Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

Институт информационных технологий и компьютерных наук
Кафедра инженерной кибернетики

Курсовая работа

по дисциплине «Технологии программирования»

по теме:

«ПО для визуализации временных рядов»

Выполнил:
Студент 4-го курса,
гр. БПМ-21-2 Сироткин С.Ю.

Проверил:
доцент, к.т.н. Полевой Д.В.

Москва, 2025

СОДЕРЖАНИЕ

1 Описание задачи

1.1 Краткое описание задачи

В условиях современного управления системами метрики играют ключевую роль в мониторинге и анализе производительности приложений и инфраструктуры. Существующие решения, такие как Grafana, предоставляют мощные инструменты для визуализации данных, однако могут оказаться избыточными для простых задач. Данная работа заключается в разработке легковесного десктопного приложения для визуализации метрик. Программа позволит пользователям подключаться к разным источникам данных, таким как Prometheus и другим базам данных временных рядов (TSDB, Time Series DataBase), и предоставит функционал для выбора данных и временных интервалов.

Рисунок 1 – интерфейс Grafana

1.2 Функциональные требования

- Основные характеристики разрабатываемого ПО:
  - Десктопное приложение с графическим интерфейсом
  - Библиотеки для получения данных из TSDB (отдельный клиент для каждой системы, приложение является потребителем библиотек)
- Выбор источника данных**:**
  - Поддержка подключения к различным источникам данных: Prometheus и других TSDB.
  - Интерфейс для ввода необходимых параметров подключения.

    - Удобный интерфейс для указания имени метрики и ее меток.
- Ввод запроса
    - Ввод запроса на языке, поддерживаемом соответствующей tsdb (`PromQL` и другие)
    - Выбор временного интервала:
    - Возможность выбора временного диапазона для отображаемых данных с удобным интерфейсом (например, последние 1 час, 24 часа или пользовательский интервал).
- Типы визуализации:
    - Поддержка нескольких типов графиков: line, column, area и т.д.
    - Предоставление пользователю возможности выбора типа графика для отображаемых данных.
- Автообновление:
    - Функция автоматического обновления графиков через заданный интервал времени.
    - Настройка интервала обновления пользователем для мониторинга системы в реальном времени.
- Отображение графиков:
    - Интерактивная визуализация временных рядов.
    - Поддержка масштабирования и навигации по графикам.

1.3 Нефункциональные требования

- Технологический стек:
    - Язык программирования: C++
    - Библиотека визуализации: ImPlot
    - Библиотека интерфейса пользователя: ImGui (в качестве основы для ImPlot)
    - Документация doxygen
    - Тесты doctest

2 Результат

2.1 Архитектура и дизайн системы

Система реализована по принципу разделения инфраструктурного слоя, отвечающего за взаимодействие с базами данных, от слоя приложения. Архитектура ориентирована на модульность и масштабируемость.

Графический интерфейс, построенный на библиотеках ImGui и ImPlot, тесно интегрирован с логикой приложения: виджеты управления (выбор метрик, настройка временного диапазона, тип графика и т.д.) напрямую взаимодействуют с механизмами формирования запросов. Основная бизнес-логика сосредоточена в модуле app, который обрабатывает пользовательский ввод, преобразует его в запросы к TSDB, выполняет прореживание данных для визуализации и управляет обновлением графиков.

Инфраструктурный слой выделен в отдельную библиотеку tsdb и построен вокруг абстрактного класса TSDBClient, декларирующего базовые методы (листинг 1).

Листинг 1 – интерфейс базового клиента к TSDB

```
class TSDBClient
{
public:
    virtual std::vector<Metric> query(const std::string &query_str,
std::time_t start, std::time_t end) = 0;

    virtual bool isAvailable() noexcept = 0;

    static std::string format_line_name(const std::string &name, const
std::map<std::string, std::string> &labels);

protected:
    virtual std::string performHttpRequest(const std::string &url, int
timeout = 5);
};
```

В рамках работы был реализован клиент к Prometheus. Для добавления новых клиентов к TSDB достаточно реализовать базовый интерфейс.

Наиболее критичные компоненты покрыты юнит-тестами с использованием фреймворка doctest. Проект собирается с помощью CMake кроссплатформенно. Код документирован с использованием Doxygen. В GitHub-репозитории проекта (приложение 1) настроены ci/cd для документации и воркфлоу для сборки и запуска юнит-тестов. Документация автоматически собирается и релизится в GitHub Pages (приложение 2) при коммите в основную ветку репозитория.

2.2 Пользовательский путь

1. **Настройка подключения к TSDB:**

Пользователь вводит базовый URL сервера Prometheus (для удобства тестирования был развернут собственный сервер, адрес `http://84.201.168.241:9090` используется по умолчанию) в поле "Prometheus Base URL". Затем нажимает кнопку "Connect" для проверки соединения. В случае успеха отображается статус "Connection successful!".

2. **Ввод PromQL-запроса:**

В разделе "Query" пользователь может указать запрос на языке PromQL для анализа (например, `process_resident_memory_bytes` используется по умолчанию для удобства тестирования).

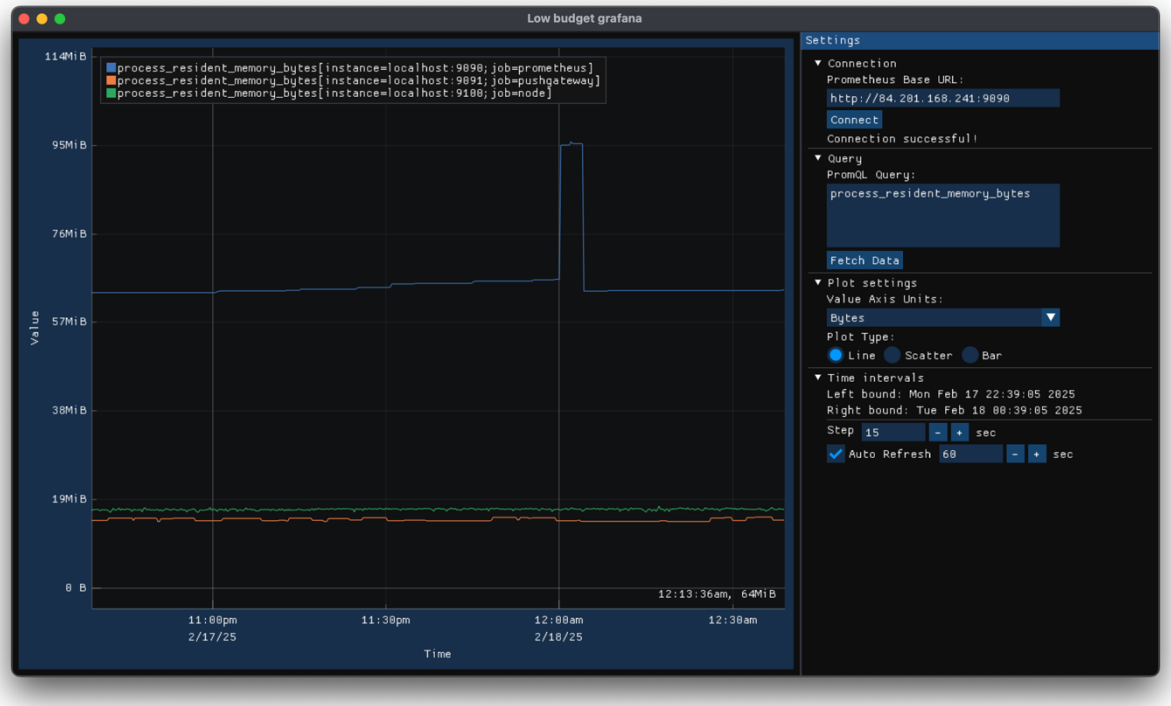

Рисунок 2 – интерфейс разработанного приложения

3. **Конфигурация графика:**

В разделе "Plot settings" выбираются: Единицы измерения оси значений (количество, секунды, байты, проценты). Тип визуализации: линия, точки или столбцы (переключатель между типами).

4. **Настройка временного интервала:**

В этом разделе отображаются точные границы временного интервала. Переместить границы или выбрать с помощью поля ввода пользователь может на графике (см. рис. 3). Также конфигурируется шаг выборки данных (Step, по умолчанию 15 секунд) и интервал

автообновления (Auto Refresh, по умолчанию 1 минута).

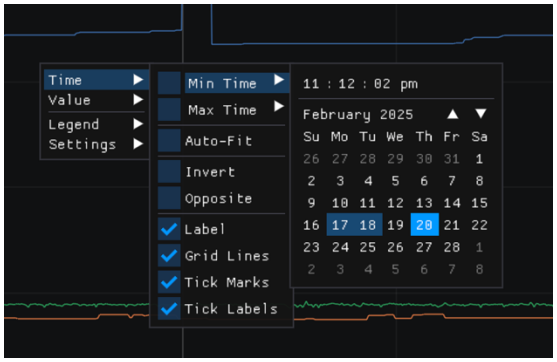

Рисунок 3 – выбор временного интервала

5. **Работа с временной шкалой:**

Пользователь может geретаскивать границы временного диапазона, увеличивать/уменьшать масштаб для детализации данных, наблюдать за обновлением графика в реальном времени благодаря функции Auto Refresh.

6. **Анализ нескольких метрик:**

Как видно на рис. 2, интерфейс позволяет одновременно отображать данные нескольких метрик (например, с метками `job=prometheus`, `job=node`). Каждая метрика представлена отдельной линией на графике.

2.3 Детальное описание исходного кода

Проект состоит из двух основных модулей: `src/app/` и `src/lib/`, содержащих входную точку в само приложение и общие переиспользуемые компоненты (библиотеки), соответственно.

2.3.1 Инициализация и запуск цикла рендеринга

В файле `src/app/app.cpp` находится функция main, инициализирующая окно с помощью `GLFW`. Затем запускается основной цикл рендеринга приложения (листинг 2).

Листинг 2 – основной цикл приложения

```
while (!glfwWindowShouldClose(window))
 {
     glfwPollEvents();
     ImGui_ImplOpenGL3_NewFrame();
     ImGui_ImplGlfw_NewFrame();
     ImGui::NewFrame();

     renderUI();

     ImGui::Render();
     glClear(GL_COLOR_BUFFER_BIT);
     ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
```

```
        glfwSwapBuffers(window);
    }
```

2.3.2 Отрисовка элементов интерфейса:

В цикле вызывается функция `renderUI` (листинг 3), вызывающая в свою очередь функции, создающие графический интерфейс с обзором метрик (листинг 4) и настройками (листинг 5).

Листинг 3 – функция отрисовки графического интерфейса

```
void renderUI()
{
    renderMetricsViewer();
    renderSettings();
}
```

Листинг 4 – фрагмент функции отрисовки окна вывода метрик

```
void renderMetricsViewer()
{
    ImGui::SetNextWindowPos(ImVec2(0, 0), ImGuiCond_Always);
    ImGui::SetNextWindowSize(ImVec2(WINDOW_WIDTH - SETTINGS_WIDTH,
WINDOW_HEIGHT), ImGuiCond_Always);
    ImGui::Begin("Metrics Viewer", nullptr,
ImGuiWindowFlags_NoDecoration);

    ImVec2 plotSize = ImVec2(ImGui::GetContentRegionAvail().x,
ImGui::GetContentRegionAvail().y);
    if (ImPlot::BeginPlot("Time Series", plotSize, ImPlotFlags_NoTitle))
    {...}
    ImGui::End();
}
```

Листинг 5 – фрагмент функции отрисовки окна с настройками

```
void renderSettings()
{
    ImGui::SetNextWindowPos(ImVec2(WINDOW_WIDTH - SETTINGS_WIDTH, 0),
ImGuiCond_Always);
    ImGui::SetNextWindowSize(ImVec2(SETTINGS_WIDTH, WINDOW_HEIGHT),
ImGuiCond_Always);
    ImGui::Begin(Strings::WINDOW_SETTINGS, nullptr,
ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove |
ImGuiWindowFlags_NoCollapse);

    if (ImGui::TreeNodeEx(Strings::NODE_CONNECTION,
ImGuiTreeNodeFlags_DefaultOpen))
    {...}

    ImGui::Separator();
    if (ImGui::TreeNodeEx(Strings::NODE_QUERY,
ImGuiTreeNodeFlags_DefaultOpen))
    {...}
```

```
    ImGui::Separator();
    if (ImGui::TreeNodeEx(Strings::NODE_PLOT_SETTINGS,
ImGuiTreeNodeFlags_DefaultOpen))
    {...}

    ImGui::Separator();
    if (ImGui::TreeNodeEx(Strings::NODE_TIME_INTERVALS,
ImGuiTreeNodeFlags_DefaultOpen))
    {...}

    ImGui::End();
}
```

Вспомогательную роль в отрисовке интерфейса выполняют функции из модуля `utils`: `formatTimestamp` – для приведения таймстемпа в ISO формат и `valueTickFormatter` – для форматирования подписей на оси значений на графике (имеет специфичную для `ImPlot` сигнатуру). Этот форматтер выбирает формат записи в зависимости от выбранных единиц измерения и величины значения (листинг 6).

Листинг 6 – фрагмент вспомогательных функций форматирования

```
std::string formatTimestamp(std::time_t timestamp)
{
    std::tm* tm = std::localtime(&timestamp);
    if (!tm) {
        return "";
    }

    std::ostringstream oss;
    oss << std::put_time(tm, "%c");
    return oss.str();
}

int valueTickFormatter(double value, char *buff, int size, void
*user_data)
{
    std::string unit = "";
    std::string value_format = "%.2f";
    switch (*(YAxisUnit *)user_data)
    {
    case YAxisUnit::No:
        break;
    case YAxisUnit::Seconds:
        ...
        break;
    case YAxisUnit::Bytes:
        ...
        break;
    case YAxisUnit::Percents:
```

```
        ...
        break;
    }

    if (abs(value) > 1e12)
        value_format = "%.6e";
    value_format += "%s";
    return ImFormatString(buff, 15, value_format.c_str(), value,
unit.c_str());
}
```

2.3.3 Запрос данных с сервера

При необходимости запроса данных (либо по нажатию кнопки Fetch data, либо по истечении Auto refresh интервала), вызывается функция `fetch_data`, которая формирует запрос и вызывает метод `query` клиента. В случае возникновения ошибки запроса, выводится человекочитаемое сообщение (листинг 7).

Листинг 7 – фрагмент функции запроса и отрисовки данных

```
void fetchData()
{
    showRequestErrorMsg = false;
    if (!prometheusClient)
    {...}

    double interval = rightTimeBound - leftTimeBound;
    int step = selectedStep;
    if (selectedStep * PROMETHEUS_MAX_POINTS_PER_REQUEST < interval)
        step = std::ceil(interval /
(double)PROMETHEUS_MAX_POINTS_PER_REQUEST);

    std::vector<Metric> metrics;
    try
    {
        metrics = prometheusClient->query(queryBuffer, leftTimeBound,
rightTimeBound, step);
    }
    catch (InvalidPrometheusRequest &error)
    {
        showRequestErrorMsg = true;
        requestErrorMsg = error.what();
        return;
    }
    seriesData.clear();
    ...
    for (auto &m : metrics)
    {
        GraphSeries s;
        s.name = prometheusClient->format_line_name(m);
        for (auto &p : m.values)
        {
```

```
            s.x.push_back(static_cast<double>(p.timestamp));
            s.y.push_back(p.value);
        }
        seriesData.push_back(s);
    }
    lastRefreshTime = glfwGetTime();
}
```

2.4 Инструкция по сборке

1. Склонировать репозиторий

```
git clone https://github.com/sssemion/low-budget-grafana.git
cd low-budget-grafana
```

2. Собрать проект

```
cmake -S src -B build -DCMAKE_BUILD_TYPE=Debug -DTEST=ON
cmake --build build
```

2.5 Инструкция по тестированию

Для дальнейшего тестирования необходимо собрать проект, проделав шаги из инструкции по сборке.

2.5.1 Тестирование библиотек

Для тестирования библиотек достаточно запустить юнит-тесты:

```
(cd build; ctest)
```

2.5.2 Тестирование приложения

Тестирование приложения с графическим интерфейсом возможно в ручном режиме. Сперва необходимо запустить собранный исполняемый файл:

```
./build/bin/app
```

1. Установить соединение с базой, нажав на кнопку “Connect” (рис. 4, шаг 1). Для удобства тестирования по умолчанию используется предварительно развернутая в облаке инсталляция Prometheus: адрес `http://84.201.168.241:9090` подставляется в поле “Prometheus Base URL”. В результате должно появится сообщение “Connection successful!”
2. Сделать запрос к базе, нажав на кнопку “Fetch Data” (рис. 4, шаг 2). Для удобства тестирования по умолчанию подставляется запрос “`process_resident_memory_bytes`” – это системная метрика

Prometheus, отражающая объем используемой памяти несколькими процессами в байтах. В результате этого шага в окне графика появятся 3 линии.

3. Выбрать единицы измерения по оси значений (рис. 4, шаг 3). Для метрики по умолчанию нужно выбрать “Bytes”. В результате метки на оси значений примут человекочитаемый вид.
4. Выбрать интервал между точками в ответе от Prometheus (рис. 4, шаг 4). Эта настройка будет применена при следующем запросе к базе.
5. Включить автообновление (рис. 4, шаг 5). В результате каждые N секунд данные будут обновляться, сдвигая правую границу временного окна к текущему времени. Это позволяет следить за графиком в режиме реального времени.
6. Изменить временное окно, двигая мышью временную ось, либо нажав на нее правой кнопкой мыши и введя нужный интервал в форме (рис 4, шаг 6).
7. Изменить запрос, добавив, например, указание конкретной метки, чтобы получить лишь один временной ряд (рис. 4, шаг 7): “`process_resident_memory_bytes{job="node"}`”.
8. Повторно нажать “Fetch Data” (рис. 4, шаг 2). В результате должна появиться лишь одна линия с обновленным временным интервалом.

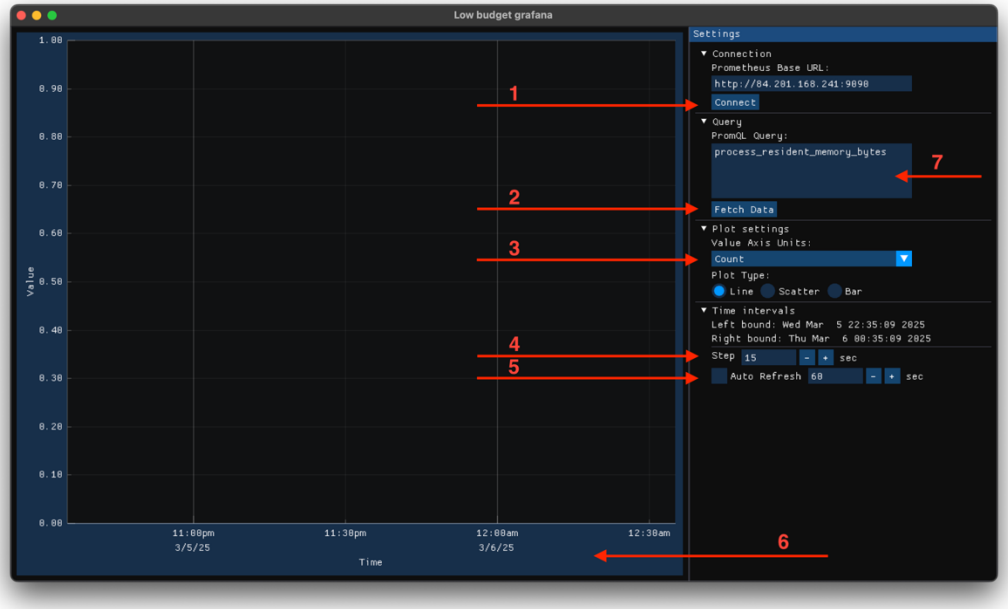

Рисунок 4 – шаги для ручного тестирования

2.6 Техническая документация

2.6.1 Библиотека tsdb

Этот модуль предоставляет интерфейс и структуры данных для взаимодействия с базами данных временных рядов (TSDB).

Классы:

- `InvalidTSDBRequest : std::exception`

  Исключение, возникающее при ошибке запроса к TSDB.

  Методы:

  - `virtual const char *what() const noexcept override`

    Возвращает строковое описание ошибки

  Объявления и описания членов классов находятся в файлах:

  - src/lib/tsdb/tsdb.h
  - src/lib/tsdb/tsdb.cpp

- `TSDBClient`

  Интерфейс клиента к TSDB

  Методы:

  - `static std::string format_line_name(const Metric &metric)`

    Форматирует имя линии графика на основе метрики.

    Аргументы:

    - `metric`: Метрика типа `Metric`.

    Возвращает: Отформатированное имя линии графика с метками.

  - `static std::string format_line_name(const std::string &name, const std::map<std::string, std::string> &labels)`

    Форматирует имя линии графика на основе имени и меток.

    Аргументы:

    - `name`: Имя метрики.
    - `labels`: Словарь меток метрики, где ключ — имя метки, а значение — её значение.

    Возвращает: Отформатированное имя линии графика с метками.

  - `bool isAvailable() noexcept`

    Проверить доступность TSDB.

    Возвращает `true`, если TSDB доступна, иначе `false`.

  - `std::vector<Metric> query(const std::string &query_str, std::time_t start, std::time_t end)`

    Выполнить запрос к TSDB для получения точек.

Аргументы

- `query`: Строка запроса
- `start`: Начало временного диапазона
- `end`: Конец временного диапазона

Возвращает: Массив метрик типа `Metric`

Исключения

- `InvalidTSDBRequest`: В случае неуспешного статуса ответа от TSDB

Объявления и описания членов классов находятся в файлах:

- src/lib/tsdb/tsdb.h
- src/lib/tsdb/tsdb.cpp

- `InvalidPrometheusRequest : InvalidTSDBRequest`

  Исключение, возникающее при ошибке запроса к Prometheus

  Конструкторы:

  - `InvalidPrometheusRequest(const std::string &errorMsg, const std::string &errorType)`

    Аргументы:

    - `errorMsg`: Сообщение об ошибке.
    - `errorType`:Тип ошибки.

  Объявления и описания членов классов находятся в файлах:

  - src/lib/tsdb/prometheus/prometheus.h
  - src/lib/tsdb/prometheus/prometheus.cpp

- `PrometheusClient : TSDBClient`

  Клиент к Prometheus.

  Конструкторы:

  - `PrometheusClient(const std::string &base_url)`

    Аргументы:

    - `base_url`: Базовый URL Prometheus API (например, `"http://localhost:9090"`)

  Методы:

  - `std::vector<Metric> query(const std::string &query_str, std::time_t start, std::time_t end)`

    Выполнить запрос к Prometheus для получения данных.

Аргументы:

- `query_str`: Строка запроса в формате PromQL
- `start`: Начало временного диапазона
- `end`: Конец временного диапазона

Возвращает: Массив метрик типа `Metric`.

Исключения:

- `InvalidPrometheusRequest`: В случае неуспешного статуса ответа от Prometheus

- `std::vector<Metric> query(const std::string &query_str, std::time_t start, std::time_t end)`

  Выполнить запрос к Prometheus для получения данных.

  Аргументы:

  - `query_str`: Строка запроса в формате PromQL
  - `start`: Начало временного диапазона
  - `end`: Конец временного диапазона

  Возвращает: Массив метрик типа `Metric`.

  Исключения:

  - `InvalidPrometheusRequest`: В случае неуспешного статуса ответа от Prometheus

- `std::vector<Metric> query(const std::string &query_str, std::time_t start, std::time_t end, int step)`

  Выполнить запрос к Prometheus с шагом между точками.
  Аргументы:

  - `query`: Строка запроса
  - `start`: Начало временного диапазона
  - `end`: Конец временного диапазона
  - `step`: Интервал между точками в секундах, по умолчанию 15

  Возвращает: Массив метрик типа `Metric`

  Исключения

  - `InvalidPrometheusRequest`: В случае неуспешного статуса ответа от Prometheus

Объявления и описания членов классов находятся в файлах:

- src/lib/tsdb/prometheus/prometheus.h
- src/lib/tsdb/prometheus/prometheus.cpp

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ImGui: Официальная документация: URL: https://github.com/ocornut/imgui
2. ImPlot: Официальная документация: URL: https://github.com/epezent/implot
3. Prometheus: Официальная документация: URL: https://prometheus.io/docs/introduction/overview/
4. Yandex Cloud: Официальная документация: URL: https://yandex.cloud/ru/docs/
5. CMake: Официальная документация: URL: https://cmake.org/documentation/
6. Doctest: Официальная документация: URL: https://github.com/doctest/doctest/

ПРИЛОЖЕНИЯ

1. GitHub-репозиторий проекта: https://github.com/sssemion/low-budget-grafana
2. Документация (GitHub Pages): https://sssemion.github.io/low-budget-grafana/