BAB III

METODE PELAKSANAAN MBKM KEWIRAUSAHAAN

3.1 Produksi

Produksi aplikasi *U-Teen* merupakan proses utama dalam pengembangan solusi digital yang ditujukan untuk meningkatkan efisiensi layanan pemesanan makanan di lingkungan Universitas Multimedia Nusantara. Aplikasi ini dirancang untuk dapat diakses melalui perangkat Android dan iOS, dan memungkinkan mahasiswa, dosen, maupun staf untuk melakukan pemesanan makanan secara praktis, cepat, dan tanpa perlu mengantre secara fisik.

Seluruh proses produksi dimulai dengan tahap perancangan antarmuka pengguna, dilanjutkan dengan pengembangan aplikasi dan koneksi ke sistem basis data, serta diakhiri dengan uji coba operasional pada mitra tenant kantin. Proses ini dilakukan secara kolaboratif oleh tim yang terdiri dari tiga anggota dengan pembagian peran yang spesifik, namun dalam laporan ini fokus akan diarahkan pada kontribusi penulis sebagai pengembang *front-end* dan *back-end*.

3.1.1. Perancangan Antarmuka Aplikasi

Tahap perancangan dimulai dengan proses penyusunan alur kerja pengguna dan perancangan antarmuka yang dikerjakan oleh Amelia Rama Setiawan selaku penanggung jawab desain *UI/UX*. Proses ini dilakukan menggunakan platform *Figma* dan mencakup seluruh elemen tampilan aplikasi mulai dari *login screen*, halaman pemesanan, riwayat transaksi, hingga notifikasi pesanan.

Tujuan dari tahap ini adalah menciptakan antarmuka yang tidak hanya menarik secara visual, tetapi juga intuitif dan mudah digunakan oleh pengguna dengan latar belakang teknologi yang berbeda-beda. Desain kemudian dijadikan acuan utama dalam proses pengembangan sistem dan pembuatan komponen

tampilan aplikasi.

3.1.2. Pengembangan Aplikasi

Setelah desain diselesaikan, penulis memulai tahap pengembangan aplikasi dengan menggunakan *Visual Studio Code* sebagai editor utama, serta *Flutter* dan *Dart* sebagai framework dan bahasa pemrograman utama. Pengembangan dilakukan secara bertahap, mencakup:

1. Pembangunan *front-end* untuk mengimplementasikan desain UI ke dalam tampilan interaktif dan dinamis
2. Pembuatan *back-end logic* untuk mengatur alur transaksi, pengelolaan data pesanan, pengaturan status pesanan, dan integrasi ke metode pembayaran digital (*GoPay* dan *OVO*)
3. Penanganan *state management* dan *provider structure* untuk menjamin kestabilan sistem pada saat aplikasi dijalankan secara *live*

Sementara itu, pengembangan *database* menggunakan *Firebase Firestore* dikerjakan oleh anggota tim lain, dengan skema penyimpanan *base64 image*, struktur koleksi produk, dan pencatatan transaksi berbasis *real-time sync*. Integrasi antara aplikasi dan database dilakukan melalui koneksi API dan layanan *Firebase SDK*.

3.1.3. Pengujian dan Evaluasi

Proses pengujian dilakukan secara menyeluruh oleh penulis secara bertahap sejak awal pengembangan. Pengujian awal dilakukan melalui emulator dan perangkat fisik untuk memastikan bahwa fitur berjalan dengan stabil dan bebas dari *bug*. Fokus utama dalam pengujian mencakup:

1. Uji fungsionalitas tiap modul (*order flow*, notifikasi, dan riwayat transaksi)
2. Uji performa dan kecepatan respon aplikasi
3. Uji kompatibilitas perangkat Android

Setelah sistem menunjukkan stabilitas, uji coba juga dilakukan dengan melibatkan tenant kantin yang bersedia mencoba aplikasi secara langsung. Uji coba ini digunakan untuk mengidentifikasi *feedback* dari pengguna akhir, serta melakukan iterasi perbaikan terhadap alur tampilan dan logika aplikasi agar semakin optimal.

3.2 Penetapan Harga

Penetapan harga dalam pengelolaan aplikasi *U-Teen* dilakukan dengan memperhitungkan nilai manfaat yang ditawarkan kepada pengguna, kelayakan finansial, serta keadilan distribusi keuntungan bagi seluruh pihak yang terlibat, yaitu tenant kantin, pengelola, dan pengembang sistem. Model harga yang digunakan tidak membebankan biaya langsung kepada konsumen, melainkan mengadopsi pendekatan *revenue sharing* berbasis transaksi.

3.2.1. Berdasarkan Nilai yang Diberikan

Aplikasi *U-Teen* dirancang untuk meningkatkan efisiensi operasional kantin kampus melalui fitur pemesanan makanan secara digital, pembayaran elektronik, sistem antrean non-fisik, dan visualisasi menu yang terpusat. Dengan mempertimbangkan nilai tambah ini, aplikasi tidak memungut biaya dari mahasiswa, dosen, atau staf sebagai pengguna akhir, tetapi menerapkan skema monetisasi melalui sistem bagi hasil dari transaksi yang terjadi di dalam platform.

3.2.2. Skema Bagi Hasil dengan Tenant

Berdasarkan simulasi dan kesepakatan pembagian pendapatan yang disusun, skema distribusi dilakukan dengan proporsi berikut untuk setiap transaksi yang dilakukan melalui aplikasi:

1. Tenant: 75%
2. Libro (pengelola kantin): 20%
3. U-Teen (pengembang sistem): 5%

Skema ini dirancang agar tenant tetap mendapatkan keuntungan terbesar sebagai penyedia jasa utama, sementara U-Teen memperoleh imbal hasil yang cukup untuk mendukung keberlanjutan pengembangan aplikasi. Persentase ini juga memungkinkan Libro menjalankan fungsi pengelolaan dan pengawasan tanpa terganggu beban operasional tambahan.

3.2.3. Integrasi Sistem Pembayaran Digital

Transaksi di dalam aplikasi menggunakan sistem pembayaran digital melalui Midtrans, yang mengakomodasi metode seperti *GoPay* dan *OVO*. Biaya yang dikenakan oleh masing-masing metode pembayaran adalah:

1. GoPay: 1,5% per transaksi
2. OVO: 2,9% per transaksi

Dengan asumsi distribusi transaksi digital terbagi secara merata (50% GoPay dan 50% OVO), maka total biaya operasional pembayaran akan dikurangkan dari pendapatan bruto yang diterima Libro dan U-Teen. Biaya ini sepenuhnya dibebankan pada komponen margin sistem dan tidak mengurangi pendapatan tenant.

3.2.4. Strategi Harga Berkelanjutan

Strategi harga yang diterapkan disusun untuk menjamin keberlanjutan operasional dan pengembangan aplikasi jangka panjang. Pendapatan yang diterima oleh U-Teen dari hasil bagi hasil transaksi akan digunakan untuk mendanai pemeliharaan sistem, pengembangan fitur tambahan, serta pengujian berkelanjutan di lingkungan kampus.

Seiring meningkatnya adopsi pengguna dan frekuensi transaksi, pendapatan sistem juga akan meningkat secara proporsional. Hal ini memungkinkan U-Teen untuk tetap relevan dan memberikan layanan yang stabil tanpa membebani pengguna akhir secara langsung.

3.2.5. Potensi Ekspansi Pasar

Model harga yang sederhana, adil, dan fleksibel ini dirancang tidak hanya untuk memenuhi kebutuhan operasional di lingkungan UMN, tetapi juga untuk dapat direplikasi di institusi pendidikan lain dengan struktur operasional kantin yang serupa. Skema bagi hasil berbasis transaksi memungkinkan pengembang untuk memperluas sistem tanpa perlu investasi awal yang besar dari mitra kampus, menjadikannya sebagai model bisnis yang scalable dan adaptif.

3.3 Promosi Target Pasar

3.3.1 Target Pasar

Target pasar utama dari aplikasi *U-Teen* adalah seluruh civitas akademika Universitas Multimedia Nusantara (UMN) yang menjadi pengguna aktif kantin kampus. Ini meliputi mahasiswa, dosen, dan staf UMN sebagai konsumen, serta tenant kantin sebagai penyedia produk makanan dan minuman.

Kelompok pengguna ini dipilih karena memiliki frekuensi interaksi yang tinggi dengan layanan kantin, serta menunjukkan kebutuhan nyata akan sistem pemesanan dan pembayaran yang lebih efisien, khususnya di jam sibuk saat waktu istirahat terbatas.

Selain itu, tenant kantin juga merupakan segmen penting dalam ekosistem aplikasi ini, karena mereka menjadi pelaksana langsung operasional sistem melalui penerimaan dan pengolahan pesanan digital. Dengan memberikan manfaat nyata pada kedua sisi (konsumen dan penjual), U-Teen menghadirkan nilai solusi yang terintegrasi.

Meskipun pengembangan awal difokuskan pada lingkungan UMN sebagai *pilot project*, aplikasi U-Teen memiliki potensi besar untuk diperluas ke kampus-kampus lain yang menghadapi masalah serupa. Skalabilitas model bisnis, arsitektur sistem modular, serta fleksibilitas

dalam integrasi menjadikan aplikasi ini layak untuk dijadikan sebagai solusi digital nasional di sektor layanan kantin pendidikan tinggi.

3.3.2 Metode Promosi

Strategi promosi U-Teen dirancang untuk menjangkau target pengguna secara langsung, memanfaatkan saluran komunikasi internal kampus serta pendekatan interaksi sosial yang efektif. Metode yang digunakan meliputi:

3.3.2.1 Blast Email

Kampanye promosi akan dilakukan melalui pengiriman *blast email* ke seluruh civitas akademika UMN menggunakan sistem email resmi kampus. Email ini akan berisi:

1. Penjelasan manfaat penggunaan aplikasi U-Teen
2. Panduan unduhan dan registrasi
3. Informasi fitur unggulan dan cara penggunaan

Langkah ini bertujuan untuk menjangkau pengguna secara serentak dan memberikan informasi formal dengan kredibilitas institusional.

3.3.2.2 Pengumuman Melalui Aplikasi Union

Aplikasi *Union* yang sudah menjadi platform utama informasi kampus UMN juga akan digunakan sebagai saluran promosi selain email. Melalui notifikasi *in-app*, pengguna Union akan diarahkan untuk mencoba U-Teen sebagai alternatif pemesanan makanan yang lebih praktis dengan isi yang sama dengan blast email UMN.

3.3.2.3 Promosi Langsung di Area Kantin

Promosi juga dilakukan melalui pendekatan interpersonal antara tenant dan konsumen. Setiap transaksi yang dilakukan secara langsung akan menjadi momen untuk memperkenalkan aplikasi U-Teen, dengan tenant menanyakan secara aktif

kepada pelanggan apakah mereka sudah menggunakan aplikasi tersebut untuk memesan makanannya. Metode ini meniru strategi *word-of-mouth* yang digunakan di sektor ritel seperti Alfamart untuk menanyakan kartu member.

3.4 Tahapan Pekerjaan yang Dilakukan Dalam MBKM Kewirausahaan

Tabel 3.1 ini menggambarkan dan menjelaskan terkait alur pekerjaan, dimulai dari tahap perencanaan hingga evaluasi.

Tabel 3.1 Detail Pekerjaan yang Dilakukan Dalam MBKM Cluster Kewirausahaan

No.	Minggu	Proyek	Keterangan
1	1-2	Perancangan ide bisnis	Tahap brainstorming terkait perencanaan ide bisnis dan aplikasi.
2	2-4	Perencanaan pengembangan aplikasi	Riset mengenai bahasa dan IDE yang cocok digunakan untuk pemgembangan aplikasi.
3	5-7	Pengembangan Aplikasi Fitur Customer	Mengembangkan aplikasi U-Teen berdasarkan desain Figma yang diberikan oleh anggota tim.
4	8-10	Pengembangan Aplikasi Fitur Tenant	Mengembangkan aplikasi U-Teen berdasarkan desain Figma yang diberikan oleh anggota tim.
5	11-14	Optimasi UI dan fitur	Melakukan perombakan desain UI dan beberapa fitur agar lebih sesuai dengan kebutuhan pengguna.

3.5 Uraian Pelaksanaan Kerja Dalam MBKM Kewirausahaan

Bagian ini berupa penjelasan secara umum mengenai pekerjaan yang dilakukan penulis selama proses MBKM Cluster Kewirausahaan.

3.5.1 Proses Pelaksanaan

3.5.1.1 Proyek U-Teen

1) **Flutter**

Flutter merupakan sebuah *open-source UI toolkit* yang dikembangkan oleh Google, digunakan untuk membangun antarmuka aplikasi lintas platform (Android, iOS, Web, dan Desktop) dari satu basis kode (*single codebase*). Flutter menggunakan bahasa pemrograman Dart, yang dirancang untuk efisiensi pengembangan *client-side application* dengan performa mendekati *native*.

Arsitektur Flutter berbeda dari pendekatan tradisional seperti React Native atau Ionic. Alih-alih menggunakan *bridge* untuk menjembatani komponen native dengan JavaScript, Flutter merender UI-nya sendiri melalui mesin grafis *Skia*, sehingga mampu menghasilkan tampilan yang konsisten di berbagai platform dengan latensi yang rendah dan kecepatan rendering tinggi. Hal ini menjadikan Flutter lebih stabil dalam hal animasi dan *UI performance*, sekaligus mengurangi ketergantungan terhadap komponen *OEM* dari masing-masing platform.

Selain itu, Flutter memiliki fitur hot reload, yang memungkinkan pengembang untuk melihat hasil perubahan kode secara langsung tanpa perlu membangun ulang (*full rebuild*) aplikasi. Fitur ini sangat membantu dalam proses pengembangan yang iteratif dan eksperimental seperti proyek U-Teen, di mana proses pengujian dan perbaikan tampilan dilakukan secara berulang setiap saat.

Dalam pengembangan U-Teen, Flutter dipilih karena kebutuhan utama aplikasi adalah efisiensi lintas platform untuk Android dan iOS, kecepatan iterasi desain, serta integrasi sistem yang relatif kompleks. Flutter memudahkan kami untuk membangun antarmuka yang responsif, melakukan pengolahan data dinamis, dan memfasilitasi komunikasi dua arah dengan *Firebase Firestore* sebagai backend.

Seluruh proses pengembangan aplikasi U-Teen dari sisi antarmuka pengguna, navigasi antar layar, pengelolaan *state*, hingga implementasi logika bisnis dan pemrosesan data ditangani secara penuh oleh penulis. Pengembangan dilakukan dengan pendekatan Flutter secara menyeluruh, di mana penulis membangun dan mengelola seluruh struktur proyek yang terdiri dari berbagai komponen utama, seperti layar (*screens*), komponen antarmuka modular (*widgets*), manajemen data dan status aplikasi (*providers*), serta logika layanan dan utilitas pendukung. Struktur kode dirancang agar modular, scalable, dan mudah dipelihara untuk kebutuhan *update* atau perawatan di masa depan. Seluruh implementasi ini disusun berdasarkan pola arsitektur MVVM agar sistem dapat berjalan secara efisien dan responsif di berbagai jenis perangkat.

2) **Instalasi**

Tahap pertama dalam pengerjaan proyek U-Teen adalah dengan melakukan instalasi terhadap Flutter berbagai *environment* pendukung, yang terdiri dari Flutter 3.29.2 (stable), Dart 3.7.2, DevTools 2.42.3, Java OpenJDK Runtime Environment Temurin-17.0.14+7, Dart SDK version: 3.7.2 (stable), git version 2.48.1.windows, gradle-8.9-bin, serta Command-line tools. Berbagai *environment* ini harus dipastikan bahwa merupakan versi *latest release* yang *stable* (tidak ada bug), serta versi yang saling dapat mendukung dan terhubung dengan tools lainnya.

3) Inisialisasi dan Login

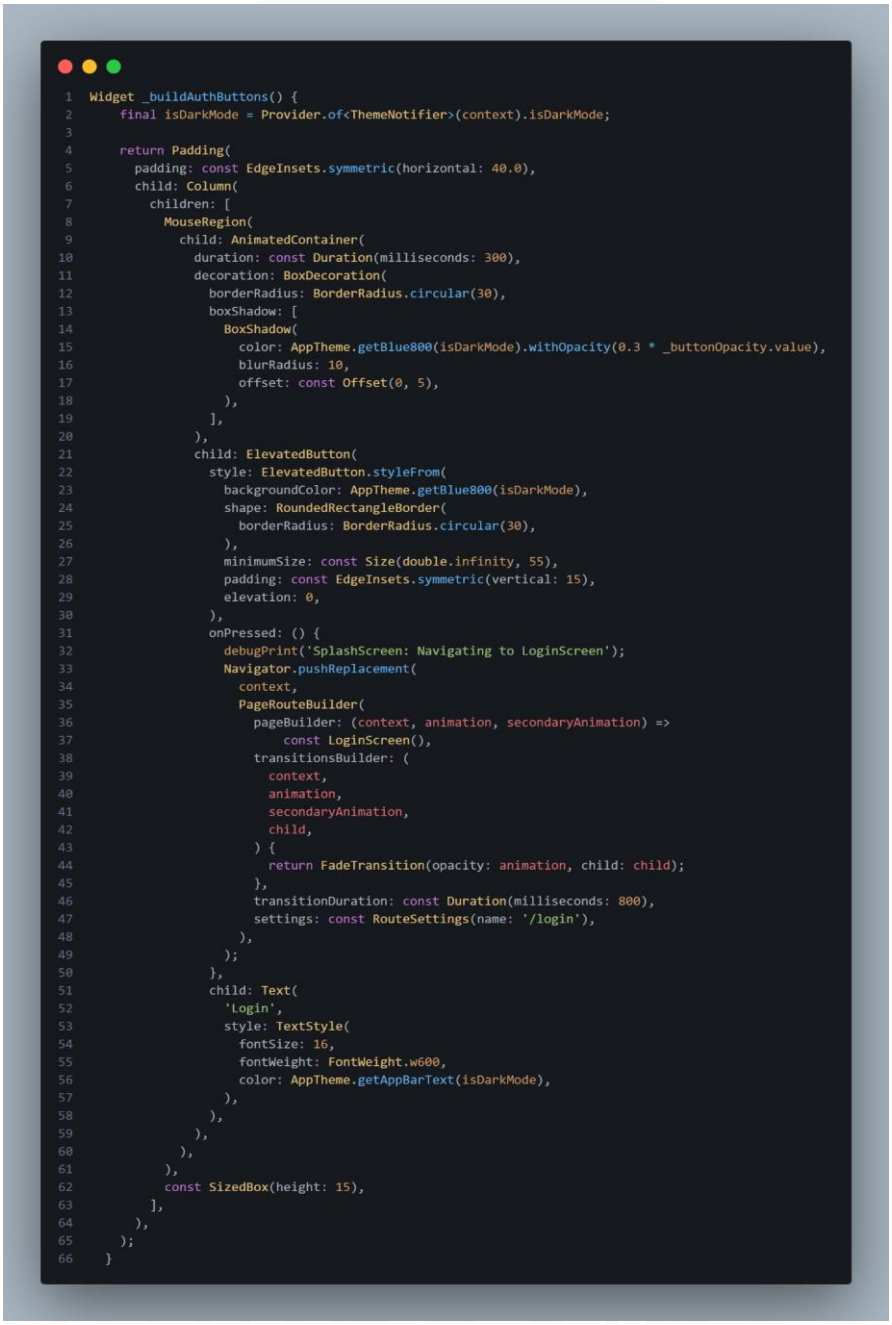
```dart
runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (context) => AuthProvider()),
        Provider(create: (context) => AuthService()),
        ChangeNotifierProvider(create: (context) => CartProvider()),
        ChangeNotifierProvider(create: (context) => FavoriteProvider()),
        ChangeNotifierProvider(create: (context) => FoodProvider()),
        ChangeNotifierProvider(create: (context) => NotificationProvider()),
        ChangeNotifierProvider(
          create: (context) => OrderProvider(
            Provider.of<NotificationProvider>(context, listen: false),
          ),
        ),
        ChangeNotifierProvider(
          create: (context) => RatingProvider(
            Provider.of<OrderProvider>(context, listen: false),
          ),
        ),
        ChangeNotifierProvider(create: (context) => ThemeNotifier()),
      ],
      child: const MyApp(),
    ),
  );
```

Gambar 3.1 Inisiasi Provider sebelum aplikasi dibuka

Bagian awal dari aplikasi U-Teen dimulai pada Gambar 3.1. dengan proses inisialisasi sistem serta konfigurasi manajemen status (state management) menggunakan pendekatan multiprovider. Pendekatan ini memungkinkan pengelolaan berbagai status aplikasi secara terpisah namun tetap terintegrasi, sehingga pengembangan fitur menjadi lebih modular, terstruktur, dan mudah untuk dikembangkan di masa depan.

Seluruh provider yang digunakan dalam aplikasi dideklarasikan sekaligus melalui struktur multiprovider, yang mencakup sejumlah penyedia data utama seperti AuthProvider, CartProvider, FavoriteProvider, FoodProvider, NotificationProvider, OrderProvider, RatingProvider, dan ThemeNotifier. Masing-masing provider diatur menggunakan pendekatan

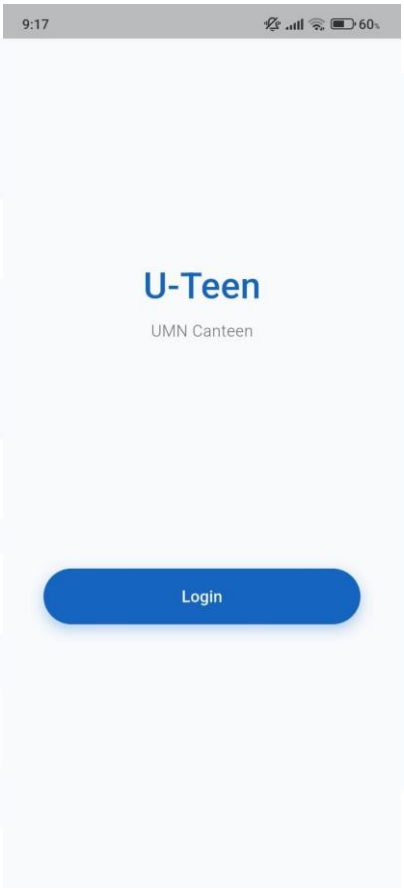
ChangeNotifierProvider, yang berarti setiap perubahan data dalam provider akan secara otomatis memberi notifikasi kepada komponen antarmuka pengguna yang relevan. Beberapa provider bersifat bergantung pada provider lainnya, seperti OrderProvider yang mengakses NotificationProvider, serta RatingProvider yang bergantung pada data dari OrderProvider. Dependensi tersebut dikelola secara efisien melalui akses konteks tanpa mendengarkan perubahan secara langsung, untuk menghindari pemanggilan ulang antarmuka yang tidak perlu.



```dart
Widget _buildAuthButtons() {
    final isDarkMode = Provider.of<ThemeNotifier>(context).isDarkMode;

    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 40.0),
      child: Column(
        children: [
          MouseRegion(
            child: AnimatedContainer(
              duration: const Duration(milliseconds: 300),
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(30),
                boxShadow: [
                  BoxShadow(
                    color: AppTheme.getBlue800(isDarkMode).withOpacity(0.3 * _buttonOpacity.value),
                    blurRadius: 10,
                    offset: const Offset(0, 5),
                  ),
                ],
              ),
              child: ElevatedButton(
                style: ElevatedButton.styleFrom(
                  backgroundColor: AppTheme.getBlue800(isDarkMode),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(30),
                  ),
                  minimumSize: const Size(double.infinity, 55),
                  padding: const EdgeInsets.symmetric(vertical: 15),
                  elevation: 0,
                ),
                onPressed: () {
                  debugPrint('SplashScreen: Navigating to LoginScreen');
                  Navigator.pushReplacement(
                    context,
                    PageRouteBuilder(
                      pageBuilder: (context, animation, secondaryAnimation) =>
                          const LoginScreen(),
                      transitionsBuilder: (
                        context,
                        animation,
                        secondaryAnimation,
                        child,
                      ) {
                        return FadeTransition(opacity: animation, child: child);
                      },
                      transitionDuration: const Duration(milliseconds: 800),
                      settings: const RouteSettings(name: '/login'),
                    ),
                  );
                },
                child: Text(
                  'Login',
                  style: TextStyle(
                    fontSize: 16,
                    fontWeight: FontWeight.w600,
                    color: AppTheme.getAppBarText(isDarkMode),
                  ),
                ),
              ),
            ),
          ),
          const SizedBox(height: 15),
        ],
      ),
    );
  }
```

Gambar 3.2 Login button untuk navigasi ke LoginScreen

Gambar 3.2 menunjukkan ketika pengguna menekan tombol login, aplikasi melakukan navigasi menuju halaman login.



Gambar 3.3 Tampilan main.dart sebagai splash screen

Ini adalah Gambar 3.3 yang menunjukkan tampilan akhir dari main.dart yang berisi tampilan login button setelah selesai menjalankan berbagai macam bentuk animasi pada awal aplikasi dibuka. Diatas menampilkan nama aplikasi, serta di bagian bawahnya terdapat button login untuk navigasi user melakukan login.

```dart
WidgetsBinding.instance.addPostFrameCallback((_) async {
      debugPrint('LoginScreen: Checking AuthProvider');
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      // Rely on AuthProvider's constructor initialization
      if (authProvider.isLoggedIn && authProvider.user != null && mounted) {
        debugPrint('LoginScreen: User is logged in, email: ${authProvider.user!.email}');
        if (mounted) {
          Navigator.pushAndRemoveUntil(
            context,
            MaterialPageRoute(
              builder: (context) => authProvider.isSeller
                  ? const SellerHomeScreen()
                  : const HomeScreen(),
            ),
            (route) => false,
          );
        }
      } else {
        debugPrint('LoginScreen: No logged-in user, showing login UI');
        _emailController.clear();
        _passwordController.clear();
        _animationController.forward();
      }
    });
```

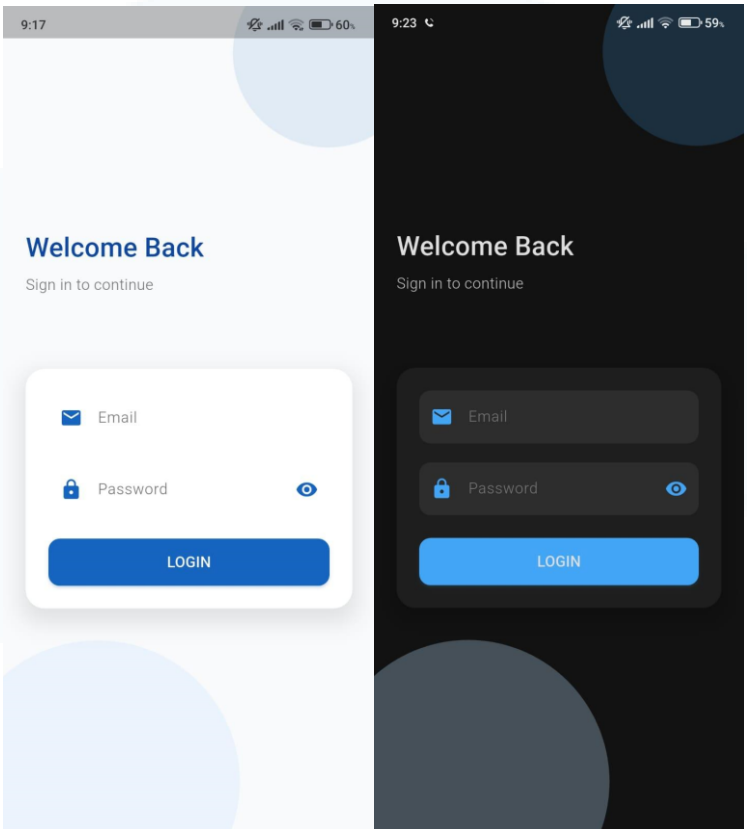
Gambar 3.4 Pengecekan status pada LoginScreen

Sistem login pada aplikasi U-Teen pada Gambar 3.4 diawali dengan pengecekan status autentikasi saat halaman login pertama kali dimuat. Melalui AuthProvider, sistem akan langsung mengarahkan pengguna ke halaman utama jika sudah login sebelumnya, berdasarkan perannya sebagai konsumen atau penjual. Jika belum login, halaman login akan ditampilkan lengkap dengan animasi transisi UI.

```dart
Future<void> _handleLogin() async {
    if (!_formKey.currentState!.validate()) return;

    setState(() => _isLoading = true);

    final isDarkMode = Provider.of<ThemeNotifier>(context, listen: false).isDarkMode;

    try {
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      final authService = AuthService();

      final email = _emailController.text.trim();
      final password = _passwordController.text.trim();

      debugPrint('LoginScreen: Attempting login for $email');
      final emailValidationError = authService.validateEmail(email);
      if (emailValidationError != null) {
        _showSnackBar(emailValidationError, AppTheme.getSnackBarError(isDarkMode));
        setState(() => _isLoading = false);
        return;
      }

      // Use AuthProvider.login directly
      final success = await authProvider.login(
        email: email,
        password: password,
        context: context,
      );

      if (!mounted) return;

      if (success && authProvider.user != null) {
        debugPrint('LoginScreen: Login successful, navigating to home');
        await Future.delayed(const Duration(milliseconds: 500));
        if (mounted) {
          Navigator.pushAndRemoveUntil(
            context,
            MaterialPageRoute(
              builder: (context) => authProvider.isSeller
                  ? const SellerHomeScreen()
                  : const HomeScreen(),
            ),
            (route) => false,
          );
        }
```

Gambar 3.5 Penanganan Login

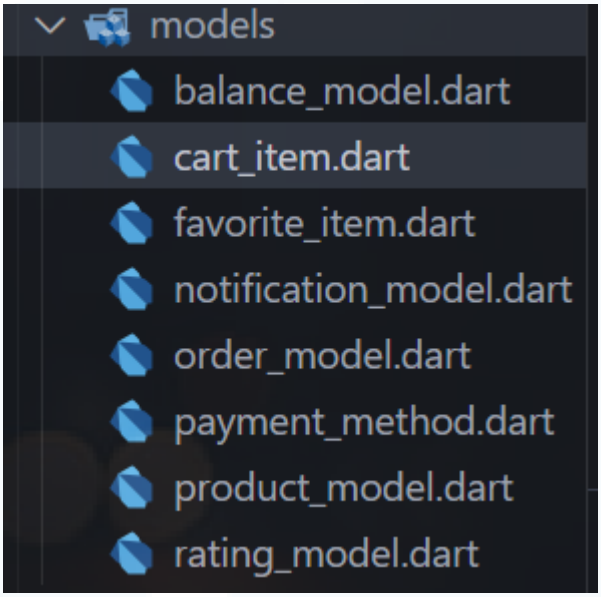
Proses login pada Gambar 3.5 dijalankan melalui fungsi handleLogin(), yang mencakup validasi input, pemrosesan kredensial, serta pemanggilan metode login() dari AuthProvider. Validasi tambahan terhadap format email dilakukan menggunakan AuthService. Jika login berhasil, pengguna akan diarahkan ke halaman utama.



Gambar 3.6 Tampilan LoginScreen

Tampilan dari LoginScreen pada Gambar 3.6 yang berfungsi memverifikasi user yang sudah terdaftar di *Firebase Firestore*. Pengguna bisa login apabila sudah memiliki akun UMN, sehingga tidak perlu melakukan pendaftaran akun. Pengguna hanya perlu mengisi email serta password saja untuk dapat mengakses aplikasi U-Teen.

4) **Penyimpanan Data**



Gambar 3.7 Struktur Folder Model

Gambar 3.7 ini menampilkan daftar file model yang digunakan dalam aplikasi. Masing-masing file berfungsi untuk merepresentasikan satu entitas data spesifik, seperti order_model untuk pemesanan, notification_model untuk notifikasi, dan seterusnya. Pemisahan file ini dilakukan agar pengelolaan data lebih terorganisir dan memudahkan pemeliharaan aplikasi dalam jangka panjang. Namun karena isinya yang sangat banyak dan lebih bersifat repetitif, maka dibawah ini penulis mengambil *file* dari order_model sebagai contoh *code* yang juga diterapkan ke semua *file* model lainnya.

```
class Order {
  final String id;
  final DateTime orderTime;
  final DateTime pickupTime;
  final List<OrderItem> items;
  final String status;
  final String paymentMethod;
  final String merchantName;
  final String merchantEmail;
  final String customerName;
  final String? cancellationReason;
  final String? notes;
  final DateTime? completedTime;
  final DateTime? cancelledTime;
  final int? _foodRating;
  final int? _appRating;
  final String? _foodNotes;
  final String? _appNotes;
  final DateTime createdAt;
  final DateTime? readyAt;
  final DateTime? completedAt;
  final DateTime? cancelledAt;
```

Gambar 3.8 Deklarasi Variabel

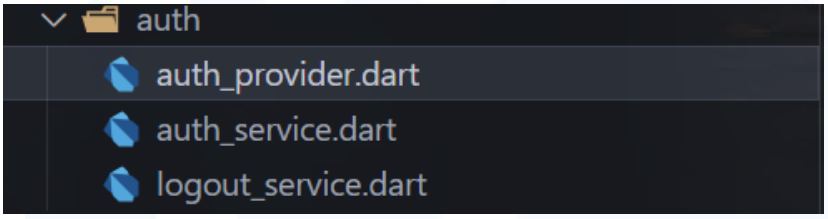
Gambar 3.8 ini menunjukkan potongan kode ini menunjukkan deklarasi atribut-atribut utama pada kelas Order. Atribut seperti id, orderTime, pickupTime, items, status, dan paymentMethod menyimpan informasi penting terkait pesanan yang dibuat oleh pengguna. Selain data utama, juga terdapat informasi tambahan seperti alasan pembatalan, waktu pesanan diselesaikan, serta rating dan catatan dari pengguna terhadap makanan atau aplikasi.

```dart
Order({
    required this.id,
    required this.orderTime,
    required this.pickupTime,
    required this.items,
    required this.paymentMethod,
    required this.merchantName,
    required this.merchantEmail,
    required this.customerName,
    required this.createdAt,
    required this.readyAt,
    required this.completedAt,
    required this.cancelledAt,
    this.status = 'pending',
    this.cancellationReason,
    this.notes,
    this.completedTime,
    this.cancelledTime,
    int? foodRating,
    int? appRating,
    String? foodNotes,
    String? appNotes,
  })  : _foodRating = foodRating,
        _appRating = appRating,
        _foodNotes = foodNotes,
        _appNotes = appNotes;

  int? get foodRating => _foodRating;
  int? get appRating => _appRating;
  String? get foodNotes => _foodNotes;
  String? get appNotes => _appNotes;

  double get totalPrice {
    return items.fold(0, (sum, item) => sum + (item.price * item.quantity));
  }
```

Gambar 3.9 Konstruktor Order Class

Pada bagian Gambar 3.9 ini ditampilkan konstruktor kelas Order yang mengatur bagaimana nilai-nilai awal ditetapkan saat objek pesanan dibuat. Konstruktor ini juga mencakup logika untuk atribut-atribut opsional seperti foodRating dan appRating. Selain itu, terdapat metode totalPrice yang secara otomatis menghitung total biaya pesanan berdasarkan daftar item yang dipesan dan kuantitasnya. Fungsi ini sangat penting untuk menampilkan informasi transaksi secara real-time kepada pengguna.

5) **Autentikasi Akun**



Gambar 3.10 Struktur Folder Auth

Folder auth pada Gambar 3.10 ini berfungsi sebagai pusat pengelolaan autentikasi dalam aplikasi U-Teen, mencakup proses login, logout, serta manajemen status pengguna selama sesi berlangsung. Auth Provider berfungsi sebagai sistem autentikasi pengguna dengan mengelola jenis pengguna (penjual atau pembeli) serta data pribadi seperti nama dan NIM.Auth Service berfungsi sebagai autentikasi kecocokan email dan password yang terdaftar di Firebase. Sedangkan logout serbice berfungsi sebagai sistem agar pengguna bisa melakukan logout.

a) Auth Provider

Manajemen autentikasi dan status pengguna dalam aplikasi U-Teen dikendalikan melalui kelas AuthProvider, yang merupakan turunan dari ChangeNotifier dan bertanggung jawab sebagai penyedia (provider) status login, jenis pengguna, serta data identitas pengguna selama sesi berjalan.

```dart
AuthProvider() {
    debugPrint('AuthProvider: Constructor called');
    _initializationFuture = initialize();
  }

  Future<void> initialize() async {
    if (_isInitialized) {
      debugPrint('AuthProvider: Already initialized, skipping');
      return;
    }
    debugPrint('AuthProvider: Starting initialization');
    _isInitializing = true;
    try {
      await _loadUserData();
      _isInitialized = true;
      debugPrint('AuthProvider: Initialization complete');
    } catch (e) {
      debugPrint('AuthProvider: Initialization error: $e');
      throw Exception('Gagal menginisialisasi AuthProvider: $e');
    } finally {
      _isInitializing = false;
      notifyListeners();
    }
    _firebaseAuth.authStateChanges().listen((fb.User? firebaseUser) async {
      debugPrint('AuthProvider: authStateChanges triggered, user: ${firebaseUser?.email}');
      if (firebaseUser == null) {
        if (_isLoggedIn && !_isLoggingOut && !_isInitializing) {
          await logout();
        }
      } else if (_user == null || _user!.email != firebaseUser.email) {
        await _loadUserData();
      }
    });
  }
```

Gambar 3.11 Inisialisasi dan Pemantauan Status Login

Kode Gambar 3.11 di atas menunjukkan bahwa setiap kali AuthProvider diinisialisasi, ia akan memuat data pengguna melalui initialize(). Sistem juga mengecek perubahan status login secara real-time menggunakan authStateChanges(). Jika pengguna keluar, sistem akan otomatis menjalankan logout() dan menghapus status sesi yang tersimpan.

```dart
Future<bool> login({
    required String email,
    required String password,
    required BuildContext context,
  }) async {
    try {
      debugPrint('AuthProvider: Starting login for $email');
      _isInitializing = true;
      final user = await _authService.login(email, password);
      if (user == null) {
        debugPrint('AuthProvider: Login failed, no user returned');
        _isInitializing = false;
        return false;
      }
      _user = user;
      await _saveToPrefs(user);
      _isLoggedIn = true;
      _isSeller = user.userType == 'seller';
      _customerNim = !_isSeller ? user.nim : null;
      _sellerNim = _isSeller ? user.nim : null;
      _customerProdi = !_isSeller ? user.prodi : null;
      _customerAngkatan = !_isSeller ? user.angkatan : null;
      final cartProvider = Provider.of<CartProvider>(context, listen: false);
      await cartProvider.initialize(user.email);
      debugPrint('AuthProvider: Logged in user: ${user.email}, type: ${user.userType}');
      notifyListeners();
      _isInitializing = false;
      return true;
```

Gambar 3.12 Proses Login Pengguna

Fungsi login() pada Gambar 3.12 bertanggung jawab dalam memanggil metode login dari AuthService, lalu menyimpan informasi pengguna ke dalam shared preferences, dan memperbarui status aplikasi menjadi login. Status pengguna akan disesuaikan dengan peran (seller atau customer) untuk kebutuhan personalisasi tampilan dan akses fitur.

```dart
Future<bool> logout() async {
    if (_isLoggingOut) {
      debugPrint('AuthProvider: Logout already in progress, skipping');
      return false;
    }
    _isLoggingOut = true;
    try {
      debugPrint('AuthProvider: Attempting logout');
      await _firebaseAuth.signOut().catchError((e) {
        debugPrint('AuthProvider: Sign out error: $e');
      });
      await _firebaseAuth.setPersistence(fb.Persistence.NONE).catchError((e) {
        debugPrint('AuthProvider: Set persistence error: $e');
      });
      final prefs = await SharedPreferences.getInstance();
      await prefs.clear();
      _user = null;
      _isLoggedIn = false;
      _isSeller = false;
      _customerNim = null;
      _sellerNim = null;
      _customerProdi = null;
      _customerAngkatan = null;
      debugPrint('AuthProvider: Logged out successfully');
      notifyListeners();
      return true;
```

Gambar 3.13 Fungsi Logout

Fungsi Gambar 3.13 ini digunakan untuk keluar dari akun yang sedang aktif, menghapus semua data lokal pengguna, dan mengatur ulang status autentikasi di seluruh aplikasi. Fungsi ini memastikan bahwa tidak ada data pengguna yang tertinggal setelah sesi berakhir.

b) Auth Service

Kelas AuthService berfungsi sebagai penghubung antara sistem autentikasi Firebase dan data pengguna yang tersimpan di Firestore. Kelas ini dipanggil oleh AuthProvider untuk melakukan proses login dan validasi input.

```dart
class AuthService {
  final fb.FirebaseAuth _auth = fb.FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<User?> login(String email, String password) async {
    try {
      // Authenticate with Firebase
      final credential = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      ).catchError((error) {
        String errorMessage;
        switch (error.code) {
          case 'user-not-found':
            errorMessage = 'No user found for this email';
            break;
          case 'wrong-password':
            errorMessage = 'Incorrect password';
            break;
          case 'invalid-email':
            errorMessage = 'Invalid email format';
            break;
          case 'user-disabled':
            errorMessage = 'This account has been disabled';
            break;
          default:
            errorMessage = 'Login failed: $error';
        }
        debugPrint('AuthService login error: $errorMessage');
        throw Exception(errorMessage);
      });

      if (credential.user == null) {
        throw Exception('Login failed: No user found');
      }

      // Fetch user data from Firestore
      final doc = await _firestore
          .collection('users')
          .doc(credential.user!.uid)
          .get();

      if (!doc.exists) {
        throw Exception('User data not found in Firestore for UID: ${credential.user!.uid}');
      }

      return User.fromFirestore(doc);
    } catch (e) {
      debugPrint('AuthService login error: $e');
      throw e; // Rethrow for AuthProvider to handle
    }
  }
```

Gambar 3.14 Login dan Otentikasi Firebase

Fungsi login() pada Gambar 3.14 ini mengautentikasi pengguna dengan menggunakan FirebaseAuth. Setelah berhasil login, sistem mengambil data lengkap

pengguna dari koleksi users di Firestore berdasarkan UID yang diberikan oleh Firebase. Proses ini memisahkan autentikasi dan otorisasi, sehingga data pengguna tetap dapat dikelola secara fleksibel di luar sistem Firebase Authentication.

```dart
String? validateEmail(String? email) {
    if (email == null || email.isEmpty) {
      return 'Email is required';
    }
    final emailRegex = RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$');
    if (!emailRegex.hasMatch(email)) {
      return 'Enter a valid email address';
    }
    return null;
  }
```

Gambar 3.15 Validasi Format Email

Sebelum proses login dijalankan pada Gambar 3.15, sistem melakukan validasi terhadap format email untuk mencegah terjadinya error saat melakukan autentikasi. Validasi ini memastikan bahwa input pengguna telah sesuai dengan pola email yang benar dan mengurangi kemungkinan kegagalan login akibat kesalahan pengetikan.

c) Logout Service

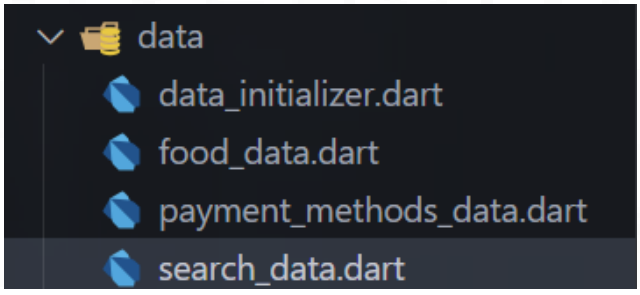
Proses logout pengguna pada aplikasi U-Teen dikemas dalam kelas LogoutService yang tidak hanya mengatur proses keluar dari akun, tetapi juga menjaga kenyamanan interaksi pengguna melalui dialog konfirmasi dan pembersihan data yang tersimpan. Pendekatan ini bertujuan menjaga integritas data pengguna dan konsistensi alur navigasi setelah sesi pengguna berakhir.

```dart
static Future<bool> logout(BuildContext context) async {
    try {
      // Close all dialogs first
      Navigator.of(context, rootNavigator: true).popUntil((route) => route.isFirst);

      // Get providers before async operations
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      final foodProvider = Provider.of<FoodProvider>(context, listen: false);

      // Perform logout
      final logoutSuccess = await authProvider.logout();
      if (!logoutSuccess) {
        debugPrint('LogoutService: AuthProvider.logout returned false');
        throw Exception('Failed to logout from authentication service');
      }

      // Clear food products
      foodProvider.clearProducts();

      // Ensure context is still mounted before navigation
      if (context.mounted) {
        // Navigate to login screen with replacement
        await Navigator.of(context, rootNavigator: true).pushAndRemoveUntil(
          MaterialPageRoute(builder: (context) => const LoginScreen()),
          (Route<dynamic> route) => false,
        );
      }

      return true;
```

Gambar 3.16 Proses Logout dan Penghapusan State

Fungsi logout() pada Gambar 3.16 ini menangani proses keluar dari sistem melalui pemanggilan metode logout() pada AuthProvider. Setelah pengguna keluar, sistem juga membersihkan produk makanan yang tersimpan di state FoodProvider, untuk menghindari data yang tersisa dari sesi sebelumnya. Navigasi dilakukan ke halaman login dengan mengganti seluruh stack halaman sebelumnya untuk menghindari pengguna kembali ke halaman sebelumnya menggunakan tombol back.

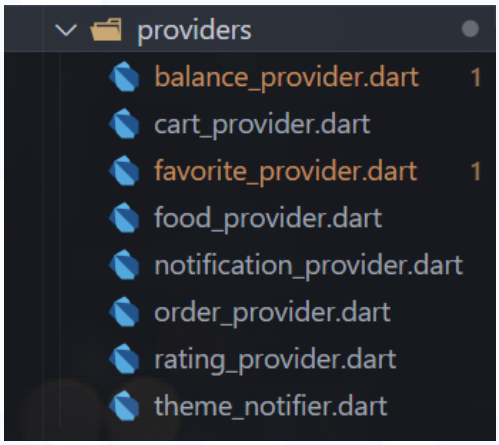
6) **Data**



Gambar 3.17 Folder Data

Folder data pada Gambar 3.17 berisi kumpulan logika dan fungsionalitas yang berperan dalam pengelolaan data pada aplikasi U-Teen, terutama dalam hal inisialisasi, pemrosesan, dan penyimpanan data ke Firebase Firestore. Beberapa data yang dikelola melalui folder ini meliputi daftar produk makanan, metode pembayaran, riwayat pencarian pengguna, serta data pendukung lainnya yang diperlukan untuk menjalankan aplikasi. Karena bagian ini bukan merupakan tanggung jawab langsung penulis dalam proyek (yang difokuskan pada pengembangan front-end dan back-end aplikasi), maka penulis tidak akan membahas lebih lanjut terhadap kode-kode yang terdapat di dalamnya.

7) **Providers**



Gambar 3.18 Folder Provider

Folder provider dalam Gambar 3.18 dalam proyek pengembangan aplikasi U-Teen berperan sebagai pusat manajemen (state management) yang mengatur logika bisnis dan data aplikasi secara terpusat. Folder ini berisi kelas-kelas yang mengimplementasikan pola Provider untuk menangani berbagai aspek fungsionalitas, seperti autentikasi pengguna, pengelolaan produk, keranjang belanja, pesanan, notifikasi, hingga pengaturan tema dan favorit.

a) Balance Provider

File balance_provider.dart mengatur logika perhitungan saldo yang diterima oleh merchant dari setiap transaksi yang berhasil, serta menyediakan mekanisme

untuk melakukan penarikan (withdrawal). Provider ini bertanggung jawab menyimpan riwayat pemasukan dan pengeluaran secara lokal dan di Firestore.

```dart
Future<void> addOrderIncome(my_orders.Order order) async {
    _isLoading = true;
    notifyListeners();

    try {
      final merchantEmail = order.merchantEmail;
      if (merchantEmail.isEmpty) return;

      if (!_balances.containsKey(merchantEmail)) {
        _balances[merchantEmail] = Balance(amount: 0.0, history: []);
      }

      final currentBalance = _balances[merchantEmail]!;
      final newAmount = currentBalance.amount + order.totalPrice;

      final history = BalanceHistory(
        id: 'income_${order.id}',
        date: DateTime.now(),
        type: 'income',
        amount: order.totalPrice,
        description: 'Income from order #${order.id}',
        orderId: order.id,
      );

      _balances[merchantEmail] = Balance(
        amount: newAmount,
        history: [...currentBalance.history, history],
      );

      await _saveBalance(merchantEmail);
    } catch (e) {
      debugPrint('Error adding order income: $e');
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }
```

Gambar 3.19 Menambahkan Pemasukan dari Order

Fungsi addOrderIncome() pada Gambar 3.19 digunakan untuk menambahkan pemasukan ke saldo merchant setiap kali terjadi transaksi. Informasi seperti total pendapatan, waktu transaksi, dan ID pesanan dicatat sebagai riwayat pemasukan. Data kemudian disimpan ke Firestore untuk menjaga sinkronisasi antara aplikasi dan server.

```dart
Future<bool> withdraw(String merchantEmail, double amount, String paymentMethod) async {
    if (!_balances.containsKey(merchantEmail) || amount > _balances[merchantEmail]!.amount) {
      return false;
    }

    _isLoading = true;
    notifyListeners();

    try {
      final currentBalance = _balances[merchantEmail]!;
      final newAmount = currentBalance.amount - amount;

      final history = BalanceHistory(
        id: 'withdraw_${DateTime.now().millisecondsSinceEpoch}',
        date: DateTime.now(),
        type: 'withdraw',
        amount: amount,
        description: 'Withdraw to $paymentMethod',
        paymentMethod: paymentMethod,
      );

      _balances[merchantEmail] = Balance(
        amount: newAmount,
        history: [...currentBalance.history, history],
      );

      await _saveBalance(merchantEmail);
      return true;
    } catch (e) {
      debugPrint('Error during withdraw: $e');
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }
```

Gambar 3.20 Penarikan Saldo oleh Merchant

Fungsi withdraw() pada Gambar 3.20 memungkinkan merchant untuk menarik saldo yang tersedia dengan menentukan jumlah dan metode pembayaran yang digunakan. Riwayat penarikan dicatat sebagai entri transaksi dengan tipe withdraw, dan jumlah saldo akan diperbarui setelah transaksi berhasil disimpan ke Firestore.

b) Cart Provider

Kelas CartProvider mengelola seluruh logika yang berkaitan dengan keranjang belanja pengguna. Provider ini berfungsi untuk menyimpan daftar produk yang ditambahkan ke keranjang, menghitung total harga, serta mengatur kuantitas item yang dibeli. Semua data disimpan ke Firestore agar tetap tersinkron saat pengguna berpindah perangkat atau sesi.

```dart
Future<void> addToCart(CartItem item) async {
    int? newItemIndex;
    try {
      debugPrint(
          'CartProvider: Adding to cart: ${item.name}, imgbase64 length: ${item.imgBase64.length}');
      if (item.name.isEmpty || item.sellerEmail.isEmpty) {
        throw Exception('Invalid cart item: name or sellerEmail is empty');
      }
      if (_userEmail == null) {
        debugPrint('CartProvider: Cannot add to cart: userEmail is null');
        throw Exception('User email not set');
      }
      final existingItemIndex = _cartItems.indexWhere(
        (i) => i.name == item.name && i.subtitle == item.subtitle,
      );
      if (existingItemIndex != -1) {
        _cartItems[existingItemIndex].quantity++;
        debugPrint('CartProvider: Incremented quantity for ${item.name}');
      } else {
        _cartItems.add(item);
        newItemIndex = _cartItems.length - 1;
        debugPrint('CartProvider: Added new item: ${item.name}');
      }
      await _saveCart();
      await Future.delayed(const Duration(milliseconds: 250));
      debugPrint('CartProvider: Notifying listeners after cart update');
      notifyListeners();
```

Gambar 3.21 Penambahan Produk ke Keranjang

Fungsi addToCart() pada Gambar 3.21 digunakan untuk menambahkan item ke dalam keranjang. Jika item sudah ada sebelumnya, maka jumlahnya akan ditambah. Jika belum ada, maka item baru akan dimasukkan ke dalam daftar. Setelah itu, data keranjang disimpan ulang ke Firestore.

```dart
void decreaseQuantity(CartItem item) {
    final existingItemIndex = _cartItems.indexWhere(
      (i) => i.name == item.name && i.subtitle == item.subtitle,
    );
    if (existingItemIndex != -1) {
      _cartItems[existingItemIndex].quantity--;
      if (_cartItems[existingItemIndex].quantity == 0) {
        _cartItems.removeAt(existingItemIndex);
      }
      _saveCart();
      notifyListeners();
    }
  }
```

Gambar 3.22 Pengurangan dan Penghapusan Item

Fungsi decreaseQuantity() menurunkan jumlah item dalam keranjang. Jika kuantitas sudah mencapai nol, item akan dihapus. Fungsi ini membantu pengguna menyesuaikan isi keranjangnya secara fleksibel dan real-time.

c) Favorite Provider

Provider ini memungkinkan pengguna menyimpan daftar makanan atau minuman favorit mereka agar mudah ditemukan dan dipesan kembali di kemudian hari.

```
Future<void> addToFavorites(FavoriteItem item) async {
    if (_userEmail == null || _userEmail!.isEmpty) {
      debugPrint('FavoriteProvider: Cannot add favorite, no user email');
      throw Exception('No user email');
    }
    if (item.name.isEmpty || item.imgBase64.isEmpty || !isValidBase64(item.imgBase64)) {
      debugPrint('FavoriteProvider: Invalid favorite item: name, imgBase64, or invalid Base64');
      throw Exception('Invalid favorite item');
    }
    if (!_favoriteItems.any((existingItem) =>
        existingItem.name == item.name && existingItem.imgBase64 == item.imgBase64)) {
      _favoriteItems.add(item);
      debugPrint('FavoriteProvider: Added to favorites: ${item.name}, imgBase64 length: ${item.imgBase64.length}');
      await _saveFavorites();
      notifyListeners();
    } else {
      debugPrint('FavoriteProvider: Item ${item.name} already in favorites');
    }
  }
```

Gambar 3.23 Penambahan Item ke Favorit

Fungsi addToFavorites() Gambar 3.23 digunakan untuk menambahkan item ke daftar favorit pengguna. Sistem akan terlebih dahulu memeriksa apakah item sudah ada untuk mencegah duplikasi. Setelah ditambahkan, data disimpan ke Firestore agar tetap tersimpan secara permanen.

```dart
Future<void> removeFromFavorites(FavoriteItem item) async {
    if (_userEmail == null || _userEmail!.isEmpty) {
      debugPrint('FavoriteProvider: Cannot remove favorite, no user email');
      return;
    }
    _favoriteItems.removeWhere((existingItem) =>
        existingItem.name == item.name && existingItem.imgBase64 == item.imgBase64);
    debugPrint('FavoriteProvider: Removed from favorites: ${item.name}');
    await _saveFavorites();
    notifyListeners();
  }
```

Gambar 3.24 Penghapusan Item dari Favorit

Fungsi removeFromFavorites() pada Gambar 3.24 menghapus item tertentu dari daftar favorit pengguna berdasarkan kecocokan nama dan gambar. Fungsi ini juga secara otomatis memperbarui data di Firestore dan memperbarui tampilan aplikasi.

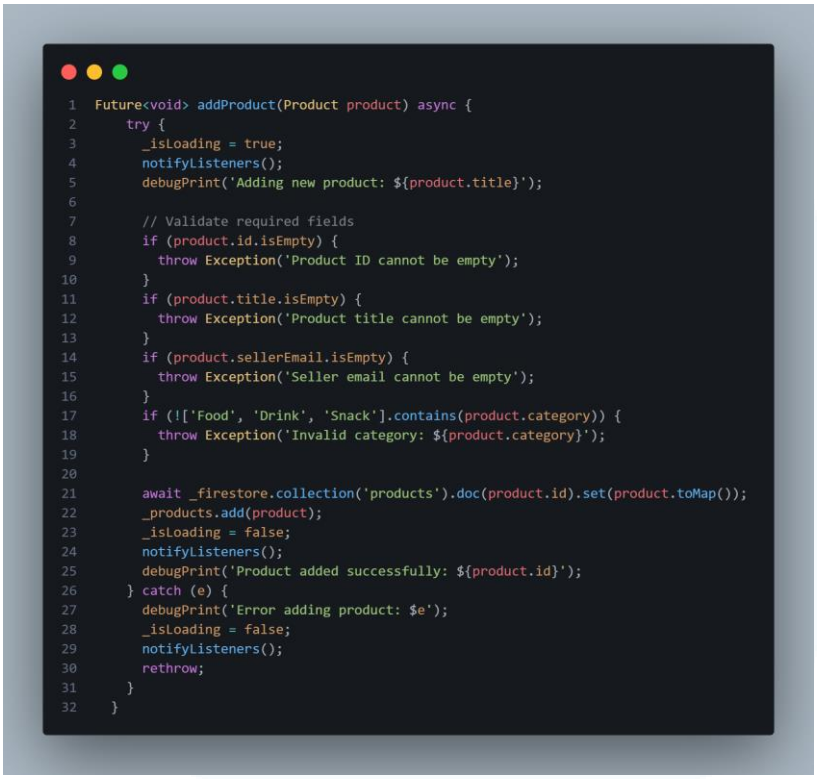
d) Food Provider

Provider ini memanfaatkan pola *state management* berbasis ChangeNotifier untuk menjembatani komunikasi antara tampilan (UI) dan database (*Firestore*), serta memastikan bahwa seluruh interaksi pengguna terhadap produk dapat dilakukan secara real-time dan terstruktur.

```dart
Future<void> loadProducts(BuildContext context, {String category = 'All', String? tenantName}) async {
    if (_isLoading || !_hasMore) return;
```

Gambar 3.25 Memuat dan Menyaring Produk

Fungsi loadProducts() pada Gambar 3.25 bertanggung jawab untuk mengambil daftar produk dari Firebase Firestore dengan menggunakan metode *pagination*. Selain itu, fungsi ini juga mendukung filter berdasarkan kategori (makanan, minuman, camilan) dan tenant tertentu. Proses pemuatan data dilakukan secara bertahap untuk mengurangi beban render sekaligus meningkatkan performa aplikasi.



```
Future<void> addProduct(Product product) async {
    try {
      _isLoading = true;
      notifyListeners();
      debugPrint('Adding new product: ${product.title}');

      // Validate required fields
      if (product.id.isEmpty) {
        throw Exception('Product ID cannot be empty');
      }
      if (product.title.isEmpty) {
        throw Exception('Product title cannot be empty');
      }
      if (product.sellerEmail.isEmpty) {
        throw Exception('Seller email cannot be empty');
      }
      if (!['Food', 'Drink', 'Snack'].contains(product.category)) {
        throw Exception('Invalid category: ${product.category}');
      }

      await _firestore.collection('products').doc(product.id).set(product.toMap());
      _products.add(product);
      _isLoading = false;
      notifyListeners();
      debugPrint('Product added successfully: ${product.id}');
    } catch (e) {
      debugPrint('Error adding product: $e');
      _isLoading = false;
      notifyListeners();
      rethrow;
    }
  }
```

Gambar 3.26 Menambahkan Produk Baru

Fungsi pada Gambar 3.26 ini memungkinkan seller (tenant) untuk menambahkan produk makanan atau minuman baru ke dalam sistem. Fungsi ini juga telah dilengkapi validasi awal untuk memastikan bahwa data produk yang dikirim ke Firestore tidak kosong atau salah format. Setelah produk ditambahkan, state lokal akan diperbarui dan UI akan merefleksikan perubahan tersebut.

```dart
Future<void> updateProduct(Product product) async {
    try {
      _isLoading = true;
      notifyListeners();
      debugPrint('Updating product: ${product.id}');

      await _firestore.collection('products').doc(product.id).update(product.toMap());
      final index = _products.indexWhere((p) => p.id == product.id);
      if (index != -1) {
        _products[index] = product;
        debugPrint('Product updated in local list at index: $index');
      } else {
        debugPrint('Product not found in local list');
      }
      _isLoading = false;
      notifyListeners();
      debugPrint('Product updated successfully: ${product.id}');
    } catch (e) {
      debugPrint('Error updating product: $e');
      _isLoading = false;
      notifyListeners();
      rethrow;
    }
  }
```

Gambar 3.27 Mengupdate Produk

Fungsi ini pada Gambar 3.27 digunakan untuk memperbarui informasi produk yang sudah ada dalam koleksi products pada Firestore. Setelah proses update dilakukan di database, data lokal (*state*) dalam aplikasi juga disinkronkan agar tampilan antarmuka pengguna mencerminkan data terbaru. Fungsi ini mencakup validasi terhadap eksistensi data produk di dalam daftar lokal, serta menangani status loading dan notifikasi kepada UI untuk memperbarui tampilan. Fungsionalitas ini penting terutama bagi tenant yang ingin mengedit harga, deskripsi, atau status ketersediaan produk mereka secara real-time.

```
Future<void> deleteProduct(String productId) async {
    try {
      _isLoading = true;
      notifyListeners();
      debugPrint('Deleting product: $productId');

      await _firestore.collection('products').doc(productId).delete();
      _products.removeWhere((p) => p.id == productId);
      _isLoading = false;
      notifyListeners();
      debugPrint('Product deleted successfully: $productId');
    } catch (e) {
      debugPrint('Error deleting product: $e');
      _isLoading = false;
      notifyListeners();
      rethrow;
    }
  }
```

Gambar 3.28 Menghapus Produk

Fungsi Gambar 3.28 ini bertugas untuk menghapus data produk dari Firestore berdasarkan productId. Setelah proses penghapusan di database berhasil, daftar produk lokal yang disimpan dalam provider juga diperbarui dengan menghapus produk yang bersangkutan. Fungsi ini memastikan bahwa data yang ditampilkan di aplikasi selalu konsisten dengan kondisi database. Proses ini dilakukan dengan pendekatan *try-catch* untuk menangani kemungkinan error, serta mengelola status loading agar pengguna mendapat umpan balik selama proses penghapusan berlangsung.

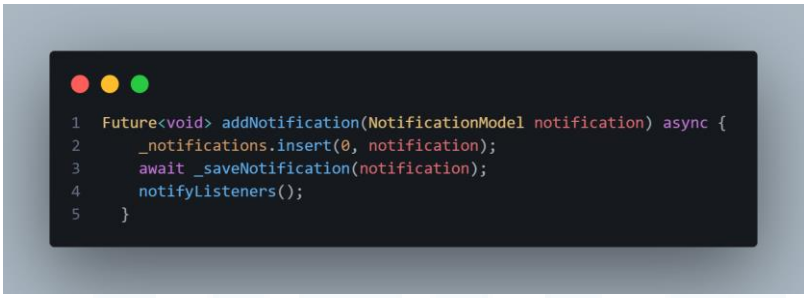
e) Notification Provider

Notification Provider adalah *state management class* yang bertanggung jawab untuk menangani seluruh alur notifikasi untuk pengguna dengan peran sebagai *customer*.

```
void _setupRealTimeListener() {
    if (_customerEmail == null) {
      debugPrint('NotificationProvider: No customer email set, skipping real-time listener');
      return;
    }
    _subscription?.cancel();
    _subscription = FirebaseFirestore.instance
        .collection('notifications')
        .where('payload.customerName', isEqualTo: _customerEmail)
        .orderBy('timestamp', descending: true)
        .snapshots()
        .listen((snapshot) {
          _notifications.clear();
          _notifications.addAll(snapshot.docs.map((doc) => NotificationModel.fromMap(doc.data())));
          _lastError = null;
          debugPrint('NotificationProvider: Real-time update: Loaded ${snapshot.docs.length} notifications for $_customerEmail');
          notifyListeners();
        }, onError: (error) {
          _lastError = error.toString();
          debugPrint('NotificationProvider: Error in real-time listener: $error');
          notifyListeners();
        });
  }
```

Gambar 3.29 Update Notifikasi dari FireStore

Fungsi Gambar 3.29 ini mengaktifkan *real-time stream listener* terhadap koleksi notifications di Firestore. Dengan ini, setiap perubahan data yang terjadi, misalnya penambahan notifikasi baru atau update status, akan langsung diperbarui pada UI pengguna secara otomatis tanpa perlu refresh manual.



```
Future<void> addNotification(NotificationModel notification) async {
    _notifications.insert(0, notification);
    await _saveNotification(notification);
    notifyListeners();
  }
```

Gambar 3.30 Menambahkan notifikasi baru

Fungsi Gambar 3.30 ini memungkinkan sistem untuk menambahkan notifikasi baru ke daftar notifikasi pengguna. Notifikasi akan langsung dimasukkan ke posisi teratas (*insert at 0*) dan disimpan di Firestore. Fungsi ini digunakan ketika aplikasi ingin mengirimkan notifikasi langsung ke pengguna sebagai hasil dari aksi tertentu, seperti pesanan yang diterima atau selesai diproses.

f) Order Provider

OrderProvider adalah kelas yang bertanggung jawab untuk mengelola seluruh proses pemesanan dalam aplikasi, baik dari sisi customer maupun seller. Fungsinya meliputi menyimpan daftar pesanan (orders), menambahkan pesanan baru, memperbarui status pesanan, menangani notifikasi terkait pesanan, serta mengelola logika withdrawal dan perhitungan saldo. Provider ini juga menyediakan real-time listener untuk memastikan perubahan status pesanan tersinkronisasi langsung dengan Firestore dan ditampilkan secara live pada UI.

```
Future<void> addOrder(order.Order order) async {
    try {
      _isLoading = true;
      notifyListeners();
      if (order.id.isEmpty || order.merchantEmail.isEmpty || order.customerName.isEmpty) {
        throw Exception('Data pesanan tidak valid: id, merchantEmail, atau customerName kosong');
      }
      debugPrint('OrderProvider: Adding order ${order.id}');
      _orders.insert(0, order);
      await _saveOrder(order);
      final notification = NotificationModel.fromOrder(order);
      // Send notification async without awaiting
      _notificationProvider.addNotification(notification).catchError((e) {
        debugPrint('OrderProvider: Gagal mengirim notifikasi untuk pesanan ${order.id}: $e');
      });
      debugPrint('OrderProvider: Added order ${order.id}');
    } catch (e) {
      debugPrint('OrderProvider: Error adding order ${order.id}: $e');
      _orders.removeWhere((o) => o.id == order.id);
      _lastError = 'Gagal menambah pesanan: $e';
      throw Exception('Gagal menambah pesanan: $e');
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }
```

Gambar 3.31 Menambahkan Pesanan Baru

Fungsi pada Gambar 3.31 ini digunakan untuk menambahkan pesanan baru ke dalam sistem. Data pesanan akan langsung dikirim dan disimpan ke Firebase Firestore, serta notifikasi otomatis dikirim ke user terkait.

```dart
Future<void> updateOrderStatus(String orderId, String newStatus, {String? reason}) async {
    final index = _orders.indexWhere((o) => o.id == orderId);
    if (index == -1) {
      debugPrint('OrderProvider: Pesanan $orderId tidak ditemukan');
      _lastError = 'Pesanan tidak ditemukan';
      throw Exception('Pesanan tidak ditemukan');
    }
    _isLoading = true;
    notifyListeners();
    try {
      debugPrint('OrderProvider: Updating order $orderId to status $newStatus');
      final now = DateTime.now();
      if (_orders[index].status == 'ready' && newStatus != 'ready') {
        _readyTimers[orderId]?.cancel();
        _readyTimers.remove(orderId);
        debugPrint('OrderProvider: Cancelled timer for order $orderId');
      }
      _orders[index] = _orders[index].copyWith(
        status: newStatus,
        cancellationReason: reason,
        completedTime: newStatus == 'completed' ? now : _orders[index].completedTime,
        cancelledTime: newStatus == 'cancelled' ? now : _orders[index].cancelledTime,
        readyAt: newStatus == 'ready' ? now : _orders[index].readyAt,
        completedAt: newStatus == 'completed' ? now : _orders[index].completedAt,
        cancelledAt: newStatus == 'cancelled' ? now : _orders[index].cancelledAt,
      );
      if (newStatus == 'ready') {
        _readyTimers[orderId] = Timer(const Duration(minutes: 2), () {
          _autoCompleteOrder(orderId);
        });
        debugPrint('OrderProvider: Started timer for order $orderId');
      }
      await _saveOrder(_orders[index]);
      // Send notification async without awaiting
      if (['ready', 'completed', 'cancelled'].contains(newStatus)) {
        _notificationProvider
            .addNotification(NotificationModel.fromOrder(_orders[index]))
            .catchError((e) {
          debugPrint('OrderProvider: Gagal mengirim notifikasi untuk pesanan $orderId: $e');
        });
      }
```

Gambar 3.32 Memperbarui Status Pesanan

Fungsi updateOrderStatus() pada Gambar 3.32 memiliki tanggung jawab utama dalam mengubah status dari suatu pesanan, seperti dari *pending* ke *processing*, *ready*, atau *completed*. Fungsi ini juga mengatur timer otomatis apabila status berubah menjadi *ready*; setelah dua menit, status akan berubah otomatis menjadi *completed* jika tidak ada intervensi dari pengguna. Selain itu, setiap perubahan status akan langsung disinkronkan ke Firestore dan diikuti oleh pengiriman notifikasi kepada pengguna yang bersangkutan, memastikan pengalaman pengguna yang informatif dan real-time.

```dart
Future<void> submitRatingAndCompleteOrder({
    required String orderId,
    required int foodRating,
    required int appRating,
    String? foodNotes,
    String? appNotes,
  }) async {
    final index = _orders.indexWhere((o) => o.id == orderId);
    if (index == -1) {
      debugPrint('Order: Pesanan $orderId tidak ditemukan');
      _lastError = 'Pesanan tidak ditemukan';
      throw Exception('Pesanan tidak ditemukan');
    }
    _isLoading = true;
    notifyListeners();
    try {
      debugPrint('Order: Submitting rating untuk order $orderId');
      final now = DateTime.now();
      _readyTimers[orderId]?.cancel();
      _readyTimers.remove(orderId);
      debugPrint('Order: Cancelled timer untuk order $orderId');
      _orders[index] = _orders[index].copyWith(
        status: 'completed',
        completedTime: now,
        completedAt: now,
        foodRating: foodRating,
        appRating: appRating,
        foodNotes: foodNotes,
        appNotes: appNotes,
      );
      await _saveOrder(_orders[index]);
      // Send notification async tanpa mengetahui
      _notificationProvider
          .addNotification(NotificationModel.fromOrder(_orders[index]))
          .catchError((e) {
        debugPrint('Order: Gagal menghasilkan notifikasi untuk pesanan $orderId: $e');
      });
      debugPrint('Order: Successfully submitted rating untuk order $orderId');
    } catch (e) {
      debugPrint('Order: Error submitting rating untuk order $orderId: $e');
      _lastError = 'Gagal mengirim rating: $e';
      throw Exception('Gagal mengirim rating: $e');
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }
```

Gambar 3.33 Rating dan Feedback Pesanan

Fungsi submitRatingAndCompleteOrder() pada Gambar 3.33 memberikan kemampuan kepada customer untuk menyelesaikan pesanan sekaligus memberikan penilaian terhadap kualitas makanan dan aplikasi. Selain mengatur status pesanan menjadi *completed*, fungsi ini juga mencatat skor penilaian dan komentar pengguna, kemudian menyimpannya ke Firestore. Informasi ini dapat digunakan untuk keperluan analitik dan evaluasi kualitas layanan. Proses ini juga mencakup pengiriman notifikasi sebagai umpan balik terhadap pihak merchant.

```dart
Future<void> addWithdrawal({
    required String merchantEmail,
    required double amount,
    required String method,
  }) async {
    try {
      _isLoading = true;
      notifyListeners();
      debugPrint('Trying: Adding withdrawal untuk $merchantEmail');
      final withdrawal = order.Order(
        id: _generateOrderId(),
        orderTime: DateTime.now(),
        pickupTime: DateTime.now(),
        items: [
          order.OrderItem(
            name: 'Withdrawal',
            imgBase64: 'assets/withdrawal.png',
            subtitle: 'Withdrawal ke $method',
            price: amount.round(),
            quantity: 1,
            sellerEmail: merchantEmail,
          )
        ],
        paymentMethod: method,
        merchantName: 'System',
        merchantEmail: merchantEmail,
        customerName: 'Withdrawal',
        status: 'processed',
        notes: 'Withdrawal ke $method',
        readyAt: null,
        completedAt: null,
        cancelledAt: null,
        createdAt: DateTime.now(),
      );
      _orders.insert(0, withdrawal);
      await _saveOrder(withdrawal);
      debugPrint('Order: Successfully added withdrawal untuk $merchantEmail');
    } catch (e) {
      debugPrint('Order: Error adding withdrawal: $e');
      _lastError = 'Gagal menambah penarikan: $e';
      throw Exception('Gagal menambah penarikan: $e');
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }
```

Gambar 3.34 Tarik Saldo Tenant

Fungsi addWithdrawal() pada Gambar 3.24 bertugas membuat entri khusus ke dalam sistem sebagai bukti transaksi penarikan dana oleh seller. Fungsi ini membentuk objek *Order* dengan item khusus bernama “Withdrawal” dan memasukkannya ke dalam daftar pesanan. Transaksi ini disimpan ke Firestore dan ditampilkan pada halaman transaksi merchant, sehingga pengguna dapat melacak riwayat penarikan dana secara transparan dan terstruktur.

```
int getAvailableBalanceForMerchant(String merchantEmail) {
    double earnings = 0;
    double withdrawals = 0;

    for (var order in _orders) {
      if (order.merchantEmail == merchantEmail) {
        if (order.status == 'completed' && order.customerName != 'Withdrawal') {
          earnings += order.totalPrice;
        } else if (order.customerName == 'Withdrawal') {
          withdrawals += order.totalPrice;
        }
      }
    }

    return (earnings - withdrawals).round();
  }
```

Gambar 3.35 Menghitung Saldo dan Transaksi Merchant

Fungsi getAvailableBalanceForMerchant() pada Gambar 3.35 menghitung saldo tersedia bagi setiap merchant dengan menjumlahkan seluruh pemasukan dari pesanan yang telah *completed* dan mengurangi total nilai transaksi penarikan (*withdrawal*). Dengan cara ini, aplikasi dapat menyajikan informasi saldo secara real-time dan akurat kepada merchant, yang berguna dalam pengambilan keputusan finansial dan pelaporan keuangan internal.

g) Rating Provider

RatingProvider adalah class yang bertugas mengelola proses penilaian (rating) dan umpan balik (*feedback*) terhadap pesanan yang telah selesai (*completed*). Provider ini mengambil data dari OrderProvider untuk menghitung rata-rata rating makanan, menampilkan komentar pelanggan, dan mengirimkan penilaian baru yang diberikan pengguna. Provider ini juga memiliki fungsi untuk memastikan bahwa setiap penilaian yang diberikan langsung dikaitkan dan disimpan bersama objek pesanan yang bersangkutan.

```
List<Order> _getCompletedOrdersWithRatings(String merchantEmail) {
    return _orderProvider.orders
        .where((order) =>
            order.status == 'completed' &&
            order.merchantEmail == merchantEmail &&
            order.foodRating != null &&
            order.completedAt != null)
        .toList();
  }
```

Gambar 3.36 Mengambil Pesanan yang Sudah Dinilai

Fungsi Gambar 3.36 ini berperan untuk menyaring pesanan yang sudah berstatus selesai (*completed*) dan telah memiliki rating makanan dari pelanggan. Data yang dihasilkan akan digunakan untuk menghitung rata-rata nilai rating dan mengambil komentar dari pengguna.

```
double getAverageFoodRating(String merchantEmail) {
    final ratedOrders = _getCompletedOrdersWithRatings(merchantEmail);
    if (ratedOrders.isEmpty) return 0.0;

    final totalRating =
        ratedOrders.fold<int>(0, (sum, order) => sum + (order.foodRating ?? 0));
    return totalRating / ratedOrders.length;
  }
```

Gambar 3.37 Mengganti Tema Secara Dinamis

Digunakan Gambar 3.37 berfungsi untuk menghitung rata-rata rating makanan yang diberikan kepada suatu tenant atau merchant. Perhitungan ini bermanfaat sebagai indikator kepuasan pelanggan terhadap kualitas makanan yang dijual.

h) Theme Notifier

ThemeNotifier adalah kelas yang bertanggung jawab untuk mengelola dan menerapkan mode tema terang (light) dan gelap (dark) dalam aplikasi U-Teen. Kelas ini menggunakan SharedPreferences untuk menyimpan preferensi tema pengguna secara lokal, sehingga pilihan tema akan tetap konsisten meskipun aplikasi ditutup dan dibuka kembali. Komponen ini berperan penting dalam meningkatkan pengalaman pengguna (UX) dengan menyesuaikan tampilan antarmuka aplikasi berdasarkan preferensi visual pengguna.

Gambar 3.38 Mengganti Tema

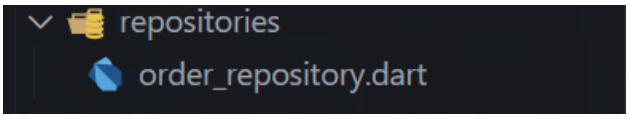
Fungsi Gambar 3.38 ini digunakan untuk mengganti mode tema secara real-time dari light ke dark atau sebaliknya. Selain memperbarui tampilan UI, perubahan juga langsung disimpan ke storage agar persistensinya terjaga

```dart
void _updateTheme() {
    _currentTheme = ThemeData(
      scaffoldBackgroundColor: AppTheme.getBackground(_isDarkMode),
      cardColor: AppTheme.getCard(_isDarkMode),
      primaryColor: AppTheme.getButton(_isDarkMode),
      textTheme: TextTheme(
        bodyLarge: TextStyle(color: AppTheme.getPrimaryText(_isDarkMode)),
        bodyMedium: TextStyle(color: AppTheme.getSecondaryText(_isDarkMode)),
      ),
      iconTheme: IconThemeData(color: AppTheme.getIcon(_isDarkMode)),
      elevatedButtonTheme: ElevatedButtonThemeData(
        style: ElevatedButton.styleFrom(
          backgroundColor: AppTheme.getButton(_isDarkMode),
          foregroundColor: Colors.white,
        ),
      ),
      dividerColor: AppTheme.getBorder(_isDarkMode),
      colorScheme: ColorScheme.fromSwatch().copyWith(
        primary: AppTheme.getButton(_isDarkMode),
        secondary: AppTheme.getRating(_isDarkMode),
        surface: AppTheme.getCard(_isDarkMode),
        background: AppTheme.getBackground(_isDarkMode),
        onPrimary: Colors.white,
        onSecondary: AppTheme.getPrimaryText(_isDarkMode),
        onSurface: AppTheme.getPrimaryText(_isDarkMode),
        onBackground: AppTheme.getPrimaryText(_isDarkMode),
        brightness: _isDarkMode ? Brightness.dark : Brightness.light,
      ),
    );
  }
```

Gambar 3.39 Penerapan Tema Secara Terpusat

Fungsi Gambar 3.39 ini digunakan untuk membangun dan menerapkan konfigurasi ThemeData berdasarkan nilai _isDarkMode. Semua komponen UI seperti warna latar, teks, ikon, tombol, dan lainnya akan menyesuaikan secara otomatis.

8) **Repository**



Gambar 3.40 Isi Folder Repositories

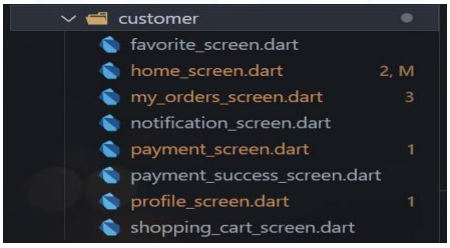
a) Order Repository

```dart
Future<String> createOrder({
    required DateTime pickupTime,
    required List<CartItem> items,
    required String paymentMethod,
  }) async {
    final merchantName = items.isNotEmpty ? items.first.subtitle : 'Unknown Merchant';

    final order = Order(
      id: _generateOrderId(),
      orderTime: DateTime.now(),
      pickupTime: pickupTime,
      createdAt: DateTime.now(),
      readyAt: null,
      completedAt: null,
      cancelledAt: null,
      items: items.map((item) => OrderItem(
        name: item.name,
        imgBase64: item.imgBase64,
        subtitle: item.subtitle,
        price: item.price,
        quantity: item.quantity, sellerEmail: '',
      )).toList(),
      paymentMethod: paymentMethod,
      merchantName: merchantName, customerName: '', merchantEmail: '', // Using first item's subtitle as merchant name
    );

    _orders.insert(0, order);
    return order.id;
  }
```

Gambar 3.41 Code Order Repository

Fungsi utama dari Gambar 3.41 repository ini adalah menyusun dan merekam pesanan baru yang dilakukan oleh pengguna dan memastikan bahwa seluruh proses pembuatan pesanan berjalan secara terstruktur dan tersimpan dalam memori lokal sebelum disinkronkan ke sistem penyimpanan berbasis *cloud*.

9) Layar Customer



Gambar 3.42 Isi Folder Customer Screen

a) Favorite Screen

```
Future<void> _initializeProvider() async {
    final favoriteProvider = Provider.of<FavoriteProvider>(context, listen: false);
    try {
      await favoriteProvider.initialize(context);
      setState(() {
        _isInitialized = true;
      });
    } catch (e) {
      debugPrint('FavoritesScreen: Error initializing FavoriteProvider: $e');
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Failed to load favorites: $e'),
          backgroundColor: AppTheme.getSnackBarError(false),
        ),
      );
    }
  }
```

Gambar 3.43 Pengambilan Data

Fungsi _initializeProvider Gambar 3.43 bertanggung jawab untuk mengambil data favorit (fetch data) dari FavoriteProvider. Fungsi ini dipanggil saat inisialisasi layar untuk memuat daftar item favorit dari penyimpanan lokal atau cloud. Proses asinkronus ini memastikan data tersedia sebelum UI dirender, dengan penanganan kesalahan melalui try-catch untuk menampilkan notifikasi jika gagal memuat data.

```
void _navigateToDetail(BuildContext context, FavoriteItem item) {
    debugPrint('Navigating to detail for ${item.name}, imgBase64 length: ${item.imgBase64.length}');
    Navigator.pop(context);
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => HomeScreen(
          initialFoodItem: item.name,
          initialFoodPrice: item.price.toString(),
          initialFoodImgBase64: item.imgBase64,
          initialFoodSubtitle: item.subtitle ?? '',
          initialSellerEmail: null,
        ),
      ),
    );
  }
```

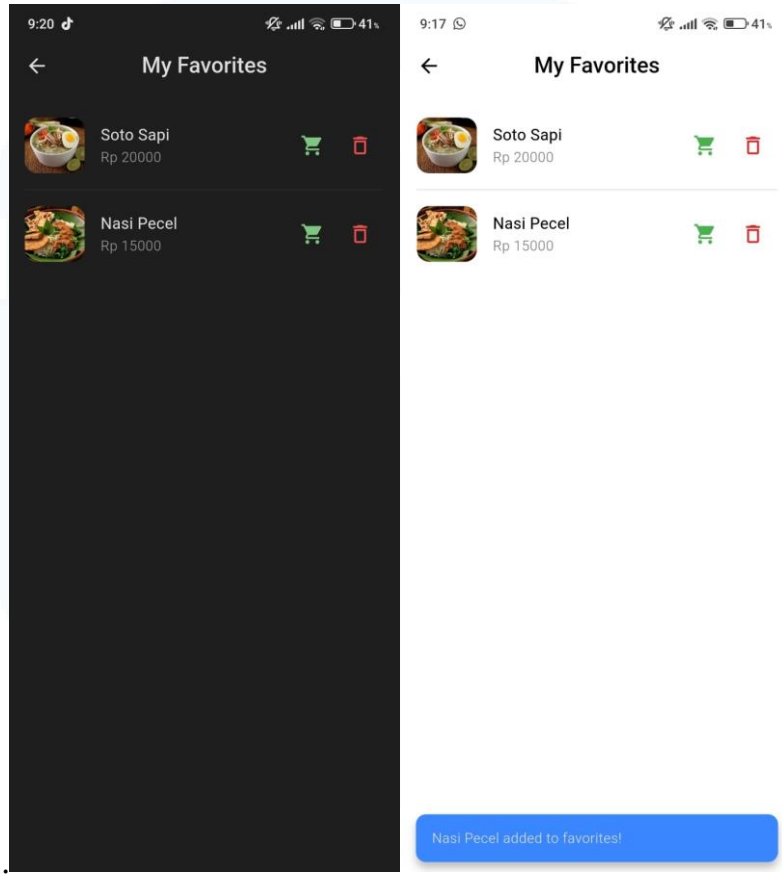
Gambar 3.44 Menambahkan ke Shopping Cart

Fungsi _navigateToDetail pada Gambar 3.44 menangani penambahan item ke keranjang (add to cart) dengan mengarahkan pengguna ke HomeScreen untuk memulai proses pemesanan. Fungsi ini meneruskan data item favorit seperti nama, harga, dan gambar dalam format Base64, memungkinkan pengguna melanjutkan transaksi dari layar favorit dengan alur yang efisien.

```
IconButton(
  icon: Icon(
    Icons.delete_outline,
    color: AppTheme.getSnackBarError(isDarkMode),
  ),
  onPressed: () {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        backgroundColor: AppTheme.getCard(isDarkMode),
        title: Text(
          "Remove from favorites?",
          style: TextStyle(color: AppTheme.getPrimaryText(isDarkMode)),
        ),
        content: Text(
          "Are you sure you want to remove ${item.name}?",
          style: TextStyle(color: AppTheme.getSecondaryText(isDarkMode)),
        ),
```

Gambar 3.45 Menghapus dari Favorit

Kode Gambar 3.45 di atas mengimplementasikan fungsi penghapusan menu dari favorit (remove from favorite) melalui IconButton. Setelah di klik, maka akan muncul *dialog box* yang berisi konfirmasi apakah pengguna ingin menghapus menu yang dimaksud.

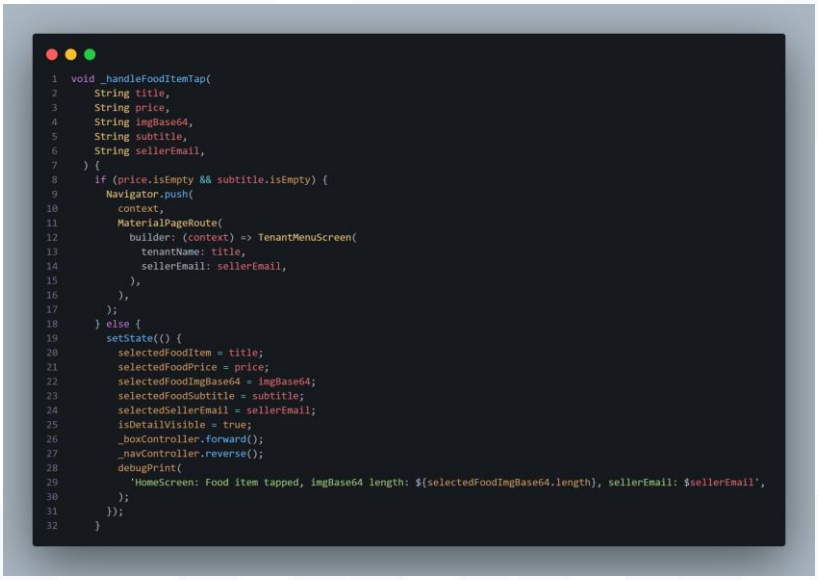


Gambar 3.46 Tampilan Favorite Screen

Gambar 3.46 diatas adalah tampilan dari Favorite Screen pada UI customer. UI ini menampilkan foto menu, nama menu, serta harga satuan di sisi kiri layar, serta button add to shopping cart dan delete from favorite di sisi kanan.

b) Home Screen

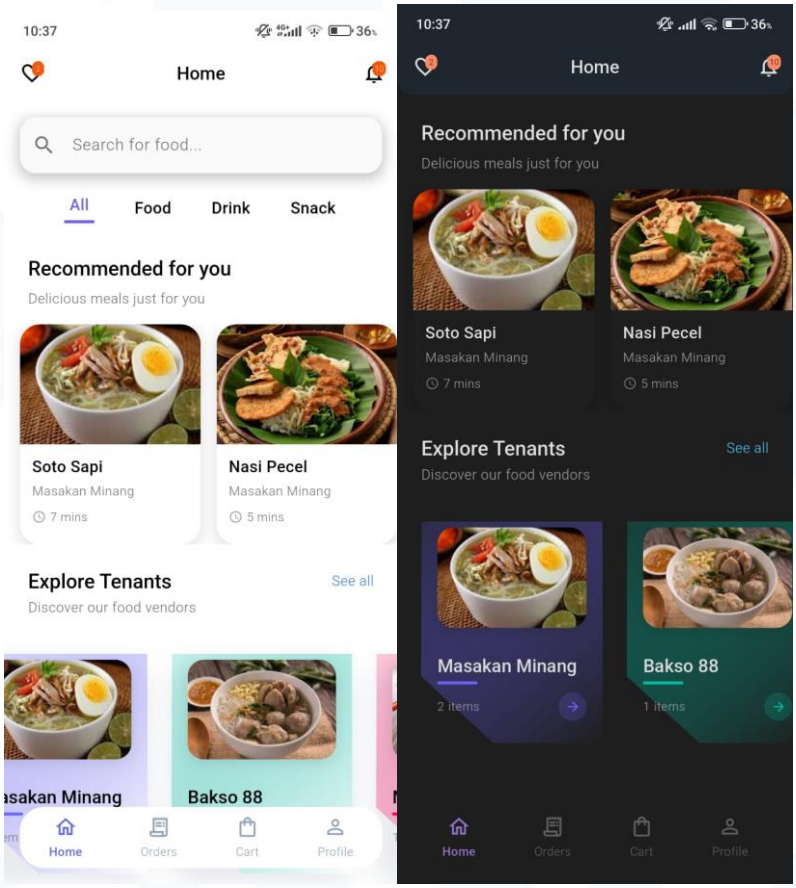
Bagian ini menjelaskan implementasi layar *HomeScreen* pada aplikasi U-Teen, yang merupakan antarmuka utama bagi pengguna untuk menjelajahi menu makanan, memilih kategori, dan melakukan pemesanan. Layar ini dikembangkan menggunakan Flutter dengan *state management* berbasis *Provider* untuk mengelola data otentikasi, favorit, notifikasi, dan pesanan, serta terintegrasi dengan tema visual *AppTheme* untuk mendukung mode terang dan gelap.



```
void _handleFoodItemTap(
    String title,
    String price,
    String imgBase64,
    String subtitle,
    String sellerEmail,
  ) {
    if (price.isEmpty && subtitle.isEmpty) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => TenantMenuScreen(
            tenantName: title,
            sellerEmail: sellerEmail,
          ),
        ),
      );
    } else {
      setState(() {
        selectedFoodItem = title;
        selectedFoodPrice = price;
        selectedFoodImgBase64 = imgBase64;
        selectedFoodSubtitle = subtitle;
        selectedSellerEmail = sellerEmail;
        isDetailVisible = true;
        _boxController.forward();
        _navController.reverse();
        debugPrint(
          'HomeScreen: Food item tapped, imgBase64 length: ${selectedFoodImgBase64.length}, sellerEmail: $sellerEmail',
        );
      });
    }
```

Gambar 3.47 Code Handle Food Item

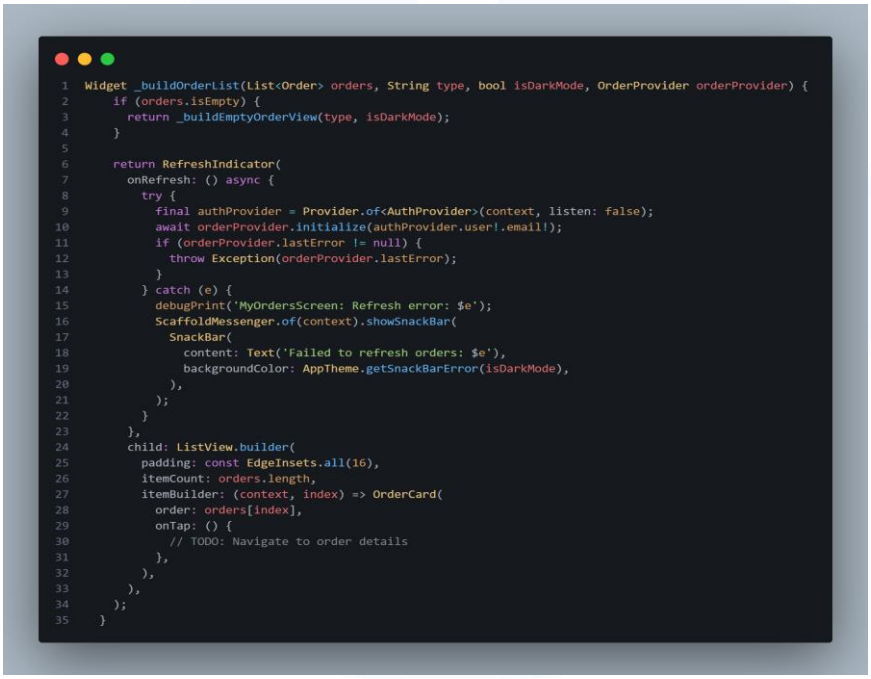
Fungsi handleFoodItemTap pada Gambar 3.47untuk menangani interaksi pengguna saat memilih item makanan untuk ditambahkan ke keranjang (*add to cart*). Fungsi ini memeriksa apakah item yang dipilih adalah tenant (tidak memiliki harga atau subtitle) atau makanan. Jika tenant, pengguna diarahkan ke *TenantMenuScreen*. Jika makanan, fungsi memperbarui state untuk menampilkan *DetailBox* dengan informasi seperti nama, harga, dan gambar menu.



Gambar 3.48 UI Customer Home Screen

Gambar 3.48 adalah tampilan home screen untuk customer. Di bagian atas kiri terdapat navigasi ke favorite screen, diatas kanan juga terdapat navigasi untuk mengarah ke notification screen, dan ditengah adalah fitur untuk melakukan pencarian makanan yang detailnya akan dibahas pada subbab widget nanti. Lalu sebagai bagian utama, layar, terdapat 2 card tampilan, yaitu daftar menu, serta daftar tenant.

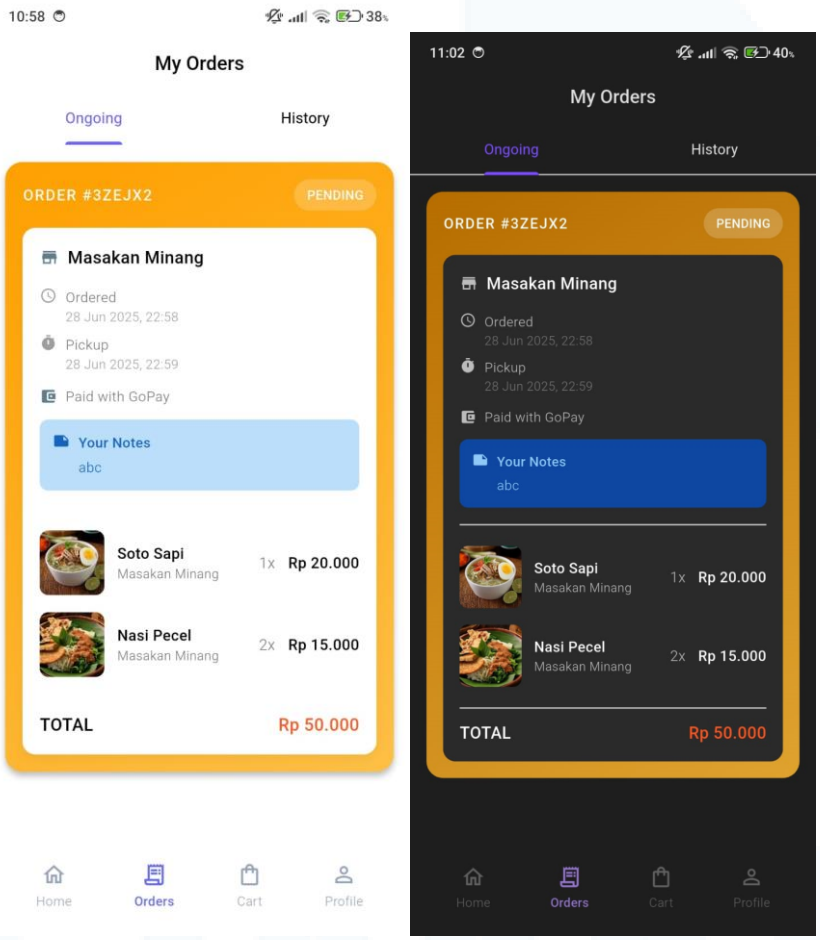
c) My Orders Screen



```dart
Widget _buildOrderList(List<Order> orders, String type, bool isDarkMode, OrderProvider orderProvider) {
    if (orders.isEmpty) {
      return _buildEmptyOrderView(type, isDarkMode);
    }

    return RefreshIndicator(
      onRefresh: () async {
        try {
          final authProvider = Provider.of<AuthProvider>(context, listen: false);
          await orderProvider.initialize(authProvider.user!.email!);
          if (orderProvider.lastError != null) {
            throw Exception(orderProvider.lastError);
          }
        } catch (e) {
          debugPrint('MyOrdersScreen: Refresh error: $e');
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(
              content: Text('Failed to refresh orders: $e'),
              backgroundColor: AppTheme.getSnackBarError(isDarkMode),
            ),
          );
        }
      },
      child: ListView.builder(
        padding: const EdgeInsets.all(16),
        itemCount: orders.length,
        itemBuilder: (context, index) => OrderCard(
          order: orders[index],
          onTap: () {
            // TODO: Navigate to order details
          },
        ),
      ),
    );
  }
```

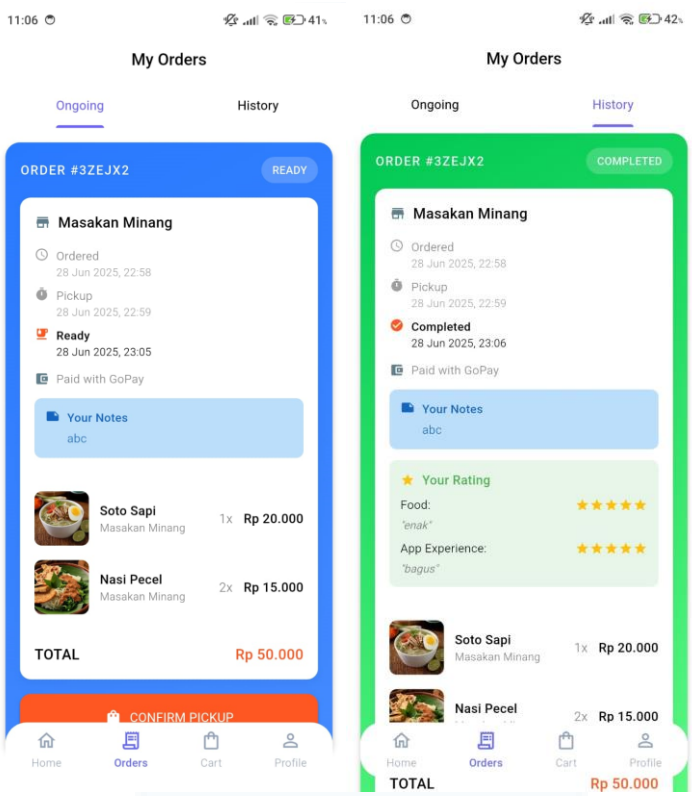
Gambar 3.49 Code My Orders Screen

Gambar 3.49 pada My Orders Screen menjelaskan tentang implementasi layar *MyOrdersScreen* pada aplikasi U-Teen, yang memungkinkan pengguna untuk melihat daftar pesanan mereka, baik yang sedang berlangsung (*ongoing*) maupun riwayat pesanan (*history*). Fungsi _buildOrderList diatasmengelola tampilan daftar pesanan (ongoing atau history) dengan menggunakan ListView.builder untuk merender OrderCard berdasarkan data dari OrderProvider. Fungsi ini juga mendukung fitur refresh melalui RefreshIndicator, yang memanggil kembali orderProvider.initialize untuk memperbarui data pesanan (fetch data). Jika daftar pesanan kosong, fungsi _buildEmptyOrderView dipanggil untuk menampilkan animasi dan pesan yang sesuai.



Gambar 3.50 UI My Orders Screen Status Ongoing

Ini adalah Gambar 3.50 tampilan untuk my orders screen apabila menu yang dipesan masih berstatus pending atau masih dalam pemrosesan oleh tenant. UI ini menampilkan berbagai macam informasi dimulai dari kapan makanan ini dipesan, waktu pengambilan, metode pembayaran, catatan tambahan, hingga detail pesanan.



Gambar 3.51 UI My Orders Screen Status Ready dan Completed

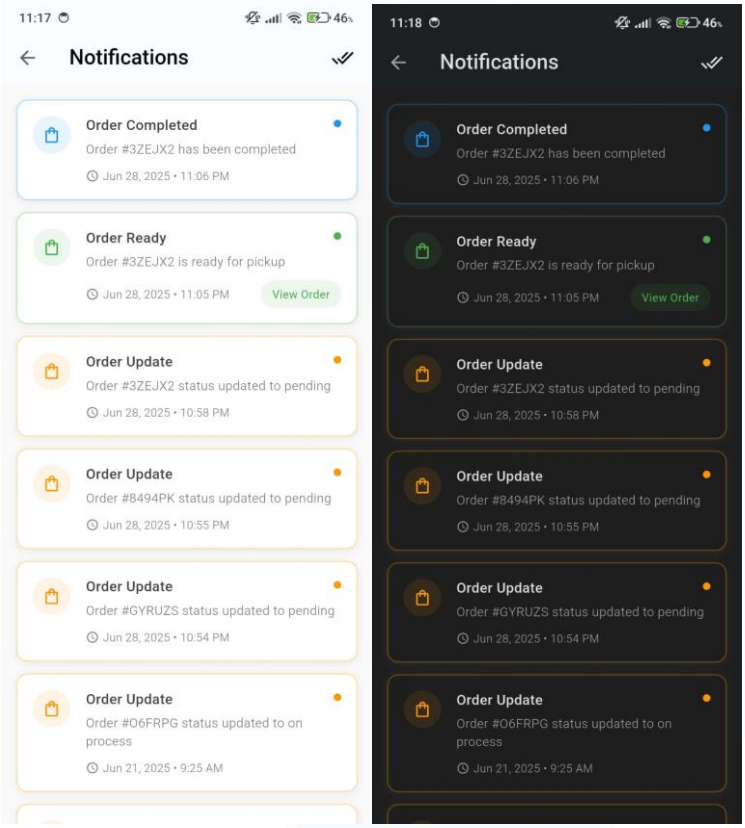
Gambar 3.51 ini adalah tampilan untuk my orders screen apabila menu yang dipesan masih berstatus *ready* dan siap diambil (kiri) dan sudah *completed* (kanan). Customer juga bisa memberikan rating terhadap tenant serta aplikasi untuk memastikan kenyamanan dari sisi kualitas makanan maupun performa aplikasi.

d) Notification Screen

```dart
Widget _buildNotificationCard(NotificationModel notification, BuildContext context, bool isDarkMode) {
    final statusColor = notification.payload?['statusColor'] != null
        ? Color(int.parse(notification.payload!['statusColor'], radix: 16))
        : AppTheme.getButton(isDarkMode);

    final isUnread = !notification.isRead;

    return InkWell(
      borderRadius: BorderRadius.circular(12),
      onTap: () {
        Provider.of<NotificationProvider>(context, listen: false)
            .markAsRead(notification.id);
        if (notification.payload != null && notification.payload!['orderId'] != null) {
          final orderId = notification.payload!['orderId'];
          debugPrint('Navigating to order details for order ID: $orderId');
          // Tambahkan navigasi ke detail order jika diperlukan
        }
      },
```

Gambar 3.52 Code Notification Screen

Fungsi _buildNotificationCard pada Gambar 3.52 ini mengelola tampilan setiap notifikasi dalam bentuk kartu interaktif (*InkWell*). Kartu ini menampilkan informasi seperti judul, pesan, dan waktu notifikasi, dengan ikon dan warna status yang disesuaikan berdasarkan jenis notifikasi (*order*, *promo*, atau *system*). Ketika kartu diketuk, fungsi markAsRead dipanggil untuk menandai notifikasi sebagai telah dibaca (*mark as read*).



Gambar 3.53 UI Notification Screen

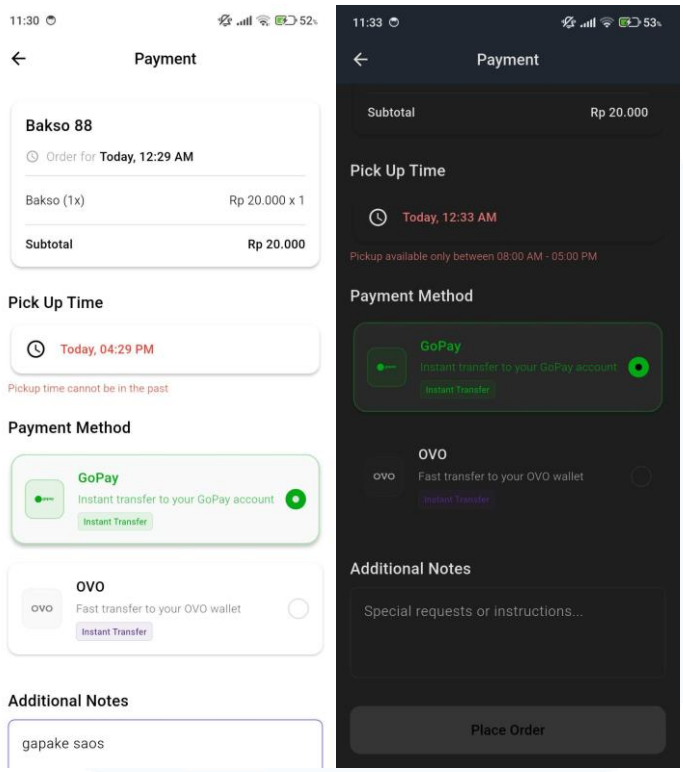
Ini adalah tampilan dari Gambar 3.53 *notification screen customer* yang dapat diakses melalui navigasi di home screen. Kartu ini menunjukkan judul dengan informasi perubahan status pesanan, isi teks dengan isi id pesanan, serta waktu notifikasi ini muncul. Apabila pengguna mengklik kartu notifikasi atau *icon* centang diatas kanan layar, maka notifikasi ini akan berubah status nya menjadi *read*.

e) Payment Screen

```dart
final newOrder = Order(
        id: _generateOrderId(context),
        orderTime: DateTime.now(),
        createdAt: DateTime.now(),
        pickupTime: pickupTime,
        readyAt: null,
        completedAt: null,
        cancelledAt: null,
        items: widget.items
            .map(
              (item) => OrderItem(
                name: item.name,
                subtitle: item.subtitle,
                price: item.price,
                quantity: item.quantity,
                sellerEmail: item.sellerEmail,
                imgBase64: item.imgBase64,
              ),
            )
            .toList(),
        paymentMethod:
            paymentMethods.firstWhere((m) => m.id == selectedPaymentMethod).name,
        merchantName: merchantName,
        merchantEmail: sellerEmail,
        customerName: customerName,
        notes: notes.isEmpty ? null : notes,
      );
```

Gambar 3.54 Code Payment Screen

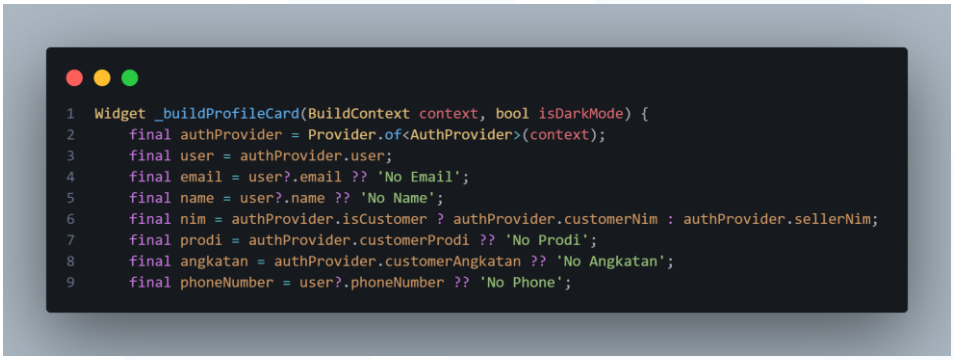
Fungsi Gambar 3.54 ini menangani logika pemrosesan pembayaran dengan memvalidasi input formulir, memastikan metode pembayaran telah dipilih, dan memeriksa status login pengguna. Jika validasi berhasil, fungsi membuat objek *Order* baru dengan detail seperti ID pesanan, waktu pemesanan, waktu pengambilan, item pesanan, metode pembayaran, dan catatan. Pesanan ditambahkan ke *OrderProvider* melalui addOrder, dan item di keranjang dihapus. Setelah pesanan berhasil, pengguna diarahkan ke *PaymentSuccessScreen* dengan notifikasi *SnackBar* untuk konfirmasi.



Gambar 3.55 UI Payment Screen

Ini adalah Gambar 3.55 yang menampilkan tampilan layar customer untuk melakukan pembayaran. Di paling atas, terdapat struk pembayaran terkait detail menu yang dipesan. Lalu dibawahnya terdapat *widget* waktu pengambilan makanan, yang dimana sistem waktu ini tidak akan valid apabila makanan diset untuk diambil pada masa lalu, atau waktu diluar jam operasional kampus pukul 08.00 - 17.00. Dibawahnya terdapat metode pembayaran dan juga notes untuk melakukan kustomisasi pesanan yang ditujukan kepada *tenant*.

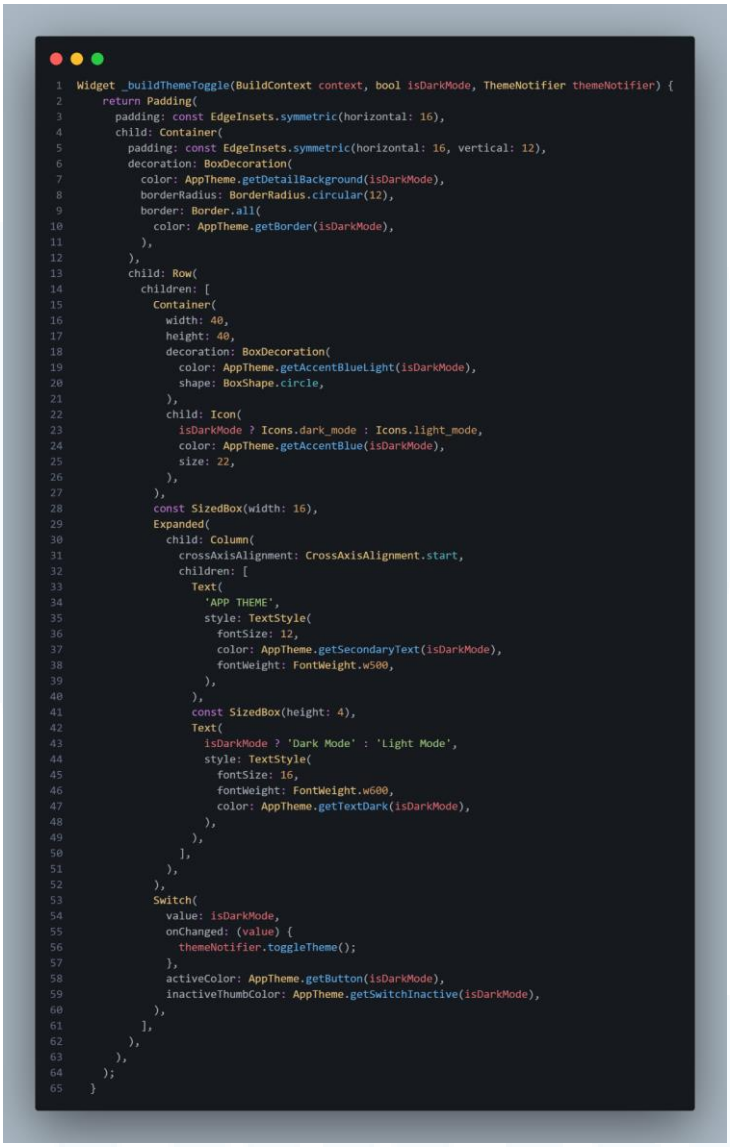
f) Profile Screen



```dart
Widget _buildProfileCard(BuildContext context, bool isDarkMode) {
    final authProvider = Provider.of<AuthProvider>(context);
    final user = authProvider.user;
    final email = user?.email ?? 'No Email';
    final name = user?.name ?? 'No Name';
    final nim = authProvider.isCustomer ? authProvider.customerNim : authProvider.sellerNim;
    final prodi = authProvider.customerProdi ?? 'No Prodi';
    final angkatan = authProvider.customerAngkatan ?? 'No Angkatan';
    final phoneNumber = user?.phoneNumber ?? 'No Phone';
```

Gambar 3.56 Code Profile Screen Customer

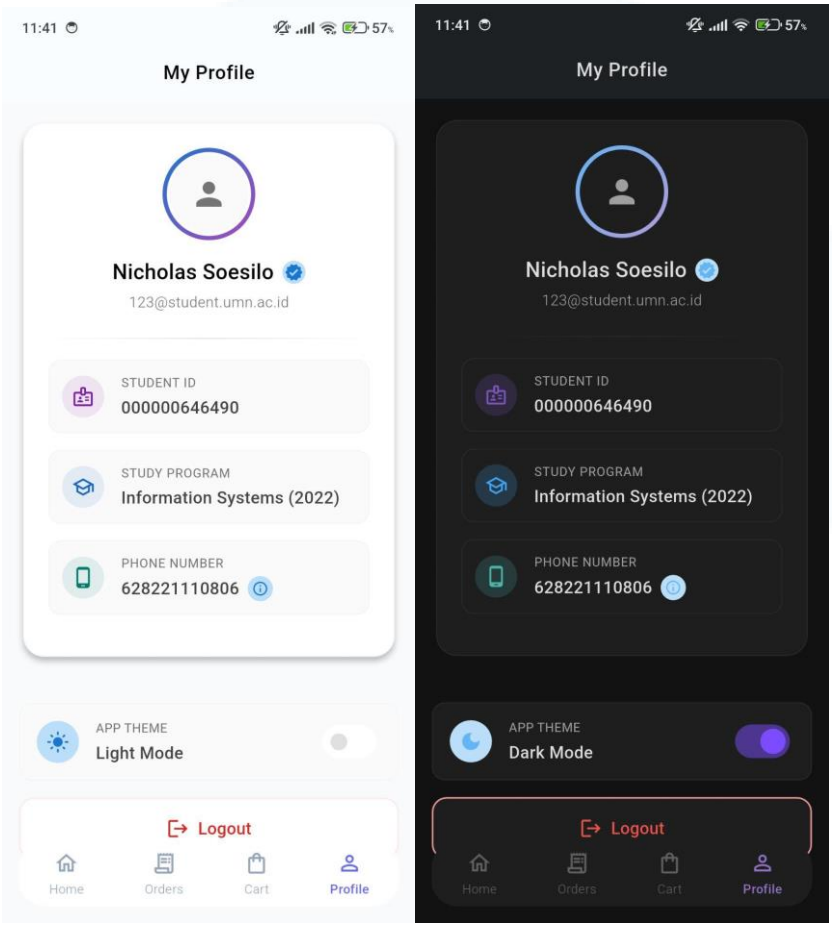
Fungsi _buildProfileCard Gambar 3.56 bertanggung jawab untuk mengambil data profil dari *AuthProvider* dan menampilkannya dalam kartu profil. Data seperti nama, email, NIM (atau ID penjual), program studi, angkatan, dan nomor telepon diambil dari objek user di AuthProvider.



```
Widget _buildThemeToggle(BuildContext context, bool isDarkMode, ThemeNotifier themeNotifier) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 16),
      child: Container(
        padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
        decoration: BoxDecoration(
          color: AppTheme.getDetailBackground(isDarkMode),
          borderRadius: BorderRadius.circular(12),
          border: Border.all(
            color: AppTheme.getBorder(isDarkMode),
          ),
        ),
        child: Row(
          children: [
            Container(
              width: 40,
              height: 40,
              decoration: BoxDecoration(
                color: AppTheme.getAccentBlueLight(isDarkMode),
                shape: BoxShape.circle,
              ),
              child: Icon(
                isDarkMode ? Icons.dark_mode : Icons.light_mode,
                color: AppTheme.getAccentBlue(isDarkMode),
                size: 22,
              ),
            ),
            const SizedBox(width: 16),
            Expanded(
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'APP THEME',
                    style: TextStyle(
                      fontSize: 12,
                      color: AppTheme.getSecondaryText(isDarkMode),
                      fontWeight: FontWeight.w500,
                    ),
                  ),
                  const SizedBox(height: 4),
                  Text(
                    isDarkMode ? 'Dark Mode' : 'Light Mode',
                    style: TextStyle(
                      fontSize: 16,
                      fontWeight: FontWeight.w600,
                      color: AppTheme.getTextDark(isDarkMode),
                    ),
                  ),
                ],
              ),
            ),
            Switch(
              value: isDarkMode,
              onChanged: (value) {
                themeNotifier.toggleTheme();
              },
              activeColor: AppTheme.getButton(isDarkMode),
              inactiveThumbColor: AppTheme.getSwitchInactive(isDarkMode),
            ),
          ],
        ),
      ),
    );
  }
```

Gambar 3.57 Code Pengaturan Light Mode dan Dark Mode

Fungsi _buildProfileCard pada Gambar 3.57 bertanggung jawab untuk mengambil data profil dari *AuthProvider* dan menampilkannya dalam kartu profil. Data seperti nama, email, NIM (atau ID penjual), program studi, angkatan, dan nomor telepon diambil dari objek user di AuthProvider.



Gambar 3.58 UI Profile Screen Customer

Fungsi _buildThemeToggle pada Gambar 3.58 mengelola penggantian tema antara mode terang dan gelap. Widget ini menggunakan switch untuk memungkinkan pengguna mengubah status tema melalui fungsi toggleTheme dari ThemeNotifier. Status tema saat ini (isDarkMode) digunakan untuk menyesuaikan ikon, teks, dan warna sesuai dengan tema aplikasi.

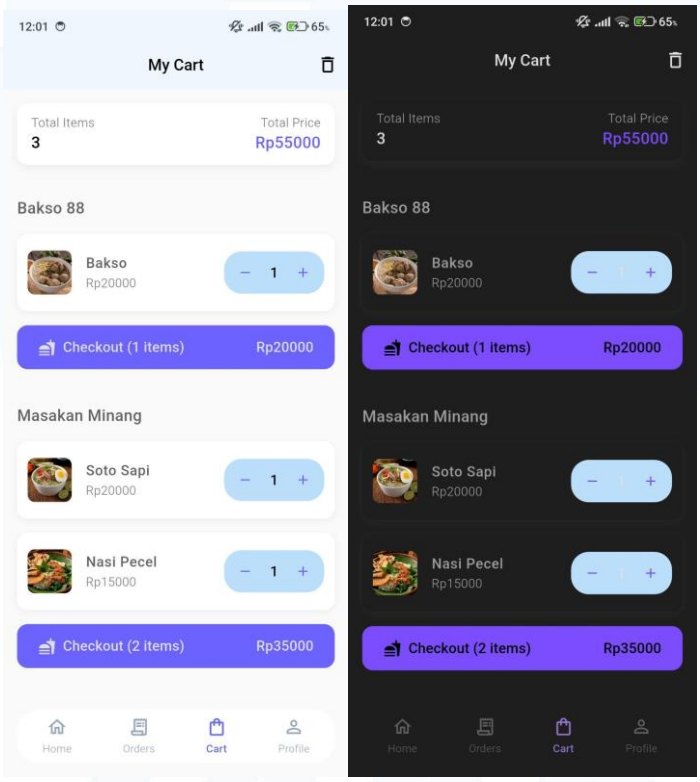
g) Shopping Cart Screen

```dart
Widget _buildCartItemCard(CartItem item, bool isDarkMode) {
    return Container(
      margin: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
      decoration: BoxDecoration(
        color: AppTheme.getCard(isDarkMode),
        borderRadius: BorderRadius.circular(12),
        boxShadow: [
          BoxShadow(
            color: AppTheme.getPrimaryText(isDarkMode).withOpacity(0.05),
            blurRadius: 6,
            offset: const Offset(0, 2),
          ),
        ],
      ),
      child: Dismissible(
        key: Key('${item.name}_${item.subtitle}'),
        direction: DismissDirection.endToStart,
        background: Container(
          decoration: BoxDecoration(
            color: AppTheme.getAccentRedLight(isDarkMode),
            borderRadius: BorderRadius.circular(12),
          ),
          alignment: Alignment.centerRight,
          padding: const EdgeInsets.only(right: 20),
          child: Icon(Icons.delete, color: AppTheme.getSnackBarError(isDarkMode)),
        ),
        confirmDismiss: (direction) => _confirmDismiss(item),
        onDismissed: (direction) => _handleDismissed(item),
        child: ListTile(
          contentPadding: const EdgeInsets.symmetric(
            horizontal: 12,
            vertical: 8,
          ),
          leading: Hero(
            tag: 'cart_${item.name}_${item.subtitle}',
            child: ClipRRect(
              borderRadius: BorderRadius.circular(8),
              child: item.imgBase64.isNotEmpty
                  ? Image.memory(
                      _decodeBase64(item.imgBase64),
                      width: 50,
                      height: 50,
                      fit: BoxFit.cover,
                      errorBuilder: (_, __, ___) {
                        debugPrint('Error loading Base64 image for ${item.name}');
                        return Container(
                          width: 50,
                          height: 50,
                          color: AppTheme.getDivider(isDarkMode),
                          child: Icon(
                            Icons.fastfood,
                            size: 30,
                            color: AppTheme.getPrimaryText(isDarkMode),
                          ),
                        );
                      },
                    )
                  : Container(
                      width: 50,
                      height: 50,
                      color: AppTheme.getDivider(isDarkMode),
                      child: Icon(
                        Icons.fastfood,
                        size: 30,
                        color: AppTheme.getPrimaryText(isDarkMode),
                      ),
                    ),
            ),
          ),
          title: Text(
            item.name,
            style: TextStyle(
              fontWeight: FontWeight.w600,
              color: AppTheme.getTextMedium(isDarkMode),
            ),
          ),
          subtitle: Text(
            'Rp${item.price}',
            style: TextStyle(color: AppTheme.getSecondaryText(isDarkMode)),
          ),
          trailing: _buildQuantityControls(item, isDarkMode),
        ),
      ),
    );
  }
```

Gambar 3.59 Code Item Card Shopping Cart Screen

Fungsi _buildCartItemCard pada Gambar 3.59 mengelola tampilan setiap item di keranjang menggunakan widget Dismissible untuk mendukung penghapusan item (remove from cart) melalui gestur swipe. Saat pengguna menggeser item, fungsi _confirmDismiss menampilkan dialog konfirmasi, dan jika dikonfirmasi, _handleDismissed memanggil removeFromCart dari CartProvider

untuk menghapus item dan menampilkan SnackBar sebagai umpan balik. Widget ini juga menampilkan gambar item (dari Base64 atau ikon default), nama, harga, dan kontrol kuantitas melalui _buildQuantityControls, yang memungkinkan pengguna menambah atau mengurangi jumlah item.



Gambar 3.60 UI Shopping Cart Screen

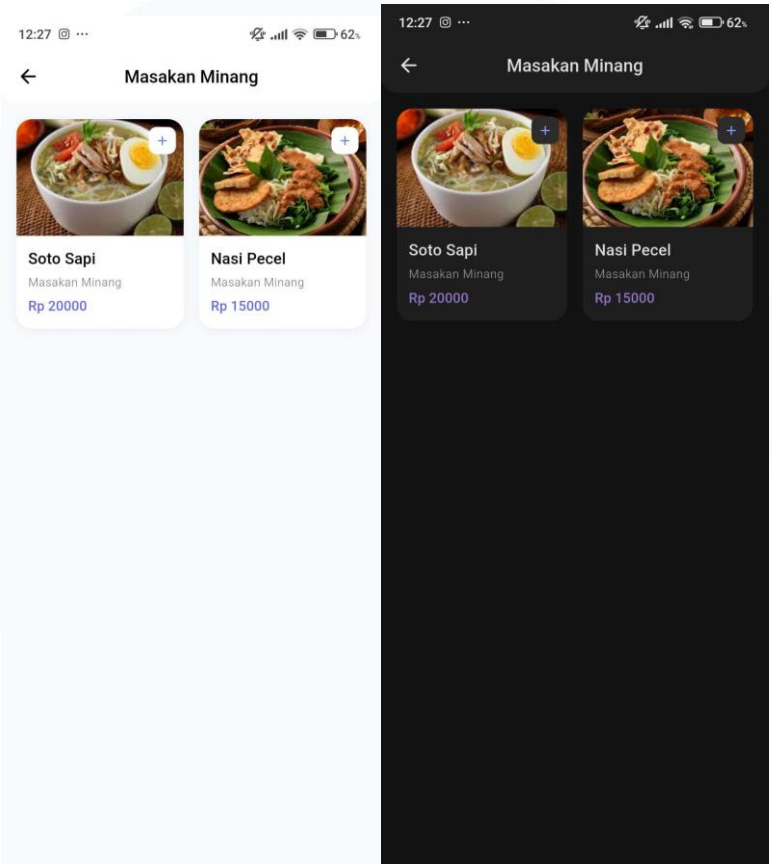
Gambar 3.60 ini adalah tampilan dari *shopping cart* screen milik *customer*. Di paling atas terdapat total jumlah item yang dipesan serta harga keseluruhan. Sementara di bagian tengah terdapat daftar menu yang sudah dimasukkan kedalam *shopping cart*. Namun untuk sistem *chekout* tetap terpisah per *tenant*, jadi tidak digabung.

h) Tenant Menu Screen

```dart
class MenuItemCard extends StatefulWidget {
  final Product product;
  final VoidCallback onTap;

  const MenuItemCard({
    Key? key,
    required this.product,
    required this.onTap,
  }) : super(key: key);

  @override
  _MenuItemCardState createState() => _MenuItemCardState();
}

class _MenuItemCardState extends State<MenuItemCard> with TickerProviderStateMixin {
  late AnimationController _hoverController;
  late Animation<double> _scaleAnimation;
  late Animation<double> _elevationAnimation;
  bool _isHovered = false;
```

Gambar 3.61 Code Tenant Menu Screen

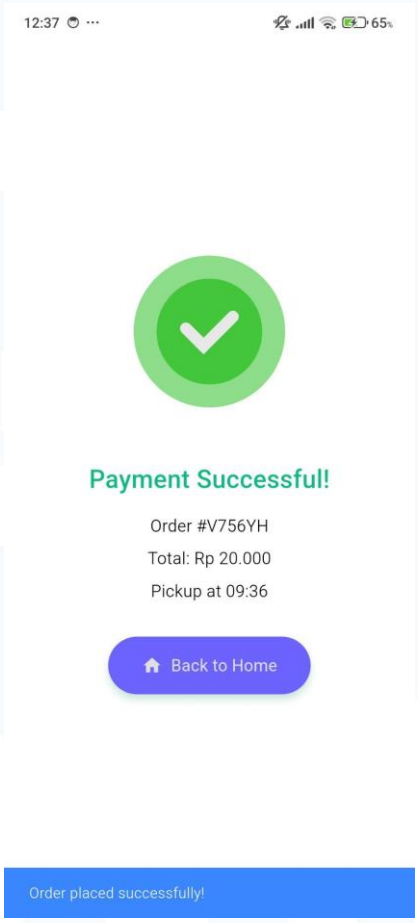
Widget MenuItemCard pada Gambar 3.61 ini menangani tampilan dan interaksi setiap item menu (handle user interaction) dalam grid. Widget ini menggunakan animasi scale dan elevation melalui AnimationController untuk memberikan efek visual saat pengguna menyentuh item (onTapDown, onTapUp, onTapCancel). Gambar produk ditampilkan dari data Base64 dengan fallback ke placeholder jika gagal memuat. Informasi seperti nama, subtitle, dan harga dirender dengan gaya sesuai tema aplikasi. Saat item disentuh, fungsi onTap memanggil _handleFoodItemTap untuk menampilkan DetailBox (show item details), memberikan pengalaman interaktif yang menarik dan responsif. Namun code secara keseluruhan tidak dapat ditampilkan karena terlalu panjang.



Gambar 3.62 UI Tenant Menu Screen

Ini adalah Gambar 3.62 yang berisi tampilan tenant menu screen yang dapat diakses melalui navigasi dari home screen customer. Tampilan ini menampilkan food card dengan gambar, nama menu, nama *tenant*, serta harga satuan. Apabila di klik, maka pengguna akan dinavigasi ke detailbox untuk menambahkan menu ke *shopping cart*.

i) Payment Success Screen

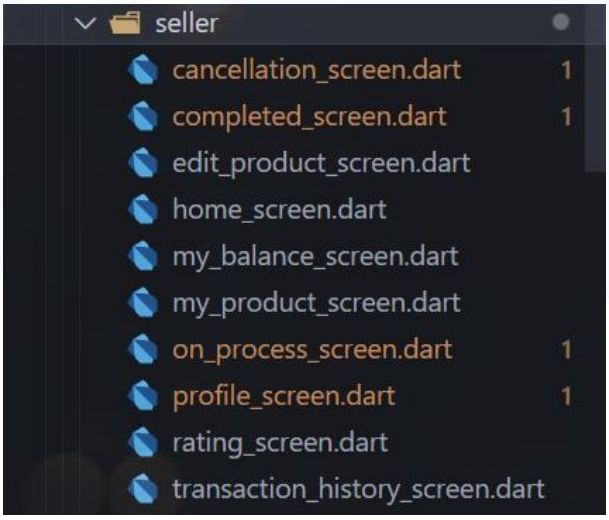


Gambar 3.63 UI Payment Success Screen

Ini adalah tampilan pada Gambar 3.63 payment success screen apabila customer sudah selesai melakukan pembayaran melalui payment screen. Pada UI diatas, menampilkan teks bahwa pesanan sudah berhasil dipesan, bersama dengan order id, total nilai pesanan, serta waktu pengambilan makanan.

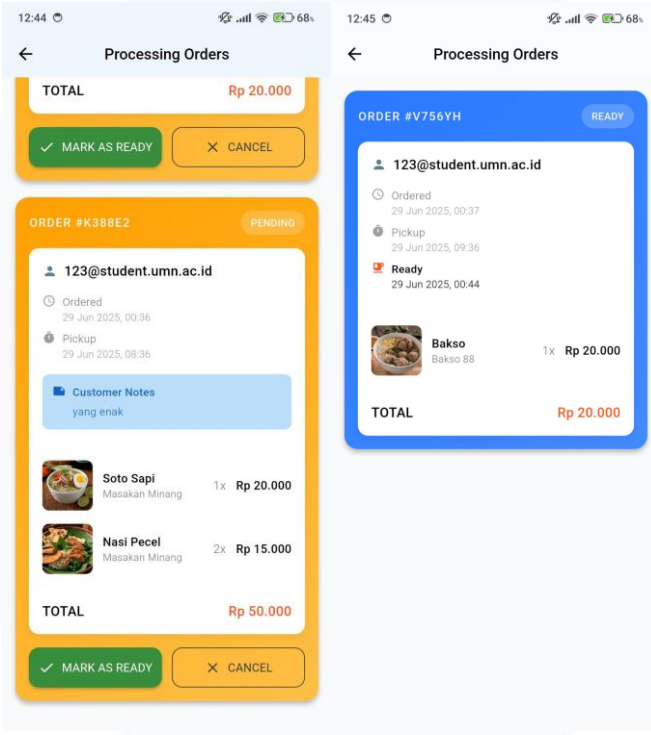
10) Tenant Screen

Seller Screen pada Gambar 3.64 ini berisi kumpulan file yang digunakan untuk menampung berbagai macam file yang dapat diakses oleh tenant. Dalam folder tenant screen ini terdiri dari total 10 file berbeda antara lain cancellation, completed, edit product, home, my balance, my product, on proess, profile, rating, serta transaction history.



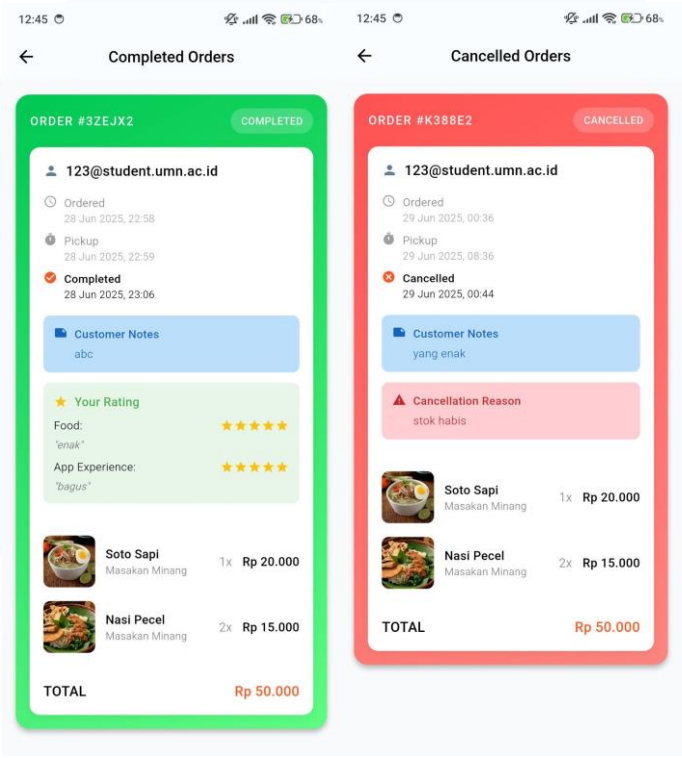
Gambar 3.64 Isi Folder Tenant Screen

a) Cancellation, Completed, and On Process Screen



Gambar 3.65 Processing Order Screen

Gambar 3.65 ini adalah tampilan pada on process screen, dengan makanan berstatus pending (kiri) setelah orderan masuk dari customer, dan status akan berubah menjadi ready (kanan) setelah tenant melakukan mark as ready sebagai penanda bahwa makanan sudah siap diambil oleh customer. Kartu ini menampilkan berbagai macam informasi seperti email (seharusnya nama, tapi ada bug), lalu waktu pemesanan, pengambilan, catatan dari kustomer apabila ada, serta detail menu pesanan.



Gambar 3.66 Completed dan Cancelled Screen

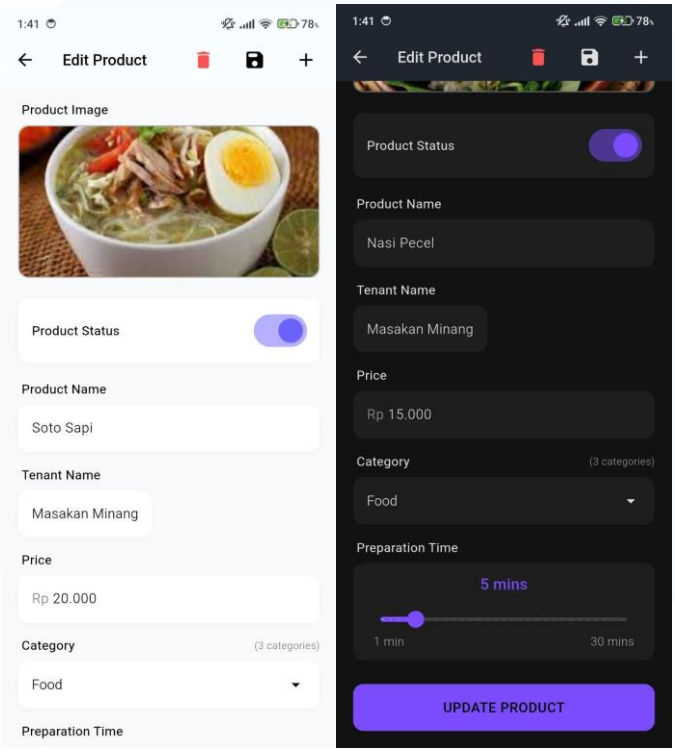
Gambar 3.66 ini adalah tampilan pada completed screen (kiri) dan cancelled screen (kanan). UI tidak mengalami banyak perubahan dengan on process screen, hanya saja pada completed screen ini terdapat tampilan rating yang diberi oleh customer, dan untuk cancellation screen terdapat alasan kenapa orderan makanan ini di cancel oleh tenant, sehingga customer juga bisa melihat dan tau alasan kenapa orderan nya di cancel.

b) Edit Product Screen

```
Future<void> _saveProduct() async {
    if (!_formKey.currentState!.validate()) return;
    setState(() => _isUploading = true);
    debugPrint('Saving product');
    debugPrint('Selected category: $_selectedCategory');

    final themeNotifier = Provider.of<ThemeNotifier>(context, listen: false);
    final isDarkMode = themeNotifier.isDarkMode;

    try {
      final foodProvider = Provider.of<FoodProvider>(context, listen: false);
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      final sellerEmail = authProvider.sellerEmail ?? '';
```

Gambar 3.67 Code Edit Product Screen

Fungsi _saveProduct pada Gambar 3.67 ini menangani penyimpanan atau pembaruan produk (save/update product). Fungsi ini memvalidasi input formulir, memastikan email penjual tersedia, dan mengelola gambar produk (Base64) dengan batas ukuran 1 MB untuk Firestore. Objek Product dibuat dengan data seperti nama, harga, waktu persiapan, nama tenant, email penjual, status aktif, dan kategori. Jika produk baru, addProduct dipanggil; jika produk ada, updateProduct dipanggil melalui FoodProvider.



Gambar 3.68 UI Edit Product Screen

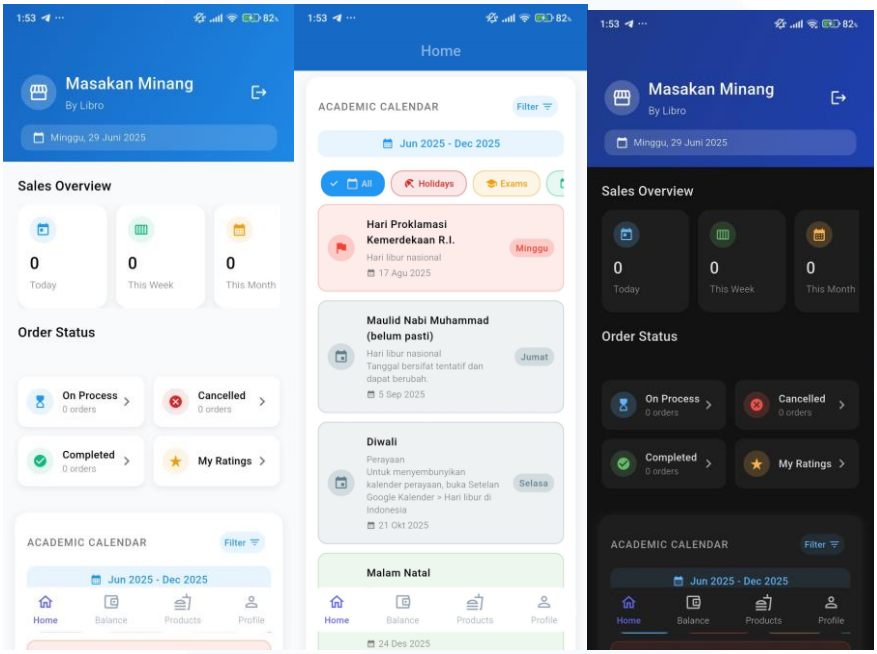
Gambar 3.68 ini adalah tampilan edit product screen untuk tenant kantin. Dari paling atas terdapat icon untuk menghapus perubahan, menyimpan perubahan, atau menambahkan produk baru. Lalu bagian utama dari tampilan ini berisi foto menu, status menu (aktif atau nonaktif), nama menu, nama tenant, harga, kategori (makanan, minuman, snack), serta berapa lama waktu untuk menyiapkan produk.

c) Home Screen

```dart
int _getSalesCount(BuildContext context, String period) {
    final orderProvider = Provider.of<OrderProvider>(context);
    final sellerEmail = Provider.of<AuthProvider>(context).user?.email ?? '';
    final now = DateTime.now();
    final startOfToday = DateTime(now.year, now.month, now.day);
    final startOfWeek = DateTime(now.year, now.month, now.day - (now.weekday - 1));
    final startOfMonth = DateTime(now.year, now.month, 1);

    switch (period) {
      case 'today':
        return orderProvider.getCompletedOrdersForMerchant(sellerEmail)
            .where((order) => order.completedTime != null && order.completedTime!.isAfter(startOfToday))
            .length;
      case 'week':
        return orderProvider.getCompletedOrdersForMerchant(sellerEmail)
            .where((order) => order.completedTime != null && order.completedTime!.isAfter(startOfWeek))
            .length;
      case 'month':
        return orderProvider.getCompletedOrdersForMerchant(sellerEmail)
            .where((order) => order.completedTime != null && order.completedTime!.isAfter(startOfMonth))
            .length;
      default:
        return 0;
    }
  }
```

Gambar 3.69 Code Pengambilan Data Penjualan Home Screen

Fungsi _getSalesCount pada Gambar 3.69 ini bertanggung jawab untuk mengambil data penjualan dari OrderProvider berdasarkan email penjual yang diperoleh dari AuthProvider. Fungsi ini menghitung jumlah pesanan yang selesai untuk periode tertentu (hari ini, minggu ini, atau bulan ini) dengan memfilter pesanan berdasarkan waktu penyelesaian (completedTime). Data ini digunakan untuk menampilkan metrik penjualan pada kartu metrik (_buildMetricCard), memberikan gambaran cepat tentang performa penjualan kepada penjual.



Gambar 3.70 UI Home Screen Tenant

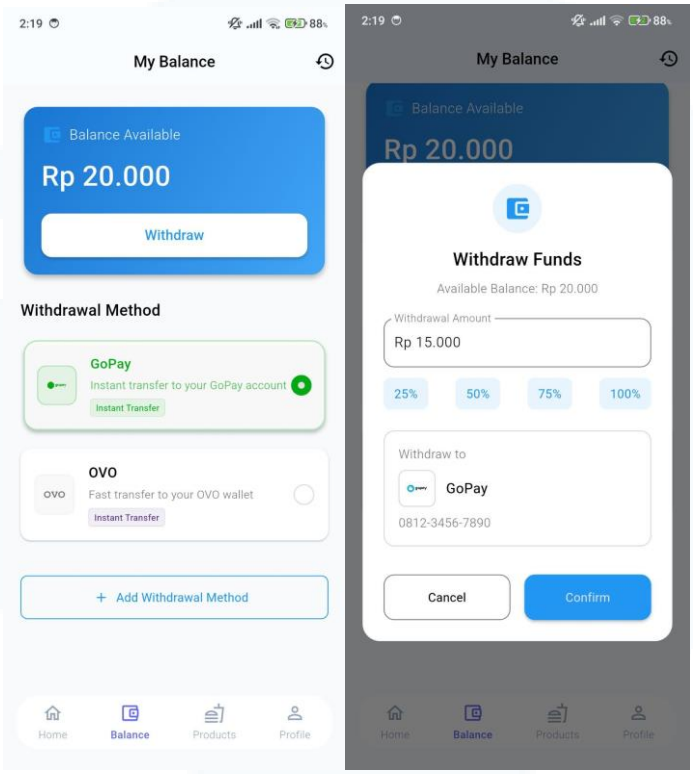
Ini adalah Gambar 3.70 yang berisi tampilan dari home screen milik tenant kantin. Diatas berisi nama tenant, *watermark* Libro sebagai pengelola kantin UMN, serta kalender yang menunjukkan tanggal di hari tersebut. Dibawah header, terdapat Sales Overview yang berisi data berapa penjualan yang berhasil diselesaikan oleh tenant per hari. minggu, dan bulan. Dibawahnya lagi terdapat button Order Status untuk mengecek pesanan yang masih dalam pemrosesan, dibatalkan, terselesaikan, serta rating tenant secara keseluruhan. Selain itu, pada posisi paling bawah terdapat kalender akademik (untuk sementara menggunakan kalender google) untuk membantu tenant dalam mengidentifikasi tanggal merah dan kegiatan mahasiswa, misalnya masa libur, UTS, dan UAS.

d) My Balance Screen

```dart
void _withdrawFunds(BuildContext context) {
    if (_balance <= 0) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: const Text('Balance is insufficient for withdrawal'),
          behavior: SnackBarBehavior.floating,
          backgroundColor: AppTheme.getSnackBarError(false),
        ),
      );
      return;
    }

    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    final orderProvider = Provider.of<OrderProvider>(context, listen: false);
    final sellerEmail = authProvider.user?.email ?? '';

    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => WithdrawalDialog(
        balance: _balance,
        selectedMethod: _selectedMethod!,
        userGopayNumber: _userGopayNumber,
        onWithdraw: (amount) async {
          await orderProvider.addWithdrawal(
            merchantEmail: sellerEmail,
            amount: amount,
            method: _selectedMethod!.name,
          );
          _showWithdrawalSuccess(amount);
          _calculateBalance();
        },
      ),
    );
  }
```

Gambar 3.71 Code My Balance Screen

Fungsi _withdrawFunds pada Gambar 3.71 berfungsi untuk menangani proses penarikan dana. Fungsi ini memeriksa apakah saldo cukup. Jika saldo cukup, fungsi menampilkan WithdrawalDialog dengan parameter saldo, metode penarikan terpilih (_selectedMethod), dan nomor GoPay/OVO pengguna. Callback onWithdraw memanggil addWithdrawal dari OrderProvider untuk mencatat transaksi penarikan di Firestore, lalu memanggil _showWithdrawalSuccess untuk umpan balik sukses dan _calculateBalance untuk memperbarui saldo di UI.



Gambar 3.72 UI My Balance Screen

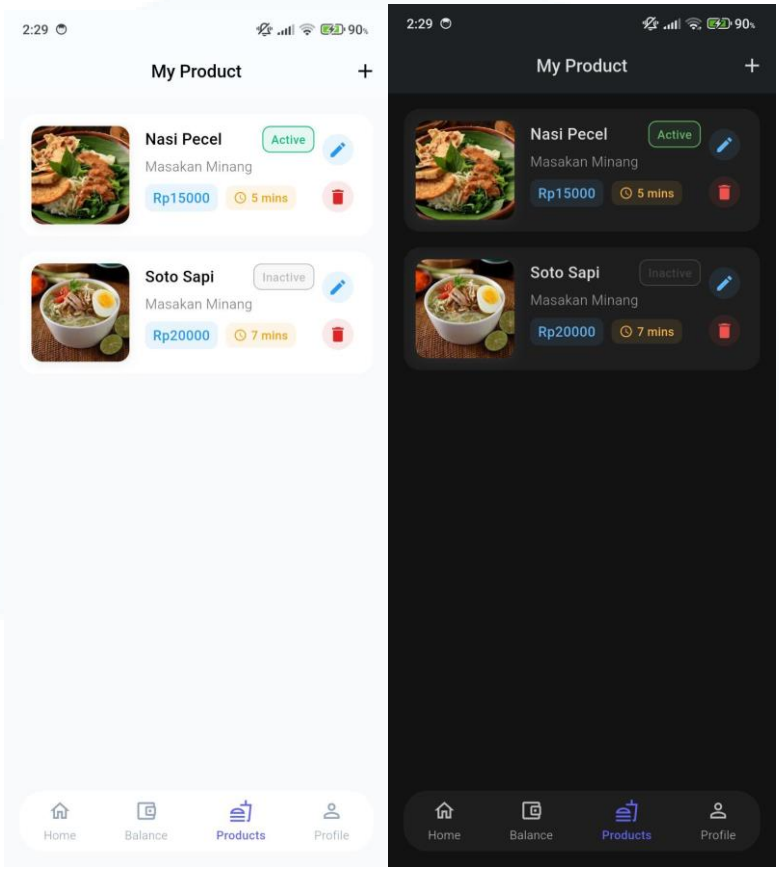
Ini adalah Gambar 3.72 yang berisi tampilan my balance screen tenant (kiri) dan penarikan saldo (kanan). *Tenant* bisa mengecek jumlah saldo yang dimiliki, serta melakukan penarikan dana menggunakan Gopay ataupun OVO. Namun sistem penarikan dana ini masih simulasi saja dan belum terintegrasi dengan *payment gateway* sungguhan.

e) My Product Screen

```dart
Widget _buildContent(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);
    final foodProvider = Provider.of<FoodProvider>(context);
    final currentTenantName = authProvider.tenantName ?? '';
    final currentSellerEmail = authProvider.user?.email ?? '';
    final sellerProducts = foodProvider.getProductsBySeller(currentSellerEmail)
        .where((product) => product.tenantName == currentTenantName)
        .toList();
    final isDarkMode = Theme.of(context).brightness == Brightness.dark;

    if (foodProvider.isLoading) {
      return Center(
        child: CircularProgressIndicator(
          valueColor: AlwaysStoppedAnimation<Color>(AppTheme.getPrimaryText(isDarkMode)),
        ),
      );
    }
```

Gambar 3.73 Code My Product Screen

Fungsi _buildContent pada Gambar 3.73 ini mengelola tampilan daftar produk penjual. Fungsi ini mengambil produk dari FoodProvider berdasarkan email penjual (currentSellerEmail) dan memfilter berdasarkan tenantName untuk memastikan hanya produk yang relevan ditampilkan. ListView.builder merender produk menggunakan ProductCard dengan melalui loadMoreProducts saat tombol "Load More" ditekan.



Gambar 3.74 UI My Product Screen

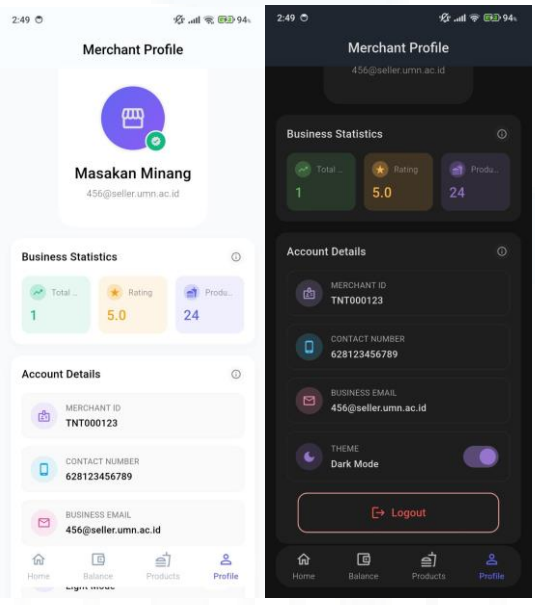
Gambar 3.74 ini adalah tampilan my product screen milik tenant. Kartu produk ini menampilkan foto menu, nama menu, nama tenant, harga, waktu persiapan, status (aktif atau nonaktif), serta button edit dan hapus. Status aktif dan nonaktif ini berfungsi untuk menghapus produk secara sementara dari tampilan milik customer jika seandainya tenant sedang tidak dapat menyediakan produk tersebut pada waktu tertentu dikarenakan stok habis atau alasan lainnya.

f) Profile Screen

```
Widget _buildBusinessStats(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);
    final ratingsProvider = Provider.of<RatingProvider>(context);
    final orderProvider = Provider.of<OrderProvider>(context);
    final merchantEmail = authProvider.sellerEmail ?? '';
    final averageRating = ratingsProvider.getAverageFoodRating(merchantEmail);
    final totalSales = orderProvider.getCompletedOrdersForMerchant(merchantEmail).length;
    final isDarkMode = Provider.of<ThemeNotifier>(context).isDarkMode;
```

Gambar 3.75 Code Profile Screen

Fungsi _buildBusinessStats pada Gambar 3.75 ini mengambil data statistik bisnisdari RatingProvider untuk rata-rata penilaian dan OrderProvider untuk jumlah total penjualan berdasarkan email penjual. Widget ini menampilkan kartu statistik untuk total penjualan, rating, dan jumlah produk (hard-coded ke '24' untuk saat ini).



Gambar 3.76 UI Profile Screen

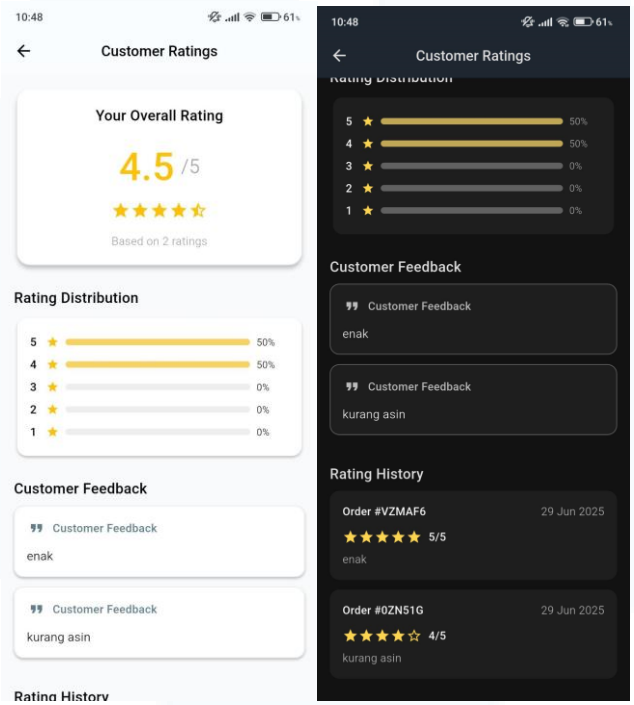
Gambar 3.76 ini adalah tampilan dari profile screen milik tenant. Profile screen ini berisi beberapa informasi seperti nama tenant, email tenant, statistik bisnis, serta identitas seperti id dan nomor hp. Dibawahnya juga terdapat icon untuk melakukan pergantian tema antara light mode dan dark mode, serta button untuk melakukan logout dari aplikasi.

g) Rating Screen

```dart
@override
  Widget build(BuildContext context) {
    final themeNotifier = Provider.of<ThemeNotifier>(context);
    final isDarkMode = themeNotifier.isDarkMode;
    return Theme(
      data: themeNotifier.currentTheme,
      child: Scaffold(
        backgroundColor: AppTheme.getBackground(isDarkMode),
        appBar: AppBar(
          title: Text(
            'Customer Ratings',
            style: TextStyle(
              color: AppTheme.getPrimaryText(isDarkMode),
              fontSize: 18,
              fontWeight: FontWeight.bold,
            ),
          ),
```

Gambar 3.77 Code Rating Screen

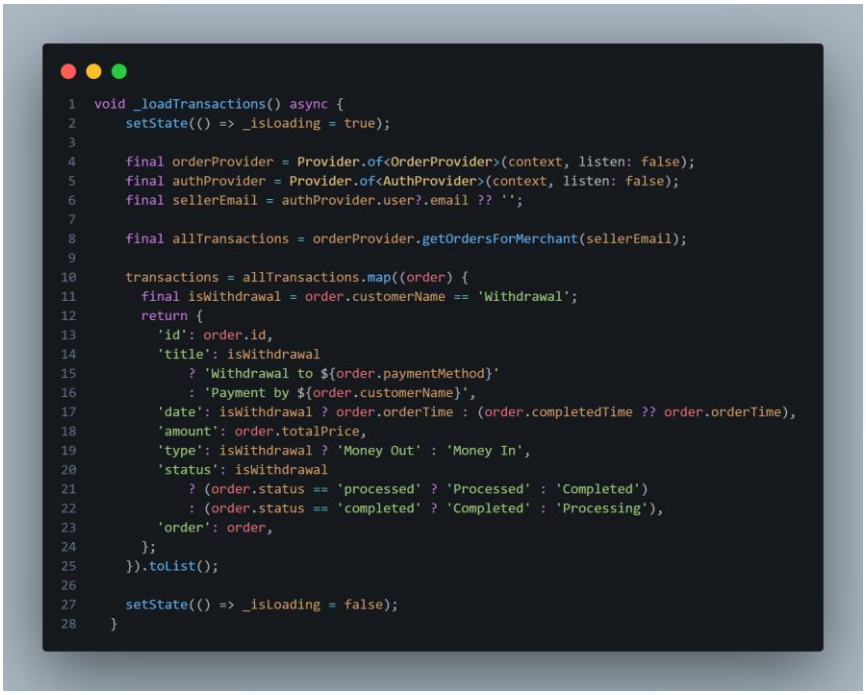
Fungsi build pada Gambar 3.77 mengelola pengambilan data penilaian untuk ringkasan penilaian melalui RatingSummaryCard. Widget ini mengambil rata-rata penilaian dan jumlah total penilaian dari RatingProvider berdasarkan email penjual dari AuthProvider. RatingSummaryCard ini memberikan gambaran cepat tentang performa penilaian penjual, meningkatkan pemahaman tentang kepuasan pelanggan.



Gambar 3.78 UI Rating Screen

Gambar 3.78 ini adalah tampilan dari rating screen yang dapat dilihat oleh tenant. Dari paling atas, terdapat overall rating atau nilai rata-rata rating dari keseluruhan pesanan customer, lalu dibawahnya ada rating distribution, yang menunjukkan pembagian persentase bintang dalam ratingnya. Dibawahnya lagi terdapat customer feedback serta rating history.

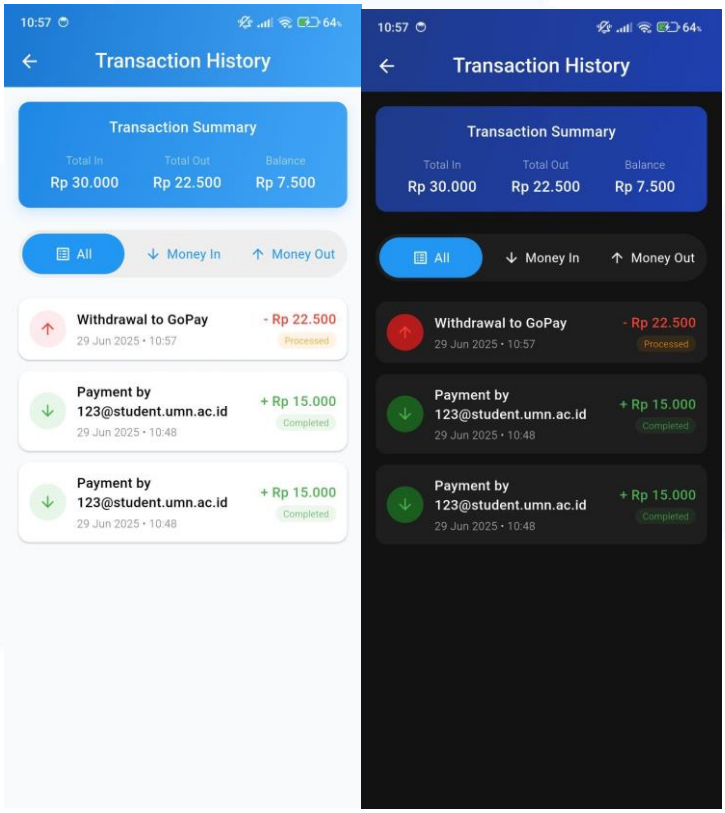
h) Transaction History Screen



```
void _loadTransactions() async {
    setState(() => _isLoading = true);

    final orderProvider = Provider.of<OrderProvider>(context, listen: false);
    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    final sellerEmail = authProvider.user?.email ?? '';

    final allTransactions = orderProvider.getOrdersForMerchant(sellerEmail);

    transactions = allTransactions.map((order) {
      final isWithdrawal = order.customerName == 'Withdrawal';
      return {
        'id': order.id,
        'title': isWithdrawal
            ? 'Withdrawal to ${order.paymentMethod}'
            : 'Payment by ${order.customerName}',
        'date': isWithdrawal ? order.orderTime : (order.completedTime ?? order.orderTime),
        'amount': order.totalPrice,
        'type': isWithdrawal ? 'Money Out' : 'Money In',
        'status': isWithdrawal
            ? (order.status == 'processed' ? 'Processed' : 'Completed')
            : (order.status == 'completed' ? 'Completed' : 'Processing'),
        'order': order,
      };
    }).toList();

    setState(() => _isLoading = false);
  }
```

Gambar 3.79 Code Transaction History Screen

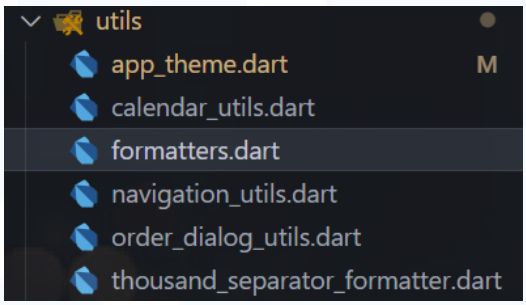
Fungsi _loadTransactions pada Gambar 3.79bertanggung jawab untuk mengambil data transaksi dari OrderProvider berdasarkan email penjual dari *AuthProvider*. Fungsi ini dijalankan pada *initState* untuk memuat transaksi saat layar dibuka. Setiap pesanan diubah menjadi objek transaksi dengan atribut seperti ID, judul, tanggal, jumlah, tipe (*Money In* atau *Money Out*), dan status (*Completed* atau *Processing*). Apabila statusnya masih processing, maka saldo tenant tidak akan bertambah dan transaksi juga belum akan dimasukkan ke transaction history.



Gambar 3.80 UI Transaction History Screen

Ini adalah Gambar 3.80 yang berisi tampilan transaction history screen yang dapat diakses oleh tenant kantin. Di paling atas menampilkan transaction summary, berisi total pemasukan dana, penarikan dana, serta saldo yang dimiliki saat ini. Lalu di bagian tengah terdapat data histori transaksi, yang dapat dikategorikan berdasarkan jenisnya (keluar atau masuk).Namun disini masih terdapat bug yang belum terselesaikan, dimana identitas yang muncul seharusnya nama customer, bukan emailnya.

11) **Utils**



Gambar 3.81 Isi Folder Utils

Bagian Gambar 3.81 ini memberikan gambaran garis besar tentang enam file utilitas (formatters.dart, navigation_utils.dart, thousand_separator_formatter.dart, order_dialog_utils.dart, calendar_utils.dart, dan app_theme.dart) yang digunakan dalam aplikasi U-Teen untuk mendukung fungsionalitas inti seperti pemformatan data, navigasi, pengelolaan dialog pesanan, pengelolaan kalender, dan tema visual. File-file ini dirancang untuk meningkatkan efisiensi pengembangan, konsistensi UI, dan pengalaman pengguna. Penjelasan berikut merangkum isi dan tujuan masing-masing file tanpa memisahkannya sebagai subbab terpisah.

File formatters.dart berisi kelas Formatters yang menyediakan dua objek NumberFormat dari paket intl untuk memformat angka sesuai standar Indonesia (id_ID). Menggunakan fungsi untuk memformat harga dengan simbol "Rp " tanpa desimal (contoh: Rp 10.000), sedangkan serta decimalFormat untuk memformat angka dengan pemisah ribuan (contoh: 1.000). File ini mendukung konsistensi pemformatan harga dan angka di seluruh aplikasi, seperti pada SellerEditProductScreen untuk input harga dan TenantMenuScreen untuk tampilan harga produk.

File navigation_utils.dart berfungsi untuk mengelola transisi antar layar dengan animasi slide dan fade. Fungsi ini membuat transisi mulus dengan durasi dan menambahkan efek transparansi (FadeTransition). Fungsi ini digunakan untuk navigasi di seluruh layar yang tidak menggunakan custom bottom navigation.

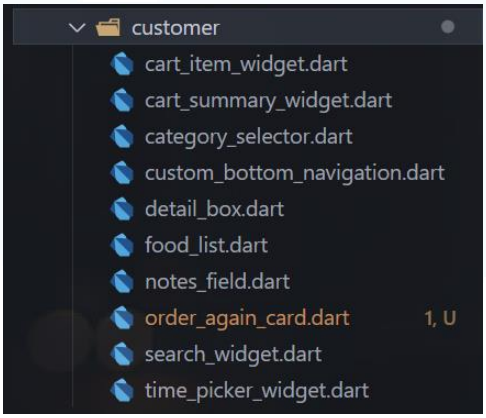
File thousand_separator_formatter.dart berisi kelas ThousandSeparatorInputFormatter yang mengimplementasikan TextInputFormatter untuk memformat input angka dengan pemisah ribuan sesuai standar Indonesia (id_ID). Kelas ini menghapus karakter non-angka, memformat input menggunakan NumberFormat.decimalPattern, dan menyesuaikan posisi kursor agar tetap akurat setelah pemformatan. Formatter ini digunakan pada TextFormField di SellerEditProductScreen untuk input harga, memastikan pengguna memasukkan harga dengan format yang benar (contoh: 100000 menjadi 100.000).

File order_dialog_utils.dart berisi kelas OrderDialogUtils yang menyediakan fungsi untuk menampilkan dialog terkait pengelolaan status pesanan, seperti showMarkAsReadyDialog untuk menandai pesanan sebagai siap dan showCancelOrderDialog untuk membatalkan pesanan. Dialog ini menggunakan tema dari AppTheme untuk konsistensi visual dan berinteraksi dengan OrderProvider untuk memperbarui status pesanan di Firestore.

File calendar_utils.dart berisi kelas CalendarUtils yang menyediakan fungsi untuk mengelola elemen kalender, seperti getEventIcon untuk memilih ikon berdasarkan jenis acara (misalnya, Icons.assignment untuk ujian, Icons.celebration untuk Natal), getEventColor untuk menetapkan warna berdasarkan acara (misalnya, oranye untuk ujian, hijau untuk Natal), dan formatEventDate untuk memformat tanggal acara sesuai standar Indonesia (id_ID). Fungsi ini digunakan pada CalendarSection di SellerHomeScreen untuk menampilkan acara dengan ikon, warna, dan format tanggal yang sesuai.

File app_theme.dart berisi kelas AppTheme yang mendefinisikan palet warna untuk mode terang dan gelap. Fungsi ini bekerja dengan cara mendefinisikan masing-masing warna terang dan gelap, lalu kemudian memasangkan warna-warna tersebut dengan membentuk nama variabel baru agar perubahan warna dapat sesuai dengan warna yang diinginkan.

12) Widget Customer



Gambar 3.82 Isi Folder Customer Widget

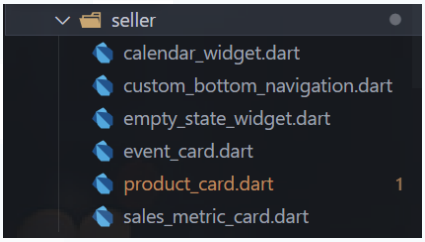
Folder *widget customer* pada Gambar 3.82 berisi serangkaian komponen antarmuka pengguna (*user interface*) yang dirancang khusus untuk mendukung fungsionalitas aplikasi U-Teen dari sisi pengguna (*customer*). Widget-widget diintegrasikan dengan Firebase untuk pengelolaan data dan autentikasi. Secara garis besar, folder ini berfungsi untuk menyediakan elemen-elemen visual dan interaktif yang memungkinkan pengguna untuk berinteraksi dengan aplikasi, seperti memesan makanan, mengelola keranjang belanja, mencari produk, dan mengatur preferensi pemesanan.

Widget utama tampilan customer yang termasuk dalam folder ini mencakup:

1. **CategorySelector**: Komponen untuk memilih kategori produk (misalnya, makanan, minuman, camilan) dengan tampilan yang responsif.
2. **CartSummaryWidget**: Menampilkan ringkasan keranjang belanja, termasuk jumlah item dan total harga, serta tombol untuk menyelesaikan pemesanan dengan validasi autentikasi dan pengecekan penjual.
3. **CustomBottomNavigation**: Navigasi bawah yang memungkinkan pengguna beralih antar layar utama seperti beranda, pesanan, keranjang, dan profil dengan animasi transisi yang mulus.
4. **CartItemWidget**: Menampilkan item dalam keranjang belanja dengan opsi untuk menambah/mengurangi jumlah atau menghapus item, dilengkapi validasi autentikasi.
5. **FoodList**: Menampilkan daftar produk makanan secara horizontal dengan data dari Firebase, mendukung filter kategori dan interaksi untuk melihat detail produk.
6. **NotesField**: Kolom input untuk menambahkan catatan khusus pada pesanan.
7. **DetailBox**: Menampilkan detail produk yang dipilih, termasuk gambar, harga, dan opsi untuk menambahkan ke keranjang atau favori.
8. **TenantList**: Menampilkan daftar tenant atau penjual sehingga memudahkan pengguna untuk menjelajahi produk berdasarkan tenant.

9. **SearchWidget**: Fitur pencarian produk makanan atau minuman. **TimePickerWidget**: Komponen untuk memilih waktu pengambilan pesanan dengan validasi waktu operasional (08:00–17:00).

13) Widget Tenant



Gambar 3.83 Isi Folder Tenant Widget

Folder *widget tenant* pada Gambar 3.83 berisi serangkaian komponen antarmuka pengguna (*user interface*) untuk mendukung kebutuhan penjual (*tenant*) dalam aplikasi U-Teen. Komponen-komponen ini dirancang untuk memfasilitasi pengelolaan produk, transaksi, dan informasi kalender akademik oleh penjual, dengan integrasi Firebase untuk pengelolaan data dan autentikasi. Widget utama dalam folder ini meliputi:

1. **SalesMetricCard**: Menampilkan metrik penjualan seperti jumlah pesanan atau pendapatan dalam bentuk kartu dengan ikon dan desain modern, mendukung visualisasi data yang jelas untuk penjual.
2. **SellerCustomBottomNavigation**: Navigasi bawah khusus untuk penjual, memungkinkan peralihan antar layar seperti beranda, saldo, produk, dan profil dengan animasi transisi.
3. **EmptyStateWidget**: Komponen untuk menampilkan pesan dan ikon ketika tidak ada data (misalnya, tidak ada pesanan atau produk).

4. **ProductCard**: Menampilkan informasi produk seperti nama, harga, waktu pembuatan, dan status (aktif atau nonaktif), dengan opsi untuk mengedit atau menghapus produk, serta menampilkan gambar produk menggunakan data Base64.
5. **CalendarSection**: Menyediakan antarmuka untuk melihat kalender akademik dengan filter berdasarkan periode waktu (1, 3, 6 bulan, atau rentang kustom) dan jenis acara (libur, ujian, acara), dengan data yang diambil dari layanan google kalender.
6. **EventCard**: Menampilkan detail acara akademik seperti ringkasan, deskripsi, dan rentang tanggal dalam format kartu dengan desain modern dan warna yang bervariasi berdasarkan jenis acara.

14) Widget Rating

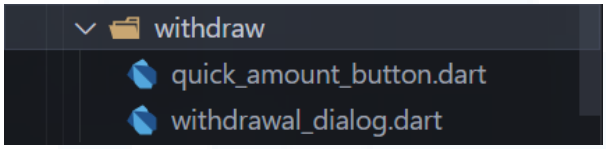
Gambar 3.84 Isi Folder Rating Widget

Folder *widget rating* pada Gambar 3.84 berisi serangkaian komponen antarmuka pengguna (*user interface*) untuk mendukung sistem penilaian dan umpan balik dalam aplikasi U-Teen. Komponen-komponen ini dirancang untuk memungkinkan pengguna memberikan penilaian terhadap makanan dan pengalaman aplikasi, serta menampilkan ringkasan dan distribusi penilaian bagi penjual. Widget-widget ini diintegrasikan dengan sistem tema (*dark/light mode*) melalui ThemeNotifier untuk memastikan tampilan yang konsisten dan responsif.

Berikut adalah penjelasan singkat tentang widget utama dalam folder ini:

1. **RatingDistributionCard**: Menampilkan distribusi penilaian makanan dalam bentuk diagram batang dengan persentase untuk setiap bintang (1-5), menggunakan data dari daftar pesanan. Widget ini membantu penjual memahami pola penilaian pelanggan.
2. **RatingDialog**: Dialog interaktif yang memungkinkan pelanggan memberikan penilaian (bintang 1-5) untuk kualitas makanan dan pengalaman aplikasi, serta menambahkan catatan custom terkait rating yang dia berikan.
3. **RatingHistoryCard**: Menampilkan riwayat penilaian untuk setiap pesanan, termasuk ID pesanan, tanggal penyelesaian, nilai penilaian, dan catatan pelanggan.
4. **RatingStars**: Komponen visual untuk menampilkan ikon bintang berdasarkan nilai penilaian, mendukung tampilan bintang penuh, setengah, atau kosong.
5. **FeedbackCard**: Menampilkan umpan balik pelanggan dalam bentuk kartu sederhana.
6. **RatingSummaryCard**: Menyajikan ringkasan penilaian keseluruhan dengan menampilkan rata-rata penilaian, jumlah total penilaian, dan visualisasi bintang, sehingga memberikan gambaran tentang performa penjual.

15) **Widget Withdraw**

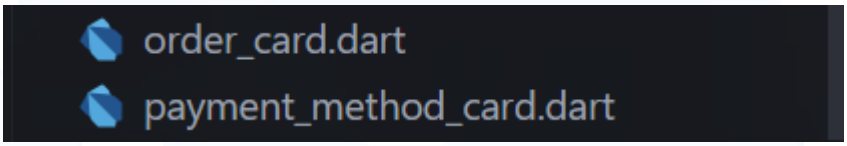


Gambar 3.85 Isi Folder Withdraw Widget

Folder widget pada Gambar 3.85 berisi penarikan saldo berisi komponen antarmuka pengguna (user interface) yang dikembangkan menggunakan Flutter dan Dart untuk mendukung fitur penarikan dana (withdrawal) bagi penjual (tenant) dalam aplikasi U-Teen. Komponen-komponen ini dirancang untuk memungkinkan penjual menarik saldo mereka dengan mudah dan aman, dengan integrasi tema (dark/light mode) melalui ThemeNotifier serta format penulisan angka yang ramah pengguna melalui utilitas seperti ThousandSeparatorInputFormatter dan Formatters. Widget ini fokus pada pengalaman pengguna yang intuitif, responsif, dan estetis. Berikut adalah penjelasan singkat tentang widget utama dalam folder ini:

a. WithdrawalDialog: Dialog untuk memungkinkan penjual menarik dana dari saldo mereka. Dialog ini menampilkan saldo yang tersedia, dan juga input jumlah penarikan dana secara manual, dan menyediakan opsi cepat untuk memilih persentase saldo (25%, 50%, 75%, 100%) yang dipanggil dari fungsi QuickAmountButton.
b. QuickAmountButton: Tombol kecil untuk memilih persentase saldo yang akan ditarik (25%, 50%, 75%, atau 100%) dengan cepat.

16) **Widget Order Card, Payment Method, and Order Status**



Gambar 3.86 File Widget Order Card dan Payment Method

Ini adalah Gambar 3.86 isi dari folder *widget* berisi komponen antarmuka pengguna yang dapat digunakan baik oleh pelanggan (*customer*) maupun penjual (*tenant*) dalam aplikasi U-Teen. Berbeda dengan widget dalam folder spesifik seperti *customer*, *tenant*, *rating*, dan *withdraw*, widget dalam folder ini bersifat umum dan tidak terbatas pada peran pengguna tertentu. Komponen-komponen ini dirancang untuk mendukung fungsionalitas bersama seperti menampilkan status pesanan, metode pembayaran, atau tombol status. Berikut adalah penjelasan singkat tentang tiga widget utama dalam folder ini:

1. **OrderCard**: Menampilkan informasi pesanan dalam format kartu yang dapat digunakan oleh pelanggan (untuk melihat pesanan mereka) atau penjual (untuk mengelola pesanan). Widget ini menyesuaikan tampilan yang muncul berdasarkan parameter isSellerView untuk mengecek apakah yang mengakses widget ini adalah tenant ataupun pelanggan.
2. **PaymentMethodCard**: Menampilkan kartu metode pembayaran yang dapat dipilih oleh pengguna (baik pelanggan maupun penjual) untuk transaksi atau penarikan dana.

3.5.2 Kendala yang Ditemukan

Berikut adalah beberapa kendala utama yang dihadapi penulis selama proses pengembangan aplikasi U-Teen, khususnya dalam lingkup pengembangan *front-end* dan *back-end*:

A. Adaptasi terhadap bahasa pemrograman Dart dan Flutter

Pengembangan aplikasi ini menggunakan Flutter, yang berbasis bahasa Dart, bahasa yang sebelumnya belum digunakan oleh penulis. Perbedaan sintaks, struktur logika, serta pendekatan *state management* yang khas pada Flutter memerlukan proses pembelajaran tambahan.

B. Kesulitan instalasi Flutter dan sinkronisasi environment

Flutter tidak dapat dijalankan dengan konfigurasi biasa. Seluruh komponen pendukung seperti Android SDK, Java Development Kit, dan Gradle emulator harus sinkron versinya. Jika salah satu komponen tidak kompatibel, maka proses build akan gagal. Penulis mengalami berbagai kendala teknis seperti error pada konfigurasi build.gradle, konflik versi SDK, serta ketidakcocokan antara Flutter SDK dengan versi *dependency* tertentu. Permasalahan ini sangat menyita waktu karena harus dipecahkan satu per satu tanpa referensi tunggal yang pasti.

C. Gagal mengintegrasikan API pembayaran dari Midtrans (sandbox)

Saat mencoba mengintegrasikan sistem pembayaran digital menggunakan API sandbox Midtrans, penulis menemui banyak kendala teknis seperti proses *server-to-server authentication* dan konfigurasi *endpoint* yang memerlukan pemahaman lanjutan tentang sistem transaksi yang aman. Proses ini cukup kompleks dan tidak dapat diselesaikan dalam waktu proyek yang terbatas.

D. Kesulitan saat debugging aplikasi

Struktur aplikasi yang dibangun cukup kompleks dan terdiri dari banyak folder antara lain *auth, data, model, provider, screen, utils, services* dan *widget*. Ketika terjadi *bug*, pelacakan kesalahan tidak bisa dilakukan secara langsung melalui tampilan antarmuka. Penulis harus menelusuri rantai logika yang tersebar di banyak file, mengecek aliran data, dan memastikan bahwa setiap komponen menerima data sesuai yang diharapkan. Hal ini mempersulit proses identifikasi error, terutama ketika aplikasi sudah mulai terhubung dengan Firebase dan melibatkan asinkronisasi data.

3.5.3 Solusi atas Kendala yang Ditemukan

Berikut adalah solusi yang diterapkan penulis untuk mengatasi setiap kendala di atas:

A. Menggunakan bantuan teknologi AI untuk memahami bahasa Dart dan Flutter

Untuk mengatasi kendala dalam memahami bahasa Dart dan struktur kerja Flutter, penulis secara aktif memanfaatkan bantuan teknologi berbasis kecerdasan buatan, antara lain ChatGPT, Grok, DeepSeek, dan Github Copilot. Bantuan ini digunakan untuk mencari penjelasan konsep pemrograman yang belum familiar, memahami logika *state management*, serta memperoleh kode yang dibutuhkan untuk pengembangan aplikasi.

Dengan mengajukan pertanyaan yang spesifik sesuai kebutuhan, penulis dapat memperoleh penjelasan yang langsung menjawab kebutuhan teknis, seperti struktur fungsi, pengelolaan data melalui *provider*, serta implementasi fitur UI secara efisien. Proses ini mempercepat waktu pengembangan aplikasi secara signifikan, dan mendukung penyelesaian pengembangan fitur dalam waktu yang lebih efisien.

B. Mengikuti instruksi teknis dan troubleshooting menggunakan bantuan AI

Proses instalasi dan konfigurasi Flutter environment juga dipermudah melalui bantuan AI interaktif yang menyediakan panduan *step-by-step* untuk menyelesaikan error yang muncul. Setiap kendala yang terjadi pada tahap instalasi seperti konflik versi Gradle, kegagalan *build*, dan error pada emulator diatasi dengan mencoba solusi yang disarankan oleh platform tersebut secara berulang.

Solusi seperti penyesuaian versi Gradle di build.gradle, sinkronisasi SDK, dan pemasangan plugin tambahan dilakukan berdasarkan saran sistematis yang dihasilkan AI. Strategi ini terbukti mempercepat penyesuaian environment yang awalnya memakan waktu cukup lama.

C. Mengganti API pembayaran Midtrans dengan sistem simulasi internal

Karena proses integrasi sistem pembayaran Midtrans membutuhkan konfigurasi teknis tingkat lanjut dan melibatkan sistem otentikasi server yang belum dapat diakses secara penuh dalam lingkup proyek, penulis memutuskan untuk menggunakan sistem simulasi transaksi sendiri. Sistem ini meniru alur transaksi nyata, termasuk input nominal dan status pembayaran dengan tujuan untuk mensimulasikan alur keluar masuk saldo, namun tanpa menggunakan API dari Midtrans.

Dengan pendekatan ini, seluruh alur pemesanan dan pembayaran tetap dapat diuji dari sisi pengguna, sekaligus menjaga struktur sistem tetap modular jika ke depan ingin diintegrasikan dengan layanan pembayaran nyata. Skema simulasi ini juga memungkinkan pengujian fitur-fitur lain tanpa terganggu kendala eksternal yang belum bisa diselesaikan selama masa pengembangan.

D. Mengimplementasikan strategi penanganan kesalahan (error handling) menggunakan pendekatan defensif

Dalam menghadapi kesulitan proses *debugging* pada sistem yang kompleks, penulis menerapkan pendekatan *error handling defensif* sebagai bagian dari strategi perbaikan dan pelacakan kesalahan. Pendekatan ini melibatkan penggunaan struktur try-catch untuk mengantisipasi kemungkinan terjadinya kesalahan saat aplikasi mengeksekusi proses yang bersifat kritis, seperti pengolahan data gambar dalam format Base64.

Selain itu, penulis juga menambahkan fungsi *log output* berupa debugPrint untuk menampilkan informasi kesalahan secara langsung ke *console log*. Dengan menampilkan pesan kesalahan secara spesifik dan terstruktur, proses pelacakan sumber error menjadi lebih efisien dan terarah. Strategi ini memungkinkan aplikasi tetap berjalan dengan stabil meskipun terjadi kesalahan pada salah satu bagian sistem dalam beberapa kasus, serta memberikan ruang untuk memperbaiki bug tanpa harus menghentikan seluruh proses.

.