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

Отчет по лабораторной работе №6
по дисциплине
«Технологии программирования»

Выполнил студент гр. ИВТб-2301-05-00 _____/Макаров С.А./
Проверил преподаватель _____/Пащенко Д.Э./

Киров 2025

Цель

Цель: Освоить практические навыки интеграции локального хранилища данных в мобильное приложение, научиться проектировать и реализовывать полный цикл работы с данными (создание, чтение, обновление, удаление) с использованием реляционной базы данных SQLite.

Задание

Разработайте многооконное мобильное приложение, которое хранит информацию в локальной базе данных SQLite и позволяет изменять её. Тему приложения выберите самостоятельно и согласуйте с преподавателем.

Решение

В ходе выполнения лабораторной работы многооконное мобильное приложение, позволяющее усправлять списком учебных работ. Для добавления новой учебной работы необходимо ввести название, ее тип, дата сдачи и описание. Для реализации данного функционала была разработана база данных включающая в себя сущности «Тип работы» и «Работа».

Сущность работы содержит следующие поля: «Название» – строковой тип данных, «Идентификатор типа работы» – идентификатор типа работы из сущности «Тип работы», «Дата сдачи» – данные в формате даты, «Описание» – строковой тип данных.

Сущность «Тип работы» содержит поле «Название» – строковый тип данных.

Экранная форма главного окна со списком учбеных работ представлена на рисунке 1.

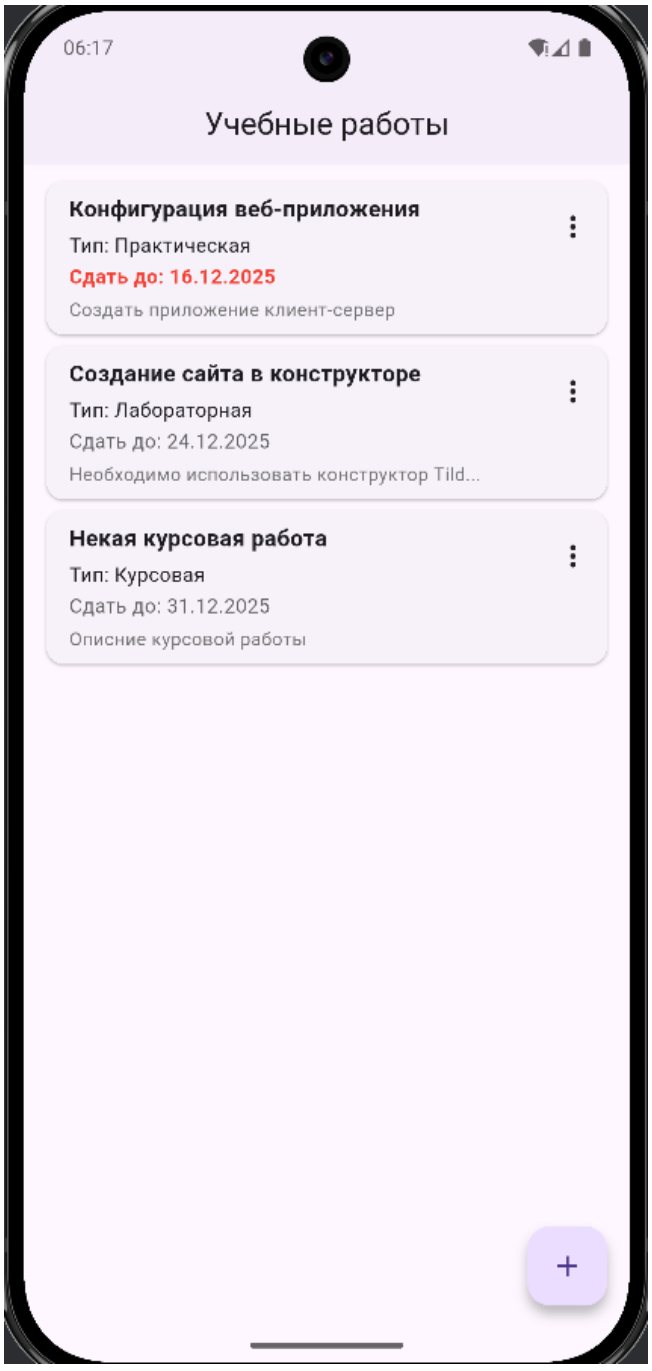

Рисунок 1 – Экранная форма главного экрана

Экранная форма добавления и редактирования учебной работы представлена на рисунке 2.

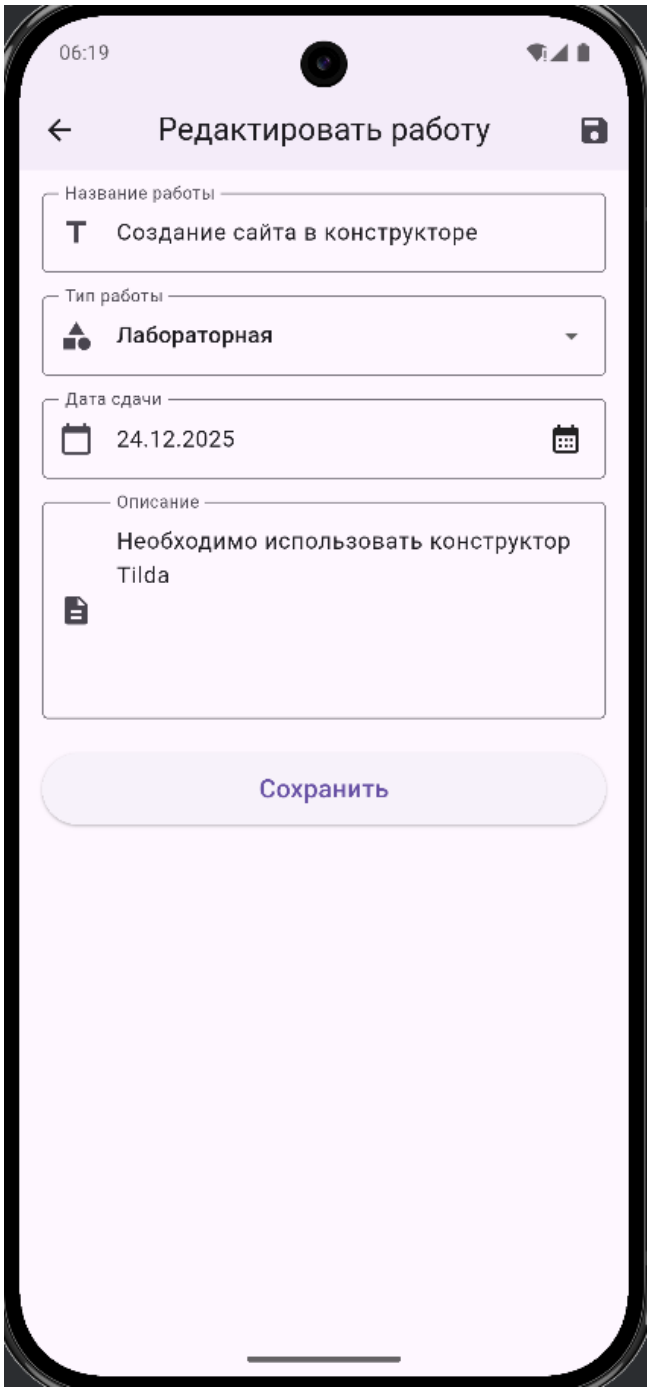

Рисунок 2 – Экранная форма добавления и редактирования

Экранная форма подтверждения удаления учебной работы представлена на рисунке 3.

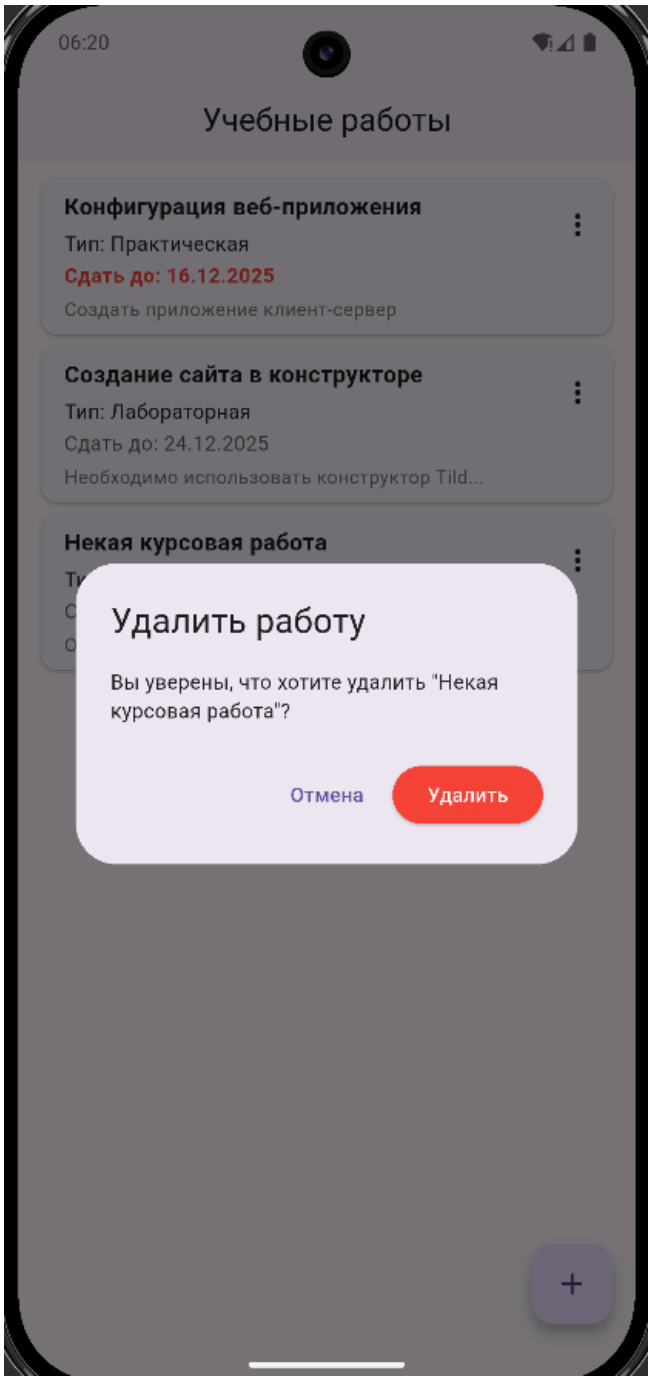

Рисунок 3 – Экранная форма главного экрана

Исходный код корневого компонента представлен ниже:

```
import 'package:flutter/material.dart';
import 'package:flutter_localizations/flutter_localizations.dart';
import 'screens/work_list_screen.dart';
import 'database/database_helper.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await DatabaseHelper.instance.initDatabase();
  runApp(const WorksApp());
}
class WorksApp extends StatelessWidget {
  const WorksApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Works App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        useMaterial3: true,
        appBarTheme: const AppBarTheme(
          centerTitle: true,
          elevation: 2,
        ),
      ),
      localizationsDelegates: const [
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      supportedLocales: const [
        Locale('ru', 'RU'),
        Locale('en', 'US'),
      ],
      home: const WorkListScreen(),
      debugShowCheckedModeBanner: false,
    );
  }
}
```

Исходный код моделей приложения представлен ниже:

```
import 'package:intl/intl.dart';
class WorkType {
  int id;
  String name;

  WorkType({
    required this.id,
    required this.name,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
    };
  }

  factory WorkType.fromMap(Map<String, dynamic> map) {
    return WorkType(
      id: map['id'] as int,
      name: map['name'] as String,
    );
  }
}

class AcademicWork {
  int id;
  String title;
  String description;
  int workTypeId;
  String workTypeName;
  DateTime dueDate;

  AcademicWork({
    this.id = 0,
    required this.title,
    required this.description,
    required this.workTypeId,
    this.workTypeName = '',
```

```
    required this.dueDate,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'title': title,
      'description': description,
      'work_type_id': workTypeId,
      'due_date': DateFormat('yyyy-MM-dd').format(dueDate),
    };
  }

  Map<String, dynamic> toMapWithoutId() {
    return {
      'title': title,
      'description': description,
      'work_type_id': workTypeId,
      'due_date': DateFormat('yyyy-MM-dd').format(dueDate),
    };
  }

  factory AcademicWork.fromMap(Map<String, dynamic> map) {
    return AcademicWork(
      id: map['id'] as int,
      title: map['title'] as String,
      description: map['description'] as String,
      workTypeId: map['work_type_id'] as int,
      dueDate: DateFormat('yyyy-MM-dd').parse(map['due_date'] as String),
    );
  }

  String get formattedDate {
    return DateFormat('dd.MM.yyyy').format(dueDate);
  }

  String get formattedDateFull {
    return DateFormat('dd MMMM yyyy', 'ru_RU').format(dueDate);
  }
}
```

Исходный код модуля с запросами к базе данных представлен ниже:

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

class DatabaseHelper {
  static final DatabaseHelper instance = DatabaseHelper._init();
  static Database? _database;

  DatabaseHelper._init();

  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, 'works_app.db');

    return await openDatabase(
      path,
      version: 1,
      onCreate: _createDatabase,
    );
  }

  Future<void> _createDatabase(Database db, int version) async {
    await db.execute('''
      CREATE TABLE work_types (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL UNIQUE
      )
    ''');

    await db.execute('''
      CREATE TABLE academic_works (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
```

```
      description TEXT,
      work_type_id INTEGER NOT NULL,
      due_date TEXT NOT NULL,
      FOREIGN KEY (work_type_id) REFERENCES work_types (id)
    )
  ''');

  await db.insert('work_types', {'name': 'Практическая'});
  await db.insert('work_types', {'name': 'Лабораторная'});
  await db.insert('work_types', {'name': 'Курсовая'});
}

Future<void> initDatabase() async {
  await database;
}

Future<List<WorkType>> getWorkTypes() async {
  final db = await database;
  final maps = await db.query('work_types');
  return maps.map((map) => WorkType.fromMap(map)).toList();
}

Future<int> insertAcademicWork(AcademicWork work) async {
  final db = await database;
  return await db.insert('academic_works', work.toMapWithoutId());
}

Future<List<AcademicWork>> getAllAcademicWorks() async {
  final db = await database;
  final maps = await db.query('academic_works', orderBy: 'due_date ASC');
  final works = <AcademicWork>[];

  for (var map in maps) {
    final work = AcademicWork.fromMap(map);
    final typeMaps = await db.query(
      'work_types',
      where: 'id = ?',
      whereArgs: [work.workTypeId],
    );
    if (typeMaps.isNotEmpty) {
```

```
      work.workTypeName = typeMaps.first['name'] as String;
    }
    works.add(work);
  }

  return works;
}

Future<AcademicWork?> getAcademicWork(int id) async {
  final db = await database;
  final maps = await db.query(
    'academic_works',
    where: 'id = ?',
    whereArgs: [id],
  );

  if (maps.isNotEmpty) {
    final work = AcademicWork.fromMap(maps.first);
    final typeMaps = await db.query(
      'work_types',
      where: 'id = ?',
      whereArgs: [work.workTypeId],
    );
    if (typeMaps.isNotEmpty) {
      work.workTypeName = typeMaps.first['name'] as String;
    }
    return work;
  }
  return null;
}

Future<int> updateAcademicWork(AcademicWork work) async {
  final db = await database;
  return await db.update(
    'academic_works',
    work.toMapWithoutId(),
    where: 'id = ?',
    whereArgs: [work.id],
  );
}
```

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

  Future close() async {
    final db = await database;
    db.close();
  }
}
```

Исходный код модуля главного окна представлен ниже:

```
import 'package:flutter/material.dart';
import '../database/database_helper.dart';
import '../models/work.dart';
import '../widgets/work_item.dart';
import 'add_edit_work_screen.dart';

class WorkListScreen extends StatefulWidget {
  const WorkListScreen({super.key});

  @override
  State<WorkListScreen> createState() => _WorkListScreenState();
}

class _WorkListScreenState extends State<WorkListScreen> {
  List<AcademicWork> _works = [];
  bool _isLoading = true;
  bool _isRefreshing = false;

  @override
  void initState() {
    super.initState();
    _loadWorks();
  }
```

```
Future<void> _loadWorks() async {
  if (!mounted) return;

  setState(() {
    _isRefreshing = true;
  });

  try {
    final works = await DatabaseHelper.instance.getAllAcademicWorks();

    if (!mounted) return;
    setState(() {
      _works = works;
      _isLoading = false;
      _isRefreshing = false;
    });
  } catch (e) {
    if (!mounted) return;
    setState(() {
      _isLoading = false;
      _isRefreshing = false;
    });
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Ошибка загрузки работ: $e'),
        backgroundColor: Colors.red,
      ),
    );
  }
}

Future<void> _editWork(AcademicWork work) async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => AddEditWorkScreen(work: work),
    ),
  );
```

```
  if (result == true) {
    await _loadWorks();
  }
}

Future<void> _addNewWork() async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => const AddEditWorkScreen(),
    ),
  );

  if (result == true) {
    await _loadWorks();
  }
}

Future<void> _deleteWork(int id) async {
  try {
    await DatabaseHelper.instance.deleteAcademicWork(id);
    await _loadWorks();

    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Работа удалена'),
        duration: Duration(seconds: 2),
      ),
    );
  } catch (e) {
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Ошибка удаления: $e'),
        backgroundColor: Colors.red,
      ),
    );
  }
}
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Учебные работы'),
      actions: [
        if (_isRefreshing)
          const Padding(
            padding: EdgeInsets.only(right: 16.0),
            child: SizedBox(
              width: 20,
              height: 20,
              child: CircularProgressIndicator(strokeWidth: 2),
            ),
          ),
      ],
    ),
    body: _isLoading
        ? const Center(child: CircularProgressIndicator())
        : _works.isEmpty
        ? Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          const Icon(Icons.assignment, size: 64, color: Colors.grey),
          const SizedBox(height: 16),
          const Text(
            'Нет учебных работ',
            style: TextStyle(fontSize: 18, color: Colors.grey),
          ),
          const SizedBox(height: 8),
          const Text(
            'Нажмите + чтобы добавить работу',
            style: TextStyle(color: Colors.grey),
          ),
          const SizedBox(height: 16),
          ElevatedButton.icon(
            onPressed: _addNewWork,
            icon: const Icon(Icons.add),
```

```
              label: const Text('Добавить работу'),
            ),
          ],
        ),
      )
          : RefreshIndicator(
        onRefresh: _loadWorks,
        child: ListView.builder(
          padding: const EdgeInsets.all(8),
          itemCount: _works.length,
          itemBuilder: (context, index) {
            final work = _works[index];
            return WorkItem(
              work: work,
              onEdit: () => _editWork(work),
              onDelete: () => _deleteWork(work.id),
            );
          },
        ),
      ),
      floatingActionButton: _works.isEmpty
          ? null
          : FloatingActionButton(
        onPressed: _addNewWork,
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

Исходный код модуля учебной работы в списке главного окна представлен ниже:

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

class WorkItem extends StatelessWidget {
  final AcademicWork work;
  final VoidCallback onEdit;
  final VoidCallback onDelete;
```

```
const WorkItem({
  super.key,
  required this.work,
  required this.onEdit,
  required this.onDelete,
});

@override
Widget build(BuildContext context) {
  return Card(
    margin: const EdgeInsets.symmetric(vertical: 4, horizontal: 8),
    child: Padding(
      padding: const EdgeInsets.only(left: 16.0, top: 8.0, bottom: 8.0),
      child: Row(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Expanded(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  work.title,
                  style: const TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 16,
                  ),
                ),
                const SizedBox(height: 4),
                Text(
                  'Тип: ${work.workTypeName}',
                  style: const TextStyle(fontSize: 14),
                ),
                const SizedBox(height: 2),
                Text(
                  'Сдать до: ${work.formattedDate}',
                  style: TextStyle(
                    fontSize: 14,
                    color: work.dueDate.isBefore(DateTime.now())
                        ? Colors.red
                        : Colors.grey[700],
```

```
            fontWeight: work.dueDate.isBefore(DateTime.now())
                ? FontWeight.bold
                : FontWeight.normal,
          ),
        ),
        if (work.description.isNotEmpty)
          Padding(
            padding: const EdgeInsets.only(top: 4),
            child: Text(
              work.description.length > 40
                  ? '${work.description.substring(0, 40)}...'
                  : work.description,
              style: TextStyle(
                fontSize: 13,
                color: Colors.grey[600],
              ),
            ),
          ),
      ],
    ),
  ),
  PopupMenuButton<String>(
    onSelected: (value) {
      if (value == 'edit') {
        onEdit();
      } else if (value == 'delete') {
        _showDeleteConfirmation(context);
      }
    },
    itemBuilder: (BuildContext context) {
      return [
        const PopupMenuItem<String>(
          value: 'edit',
          child: Row(
            children: [
              Icon(Icons.edit, size: 20),
              SizedBox(width: 8),
              Text('Редактировать'),
            ],
          ),
```

```
                ),
                const PopupMenuItem<String>(
                  value: 'delete',
                  child: Row(
                    children: [
                      Icon(Icons.delete, color: Colors.red, size: 20),
                      SizedBox(width: 8),
                      Text('Удалить', style: TextStyle(color: Colors.red)),
                    ],
                  ),
                ),
              ];
            },
          ),
        ],
      ),
    ),
  );
}

void _showDeleteConfirmation(BuildContext context) {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      title: const Text('Удалить работу'),
      content: Text('Вы уверены, что хотите удалить "${work.title}"?'),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('Отмена'),
        ),
        ElevatedButton(
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.red,
            foregroundColor: Colors.white,
          ),
          onPressed: () {
            Navigator.pop(context);
            onDelete();
          },
```

```
            child: const Text('Удалить'),
          ),
        ],
      ),
    );
  }
}
```

Исходный код модуля окна добавления и редактирования учебной работы представлен ниже:

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import '../database/database_helper.dart';
import '../models/work.dart';

class AddEditWorkScreen extends StatefulWidget {
  final AcademicWork? work;

  const AddEditWorkScreen({super.key, this.work});

  @override
  State<AddEditWorkScreen> createState() => _AddEditWorkScreenState();
}

class _AddEditWorkScreenState extends State<AddEditWorkScreen> {
  final _formKey = GlobalKey<FormState>();
  final _titleController = TextEditingController();
  final _descriptionController = TextEditingController();

  List<WorkType> _workTypes = [];
  WorkType? _selectedWorkType;
  DateTime _selectedDate = DateTime.now();
  bool _isSaving = false;

  @override
  void initState() {
    super.initState();
    _loadWorkTypes();
    _initForm();
  }
```

```
Future<void> _loadWorkTypes() async {
  final types = await DatabaseHelper.instance.getWorkTypes();
  setState(() {
    _workTypes = types;
    if (widget.work != null && _selectedWorkType == null) {
      _selectedWorkType = _workTypes.firstWhere(
            (type) => type.id == widget.work!.workTypeId,
        orElse: () => _workTypes.first,
      );
    } else if (_selectedWorkType == null && _workTypes.isNotEmpty) {
      _selectedWorkType = _workTypes.first;
    }
  });
}

void _initForm() {
  if (widget.work != null) {
    _titleController.text = widget.work!.title;
    _descriptionController.text = widget.work!.description;
    _selectedDate = widget.work!.dueDate;
  }
}

Future<void> _selectDate(BuildContext context) async {
  final DateTime? picked = await showDatePicker(
    context: context,
    initialDate: _selectedDate,
    firstDate: DateTime(2000),
    lastDate: DateTime(2100),
  );
  if (picked != null && picked != _selectedDate) {
    setState(() {
      _selectedDate = picked;
    });
  }
}

Future<void> _saveWork() async {
  if (!_formKey.currentState!.validate()
```

```
      || _selectedWorkType == null) return;
  if (_isSaving) return;

  setState(() {
    _isSaving = true;
  });

  try {
    final work = AcademicWork(
      id: widget.work?.id ?? 0,
      title: _titleController.text.trim(),
      description: _descriptionController.text.trim(),
      workTypeId: _selectedWorkType!.id,
      dueDate: _selectedDate,
    );

    if (work.id == 0) {
      await DatabaseHelper.instance.insertAcademicWork(work);
    } else {
      await DatabaseHelper.instance.updateAcademicWork(work);
    }

    if (!mounted) return;
    Navigator.pop(context, true);
  } catch (e) {
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Ошибка сохранения: $e'),
        backgroundColor: Colors.red,
      ),
    );
    setState(() {
      _isSaving = false;
    });
  }
}

@override
void dispose() {
```

```
  _titleController.dispose();
  _descriptionController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.work == null ? 'Добавить работу'
          : 'Редактировать работу'),
      actions: [
        IconButton(
          icon: _isSaving
              ? const SizedBox(
            width: 20,
            height: 20,
            child: CircularProgressIndicator(strokeWidth: 2),
          )
              : const Icon(Icons.save),
          onPressed: _isSaving ? null : _saveWork,
        ),
      ],
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: ListView(
          children: [
            TextFormField(
              controller: _titleController,
              decoration: const InputDecoration(
                labelText: 'Название работы',
                border: OutlineInputBorder(),
                prefixIcon: Icon(Icons.title),
              ),
              validator: (value) {
                if (value == null || value.trim().isEmpty) {
                  return 'Введите название работы';
```

```
    }
    return null;
  },
),
const SizedBox(height: 16),
DropdownButtonFormField<WorkType>(
  value: _selectedWorkType,
  decoration: const InputDecoration(
    labelText: 'Тип работы',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.category),
  ),
  items: _workTypes.map((WorkType type) {
    return DropdownMenuItem<WorkType>(
      value: type,
      child: Text(type.name),
    );
  }).toList(),
  onChanged: (WorkType? newValue) {
    setState(() {
      _selectedWorkType = newValue;
    });
  },
  validator: (value) {
    if (value == null) {
      return 'Выберите тип работы';
    }
    return null;
  },
),
const SizedBox(height: 16),
InkWell(
  onTap: () => _selectDate(context),
  child: InputDecorator(
    decoration: const InputDecoration(
      labelText: 'Дата сдачи',
      border: OutlineInputBorder(),
      prefixIcon: Icon(Icons.calendar_today),
    ),
    child: Row(
```

```
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Text(
          DateFormat('dd.MM.yyyy').format(_selectedDate),
          style: const TextStyle(fontSize: 16),
        ),
        const Icon(Icons.calendar_month),
      ],
    ),
  ),
),
const SizedBox(height: 16),
TextFormField(
  controller: _descriptionController,
  decoration: const InputDecoration(
    labelText: 'Описание',
    border: OutlineInputBorder(),
    alignLabelWithHint: true,
    prefixIcon: Icon(Icons.description),
  ),
  maxLines: 5,
  validator: (value) {
    if (value == null || value.trim().isEmpty) {
      return 'Введите описание работы';
    }
    return null;
  },
),
const SizedBox(height: 24),
ElevatedButton(
  onPressed: _isSaving ? null : _saveWork,
  style: ElevatedButton.styleFrom(
    minimumSize: const Size(double.infinity, 50),
  ),
  child: _isSaving
      ? const SizedBox(
    width: 20,
    height: 20,
    child: CircularProgressIndicator(
      strokeWidth: 2,
```

```
                    color: Colors.white,
                  ),
                )
                    : const Text(
                  'Сохранить',
                  style: TextStyle(fontSize: 18),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Вывод

В ходе выполнения лабораторной работы освоены практические навыки интеграции локального хранилища данных в мобильное приложение. Разработано приложение, которое позволяет усправлять списком учбеных работ, присутствует возможность создавать, редактировать и удалять учебные работы. Приложение разработано с помощью фреймворка Flutter.