МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

Разработка мобильного приложения с базой данных.

Отчёт по лабораторной работе №6

по дисциплине

«Технологии программирования»

Разработал студент гр. ИВТб-2301-05-00 _____ /Черкасов А. А./

(подпись)

Преподаватель _____ /Пащенко Д. Э./

(подпись)

Киров

2025

Цели лабораторной работы

- освоить практические навыки создания мобильных приложений с локальной базой данных;
- познакомиться с основами работы с SQLite в Flutter;
- изучить паттерн управления состоянием Riverpod;
- понять принципы построения многооконных приложений с CRUD-операциями.

Задание

Разработать многооконное мобильное приложение, которое хранит информацию в локальной базе данных SQLite и позволяет изменять её. Тема приложения: «Моя Библиотека» — управление коллекцией книг и авторов.

Реализация приложения

В рамках лабораторной работы разработано Flutter-приложение «Моя Библиотека», предназначенное для управления коллекцией книг и авторов с использованием локальной базы данных SQLite.

Структура приложения

Приложение состоит из трёх основных экранов, доступных через нижнюю навигационную панель:

- **Добавить** — экран для добавления новых книг и авторов;
- **Книги** — список всех книг с возможностью редактирования и удаления;
- **Авторы** — список всех авторов с возможностью удаления.

База данных

Приложение использует SQLite базу данных с двумя таблицами:

1. **books** — хранит информацию о книгах:
    - `id` (INTEGER PRIMARY KEY) — уникальный идентификатор;
    - `title` (TEXT NOT NULL) — название книги;
    - `publication_year` (INTEGER NOT NULL) — год издания;
    - `genre` (TEXT NOT NULL) — жанр;
    - `isbn` (TEXT NOT NULL) — ISBN книги.
2. **authors** — хранит информацию об авторах:
    - `id` (INTEGER PRIMARY KEY) — уникальный идентификатор;
    - `book_id` (INTEGER, FOREIGN KEY) — связь с книгой;
    - `first_name` (TEXT NOT NULL) — имя автора;
    - `last_name` (TEXT NOT NULL) — фамилия автора;
    - `nationality` (TEXT NOT NULL) — национальность;
    - `birth_year` (INTEGER NOT NULL) — год рождения.

Управление состоянием

Для управления состоянием приложения используется Riverpod — декларативный фреймворк для управления состоянием:

- `booksProvider` — управление списком книг;
- `authorsProvider` — управление списком авторов;
- `fastDeleteProvider` — настройка быстрого удаления.

CRUD-операции

Приложение поддерживает полный набор CRUD-операций:

- **Create** — добавление новых книг и авторов через формы ввода;
- **Read** — отображение списков книг и авторов;
- **Update** — редактирование информации о книгах;
- **Delete** — удаление книг и авторов с подтверждением.

Пользовательский интерфейс

Интерфейс построен с использованием Material Design 3:

- **Навигация** — нижняя панель с тремя вкладками;
- **Формы** — валидированные формы для ввода данных;
- **Списки** — карточки с информацией и кнопками действий;
- **Диалоги** — модальные окна для редактирования и подтверждения удаления.

Скриншоты работы приложения

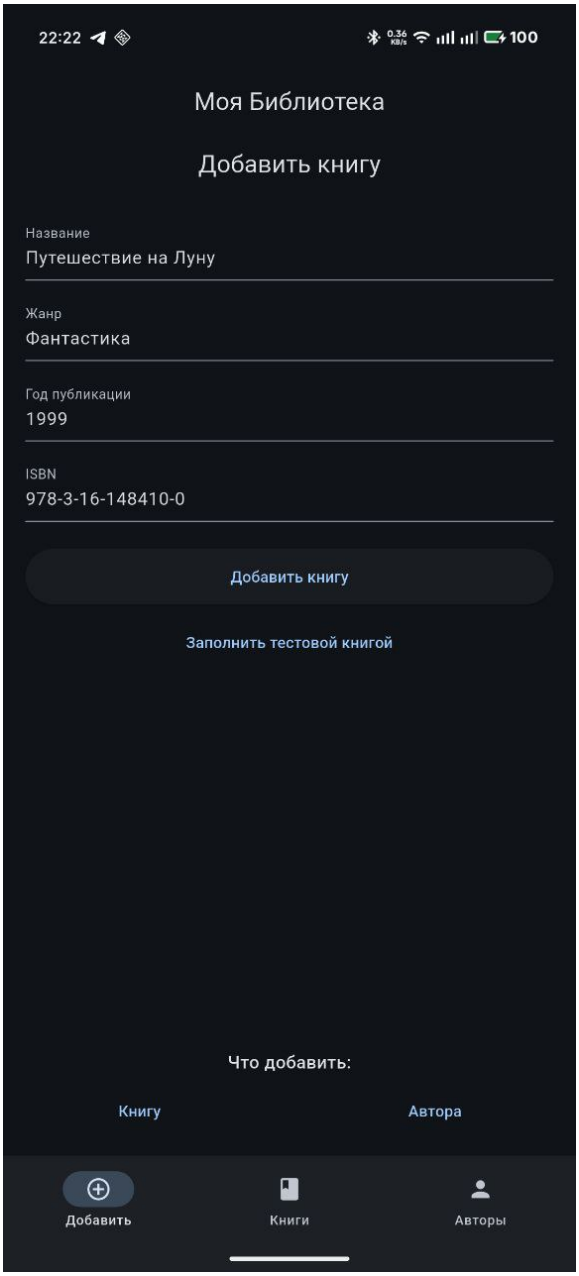

Рисунок 1 — Главный экран приложения

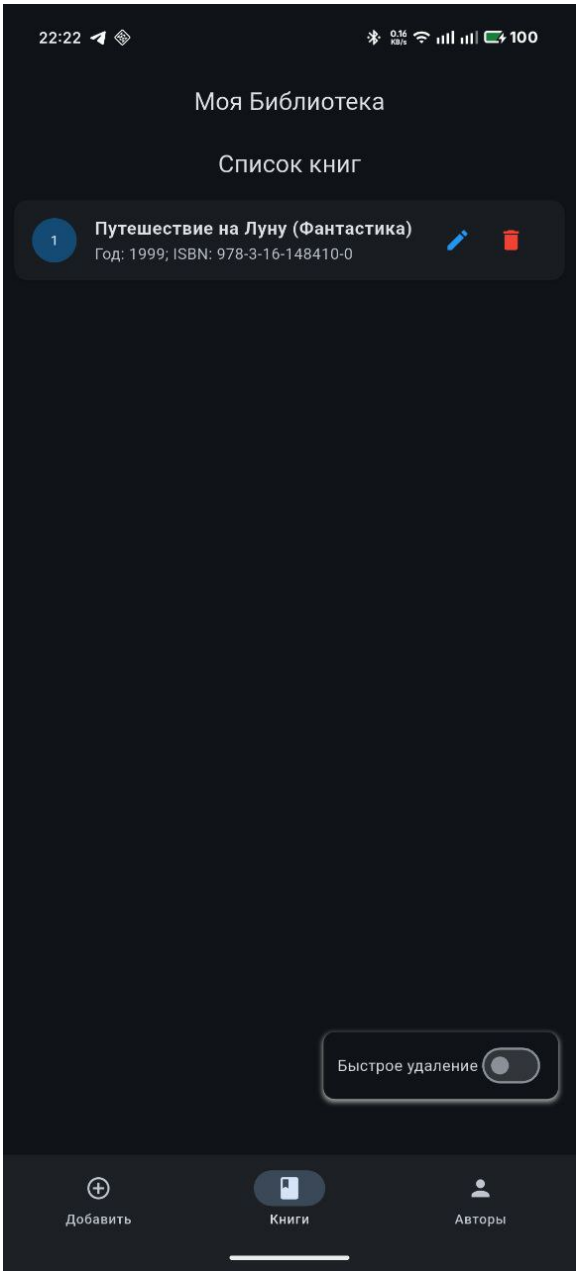

Рисунок 2 — Экран списка книг

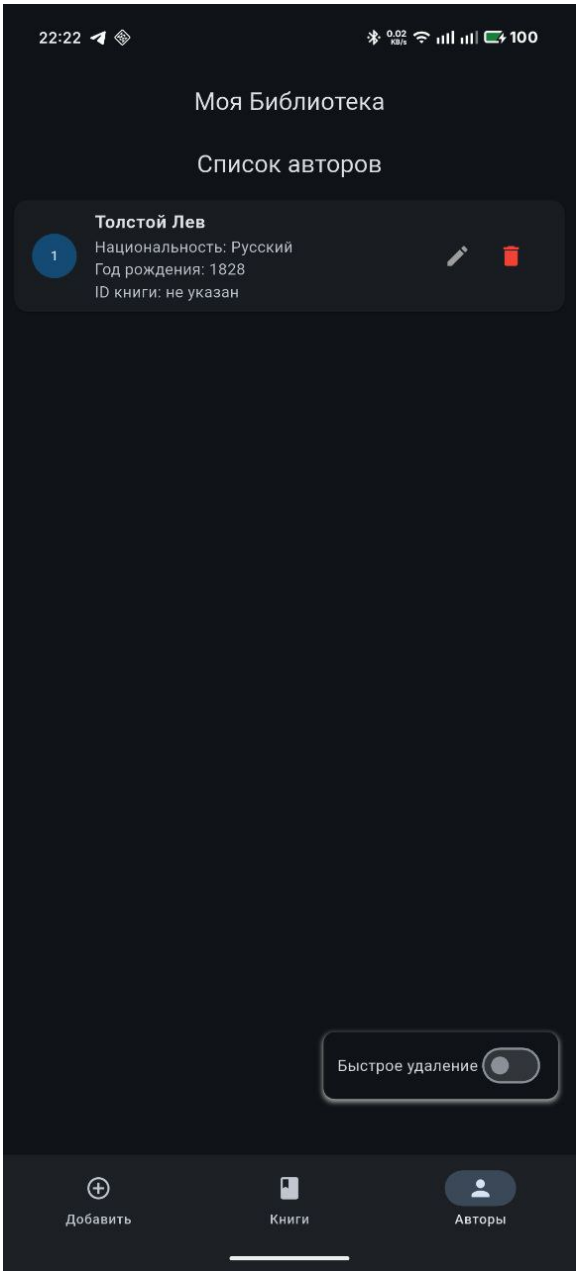

Рисунок 3 — Экран списка авторов

Контрольные вопросы

1. Что такое SQLite?

SQLite — это встраиваемая реляционная база данных, которая не требует отдельного серверного процесса. Она хранит всю базу данных в одном файле на диске и предоставляет полный набор SQL-функций.

2. Преимущества SQLite в мобильной разработке

- Не требует установки сервера;
- Хранит данные в одном файле;
- Поддерживает ACID-транзакции;
- Имеет небольшой размер библиотеки;
- Поддерживает большинство SQL-стандартов.

3. Что такое Riverpod?

Riverpod — это фреймворк для управления состоянием во Flutter, предоставляющий декларативный подход к управлению зависимостями и состоянием приложения. Он является развитием Provider с улучшенной типобезопасностью и API.

4. Основные компоненты Riverpod

- **Provider** — контейнер для хранения состояния или зависимостей;
- **ConsumerWidget/ConsumerStatefulWidget** — виджеты, которые могут читать провайдеры;
- **WidgetRef** — объект для чтения и изменения состояния провайдеров;
- **Notifier** — классы для управления изменяемым состоянием.

5. Что такое CRUD?

CRUD — акроним, обозначающий четыре основные операции с данными в базе данных:

- **Create** — создание новых записей;
- **Read** — чтение/получение данных;
- **Update** — обновление существующих записей;
- **Delete** — удаление записей.

6. Внешние ключи в SQLite

Внешний ключ (FOREIGN KEY) — это поле в таблице, которое ссылается на первичный ключ другой таблицы. Он обеспечивает целостность данных и позволяет устанавливать связи между таблицами. В SQLite внешние ключи включаются с помощью `PRAGMA foreign_keys = ON`.

7. Асинхронное программирование в Flutter

Flutter использует `async/await` для работы с асинхронными операциями. Ключевые слова:

- `async` — помечает функцию как асинхронную;
- `await` — ожидает завершения асинхронной операции;
- `Future` — тип для представления асинхронного результата.

8. Валидация форм во Flutter

Валидация форм осуществляется через callback `validator` в `TextFormField`. Функция валидации возвращает строку с ошибкой или `null`, если данные корректны. Для запуска валидации используется метод `FormState.validate()`.

Вывод

В ходе выполнения лабораторной работы №6 было разработано полнофункциональное мобильное приложение «Моя Библиотека» с использованием локальной базы данных SQLite. Реализован полный набор CRUD-операций для управления книгами и авторами. Изучены принципы работы с SQLite в Flutter, включая создание таблиц, внешние ключи и асинхронные операции. Освоен паттерн управления состоянием Riverpod для декларативного управления данными приложения. Приложение демонстрирует современные подходы к разработке мобильных приложений с персистентным хранением данных.

Приложение А. Исходный код main.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'screens/books_page.dart';
import 'screens/authors_page.dart';
import 'screens/home_page.dart';

void main() {
  runApp(const ProviderScope(child: MyApp()));
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    final lightColorScheme = ColorScheme.fromSeed(
      seedColor: const Color.fromARGB(255, 68, 138, 201),
      brightness: Brightness.light,
    );

    final darkColorScheme = ColorScheme.fromSeed(
      seedColor: const Color.fromARGB(255, 68, 138, 201),
      brightness: Brightness.dark,
    );

    return MaterialApp(
      themeMode: ThemeMode.system,
      title: 'Моя Библиотека',
      darkTheme: ThemeData(colorScheme: darkColorScheme, useMaterial3: true),
      theme: ThemeData(
        colorScheme: lightColorScheme,
        useMaterial3: true,
        appBarTheme: AppBarTheme(
          backgroundColor: lightColorScheme.primary,
          foregroundColor: lightColorScheme.onPrimary,
```

```
),
inputDecorationTheme: InputDecorationTheme(
  labelStyle: TextStyle(
    fontSize: 18,
    color: lightColorScheme.onSurfaceVariant,
  ),
  border: const OutlineInputBorder(
    borderRadius: BorderRadius.all(Radius.circular(8)),
  ),
  focusedBorder: OutlineInputBorder(
    borderSide: BorderSide(color: lightColorScheme.primary, width: 2),
    borderRadius: const BorderRadius.all(Radius.circular(8)),
  ),
  errorBorder: OutlineInputBorder(
    borderSide: BorderSide(color: lightColorScheme.error, width: 2),
    borderRadius: const BorderRadius.all(Radius.circular(8)),
  ),
  focusedErrorBorder: OutlineInputBorder(
    borderSide: BorderSide(color: lightColorScheme.error, width: 2),
    borderRadius: const BorderRadius.all(Radius.circular(8)),
  ),
  enabledBorder: OutlineInputBorder(
    borderSide: BorderSide(color: lightColorScheme.outline),
    borderRadius: const BorderRadius.all(Radius.circular(8)),
  ),
  floatingLabelBehavior: FloatingLabelBehavior.always,
),
cardTheme: CardThemeData(
  elevation: 4,
  margin: const EdgeInsets.symmetric(horizontal: 10, vertical: 5),
  color: lightColorScheme.surface,
  shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(12),
  ),
),
navigationBarTheme: NavigationBarThemeData(
  indicatorColor: lightColorScheme.primaryContainer,
```

```
          labelTextStyle: WidgetStateProperty.resolveWith((states) {
            if (states.contains(WidgetState.selected)) {
              return TextStyle(
                color: lightColorScheme.onSurface,
                fontWeight: FontWeight.bold,
              );
            }
            return TextStyle(color: lightColorScheme.onSurfaceVariant);
          }),
        ),
      ),
      home: const MainScreen(title: 'Моя Библиотека'),
    );
  }
}

class MainScreen extends StatefulWidget {
  const MainScreen({super.key, required this.title});

  final String title;

  @override
  State<MainScreen> createState() => _MainScreenState();
}

class _MainScreenState extends State<MainScreen> {
  final List<Widget> _screens = const [
    HomePage(),
    BooksListPage(),
    AuthorsPage(),
  ];
  int _currentIndex = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(widget.title), centerTitle: true),
```

```
      body: IndexedStack(index: _currentIndex, children: _screens),
      bottomNavigationBar: NavigationBar(
        selectedIndex: _currentIndex,
        onDestinationSelected: (index) {
          setState(() => _currentIndex = index);
        },
        destinations: const [
          NavigationDestination(
            icon: Icon(Icons.add_circle_outline),
            label: 'Добавить',
          ),
          NavigationDestination(icon: Icon(Icons.book), label: 'Книги'),
          NavigationDestination(icon: Icon(Icons.person), label: 'Авторы'),
        ],
      ),
    );
  }
}
```

Приложение А1. Исходный код database.dart

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

class DatabaseHelper {
  static final DatabaseHelper _instance = DatabaseHelper._internal();
  factory DatabaseHelper() => _instance;
  DatabaseHelper._internal();

  static Database? _database;

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDatabase();
    return _database!;
  }

  Future<Database> _initDatabase() async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, 'bookstore_database_v1.db'); // Изменено имя базы данных

    return await openDatabase(
      path,
      version: 1,
      onConfigure: (db) async {
        await db.execute('PRAGMA foreign_keys = ON');
      },
      onCreate: (db, version) async {
        // Таблица для книг
        await db.execute('''
          CREATE TABLE books (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT NOT NULL,
            publication_year INTEGER NOT NULL,
            genre TEXT NOT NULL,
            isbn TEXT NOT NULL
```

```
        );
      ''');

      // Таблица для авторов
      await db.execute('''
        CREATE TABLE authors (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          book_id INTEGER,
          first_name TEXT NOT NULL,
          last_name TEXT NOT NULL,
          nationality TEXT NOT NULL,
          birth_year INTEGER NOT NULL,
          FOREIGN KEY (book_id) REFERENCES books (id) ON DELETE SET NULL
        );
      ''');
    },
  );
}

Future<int> insert(String table, Map<String, dynamic> data) async {
  final db = await database;
  return await db.insert(
    table,
    data,
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}

Future<List<Map<String, dynamic>>> getAll(String table) async {
  final db = await database;
  return await db.query(table);
}

Future<Map<String, dynamic>?> getById(String table, int id) async {
  final db = await database;
  final res = await db.query(
    table,
```

```
      where: 'id = ?',
      whereArgs: [id],
      limit: 1,
    );
    return res.isNotEmpty ? res.first : null;
  }

  Future<int> update(String table, int id, Map<String, dynamic> data) async {
    final db = await database;
    return await db.update(table, data, where: 'id = ?', whereArgs: [id]);
  }

  Future<int> delete(String table, int id) async {
    final db = await database;
    return await db.delete(table, where: 'id = ?', whereArgs: [id]);
  }
}

// Модель данных для Книги
class Book {
  final int id;
  final String title;
  final int publicationYear;
  final String genre;
  final String isbn;

  Book({
    this.id = 0,
    required this.title,
    required this.publicationYear,
    required this.genre,
    required this.isbn,
  });

  Map<String, dynamic> toMap() {
    return {
      if (id != 0) 'id': id,
```

```
      'title': title,
      'publication_year': publicationYear,
      'genre': genre,
      'isbn': isbn,
    };
  }

  factory Book.fromMap(Map<String, dynamic> map) {
    return Book(
      id: map['id'],
      title: map['title'],
      publicationYear: map['publication_year'],
      genre: map['genre'],
      isbn: map['isbn'],
    );
  }
}

// Модель данных для Автора
class Author {
  final int id;
  final int? bookId; // Связь с книгой
  final String firstName;
  final String lastName;
  final String nationality;
  final int birthYear;

  Author({
    this.id = 0,
    this.bookId,
    required this.firstName,
    required this.lastName,
    required this.nationality,
    required this.birthYear,
  });

  Map<String, dynamic> toMap() {
```

```
    return {
      if (id != 0) 'id': id,
      'book_id': bookId,
      'first_name': firstName,
      'last_name': lastName,
      'nationality': nationality,
      'birth_year': birthYear,
    };
  }

  factory Author.fromMap(Map<String, dynamic> map) {
    return Author(
      id: map['id'],
      bookId: map['book_id'],
      firstName: map['first_name'],
      lastName: map['last_name'],
      nationality: map['nationality'],
      birthYear: map['birth_year'],
    );
  }
}
```

Приложение А2. Исходный код error_handle.dart

```
class DogValidator {
  String? validateBreed(String? value) {
    if (value == null || value.isEmpty) {
      return 'respect';
    }
    return null;
  }

  String? validateName(String? value) {
    if (value == null || value.isEmpty) {
      return 'Введите имя собаки';
    }
    return null;
  }

  String? validateAge(String? value) {
    if (value == null || value.isEmpty) {
      return 'Собака существует?';
    }

    final age = int.tryParse(value);
    if (age == null) {
      return 'Введите целое число';
    }

    if (age > 115) {
      return 'Введите действительный возраст';
    }

    return null;
  }
}

class OwnerValidator {
  String? validateFirstName(String? value) {
```

```
  if (value == null || value.trim().isEmpty) {
    return "Введите имя";
  }
  if (value.length < 2) {
    return "Имя слишком короткое";
  }
  return null;
}

String? validateMiddleName(String? value) {
  if (value == null || value.trim().isEmpty) {
    return "Введите фамилию";
  }
  return null;
}

String? validateLastName(String? value) {
  if (value == null || value.trim().isEmpty) {
    return "Введите отчество";
  }
  return null;
}

String? validateAddress(String? value) {
  if (value == null || value.trim().isEmpty) {
    return "Введите адрес";
  }
  return null;
}

String? validatePhone(String? value) {
  if (value == null || value.trim().isEmpty) {
    return "Введите номер телефона";
  }

  final phoneExp = RegExp(r'^[0-9+\-\s]{6,18}$');
  if (!phoneExp.hasMatch(value)) {
```

```
      return "Некорректный номер телефона";
    }
    return null;
  }
}
```

Приложение А3. Исходный код providers.dart

```dart
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'database.dart'; // Обновленный путь
import 'package:riverpod/legacy.dart';

// Notifier для управления состоянием списка книг
class BookNotifier extends AsyncNotifier<List<Book>> {
  @override
  Future<List<Book>> build() async {
    final list = await DatabaseHelper().getAll("books");
    return list.map((e) => Book.fromMap(e)).toList();
  }

  Future<void> addBook(Book book) async {
    state = const AsyncValue.loading();
    state = await AsyncValue.guard(() async {
      await DatabaseHelper().insert("books", book.toMap());
      return await _fetchBooks(); // Обновление списка
    });
  }

  Future<void> deleteBook(int id) async {
    state = const AsyncValue.loading();
    state = await AsyncValue.guard(() async {
      await DatabaseHelper().delete("books", id);
      return await _fetchBooks(); // Обновление списка
    });
  }

  Future<void> updateBook(Book book) async {
    state = const AsyncValue.loading();
    state = await AsyncValue.guard(() async {
      await DatabaseHelper().update("books", book.id, book.toMap());
      return await _fetchBooks(); // Обновление списка
    });
  }
```

```
  Future<List<Book>> _fetchBooks() async {
    final list = await DatabaseHelper().getAll("books");
    return list.map((e) => Book.fromMap(e)).toList();
  }
}

final booksProvider = AsyncNotifierProvider<BookNotifier, List<Book>>(
  () => BookNotifier(),
);

// Notifier для управления состоянием списка авторов
class AuthorNotifier extends AsyncNotifier<List<Author>> {
  @override
  Future<List<Author>> build() async {
    final list = await DatabaseHelper().getAll("authors");
    return list.map((e) => Author.fromMap(e)).toList();
  }

  Future<void> addAuthor(Author author) async {
    state = const AsyncValue.loading();
    state = await AsyncValue.guard(() async {
      await DatabaseHelper().insert("authors", author.toMap());
      return await _fetchAuthors(); // Обновление списка
    });
  }

  Future<void> updateAuthor(Author author) async {
    state = const AsyncValue.loading();
    state = await AsyncValue.guard(() async {
      await DatabaseHelper().update("authors", author.id, author.toMap());
      return await _fetchAuthors(); // Обновление списка
    });
  }

  Future<void> deleteAuthor(int id) async {
    state = const AsyncValue.loading();
```

```
    state = await AsyncValue.guard(() async {
      await DatabaseHelper().delete("authors", id);
      return await _fetchAuthors(); // Обновление списка
    });
  }

  Future<List<Author>> _fetchAuthors() async {
    final list = await DatabaseHelper().getAll("authors");
    return list.map((e) => Author.fromMap(e)).toList();
  }
}

final authorsProvider = AsyncNotifierProvider<AuthorNotifier, List<Author>>(
  () => AuthorNotifier(),
);

// Провайдер для быстрого удаления
final fastDeleteProvider = StateProvider<bool>((ref) => false);
```

Приложение А4. Исходный код screens/authors_page.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../providers.dart'; // Обновленный путь
import '../utils/confirm_delete_popup.dart'; // Обновленный путь

class AuthorsPage extends ConsumerWidget {
  const AuthorsPage({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final isFastDelete = ref.watch(
      fastDeleteProvider,
    ); // Изменено имя провайдера
    final authorsAsyncValue = ref.watch(
      authorsProvider,
    ); // Изменено имя провайдера

    return Scaffold(
      appBar: AppBar(
        centerTitle: true,
        title: const Text('Список авторов'),
      ), // Изменен заголовок
      body: authorsAsyncValue.when(
        loading: () => const Center(child: CircularProgressIndicator()),
        data: (authors) {
          if (authors.isEmpty) {
            return const Center(child: Text('Авторов нет!')); // Изменен текст
          }
          return ListView.builder(
            itemCount: authors.length,
            itemBuilder: (context, index) {
              final author = authors[index];
              return Card(
                margin: const EdgeInsets.symmetric(horizontal: 10, vertical: 4),
                child: ListTile(
```

```
leading: CircleAvatar(
  // Небольшое изменение дизайна - круглая иконка с ID
  child: Text(
    '${author.id}',
    style: const TextStyle(fontSize: 12),
  ),
),
title: Text(
  '${author.lastName} ${author.firstName}', // Обновлен вывод информации
  style: const TextStyle(fontWeight: FontWeight.bold),
),
subtitle: Text(
  'Национальность: ${author.nationality}\nГод рождения: ${author.birthYea
),
trailing: Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    // Кнопка редактирования автора (реализация не предусмотрена в ориги
    IconButton(
      onPressed: () {
        // TODO: Реализовать диалог редактирования автора
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text(
              'Редактирование автора пока не реализовано',
            ),
          ),
        );
      },
      icon: const Icon(
        Icons.edit,
        color: Colors.grey,
      ), // Серая иконка для неактивной функции
    ),
    IconButton(
      onPressed: () async {
        showDeleteConfirmation(
```

```
                        context: context,
                        isFastDelete: isFastDelete,
                        title: 'Удалить автора?', // Изменен заголовок
                        itemName:
                            '${author.firstName} ${author.lastName}', // Имя автора
                        onDelete: () async {
                          await ref
                              .read(authorsProvider.notifier)
                              .deleteAuthor(
                                author.id,
                              ); // Изменено имя провайдера
                        },
                      );
                    },
                    icon: const Icon(Icons.delete, color: Colors.red),
                  ),
                ],
              ),
            ),
          );
        },
      );
    },
    error: (err, stack) => Center(child: Text('Ошибка: $err')),
  ),
  floatingActionButton: Padding(
    padding: const EdgeInsets.only(
      bottom: 50.0,
    ), // Отступ от нижней навигации
    child: Column(
      mainAxisSize: MainAxisSize.min,
      crossAxisAlignment: CrossAxisAlignment.end,
      children: [
        Container(
          padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 6),
          decoration: BoxDecoration(
            color: Theme.of(context).cardColor,
```

```
                borderRadius: BorderRadius.circular(10),
                boxShadow: [
                  BoxShadow(
                    color: Colors.grey.withValues(alpha: 51),
                    spreadRadius: 1,
                    blurRadius: 3,
                    offset: const Offset(0, 2),
                  ),
                ],
              ),
              child: Row(
                mainAxisSize: MainAxisSize.min,
                children: [
                  const Text('Быстрое удаление'),
                  Switch(
                    value: isFastDelete,
                    onChanged: (newValue) {
                      ref.read(fastDeleteProvider.notifier).state =
                          newValue; // Изменено имя провайдера
                    },
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
      floatingActionButtonLocation:
          FloatingActionButtonLocation.endDocked, // Расположение FAB
    );
  }
}
```

Приложение А5. Исходный код screens/books_page.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../providers.dart'; // Обновленный путь
import '../utils/confirm_delete_popup.dart'; // Обновленный путь
import '../widgets/book_redacting_popup.dart'; // Изменено

class BooksListPage extends ConsumerWidget {
  const BooksListPage({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final isFastDelete = ref.watch(
      fastDeleteProvider,
    ); // Изменено имя провайдера
    final booksAsyncValue = ref.watch(booksProvider); // Изменено имя провайдера

    return Scaffold(
      appBar: AppBar(
        centerTitle: true,
        title: const Text(
          'Список книг',
          textAlign: TextAlign.center,
        ), // Изменен заголовок
      ),
      body: booksAsyncValue.when(
        loading: CircularProgressIndicator.adaptive,
        data: (books) {
          if (books.isEmpty) {
            return const Center(child: Text('Книг нет!')); // Изменен текст
          }
          return ListView.builder(
            itemCount: books.length,
            itemBuilder: (context, index) {
              final book = books[index];
              return Card(
```

```
margin: const EdgeInsets.symmetric(horizontal: 10, vertical: 4),
child: ListTile(
  leading: CircleAvatar(
    // Небольшое изменение дизайна - круглая иконка с ID
    child: Text(
      '${book.id}',
      style: const TextStyle(fontSize: 12),
    ),
  ),
  title: Text(
    '${book.title} (${book.genre})', // Обновлен вывод информации о книге
    style: const TextStyle(fontWeight: FontWeight.bold),
  ),
  subtitle: Text(
    'Год: ${book.publicationYear}; ISBN: ${book.isbn}', // Обновлен вывод
  ),
  trailing: Row(
    mainAxisSize: MainAxisSize.min,
    children: [
      IconButton(
        onPressed: () => showBookEditingDialog(
          context,
          book,
          ref,
        ), // Изменено название функции
        icon: const Icon(Icons.edit, color: Colors.blue),
      ),
      IconButton(
        onPressed: () => showDeleteConfirmation(
          context: context,
          itemName: book.title, // Используем название книги
          title: 'Удалить книгу?', // Изменен заголовок
          isFastDelete: isFastDelete,
          onDelete: () async {
            await ref
                .read(booksProvider.notifier)
                .deleteBook(book.id); // Изменено имя провайдера
```

```
                    },
                  ),
                  icon: const Icon(Icons.delete, color: Colors.red),
                ),
              ],
            ),
          ),
        );
      },
    );
  },
  error: (err, stack) => Center(child: Text('Ошибка: $err')),
),
floatingActionButton: Padding(
  padding: const EdgeInsets.only(
    bottom: 50.0,
  ), // Отступ от нижней навигации
  child: Column(
    mainAxisSize: MainAxisSize.min,
    crossAxisAlignment: CrossAxisAlignment.end,
    children: [
      Container(
        padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 6),
        decoration: BoxDecoration(
          color: Theme.of(context).cardColor,
          borderRadius: BorderRadius.circular(10),
          boxShadow: [
            BoxShadow(
              color: Colors.grey.withValues(alpha: 51),
              spreadRadius: 1,
              blurRadius: 3,
              offset: const Offset(0, 2),
            ),
          ],
        ),
        child: Row(
          mainAxisSize: MainAxisSize.min,
```

```
                children: [
                  const Text('Быстрое удаление'),
                  Switch(
                    value: isFastDelete,
                    onChanged: (newValue) {
                      ref.read(fastDeleteProvider.notifier).state =
                          newValue; // Изменено имя провайдера
                    },
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
      floatingActionButtonLocation:
          FloatingActionButtonLocation.endDocked, // Расположение FAB
    );
  }
}
```

Приложение А6. Исходный код screens/home_page.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../widgets/book_adding_window.dart'; // Изменено
import '../widgets/author_adding_window.dart'; // Изменено

class HomePage extends ConsumerStatefulWidget {
  const HomePage({super.key});

  @override
  ConsumerState<HomePage> createState() => _HomePageState();
}

class _HomePageState extends ConsumerState<HomePage> {
  String? _appBarTitle = 'Добавить книгу'; // Изменено
  int _showWindow = 0; // Изменено имя переменной

  Widget _buildWindow() {
    switch (_showWindow) { // Изменено имя переменной
      case 0:
        return const BookAddingWindow(); // Изменено
      case 1:
        return const AuthorAddingWindow(); // Изменено
      default:
        return const SizedBox();
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(centerTitle: true, title: Text('$_appBarTitle')),
      body: Column(
        children: [
          Expanded(child: _buildWindow()), // Wrap with Expanded
          Container(
```

```
  padding: const EdgeInsets.only(bottom: 16, top: 16), // Добавлен top padding
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
      const Text(
        'Что добавить:', // Изменен текст
        style: TextStyle(fontSize: 16, fontWeight: FontWeight.w500),
      ),
      const SizedBox(height: 8),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          TextButton(
            onPressed: () {
              setState(() {
                _showWindow = 0;
                _appBarTitle = 'Добавить книгу'; // Изменено
              });
            },
            child: const Text('Книгу'), // Изменено
          ),
          const SizedBox(width: 16),
          TextButton(
            onPressed: () {
              setState(() {
                _showWindow = 1;
                _appBarTitle = 'Добавить автора'; // Изменено
              });
            },
            child: const Text('Автора'), // Изменено
          ),
        ],
      ),
    ],
  ),
),
```

```
        ],
      ),
    );
  }
}
```

Приложение А7. Исходный код widgets/author_adding_window.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../database.dart'; // Обновленный путь
import '../providers.dart'; // Обновленный путь
import '../validators/author_validator.dart'; // Обновленный путь

class AuthorAddingWindow extends ConsumerStatefulWidget {
  const AuthorAddingWindow({super.key});

  @override
  ConsumerState<AuthorAddingWindow> createState() => _AuthorAddingWindowState();
}

class _AuthorAddingWindowState extends ConsumerState<AuthorAddingWindow> {
  final _formKey = GlobalKey<FormState>();

  final _firstNameController = TextEditingController();
  final _lastNameController = TextEditingController();
  final _nationalityController = TextEditingController();
  final _birthYearController = TextEditingController();

  Book? selectedBook; // Изменено с Dog на Book
  final AuthorValidator _validator =
      AuthorValidator(); // Изменено имя валидатора

  @override
  void dispose() {
    _firstNameController.dispose();
    _lastNameController.dispose();
    _nationalityController.dispose();
    _birthYearController.dispose();
    super.dispose();
  }

  void _submitForm() async {
```

```
  if (_formKey.currentState!.validate()) {
    try {
      final author = Author(
        bookId: selectedBook?.id, // Изменено с dogId на bookId
        firstName: _firstNameController.text,
        lastName: _lastNameController.text,
        nationality: _nationalityController.text,
        birthYear: int.parse(_birthYearController.text),
      );

      await ref
          .read(authorsProvider.notifier)
          .addAuthor(author); // Изменено имя провайдера

      _formKey.currentState!.reset();
      _firstNameController.clear();
      _lastNameController.clear();
      _nationalityController.clear();
      _birthYearController.clear();
      setState(() {
        selectedBook = null; // Изменено с selectedDog на selectedBook
      });

      if (mounted) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text("Автор добавлен!")),
        ); // Изменен текст
      }
    } catch (e) {
      if (mounted) {
        ScaffoldMessenger.of(
          context,
        ).showSnackBar(SnackBar(content: Text("Ошибка: $e")));
      }
    }
  }
}
```

```
@override
Widget build(BuildContext context) {
  final booksList = ref.watch(
    booksProvider,
  ); // Изменено с dogsProvider на booksProvider

  return SingleChildScrollView(
    // Обертка в SingleChildScrollView
    padding: const EdgeInsets.fromLTRB(20, 20, 20, 10),
    child: Form(
      key: _formKey,
      child: Column(
        children: [
          TextFormField(
            controller: _firstNameController,
            decoration: const InputDecoration(labelText: 'Имя'),
            validator: _validator.validateFirstName,
          ),
          const SizedBox(height: 16),
          TextFormField(
            controller: _lastNameController,
            decoration: const InputDecoration(labelText: 'Фамилия'),
            validator: _validator.validateLastName,
          ),
          const SizedBox(height: 16),
          TextFormField(
            controller: _nationalityController,
            decoration: const InputDecoration(labelText: 'Национальность'),
            validator: _validator.validateNationality,
          ),
          const SizedBox(height: 16),

          booksList.when(
            loading: () => const CircularProgressIndicator(),
            error: (err, st) =>
                Text('Ошибка загрузки книг: $err'), // Обновлен текст ошибки
```

```
  data: (books) {
    final List<DropdownMenuItem<Book?>> menuItems = [
      const DropdownMenuItem<Book?>(
        value: null,
        child: Text('Нет (без книги)'), // Изменен текст
      ),
    ];
    menuItems.addAll(
      books.map((book) {
        // Изменено с dog на book
        return DropdownMenuItem<Book?>(
          // Изменено с Dog на Book
          value: book,
          child: Text(
            '${book.title} (${book.genre})',
          ), // Обновлен вывод информации о книге
        );
      }),
    );
    return DropdownButtonFormField<Book?>(
      // Изменено с Dog на Book
      decoration: const InputDecoration(
        labelText: 'Книга автора', // Изменен текст
      ),
      initialValue: selectedBook, // Изменено на initialValue
      items: menuItems,
      onChanged: (value) => setState(
        () => selectedBook = value,
      ), // Изменено с selectedDog на selectedBook
    );
  },
),
const SizedBox(height: 16),
TextFormField(
  controller: _birthYearController,
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(labelText: 'Год рождения'),
```

```
            validator: _validator.validateBirthYear,
          ),
          const SizedBox(height: 24),
          ElevatedButton(
            onPressed: _submitForm,
            style: ElevatedButton.styleFrom(
              minimumSize: const Size.fromHeight(50),
            ),
            child: const Text("Добавить автора"), // Изменен текст
          ),
          const SizedBox(height: 10),
          TextButton(
            // Изменено на TextButton
            onPressed: () {
              _birthYearController.text = '1828';
              _nationalityController.text = 'Русский';
              _lastNameController.text = 'Толстой';
              _firstNameController.text = 'Лев';
            },
            child: const Text('Заполнить тестовым автором'), // Изменен текст
          ),
        ],
      ),
    ),
  );
 }
}
```

Приложение А8. Исходный код widgets/book_adding_window.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../database.dart'; // Обновленный путь
import '../providers.dart'; // Обновленный путь
import '../validators/book_validator.dart'; // Обновленный путь

class BookAddingWindow extends ConsumerStatefulWidget {
  const BookAddingWindow({super.key});

  @override
  ConsumerState<BookAddingWindow> createState() => _BookAddingWindowState();
}

class _BookAddingWindowState extends ConsumerState<BookAddingWindow> {
  final _formKey = GlobalKey<FormState>();

  final TextEditingController _titleController = TextEditingController();
  final TextEditingController _genreController = TextEditingController();
  final TextEditingController _publicationYearController = TextEditingController();
  final TextEditingController _isbnController = TextEditingController();

  final BookValidator _validator = BookValidator();

  @override
  void dispose() {
    _titleController.dispose();
    _genreController.dispose();
    _publicationYearController.dispose();
    _isbnController.dispose();
    super.dispose();
  }

  void _submit() async {
    if (_formKey.currentState!.validate()) {
      try {
```

```
      final book = Book(
        title: _titleController.text,
        publicationYear: int.parse(_publicationYearController.text),
        genre: _genreController.text,
        isbn: _isbnController.text,
      );

      await ref.read(booksProvider.notifier).addBook(book); // Изменено имя провайдера

      _formKey.currentState!.reset();
      _titleController.clear();
      _genreController.clear();
      _publicationYearController.clear();
      _isbnController.clear();

      if (mounted) {
        ScaffoldMessenger.of(
          context,
        ).showSnackBar(const SnackBar(content: Text('Книга добавлена!'))); // Изменен те
      }
    } catch (e) {
      if (mounted) {
        ScaffoldMessenger.of(
          context,
        ).showSnackBar(SnackBar(content: Text('Ошибка: $e')));
      }
    }
  }
}

@override
Widget build(BuildContext context) {
  return SingleChildScrollView(
    padding: const EdgeInsets.all(20.0),
    child: Form(
      key: _formKey,
      child: Column(
```

```
children: [
  TextFormField(
    controller: _titleController,
    decoration: const InputDecoration(
      labelText: 'Название',
      hintText: 'Введите название книги',
    ),
    validator: _validator.validateTitle,
  ),
  const SizedBox(height: 16), // Добавлен SizedBox для отступа
  TextFormField(
    controller: _genreController,
    decoration: const InputDecoration(
      labelText: 'Жанр',
      hintText: 'Введите жанр книги',
    ),
    validator: _validator.validateGenre,
  ),
  const SizedBox(height: 16),
  TextFormField(
    controller: _publicationYearController,
    keyboardType: TextInputType.number,
    decoration: const InputDecoration(
      labelText: 'Год публикации',
      hintText: 'Введите год публикации',
    ),
    validator: _validator.validatePublicationYear,
  ),
  const SizedBox(height: 16),
  TextFormField(
    controller: _isbnController,
    decoration: const InputDecoration(
      labelText: 'ISBN',
      hintText: 'Введите ISBN',
    ),
    validator: _validator.validateISBN,
  ),
```

```
            const SizedBox(height: 24), // Увеличен отступ
            ElevatedButton(
              onPressed: _submit,
              style: ElevatedButton.styleFrom(
                minimumSize: const Size.fromHeight(50), // Увеличен размер кнопки
              ),
              child: const Text('Добавить книгу'), // Изменен текст
            ),
            const SizedBox(height: 10),
            TextButton( // Изменено на TextButton для тестовой кнопки
              onPressed: () {
                _isbnController.text = '978-3-16-148410-0';
                _genreController.text = 'Фантастика';
                _publicationYearController.text = '1999';
                _titleController.text = 'Путешествие на Луну';
              },
              child: const Text('Заполнить тестовой книгой'), // Изменен текст
            ),
          ],
        ),
      ),
    );
  }
}
```

Приложение А9. Исходный код widgets/book_redacting_popup.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../database.dart'; // Обновленный путь
import '../providers.dart'; // Обновленный путь
import '../validators/book_validator.dart'; // Обновленный путь
import 'dart:async';

void showBookEditingDialog(BuildContext context, Book book, WidgetRef ref) {
  // Изменено название функции и тип данных
  final titleCtrl = TextEditingController(
    text: book.title,
  ); // Изменено с breed на title
  final genreCtrl = TextEditingController(
    text: book.genre,
  ); // Изменено с age на genre
  final publicationYearCtrl = TextEditingController(
    text: '${book.publicationYear}',
  ); // Изменено с name на publicationYear
  final isbnCtrl = TextEditingController(
    text: book.isbn,
  ); // Изменено с collar на isbn

  final focusNode = FocusNode();
  String? titleError; // Изменено
  String? genreError; // Изменено
  String? publicationYearError; // Изменено
  String? isbnError; // Добавлено

  final BookValidator validator = BookValidator(); // Изменено имя валидатора

  void closeDialog() {
    if (context.mounted) {
      Navigator.pop(context);
    }
  }
```

```
showDialog(
  context: context,
  builder: (context) => StatefulBuilder(
    builder: (context, setState) => Dialog(
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text(
              'Редактирование книги', // Изменен заголовок
              style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
            const SizedBox(height: 16),

            // Поле "Название"
            TextField(
              controller: titleCtrl,
              onChanged: (value) {
                setState(() {
                  titleError = validator.validateTitle(value);
                });
              },
              focusNode: focusNode,
              decoration: InputDecoration(
                errorText: titleError,
                labelText: 'Название',
                hintText: 'Введите название книги',
                border: OutlineInputBorder(),
              ),
            ),

            const SizedBox(height: 12),

            // Поле "Жанр"
```

```
TextField(
  controller: genreCtrl,
  onChanged: (value) {
    setState(() {
      genreError = validator.validateGenre(value);
    });
  },
  decoration: InputDecoration(
    errorText: genreError,
    labelText: 'Жанр',
    hintText: 'Введите жанр',
    border: OutlineInputBorder(),
  ),
),

const SizedBox(height: 12),

// Поле "Год публикации"
TextField(
  controller: publicationYearCtrl,
  keyboardType: TextInputType.number,
  onChanged: (value) {
    setState(() {
      publicationYearError = validator.validatePublicationYear(
        value,
      );
    });
  },
  decoration: InputDecoration(
    errorText: publicationYearError,
    labelText: 'Год публикации',
    hintText: 'Введите год публикации',
    border: OutlineInputBorder(),
  ),
),

const SizedBox(height: 12),
```

```
// Поле "ISBN"
TextField(
  controller: isbnCtrl,
  onChanged: (value) {
    setState(() {
      isbnError = validator.validateISBN(value);
    });
  },
  decoration: InputDecoration(
    errorText: isbnError,
    labelText: 'ISBN',
    hintText: 'Введите ISBN',
    border: OutlineInputBorder(),
  ),
),

const SizedBox(height: 24),

Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    ElevatedButton(
      onPressed: () => Navigator.pop(context),
      child: const Text("Отмена"),
    ),
    ElevatedButton(
      onPressed: () {
        final localTitleError = validator.validateTitle(
          titleCtrl.text,
        );
        final localGenreError = validator.validateGenre(
          genreCtrl.text,
        );
        final localPublicationYearError = validator
            .validatePublicationYear(publicationYearCtrl.text);
        final localIsbnError = validator.validateISBN(
```

```
  isbnCtrl.text,
);

if (localTitleError != null ||
    localGenreError != null ||
    localPublicationYearError != null ||
    localIsbnError != null) {
  setState(() {
    titleError = localTitleError;
    genreError = localGenreError;
    publicationYearError = localPublicationYearError;
    isbnError = localIsbnError;
  });
  return;
}

final updatedBook = Book(
  // Изменено с Dog на Book
  id: book.id,
  title: titleCtrl.text, // Изменено
  publicationYear:
      int.tryParse(publicationYearCtrl.text) ??
      0, // Изменено
  genre: genreCtrl.text, // Изменено
  isbn: isbnCtrl.text, // Изменено
);

ref
    .read(booksProvider.notifier)
    .updateBook(
      updatedBook,
    ); // Изменено имя провайдера и метод

closeDialog();
ScaffoldMessenger.of(context).showSnackBar(
  const SnackBar(
    content: Text('Книга обновлена.'),
```

```
                        ), // Изменен текст
                      );
                    },
                    child: const Text('Сохранить'), // Изменен текст
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    ),
  );

  Future.delayed(const Duration(milliseconds: 200), () {
    if (context.mounted) {
      FocusScope.of(context).requestFocus(focusNode);
    }
  });
}
```

Приложение А10. Исходный код utils/confirm_delete_popup.dart

```dart
import 'package:flutter/material.dart';

void showDeleteConfirmation({
  required BuildContext context,
  required bool isFastDelete,
  required String title,
  required String itemName,
  required Future<void> Function() onDelete,
}) {
  Future<void> performDelete() async {
    try {
      await onDelete();

      if (context.mounted) {
        if (!isFastDelete) {
          Navigator.of(context).pop();
        }

        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Text('$itemName успешно удален(а)'), // Обновлен текст
            action: SnackBarAction(label: 'OK', onPressed: () {}),
          ),
        );
      }
    } catch (e) {
      if (context.mounted) {
        if (!isFastDelete) {
          Navigator.of(context).pop();
        }

        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Text('Ошибка удаления: $e'),
            backgroundColor: Colors.red,
```

```
          ),
        );
      }
    }
  }

  if (isFastDelete) {
    performDelete();
  } else {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: Text(title, textAlign: TextAlign.center),
        content: Text(
          'Вы действительно хотите удалить "$itemName"? Это действие нельзя отменить.',
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context),
            child: const Text("Отмена"),
          ),
          TextButton(
            onPressed: () {
              performDelete();
              // Закрыть диалог после вызова performDelete, чтобы избежать двойного удален
              // если performDelete делает что-то асинхронное и потом обновляет UI
              // Здесь это уже учтено: performDelete может закрыть, но только если !isFast
              if (context.mounted) Navigator.pop(context);
            },
            child: const Text("Удалить", style: TextStyle(color: Colors.red)),
          ),
        ],
      ),
    );
  }
}
```

Приложение А11. Исходный код validators/author_validator.dart

```dart
class AuthorValidator {
  String? validateFirstName(String? value) {
    if (value == null || value.trim().isEmpty) {
      return "Введите имя автора";
    }
    if (value.length < 2) {
      return "Имя слишком короткое";
    }
    return null;
  }

  String? validateLastName(String? value) {
    if (value == null || value.trim().isEmpty) {
      return "Введите фамилию автора";
    }
    return null;
  }

  String? validateNationality(String? value) {
    if (value == null || value.trim().isEmpty) {
      return "Введите национальность";
    }
    return null;
  }

  String? validateBirthYear(String? value) {
    if (value == null || value.isEmpty) {
      return 'Введите год рождения';
    }

    final year = int.tryParse(value);
    if (year == null) {
      return 'Введите целое число';
    }
```

```
    if (year < 0 || year > DateTime.now().year) {
      return 'Введите действительный год рождения';
    }

    return null;
  }
}
```

Приложение А12. Исходный код validators/book_validator.dart

```
class BookValidator {
  String? validateTitle(String? value) {
    if (value == null || value.isEmpty) {
      return 'Введите название книги';
    }
    return null;
  }

  String? validateGenre(String? value) {
    if (value == null || value.isEmpty) {
      return 'Введите жанр';
    }
    return null;
  }

  String? validatePublicationYear(String? value) {
    if (value == null || value.isEmpty) {
      return 'Введите год публикации';
    }

    final year = int.tryParse(value);
    if (year == null) {
      return 'Введите целое число';
    }

    if (year < 0 || year > DateTime.now().year) {
      return 'Введите действительный год публикации';
    }

    return null;
  }

  String? validateISBN(String? value) {
    if (value == null || value.isEmpty) {
      return 'Введите ISBN';
```

```
    }
    // Простая валидация ISBN (можно усложнить)
    final isbnExp = RegExp(r'^(?:ISBN(?::?)(?=\s*\d{3}[\s-]?\d{10}$))?((?:97[89][\s-]?)?\d
    if (!isbnExp.hasMatch(value)) {
      return 'Некорректный ISBN';
    }
    return null;
  }
}
```