Apakah Navigator 2.0 selalu lebih baik daripada Navigator 1.0?

Jika Anda memiliki proyek yang sudah ada, Anda tidak perlu memigrasikan atau mengonversi kode yang ada untuk menggunakan API baru.

Berikut adalah beberapa tips untuk membantu Anda memutuskan mana yang lebih berguna untuk Anda:

- **Untuk aplikasi sedang hingga besar**: Pertimbangkan untuk menggunakan API deklaratif dan widget router. Anda mungkin harus mengelola banyak status navigasi Anda.
- **Untuk aplikasi kecil**: Untuk pembuatan prototipe cepat atau pembuatan aplikasi kecil untuk demo, API imperatif cocok. Terkadang hanya push dan pop yang Anda butuhkan!

Selanjutnya, Anda akan mendapatkan pengalaman langsung dengan Navigator 2.0.

> **Catatan**: Bab ini akan fokus pada penerapan Navigator 2.0. Untuk mempelajari lebih lanjut tentang Navigator 1.0, periksa:
>
> **Tutorial Buku Masak Dev Flutter**: https://flutter.dev/docs/cookbook/navigasi.
>
> **Navigasi Flutter: Memulai oleh Filip Babić**: https://www.raywenderlich.com/4562634-flutter-navigation-getting-started.

Mulai

Buka proyek pemula di Android Studio, jalankan bergetar pub dapatkan, lalu jalankan aplikasinya.

> **Catatan**: Lebih baik memulai dengan proyek awal daripada melanjutkan proyek dari bab terakhir karena proyek awal berisi beberapa perubahan khusus untuk bab ini.

Anda akan melihat bahwa **makanan** aplikasi hanya menampilkan layar Splash.

Jangan khawatir, Anda akan segera menghubungkan semua layar. Anda akan membuat alur sederhana yang menampilkan layar masuk dan widget orientasi sebelum menampilkan aplikasi berbasis tab yang telah Anda buat sejauh ini. Tapi pertama-tama, Anda akan melihat perubahan pada file proyek.

Perubahan pada file proyek

Sebelum Anda mendalami navigasi, ada file baru di proyek awal ini untuk membantu Anda.

Di dalam **main.dart**, makanan sekarang adalah widget stateful. Ini akan mendengarkan perubahan status dan membangun kembali widget yang sesuai.

makanan sekarang mendukung pengaturan pengguna untuk mode gelap.

Apa yang baru di folder layar

Ada delapan perubahan baru di **lib/layar/**:

- **splash_screen.dart**: Mengonfigurasi layar Splash awal.
- **login_screen.dart**: Memungkinkan pengguna untuk masuk.
- **onboarding_screen.dart**: Memandu pengguna melalui serangkaian langkah untuk mempelajari lebih lanjut tentang aplikasi.
- **profile_screen.dart**: Memungkinkan pengguna untuk memeriksa profil mereka, memperbarui pengaturan, dan keluar.
- **home.dart**: Sekarang termasuk **Profil** tombol di kanan atas bagi pengguna untuk melihat profil mereka.
- **layar.dart**: File barel yang mengelompokkan semua layar menjadi satu impor.

Nanti, Anda akan menggunakan ini untuk membuat alur UI autentikasi Anda.

Perubahan ke folder model

Ada beberapa perubahan pada file di **lib/model/**.

tab_manager.dart telah dihapus. Sebagai gantinya, Anda akan mengelola pemilihan tab pengguna di**app_state_manager.dart**, yang akan segera Anda buat.

Selain itu, ada tiga objek model baru:

- **fooderlich_pages.dart**: Menjelaskan daftar kunci unik untuk setiap halaman.
- **pengguna.dart**: Menjelaskan satu pengguna. Termasuk informasi seperti peran pengguna, gambar profil, nama lengkap, dan pengaturan aplikasi.
- **profile_manager.dart**: Mengelola status profil pengguna dengan, misalnya, mendapatkan info pengguna, memeriksa apakah pengguna melihat profilnya dan menyetel mode gelap.

Aset tambahan

aset/contoh_data/ berisi data tiruan berikut:

- **sample_explore_recipes.json**, **sample_friends_feed.json** dan **sample_recipes.json**: Ini semua termasuk id field, untuk memberi setiap ubin yang ditampilkan sebuah kunci unik.
- JelajahiRecipe, SimpleRecipe dan Pos juga termasuk tambahan id fitua.

aktiva/ berisi gambar baru, yang akan Anda gunakan untuk membuat panduan orientasi baru.

paket baru

Ada dua paket baru di **pubspec.yaml**:

```
indikator_halaman_halus: ^0.2.3
tampilan web_flutter: ^2.0.7
```

Inilah yang mereka lakukan:

- **indikator_halaman_halus**: Menampilkan indikator halaman saat Anda menggulir halaman.
- **tampilan web_flutter**: Menyediakan sebuah Tampilan Web widget untuk menampilkan konten web di platform iOS atau Android.

Versi Android SDK

Jika Anda membuka **android/app/build.gradle** Anda akan melihat bahwa minSdkVersion adalah sekarang 19, seperti yang ditunjukkan di bawah ini:

```
android {
    konfigurasi default {
        . . .
        minSdkVersion 19
        . . .
    }
}
```

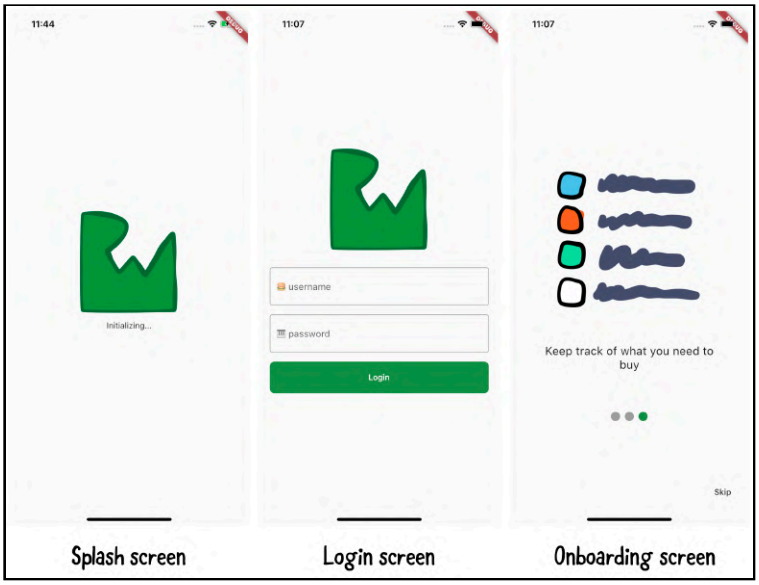
Ini karena webview_flutter bergantung pada Android SDK 19 atau lebih tinggi untuk mengaktifkan komposisi hybrid.

> **Catatan**: Untuk informasi lebih lanjut, lihat dokumentasi webview_flutter https://pub.dev/packages/webview_flutter

Sekarang setelah Anda mengetahui apa yang berubah, Anda akan mendapatkan gambaran singkat tentang alur UI yang akan Anda buat dalam bab ini.

Melihat alur UI

Berikut adalah tiga layar pertama yang Anda tunjukkan kepada pengguna:

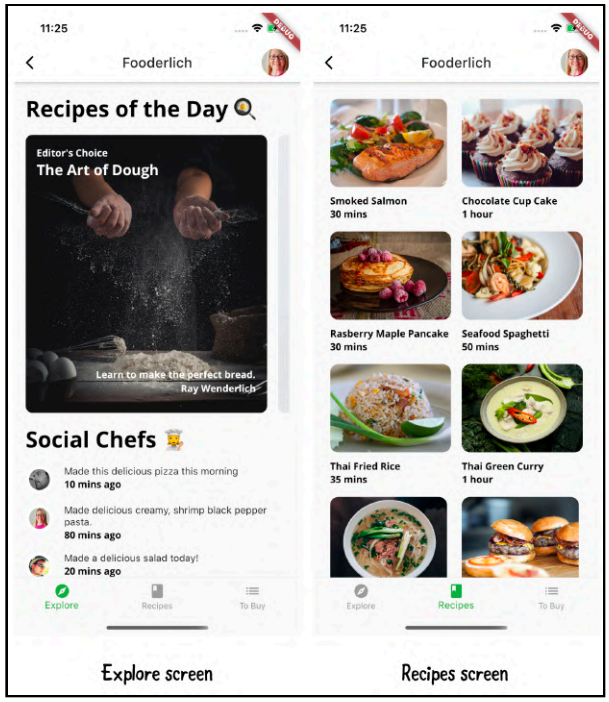


1. Saat pengguna meluncurkan aplikasi, layar pertama yang akan mereka lihat adalah **Layar percikan**. Ini memberi pengembang kesempatan untuk menginisialisasi dan mengonfigurasi aplikasi.

2. Setelah diinisialisasi, pengguna menavigasi ke **Layar login**. Pengguna sekarang harus memasukkan**nama pengguna** dan **kata sandi**, lalu ketuk **Gabung**.

3. Setelah pengguna masuk, dan **Layar orientasi** menunjukkan kepada mereka cara menggunakan aplikasi. Pengguna memiliki dua pilihan: geser melalui panduan untuk mempelajari lebih lanjut tentang aplikasi atau lewati.

Dari layar Orientasi, pengguna pergi ke aplikasi **Rumah**. Mereka sekarang dapat mulai menggunakan aplikasi.



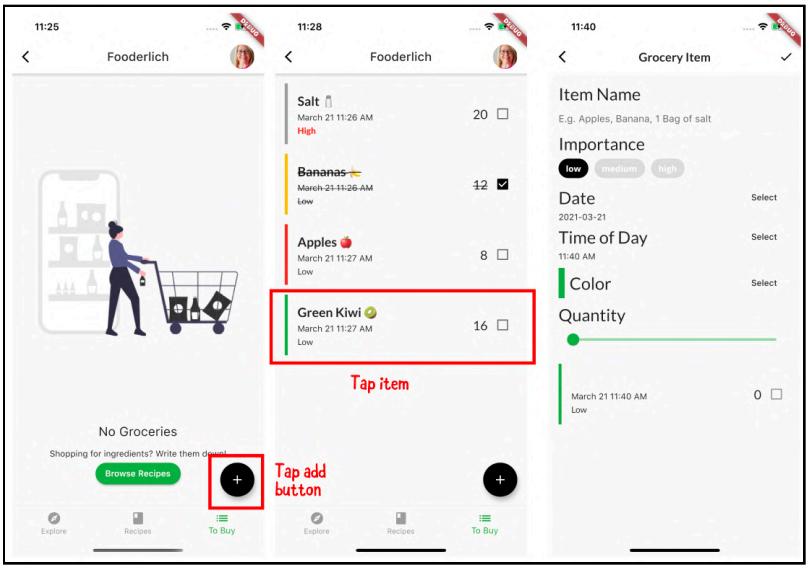
Aplikasi ini memberi pengguna tiga tab dengan opsi ini:

1. **Mengeksplorasi**: Lihat resep untuk hari itu dan lihat apa yang sedang dimasak teman mereka.

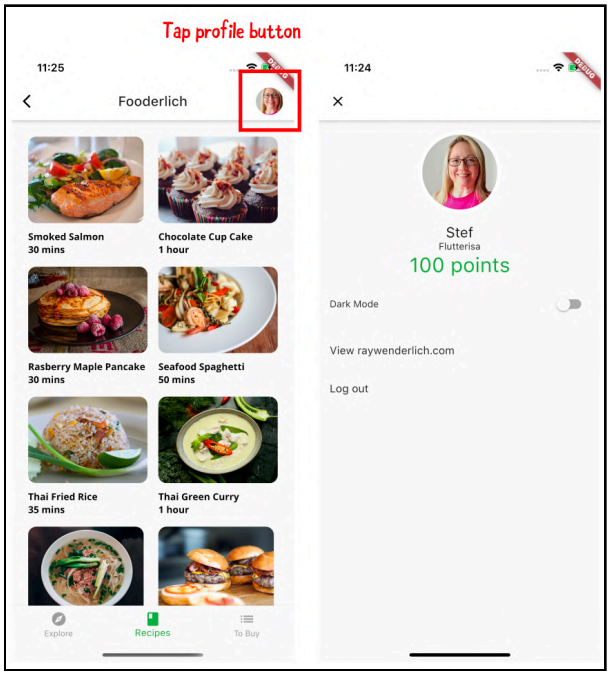
2. **resep**: Menelusuri kumpulan resep yang ingin mereka masak.

3. **Untuk membeli**: Tambahkan bahan atau item ke daftar belanjaan mereka.

Selanjutnya, pengguna dapat mengetuk **Menambahkan** atau, jika daftar belanjaan tidak kosong, mereka dapat mengetuk item yang ada. Ini akan menyajikan**Barang Kelontong** layar, seperti gambar di bawah ini:



Sekarang, bagaimana pengguna melihat profil mereka atau keluar? Mereka mulai dengan mengetuk avatar profil, seperti yang ditunjukkan di bawah ini:



pada **Profil** layar, mereka dapat melakukan hal berikut:

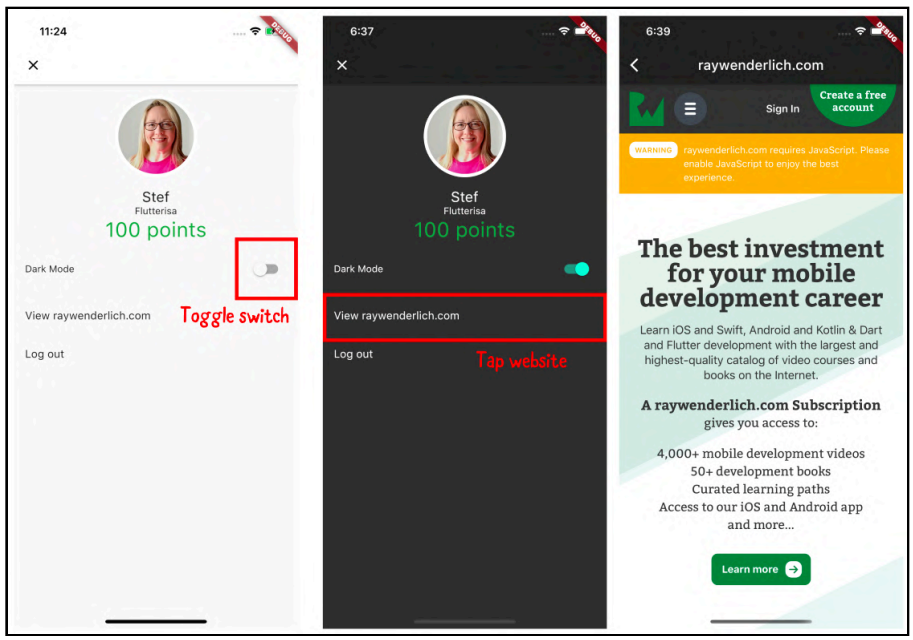
• Lihat profil mereka dan lihat berapa banyak poin yang mereka peroleh.

• Ubah tema aplikasi ke mode gelap.

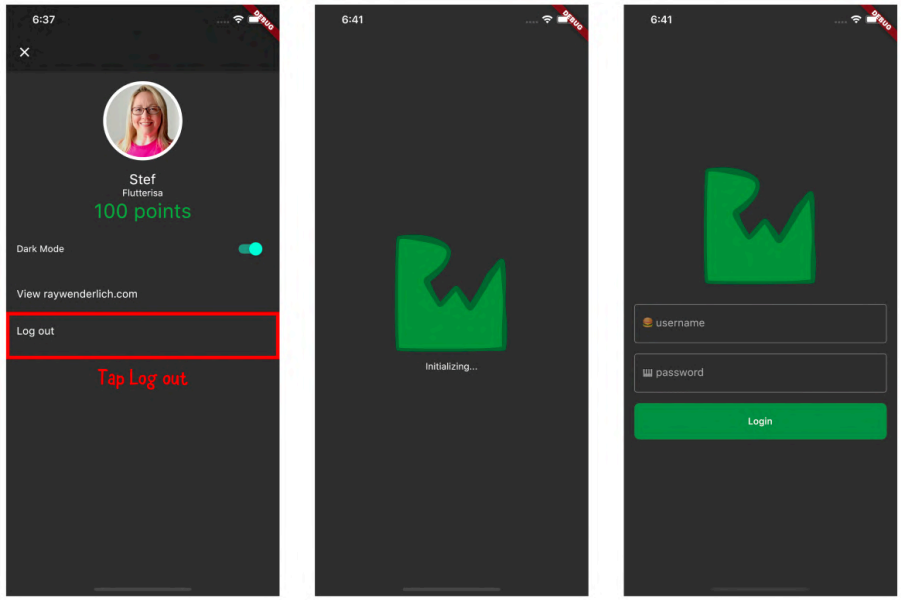
• Kunjungi situs web raywenderlich.com.

• Keluar dari aplikasi.

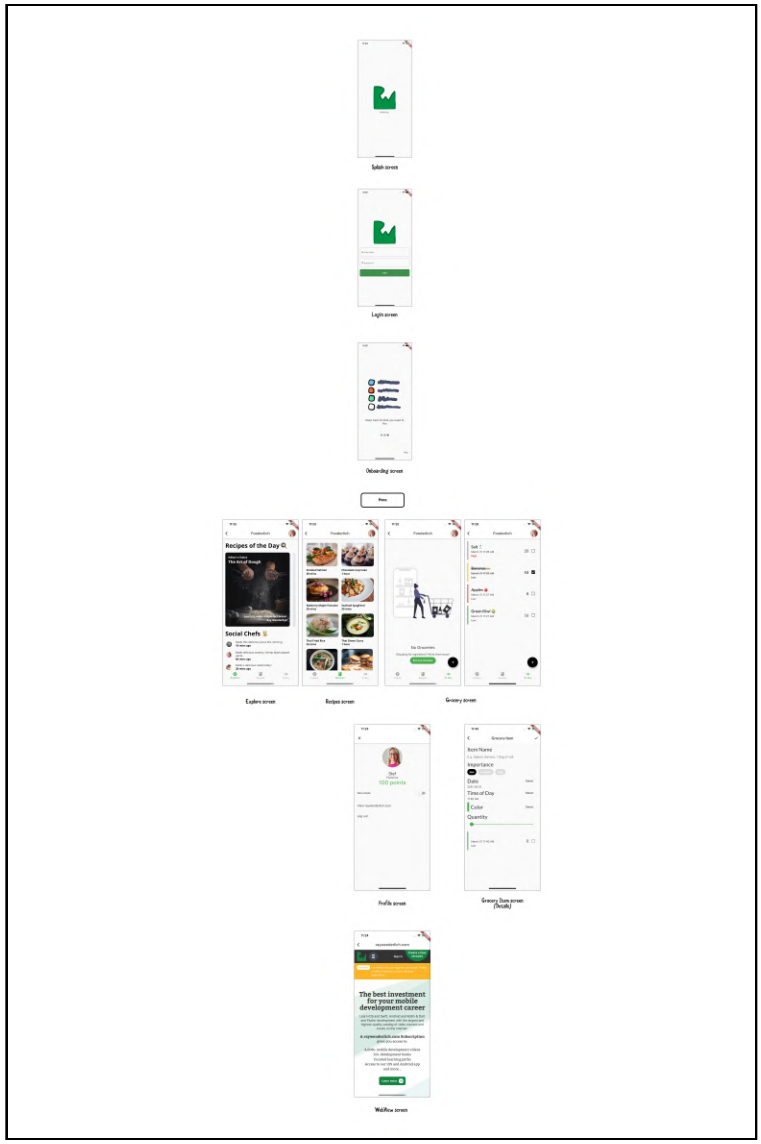
Di bawah ini adalah contoh pengguna yang mengaktifkan mode gelap dan kemudian membuka raywenderlich.com.



Saat Anda mengetuk **Keluar**, itu menginisialisasi ulang aplikasi dan masuk ke layar Login, seperti yang ditunjukkan di bawah ini:



Berikut adalah pandangan sekilas dari seluruh hierarki navigasi:



> **Catatan**: Ada versi gambar skala besar di **aktiva** folder materi bab ini.

Aplikasi Anda akan menjadi luar biasa setelah selesai. Sekarang, saatnya menambahkan beberapa kode!

Mengelola status aplikasi Anda

Langkah pertama adalah menentukan status aplikasi Anda, bagaimana itu bisa berubah, dan komponen mana yang diberi tahu saat terjadi perubahan.

Dalam **model** direktori, buat file baru bernama **app_state_manager.dart** dan tambahkan yang berikut ini:

```
impor 'panah: asinkron';
impor 'paket: flutter/material.dart';

// 1
kelas FooderlichTab {
  konstanta statis ke dalam jelajahi = 0;
  konstanta statis ke dalam resep = 1;
  konstanta statis ke dalam untuk Membeli = 2;
}

kelas AppStateManager meluas UbahPemberitahu {
  // 2
  bool _diinisialisasi = Salah;
  // 3
  bool _masuk = Salah;
  // 4
  bool _onboardingComplete = Salah;
  // 5
  ke dalam _selectedTab = FooderlichTab.explore;

  // 6
  bool Dapatkan isInitialized => _initialized;
  bool Dapatkan isLoggedIn => _loggedIn;
  bool Dapatkan isOnboardingComplete => _onboardingComplete;
  ke dalam Dapatkan getSelectedTab => _selectedTab;

  // TODO: Tambahkan initializeApp //
  TODO: Tambahkan login
  // TODO: Tambahkan completeOnboarding //
  TODO: Tambahkan goToTab
  // TODO: Tambahkan goToRecipes //
  TODO: Tambahkan logout
}
```

AppStateManager mengelola status navigasi aplikasi. Luangkan waktu sejenak untuk mengerti properti yang Anda tambahkan:

1. Membuat konstanta untuk setiap tab yang diketuk pengguna.

2. _diinisialisasi memeriksa apakah aplikasi diinisialisasi.

3. _masuk memungkinkan Anda memeriksa apakah pengguna telah masuk.

4. _onboardingSelesai memeriksa apakah pengguna menyelesaikan alur orientasi.

5. _dipilihTab melacak tab mana yang digunakan pengguna.

6. Ini adalah metode pengambil untuk setiap properti. Anda tidak dapat mengubah properti ini di luarAppStateManager. Ini penting untuk arsitektur aliran searah, di mana Anda tidak mengubah status secara langsung tetapi hanya melalui panggilan fungsi atau peristiwa yang dikirim.

Sekarang, saatnya mempelajari cara mengubah status aplikasi. Anda akan membuat fungsi untuk mengubah setiap properti yang dideklarasikan di atas.

Menginisialisasi aplikasi

Dalam file yang sama, cari // TODO: Tambahkan initializeApp dan ganti dengan yang berikut ini:

```
ruang kosong inisialisasiAplikasi() {
  // 7
  pengatur waktu (konstan Durasi(milidetik: 2000), () {
    // 8
    _diinisialisasi = benar;
    // 9
    notifyListeners();
  });
}
```

Berikut cara kerja kode:

7. Menyetel timer tunda selama 2.000 milidetik sebelum menjalankan penutupan. Ini set berapa lama layar aplikasi akan ditampilkan setelah pengguna memulai aplikasi.

8. Set diinisialisasi ke **benar**.

9. Memberitahu semua pendengar.

Masuk

Selanjutnya, cari // TODO: Tambahkan login dan ganti dengan yang berikut ini:

```
ruang kosong Gabung(Rangkaian nama pengguna, Rangkaian kata sandi) {
  // 10
  _masuk = benar;
  // 11
  notifyListeners();
}
```

Fungsi ini mengambil nama pengguna dan kata sandi. Inilah yang dilakukannya:

10. Set masuk ke **benar**.

11. Memberi tahu semua pendengar.

> **Catatan**: Dalam skenario nyata, Anda akan membuat permintaan API untuk masuk. Namun, dalam kasus ini, Anda hanya menggunakan tiruan.

Menyelesaikan orientasi

Selanjutnya, cari // TODO: Tambahkan Onboarding lengkap dan ganti dengan yang berikut ini:

```
ruang kosong selesaiOnboarding() {
  _onboardingComplete = benar;
  notifyListeners();
}
```

Panggilan selesaiOnboarding() akan memberi tahu semua pendengar bahwa pengguna telah selesai panduan orientasi.

Mengatur tab yang dipilih

Cari // TODO: Tambahkan goToTab dan ganti dengan yang berikut ini:

```
ruang kosong goToTab(indeks) {
  _selectedTab = indeks;
  notifyListeners();
}
```

goToTab menetapkan indeks _dipilihTab dan memberitahu semua pendengar.

Menavigasi ke tab Resep

Cari // TODO: Tambahkan goToRecipes dan ganti dengan yang berikut ini:

```
ruang kosong goToRecipes() {
  _selectedTab = FooderlichTab.recipes;
  notifyListeners();
}
```

Ini adalah fungsi pembantu yang langsung menuju ke tab resep.

Menambahkan kemampuan logout

Cari // TODO: Tambahkan logout dan ganti dengan yang berikut ini:

```
ruang kosong keluar() {
  // 12
  _masuk = Salah;
  _onboardingComplete =Salah;
  _diinisialisasi =Salah; _selectedTab =0;


  // 13
  inisialisasi Aplikasi();
  // 14
  notifyListeners();
}
```

Ketika pengguna logout, kode di atas:

12. Mereset semua properti status aplikasi.

13. Inisialisasi ulang aplikasi.

14. Memberi tahu semua pendengar tentang perubahan status.

Perhatikan bahwa semua fungsi ini mengikuti pola yang sama: mereka menetapkan beberapa nilai yang tidak diekspos secara publik dan kemudian memberi tahu pendengar. Ini adalah inti dari arsitektur aliran data searah yang Anda terapkan.

Akhirnya buka **lib/model/models.dart** dan tambahkan yang berikut ini:

```
ekspor 'app_state_manager.dart';
```

Dengan cara ini, Anda menambahkan yang baru dibuat AppStateManager ke file barel. Anda sekarang memiliki model status aplikasi yang terdefinisi dengan baik dan mekanisme yang memberi tahu pendengar tentang perubahan status. Ini adalah kemajuan besar. Sekarang, Anda akan menggunakannya di aplikasi!

Menggunakan newAppStateManager

Membuka **lib/main.dart**, temukan // TODO: Buat AppStateManager dan ganti dengan pengikut:

```
terakhir _appStateManager = AppStateManager();
```

Di sini, Anda menginisialisasi AppStateManager.

Selanjutnya, cari // TODO: Tambahkan AppStateManager ChangeNotifierProvider dan ganti dengan yang berikut ini:

```
ChangeNotifierProvider(buat: (konteks) => _appStateManager,),
```

Ini menciptakan penyedia perubahan untuk AppStateManager, sehingga turunan widget dapat mengakses atau mendengarkan status aplikasi.

Itu saja! Perhatikan bagaimana Anda mendefinisikan status aplikasi Anda terlebih dahulu? Setiap pengembang yang melihat file ini dapat mengetahui bagaimana pengguna berinteraksi dengan aplikasi Fooderlich.

Jangan tutup **main.dart**, Anda akan segera memperbaruinya lagi. Selanjutnya, Anda akan menambahkan router.

Membuat router

Router mengonfigurasi daftar halaman yang ditampilkan Navigator. Ini mendengarkan manajer negara bagian dan, berdasarkan perubahan status, mengonfigurasi daftar rute halaman.

Dibawah **lib/**, buat direktori baru bernama **navigasi**. Di dalam folder itu, buat yang baru file bernama **app_router.dart**. Tambahkan kode berikut:

```
impor 'paket: flutter/material.dart';
impor '../model/models.dart';
impor '../screens/screens.dart';

// 1
kelas AppRouter meluas RouterDelegasi
      dengan ChangeNotifier, PopNavigatorRouterDelegateMixin {
  // 2
  @mengesampingkan
  terakhir GlobalKey<NavigatorState> navigatorKey;

  // 3
  terakhir AppStateManager appStateManager;
  // 4
  terakhir GroceryManager GroceryManager;
  // 5
  terakhir ProfilManager profilManager;

  AppRouter({
    ini.appStateManager,
    ini.groceryManager,
          ini.profilManajer
  })
      : navigatorKey = GlobalKey<NavigatorState>() {
    // TODO: Tambahkan Pendengar
```

```
    }

    // TODO: Buang pendengar

    // 6
    @mengesampingkan
    Pembuatan widget (konteks BuildContext) {
      // 7
      kembali Navigator (
        // 8
        kunci: navigatorKey,
        // TODO: Tambahkan
        padaPopPage // 9
        halaman: [
          // TODO: Tambahkan SplashScreen //
          TODO: Tambahkan LoginScreen
          // TODO: Tambahkan OnboardingScreen //
          TODO: Tambahkan Beranda
          // TODO: Buat item baru
          // TODO: Pilih GroceryItemScreen // TODO:
          Tambahkan Layar Profil // TODO:
          Tambahkan Layar WebView
        ],
      );
    }

    // TODO: Tambahkan _handlePopPage

    // 10
    @mengesampingkan
    Masa Depan<ruang kosong> setNewRoutePath(konfigurasi) tidak sinkron => batal;
}
```

Berikut cara kerja widget router:

1. Ini memanjang RouterDelegasi. Sistem akan memberi tahu router untuk membuat dan mengonfigurasi widget navigator.

2. Menyatakan Kunci Global, kunci unik di seluruh aplikasi.

3. Menyatakan AppStateManager. Router akan mendengarkan perubahan status aplikasi untuk mengonfigurasi daftar halaman navigator.

4. Menyatakan Manajer Toko Kelontong untuk mendengarkan status pengguna saat Anda membuat atau mengedit item.

5. Menyatakan Manajer Profil untuk mendengarkan status profil pengguna.

6. RouterDelegasi mengharuskan Anda untuk menambahkan membangun(). Ini mengonfigurasi navigator dan halaman Anda.

7. Mengonfigurasi Navigator.

8. Menggunakan navigatorKey, yang diperlukan untuk mengambil navigator saat ini.

9. Menyatakan halaman, tumpukan halaman yang menjelaskan tumpukan navigasi Anda.

10. Set setNewRoutePath ke batal karena Anda belum mendukung aplikasi web Flutter. Jangan khawatir tentang itu untuk saat ini, Anda akan belajar lebih banyak tentang topik itu di bab berikutnya.

> **Catatan**: Bagaimana deklaratif ini? Alih-alih memberi tahu navigator apa yang harus dilakukan dorongan() dan pop(), Anda mengatakannya: ketika keadaannya **x**, render **kamu** halaman.

Sekarang setelah Anda menentukan router Anda, Anda akan membiarkannya menangani permintaan perutean.

Menangani acara pop

Cari // JUGA: Tambahkan _handlePopPage dan ganti dengan yang berikut ini:

```
bool _handlePopPage(
    // 1
    Rute<dinamis> rute,
    // 2
    hasil) {
    // 3
    jika (!route.didPop(hasil)) {
        // 4
        kembali salah;
    }

    // 5
    // TODO: Menangani Orientasi dan splash
    // TODO: Menangani status saat pengguna menutup layar item belanjaan //
    TODO: Menangani status saat pengguna menutup layar profil // TODO:
    Menangani status saat pengguna menutup layar WebView
        // 6
    kembali benar;
}
```

Saat pengguna mengetuk **Kembali** tombol atau memicu acara tombol kembali sistem, itu mengaktifkan metode pembantu, di PopPage.

Berikut cara kerjanya:

1. Ini adalah arusnya Rute, yang berisi informasi seperti Pengaturan Rute untuk mengambil nama rute dan argumen.
2. hasil adalah nilai yang dikembalikan saat rute selesai —nilai yang dikembalikan oleh dialog, misalnya.
3. Memeriksa apakah pop rute saat ini berhasil.
4. Jika gagal, kembalikan Salah.
5. Jika pop rute berhasil, ini akan memeriksa rute yang berbeda dan memicu perubahan status yang sesuai.

Sekarang, untuk menggunakan pembantu panggilan balik ini, cari // TODO: Tambahkan diPopPage dan ganti dengan yang berikut ini:

```
onPopPage: _handlePopPage,
```

Dengan cara ini, ini dipanggil setiap kali halaman muncul dari tumpukan.

Menambahkan pendengar status

Sekarang, Anda perlu menghubungkan manajer negara bagian. Ketika status berubah, router akan mengkonfigurasi ulang navigator dengan set halaman baru.

Cari // TODO: Tambahkan Pendengar dan ganti dengan yang berikut ini:

```
appStateManager.addListener(notifyListeners);
grosirManager.addListener(notifyListeners);
profileManager.addListener(notifyListeners);
```

Inilah yang dilakukan manajer negara bagian:

- **aplikasiStateManager**: Menentukan status aplikasi. Ini mengelola apakah aplikasi menginisialisasi login dan jika pengguna menyelesaikan orientasi.
- **manajer kelontong**: Mengelola daftar barang belanjaan dan status pemilihan barang.
- **profilManajer**: Mengelola profil dan pengaturan pengguna.

Saat Anda membuang router, Anda harus menghapus semua pendengar. Lupa melakukan ini akan menimbulkan pengecualian.

Cari // TODO: Buang pendengar dan ganti dengan yang berikut ini:

```
@mengesampingkan
ruang kosong buang() {
        appStateManager.removeListener(notifyListeners);
        grosirManager.removeListener(notifyListeners);
        profileManager.removeListener(notifyListeners);
    super.membuang();
}
```

Selamat, Anda baru saja mengatur widget router Anda. Sekarang, saatnya menggunakannya! Menyimpan **app_router.dart** terbuka, Anda akan segera menggunakannya lagi.

Menggunakan router aplikasi Anda

Router yang baru dibuat perlu mengetahui siapa manajernya, jadi sekarang Anda akan menghubungkannya ke manajer negara bagian, grosir, dan profil.

Membuka **main.dart** dan temukan // TODO: Impor app_router. Ganti dengan mengikuti:

```
impor 'navigasi/app_router.dart';
```

Selanjutnya, cari // TODO: Tentukan AppRouter dan ganti dengan yang berikut ini:

```
AppRouter _appRouter;
```

Setelah Anda mendeklarasikan router aplikasi Anda, cari // TODO: Inisialisasi router aplikasi dan ganti dengan yang berikut ini:

```
@mengesampingkan
ruang kosong keadaan init() {
    _appRouter = AppRouter(
        appStateManager: _appStateManager,
        grosirManager: _groceryManager,
        profileManager: _profileManager,
    );
    super.initState();
}
```

Anda sekarang telah menginisialisasi router aplikasi Anda di initState() sebelum Anda menggunakannya. Menyimpan **main.dart** membuka.

Untuk langkah selanjutnya, cari // TODO: Ganti dengan widget Router. Ganti ada rumah: const SplashScreen(), baris dengan kode berikut:

```
rumah: Router(
        routerDelegate: _appRouter,
    // TODO: Tambahkan backButtonDispatcher
),
```

Anda tidak perlu mengimpor layar Splash lagi. Silakan dan**menghapus** kode berikut:

```
impor 'screens/splash_screen.dart';
```

Router Anda sudah siap sekarang! Saatnya untuk membiarkannya bermain dengan layar.

Menambahkan layar

Dengan semua infrastruktur di tempat, sekarang saatnya untuk menentukan layar mana yang akan ditampilkan menurut rute. Tapi pertama-tama, periksa situasi saat ini. Bangun dan jalankan di iOS. Anda akan melihat pengecualian di**Lari** tab:

```
======== Exception caught by widgets library =======================
Navigator.onGenerateRoute was null, but the route named "/" was referenced.
The relevant error-causing widget was:
  Router<dynamic> file:///Users/
```

Lebih buruk lagi, simulator mungkin menampilkan layar merah kematian:

Itu karena Navigator halaman tidak boleh kosong. Aplikasi memberikan pengecualian karena tidak dapat menghasilkan rute. Anda akan memperbaikinya dengan menambahkan layar berikutnya.

Menampilkan layar Splash

Anda akan mulai dari awal, menampilkan layar Splash.

Membuka **lib/screens/splash_screen.dart** dan tambahkan impor berikut:

```
impor 'paket: penyedia/penyedia.dart';
impor '../model/models.dart';
```

Selanjutnya, cari // TODO: SplashScreen MaterialPage Helper dan ganti dengan pengikut:

```
statis Halaman MaterialHalaman() {
    kembali Halaman Materi(
        nama: FooderlichPages.splashPath,
            kunci: ValueKey(FooderlichPages.splashPath), anak:
            konstan SplashScreen(),);
}
```

Di sini, Anda menentukan metode statis untuk membuat Halaman Materi yang menetapkan pengidentifikasi unik yang sesuai dan membuat Layar Splash.

Selanjutnya temukan // TODO: Inisialisasi Aplikasi dan ganti dengan yang berikut ini:

```
Provider.of<AppStateManager>(konteks, dengarkan:
Salah).initializeApp();
```

Di sini, Anda menggunakan konteks saat ini untuk mengambil AppStateManager untuk menginisialisasi aplikasi.

Sekarang, Anda ingin menambahkan layar Splash yang ditampilkan saat aplikasi dimulai.

Kembali ke **app_router.dart**, temukan // TODO: Tambahkan SplashScreen dan menggantinya dengan berikut ini:

```
jika (!appStateManager.isInitialized) SplashScreen.page(),
```

Di sini, Anda memeriksa apakah aplikasi diinisialisasi. Jika tidak, Anda menunjukkan layar Splash.

Lakukan restart panas dan Anda akan melihat layar berikut berkedip oleh:

Anda masih akan melihat kesalahan tapi jangan khawatir, itu akan segera hilang.

Selamat, Anda baru saja mengatur rute pertama Anda! Sekarang, akan jauh lebih mudah untuk mempersiapkan rute lainnya. Meninggalkan**app_router.dart** membuka.

Rangkaian pembaruan kode berikutnya akan mengikuti pola serupa:

• Perbarui kode layar untuk memicu perubahan status melalui pengelola.

• Perbarui kode router untuk menangani perubahan status baru, sesuai dengan rute yang ditetapkan saat ini.

Menampilkan layar Login

Sekarang Anda akan menerapkan langkah pertama dari logika perutean: menampilkan layar Login setelah layar Splash jika pengguna tidak login.

Membuka **lib/layar/login_screen.dart** dan tambahkan impor berikut:

```
impor 'paket: penyedia/penyedia.dart';
impor '../model/models.dart';
```

Selanjutnya, cari // TODO: LoginScreen MaterialPage Helper dan ganti dengan mengikuti:

```
statis Halaman MaterialHalaman() {
  kembali Halaman Materi(
        nama: FooderlichPages.loginPath,
        kunci: ValueKey(FooderlichPages.loginPath), anak:
        konstan Layar login());
}
```

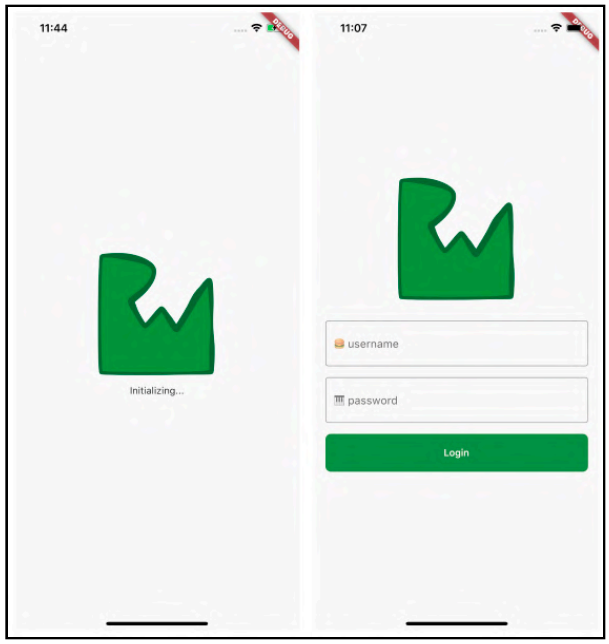
Di sini, Anda mendefinisikan metode statis yang membuat Halaman Bahan, menetapkan kunci unik dan menciptakan Layar login. Menyimpan **login_screen.dart** membuka.

Beralih kembali ke **app_router.dart**, temukan // TODO: Tambahkan Layar Masuk dan ganti dengan yang berikut ini:

```
jika (appStateManager.isInitialized && !
appStateManager.isLoggedIn)
LoginScreen.page(),
```

Kode ini mengatakan bahwa jika aplikasi diinisialisasi dan pengguna belum masuk, itu akan menampilkan halaman masuk.

Memicu restart panas. Anda akan melihat layar Splash selama beberapa detik, diikuti oleh layar Login:



Selamat, kesalahan telah hilang dan Anda telah berhasil menerapkan rute. Langkah terakhir adalah menangani perubahan status login.

Kembali **login_screen.dart**, temukan // TODO: Masuk -> Arahkan ke rumah dan ganti dengan yang berikut ini:

```
Provider.of<AppStateManager>(konteks, dengarkan: Salah)
    . Gabung('nama pengguna tiruan', 'kata sandi tiruan');
```

Ini menggunakan AppStateManager untuk memanggil fungsi yang memperbarui status login pengguna. Apa yang terjadi ketika status login berubah? Senang Anda bertanya, itu langkah selanjutnya. :]

Transisi dari Login ke layar Orientasi

Saat pengguna masuk, Anda ingin menampilkan layar Orientasi.

Membuka **lib/screens/onboarding_screen.dart** dan tambahkan impor berikut:

```
impor 'paket: penyedia/penyedia.dart';
impor '../model/models.dart';
```

Selanjutnya, cari // TODO: Tambahkan Pembantu MaterialPage OnboardingScreen dan ganti itu dengan berikut ini:

```
statis Halaman MaterialHalaman() {
  kembali Halaman Materi(
        nama: FooderlichPages.onboardingPath,
        kunci: ValueKey(FooderlichPages.onboardingPath), anak:
        konstan Layar Onboarding(),);
}
```

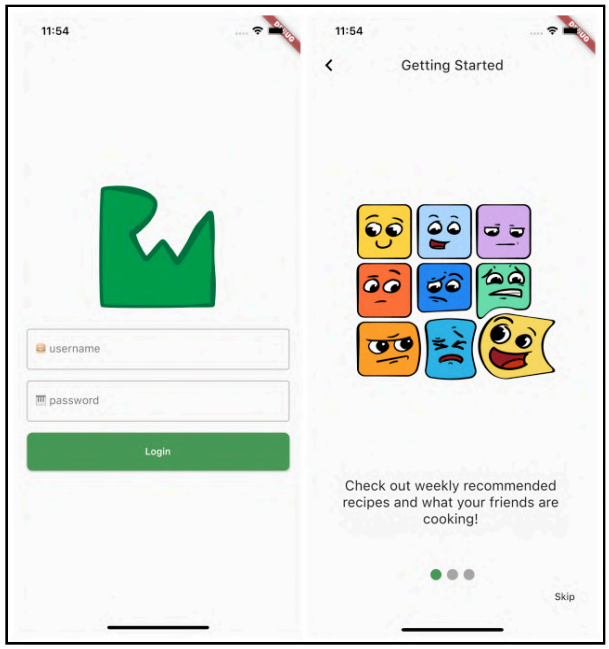
Di sini, Anda mengonfigurasi Halaman Bahan, atur kunci unik halaman orientasi dan buat widget layar orientasi.

Kembali ke **app_router.dart**, temukan // TODO: Tambahkan Layar Orientasi dan menggantinya dengan berikut ini:

```
jika (appStateManager.isLoggedIn &&
      !appStateManager.isOnboardingComplete)
      OnboardingScreen.page(),
```

Di sini, Anda menampilkan layar Orientasi jika pengguna masuk tetapi belum menyelesaikan Panduan Orientasi.

Lakukan restart panas lainnya lalu ketuk **Gabung** tombol. Anda akan melihat layar Orientasi muncul.



Selamat, ini kemajuan yang bagus. Sekarang, Anda akan menambahkan logika untuk menangani perubahan yang dipicu dalam layar Orientasi.

Menangani tombol Lewati dan Kembali di Orientasi

Saat pengguna mengetuk **Melewati** daripada melalui panduan Orientasi, Anda ingin menampilkan layar beranda biasa.

Di dalam **onboarding_screen.dart**, temukan // TODO: Orientasi -> Navigasi ke rumah dan ganti dengan yang berikut ini:

```
Provider.of<AppStateManager>(konteks, dengarkan: Salah)
    . selesaiOnboarding();
```

Di sini, mengetuk **Melewati** pemicu selesaiOnboarding(), yang memperbarui status dan menunjukkan bahwa pengguna telah menyelesaikan orientasi. Ini belum berfungsi, jadi jangan panik jika Anda melihat kesalahan.

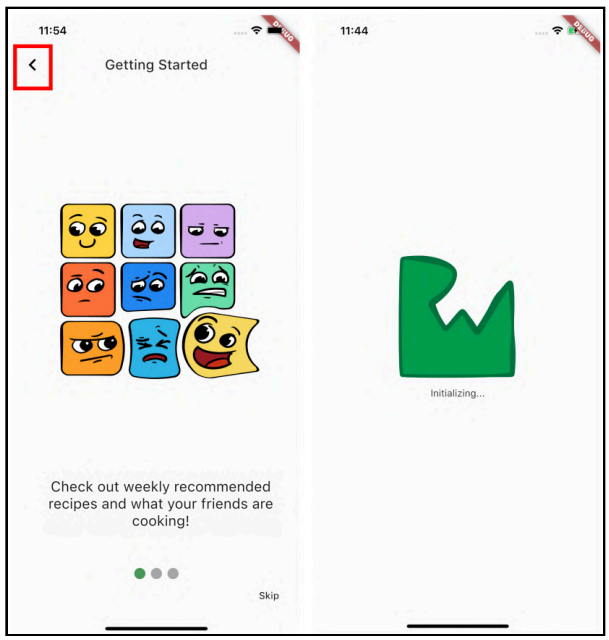
Selanjutnya, Anda ingin berurusan dengan apa yang terjadi ketika pengguna mengetuk **Kembali** di layar Orientasi.

Kembali ke **app_router.dart**, temukan TODO: Menangani Orientasi dan Splash dan ganti dengan yang berikut ini:

```
jika (route.settings.name == FooderlichPages.onboardingPath) {
    appStateManager.logout();
}
```

Jika pengguna mengetuk **Kembali** tombol dari layar Orientasi, itu memanggil keluar(). Ini mengatur ulang seluruh status aplikasi dan pengguna harus masuk lagi.

Aplikasi akan kembali ke layar Splash untuk menginisialisasi ulang, seperti yang ditunjukkan di bawah ini:



Transisi dari Orientasi ke Beranda

Saat pengguna mengetuk **Melewati**, aplikasi akan menampilkan layar Utama. Membuka**lib/layar/home.dart** dan tambahkan impor berikut:

```
impor 'paket: penyedia/penyedia.dart';
impor '../model/models.dart';
```

Selanjutnya, cari // TODO: Home MaterialPage Helper dan ganti dengan yang berikut ini:

```
statis halaman halaman materi(ke dalam tab saat ini) {
    kembali Halaman Materi(
            nama: FooderlichPages.home,
            kunci: ValueKey(FooderlichPages.home), anak:
            Home(
                tab saat ini: tab saat ini,
            ), ) ;
}
```

Di sini, Anda telah membuat static Halaman Materi helper dengan tab saat ini untuk ditampilkan di layar Beranda. Menyimpan**home.dart** membuka.

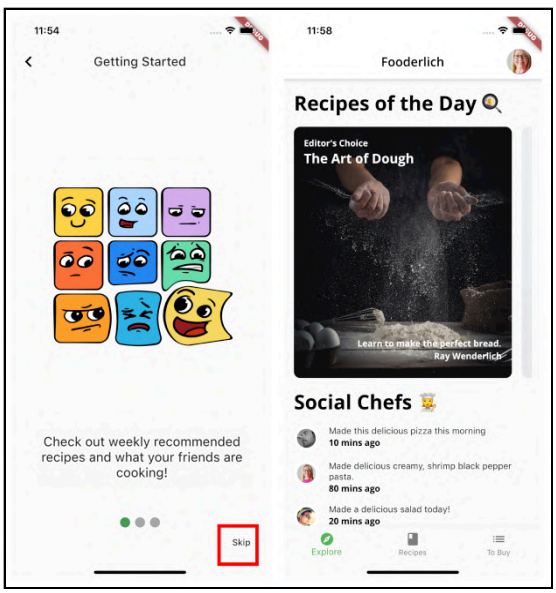
Kembali ke **app_router.dart**, temukan // TODO: Tambahkan Beranda dan ganti dengan yang berikut ini:

```
jika (appStateManager.isOnboardingComplete)
Beranda.halaman(appStateManager.getSelectedTab),
```

Ini memberi tahu aplikasi Anda untuk menampilkan halaman beranda hanya ketika pengguna menyelesaikan orientasi.

Akhirnya, Anda dapat melihat orientasi beraksi!

Restart panas, navigasikan ke layar Orientasi dengan mengetuk **Gabung** tombol lalu ketuk **Melewati** tombol. Anda sekarang akan melihat layar Beranda. Selamat!



Anda akan melihat bahwa Anda tidak dapat beralih ke tab yang berbeda. Itu karena Anda belum menyiapkan penanganan status. Anda akan melakukannya selanjutnya.

Menangani pemilihan tab

Membuka **home.dart**, temukan // TODO: Bungkus Konsumen untuk AppStateManager dan ganti dengan yang berikut ini:

```
kembali Konsumen<AppStateManager>(
    pembangun: (konteks, appStateManager, anak) {
```

Abaikan coretan merah untuk saat ini.

Selanjutnya, gulir ke bawah hingga akhir widget dan, tepat sebelum penutup }, tambahkan yang berikut ini:

```
    },);
```

Pastikan Anda telah mengaktifkan format otomatis dan menyimpan file untuk memformat ulang.

Anda baru saja membungkus seluruh widget Anda di dalam a Konsumen. Konsumenakan mendengarkan perubahan status aplikasi dan membangun kembali widget dalamnya yang sesuai.

Selanjutnya, cari // TODO: Perbarui tab yang dipilih pengguna dan ganti dengan mengikuti:

```
Provider.of<AppStateManager>(konteks, dengarkan: Salah)
        . goToTab(indeks);
```

Di sini, Anda menentukan bahwa mengetuk panggilan tab goToTab().

Menangani tombol Telusuri Resep

Sekarang, Anda ingin menambahkan ketukan itu **Jelajahi Resep** tombol membawa pengguna ke **resep** tab.

Membuka **blank_grocery_screen.dart** tambahkan impor berikut:

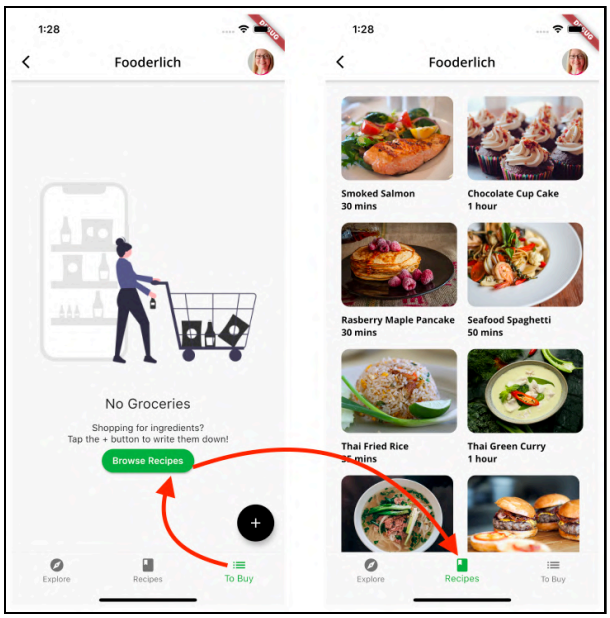
```
impor 'paket: penyedia/penyedia.dart';
impor '../model/models.dart';
```

Selanjutnya, cari // TODO: Perbarui tab yang dipilih pengguna dan ganti dengan mengikuti:

```
Provider.of<AppStateManager>(konteks, dengarkan: Salah)
    . goToRecipes();
```

Di sini, Anda menentukan penyadapan itu **Jelajahi Resep** panggilan goToRecipes(). Ini mirip dengan apa yang Anda lakukan untuk tab.

Untuk mengujinya, ketuk **Untuk membeli** tab di bilah navigasi bawah, lalu ketuk **Jelajahi Resep** tombol. Perhatikan bahwa aplikasi masuk ke**resep** tab, seperti gambar di bawah ini:



Menampilkan layar Barang Kelontong

Selanjutnya, Anda akan menghubungkan layar Grocery Item. Membuka**lib/layar/ grosir_item_screen.dart**. Cari //TODO: GroceryItemScreen MaterialPage Pembantu dan ganti dengan yang berikut ini:

```
statis halaman halaman materi(
        {Item Barang Kelontong,
        ke dalam indeks,
        Fungsi(Item Kelontong) diBuat,
        Fungsi(Barang Kelontong, ke dalam) pada Pembaruan}) {
    kembali Halaman Materi(
```

```
            nama: FooderlichPages.groceryItemDetails,
            kunci: ValueKey(FooderlichPages.groceryItemDetails), anak:
            GroceryItemScreen(
                barang asli: barang,
                indeks: indeks,
                diBuat: diBuat,
                onUpdate: onUpdate,
            ), ) ;
}
```

Di sini, Anda membuat pembantu halaman statis yang membungkus Layar Barang Kelontong di sebuah Halaman Materi. Layar Item Grocery membutuhkan:

1. Barang belanjaan asli, jika ada. Jika tidak, itu mengasumsikan pengguna membuat barang belanjaan baru.
2. Indeks barang belanjaan yang dipilih.
3. diBuat ketika pengguna selesai membuat item baru.
4. diUpdate ketika pengguna selesai memperbarui item.

Selanjutnya, Anda akan menerapkan layar Grocery Item. Ada dua cara untuk menunjukkannya:

1. Pengguna mengetuk **+** tombol untuk membuat item belanjaan baru.
2. Pengguna mengetuk item belanjaan yang ada untuk mengeditnya.

Anda akan mengaktifkan fitur ini selanjutnya.

Membuat item kelontong baru

Membuka **lib/screens/grocery_screen.dart** dan temukan // TODO: Buat Item Baru. Ganti dengan yang berikut ini:

```
Provider.of<GroceryManager>(konteks, dengarkan:
Salah).createNewItem();
```

Di sini, Anda memicu panggilan ke buatBarangBaru() saat pengguna mengetuk **+** tombol.

Selanjutnya, kembali ke **app_router.dart**, temukan // TODO: Buat item baru dan menggantinya dengan berikut ini:

```
// 1
jika (groceryManager.isCreatingNewItem)
// 2
Halaman GroceryItemScreen.(
    diBuat: (item) {
```

```
        // 3
        grosirManager.addItem(item); },),
```

Inilah cara ini memungkinkan Anda menavigasi ke item belanjaan baru:

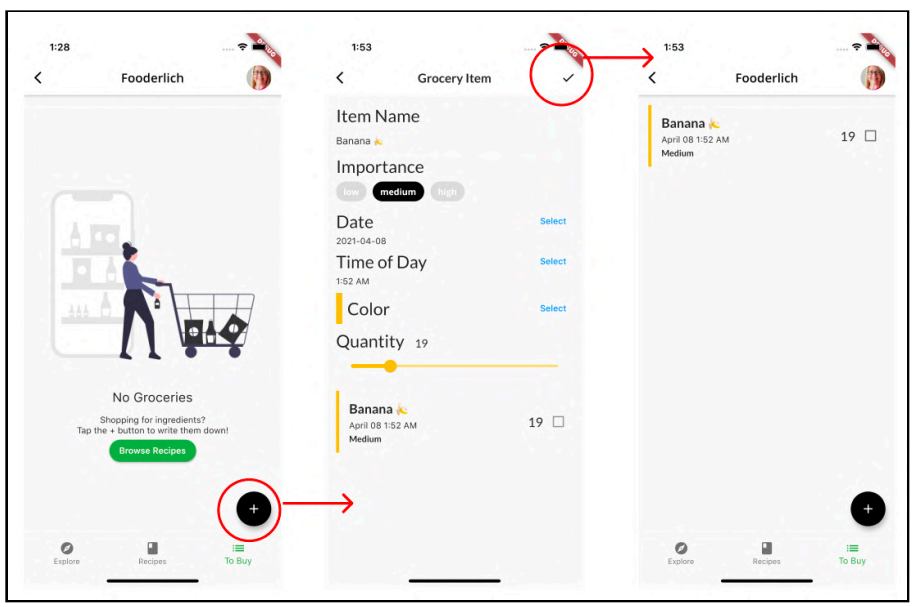
1. Memeriksa apakah pengguna membuat barang belanjaan baru.

2. Jika ya, tampilkan layar Barang Kelontong.

3. Setelah pengguna menyimpan item, memperbarui daftar belanjaan.

Dengan aplikasi Anda berjalan, lakukan restart panas. Anda sekarang dapat membuat yang baru

bahan makanan, seperti yang ditunjukkan di bawah ini:



Mengedit item belanjaan yang ada

Membuka **grosir_list_screen.dart**, temukan // TODO: Ketuk item belanjaan dan ganti dengan yang berikut ini:

```
    manager.groceryItemTapped(indeks);
```

Ini kebakaran kelontongItemTapped() untuk memberi tahu pendengar bahwa pengguna memilih barang belanjaan.

Sekarang, kembali ke **app_router.dart**, temukan // TODO: Pilih GroceryItemScreen dan ganti dengan yang berikut ini:

```
// 1
jika (groceryManager.selectedIndex != batal)
// 2
Halaman GroceryItemScreen.(
    item: grosirManager.selectedGroceryItem, indeks:
    grosirManager.selectedIndex, onUpdate: (item, indeks)
    {
        // 3
        grosirManager.updateItem(item, indeks); },),
```

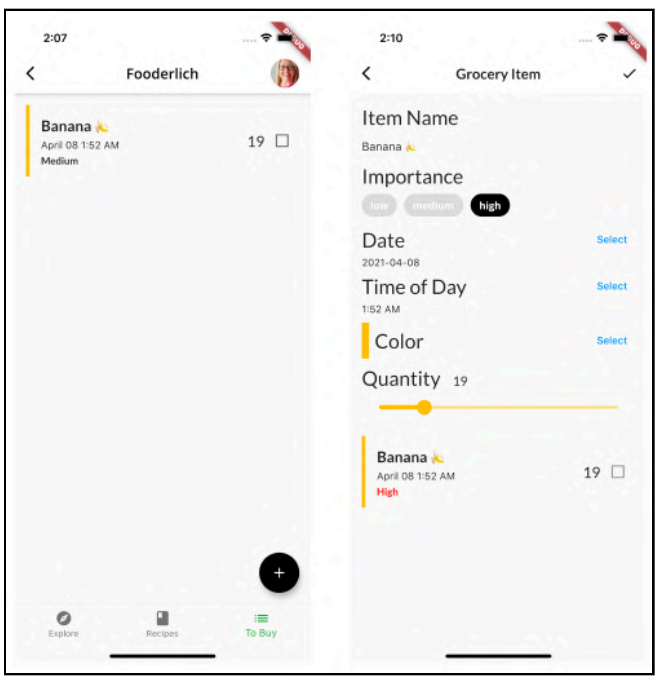
Berikut cara kerja kode:

1. Cek untuk melihat apakah item belanjaan dipilih.

2. Jika demikian, buat halaman layar Barang Kelontong.

3. Saat pengguna mengubah dan menyimpan item, item tersebut akan diperbarui pada indeks saat ini.

Sekarang, Anda dapat mengetuk item belanjaan, mengeditnya, dan menyimpannya!



Menutup layar Barang Kelontong

Terkadang, pengguna mulai menambahkan barang belanjaan, lalu berubah pikiran. Untuk menutupi kasus ini, buka**app_router.dart**, temukan // TODO: Menangani status saat pengguna menutup layar barang belanjaan dan ganti dengan yang berikut ini:

```
jika (route.settings.name == FooderlichPages.groceryItemDetails) {
        grosirManager.groceryItemTapped(batal);
}
```

Ini memastikan bahwa status yang sesuai diatur ulang saat pengguna mengetuk tombol kembali dari layar Barang Kelontong.

Restart panas dan kemudian uji urutannya lagi:

1. Ketuk **+** tombol untuk membuat item belanjaan baru.

2. Ketuk **<** tombol untuk kembali.

Perhatikan bahwa aplikasi sekarang berfungsi seperti yang diharapkan.

Menavigasi ke layar Profil

Pengguna belum dapat menavigasi ke layar Profil. Sebelum Anda dapat memperbaikinya, Anda perlu menangani perubahan status.

Membuka **home.dart**, temukan // TODO: rumah -> profil dan ganti dengan yang berikut ini:

```
Provider.of<ProfileManager>(konteks, dengarkan: Salah)
. ketukDiProfil(benar);
```

Ini memicu ketukDiProfil() setiap kali pengguna mengetuk tombol Profil.

Sekarang pengguna dapat membuka layar Profil, mereka harus dapat menutupnya lagi.

Membuka **lib/screens/profile_screen.dart**, temukan // TODO: Tutup Layar Profil dan ganti dengan yang berikut ini:

```
Provider.of<ProfileManager>(konteks, dengarkan: Salah)
   . ketukDiProfil(Salah);
```

Ini menangani tindakan yang terjadi saat pengguna mengetuk **x** (tutup) tombol. Ini memperbarui status profil sehingga navigator menghapus layar Profil.

Sekarang, cari // TODO: ProfileScreen MaterialPage Helper dan ganti dengan pengikut:

```
statis Halaman MaterialPage(Pengguna pengguna) {
    kembali Halaman Materi(
            nama: FooderlichPages.profilePath,
            kunci: ValueKey(FooderlichPages.profilePath), anak:
            ProfileScreen(pengguna: pengguna),);
}
```

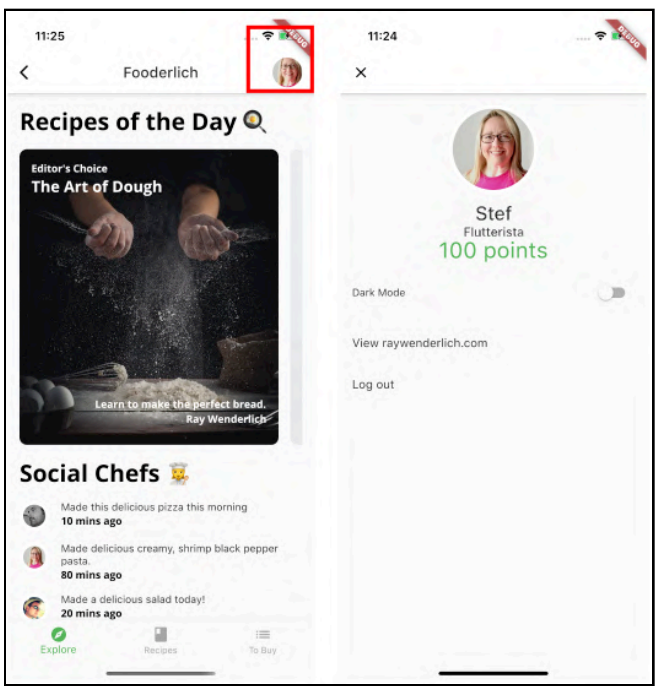
Di sini, Anda membuat pembantu Halaman Materi untuk layar Profil. Ini membutuhkan objek pengguna.

Selanjutnya, buka **app_router.dart**, temukan // TODO: Tambahkan Layar Profil dan ganti dengan yang berikut ini:

```
jika (profileManager.didSelectUser)
ProfileScreen.page(profileManager.getUser),
```

Ini memeriksa manajer profil untuk melihat apakah pengguna memilih profil mereka. Jika demikian, itu menunjukkan layar Profil.

Lakukan hot reload dan ketuk avatar pengguna. Sekarang akan menampilkan layar Profil:



Membuka **app_router.dart**, temukan // TODO: Menangani status saat pengguna menutup layar profil dan ganti dengan yang berikut ini:

```
jika (route.settings.name == FooderlichPages.profilePath) {
    profileManager.tapOnProfile(Salah);
}
```

Ini memeriksa untuk melihat apakah rute yang Anda lewati memang benar profilPath, kemudian memberitahu profilManajer bahwa layar Profil tidak terlihat lagi.

Sekarang ketuk **x** tombol dan layar Profil akan hilang.

Menavigasi ke raywenderlich.com

Di dalam layar Profil, Anda dapat melakukan tiga hal:

1. Ubah pengaturan mode gelap.

2. Kunjungi raywenderlich.com.

3. Logout.

Selanjutnya, Anda akan menangani layar WebView.

Transisi dari Profil ke Tampilan Web

Kembali ke **profile_screen.dart**, temukan // TODO: Buka raywenderlich.com WebView dan ganti dengan yang berikut ini:

```
Provider.of<ProfileManager>(konteks, dengarkan: Salah)
    . ketuk PadaRaywenderlich(benar);
```

Di sini, Anda mengatakan untuk menelepon tapOnRaywenderlich() ketika pengguna mengetuk tombol yang sesuai. Ini memicu pembangunan kembali pada widget router Anda dan menambahkan layar Tampilan Web.

Sekarang buka **webview_screen.dart** dan impor berikut ini:

```
impor '../model/models.dart';
```

Selanjutnya, cari // TODO: WebViewScreen MaterialPage Helper dan ganti dengan pengikut:

```
statis Halaman MaterialHalaman() {
    kembali Halaman Materi(
            nama: FooderlichPages.raywenderlich,
            kunci: ValueKey(FooderlichPages.raywenderlich), anak:
            konstan WebViewScreen(),);
}
```

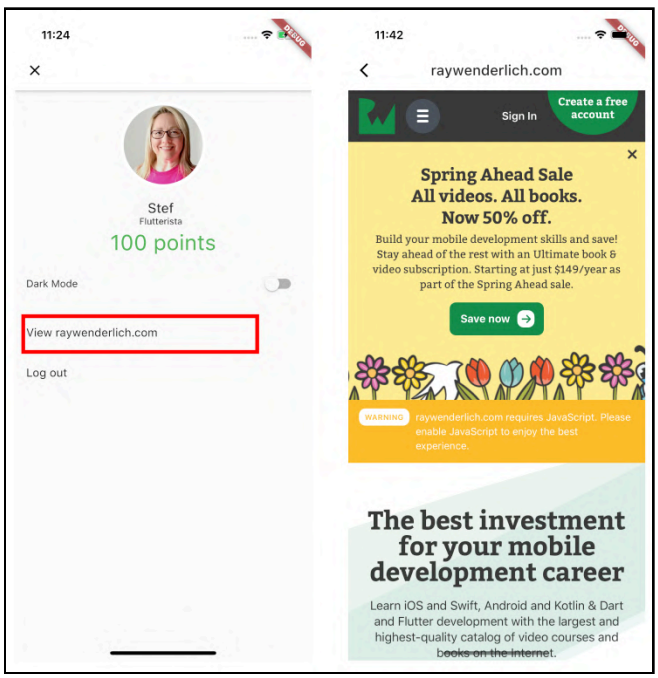
Di sini, Anda membuat statis Halaman Materi yang membungkus widget layar WebView.

Selanjutnya, kembali ke **app_router.dart**. Cari //TODO: Tambahkan Layar WebView dan ganti dengan yang berikut ini:

```
jika (profileManager.didTapOnRaywenderlich)
WebViewScreen.page(),
```

Ini memeriksa apakah pengguna mengetuk opsi untuk membuka situs web raywenderlich.com. Jika demikian, ini menyajikan layar WebView.

Muat ulang panas dan buka layar Profil. Sekarang, ketuk**Lihat raywenderlich.com** dan Anda akan melihatnya hadir dalam tampilan web, seperti yang ditunjukkan di bawah ini:



Bagaimana dengan menutup tampilan?

Tetap **app_router.dart**, temukan // TODO: Menangani status saat pengguna menutup Layar Tampilan Web dan ganti dengan yang berikut ini:

```
jika (route.settings.name == FooderlichPages.raywenderlich) {
    profileManager.tapOnRaywenderlich(Salah);
}
```

Di sini, Anda memeriksa apakah nama pengaturan rute adalah **raywenderlich**, lalu panggil metode yang sesuai di manajer profil.

Selanjutnya, Anda akan mengerjakan fungsi logout.

Keluar

Untuk menangani pengguna yang logout, buka **profile_screen.dart** dan temukan // TODO: Logout pengguna. Ganti dengan yang berikut ini:

```
// 1
Provider.of<ProfileManager>(konteks, dengarkan: Salah)
        . ketukDiProfil(Salah);
// 2
Provider.of<AppStateManager>(konteks, dengarkan: Salah).keluar();
```

Inilah yang terjadi ketika tindakan logout dipicu:

1. Setel status ketukan profil pengguna ke **Salah**.

2. Panggilan keluar(), yang mengatur ulang seluruh status aplikasi.

Simpan perubahan Anda. Sekarang, ketuk**Keluar** dari layar Profil dan Anda akan melihatnya kembali ke layar Splash, seperti yang ditunjukkan di bawah ini:

Selanjutnya, Anda akan membahas sistem Android **Kembali** tombol.

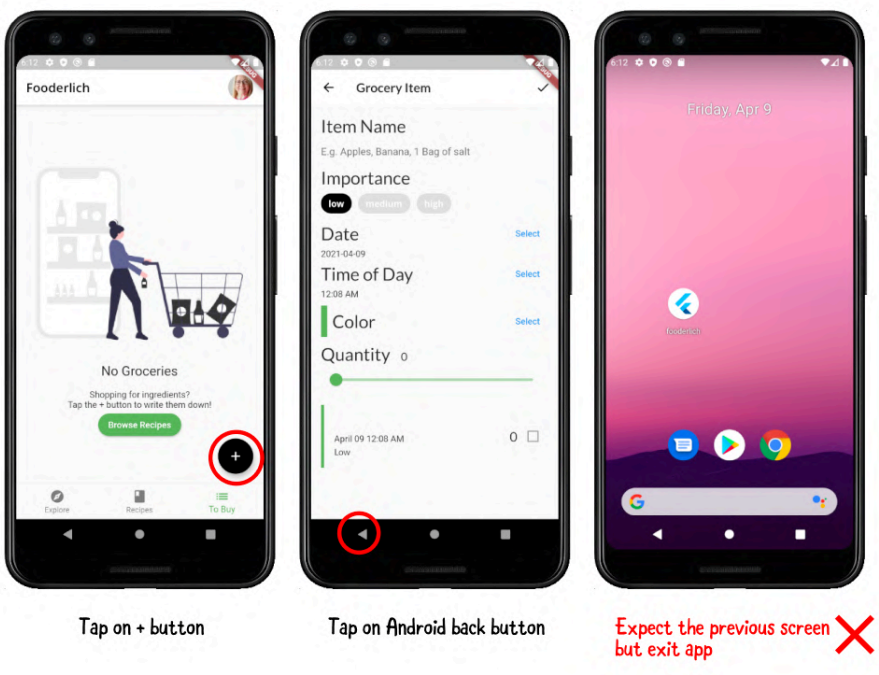
Menangani tombol Kembali sistem Android

Jika Anda telah menjalankan proyek di iOS, hentikan aplikasi di perangkat atau simulator yang ada. Sekarang, buat dan jalankan aplikasi Anda di **Android** perangkat atau emulator. Lakukan tugas-tugas berikut:

1. Navigasikan melalui aplikasi ke **Untuk membeli** tab.

2. Ketuk **+** tombol.

3. Ketuk sistem Android **Kembali** tombol, bukan aplikasi **Kembali** tombol.



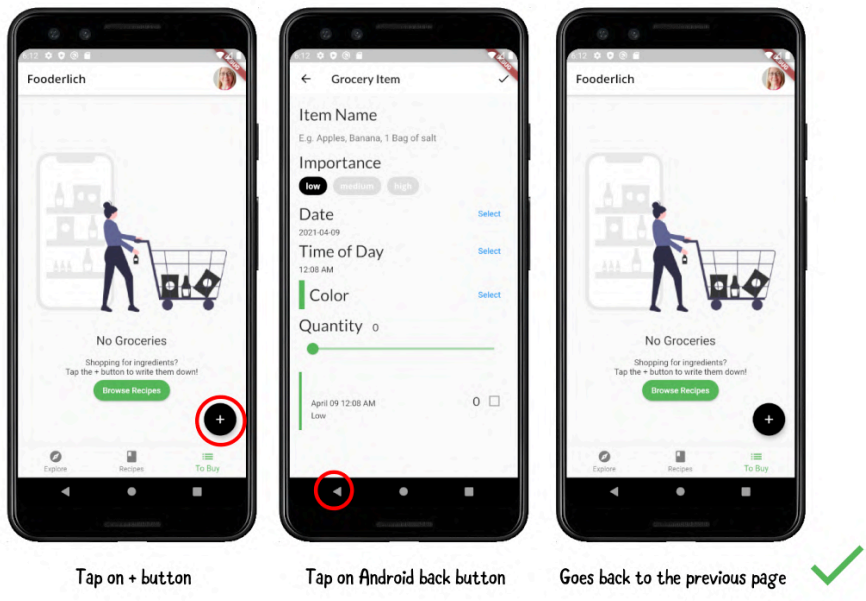
Anda mengharapkannya untuk kembali ke halaman sebelumnya. Sebaliknya, itu keluar dari seluruh aplikasi!

Untuk memperbaikinya, buka **main.dart**, temukan // TODO: Tambahkan kembaliButtonDispatcher dan ganti dengan yang berikut ini:

```
backButtonDispatcher: RootBackButtonDispatcher(),
```

Di sini, Anda mengatur widget router KembaliButtonDispatcher, yang mendengarkan pemberitahuan rute pop platform. Saat pengguna mengetuk sistem Android**Kembali** tombol, itu memicu delegasi router onPopPage panggilan balik.

Mulai ulang aplikasi Anda dan coba langkah yang sama lagi.



Woo-hoo, itu berperilaku seperti yang diharapkan! Selamat, Anda sekarang telah menyelesaikan seluruh alur navigasi UI.