

# Индивидуальное задание на курсовое проектирование

Тема: «Разработка многопоточного агрегатора данных из REST API»

## 1. Постановка задачи

### 1.1 Общая формулировка

В рамках курсового проекта необходимо разработать консольное Java-приложение для агрегации данных из нескольких открытых REST API.

Приложение должно уметь:

- Получать данные как минимум из трёх различных REST API по HTTP
- Сохранять агрегированные данные в файл заданного формата и читать из него
- Выполнять опрос источников параллельно, с контролем количества одновременно выполняемых задач и интервалов между запросами

Проект реализуется поэтапно, для получения зачёта необходимо сдать все три этапа.

### 1.2 Первый этап

Выбрать минимум три открытых API, удовлетворяющих следующим требованиям:

- Получаемый в качестве ответа JSON должен содержать не менее 5 полей, среди которых как минимум один вложенный объект или массив
- API должен поддерживать отправку query-параметров
- Данные должны изменяться по времени или в зависимости от параметров

Затем, реализовать однопоточное консольное java-приложение, которое:

- Может быть запущено в двух режимах:
  - Автоматический – принимает на вход набор параметров:
    - Список используемых API
    - Формат выходного файла (CSV или JSON)
  - Интерактивный
    - Пользователь может выбрать API для опроса из предложенного списка
    - Задать формат выходного файла (CSV или JSON)
    - Выбрать режим работы с выходным файлом – создать новый или дозаписать в существующий
    - Вывести содержимое файла на экран либо целиком, либо по конкретному API
- Отправляет HTTP-запрос к выбранным API
- Обрабатывает JSON-ответы и преобразует данные в java-объекты
- Сохраняет агрегированные данные в файл заданного формата (см. раздел 4)

Требования к реализации:

- Приложение должно быть реализовано с использованием принципов ООП
- Архитектура должна позволять добавление нового API без изменения основной логики приложения
- Логика получения, обработки и сохранения данных должна быть разделена по ответственности
- Логика пользовательского ввода должна быть отделена от бизнес-логики
- Запрещается смешивать логику CLI и логику агрегации данных
- Запрещается выполнять HTTP-запросы прямо из обработчиков ввода
- Проект должен собираться с использованием Maven или Gradle

### 1.3 Второй этап

Дополнить написанное приложение, реализовав:

- Расширение списка входных параметров
  - Максимальное количество одновременно выполняемых задач  $n$
  - Интервал опроса источников  $t$
  - Для интерактивного режима – запуск и остановка опроса
- Параллельный опрос API
- Управление количеством одновременно выполняемых задач
- Периодический опрос источников с заданным интервалом

Требования к реализации:

- Для управления потоками использовать средства пакета `java.util.concurrent`
- Максимальное количество одновременно выполняемых задач ограничивается параметром  $n$
- Повторный опрос одного и того же API должен происходить не чаще, чем через  $t$  секунд после завершения предыдущего запроса
- Запись данных в файл должна быть потокобезопасной
- Приложение должно корректно завершать работу (освобождать ресурсы, останавливать потоки)
- Логика управления потоками должна быть отделена от бизнес-логики приложения

### 1.4 Третий этап

Завершающий этап разработки:

- Реализовать unit-тесты для ключевой бизнес-логики и пользовательского ввода
- Подготовить отчёт по курсовой, содержащий:
  - Описание назначения приложения

- Описание архитектуры
- Схему обработки данных
- Инструкцию по сборке и запуску

Требования к реализации тестов:

- Unit-тестами должна быть покрыта бизнес-логика проекта
- В unit-тестах не должны выполняться сетевые вызовы
- Минимальное покрытие – 70% по критерию строк (line coverage) согласно отчёту инструмента покрытия (например, JaCoCo)

## 2. Список источников по темам, отсутствующим в лекциях

### 1. Про REST API:

- <https://yandex.cloud/ru/docs/glossary/rest-api>
- <https://gb.ru/blog/rest-api/>
- <https://restfulapi.net/>

### 2. Про форматы JSON и CSV:

- <https://tproger.ru/articles/что-такое-json-введение>
- <https://www.json.org/json-en.html>
- <https://ru.wikipedia.org/wiki/CSV>
- <https://www.rfc-editor.org/rfc/rfc4180>

### 3. Списки открытых API:

- <https://github.com/public-apis/public-apis>
- <https://github.com/public-api-lists/public-api-lists>
- <https://habr.com/ru/articles/769384/>
- <https://developer.donnoval.ru/free-api/>

### 4. Мокирование в unit-тестах:

- <https://site.mockito.org/>
- <https://www.baeldung.com/mockito-series>

## 3. Рекомендованные библиотеки и инструменты

Использование указанных ниже библиотек не является обязательным. Вы в праве выбрать любые другие инструменты при условии выполнения требований задания.

Работа с REST API:

- Apache HttpClient - <https://hc.apache.org/httpcomponents-client-5.6.x/>
- OkHttp - <https://square.github.io/okhttp/>

Парсинг:

- Jackson - <https://github.com/FasterXML/jackson>
- Gson - <https://github.com/google/gson>

- Apache Commons CSV - <https://commons.apache.org/proper/commons-csv/>
- OpenCSV - <http://opencsv.sourceforge.net/>

Тестирование:

- JUnit 6 - <https://junit.org/>
- Mockito - <https://site.mockito.org/>

#### 4. Рекомендуемый формат сохранения данных

В данном пункте предложен один из вариантов сохранения данных, полученных из API. Вы также можете продумать свой формат и согласовать его с преподавателем.

При сохранении данных в формате JSON рекомендуется использовать следующую структуру:

```
[
  {
    "id": 1, // идентификатор записи (в формате
    // инкрементируемого числа или UUID)
    "source": "api_name1", // название api
    "timestamp": "2026-02-04T11:56:23+00:00", // время
    // сохранения записи (в формате ISO 8601 или Epoch Unix Timestamp)
    "data": {
      // json-запись из api_name1
    }
  }
]
```

При сохранении данных в формате CSV рекомендуется использовать денормализованную таблицу:

```
Id,source,timestamp,api_name1_field_name1,api_name1_field_name1,api_name2_field_name1,api_name2_field_name2,...
1,api_name1,2026-02-04T11:56:23+00:00,a1f1_value,a1f2_value,,
2,api_name2,2026-02-04T11:58:21+00:00,,,a2f1_value,a2f2_value
```

Рассмотрим на примере, пусть извлечено 2 записи из различных API:

1. Запись о заказе из онлайн магазина “Самокат”:

```
{
  "id": 12,
  "timestamp": "2024-01-06T16:45:22+00:00",
  "customer": {
    "id": 1,
    "name": "Petr",
    "surname": "Ivanov",
    "phone_number": "+70123456789"
  }
}
```

```

    },
    "delivery_address": {
      "city": "Saint-Petersburg",
      "street": "Nevskiy pr.",
      "home": 3
    },
    "items": [
      {
        "id": 1,
        "name": "Milk Parmalat",
        "price": 124
      },
      {
        "id": 2,
        "name": "Bread Harris",
        "price": 85
      }
    ]
  }
}

```

## 2. Запись о текущей погоде в сервисе “Яндекс погода”:

```

{
  "id": 14,
  "timestamp": "2024-01-15T14:30:00+03:00",
  "location": {
    "city": "Москва",
    "country": "Россия",
    "coordinates": {
      "latitude": 55.7558,
      "longitude": 37.6173
    },
    "timezone": "Europe/Moscow"
  },
  "current_weather": {
    "temperature": {
      "celsius": -5.2,
      "fahrenheit": 22.6
    }
  }
}

```

Результат сохранения записей:

### 1. В формате JSON:

```

[
  {
    "id": 1,
    "source": "samokat",
    "timestamp": "2026-02-04T11:56:23+00:00",
    "data": {
      "id": 12,
      "timestamp": "2024-01-06T16:45:22+00:00",
      "customer": {
        "id": 1,
        "name": "Petr",
        "surname": "Ivanov",
        "phone_number": "+70123456789"
      },
      "delivery_address": {
        "city": "Saint-Petersburg",
        "street": "Nevskiy pr.",

```

```

        "home": 3
      },
      "items": [
        {
          "id": 1,
          "name": "Milk Parmalat",
          "price": 124
        },
        {
          "id": 2,
          "name": "Bread Harris",
          "price": 85
        }
      ]
    },
    {
      "id": 2,
      "source": "yandex_weather",
      "timestamp": "2026-02-04T11:58:21+00:00",
      "data": {
        "id": 14,
        "timestamp": "2024-01-15T14:30:00+03:00",
        "location": {
          "city": "Москва",
          "country": "Россия",
          "coordinates": {
            "latitude": 55.7558,
            "longitude": 37.6173
          }
        },
        "timezone": "Europe/Moscow"
      },
      "current_weather": {
        "temperature": {
          "celsius": -5.2,
          "fahrenheit": 22.6
        }
      }
    }
  ]
}

```

## 2. В формате CSV:

```

id,source,timestamp,data.id,data.timestamp,customer.id,customer.name,customer.sur
name,customer.phone_number,delivery_address.city,delivery_address.street,delivery_a
ddress.home,items.id,items.name,items.price,data.location.city,data.location.country,d
ata.location.coordinates.latitude,data.location.coordinates.longitude,data.location.time
zone,data.current_weather.temperature.celsius,data.current_weather.temperature.fahre
nheit
1,samokat,2026-02-04T11:56:23+00:00,12,2024-01-
06T16:45:22+00:00,1,Petr,Ivanov,+70123456789,Saint-Petersburg,Nevskiy pr.,3,1,Milk
Parmalat,124,,,,,
1,samokat,2026-02-04T11:56:23+00:00,12,2024-01-
06T16:45:22+00:00,1,Petr,Ivanov,+70123456789,Saint-Petersburg,Nevskiy pr.,3,2,Bread
Harris,85,,,,,
2,yandex_weather,2026-02-04T11:58:21+00:00,14,2024-01-
15T14:30:00+03:00,,,,,,,,,Москва,Россия,55.7558,37.6173,Europe/Moscow,-5.2,22.6

```