Project Defense Guide

🎓 Dee's Budget App - Project Defense Guide

Title: Design and Implementation of a Mobile Budget & Expense Tracking App
Student: Deborah Chibuzo Okere
Matric Number: NOU213070643
Institution: National Open University of Nigeria
Academic Session: 2024/2025

Table of Contents

1. Project Overview
2. Technology Stack
3. Folder Structure
4. Core Application Architecture
5. Data Models
6. Services Layer
7. User Interface Screens
8. Firebase Database Schema
9. Key Innovations
10. Application Flow
11. Dependencies
12. Defense Tips

Project Overview

Dee's Budget App is a mobile financial management application designed specifically for Nigerian users with diverse income structures. The application addresses the critical gap in existing Personal Finance Management (PFM) tools by providing **income-type personalized** budgeting experiences.

Target Users

Income Type	Description	Examples
Fixed Earners	Stable monthly income	Salaried workers, civil servants
Variable Earners	Irregular income patterns	Freelancers, gig workers, traders
Hybrid Earners	Combination of both	Employees with side businesses

Key Problem Solved

Traditional budgeting apps assume stable income patterns, which fails 40%+ of Nigerians in the gig/informal economy. This app provides **personalized dashboards** tailored to each earning pattern.

🛠 Technology Stack

Frontend Technologies

Technology	Version	Purpose
Flutter	3.9.0	Cross-platform UI framework
Dart	^3.9.0	Programming language
Material Design	Built-in	Google's UI design system

Backend Technologies (Firebase)

Service	Purpose
Firebase Authentication	User sign-in/sign-up (Email & Google OAuth)
Cloud Firestore	NoSQL real-time database for all app data
Firebase Messaging	Push notifications
Firebase Hosting	Web app deployment

Development Tools

Tool	Purpose
Android Studio / VS Code	IDE for development
Git	Version control
Figma	UI/UX design prototyping
Gradle	Android build system

Folder Structure

Root Level

```
budget_tracker_app/
├── android/              # Android-specific configuration
├── ios/                  # iOS-specific configuration
├── web/                  # Web platform files
├── lib/                  # Main Dart source code
├── test/                 # Unit and widget tests
├── assets/               # Images, fonts, static files
├── pubspec.yaml          # Dependencies & project config
├── firebase.json         # Firebase hosting configuration
└── README.md             # Project documentation
```

Root Configuration Files

File	Purpose
`pubspec.yaml`	Defines app dependencies, version (1.0.0+1), and Flutter settings
`firebase.json`	Firebase Hosting configuration, project ID: `dee-s-budget-app`
`analysis_options.yaml`	Dart/Flutter code linting rules
`.firebaserc`	Links local project to Firebase project

Platform Folders

Folder	Key Files	Purpose
	`google-services.json`,	Android build

`android/`	`build.gradle`, `AndroidManifest.xml`	configuration and Firebase setup
`ios/`	`Info.plist`, `GoogleService-Info.plist`	iOS build configuration
`web/`	`index.html`, `manifest.json`	Web entry point and PWA configuration

🏗 Core Application Architecture

`lib/` Directory Structure

```
lib/
├── main.dart                    # App entry point
├── firebase_options.dart        # Firebase credentials (auto-
generated)
├── constants/                   # App-wide constants
├── models/                      # Data structures (5 files)
├── screens/                     # UI components (15+ files)
├── services/                    # Business logic (5 files)
├── theme/                       # Visual styling
└── utils/                       # Utility functions
```

Entry Point (`main.dart`)

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform);
  await NotificationService.initialize();
  runApp(const BudgetTrackerApp());
}
```

Key Responsibilities:

1. Initialize Firebase connection
2. Setup notification service
3. Configure app routes (`/`, `/login`, `/signup`, `/home`, `/rules`)
4. Listen for authentication state changes
5. Setup periodic alert checks (every 30 minutes)

📊 Data Models

Location: `lib/models/`

1. UserModel (`user_model.dart`)

```
class UserModel {
  final String uid;
  final String email;
  final String fullName;
  final String username;
  final String incomeType;     // 'fixed', 'variable', 'hybrid'
  final DateTime createdAt;
  final double? monthlyIncome;
  final double? targetSavings;
}
```

Purpose: Stores user profile information including the crucial `incomeType` field that determines which dashboard to display.

2. TransactionModel (`transaction_model.dart`)

```
        class TransactionModel {
          final String id;
          final String type;            // 'income' or 'expense'
          final String category;       // e.g., 'Food', 'Transport',
'Salary'
          final double amount;
          final String description;
          final DateTime date;
          final String? source;         // Income source for variable earners
          final double? savingsAllocation;  // Amount auto-allocated to
savings
        }
```

Purpose: Records all financial transactions with support for savings allocation tracking.

3. BudgetModel (`budget_model.dart`)

```
        class BudgetModel {
          String id;
          String category;
          double amount;              // Budget limit
          double spent;               // Current spending
          String period;              // 'weekly' or 'monthly'
          DateTime startDate;
          DateTime endDate;
          bool isAutoCreated;         // Created by allocation rule?
        }
```

Computed Properties:

- `remaining` → `amount - spent`
- `percentSpent` → `(spent / amount * 100)`

4. RuleModel (`rule_model.dart`)

```
        class RuleModel {
          String id;
          String name;
          String type;                // 'allocation', 'savings', 'alert',
'income_allocation'
          Map<String, dynamic> conditions;
          Map<String, dynamic> actions;
          int priority;               // 1-5, higher = processed first
          bool isActive;

          // Income allocation fields
          String? incomeSource;       // 'all', 'Gig Work', 'Gift', etc.
          String? allocationType;     // 'percentage' or 'fixed'
          double? allocationValue;
          String? targetCategory;
        }
```

Purpose: Powers the rule engine for automated fund allocation and alerts.

5. ConflictModel (`conflict_model.dart`)

Purpose: Detects and manages conflicts between rules (e.g., two rules allocating 60% each = 120% conflict).

⚙ Services Layer

Location: `lib/services/`

1. FirebaseService (`firebase_service.dart`) - 1,722 lines

The core service handling ALL Firebase operations:

Category	Methods
Authentication	`signUp()`, `signIn()`, `signOut()`
User Profile	`getUserProfile()`, `updateUserProfile()`
Transactions	`addTransaction()`, `getTransactions()`, `deleteTransaction()`
Budgets	`addBudget()`, `getBudgets()`, `updateBudget()`, `deleteBudget()`
Rules	`addRule()`, `getRules()`, `updateRule()`, `deleteRule()`
Reset Logic	`checkAndPerformMonthlyReset()`, `resetWeeklyBudgets()`
Analytics	`getSpendingTrends()`, `getArchivedBudgets()`

Smart Features:

- Auto-archives budget data monthly for trend analysis
- Triggers allocation rules when income is added
- Updates budget spending when expenses are logged

2. IncomeAllocationService (`income_allocation_service.dart`)

Purpose: Automatically allocates income to budget categories based on user-defined rules.

Example Flow:

1. User receives ₦100,000 salary
2. Rule: "Allocate 20% to Savings" triggers
3. ₦20,000 automatically added to Savings budget

3. AlertService (`alert_service.dart`)

Manages smart notifications:

- Budget overspending alerts (e.g., "Food budget at 90%!")
- Salary alerts for fixed earners
- Runway warnings for variable earners (days until funds depleted)
- Low balance warnings

4. NotificationService (`notification_service.dart`)

Purpose: Handles local push notifications using `flutter_local_notifications` package.

5. ConflictResolutionService (`conflict_resolution_service.dart`)

Detects rule conflicts:

- Duplicate category rules
- Over-allocation (rules allocating >100% of income)
- Priority conflicts

📱 User Interface Screens

Location: `lib/screens/`

Authentication Flow (`screens/auth/`)

Screen	File	Purpose

Splash	`splash_screen.dart`	App loading with branding
Login	`login_screen.dart`	Email/password authentication
Sign Up	`signup_screen.dart`	Registration with **income type selection**

Home & Dashboard

Screen	File	Purpose
Home	`home_screen.dart`	Navigation hub, loads appropriate dashboard

Personalized Dashboards (`screens/widgets/`)

Dashboard	File	Features
Fixed Earner	`fixed_earner_dashboard.dart`	Monthly budget tracking, payday countdown, safe-to-spend calculation
Variable Earner	`variable_earner_dashboard.dart`	Weekly tracking, **runway calculator**, income volatility alerts
Hybrid Earner	`hybrid_earner_dashboard.dart`	Dual-stream tracking, cross-funding journal

Supporting Widgets:

- `budget_tracker_screen.dart` - Visual budget progress bars
- `enhanced_alert_banner.dart` - Smart alert display
- `monthly_reset_manager.dart` - Handles period resets

Transaction Management (`screens/transactions/`)

Screen	File	Purpose
Add Transaction	`add_transaction_screen.dart`	Form with categories, amounts, optional savings allocation
Transaction List	`transactions_list_screen.dart`	Chronological history with filters

Rules Engine (`screens/rules/`)

Screen	File	Purpose
Rules List	`rules_screen.dart`	View/manage automation rules
Add Rule	`add_rule_screen.dart`	Create allocation, savings, or alert rules

Utilities

Folder	Files	Purpose
`lib/utils/`	`currency_formatter.dart`	Formats with **Nigerian Naira (₦)**
	`data_validator.dart`	Input validation
	`financial_calculator.dart`	Runway, percentage calculations

`lib/theme/`	`app_theme.dart`	Color palette, button styles

Firebase Database Schema

```
Firestore Database Structure
============================

users/{userId}
├── uid: string
├── email: string
├── fullName: string
├── username: string
├── incomeType: "fixed" | "variable" | "hybrid"
├── monthlyIncome: number
├── targetSavings: number
├── createdAt: timestamp
│
├── userTransactions/{transactionId}
│   ├── type: "income" | "expense"
│   ├── category: string
│   ├── amount: number
│   ├── description: string
│   ├── date: timestamp
│   ├── source: string (optional)
│   └── savingsAllocation: number (optional)
│
├── userBudgets/{budgetId}
│   ├── category: string
│   ├── amount: number
│   ├── spent: number
│   ├── period: "weekly" | "monthly"
│   ├── startDate: timestamp
│   └── endDate: timestamp
│
├── userRules/{ruleId}
│   ├── name: string
│   ├── type: "allocation" | "savings" | "alert"
│   ├── conditions: map
│   ├── actions: map
│   ├── priority: number (1-5)
│   ├── isActive: boolean
│   └── [allocation-specific fields]
│
└── archivedBudgets/{year-month}
    └── [historical budget data for analytics]
```

Security Rules

```
        rules_version = '2';
        service cloud.firestore {
          match /databases/{database}/documents {
            match /users/{userId} {
              // Only authenticated users can access their own data
              allow read, write: if request.auth != null && request.auth.uid
== userId;

              match /userTransactions/{transactionId} {
                allow read, write: if request.auth != null &&
request.auth.uid == userId;
              }

              match /userBudgets/{budgetId} {
                allow read, write: if request.auth != null &&
request.auth.uid == userId;
              }

              match /userRules/{ruleId} {
                allow read, write: if request.auth != null &&
request.auth.uid == userId;
              }
            }
        }
      }
```

Key Innovations

1. Income-Type Personalization

Unlike generic budgeting apps, this app provides **three distinct user experiences**:

Income Type	Dashboard Cycle	Unique Features
Fixed	Monthly	Payday countdown, safe-to-spend
Variable	Weekly	**Runway calculator** (days until ₦0)
Hybrid	Both	Dual-stream tracking, cross-funding

2. Rule Engine (Automation)

Users can create intelligent rules:

Rule Type	Example
Allocation	“Allocate 20% of all gig income to Savings”
Alert	“Notify when Food budget reaches 80%”
Savings	“Auto-save ₦5,000 from every salary”

Features:

- Priority-based execution (1-5)
- Conflict detection
- Income source filtering

3. Context-Aware Design

Feature	Nigerian Context
Cash-first	Supports manual entry (Nigeria’s cash-dominant economy)
Naira (₦)	Default currency formatting
Offline-capable	Works with intermittent connectivity
Flexible income	Accommodates irregular payments

4. Smart Budget Management

- **Auto-reset**: Monthly/weekly based on income type
- **Archiving**: Historical data preserved for trend analysis
- **Conflict detection**: Prevents over-allocation errors

Application Flow

```
┌─────────────────────────────────────────────────────────────┐
│                         APP LAUNCH                          │
└─────────────────────────────┬───────────────────────────────┘
                              ▼
                    ┌─────────────────┐
                    │  Firebase Init  │
                    └────────┬────────┘
                             ▼
                    ┌─────────────────┐
                    │ Authenticated? │
                    └───────┬────────┘
                            │
                ┌───────────┴───────────┐
                ▼                       ▼
         ┌─────────────────┐     ┌─────────────────┐
         │  Login Screen   │     │  Home Screen    │
         └────────┬────────┘     └────────┬────────┘
                  │                        │
                  └──────────┬─────────────┘
                             ▼
                    ┌─────────────────┐
                    │ Get Income Type │
                    └────────┬────────┘
                             │
          ┌──────────────────┼──────────────────┐
          ▼                  ▼                  ▼
    ┌───────────┐     ┌───────────┐     ┌───────────┐
    │   Fixed   │     │ Variable  │     │  Hybrid   │
    │ Dashboard │     │ Dashboard │     │ Dashboard │
    └─────┬─────┘     └─────┬─────┘     └─────┬─────┘
          │                 │                 │
          └─────────────────┼─────────────────┘
                            ▼
              ┌─────────────────────────┐
              │      User Actions:      │
              │  • Add Transaction      │
              │  • Create Budget        │
              │  • Setup Rules          │
              │  • View Reports         │
              └────────────┬────────────┘
                           ▼
              ┌─────────────────────────┐
              │   Firebase Sync &       │
              │   Rule Processing       │
              └────────────┬────────────┘
                           ▼
              ┌─────────────────────────┐
              │   Alerts Triggered?     │
              │   → Send Notification   │
              └─────────────────────────┘
```

📦 Dependencies

From `pubspec.yaml`

Package	Version	Purpose
`firebase_core`	^4.1.1	Firebase SDK initialization
`firebase_auth`	^6.1.0	Email/Google authentication
`cloud_firestore`	^6.0.2	NoSQL database operations
`firebase_messaging`	^16.0.2	Push notification handling
`flutter_local_notifications`	^15.1.3	Local notifications display
`shared_preferences`	^2.2.2	Local storage for preferences
`intl`	^0.20.2	Date formatting & internationalization
`cupertino_icons`	^1.0.8	iOS-style icons

Dev Dependencies

Package	Purpose
`flutter_test`	Widget & unit testing
`flutter_lints`	Code quality rules

🎯 Defense Tips

Technical Questions

Q: Why Flutter?

> Flutter enables cross-platform development (Android, iOS, Web) from a single codebase, reducing development time by 40-50%. Hot reload accelerates development. Rich widget library provides consistent Material Design UI.

Q: Why Firebase?

> Firebase is serverless (no backend management needed), provides real-time sync, has built-in authentication, and scales automatically. The free tier is sufficient for academic projects.

Q: Why NoSQL (Firestore) over SQL?

> Firestore's document model naturally fits our hierarchical data (users → transactions/budgets/rules). Real-time listeners provide instant UI updates. Offline caching ensures app works without internet.

Q: How does the income-type personalization work?

> During signup, users select their income type. The `incomeType` field in `UserModel` determines which dashboard component to render in `home_screen.dart`. Each dashboard has tailored metrics and reset cycles.

Innovation Defense

Q: What makes this different from existing apps?

> 1. Income-type personalization (fixed/variable/hybrid dashboards)
> 2. Nigerian context (Naira currency, cash-first design)
> 3. Rule engine for automated allocations
> 4. Runway calculator for gig workers

Q: What is the "Runway" feature?

> Runway calculates "days until funds depleted" based on current balance and daily spending average. Critical for variable earners who need to know how long their funds will last.

Code Metrics

Metric	Value
Total Dart files	~30
Firebase Service	1,722 lines
Data Models	5 classes
Personalized Dashboards	3
Rule Types Supported	4 (allocation, savings, alert, income_allocation)

📚 References

- Flutter Documentation
- Firebase Documentation

- Material Design Guidelines
- Dart Language Tour

Document Version: 1.0
Last Updated: January 2026
Prepared For: Project Defense

Good luck with your defense! 🎓