Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 1

По предмету: «Проектирование программного обеспечения»

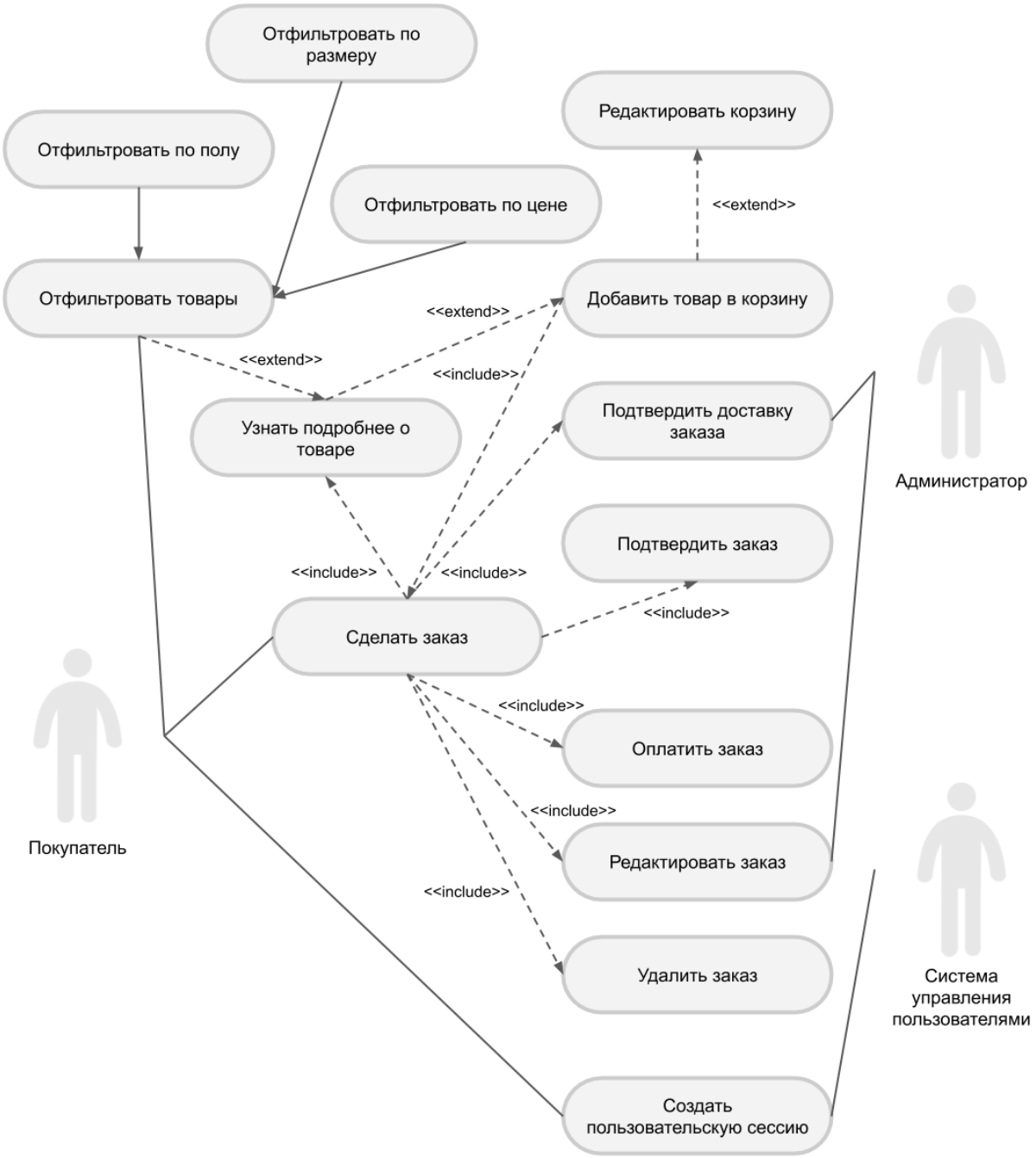
На тему
«Программная реализация доступа к данным»

Студент: Юмаев Артур Русланович
Группа: ИУ7-65Б

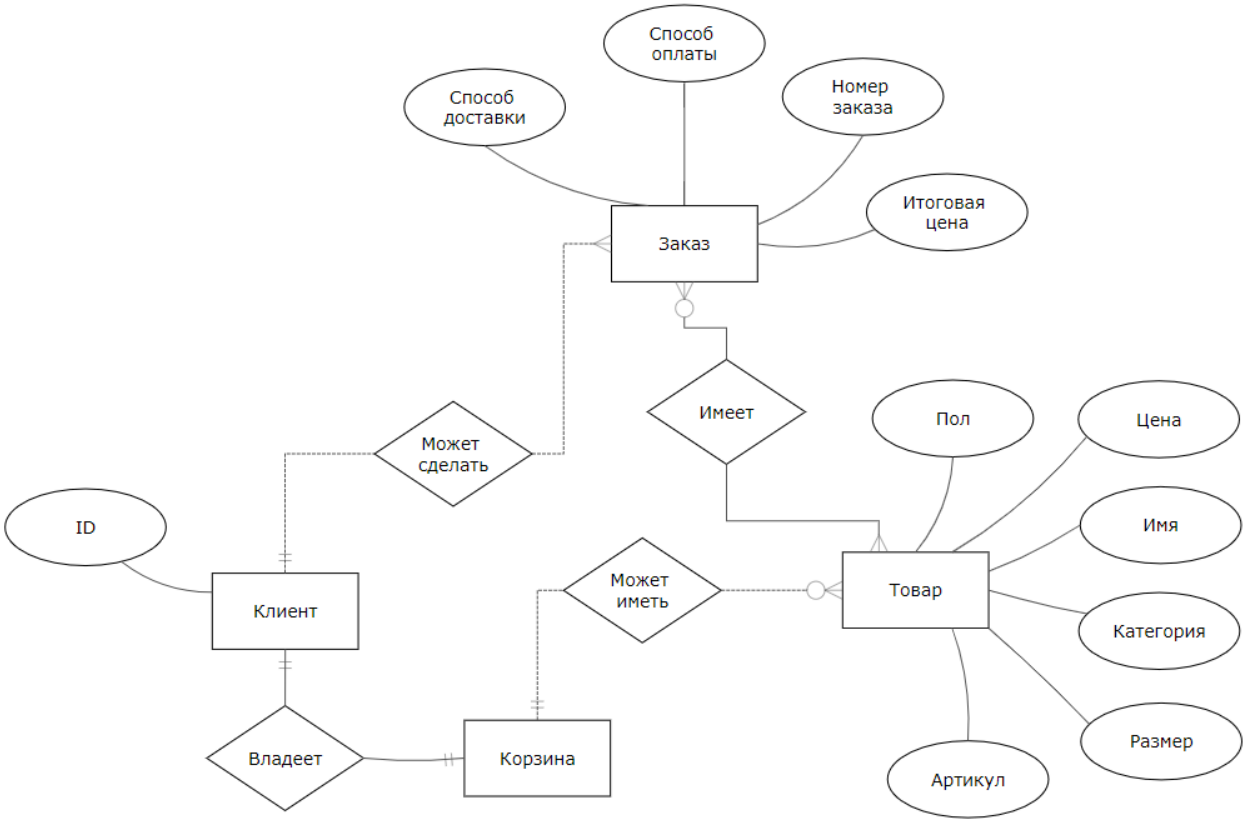
Москва, 2020 г.

Оглавление

1. Use – Case диаграмма



2. ER – диаграмма сущностей



3. Технологический стек

Используемые технологии и подходы

Бэкенд

- Язык программирования Python
- Фреймворк Flask для быстрой реализации простых одностраничных веб приложений на модели MVC (Model – View – Controller)
- Apache HTTP-сервер – кроссплатформенный веб сервер для обработки соединений с юзер-агентом
- SQLite реляционная база данных
- Система контроля версий Git (GitHub)
- Docker — разработка, развертывание на сервере
- Microsoft Azure – облачный сервер с Ubuntu 16.04

Фронтенд

- Верстка макета с помощью HTML, CSS, JavaScript
- Twitter Bootstrap для быстрой и адаптивной разработки интерфейсов

4. Структурная схема программы

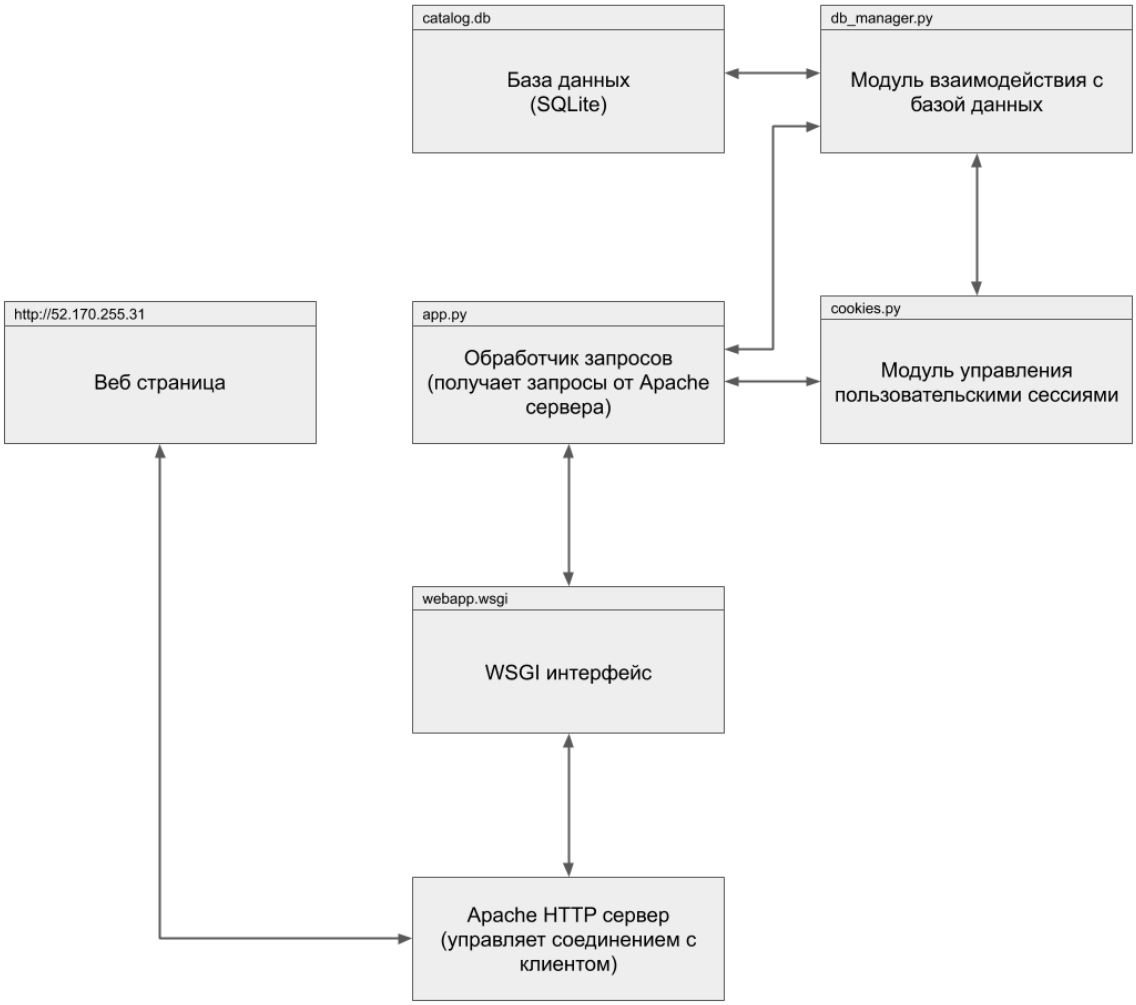


Рисунок 2. Структурная схема программы

5. UML диаграмма классов

Рисунок 3. UML диаграмма классов

6. Программная реализация компонента доступа к данным

В качестве программной реализации доступа к данным был выбран фреймворк sqlite3 версии 3.8.2 для Python 3.7.1. СУБД инициализируется скриптом запуска. В качестве шаблона проектирования компонента работы с базой данных был выбран шаблон репозиторий.

Листинг 1. Класс SQLiteStorage для взаимодействия с базой данных SQLite

```
class SQLiteStorage(StorageInterface):
    def __init__(self):
        self.connection = Connection(sqlite3).getInstance()

    def create(self, query):
        c = self.connection.cursor()
        c.execute(query)
        self.connection.commit()

    def read(self, query):
        c = self.connection.cursor()
        data = [row for row in c.execute(query)]
        self.connection.commit()

        return data

    def insertIntoCart(self, userid, vendor, size):
        insertQuery = GenerateSQLiteQueriesCart().generateInsertIntoCart(userid, vendor, size)
        self.create(insertQuery)
        self.getItemsAmount(userid)

    def getItemsAmount(self, userid):
        selectQuery = GenerateSQLiteQueriesCart().generateGetCartItemsAmount(userid)
        data = self.read(selectQuery)
        goodsAmount = [_ for _ in data][0][0]

        return goodsAmount

    def getData(self, gender, sortby, cats):
        selectQuery = GenerateSQLiteQueriesGoods().generateGetGoods(gender, sort_by, cats)
        self.read(selectQuery)

    def selectCart(self, userid):
        selectQuery = GenerateSQLiteQueriesCart().generateGetCart(userid)
        self.read(selectQuery)

    def update(self):
        pass

    def delete(self):
        pass
```

Листинг 2. Класс GenerateSQLiteQueriesCart для генерации SQL запросов в таблицу Cart

```
class GenerateSQLiteQueriesCart():
    def __init__(self):
        pass

    def generateInsertIntoCart(self, userid, vendor, size):
        sqlQuery = """
        insert into cart (userid, vendor, size) values
        (\'{}\',{},{});
        """.format(userid, vendor, size)

        return sqlQuery

    def generateGetCartItemsAmount(self, userid):
        sqlQuery = """
        select count(userid) from cart where userid = '{}';
        """.format(userid)

        return sqlQuery

    def generateGetCart(self, userid):
        sqlQuery = """
        select *, count(*) as amount from (
            select g.description,
                g.category,
                g.gender,
                c.size,
                g.color,
                g.price * (1 - g.discount) as price,
                c.vendor
            from cart c
            join goods g
            on c.vendor = g.vendor
            where userid = '{0}'
        ) as items
        group by vendor, size
        having count(*) >= 1
        """.format(userid)

        return sqlQuery
```

Листинг 3. Интерфейс работы с хранилищем

```
class StorageInterface(ABC):
    @abstractmethod
    def create(self):
        pass

    @abstractmethod
    def read(self):
        pass

    @abstractmethod
    def update(self):
        pass

    @abstractmethod
    def delete(self):
        pass
```

Листинг 4. Статический коннект менеджер

```
class Connection:
    def __init__(self, server):
        self.server = server
        self.instance = None
        self.configuration = None

    def getInstance(self):
        if self.server.__name__ == "sqlite3":

            with open("./db.conf", "r") as sqliteConfig:
                self.configuration = sqliteConfig.readline()

            try:
                self.instance = self.server.connect(self.configuration)

                return self.instance
            except IOError:
                print('An error occurred trying to connect to sqlite3')
        else:
            # Handling of other database servers may be added
            return None
```