

Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Домашнее задание
Реализация программной модели инфокоммуникационной
системы

Выполнил
Стафеев И.А.
Проверила
Казанова П. П.

Санкт-Петербург,
2024

Задание на выполнение домашнего задания

Требуется создать программное обеспечение системы обработки данных: «Программа для контроля собственных денежных средств».

Необходимо реализовать следующие функции, позволяющие:

1. Добавлять продукт в коллекцию (тип коллекции на ваш выбор).
2. Просматривать все записанное в программу.
3. Просматривать покупки по дате и категории.
4. Распределять их по стоимости от минимальной к максимальной или наоборот.
5. Удалять требуемые записи и выходить из программы.

Дополнительные указания: интерфейс программы реализовать в отдельной функции на ваше усмотрение.

Рекомендуется реализовать возможность сохранения данных в файл.

СОДЕРЖАНИЕ

	Стр.
ВВЕДЕНИЕ	4
1 Анализ предметной области	5
2 Реализация информационных моделей	6
3 Создание пользовательского интерфейса	11
4 Блок-схемы работы программы	20
5 Диаграмма классов	23
6 Схема базы данных	24
ЗАКЛЮЧЕНИЕ	25
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	26

ВВЕДЕНИЕ

Цель работы: разработать программное обеспечение информационной системы для контроля денежных средств.

Для достижения цели были поставлены следующие задачи:

- проанализировать предметную область и требования к выполнению задания;
- выбрать способ представления данных в программе и способы взаимодействия с ними через код
- написать код, использующий выбранные ранее формы и способы взаимодействия с данными и отвечающий требованиям задания;
- определить удобную для пользователя форму для просмотра данных и действий над ними;
- реализовать форму использования программы пользователем.
- провести отладку и тестирование, исправить ошибки.

1 Анализ предметной области

Первым этапом разработки программы является анализ предметной области. В ходе анализа были определены следующие требования к системе:

1. Система должна поддерживать ввод данных пользователем;
2. Система должна иметь возможность добавления, сохранения и удаления введенных данных;
3. Система должна иметь возможность отображать сохраненные ранее данные;
4. Система должна иметь настраиваемые параметры отображения данных: отображение отсортированных данных по дате или по стоимости и отображение отфильтрованных данных по категориям или по датам;
5. Система должна грамотно обрабатывать возможные ошибки, возникающие при использовании программного продукта пользователем и/или предотвращать появление таких ошибок.

В качестве языка программирования, на котором будет написано программное обеспечение, был выбран Python в силу наличия наибольшего опыта в применении его для решения практических задач.

2 Реализация информационных моделей

В соответствии с требованиями задания в программе предполагается две идейные информационные модели - продукт и категория продукта. Для их реализации было принято решение создать базу данных с таблицами, отождествляющими продукт и категорию соответственно. В качестве СУБД была выбрана SQLite, а программная реализация осуществлена с помощью ORM-моделей, написанных с использованием библиотеки SQLAlchemy [1]. Первичным для работы с ORM-моделями является создание подключения к БД, код которого представлен на рисунке 1.

```
models > db_session.py > ...
1  import sqlalchemy as sa
2  import sqlalchemy.orm as orm
3  from sqlalchemy.orm import Session
4  import sqlalchemy.ext.declarative as dec
5
6  SQLAlchemyBase = dec.declarative_base()
7  __factory = None
8
9
10 def global_init(db_file):
11     global __factory
12     if __factory:
13         return
14     if not db_file or not db_file.strip():
15         raise Exception("Необходимо указать файл базы данных.")
16     conn_str = f'sqlite:/// {db_file.strip()}?check_same_thread=False'
17     print(f"Подключение к базе данных по адресу {conn_str}")
18     engine = sa.create_engine(conn_str, echo=False)
19     __factory = orm.sessionmaker(bind=engine)
20     from . import __all_models
21     SQLAlchemyBase.metadata.create_all(engine)
22
23
24 def create_session() -> Session:
25     global __factory
26     return __factory()
27
```

Рисунок 1 — Код, реализующий соединение с базой данных и создание пула подключения

Для таблиц написаны классы *Product* и *Category*. Таблицы в базе данных связаны отношением многие-к-одному, т.е. продукт может иметь толь-

ко одну категорию, но к одной категории может принадлежать множество продуктов.

Код класса ORM-модели для категории продукта представлен на рисунке 2. Поля таблицы - id, имя и цвет в hex-формате, который будет использован позднее при создании пользовательского интерфейса. Метод *validate_name* выполняет проверку допустимой длины имени при добавлении объекта в базу данных.

```
models > category.py > ...
1  from sqlalchemy import Column, Integer, String
2  from sqlalchemy.orm import relation, validates
3  from .db_session import SqlAlchemyBase
4  from random import randint
5
6
7  def get_random_color():
8      """Возвращает случайный цвет для категории"""
9      return "#%02X%02X%02X" % (randint(0, 255), randint(0, 255), randint(0, 255))
10
11
12  class Category(SqlAlchemyBase):
13      """Класс для ORM-модели категории товара"""
14      __tablename__ = "category"
15      id = Column(Integer, primary_key=True, autoincrement=True)
16      name = Column(String, nullable=False, unique=True)
17      color = Column(String, nullable=False, default=get_random_color)
18      products = relation("Product", back_populates="category", cascade="all, delete")
19
20      @validates("name")
21      def validate_name(self, _, value):
22          """Проверка допустимых значений для имени"""
23          if not value:
24              raise ValueError("Name can't be empty")
25          if len(value) > 1000:
26              raise ValueError("Name length can't be more than 1000 symbols")
27          return value
28
29      def __repr__(self):
30          return f"Category(name={self.name})"
31
32      def __str__(self):
33          return self.name
34
```

Рисунок 2 — Код класса ORM-модели для таблицы категории продукта

Код класса ORM-модели для продукта представлен на рисунке 3. Таблица имеет поля `id`, имени, даты совершения покупки, стоимости и `id` категории покупки. Методы `validate_name` и `validate_cost` проверяют допустимую длину имени продукта и допустимую стоимость.

```
models > product.py > ...
1  from sqlalchemy import Column, Integer, ForeignKey, String, Float, DateTime
2  from sqlalchemy.orm import relation, validates
3  from .db_session import SQLAlchemyBase
4  import datetime
5
6
7  class Product(SQLAlchemyBase):
8      """Класс для ORM-модели товара"""
9      __tablename__ = "product"
10     id = Column(Integer, primary_key=True, autoincrement=True)
11     name = Column(String, nullable=False)
12     date = Column(DateTime, default=datetime.datetime.now)
13     cost = Column(Float, nullable=False)
14     category_id = Column(Integer, ForeignKey("category.id"), nullable=False)
15     category = relation("Category", back_populates="products", cascade="all, delete")
16
17     @validates("name")
18     def validate_name(self, _, value):
19         """Проверка допустимых значений для имени"""
20         if len(value) > 1000:
21             raise ValueError("Name length can't be more than 1000 symbols")
22         return value
23
24     @validates("cost")
25     def validate_cost(self, _, value):
26         """Проверка допустимых значений для цены"""
27         if value < 0:
28             raise ValueError("Cost must be positive float")
29         if value > 9223372036854775807:
30             raise ValueError("Cost can't be so enormous")
31         return value
32
33     def __repr__(self):
34         return f"Product(name={self.name}, cost={self.cost}, date={self.date})"
35
36     def __str__(self):
37         return self.name
38
```

Рисунок 3 — Код класса ORM-модели для таблицы продукта

Для удобной работы были созданы несколько функций, позволяющих выполнять запросы в базу данных.

Функции для работы с категориями - получение всех категорий, получение категории по имени, добавление категории и удаление категории по имени - представлены на рисунке 4.

```
db > db_control_functions.py > change_purchase
1  import sys
2  import os
3  sys.path.append(os.getcwd())
4  from models.product import Product
5  from models.category import Category
6  import datetime as dt
7
8
9  def get_categories(session) -> list[Category]:
10     """Возвращает данные о категориях из БД"""
11     return session.query(Category).all()
12
13
14  def get_category_by_name(session, category_name: str) -> Category:
15     """Получение категории по ее имени"""
16     category = session.query(Category).filter(Category.name == category_name).first()
17     return category
18
19
20  def add_category(session, category_name: str) -> Category:
21     """Создание новой категории"""
22     category = Category(name=category_name)
23     session.add(category)
24     session.commit()
25     return category
26
27
28  def delete_category_by_name(session, category_name: str):
29     """Удаление категории по имени"""
30     category = session.query(Category).filter(Category.name == category_name).first()
31     session.delete(category)
32     session.commit()
33
```

Рисунок 4 — Функции для работы с запросами в БД, связанными с категориями товаров

Функции для работы с товарами - получение всех товаров, добавления и изменения товара, удаления товаров по списку id - представлены на рисунке 5.

```
34
35 def get_products(session) -> list[Product]:
36     """Возвращает данные о товарах из БД"""
37     return session.query(Product).all()
38
39
40 def add_purchase(session, product_name: str, cost: float, date: dt.date, category: Category) -> Product:
41     """Добавление новой покупки"""
42     product = Product(name=product_name, cost=cost, date=date)
43     product.category = category
44     session.add(product)
45     session.commit()
46     return product
47
48
49 def delete_purchases(session, ids: list):
50     """Удаление покупок по id"""
51     session.query(Product).filter(Product.id.in_(ids)).delete()
52     session.commit()
53
54
55 def change_purchase(session, id_, product_name: str, cost: float, date: dt.date, category: Category):
56     """Изменение существующей покупки"""
57     product = session.query(Product).filter(Product.id == id_).first()
58     product.name = product_name
59     product.cost = cost
60     product.date = date
61     product.category = category
62     session.commit()
```

Рисунок 5 — Функции для работы с запросами в БД, связанными с товарами

На данный момент в программе реализованы информационные модели и функции, позволяющие выполнять различные действия над этими моделями. В совокупности с функционалом стандартной библиотеки Python требуемый к программному продукту функционал может быть реализован полностью через код. Следующий этап разработки - разработка формы и способов взаимодействия пользователя с исходным кодом программы.

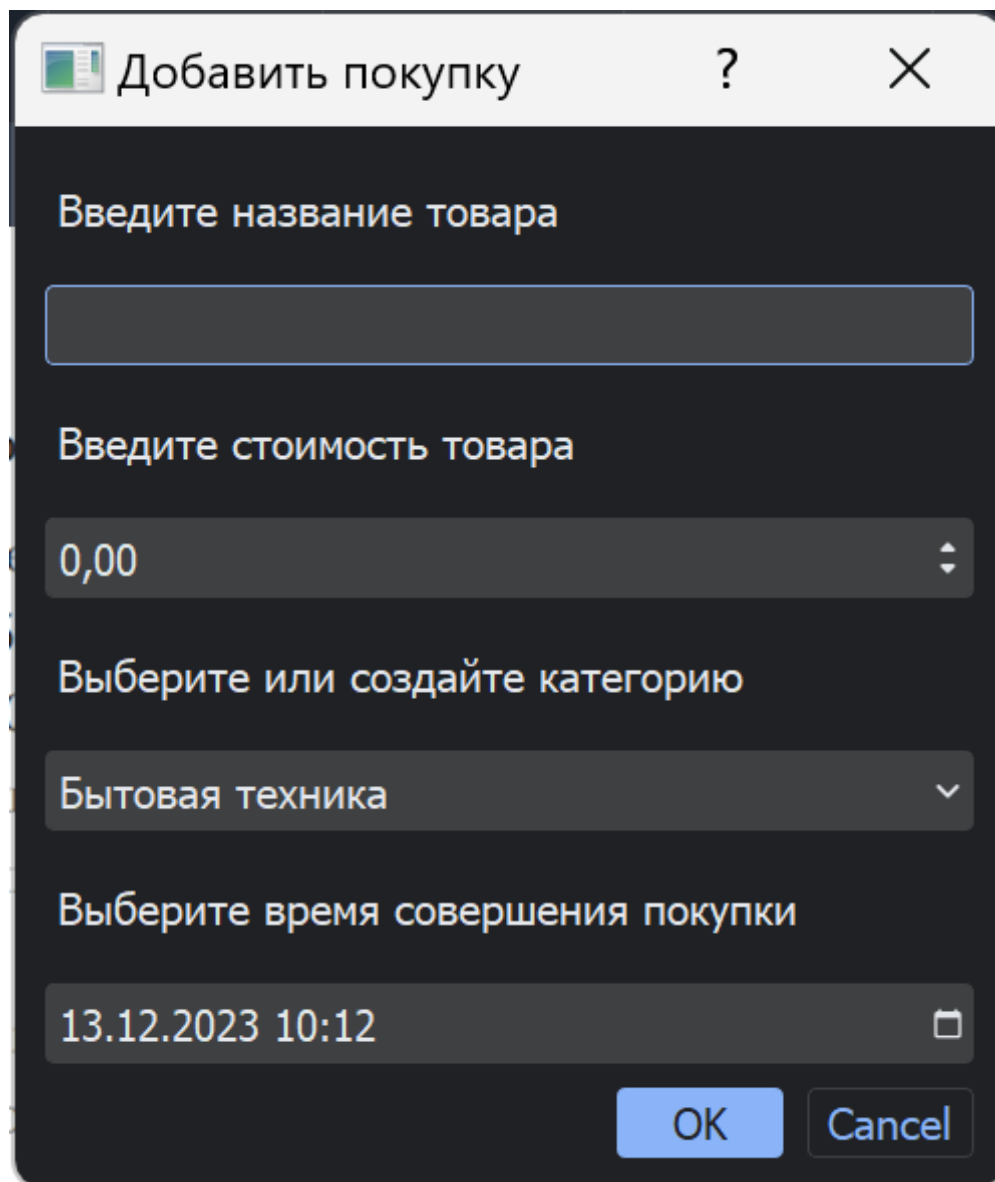
3 Создание пользовательского интерфейса

Чтобы пользователь, незнакомый с программированием, мог пользоваться программой, был разработан графический пользовательский интерфейс на фреймворке Qt, который в коде реализуется с помощью библиотеки PyQt5 [2]. В пользу создания GUI вместо обычного консольного приложения можно отметить, во-первых, большее удобство для пользователя, поскольку взаимодействие с элементами интерфейса гораздо более простое и понятное, нежели ввод команд в консоль, а во-вторых, таким образом проще предотвращать возможные ошибки, которые могут быть вызваны пользователем из-за неправильного ввода, поскольку параметризация виджетов интерфейса позволяет запретить пользователю вводить значения, не соответствующие нужным типам данных или не соответствующие определенной ранее валидации.

В силу объёма кода, реализующего пользовательский интерфейс, в отчете он не будет представлен полностью. Полный исходный код можно посмотреть в репозитории [3].

Пользовательский интерфейс представляет собой два основных окна. Первое предназначено для отображения всех покупок в таблице, второе — это форма для добавления новой покупки или изменения существующей.

Форма добавления покупки представляет из себя виджет с четырьмя полями различных типов, в которые пользователь должен внести информацию о товаре (название, стоимость, категорию и дату совершения покупки) (см. рисунок 6).



Добавить покупку ? X

Введите название товара

Введите стоимость товара

0,00

Выберите или создайте категорию

Бытовая техника

Выберите время совершения покупки

13.12.2023 10:12

OK Cancel

Рисунок 6 — Виджет с формой добавления покупки

Выбранные типы полей предотвращают возможные ошибки ввода. Например, в поле стоимости пользователь не может ввести ничего другого, кроме цифр. Если пользователь не ввел название товара или категорию, он будет об этом уведомлен, и форма работу не завершит (см рис. 7).

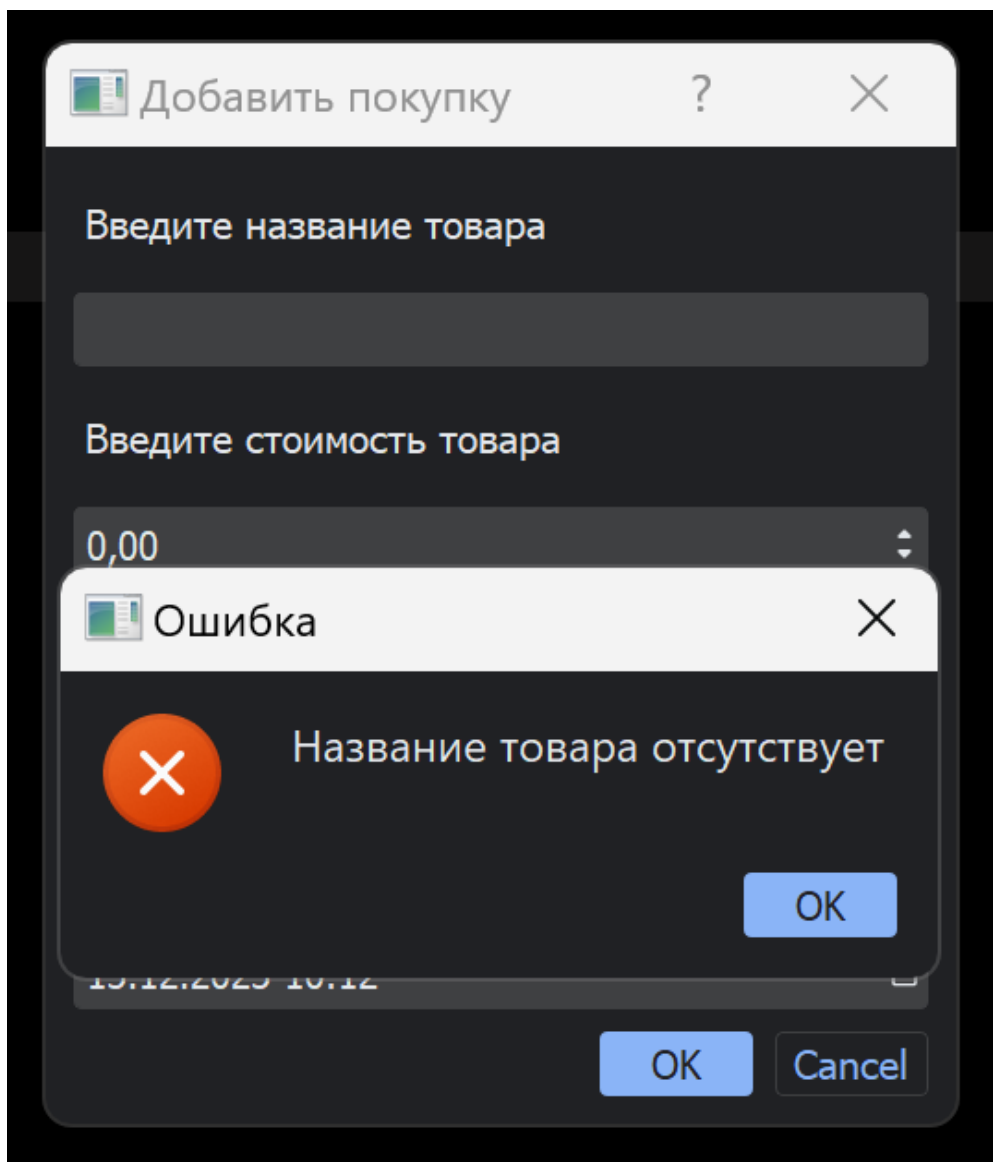


Рисунок 7 — Пример обработки ошибки в форме добавления покупки

После нажатия на кнопку ОК, если введены корректные данные, форма завершает работу, и обработка введенных данных происходит в двух методах класса основного окна. Эти методы используют созданные ранее функции для работы с запросами в БД (см. рис. 8).

```
def exec_add_purchase_form(self, _):
    """Метод для работы с формой добавления покупки"""
    form = AddForm(self.all_categories)
    if not form.exec():
        return
    self.process_purchase(*form.get_data())

def process_purchase(self, *args, id_to_update=False):
    """Обработка данных новой покупке"""
    product_name, cost, category_name, date = args
    print(args)
    date = date.toPyDateTime()
    if category_name not in list(map(str, self.all_categories)): # новая категория
        category = add_category(self.session, category_name)
        self.all_categories.append(category_name)
        self.load_all_categories()
    else:
        category = get_category_by_name(self.session, category_name)

    if not id_to_update: # новая запись
        purchase = add_purchase(self.session, product_name, cost, date, category)
        self.all_purchases.append(purchase)
        self.update_shown_purchases()
    else: # измененная запись
        change_purchase(self.session, id_to_update, product_name, cost, date, category)
        self.reload_all_purchases()
```

Рисунок 8 — Код обработки данных, введенных в форме добавления покупки

Если пользователь ввел название категории, которой нет в списке, то будет создана новая категория с этим именем.

Основное окно программы (рис. 9) состоит из таблицы, в которой отображаются покупки с примененными фильтрами, кнопки, вызывающей открытие формы для добавления покупки, нескольких выпадающих списков с фильтрами отображения данных и текстовой метки, показывающей суммарную стоимость всех отображаемых товаров.

	Дата	Наименование	Стоимость	Категория
1	06-12-2023 13:19	Поездка на электричке	30.0	Транспорт
2	05-12-2023 18:20	Проезд в метро	70.0	Транспорт
3	04-12-2023 20:03	Покупка еды в Дикси	62.98	Продукты
4	01-12-2023 08:00	Оплата месячного ...	200.0	Мобильная связь
5	06-12-2022 13:21	Покупка ...	12500.0	Бытовая техника

Рисунок 9 — Интерфейс основного окна программы

Пользователь может сортировать покупки по возрастанию/убыванию даты совершения или по возрастанию/убыванию стоимости. В выпадающем списке категорий пользователь может выбрать необходимые для отображения категории, поставив у них галочки. Для этого была реализована модель выпадающего списка, содержащего в качестве элементов чекбоксы. В выпадающем списке периода пользователь может выбрать один из предустановленных вариантов отображения - покупки, совершенные за последний день, неделю, месяц или год - или выбрать в виджете календаря необходимый промежуток времени.

Пример использования фильтров пользователем приведен на рисунке 10.

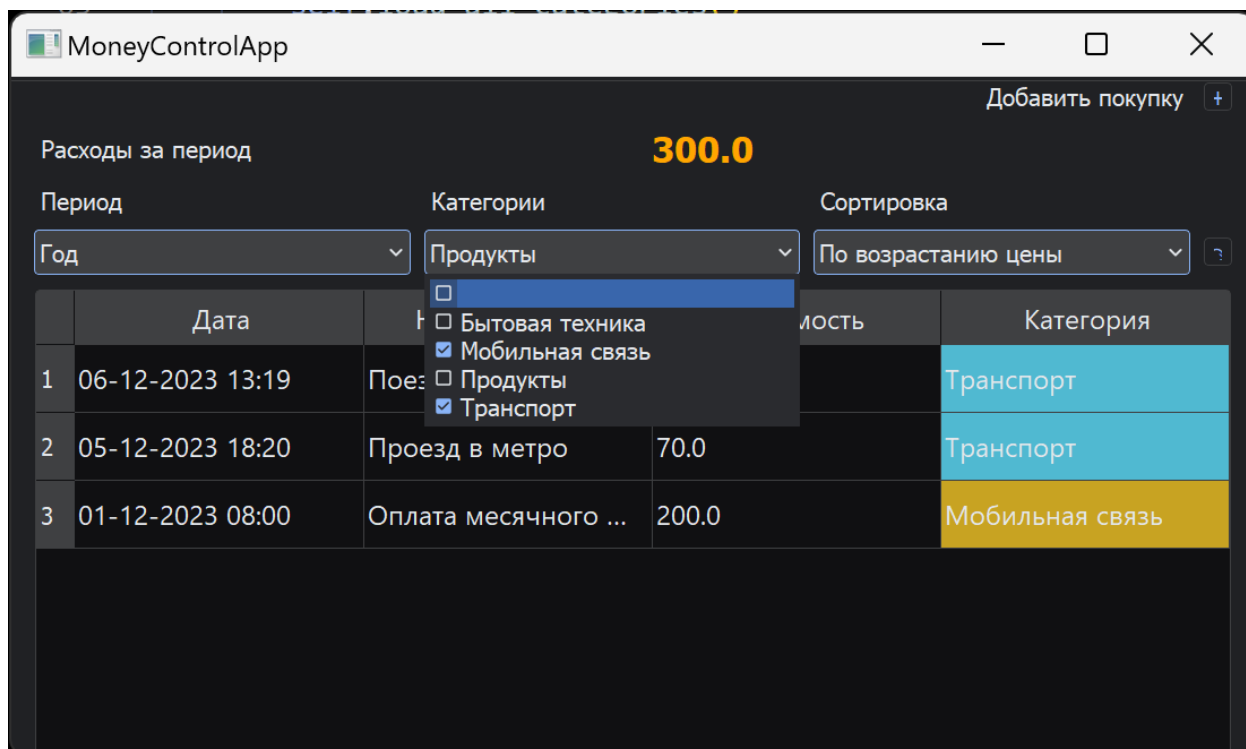


Рисунок 10 — Пример использования фильтров и сортировки в основном окне программы

Пример выбора пользователем временного промежутка отображения покупок приведен на рисунке 11.

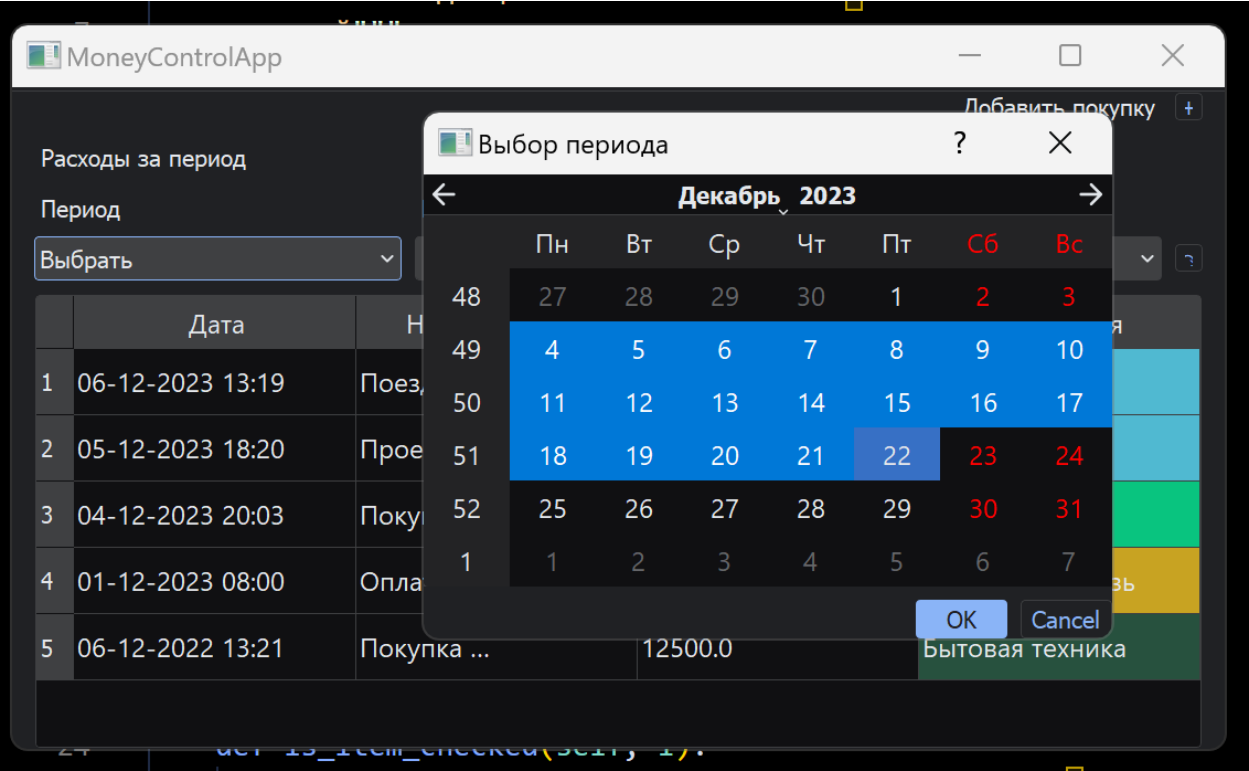


Рисунок 11 — Пример выбора промежутка времени для отображения покупок

Редактирование и удаление покупок было реализовано с помощью контекстного меню. Пользователь может удалить покупки, если выберет нужные строки в таблице и выберет соответствующий пункт в контекстном меню (см. рис. 12). Для удаления категорий пользователь должен выбрать ячейки с категориями в таблице и выбрать соответствующее действие в контекстном меню. При удалении категории будут удалены все покупки, имеющие эту категорию.

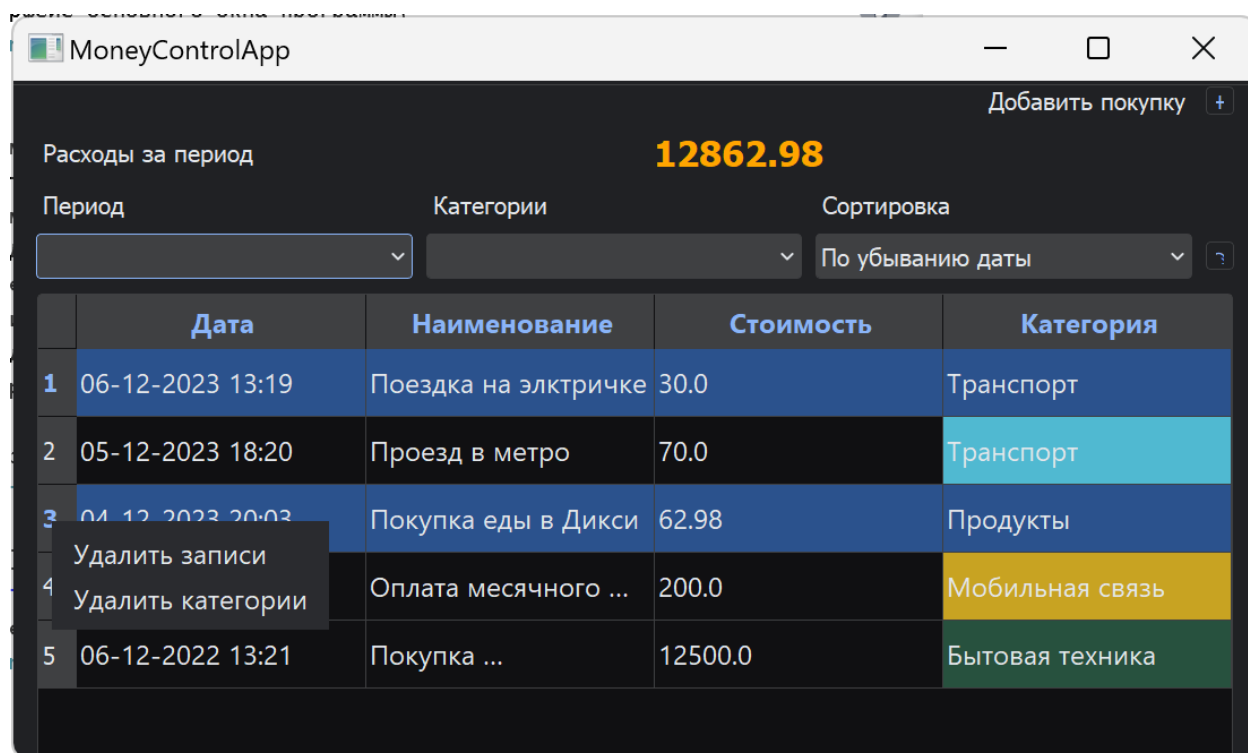


Рисунок 12 — Пример использования контекстного меню в основном окне программы

Если пользователь выберет только одну строку в таблице, он сможет изменить данные о покупке, нажав на нужное действие в контекстном меню (см . рис. 13). После выбора действия будет открыта форма для добавления покупки, но с уже созданными данными. После завершения работы формы данные о покупке будут обновлены.

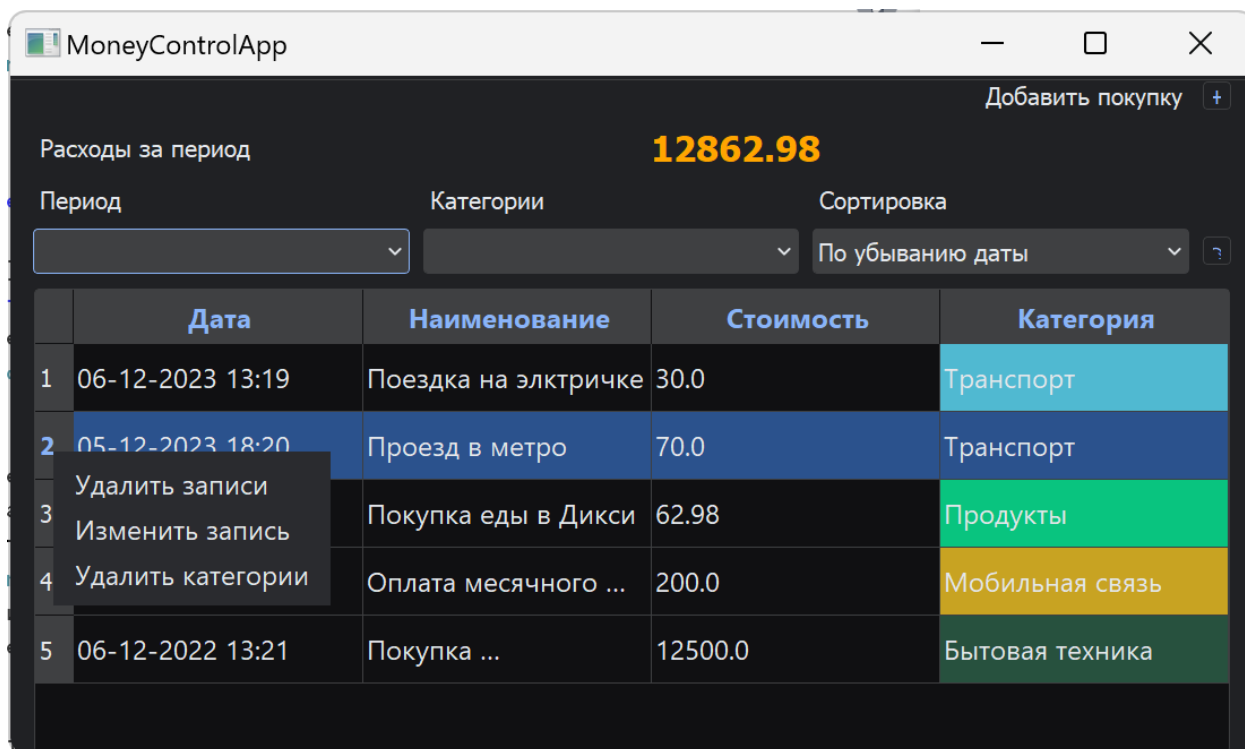


Рисунок 13 — Пример возможности изменить данные о покупке при работе с контекстным меню

4 Блок-схемы работы программы

Блок-схема основного процесса работы программы представлена на рисунке 14.

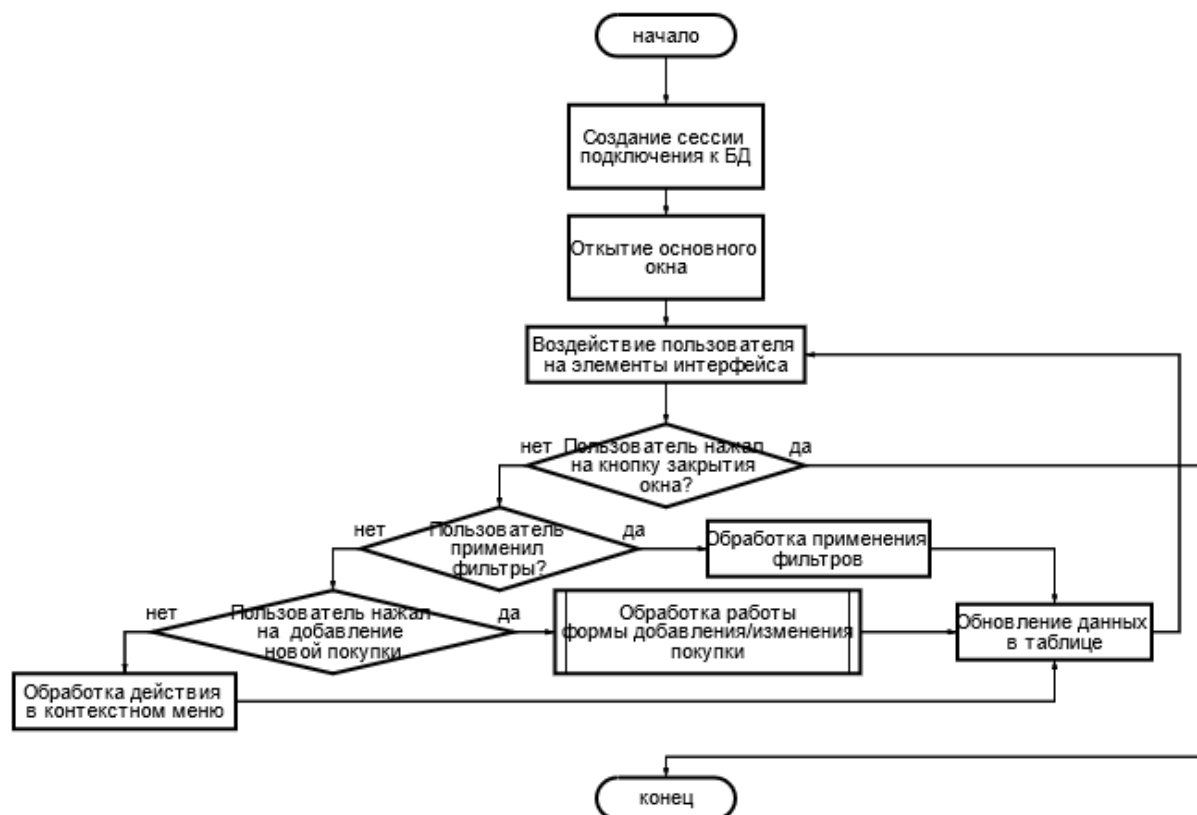


Рисунок 14 — Блок-схема основного процесса работы программы

Блок-схема обработки работы формы добавления/изменения покупок представлена на рисунке 15.

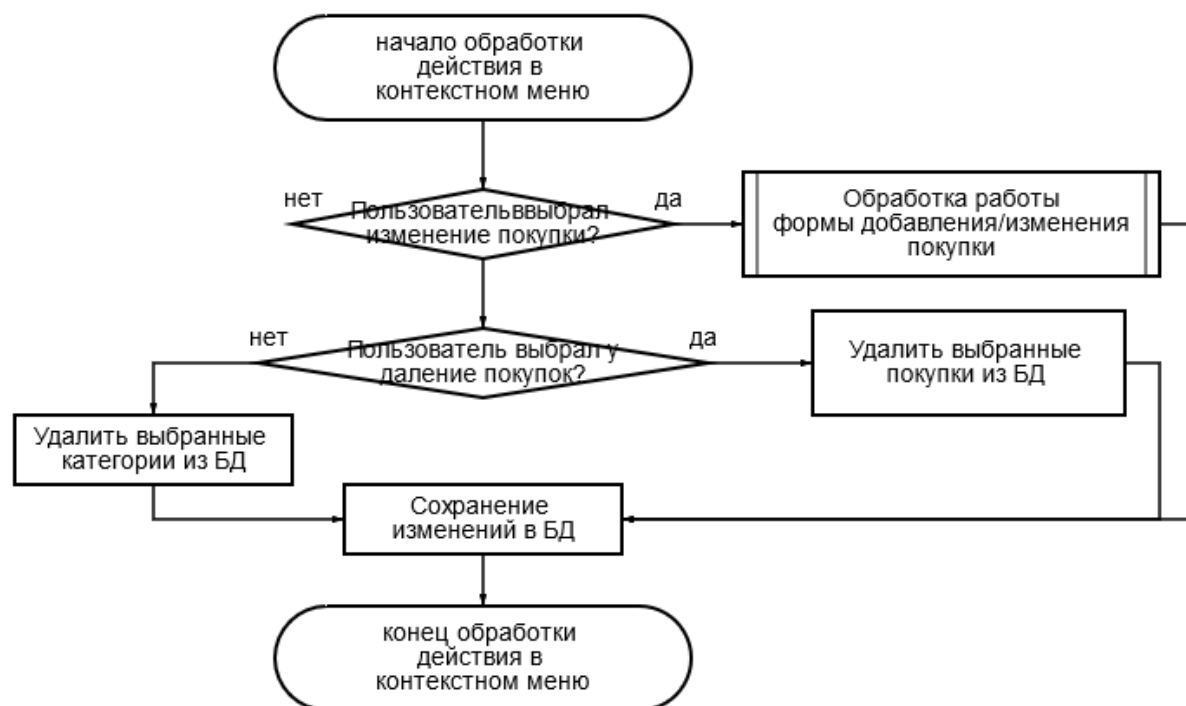


Рисунок 15 — Блок-схема обработки работы формы добавления/изменения покупок

Блок-схема обработки действия в контекстном меню представлена на рисунке 16.

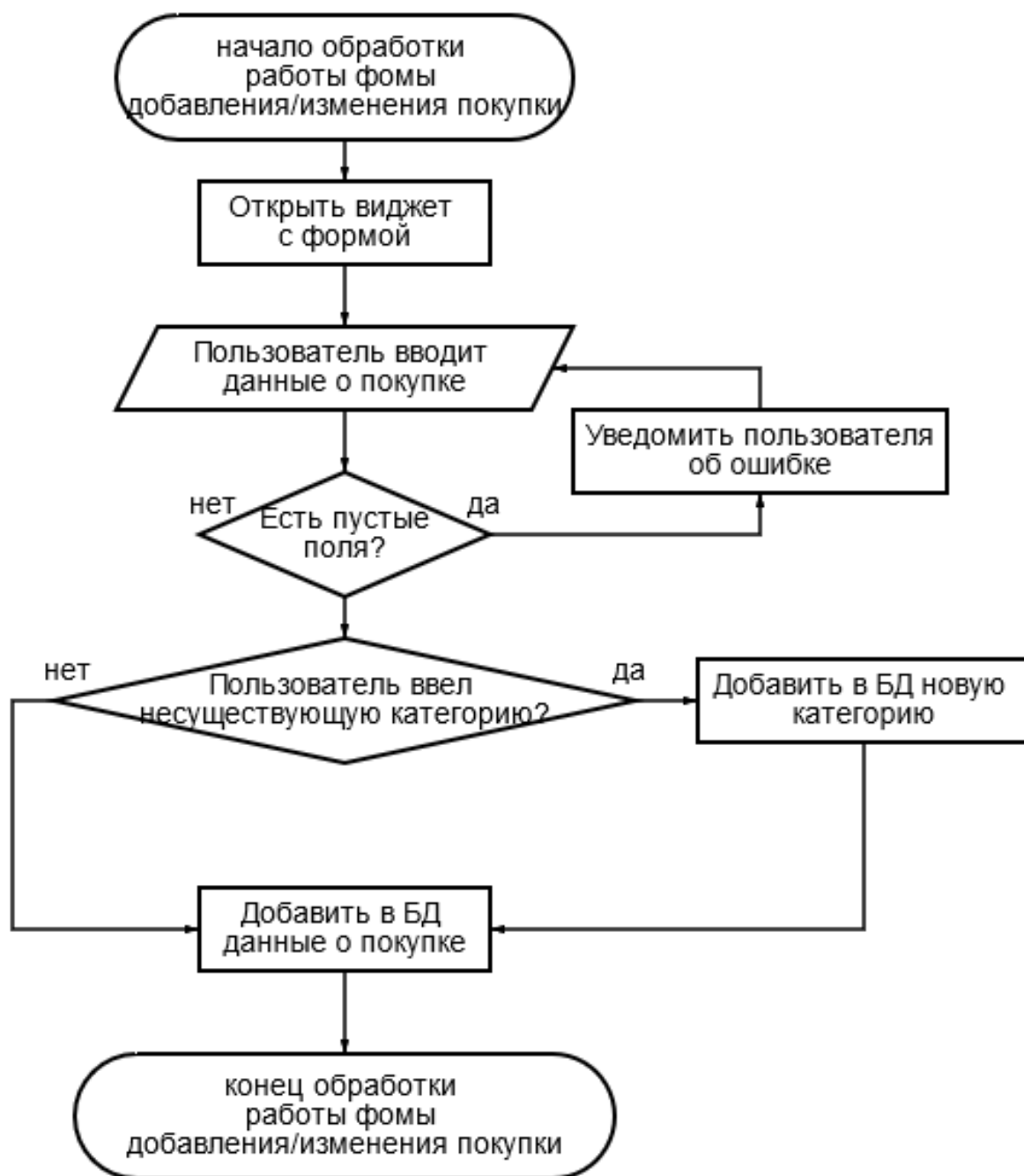


Рисунок 16 — Блок-схема обработки действия в контекстном меню

5 Диаграмма классов

Диаграмма классов на языке UML приведена на рисунке 17

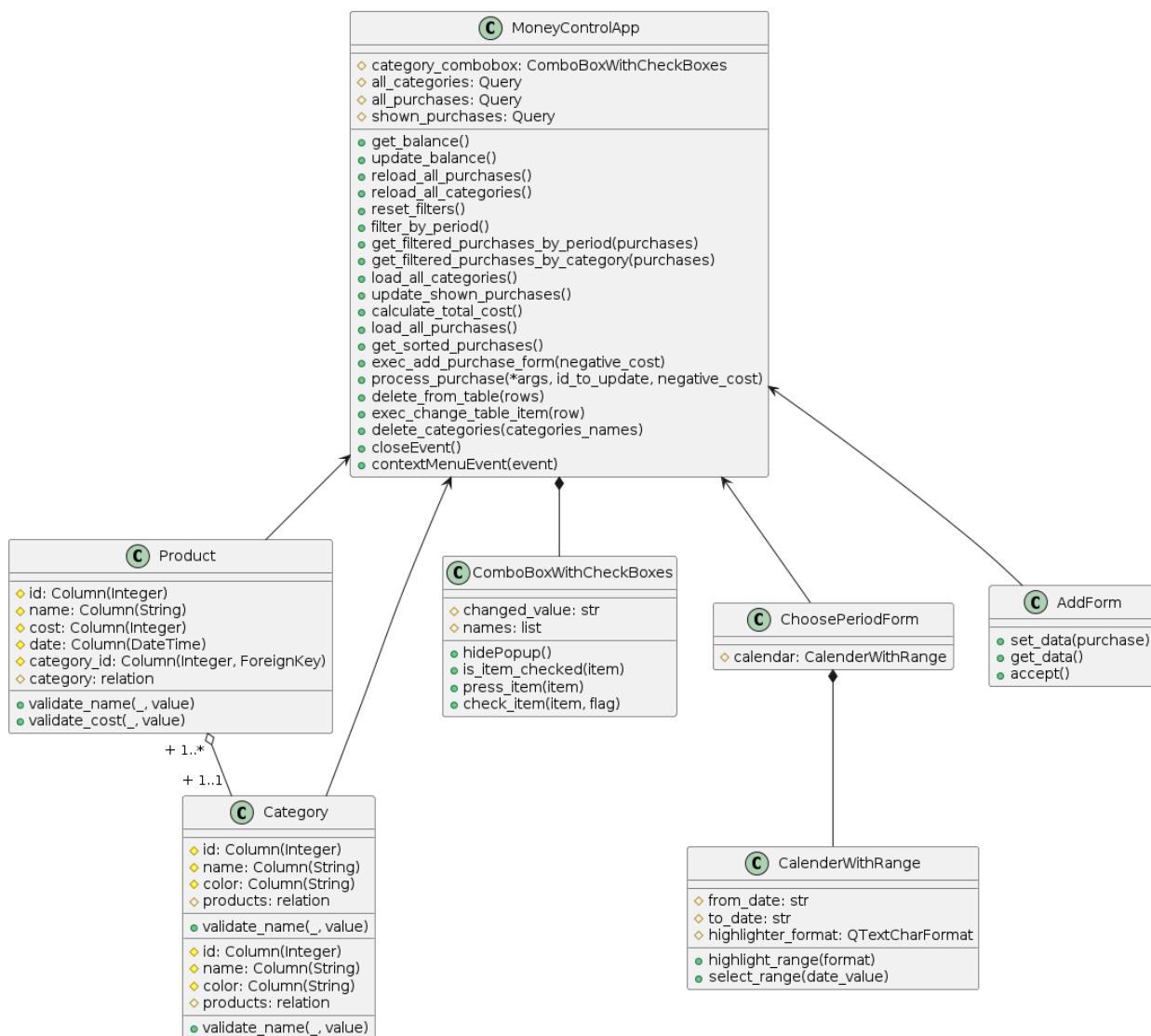


Рисунок 17 — Диаграмма классов

6 Схема базы данных

Схема базы данных, используемой в программе, представлена на рисунке 18.

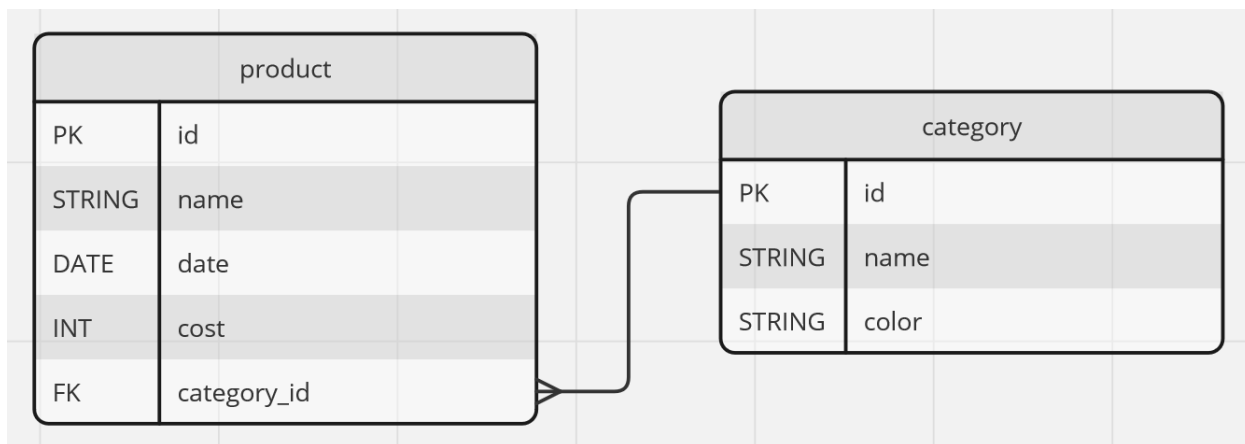


Рисунок 18 — Схема базы данных

В БД реализованы две сущности - продукт (товар) и категория товара. Таблица с товарами содержит информацию о наименовании, стоимости покупки, дате совершения покупки и id категории, к которой товар относится. Таблица с категориями содержит информацию о названии категории и цвете категории для отображения в программе. сущности связаны отношением один-ко-многим, так как к одной категории может относиться много разных товаров.

ЗАКЛЮЧЕНИЕ

Цель работы выполнена. Все поставленные требования к системе удовлетворены. В ходе работы была создана программа с графическим пользовательским интерфейсом, обладающая функционалом для контроля денежных средств. В результате работы были повышены навыки использования языка Python, использования баз данных и графического интерфейса для разработки клиенто-ориентированного программного обеспечения. Полученные навыки будут полезны в дальнейшей профессиональной деятельности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. SQLAlchemy. The Python SQL Toolkit and Object Relational Mapper [Электронный ресурс]. - URL: <https://www.sqlalchemy.org/> (дата обращения: 15.05.2024)
2. Qt for Python Documentation [Электронный ресурс]. - URL: <https://doc.qt.io/qtforpython-5/contents.html> (дата обращения: 15.05.2024)
3. GitHub. Репозиторий проекта [Электронный ресурс]. - URL: https://github.com/staffeev/itmo_prog_hw (дата обращения: 15.05.2024)