



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РГУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных
технологий

Отчет по практической работе №5

по дисциплине «Разработка кроссплатформенных мобильных приложений»

Выполнил:

Студент группы ИКБО-07-22

Михеева Д.С.

Проверил:

Старший преподаватель кафедры
МОСИТ

Шешуков Л.С.

Москва 2025 г.

Содержание

Цель и задание	3
Выбор темы приложения и определение бизнес-функций.....	4
Подготовка структуры проекта.....	6
Создание экранных форм	9
Разработка виджетов.....	15
Реализация логики приложения.....	18
Выполнение практической работы №5	21
Вывод.....	26

Цель и задание

Цель работы: создание структуры проекта и экранных форм, разработка логики и виджетов приложения.

Задание: необходимо выбрать индивидуальную тему и подготовить приложение, демонстрирующее реализацию бизнес-логики продукта. В ходе выполнения практического задания студенту нужно предоставить описания логики и выполняемых функций приложения, а также произвести описания принципов его работы. Приложение должно включать в себя обработку пользовательского ввода, работу с данными и вывод информации. Приложение должно состоять из базовых элементов и строится на примитивных системах контроля состояния

План практической работы:

- Выбор темы приложения и определение бизнес-функций
- Подготовка структуры проекта
- Создание экранных форм
- Разработка виджетов
- Реализация логики приложения
- Выполнение практической работы №5;

Выбор темы приложения и определение бизнес-функций

Тема: Приложение для учета расходов

1. Исследование целевой аудитории

Целевая аудитория приложения включает широкий круг пользователей, стремящихся лучше контролировать свои личные финансы. Ключевые группы — студенты, молодые специалисты и семьи, желающие понимать, куда уходят их деньги. Студенты заинтересованы в отслеживании ежедневных расходов на транспорт, питание и развлечения, чтобы планировать ограниченный бюджет. Молодые специалисты хотят контролировать траты по категориям — например, еда, транспорт, покупки — и видеть статистику за неделю или месяц. А семейные пары используют подобные инструменты для планирования семейных расходов, выявления лишних трат и накопления средств.

2. Анализ пользовательских потребностей

Современный пользователь стремится эффективно управлять своими расходами, но часто сталкивается с нехваткой прозрачности в учёте личных финансов.

Основные проблемы:

- отсутствие простого инструмента для быстрого внесения покупок;
- сложность анализа, куда уходит большая часть бюджета;
- сложность в быстром просмотре общей суммы расходов за период.

Приложение решает эти задачи, предоставляя пользователю удобный способ добавлять расходы, распределять их по категориям (еда, транспорт, покупки и т.д.) и просматривать общие суммы за весь период или выбранный день.

3. Определение функциональных требований

Основная функциональность приложения включает:

- добавление новой записи о расходе (сумма, категория, описание, дата);
- отображение списка всех расходов;
- фильтрацию по дате;

- просмотр общей суммы расходов за выбранный период;
- возможность удаления.

4. Разработка бизнес-логики

При добавлении расхода пользователь вводит сумму, выбирает категорию и при желании добавляет описание. Система сохраняет данные в локальной структуре.

Приложение позволяет: просматривать список расходов, фильтровать по дате, удалять записи, автоматически подсчитывать сумму всех расходов за выбранный период.

Подготовка структуры проекта

Разработка мобильного приложения для учёта расходов на фреймворке Flutter требует выбора оптимальной архитектуры, обеспечивающей простоту, читаемость и лёгкость сопровождения проекта. Так как приложение является учебным и имеет ограниченный набор функций (добавление, удаление и просмотр расходов), наиболее подходящим решением будет функционально-ориентированная (feature-based) структура проекта. С ней можно ознакомиться на рисунке 1.

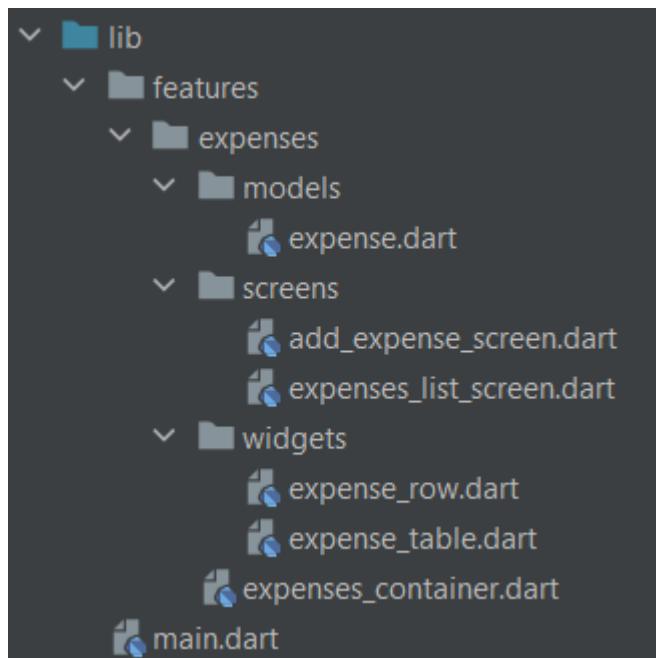


Рисунок 1 – Структурное дерево проекта

Описание структуры:

- models/expense.dart — определяет структуру данных одной записи о расходе. Это основная сущность предметной области приложения.
- screens/expenses_list_screen.dart — отображает список всех расходов, позволяет удалять записи и переходить к экрану добавления новой.
- screens/add_expense_screen.dart — форма для ввода суммы, категории и описания нового расхода.
- widgets/expense_row.dart — отдельная строка списка (один расход).
- widgets/expense_table.dart — агрегирующий виджет, отображающий все строки в виде прокручиваемого списка.
- expenses_container.dart — главный контейнер состояния,

управляющий списком расходов и переключением между экранами. Он отвечает за добавление, редактирование, удаление и фильтрацию расходов, а также за навигацию между формами.

Данный подход позволяет сгруппировать все файлы, относящиеся к одной функциональной области — учёту расходов — в пределах одной директории. Это делает кодовую базу логически цельной и интуитивно понятной для разработчика. В отличие от классической трёхслойной архитектуры (data, domain, presentation), feature-based структура в данном случае исключает избыточные слои, так как приложение не использует удалённые источники данных и сложную бизнес-логику.

Основная цель архитектуры — упростить реализацию и сопровождение кода, сохранив при этом чёткое разделение ответственности. Так как приложение не взаимодействует с сервером и хранит данные только в памяти устройства, целесообразно хранить состояние (список расходов) внутри одного контейнера состояния (StatefulWidget), а операции (добавление, удаление, фильтрация) реализовывать как методы внутри этого контейнера. Это исключает необходимость выделения отдельных уровней data и domain, которые в данном контексте были бы искусственными. Вся логика остаётся внутри одной фичи — expenses.

Для корректной работы приложение требуется описать структуру Expense. Она состоит из достаточно простых данных:

- id — уникальный идентификатор записи, формируется автоматически;
- amount — сумма расхода;
- category — категория траты (например, “Еда”, “Транспорт”);
- description — optionalное описание (например, “Кофе в кафе”);
- date — дата добавления записи.

Реализацию данной структуры можно увидеть на рисунке 2.

The screenshot shows a code editor with two tabs open: 'expense.dart' and 'expenses_list_screen.dart'. The 'expense.dart' tab is active and displays the following Dart code:

```
1 class Expense {  
2     final String id;  
3     final double amount;  
4     final String category;  
5     final String? description;  
6     final DateTime date;  
7  
8     Expense({  
9         required this.id,  
10        required this.amount,  
11        required this.category,  
12        this.description,  
13        required this.date,  
14    });  
15 }
```

Рисунок 2 – Реализация структуры Expense

Создание экранных форм

1. Экран списка расходов

Экран списка — основная точка входа пользователя в приложение. Его назначение:

- показать пользователю текущий набор записей о расходах;
- позволить быстро добавить новую запись;
- дать возможность удалить;
- предоставить простой фильтр (показ общей суммы за выбранный период).

Экран выполняет как информирующую (показывает список и суммы), так и управляющую (инициирует навигацию к форме добавления, предоставляет возможность удаления расхода) функцию. Экран не хранит собственное состояние, а получает его в виде параметров от контейнера состояния (ExpensesContainer).

Он принимает:

- список расходов List<Expense> expenses;
- функции обратного вызова (onAddTap, onDelete, onFilter, onClearFilter);
- выбранную дату фильтра selectedDate.

Такой подход полностью соответствует принципу явной инъекции зависимостей (explicit dependency injection) и разделения ответственности. В терминах Flutter-архитектуры экран реализован на базе следующих базовых элементов:

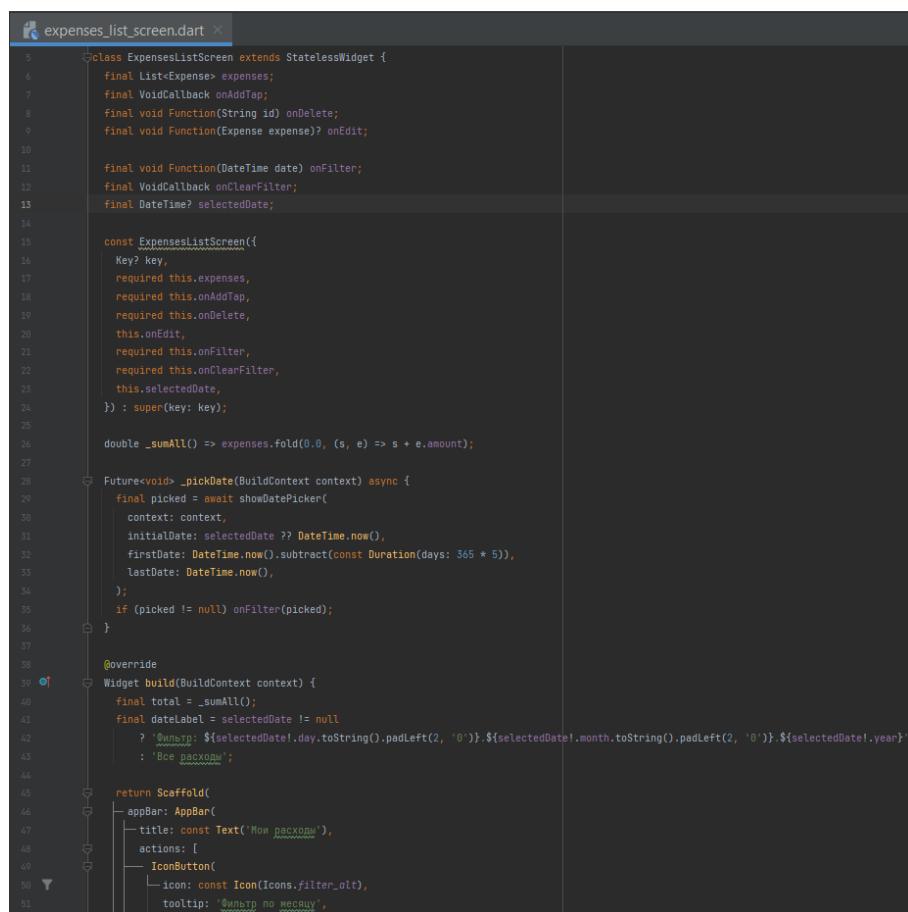
- Scaffold — базовый контейнер приложения, включающий верхнюю панель (AppBar), основную область (body) и плавающую кнопку действий (FloatingActionButton).
- AppBar — содержит заголовок «Мои расходы» и две иконки действий: Icons.filter_alt — открывает диалог выбора даты (DatePicker) для фильтрации и Icons.clear — сбрасывает активный фильтр.
- Container — информационная панель под заголовком,

отображающая активный фильтр и сумму расходов.

- `ExpenseTable` — отдельный виджет списка, отображающий все записи о расходах.
- `FloatingActionButton` — инициирует добавление нового расхода, вызывая `onAddTap`.

Ключевым дополнением экрана является механизм фильтрации по дате, реализованный через `showDatePicker()`. При выборе даты вызывается переданный колбэк `onFilter`, который изменяет состояние в контейнере. Контейнер, в свою очередь, пересчитывает список `_filteredExpenses`, и экран получает уже обновлённые данные.

Реализацию экранной формы в коде можно увидеть на рисунках 3-4.



```
5   class ExpensesListScreen extends StatelessWidget {
6     final List<Expense> expenses;
7     final VoidCallback onAddTap;
8     final void Function(String id) onDelete;
9     final void Function(Expense expense)? onEdit;
10
11    final void Function(DateTime date) onFilter;
12    final VoidCallback onClearFilter;
13    final DateTime? selectedDate;
14
15    const ExpensesListScreen({
16      Key? key,
17      required this.expenses,
18      required this.onAddTap,
19      required this.onDelete,
20      this.onEdit,
21      required this.onFilter,
22      required this.onClearFilter,
23      this.selectedDate,
24    }) : super(key: key);
25
26    double _sumAll() => expenses.fold(0.0, (s, e) => s + e.amount);
27
28    Future<void> _pickDate(BuildContext context) async {
29      final picked = await showDatePicker(
30        context: context,
31        initialDate: selectedDate ?? DateTime.now(),
32        firstDate: DateTime.now().subtract(const Duration(days: 365 * 5)),
33        lastDate: DateTime.now(),
34      );
35      if (picked != null) onFilter(picked);
36    }
37
38    @override
39    Widget build(BuildContext context) {
40      final total = _sumAll();
41      final dateLabel = selectedDate != null
42        ? DateFormat('дд.мм.гг').format(selectedDate)
43        : 'Все расходы';
44
45      return Scaffold(
46        appBar: AppBar(
47          title: const Text('Мои расходы'),
48          actions: [
49            IconButton(
50              icon: const Icon(Icons.filter_alt),
51              tooltip: 'Фильтр по месяцу',
52            ),
53          ],
54        ),
55        body: Column(
56          children: [
57            Padding(
58              padding: const EdgeInsets.all(8.0),
59              child: ExpenseTable(expenses: expenses),
60            ),
61            Padding(
62              padding: const EdgeInsets.all(8.0),
63              child: FABContainer(onAddTap: onAddTap),
64            ),
65          ],
66        ),
67      );
68    }
69  }
```

Рисунок 3 – Реализация экрана списка расходов

```
        tooltip: 'Фильтр по месяцу',
        onPressed: () => _pickDate(context),
      ), // IconButton
      if (selectedDate != null)
        IconButton(
          icon: const Icon(Icons.clear),
          tooltip: 'Сбросить фильтр',
          onPressed: onClearFilter,
        ), // IconButton
      ],
    ), // AppBar
    body: Column(
      children: [
        Container(
          width: double.infinity,
          color: Colors.grey[100],
          padding: const EdgeInsets.all(12),
          child: Row(
            children: [
              Expanded(
                child: Text(
                  dateLabel,
                  style: const TextStyle(fontSize: 15, fontWeight: FontWeight.bold),
                ), // Text
              ), // Expanded
              Text(
                '${total.toStringAsFixed(2)} ₽',
                style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
              ), // Text
            ],
          ), // Row
        ), // Container
        Expanded(
          child: ExpenseTable(
            expenses: expenses,
            onDelete: onDelete,
            onTap: onEdit,
          ), // ExpenseTable
        ), // Expanded
      ],
    ), // Column
    floatingActionButton: FloatingActionButton(
      onPressed: onAddTap,
      child: const Icon(Icons.add),
    ), // FloatingActionButton
  ); // Scaffold

```

Рисунок 4 – Продолжение реализация экрана списка расходов

2. Экран добавления расхода

Экран добавления расхода предназначен для ввода новых записей в систему учёта финансов. Его основная функция — сбор и валидация пользовательских данных, необходимых для формирования новой сущности

«Расход». Пользователь может ввести сумму, выбрать категорию, при необходимости добавить описание и установить дату расхода. После заполнения формы данные передаются в контейнер состояния (`ExpensesContainer`), где происходит их сохранение в общий список расходов.

Экран реализован в виде `StatefulWidget`, так как содержит внутренние поля, изменяющиеся в процессе работы (например, выбранная дата и категория). Он не управляет глобальным состоянием приложения напрямую, а взаимодействует с ним через функцию обратного вызова `onSave`, передаваемую из контейнера. Такое решение соответствует принципам инверсии управления и слабой связанности компонентов, что делает экран легко переиспользуемым и изолированным от остальной логики приложения.

Для ввода данных используется комбинация стандартных компонентов Flutter:

- `Form` и `GlobalKey<FormState>` — обеспечивают валидацию и контроль правильности ввода перед сохранением.
- `TextFormField` — поля ввода для суммы и описания. Поле суммы содержит префикс ₽ и ограничено числовым типом клавиатуры. Валидация гарантирует, что введено положительное число.
- `DropdownButtonFormField` — выпадающий список для выбора категории расхода.
- `TextButton` с вызовом `showDatePicker()` — позволяет выбрать дату расхода.
- `ElevatedButton` — кнопка сохранения данных.

Реализацию экрана добавления расходов в коде можно увидеть на рисунках 5-7.

```
add_expense_screen.dart ×

3   class AddExpenseScreen extends StatefulWidget {
4     final void Function({
5       required double amount,
6       required String category,
7       String? description,
8       required DateTime date,
9     }) onSave;
10    final List<String> categories;
11
12    const AddExpenseScreen({
13      Key? key,
14      required this.onSave,
15      required this.categories,
16    }) : super(key: key);
17
18    @override
19    State<AddExpenseScreen> createState() => _AddExpenseScreenState();
20  }
21
22  class _AddExpenseScreenState extends State<AddExpenseScreen> {
23    final _formKey = GlobalKey<FormState>();
24    final _amountController = TextEditingController();
25    String? _selectedCategory;
26    final _descriptionController = TextEditingController();
27    DateTime _selectedDate = DateTime.now();
28
29    @override
30    void initState() {
31      super.initState();
32      if (widget.categories.isNotEmpty) {
33        _selectedCategory = widget.categories.first;
34      }
35    }
36
37    @override
38    void dispose() {
39      _amountController.dispose();
40      _descriptionController.dispose();
41      super.dispose();
42    }
43
44    Future<void> _pickDate() async {
45      final picked = await showDatePicker(
46        context: context,
47        initialDate: _selectedDate,
48        firstDate: DateTime.now().subtract(const Duration(days: 365 * 5)),
49        lastDate: DateTime.now(),
50      );
51    }
52  }
```

Рисунок 5 – Реализация экрана добавления расхода

```
add_expense_screen.dart

51     if (picked != null) {
52         setState(() {
53             _selectedDate = picked;
54         });
55     }
56 }
57
58 void _submit() {
59     if (_formKey.currentState?.validate() ?? false) {
60         final amount = double.parse(_amountController.text.replaceAll(',', '.'));
61         final category = _selectedCategory ?? 'Поесть';
62         final description = _descriptionController.text.trim().isEmpty ? null : _descriptionController.text.trim();
63         widget.onSave(amount: amount, category: category, description: description, date: _selectedDate);
64     }
65 }
66
67 @override
68 Widget build(BuildContext context) {
69     final dateStr = "${_selectedDate.day.toString().padLeft(2, '0')}.${_selectedDate.month.toString().padLeft(2, '0')}.${_selectedDate.year}";
70     return Scaffold(
71         appBar: AppBar(title: const Text('Добавить расход')),
72         body: Padding(
73             padding: const EdgeInsets.all(16.0),
74             child: Form(
75                 key: _formKey,
76                 child: Column(
77                     children: [
78                         TextFormField(
79                             controller: _amountController,
80                             decoration: const InputDecoration(labelText: 'Сумма', prefixText: 'Р '),
81                             keyboardType: const TextInputType.numberWithOptions(decimal: true),
82                             validator: (v) {
83                                 if (v == null || v.trim().isEmpty) return 'Введите сумму';
84                                 final parsed = double.tryParse(v.replaceAll(',', '.'));
85                                 if (parsed == null || parsed <= 0) return 'Введите корректную сумму > 0';
86                                 return null;
87                             },
88                         ), // TextFormField
89                         const SizedBox(height: 12),
90                         DropdownButtonFormField<String>(
91                             value: _selectedCategory,
92                             decoration: const InputDecoration(labelText: 'Категория'),
93                             items: widget.categories.map((c) => DropdownMenuItem(value: c, child: Text(c))).toList(),
94                             onChanged: (v) => setState(() => _selectedCategory = v),
95                             validator: (v) => (v == null || v.isEmpty) ? 'Выберите категорию' : null,
96                         ), // DropdownButtonFormField
97                         const SizedBox(height: 12),
98                         TextFormField(
99                         decoration: const InputDecoration(labelText: 'Описание'))),
100                     ],
101                 ),
102             ),
103         ),
104     );
105 }
```

Рисунок 6 – Продолжение реализация экрана добавления расхода

```
    TextFormField(
      controller: _descriptionController,
      decoration: const InputDecoration(labelText: 'Описание (необязательно)'),
      maxLines: 2,
    ), // TextFormField
    const SizedBox(height: 12),
    Row(
      children: [
        Text('Дата: $dateStr'),
        const Spacer(),
        TextButton(onPressed: _pickDate, child: const Text('Изменить дату')),
      ],
    ), // Row
    const Spacer(),
    SizedBox(
      width: double.infinity,
      child: ElevatedButton(
        onPressed: _submit,
        child: const Text('Сохранить'),
      ), // ElevatedButton
    ), // SizedBox
  ],
), // Column
), // Form
), // Padding
); // Scaffold
}
```

Рисунок 7 – Продолжение реализация экрана добавления расхода

Разработка виджетов

1. Виджет строки расходы

ExpenseRow — это Flutter-виджет, который отображает один расход в виде строки карточки (Card) с основными деталями. Он показывает категорию расхода (expense.category) как заголовок, описание расхода (expense.description) или дату, если описания нет. Справа отображает: сумму расхода (expense.amount) в рублях и дату расхода в формате дд.мм.гггг.

Основные компоненты:

- Flutter StatelessWidget для статического отображения.
- Card и ListTile для удобного оформления строки.
- Форматирование даты и суммы.
- Callback функции для интерактивности (onTap, onDelete).

Реализацию виджета строки расхода в коде можно увидеть на рисунке 8.

```
class ExpenseRow extends StatelessWidget {
  final Expense expense;
  final VoidCallback? onTap;
  final VoidCallback? onDelete;

  const ExpenseRow({
    Key? key,
    required this.expense,
    this.onTap,
    this.onDelete,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final dateStr = "${expense.date.day.toString().padLeft(2, '0')}.${expense.date.month.toString().padLeft(2, '0')}.${expense.date.year}";
    return Card(
      margin: const EdgeInsets.symmetric(horizontal: 8, vertical: 4),
      child: ListTile(
        onTap: onTap,
        title: Text(
          expense.category,
          style: const TextStyle(fontWeight: FontWeight.bold),
        ), // Text
        subtitle: Text(expense.description ?? dateStr),
        trailing: Column(
          mainAxisAlignment: MainAxisAlignment.end,
          children: [
            Text('${expense.amount.toStringAsFixed(2)} ₽', style: const TextStyle(fontSize: 16)),
            const SizedBox(height: 4),
            Text(dateStr, style: TextStyle(fontSize: 12, color: Colors.grey[600])),
          ],
        ), // Column
      ), // ListTile
    );
  }
}
```

Рисунок 8 – Реализация виджета строки расхода

2. Виджет таблицы расходов

ExpenseTable — это Flutter-виджет, который отображает список расходов в виде прокручиваемой таблицы (списка ListView), используя виджет ExpenseRow для каждой строки. Данный виджет принимает список расходов (expenses) и функции для удаления и обработки нажатия на расход. Если список пуст, показывает сообщение: "Нет расходов для отображения". Для каждого расхода виджет: оборачивает его в Dismissible, чтобы можно было удалить свайпом влево, использует ExpenseRow для отображения информации о расходе, вызывает onDelete при свайпе, onTap при нажатии.

Основные компоненты:

- Flutter StatelessWidget и ListView.builder для прокручиваемого списка.
- Dismissible для свайпа и удаления элементов.
- Виджет ExpenseRow для отображения каждой строки расходов.
- Callback функции для управления событиями.

Реализацию виджета таблицы расходов в коде можно увидеть на рисунке 9.

```
class ExpenseTable extends StatelessWidget {
    final List<Expense> expenses;
    final void Function(String id) onDelete;
    final void Function(Expense expense)? onTap;

    const ExpenseTable({
        Key? key,
        required this.expenses,
        required this.onDelete,
        this.onTap,
    }) : super(key: key);

    @override
    Widget build(BuildContext context) {
        if (expenses.isEmpty) {
            return const Center(
                child: Text(
                    'Нет расходов для отображения',
                    style: TextStyle(fontSize: 16),
                ),
            );
        }
    }

    return ListView.builder(
        itemCount: expenses.length,
        itemBuilder: (context, index) {
            final e = expenses[index];
            return Dismissible(
                key: ValueKey(e.id),
                direction: DismissDirection.endToStart,
                background: Container(
                    color: Colors.red,
                    alignment: Alignment.centerRight,
                    padding: const EdgeInsets.only(right: 20),
                    child: const Icon(Icons.delete, color: Colors.white),
                ),
                onDismissed: (_) => onDelete(e.id),
                child: ExpenseRow(
                    expense: e,
                    onTap: onTap != null ? () => onTap!(e) : null,
                ),
            );
        },
    );
}
```

Рисунок 9 – Реализация виджета таблицы расходов

Реализация логики приложения

Виджет ExpensesContainer реализует центральную логику всего приложения для учёта расходов и объединяет работу экранов добавления и отображения расходов в одном контейнере. Приложение построено на Flutter с использованием StatefulWidget, чтобы динамически менять состояние интерфейса и данные в ответ на действия пользователя. Основной класс ExpensesContainer хранит список всех расходов в виде _expenses, а также текущий выбранный экран (_current) — либо список расходов, либо экран добавления нового расхода.

Пользователь может выполнять несколько действий: добавлять новые расходы, просматривать список существующих, фильтровать их по дате и удалять ненужные.

Добавление нового расхода происходит через экран AddExpenseScreen, где пользователь вводит сумму, категорию, описание и дату. После сохранения данные создаются в виде объекта Expense с уникальным id, добавляются в начало списка _expenses, и экран автоматически переключается обратно на список расходов.

Список расходов отображается через экран ExpensesListScreen, который получает уже отфильтрованные расходы через геттер _filteredExpenses. Фильтрация работает по дате: если выбранная дата _selectedDate установлена, в список попадают только расходы с совпадающей датой; если фильтр не выбран, отображаются все расходы. Пользователь может устанавливать фильтр даты или очищать его с помощью методов _setFilterDate и _clearFilter.

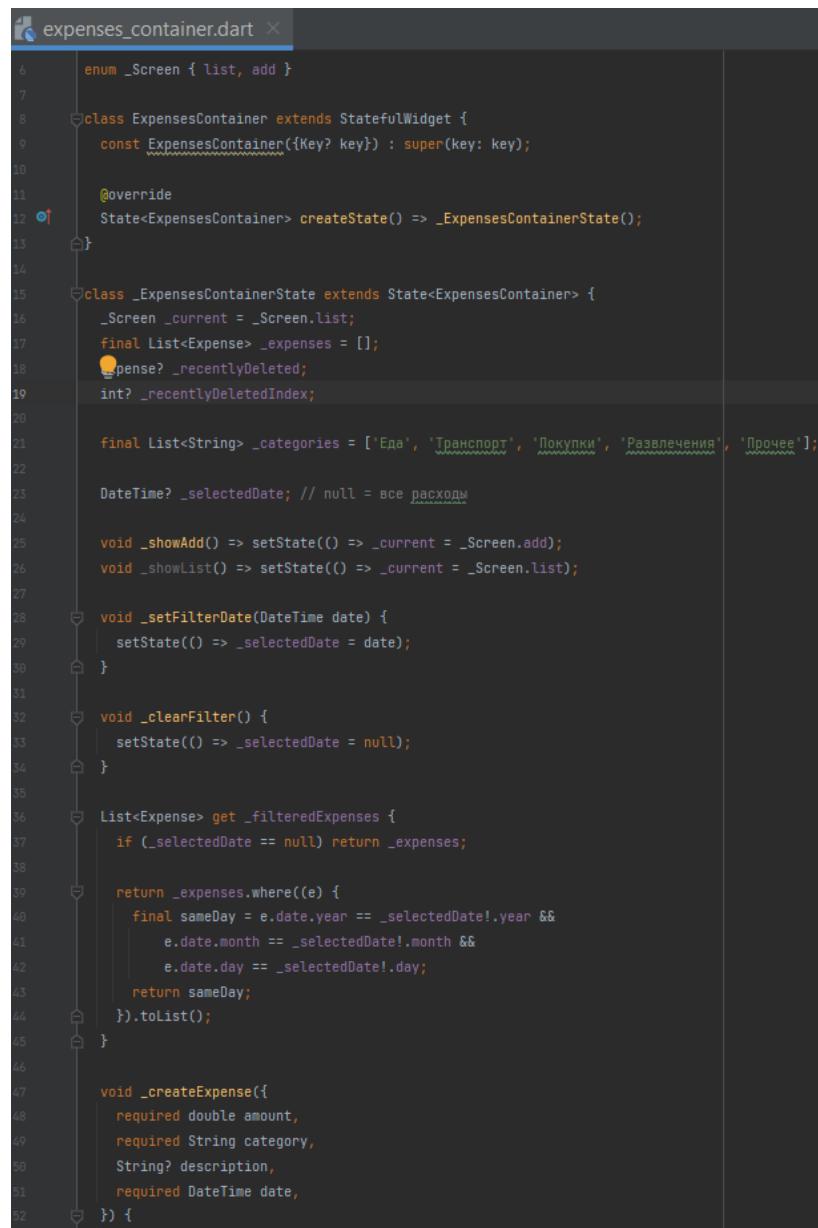
Удаление расхода реализовано с возможностью отмены. Когда пользователь свайпает элемент в списке, вызывается _deleteExpense, который удаляет расход из списка, сохраняет его временно в _recentlyDeleted, а также индекс удалённого элемента. Затем отображается Snackbar с уведомлением об удалении и кнопкой «Отменить». Если пользователь нажимает «Отменить», расход восстанавливается на прежнюю позицию.

Навигация между экраном списка и экраном добавления реализована через

смену состояния `_current` и анимированное переключение виджетов с помощью `AnimatedSwitcher`. Таким образом, интерфейс остаётся плавным, а пользователь видит переход без резких изменений.

Кроме того, контейнер использует предопределённый список категорий расходов `_categories`, что упрощает ввод данных на экране добавления. Все действия интегрированы с состоянием приложения через `setState`, что обеспечивает мгновенное обновление интерфейса при изменении списка расходов, фильтрации или переключении экранов.

Реализацию в коде можно увидеть на рисунках 10-12.



The screenshot shows a code editor with a dark theme. The file is named `expenses_container.dart`. The code defines a `StatefulWidget` named `ExpensesContainer` with a `createState` method returning a `_ExpensesContainerState` object. The state class contains fields for the current screen (`_Screen.list` or `_Screen.add`), a list of expenses (`_expenses`), a recently deleted expense (`_expenseRecentlyDeleted`), and the index of the recently deleted item (`_recentlyDeletedIndex`). It also lists categories like 'Еда', 'Транспорт', etc. Methods include `_showAdd`, `_showList`, `_setFilterDate`, `_clearFilter`, and a getter for filtered expenses that filters by date if a specific date is selected. Finally, there's a `_createExpense` method with parameters for amount, category, description, and date.

```
enum _Screen { list, add }

class ExpensesContainer extends StatefulWidget {
  const ExpensesContainer({Key? key}) : super(key: key);

  @override
  State<ExpensesContainer> createState() => _ExpensesContainerState();
}

class _ExpensesContainerState extends State<ExpensesContainer> {
  _Screen _current = _Screen.list;
  final List<Expense> _expenses = [];
  Expense? _expenseRecentlyDeleted;
  int? _recentlyDeletedIndex;

  final List<String> _categories = ['Еда', 'Транспорт', 'Докупки', 'Развлечения', 'Прочее'];

  DateTime? _selectedDate; // null = все расходы

  void _showAdd() => setState(() => _current = _Screen.add);
  void _showList() => setState(() => _current = _Screen.list);

  void _setFilterDate(DateTime date) {
    setState(() => _selectedDate = date);
  }

  void _clearFilter() {
    setState(() => _selectedDate = null);
  }

  List<Expense> get _filteredExpenses {
    if (_selectedDate == null) return _expenses;

    return _expenses.where((e) {
      final sameDay = e.date.year == _selectedDate!.year &&
        e.date.month == _selectedDate!.month &&
        e.date.day == _selectedDate!.day;
      return sameDay;
    }).toList();
  }

  void _createExpense({
    required double amount,
    required String category,
    String? description,
    required DateTime date,
  }) {
}
```

Рисунок 10 – Реализация основной логики приложения

```
expenses_container.dart
52     }
53     final newExpense = Expense(
54         id: DateTime.now().microsecondsSinceEpoch.toString(),
55         amount: amount,
56         category: category,
57         description: description,
58         date: date,
59     ); // Expense
60     setState(() {
61         _expenses.insert(0, newExpense);
62         _current = _Screen.list;
63     });
64 }
65
66 void _deleteExpense(String id) {
67     final index = _expenses.indexWhere((e) => e.id == id);
68     if (index == -1) return;
69     setState(() {
70         _recentlyDeleted = _expenses.removeAt(index);
71         _recentlyDeletedIndex = index;
72     });
73     ScaffoldMessenger.of(context).hideCurrentSnackBar();
74     ScaffoldMessenger.of(context).showSnackBar(
75         SnackBar(
76             content: const Text('Расход удалён'),
77             action: SnackBarAction(
78                 label: 'Отменить',
79                 onPressed: () {
80                     if (_recentlyDeleted != null && _recentlyDeletedIndex != null) {
81                         setState(() {
82                             _expenses.insert(_recentlyDeletedIndex!, _recentlyDeleted!);
83                             _recentlyDeleted = null;
84                             _recentlyDeletedIndex = null;
85                         });
86                     }
87                 },
88             ),
89         ), // SnackBar
90     );
91 }
92
93 @override
94 Widget build(BuildContext context) {
95     Widget child;
96     switch (_current) {
97         case _Screen.list:
98             child = ExpensesListScreen(
99                 expenses: List.unmodifiable(_filteredExpenses),
100            
```

Рисунок 11 – Продолжение реализации основной логики приложения

```

expenses: List.unmodifiable(_filteredExpenses),
onAddTap: _showAdd,
onDelete: _deleteExpense,
onEdit: null,
onFilter: _setFilterDate,
onClearFilter: _clearFilter,
selectedDate: _selectedDate,
); // ExpensesListScreen
break;
case _Screen.add:
    child = AddExpenseScreen(
        categories: _categories,
        onSave: ({required amount, required category, description, required date}) {
            _createExpense(amount: amount, category: category, description: description, date: date);
        },
    );
    break;
}

return AnimatedSwitcher(duration: const Duration(milliseconds: 200), child: child);
}

```

Рисунок 12 – Продолжение реализации основной логики приложения

Выполнение практической работы №5

В пунктах выше было сформировано полное описание работы приложения с предоставлением кода для каждого элемента. Теперь необходимо проверить работу приложения.

На рисунке 13 видно экран отображения списка расходов, при первой загрузке он пустой, соответственно выводится сообщение «Нет расходов для отображения».

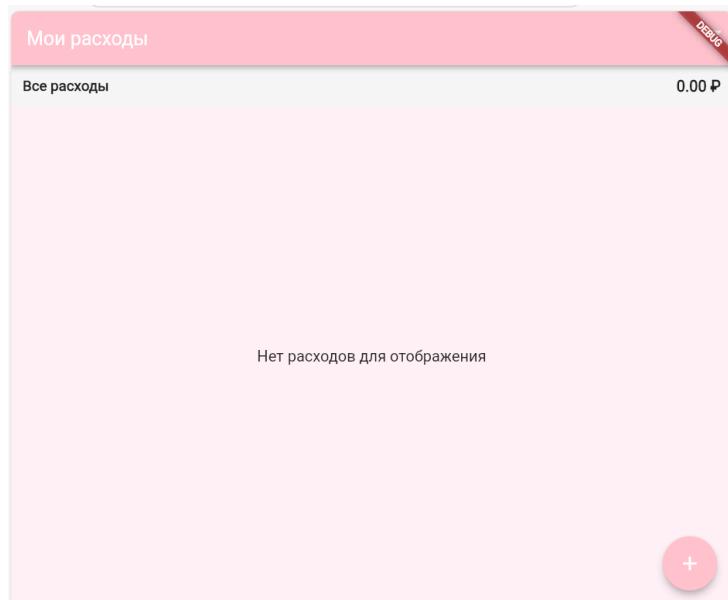


Рисунок 13 – Список до добавления расхода

Далее с помощью кнопки «+» перейдем на экран добавления нового расхода (Рисунок 14).

A screenshot of a mobile application interface titled 'Добавить расход' (Add expense). The top bar is pink with the title and a 'DEBUG' badge. The form consists of several input fields: 'Сумма' (Sum) with a placeholder '0.00 ₽', 'Категория' (Category) set to 'Еда' (Food), 'Описание (необязательно)' (Description (optional)), 'Дата: 17.10.2025' (Date: 17.10.2025), and a 'Изменить дату' (Change date) link. At the bottom is a pink 'Сохранить' (Save) button.

Рисунок 14 – Экран добавления расходов

Внесем данные для нового расхода (Рисунок 15).

Добавить расход

Сумма
₽ 55

Категория
Покупки

Описание (необязательно)
носки

Дата: 17.10.2025 Изменить дату

Сохранить

Рисунок 15 – Заполнение данных расхода

Видим, что внесенные данные теперь отображаются в списке (Рисунок 16).

Мои расходы	
Все расходы	55.00 ₽
Покупки носки	55.00 ₽ 17.10.2025

Рисунок 16 – Список после добавления нового расхода

Проверка корректных данных в поле суммы расхода имеет ограничения по вводу, на рисунке 17 видно, что приложение не позволяет вводить отрицательные значения.

The screenshot shows a mobile application interface for adding an expense. At the top, a pink header bar says 'Добавить расход'. Below it, there's a red error message: 'Сумма P -97' followed by 'Введите корректную сумму > 0'. A dropdown menu for 'Категория' is open, showing 'Еда' selected. There's a note below: 'Описание (необязательно)'. At the bottom, the date 'Дата: 17.10.2025' is shown next to a 'Изменить дату' button, and a large blue 'Сохранить' button at the bottom right.

Рисунок 17 – Некорректный ввод суммы

Проверим функционал фильтрации. Заполним список некоторыми записями о расходах (Рисунок 18).

The screenshot shows a list of expenses under a pink header 'Мои расходы'. The total amount is listed as '18697.00 ₽'. The list contains four items:

Все расходы	18697.00 ₽
Развлечения 17.10.2025	10000.00 ₽ 17.10.2025
Транспорт 15.10.2025	865.00 ₽ 15.10.2025
Еда 13.10.2025	7777.00 ₽ 13.10.2025
Покупки НОСКИ	55.00 ₽ 17.10.2025

A blue '+' button is located at the bottom right of the list area.

Рисунок 18 – Список до фильтрации

Выберем определенную дату и получим обновленный список (Рисунок 19).

Мои расходы		DEBUG
Фильтр: 17.10.2025	10055.00 ₽	
Развлечения 17.10.2025	10000.00 ₽	17.10.2025
Покупки НОСКИ	55.00 ₽	17.10.2025

Рисунок 19 – Список после фильтрации

Проверим работу функции удаления расхода. Заполним список записями (Рисунок 20).

Мои расходы		DEBUG
Все расходы	13673.00 ₽	
Покупки 17.10.2025	6353.00 ₽	17.10.2025
Развлечения 07.10.2025	6543.00 ₽	07.10.2025
Еда 17.10.2025	777.00 ₽	17.10.2025

Рисунок 20 – Список до удаления

С помощью свайпа влево удалим один из элементов. Обновленный список изображен на рисунке 21. Также видно сообщение об удалении расхода и временную возможность отменить удаление.

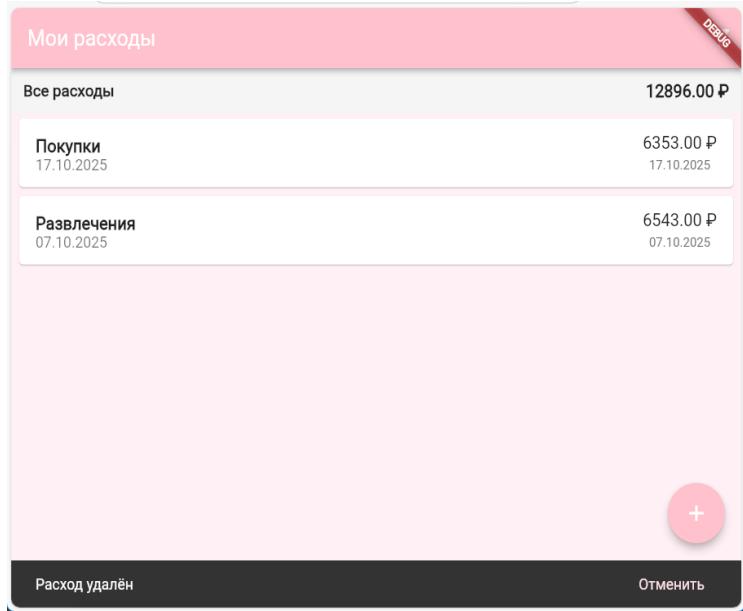


Рисунок 21 – Список после удаления

Вывод

В ходе выполнения практической работы были освоены: создание структуры проекта и экранных форм, разработка логики и виджетов приложения.

Также была выбрана тема и подготовлено приложение, демонстрирующее реализацию бизнес-логики продукта. Было сформировано описание логики и выполняемых функций приложения, а также описания принципов его работы. Приложение включает в себя обработку пользовательского ввода, работу с данными и вывод информации. Приложение состоит из базовых элементов и строится на примитивных системах контроля состояния.

Исходный код приложения можно посмотреть по [ссылке на GitHub](#).