Flutter Form Application Documentation

App Overview

This Flutter application implements a multi-tab form system with SQLite database integration. The app allows users to create, view, edit, and delete form submissions through an intuitive interface. The form is divided into three sequential tabs, with data persistence across tab navigation and app lifecycle events.

Key Features

1. Multi-Tab Form Implementation

- **Three-Tab Structure**: Form divided into Personal Details, Address, and Contact Details
- **Sequential Navigation**: Users move through tabs using "Next" buttons
- **TabController**: Manages tab state and transitions
- **Form Data Preservation**: Data persists when switching between tabs

2. SQLite Database Integration

- **Relational Database Design**: Three tables with proper relationships
  - `personal_details`: Primary table (one-to-many relationship with other tables)
  - `address`: Contains address information linked to personal details
  - `contact_details`: Stores contact information linked to personal details
- **CRUD Operations**: Complete implementation of Create, Read, Update, and Delete functionalities
- **Foreign Key Constraints**: Ensures data integrity with cascading delete operations

3. Data Persistence & Lifecycle Management

- **Form Data Object**: Central data model that persists across tab transitions
- **Widget Lifecycle Hooks**: `initState()` and `dispose()` methods used to load and save data
- **Controller Management**: Proper initialization and disposal of text controllers
- **State Restoration**: Form values preserved when navigating through tabs

4. Record Management Features

- **Dashboard View**: Lists all submitted forms with swipe-to-delete functionality
- **View Mode**: Detailed view of all form fields in read-only mode
- **Edit Mode**: Toggle to edit existing form data with immediate database updates
- **Delete Function**: Swipe gesture for record deletion with confirmation feedback

5. User Experience Enhancements

- **Validation**: Form validation ensures all required fields are completed
- **Visual Feedback**: Success/error messages via SnackBar notifications
- **Consistent UI**: Unified styling and theme throughout the application
- **Loading States**: Visual indicators during data loading operations

Technical Implementation Details

Data Models

The app implements three primary data models:

- `PersonalDetails`: Stores basic user information (first name, last name, date of birth, gender)
- `Address`: Contains address information (address line, pin code, city, state, country)
- `ContactDetails`: Manages contact information (email/phone) with verification status
- `FormData`: Container class that holds instances of all three models for unified form management

Database Architecture

The SQLite database is structured with:

- **Three Tables**: Each corresponding to a form tab
- **Primary Keys**: Auto-incremented integer IDs
- **Foreign Keys**: Linking child tables to the personal_details table
- **Cascading Deletes**: Ensuring related records are removed when a personal details record is deleted

Screen Flow

1. **Dashboard Screen**: Entry point showing all existing form submissions
2. **Form Screen**: Contains the three-tab form for new submissions
3. **View Screen**: Displays completed form data with edit capability

State Management

- Local state management using Flutter's `setState()`
- Form data passed between tabs using a shared `FormData` object
- Database operations abstracted in the `DatabaseHelper` class

Code Organization

- **Models**: Data structures and transformation logic

- **Screens**: User interface components and screens
- **Utils**: Helper classes, constants, and database operations
- **Widgets**: Reusable UI components

Implementation Challenges & Solutions

Challenge 1: Data Persistence Between Tabs

Solution: Implemented a shared `FormData` object passed to each tab, with proper lifecycle methods to save and retrieve data when tabs are navigated.

Challenge 2: Database Relationships

Solution: Created a one-to-many relationship between tables using foreign keys and cascading delete operations, ensuring data integrity.

Challenge 3: Edit Functionality

Solution: Built a toggle mechanism in the View screen that switches between read-only and editable states, with database updates on save.

Challenge 4: Form Validation

Solution: Added validation checks before tab navigation and form submission to ensure data completeness and integrity.

Technical Architecture Diagram

```
App Structure
├── Models
│   ├── PersonalDetails
│   ├── Address
│   ├── ContactDetails
│   └── FormData
├── Screens
│   ├── DashboardScreen
│   ├── FormScreen
│   │   ├── FormTabOne (PersonalDetails)
│   │   ├── FormTabTwo (Address)
│   │   └── FormTabThree (ContactDetails)
│   └── ViewScreen
├── Utils
│   ├── Constants
│   ├── DatabaseHelper
│   └── StringHelper
└── Widgets
    ├── CardWidget
    └── IconButtonWidget
```

Database Schema

```
personal_details
├── id (PRIMARY KEY)
├── first_name
├── last_name
├── date_of_birth
└── gender

address
├── id (PRIMARY KEY)
├── personal_details_id (FOREIGN KEY)
├── address_line
├── pin_code
├── city
├── state
└── country

contact_details
├── id (PRIMARY KEY)
├── personal_details_id (FOREIGN KEY)
├── contact_type
├── contact_value
└── is_verified
```