МИНОБРНАУКИ РОССИИ

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

**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.

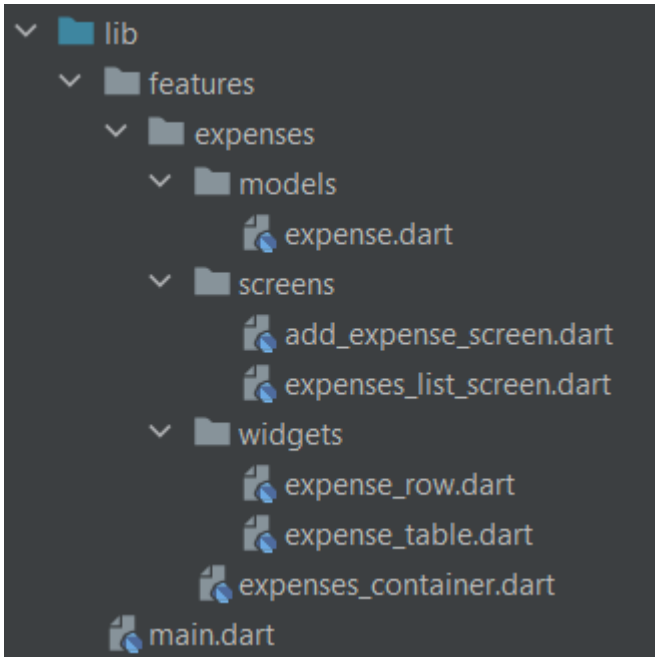


Рисунок 1 – Структурное дерево проекта

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 — опциональное описание (например, “Кофе в кафе”);
- date — дата добавления записи.

Реализацию данной структуры можно увидеть на рисунке 2.

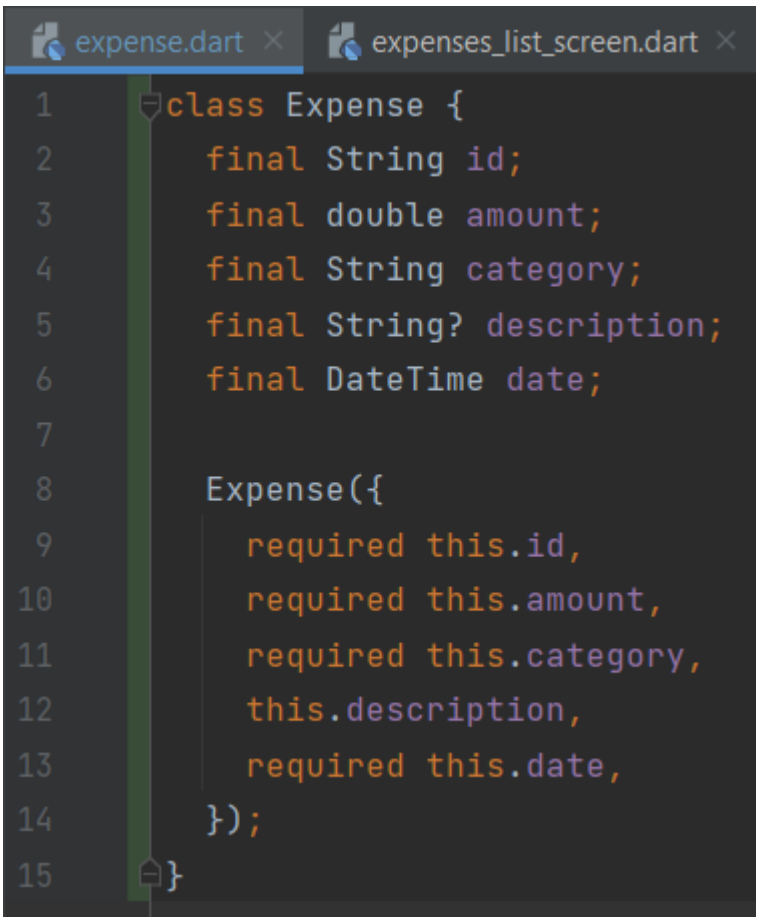


Рисунок 2 – Реализация структуры Expense

## 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.

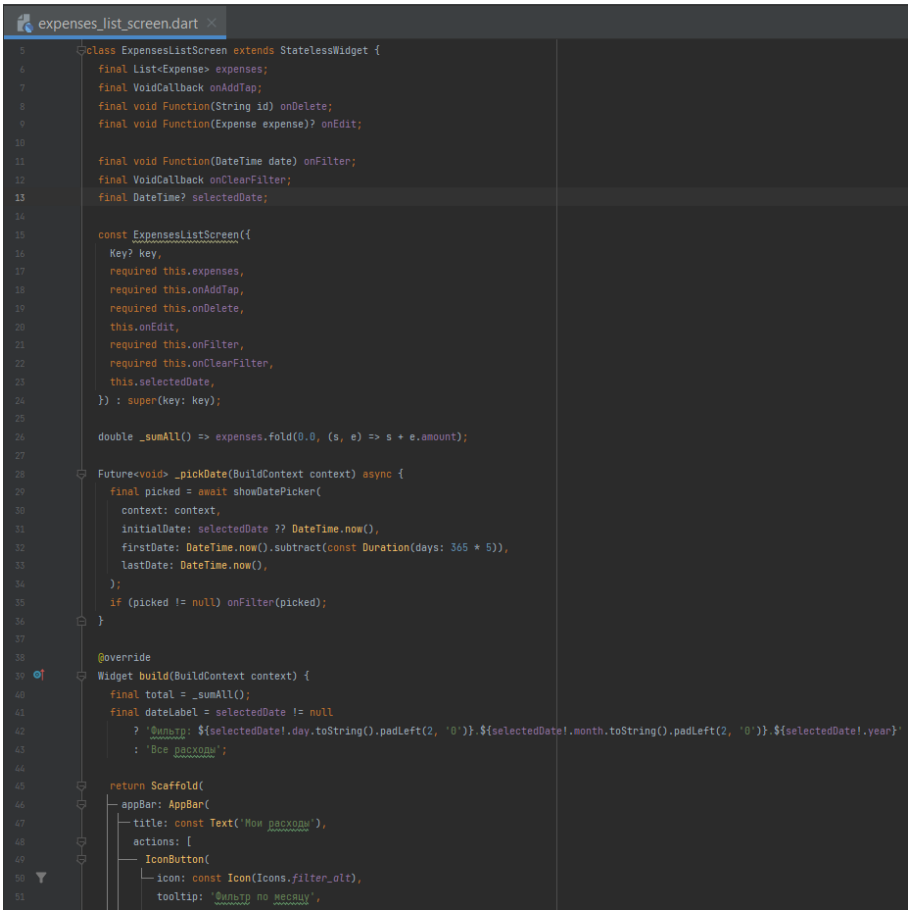


Рисунок 3 – Реализация экрана списка расходов


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

```dart
class AddExpenseScreen extends StatefulWidget {
  final void Function({
  required double amount,
  required String category,
  String? description,
  required DateTime date,
  }) onSave;
  final List<String> categories;

  const AddExpenseScreen({
    Key? key,
    required this.onSave,
    required this.categories,
  }) : super(key: key);

  @override
  State<AddExpenseScreen> createState() => _AddExpenseScreenState();
}

class _AddExpenseScreenState extends State<AddExpenseScreen> {
  final _formKey = GlobalKey<FormState>();
  final _amountController = TextEditingController();
  String? _selectedCategory;
  final _descriptionController = TextEditingController();
  DateTime _selectedDate = DateTime.now();

  @override
  void initState() {
    super.initState();
    if (widget.categories.isNotEmpty) {
      _selectedCategory = widget.categories.first;
    }
  }

  @override
  void dispose() {
    _amountController.dispose();
    _descriptionController.dispose();
    super.dispose();
  }

  Future<void> _pickDate() async {
    final picked = await showDatePicker(
      context: context,
      initialDate: _selectedDate,
      firstDate: DateTime.now().subtract(const Duration(days: 365 * 5)),
      lastDate: DateTime.now(),
    );
```

Рисунок 5 – Реализация экрана добавления расхода

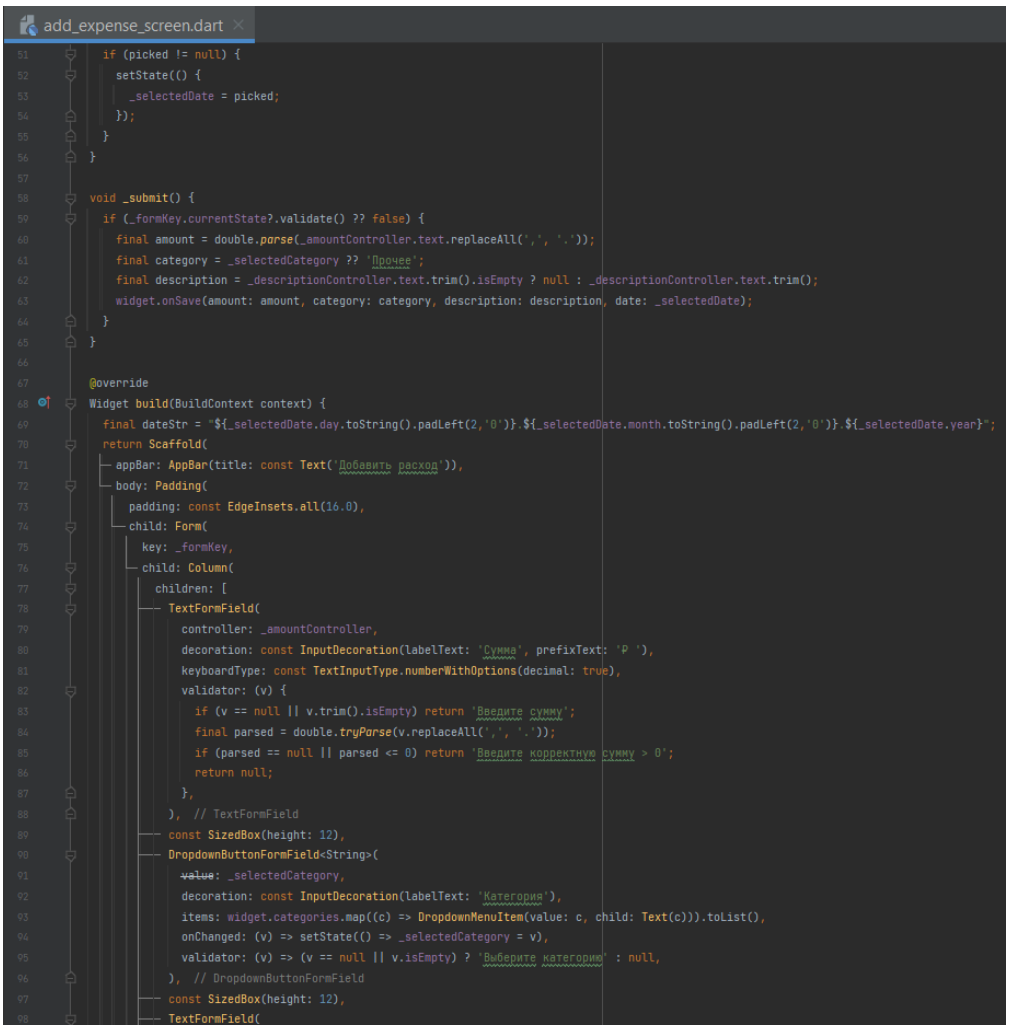


Рисунок 6 – Продолжение реализация экрана добавления расхода

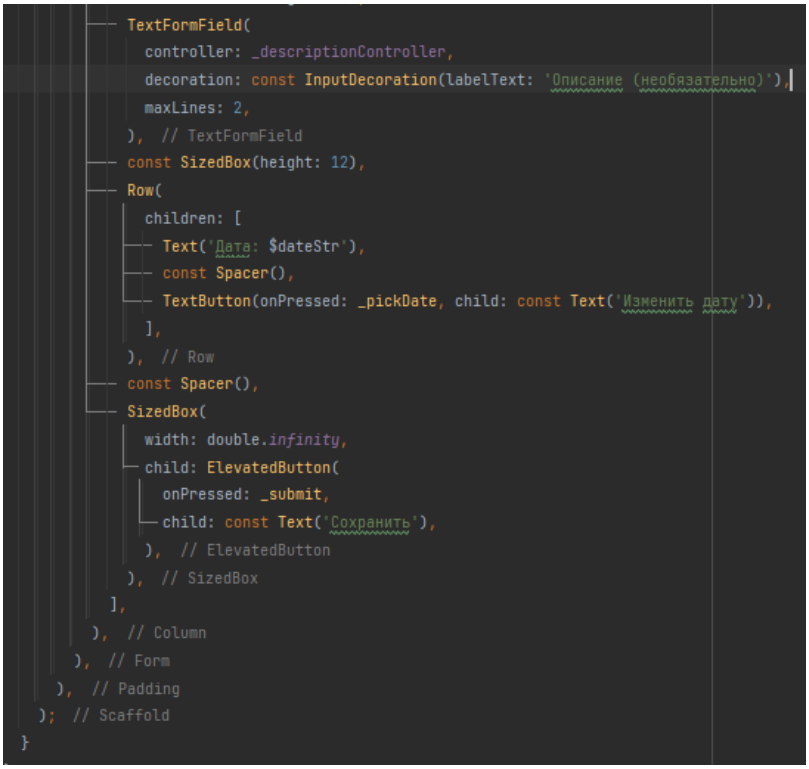


Рисунок 7 – Продолжение реализация экрана добавления расхода

## 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')}.${expense.date.year
    return Card(
      margin: const EdgeInsets.symmetric(horizontal: 8, vertical: 4),
      child: ListTile(
        onTap: onTap,
        title: Text(
          expense.category,
          style: const TextStyle(fontWeight: FontWeight.bold),
        ), // Text
        subtitle: Text(expense.description ?? dateStr),
        trailing: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('${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
    ); // Card
  }
}
```

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.

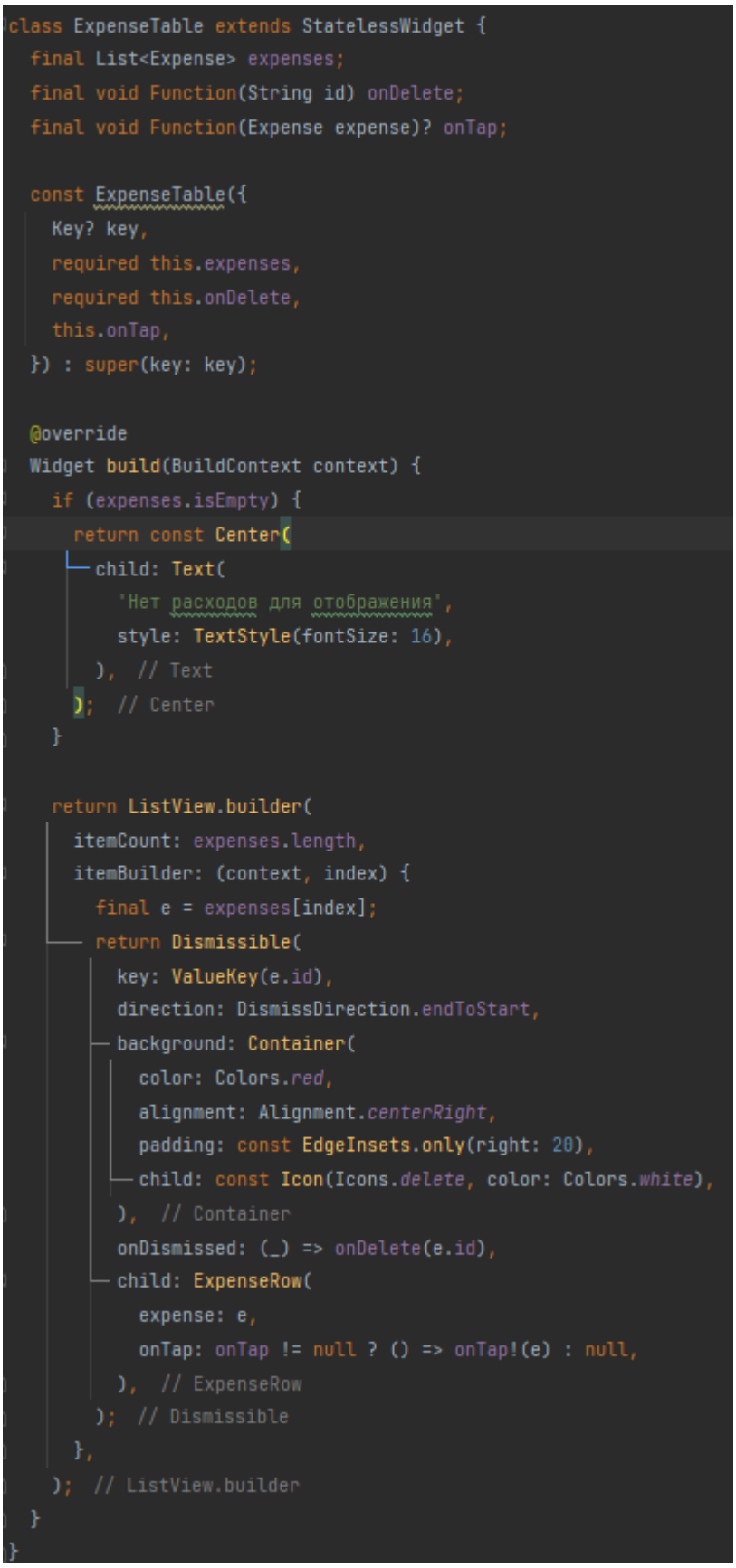


Рисунок 9 – Реализация виджета таблицы расходов

**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.

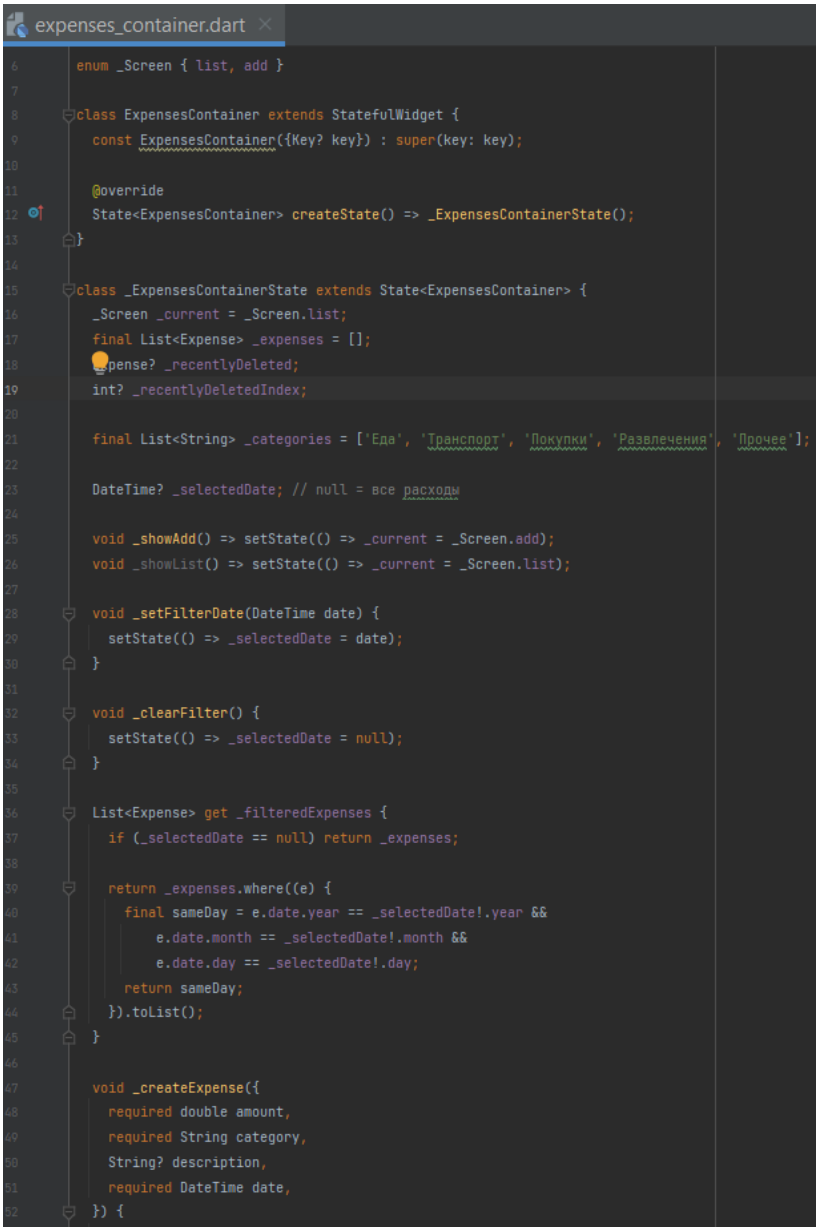


Рисунок 10 – Реализация основной логики приложения


Рисунок 11 – Продолжение реализация основной логики приложения

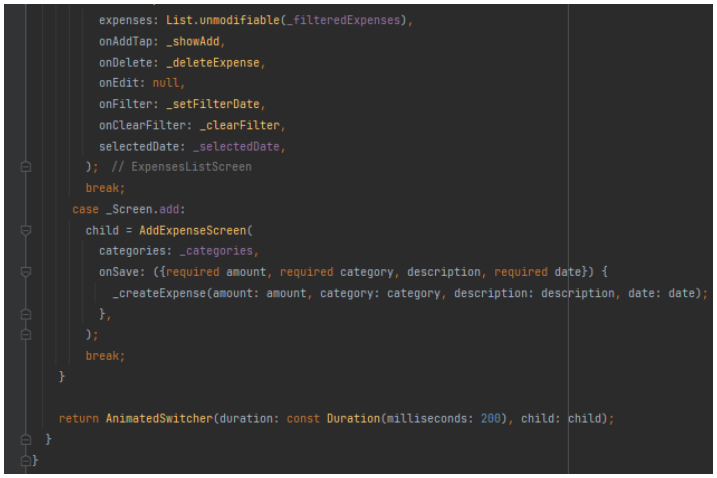


Рисунок 12 – Продолжение реализация основной логики приложения

## 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».

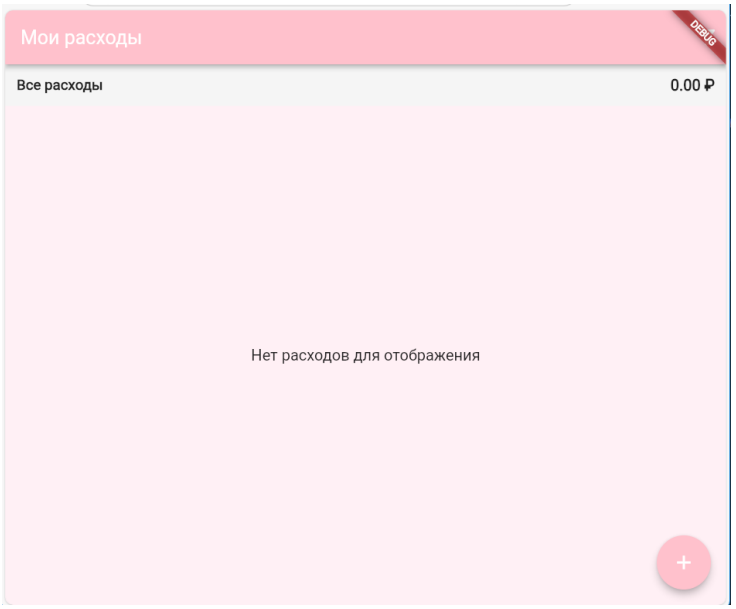


Рисунок 13 – Список до добавления расхода

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


Рисунок 14 – Экран добавления расходов

Внесем данные для нового расхода (Рисунок 15).

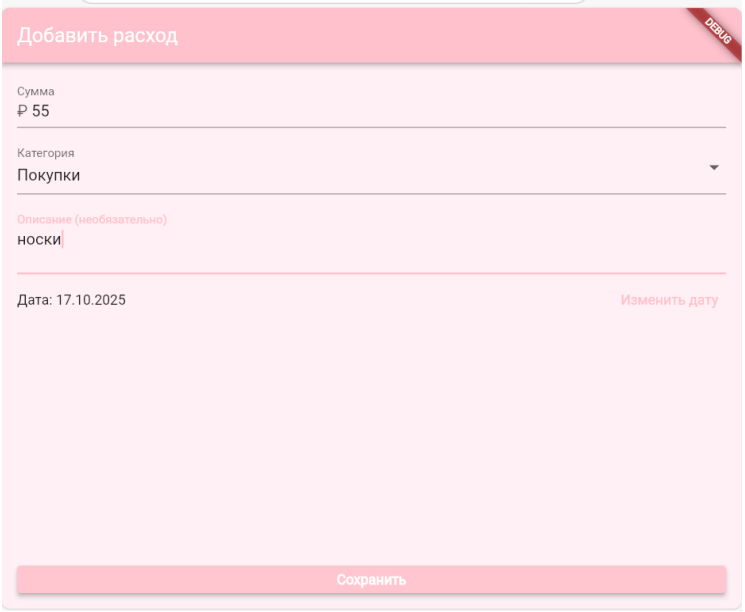


Рисунок 15 – Заполнение данных расхода

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

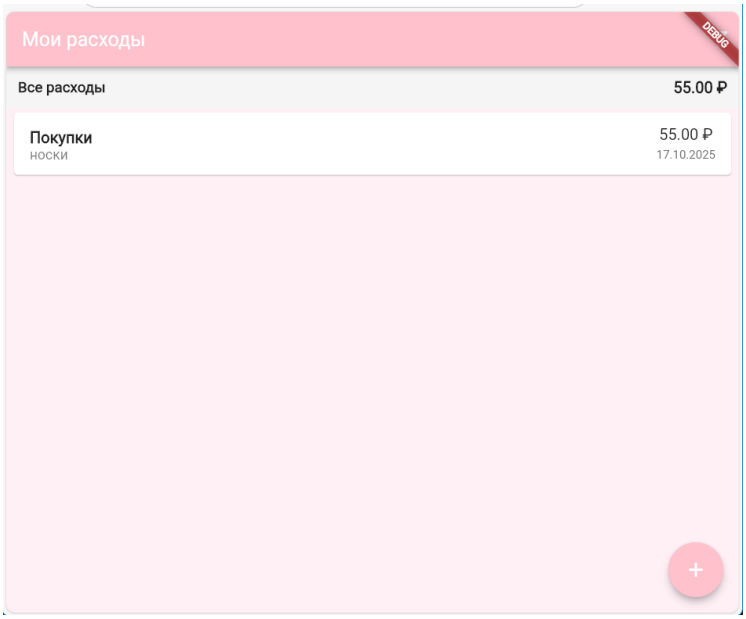


Рисунок 16 – Список после добавления нового расхода

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

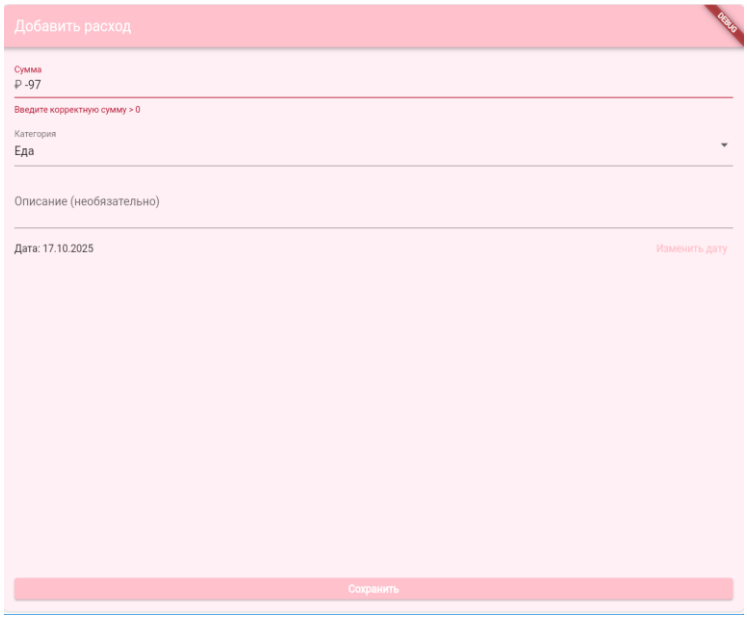


Рисунок 17 – Некорректный ввод суммы

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

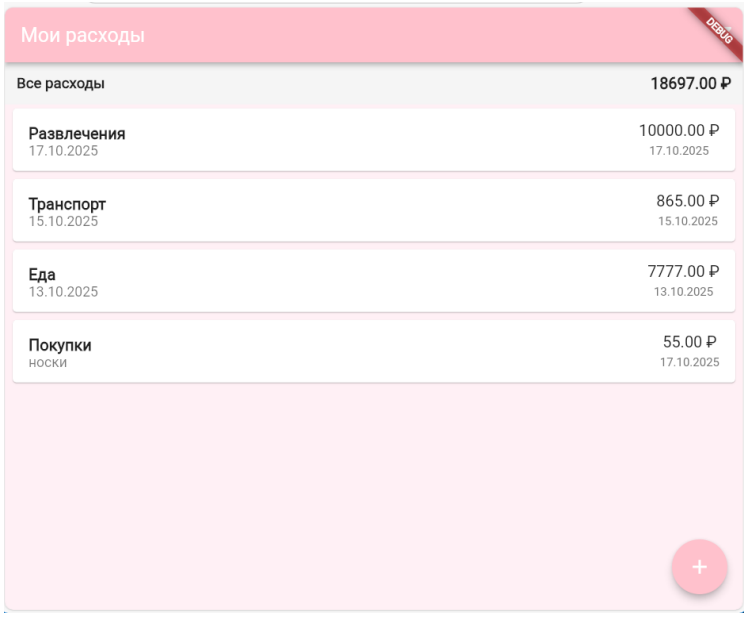


Рисунок 18 – Список до фильтрации

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

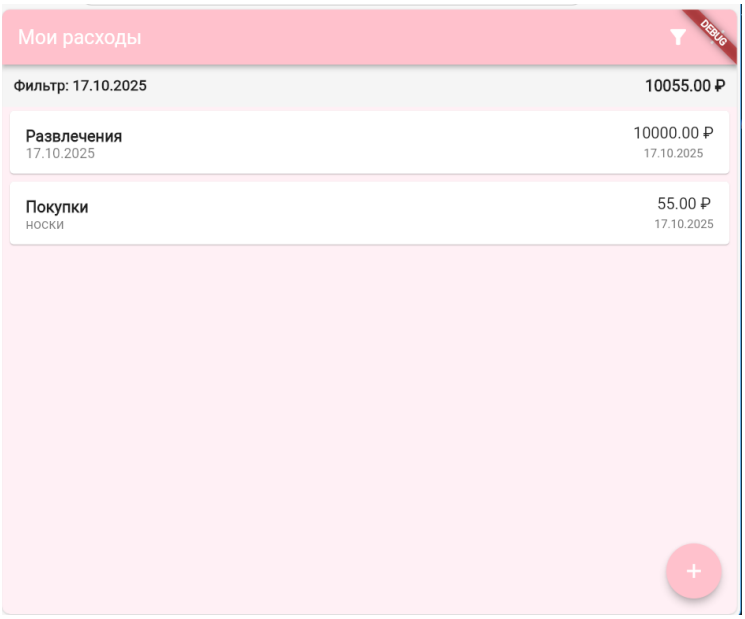


Рисунок 19 – Список после фильтрации

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

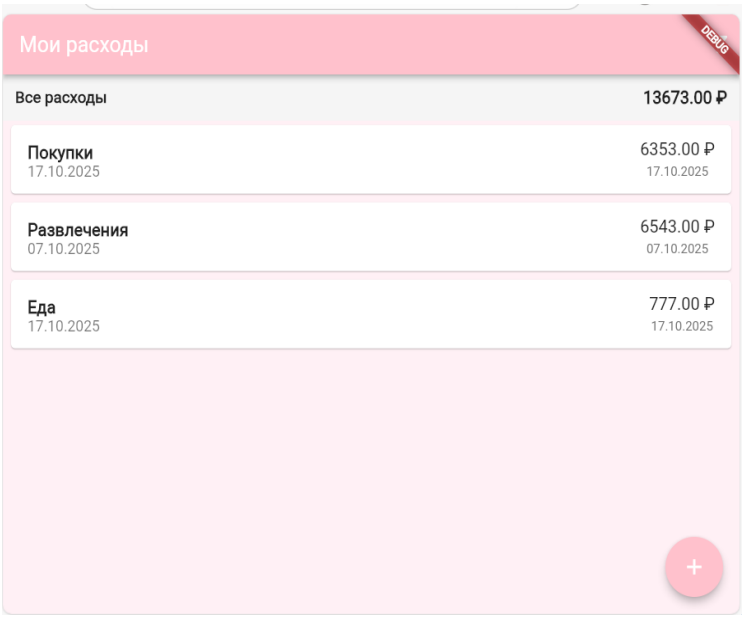


Рисунок 20 – Список до удаления

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

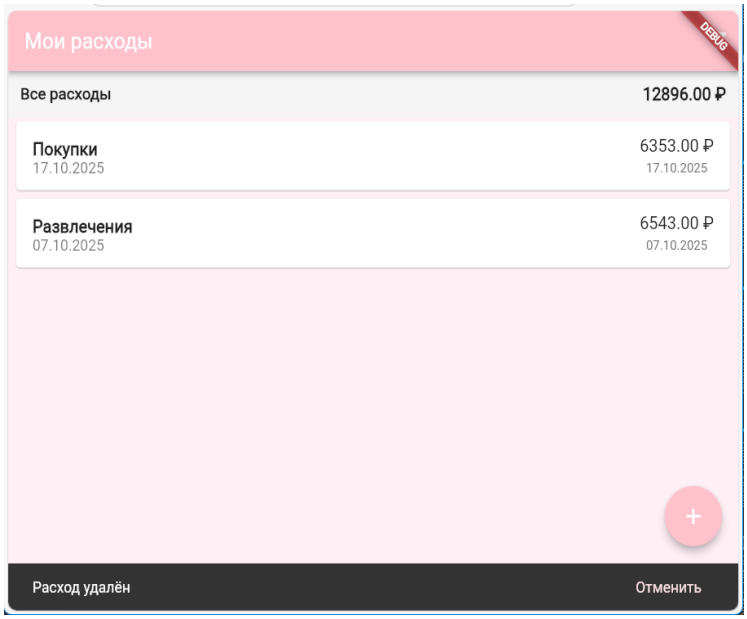


Рисунок 21 – Список после удаления

## 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.