

Сервис обмена информацией о стоимости и качестве товара с использованием геолокации

Пояснительная записка

Оглавление

1. Методы и инструменты программной инженерии.....	2
1.1 Техническое задание	2
1.2 Перечень задач, подлежащих решению в процессе разработки	2
1.3 Обоснование выбора инструментов и платформы для разработки приложения	2
2. Проектирование компонентов программного продукта.....	4
2.1 Архитектура приложения	4
2.2 Разработка API.....	6
2.3 Разработка серверной части приложения	9
2.4 Разработка клиентской части приложения	11
3. Тестирование.....	13
3.1 Руководство администратора	13
3.2 Руководство пользователя	14
Заключение.....	19
Список использованных источников.....	19

1. Методы и инструменты программной инженерии

1.1 Техническое задание

Задача сервиса заключается в том, чтобы собирать информацию о различных товарах и услугах. Информацию предоставляют сами пользователи. В дальнейшем пользователь может получить информацию от сервиса.

Например, человек может ввести в приложение данные о стоимости или качестве товара, а также адрес магазина, в котором товар был приобретен. Эти данные должны быть сохранены в базе данных приложения (информационной системы). Также пользователь может узнать, где можно купить самый дешевый товар по мнению других пользователей приложения. Для этого он должен ввести в поиск название искомого товара и выбрать соответствующий фильтр (стоимость). После этого пользователю должен быть предоставлен список товаров, отсортированных по фильтру (стоимости) с адресами магазинов, в которых их можно приобрести.

1.2 Перечень задач, подлежащих решению в процессе разработки

В процессе разработки приложения нужно решить ряд задач. Поскольку приложение будет клиент-серверным, существуют две глобальные задачи, которые разбиваются на ряд мелких задач:

1. Разработать серверную часть приложения;
2. Разработать клиентскую часть приложения.

Далее приведем список более конкретных задач:

1. Создать базу данных информационной системы;
2. Разработать систему для записи и чтения данных из базы данных;
3. Разработать систему, позволяющую отправлять сообщения;
4. Подготовить API для связи клиентов с сервером;
5. Разработать графический интерфейс клиентской части приложения.

1.3 Обоснование выбора инструментов и платформы для разработки приложения

Приложение будет написано на языке Python 3 [1], поскольку данный язык является языком высокого уровня с огромным количеством свободно распространяемых библиотек, которые могут оказаться полезными в процессе разработки. Большим достоинством Python является его относительная простота, что значительно ускоряет процесс написания приложения.

Для обмена сообщениями между сервером и клиентом будет использоваться технология socket [2]. База данных будет создана с помощью SQLite – компактной встраиваемой СУБД [3]. Работа с базой данных будет осуществляться с помощью библиотеки SQLAlchemy [4]. Графический интерфейс клиентской части приложения будет создан с помощью PyQt5 [5]. Это набор расширений графического фреймворка Qt для языка программирования Python. PyQt5 работает на всех платформах, поддерживаемых Qt (Linux, Mac OS и Windows), и практически полностью реализует возможности Qt.

2. Проектирование компонентов программного продукта

2.1 Архитектура приложения

Приложение является клиент-серверным. Архитектура приложения показана на рис. 2.1.

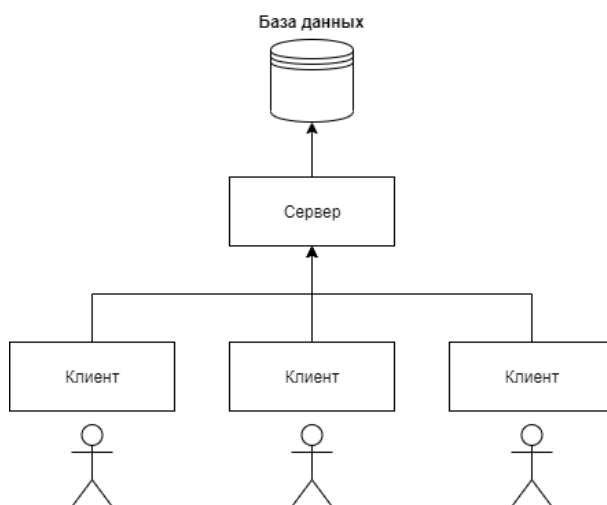


Рис. 2.1. Архитектура приложения

Схема базы данных для разрабатываемой информационной системы приведена на рис.

2.2.

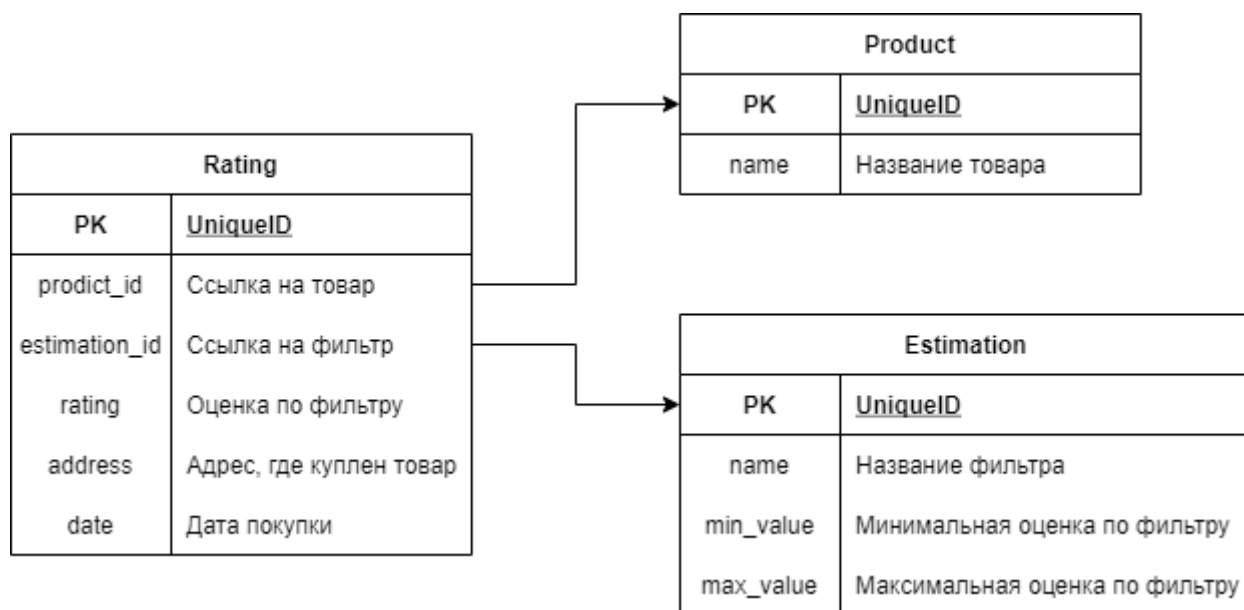


Рис. 2.2. Схема базы данных

База данных состоит из трех таблиц. В таблице Product хранятся общие данные о товарах (наименования). В табл. 2.1 приведены типы и назначения полей таблицы Rating.

Таблица 2.1

Поля таблицы Product

Название поля	Тип	Назначение
PK	Integer	Первичный ключ
name	String	Название товара

В таблицу Estimation записывается информация о фильтрах, которые можно использовать для оценки товаров. В табл. 2.2 приведены типы и назначения полей таблицы Estimation.

Таблица 2.2

Поля таблицы Estimation

Название поля	Тип	Назначение
PK	Integer	Первичный ключ
name	String	Название фильтра (например, стоимость или качество)
min_value	Float	Минимальное значение, которое может принимать оценка товара по данному фильтру (если значение None, то у оценки нет минимального значения)
max_value	Float	Максимальное значение, которое может принимать оценка товара по данному фильтру (если значение None, то у оценки нет максимального значения)

В таблицу Rating записывается информация о поставленных оценках товарам по разным фильтрам. В табл. 2.3 приведены типы и назначения полей таблицы Rating.

Таблица 2.3

Поля таблицы Rating

Название поля	Тип	Назначение
PK	Integer	Первичный ключ

product_id	Integer	Вторичный ключ, указывает на РК товара, которому поставлена оценка
estimation_id	Integer	Вторичный ключ, указывает на РК фильтра, по которому поставлена оценка
rating	Float	Поставленная оценка
address	String	Адрес магазина, в котором приобретен оцениваемый товар
date	DateTime	Дата оценки

2.2 Разработка API

Для того чтобы серверная и клиентская части приложения понимали друг друга при обмене сообщениями, был придуман следующий API.

1. Клиент может запросить получение всех фильтров и названий товаров, отправив json-объект:

```
{“action”: “get_filters_and_products”}
```

От сервера в ответ приходит сообщение со списком названий фильтров и товаров:

```
{“action”: “get_filters_and_products”,
“status”: 200,
“content”: {
    “filter”: [список с данными фильтров],
    “product”: [список с данными товаров]}}
```

Данные о фильтре – это json-объект:

```
{“id”: ID фильтра,
“filter”: название фильтра,
“min”: минимальное значение,
“max”: максимальное значение}
```

Данные о товаре – это json-объект:

```
{“id”: ID товара,
“product”: название товара}
```

Сообщение об ошибке имеет формат:

```
{“action”: “get_filters_and_products”,  
“status”: 400}
```

2. Клиент может запросить получение всех оценок товара по заданному фильтру, отправив json-объект:

```
{  
“action”: “get_ratings”,  
“content”: {  
    “filter”: название фильтра,  
    “product”: название товара}}
```

Если запрос выполнен без ошибок, то приходит ответ:

```
{“action”: “get_ratings”,  
“status”: 200,  
“content”: [{  
    “id”: ID оценки,  
    “address”: адрес покупки,  
    “date”: дата оценки,  
    “rating”: оценка,  
    },]}
```

Если произошла ошибка, то приходит ответ:

```
{“action”: “get_ratings”,  
“status”: 400}
```

3. Добавить новый фильтр можно запросом:

```
{“action”: “add_filter”,  
“content”: {  
    “filter”: название нового фильтра,  
    “min”: минимальное значение оценки по фильтру,  
    “max”: максимальное значение оценки по фильтру}}
```

Если сервер успешно добавил фильтр, приходит ответ:

```
{“action”: “add_filter”,  
“status”: 200,  
“content”: {  
    “id”: ID нового фильтра,  
    “filter”: название нового фильтр,  
    “min”: минимальное значение оценки по фильтру,  
    “max”: максимальное значение оценки по фильтру}}
```

Если фильтр не был добавлен, приходит ответ:

```
{“action”: “add_filter”,
```

```
"status": 400}
```

4. Добавить новый товар можно запросом:

```
{ "action": "add_product",  
  "content": {  
    "product": название нового товара}}
```

Если товар был добавлен, приходит ответ:

```
{ "action": "add_product",  
  "status": 200,  
  "content": {  
    "id": ID добавленного товара,  
    "product": название добавленного товара}}
```

Если товар не был добавлен, приходит ответ:

```
{ "action": "add_product",  
  "status": 400}
```

5. Добавить оценку товара можно запросом:

```
{ "action": "add_rating",  
  "content": {  
    "product": название продукта,  
    "address": адрес покупки товара,  
    "filter": фильтр,  
    "rating": оценка,  
    "date": дата оценки}}
```

Если оценка добавлена, приходит ответ (все оценки товара по выбранному фильтру):

```
{ "action": "add_rating",  
  "status": 200,  
  "content": [{  
    "id": ID оценки,  
    "address": адрес покупки,  
    "date": дата оценки,  
    "rating": оценка,  
  }]  
}
```

Если оценка не была добавлена, приходит ответ:

```
{ "action": "add_rating",  
  "status": 400}
```


2.3 Разработка серверной части приложения

Серверная часть приложения должна записывать и читать данные из базы данных. Для реализации данного функционала был написан класс Database (описание класса содержится в файле database.py). В табл. 2.4 приведены методы класса Database.

Таблица 2.4

Методы класса Database

Название	Назначение
add_estimation	Метод добавляет фильтр в таблицу Estimation базы данных
add_product	Метод добавляет товар в таблицу Product базы данных
add_rating	Метод добавляет оценку в таблицу Rating базы данных
change_estimation	Метод изменяет пределы значений оценки по фильтру
get_estimation	Метод возвращает фильтр с заданным названием
get_estimations	Метод возвращает все фильтры
get_estimation_limits	Метод возвращает пределы значений оценки по фильтру
get_estimations_names	Метод возвращает названия всех фильтров
get_product	Метод возвращает информацию о товаре с заданным названием
get_products	Метод возвращает информацию о всех товарах
get_ratings	Метод возвращает товары с заданным названием, оцененные по заданному фильтру

Для чтения-отправки сообщений на основе технологии socket был написан класс Messenger (описание класса содержится в файле messenger.py). В табл. 2.5 приведены методы класса Messenger.

Таблица 2.5

Методы класса Messenger

Название	Назначение
get_msg	Метод принимает и декодирует сообщение, пришедшее из сокета
receive_all_msg	Метод для чтения сообщения целиком
receive_given_size_msg	Метод для чтения сообщения заданного размера
send_msg	Метод кодирует и отправляет сообщение в сокет

Наконец, для обработки сообщений от клиентов и отправки ответов был написан класс Server (описание класса содержится в файле server.py). В табл. 2.6 приведены методы класса Server.

Таблица 2.6

Методы класса Server

Название	Назначение
process_add_estimation	Метод обрабатывает запрос, пришедший от клиента, на добавление нового фильтра в базу данных
process_add_product	Метод обрабатывает запрос, пришедший от клиента, на добавление нового товара в базу данных
process_add_rating	Метод обрабатывает запрос, пришедший от клиента, на добавление оценки товара
process_get_estimations_and_products	Метод обрабатывает запрос на получение всех фильтров и товаров
process_get_ratings	Метод обрабатывает запрос на получение оценок товаров по заданному фильтру
process_msg	Метод обрабатывает сообщение от клиента
read_messages	Метод читает сообщения из сокетов клиентов
write_responses	Метод отправляет ответы в сокеты клиентов

2.4 Разработка клиентской части приложения

Для клиентской части приложения разработаны два класса: Client и Window. Класс Client предназначен для получения и отправки сообщений на сервер (описание класса содержится в файле client.py). В табл. 2.7 приведены методы класса Client.

Таблица 2.7

Методы класса Client

Название	Назначение
connect	Метод подключает клиента к серверу
create_msg	Метод создает сообщение для отправки на сервер (получает сообщение от Window)
process_msg	Метод разбирает сообщение из сервера
send_msg	Метод отправляет сообщение на сервер

Класс Window создает графический интерфейс клиентской части приложения (описание класса содержится в файле client_gui.py). В табл. 2.8 приведены основные методы класса Window.

Таблица 2.8

Методы класса Window

Название	Назначение
add_filter	Метод выполняется для добавления нового фильтра
add_product	Метод выполняется для добавления нового товара
estimate	Метод выполняется при клике по кнопке 'Оценить' в окне приложения
get_ratings	Метод отправляет сообщение в Client для получения рейтинга товара по фильтру
init_ui	Метод инициализирует виджеты окна клиентской части приложения
process_add_filter_or_product	Метод обрабатывает ответ, пришедший из сервера и полученный в Client, на запрос на добавление фильтра или товара

process_add_rating	Метод обрабатывает ответ, пришедший из сервера и полученный в Client, на запрос на добавление оценки товара
process_get_filters_and_products	Метод обрабатывает ответ, пришедший из сервера и полученный в Client, на запрос на получение фильтров и товаров
process_get_ratings	Метод обрабатывает ответ, пришедший из сервера и полученный в Client, на запрос на получение рейтинга товара

3. Тестирование

3.1 Руководство администратора

Для запуска серверной части приложения нужно выполнить следующие команды (далее будут описаны команды для ОС Windows).

1. В командной строке перейдите в папку GoodsInfoExchanger, где лежит проект.

2. Создайте виртуальную среду командой

```
python -m venv venv
```

3. Активируйте виртуальную среду:

```
venv\scripts\activate.bat
```

4. Установите обновления pip:

```
python -m pip install --upgrade pip
```

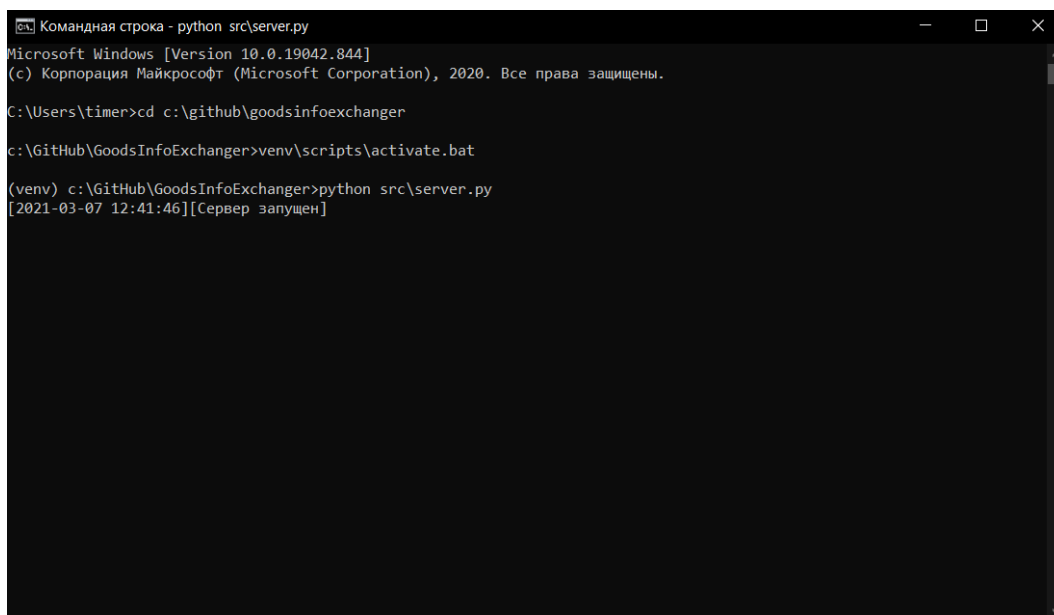
5. Установите необходимые библиотеки:

```
python -m pip install -r requirements.txt
```

6. Запустите серверную часть приложения:

```
python src/server.py
```

В результате должно появиться сообщение, как на рис. 3.1.



```
Командная строка - python src\server.py
Microsoft Windows [Version 10.0.19042.844]
(c) Корпорация Майкрософт (Microsoft Corporation), 2020. Все права защищены.

C:\Users\timer>cd c:\github\goodsinfoexchanger

c:\Github\GoodsInfoExchanger>venv\scripts\activate.bat

(venv) c:\Github\GoodsInfoExchanger>python src\server.py
[2021-03-07 12:41:46][Сервер запущен]
```

Рис. 3.1. Сообщение из серверной части приложения о запуске

3.2 Руководство пользователя

Для запуска клиентской части приложения нужно выполнить следующие команды (далее будут описаны команды для ОС Windows).

1. В командной строке перейдите в папку GoodsInfoExchanger, где лежит проект.

2. Создайте виртуальную среду командой

```
python -m venv venv
```

3. Активируйте виртуальную среду:

```
venv\scripts\activate.bat
```

4. Установите обновления pip:

```
python -m pip install --upgrade pip
```

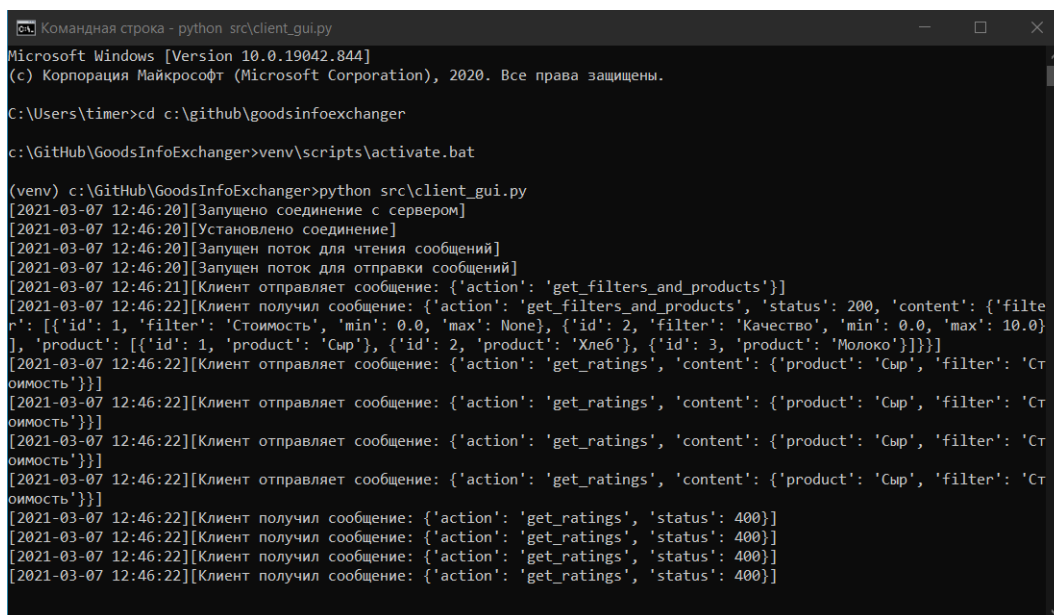
5. Установите необходимые библиотеки:

```
python -m pip install -r requirements.txt
```

6. Запустите клиентскую часть приложения:

```
python src/client_gui.py
```

В результате должно появиться сообщение, как на рис. 3.2, а также графическое окно приложения, как на рис. 3.3.



```
Командная строка - python src/client_gui.py
Microsoft Windows [Version 10.0.19042.844]
(c) Корпорация Майкрософт (Microsoft Corporation), 2020. Все права защищены.

C:\Users\timer>cd c:\github\goodsinfoexchanger

c:\Github\GoodsInfoExchanger>venv\scripts\activate.bat

(venv) c:\Github\GoodsInfoExchanger>python src\client_gui.py
[2021-03-07 12:46:20][Запущено соединение с сервером]
[2021-03-07 12:46:20][Установлено соединение]
[2021-03-07 12:46:20][Запущен поток для чтения сообщений]
[2021-03-07 12:46:20][Запущен поток для отправки сообщений]
[2021-03-07 12:46:21][Клиент отправляет сообщение: {'action': 'get_filters_and_products'}]
[2021-03-07 12:46:22][Клиент получил сообщение: {'action': 'get_filters_and_products', 'status': 200, 'content': {'filters': [{'id': 1, 'filter': 'Стоимость', 'min': 0.0, 'max': None}, {'id': 2, 'filter': 'Качество', 'min': 0.0, 'max': 10.0}], 'product': [{'id': 1, 'product': 'Сыр'}, {'id': 2, 'product': 'Хлеб'}, {'id': 3, 'product': 'Молоко'}]}}]
[2021-03-07 12:46:22][Клиент отправляет сообщение: {'action': 'get_ratings', 'content': {'product': 'Сыр', 'filter': 'Стоимость'}}]
[2021-03-07 12:46:22][Клиент отправляет сообщение: {'action': 'get_ratings', 'content': {'product': 'Сыр', 'filter': 'Стоимость'}}]
[2021-03-07 12:46:22][Клиент отправляет сообщение: {'action': 'get_ratings', 'content': {'product': 'Сыр', 'filter': 'Стоимость'}}]
[2021-03-07 12:46:22][Клиент отправляет сообщение: {'action': 'get_ratings', 'content': {'product': 'Сыр', 'filter': 'Стоимость'}}]
[2021-03-07 12:46:22][Клиент получил сообщение: {'action': 'get_ratings', 'status': 400}]
[2021-03-07 12:46:22][Клиент получил сообщение: {'action': 'get_ratings', 'status': 400}]
[2021-03-07 12:46:22][Клиент получил сообщение: {'action': 'get_ratings', 'status': 400}]
[2021-03-07 12:46:22][Клиент получил сообщение: {'action': 'get_ratings', 'status': 400}]
```

Рис. 3.2. Сообщение из клиентской части приложения о запуске

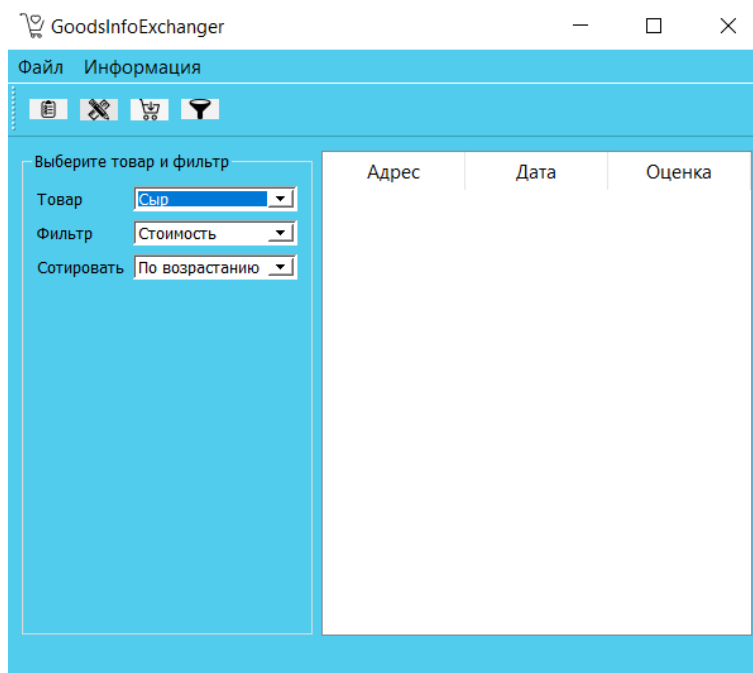


Рис. 3.3. Окно клиентской части приложения

В окне клиента имеется панель инструментов с четырьмя кнопками.



Кнопка для показа рейтинга товара по выбранному фильтру.

В окне имеются поля «Товар», «Фильтр» и «Сортировать». В поле «Товар» можно выбрать название товара. В поле «Фильтр» можно выбрать фильтр, по которому будут выведены оценки товара. В поле «Сортировать» можно выбрать тип сортировки (по возрастанию или убыванию оценок) товаров при выводе их в окне приложения. На рис. 3.4 приведен пример сортировки товара «Сыр» по стоимости.

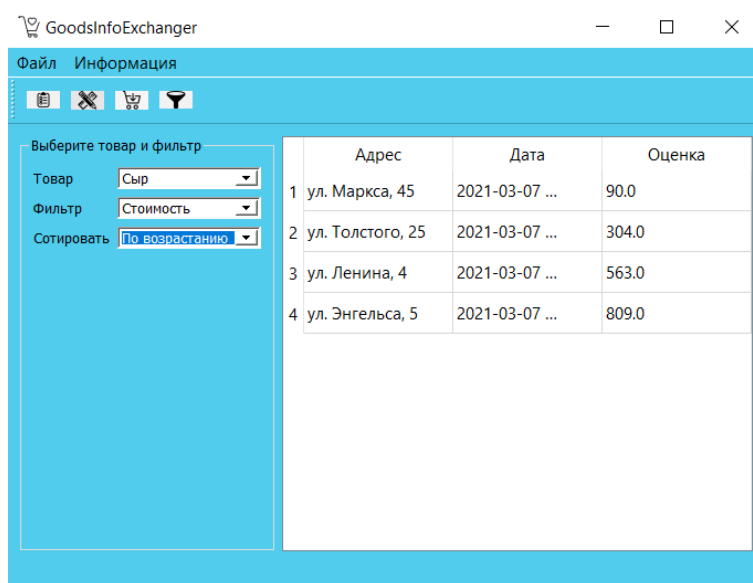


Рис. 3.4. Окно с показом рейтинга выбранного товара



Кнопка для добавления оценки товару.

В окне имеются поля «Товар», «Фильтр», «Адрес» и «Оценка». В поле «Товар» можно выбрать название товара. В поле «Фильтр» можно выбрать фильтр, по которому будет сделана оценка товара. В поле «Адрес» нужно указать адрес магазина, в котором был приобретен оцениваемый товар. Наконец, в поле «Оценка» нужно указать оценку товара по выбранному фильтру. На рис. 3.5 приведен пример оценки товара.

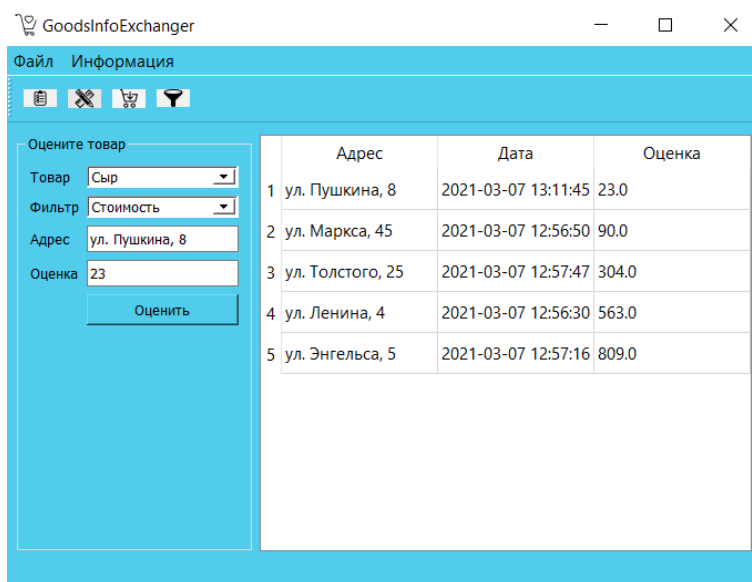
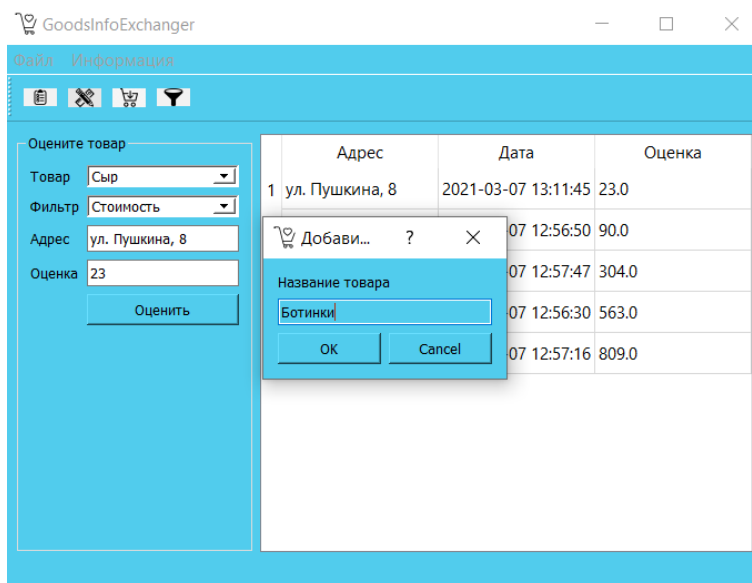


Рис. 3.5. Окно с оцениванием товара



Кнопка для добавления нового наименования товара.

После клика по этой кнопке появится окошко для ввода нового наименования товара (рис. 3.6). Добавленный товар можно будет впоследствии оценить (выбрать в поле «Товары» в окне оценивания товаров).



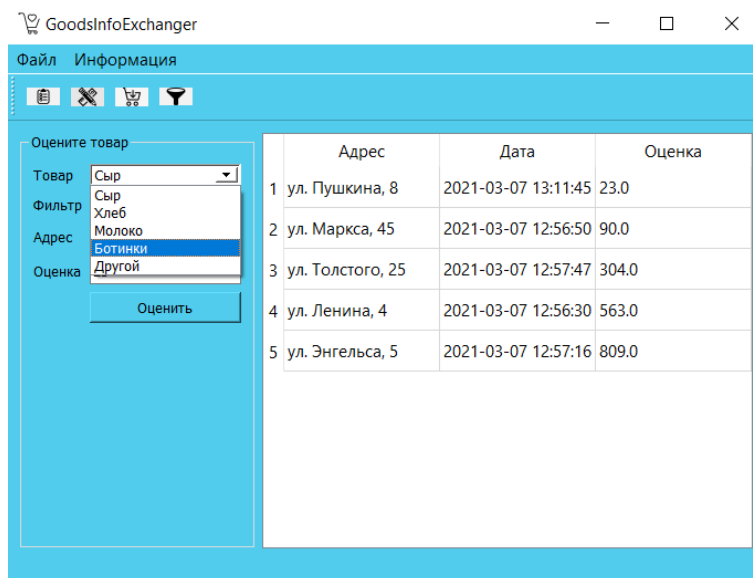
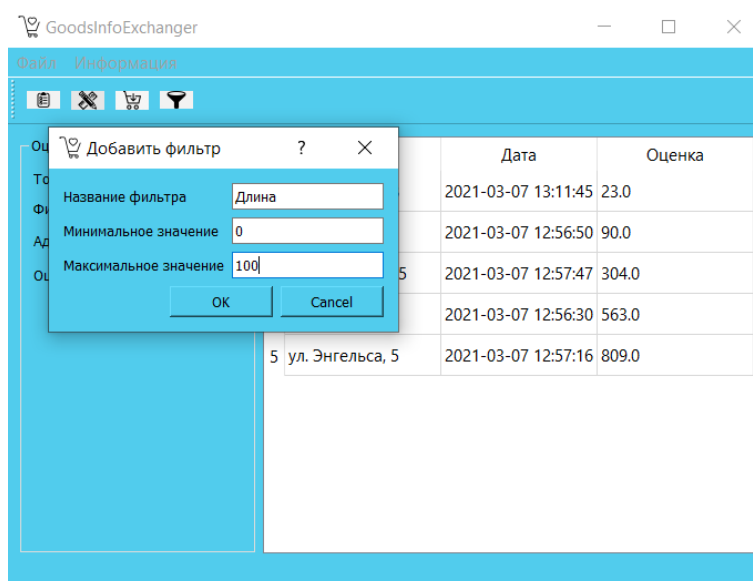


Рис. 3.6. Добавление нового наименования товара



Кнопка для добавления нового фильтра товара.

После клика по этой кнопке появится окошко для ввода нового фильтра товара (рис. 3.7). Добавленный фильтр можно будет впоследствии применить для оценивания товара (выбрать в поле «Фильтр» в окне оценивания товаров).



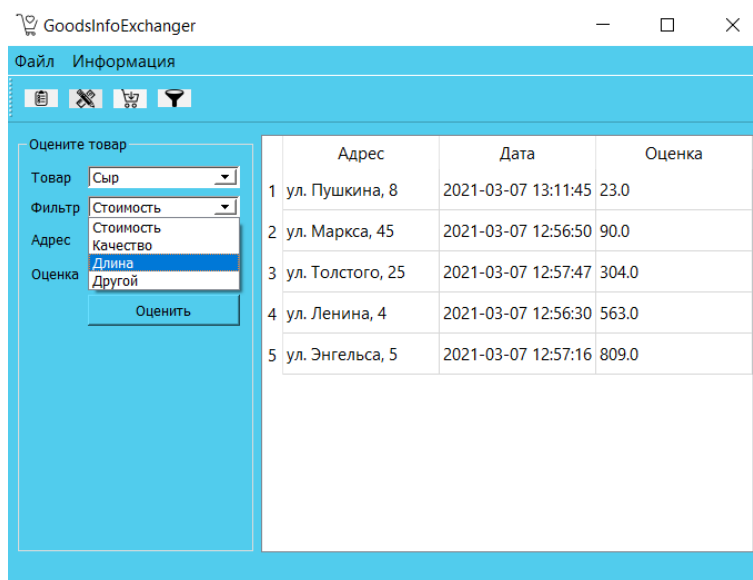


Рис. 3.7. Добавление нового фильтра товара

Заключение

Разработано клиент-серверное приложение по обмену информацией о стоимости и качестве товара с использованием геолокации. Приложение написано на языке Python 3 с применением технологии socket для обмена сообщениями между клиентом и сервером. Графический интерфейс клиентской части приложения реализован с помощью фреймворка Qt.

Список использованных источников

1. Лутц М. Изучаем Python // Символ-Плюс, 2011.
2. socket – Low-level networking interface // <https://docs.python.org/3/library/socket.html>
3. SQLite // <https://ru.wikipedia.org/wiki/SQLite>
4. SQLAlchemy // <https://www.sqlalchemy.org/>
5. PyQt5 // <https://pypi.org/project/PyQt5/>