

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ГОРОДА МОСКВЫ
Государственное бюджетное общеобразовательное учреждение города Москвы
«Школа № 1387»
(ГБОУ школа № 1387)

Кейс №5: Сбор и обработка данных температуры

Авторы работы:
Марков Александр Сергеевич
Прошина Евгения Александровна
Аваков Артём Артурович
11 И класс
ГБОУ школа № 1387
Научный руководитель:

Москва, 2020

Оглавление

Анализ технических требований	2
Задание	2
Условия выполнения	2
Пользовательский интерфейс	3
Язык программирования и программные средства	3
Языки программирования	3
Сравнение языков программирования	4
Модули стандартной библиотеки Python	4
Библиотеки Python	4
Программные средства	4
Описание основных этапов разработки	4
Первый этап	4
Второй этап	5
Третий этап	5
Четвёртый этап	5
Пятый этап	5
Шестой этап	5
Структурная и функциональная схема	6
Функциональные схемы: Parser	6
Алгоритм работы программного продукта	7
База данных	10
Результаты разработки	10
Программный код	11
Модуль 1	11
База данных	15
Таймер	18

Анализ технических требований

Задание

Реализовать программный модуль для сбора, хранения и обработки данных с удалённых температур данных

Условия выполнения

Посредством специализированного сервиса, расположенного по адресу http://dt.miet.ru/ppo_it, осуществить сбор данных об уличной температуре в 16

городах. Необходимо использовать показатели датчиков, находящихся в 10 квартирах не менее, чем пяти районов города. Время осуществления – 48 часов реального времени.

Обращение к сервису происходит не реже, чем один раз в 10 минут.

Полученные данные хранятся с помощью реляционной СУБД, реализованной на основе ER-модели.

Взаимодействие с данными осуществляется через пользовательский интерфейс, который по запросу пользователя отображает информацию о температуре в квартирах и на улице в виде графика/диаграммы (см. пункт [пользовательский интерфейс](#))

Пользовательский интерфейс является кроссплатформенным, а также организован в соответствии со стандартами построения UI

Для облегчения работы с вносящимися в программный код изменениями используется система управления версиями git

Пользовательский интерфейс

Согласно регламенту испытаний функционал UI подразумевает выведение следующих данных в виде графика/диаграммы/величины:

- Данные температуры в реальном времени в определённой квартире
- График изменения уличной температуре на протяжении суток реального времени в одном из городов
- График изменения средней температуры в квартирах в одном из городов на протяжении суток реального времени
- График изменения температуры в одной квартире в каждом из городов
- Диаграмма максимальных температур в квартирах в каждом из районов (не менее пяти) одного города

Язык программирования и программные средства

Языки программирования

Учитывая изложенные выше особенности технического задания, было решено, что в качестве основного языка программирования будет использоваться Python.

При выборе важную роль сыграли такие его качества, как минималистичный синтаксис ядра и богатая стандартная библиотека, позволяющая работать с высокоуровневыми структурами данных и взаимодействовать со многими сетевыми протоколами, в частности HTTP.

Управление БД осуществляется посредством СУБД SQLite ввиду того, что его легко использовать при кроссплатформенном переносе, а также он очень надёжен с точки зрения программного кода.

Сравнение языков программирования

	Python	C++	Java
Опыт работы	+	-	-
Простота синтаксиса и удобство	+	-	-
Производительность	-	+	+
Работа с БД	+	-	-
Динамическая типизация	+	-	-

Таблица 1

Производительность языка не играла важную роль при выборе языка, так как программный продукт используется малым числом лиц и объёмы обрабатываемых данных относительно малы

Модули стандартной библиотеки Python

- модуль requests – инструмент составления HTTP и GET-запросов для взаимодействия с сервером
- модуль time – для работы с реальным/серверным временем
- модуль sqlite3 – кроссплатформенное средство работы с БД

Библиотеки Python

- Matplotlib – осуществляет визуализацию информации из БД в UI
- IPyWidgets – интерактивные HTML виджеты для Jupyter Notebook

Программные средства

Дистрибутив Anaconda предусматривает инструмент интерактивной разработки Jupyter Notebook, удобный с точки зрения разработки и использования интерфейса, в том числе он поддерживает создание графического интерфейса для пользователя. В качестве IDE используется Spyder из дистрибутива Anaconda. Одна из важных особенностей этой среды разработки – это интеграция с научными библиотеками Python, к примеру Matplotlib

Описание основных этапов разработки

Первый этап

С помощью интернет-ресурсов мы изучили синтаксис и принципы работы описанных выше модулей и библиотек языка Python, СУБД SQLite, а также ознакомились с основами работы в программных средствах Jupyter Notebook и Spyder

Второй этап

Для реализации программного кода выбраны методы структурного программирования. Составлены концепции функций, образующих модуль взаимодействия с сервером и обработкой полученной информации и модуль пользовательского интерфейса и визуализации данных (далее «модуль 1» и «модуль 2» соответственно; см. [Алгоритм работы программного продукта](#))

Третий этап

На сайте <https://github.com/> создана система контроля версий продукта, доступная по следующей ссылке: <https://github.com/CangCiwei/PredProf>. Составлен план выполнения технического задания согласно функциональности продукта.

Четвёртый этап

Проектируется UI. Осуществляется написание программного кода для модуля 1 и модуля 2. Код комментируется. Каждая составная функцию тестируется на корректность возвращаемых данных.

Оформляется техническая документация.

Пятый этап

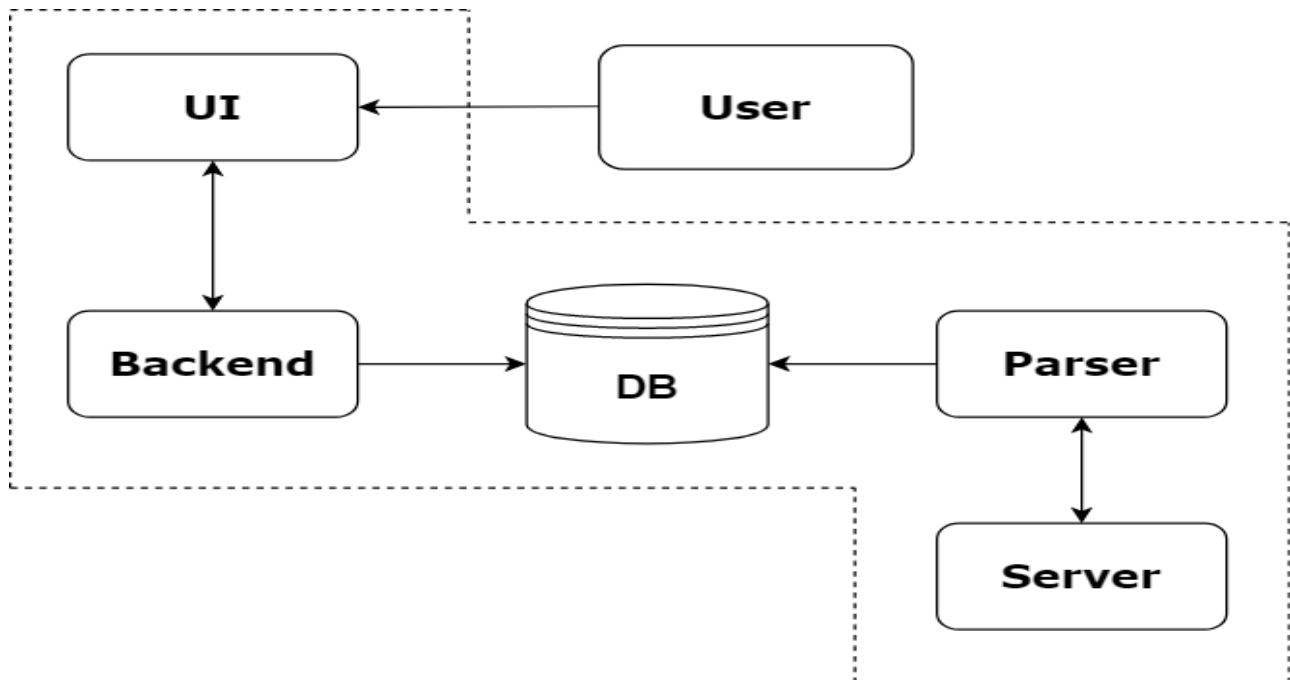
Тестируется функциональность программы. Проводятся необходимые корректировки программного кода (см. [Третий этап](#)) и повторные тестирования.

Шестой этап

Производится сбор данных температуры на протяжении 48 часов реального времени. По ним строятся графики, описанные в разделе [Пользовательский интерфейс](#)

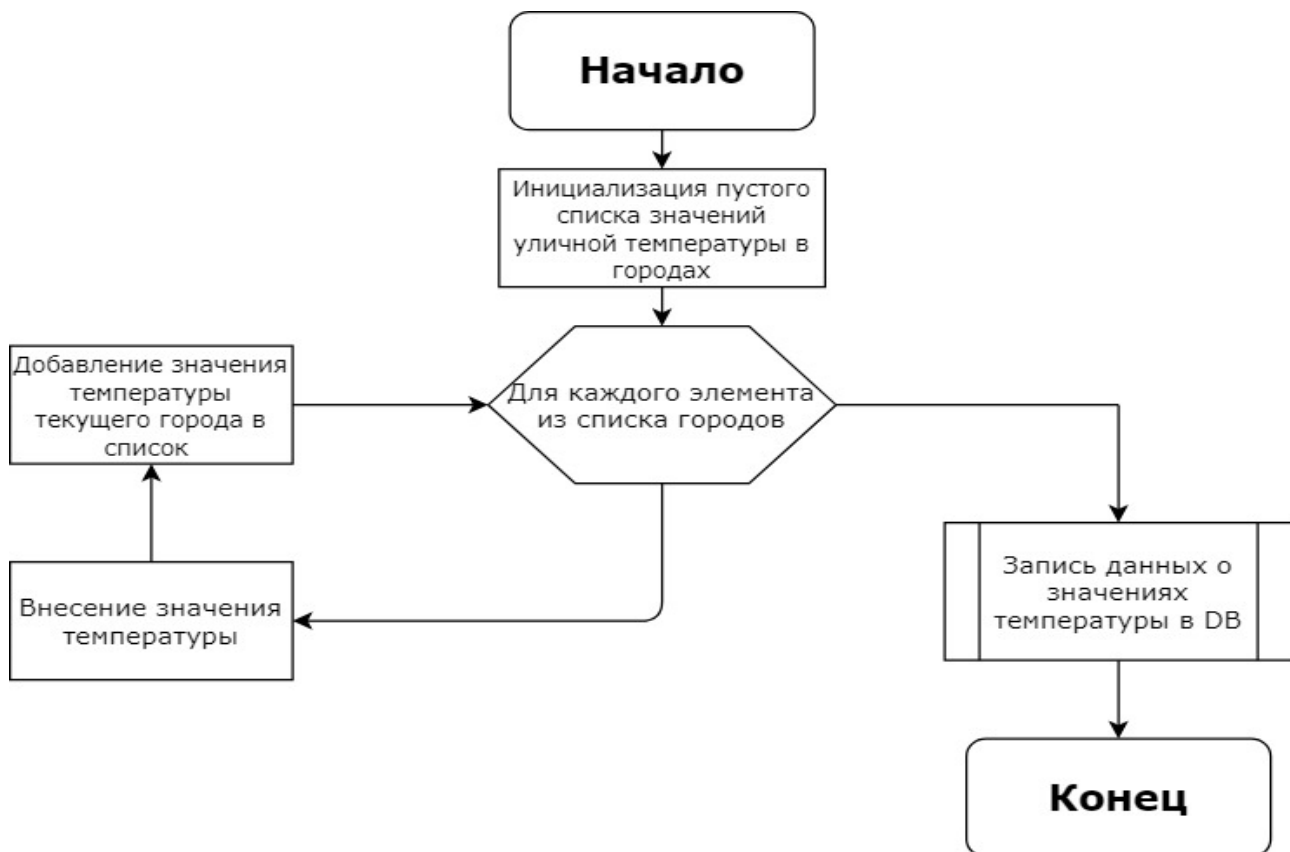
Структурная и функциональная схема

1. Структурная схема работы алгоритма

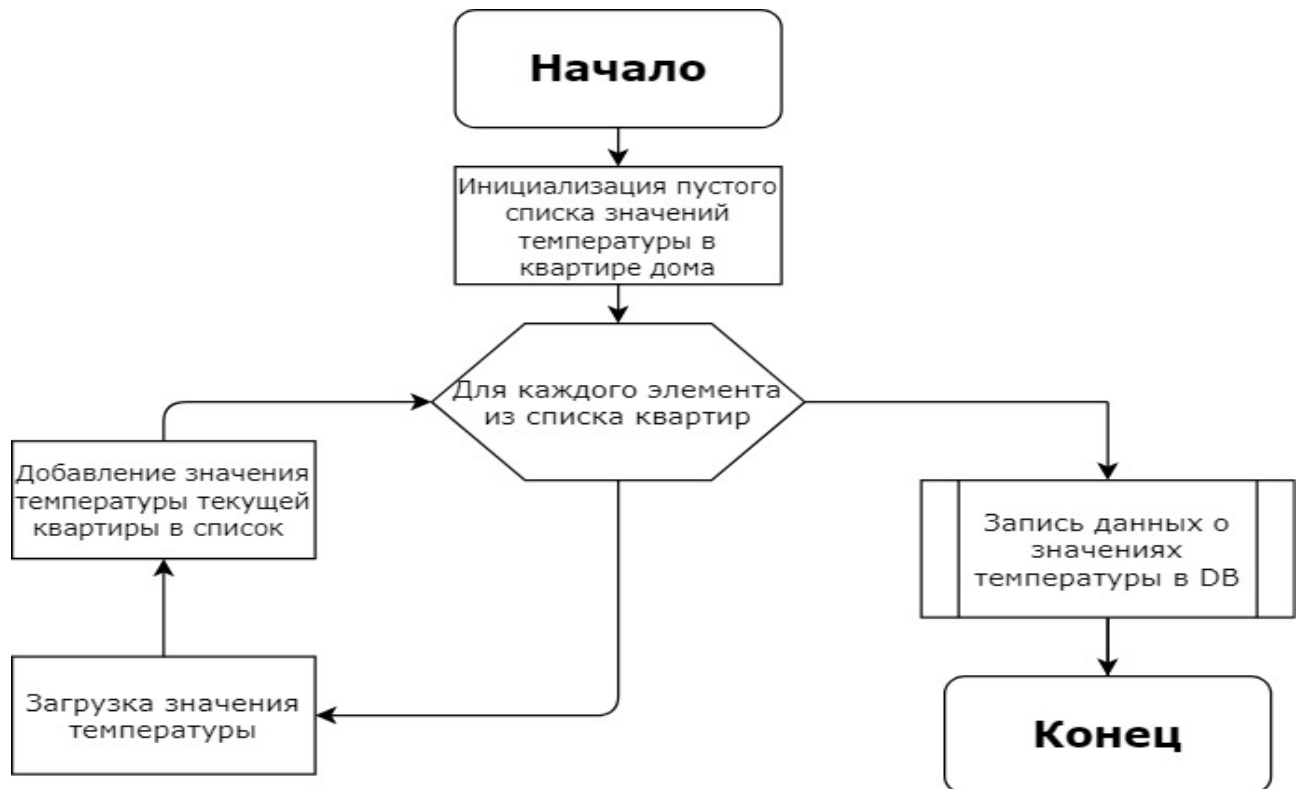


Функциональные схемы: Parser

2. Функциональная схема: внесения данных уличной температуры в городах



3. Функциональная схема: внесение значений температуры в квартире



Алгоритм работы программного продукта

Пользователь посредством специализированного UI взаимодействует с данными находящимися в DB посредством Backend

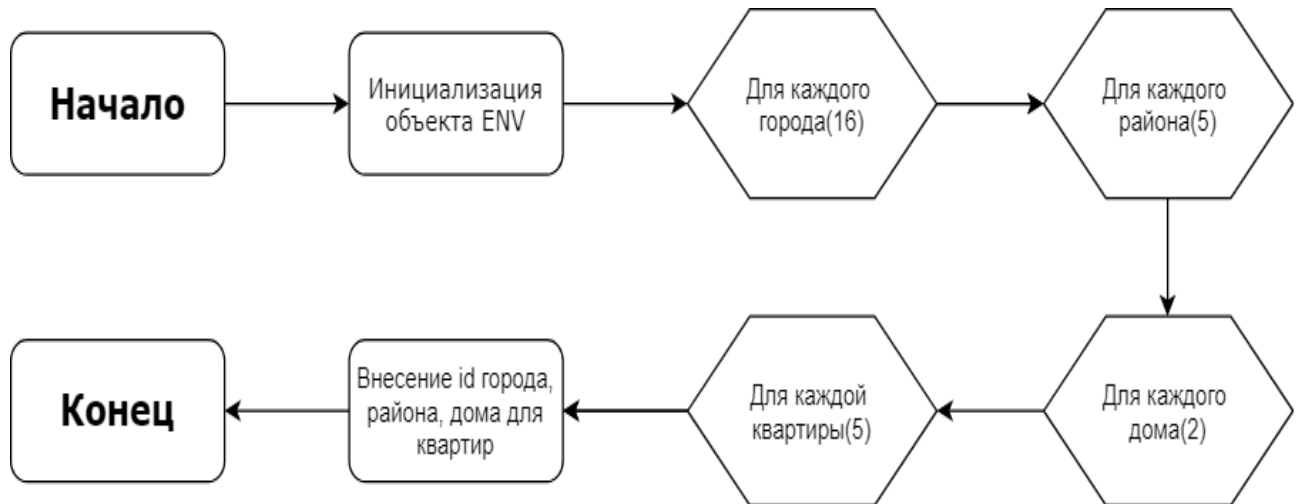
Раздел Backend производит визуальное отображение информации в качестве графиков и диаграмм, которое реализовано с помощью Matplotlib, также здесь осуществлена некоторая обработка информации в соответствии с регламентом, например, определение максимальных значений температуры.

DB хранит информацию, полученную с сервера в виде нескольких таблиц: таблица показаний уличной температуры в городах, таблица показаний температуры в множестве квартир одного города, таблица показаний температуры в одной квартире в каждом из городов, таблица показаний температуры в квартирах в нескольких (5) районах одного города.

Parser производит обработку данных полученных сервиса и передаёт их в DB в соответствующие таблицы. В том числе, с помощью модуля requests здесь извлекается токен для взаимодействия с сервиса, производится извлечение данных на разных уровнях сервера; создаются «цели» для создания графика «температур множества квартир» и здесь же находится подпрограмма, отвечающая за время сбора данных с сервиса, реализованная с помощью модуля time

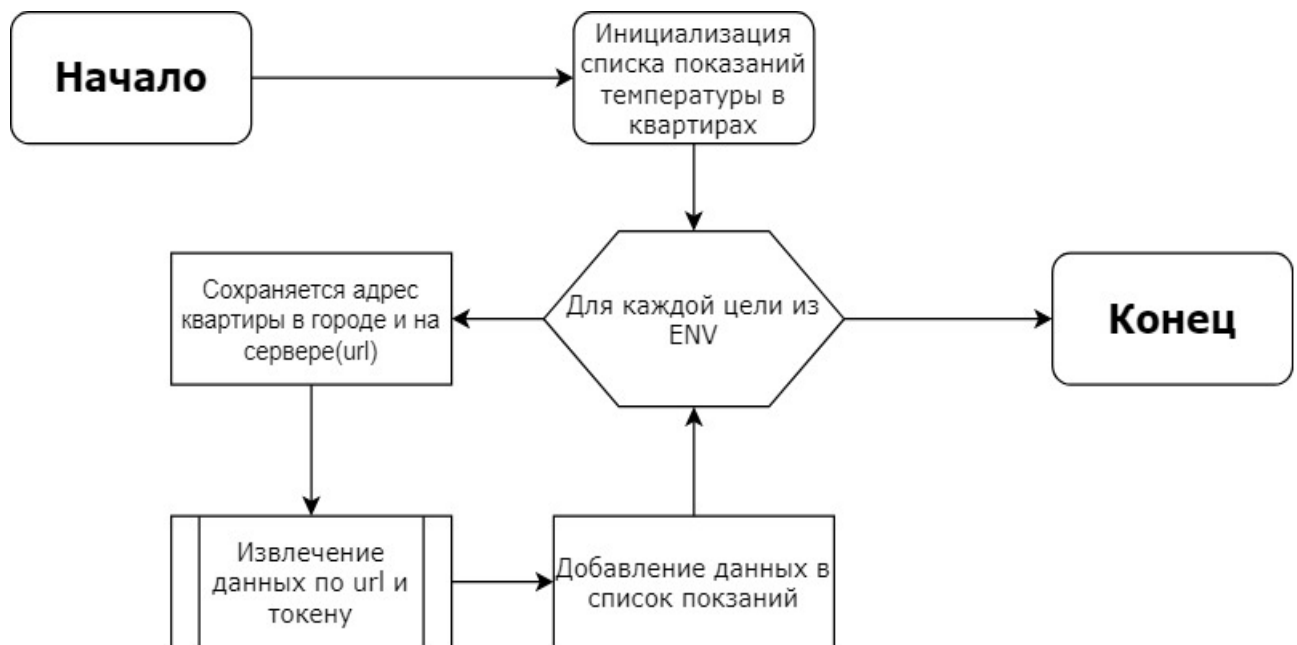
Ниже представлены функциональные схемы работы некоторых частей алгоритма:

1. Загрузка «целей» - адресов квартир, из которых будут считываться данные температуры



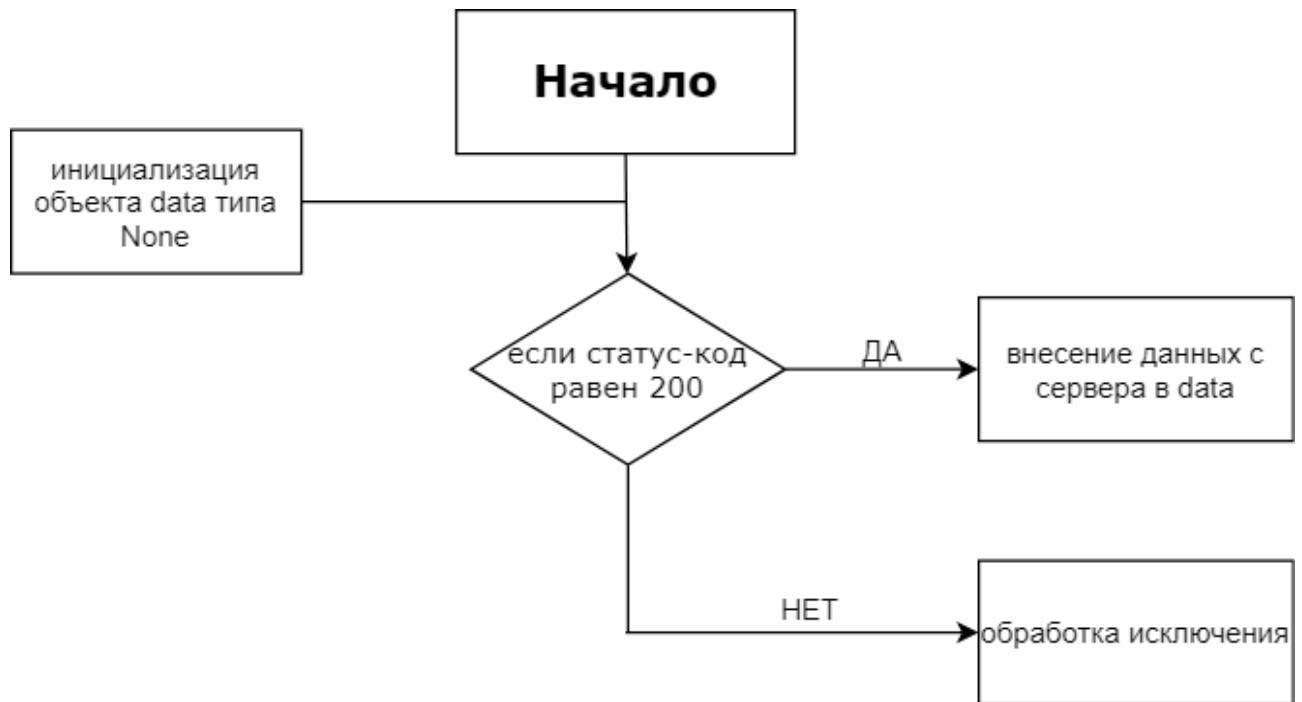
ENV – это объект типа dict, содержащий список целей targets, url-адрес сервиса, token и время работы программы.

2. Сбор данных о каждой цели



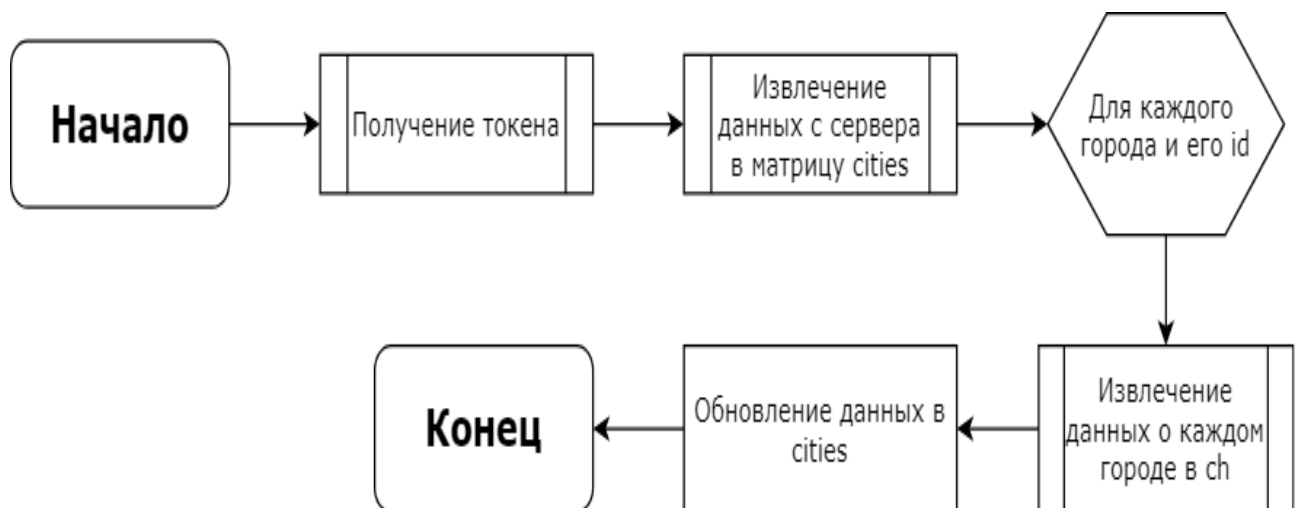
Алгоритм возвращает список показаний температуры в квартирах

3. Извлечение данных с сервера



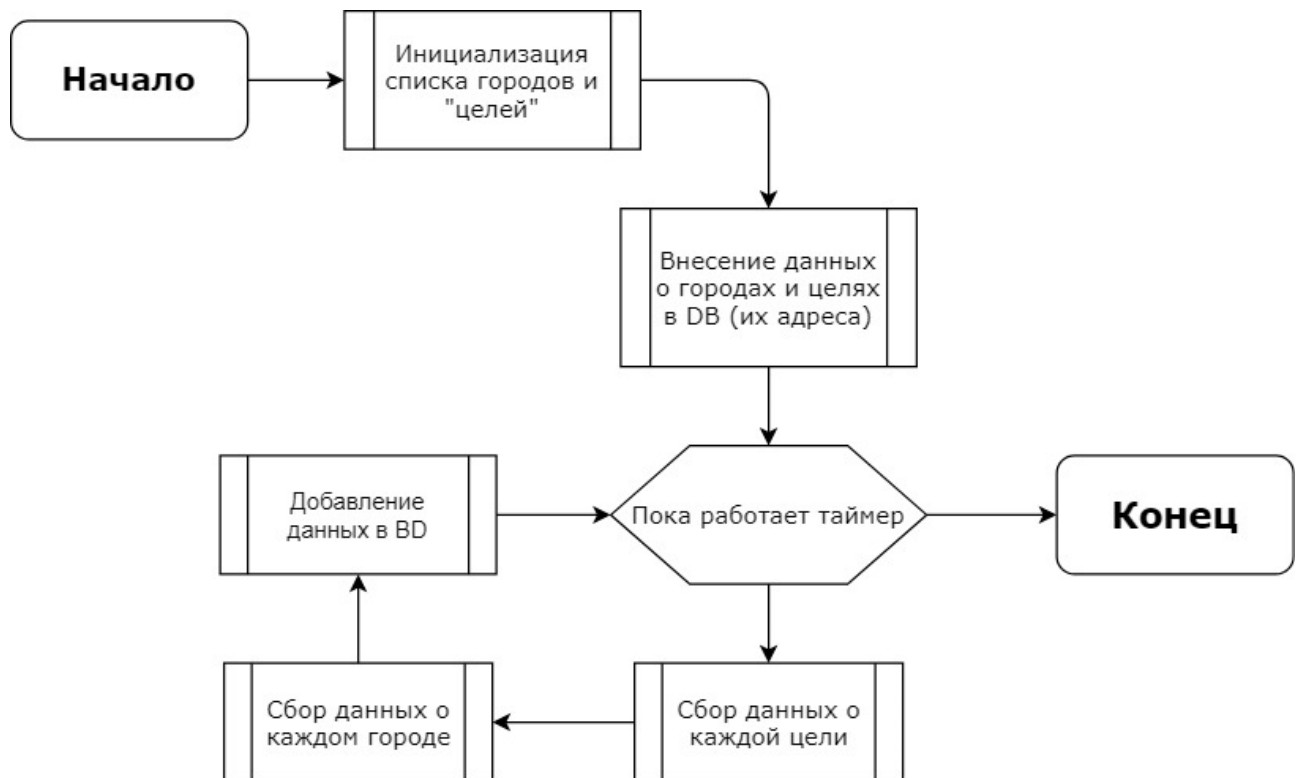
Алгоритм возвращает data в формате json

4. Извлечение подробных данных о каждом городе



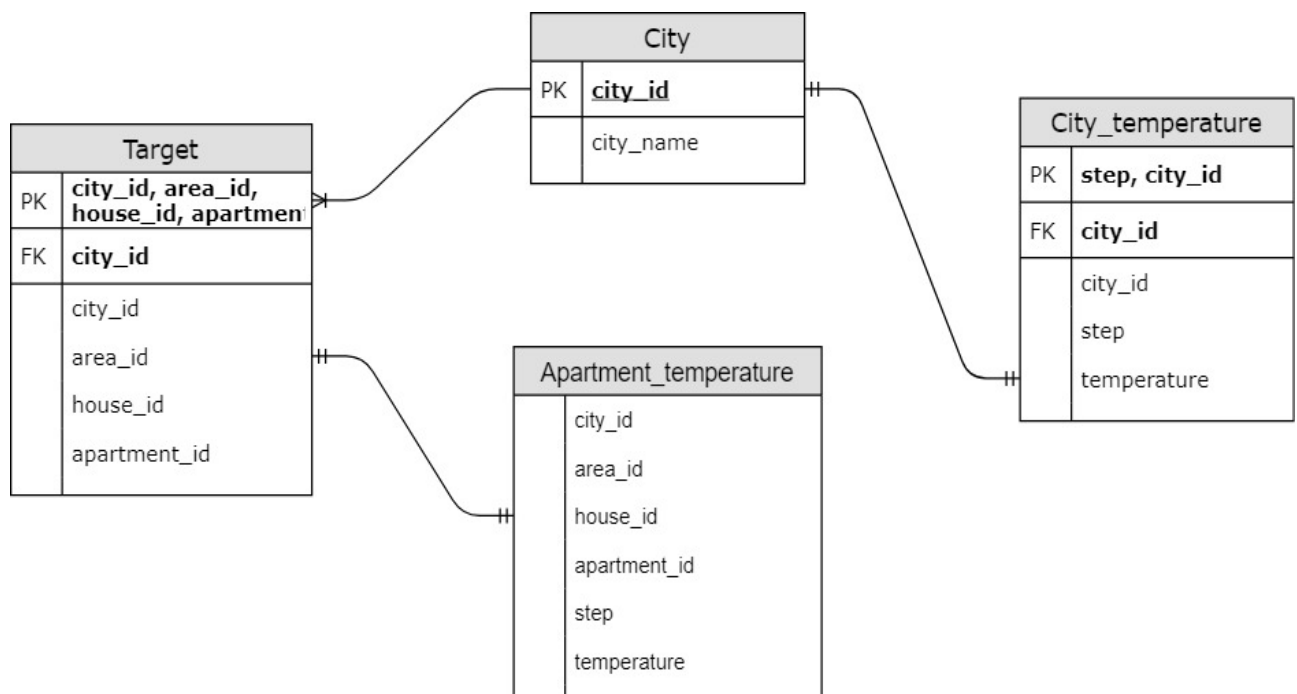
Дополнительно, перед использованием объектов cities и ch они проверяются на корректность (не тип None). Алгоритм возвращает cities – объект типа dict. Использование этой программы подразумевает постоянные изменения в показаниях температуры

5. Основной алгоритм



База данных

6. ER-модель



Результаты разработки

Реализован программный продукт, позволяющий выводить данные температуры в различных квартирах и регионах. Разработан программный модуль, обеспечивающий взаимодействие с сервисом. Полученная с него

информация хранится в реляционной базе данных. Взаимодействие с пользователем происходит через специализированный UI. Интерфейс **является самоадаптирующимся для различных ОС/имеет единый стиль, общий для всех ОС/гибридное решение.**

Выполнены все пункты технического задания, описанные в начале документации.

Внешний вид интерфейса

Город

Алмазный ▾

Район

1 ▾

Дом

1 ▾

Квартира

1 ▾

23 °C

Программный код

Модуль 1

```
1. #!/usr/bin/env python3
2. u"""Модуль сбора данных."""
3. import os
4. import json
5. import requests
6. from stimer import STimer
7. from random import shuffle
8. from database import DB
9.
10. ENV = {
11.     "token_path": "../token",
12.     "cache": ".cache.json",
13.     "targets": [],
14.     "url": "http://dt.miet.ru/ppo_it/api",
15.     "dbname": "../database.db",
16.     "delay": 60,
17.     "end": 24 * 3600
18. }
19.
20. with open(ENV["token_path"]) as file:
21.     ENV["token"] = file.read().strip()
22.
23.
24. def init_random(size, max_val):
25.     """Генерация диапазона случайных значений."""
26.     keys = list(range(1, max_val + 1))
27.     shuffle(keys)
28.     return dict.fromkeys(keys[0:size])
29.
```

```

30.
31. def gen_random_targets(cities):
32.     """Генерация целей для сбора данных."""
33.     targets = {}
34.     for city in cities:
35.         # случайные номера значения районов
36.         areas = init_random(4, city["area_count"])
37.         for i, area_id in enumerate(areas.keys()):
38.             # список для хранения номеров домов для текущего города и района
39.             # запрос данных для
40.             res = load_data(f"{ENV['url']}/{city['city_id']}/{area_id}")
41.             houses = init_random(2, len(res["data"]["data"]))
42.             for j, house_id in enumerate(houses.keys()):
43.                 url = f"{ENV['url']}/{city['city_id']}/{area_id}/{house_id}"
44.                 res2 = load_data(url)
45.                 size = 1
46.                 if i == 3 and j == 1:
47.                     size += 2
48.                 apartments = init_random(
49.                     size, res2["data"]["data"]["apartment_count"]
50.                 )
51.                 houses[house_id] = list(apartments.keys())
52.                 areas[area_id] = houses
53.                 targets[city["city_id"]] = areas
54.     return targets
55.
56.
57. def make_targets(cities):
58.     """Создание списка целей."""
59.     targets_map = gen_random_targets(cities)
60.     targets = []
61.     for city_id in targets_map.keys():
62.         target = {}
63.         target["city_id"] = city_id
64.         for area_id in targets_map[city_id]:
65.             target["area_id"] = area_id
66.             for house_id in targets_map[city_id][area_id]:
67.                 target["house_id"] = house_id
68.                 for apartment_id in targets_map[city_id][area_id][house_id]:
69.                     target["apartment_id"] = apartment_id
70.             targets.append(target.copy())
71.     return targets
72.
73.
74. def load_data(url=ENV["url"], token=ENV["token"]):
75.     """Функция загрузки данных."""
76.     result = {"data": None, "error": None}
77.     try:
78.         print(f">>> REQ: {url} ")
79.         res = requests.get(url, headers={"X-Auth-Token": token})

```

```

80.         print(f"<<< RES: {res.status_code}\n\t{res.text}")
81.         if res.status_code == 200:
82.             result["data"] = res.json()
83.         else:
84.             result["error"] = res.text
85.     except Exception as error:
86.         print(error)
87.     return result
88.
89.
90. def get_cities():
91.     res = load_data()
92.     cities = []
93.     if res["data"] is not None:
94.         cities = res["data"]["data"]
95.     return cities
96.
97.
98. def get_cities_data(cities=None):
99.     """ извлечение подробных данных о каждом городе """
100.    if cities is None:
101.        cities = get_cities()
102.    for city in cities:
103.        res = load_data(f"{ENV['url']}/{city['city_id']}")
104.        if res["data"] is not None:
105.            city.update(res["data"]["data"])
106.    return cities
107.
108.
109.    def get_apatments_data(targets):
110.        """сбор данных о каждой цели"""
111.        data_set = []
112.        for target in targets:
113.            url =
114.                f"{ENV['url']}/{target['city_id']}/{target['area_id']}/{target['house_id']}/{target['apartment_id']}"
115.            res = load_data(url, ENV["token"])
116.            dataset = {
117.                "city_id": target['city_id'],
118.                "area_id": target['area_id'],
119.                "house_id": target['house_id'],
120.                "apartment_id": target['apartment_id'],
121.                "temperature": None
122.            }
123.            if res["data"] is not None:
124.                dataset["temperature"] = res["data"]["data"]["temperature"]
125.            else:
126.                dataset["temperature"] = -273
127.            data_set.append(dataset)
128.        return data_set

```

```

128.
129.
130.     def get_one_target(city, area, house, apartment):
131.         """Сбор данных об одной цели в реальном времени"""
132.         data = load_data(f"{ENV['url']}/{city}/{area}/{house}/{apartment}")
133.         return data
134.
135.     def read_cache():
136.         """Чтение закешированных значений городов и целей"""
137.         data = None
138.         if os.path.isfile(ENV["cache"]):
139.             with open(ENV["cache"]) as fd:
140.                 data = json.load(fd)
141.         return data
142.
143.     def write_cache():
144.         with open(ENV["cache"], "w") as fd:
145.             json.dump({"cities": ENV["cities"], "targets": ENV["targets"]}, fd)
146.
147.     def initialize():
148.         """Функция инициализации."""
149.         data = read_cache()
150.         if data is not None:
151.             ENV["cities"] = data["cities"]
152.             ENV["targets"] = data["targets"]
153.         else:
154.             # получение списка городов
155.             ENV["cities"] = get_cities()
156.             # генерация списка целей
157.             ENV["targets"] = make_targets(get_cities_data(ENV["cities"]))
158.             write_cache()
159.
160.
161.     def main():
162.         """Главная функция."""
163.         initialize()
164.         if os.path.isfile(ENV["dbname"]):
165.             os.remove(ENV["dbname"])
166.         db = DB(ENV["dbname"])
167.         db.add_cities(ENV["cities"])
168.         db.add_targets(ENV["targets"])
169.         timer = STimer(ENV["end"])
170.         count = 0
171.         while not timer.is_stop():
172.             # Главный цикл сбора данных
173.             cities_data = get_cities_data(ENV["cities"])
174.             targets_data = get_apatments_data(ENV["targets"])
175.             print(f"Received {len(cities_data) + len(targets_data)} out of
176.                 {len(ENV['cities'])+len(ENV['targets'])} objects")
177.             print("Saving...")

```

```

177.         for city in get_cities_data(ENV["cities"]):
178.             db.add_city_temperature(
179.                 city_id=city["city_id"],
180.                 step=count,
181.                 temperature=city["temperature"]
182.             )
183.         for target in get_apatments_data(ENV["targets"]):
184.             db.add_apartment_temperature(
185.                 city_id=target["city_id"],
186.                 area_id=target["area_id"],
187.                 house_id=target["house_id"],
188.                 apartment_id=target["apartment_id"],
189.                 step=count,
190.                 temperature=target["temperature"]
191.             )
192.         count += 1
193.         print(f"Timeout {ENV['delay']}s.")
194.         timer.sleep(ENV['delay'])
195.
196.     if __name__ == "__main__":
197.         main()

```

База данных

```

1.  #!/usr/bin/env python3
2.  """Модуль работы с БД sqlite3."""
3.  import sqlite3
4.  import sys
5.
6.
7.  class DB:
8.      """Класс-обертка над sqlite3."""
9.
10.     def __init__(self, name: str):
11.         """Метод инициализации.
12.            name - имя БД
13.            """
14.         # открытие соединения с БД
15.         self.db_connection = sqlite3.connect(name)
16.         # создание таблиц в БД
17.         self._create_tables()
18.
19.     def __del__(self):
20.         """Destructor."""
21.         # закрытие соединения с БД
22.         self.db_connection.close()
23.
24.

```

```

25.     def add_city_temperature(self, *, city_id, step, temperature):
26.         """Метод добавляет запись в таблицу city_temperature.
27.             city_id      - идентификатор (номер) города
28.             step         - шаг измерений
29.             temperature  - значение температуры
30.         """
31.         try:
32.             self.db_connection.executescript(
33.                 "\n".join([
34.                     "INSERT INTO city_temperature VALUES (",
35.                     f"{city_id},",
36.                     f"{step},",
37.                     f"{temperature}",
38.                     f");"
39.                 ])
40.             )
41.         except Exception as err:
42.             # вывод сообщения об ошибке в стандартный поток ошибок
43.             print(err)
44.
45.     def add_apartment_temperature(self, *, city_id, area_id, house_id,
46.         apartment_id, step, temperature):
47.         """Метод добавляет запись в таблицу apartment_temperature.
48.             city_id      - идентификатор (номер) города
49.             area_id      - идентификатор (номер) района
50.             house_id     - идентификатор (номер) дома
51.             step         - шаг измерений
52.             temperature  - значение температуры
53.         """
54.         try:
55.             self.db_connection.executescript(
56.                 "\n".join([
57.                     "INSERT INTO apartment_temperature VALUES (",
58.                     f"{city_id},",
59.                     f"{area_id},",
60.                     f"{house_id},",
61.                     f"{apartment_id},",
62.                     f"{step},",
63.                     f"{temperature}",
64.                     f");"
65.                 ])
66.             )
67.         except Exception as err:
68.             # вывод сообщения об ошибке в стандартный поток ошибок
69.             sys.stderr.write(err)
70.
71.     def get_apartment_temperature(self, *, city_id, area_id, house_id,
72.         apartment_id):
73.         """
74.         return self.db_connection.executescript(

```



```

73.         "\n".join([
74.             "SELECT * FROM apartment_temperature WHERE",
75.             f"city_id = {city_id} and",
76.             f"area_id = {area_id} and",
77.             f"house_id = {house_id} and",
78.             f"apartment_id = {apartment_id};"
79.         ])
80.     )
81.
82.     def add_targets(self, targets):
83.         """Добавление целей для сбора данных."""
84.         sql = "INSERT INTO target VALUES"
85.         values = ""
86.         for target in targets:
87.             sql += f"\n\t({target['city_id']}, {target['area_id']},
            {target['house_id']}, {target['apartment_id']}),"
88.         sql = f"{sql[0:-1]};"
89.         self.db_connection.executescript(sql)
90.
91.     def add_cities(self, cities):
92.         """Добавление городов в БД"""
93.         sql = "".join([
94.             "INSERT INTO city VALUES\n\t",
95.             ',\n\t'.join([f'({city["city_id"]}, \'{city["city_name"]}\')' for city
            in cities]),
96.             ";"
97.         ])
98.         self.db_connection.executescript(sql)
99.
100.    def _create_tables(self):
101.        """Создание таблицы БД"""
102.        SQL = """PRAGMA FOREIGN_KEYS = on;
103.        CREATE TABLE IF NOT EXISTS city(
104.            city_id int PRIMARY KEY NOT NULL,
105.            city_name text NOT NULL
106.        );
107.
108.        CREATE TABLE IF NOT EXISTS city_temperature(
109.            city_id int NOT NULL,
110.            step int NOT NULL,
111.            temperature int NOT NULL,
112.            CONSTRAINT pk_city_temperature PRIMARY KEY (step, city_id),
113.            FOREIGN KEY (city_id) REFERENCES city(city_id)
114.        );
115.        CREATE INDEX IF NOT EXISTS idx_city_tmpr_cid ON
            city_temperature(city_id);
116.
117.        CREATE TABLE IF NOT EXISTS target(
118.            city_id int NOT NULL,
119.            area_id int NOT NULL,

```

```

120.             house_id int NOT NULL,
121.             apartment_id int NOT NULL,
122.             CONSTRAINT pk_target PRIMARY KEY (city_id, area_id,
             house_id, apartment_id),
123.             FOREIGN KEY (city_id) REFERENCES city(city_id)
124.         );
125.         CREATE INDEX IF NOT EXISTS idx_target_cid ON target(city_id);
126.         CREATE INDEX IF NOT EXISTS idx_target_cid_aid ON target(city_id,
             area_id);
127.         CREATE INDEX IF NOT EXISTS idx_target_cid_aid_hid ON
             target(city_id, area_id, house_id);
128.
129.         CREATE TABLE IF NOT EXISTS apartment_temperature(
130.             city_id int NOT NULL,
131.             area_id int NOT NULL,
132.             house_id int NOT NULL,
133.             apartment_id int,
134.             step int,
135.             temperature int
136.         );
137.         CREATE INDEX IF NOT EXISTS idx_aprtm_tmpr ON
             apartment_temperature(city_id, area_id, house_id, apartment_id);
138.         """
139.         self.db_connection.executescript(SQL)

```

Таймер

```

1  #!/usr/bin/env .python3
2  """The Simple timer."""
3  import time
4
5
6  class STimer:
7      """The simple timer class."""
8
9      def __init__(self, duration):
10         """Initializer."""
11         self.start = self.timestamp()
12         self.duration = duration
13
14     @classmethod
15     def timestamp(cls):
16         """Current time stamp."""
17         return int(time.time())
18
19     def is_stop(self):
20         """Check."""
21         return self.timestamp() - self.start >= self.duration
22
23     def sleep(self, nsecond=0):
24         """Sleep."""
25         time.sleep(nsecond)

```

