Flutter Bootcamp

Navigator & GoRouter

Emanuel López – emanuel.lopez@globant.com

Globant

Repaso

- Clean Architecture
- `WidgetsFlutterBinding.ensureInitialized()`
- `WidgetsBinding.instance.addPostFrameCallback()`
- Mixin `WidgetsBindingObserver`
- Barrel files
- Librerías o paquetes
  - `Shared_preferences`, `http`, `dio`
  - `Cached_network_image`, `flutter_launcher_icons`
  - `intl`, `lottie`, `sqflite` y `path`

Instalación Flutter SDK:
youtube.com/playlist?list=PLutrh4gv1YI96buJoWiyG98qnBG26M9U9

Globant

Navigator

Se utiliza para movernos entre distintas pantallas dentro de una aplicación. Se basa en el uso de una pila o stack de rutas.

Cuando navegamos a una nueva pantalla, se inserta un nuevo `Route` en la pila de Navigator y, cuando navegamos hacia atrás, se elimina el `Route` actual.

`push()`: este método agrega una ruta a la parte superior de la pila del navegador, lo que da como resultado la navegación a una nueva pantalla. La nueva ruta se convierte en la pantalla activa de la aplicación.

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => const AnotherScreen()),
);
```

Navigator

`pop()`: este método elimina la ruta actual de la parte superior de la pila y regresa a la pantalla anterior. Opcionalmente, puede devolver datos a la ruta anterior.

```
Navigator.pop(context);
Navigator.pop(context, true);
```

```
static void pop<T extends Object?>(
  BuildContext context,
  [ T? result ]) => Navigator.of(context).pop<T>(result);
```

Navegación a una pantalla que devuelve un resultado:

```
final result = await Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => const AnotherScreen()),
);
```

Navigator

`pushReplacement()`: reemplaza la ruta actual en la pila con una nueva ruta. Útil para situaciones como el inicio de sesión, donde desea reemplazar la pantalla de inicio de sesión con la pantalla de inicio.

```
Navigator.pushReplacement(
  context,
  MaterialPageRoute(builder: (context) => HomeScreen()),
);
```

`popUntil()`: extrae rutas de la pila hasta que se alcanza una ruta con un nombre determinado.

```
Navigator.popUntil(context, ModalRoute.withName('/'));
```

Navigator

`pushAndRemoveUntil()`: inserta una nueva ruta en la pila y elimina todas las rutas anteriores hasta que el predicado devuelva verdadero.

```
Navigator.pushAndRemoveUntil(
  context, MaterialPageRoute(builder: (context) => HomeScreen()),
  (Route<dynamic> route) => false,// Remove all routes beneath
);

Navigator.pushAndRemoveUntil(
  ..., ...,(Route<dynamic> route) => route.isFirst,
);

Navigator.pushAndRemoveUntil(
  ..., ..., ModalRoute.withName('/home'),
);
```

Navigator & Named Routes

Las named routes es una forma de gestionar la navegación de nuestra aplicación usando identificadores (o nombres) en lugar de tener que definir directamente cada ruta con `MaterialPageRoute`.

Definimos todas las rutas al inicio de la aplicación en el parámetro `routes` de `MaterialApp` y luego navegamos entre ellas usando sus nombres usando un mapa clave-valor donde la clave es el nombre de la ruta y el valor es la pantalla asociada.

También podremos definir la ruta inicial (`initialRoute`) que se cargará por defecto al iniciar la aplicación. Por lo general, la pantalla principal o de bienvenida tiene la ruta `'/'`.

Navigator & Named Routes

```
MaterialApp(
  title: 'Soccer Scores App',
  initialRoute: '/',
  routes: {
    '/': (context) => const SplashScreen(),
    '/login': (context) => const LoginScreen(),
    '/signup': (context) => const SignUpScreen(),
    '/home': (context) => const MatchListScreen(),
    '/details': (context) => const MatchDetailsScreen(),
    '/profile': (context) => const UserProfileScreen(),
    '/settings': (context) => const SettingsScreen(),
  },
);
```

Navigator & Named Routes

Ventajas de las named routes:

- **Código más limpio:** Al usar nombres en lugar de definir la ruta completa cada vez, el código de navegación es más simple y fácil de mantener.
- **Organización centralizada:** Todas las rutas se definen en un solo lugar, lo que facilita la gestión de las rutas y su mantenimiento en aplicaciones grandes.
- **Reutilización:** Puedes usar la misma ruta desde cualquier parte de la aplicación sin tener que redefinirla.

Navigator & Named Routes

`pushNamed()`: se utiliza para navegar a una ruta con nombre. Las rutas suelen declararse en la tabla de rutas de `MaterialApp`.

```
Navigator.pushNamed(context, '/details');
```

`popAndPushNamed()`: extrae la ruta actual de la pila y navega hacia una ruta con nombre.

```
Navigator.popAndPushNamed(context, '/details');
```

`pushReplacementNamed()`: similar a `pushReplacement`, pero utiliza una ruta con nombre.

```
Navigator.pushReplacementNamed(context, '/home');
```

Navigator 2.0

`Navigator 1.0` sigue un **modelo imperativo**, el desarrollador indica explícitamente cuándo navegar a una nueva pantalla o cuándo regresar a la anterior utilizando métodos como `push` o `pop`. Esto funciona bien para aplicaciones simples, pero resulta problemático para aplicaciones más complejas que requieren una navegación más flexible y personalizada.

`Navigator 2.0` surgió como una solución a las siguientes limitaciones de `Navigator 1.0`:

- **Falta de control total sobre la pila de navegación:** Es complejo modificar la pila de navegación
- **Dificultad con el manejo de la URL en la web:** No hay una forma sencilla de sincronizar la URL del navegador con el estado de la navegación de la aplicación
- **Navegación dependiente del contexto**

Navigator 2.0

`Navigator 2.0` introduce un **enfoque declarativo** para la navegación, que permite tener un mayor control sobre el estado de la navegación y manejar la navegación de manera más flexible y adaptada a aplicaciones complejas, como las que pueden tener múltiples flujos de navegación.

Aunque `Navigator 2.0` resolvió muchas limitaciones y ofreció un control más granular sobre la navegación, su complejidad también hizo que los desarrolladores buscaran alternativas más simples, centradas en casos de uso comunes.

Estos paquetes ayudan a abstraer parte de esa complejidad, ofreciendo soluciones más accesibles, menos verbosas y más específicas para las necesidades de las aplicaciones modernas.

Navigator 2.0

Si revisamos la documentación oficial actual en flutter.dev, el uso de Navigator 2.0 se explica mediante la librería/paquete de terceros `go_router`.

A partir de la versión 3.0.2 (febrero de 2022), `go_router` comienza a ser publicado y mantenido en el github oficial de Flutter.

- Repositorio original: github.com/csells/go_router
- Repositorio actual: github.com/flutter/packages/tree/main/packages/go_router

Fuente: twitter.com/RydMike/status/1596645324793446400

Globant

GoRouter

```
MaterialApp.router(
  routerConfig: GoRouter(
    routes: [
      GoRoute(
        path: '/',
        builder: (context, state) => ItemListScreen(),
      ),
      GoRoute(
        path: '/details,
        builder: (context, state) => ItemDetailsScreen(),
      ),
    ],
  );
);
```

GoRouter

```
GoRoute(
  name: 'details',
  path: '/match/:id',
  builder: (context, state) => MatchDetailsScreen(
    id: state.pathParameters['id']!,
  ),
),
```

El parámetro `path` define la URL que se muestra en la barra de direcciones del navegador (para web) o la ruta que se usa en la aplicación para navegar entre pantallas.

El parámetro `name` es un identificador simbólico de la ruta. Se usa para referirse a esa ruta de manera más cómoda y segura cuando se navega.

GoRouter

`context.go() / goNamed()`: Navega a una nueva ruta reemplazando la ruta actual en la pila de navegación. Al usar `go()`, la ruta actual se elimina y no podremos regresar a ella usando el botón de retroceso. Sería el `pushAndRemoveUntil` del Navigator 1.0

Ejemplo: Navegación desde la pantalla de login o registro al home. Una vez finalizado el login/registro, no queremos volver hacia atrás

`context.push() / pushNamed()`: Navega a una nueva ruta agregando esta ruta a la pila de navegación. En este caso si podremos regresar a la ruta anterior utilizando el botón de retroceso, ya que esta ruta permanecerá en la pila.

Ejemplo: Navegación desde una pantalla donde se muestra una lista de items hacia la pantalla del detalle de un item. Podemos volver a la pantalla anterior para seguir revisando otros items.

GoRouter

```
context.pushNamed(
  'name', pathParameters: { … }, queryParameters: { … }, extra: extra,
),
```

`pathParameters:` Son valores que se incluyen directamente en la ruta. Se definen usando una sintaxis específica que incluye dos puntos (:) antes del nombre del parámetro. Ideales para datos que son parte fundamental de la ruta, como identificadores.

`queryParameters`: Son pares clave-valor que se agregan al final de la URL después de un signo de interrogación (?). Se utilizan para datos opcionales que pueden modificar el comportamiento de la ruta, pero que no son parte de la estructura de la URL.

`extra` es un objeto que puedes pasar a la ruta como un dato adicional que no forma parte de los parámetros de ruta o de consulta. Este objeto se puede utilizar para pasar información compleja o tipos de datos que no se pueden representar fácilmente en la URL.

build_runner

Es una herramienta de generación de código para aplicaciones Dart y Flutter.

Casos de uso comunes:

- **Serialización de JSON:** Generación de código para convertir objetos a y desde JSON (ejemplo: usando `json_serializable`).
- **Inyección de Dependencias:** Generación de código para administrar la inyección de dependencias (ejemplo: usando `injectable`).
- **Clases Inmutables:** Creación de clases inmutables y generación de métodos como copyWith (ejemplo: usando `freezed`).
- **Génesis de Código de Pruebas:** Automatización de la creación de pruebas y mocks (ejemplo: usando `mockito`).
- **Generación de Configuraciones:** Generación de archivos de configuración y clases a partir de archivos de configuración personalizados.

build_runner

Ejemplo de uso: JSON serialization and deserialization.

```
flutter pub add
      json_annotation
      dev:build_runner
      dev:json_serializable
```

Generación de código:

```
flutter pub run build_runner build
```

Generación de código continua:

```
flutter pub run build_runner watch
```

Globant

Algunas librerías/paquetes interesantes

- `flutter_secure_storage`: Librería para almacenar datos de forma segura y persistente en dispositivos móviles.
- `flutter_native_splash`: Librería para crear splash screens nativas. Permite personalizar la apariencia de la pantalla que se muestra mientras la aplicación se está cargando.
- `url_launcher`: Se utiliza para abrir URLs en un navegador web o en aplicaciones externas. Permite agregar funcionalidades como links a páginas web, hacer llamadas telefónicas, enviar correos electrónicos, etc.
- `google_fonts`: Se utiliza para aplicar fuentes personalizadas de Google Fonts en nuestras aplicaciones.
- `shimmer`: Se utiliza para crear efectos de carga visualmente atractivos, conocidos como efecto shimmer

Links adicionales

- Navigation and routing: docs.flutter.dev/ui/navigation
- Using Flutter Navigator 2.0 With Routes: medium.com/@lkay_codes/using-flutter-navigator-2-0-with-routes-1e8b2dcb0e84
- go_router documentation: pub.dev/documentation/go_router/latest/index.html
- Navigation and routing example app: flutter.github.io/samples/navigation_and_routing.html

Preguntas?

Globant

Gracias!

Globant