МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ П.М. МАШЕРОВА»

Факультет математики и информационных технологий

Кафедра прикладного и системного программирования

Допущен к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г.

Заведующая кафедрой

\_\_\_\_\_ Е. А. Корчевская

ДИПЛОМНАЯ РАБОТА

ИСПОЛЬЗОВАНИЕ ПЛАТФОРМЫ NODEJS ДЛЯ РАЗРАБОТКИ СЕРВЕРНОЙ ЧАСТИ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Специальность 1-40 01 01 03 Программное обеспечение информационных технологий. Базы данных и программное обеспечение информационных систем

Царенко Антон Дмитриевич

4 курс, 44 группа

Научный руководитель:

Новый Вадим Владимирович,

старший преподаватель

Витебск, 2024

# РЕФЕРАТ

Дипломная работа53 с., 25 рис., 31 листинг, 15 источников, 1 приложение.

C#, .NET, WINDOWS, СЛУЖБА WINDOWS, ЖУРНАЛ СОБЫТИЙ WINDOWS, NODEJS, NESTJS, TYPESCRIPT, TYPEORM, БАЗА ДАННЫХ, POSTGESQL, WEB APPLICATION, AUTHORIZATION, JWT, PASSPORT, BACKEND.

Объект исследования – учет времени работы компьютерной техники.

Предмет исследования – технологии разработки программного обеспечения для учета времени работы компьютерной техники.

Цель работы – разработать серверную часть приложения для учета времени работы компьютерной техники с использованием платформы NodeJS.

Методы исследования: анализ предметной области, синтез, изучение литературы и технической документации по используемым технологиям, методы объектно-ориентированного проектирования и программирования.

Элементы новизны: в результате изучения принципов функционирования операционных систем, а именно процесса авторизации пользователей, были разработаны механизмы сбора информации о входе и выходе пользователей из ОС.

Теоретическая и практическая значимость: разработанное приложение отслеживает вход и выход пользователей из операционной системы, а также предоставляет статистику о времени работы пользователей, что позволяет оптимизировать расписание занятий в аудиториях с компьютерной техникой.

# СОДЕРЖАНИЕ

[РЕФЕРАТ 2](#_Toc170195933)

[СОДЕРЖАНИЕ 3](#_Toc170195934)

[ВВЕДЕНИЕ 5](#_Toc170195935)

[1 Характеристика приложения и средства разработки 7](#_Toc170195936)

[1.1 Краткая характеристика приложения 7](#_Toc170195937)

[1.2 Описание используемых технологий 8](#_Toc170195938)

[1.2.1 C# 8](#_Toc170195939)

[1.2.2 NodeJS 8](#_Toc170195940)

[1.2.3 NestJS 9](#_Toc170195941)

[1.2.4 TypeORM 10](#_Toc170195942)

[1.2.5 PostgreSQL 11](#_Toc170195943)

[1.3 Выводы из первого раздела 12](#_Toc170195944)

[2 Практическая реализация 13](#_Toc170195945)

[2.1 Реализация службы Windows 13](#_Toc170195946)

[2.2 Общая структура серверной части приложения 17](#_Toc170195947)

[2.3 Реализация записи времени входа и выхода 18](#_Toc170195948)

[2.4 Аутентификация 38](#_Toc170195949)

[2.5 Тестирование службы 42](#_Toc170195950)

[2.5.1 Установка службы 42](#_Toc170195951)

[2.5.2 Нагрузочное тестирование 47](#_Toc170195952)

[2.6 Выводы из второго раздела 49](#_Toc170195953)

[Заключение 50](#_Toc170195954)

[Список использованных источников 51](#_Toc170195955)

[ПРИЛОЖЕНИЕ А 53](#_Toc170195956)

# ВВЕДЕНИЕ

Актуальность данной работы заключается в необходимости учета времени работы компьютерной техники в аудиториях. Разрабатываемое приложение может быть полезным для различных учреждений образования, так как предоставляет информацию, необходимую для оптимизации расписания занятий в аудиториях с компьютерной техникой. Это позволит использовать компьютерную технику более эффективно, уменьшая время простоя и улучшая общую производительность в связи с равномерным распределением нагрузки на компьютеры.

Проект представляет из себя 2 модуля. Первый модуль представляет собой службу, которая работает на компьютере. Она отслеживает вход и выход пользователей из операционной системы и передает эти данные на сервер. Второй модуль – клиент-серверное приложение, предоставляющее пользователям интерфейс для просмотра статистики времени работы компьютеров.

Объектом исследования является учет времени работы компьютерной техники. Предметом исследования являются технологии разработки программного обеспечения для учета времени работы компьютерной техники. Таким образом, целью дипломной работы является разработка серверной части приложения для учета времени работы компьютерной техники с использованием платформы NodeJS.

Задачи дипломной работы:

1. Изучить особенности разработки серверной части приложения с использованием платформы NodeJS;
2. Разработать серверную часть приложения для учета времени работы компьютерной техники.
3. Изучить особенности процесса авторизации пользователей в операционной системе Windows и разработки служб Windows;
4. Разработать службу для операционной системы Windows с использованием языка программирования C#;

Источниками изучения информации о процессе авторизации пользователей в системе являются: статья об источниках наиболее значимых событий при мониторинге информационной безопасности [1], статья об аудите событий входа пользователей [10], статья с описанием события входа (код 4624) в журнале безопасности [11].

Статья «Знакомство с приложениями служб Windows» [2] и практическое руководство по созданию служб Windows [9] используется в качестве источника информации для создания службы.

Документация NodeJS [3], статья о преимуществах и недостатках использования NodeJS [2], документация NestJS [5], статья об использовании паттерна MVC при разработке web-приложений [6], статья об особенностях использования TypeORM [7], статья об особенностях СУБД PostgreSQL [8], статьи об авторизации и аутентификации пользователя при помощи JWT [12][13] используются для изучения особенностей реализации серверной части приложения на платформе NodeJS и для самой реализации приложения. В качестве источников информации для тестирования приложения используются статья о методах тестирования программного обеспечения [14] и руководство по использованию JMeter [9].

# 1 Характеристика приложения и средства разработки

## 1.1 Краткая характеристика приложения

Приложение должно поддерживать следующий функционал:

* Отслеживать время входа и выхода пользователей из операционной системы;
* Предоставлять статистику о времени работы пользователей на компьютерах.

Для отслеживания времени входа и выхода пользователей из операционной системы необходимо выбрать источник информации. В операционной системе Windows разработан специальный журнал событий, где пользователь может просматривать все события и анализировать их [1]. Данный журнал разбит на несколько разделов, нужная нам информация (успешные и неуспешные попытки входа в систему, использование каких-либо ресурсов (создание, открытие и удаление файлов) [1]) хранится в журнале безопасности.

В журнале безопасности нужно отслеживать события входа и выхода пользователей из системы, и для получения данной информации в фоновом режиме работы подходит приложение службы.

Службы Microsoft Windows, ранее известные как службы NT, позволяют создавать долговременные исполняемые приложения, которые запускаются в собственных сеансах Windows. Для этих служб не предусмотрен пользовательский интерфейс. Они могут запускаться автоматически при загрузке компьютера, их также можно приостанавливать и перезапускать. Благодаря этому службы идеально подходят для реализации долго выполняемых процессов, которые не мешают работе пользователей на том же компьютере [2].

Служба будет запущена на каждом компьютере, и будет отсылать данные о входе и выходе пользователей из системы на сервер. На сервере эти данные будут сохраняться в базе данных. Затем по этим данным можно будет получить статистику о времени работы пользователей.

Статистику можно будет получить в двух вариантах:

* Статистика по периодам – для каждого компьютера будут указываться дата и время входа и выхода пользователя из системы, часы работы и операционная система, в которой работал пользователь;
* Статистика по часам – для каждого компьютера будут указываться только часы работы и операционная система. Время получения данной статистики будет быстрее (т.к. промежуточные данные о времени работы будут хранится в БД).

В приложении будут реализованы следующие роли – администратор (может просматривать логи входа/выхода пользователей из ОС, редактировать список компьютеров, добавлять новых пользователей в приложение) и обычный пользователь (может просматривать статистику о времени и периодах работы компьютеров).

## 1.2 Описание используемых технологий

### **1.2.1 C#**

**Для разработки службы Windows будет использоваться язык программирования C#.**

C# – это гибкий, современный объектно-ориентированный язык программирования, разработанный компанией Microsoft в рамках платформы .NET. Он был представлен в начале 2000-х годов и получил значительную популярность среди разработчиков для создания различных типов приложений, включая настольные, веб, мобильные и игровые приложения.

C# является одним из наиболее популярных языков программирования в экосистеме разработки Microsoft.

Язык C# содержит шаблоны и классы для создания служб Windows, а также для работы с журналом событий Windows, что является полезным для разрабатываемого приложения.

### **1.2.2 NodeJS**

**Node.js** – это программная платформа, которая предоставляет среду для выполнения кода JavaScript на стороне сервера [3]. Node.js построен на основе движка JavaScript Chrome V8, который позволяет транслировать вызовы на языке JavaScript в машинный код. Node.js прежде всего предназначен для создания серверных приложений на языке JavaScript.

Особенностью Node.js является то, что Node.js использует один (основной) поток, который получает все запросы и управляет ими через очередь запросов (таким образом, Node.js является однопоточным сервером). Внутри этого потока выполняется так называемый цикл событий (event loop). Он представляет собой цикл, который непрерывно проверяет запросы из очереди событий и обрабатывает события ввода и вывода.

К преимуществам NodeJS относятся:

* Быстрое подключение к серверу

Node.js использует цикл событий, который может обрабатывать несколько запросов одновременно. Другие платформы не могут обрабатывать несколько задач до тех пор, пока сервер не выполнит запрос, отправленный от первоначальной пользовательской формы на стороне клиента, в результате для их выполнения потребуется несколько потоков. Причина масштабируемости Node.js заключается в том, что он использует очень мало потоков для обработки запросов нескольких клиентов и больше внимания уделяет вычислительным возможностям системы, работающей с клиентами, а не использованию пространства и времени для обработки потоков [4].

* Единый язык программирования

Преимущество использования Node.js перед другими платформами заключается в том, что разработчик использует только язык JavaScript для программирования как на стороне клиента, так и на стороне сервера [4].

* Большое количество пакетов

Также к преимуществам NodeJS можно отнести большое количество пакетов, которыми можно управлять с помощью пакетного менеджера npm (Node Package Manager). npm предоставляет доступ к множеству готовых модулей и библиотек, которые значительно упрощают разработку приложений.

Однопоточность можно отнести и к недостатку. Слишком большое количество асинхронных запросов в одном потоке может привести к замедлению работы программы. Когда Node.js выполняет задачу, управляемую ЦП (центральным процессором), в своем цикле событий, он использует всю доступную мощность ЦП для выполнения асинхронной операции, что в результате может привести к снижению производительности цикла событий целиком [4].

### **1.2.3 NestJS**

NestJS – это фреймворк для создания эффективных, масштабируемых серверных приложений на Node.js. Он использует прогрессивный JavaScript, хотя Nest создан с помощью TypeScript и полностью его поддерживает (но все же позволяет разработчикам писать код на чистом Javascript) и сочетает в себе элементы ООП (объектно-ориентированного программирования), ФП (функционального программирования) и ФРП (функционально-реактивного программирования) [5].

NestJS использует модульную структуру, которая упрощает организацию и повторное использование кода. Модули позволяют объединять компоненты, провайдеры, контроллеры и другие элементы в логические группы, что улучшает масштабируемость и поддержку приложения. К преимуществам NestJS относится использование декораторов для определения метаданных и связывания элементов приложения.

NestJS использует шаблон MVC для разделения приложения на отдельные части (рисунок 1.1). Архитектура Model-View-Controller позволяет разделить модель данных (Models), пользовательский интерфейс (Views) и управляющую логику (Controllers) на три отдельные составляющих. Причем разделить так, что изменение одного компонента не повлияет на работу остальных [6].

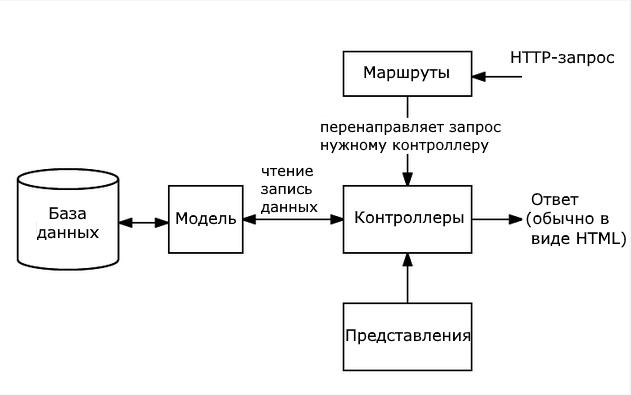


Рисунок 1.1 – схематичное изображение шаблона MVC

### **1.2.4 TypeORM**

TypeORM (ORM – объект-реляционное отображение) — это инструмент, который можно установить поверх Node.js. TypeORM можно использовать в приложениях, использующих JavaScript или TypeScript, и он является устанавливаемым пакетом, который можно использовать для упрощения работы с базами данных [7].

TypeORM легко интегрируется с NestJS, предоставляя удобные декораторы и классы для определения сущностей базы данных, репозиториев и сервисов.

Основные используемые классы:

1. Entity (Сущность) – это класс, который представляет таблицу в базе данных, и его свойства соответствуют полям этой таблицы. Сущности обычно декорируются декоратором @Entity() из пакета TypeORM;
2. Service (Сервис) – класс, который содержит бизнес-логику и выполняет определенные операции или функции в приложении. Сервисы являются основным местом для обработки бизнес-логики и взаимодействия с другими компонентами, такими как репозитории, внешние API или другие сервисы. Используют декоратор @Service();
3. Controller (Контроллер) - отвечает за обработку входящих HTTP-запросов и взаимодействие с клиентами. Контроллеры определяют маршруты, по которым приложение будет принимать запросы, и содержат методы, которые обрабатывают эти запросы.

### **1.2.5 PostgreSQL**

PostgreSQL — это бесплатная СУБД c открытым исходным кодом. С помощью PostgreSQL можно создавать, хранить базы данных и работать с данными с помощью запросов на языке SQL [8].

К основным преимуществам PostgreSQL относятся:

* Мультиплатформенность - PostgreSQL подходит для работы в любой операционной системе: Linux, macOS, Windows [8];
* Свободный доступ – любой специалист может бесплатно скачать, установить СУБД и сразу начать работу с базами данных [8];
* Поддержка больших размеров данных - размер базы данных в PostgreSQL не ограничен и зависит от того, сколько свободной памяти есть в месте хранения: на сервере, локальном компьютере или в облаке. Максимальный размер таблицы — 32 терабайта [8]. Это позволяет хранить большое количество записей (например, логов о входе и выходе пользователей);

Также одним из преимуществ для данного проекта является то, что PostgreSQL легко интегрируется с TypeORM.

Данные в БД будут хранится в соответствии со следующей схемой (рисунок 1.2). Схема содержит следующие таблицы:

* user – таблица, хранящая информацию о пользователях приложения;
* role – хранит роли, которые используются в приложении;
* computer – хранит информацию о компьютерах;
* day\_computre\_work, month\_computer\_work, year\_computer\_work – хранят время работы компьютеров по дням, месяцам, годам;
* period\_computer\_work – хранит периоды работы компьютеров;
* log – хранит первичные записи с данными о входе/выходе пользователей, приходящие с компьютеров;

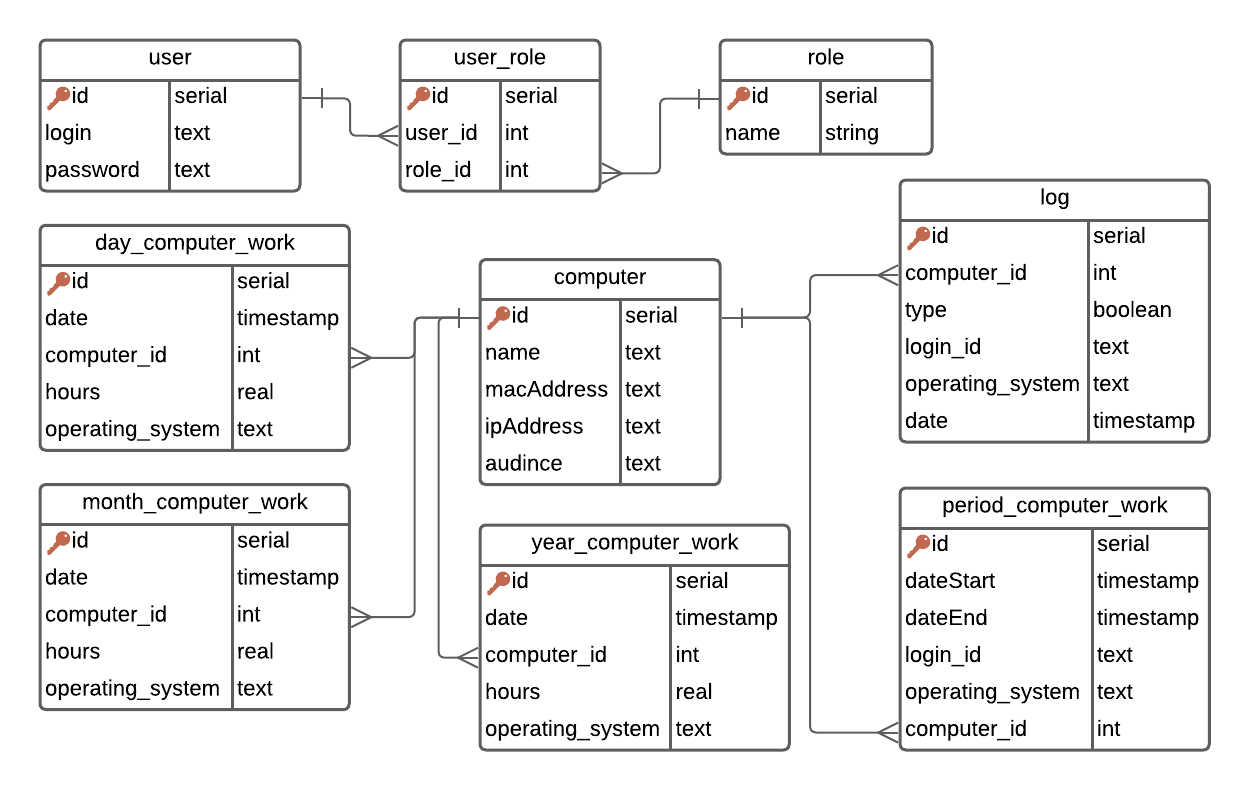


Рисунок 1.2 – схема базы данных

## 1.3 Выводы из первого раздела

В первом подразделе были описаны требования к приложению для учета времени работы компьютеров, а также функционал серверной части разрабатываемого приложения и службы Windows.

Во втором подразделе были описаны средства разработки: NodeJS и его фреймворк NestJS в качестве платформы для создания серверной части приложения, TypeORM в качестве ORM для упрощения работы с базами данных, C# для разработки службы Windows. PostgreSQL был выбран в качестве СУБД.

# 2 Практическая реализация

## 2.1 Реализация службы Windows

При создании службы будем использовать шаблон проекта Visual Studio, который называется Служба Windows. Этот шаблон автоматически выполняет основную часть работы, ссылаясь на необходимые классы и пространства имен, устанавливая наследование от базового класса для служб [9].

При создании службы укажем следующие свойства:

* Имя службы – AuthLog Service;
* CanShutdown – true, чтобы указать, что служба хочет принимать уведомления о выключении компьютера, на котором она работает, позволяя ему вызывать процедуру OnShutdown;
* CanHandlePowerEvent – false, чтобы не сообщать службе об изменениях состояния питания компьютера;
* AutoLog – true, для внесения информационных записей в журнал событий приложения, когда служба выполняет некоторое действие.

Отслеживать вход и выход пользователей можно с помощью журнала событий Windows, а именно журнала безопасности. Идентифицировать событие можно по коду:

* 4624 – пользователь успешно вошел в учетную запись [10];
* 4634 – процесс выхода был завершен для пользователя [10];
* 4647 – пользователь инициировал процесс выхода из системы [10].

В программе будет использоваться код 4624 для отслеживания входа пользователя и код 4647 для выхода. Однако событие с кодом 4624 генерируется не только при входе пользователей в систему, но и при других обстоятельствах, которые можно различать по типу входа. Вот список основных кодов:

* 0 – System (используется только системной учетной записью, например при запуске системы) [11];
* 2 – Interactive (пользователь успешно вошел в систему на данном компьютере) [11];
* 3 – Network (пользователь вошел в систему на данном компьютере через сеть) [11];
* 4 – Batch (пакетный тип входа используется пакетными серверами, исполнение процессов на которых производится по поручению пользователя, но без его прямого вмешательства) [11];
* 5 – Service (служба была запущена диспетчером служб) [11].

Будем отслеживать события с кодом 2. События с таким кодом также генерируют виртуальные учетные записи (учетная запись, используемая какой-либо службой), поэтому такие записи следует игнорировать (рисунок 2.1).

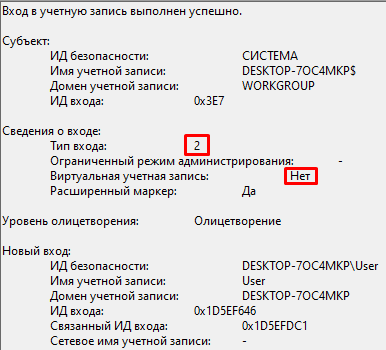


Рисунок 2.1 – событие входа пользователя

Для идентификации события выхода используем код 4647, так как такие события порождают только обычные пользователи (что упрощает работу с такими событиями, так как отсутствует необходимость отслеживания типа входа и виртуальных учетных записей, как в событиях с кодом 4634) и согласно описанию этого кода, данное событие можно рассматривать как событие выхода (рисунок 2.2).

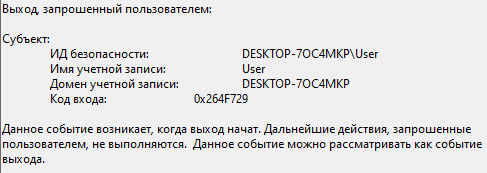


Рисунок 2.2 – событие выхода пользователя

Для реализации службы нужно переопределить следующие методы:

* OnStart – содержит код, который выполняется при запуске службы;
* OnStop – содержит код, который выполняется при остановке службы;
* OnShutdown – содержит код, который выполняется при выключении компьютера.

При занесении записи в журнал генерируется событие EntryWritten. В методе OnStart нужно получить журнал безопасности Windows и создать для него обработчик событий (листинг 2.1).

Листинг 2.1 – получение журнала безопасности и добавление обработчика событий

EventLog securityLog = new EventLog("Security");

securityLog.EntryWritten += new EntryWrittenEventHandler(SecurityLogEntryWritten);

Где SecurityLogEntryWritten – функция обработчик события. В ней мы получаем добавленную запись, а затем в зависимости от кода события выполняем определенные действия (листинг 2.2).

Листинг 2.2 – обработка события входа

public async void SecurityLogEntryWritten(object source, EntryWrittenEventArgs e)

{

    EventLogEntry eventLogEntry = e.Entry;

    switch (eventLogEntry.InstanceId)

    {

        case 4624:

            if (GetType(eventLogEntry.Message) == 2)

            {

                if (IsVirtual(eventLogEntry.Message))

                {

                    this.loginId = GetEnterId(eventLogEntry.Message);

                    var packetJSON = new PacketJSON

                    {

                        computerName = this.machineName,

                        macAddress = this.macAddress,

                        ipAddress = this.ipAddress,

                        date = eventLogEntry.TimeGenerated,

                        loginId = this.loginId,

                        operatingSystem = this.operatingSystem,

                        type = true

                    };

                    await sendPacket(packetJSON);

                }

            }

            break;

Здесь функция sendPacket отправляет JSON пакет с информацией о входе пользователя на сервер (листинг 2.3).

Листинг 2.3 – функция sendPacket

private async Task sendPacket(PacketJSON packetJSON)

{

    var payload = JsonSerializer.Serialize(packetJSON);

    var content = new StringContent(payload, Encoding.UTF8, "application/json");

    var response = await httpClient.PostAsync(url, content);

    if(response.IsSuccessStatusCode)

    {

        serviceEventLog.WriteEntry("Request sent successfully");

    } else

    {

        serviceEventLog.WriteEntry("Error in sending request to url: " + url + " statusCode: " + response.StatusCode);

    }

}

Где serviceEventLog – журнал событий, в который заносится информация о работе службы. Данные будут отсылаться в следующем виде (рисунок 2.3).

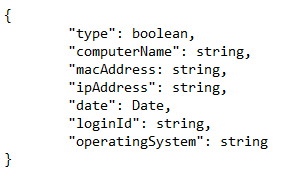


Рисунок 2.3 – пакет с информацией о входе/выходе пользователей

В данном пакете содержится следующая информация:

* type – указывает, произошел вход или выход (true – вход, false – выход);
* computerName, macAddress и ipAddress – имя компьютера, его mac-адрес и ip-адрес в локальной сети. Эти поля используются для идентификации компьютера;
* date – дата и время входа/выхода пользователя из операционной системы;
* loginId – ID входа – уникальный идентификатор, который назначается пользователю при входе в систему. С его помощью можно связать запись о входе пользователя с записью о его выходе из системы;
* operatingSystem – название и версия операционной системы.

Отправка пакета с данными о выходе пользователя реализуется следующим образом (листинг 2.4).

Листинг 2.4 – отправка данных о выходе пользователя

case 4647:

    var packetJSON = new PacketJSON

    {

        computerName = this.machineName,

        macAddress = this.macAddress,

        ipAddress = this.ipAddress,

        date = eventLogEntry.TimeGenerated,

        loginId = this.loginId,

        operatingSystem = this.operatingSystem,

        type = false

     };

     await sendPacket(packetJSON);

     break;

Аналогичные данные отправляются при остановке службы (метод OnStop) и при выключении компьютера (метод OnShutdown).

## 2.2 Общая структура серверной части приложения

Серверная часть приложения имеет следующую структуру (рисунок 2.4):

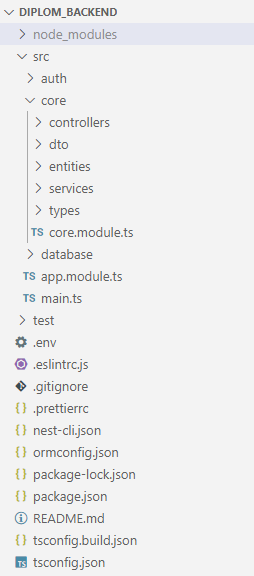


Рисунок 2.4 – структура серверной части проекта

Модуль core содержит сущности, сервисы, контроллеры и прочие классы, которые обеспечивают функциональность записи информации о входе и выходе пользователей, а также получения статистики.

Модуль auth содержит компоненты, необходимые для организации авторизации и аутентификации пользователей в приложении.

Модуль database содержит компоненты, описывающие конфигурацию базы данных.

## 2.3 Реализация записи времени входа и выхода

Для реализации хранения информации о входе и выходе пользователей из операционной системы нужно описать класс, содержащий информацию о компьютере (листинг 2.5).

Листинг 2.5 – класс Computer

@Entity({ name: 'computer', engine: 'InnoDB' })

export class Computer {

    @PrimaryGeneratedColumn()

    id: number;

    @Column({ length: 50 })

    name: string;

    @Column({ length: 12, unique: true })

    macAddress: string;

    @Column({ length: 15 })

    ipAddress: string;

    @Column({ length: 20 })

    audince: string;

    @OneToMany(

        () => LogWindows,

        logWindows => logWindows.computer,

    )

    logsWindows: LogWindows[];

    @OneToMany(

        () => DayComputerWork,

        dayComputerWork => dayComputerWork.computer,

    )

    daysComputerWork: DayComputerWork[];

    @OneToMany(

        () => MonthComputerWork,

        monthComputerWork => monthComputerWork.computer,

    )

    monthsComputerWork: MonthComputerWork[];

    @OneToMany(

        () => YearComputerWork,

        yearComputerWork => yearComputerWork.computer,

    )

    yearsComputerWork: YearComputerWork[];

    @OneToMany(

        () => PeriodComputerWork,

        periodComputerWork => periodComputerWork.computer,

    )

    periodsComputerWork: PeriodComputerWork[];

    @DeleteDateColumn({ name: 'deleted\_date', type: 'timestamp' })

    deletedDate: Date;

}

Класс Computer содержит имя компьютера, его MAC адрес и IP адрес в сети, а также аудиторию, в которой находится компьютер.

Каждое событие входа и выхода пользователей из системы будет описываться классом Log (листинг 2.6).

Листинг 2.6 – класс Log

@Entity({ name: 'log', engine: 'InnoDB' })

export class Log {

    @PrimaryGeneratedColumn()

    id: number;

    @Column()

    type: boolean;

    @Column()

    date: Date;

    @Column({ name: 'login\_id', length: 15 })

    loginId: string;

    @Column({ name: 'operating\_system', length: 50 })

    operatingSystem: string;

    @ManyToOne(

        () => Computer,

        computer => computer.logs,

        { onDelete: 'RESTRICT', onUpdate: 'RESTRICT' },

    )

    @JoinColumn({ name: 'computer\_id', referencedColumnName: 'id'

    computer: Computer;

    @DeleteDateColumn({ name: 'deleted\_date', type: 'timestamp' })

    deletedDate: Date;

}

Класс LogWindows содержит поля type – поле типа boolean (true – вход, false – выход), дату события, loginId – специальный идентификатор, генерируемый операционной системой при входе пользователя (необходим для сопоставления события входа и выхода пользователя из системы), наименование операционной системы и информацию о компьютере, на котором произошло событие входа/выхода.

Для быстрого получения времени работы компьютеров за определенный период создадим 3 дополнительных класса, каждый из которых будет хранить время работы компьютера на определенный день (листинг 2.7), месяц и год.

Листинг 2.7 – класс DayComputerWork

@Entity({ name: 'day\_computer\_work', engine: 'InnoDB' })

export class DayComputerWork {

    @PrimaryGeneratedColumn()

    id: number;

    @Column()

    date: Date;

    @Column()

    hours: number;

    @ManyToOne(

        () => Computer,

        computer => computer.daysComputerWork,

        { onDelete: 'RESTRICT', onUpdate: 'RESTRICT' },

    )

    @JoinColumn({ name: 'computer\_id', referencedColumnName: 'id' })

    computer: Computer;

    @DeleteDateColumn({ name: 'deleted\_date', type: 'timestamp' })

    deletedDate: Date;

}

Данный класс содержит дату начала периода (в данном случае – день, для класса MonthComputerWork – день начала месяца и т.д.), информацию о компьютере и его время работы за данный период. Классы для хранения времени работы по месяцам и годам реализованы аналогичным способом.

Для хранения периодов работы компьютера создадим класс PeriodComputerWork (листинг 2.8).

Листинг 2.8 – класс PeriodComputerWork

@Entity({ name: 'period\_computer\_work', engine: 'InnoDB' })

export class PeriodComputerWork {

    @PrimaryGeneratedColumn()

    id: number;

    @Column()

    dateStart: Date;

    @Column()

    dateEnd: Date;

    @Column()

    operatingSystem: string;

    @Column()

    loginId: string;

    @ManyToOne(

        () => Computer,

        computer => computer.periodsComputerWork,

        { onDelete: 'RESTRICT', onUpdate: 'RESTRICT' },

    )

    @JoinColumn({ name: 'computer\_id', referencedColumnName: 'id' })

    computer: Computer;

    @DeleteDateColumn({ name: 'deleted\_date', type: 'timestamp' })

    deletedDate: Date;

}

Данный класс содержит поля dateStart и dateEnd – время начала и конца работы компьютера соответственно.

Рассмотрим метод, который записывает данные о входе или выходе пользователей из операционной системы (листинг 2.9):

Листинг 2.9 – получение информации о компьютере из пакета

const readComputerOptions = {

    macAddress: packet.macAddress,

};

let computer = await this.computerService.readOne(readComputerOptions);

if(computer == null) {

    computer = await this.computerService.create({

      macAddress: packet.macAddress,

        ipAddress: packet.ipAddress,

        name: packet.computerName,

    });

}

Первым шагом получаем информацию о компьютере, которая пришла в пакете, из БД. Если записи о таком компьютере в базе данных не существует, создаем её.

Листинг 2.10 – создание записи в таблице логов

const logWindowsOptions = {

    computerId: computer.id,

    type: packet.type,

    loginId: packet.loginId,

    date: packet.date,

    operatingSystem: packet.operatingSystem,

};

this.logWindowsService.create(logWindowsOptions);

Получаем из пакета данные, необходимые для создания лога, и создаем его (листинг 2.10).

Следующим шагом создаем запись в таблице периодов (листинг 2.11).

Листинг 2.11 – запись в таблицу периодов работы компьютера

const periodComputerWorkOptions = {

    computerId: computer.id,

    dateStart: packet.date,

    operatingSystem: packet.operatingSystem,

    loginId: packet.loginId,

};

if(packet.type) {

    this.periodComputerWorkService.create(periodComputerWorkOptions);

} else {

    const periodComputerWorkService = await this.periodComputerWorkService.readOne(periodComputerWorkOptions);

    const updatedComputerWork = await this.periodComputerWorkService.update(periodComputerWorkService.id, { dateEnd: packet.date });

}

Если пользователь вошел в систему, то просто создаем новую запись в таблице периодов с пустой датой выхода. Иначе ищем существующую запись с данными о входе и обновляем ей время выхода.

Следующим шагом идет запись в таблицы с временем работы компьютера по дням, месяцам и годам. Рассмотрим запись в таблицу с данными по дням (листинг 2.12), для таблиц с данными по годам и месяцам код аналогичный.

Листинг 2.12 – получение дней, на которые нужно занести статистику

if(getDayStart(dateStart).getTime() !== getDayStart(dateEnd).getTime()) {

    let currentDay = dateStart;

    let nextDay = getNextDay(dateStart);

    while(nextDay < dateEnd) {

        dayHours.push({ hours: getDateDiffHours(currentDay, nextDay), day: getDayStart(currentDay) });

        currentDay = nextDay;

        nextDay = getNextDay(currentDay);

    }

    dayHours.push({ hours: getDateDiffHours(currentDay, dateEnd), day: currentDay });

} else {

    dayHours.push({ hours: getDateDiffHours(dateStart, dateEnd), day: getDayStart(dateStart) });

}

Если день даты начала не совпадает с днем даты окончания работы компьютера (например, пользователь зашел поздно вечером и вышел из системы только на следующий день), то данные нужно заносить сразу на несколько дней. Для этого перебираем все дни между датами начала и конца и считаем количество часов работы компьютера на эти дни. Если день даты начала совпадает с днем даты конца работы, то просто вносим одну запись. Для месяца и года код идентичен.

Следующим шагом идет добавление (или обновление) записей о времени работы компьютера по дням (листинг 2.13), месяцам и годам.

Листинг 2.13 – создание и обновление записей в таблице времени работы компьютера по дням

for(let el of dayHours) {

  const dayComputerWorkOptions = {

        date: el.day,

        computerId: computer.id,

        operatingSystem: packet.operatingSystem,

    }

    let dayComputerWork = await this.dayComputerWorkService.readOne(dayComputerWorkOptions);

    if(dayComputerWork == null) {

        dayComputerWork = await this.dayComputerWorkService.create({

            hours: 0,

            ...dayComputerWorkOptions,

        });

    }

    if(!packet.type) {

    await this.dayComputerWorkService.update(dayComputerWork.id, { hours: dayComputerWork.hours + el.hours });

    }

}

Перебираем все дни, на которые нужно внести данные. Если записи о данном компьютере на определенный день не существует, то создаем ее. Если type = false (т.е. выход пользователя), то обновляем время работы компьютера в этот день.

Следующим шагом реализуем метод, который для заданных компьютеров, а также даты начала и конца периода вернет периоды, в течении которых компьютер работал.

Создадим DTO (Data Transfer Object – объект, который описывает, как данные передаются в сети), который будет передаваться для получения периодов и времени работы компьютеров за период (листинг 2.14).

Листинг 2.14 – класс ReadStatisticsDto

export class ReadStatisticsDto {

    @IsOptional()

    @IsObject()

    @ValidateNested()

    @Type(() => PaginationDto)

    public pagination?: PaginationDto;

    @IsOptional()

    @IsObject()

    @ValidateNested()

    @Type(() => SortingDto)

    public sorting?: SortingDto;

    @IsOptional()

    @IsArray()

    @IsInt({ each: true })

    @Transform(computers => Array.from(computers).map(id => Number(id)))

    computers?: number[];

    @IsNotEmpty()

    @IsDate()

    dateStart: Date;

    @IsNotEmpty()

    @IsDate()

    dateEnd: Date;

    @IsOptional()

    @IsString()

    @MaxLength(50)

    operatingSystem?: string;

}

В этом классе используются декораторы из пакета class-validator для валидации отправляемых данных. Поле computers содержит массив id компьютеров, для которых нужно получить статистику, operatingSystem – операционную систему, а поля dateStart и dateEnd содержат время начала и конца периода. Поля sorting и pagination используются для сортировки данных и пагинации соответственно.

Статистику по периодам работы будем получать в следующем виде (рисунок 2.5).

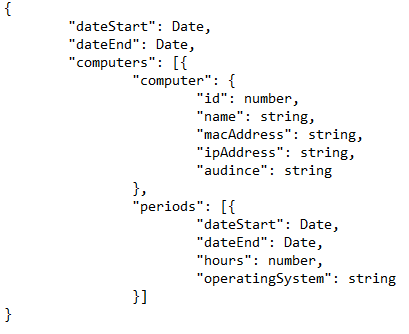


Рисунок 2.5 – пакет со статистикой по периодам работы

Пакет со статистикой по периодам работы содержит следующие поля: dateStart и dateEnd – период, за который требуется получить статистику; computers – массив записей, содержащих информацию о компьютере (поле computer с соответствующей информацией) и массив периодов его работы (поле periods).

Для обеспечения требуемого формата получения данных создадим следующие классы.

Листинг 2.15 – класс StatisticsPeriodDate

export class StatisticsPeriodDate {

    dateStart: Date;

    dateEnd: Date;

    hours: number;

}

Этот класс (листинг 2.15) хранит дату начала и конца периода работы компьютера, а также время работы.

Листинг 2.16 – класс StatisticsPeriodMember

export class StatisticsPeriodMember {

    computer: Computer;

    periods: StatisticsPeriodDate[]

}

Данный класс (листинг 2.16) содержит компьютер и массив периодов его работы.

Листинг 2.17 – класс StatisticsPeriod

export class StatisticsPeriod {

    dateStart: Date;

    dateEnd: Date;

    computers: StatisticsPeriodMember[];

}

Класс StatisticsPeriod (листинг 2.17) содержит дату начала и конца периода, за который получаем данные о периодах работы компьютера, а также массив экземпляров класса StatisticsPeriodMember.

Теперь создадим метод для получения требуемых данных из БД (листинг 2.18).

Листинг 2.18 – получение данных из БД

const queryBuilder = this.computerRepository.createQueryBuilder("computer");

queryBuilder

    .select(['computer.id', 'computer.name', 'computer.macAddress', 'computer.ipAddress', 'computer.audince'])

    .leftJoin('computer.periodsComputerWork', 'periodsComputerWork')

    .addSelect([

        'periodsComputerWork.dateStart',

        'periodsComputerWork.dateEnd',

        'periodsComputerWork.loginId',

        'periodsComputerWork.operatingSystem',

    ])

    .where('periodsComputerWork.dateEnd IS NOT NULL')

    .andWhere(`(periodsComputerWork.dateStart >= :dateStart AND periodsComputerWork.dateStart <= :dateEnd) OR

    (periodsComputerWork.dateEnd <= :dateEnd AND periodsComputerWork.dateEnd >= :dateStart)

    `, {

        dateStart: readStatisticsDto.dateStart,

        dateEnd: readStatisticsDto.dateEnd

});

if (readStatisticsDto.operatingSystem) {

            queryBuilder.andWhere('periodComputerWork.operatingSystem LIKE :operatingSystem', {

                operatingSystem: "%" + readStatisticsDto.operatingSystem + "%",

            });

        }

        if(readStatisticsDto.computers.length !== 0) {

            queryBuilder.andWhere('computer.id IN (:...computers)', {

                computers: readStatisticsDto.computers,

            });

        }

        if(readStatisticsDto.sorting) {

queryBuilder.orderBy(readStatisticsDto.sorting.column, readStatisticsDto.sorting.direction);

}

if(readStatisticsDto.pagination) {

queryBuilder.skip(readStatisticsDto.pagination.page \* readStatisticsDto.pagination.size).take(readStatisticsDto.pagination.size);

}

const entities = await queryBuilder.getMany();

Для начала создаем объект queryBuilder, который позволяет описывать требуемый запрос вызовом соответствующих методов. Из базы данных получаем записи, удовлетворяющие условиям, описанным в readStatisticsDto: выбираем записи с датой начала и окончания, которые входят в заданный интервал, и дата окончания работы у которых известна. К запросу также добавляем фильтры по компьютерам и операционной системе (если они заданы), настраиваем сортировку по заданному столбцу и в заданной последовательности. В конце добавляем пагинацию и получаем результат.

Следующим шагом создаем объект класса StatisticsPeriod (листинг 2.19).

Листинг 2.19 – возврат результата

const arrayPeriod = entities.map(function(el) {

    let computer = new Computer;

    computer.id = el.id;

    computer.name = el.name;

    computer.ipAddress = el.ipAddress;

    computer.macAddress = el.macAddress;

    computer.audince = el.audince;

return {

        computer: computer,

        periods: el.periodsComputerWork.map(function(el1) {

            const realDateStart = dateStart.getTime() > el1.dateStart.getTime() ? dateStart : el1.dateStart;

            const realDateEnd = dateEnd.getTime() < el1.dateEnd.getTime() ? dateEnd : el1.dateEnd;

            return {

                dateStart: realDateStart,

                dateEnd: realDateEnd,

                hours: Number(getDateDiffHours(realDateStart, realDateEnd).toFixed(4)),

                operatingSystem: el1.operatingSystem,

            }

        }),

    }

});

С помощью метода массива map преобразуем полученный из базы данных массив записей в объект класса StatisticsPeriod. Создавая экземпляры класса StatisticsPeriodMember, проверяем, чтобы дата начала полученной записи была позднее даты начала, на которую нужно получить статистику. Это может произойти, когда, например, статистика нужна с 20:00:00 до 24:00:00 какого-нибудь дня, а компьютер работал с 19:00:00 до 23:00:00. В этом случае время работы компьютера нужно брать с 20:00:00 до 23:00:00. То же самое действие производится и с датой конца.

Реализуем метод, который будет получать время работы компьютеров в часах за определенный период.

Статистику по часам работы будем получать в следующем виде (рисунок 2.5).

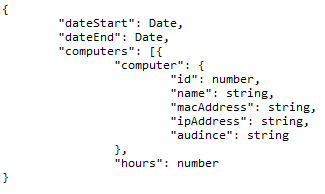


Рисунок 2.6 – пакет с статистикой по часам работы

Данный пакет содержит следующие поля: dateStart и dateEnd – период, за который требуется получить статистику; computers – массив записей с сведениями о компьютере (поле computer) и часами его работы (поле hours).

Для обеспечения нужного формата создадим следующие классы.

Листинг 2.20 – класс StatisticsHoursMember

export class StatisticsHoursMember {

    computer: Computer;

    hours: number;

}

Листинг 2.21 – класс StatisticsHours

export class StatisticsHours {

    dateStart: Date;

    dateEnd: Date;

    computers: StatisticsHoursMember[];

}

Класс StatisticsHoursMember (листинг 2.20) содержит компьютер и количество часов его работы, а класс StatisticsHours (листинг 2.21) – дату начала и конца периода, на которые нужно получить данные, и список объектов класса StatisticsHoursMember.

Теперь реализуем алгоритм, который будет получать время работы компьютера за период, используя таблицы с данными на день, месяц и год.

Допустим, нужно получить время работы с 11 сентября 2020 года по 10 апреля 2022 (рисунок 2.7).

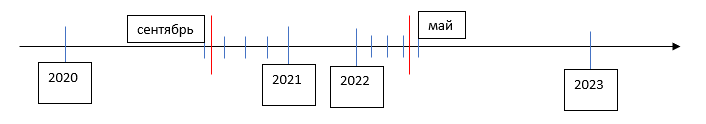


Рисунок 2.7 – ось времени

Перебираем все года, что лежат между датой начала и датой конца периода, и получаем статистику по времени работы компьютеров на эти года (рисунок 2.8).

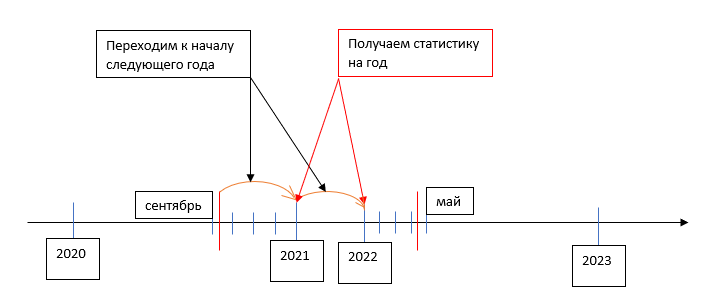


Рисунок 2.8 – получение статистики по годам

В данном случае мы за раз получим статистику за весь 2021 год, но за 2022 статистику брать нельзя, т.к. она нужна не за весь год целиком, а только за его часть.

Следующим шагом получаем статистику по месяцам (рисунок 2.9).

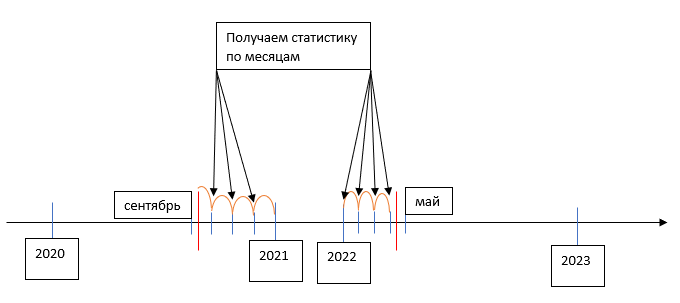


Рисунок 2.9 – получение статистики по месяцам

В данном случае мы сразу получим статистику за октябрь, ноябрь, декабрь 2020 года и январь, февраль, март 2022 года. Осталось только перебрать оставшиеся дни сентября 2020 года и мая 2022 года.

Создадим метод, который по переданным датам дней, месяцев и лет получит для них время работы, просуммирует и вернет экземпляр класса StatisticsHours.

Первым делом получим данные из базы данных (листинг 2.22).

Листинг 2.22 – получение данных из базы данных

const queryBuilder = this.computerRepository.createQueryBuilder("computer");

queryBuilder

    .select(['computer.id', 'computer.name', 'computer.macAddress', 'computer.ipAddress', 'computer.audince']);

if(options.datesDay.length !== 0) {

queryBuilder.leftJoin('computer.daysComputerWork', 'daysComputerWork', 'daysComputerWork.date IN (:...dates)', {

        dates: options.datesDay,

    })

    .addSelect('daysComputerWork.hours');

}

if(options.datesMonth.length !== 0) {

queryBuilder.leftJoin('computer.monthsComputerWork', 'monthsComputerWork', 'monthsComputerWork.date IN (:...dates)', {

    dates: options.datesMonth,

    })

    .addSelect('monthsComputerWork.hours');

}

if(options.datesYear.length !== 0) {

queryBuilder.leftJoin('computer.yearsComputerWork', 'yearsComputerWork', 'yearsComputerWork.date IN (:...dates)', {

    dates: options.datesYear,

    })

    .addSelect('yearsComputerWork.hours');

}

if (options.operatingSystem) {

queryBuilder.andWhere('dayComputerWork.operatingSystem LIKE :operatingSystem', {

    operatingSystem: "%" + options.operatingSystem + "%",

});

}

if(options.computerIds.length !== 0) {

  queryBuilder.andWhere('computer.id IN (:...computers)', {

        computers: options.computerIds,

});

}

if(options.sorting) {

    queryBuilder.orderBy(options.sorting.column, options.sorting.direction);

}

if(options.pagination) {

    queryBuilder.skip(options.pagination.page \* options.pagination.size).take(options.pagination.size);

}

const entities = await queryBuilder.getMany();

Получаем из базы данных информацию о всех компьютерах, затем, если массив с датами лет не пустой, присоединяем таблицу с временем работы компьютеров по годам. Так же поступаем с массивами с датами месяцев и дней. Добавляем фильтры на компьютеры и операционную систему, сортировку и пагинацию и получаем массив записей.

Следующим шагом нужно сформировать массив экземпляров класса StatisticsHoursMember (листинг 2.23).

Листинг 2.23 – формирование массива результатов

const computersArray = entities.map(function(el) {

    let computer = new Computer;

    computer.name = el.name;

    computer.macAddress = el.macAddress;

    computer.ipAddress = el.ipAddress;

    computer.audince = el.audince;

    computer.id = el.id;

    const dayHours = el.daysComputerWork ? el.daysComputerWork.reduce((acc, el1) => acc + el1.hours, 0) : 0;

    const monthHours = el.monthsComputerWork ? el.monthsComputerWork.reduce((acc, el1) => acc + el1.hours, 0) : 0;

    const yearHours = el.yearsComputerWork ? el.yearsComputerWork.reduce((acc, el1) => acc + el1.hours, 0) : 0;

    return {

        computer: computer,

        hours: dayHours + monthHours + yearHours,

    }

});

Перебираем массив записей, извлекаем информацию о компьютере и, если записи о времени компьютера на переданные дни существуют, суммируем их. То же делаем и для массивов с записями о времени работы компьютеров по месяцам и годам. При формировании результата суммируем полученные часы работы.

Листинг 2.24 – получение времени работы по годам и по оставшимся месяцам и дням последнего года

let dateStart = readStatisticsDto.dateStart;

let dateEnd = readStatisticsDto.dateEnd;

let datesYear = [];

let datesMonth = [];

let datesDay = [];

let nextYear = getYearNext(dateStart);

while(nextYear.getTime() <= dateEnd.getTime()) {

    datesYear.push(nextYear);

    nextYear = getNextYear(nextYear);

}

if(datesYear.length !== 0) {

    const lastYear = datesYear[datesYear.length - 1];

    if(getYearEnd(lastYear).getTime() !== dateEnd.getTime()) {

        let currentMonth = datesYear.pop();

        while(currentMonth.getTime() <= dateEnd.getTime()){

            datesMonth.push(currentMonth);

            currentMonth = getNextMonth(currentMonth);

        }

        if(datesMonth.length !== 0) {

            const lastMonth = datesMonth[datesMonth.length - 1];

            if(getMonthEnd(lastMonth).getTime() !== dateEnd.getTime()) {

                let currentDay = lastYear;

                currentDay = datesMonth.pop();

                while(currentDay.getTime() <= dateEnd.getTime()) {

                    datesDay.push(currentDay);

                    currentDay = getNextDay(currentDay);

                }

            }

        }

    }

}

Создаем массивы, в которых будут хранится даты начала периодов, на которые нужно получить данные о времени работы компьютеров (листинг 2.24). Затем перебираем последующие года, постепенно добавляя их в массив. Если этот массив оказался не пустой, то проверяем, нужно ли брать статистику за последний год целиком (это нужно делать в том случае, если конец последнего занесенного в массив года совпадает с датой конца периода, на который нужно получить статистику). Если статистику за последний год нужно брать не целиком, то убираем его из массива. Последовательно перебираем месяцы последнего года, вновь проверяем, нужно ли брать статистику за последний месяц целиком. Если нет, то убираем этот месяц из массива месяцев и перебираем оставшиеся дни этого месяца.

Осталось получить статистику за оставшиеся месяцы и дни неполного первого года (листинг 2.25). Если дата начала, на которую нужно получить статистику, совпадает с датой начала года, то просто берем статистику за весь год, иначе перебираем месяцы этого года и добавляем их в массив месяцев, на которые нужно получить статистику. Проверяем, нужно ли брать статистику за первый месяц целиком, и, если это не так, перебираем оставшиеся дни первого месяца.

Листинг 2.25 – получение времени работы по оставшимся месяцам и дням первого года

if(getYearStart(dateStart).getTime() === dateStart.getTime()) {

    datesYear.push(dateStart);

} else {

    let nextMonth = getNextMonth(dateStart);

    while(nextMonth.getTime() <= dateEnd.getTime()) {

        datesMonth.push(nextMonth);

        nextMonth = getNextMonth(nextMonth);

    }

    if(datesMonth.length !== 0) {

        dateEnd = getNextMonth(dateStart);

    }

    if(getMonthStart(dateStart).getTime() !== dateStart.getTime()) {

        datesDay.push(dateStart);

        let nextDay = getNextDay(dateStart);

        while(nextDay.getTime() <= dateEnd.getTime()) {

            datesDay.push(nextDay);

            nextDay = getNextDay(nextDay);

      }

    } else {

        datesMonth.push(dateStart);

    }

}

const computersWork = await this.dayStatisticsService.readWorkHours({

    datesDay: datesDay,

    datesMonth: datesMonth,

    datesYear: datesYear,

    dateStart: dateStart,

    dateEnd: dateEnd,

    operatingSystem: readStatisticsDto.operatingSystem,

    computerIds: computersArray,

    sorting: readStatisticsDto.sorting,

    pagination: readStatisticsDto.pagination,

});

return computersWork;

После получения дней, месяцев и лет, на которые нужно получить статистику, вызываем метод readWorkHours, передавая в качестве аргумента соответствующие массивы и возвращаем результат.

## 2.4 Аутентификация

Для авторизации и аутентификации в приложении будут использоваться JWT-токены.

JWT или JSON Web Token представляет собой надежный и безопасный метод передачи информации между сторонами в виде JSON-объектов. JWT-токен состоит из трех частей, закодированных в формате Base64. Эти части (заголовок, полезная нагрузка, подпись) разделены между собой точками [12]. Заголовок содержит информацию о токене, а именно тип токена и алгоритм шифрования, полезная нагрузка хранит данные, которые разработчик хочет передавать в токене (например, JSON-объект), а подпись позволяет проверить, что токен не был изменен в процессе передачи по сети.

Токен создается сервером при успешной аутентификации, передается клиенту, который его в дальнейшем использует для подтверждения подлинности. В полезной нагрузке токена будет храниться логин и id пользователя, а также список его ролей.

Для работы с JWT токенами используем пакет PassportJS. PassportJS — это пакет npm, который обеспечивает аутентификацию в Node.js. Passport — это универсальная модульная платформа веб-приложений, которую можно легко интегрировать в любое веб-приложение на платформе Node.js [13].

Первым шагом создаем сущности для пользователя (листинг 2.26) и роли (листинг 2.27).

Листинг 2.26 – класс User

@Entity({ name: 'user', engine: 'InnoDB' })

export class User {

    @PrimaryGeneratedColumn()

    id: number;

    @Column({ length: 100, unique: true })

    login: string;

    @Column({ length: 100 })

    password: string;

    @ManyToMany(() => Role)

    @JoinTable()

    roles: Role[];

    @DeleteDateColumn({ name: 'deleted\_date', type: 'timestamp' })

    deletedDate: Date;

}

Листинг 2.27 – класс Role

@Entity({ name: 'role', engine: 'InnoDB' })

export class Role {

    @PrimaryGeneratedColumn()

    id: number;

    @Column({ length: 100, unique: true })

    name: string;

    @ManyToMany(() => User)

    users: User[];

    @DeleteDateColumn({ name: 'deleted\_date', type: 'timestamp' })

    deletedDate: Date;

}

Класс User содержит логин и хешированный пароль пользователя, а класс Role – название роли.

Создадим декоратор HasRoles, который будет передавать список разрешенных ролей для их последующей сверки с ролями пользователя (листинг 2.28).

Листинг 2.28 – декоратор HasRoles

export const HasRoles = (...roles: String[]) => SetMetadata('roles', roles);

В сервисе авторизации создадим 2 метода – validateUser() (листинг 2.29), который по полученному логину и паролю ищет пользователя в БД, и login() (листинг 2.30), который создает для пользователя JWT токен.

Листинг 2.29 – метод validateUser

async validateUser(login: string, password: string): Promise<any> {

    const user = await this.userService.readOne({ login: login });

    if(user === null) {

      return null;

    }

    const hashPassword = await argon2.verify(user.password, password);

    if(!hashPassword) {

      return null;

    }

    const result = {

      id: user.id,

      login: user.login,

      roles: (await this.userService.readUserRoles(user.id)).map(role => ({

          id: role.id,

          name: role.name

        })

      ),

    }

    return result;

}

Листинг 2.30 – метод login

async login(user: any) {

    const payload = {

        login: user.login,

        roles: user.roles,

    };

    return {

        access\_token: this.JwtService.sign(payload),

    };

}

Создадим класс RolesGuard с методом canActivate (листинг 2.31). Если данный метод возвращает true, то доступ авторизированному пользователю разрешен.

Листинг 2.31 – метод canActivate

canActivate(context: ExecutionContext): boolean {

    const requiredRoles = this.reflector.getAllAndOverride<String[]>('roles', [

      context.getHandler(),

      context.getClass(),

    ]);

    if (!requiredRoles) {

      return true;

    }

    const request = context.switchToHttp().getRequest();

    const token = this.extractTokenFromHeader(request);

    const user = Object(this.jwtService.decode(token));

    return requiredRoles.some((role) => (user?.roles?.map(role => role.name)).includes(role));

}

Данный метод извлекает список ролей, для которых разрешен доступ, из декоратора HasRoles, а также список ролей из JWT токена. При совпадении хотя бы одной роли доступ разрешается.

Теперь для того, чтобы разрешить доступ только авторизованным пользователям, перед соответствующим методом (или классом целиком) будем использовать декоратор UseGuards(AuthGuard(“jwt”)). Для того, чтобы разрешить доступ только пользователям с определенной ролью, будем использовать декоратор HasRoles (передав в него список доступных ролей) и декоратор UseGuards(RolesGuard).

## 2.5 Тестирование службы

### **2.5.1 Установка службы**

Для запуска службы воспользуемся исполняемым файлом sc.exe, который установит службу (рисунок 2.10).

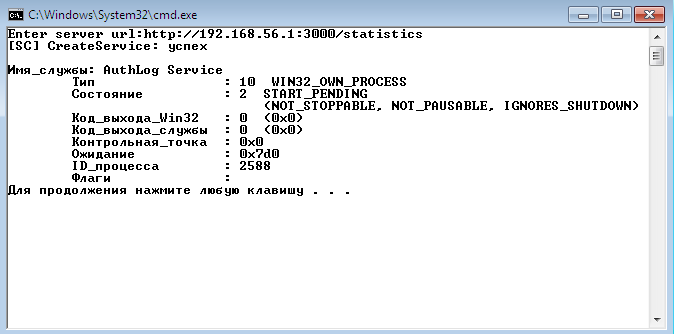


Рисунок 2.10 – запуск службы

Для тестирования установим службу на виртуальной машине (рисунок 2.11). При запуске указываем адрес, на который будут отправляется запросы, в нашем случае адрес основной машины.

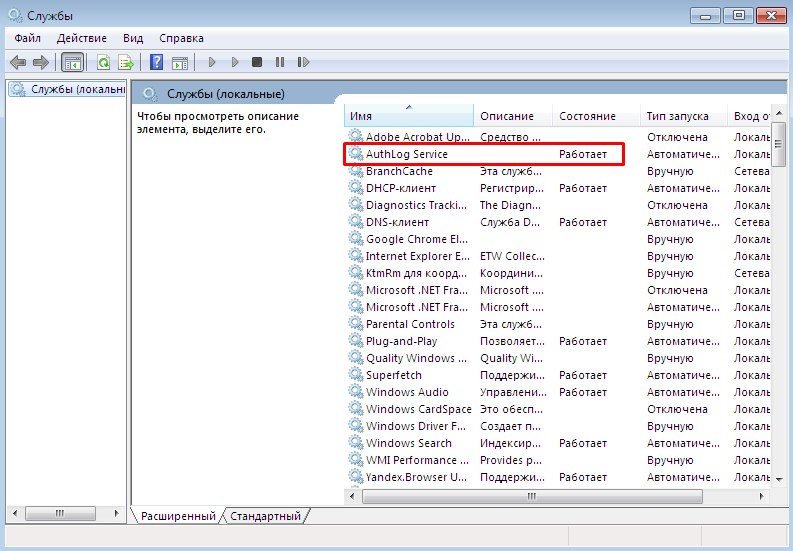


Рисунок 2.11 – служба запущена

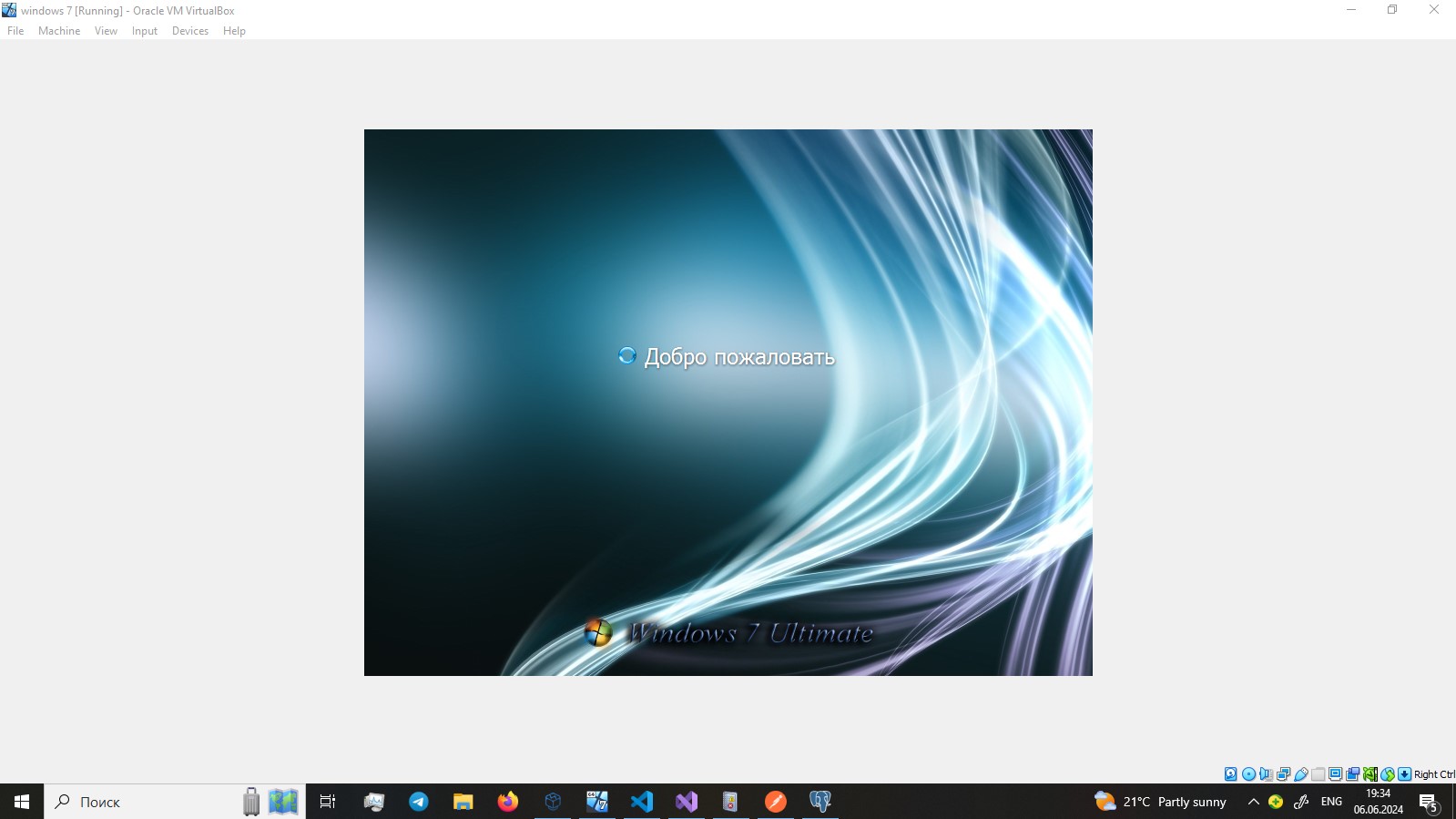


Рисунок 2.12 – вход пользователя в систему

После входа пользователя (рисунок 2.12) в базе данных создаются следующие записи:

* В таблице log добавляется запись о входе пользователя (рисунок 2.13);

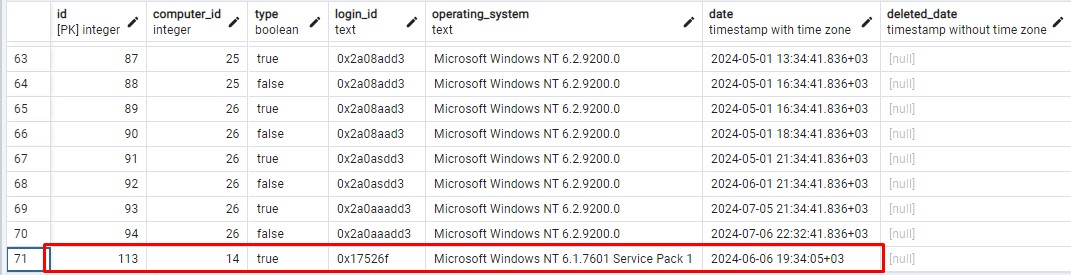


Рисунок 2.13 – запись в таблице log

* В таблице period\_computer\_work – запись о начале сеанса пользователя (дата конца на данный момент равна null) (рисунок 2.14);

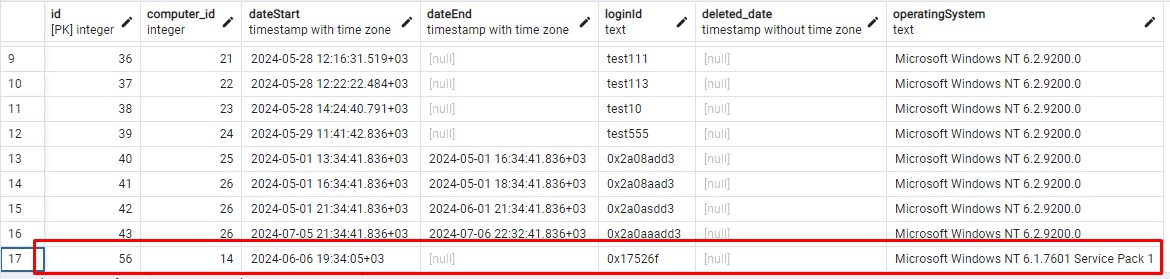


Рисунок 2.14 – запись в таблице period\_computer\_work

* В таблицах day\_computer\_work (рисунок 2.15), month\_computer\_work (рисунок 2.16), year\_computer\_work (рисунок 2.17) – записи на соответствующий день, месяц, год.

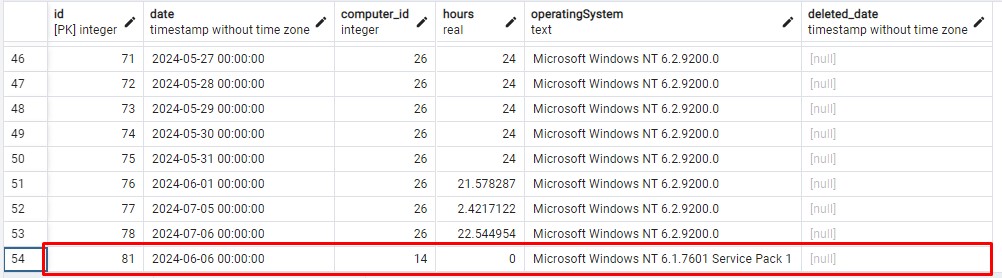


Рисунок 2.15 – запись в таблице day\_computer\_work

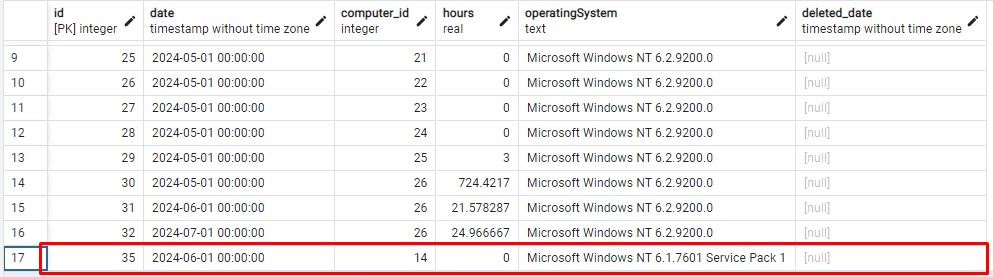


Рисунок 2.16 – запись в таблице month\_computer\_work

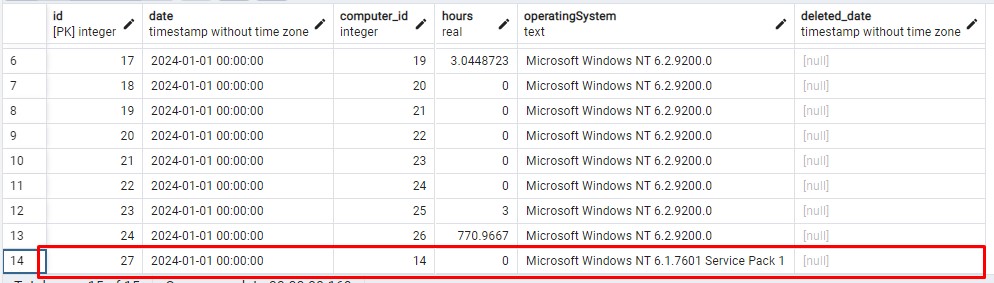


Рисунок 2.17 – запись в таблице year\_computer\_work

После выхода пользователя из системы в базе данных создаются следующие записи:

* В таблице log – запись о выходе пользователя из системы (рисунок 2.18);

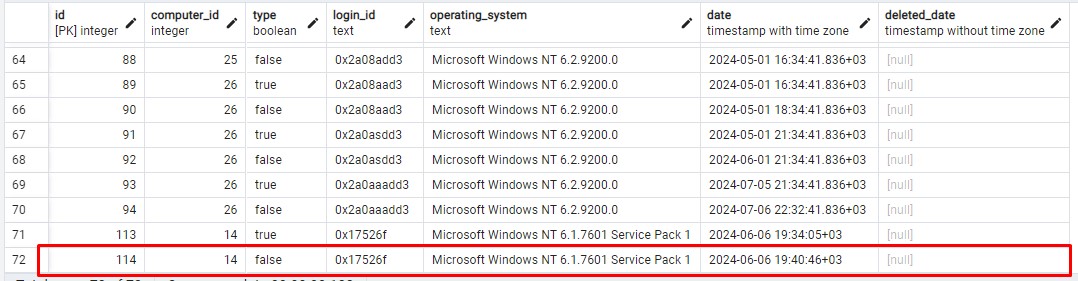


Рисунок 2.18 – запись в таблице log

* В таблице period\_computer\_work – обновляется запись о входе пользователя (добавляется дата выхода) (рисунок 2.19);

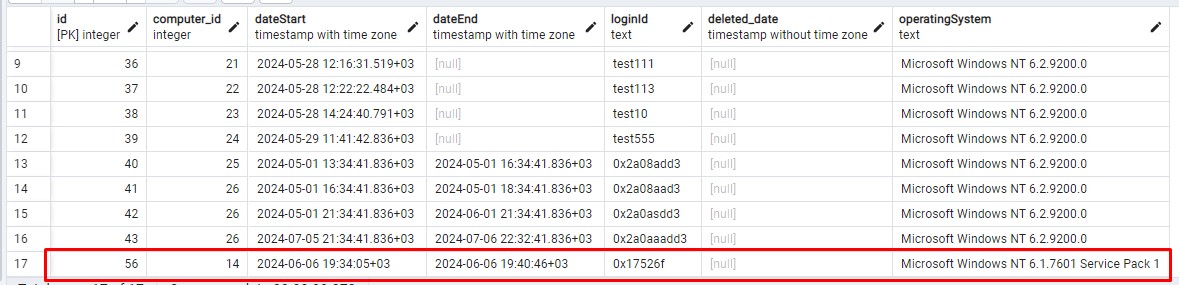


Рисунок 2.19 – запись в таблице period\_computer\_work

* В таблице day\_computer\_work (рисунок 2.20) обновляется время работы компьютера (то же самое для таблиц month\_computer\_work и year\_computer).

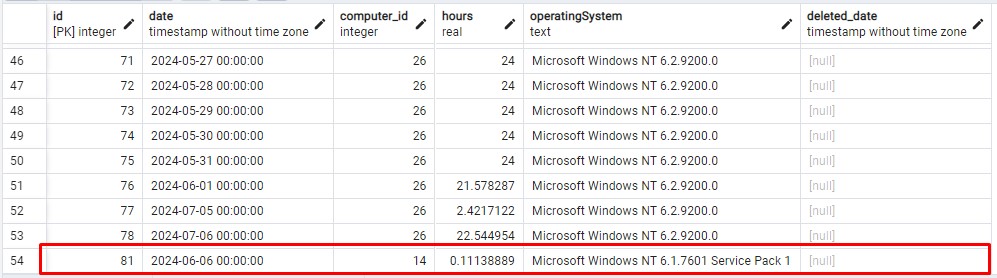


Рисунок 2.20 – запись в таблице day\_computer\_work

Теперь можно получить статистику по работе компьютера, отправив соответствующий запрос. Рисунок 2.21 показывает статистику по периодам, а рисунок 2.22 – статистику по часам работы компьютера.

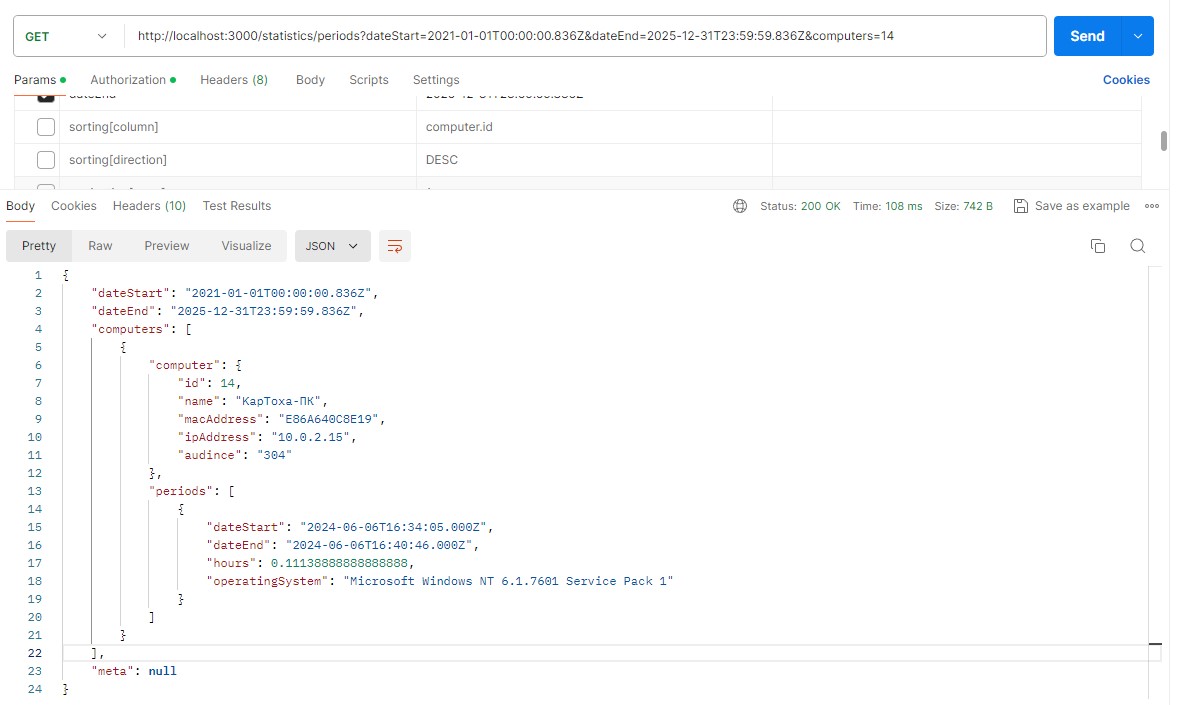


Рисунок 2.21 – статистика по работе компьютера по периодам

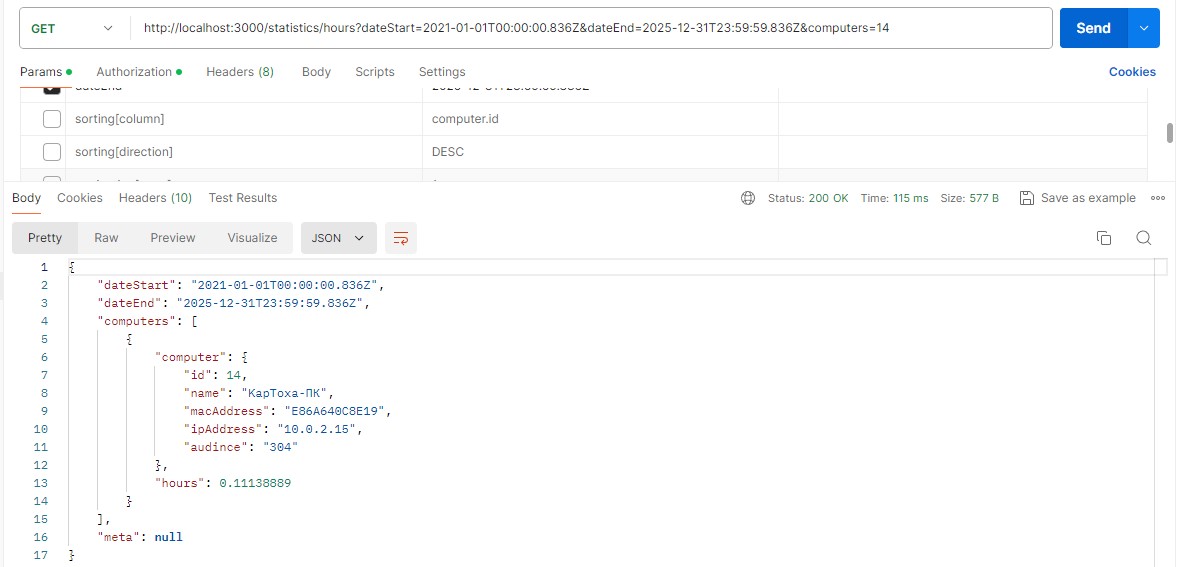


Рисунок 2.22 – статистика по работе компьютера по часам

### **2.5.2 Нагрузочное тестирование**

Нагрузочное тестирование (Perfomance testing) – это автоматизированное тестирование, имитирующее работу определенного количества пользователей на каком-либо общем ресурсе. Оно проводится с целью определения скорости работы ПО под нагрузкой, а также масштабируемости и потребления ресурсов [14].

Нагрузочное тестирование можно провести с помощью инструмента Apache JMeter.

Apache JMeter — это инструмент для проведения нагрузочного тестирования, представляющий собой десктопное приложение с открытым исходным кодом на базе Java. JMeter позволяет определить, может ли тестируемое веб-приложение удовлетворять требованиям высокой нагрузки или нет. Он также помогает проанализировать работу всего сервера под высокой нагрузкой [15].

Для нагрузочного тестирования отправим 5000 запросов о входе и выходе пользователей на сервер, данные для которых сформируем следующим образом: каждую вторую запись будем чередовать type (вход или выход) и loginId (имитируя тем самым вход и выход пользователя из операционной системы). Каждую четвертую запись будем менять операционную систему, каждую десятую запись будем менять компьютер. На каждой итерации к дате отправления будем случайно добавлять от 0 до 3 часов.

Результаты тестирования предоставлены на следующем рисунке (рисунок 2.23).

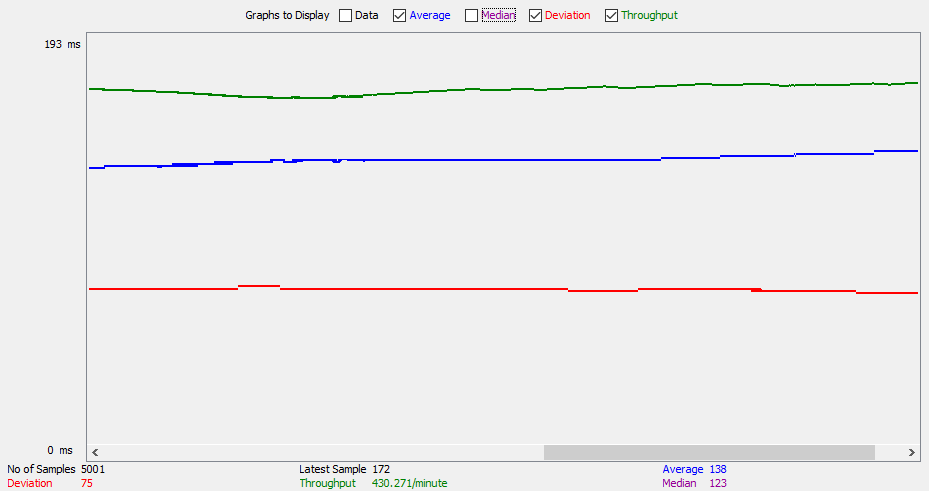


Рисунок 2.23 – результаты тестирования для внесения записи о входе/выходе пользователей

На данном рисунке значения Throughput (пропускная способность), Deviation (отклонение) и Average (среднее) являются параметрами, по которым можно проанализировать производительность сервера. На графике зеленым цветом обозначена пропускная способность, красным отклонение и синим среднее значение.

Пропускная способность является наиболее важным параметром. Она отражает способность сервера справляться с большой нагрузкой. Чем выше пропускная способность, тем выше производительность сервера [15]. В данном тесте пропускная способность составляет 430.271 в минуту, что означает, что сервер может обрабатывать 430.271 запросов о входе/выходе пользователей из ОС в минуту.

Average (среднее) указывает среднее время отклика сервера (в нашем случае 138 мс), а Deviation указывает на отклонение от среднего значения (в нашем случае 75 мс).

Проведя таким же образом тестирования для получения статистики по периодам и часам (используя данные, сгенерированные предыдущими запросами) получили, что среднее время выполнения запроса для получения статистики по периодам составляет 895 мс (67.039 запросов в минуту), а для статистики по часам работы – 636 мс (94.339 запросов в минуту).

## 2.6 Выводы из второго раздела

В первом подразделе были рассмотрены коды событий, которые будут использоваться для отслеживания входа и выхода пользователей из операционной системы, описан ход разработки службы и её конфигурация.

В последующих подразделах была рассмотрена общая структура серверной части приложения, описана разработка функционала записи о входе/выходе пользователей из ОС и получения статистики о времени работы компьютеров.

В четвертом подразделе был описан ход разработки функционала авторизации и аутентификации пользователей в приложении.

В последнем подразделе было проведено тестирование службы Windows, а также нагрузочное тестирование сервера.

# Заключение

В результате выполнения дипломной работы была разработана серверная часть приложения для учета времени работы компьютерной техники и служба Windows для сбора данных о времени входа и выхода пользователей из ОС. Также в ходе выполнения работы были изучены особенности разработки серверной части приложения с использованием платформы NodeJS, процесса авторизации пользователей в операционной системе и разработки служб Windows на языке программирования C#.

Серверная часть приложения выполняет следующие функции: записывает данные о входе и выходе пользователей в базу данных, предоставляет статистику о времени и периодах работы компьютеров. Для полученной статистики добавлены фильтры по определенным компьютерам, по определенной операционной системе и по определенному периоду. Добавлена сортировка по всем полям в любом направление (по возрастанию или по убыванию) и пагинация получаемых данных. Также серверная часть обеспечивает авторизацию и аутентификацию пользователей в приложении. В приложении предусмотрено 2 роли: обычный пользователь может просматривать статистику о времени и периодах работы компьютеров, администратор может добавлять пользователей, просматривать список логов и редактировать список компьютеров.

Служба Windows имеет следующий функционал: в фоновом режиме отслеживает вход и выход пользователей из операционной системы, получает эти данные и отправляет их на сервер.

Данное приложение может быть полезным для различных учреждений образования, так как позволяет получить информацию для оптимизации расписания занятий в аудиториях с компьютерной техникой. Это позволит использовать компьютерную технику более эффективно, уменьшая время простоя и улучшая общую производительность в связи с равномерным распределением нагрузки на компьютеры.

Все поставленные в начале работы задачи были решены. Основываясь на полученных результатах, можно считать, что цель дипломной работы была достигнута.

# Список использованных источников

1. Михальченко С. В. Анализ источников событий и определение наиболее значимых событий для мониторинга информационной безопасности / С. В. Михальченко // Международный студенческий научный вестник. – 2018. – №. 2. – С. 63-63.
2. Знакомство с приложениями служб Windows – URL: <https://learn.microsoft.com/ru-ru/dotnet/framework/windows-services/introduction> -to-windows-service-applications (дата доступа 26.09.2023)
3. NodeJS документация – URL: <https://nodejsdev.ru/guides/webdraftt/> (дата доступа: 03.11.2023)
4. Basumatary, B. Benefits and Challenges of Using NodeJS / B. Basumatary, N. Agnihotri // International Journal of Innovative Research in Computer Science & Technology. – 2022. – T. 10. – №. 3. – P. 67-70.
5. NestJS документация – URL: <https://ru-nestjs-docs.netlify.app/> (дата доступа 03.11.2023)
6. Беликова Н. В. Использование паттерна проектирования Model-ViewController при разработке web-приложений / Н.В. Беликова, М.Г. Галич // Объектные системы. – 2011. – №. 1 (3). – С. 99-101.
7. Neumann F. Developing a content management system with drag and drop functionality / F. Neumann – 2022 – 38 p.
8. СУБД PostgreSQL: почему её стоит выбрать для работы с данными и как установить – URL: <https://practicum.yandex.ru/blog/chto-takoe-subd-postgresql/> (дата доступа 03.11.2023)
9. Практическое руководство. Создание служб Windows – URL: <https://learn.microsoft.com/ru-ru/dotnet/framework/windows-services/how-to-creat> e-windows-services (дата доступа: 26.09.2023)
10. Audit logon events – URL: <https://learn.microsoft.com/ru-ru/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/basic-audi> t-logon-events (date of access: 27.09.2023)
11. 4624(S): An account was successfully logged on. – URL: [https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/se curity/threat-protection/auditing/event-4624](https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/se%20curity/threat-protection/auditing/event-4624) (date of access: 27.09.2023)
12. Авторизация и аутентификация пользователя при помощи JWT – URL: <https://timeweb.com/ru/community/articles/avtorizaciya-i-autentifikaciya-polzovatelya-pri-pomoshchi-jwt-chast-1-chto-takoe-jwt-i-kak-ego-sozdat> (дата доступа 24.11.2023)
13. Pant P. et al. Authentication and authorization in modern web apps for data security using Nodejs and role of dark web / P. Pant // Procedia Computer Science. – 2022. – Т. 215. – P. 781-790.
14. Золотухина Е. Б. Обзор методов тестирования программного обеспечения / Е. Б. Золотухина, Е. А. Макарова, А. А. Беляков // Аллея науки. – 2018. – Т. 4. – №. 6. – С. 10-18.
15. Как использовать JMeter для нагрузочного тестирования и тестирования производительности – URL: https://habr.com/ru/companies/otus/articles/746504/ (дата доступа 26.03.2024)

# ПРИЛОЖЕНИЕ А

**Содержание электронного носителя**

На электронном носителе расположены следующие файлы и директории:

* Проект auth-log-service-windows с исходными файлами службы Windows.
* Проект auth-log-backend с исходными файлами серверной части приложения.
* Файл «Презентация Царенко А. Д. 2024.pptx» с презентацией к дипломной работе.
* Файл «Диплом Царенко А. Д. 2024.docx» с пояснительной запиской к дипломной работе.