|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***«Мониторинг состояния серверов»***

Студент ИУ7-75Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Т.М.Оберган

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Ю.В.Строганов

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

**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)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_ИУ7\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_И.В.\_Рудаков\_\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине \_\_\_\_\_\_Компьютерные Сети\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы ИУ7-75Б

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Оберган Татьяна Максимовна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Тема курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Мониторинг состояния серверов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направленность КП (учебный, исследовательский, практический, производственный, др.)

\_\_\_\_\_\_\_\_учебный\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения проекта: 25% к \_4\_ нед., 50% к \_7\_ нед., 75% к \_11 нед., 100% к \_14 нед.

***Задание\_\_****Разработать программу, выполняющую мониторинг выбранных серверов.*

***Оформление курсового проекта:***

Расчетно-пояснительная записка на \_25-30\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

На защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов.\_\_\_

На слайдах должны быть отражены: постановка задачи, использованные методы и\_\_\_\_\_\_\_\_\_

алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс,\_\_\_\_\_\_\_\_\_\_\_

результаты проведенных исследований.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «17 » сентября 2020г.

**Руководитель курсового проекта**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.В.Строганов

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Т.М.Оберган

(Подпись, дата) (И.О.Фамилия)

Оглавление

[Введение 4](#_Toc59987718)

[1. Аналитическая часть 5](#_Toc59987719)

[1.1 Формализация задачи 5](#_Toc59987720)

[1.2 Анализ аналогов 5](#_Toc59987721)

[1.3 Виды систем мониторинга 7](#_Toc59987722)

[1.4 HTTP 7](#_Toc59987723)

[1.5 SNMP 8](#_Toc59987724)

[1.6 WMI 9](#_Toc59987725)

[1.7 MVC 10](#_Toc59987726)

[1.8 Выводы из аналитического раздела 11](#_Toc59987727)

[2. Конструкторская часть 12](#_Toc59987728)

[2.1 Требования к программе 12](#_Toc59987729)

[2.2 Структура решения 12](#_Toc59987730)

[2.3 Общий алгоритм работы системы 14](#_Toc59987731)

[2.4 Менеджер 15](#_Toc59987732)

[2.5 Наблюдаемый сервер 16](#_Toc59987733)

[2.5 Веб приложение 16](#_Toc59987734)

[2.6 Выводы из конструкторского раздела 16](#_Toc59987735)

[3. Технологическая часть 17](#_Toc59987736)

[3.1 Выбор и обоснование языка программирования и среды разработки 17](#_Toc59987737)

[3.3 Сведения о веб приложении 18](#_Toc59987738)

[3.4 Сведения о менеджере серверов 19](#_Toc59987739)

[3.5 Сведения о наблюдаемом сервере 20](#_Toc59987740)

[3.4 Интерфейс программы 21](#_Toc59987741)

[3.5 Выводы из технологического раздела 22](#_Toc59987742)

[Заключение 23](#_Toc59987743)

[Список использованной литературы 24](#_Toc59987744)

# Введение

Термином мониторинг серверов называют непрерывное наблюдение за основными показателями работы сервера [1]. Целью такого наблюдения является контроль непрерывной работы сервера, предотвращение аварий и инцидентов. Преимуществами мониторинга являются: снижение времени простоя, повышение производительности работы, помощь в планировании пропускной способности сети, снижение производственных затрат в сфере IT [2].

Целью данной курсовой работы является создание программы, выполняющей мониторинг серверов.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

1. формализация цели;
2. анализ аналогов;
3. анализ и выбор способов достижения поставленной цели;
4. разработка программного обеспечения, которое позволит решить задачу мониторинга серверов.

# 1. Аналитическая часть

В данном разделе будет формализована задача, проанализированы аналоги и рассмотрены способы решения поставленной задачи.

## 1.1 Формализация задачи

Необходимо реализовать приложение для мониторинга состояния и управлением группой серверов, которые направлены на выполнение одинаковых задач.

Каждый сервер имеет следующие характеристики:

* имя;
* адрес;
* показатель использования;
* процент выполнения текущей задачи;
* показатель загруженности процессора;
* показатель загруженности памяти.

## 1.2 Анализ аналогов

**UptimeRobot** – веб приложение для мониторинга состояния серверов [3].

Достоинства:

* оповещения при падении сервера (e-mail, sms, звонок, twitter, slack, и другие);
* логирование событий;
* мобильное приложение.

Недостатки:

* большая часть функционала доступна по платной подписке;
* не предоставляет информации о серверных ресурсах и процент выполнения текущей задачи.

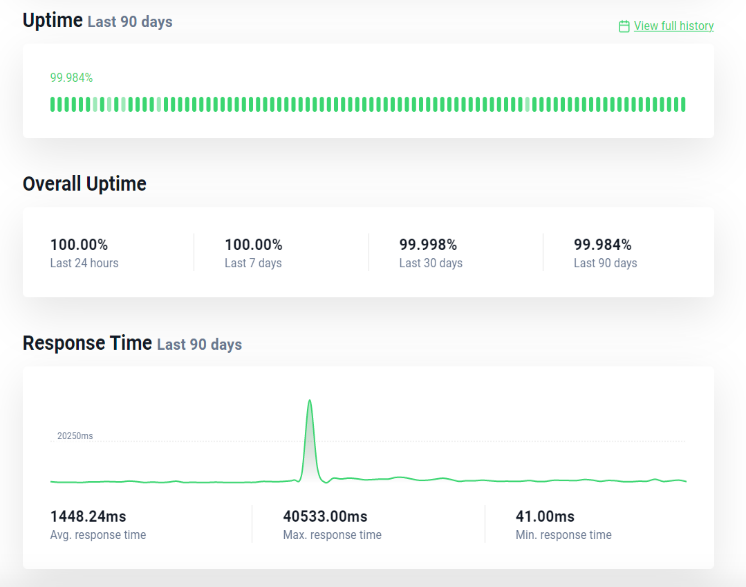




Рис. 1.2.1 – интерфейс приложения UptimeRobot

**Cacti** – open-source веб-приложение для мониторинга компьютерной сети [4]. Собирает статистику за определенные временные интервалы и отображает их в графическом виде.

Достоинства:

* открытый исходный код;
* поддержка циклических баз данных с более чем одним источником данных [5];
* присутствует статистика по загрузке процессора, выделению оперативной памяти, количеству запущенных процессов.

Недостатки:

* высокий порог вхождения к пользованию программой;
* требуется MySQL, Apache или IIS с поддержкой PHP.

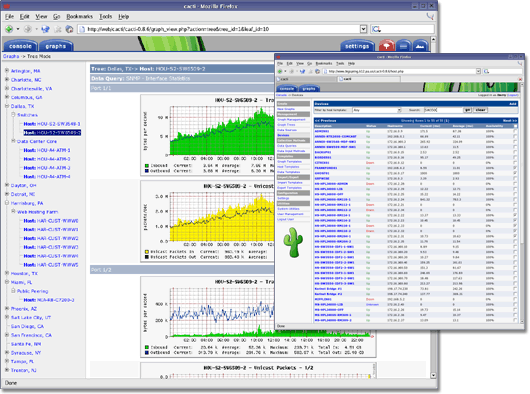


Рис. 1.2.2 – интерфейс программы cacti

## 1.3 Виды систем мониторинга

По типу установки системы мониторинга делятся на:

1. системы, которые устанавливаются непосредственно на сервер;
2. клиент-серверные системы мониторинга. [1]

Первый тип хорош простотой администрирования и настройки, но такая система перестанет работать при выходе из строя сервера при этом во время работы будет потреблять ресурсы самого сервера, снижая его производительность.

Со вторым типом все ровно наоборот: сложная настройка, но надежный мониторинг и сохранность ресурсов сервера. В этом случае на наблюдаемый сервер устанавливается программный агент, который собирает информацию о работе сервера и пересылает ее на сервер мониторинга, который в свою очередь анализирует и сохраняет полученные данные.

## 1.4 HTTP

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи произвольных данных. Задача, которая традиционно решается с помощью этого протокола – обмен данными между пользовательским приложением, осуществляющим доступ к веб ресурсам и веб-сервером. API многих программных продуктов также подразумевает использование HTTP для передачи данных [6].

Все ПО для работы с HTTP разделяется на:

* клиенты – оправляют запросы;
* серверы – обрабатывают запросы;
* прокси – посредники для выполнения транспортных служб.

Каждое HTTP сообщение состоит из трех частей:

* стартовая строка, определяющая тип сообщения;
* заголовки, характеризующие тело сообщения и прочие сведения;
* тело сообщения – данные.

## 1.5 SNMP

SNMP (Simple Network Management Protocol) – интернет-протокол для управления устройствами в IP-сетях на основе архитектур TCP/UDP. Данный протокол используется в системах сетевого управления для контроля подключенных к сети устройств [7].

К поддерживающим SNMP устройствам относятся:

* маршрутизаторы;
* коммутаторы;
* серверы;
* рабочие станции;
* принтеры;
* другие.

При использовании SNMP один или более административных компьютеров выполняют отслеживание или управление группой хостов или устройств в компьютерной сети. На каждой управляемой системе есть постоянно запущенная программа, называемая агент, которая через SNMP передаёт информацию менеджеру. [8]

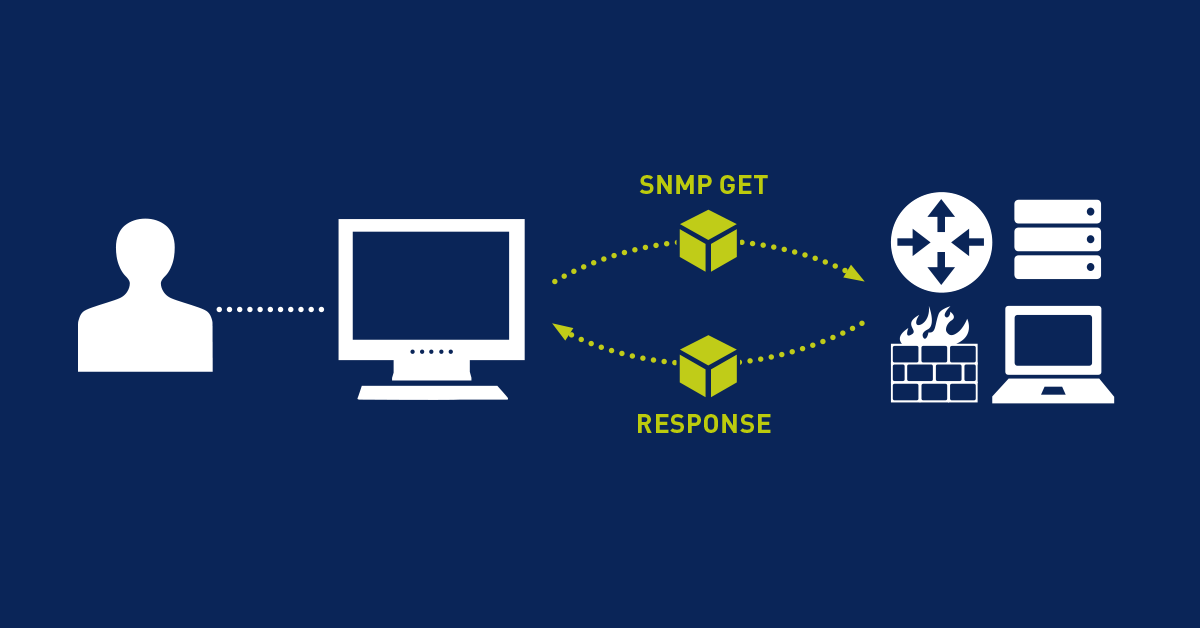


Рис. 1.4.1 – схема работы SNMP [9]

## 1.6 WMI

WMI (Windows Management Instrumentation) – это одна из базовых технологий для централизованного управления и слежения за работой различных компонентов как локальной, так и удаленной компьютерной инфраструктуры под управлением платформы Windows. WMI позволяет писать скрипты или приложения, призванные автоматизировать задачи администрирования. [10]

WMI построен по объектно-ориентированному принципу и рассчитан на программистов, использующих C#, C/C++, Microsoft Visual Basic, скриптовые ЯП, обрабатывающие Microsoft ActiveX объекты.

Wmic.exe – консольная утилита для взаимодействия со структурой WMI на локальном или удаленном компьютере.

Листинг 1.5: пример команд, возвращающих загруженность процессора и количество свободной памяти на удаленной машине [11]

wmic /node:HOSTNAME cpu get loadpercentage

wmic /node:HOSTNAME OS get FreePhysicalMemory

## 1.7 MVC

MVC (Model-View-Controller) – паттерн проектирования ПО. Необходим для разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента, таким образом, что модификация каждого компонента может осуществляться независимо. [12]

Ниже описаны виды компонентов и их различия.

* **Модель** предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* **Представление** отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* **Контроллер** интерпретирует действия пользователя, оповещая модель о необходимости изменений.



Рис. 1.7.1 – схема паттерна MVC [13]

## 1.8 Выводы из аналитического раздела

В данном разделе была формализована задача, проанализированы аналоги и рассмотрены способы решения поставленной задачи. В частности, рассмотрены протоколы передачи данных HTTP, SNMP. В качестве основной, была выбрана клиент серверная архитектура с использованием MVC паттерна.

# 2. Конструкторская часть

В данном разделе будут рассмотрены требования к программе и ее компонентам.

## 2.1 Требования к программе

Программа-менеджер должна предоставлять следующие возможности:

* опрос всех серверов для обновления их статуса в системе;
* добавление нового сервера в пул серверов;
* удаление сервера из пула серверов;
* отправление запроса выбранному серверу;
* отправление запроса на автоматически выбранный сервер;
* мониторинг состояния запроса.

Для удобства пользования программой-менеджером предлагается написать веб приложение, имеющее следующий функционал:

* просмотр состояния отслеживаемых серверов;
* добавление отслеживаемого сервера;
* удаление сервера из отслеживаемых;

Для тестирования программы-менеджера предлагается создать сервер, выполняющий итеративные вычисления, например подсчет факториала. Если в любой момент времени известен номер и количество действий внутри каждой итерации, то также и известен процент выполнения текущей задачи. Несколько программ-серверов может быть запущено на одной аппаратной платформе.

## 2.2 Структура решения

Предполагается MVC взаимодействие системы. Где View – визуальная составляющая веб приложения; Controller – управляющая прослойка веб приложения, которая осуществляет получение информации из менеджера для веб страницы и управление его состоянием; Model – менеджер серверов, бизнес-логика приложения.

На рис. 2.2.1 и 2.2.2 показаны выделенные компоненты и их взаимодействие.

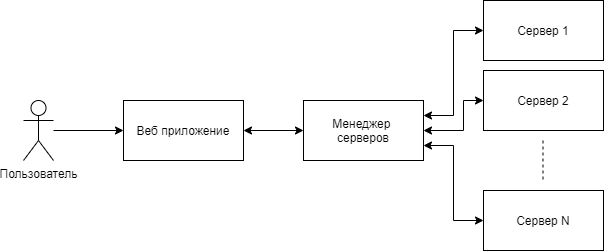


Рис. 2.2.1 – структура решения

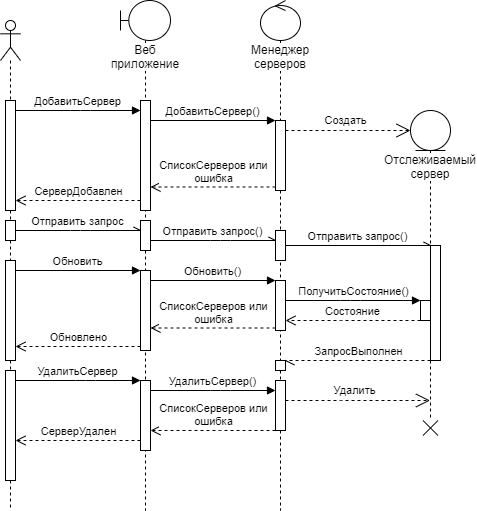


Рис. 2.2.2 – диаграмма последовательностей

## 2.3 Общий алгоритм работы системы

1. Запуск наблюдаемых программ-серверов.
2. Запуск программы-менеджера и веб приложения.
3. Если нужно отправить запрос вручную:
   1. при помощи программы postman [14] (или любой другой) формируется и отправляется запрос;
   2. во время обработки запроса на сервере, статус можно просмотреть через веб приложение или при соответствующем запросе к менеджеру, при условии добавления сервера в наблюдаемые.
   3. результат выполнения запроса приходит программе, инициировавшей запрос.
4. Если нужно отправить запрос через менеджера:
   1. менеджеру передаются данные запроса;
   2. менеджер выбирает первый свободный сервер, если такового нет, помещает запрос в очередь;
   3. менеджер возвращает уникальный внутри сессии идентификатор запроса;
   4. по идентификатору запроса через менеджер можно получить состояние запроса или результат.
5. Для добавления или удаления наблюдаемых серверов нужно послать соответствующую команду менеджеру.



Рис. 2.3.1 – use-case диаграмма

## 2.4 Менеджер

Менеджер ответственен за получение информации о серверах. Это порождает связь клиент-сервер. Менеджер в данном случае выступает в роли клиента, при получении информации от наблюдаемого сервера.

Предполагается, что сервер знает свое текущее состояние и готов его вернуть посредством get-запроса. Далее будет предполагаться, что у каждого наблюдаемого сервера реализовано API /status. Ответ от сервера должен быть в формате json и содержать основную информацию о сервере:

* Name: string – наименование сервера;
* CurrentTaskNumber: int – текущий номер подзадачи;
* TotalTasksAmount: int – общее количество подзадач внутри текущей задачи;

На основании полученных данных, менеджер знает состояние сервера (занят/свободен), может рассчитать процент выполнения текущей задачи, при хранении прошлого состоянии сервера, спрогнозировать время завершения задачи.

Менеджер ответственен за хранение поля «запрос» и получение и сохранение ответа от сервера, при условии, что запрос был отправлен с его помощью.

## 2.5 Наблюдаемый сервер

Из прошлого раздела можно почерпнуть главные требования к наблюдаемому серверу:

* наличие обработчика get-запроса /status;
* сервер должен быть в состоянии обрабатывать несколько запросов одновременно (один – основной, другой – получение статуса);
* сервер должен быть в состоянии в любой момент времени предоставить номер текущей итерации. Для выполнения данного требования создается глобальная (разделяемая потоком основного запроса и потоком запроса статуса) переменная, которая обновляется в главном потоке, а поток статуса ее считывает.

## 2.5 Веб приложение

Веб приложение призвано улучшить пользовательский опыт взаимодействия с менеджером: данные о серверах и их состояние должны быть представлены в наглядном виде, реализовано добавление наблюдаемых серверов.

## 2.6 Выводы из конструкторского раздела

В данном разделе были проанализированы требования к программе и ее компонентам. Между менеджером и наблюдаемыми серверами было установлена клиент-серверная связь с жестко определенным запросом статуса сервера. Было решено использовать веб приложение для наглядного отображения информации.

# 3. Технологическая часть

## 3.1 Выбор и обоснование языка программирования и среды разработки

В качестве языка программирования был выбран C# т.к.:

* я знакома с этим языком программирования, что сократит время написания программы;
* данный язык программирования кроссплатформенный и объектно-ориентированный, что даст в полной мере использовать наследование, абстрактные классы и т.д.; [15]

В качестве среды разработки была выбрана «Visual Studio 2017» по следующим причинам:

* она бесплатна в пользовании студентами;
* она имеет множество удобств, которые облегчают процесс написания и отладки кода;
* она обеспечивает работу с Windows Forms – интерфейсом, который упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обертки для существующего Win32 API в управляемом коде;
* я знакома с данной средой разработки, что сократит время изучения возможностей среды.

Для создания визуальной составляющей был использован фреймворк Angular по следующим причинам:

* все необходимое для начала разработки уже есть внутри Angular;
* компонентный подход;
* dependency injection дает возможность изменять поведение фреймворка, позволяет реализовывать такие паттерны, как Singleton, Factory, Façade.

Для реализации серверной части был использован swi-prolog т.к. этот язык позволит, в рамках курсовой работы, ознакомиться со способами написания серверов на логических языках.

## 3.3 Сведения о веб приложении

Для отображения визуальной части использовался шаблон ASP.NET Core с Angular.

Были выделены следующие компоненты:

* app.component – оболочка для всего приложения;
* home.component – главная страница приложения;
* nav-menu.component – компонент навигации;

Для каждого компонента созданы основной файл typescript и html файл.

Листинг 3.3.1: пример ts файла компонента со статической информацией

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'app-home',
4. templateUrl: './home.component.html',
5. })
6. export class HomeComponent {
7. }

В более сложных компонентах присутствует получение данных с контроллера через get запрос.

Листинг 3.3.2: компонент, реализующий доступ к контроллеру

1. import { Component, Inject } from '@angular/core';
2. import { HttpClient, HttpHeaders } from '@angular/common/http';
3. import { Url } from 'url';
4. @Component({
5. selector: 'servers-info-data',
6. templateUrl: './servers-info.component.html'
7. })
8. export class ServersInfoComponent {
9. private http: HttpClient;
10. private baseUrl: string;
11. public servers: ServerInfo[];
12. constructor(http: HttpClient, @Inject('BASE\_URL') baseUrl: string) {
13. this.http = http;
14. this.baseUrl = baseUrl;
15. this.updServers();
16. }
17. addServer(address: string) {
18. var headers = new HttpHeaders({ 'Content-Type': 'application/json; charset=utf-8' });
19. this.http.post(this.baseUrl + 'api/Manager/AddServer', { uri: address }, { headers: headers }).subscribe(result => { }, error => console.error(error));
20. }
21. updServers() {
22. this.http.get<ServerInfo[]>(this.baseUrl + 'api/Manager/GetServers').subscribe(result => {
23. this.servers = result;
24. }, error => console.error(error));
25. }
26. }

## 3.4 Сведения о менеджере серверов

Менеджер серверов написан на .NET Core и реализует бизнес-логику приложения.

Два выделенных класса: Manager, Server.

Класс менеджера хранит все сервера внутри себя, запрашивает обновление информации серверов, отслеживает выполнение запросов. Через менеджер можно отправить запрос на один из активных и свободных серверов, причем выбор будет сделан автоматически (предполагается, что добавленные сервера являются эквивалентными по доступному api). Менеджер вернет уникальный внутри сессии идентификатор запроса, по которому можно отследить его состояние. В случае, если нет доступных серверов, запрос помещается в очередь ожидания.

Класс сервера внутри себя реализует асинхронную задачу отправки get-запроса и получения его результата, функцию обновления состояния сервера внутри менеджера на основании реального состояния сервера (запрос /state).

Листинг 3.4.1: асинхронная задача get-запроса

1. public async Task<string> GetMethod(string path)
2. {
3. HttpClient client = new HttpClient();
4. client.Timeout = TimeSpan.FromMinutes(1);
5. client.BaseAddress = Uri;
6. var response = await client.GetAsync(path);
7. var responseString = await response.Content.ReadAsStringAsync();
8. return responseString;
9. }

## 3.5 Сведения о наблюдаемом сервере

В рамках данной курсовой работы был реализован сервер на swi-prolog, в основе которого лежит библиотека «thread\_httpd». Эта библиотека позволяет создать многопоточный сервер, который в состоянии обрабатывать несколько запросов одновременно.

Листинг 3.4.1: инициализация сервера

1. :- initialization server.
2. **server**() :-
3. http\_server(http\_dispatch, [port(8080)]).

Для демонстрации работы программы мониторинга было решено создать функцию расчета факториала и обработчик соответствующего get запроса. Подсчет факториала хорошо подходит для данных целей т.к. имеет итеративную составляющую, количество итераций известно и фиксировано.

Листинг 3.4.2: обработка get запроса

1. % http://localhost:8080/factorial/15000
2. :- http\_handler(root(factorial/N), getFact(M, N),[method(M),methods([**get**])]).
3. **getFact**(get, AtomN, R) :-
4. **atom\_number**(AtomN, N),
5. factorial(N, Ans),
6. reply\_json(json{answer:Ans}).

Листинг 3.4.3: Функция подсчета факториала

1. **factorial**(N, -1) :- N < 0, !. *% error*
2. **factorial**(0, 1) :- !.
3. **factorial**(N, Res) :-
4. zero\_out\_state(N),
5. factorial(N, 1, Res).
6. **factorial**(1, Res, Res) :-
7. update\_state, !.
8. **factorial**(N, Cur, Res) :-
9. update\_state,
10. NewN is N - 1,
11. NewMult is Cur \* N,
12. factorial(NewN, NewMult, Res).

Обязательное условие работы программы мониторинга: наличие в наблюдаемом сервере обработчика запроса на получение состояния.

Листинг 3.4.4: обработка get запроса /state

1. :- http\_handler(root(state), getState, []).
2. **getState**(\_) :-
3. curTasks(Cur),
4. totalTasks(Total),
5. reply\_json(json{**name**: swipl, current: Cur, total:Total}).

Для запуска тестируемого сервера требуется установить swi-prolog, перейти в директорию с реализацией и напечатать команду “swipl <имя файла-реализации> <номер порта>”.

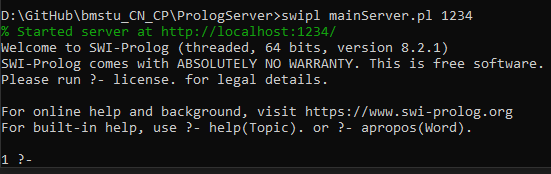


Рис. 3.5.1 – пример запуска сервера на прологе

## 3.4 Интерфейс программы

На рисунках 3.4.1, 3.4.2, 3.4.3 показан интерфейс веб приложения.

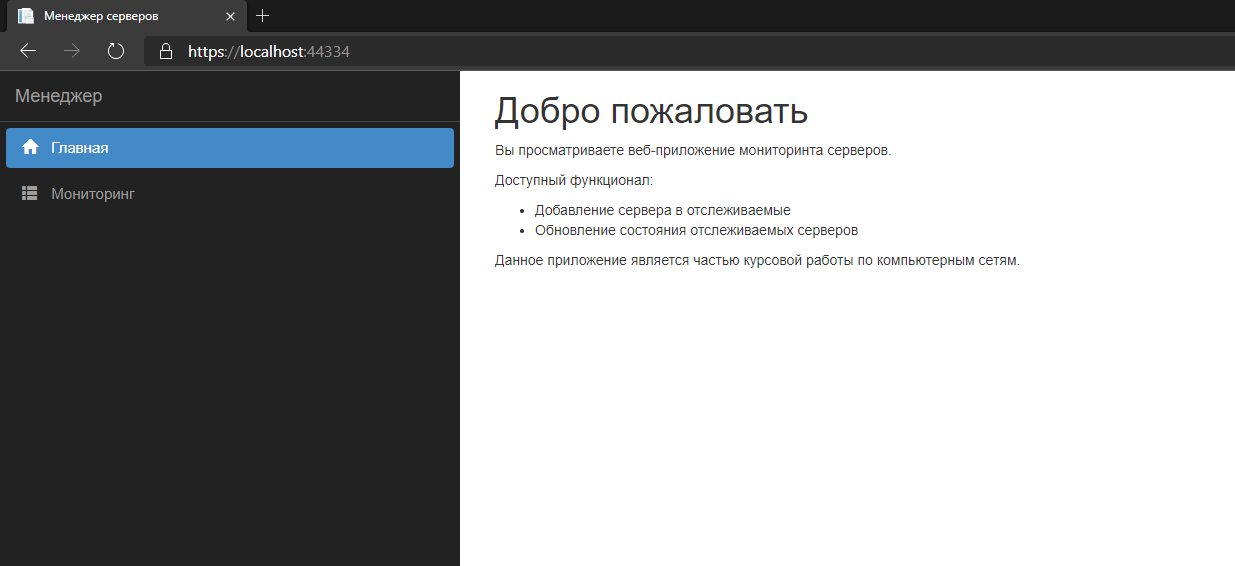


Рис. 3.4.1 – главная страница веб приложения

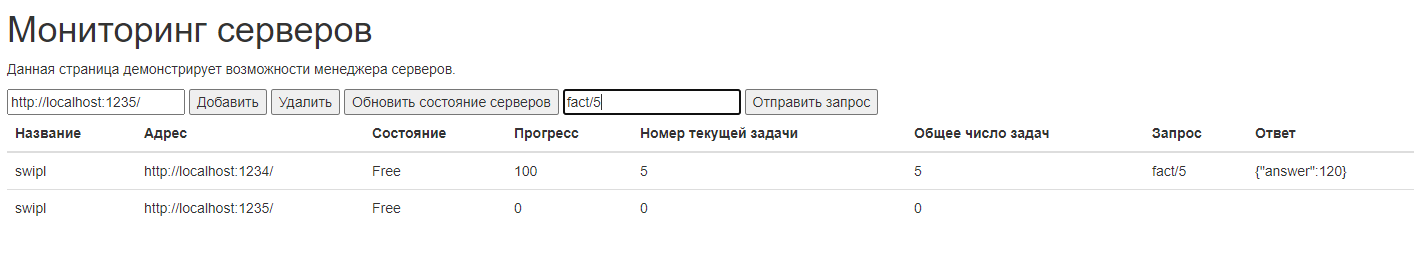


Рис. 3.4.2 – запрос был выполнен

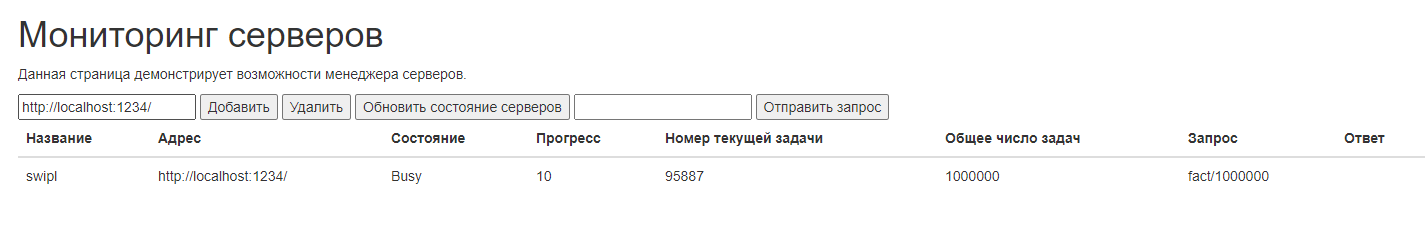


Рис. 3.4.3 – запрос в стадии выполнения, сервер занят

## 3.5 Выводы из технологического раздела

В данном разделе был выбран стек используемых технологий: .Net Core, Angular, Swi-Prolog. Для отображения визуальной части использовался шаблон ASP.NET Core с Angular. Менеджер серверов написан на .NET Core и реализует бизнес-логику приложения. Сервер для тестирования был реализован на Swi-Prolog.

# Заключение

Во время выполнения курсового проекта были достигнуты поставленные цель и задачи: формализована цель; проанализированы аналоги; выбран способ достижения поставленной цели; разработана программа, которая позволяет решить задачу мониторинга серверов.

Был реализован менеджер серверов на языке C# и веб приложение Angular для создания пользовательского интерфейса, через который можно наблюдать за отслеживаемыми серверами.

В ходе выполнения поставленных задач были изучены возможности языка C#, фреймворка Angular, получены знания в области компьютерных сетей.

# Список использованной литературы

1. Мониторинг серверов и системное администрирование. [Электронный ресурс]. – Режим доступа: https://www.reg.ru/support/vydelennyye-servery-i-dts/administrirovanie-vydelennyh-serverov/monitoring-vydelennogo-servera-dedicated

2. Мониторинг серверов: что это и для необходимо. [Электронный ресурс]. – Режим доступа: https://www.ittelo.ru/news/monitoring-serverov-chto-eto-i-dlya-neobkhodimo/

3. Uptime Robot, официальный веб-сайт. [Электронный ресурс]. – Режим доступа: https://www.ittelo.ru/news/monitoring-serverov-chto-eto-i-dlya-neobkhodimo/

4. Cacti, официальный веб-сайт. [Электронный ресурс]. – Режим доступа: https://www.cacti.net/

5. 5 лучших бесплатных систем мониторинга ИТ-инфраструктуры. [Электронный ресурс]. – Режим доступа: https://networkguru.ru/5-besplatnykh-sistem-monitoringa-it-infrastruktury/

6. Простым языком об HTTP. [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/215117/

7. Douglas R. Mauro & Kevin J. Schmidt. (2001). *Essential SNMP* (1st ed.). Sebastopol, CA: O’Reilly & Associates

8. J. Case; K. McCloghrie; M. Rose; S. Waldbusser (April 1993). Internet Engineering Task Force.

9. What is SNMP. [Электронный ресурс]. – Режим доступа: https://www.paessler.com/it-explained/snmp

10. Windows Management Instrumentation. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-gb/windows/win32/wmisdk/wmi-start-page

11. wmic command. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmic

12. Сергей Рогачев*.* Обобщенный Model-View-Controller. [Электронный ресурс]. – Режим доступа: http://rsdn.org/article/patterns/generic-mvc.xml 2007.

13. Joseph Spinelli. MVC Overview. [Электронный ресурс]. – Режим доступа: https://medium.com/@joespinelli\_6190/mvc-model-view-controller-ef878e2fd6f5

14. Официальный веб-сайт Postman. [Электронный ресурс]. – Режим доступа: https://www.postman.com/explore

15. C# документация. Microsoft Docs. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/dotnet/csharp/