



TECNOLÓGICO
NACIONAL DE MÉXICO®



INSTITUTO TECNOLÓGICO DE MEXICALI

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Desarrollo de aplicaciones móviles

“Aplicación móvil: Drinkify”

Integrantes:

Hernández López Jose Carlos - 22150084

Hernández Cortina Adolfo - 21490555

Docente:

Bogarin Valenzuela Jose Ramon

Mexicali, Baja California a 07 de noviembre 2025

Índice

Introducción.....	3
Problemática.....	4
Objetivo del Proyecto.....	4
Historias de Usuario.....	4
Requerimientos Funcionales y No Funcionales.....	5
Funcionales.....	5
No funcionales.....	6
Tipo de Arquitectura.....	7
Capas y responsabilidades.....	7
Reglas de dependencia.....	7
Wireframes.....	8
Plan de Sprints.....	12
Criterios de Aceptación.....	12
Sprint 1.....	13
Sprint 2.....	33

Introducción

En el presente documento se explica por partes cómo se llevará a cabo el desarrollo del proyecto sobre la aplicación móvil “Drinkify”, desde la problemática encontrada, el objetivo de la app, los requerimientos no funcionales y funcionales encontrados, el tipo de arquitectura que utilizaremos y los sprints y que haremos en cada uno.

Problemática

Actualmente no existe una aplicación móvil que permita a los usuarios visualizar recetas de bebidas organizadas por categorías de manera intuitiva. Los usuarios deben recurrir a redes sociales, blogs o notas personales, lo que hace que encontrar recetas confiables y organizadas sea difícil, no existe una app que:

- Organice un conjunto de bebidas en un solo lugar.
- Permita descubrir bebidas nuevas de forma fácil.
- Genere listas personalizadas automáticamente según categorías.
- Ofrezca recomendaciones aleatorias.

Objetivo del Proyecto

Desarrollar una aplicación móvil para Android usando Flutter bajo el patrón de diseño MVC (Modelo Vista Controlador) que facilite la recopilación de recetas de bebidas mediante la organización en listas.

- Ver recetas de bebidas
- Guardar recetas favoritas
- Crear recetas de bebidas en listas personalizadas

Historias de Usuario

Usuario:

- Como usuario, quiero ver recetas de bebidas divididas en categorías.
- Como usuario, quiero ver las bebidas de temporada.

- Como usuario, quiero generar una lista de diferentes bebidas de categorías específicas y darle un nombre.
- Como usuario, quiero ver las listas que he generado.
- Como usuario, quiero ver recomendaciones del día.
- Como usuario, quiero usar una barra de búsqueda para encontrar bebidas específicas.
- Como usuario, quiero ver las instrucciones para preparar las bebidas que me interesan.
- Como usuario, quiero marcar bebidas como favoritas y verlas en una lista de favoritos.

Requerimientos Funcionales y No Funcionales

Funcionales

- Gestión de usuario
- La app debe permitir registrar usuarios con correo, contraseña y nombre de usuario.
- La app debe permitir iniciar sesión con credenciales válidas.

Home

- Mostrar un mensaje de bienvenida al usuario logueado.
- Incluir una barra de búsqueda de bebidas.
- Mostrar un carrusel de bebidas de temporada.
- Mostrar un botón para ver categorías y, al seleccionarlas, desplegar la lista de bebidas.
- Mostrar un botón para la bebida recomendada del día.

Favoritos

- El usuario debe poder marcar o desmarcar bebidas como favoritas.
- La app debe mostrar bebidas guardadas en favoritos mediante un scroll..

Generación de listas

- El usuario debe poder generar máximo 4 listas seleccionando categorías de bebidas.
- La app debe crear una lista automática con las bebidas que pertenecen a esas categorías.

Gestión de listas

- El usuario debe poder visualizar todas las listas generadas.
- El usuario debe poder eliminar una lista generada.
- El usuario debe poder modificar únicamente el nombre de la lista.

No funcionales

Seguridad

- Las credenciales del usuario deben transmitirse de forma segura.

Tiempo

- Tiempo de carga no mayor a 5 segundos entre pantallas.

Diseño

- Diseño atractivo y coherente.

Intuitivo

- Fácil uso para usuarios no técnicos

Tipo de Arquitectura

El patrón de diseño que utilizaremos será “MVC” Modelo Vista Controlador.

Capas y responsabilidades

Vista

- Es la parte que el usuario ve y con la que interactúa.
- Su trabajo es mostrar la información de forma clara y bonita.

Controlador

- Es como el "puente" entre la vista y el modelo.
- Recibe lo que hace el usuario y decide qué debe pasar.
- Se asegura de que la vista muestre lo que el modelo tiene.

Modelo

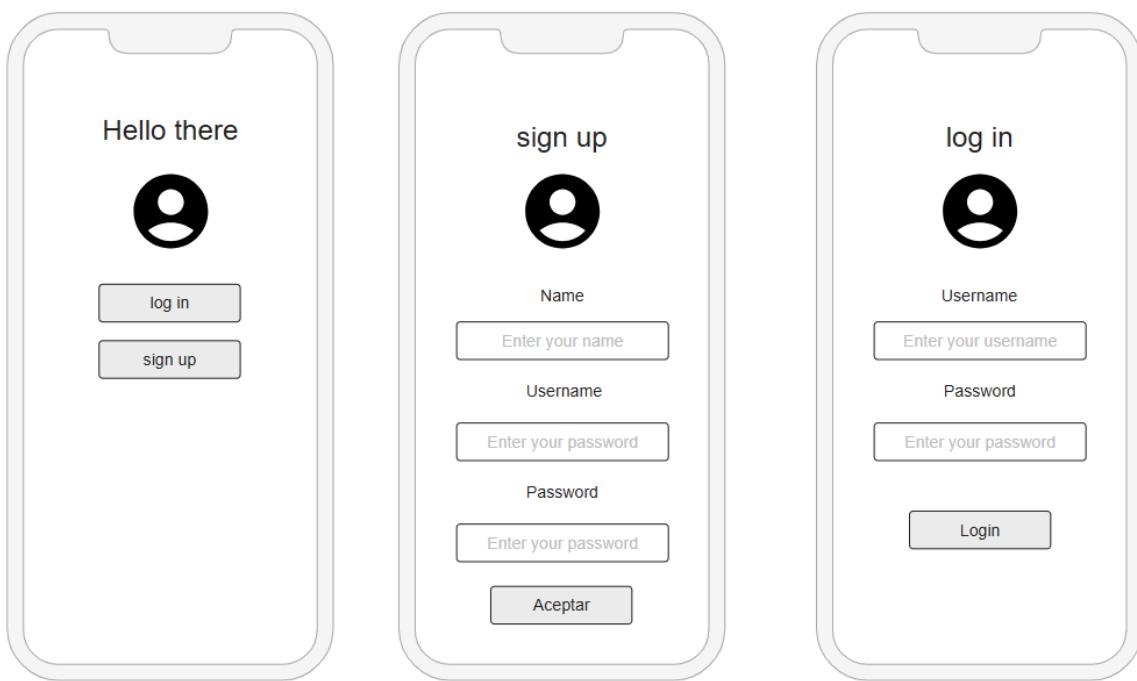
- Contiene la información y las reglas.
- Aquí estarían guardadas las recetas.
- Solo guarda y organiza los datos.

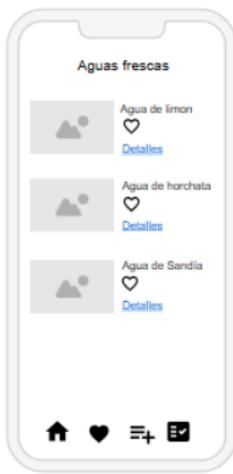
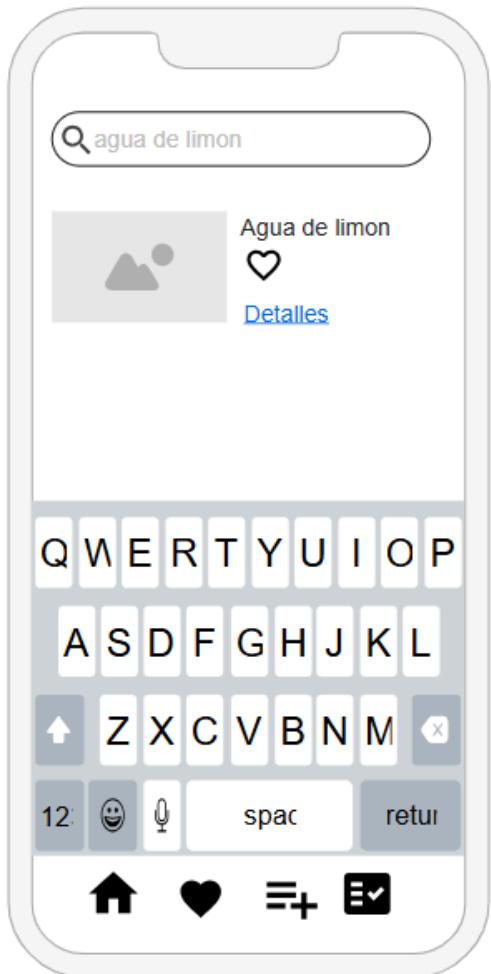
Reglas de dependencia

- La Vista depende del Controlador, porque necesita instrucciones para mostrar datos.

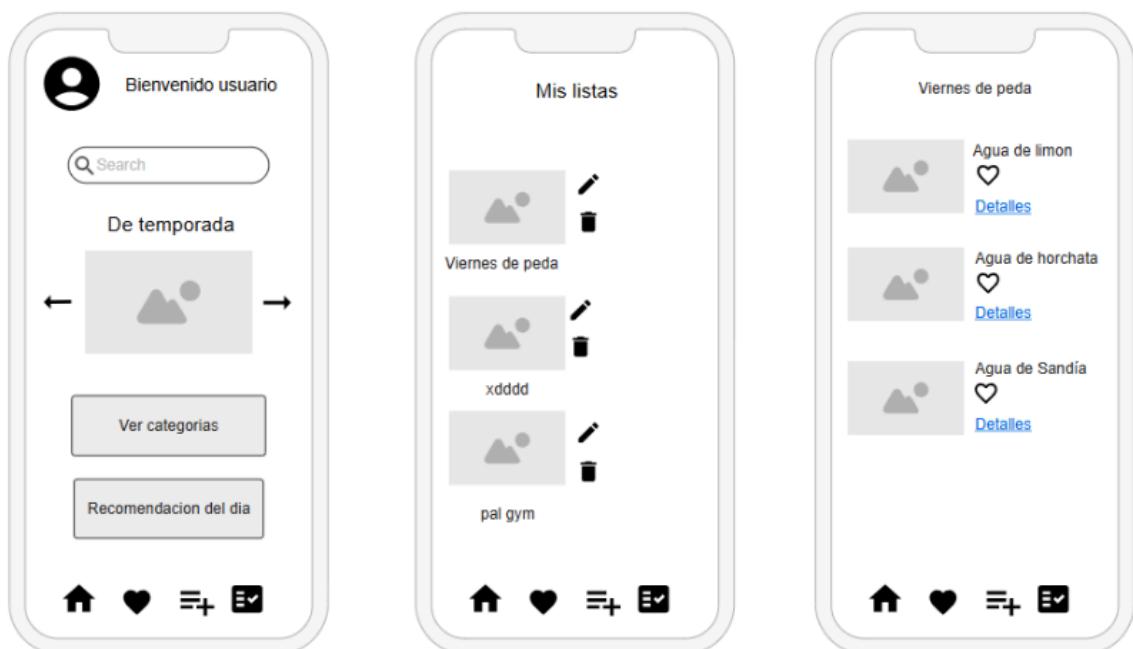
- El Controlador depende del Modelo, porque necesita obtener la información ahí.
- El Modelo no depende de nadie, es independiente, solo guarda y organiza la info.

Wireframes









Plan de Sprints

SPRINTS	DESCRIPCIÓN	Fecha
Sprint 1 – Estructura básica	<ul style="list-style-type: none">• Pantalla Iniciar sesión (UI)• Pantalla Registro (UI).• barra inferior con 4 secciones: Home, Favoritos, Crear, Mis Listas.• Pantallas vacías para cada sección.	6 - 10 oct
Sprint 2 – Pantalla Home y autenticación de usuarios.	<ul style="list-style-type: none">• Pantalla Iniciar sesión (validación)• Pantalla Registro (validación).• Mensaje de bienvenida con nombre de usuario logeado.• Barra de búsqueda (sin lógica, solo UI).• Carrusel de imágenes (estático, ejemplo).• Botones visibles: "Ver categorías" y "Recomendación del día".	20 - 24 oct
Sprint 3 – Búsqueda y categorías	<ul style="list-style-type: none">• Implementar búsqueda real conectada a la base de datos.• Pantalla de resultados de búsqueda.• Sección de categorías.• Mostrar bebidas dentro de una categoría seleccionada.	3 - 7 nov
Sprint 4 – Recomendaciones y carrusel dinámico	<ul style="list-style-type: none">• Carrusel dinámico con bebidas de temporada.• Botón "Recomendación del día" con bebida random o destacada.• Página de detalles de bebida.• Lógica para agregar/quitar favoritos.• Pantalla de favoritos mostrando bebidas guardadas.	17 - 21 nov
Sprint 5 – Favoritos y generación de listas	<ul style="list-style-type: none">• Pantalla y lógica para Generar listas seleccionando categorías.• Pantalla de "Mis listas" mostrando todas las listas generadas.• Funcionalidad para eliminar lista.• Funcionalidad para modificar nombre de lista.• Pulido general de UI (colores, iconos, tipografía).	1 - 5 dic

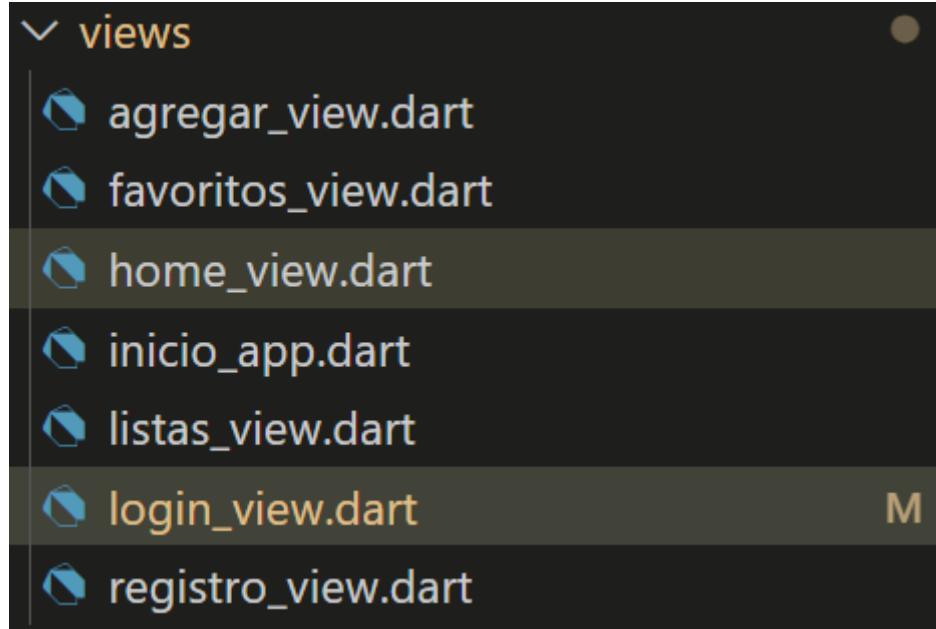
Criterios de Aceptación

Los criterios de aceptación que tomaremos una vez dado por concluido el proyecto serán los siguientes:

- Un usuario puede buscar bebidas específicas.
- Un usuario puede iniciar sesión con credenciales válidas.
- Un usuario puede crear máximo 4 listas.
- Un usuario puede dar me gusta y ver sus bebidas favoritas.

Sprint 1

Carpeta de vistas, en esta carpeta se encuentran la parte de “Vista” de la arquitectura MVC (Modelo Vista Controlador).



inicio_app.dart

En esta pantalla se muestra un título de bienvenida, un ícono y dos botones, uno para iniciar sesión y otro para registrarse, cada widget se separa con una altura para que no se vean todos pegados. También los textos están estilizados, en algunos se cambia el tamaño de la letra y la negrita a la letra, a los botones se les colocó un padding para hacerlos más grandes. Estamos utilizando la “plantilla Scaffold”. Es la primer pantalla al iniciar la aplicación.



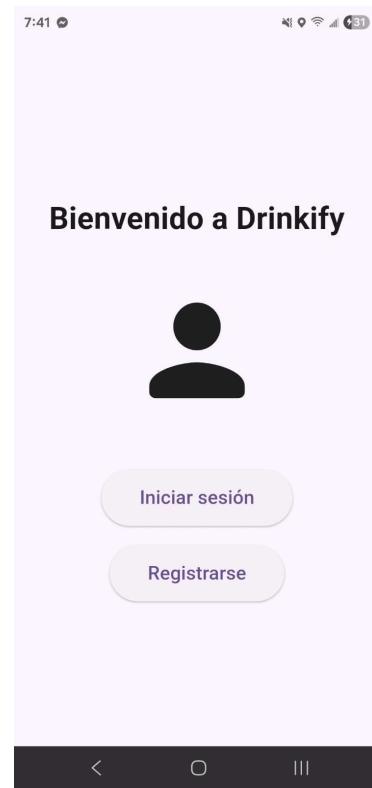
```
1  Widget build(BuildContext context) {
2      return Scaffold(
3          body: Center(
4              child: Column(
5                  mainAxisAlignment: MainAxisAlignment.center,
6                  children: [
7                      const Text(
8                          'Bienvenido a Drinkify',
9                          style: TextStyle(
10                             fontSize: 32,
11                             fontWeight: FontWeight.bold,
12                         ),
13                         textAlign: TextAlign.center,
14                     ),
15
16                     const SizedBox(height: 40),
17
18                     const Icon(
19                         Icons.person_rounded,
20                         size: 150,
21                     ),

```

Con la función definida en “onPressed” indicamos que debe llevar a la siguiente pantalla, la estructura del botón es la misma para “login” y “registro”, solo cambia a dónde lo dirige y el texto, según el botón.



Así es como se ve la pantalla ya ejecutada.



registro_view.dart

Esta es la pantalla que se muestra después de presionar el botón “Registrarse” en la pantalla “inicio_app”, aquí tenemos un appBar con el título de la pantalla, en el body un icono, tres “textfield”, un botón y un textbutton. Estamos utilizando “SingleChildScrollView” el cual nos sirve para evitar el desbordamiento de la pantalla cuando abrimos el teclado al hacer click en un textfield, permitiendo hacer “scroll” al abrirse el teclado.

```
1  return Scaffold(  
2      appBar: AppBar(  
3          title: const Text('Registro de Usuario'),  
4          centerTitle: true,  
5      ),  
6      body: Center(  
7          child: SingleChildScrollView(  
8              padding: const EdgeInsets.symmetric(horizontal: 30),  
9              child: Column(  
10                  mainAxisAlignment: MainAxisAlignment.center,  
11                  children: [  
12                      const Icon(  
13                          Icons.person_add_rounded,  
14                          size: 120,  
15                  ),
```

En cada textfield tenemos un controlador, la variable de cada controlador para cada textfield es diferente, colocamos un icono y un texto para indicar que va en cada textfield, en los tres textfield de la pantalla se conserva la misma estructura, lo único que cambia es el nombre, el icono y el controlador, a excepción del textfield de la contraseña donde se agrega “obscureText” para ocultar la contraseña.

```
1  TextField(  
2      controller: nameController,  
3      decoration: const InputDecoration(  
4          LabelText: 'Nombre',  
5          border: OutlineInputBorder(),  
6          prefixIcon: Icon(Icons.badge_outlined),  
7      ),  
8  ),
```

```
1  final TextEditingController nameController = TextEditingController();  
2  final TextEditingController usernameController = TextEditingController();  
3  final TextEditingController passwordController = TextEditingController();
```

Al presionar este botón lo que hace es mostrar un “snackbar” que lo que es es mostrar un mensaje en la parte inferior de la app indicando que el registro fue exitoso. Al igual que en los botones de la pantalla anterior se le aplicó un padding.

```
1 ElevatedButton(  
2   onPressed: () {  
3     // Simula registro: cambia estado a logueado  
4     //auth.login(); // Esto activa el authController  
5     ScaffoldMessenger.of(context).showSnackBar(  
6       const SnackBar(content: Text('Usuario registrado con éxito!')),  
7     );  
8   },  
9   style: ElevatedButton.styleFrom(  
10    padding: const EdgeInsets.symmetric(horizontal: 50, vertical: 15),  
11  ),  
12  child: const Text(  
13    'Registrar',  
14    style: TextStyle(fontSize: 18),  
15  ),  
16),
```

En este botón lo que se indica es que al presionarlo lo regresa a la pantalla inicial, lo cual su mismo nombre indica “Regresar”.

```
1 TextButton(  
2   onPressed: () {  
3     NavigationController.navigateTo(context, '/inicio'); // Regresa a InicioApp  
4   },  
5   child: const Text(  
6    'Regresar',  
7    style: TextStyle(fontSize: 16),  
8  ),  
9 ),
```

Así es como se ve la pantalla ya ejecutada y presionando el botón para registrarse.



login_view.dart

Esta es la pantalla que se muestra después de presionar el botón “Iniciar sesión” en la pantalla “inicio_app”, aquí tenemos un appBar con el título de la pantalla centrado, en el body un ícono, dos “textfield”, un botón y un textbutton. Estamos utilizando “SingleChildScrollView” el cual nos sirve para evitar el desbordamiento de la pantalla cuando abrimos el teclado al hacer click en un textfield, permitiendo hacer “scroll” al abrirse el teclado.

En cada textfield tenemos un controlador, la variable de cada controlador para cada textfield es diferente, colocamos un ícono y un texto para indicar que va en cada textfield, en los dos textfield de la pantalla se conserva la misma estructura, lo único que cambia es el nombre, el ícono y el controlador, a excepción del textfield de la contraseña donde se agrega “obscureText” para ocultar la contraseña.

```
1 final TextEditingController usernameController = TextEditingController();
2 final TextEditingController passwordController = TextEditingController();
3
4 return Scaffold(
5   appBar: AppBar(
6     title: const Text('Iniciar Sesión'),
7     centerTitle: true,
8   ),
9   body: Center(
10     child: SingleChildScrollView(
11       padding: const EdgeInsets.symmetric(horizontal: 30),
12       child: Column(
13         mainAxisAlignment: MainAxisAlignment.center,
14         children: [
15           const Icon(
16             Icons.person_rounded,
17             size: 120,
18           ),
19
20           const SizedBox(height: 30),
21
22           TextField(
23             controller: usernameController,
24             decoration: const InputDecoration(
25               labelText: 'Usuario',
26               border: OutlineInputBorder(),
27               prefixIcon: Icon(Icons.person_outline),
28             ),
29           ),

```

Al presionar este botón lo que hace es mostrar un “snackbar” que lo que es es mostrar un mensaje en la parte inferior de la app indicando que el registro fue exitoso. Al igual que en los botones de la pantalla anterior se le aplicó un padding.

```
● ● ●  
1 ElevatedButton(  
2   onPressed: () {  
3     // Aquí luego conectaremos la lógica de autenticación  
4     NavigatorController.navigateTo(context, '/home');  
5   },  
6   style: ElevatedButton.styleFrom(  
7     padding: const EdgeInsets.symmetric(horizontal: 50, vertical: 15),  
8   ),  
9   child: const Text(  
10    'Entrar',  
11    style: TextStyle(fontSize: 18),  
12  ),  
13 ),
```

En este botón lo que se indica es que al presionarlo lo regresa a la pantalla inicial, lo cual su mismo nombre indica “Regresar”.

```
● ● ●  
1 TextButton(  
2   onPressed: () {  
3     //NavigatorController.navigateTo(context, '/inicio');// Regresa a InicioApp  
4     context.push('/inicio');  
5   },  
6   child: const Text(  
7    'Regresar',  
8    style: TextStyle(fontSize: 16),  
9  ),  
10 ),
```

Así es como se ve la pantalla ya ejecutada.



home_view.dart

En esta pantalla tenemos un appBar con el título y IconButton para regresar a la pantalla inicial, en el body tenemos un texto e implementamos una barra de navegacion en la parte inferior para navegar entre home, favoritos, agregar y listas.

```
 1 import 'package:appdrinkify/controllers/navigation_controller.dart';
 2 import 'package:flutter/material.dart';
 3 import '../widgets/bottom_nav_bar.dart';
 4
 5 class HomeView extends StatelessWidget {
 6   const HomeView({super.key});
 7
 8   @override
 9   Widget build(BuildContext context) {
10
11     return Scaffold(
12       appBar: AppBar(
13         title: const Text('Ventana de Home'),
14         actions: [
15           IconButton(
16             onPressed: ()=> NavigationController.navigateTo(context,'/inicio'), // cerrar sesión → redirige automáticamente
17             icon: const Icon(Icons.logout),
18           ),
19         ],
20       ),
21       body: const Center(child: Text('Bienvenido a Home')),
22       bottomNavigationBar: const BottomNavBar(),
23     );
24   }
25 }
```

Así es como se ve la pantalla ya ejecutada.



tiliiiiin que paso tilinajj?, putas imagenes no me deja copiarlas, les tengo que sacar captura ni pedo we xdxwdx, las del snap?, y las del celular tmb, te las paso?, ya no agregues las de home ni nada de eso, por?, ya lo quiero acabar we, nomas deja esa para agarrar la barra de navegacion, borrar, vavava todas las pantalla que pegue?, ya las agregue, y voy justo aqui xdx no papu :,v avavvavavavava

favoritos_view.dart

En esta pantalla tenemos lo mismo que contiene la pantalla home con la diferencia de que no tenemos el botón en el navbar, solo tenemos el navbar con el título, el texto en el body y la barra de navegación inferior.

```
● ● ●  
1 import 'package:appdrinkify/widgets/bottom_nav_bar.dart';  
2 import 'package:flutter/material.dart';  
3 class FavoritosView extends StatelessWidget {  
4     const FavoritosView({super.key});  
5  
6  
7     @override  
8     Widget build(BuildContext context) {  
9  
10        return Scaffold(  
11            appBar: AppBar(title: const Text('Ventana de Favoritos')),  
12            body: const Center(child: Text('Ventana de Favoritos')),  
13            bottomNavigationBar: const BottomNavBar(),  
14        );  
15    }  
16}
```

Así es como se ve la pantalla ya ejecutada.



agregar_view.dart

En esta pantalla tenemos lo mismo que contiene la pantalla home con la diferencia de que no tenemos el botón en el navbar, solo tenemos el navbar con el título, el texto en el body y la barra de navegación inferior.

```
● ● ●  
1 import 'package:appdrinkify/widgets/bottom_nav_bar.dart';  
2 import 'package:flutter/material.dart';  
3 class AgregarView extends StatelessWidget {  
4     const AgregarView({super.key});  
5  
6  
7     @override  
8     Widget build(BuildContext context) {  
9  
10        return Scaffold(  
11            appBar: AppBar(title: const Text('Ventana de Agregar')),  
12            body: const Center(child: Text('Ventana de Agregar')),  
13            bottomNavigationBar: const BottomNavBar(),  
14        );  
15    }  
16}
```

Así es como se ve la pantalla ya ejecutada.



listas_view.dart

En esta pantalla tenemos lo mismo que contiene la pantalla home con la diferencia de que no tenemos el botón en el navbar, solo tenemos el navbar con el título, el texto en el body y la barra de navegación inferior.

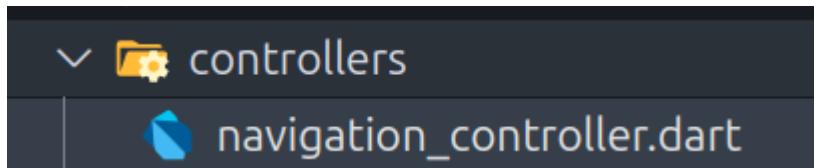
```
1 import 'package:appdrinkify/widgets/bottom_nav_bar.dart';
2 import 'package:flutter/material.dart';
3
4 class ListasView extends StatelessWidget {
5   const ListasView({super.key});
6
7
8   @override
9   Widget build(BuildContext context) {
10
11     return Scaffold(
12       appBar: AppBar(title: const Text('Ventana de Listas')),
13       body: const Center(child: Text('Ventana de Listas')),
14       bottomNavigationBar: const BottomNavBar(),
15     );
16   }
17 }
```

Así es como se ve la pantalla ya ejecutada.



Carpeta controllers.

navigation_controller.dart



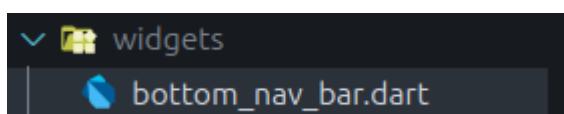
Aquí creamos una clase estática llamada NavigationController y dentro, definimos un método estático llamado navigateTo, que recibe el contexto (BuildContext context) y el nombre de la ruta (String route).

Llama a context.go(route) para redirigir a la vista indicada.



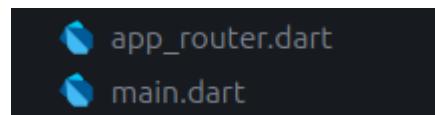
Carpeta widgets.

bottom_nav_bar.dart.



Aquí creamos un Widget para la barra inferior de navegación, que va a manejar la navegación en la app en las distintas vistas que son (Home,Favoritos,Agregar y Listas).

```
1  @override
2  Widget build(BuildContext context) {
3      final location = GoRouterState.of(context).uri.toString();
4
5      return BottomNavigationBar(
6          currentIndex: _getCurrentIndex(location),
7          type: BottomNavigationBarType.fixed,
8          onTap: (index) {
9              switch (index) {
10                  case 0:
11                      NavigatorController.navigateTo(context, '/home');
12                      break;
13                  case 1:
14                      NavigatorController.navigateTo(context, '/favoritos');
15                      break;
16                  case 2:
17                      NavigatorController.navigateTo(context, '/agregar');
18                      break;
19                  case 3:
20                      NavigatorController.navigateTo(context, '/listas');
21                      break;
22              }
23          },
24          items: const [
25              BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
26              BottomNavigationBarItem(icon: Icon(Icons.favorite), label: 'Favoritos'),
27              BottomNavigationBarItem(icon: Icon(Icons.add), label: 'Aregar'),
28              BottomNavigationBarItem(icon: Icon(Icons.list), label: 'Listas'),
29          ],
30      );
31 }
```



app_router.dart

Aquí creamos el método `createRouter()` devuelve un objeto `GoRouter`, que el `main.dart` usa para controlar la navegación.

En `initialLocation: '/inicio'` indica que al abrir la app, la primera pantalla visible será `/inicio`, es decir, `InicioApp`.

```
1 GoRouter createRouter() {
2   return GoRouter(
3     initialLocation: '/inicio',
4     routes: [
5       GoRoute(path: '/inicio', builder: (context, state) => const InicioApp()),
6       GoRoute(path: '/login', builder: (context, state) => const LoginView()),
7       GoRoute(path: '/registro', builder: (context, state) => const RegistroView()),
8       GoRoute(path: '/home', builder: (context, state) => const HomeView()),
9       GoRoute(path: '/favoritos', builder: (context, state) => const FavoritosView()),
10      GoRoute(path: '/agregar', builder: (context, state) => const AgregarView()),
11      GoRoute(path: '/listas', builder: (context, state) => const ListasView()),
12    ],
13  );
14 }
```

main.dart

Aquí se llama a la función createRouter() definida en app_router.dart, esta función devuelve un objeto GoRouter, que administra toda la navegación.

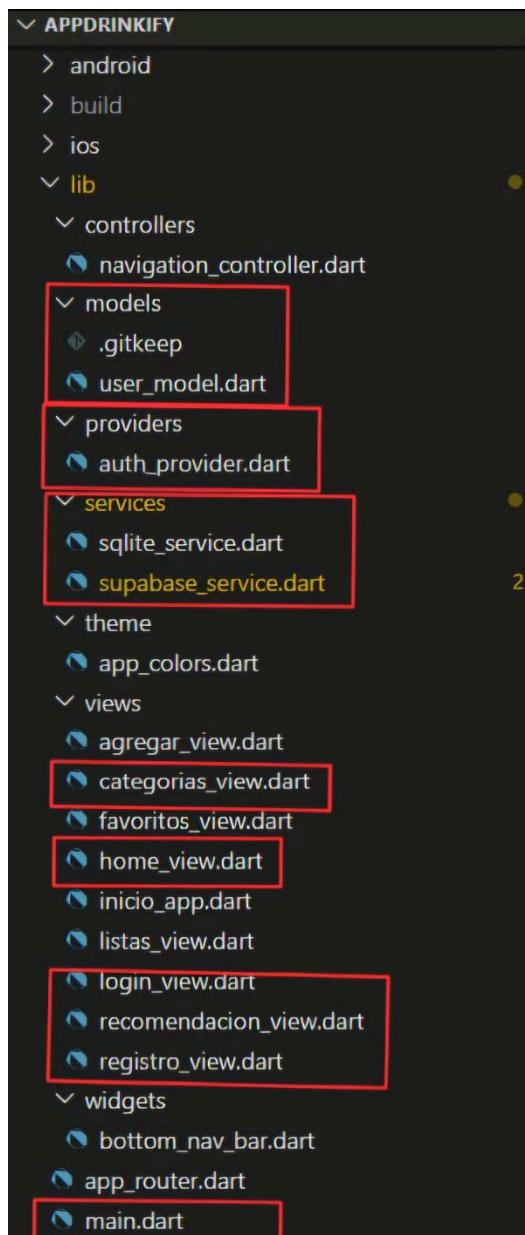
```
1 @override
2 Widget build(BuildContext context) {
3   final router = createRouter();
4
5   return MaterialApp.router(
6     debugShowCheckedModeBanner: false,
7     title: 'Drinkify',
8     theme: ThemeData(
9       colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
10      ),
11      routerConfig: router,
12    );
13 }
```

Sprint 2

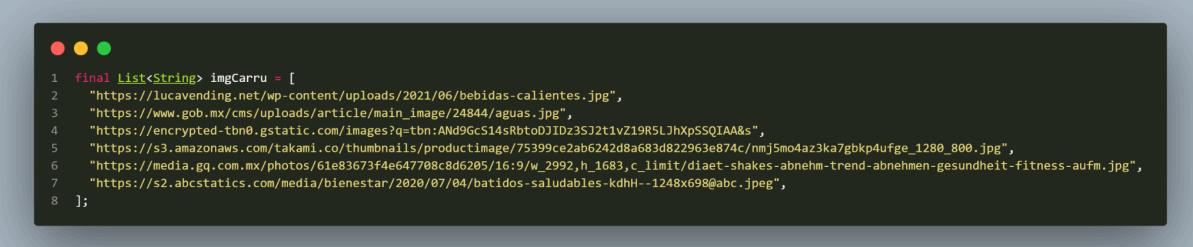
Observaciones del profesor

- Validar los textfields para el registro y el login.
- Mostrar el nombre del usuario logueado en la pantalla “Home”.
- Crear una carpeta llamada “config” y guardar ahí las conexiones entrantes y las variables de entorno.

Estos son los archivos y/o carpetas que se crearon y/o modificaron para el segundo sprint.

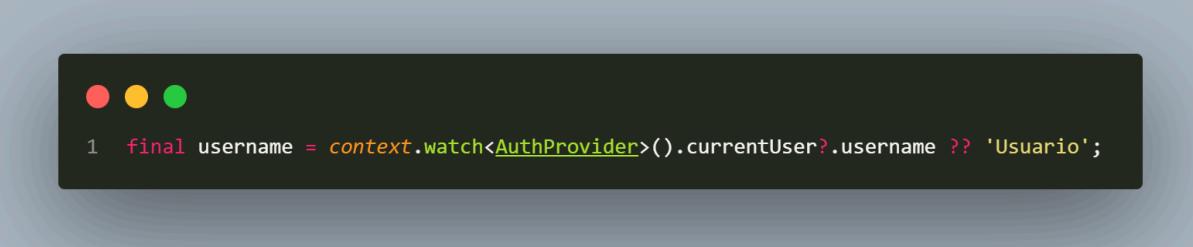


home_view.dart



```
1 final List<String> imgCarru = [
2   "https://lucavending.net/wp-content/uploads/2021/06/bebidas-calientes.jpg",
3   "https://www.gob.mx/cms/uploads/article/main_image/24844/aguas.jpg",
4   "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9Gcs14skbt0JJDz35J2t1vZ19R5LJhXpSSQIA&s",
5   "https://s3.amazonaws.com/takami.co-thumbnails/productimage/75399ce2ab6242d8a683d822963e874c/nmj5mo4az3ka7gbkp4ufge_1280_800.jpg",
6   "https://media.gq.com.mx/photos/61e83673f4ef647708c8d6205/16:9/w_2992,h_1683,c_limit/diaet-shakes-abnehm-trend-abnehmen-gesundheit-fitness-aufm.jpg",
7   "https://s2.abcstatics.com/media/bienestar/2020/07/04/batidos-saludables-kdhH--1248x698@abc.jpeg",
8 ];
```

Aquí se creó una lista llamada “imgCarru” con las url de las imágenes en “String” que se usarán en el carrusel.



```
1 final username = context.watch<AuthProvider>().currentUser?.username ?? 'Usuario';
```

En esta variable se está usando context.watch para esperar a el provider para obtener el nombre del usuario logueado y de no encontrarlo usar “Usuario”.



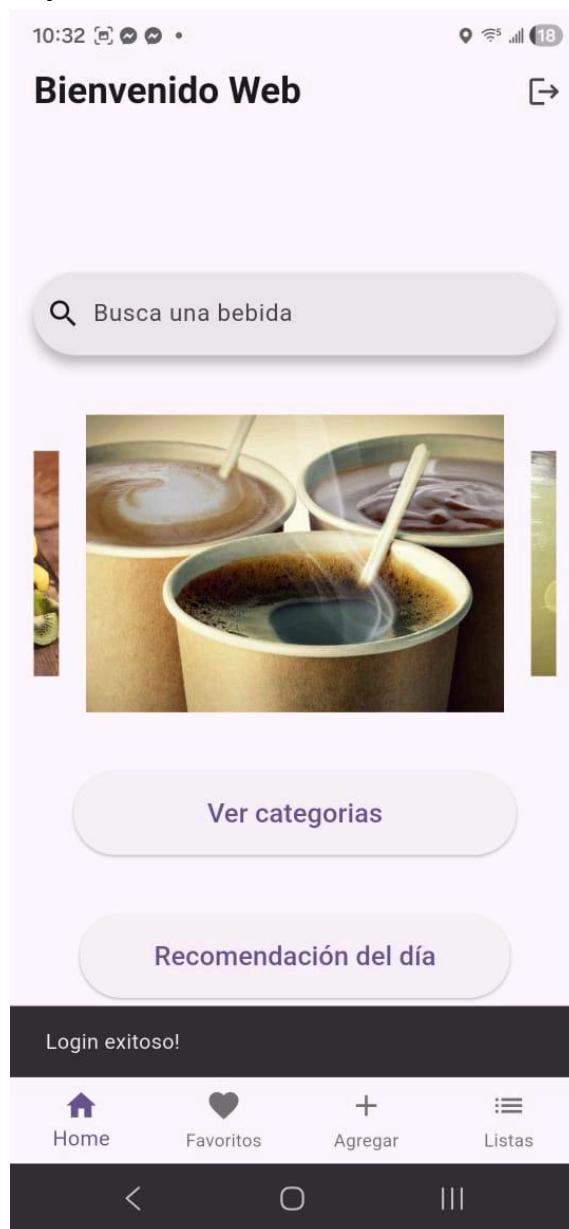
```
1 appBar: AppBar(
2   //leading: Icon(Icons.person_rounded, size: 80),
3   title: Text('Bienvenido $username',
4     style: const TextStyle(
5       fontSize: 25,
6       fontWeight: FontWeight.bold,
7     ),
8   ),
9   actions: [
10     IconButton(
11       onPressed:(){
12         final authProvider = Provider.of<AuthProvider>(context, listen: false);
13         authProvider.logout();
14         NavigatorController.navigateTo(context, '/inicio');
15       },
16       icon: const Icon(Icons.logout),
17     ),
18   ],
19 ),
```

Y en el “title” del appBar se llama a la variable que obtiene el nombre del usuario.

```
1  body: Center(
2    child: Padding(
3      padding: const EdgeInsets.symmetric(horizontal: 16.0),
4      child: SingleChildScrollView(
5        child: Column(
6          mainAxisAlignment: MainAxisAlignment.center,
7          children: [
8            const SizedBox(height: 40),
9            SearchBar(
10              Leading: const Icon(Icons.search),
11              hintText: "Busca una bebida",
12            ),
13
14            const SizedBox(height: 40),
15            CarouselSlider(
16              items: imgCarru.map((e) => Center(
17                child: Image.network(
18                  e,
19                  width: MediaQuery.of(context).size.width,
20                  height: 200,
21                  fit: BoxFit.cover,)
22            ).toList(),
23            options: CarouselOptions(
24              autoPlay: true,
25              autoPlayInterval: Duration(seconds: 3),
26              enlargeCenterPage: true,
27              enlargeFactor: 0.3,
28              height: 200,
29            ),
30          ),
31
32          const SizedBox(height: 40),
33          ElevatedButton(
34            //onPressed: () {},
35            onPressed:()=> NavigationController.navigateTo(context,'/categorias'),
36            style: ElevatedButton.styleFrom(
37              padding: const EdgeInsets.symmetric(horizontal: 90, vertical: 15),
38            ),
39            child: const Text(
40              'Ver categorias',
41              style: TextStyle(fontSize: 18),
42            ),
43          ),
44          const SizedBox(height: 40),
45          ElevatedButton(
46            onPressed:()=> NavigationController.navigateTo(context,'/recomendacion'),
47            style: ElevatedButton.styleFrom(
48              padding: const EdgeInsets.symmetric(horizontal: 50, vertical: 15),
49            ),
50            child: const Text(
51              'Recomendación del día',
52              style: TextStyle(fontSize: 18),
53            ),
54          ),
55        ],
56      ),
57    ),
58  ),
59 ),
```

Más abajo en el body se usa el widget “searchBar”, el cual ya viene en la librería de “material.dart”, se coloca un icono de búsqueda y un hintText. también se usa el widget del carrusel el cual proviene de una librería, ahí se mapea la lista de imágenes y se almacena en la variable “e”, se centra y se llaman a las imágenes con “image.network”, se llama a la variable “e” y para que todas las imágenes tengan el mismo tamaño se usa “width, height y BoxFit.cover” y con “.toList” se convierte en una lista, en las opciones del carrusel se configura por así decirlo el carrusel, cambiará de imagen automáticamente cada 3 segundos, la página central será más grande que las demás y cada imagen tendrá un separación. Y finalmente dos botones para llevar a las pantallas correspondientes, categorías y recomendación del día.

Así se ve la pantalla ya ejecutada:



categorias_view.dart

```
1 import 'package:flutter/material.dart';
2 import 'package:appdrinify/controllers/navigation_controller.dart';
3
4 class CategoriesView extends StatelessWidget {
5   const CategoriesView({super.key});
6
7   @override
8   Widget build(BuildContext context) {
9     return Scaffold(
10       appBar: AppBar(
11         leading: IconButton(
12           onPressed: () => NavigationController.navigateTo(context, '/home'),
13           icon: const Icon(Icons.arrow_back),
14         ),
15         title: const Text('Explora distintas categorías'),
16         centerTitle: true,
17       ),
18       body: Center(
19         child: Column(
20           mainAxisAlignment: MainAxisAlignment.spaceAround,
21           children: [
22             Row(
23               mainAxisSize: MainAxisSize.spaceAround,
24               children: [
25                 Column(
26                   children: [
27                     TextButton(
28                       onPressed: () => NavigationController.navigateTo(context, '/home'),
29                       child: Image.network("https://www.gob.mx/cms/uploads/article/main_image/24844/aguas.jpg",
30                         width: 160,
31                         height: 120,
32                         fit: BoxFit.cover),
33                     ),
34                     const Text("Aguas frescas", style: TextStyle(fontSize: 20)),
35                   ],
36                 ),
37                 TextButton(
38                   onPressed: () => NavigationController.navigateTo(context, '/home'),
39                   child: Image.network("https://lucavending.net/wp-content/uploads/2021/06/bebidas-calientes.jpg",
40                     width: 160,
41                     height: 120,
42                     fit: BoxFit.cover),
43                 ),
44                 const Text("Calientes", style: TextStyle(fontSize: 20)),
45               ],
46             ),
47             Column(
48               children: [
49                 TextButton(
50                   onPressed: () => NavigationController.navigateTo(context, '/home'),
51                   child: Image.network("https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS14sRbt0DJIdz3SJ2t1vZ19R5LJhXpSSQ1AA&s",
52                     width: 160,
53                     height: 120,
54                     fit: BoxFit.cover),
55                 ),
56                 const Text("Con alcohol", style: TextStyle(fontSize: 20)),
57               ],
58             ),
59             Column(
60               children: [
61                 TextButton(
62                   onPressed: () => NavigationController.navigateTo(context, '/home'),
63                   child: Image.network("https://s3.amazonaws.com/takami.co-thumbnails/productimage/75399ce2ab6242d8a683d822963e874c/nmj5mo4azka7gbkp4ufge_1280_800.jpg",
64                     width: 160,
65                     height: 120,
66                     fit: BoxFit.cover),
67                 ),
68                 const Text("Jugos Clásicos", style: TextStyle(fontSize: 20)),
69               ],
70             ),
71             Column(
72               children: [
73                 TextButton(
74                   onPressed: () => NavigationController.navigateTo(context, '/home'),
75                   child: Image.network("https://media.gq.com.mx/photos/61e83673f4e647708c8d6205/16:9/w_2992,h_1683,c_limit/diaet-shakes-abnehmen-gesundheit-fitness-aufm.jpg",
76                     width: 160,
77                     height: 120,
78                     fit: BoxFit.cover),
79                 ),
80                 const Text("Jugos fitness", style: TextStyle(fontSize: 20)),
81               ],
82             ),
83             Column(
84               children: [
85                 TextButton(
86                   onPressed: () => NavigationController.navigateTo(context, '/home'),
87                   child: Image.network("https://s2.abcstatics.com/media/bienestar/2020/07/04/batidos-saludables-kdhH-1248x698@abc.jpeg",
88                     width: 160,
89                     height: 120,
90                     fit: BoxFit.cover),
91                 ),
92                 const Text("Batidos", style: TextStyle(fontSize: 20)),
93               ],
94             ),
95           ],
96         ],
97       ),
98       //bottomNavigationBar: const BottomNavBar(),
99     );
100 }
101 }
```

En esta pantalla se mostrarán las categorías de las bebidas, se muestra un textButton con una imagen sacada de internet y un texto debajo indicando el nombre de la categoría, esto se repite seis veces, para que aparezcan uno al lado del otro se colocó “Row”, en cada textButton se puso la imagen de internet con “Image.network” y se establece un tamaño igual para todos con “width, height y BoxFit.cover”.

Así se ve la pantalla ya ejecutada:

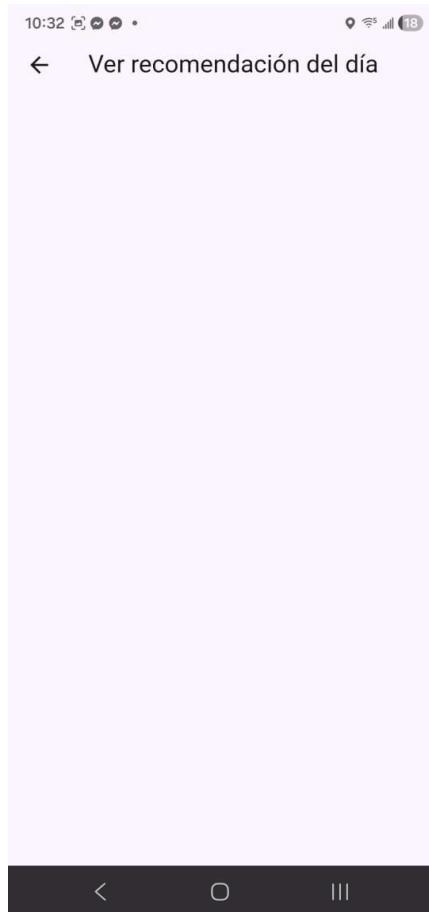


recomendacion_view.dart

```
 1 import 'package:flutter/material.dart';
 2 import 'package:appdrinky/controllers/navigation_controller.dart';
 3
 4 class RecomendacionView extends StatelessWidget{
 5   const RecomendacionView({super.key});
 6
 7   @override
 8   Widget build(BuildContext context) {
 9     return Scaffold(
10       appBar: AppBar(
11         leading: IconButton(
12             onPressed: ()=> NavigationController.navigateTo(context,'/home'),
13             icon: const Icon(Icons.arrow_back),
14         ),
15         title: const Text('Ver recomendación del día'),
16         centerTitle: true,
17       ),
18       body: Center(
19
20     ),
21     //bottomNavigationBar: const BottomNavBar(),
22   );
23 }
24 }
```

Al hacer click sobre el botón “Recomendación del día” lo redirige a esta pantalla donde se mostrará la bebida recomendada, por el momento solo tenemos en el appBar un IconButton que lo regresa a la pantalla anterior, con un leading para mostrar primero el icono que el texto que indica la página.

Así se ve la pantalla ya ejecutada:

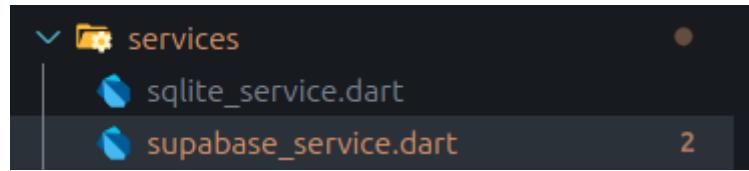


pubspec.yaml

Estas fueron las librerías que se instalaron para poder llevar a cabo el segundo sprint.

```
dependencies:
  flutter:
    sdk: flutter
  go_router: ^16.2.4
  provider: ^6.0.5
  carousel_slider: ^5.1.1
  supabase_flutter: ^2.10.3
  sqflite: ^2.2.0
  path: ^1.8.3
```

Carpeta services.



sqlite services tiene código, pero de momento no es relevante para aplicación.
Y en supabase_service, tenemos 2 funciones asíncronas, **registerUser**, **loginUser**.

registerUser.

```
1 Future<bool> registerUser(UserModel user) async {
2     try {
3
4         // 1 Revisar si el email ya existe
5         final existing = await client
6             .from('users')
7             .select()
8             .eq('email', user.email)
9             .maybeSingle(); // devuelve null si no existe
10
11        if (existing != null) {
12            print('El email ya está registrado');
13            return false; // No permitimos registro duplicado
14        }
15
16        final response = await client
17            .from('users')
18            .insert({
19                'email': user.email,
20                'username': user.username,
21                'password': user.password,
22            })
23            .select();
24
25        // Revisamos si hay error manualmente
26        if (response == null || (response as List).isEmpty) {
27            print('Error al registrar en Supabase');
28            return false;
29        }
30
31        print('Usuario registrado en Supabase: $response');
32        return true;
33    } catch (e) {
34        print('Excepción al registrar usuario: $e');
35        return false;
36    }
37}
```

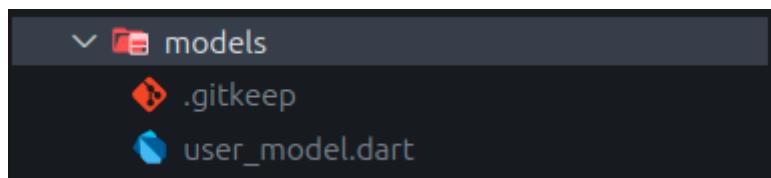
Donde primero en la variable `existing`, hace una consulta a la base de datos, y no permite registrar correos que ya fueron duplicados, sino entonces hará el registro a Supabase con la variable `response`.

LoginUser

```
1 Future<UserModel?> loginUser(String email, String password) async {
2     try {
3         final response = await client
4             .from('users')
5             .select()
6             .eq('email', email)
7             .eq('password', password)
8             .maybeSingle(); // No usamos execute()
9
10        // Revisamos si no hay datos
11        if (response == null) {
12            print('Usuario no encontrado o contraseña incorrecta');
13            return null;
14        }
15
16        return UserModel.fromMap(response as Map<String, dynamic>);
17    } catch (e) {
18        print('Excepción al loguear usuario: $e');
19        return null;
20    }
21 }
```

Aquí recibimos 2 parámetros el email y el password y hace la verificación con la variable cliente de supabase, si regresa null, manda un mensaje indicando que el usuario no fue encontrado o que el password esta mal, si no regresa un UserModel con la información del usuario.

Carpeta models.



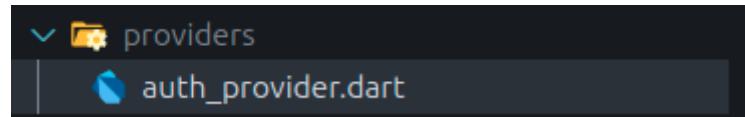
user_model.

Dentro de UserModel tenemos sus variables, el constructor, el metodo toMap, para convertir el objeto en un json, y fromMap, para hacer el inverso del metodo toMap.

```
● ● ●
```

```
1 class UserModel {  
2     final int? id;  
3     final String email;  
4     final String username;  
5     final String password;  
6  
7     UserModel({  
8         this.id,  
9         required this.email,  
10        required this.username,  
11        required this.password,  
12    });  
13  
14    Map<String, dynamic> toMap() {  
15        return {  
16            'id': id,  
17            'email': email,  
18            'username': username,  
19            'password': password,  
20        };  
21    }  
22  
23    factory UserModel.fromMap(Map<String, dynamic> map) {  
24        return UserModel(  
25            id: map['id'],  
26            email: map['email'],  
27            username: map['username'],  
28            password: map['password'],  
29        );  
30    }  
31 }  
32
```

Providers.



auth_provider.dart

Aquí manejamos se maneja la autenticación del usuario utilizando un servicio de Supabase (SupabaseService), permitiendo registrar y autenticar usuarios mediante métodos como login y register.

Y también la gestión del estado, aquí utilizamos el patrón ChangeNotifier de Flutter para notificar a los widgets cuando el estado del usuario cambia (inicio de sesión, registro, cierre de sesión).

```
1 class AuthProvider extends ChangeNotifier {
2   final SupabaseService _supabaseService = SupabaseService();
3
4   UserModel? _currentUser;
5
6   UserModel? get currentUser => _currentUser;
7   bool get isLoggedIn => _currentUser != null;
8
9   Future<void> login(String email, String password) async {
10     final user = await _supabaseService.loginUser(email, password);
11     if (user != null) {
12       _currentUser = user;
13       notifyListeners();
14     }
15   }
16
17   Future<void> register(String username, String email, String password) async {
18     final user = UserModel(
19       username: username,
20       email: email,
21       password: password,
22     );
23     final success = await _supabaseService.registerUser(user);
24     if (success) {
25       _currentUser = user;
26       notifyListeners();
27     }
28   }
29
30   void logout() async {
31   // Limpiamos el usuario en memoria
32   _currentUser = null;
33
34   // Opcional: limpiar SQLite
35   await SQLiteService.deleteAllUsers();
36
37   notifyListeners();
38 }
```

The code is a Dart class named 'AuthProvider' that extends 'ChangeNotifier'. It has a private field '_supabaseService' of type 'SupabaseService'. It also has a private field '_currentUser' of type 'UserModel?' and two public getters: 'currentUser' and 'isLoggedIn'. The 'login' method takes an email and password, uses 'await' to call the 'loginUser' method from '_supabaseService', and then updates '_currentUser' and calls 'notifyListeners'. The 'register' method takes a username, email, and password, creates a new 'UserModel' instance, uses 'await' to call the 'registerUser' method from '_supabaseService', and then updates '_currentUser' and calls 'notifyListeners'. The 'logout' method sets '_currentUser' to null and calls 'notifyListeners'. There is also a comment in the code indicating that optional SQLite cleanup is being performed.

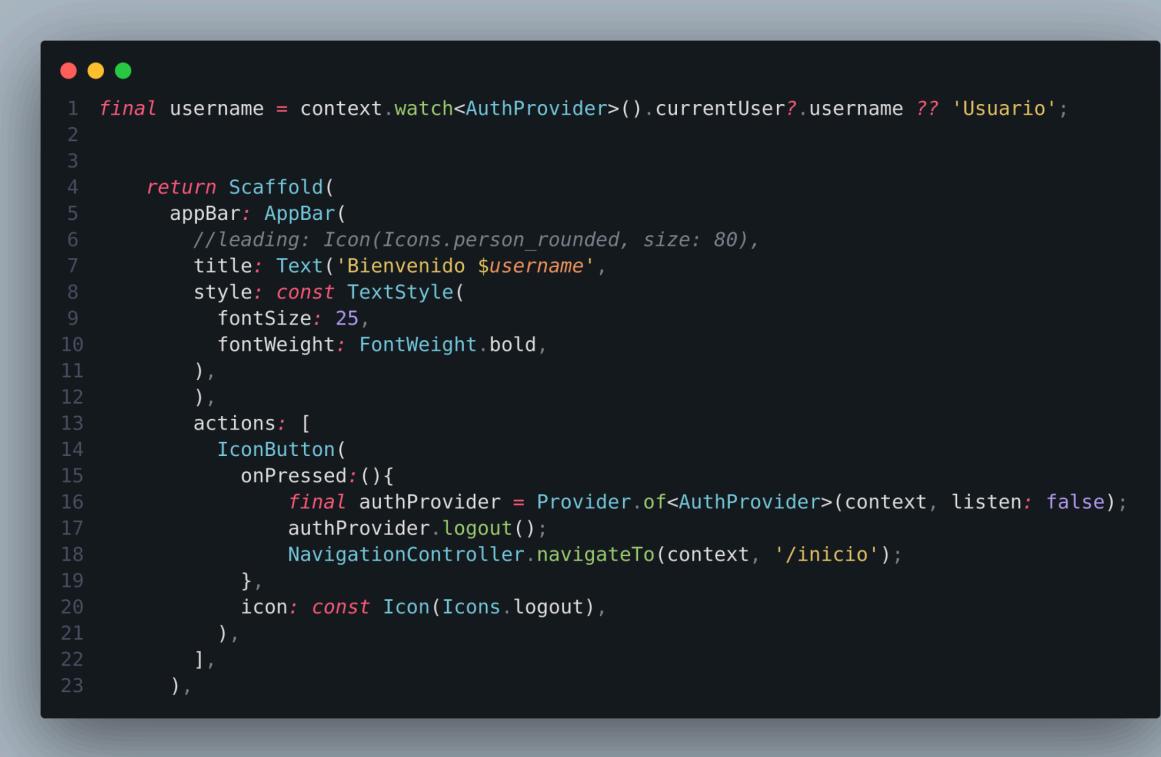
Cambios en el home_view.

El código muestra el nombre del usuario actual en la barra superior (AppBar) usando los datos del AuthProvider y si no hay usuario logueado, muestra “Usuario” por defecto.

Además, incluye un botón de cerrar sesión, que:

Llama a authProvider.logout() para limpiar el estado del usuario.

Redirige a la pantalla de inicio (/inicio)



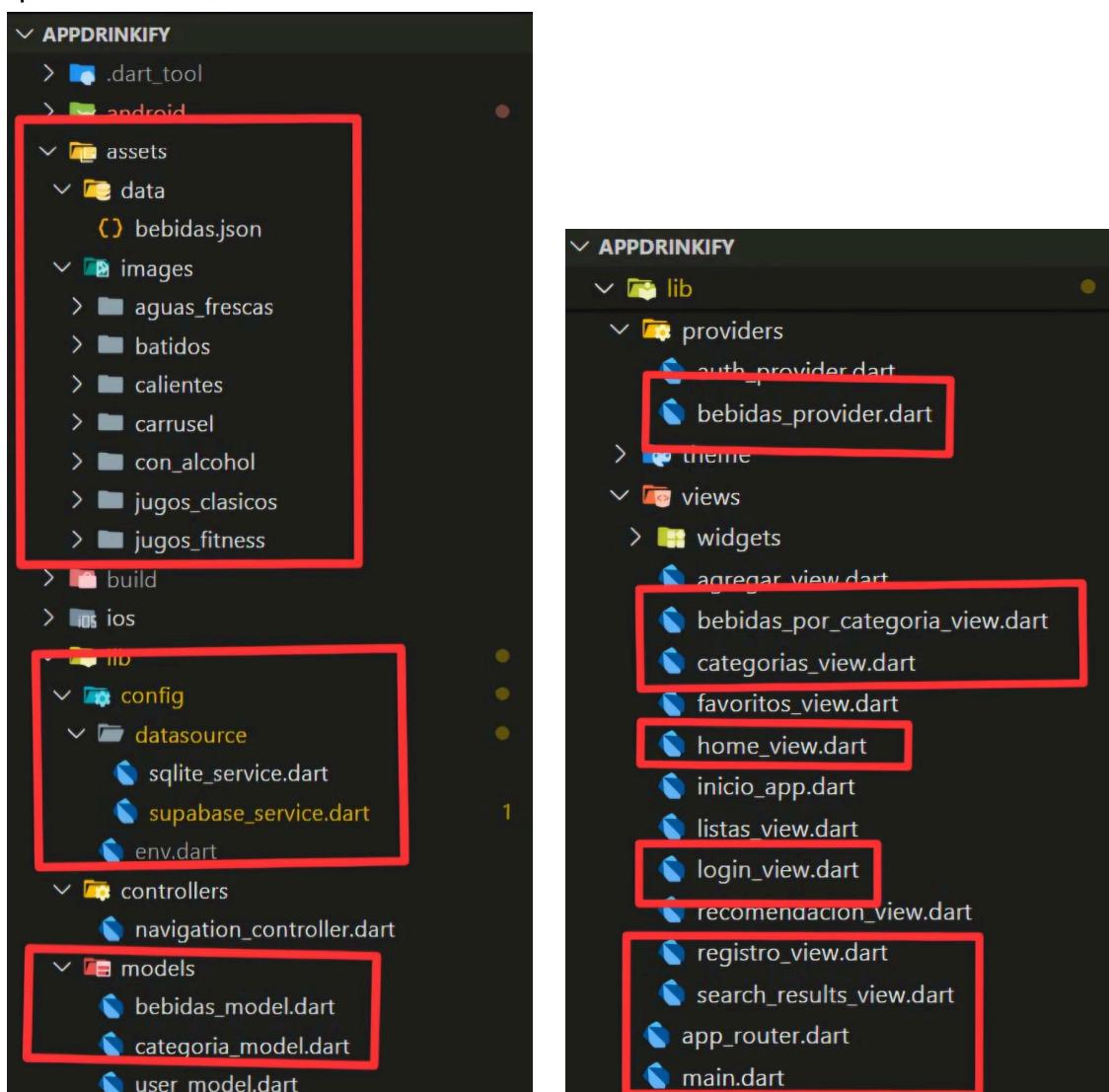
```
1 final username = context.watch<AuthProvider>().currentUser?.username ?? 'Usuario';
2
3
4 return Scaffold(
5   appBar: AppBar(
6     //leading: Icon(Icons.person_rounded, size: 80),
7     title: Text('Bienvenido $username',
8     style: const TextStyle(
9       fontSize: 25,
10      fontWeight: FontWeight.bold,
11    ),
12    ),
13    actions: [
14      IconButton(
15        onPressed:(){
16          final authProvider = Provider.of<AuthProvider>(context, listen: false);
17          authProvider.logout();
18          NavigatorController.navigateTo(context, '/inicio');
19        },
20        icon: const Icon(Icons.logout),
21      ),
22    ],
23  ),
```

Sprint 3

En el sprint 2 se cometieron errores y el profesor nos los hizo ver y encargo solucionarlo para el tercer sprint, el cual se entregó el 07 de noviembre de 2025, las observaciones del profesor fueron las siguientes:

- Validar los textfields para el registro y el login.
- Mostrar el nombre del usuario logueado en la pantalla “Home”.
- Crear una carpeta llamada “config” y guardar ahí las conexiones entrantes y las variables de entorno.

Estos son los archivos y/o carpetas que se crearon y/o modificaron para el segundo sprint:



Assets (carpeta)

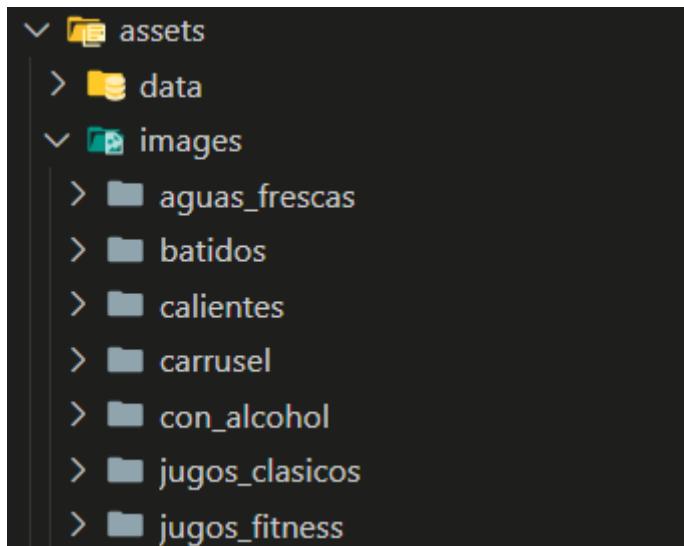
Se creó la carpeta “Assets”, dentro de ella se creó la carpeta “data” e “images”.



En “data” se creó “bebidas.json” que nos sirve para.

```
1  [
2  {
3      "id": 1,
4      "nombre": "Mojito",
5      "categoria": "Cocteles",
6      "descripcion": "Refrescante bebida con hierbabuena y limón",
7      "imagen": "assets/images/bebidas/mojito.jpeg"
8  },
9  {
10     "id": 2,
11     "nombre": "Piña Colada",
12     "categoria": "Cocteles",
13     "descripcion": "Dulce mezcla de piña y coco",
14     "imagen": "assets/images/bebidas/pina_colada.jpeg"
15 }
16 ]
```

En “images”, estamos guardando las imágenes que estamos utilizando para las bebidas.



sqlite_service.dart.

Implementación de un Singleton que gestiona la conexión y operaciones con la base de datos local sqlite. Abstacta el acceso a los datos (DAO) para las entidades Bebida y Categoría.

Patrón: Singleton (.instance) para garantizar una única instancia de Database (_database) durante el ciclo de vida de la app.

Inicialización (_initDB, _createDB):

_initDB localiza la ruta de la base de datos en el dispositivo.

_createDB (ejecutado por onCreate) define el esquema DDL.

Crea la tabla categorías (Padre) y bebidas (Hijo).

Establece una restricción FOREIGN KEY en bebidas.categoria_id que referencia a categorias.id para mantener la integridad relacional.

```
eltrillero33, yesterday | 2 authors (You and one other)
class SqliteService {
    static final SqliteService instance = SqliteService._init();
    static Database? _database;
    SqliteService._init();

    Future<Database> get database async {
        if (_database != null) return _database!;
        _database = await _initDB('bebidas.db');
        return _database!;
    }

    Future<Database> _initDB(String filePath) async {
        final dbPath = await getDatabasesPath();
        final path = join(dbPath, filePath);
        return await openDatabase(path, version: 1, onCreate: _createDB);
    }
}
```

Métodos Principales:

popularDatosIniciales(): Lógica de seeding. Ejecuta un COUNT(*) en ambas tablas y, si están vacías, inserta un conjunto de datos predefinido (Categorías y Bebidas). Utiliza createBebida para las inserciones.

getAllBebidas(): Ejecuta una rawQuery con un JOIN entre bebidas (b) y categorias (c) sobre b.categoria_id = c.id. Esta consulta enriquece el resultado con c.nombre as categoria_nombre, facilitando el mapeo en el modelo Bebida.

getAllCategorias(): Retorna una lista de todas las categorías, ordenadas por nombre.

createBebida(Bebida bebida): Recibe un objeto Bebida, lo serializa (bebida.toMap()) y lo inserta en la tabla bebidas.

```
Future<int> createBebida(Bebida bebida) async {
    final db = await instance.database;
    return await db.insert('bebidas', bebida.toMap());
}

Future<List<Categoria>> getAllCategorias() async {
    final db = await instance.database;
    final result = await db.query('categorias', orderBy: 'nombre ASC');
    return result.map((json) => Categoria.fromMap(json)).toList();
}

Future<List<Bebida>> getAllBebidas() async {
    final db = await instance.database;
    final result = await db.rawQuery('''
        SELECT
            b.id,
            b.nombre,
            b.descripcion,
            b.preparacion,
            b.image_url,
            b.categoria_id,
            c.nombre as categoria_nombre
        FROM bebidas b
        JOIN categorias c ON b.categoria_id = c.id
        ORDER BY b.nombre ASC
    ''');
    return result.map((json) => Bebida.fromMap(json)).toList();
}
```

env.dart

Aquí definimos nuestras variables de entorno para hacer la conexión a supabase.

```
class Env {
  static const supabaseUrl = 'https://tbgdjohrllhrcmwigkv.supabase.co';
  static const supabaseAnonKey ='eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3Mi0iJzdXBhYmFzZSIsInJlZiI6InRiZ2RqZ29ocmxsaHjbXdpZ2t2Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3NjA1Mzg4MjIsImV4cCI6MjA3NjExNDgyMn0
.V6rI0L7f0w8fMB3dFT1S4dkn-nMj0v45tXk3nncoT8g';}
```

bebidas_model.dart

Aquí definimos el modelo para poder insertar bebidas.

```
class Bebida {      You, 4 days ago • S
  final int? id;
  final String nombre;
  final String descripcion;
  final String preparacion;
  final String imageUrl;
  final int categoria_id;

  final String? categoria_nombre;

  Bebida({
    this.id,
    required this.nombre,
    required this.descripcion,
    required this.preparacion,
    required this.imageUrl,
    required this.categoria_id,
    this.categoria_nombre,
  });
}
```

categoria_model.dart

De igual manera, aquí definimos el modelo para la creación de categorías.

```
class Categoria {  
    final int? id;  
    final String nombre;  
  
    Categoria({  
        required this.id,  
        required this.nombre,  
    });
```

bebidas_provider.dart

Gestor de estado para el catálogo de bebidas. Actúa como intermediario entre la UI y el SqliteService, implementando un patrón de caché en memoria.

Estado Interno:

_listaCompletaBebidas (List<Bebida>): Mantiene una caché en memoria de todas las bebidas después de la carga inicial.

_isLoading (bool): Flag de estado para que la UI muestre un loader durante la inicialización.

```
final SqliteService _sqliteService = SqliteService.instance;  
  
List<Bebida> _listaCompletaBebidas = [];  
bool _isLoading = false;  
  
bool get isLoading => _isLoading;  
List<Bebida> get listaCompletaBebidas => _listaCompletaBebidas;  
  
BebidaProvider() {  
    _inicializar();  
}
```

Flujo de Inicialización (_inicializar):

Se llama desde el constructor del Provider.

Establece _isLoading = true y notifica (para mostrar loader).

Ejecuta `_sqliteService.popularDatosIniciales()` (para el seeding).

Carga todas las bebidas usando `_sqliteService.getAllBebidas()` y las almacena en `_listaCompletaBebidas`.

Establece `_isLoading = false` y notifica (para mostrar datos).

```
Future<void> _inicializar() async {
    _isLoading = true;
    notifyListeners();
    await _sqliteService.popularDatosIniciales();
    _listaCompletaBebidas = await _sqliteService.getAllBebidas();
    _isLoading = false;
    notifyListeners();
}

List<Bebida> buscarBebidas(String query) {
    if (query.isEmpty) {
        return [];
    }
}
```

Métodos de UI (Filtros en Memoria):

`buscarBebidas(String query)`: Ejecuta un filtro `.where()` sobre la lista en caché (`_listaCompletaBebidas`). No accede a la base de datos, lo que resulta en búsquedas instantáneas.

`getBebidasPorCategoria(String nombreCategoria)`: Filtra la lista en caché por `categoria_nombre`. Al igual que la búsqueda, opera en memoria para un rendimiento óptimo.

```
List<Bebida> buscarBebidas(String query) {
    if (query.isEmpty) {
        return [];
    }

    final queryMinusculas = query.toLowerCase().trim();
    final resultados = _listaCompletaBebidas.where((bebida) {
        final nombreMinusculas = bebida.nombre.toLowerCase();
        final descripcionMinusculas = bebida.descripcion.toLowerCase();
        final categoriaMinusculas = bebida.categoria_nombre?.toLowerCase() ?? '';
        return nombreMinusculas.contains(queryMinusculas) ||
            descripcionMinusculas.contains(queryMinusculas) ||
            categoriaMinusculas.contains(queryMinusculas);
    }).toList();
    return resultados;
}

List<Bebida> getBebidasPorCategoria(String nombreCategoria) {
    final nombreMinusculas = nombreCategoria.toLowerCase();
    final resultados = _listaCompletaBebidas.where((bebida) {
        final catNombreBebida = bebida.categoria_nombre?.toLowerCase() ?? '';
        return catNombreBebida == nombreMinusculas;
    }).toList();

    return resultados;
}
```

bebidas_por_categoria_view.dart

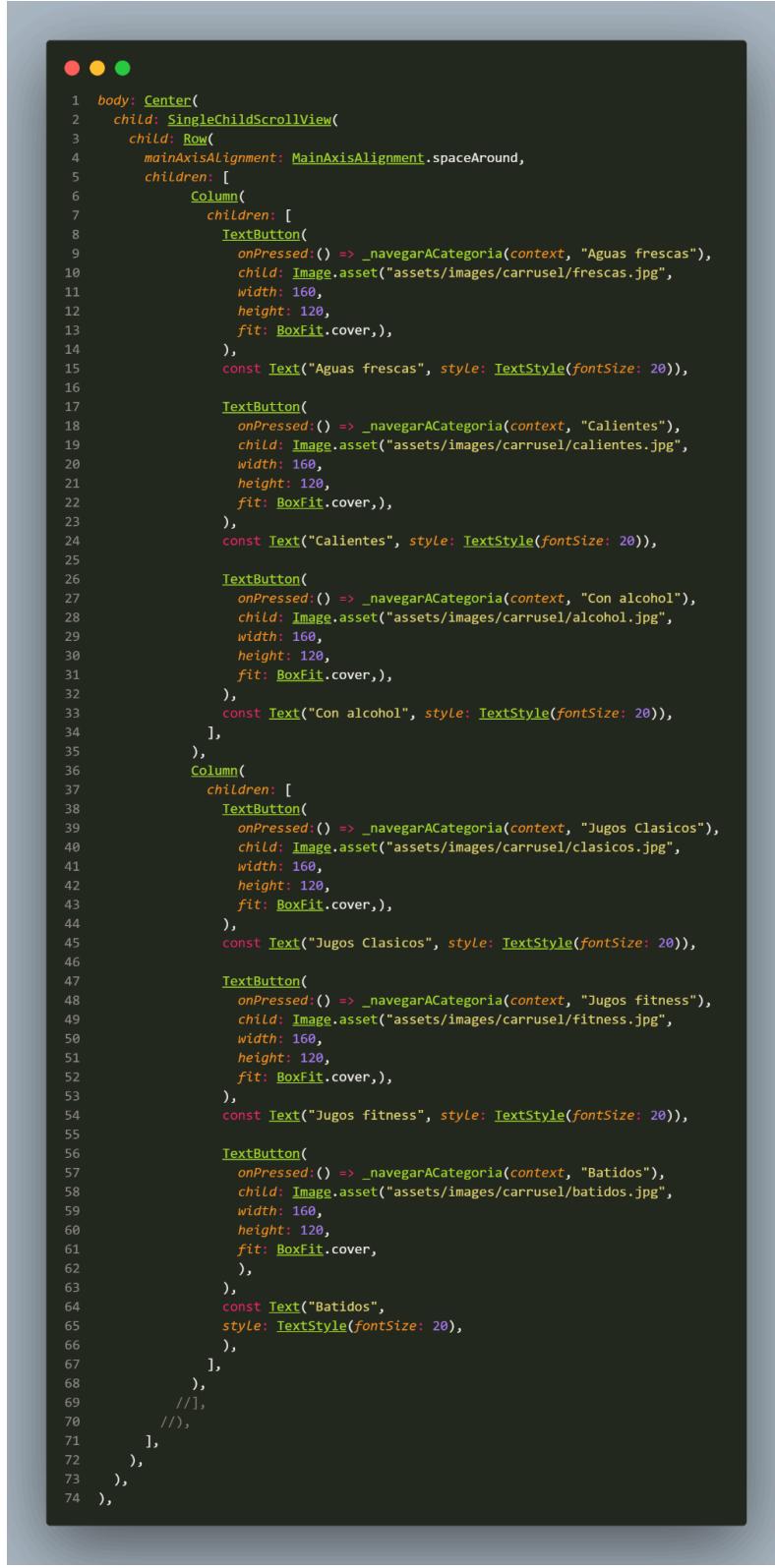
Esta pantalla recibe y muestra una lista de bebidas (bebidas) que pertenecen a una categoría específica (categoriaNombre). El nombre de la categoría se usa como título en el AppBar. El body es muy similar al de search_results_view: comprueba si la lista de bebidas está vacía (mostrando un mensaje de "No hay bebidas en esta categoría.") o, de lo contrario, usa un ListView.builder para mostrar cada bebida en un ListTile (con su imagen, nombre y descripción).

```
● ● ●
```

```
1 // lib/views/bebidas_por_categoria_view.dart
2 import 'package:flutter/material.dart';
3 import 'package:appdrinkify/models/bebidas_model.dart';
4
5 class BebidasPorCategoriaView extends StatelessWidget {
6   final List<Bebida> bebidas;
7   final String categoriaNombre;
8
9   const BebidasPorCategoriaView({
10     super.key,
11     required this.bebidas,
12     required this.categoriaNombre,
13   });
14
15   @override
16   Widget build(BuildContext context) {
17     return Scaffold(
18       appBar: AppBar(
19         title: Text(categoriaNombre),
20       ),
21       body: bebidas.isEmpty
22           ? const Center(
23               child: Text("No hay bebidas en esta categoría."),
24           )
25           : ListView.builder(
26             itemCount: bebidas.length,
27             itemBuilder: (context, index) {
28               final bebida = bebidas[index];
29
30               return ListTile(
31                 Leading: Image.asset(
32                   bebida.imageUrl,
33                   width: 50,
34                   height: 50,
35                   fit: BoxFit.cover,
36                   errorBuilder: (context, error, stackTrace) =>
37                     Icon(Icons.no_photography, color: Colors.grey),
38               ),
39                 title: Text(bebida.nombre),
40                 subtitle: Text(
41                   bebida.descripcion,
42                   maxLines: 1,
43                   overflow: TextOverflow.ellipsis,
44                 ),
45                 onTap: () {
46                 },
47               );
48             },
49           ),
50         );
51   }
52 }
```

categorias_view.dart

En esta pantalla lo único que se modificó fue el “Row”, fue una observación del profesor, se eliminó un “column” que contenía el “row” y dentro de ese venían dos “column”.

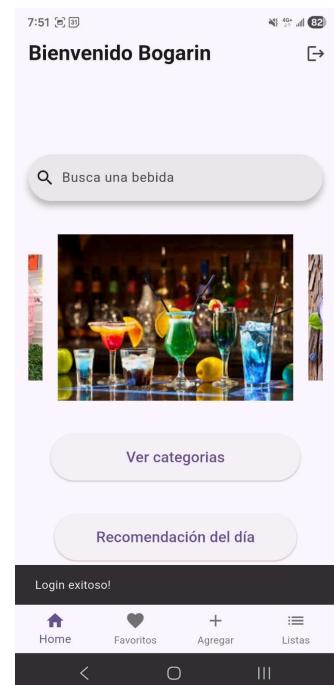


Así se ve la pantalla ya ejecutada:



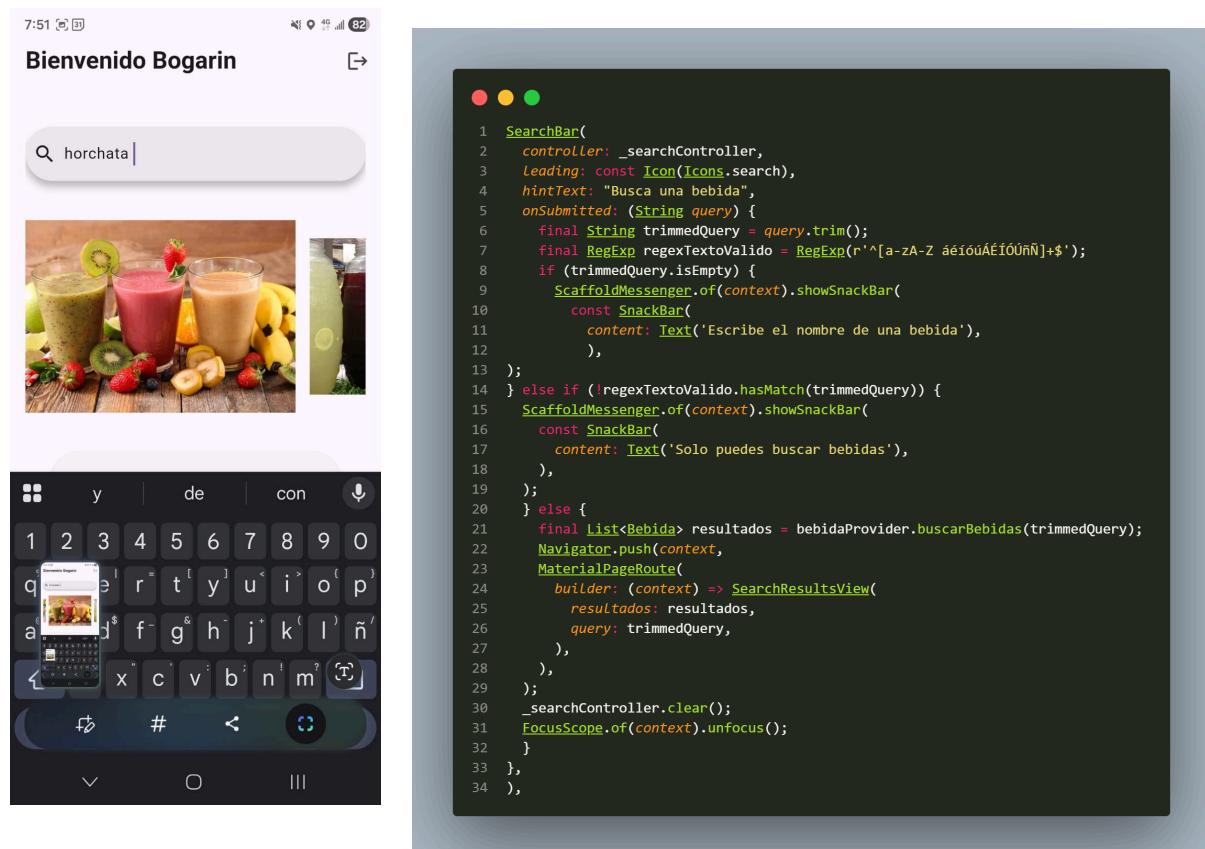
home_view.dart

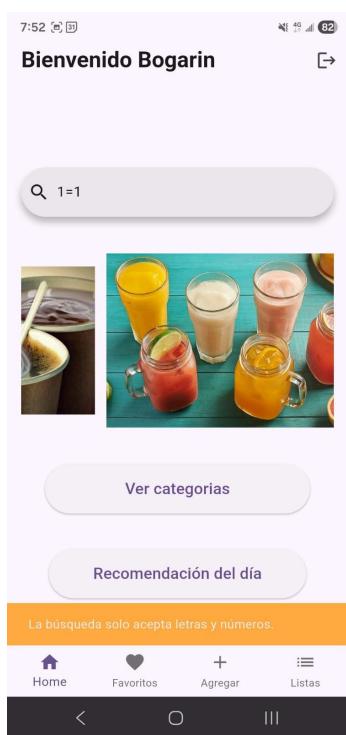
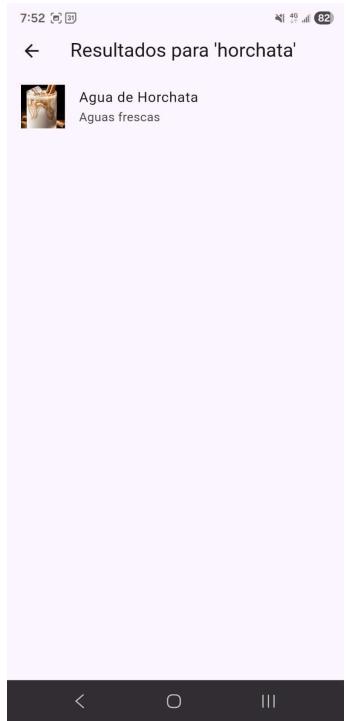
En esta pantalla el problema en el sprint anterior era que no mostraba el nombre del usuario logueado, no se hicieron modificaciones al código, solo funcionó al seguir haciendo pruebas.



Se agregó la funcionalidad a la barra de búsqueda: En la barra de búsqueda (SearchBar), se implementó la lógica de validación y navegación en el evento onSubmit, que se dispara cuando el usuario confirma la búsqueda, se captura el texto ingresado (query) y se limpia usando .trim() para eliminar espacios en blanco

al inicio y al final, se define una expresión regular (RegExp) que solo permite letras, espacios y acentos. Se revisa si la consulta (trimmedQuery) está vacía (isEmpty). Si lo está, se muestra un Snackbar (usando ScaffoldMessenger), si la consulta no está vacía, se valida con la expresión regular. Si el texto no coincide se muestra un snackBar. Si la consulta pasa ambas validaciones, se llama al método buscarBebidas() del bebidaProvider, pasándole la consulta limpia (trimmedQuery) y el resultado (la List<Bebida>) se almacena en la variable resultados. Se usa Navigator.push para navegar a la nueva pantalla SearchResultsView. Es importante notar que a esta nueva pantalla se le pasan los datos que necesita: la lista de resultados y el query (el texto que buscó el usuario) y se limpia el controlador de la barra (_searchController.clear()) y se oculta el teclado (FocusScope.of(context).unfocus()) para que la interfaz quede limpia al regresar.





registro_view.dart

Atendiendo a la observación de validar los campos, se agregaron Expresiones Regulares (Regex). Antes de intentar el registro, el código ahora comprueba secuencialmente:

- Email (emailRegex): Valida que el texto tenga un formato de correo estándar.
- Usuario (userRegex): Asegura que el nombre empiece con una letra y tenga entre 3 y 20 caracteres (permitiendo letras, números, _ o -).

- Contraseña (passRegex): Verifica que tenga un mínimo de 8 caracteres. Si alguna de estas validaciones falla (!hasMatch), se muestra un Snackbar con el mensaje de error correspondiente y se detiene la función con return, impidiendo que se intente el registro con datos incorrectos.

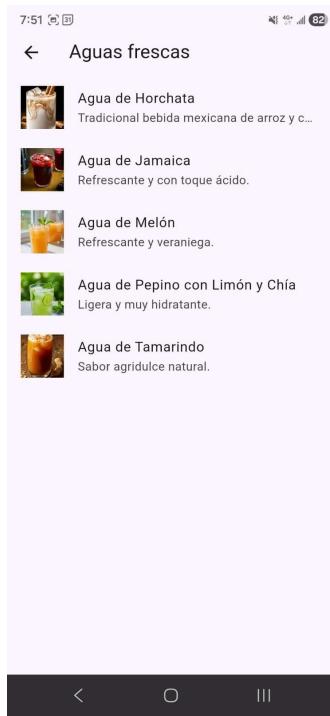


```
1 final emailRegex =
2     RegExp(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}+$');
3 if (!emailRegex.hasMatch(email)) {
4     ScaffoldMessenger.of(context).showSnackBar(
5         const SnackBar(content: Text('Por favor, ingresa un correo válido')),
6     );
7     return;
8 }
9
10 // 2. Validar Usuario
11 // Debe empezar con letra
12 // Puede contener letras, números, _ o -
13 // Longitud total entre 3 y 20 caracteres
14 final userRegex = RegExp(r'^[a-zA-Z][a-zA-Z0-9_-]{2,19}$');
15 if (!userRegex.hasMatch(username)) {
16     ScaffoldMessenger.of(context).showSnackBar(
17         const SnackBar(
18             content:
19                 Text('Usuario inválido (3-20 caracteres, debe empezar con letra)')),
20     );
21     return;
22 }
23
24 // 3. Validar Contraseña
25 // Acepta cualquier carácter
26 // Longitud mínima de 8 caracteres
27 final passRegex = RegExp(r'^.{8,}$');
28 if (!passRegex.hasMatch(password)) {
29     ScaffoldMessenger.of(context).showSnackBar(
30         const SnackBar(
31             content: Text('La contraseña debe tener al menos 8 caracteres')),
32     );
33 }
```

Así se ve la pantalla ya ejecutada:



search_result_view.dart



En esta pantalla se muestran los resultados de la búsqueda, recibe y muestra los resultados de una búsqueda. En su constructor, requiere la lista de resultados (bebidas) y el query (texto buscado), el cual muestra en el título del AppBar. El body comprueba si la lista de resultados está vacía; si lo está, muestra el texto "No se encontraron bebidas." Si hay resultados, usa un ListView.builder para mostrar cada bebida en un ListTile, el cual incluye la imagen (Image.asset), el nombre (title) y la categoría (subtitle) de la bebida.

```
● ● ●
```

```
1 import 'package:flutter/material.dart';
2 import 'package:appdrinkify/models/bebidas_model.dart';
3
4 class SearchResultsView extends StatelessWidget {
5   final List<Bebida> resultados;
6   final String query;
7
8   const SearchResultsView({
9     super.key,
10    required this.resultados,
11    required this.query,
12  });
13
14   @override
15   Widget build(BuildContext context) {
16     return Scaffold(
17       appBar: AppBar(
18         title: Text("Resultados para '$query'"),
19       ),
20       body: resultados.isEmpty
21         ? const Center(
22             child: Text("No se encontraron bebidas."),
23           )
24         : ListView.builder(
25           itemCount: resultados.length,
26           itemBuilder: (context, index) {
27             final bebida = resultados[index];
28
29             return ListTile(
30               Leading: Image.asset(
31                 bebida.imageUrl,
32                 width: 50,
33                 height: 50,
34                 fit: BoxFit.cover,
35                 errorBuilder: (context, error, stackTrace) =>
36                   Icon(Icons.no_photography, color: Colors.grey),
37               ),
38               title: Text(bebida.nombre),
39               subtitle: Text(
40                 bebida.categoría_nombre ?? 'Sin categoría',
41                 maxLines: 1,
42                 overflow: TextOverflow.ellipsis,
43               ),
44               onTap: () {},
45             );
46           },
47         ),
48       );
49   }
50 }
```

app_router.dart

En este archivo lo único que se modificó fue agregar las nuevas pantallas para redirigirlo.