

**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**ITMO University**

**ДОМАШНЕЕ ЗАДАНИЕ**

**По дисциплине Программирование**

**Тема работы** Разработка программного обеспечения системы обработки  
данных: «Учет собственных денежных средств»

**Обучающийся** Озеров Антон Александрович

**Факультет** факультет инфокоммуникационных технологий

**Группа** К3123

**Направление подготовки** 11.03.02 Инфокоммуникационные технологии и  
системы связи

**Образовательная программа** Программирование в инфокоммуникационных  
системах

**Обучающийся**

\_\_\_\_\_

(дата)

\_\_\_\_\_

(подпись)

Озеров А.А.

(Ф.И.О.)

**Руководитель**

\_\_\_\_\_

(дата)

\_\_\_\_\_

(подпись)

Казанова П.П.

(Ф.И.О.)

**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**ITMO University**

**ЗАДАНИЕ НА ДОМАШНЕЕ ЗАДАНИЕ**

**По дисциплине Программирование**

**Обучающийся Озеров Антон Александрович**

**Факультет факультет инфокоммуникационных технологий**

**Группа К3123**

**Направление подготовки 11.03.02 Инфокоммуникационные технологии и  
системы связи**

**Образовательная программа Программирование в инфокоммуникационных  
системах**

**Тема курсовой работы Разработка программного обеспечения системы  
обработки данных: «Учет собственных денежных средств»**

**Руководитель курсовой работы Казанова Полина Петровна, Университет  
ИТМО, факультет инфокоммуникационных технологий, тьютор**

**Основные вопросы, подлежащие разработке В рамках домашнего задания  
необходимо разработать приложение с помощью технологий ООП, хранение  
данных осуществить в базе данных, предусмотреть графический интерфейс,  
обработать все возможные ошибки**

**Дата выдачи задания: 14.02.2024**

**Срок предоставления готовой курсовой работы: 08.05.2024**

**Руководитель**

\_\_\_\_\_  
(дата)

\_\_\_\_\_  
(подпись)

**Казанова П.П.**

\_\_\_\_\_  
(Ф.И.О.)

**Задание принял  
к исполнению**

\_\_\_\_\_  
(дата)

\_\_\_\_\_  
(подпись)

**Озеров А.А.**

\_\_\_\_\_  
(Ф.И.О.)

# СОДЕРЖАНИЕ

Стр.

<b>ВВЕДЕНИЕ .....</b>	<b>4</b>
<b>1 ДИАГРАММЫ И СХЕМЫ.....</b>	<b>5</b>
1.1 Диаграмма классов .....	5
1.2 Схема быза данных .....	5
<b>2 СОЗДАНИЕ ПРОЕКТА.....</b>	<b>7</b>
2.1 Файловая структура проекта и используемые модули .....	7
2.2 Подключение модулей и файлов .....	7
2.3 Создание дизайна системы.....	9
<b>3 ФУНКЦИОНАЛ ПРОГРАММЫ .....</b>	<b>10</b>
3.1 Добавление транзакции .....	10
3.2 Удаление транзакции .....	11
3.3 Добавление категории .....	12
3.4 Удаление категории .....	13
3.5 Фильтр и сортировка транзакций .....	13
<b>4 РАБОТА С БАЗОЙ ДАННЫХ .....</b>	<b>15</b>
4.1 Добавление и удаление транзакции на уровне базы данных	15
4.2 Добавление и удаление категории на уровне базы данных..	15
4.3 Получение транзакций и категорий на уровне базы данных	17
<b>5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ .....</b>	<b>18</b>
5.1 Запуск приложения и основной сценарий .....	18
5.2 Отлов ошибок .....	21
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>22</b>
<b>ПРИЛОЖЕНИЕ А Диаграмма классов .....</b>	<b>23</b>
<b>ПРИЛОЖЕНИЕ Б Дизайн системы.....</b>	<b>26</b>

## ВВЕДЕНИЕ

Актуальность создания приложения системы обработки данных: «Учет собственных денежных средств» в данном контексте обусловлена необходимостью эффективного контроля и управления личными финансами. В условиях повседневных финансовых операций все чаще возникает потребность в учете доходов и расходов, а также в планировании и контроле трат. Такое приложение поможет пользователям не только вести учет своих финансов, но и управлять ими на более осознанном уровне, соблюдая бюджет и избегая лишних трат.

Предметная область приложения включает в себя учет и контроль личных финансов пользователя. Это подразумевает фиксацию поступлений средств, отслеживание баланса кошелька, добавление и удаление трат, а также фильтрацию и просмотр трат по дате и категории.

Цель создания приложения – предоставить пользователю инструмент для организации и контроля своих финансов, повысить финансовую грамотность и помочь в управлении денежными средствами.

Задачи приложения будут следующие:

1. Реализация возможности внесения денег в кошелек только в определенный день месяца (поступление)
2. Просмотр баланса кошелька
3. Добавление трат
4. Удаление трат из списка покупок (возврат товара)
5. Просмотр трат по дате (фильтрация данных)
6. Просмотр трат по категории (фильтрация данных)

Таким образом, создание данного приложения позволит пользователям более целенаправленно управлять своими личными финансами, осуществлять контроль над тратами и планировать свой бюджет с учетом доступных средств.

# 1 ДИАГРАММЫ И СХЕМЫ

## 1.1 Диаграмма классов

В программе существует 9 классов: `DateBase` - класс для взаимодействия с базой данных (рис. A.1), `Transaction` и `Category` - классы для работы с таблицами базы данных (рис. A.1), `Ui_MainWindow` - класс, в котором объявляются все основные виджеты (рис. A.2), `MainWindow` - класс, в котором прописывается основной функционал приложения (рис. A.2), `DeleteCategory` - класс объявления окна для удаления категории (рис. A.3), `AddCategory` - класс объявления окна для добавления новой категории (рис. A.4), `ErrorDialog` - класс объявления окна для информирования об ошибке (рис. A.5), `CompleteDialog` - класс объявления окна для информирования об успешном добавлении новой транзакции (рис. A.5).

## 1.2 Схема база данных

База данных содержит в себе 2 таблицы: `Categories` и `Products`. Диаграмма классов представлена на рисунке 1.1.

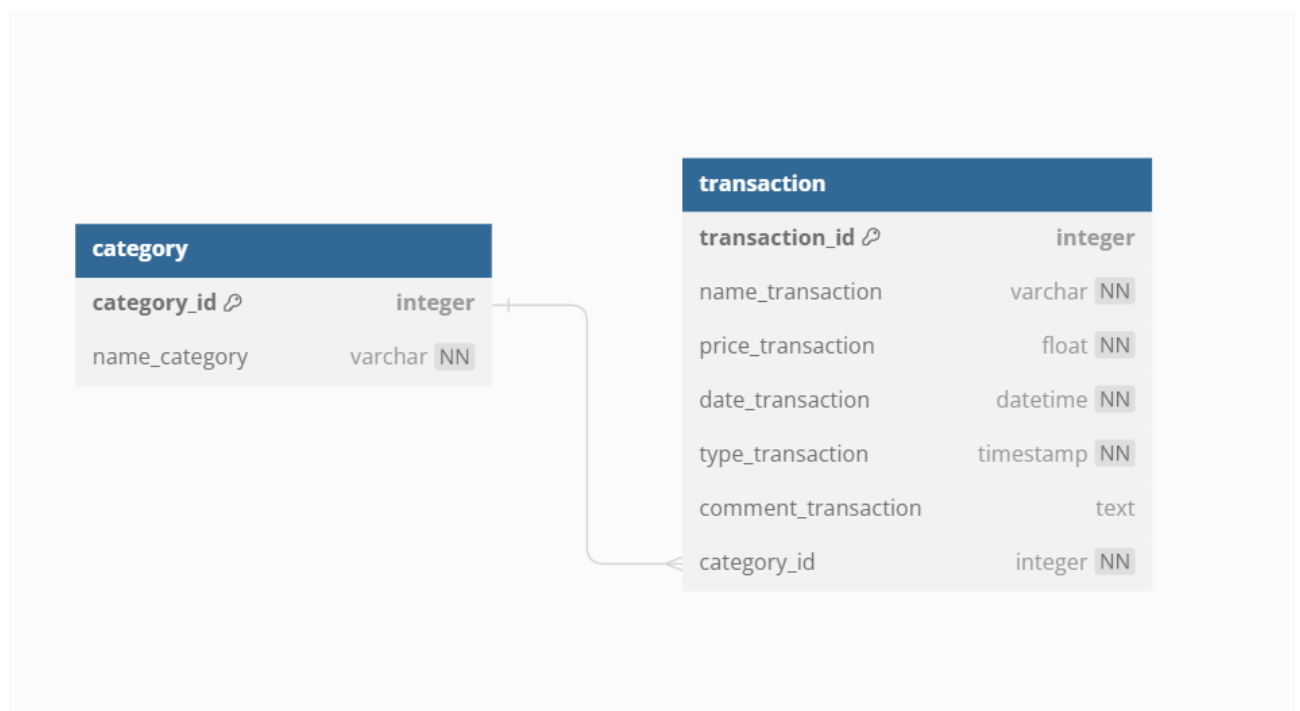


Рисунок 1.1 — Схема базы данных

В первой таблице хранятся: уникальный номер категории и ее название. Во второй хранится информация о транзакциях. А именно: уникальный номер транзакции, название, цена, дата совершения транзакции, тип транзакции (доход или расход), комментарий к транзакции и уникальный номер категории (указывается при добавлении расхода в приложении, к которой относится данный товар).

## 2 СОЗДАНИЕ ПРОЕКТА

### 2.1 Файловая структура проекта и используемые модули

Файловая структура проекта состоит из нескольких исполняемых файлов питона, главным из которых является `main.py`, базой данных и файлом дизайна интерфейса (рис. 2.1).

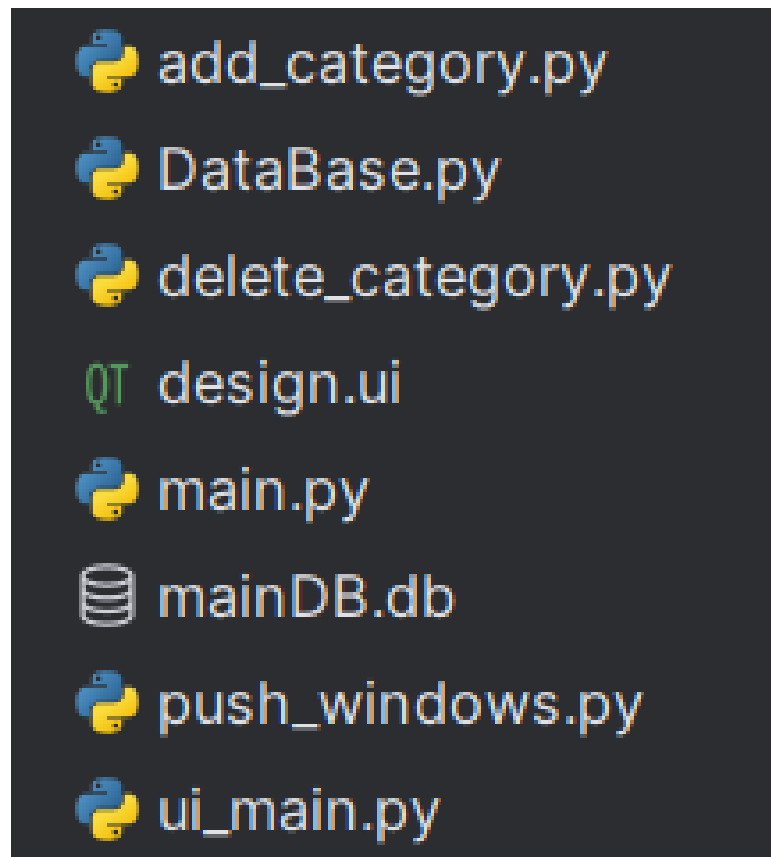
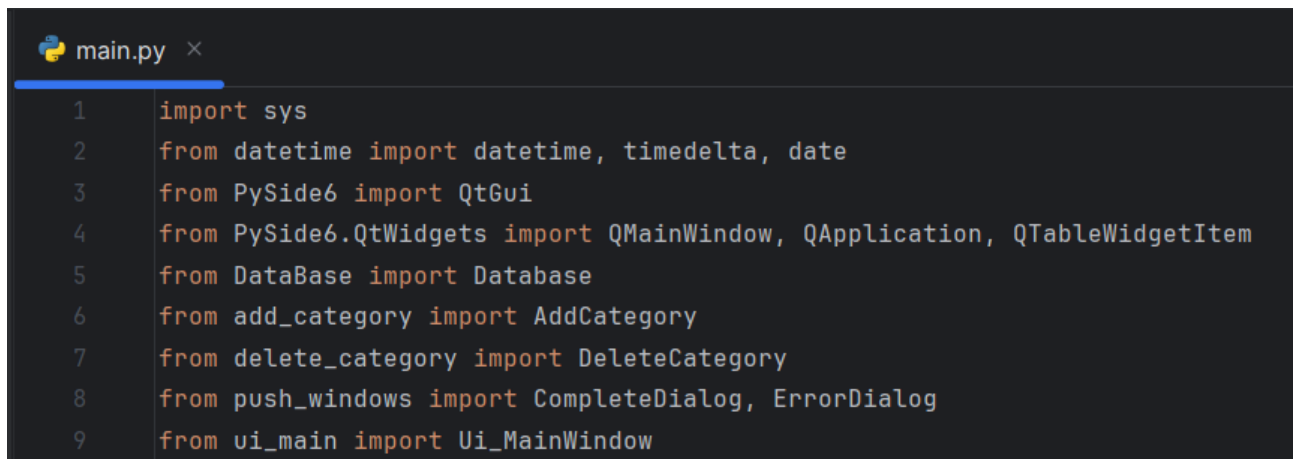


Рисунок 2.1 — Файловая структура проекта

### 2.2 Подключение модулей и файлов

На рисунках 2.2 и 2.3 приведены все используемые библиотеки и их модули. В остальных файлах идёт повторение.



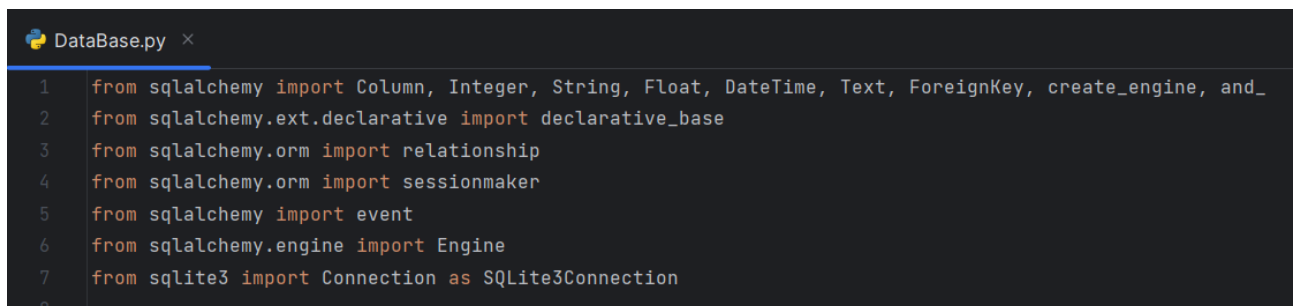
```

main.py x
1  import sys
2  from datetime import datetime, timedelta, date
3  from PySide6 import QtGui
4  from PySide6.QtWidgets import QMainWindow, QApplication, QTableWidgetItem
5  from DataBase import Database
6  from add_category import AddCategory
7  from delete_category import DeleteCategory
8  from push_windows import CompleteDialog, ErrorDialog
9  from ui_main import Ui_MainWindow

```

Рисунок 2.2 — Используемые библиотеки в основном файле

Так как по сравнению с консольным приложением, в графической версии будет больше библиотек и зависимостей, то для данной программы вся графическая часть была построена на основе библиотеки PySide6, которая имеет крайне большой спектр возможностей. Она содержит в себе все необходимые модули для работы с графическим интерфейсом и базой данных. В main.py, помимо PySide6, были подключены также и другие библиотеки, такие как sys для работы с оболочкой приложения и datetime для работы с датами.



```

DataBase.py x
1  from sqlalchemy import Column, Integer, String, Float, DateTime, Text, ForeignKey, create_engine, and_
2  from sqlalchemy.ext.declarative import declarative_base
3  from sqlalchemy.orm import relationship
4  from sqlalchemy.orm import sessionmaker
5  from sqlalchemy import event
6  from sqlalchemy.engine import Engine
7  from sqlite3 import Connection as SQLite3Connection
8

```

Рисунок 2.3 — Для работы с базой данных

Для работы с базой данных SQLite использовался ORM SQLAlchemy, которая взаимодействует с SQLite. Выбрана именно система управления базами данных в силу её легковесности и простоты.



## 2.3 Создание дизайна системы

Перед началом разработки был создан статичный дизайн программы для дальнейшей работы. В качестве графического редактора PySide6 имеет инструмент QtDesigner. В нем был создан главный экран приложения и окно взаимодействия с транзакциями (рис. Б.1 и Б.2) в файле `design.ui`. В дальнейшем этот файл, с помощью PySide6, был преобразован в файл `ui_main.py`. Все остальные окна были прописаны вручную (рис. Б.3, Б.4 и Б.5).

### 3 ФУНКЦИОНАЛ ПРОГРАММЫ

#### 3.1 Добавление транзакции

Одной из основных функций программы является добавление новой транзакции. Реализация показана на рисунке 3.1:

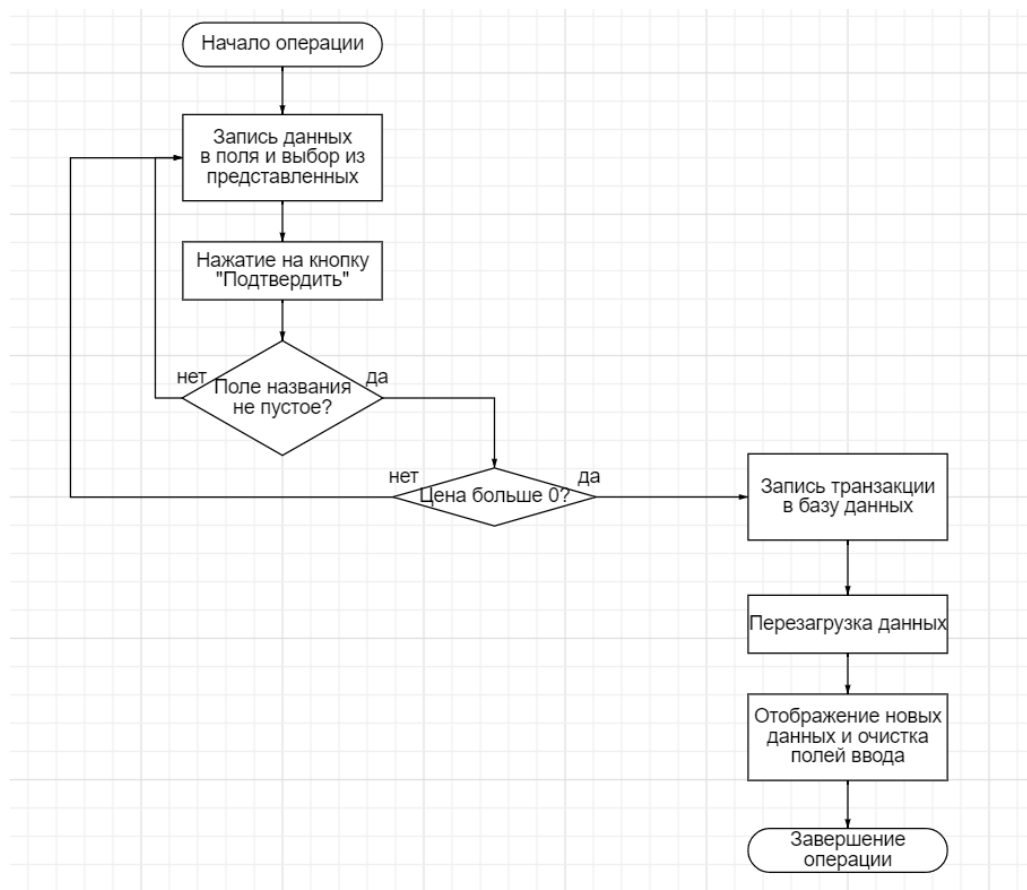


Рисунок 3.1 — Схема добавление новой транзакции

При входе в приложение, пользователь видит основное окно, где есть отдельное поле, в котором пользователь заполняет такие данные, как: название транзакции, цена, тип (доход/расход), категория (если в типе выбран расход), дата транзакции и комментарий (при необходимости). В стоимость пользователь не сможет ввести ничего, кроме числа, так как в QtDesigner для чисел есть специальное поле, которое ограничивает вводимые данные на уровне моделирования системы. Также было введено ограничение на числа от 0 до 10000000000 (рис. 3.2).

```

def commit_transaction(self):
    name = self.product_name_lineEdit.text().strip()
    price = self.price_doubleSpinBox.value()
    type_ = self.type_comboBox.currentData()
    category = self.category_comboBox.currentData()
    date_ = self.date_comboBox.currentData()
    comment = self.comment_textEdit.toPlainText().strip()
    if name.strip() and price != 0:
        self.DB.add_transaction(name, price, type_, category, date_, comment)
        self.update_fields()
        dialog = CompleteDialog()
    else:
        dialog = ErrorDialog()
    dialog.exec()

```

Рисунок 3.2 — Добавление новой транзакции

При нажатии на кнопку “Подтвердить”, следуют стандартные функции обновления экрана и внесение информации в базу данных. Баланс также обновляется автоматически после каждого действия пользователя.

### 3.2 Удаление транзакции

Удаление транзакции происходит при выборе пользователя нужной в таблице (рис. 3.3).

```

def delete_transaction(self):
    if self.tableWidget.selectedItems():
        row = self.tableWidget.selectedItems()[0].row()
        id_ = self.tableWidget.takeItem(row, 0).text()
        self.DB.delete_transaction(id_)
        self.update_fields()

```

Рисунок 3.3 — Удаление транзакции

Функция находит выбранную транзакцию и вызывает функцию удаления её на уровне базы данных.

### 3.3 Добавление категории

Добавление категории происходит при выборе пользователя кнопки “НОВАЯ КАТЕГОРИЯ” (рис. 3.4).

```
def add_category(self):  
    dialog = AddCategory()  
    dialog.exec()  
    self.update_fields()
```

Рисунок 3.4 — Добавление категории - вызов окна

Выводится диалоговое окно, где пользователь вводит название и, при подтверждении, нажатием на кнопку, вызывается функция создания её на уровне базы данных (рис. 3.5).

```
def bt_clicked(self):  
    if not self.ledit.text().strip():  
        dialog = ErrorDialog()  
        dialog.exec()  
    else:  
        self.DB.add_category(self.ledit.text())  
        self.close()
```

Рисунок 3.5 — Добавление категории - обращение к базе данных

### 3.4 Удаление категории

Удаление категории происходит при выборе пользователя кнопки “УДАЛИТЬ КАТЕГОРИЮ” (рис. 3.6).

```
def delete_category(self):  
    dialog = DeleteCategory()  
    dialog.exec()  
    self.update_fields()
```

Рисунок 3.6 — Удаление категории - вызов окна

Выводится диалоговое окно, где пользователь выбирает одну из доступных категорий для удаления и, при подтверждении, нажатием на кнопку, вызывается функция удаления её на уровне базы данных (рис. 3.7).

```
def bt_clicked(self):  
    self.DB.delete_category(self.ledit.currentData())  
    self.close()
```

Рисунок 3.7 — Удаление категории - обращение к базе данных

### 3.5 Фильтр и сортировка транзакций

Фильтрация и сортировка происходит при выборе нужных пунктов в окне транзакций (рис. 3.8).

```

def load_TableWidget(self):
    self.tableWidget.clear()
    date_ = self.filter_date_comboBox.currentData()
    type_ = self.filter_type_comboBox.currentData()
    category = self.filter_categories_comboBox.currentData()
    transactions = self.DB.get_transactions(date_transaction=date_, type_transaction=type_, category_id=category)
    transactions.sort(key=lambda x: x.price_transaction, reverse=self.sort_comboBox.currentData())
    self.tableWidget.setRowCount(len(transactions))
    self.tableWidget.setColumnCount(7)

    for row, transaction in enumerate(transactions):
        if transaction.type_transaction == 'income':
            type_ = 'Доход'
        else:
            type_ = 'Расход'
        self.tableWidget.setItem(row, 0, QTableWidgetItem(str(transaction.transaction_id)))
        self.tableWidget.setItem(row, 1, QTableWidgetItem(str(transaction.name_transaction)))
        self.tableWidget.setItem(row, 2, QTableWidgetItem(str(transaction.price_transaction)))
        self.tableWidget.setItem(row, 3, QTableWidgetItem(type_))
        self.tableWidget.setItem(row, 4, QTableWidgetItem(str(transaction.category.name_category)))
        self.tableWidget.setItem(row, 5, QTableWidgetItem(str(transaction.date_transaction.strftime('%d-%m-%Y'))))
        self.tableWidget.setItem(row, 6, QTableWidgetItem(str(transaction.comment_transaction)))

    self.tableWidget.setColumnHidden(0, True)
    self.tableWidget.setHorizontalHeaderLabels(['id_hidden', 'Название', 'Цена', 'Тип', 'Категория', 'Дата',
                                                'Комментарий'])

```

Рисунок 3.8 — Фильтр и сортировка транзакций

После выбора некоторых фильтров или типа сортировки, обновляется таблица.

## 4 РАБОТА С БАЗОЙ ДАННЫХ

### 4.1 Добавление и удаление транзакции на уровне базы данных

На рисунке 4.1 представлены функции добавления и удаления транзакции на уровне базы данных.

```
1 usage
def add_transaction(self, name, price, type_, category_id, date_, comment):
    category = self.session.query(Category).filter_by(category_id=category_id).first()
    new_transaction = Transaction(name_transaction=name, price_transaction=price, date_transaction=date_,
                                  type_transaction=type_, comment_transaction=comment,
                                  category_id=category.category_id)
    self.session.add(new_transaction)
    self.session.commit()

1 usage
def delete_transaction(self, transaction_id):
    transaction_to_delete = self.session.query(Transaction).filter_by(transaction_id=transaction_id).first()
    if transaction_to_delete:
        self.session.delete(transaction_to_delete)
        self.session.commit()
```

Рисунок 4.1 — Добавление и удаление транзакции

Функция добавления транзакции позволяет пользователям внести новые данные о транзакции в базу данных. При этом данные вводятся на начальном экране приложения.

Функция удаления транзакции дает возможность пользователям удалить определенную транзакцию из базы данных после клика на нужную в таблице.

### 4.2 Добавление и удаление категории на уровне базы данных

На рисунке 4.2 представлены функции добавления и удаления категории на уровне базы данных.

```
def add_category(self, category_name):
    new_category = Category(name_category=category_name)
    self.session.add(new_category)
    self.session.commit()

def delete_category(self, category_id):
    category_to_delete = self.session.query(Category).filter_by(category_id=category_id).first()
    if category_to_delete:
        self.session.delete(category_to_delete)
        self.session.commit()
```

Рисунок 4.2 — Добавление и удаление категории

Функция добавления категории позволяет пользователям внести информацию о новой категории в базу данных

Функция удаления категории дает возможность пользователям удалить выбранную в выпадающем списке категорию из базы данных, кроме категории “Разное” (рис. Б.4).



### 4.3 Получение транзакций и категорий на уровне базы данных

На рисунке 4.3 представлены функции получения транзакций и категорий на уровне базы данных.

```
4 usages
def get_categories(self, **kwargs):
    if kwargs:
        filters = []
        for key, value in kwargs.items():
            filters.append(getattr(Category, key) == value)
        query = self.session.query(Category).filter(and_(*filters)).all()
    else:
        query = self.session.query(Category).all()
    return query

4 usages
def get_transactions(self, **kwargs):
    if kwargs:
        filters = []
        for key, value in kwargs.items():
            if value != -1:
                if key != 'date_transaction':
                    filters.append(getattr(Transaction, key) == value)
                else:
                    filters.append(getattr(Transaction, key) >= value)
        query = self.session.query(Transaction).filter(and_(*filters)).all()
    else:
        query = self.session.query(Transaction).all()
    return query
```

Рисунок 4.3 — Получение транзакций и категорий

Обе функции предназначены для предоставления пользователю доступа к данным о транзакциях и категориях, сохраненных в базе данных с теми аргументами, которые мы указали в фильтрах, с целью удобного и эффективного управления финансовыми данными.

## 5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

### 5.1 Запуск приложения и основной сценарий

При запуске приложения пользователя встречает окно, представленное на рисунке 5.1.

Учет собственных денежных средств

Новая транзакция

Название:

Цена:  ^ v

Тип:  v

Категория:  v

Дата:  v

Комментарий:

Подтвердить

БАЛАНС ЗА МЕСЯЦ

Баланс: 9180.98

Доходы: 10001.09

Расходы: 820.11

ТРАНЗАКЦИИ

УДАЛИТЬ КАТЕГОРИЮ

НОВАЯ КАТЕГОРИЯ

Рисунок 5.1 — Получение транзакций и категорий

При нажатии на “УДАЛИТЬ КАТЕГОРИЮ”, пользователь увидит всплывающее окно (рис. 5.2).

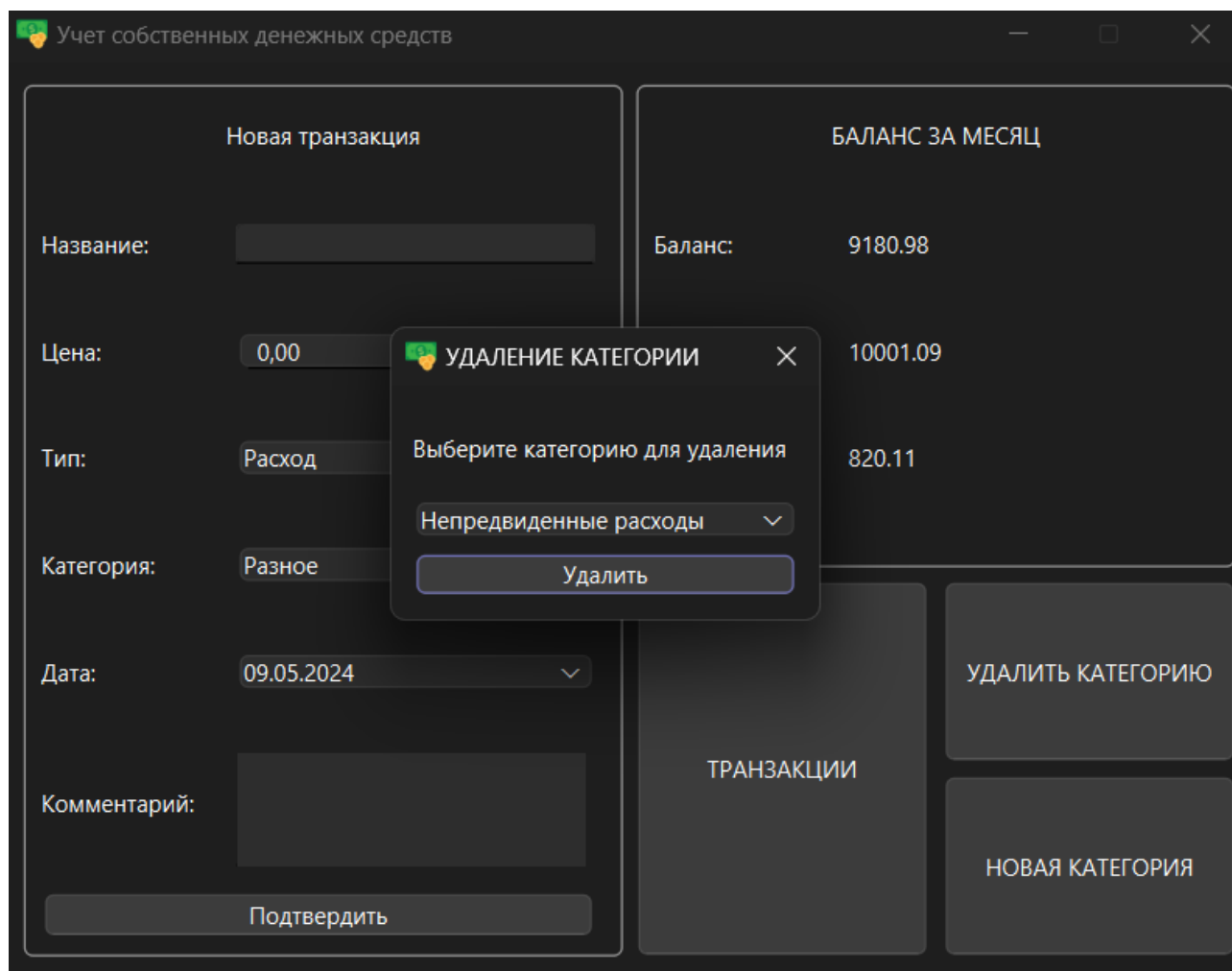


Рисунок 5.2 — Получение транзакций и категорий

При нажатии на “НОВАЯ КАТЕГОРИЯ” пользователь увидит всплывающее окно (рис. 5.3).

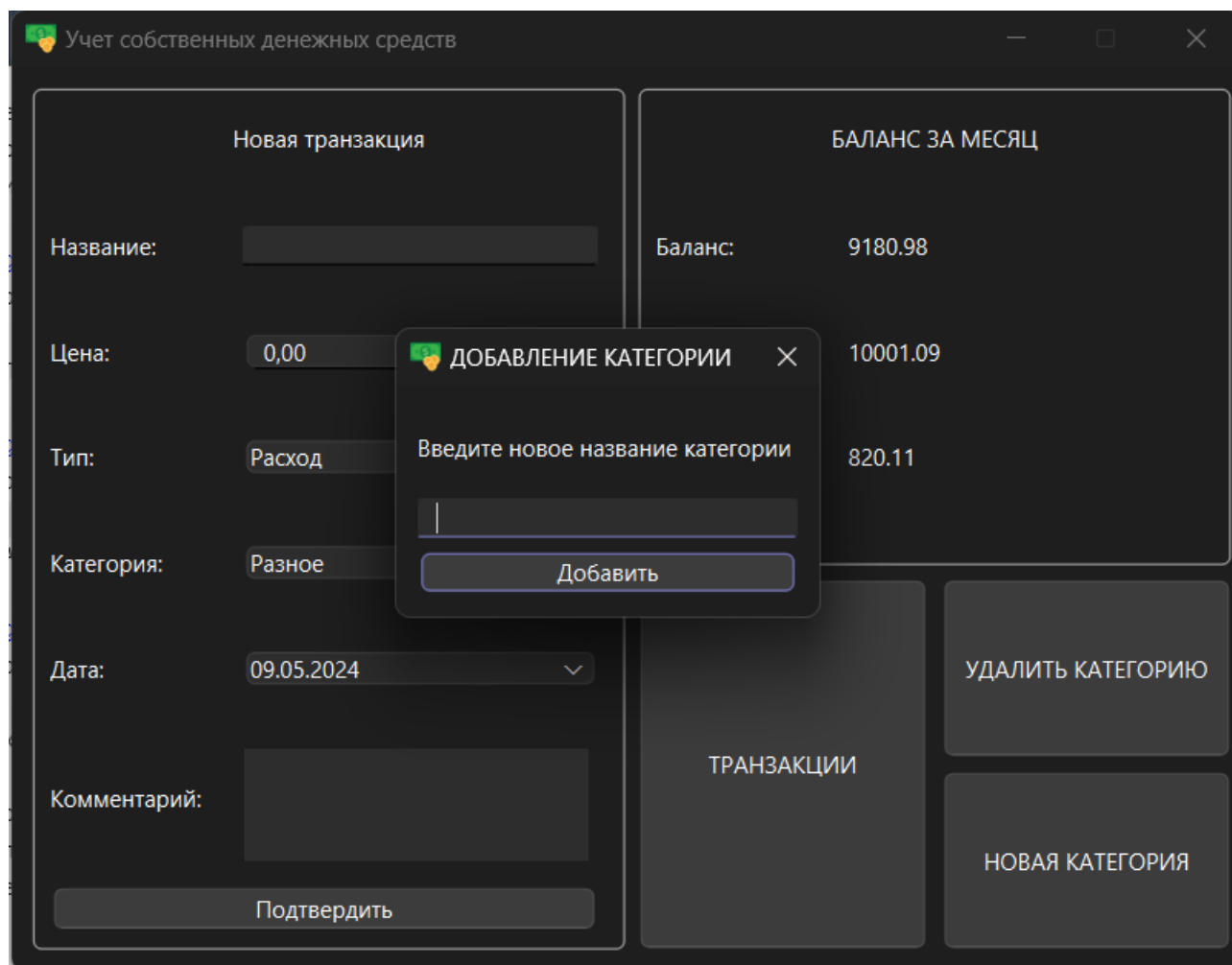


Рисунок 5.3 — Получение транзакций и категорий

При нажатии на “ТРАНЗАКЦИИ” пользователь увидит смену основного окна (рис. 5.4).

Учет собственных денежных средств

Назад

ФИЛЬТР: 

Все категории

За всё время

Доходы/Расходы

Сортировать по цене: 

От min к max

Удалить транзакцию

	Название	Цена	Тип	Категория	Дата	Комментарий
1	душа	0.11	Расход	Разное	01-05-2024	
2	From mum	1.09	Доход	Разное	06-05-2024	
3	Яблоки	120.0	Расход	Продукты	06-05-2024	Вкусно!
4	Бананы	200.0	Расход	Разное	06-05-2024	
5	тортик	500.0	Расход	Разное	08-05-2024	маме на ДР
6	С онлайн-...	10000.0	Доход	Разное	01-05-2024	СПАСИБО!

Рисунок 5.4 — Получение транзакций и категорий

## 5.2 Отлов ошибок

Благодаря PySide6 и его встроенным модулей, в дизайне системы были заранее настроены возможные входные данные, которые может ввести пользователь. Например, в модальном окне в поле цены можно ввести только положительное число, а поле ввода названия ограничено по длине так, чтобы быть в пределах ограничений python’a.

## **ЗАКЛЮЧЕНИЕ**

В ходе данной работы получилось выполнить все поставленные задачи. Было создано приложение с дизайном и базой данных для контроля собственных денежных средств, описан ее функционал и составлена блок-схема одной из основных функций.

## ПРИЛОЖЕНИЕ А

### Диаграмма классов

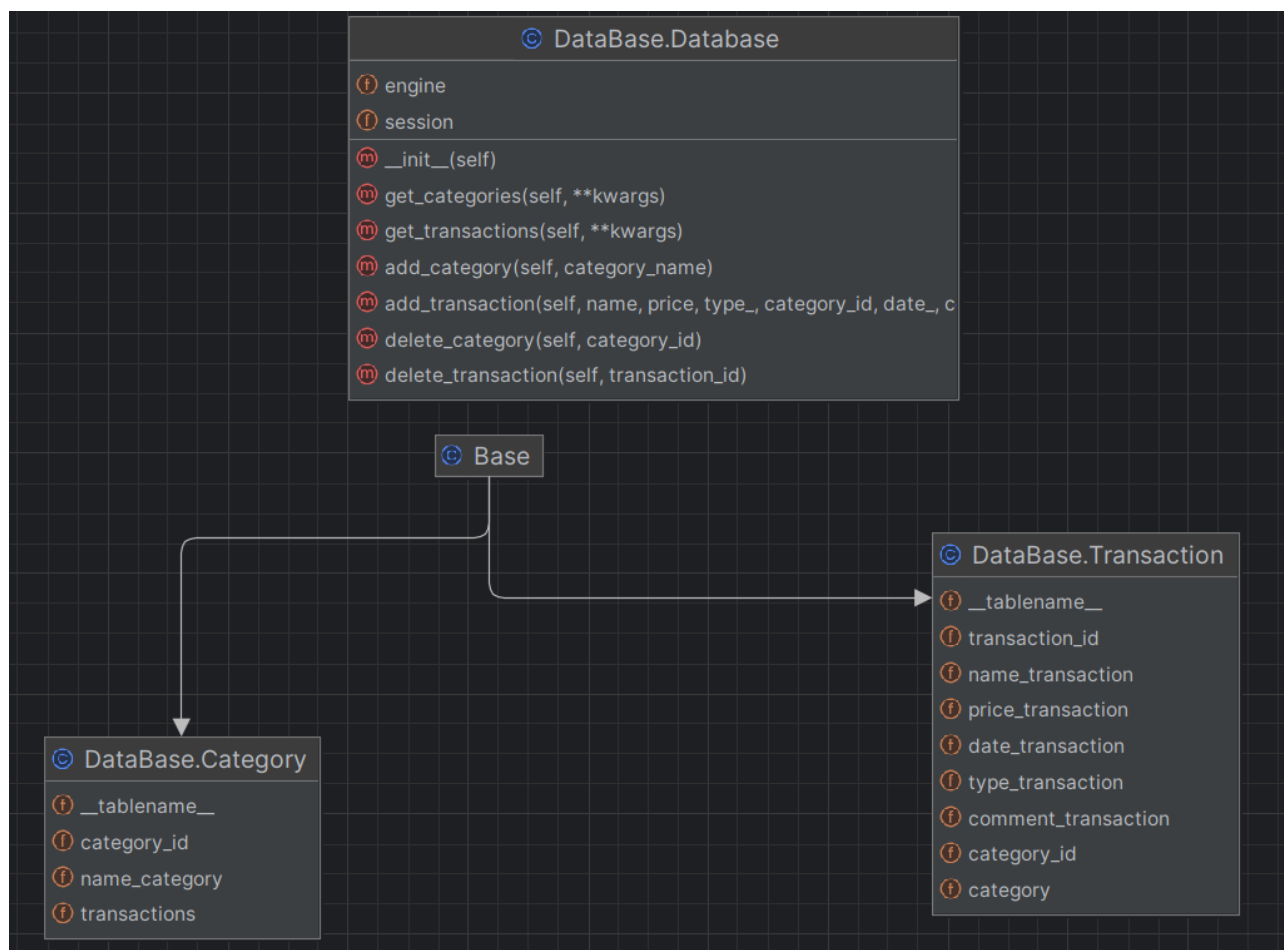


Рисунок А.1 — Диаграмма классов для работы с базой данных

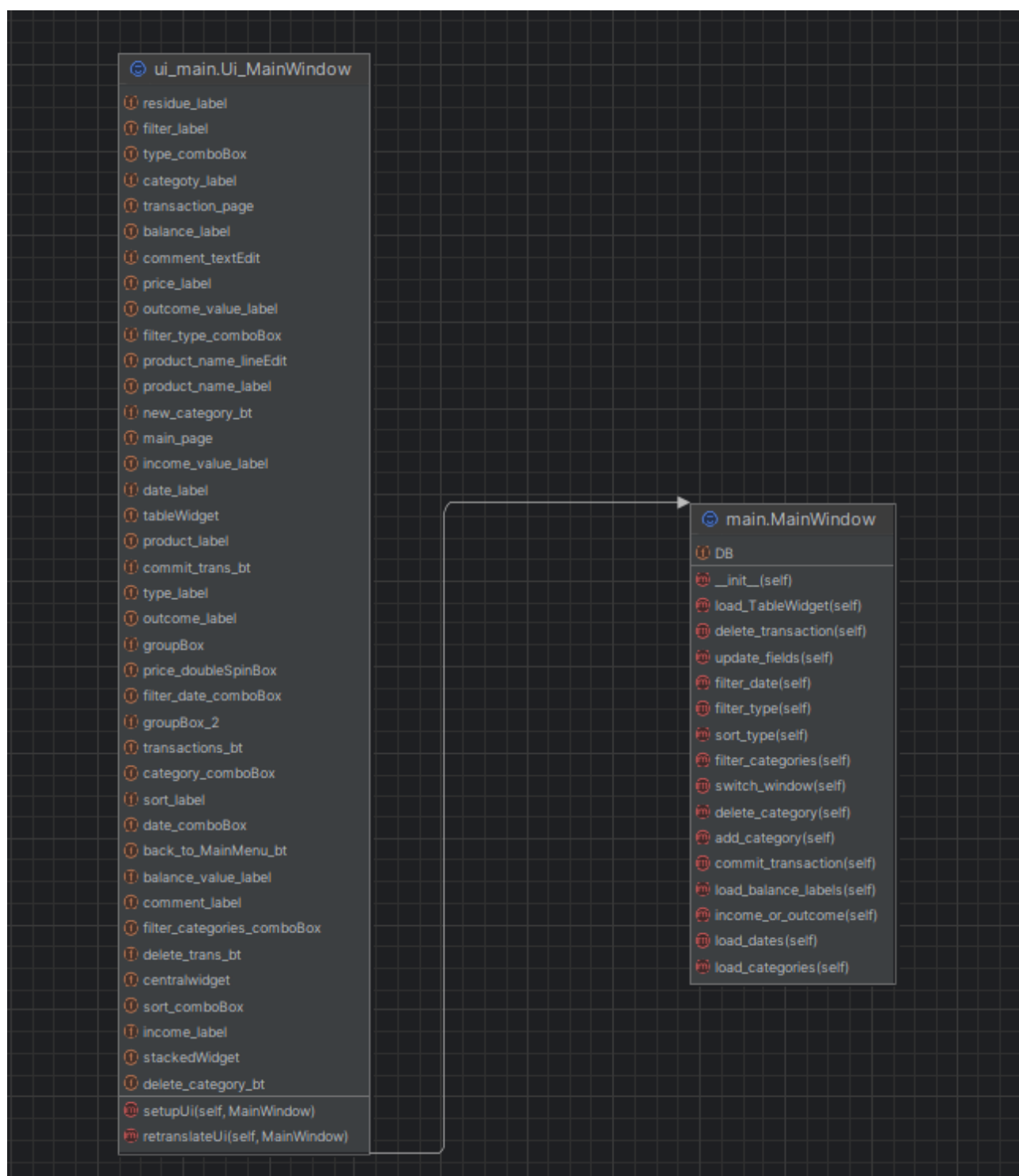


Рисунок А.2 — Диаграмма классов основного окна



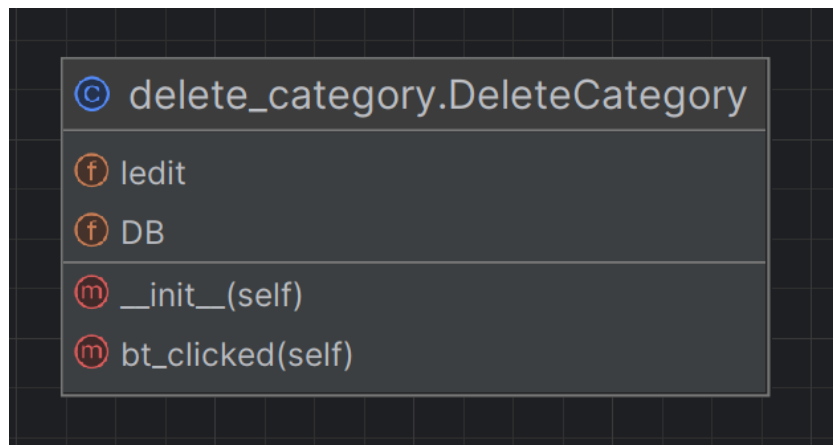


Рисунок А.3 — Диаграмма класса удаления категории

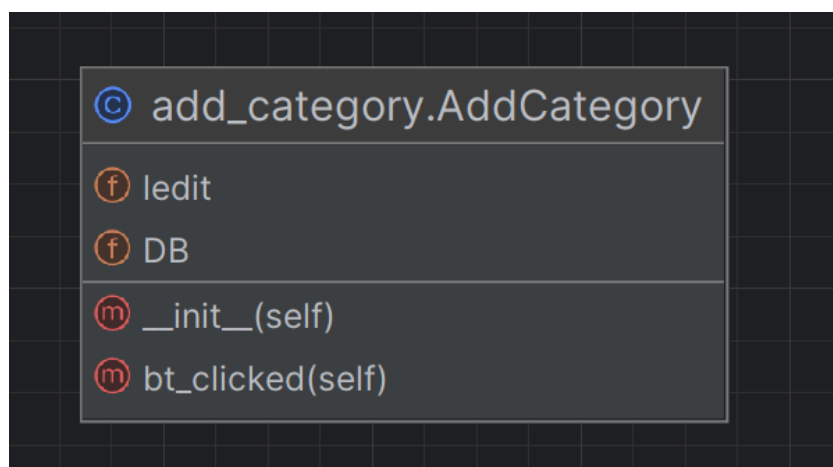


Рисунок А.4 — Диаграмма класса добавления новой категории

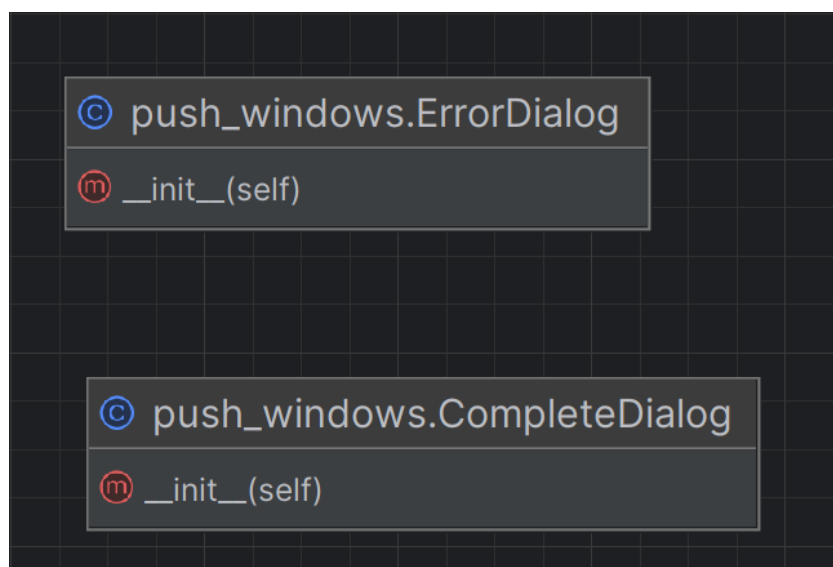


Рисунок А.5 — Диаграмма классов информирования

## ПРИЛОЖЕНИЕ Б

### Дизайн системы

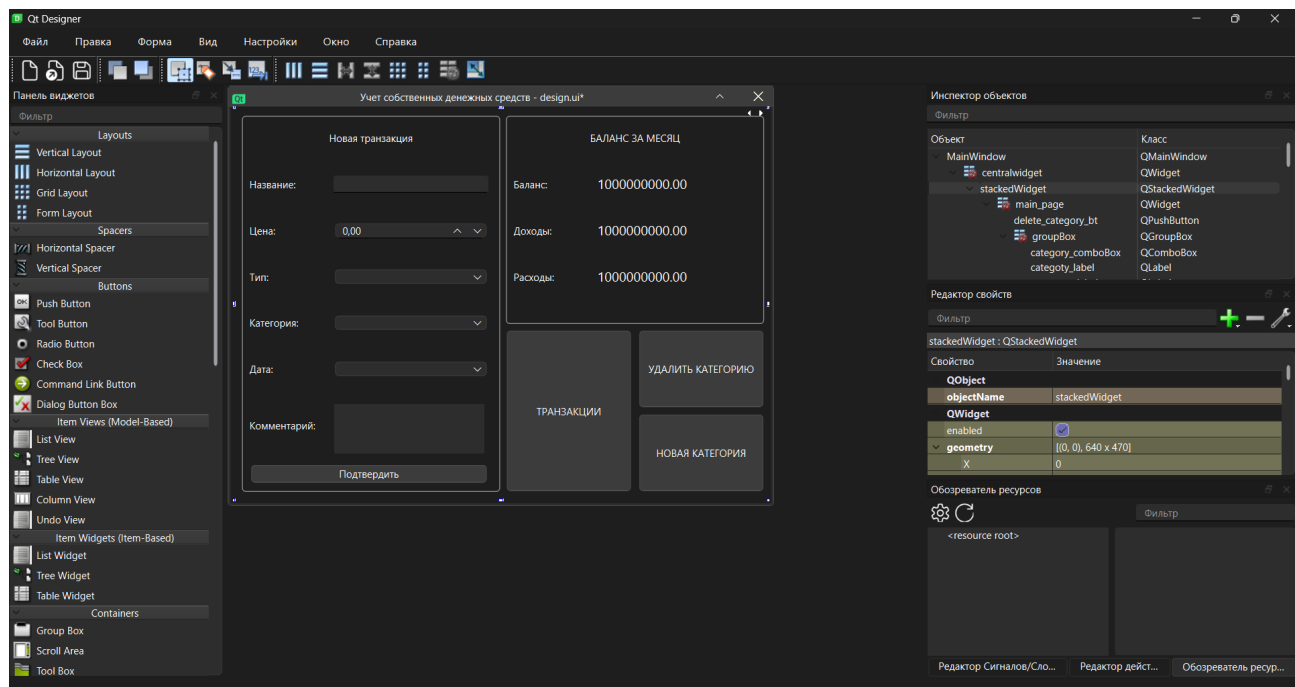


Рисунок Б.1 — Основное окно

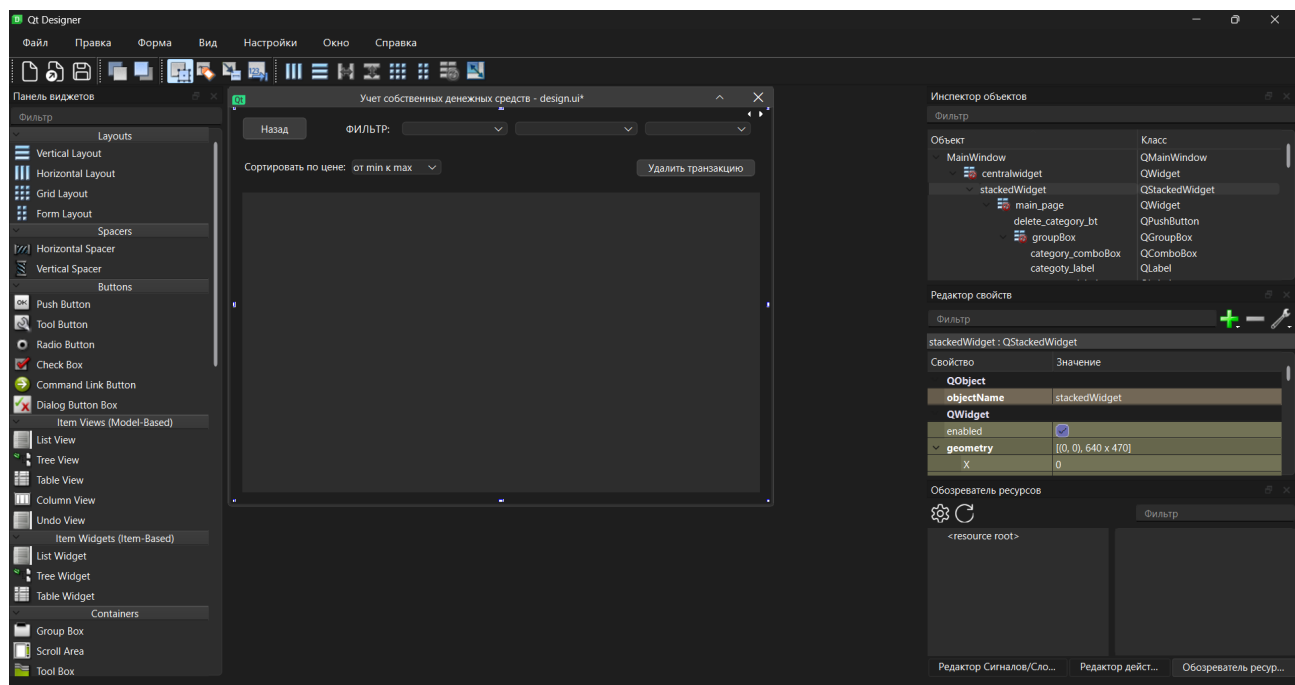


Рисунок Б.2 — Окно для работы с транзакциями

```

class AddCategory(QDialog):
    def __init__(self):
        super().__init__()
        self.DB = Database()
        self.setWindowTitle('ДОБАВЛЕНИЕ КАТЕГОРИИ')
        self.setFixedSize(220, 120)
        layout = QVBoxLayout()
        label = QLabel("Введите новое название категории")
        layout.addWidget(label)
        self.ledit = QLineEdit()
        layout.addWidget(self.ledit)
        bt = QPushButton()
        bt.setText('Добавить')
        bt.clicked.connect(self.bt_clicked)
        layout.addWidget(bt)
        self.setLayout(layout)

```

Рисунок Б.3 — Окно добавления новой категории

```

class DeleteCategory(QDialog):
    def __init__(self):
        super().__init__()
        self.DB = Database()
        self.setWindowTitle('УДАЛЕНИЕ КАТЕГОРИИ')
        self.setFixedSize(220, 120)
        layout = QVBoxLayout()
        label = QLabel("Выберите категорию для удаления")
        layout.addWidget(label)
        self.ledit = QComboBox()
        for i in self.DB.get_categories()[1:]:
            self.ledit.addItem(i.name_category, i.category_id)
        layout.addWidget(self.ledit)
        bt = QPushButton()
        bt.setText('Удалить')
        bt.clicked.connect(self.bt_clicked)
        layout.addWidget(bt)
        self.setLayout(layout)

```

Рисунок Б.4 — Окно удаления категории

```

4 usages
class ErrorDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('ОШИБКА')
        self.setFixedSize(210, 70)
        layout = QVBoxLayout()
        label = QLabel("Информация заполнена неверно!")
        layout.addWidget(label)
        self.setLayout(layout)

2 usages
class CompleteDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('УСПЕШНО')
        self.setFixedSize(210, 70)
        layout = QVBoxLayout()
        label = QLabel("Транзакция успешно добавлена!")
        layout.addWidget(label)
        self.setLayout(layout)

```

Рисунок Б.5 — Окна информирования пользователя