

**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,
СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)**

Санкт-Петербургский колледж телекоммуникаций им. Э.Т. Кренкеля

Отделение *Информационных технологий*

Цикловая комиссия *Информатики и программирования в компьютерных системах*

КУРСОВОЕ ПРОЕКТИРОВАНИЕ

Тема: **Разработка системы для формирования политик
безопасности на предприятии**

Вид курсового проектирования *курсовой проект*

Специальность *09.02.03 «Программирование в компьютерных системах»*

Студент _____

М.Е.Сосновский
« » _____ 2023 г.

Руководитель _____

Н.В. Кривоносова
« » _____ 2023 г.

Оценка _____
(подпись)

Санкт-Петербург
2023

**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,
СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)**

Санкт-Петербургский колледж телекоммуникаций им. Э.Т. Кренкеля

Отделение *Информационных технологий и управления в телекоммуникациях*

Цикловая комиссия *Информатики и программирования в компьютерных системах*

**ЗАДАНИЕ
на курсовое проектирование**

Студент _____ Сосновский Максим Евгеньевич _____ Группа 511
(ФИО)

Руководитель _____ Кривоносова Наталья Викторовна _____
(фамилия, имя, отчество, должность, уч. степень и звание)

Профессиональный модуль _____

Вид работы _____ курсовой проект _____
(курсовой проект, курсовая работа)

Тема Разработка системы для формирования политик безопасности на предприятии

Исходные данные: разработать систему для внедрения политик безопасности
(технические требования)

Содержание работы: проведение анализа проблемы, проектирование системы,
разработка системы, тестирование системы _____
(анализ состояния проблемы, проведение исследований, разработка, расчеты параметров, экономическое обоснование и др.)

Вид отчетных материалов, представляемых в ПЦК: пояснительная записка, исходные
коды программного продукта
(пояснительная записка, перечень, графического материала, отчет о НИР, технический проект, образцы и др.)

(подпись студента)

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ	4
ВВЕДЕНИЕ.....	5
1. ПОСТАНОВКА ЗАДАЧИ.....	6
1.1. Обзор аналогов	6
1.2. SWOT-анализ.....	6
1.3 PEST-анализ.....	7
1.4 Формирование требований к программному продукту	8
1.4.1 Бизнес-требования	8
1.4.2 Пользовательские требования	9
1.4.3 Функциональные требования	11
1.4.4 Нефункциональные требования	12
1.4.5 Ограничения	12
1.4.6 Требования к интерфейсам	14
1.4.7 Требования к данным	14
1.5 Программные средства разработки	15
1.6 Аппаратные средства разработки	16
2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ	17
2.1. Архитектура системы	17
2.2. Моделирование основных сценариев системы	18
2.3. Проектирование графического интерфейса пользователя.....	22
2.4. Проектирование и разработка модели данных	24
3. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА.....	26
4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ	46
4.1. История изменений.....	46
4.2. Терминология	47
4.3. Стратегия тестирования.....	47
4.4. Определение объектов тестирования	48
4.5. Архитектура тестируемой системы	49
4.6. Описание процесса тестирования	49
ЗАКЛЮЧЕНИЕ	51
СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ.....	52
ПРИЛОЖЕНИЕ 1. ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ	53

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

РФ – Российская Федерация;
ФЗ – Федеральный закон;
SWOT – Strengths (сильные стороны), Weaknesses (слабые стороны), Opportunities (возможности), Threats (угрозы);
PEST – Political (политические факторы), Economic (экономические факторы), Social (социальные факторы), Technological (технологические факторы);
ПО – программное обеспечение;
DTO – Data Transfer Object (объект передачи данных)
DOM – document object model (объектная модель документа);
IDE – integrated development environment (интегрированная среда разработки);
ЦП – центральный процессор;
ОЗУ – оперативно-запоминающее устройство;
ОС – операционная система;
API – Application Programming Interface;
HTTP – HyperText Transfer Protocol (протокол передачи гипертекста);
HTTPS – HyperText Transfer Protocol Secure (безопасный протокол передачи гипертекста);
SQL – Structured Query Language (структурированный язык запросов);
XSS – Cross Site Scripting (межсайтовый скриптинг);
CSP – Content Security Policy (политика защиты содержимого);
GUI – Graphical User Interface (графический интерфейс пользователя);
IDEF – Integrated Definition (Интегрированное определение);
UML – Unified Modeling Language (Унифицированный язык моделирования);
MVC – Model-View-Controller (Модель-Представление-Контроллер)

ВВЕДЕНИЕ

В современном информационном обществе, где технологические трансформации играют ключевую роль в повседневной жизни и деятельности предприятий, вопросы кибербезопасности выходят на первый план. Неотъемлемой частью этой реальности становится необходимость обеспечения безопасности информационных ресурсов. В этом контексте компания "SecureGuard Solutions" осознает значимость обеспечения надежности и безопасности данных, что мотивирует нас рассмотреть создание системы, направленной на мониторинг и обеспечение безопасности клиентских машин.

Актуальность предложенного проекта ощущается болезненно в условиях постоянно меняющейся киберугрозы. Утечки конфиденциальных данных, кибератаки и другие события, подчеркивают насущную потребность в эффективных решениях, способных адаптироваться к новым угрозам и обеспечивать устойчивость информационных систем.

Цель настоящего проекта заключается в разработке системы, предназначенной для мониторинга и обеспечения безопасности данных на клиентских машинах. Для этого были поставлены следующие задачи:

- 1) Провести анализ предметной области
- 2) Провести анализ аналогичных систем, если таковые существуют
- 3) Разработать функциональные требования к программному продукту
- 4) Разработать нефункциональные требования к программному продукту
- 5) Рассмотреть возможности интеграции модуля с уже существующими инструментами
- 6) Разработать требования к архитектуре программного продукта
- 7) Определить как, кем и в каких случаях будет использоваться система
- 8) Определить схему тестирования готового программного продукта
- 9) Разработать программный продукт

Таким образом, практической значимостью разработки системы является обеспечение высокого уровня кибербезопасности информационных ресурсов пользователей. Этот инструмент предоставляет надежный механизм мониторинга и реагирования на потенциальные угрозы, связанные с аппаратными компонентами и историей браузера. Разработанная система станет неотъемлемым решением для организаций, стремящихся обеспечить защиту конфиденциальных данных, и для администраторов, желающих эффективно управлять безопасностью информационных систем. Такой инструмент позволит укрепить киберсреду компаний и поддерживать высокий стандарт безопасности, что особенно важно в условиях постоянно меняющихся киберугроз.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Обзор аналогов

На рынке средств мониторинга и обеспечения безопасности информационных ресурсов существует ряд инновационных продуктов, предназначенных для решения аналогичных задач. Рассмотрим несколько ключевых аналогов системы "PolicyEnforcer" и их особенности.

1. SecurityGuard Pro:

SecurityGuard Pro представляет собой комплексное решение для обеспечения кибербезопасности организаций. Этот продукт обладает широким функционалом, включая мониторинг активности пользователей, анализ сетевого трафика и обнаружение угроз. Однако, отсутствует интегрированная система для мониторинга аппаратных компонентов клиентских машин.

2. SecureWatch Solutions:

SecureWatch Solutions специализируется на анализе событий безопасности и предоставлении реального времени для оперативного реагирования. Этот продукт хорошо подходит для мгновенного обнаружения угроз, но не обеспечивает полноценного мониторинга состояния аппаратных компонентов.

3. GuardianSuite:

GuardianSuite предоставляет обширные средства аудита и мониторинга безопасности, включая анализ логов и сетевого трафика. Однако, система ограничивается в возможности анализа истории браузера, что является ключевым компонентом для полноценного мониторинга.

4. SafetyNet Advanced:

SafetyNet Advanced выделяется своей сфокусированностью на мониторинге аппаратных компонентов. Однако, отсутствует централизованный хаб для оперативного управления и взаимодействия с клиентскими машинами.

В контексте сравнения, разрабатываемая система стремится объединить в себе все перечисленные возможности, предоставляя комплексный и интегрированный подход к мониторингу и обеспечению безопасности информационных ресурсов. Она выделяется централизованным хабом, а также широким спектром функциональности, включающим как мониторинг аппаратных компонентов, так и анализ истории браузера.

1.2. SWOT-анализ

SWOT-анализ является эффективным инструментом для выявления сильных и слабых сторон, а также возможностей и угроз, с которыми может столкнуться система "PolicyEnforcer". Этот анализ позволяет получить глубокое понимание внутренней и внешней среды проекта, определить стратегические направления развития и разработать меры по улучшению.

Сильные стороны (Strengths):

- Интегрированный мониторинг: разрабатываемая система обеспечивает комплексный мониторинг как аппаратных компонентов, так и истории браузера, что предоставляет полную картину безопасности информационных ресурсов.
- Система реального времени: Система работает в режиме реального времени, что позволяет оперативно выявлять и реагировать на потенциальные угрозы.

- **Масштабируемость:** Система разработана с учетом возможности масштабирования, что обеспечивает эффективную работу в условиях увеличения числа клиентов.

Слабые стороны (Weaknesses):

- **Сложность внедрения:** Внедрение системы может потребовать определенного времени и ресурсов, особенно в случае интеграции с существующей инфраструктурой.
- **Необходимость обучения:** Использование системы требует обучения персонала, особенно администраторов, что может вызвать временные сложности в первоначальной эксплуатации.

Возможности (Opportunities):

- **Растущий рынок кибербезопасности:** С увеличением угроз в сфере кибербезопасности, система имеет возможность привлечь внимание новых клиентов, ищущих надежные решения.
- **Развитие партнерских отношений:** Установка партнерских отношений с другими поставщиками кибербезопасности может расширить функциональность и конкурентоспособность продукта.

Угрозы (Threats):

- **Конкуренция на рынке:** Наличие конкурентов с подобными продуктами может создать конкуренцию за клиентскую базу и ресурсы.
- **Технологические изменения:** Быстро меняющаяся технологическая среда может потребовать постоянных обновлений системы для соответствия новым стандартам и требованиям безопасности.

SWOT-анализ позволяет выявить ключевые факторы, которые могут повлиять на успешность системы. Понимание собственных сильных и слабых сторон, а также внешних возможностей и угроз, является важным этапом стратегического планирования для обеспечения долгосрочной эффективности продукта.

1.3 PEST-анализ

Проведем анализ внешней среды, оценив воздействие политических, экономических, социокультурных и технологических факторов на разрабатываемую систему.

Политические факторы:

- **Законодательство о кибербезопасности:** Введение строгих законов и нормативов в области кибербезопасности может повысить потребность в продуктах, таких как разрабатываемая система, для соблюдения требований и обеспечения защиты информации.
- **Глобальные регуляторы:** Работа с мировыми регуляторами по стандартам безопасности может стать ключевым элементом в успешной международной экспансии системы.

Экономические факторы:

- **Финансовая устойчивость клиентов:** Экономические колебания могут повлиять на решение клиентов инвестировать в кибербезопасность, включая системы мониторинга.
- **Стоимость внедрения:** Доступность и стоимость внедрения могут быть определяющими факторами при выборе системы безопасности для компаний с различными бюджетами.

Социокультурные факторы:

- Осведомленность пользователей: Рост уровня осведомленности пользователей о киберопасности может увеличить спрос на продукты, направленные на обеспечение безопасности данных.
- Ценности в области конфиденциальности: Растущее внимание к защите конфиденциальности данных создает повышенный спрос на продукты, обеспечивающие цифровую безопасность.

Технологические факторы:

Быстрый темп технологического развития: Готовность к интеграции с новыми технологиями, такими как искусственный интеллект и аналитика больших данных, может определить успех разрабатываемой системы.

Конкуренция на рынке технологий: Динамичная конкуренция в технологическом секторе требует постоянного развития и обновления системы для поддержания конкурентоспособности на рынке.

PEST-анализ обеспечивает глубокое понимание внешней среды, помогая лучше адаптировать стратегии развития системы к внешним изменениям и вызовам.

1.4 Формирование требований к программному продукту

Процесс разработки любого программного продукта начинается не с написания кода, а с четкого определения его требований. Правильное формирование требований — это ключевой этап, от которого зависит успешность всего проекта, так как именно на основе этих требований будут построены архитектура решения, функционал и интерфейс продукта.

Формирование требований — это не только технический процесс, но и в значительной степени коммуникативный, включающий в себя работу с заинтересованными сторонами, проведение опросов, исследование рынка и анализ конкурентов. В результате такой работы создается документ, который будет служить основой для дальнейшей разработки и внедрения программного продукта. Данный раздел позволит читателю ознакомиться с процессом формирования требований, методами их сбора, анализа и верификации, а также понять, какие критерии и параметры были установлены для обеспечения качества и соответствия разрабатываемого решения задачам и потребностям пользователей.

1.4.1 Бизнес-требования

Бизнес-требования определяют, какие функции и характеристики продукта должны быть реализованы, чтобы удовлетворить бизнес-потребности заказчика и конечного пользователя.

Контекстом для создания системы стала необходимость в удобном сбалансированном продукте, позволяющем пользователю фиксировать свои задачи, распределять их, делать записи в заметках и структурировать необходимую ему информацию с целью оптимизации жизненных процессов.

В ходе разработки инструмента были выделены следующие бизнес-требования:

- 1) Производительность: приложение должно обеспечивать быструю и эффективную работу. Это достигается за счет оптимизации кода, использования новых технологий, а также правильного выбора серверных решений. Высокая производительность обеспечивает стабильную работу приложения в условиях работы в корпоративных масштабах.

- 2) **Безопасность:** безопасность является первостепенной задачей. Использование хэширования для защиты конфиденциальных данных, таких как пароли, и применение протокола HTTPS для защиты передачи данных между клиентом и сервером, помогает обеспечить безопасность данных компании.
- 3) **Масштабируемость:** важным фактором является способность системы политик безопасности адаптироваться к растущему числу персонала, что влечет увеличение объема одновременно обрабатываемых и хранимых данных.
- 4) **Удобство использования:** для обеспечения быстрой адаптации персонала и уменьшения задержек при работе с системой человеческого персонала необходимо сделать интерфейс интуитивно понятным и предлагающим достаточный инструментарий для администрирующего персонала.
- 5) **Надежность:** надежная работа без сбоев и ошибок является залогом стабильности компании. Тщательное тестирование, включая юнит-тесты и интеграционные тесты, помогает обнаружить и устранить потенциальные проблемы.

1.4.2 Пользовательские требования

Основой для разработки инструмента по управлению личной эффективностью является глубокое понимание и четкое определение пользовательских требований, которые должны быть четко сформулированы. В данном контексте, ключевым аспектом является анализ действий, которые будут выполняться пользователями в сервиса, что позволит обеспечить их эффективное взаимодействие и достичь поставленных целей. Основным действующим лицом является авторизированный пользователь.

Для упрощения выявления пользовательских требований необходимо выявить варианты использования инструмента, а также выявить альтернативные потоки действия при различных состояниях системы.

Вариант использования № 1

Название прецедента: Запрос на сбор истории браузера

Действующее лицо: Администратор

Цель: Создание запросов к подключенным клиентам на сбор истории посещений сайтов

Предусловия: Администратор вошел в систему под учетной записью с правами администратора

Главная последовательность:

- 1) Администратор нажимает на кнопку «Админ»
- 2) Система открывает вкладку со списком всех зарегистрированных клиентов.
- 3) Администратор нажимает на кнопку «Сбор истории»
- 4) Система оповещает администратора о результате создания запросов.
- 5) Администратор закрывает уведомление.

Вариант использования № 2

Название прецедента: Запрос на сбор данных об аппаратных компонентах

Действующее лицо: Администратор

Цель: Создание запросов к подключенным клиентам на сбор информации об аппаратных компонентах вычислительных машин.

Предусловия: Администратор вошел в систему под учетной записью с правами администратора.

Главная последовательность:

- 1) Администратор нажимает на кнопку «Админ».

- 2) Система открывает вкладку со списком всех зарегистрированных клиентов.
- 3) Администратор нажимает на кнопку «Сбор данных о машинах»
- 4) Система оповещает администратора о результате создания запросов.
- 5) Администратор закрывает уведомление.

Вариант использования № 3

Название прецедента: Просмотр истории браузера пользователя

Действующее лицо: Администратор

Цель: Ознакомиться с историей посещения сайтов пользователя

Предусловия: Запрос на сбор истории браузера был создан после регистрации клиента в системе.

Главная последовательность:

- 1) Администратор нажимает на кнопку «Админ».
- 2) Система открывает вкладку со списком всех зарегистрированных клиентов.
- 3) Администратор находит нужного пользователя в списке зарегистрированных клиентов.
- 4) Администратор нажимает на кнопку «История браузера» в строке, соответствующей требуемому пользователю.
- 5) Система открывает вкладку с историей посещения сайтов пользователем.

Вариант использования № 4

Название прецедента: Просмотр информации об аппаратных компонентах клиента

Действующее лицо: Администратор

Цель: Ознакомиться с информацией о состоянии комплектующих компьютера клиента.

Предусловия: Запрос на сбор информации о компонентах был запущен после регистрации клиента в системе.

Главная последовательность:

- 1) Администратор нажимает на кнопку «Админ».
- 2) Система открывает вкладку со списком всех зарегистрированных клиентов.
- 3) Администратор находит нужного пользователя в списке зарегистрированных клиентов.
- 4) Администратор нажимает на кнопку «Аппаратные компоненты» в строке, соответствующей требуемому пользователю.
- 5) Система открывает вкладку с информацией о компонентах ЭВМ клиента на момент замера по запросу сервера.

После выявления пользовательских требований необходимо создать диаграмму вариантов использования, которая в удобном виде позволит ознакомиться с вариантами использования сервиса по управлению личной эффективностью (рис. 1.1)

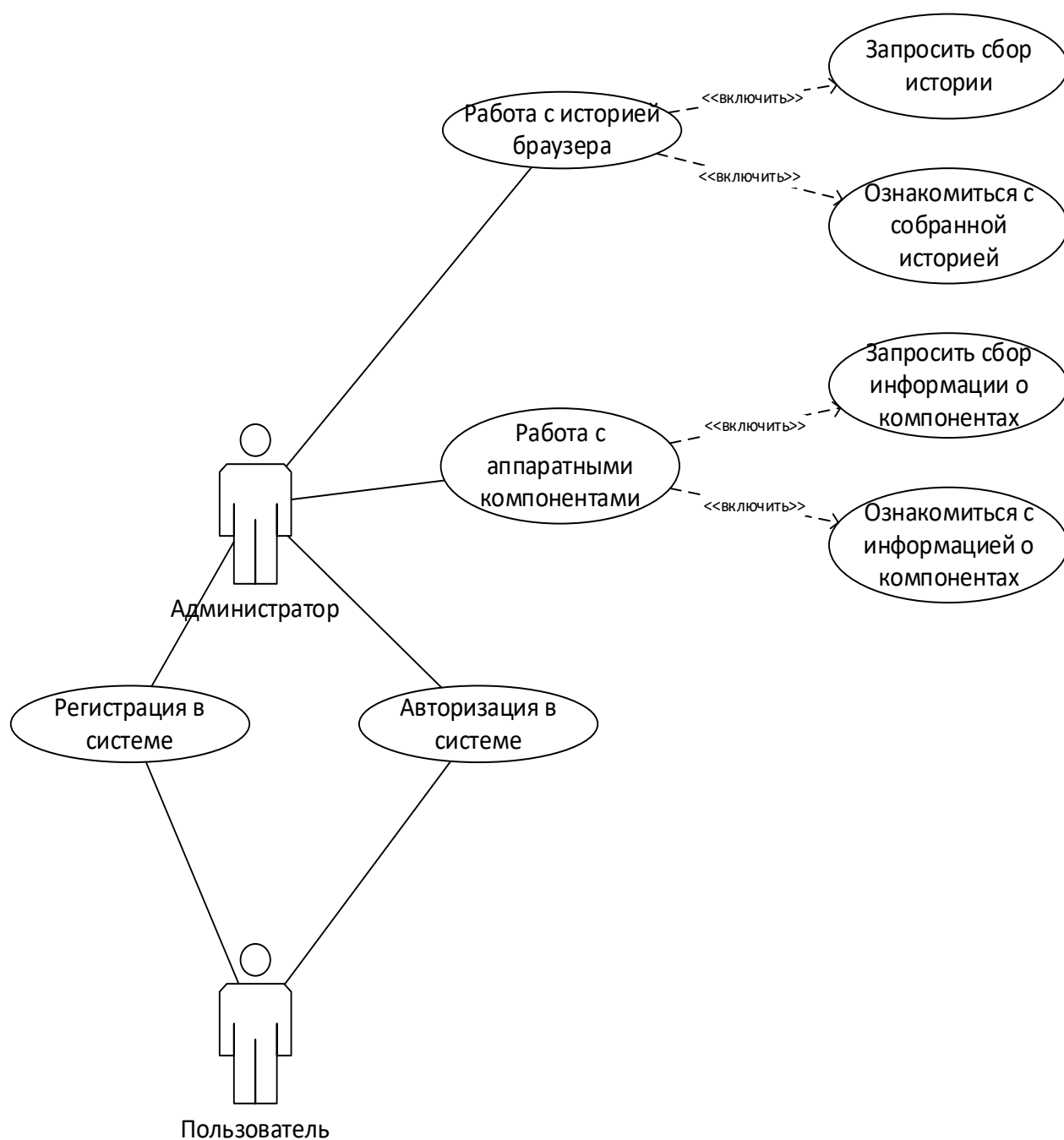


Рисунок 1.1 Диаграмма сценариев использования системы для формирования политик безопасности на предприятии

В заключение раздела о пользовательских требованиях можно подчеркнуть, что детальный анализ и понимание потребностей всех участников процесса работы с сервисом является фундаментальными для успешной разработки и внедрения инструмента.

1.4.3 Функциональные требования

Функциональные требования играют важную роль в процесс разработки программного обеспечения, так как они определяют спецификации, которые должны быть реализованы, чтобы система отвечала бизнес-потребностям пользователей. Эти требования обеспечивают основу для разработки модуля и дизайна. Операции оформлены в виде

удобной навигации и четком разделении функций для лучшего освоения сервиса управляющим персоналом.

После прохождения процесса установки система за исключением модуля администратора способна существовать абсолютно автономно, что убирает необходимость создавать пользовательский интерфейс для модулей клиента и сервера и исключает человеческий фактор в их работе. Для использования приложения и оценки результатов его работы в рамках компании разработан веб-клиент для администрирующего персонала.

Основной функционал администратора заключается во взаимодействии с историей посещения сайтов и информацией об аппаратных компонентах вычислительных машин, на которые установлен клиентский модуль системы. Базовый функционал администратора представляет собой следующие пункты:

- 1) Авторизация в системе
- 2) Создание запросов на сбор истории посещения сайтов клиентами
- 3) Создание запросов на сбор информации об аппаратных компонентах клиентов
- 4) Просмотр истории посещения сайтов конкретных клиентов
- 5) Просмотр информации об аппаратных компонентах конкретных клиентов

Функционал будет пополняться по мере увеличения нужд компании в сфере безопасности.

1.4.4 Нефункциональные требования

В рамках данной работы особое внимание уделяется не только функциональным аспектам создаваемого продукта, но и широкому спектру нефункциональных требований. Эти требования играют ключевую роль в обеспечении качества, эффективности и удобства использования разрабатываемой системы.

В ходе разработки инструмента были выделены следующие нефункциональные требования:

- 1) Легкость и простота использования. Если пользователю необходимо слишком много времени для освоения инструмента, значит, данный продукт недостаточно проработан и требует изменений, так как мы ориентируемся на эффективность внедрения продукта в условия работы на корпоративном уровне.
- 2) Ограничения дизайна и реализации. Они касаются выбора возможности разработки внешнего вида и структуры продукта. В данном случае ограничением дизайна и реализации выступает минималистичный стиль с акцентом на основные элементы управления

С помощью комплексного подхода к соблюдению этих дополнительных требований, можно разработать веб-приложение, которое не только успешно выполняет свои основные функции, но и обладает высоким уровнем производительности. Это, в свою очередь, предполагает положительный опыт пользователей и внушает доверие к системе.

1.4.5 Ограничения

В ходе разработки любого программного продукта неизбежно возникают различные ограничения, которые могут существенно воздействовать на весь процесс – от выбора технологий и функциональности до успешного внедрения проекта. Учет и понимание этих ограничений с самого начала проекта позволяет установить реалистичные ожидания, эффективно планировать ресурсы и разрабатывать стратегии для минимизации их воздействия.

В данном разделе мы рассмотрим ключевые виды ограничений, с которыми сталкиваемся при разработке модуля системы деловых переговоров: технические, временные, финансовые, функциональные и ограничения в области безопасности.

Проанализировав каждое из этих ограничений, мы стремимся выявить их потенциальное влияние на весь проект. Осознание данных ограничений и эффективное управление ими являются ключевыми аспектами обеспечения успешной и беспрепятственной разработки инструмента.

Для разработки в рамках данной работы предусмотрены определенные технические ограничения. При разработке серверной части – “PolicyEnforcer.ServerCore” требуется разработка WEB-API, предлагающая методы для авторизации и административной работы, а также центры для возможности двухстороннего общения сервера с клиентами.

Клиентский модуль – “PolicyEnforcer.Service” должен быть выполнен в виде сервиса, предназначенного для автономной работы. Сервис должен подключаться к центру общения на сервере и быть готовым собрать и отправить данные при поступлении запросов с сервера.

Модуль администратора был реализован в виде веб-клиента, предоставляя базовый функционал для контроля работы сервера.

В дополнение к техническим ограничениям необходимо учесть требование кроссбраузерности модуля администратора. Гарантировать стабильную и однородную работу модуля во всех популярных веб-браузерах становится приоритетной задачей. Это предполагает, что внешний вид и функциональность системы должны оставаться постоянными, независимо от того, через какой браузер осуществляется доступ. Это техническое ограничение подразумевает проведение тщательного тестирования и использование специализированных техник и инструментов веб-разработки для обеспечения совместимости и однородного визуального представления в различных браузерах.

При разработке сервиса также должны быть установлены временные ограничения, которые влияют на все этапы проекта, начиная от исследования и заканчивая выпуском и поддержкой продукта. После согласования сроков, обозначены следующие сроки:

- 1) Этап исследования и анализ требований – с 08.08.2023 до 08.09.2023
- 2) Проектирование (включается в себя разработку архитектуры и создание дизайна интерфейса) – с 09.09.2023 до 09.10.2023
- 3) Разработка – с 10.10.2023 до 25.11.2023
- 4) Этап тестирования – с 25.11.2023 до 15.12.2023
- 5) Сдача проекта – 23.12.2023

Также при согласовании сроков разработки было выделено 5 резервных дней в случае непредвиденных трудностей. Таким образом, общий срок разработки сервиса для управления личной эффективностью составляет чуть больше 4 месяцев.

Финансовым ограничением выступает использование бесплатных ресурсов при проектировании и разработке продукта. Начиная от программной части, используемой для создания сервиса, заканчивая оформлением пользовательского интерфейса.

Такое ограничение накладывает определенные рамки на выбор технологий и ресурсов. Важно отметить, что их использование никак не должно ухудшать качество конечного продукта, что предполагает грамотное планирование процесса разработки с учетом потребностей проекта.

Переходя к функциональным ограничениям для разрабатываемой системы необходимо учитывать требования к автономности, производительности и поддержке большого числа клиентов, которые включают в себя:

- Обеспечение высокой эффективности обработки запросов – модуль должен оперативно обрабатывать поступающие запросы, минимизируя временные задержки и обеспечивая мгновенный отклик на пользовательские действия.
- Обеспечение автономной работы – клиентский модуль должен работать без человеческого вмешательства, что требует высокой отказоустойчивости, способности автоматического запуска и самоидентификации, и полного покрытия тестами.

- Уменьшение времени загрузки страниц – необходимо существенно сократить время загрузки страниц за счет оптимизации как серверной, так и клиентской частей приложения. Это включает в себя сжатие ресурсов, эффективное кэширование и минимизацию HTTP-запросов.

Соблюдение функциональных ограничений является фундаментальным для разработки. Последними ограничениями при разработке модуля являются ограничения безопасности. К таким ограничениям относится следующее:

- 1) Применение HTTPS для шифрования трафика между клиентом и сервером, что предотвращает возможность перехвата и чтения информации третьими лицами, включая личные данные пользователей и конфиденциальную информацию
- 2) Шифрование конфиденциальных данных во всех модулях системы предотвращает возможность кражи критически важной информации.

Таким образом, каждое из вышеперечисленных ограничений вносит свой вклад в формирование качественного и надежного программного продукта, который будет отвечать всем требованиям и ожиданиям как разработчиков, так и конечных пользователей. Учет и управление этими ограничениями являются неотъемлемой частью процесса разработки.

1.4.6 Требования к интерфейсам

Одним из ключевых требований к сервису выступает требование к созданию условий, при которых каждый пользователь, который заходит в веб-приложение, мог сразу же воспользоваться и понять весь функционал. Также необходимо отследить единую стилистику всего проекта, используя типизированные цвета и стили, с целью лаконичности сервиса.

Исходя из минималистичного стиля оформления интерфейса, были выбраны следующие цвета:

- 1) # fafafa – светлый, для фона основных блоков
- 2) # 1e1e1e – темно-серый, для более маленьких блоков и блоков управления в пределах основных блоков
- 3) # 343434 – темно-серый, для оформления текста в пределах основных блоков
- 4) # ffffff – белый, для оформления текста в пределах дополнительных блоков и выделения элементов управления в пределах дополнительных блоков
- 5) # b2b2b2 – светло-серый, для подчеркивания выбранных элементов
- 6) # 2727fd – синий, для дополнительных элементов управления

Также при разработке необходимо учесть стиль текста, так чтобы он был прост и удобен для восприятия. Для проекта был выбран шрифт Nunito, при необходимости и иерархии текста использовались разная жирность и размер.

Следующим требованием к пользовательским интерфейсам является размещение основных кнопок страницы на видном месте, благодаря такой расстановки кнопок можно достигнуть удобства использования сервиса. Все кнопки в разрабатываемом модуле на каждой странице должны иметь одинаковый размер и единый стиль.

Так как модуль веб-клиента создается для использования в корпоративных условиях, требования к адаптивности были менее строгими, требуя только возможность работы на стандартных компьютерных экранах, использующихся в офисных условиях.

1.4.7 Требования к данным

В современном цифровом мире, где данные становятся ключевым активом в любой информационной системе, крайне важно обеспечить эффективное и безопасное управление данными. В контексте разработки данного проекта особое внимание уделяется организации, обработке и передаче данных. Эти аспекты играют важную роль в обеспечении надежности, производительности и безопасности системы, а также в создании удобного и функционального пользовательского интерфейса.

Необходимо сосредоточиться на следующих аспектах: формате данных, их хранении, обработке, передаче и защите. Каждый из этих аспектов включает в себя конкретные требования и спецификации, необходимые для эффективной работы системы и удовлетворения потребностей пользователей.

Конфиденциальные данные на стороне клиентов с момента их получения и до момента их отправки хранятся в виде защищенных типов данных, таких как `SecureString` для строк.

Касательно защиты данных на серверной части, стоит сказать, что для хранения данных на сервере используется СУБД Microsoft SQL Server. Данная СУБД предлагает: возможность значительно уменьшить сроки разработки, предлагая инструментарий для генерации программного кода EntityFramework, позволяя одной командой дублировать структуру и отношения внутри базы данных; большой выбор типов данных, имеющих эквивалент в языке программирования, что позволяет избавиться от необходимости их конвертации при чтении и записи в базу данных, увеличивая производительность и облегчая разработку модуля.

Для общения веб-клиента с сервером выделены модели DTO, что гарантирует невозможность получения потенциально критичных данных при выполнении запросов к серверу – отправляются только те данные, которые требуются для работы клиента.

Таким образом, внедрение этих требований и технологий в данный сервис позволит создать мощную и гибкую систему, способную эффективно обрабатывать, хранить и передавать данные. Это будет способствовать улучшению общей работы системы, повышая её производительность.

1.5 Программные средства разработки

Современное программное обеспечение часто создается в сложной и динамичной среде. При этом каждый инструмент или технология выбирается с учетом конкретных требований и задач проекта. Правильный выбор программных средств разработки может значительно ускорить процесс создания приложения, повысить его стабильность, упростить масштабирование и обеспечить высокое качество конечного продукта.

Для разработки серверного модуля системы контроля политик безопасности был выбран язык программирования C#, фреймворк .NET Core 7.0. WEB-API было решено выполнить используя ASP.NET.Core, а центр общения был выполнен с использованием технологии SignalR.

Для реализации модуля клиента также был выбран фреймворк .NET Core 7.0. Сбор данных об аппаратных компонентах будет осуществляться с помощью библиотеки LibreHardwareMonitorLib, а для получения истории браузеров будут производиться обращения к SQLite базам данных самых популярных браузеров.

Клиент администратора было решено сделать в виде веб-приложения, с использованием фреймворка React.JS и пакетного менеджера Yarn.

В качестве среды разработки используется Visual Studio 2022 Enterprise, предоставляющая обширный инструментарий, значительно облегчающий разработку данной системы.

1.6 Аппаратные средства разработки

Для разработки сервиса для управления личной эффективностью были использованы аппаратные средства со следующими характеристиками (Таблица 1):

Таблица 1

Аппаратные средства разработки и их характеристики

Аппаратное средство	Характеристика
Персональный компьютер	ОС – Windows 11 ЦП – AMD Ryzen 9 7900X ОЗУ – Kingston Fury Beast 5600 MHz 64 GB (2x32 GB) Видеокарта: NVIDIA GeForce RTX 4070 12 GB SSD: NVMe Samsung 980 PRO 1 T6 HDD: Seagate 5400 IronWolf 4 TB
Ноутбук	ОС – Windows 11 ЦП – AMD Ryzen 5 4600H ОЗУ – Generic Memory 16 GB Видеокарта: NVIDIA GeForce GTX 1650 4GB SSD: Integrated 512GB
Серверный компьютер	ОС – Windows Server 2022 ЦП – AMD Ryzen 3 2200G ОЗУ – Unknown Memory 8GB (1x8GB) GPU: Integrated SSD: NVMe Samsung 980 PRO 1 T6 HDD: Seagate 5400 IronWolf 4 TB

2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

2.1. Архитектура системы

Архитектурно система представляет собой клиент-серверную архитектуру (рис.2.1)

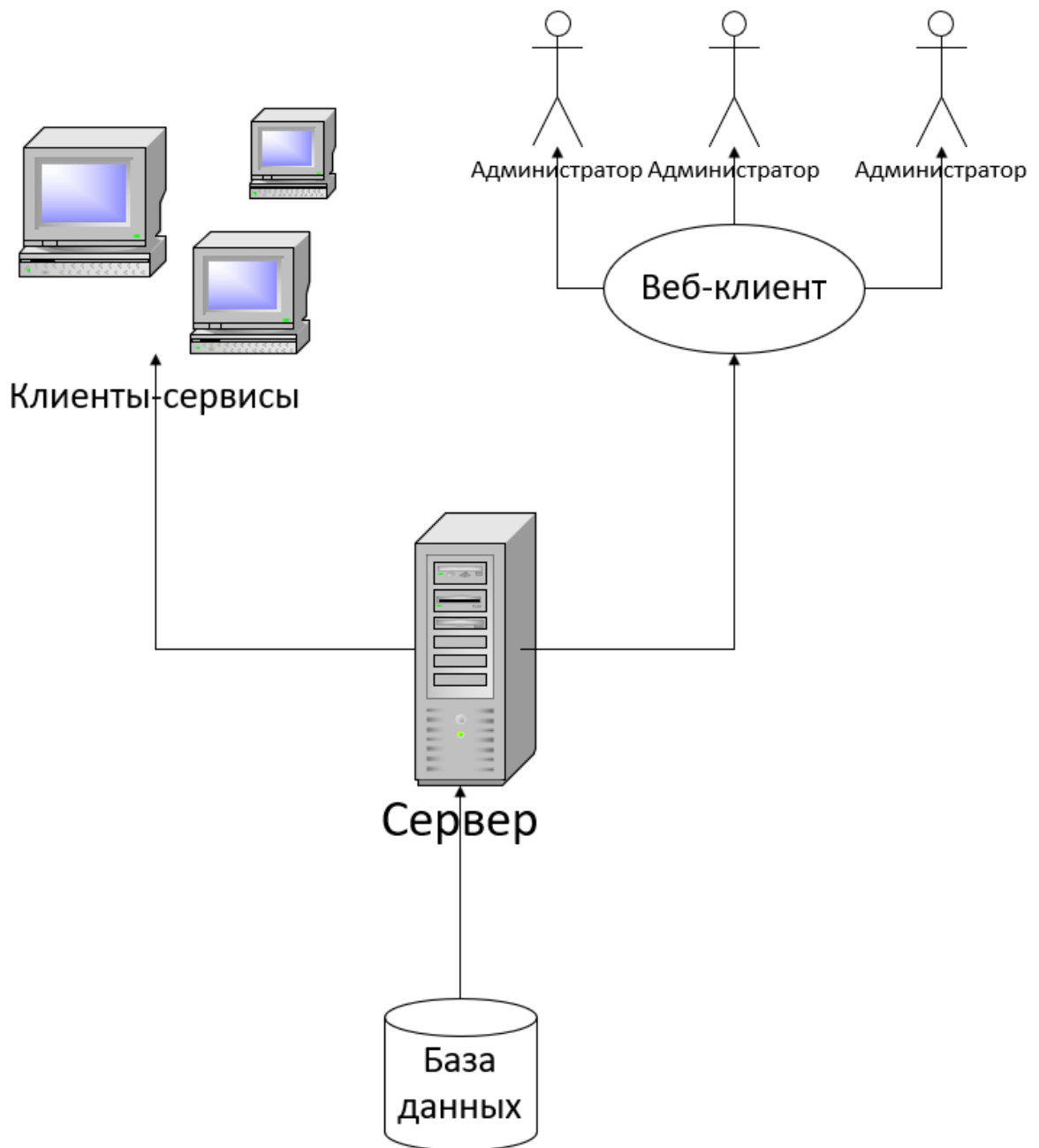


Рисунок 1.1. Архитектура системы

На данной диаграмме представлена упрощенная клиент-серверная архитектура для большего понимания, как и какие взаимодействия происходят на стороне сервера. В целом архитектура системы состоит из следующих частей:

- 1) Клиент-сервисы – это машины, на которых установлен и запущен клиентский модуль системы.
- 2) Сервер (Server): на сервере реализован модуль, отвечающий за обработку и сохранение данных.

- 3) Веб-клиент – модуль системы, производящий общение с сервером и исполняющий административные функции
- 4) База данных (Database): это набор структурированных данных, хранящихся и обрабатываемых с помощью SQL-запросов. База данных используется для эффективного хранения и обработки больших объемов информации в организованном виде.

Данная архитектура обеспечивает четкое разделение ответственности между клиентской и серверной частями, что упрощает поддержку и развитие системы, а также позволяет адаптировать ее под различные бизнес-требования и условия эксплуатации. Эта архитектура является фундаментом для создания эффективного и удобного в использовании сервиса.

2.2. Моделирование основных сценариев системы

В данном разделе мы сосредоточимся на моделировании основных сценариев использования сервиса. Этот процесс является ключевым этапом в проектировании системы, поскольку он позволяет визуализировать и анализировать различные аспекты ее работы, что обеспечивает более глубокое понимание функциональных и нефункциональных требований к программному продукту. Моделирование основных сценариев использования включает в себя применение двух ведущих методологий: IDEF и UML.

IDEF – методология, которая фокусируется на моделировании бизнес-процессов с использованием графических обозначений для описания функциональности, данных, объектов и процессов. Мы применим контекстные диаграммы IDEF до 3 уровня для описания общей концептуальной модели системы деловых переговоров.

UML являясь стандартным языком для моделирования программных систем, UML поможет нам создать диаграммы, описывающие структуру системы, ее поведение и взаимодействие между компонентами.

Важно отметить, что все модели и диаграммы, представленные в этом разделе, будут основаны на предварительно проведенном анализе пользовательских и бизнес-требований, а также учитывать аппаратные и программные ресурсы, необходимые для разработки и эксплуатации системы.

Для начала составим диаграмму IDEF0 для нашего модуля системы деловых переговоров, которая будет описывать процесс добавления содержимого заметки. (рис. 2.2.1)

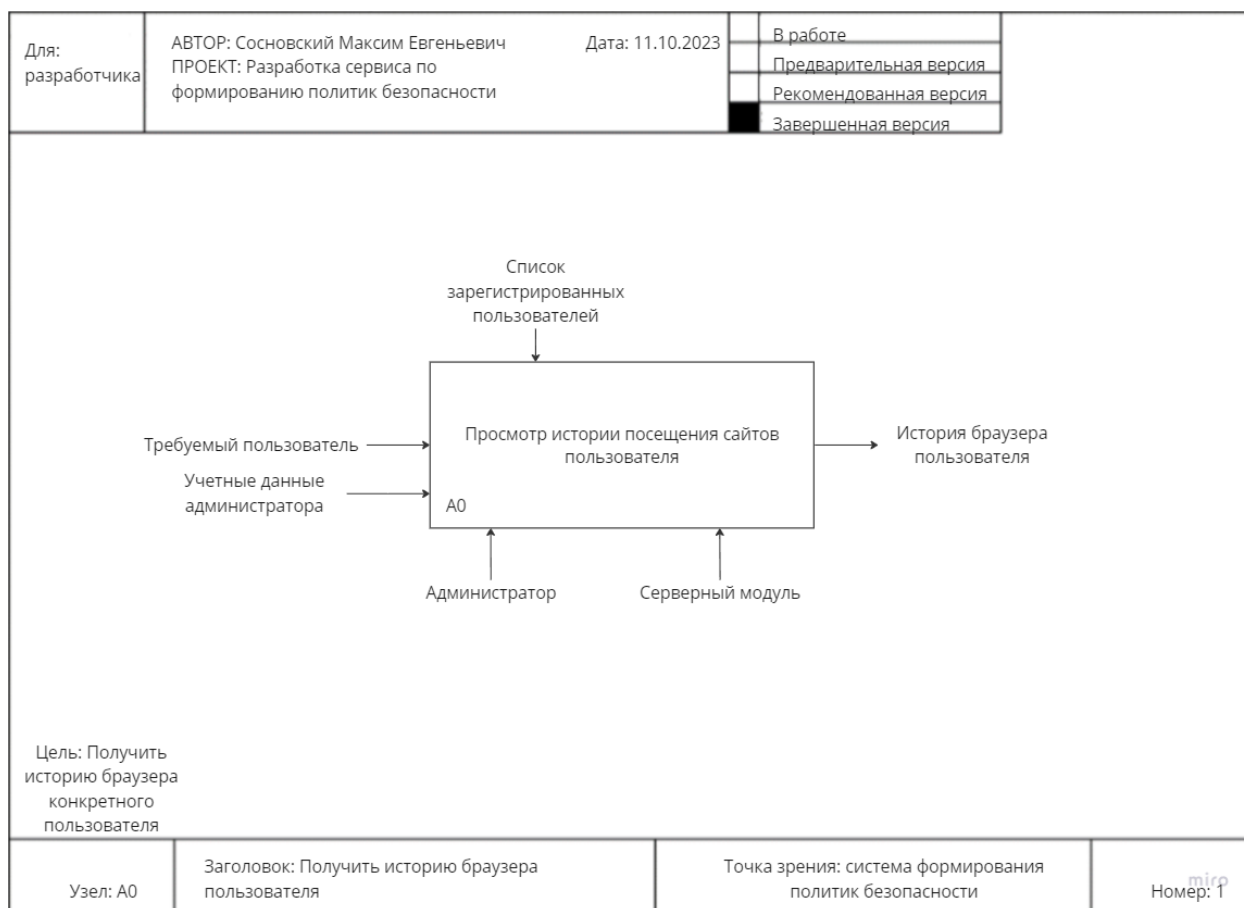


Рисунок 2.2.1. Диаграмма моделирования процесса “Просмотр истории посещения сайтов пользователя”

На данной диаграмме видно, что для данного в качестве элементов управления выступает список зарегистрированных в системе клиентов. Информацией для получения конечного результата в данном процессе является требуемый пользователь и учетные данные аккаунта с правами администратора. Результатом данного бизнес-процесса, то есть выходом, является список посещенных сайтов требуемым клиентом.

Далее необходимо углубиться в данный бизнес-процесс и разбить его на более маленькие процессы (рис. 2.2.2).

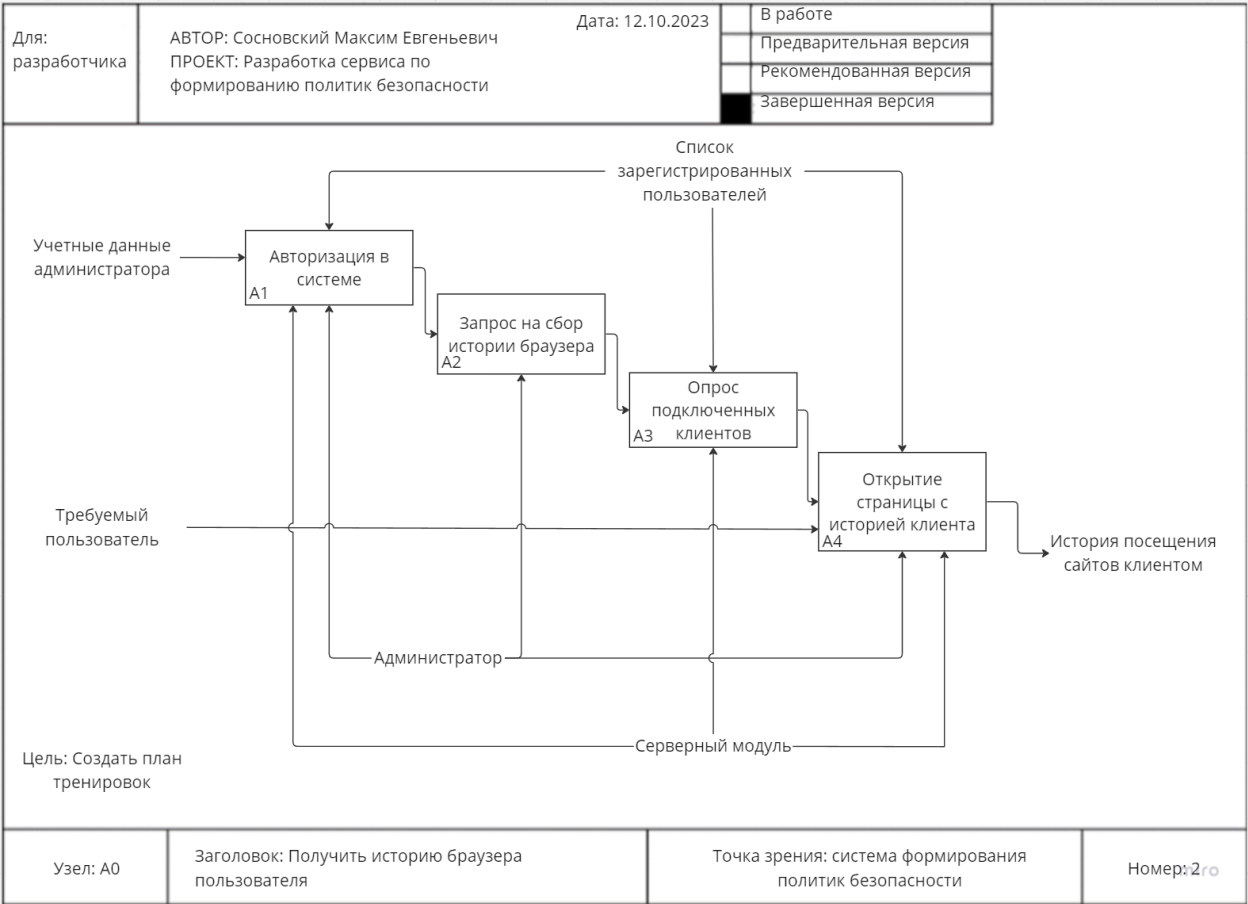


Рисунок 2.2.2 Диаграмма моделирования процесса “Создание плана тренировок”

Основная цель этой диаграммы — показать, как проходит процесс создания и добавления данных в пользовательский план тренировок. Это помогает понять и стандартизировать шаги, необходимые для успешного выполнения процесса. Из диаграммы видно, что сначала происходит переход на страницу создания плана, затем, добавление данных и упражнений, далее – сохранение. Результатом является личный план тренировок пользователя.

Далее будут представлены диаграммы последовательности для некоторых процессов в сервисе. Первой диаграммой последовательности является диаграмма процесса авторизации. (рис. 2.3)

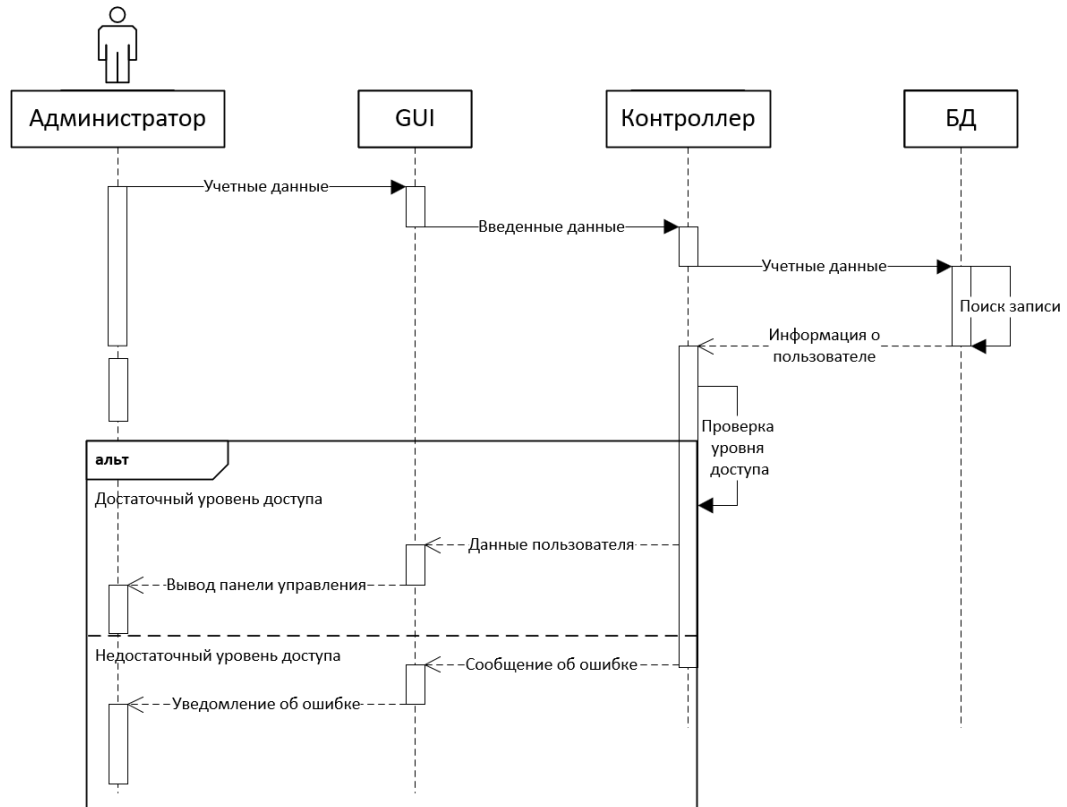


Рисунок 2.3. Диаграмма последовательности для процесса создания заметки

Следующим процессом для создания диаграммы последовательности является процесс создания запроса на сбор истории браузера (рис. 2.4)

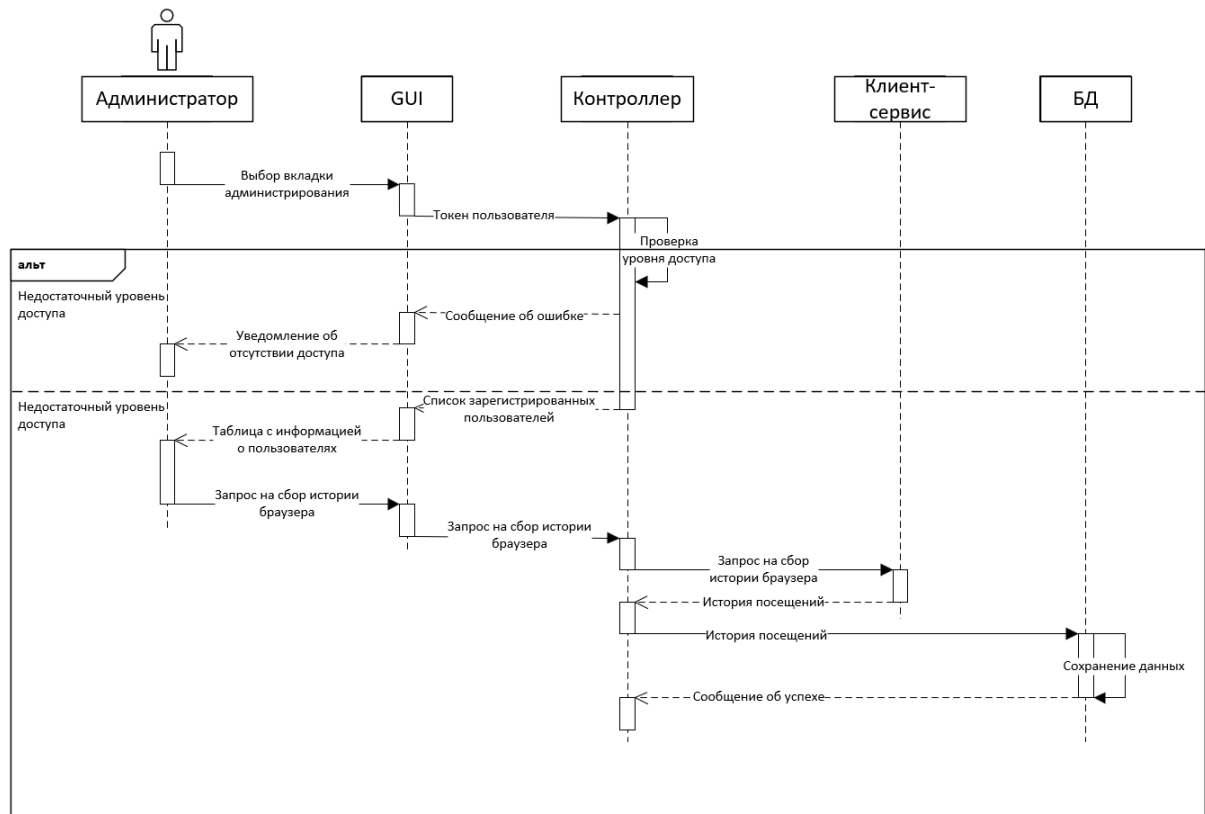


Рисунок 2.4 Диаграмма последовательности для процесса создания запроса на сбор истории посещений сайтов клиентом

Далее рассмотрим диаграмму деятельности сбора информации о технических характеристиках (рис. 2.5)

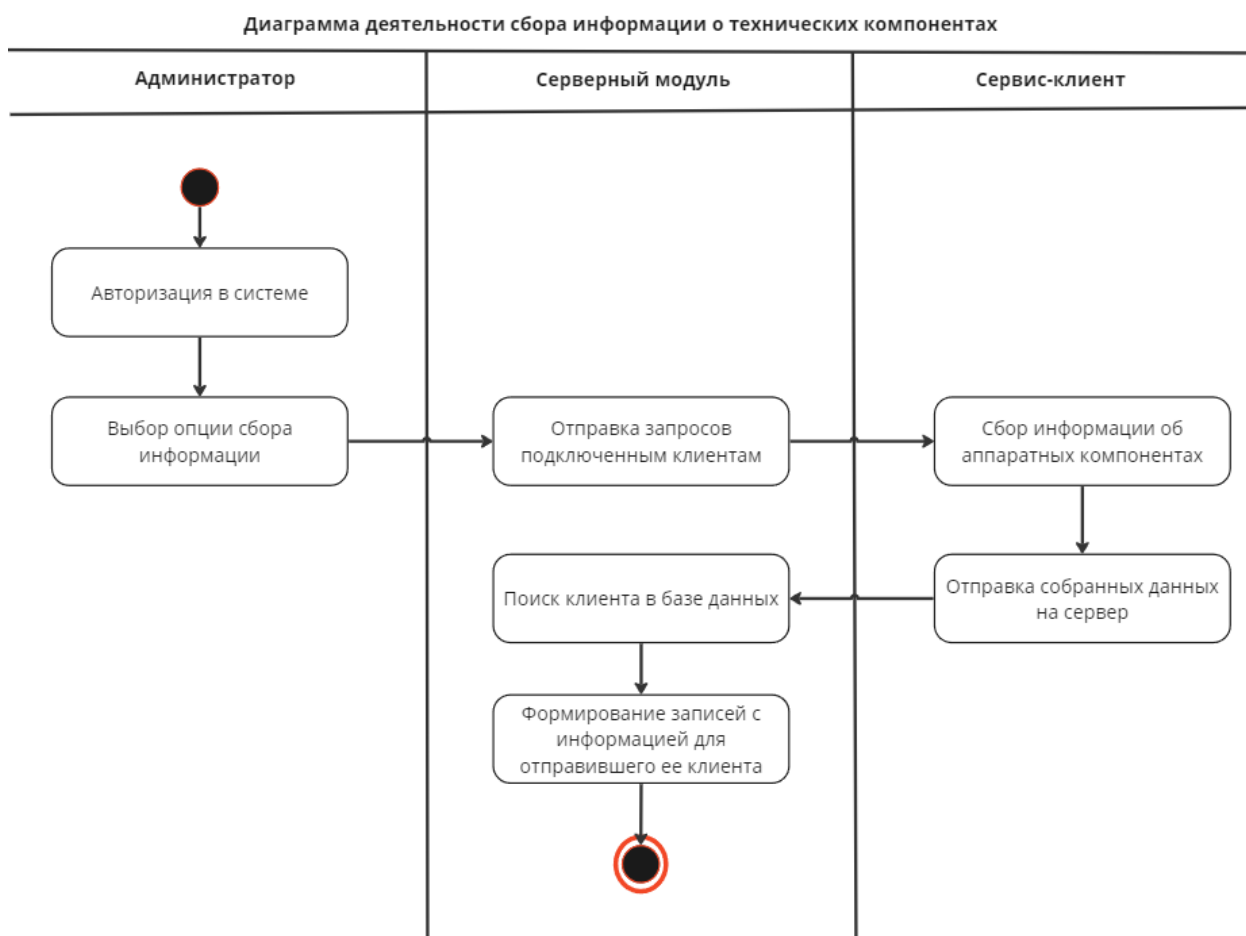


Рисунок 2.5 Диаграмма деятельности для процесса сбора информации о технических характеристиках

Моделирование основных сценариев использования сервиса для управления личной эффективностью, основанное на методологиях IDEF и UML, позволило глубоко проанализировать и структурировать бизнес-процессы. Детальные диаграммы последовательности и деятельности способствовали формированию четкой картины структуры и поведения системы, что является ключевым для её эффективной разработки и оптимизации.

2.3. Проектирование графического интерфейса пользователя

В эпоху цифровизации немаловажным аспектом является создание эффективного и интуитивно понятного GUI – графического интерфейса пользователя. Необходимым требованием для GUI является удобное расположение управляющих кнопок и интуитивность графического интерфейса, что позволит пользователям системы работать в ней эффективнее и быстрее. Также при создании графического интерфейса необходимо придерживаться единой стилистики.

Следующим не мало важным требованием к GUI является адаптивность под различные экраны устройств, то есть необходимо расположить все элементы управления и

данные таким образом, чтобы система оставалась удобной и эффективной в использовании при любом размере экрана.

В ходе разработки GUI были созданы макеты следующих страниц:

- 1) Страница авторизации (рис. 3.1)
- 2) Страница с панелью управления администратора (рис. 3.2)
- 3) Страница с информацией об аппаратных компонентах клиента (рис 3.3)
- 4) Страница с историей браузера клиента (рис 3.4)

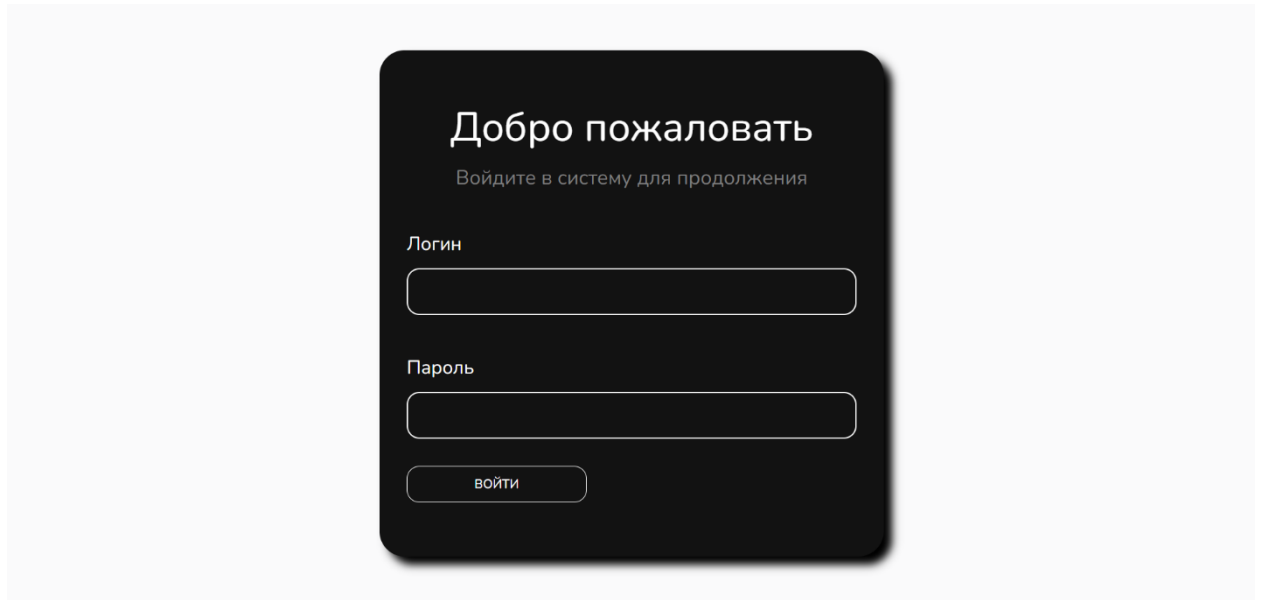


Рисунок 3.1 Страница авторизации в системе

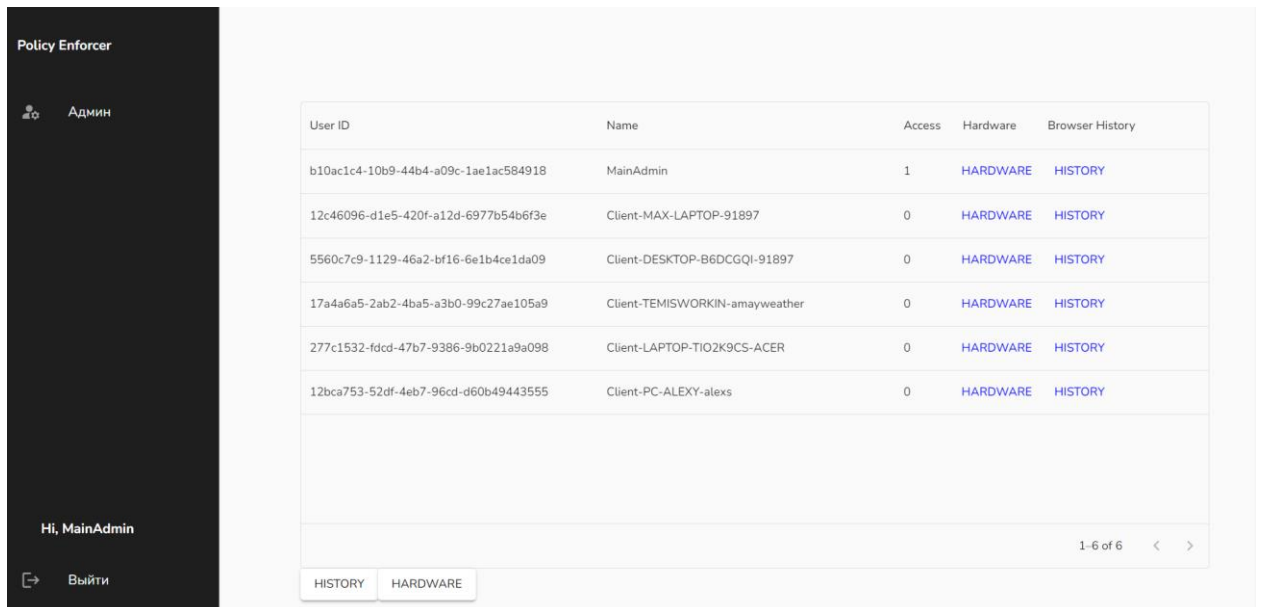
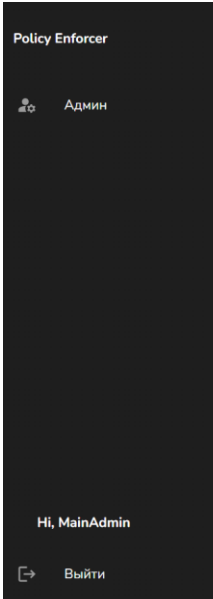


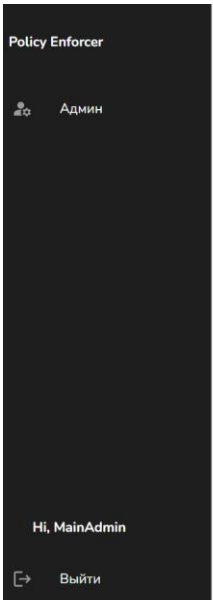
Рисунок 3.2 Панель управления администратора



Name	Temperature	Load	Date
Generic Memory		61.480323791503906	2023-12-21T19:57:42.513
AMD Ryzen 9 7900X	0	11.288243293762207	2023-12-21T19:30:50.307
Generic Memory		61.480323791503906	2023-12-21T19:57:42.513
AMD Ryzen 9 7900X	0	11.288243293762207	2023-12-21T19:30:50.307
AMD Ryzen 9 7900X	0	11.288243293762207	2023-12-21T19:30:50.307
NVIDIA GeForce RTX 4070	66.78125	30.3713321685791	2023-12-21T19:56:06.383
Generic Memory		61.480323791503906	2023-12-21T19:57:42.513
Generic Memory		61.480323791503906	2023-12-21T19:57:42.513
Generic Memory		61.480323791503906	2023-12-21T19:57:42.513

1-8 of 36 < >

Рисунок 3.3 Страница просмотра данных о технических компонентах клиента



Url	Date	Browser Name
https://web.telegram.org/a/#1046604148	2023-12-21T19:53:35.637	Opera GX
https://web.telegram.org/a/#1046604148	2023-12-21T19:53:35.637	Opera GX
https://stackoverflow.com/questions/65842591/net-core-solution-build-giv...	2023-12-21T19:45:48.24	Opera GX
https://stackoverflow.com/questions/65842591/net-core-solution-build-giv...	2023-12-21T19:45:48.24	Opera GX
https://www.google.com/search?client=opera-gx&q=generatedepsfile+fail...	2023-12-21T19:45:40.543	Opera GX
https://www.google.com/search?client=opera-gx&q=generatedepsfile+fail...	2023-12-21T19:45:40.543	Opera GX
https://stackoverflow.com/questions/55806053/the-generatedepsfile-task...	2023-12-21T19:45:39.733	Opera GX
https://stackoverflow.com/questions/55806053/the-generatedepsfile-task...	2023-12-21T19:45:39.733	Opera GX

1-8 of 150 < >

Рисунок 3.4 Страница просмотра истории браузера клиента

Выше были представлены основные спроектированные элементы пользовательского интерфейса данного сервиса.

2.4. Проектирование и разработка модели данных

В разработке любого программного продукта фундаментальное значение также имеет эффективная система хранения данных, обеспечивающая надежность, производительность и масштабируемость. В рамках данной работы было принято решение использовать систему управления базами данных Microsoft SQL Server.

Структура хранения будет представлена 3 таблицами, которые представлены в таблице ниже. (табл. 2.1)

Таблица 2.1

Описание структур таблиц базы данных для системы внедрения политик безопасности

Наименование поля	Тип поля	Описание поля
Users – таблица пользователей		
UserID	uniqueidentifier	Идентификатор пользователя
Login	nvarchar(40)	Логин
Password	binary(16)	Пароль
AccessLevel	int	Уровень доступа
HardwareInfo – таблица с информацией об аппаратных компонентах		
MeasurementID	uniqueidentifier	Идентификатор записи
InstanceName	nvarchar(70)	Название компонента
Temperature	float	Температура
Load	float	Процент загрузки
DateMeasured	datetime	Дата замера
UserID	uniqueidentifier	Идентификатор пользователя
BrowserHistory – таблица с историей браузера клиентов		
ID	uniqueidentifier	Идентификатор записи
UserID	uniqueidentifier	Идентификатор пользователя
URL	text	Адрес ссылки
DateVisited	datetime	Дата посещения
Browser	nvarchar(30)	Название браузера

3. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

Разработка была начата с создания серверного модуля. Первым следует рассмотреть класс Program, так как с него начинается исполнение всей серверной логики. В первую очередь создается «построитель» приложения, где регистрируются и настраиваются необходимые сервисы, после чего приложение строится, для него проводится дополнительная конфигурация и производится его запуск. (рис 4.1.1 и рис. 4.2.2)

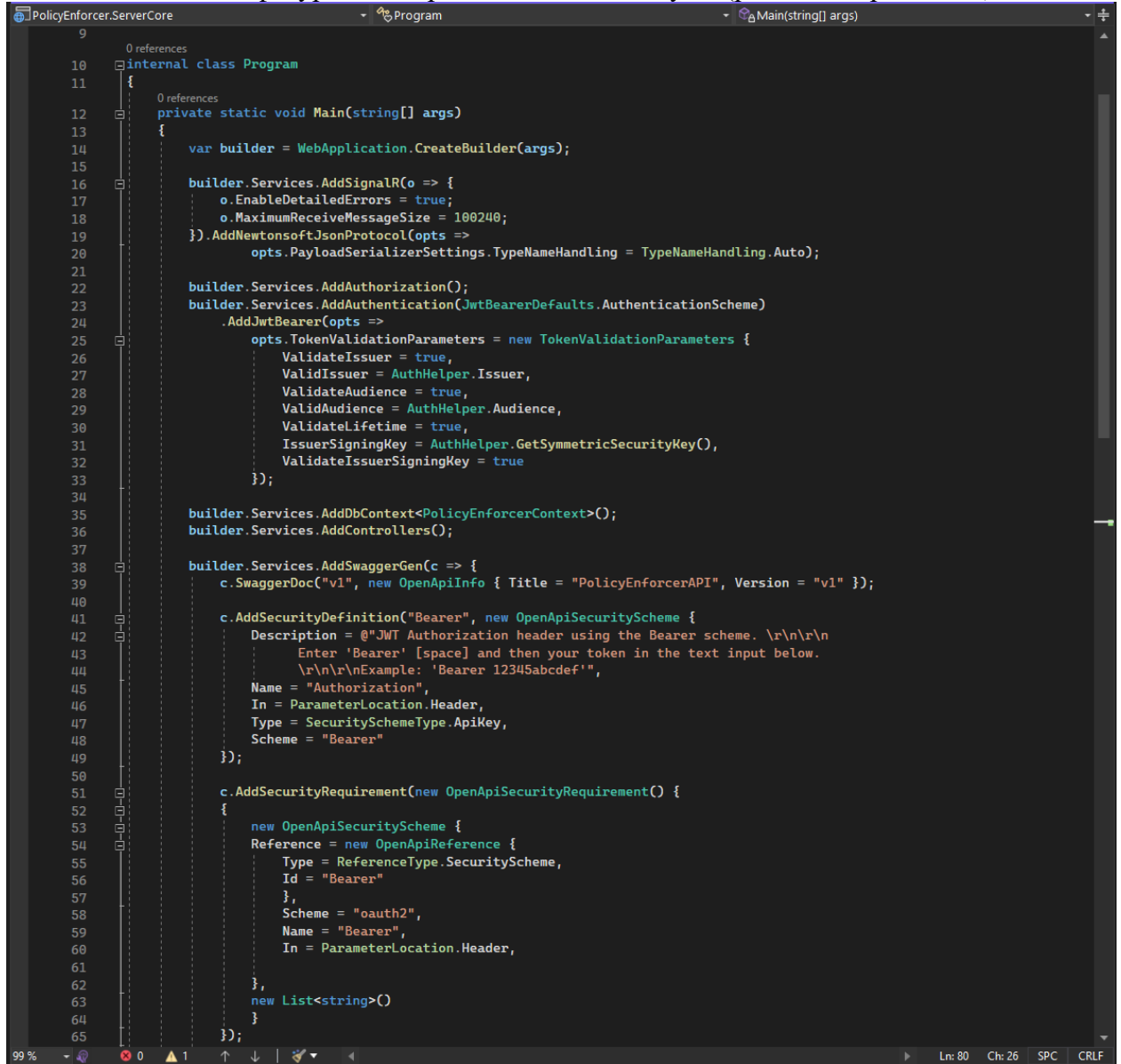


Рисунок 4.1.1. Конфигурация построителя приложения.

```

        // Set the comments path for the Swagger JSON and UI.
        var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
        var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
        c.IncludeXmlComments(xmlPath);
    });

    var configBuilder = new ConfigurationBuilder()
        .AddJsonFile("appsettings.json");

    var config = configBuilder.Build();
    var url = config.GetValue<string>("WorkingURL");

    builder.WebHost.UseUrls(url);
    var app = builder.Build();

    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI(options =>
        {
            options.SwaggerEndpoint("/swagger/v1/swagger.json", "v1");
        });
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();

    app.UseCors(x => x
        .AllowAnyMethod()
        .AllowAnyHeader()
        .SetIsOriginAllowed(origin => true) // allow any origin
        .AllowCredentials());

    app.MapControllers();
    app.MapHub<DataCollectionHub>("/data");

    app.Run();
}

```

Рисунок 4.1.2. Конфигурация и запуск приложения.

Взаимодействие с сервером производится посредством обращения к его методам API и предназначенному для клиентов хабу. Для авторизации был создан контроллер API под названием UsersController (рис. 4.2.1)

```

namespace PolicyEnforcer.ServerCore.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class UsersController : ControllerBase
    {
        private readonly PolicyEnforcerContext _context;
        0 references
        public UsersController(PolicyEnforcerContext context)
        {
            _context = context;
        }
    }
}

```

Рисунок 4.2.1. Контроллер UsersController.

Данный контроллер описывает три метода:

1. Login (рис. 4.2.2) – вход в систему используя имеющиеся учетные данные. Если авторизация проходит успешно, метод возвращает токен для дальнейшего использования и идентификатор пользователя в системе.
2. Register (рис. 4.2.3) – регистрация нового пользователя. Если переданные данные не конфликтуют с уже созданными учетными записями, метод возвращает токен для дальнейшего использования и идентификатор пользователя в системе

3. GetUser (рис. 4.2.4) – получение информации о пользователе в систему. Если пользователь был найден, система возвращает его идентификатор, имя и уровень доступа.

```

/// <summary>
/// Авторизация в системе
/// </summary>
/// <param name="loginInfo">Данные для входа</param>
/// <returns>токен</returns>
[HttpPost("login")]
0 references
public IActionResult Login(
    [FromBody] UserDTO loginInfo)
{
    var hashedPassword = AuthHelper.HashString(loginInfo.Password);
    var user = _context.Users.FirstOrDefault(x => x.Login == loginInfo.Login && x.Password == hashedPassword);

    if (user is null)
    {
        return Unauthorized();
    }

    var claims = new List<Claim> { new Claim(ClaimTypes.Name, user.Login), new Claim(ClaimTypes.Role, user.AccessLevel.ToString()) };
    var jwt = new JwtSecurityToken(
        issuer: AuthHelper.Issuer,
        audience: AuthHelper.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddDays(1),
        signingCredentials: new SigningCredentials(AuthHelper.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));

    return Ok(new TokenDTO { Token = new JwtSecurityTokenHandler().WriteToken(jwt), UserID = user.UserID });
}

```

Рисунок 4.2.2. Метод Login.

```

/// <summary>
/// Регистрация в системе
/// </summary>
/// <param name="loginInfo">данные для входа</param>
/// <returns>токен</returns>
[HttpPost("register")]
0 references
public IActionResult Register(
    [FromBody] UserDTO loginInfo)
{
    var user = _context.Users.FirstOrDefault(x => x.Login == loginInfo.Login);

    if (user is not null)
    {
        return Conflict();
    }

    user = new User()
    {
        Login = loginInfo.Login,
        AccessLevel = 0,
        Password = AuthHelper.HashString(loginInfo.Password),
        UserID = Guid.NewGuid()
    };

    _context.Users.Add(user);
    _context.SaveChanges();

    var claims = new List<Claim> { new Claim(ClaimTypes.Name, user.Login), new Claim(ClaimTypes.Role, user.AccessLevel.ToString()) };
    var jwt = new JwtSecurityToken(
        issuer: AuthHelper.Issuer,
        audience: AuthHelper.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddDays(1),
        signingCredentials: new SigningCredentials(AuthHelper.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));

    return Ok(new TokenDTO { Token = new JwtSecurityTokenHandler().WriteToken(jwt), UserID = user.UserID });
}

```

Рисунок 4.2.3. Метод Register.

```

/// <summary>
/// Поиск пользователя по идентификатору
/// </summary>
/// <param name="userID">идентификатор пользователя</param>
/// <returns></returns>
[Authorize]
[HttpGet("{userID}")]
0 references
public IActionResult GetUser(Guid userID)
{
    var user = _context.Users.FirstOrDefault(x => x.UserId == userID);

    if (user is null)
    {
        return BadRequest();
    }

    // Создание конфигурации сопоставления
    var config = new MapperConfiguration(cfg => cfg.CreateMap<User, UserResponseDTO>());
    // Настройка AutoMapper
    var mapper = new Mapper(config);
    // сопоставление
    var result = mapper.Map<UserResponseDTO>(user);

    return Ok(result);
}

```

Рисунок 4.2.4. Метод GetUser.

После авторизации, веб-клиент переходит к использованию методов, описанных в контроллере AdminController (рис. 4.3.1)

```

namespace PolicyEnforcer.ServerCore.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize(Roles = "admin")]
    1 reference
    public class AdminController : ControllerBase
    {
        private IHubContext<DataCollectionHub> _dataHub;
        private PolicyEnforcerContext _context;
        0 references
        public AdminController(IHubContext<DataCollectionHub> dataHub, PolicyEnforcerContext context)
        {
            _dataHub = dataHub;
            _context = context;
        }
    }
}

```

Рисунок 4.3.1. Контроллер AdminController.

Данный контроллер требует авторизации с правами уровня администратора и предлагает следующий перечень методов:

1. GetUsers (рис. 4.3.2) – возвращает список всех зарегистрированных пользователей.
2. RequestBrowserHistory (рис. 4.3.3) – отправляет всем подключенным клиентам запрос на сбор истории браузера
3. RequestHardwareInfo (рис. 4.3.4) – отправляет всем подключенным клиентам запрос на сбор информации об аппаратных компонентах
4. GetHardwareReadings (рис. 4.3.5) – получает собранную информацию об аппаратных компонентах для конкретного клиента
5. GetBrowserHistory (рис. 4.3.6) – получает собранную историю браузера для конкретного клиента

```

/// <summary>
/// Возвращает список всех пользователей
/// </summary>
/// <returns></returns>
[HttpGet("getusers")]
0 references
public IActionResult GetUsers()
{
    // Создание конфигурации сопоставления
    var config = new MapperConfiguration(cfg => cfg.CreateMap<User, UserResponseDTO>());
    // Настройка AutoMapper
    var mapper = new Mapper(config);
    // сопоставление
    var users = mapper.Map<List<UserResponseDTO>>(_context.Users.ToList());

    return Ok(users);
}

```

Рисунок 4.3.2. Метод GetUsers

```

/// <summary>
/// Генерирует запросы на сбор данных об истории посещения сайтов подключенных клиентов
/// </summary>
[HttpGet("requestbrowserhistory")]
0 references
public async Task<IActionResult> RequestBrowserHistory()
{
    await _dataHub.Clients.All.SendAsync("GetBrowserHistory", 10);
    return Ok();
}

```

Рисунок 4.3.3. Метод RequestBrowserHistory

```

/// <summary>
/// Генерирует запросы на сбор данных об аппаратных компонентах подключенных клиентов
/// </summary>
[HttpGet("requesthardwarereadings")]
0 references
public async Task<IActionResult> RequestHardwareInfo()
{
    await _dataHub.Clients.All.SendAsync("GetHardwareInfo");
    return Ok();
}

```

Рисунок 4.3.4. Метод RequestHardwareInfo

```

[HttpGet("gethardwarereadings/{userID}")]
0 references
public async Task<IActionResult> GetHardwareReadings(Guid userID)
{
    var target = _context.HardwareInfos.Where(x => x.UserId == userID).OrderByDescending(x => x.DateMeasured);

    // Создание конфигурации сопоставления
    var config = new MapperConfiguration(cfg => cfg.CreateMap<HardwareInfo, HardwareInfoDTO>());
    // Настройка AutoMapper
    var mapper = new Mapper(config);
    // сопоставление
    var result = mapper.Map<List<HardwareInfoDTO>>(target.ToList());

    return Ok(result);
}

```

Рисунок 4.3.5. Метод GetHardwareReadings

```

/// <summary>
/// Возвращает историю посещений сайтов для клиента
/// </summary>
/// <param name="userID">клиент-цель</param>
/// <returns></returns>
[HttpGet("getbrowserhistory/{userID}")]
0 references
public async Task<IActionResult> GetBrowserHistory(Guid userID)
{
    var target = _context.BrowserHistories.Where(x => x.UserId == userID).OrderByDescending(x => x.DateVisited).Take(150);

    // Создание конфигурации сопоставления
    var config = new MapperConfiguration(cfg => cfg.CreateMap<BrowserHistory, BrowserHistoryDTO>());
    // Настройка AutoMapper
    var mapper = new Mapper(config);
    // сопоставление
    var result = mapper.Map<List<BrowserHistoryDTO>>(target.ToList());

    return Ok(result);
}

```

Рисунок 4.3.6. Метод GetBrowserHistory

Для реализации общения клиентов-сервисов с сервером на стороне сервера был реализован хаб DataCollectionHub, куда клиенты могут подключаться и отправлять данные при поступлении запросов (рис. 4.4.1)

```

4 references
public class DataCollectionHub : Hub
{
    private PolicyEnforcerContext _context;
    0 references
    public DataCollectionHub(PolicyEnforcerContext dbContext)
    {
        _context = dbContext;
    }
}

```

Рисунок 4.4.1 Хаб DataCollectionHub

Данный хаб содержит следующий набор методов, доступный подключенным клиентам:

- 1) ReturnHardwareReadings (рис. 4.4.2) – метод, с помощью которого клиенты возвращают данные об аппаратных компонентах.
- 2) ReturnBrowserHistory (рис. 4.4.3) – метод, с помощью которого клиенты возвращают историю браузера.

```

0 references
public async Task ReturnHardwareReadings(List<string> readings)
{
    foreach (var pieceRaw in readings)
    {
        var piece = JsonConvert.DeserializeObject<HardwarePiece>(pieceRaw);
        _context.HardwareInfos.Add(new()
        {
            DateMeasured = piece.TimeMeasured,
            MeasurementId = Guid.NewGuid(),
            InstanceName = piece.InstanceName,
            Load = piece.Load,
            Temperature = piece.Temperature,
            UserId = piece.UserID
        });
    }

    await _context.SaveChangesAsync();
}

```

Рисунок 4.4.2. Метод ReturnHardwareReadings

```

0 references
public async Task ReturnBrowserHistory(List<string> readings)
{
    foreach (var pieceRaw in readings)
    {
        var piece = JsonConvert.DeserializeObject<BrowserHistory>(pieceRaw);
        _context.BrowserHistories.Add(new()
        {
            DateVisited = piece.DateVisited,
            Id = Guid.NewGuid(),
            UserId = piece.UserId,
            BrowserName = piece.BrowserName,
            Url = piece.Url,
        });
    }

    await _context.SaveChangesAsync();
}

```

Рисунок 4.4.3. Метод ReturnBrowserHistory

Стоит отметить, что методы отправки запросов на клиент в хабе отсутствуют, а запросы клиентам производятся из класса AdminController, куда хаб передается при помощи внедрения зависимостей.

Касаемо клиента-сервиса, он выполнен в виде приложения-сервиса, которое функционирует без вмешательства человека. При первом запуске клиент генерирует свои учетные данные, используя данные о вычислительной машине, авторизуется на сервере путем запроса к API, подключается к хабу и ожидает дальнейших указаний от сервера.

Учетные данные сохраняются в файл конфигурации и при последующих запусках используются для авторизации в системе.

Исполнение программы начинается с класса Program (рис. 4.5.1), где регистрируются и конфигурируются необходимые для работы приложения сервисы.

```

0 references
internal class Program
{
    0 references
    private static void Main(string[] args)
    {
        var configBuilder = new ConfigurationBuilder()
            .AddJsonFile("appsettings.json");

        var configuration = configBuilder.Build();

        IHost host = Host.CreateDefaultBuilder(args)
            .ConfigureServices(services =>
            {
                services.AddHostedService<ServerConnectionService>();
                services.AddTransient<IHistoryCollectionService, HistoryCollectionService>();
                services.AddTransient<IHardwareMonitoringService, HardwareMonitoringService>();
                services.Configure<PolicyEnforcer.Service.POCO.LoginInfo>(configuration.GetSection("LoginInfo"));
            })
            .ConfigureLogging((context, logging) =>
            {
                logging.ClearProviders();
                logging.AddConfiguration(context.Configuration.GetSection("Logging"));

                // для отладки
                if (context.HostingEnvironment.IsDevelopment())
                {
                    AllocConsole();
                    logging.AddConsole();
                    logging.AddDebug();
                }
            })
            .Build();

        host.Run();
    }
}

[DllImport("kernel32.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
1 reference
static extern bool AllocConsole();
}

```

Рисунок 4.5.1. Класс Program.

В данном коде стоит обратить внимание на параметры при вызове метода ConfigureServices. В нем регистрируются три сервиса, необходимых для работы приложения:

1. ServerConnectionService (рис. 4.6.1) – основной сервис, который проводит авторизацию и устанавливает подключение к серверу.
2. HistoryCollectionService (рис. 4.7.1) – сервис, отвечающий за сбор истории браузера.
3. HardwareMonitoringService (рис. 4.8.1) – сервис, отвечающий за сбор данных об аппаратных компонентах компьютера.

```

4 references
public class ServerConnectionService : IServerConnectionService, IHostedService
{
    private readonly IHardwareMonitoringService _monitoringService;
    private readonly IHistoryCollectionService _historyCollectionService;
    private readonly ILogger<ServerConnectionService> _logger;
    private readonly HttpClient _httpClient;
    private Guid _userId;
    private readonly POCO.LoginInfo _loginInfo;

    6 references
    HubConnection _connection { get; set; }

    0 references
    public ServerConnectionService(IHistoryCollectionService historyCollectionsvc, IHardwareMonitoringService monitoringsvc, ILogger<ServerConnectionService> logger, IOptions<POCO.LoginInfo> loginConfig)
    {
        _monitoringService = monitoringsvc;
        _historyCollectionService = historyCollectionsvc;
        _logger = logger;
        _loginInfo = loginConfig.Value;

        var clientHandler = new HttpClientHandler()
        {
            ServerCertificateCustomValidationCallback = (message, cert, chain, errors) => { return true; }
        };
        _httpClient = new HttpClient(clientHandler);
    }
}

```

Рисунок 4.6.1. Класс ServerConnectionService.

ServerConnectionService содержит следующие методы:

1. StartAsync (рис. 4.6.2) – запуск сервиса.
2. StopAsync (рис. 4.6.3) – остановка сервиса.
3. ConfigureConnection (рис. 4.6.4) – установка и настройка подключения к серверу.
4. Login (рис. 4.6.5) – авторизация в системе.
5. Register (рис. 4.6.6) – регистрация в системе.
6. RenewToken (рис. 4.6.7) – обновление токена.
7. GetBrowserHistory (рис. 4.6.8) – метод, вызываемый при получении запроса “GetBrowserHistory”, запрашивает у соответствующего сервиса историю браузера, после чего отправляет ее на сервер.
8. GetTemps (рис. 4.6.8) – метод, вызываемый при получении запроса “GetHardwareReadings”, запрашивает у соответствующего сервиса информацию об аппаратных компонентах, после чего отправляет ее на сервер.

```

0 references
public async Task StartAsync(Cancellation_token cancellation_token)
{
    try
    {
        await ConfigureConnection();

        await Task.CompletedTask;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex.Message);
    }
}

```

Рисунок 4.6.2. Метод StartAsync.

```

0 references
public async Task StopAsync(Cancellation_token cancellation_token)
{
    _logger.LogInformation($"Service stopped at {DateTime.Now}");
    await Task.CompletedTask;
}

```

Рисунок 4.6.3. Метод StopAsync.

```

1 reference
private async Task ConfigureConnection()
{
    var configFile = System.Configuration.ConfigurationManager.OpenExeConfiguration(System.Configuration.ConfigurationUserLevel.None);
    var settings = configFile.AppSettings.Settings;

    var loginInfo = await Login(settings);

    var url = settings["WorkingURL"]; // Адрес сервера хранится в конфиге

    _connection = new HubConnectionBuilder()
        .WithUrl($"{url}/data",
            options =>
            {
                options.UseDefaultCredentials = true;
                options.HttpMessageHandlerFactory = (msg) =>
                {
                    if (msg is HttpClientHandler clientHandler)
                    {
                        // bypass SSL certificate
                        clientHandler.ServerCertificateCustomValidationCallback +=
                            (sender, certificate, chain, sslPolicyErrors) => { return true; };
                    }

                    return msg;
                }
            })
        .AccessTokenProvider = () => Task.FromResult("Bearer " + loginInfo.Token);
    .WithAutomaticReconnect()
    .AddNewtonsoftJsonProtocol(opts =>
        opts.PayloadSerializerSettings.TypeNameHandling = Newtonsoft.Json.TypeNameHandling.Auto)
    .Build();

    await _connection.StartAsync();
    _logger.LogInformation($"Connection established at: {DateTimeOffset.Now}");

    _connection.On("GetHardwareInfo", GetTemps);
    _connection.On<int>("GetBrowserHistory", GetBrowserHistory);

    configFile.Save(ConfigurationSaveMode.Modified);
    System.Configuration.ConfigurationManager.RefreshSection(configFile.AppSettings.SectionInformation.Name);
}

```

Рисунок 4.6.4. Метод ConfigureConnection.

```

1 reference
private async Task<LoginInfo> Login(KeyValueCollection? settings)
{
    var filepath = Path.Combine(Environment.CurrentDirectory, "config.xml");
    if (!File.Exists(filepath))
    {
        var loginInfo = await Register(settings);
        return loginInfo;
    }

    var config = XMLHelper.FromXmlFile<LoginConfig>(filepath);

    var login = config.Login;
    var password = config.Password;

    var token = await RenewToken(new UserDTO() { login = login, password = password });

    return new LoginInfo { Username = login, Password = password, Token = token };
}

```

Рисунок 4.6.5. Метод Login.

```

1 reference
private async Task<LoginInfo> Register(KeyValueCollection? settings)
{
    var loginInfo = LoginInfo.Generate();

    var content = new UserDTO() { login = loginInfo.Username, password = loginInfo.Password };

    string json = JsonConvert.SerializeObject(content);
    var httpContent = new StringContent(json, Encoding.UTF8, "application/json");

    var message = new HttpRequestMessage
    {
        Method = HttpMethod.Post,
        Content = httpContent,
        RequestUri = new UriBuilder("https://26.85.180.83:6969/api/Users/register").Uri
    };

    var response = _httpClient.Send(message);

    response.EnsureSuccessStatusCode();

    var token = await response.Content.ReadFromJsonAsync<TokenDTO>();

    this._userID = token.UserID;
    loginInfo.Token = token.Token;

    var loginConfig = new LoginConfig
    {
        Login = loginInfo.Username,
        Password = loginInfo.Password,
        Token = token.Token,
    };

    var filepath = Path.Combine(Environment.CurrentDirectory, "config.xml");
    XMLHelper.ToXmlFile(loginConfig, filepath);

    return loginInfo;
}

```

Рисунок 4.5.6. Метод Register.

```

1 reference
private async Task<string> RenewToken(UserDTO userinfo)
{
    var message = new HttpRequestMessage
    {
        Method = HttpMethod.Post,
        Content = JsonConvert.Create(userinfo),
        RequestUri = new UriBuilder("https://26.85.180.83:6969/api/Users/login").Uri
    };

    var response = _httpClient.Send(message);
    response.EnsureSuccessStatusCode();

    var token = await response.Content.ReadFromJsonAsync<TokenDTO>();
    this._userID = token.UserID;

    return token.Token;
}

```

Рисунок 4.5.7. Метод RenewToken.

```

1 reference
public async void GetBrowserHistory(int batchSize = 10)
{
    _logger.LogInformation($"Received history collection request at {DateTimeOffset.Now}");
    var readings = _historyCollectionService.GetBrowsersHistory(DateTime.Now.AddDays(-2), _userID);

    while (readings.Count > 0)
    {
        var ceiling = readings.Count > batchSize ? batchSize : readings.Count;
        _connection.InvokeAsync("ReturnBrowserHistory", readings.GetRange(0, ceiling));
        readings.RemoveRange(0, ceiling);
    }
}

```

Рисунок 4.5.8. Метод GetBrowserHistory.

```

1 reference
public async void GetTemps()
{
    _logger.LogInformation($"Received hardware poll request at {DateTimeOffset.Now}");
    var readings = _monitoringService.PollHardware(_userID);

    await _connection.InvokeAsync("ReturnHardwareReadings", readings);
}

```

Рисунок 4.5.9. Метод GetTemps.

```

1 reference
public class HistoryCollectionService : IHistoryCollectionService
{
    2 references
    private static string AppDataLocal => Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
    2 references
    private static string AppDataRoaming => Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
}

```

Рисунок 4.6.1. Класс HistoryCollectionService.

Класс HistoryCollectionService содержит следующий перечень методов:

1. GetBrowsersHistory (рис. 4.6.2) – возвращает историю посещений, хранимую в базах данных chromium-браузеров.
2. GetHistoryFiles (рис. 4.6.3) – вспомогательный метод, возвращающий пути к базам данных самых популярных веб-сайтов.

```

2 references
public List<string> GetBrowsersHistory(DateTime from, Guid userID)
{
    var conStrings = GetHistoryFiles();

    var result = new List<string>();
    foreach (var con in conStrings)
    {
        string tempFilePath = Path.Combine(Environment.CurrentDirectory, DateTime.Now.Ticks.ToString());
        File.Copy(con.DBPath, tempFilePath);

        try
        {
            using var connection = new SQLiteConnection($"Data Source={tempFilePath}", true);
            using SQLiteCommand command = connection.CreateCommand();

            connection.Open();

            const long secondsBetween19701601 = 11644473600;
            var timestamp = ((DateTimeOffset)from).ToUnixTimeSeconds() + secondsBetween19701601;
            command.CommandText = $"select * from urls where last_visit_time >= {timestamp * 1000000}";

            using (SQLiteDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    // Структура БД истории посещений в chromium-браузерах унифицирована
                    var url = new VisitedURL(con.BrowserName, reader["url"].ToString(), reader["last_visit_time"].ToString(), userID);
                    result.Add(JsonConvert.SerializeObject(url));
                }
            }
            connection.Close();
        }
        finally
        {
            SQLiteConnection.ClearAllPools();
            GC.Collect();
            GC.WaitForPendingFinalizers();
            File.Delete(tempFilePath);
        }
    }

    return result;
}

```

Рисунок 4.6.2. Метод GetBrowsersHistory.

```

1 reference
private List<BrowserModel> GetHistoryFiles()
{
    var result = new List<BrowserModel>();

    result.AddBrowser(AppDataLocal, "Google\\Chrome\\User Data\\Default\\History", "Google Chrome");
    result.AddBrowser(AppDataRoaming, "Opera Software\\Opera GX Stable\\History", "Opera GX");
    result.AddBrowser(AppDataRoaming, "Opera Software\\Opera Stable\\History", "Opera");
    result.AddBrowser(AppDataLocal, "Microsoft\\Edge\\User Data\\Default\\History", "Microsoft Edge");

    return result;
}

```

Рисунок 4.6.3. Метод GetHistoryFiles.

```

2 references
public class HardwareMonitoringService : IHardwareMonitoringService
{
    private Computer computer;

    0 references
    public HardwareMonitoringService()
    {
        UpdateComputerInfo();
    }
}

```

Рисунок 4.7.1 Класс HardwareMonitoringService.

Класс HardwareMonitoringService содержит следующий перечень методов:

1. UpdateComputerInfo (рис. 4.7.2) – инициализирует необходимые для сбора данных переменные.
2. PollHardware (рис. 4.7.3) – организует собранные данные для обнаруженных компонент.
3. GetHardwareReadings (рис. 4.7.4) – опрашивает сенсоры на обнаруженных компонентах, и возвращает их показатели.

```
1 reference
private void UpdateComputerInfo()
{
    computer = new Computer
    {
        IsCpuEnabled = true,
        IsGpuEnabled = true,
        IsMemoryEnabled = true,
        IsMotherboardEnabled = true,
        IsStorageEnabled = true,
    };

    var visitor = new UpdateVisitor();

    computer.Open();
    computer.Accept(visitor);
}
```

Рисунок 4.7.2. Метод UpdateComputerInfo.

```
2 references
public List<string> PollHardware(Guid userID)
{
    var result = new List<string>();

    foreach (var hw in computer.Hardware)
    {
        result.AddRange(GetHardwareReadings(hw, userID));
    }

    return result;
}
```

Рисунок 4.7.3. Метод PollHardware.


```

2 references
private List<string> GetHardwareReadings(IHardware hw, Guid userID)
{
    var result = new List<string>();

    var hwPiece = new HardwarePiece { InstanceName = hw.Name, UserID = userID };
    foreach (var sensor in hw.Sensors)
    {
        if (sensor.SensorType == SensorType.Temperature)
        {
            hwPiece.Temperature = sensor.Value;
        }
        if (sensor.SensorType == SensorType.Load)
        {
            hwPiece.Load = sensor.Value;
        }
    }

    hwPiece.TimeMeasured = DateTime.Now;

    result.Add(JsonConvert.SerializeObject(hwPiece));

    foreach (var childHw in hw.SubHardware)
    {
        result.AddRange(GetHardwareReadings(childHw, userID));
    }
    return result;
}

```

Рисунок 4.7.4. Метод GetHardwareReadings.

Также, для унификации используемых моделей на клиенте и сервере, была разработана библиотека PolicyEnforcer.Interfaces на платформе .NET Standard 2.0, содержащая интерфейсы для моделей информации, обмен которой происходит при выполнении основной деятельности приложения. Используемые модели в модулях должны наследовать данные интерфейсы. Всего было разработано 2 интерфейса – IHardwarePiece (рис. 4.8.1) для аппаратных компонент и IVisitedURL (рис. 4.8.2) для посещенных страниц.

```

2 references
public interface IHardwarePiece
{
    4 references
    string InstanceName { get; set; }
    4 references
    Guid UserID { get; set; }
    4 references
    float? Temperature { get; set; }
    4 references
    float? Load { get; set; }
    4 references
    DateTime TimeMeasured { get; set; }
}

```

Рисунок 4.8.1. Интерфейс IHardwarePiece.


```

1 reference
public interface IVisitedURL
{
    3 references
    string Url { get; set; }
    2 references
    Guid UserID { get; set; }
    3 references
    string BrowserName { get; set; }
    3 references
    DateTime DateVisited { get; set; }
}

```

Рисунок 4.8.2. Интерфейс IVisitedURL.

Для эффективного управления данными и обеспечения взаимодействия с бэкенд-частью нашего проекта был создан веб-клиент. В рамках его разработки применялись React и TypeScript, а также инструменты Vite и Yarn для оптимизации процесса сборки и управления зависимостями.

Одним из ключевых компонентов, реализованных в веб-клиенте, является форма авторизации (LoginForm) (рис.4.9.1). Этот компонент отвечает за передачу данных, необходимых для процесса авторизации, обеспечивая безопасность и удобство пользовательского входа.

```

src > components > Auth > loginForm.tsx
1 > import { Box, Button, InputLabel, TextField, Typography } from '@mui/material';
2
3 const LoginForm: React.FC = () => {
4   const navigate = useNavigate();
5
6   const { control, handleSubmit } = useForm<ILoginCred>({
7     mode: 'onChange',
8   });
9
10  const { errors } = useFormState({
11    control,
12  });
13
14  const onSubmit: SubmitHandler<ILoginCred> = async (data) => {
15    await login(data.login, data.password);
16    navigate('/admin');
17  };
18
19  return (
20    <Box
21      width="400px"
22      sx={{
23        bgcolor: '#121212',
24        px: 4,
25        py: 8,
26        borderRadius: '30px',
27        boxShadow: '10px 10px 10px black',
28      }}
29    >
30      <Typography variant="h3" textAlign="center" sx={{ color: 'white', mb: 2 }}>
31        Welcome
32      </Typography>
33
34      <Typography variant="h5" textAlign="center" sx={{ color: 'gray', mb: 6 }}>
35        Please sign in to continue.
36      </Typography>
37
38      <form onSubmit={handleSubmit(onSubmit)}>
39        <Controller
40          name="login"
41          control={control}
42          rules={{ required: true }}
43        >
44          <TextField
45            type="text"
46            value={login}
47            onChange={e => login(e.target.value)}
48            placeholder="Email or phone"
49            sx={{ width: '100%', }}
50          />
51        </Controller>
52
53        <Controller
54          name="password"
55          control={control}
56          rules={{ required: true }}
57        >
58          <TextField
59            type="password"
60            value={password}
61            onChange={e => password(e.target.value)}
62            placeholder="Password"
63            sx={{ width: '100%', }}
64          />
65        </Controller>
66
67        <Button
68          type="submit"
69          size="large"
70          sx={{
71            mt: 4,
72            border: '1px solid',
73            px: 10,
74            py: 1,
75            borderRadius: '15px',
76          }}
77        >
78          Log In
79        </Button>
80      </form>
81    </Box>
82  );
83
84 export default LoginForm;

```

Рисунок 4.9.1. Форма авторизации.

Дополнительно, были созданы компоненты для отображения информации о пользователях (UserList) (рис. 4.9.2), а также для представления данных об аппаратных характеристиках системы (рис. 4.9.3) и истории браузера (рис. 4.9.4).

```

1 > import { Box, Button } from '@mui/material';...
7
8 const GameList: React.FC = () => {
9   const navigate = useNavigate();
10
11 > const columns: GridColDef[] = [ ...
41 ];
42
43 const [users, setUsers] = useState<IUser[]>([]);
44
45 const requestHistory = () => {
46   requestBrowserHistory();
47 };
48
49 const requestHardware = () => {
50   requestHardwareReadings();
51 };
52
53 useEffect(() => {
54   const getUsers = async () => {
55     const response = await getAllUsers();
56     setUsers(response.data);
57   };
58   getUsers();
59 }, []);
60
61 return (
62   <Box>
63     <Box sx={{ height: 550, width: '100%' }}>
64       <DataGrid
65         getRowId={(row) => row.userId}
66         rows={users}
67         columns={columns}
68         initialState={{
69           pagination: {
70             paginationModel: {
71               pageSize: 8,
72             },
73           },
74         }}
75         pageSizeOptions={[8]}
76         disableRowSelectionOnClick
77       />
78     </Box>
79     <Button variant="contained" onClick={() => requestHistory()}>History</Button>
80     <Button variant="contained" onClick={() => requestHardware()}>Hardware</Button>
81   </Box>
82 );
83 };
84
85 export default GameList;
86
87
88

```

Рисунок 4.9.2. Страница данных о пользователях.

```

1 > import { Box, Button, Stack } from '@mui/material'; ...
8
9 const columns: GridColDef[] = [
10   { field: 'instanceName', headerName: 'Name', width: 350 },
11   { field: 'temperature', headerName: 'Temperature', width: 150 },
12   { field: 'load', headerName: 'Load', width: 200 },
13   { field: 'dateMeasured', headerName: 'Date', width: 200 },
14 ];
15
16 const Hardware: React.FC = () => {
17   const [hardware, setHardware] = useState<IHardware[]>([]);
18   const params = useParams();
19
20   console.log(params.userId);
21
22
23   useEffect(() => {
24     const getHardwareInfo = async () => {
25       const response = await getHardware(params.userId);
26       console.log(response);
27       setHardware(response.data);
28     };
29     getHardwareInfo();
30     console.log(hardware);
31   }, []);
32
33   return (
34     <Stack alignItems="center" justifyContent="center" height="100vh" width="100%">
35       <Box sx={{ height: 550, width: '100%' }}>
36         <DataGrid
37           getRowId={(row) => row.instanceName}
38           rows={hardware}
39           columns={columns}
40           initialState={{
41             pagination: {
42               paginationModel: {
43                 pageSize: 8,
44               },
45             },
46           }}
47           pageSizeOptions={[8]}
48           disableRowSelectionOnClick
49         />
50       </Box>
51     </Stack>
52   );
53 };
54
55 export default Hardware;
56

```

Рисунок 4.9.3. Страница данных об аппаратных компонентах.

```

1 > import { Box, Button, Stack } from '@mui/material';
8
9 const columns: GridColDef[] = [
10   { field: 'url', headerName: 'Url', width: 500 },
11   { field: 'dateVisited', headerName: 'Date', width: 300 },
12   { field: 'browserName', headerName: 'Browser Name', width: 200 },
13 ];
14
15 const BrowserHistory: React.FC = (ur) => {
16   const params = useParams();
17
18   const [history, setHistory] = useState<IBrowserHistory[]>([]);
19
20   useEffect(() => {
21     const getHistory = async () => {
22       const response = await getBrowserHistory(params.userId);
23       setHistory(response.data);
24     };
25     getHistory();
26     console.log(history);
27   }, []);
28
29   return (
30     <Stack width="100%" alignItems="center" justifyContent="center" height="100vh">
31       <Box sx={{ height: 550, width: '100%' }}>
32         <DataGrid
33           getRowId={(row) => row.url}
34           rows={history}
35           columns={columns}
36           initialState={{
37             pagination: {
38               paginationModel: {
39                 pageSize: 8,
40               },
41             },
42           }}
43           pageSizeOptions={[8]}
44           disableRowSelectionOnClick
45         />
46       </Box>
47     </Stack>
48   );
49 };
50
51 export default BrowserHistory;
52

```

Рисунок 4.9.4. Страница данных об истории посещения страниц.

Для взаимодействия с веб-API была использована библиотека Axios. Это обеспечивает надежный и удобный способ отправки запросов на сервер, что в свою очередь поддерживает эффективное взаимодействие между веб-клиентом и бэкенд-частью. (рис. 4.9.5 и 4.9.6)

```

1 import axios from 'axios';
2 import { IUser } from '../models/user';
3 import authHeader from '../auth-service/auth-header';
4 import { IBrowserHistory } from '../models/browserHistory';
5 import { IHardware } from '../models/hardware';
6
7 const API_URL = 'https://26.85.180.83:6969/api/Admin/';
8
9 export const getAllUsers = async () => {
10   return await axios.get<IUser[]>(API_URL + 'getusers', { headers: authHeader() });
11 };
12
13 export const getBrowserHistory = async (userId?: string) => {
14   return await axios.get<IBrowserHistory[]>(API_URL + 'getbrowserhistory/' + userId, {
15     headers: authHeader(),
16   });
17 };
18
19 export const getHardware = async (userId?: string) => {
20   return await axios.get<IHardware[]>(API_URL + 'gethardware readings/' + userId, {
21     headers: authHeader(),
22   });
23 };
24
25 export const requestBrowserHistory = async () => {
26   return await axios.get(API_URL + 'requestbrowserhistory', {
27     headers: authHeader(),
28   });
29 };
30
31 export const requestHardwareReadings = async () => {
32   return await axios.get(API_URL + 'requesthardware readings', {
33     headers: authHeader(),
34   });
35 };
36
37

```

Рисунок 4.9.5. Работа с административными методами API.

```

1 > import axios from "axios";...
5
6 const API_URL = "https://26.85.180.83:6969/api/Users/";
7
8 export const login = async (login: string, password: string) => {
9   const response = await axios.post(API_URL + "login", {
10     login,
11     password,
12   });
13   if (response.data.token) {
14     localStorage.setItem("user", JSON.stringify(response.data));
15   }
16   return response.data;
17 };
18
19 export const logout = () => {
20   localStorage.removeItem("user");
21 };
22
23 export const getUserById = async (userId?: string) => {
24   return await axios.get<IUser>(API_URL + userId, { headers: authHeader() });
25 };
26
27 export const getCurrentUserId = () => {
28   const userStr = localStorage.getItem("user");
29   if (userStr) {
30     const authResult: IAuthResult = JSON.parse(userStr);
31     return authResult.userId;
32   }
33   return undefined;
34 };
35
36
37 export const getToken = () => {
38   const userStr = localStorage.getItem("user");
39   if (userStr) {
40     const authResult: IAuthResult = JSON.parse(userStr);
41     return authResult.token;
42   }
43   return "";
44 };
45

```

Рисунок 4.9.6. Работа с методами авторизации API.

4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1. История изменений

В динамическом мире информационных технологий, где бизнес-требования и технологические возможности непрерывно эволюционируют, систематическое отслеживание и управление изменениями ПО становится критически важным аспектом разработки и поддержки систем. История изменений ПО играет важную роль в жизненном цикле разработки продукта, предоставляя ценный контекст, улучшая поддержку и обслуживание, и обеспечивая отчетность в случае необходимости. Для того чтобы отобразить все изменения разрабатываемого модуля системы деловых переговоров была создана таблица истории изменений (таблица 3.1)

Таблица 3.1

История изменений программного продукта

Продолжение таблицы 3.1

Дата	Версия	Автор	Описание изменений
13.10.2023	0.1	Сосновский Максим Евгеньевич	Инициализация решения. Создание проекта серверной части. Конфигурация приложения.
13.10.2023	0.2	Сосновский Максим Евгеньевич	Реализация взаимодействия с базой данных. Описание основных сущностей.
14.10.2023	0.3	Сосновский Максим Евгеньевич	Реализация функционала авторизации. Создание контроллера и объектов передачи данных.
20.10.2023	0.4	Сосновский Максим Евгеньевич	Реализация функционала опроса клиентов. Создание хаба и объектов передачи данных.
21.10.2023	0.4.1	Сосновский Максим Евгеньевич	Создание библиотеки интерфейсов. Изменение объектов передачи данных для реализации интерфейсов.
21.10.2023	0.5	Сосновский Максим Евгеньевич	Создание проекта клиента-сервиса. Конфигурация приложения.
25.10.2023	0.5.1	Сосновский Максим Евгеньевич	Реализация функционала опроса аппаратного обеспечения.
30.10.2023	0.5.2	Сосновский Максим Евгеньевич	Реализация функционала сбора истории посещения сайтов.
18.11.2023	0.6	Сосновский Максим Евгеньевич	Реализация общения клиента с сервером. Создание сервиса, производящего взаимодействие с хабом. Интеграция сервисов опроса аппаратного обеспечения и сбора истории браузеров.
20.11.2023	0.6.1	Сосновский Максим Евгеньевич	Фаза тестирования и исправления ошибок: устранение выявленных недочетов в логике опроса клиентов и реализация уровней доступа при работе с сервером.

Продолжение таблицы 3.1

Дата	Версия	Автор	Описание изменений
24.11.2023	0.6.2	Сосновский Максим Евгеньевич	Оптимизация производительности и повышение конфиденциальности данных серверной части.
26.11.2023	0.6.3	Сосновский Максим Евгеньевич	Оптимизация производительности клиентского модуля.
30.11.2023	0.7	Сосновский Максим Евгеньевич	Реализация веб-клиента.
05.12.2023	0.7.1	Сосновский Максим Евгеньевич	Проведение интеграционного тестирования. Устранения недочетов в логике основной деятельности системы.
06.12.2023	0.7.2	Сосновский Максим Евгеньевич	Незначительная оптимизация серверной логики.
10.12.2023	1.0	Сосновский Максим Евгеньевич	Релиз первой стабильной версии системы.

В данной таблице каждый этап разработки сопровождается описанием того, что было сделано, а также когда именно это изменение произошло. Такой подход обеспечивает четкое понимание процесса разработки и изменений, которые были внесены на каждом этапе. Это позволяет не только отслеживать прогресс, но и легко идентифицировать и исправлять возможные ошибки, а также улучшать продукт в будущем.

4.2. Терминология

Список терминов и определений, используемых для тестирования программного продукта:

- 1) Стратегия тестирования – план или подход, определяющий цели тестирования, методы, ресурсы, временные рамки и этапы проведения тестов программного обеспечения.
- 2) Функциональное тестирование – тестирование, направленное на проверку соответствия функций программного обеспечения его спецификациям и требованиям. Проверяется корректность выполнения заданных функций.
- 3) Модульное тестирование – тестирование отдельных компонентов или модулей программного обеспечения в изоляции от остальной части системы, обычно выполняемое разработчиками.
- 4) Интеграционное тестирование – тестирование, при котором отдельные модули программного обеспечения объединяются и тестируются как группа для выявления ошибок в их взаимодействии.
- 5) Клиент-серверная архитектура – архитектура программного обеспечения, в которой клиентские приложения или устройства взаимодействуют с серверами для получения ресурсов или услуг. Тестирование в такой архитектуре включает проверку взаимодействия между клиентскими и серверными компонентами.

4.3. Стратегия тестирования

Стратегия тестирования программного обеспечения методом "белого ящика" ориентирована на детальное изучение и анализ внутренней структуры кода. Это подразумевает глубокое погружение в логику работы программы, проверку правильности циклов, условных операторов и других элементов кода. Такой подход требует обеспечения выполнения каждой строки кода в процессе тестирования, что помогает максимизировать обнаружение скрытых ошибок. В рамках стратегии "белого ящика" тестировщики проверяют все возможные пути выполнения кода, что позволяет выявить уязвимые места в логике и работе программы. Особое внимание уделяется искусственному введению ошибок в данные или имитации их отсутствия, чтобы проверить, насколько устойчиво ПО работает при нестандартных ситуациях и как оно обрабатывает исключения. Также важным аспектом является тестирование на безопасность, которое включает в себя анализ кода на наличие уязвимостей, таких как переполнение буфера, SQL-инъекции и прочие потенциальные угрозы. Статический анализ, который проводится с помощью специализированных инструментов, позволяет выявлять ошибки и проблемные места без непосредственного выполнения кода. Наряду с этим, отслеживание потоков данных играет ключевую роль в тестировании "белого ящика". Анализируется, как данные перемещаются внутри системы: откуда они берутся, как изменяются в процессе и куда направляются в итоге. Все это в комплексе обеспечивает высокий уровень доверия к стабильности и надежности программного продукта перед его релизом.

4.4. Определение объектов тестирования

В процессе разработки информационных систем критически важным этапом является тестирование. Это систематический процесс проверки и оценки качества программного продукта с целью выявления и устранения ошибок, а также подтверждения соответствия разработанного продукта установленным требованиям и ожиданиям пользователя. Определение объектов тестирования лежит в основе этого процесса, поскольку именно на этом этапе устанавливаются компоненты и функциональные аспекты системы, подлежащие проверке.

При выявлении были выделены следующие объекты тестирования (Таблица 3.2)

Таблица 3.2

Определение объектов тестирования

Продолжение таблицы 3.2

Тесты	Объект тестирования
Виды тестирования	
Функциональное тестирование	1) Работа функции сбора истории браузера 2) Работа функции сбора информации об аппаратных комплектующих 3) Работа функции просмотра истории браузера 4) Работа функции просмотра информации об аппаратных комплектующих
Уровни тестирования	
Модульное тестирование	1) Тестирование сбора информации об истории браузера на стороне клиента 2) Тестирование сбора информации об аппаратных комплектующих на стороне клиента

Продолжение таблицы 3.2

	3) Тестирование работы центра опроса на сервере 4) Тестирование модуля авторизации и аутентификации для клиентов и администраторов 5) Тестирование работы модуля с функционалом администратора 6) Тестирование работы модуля взаимодействия с базой данных
Интеграционное тестирование	1) Тестирование общения сервера с подключенными клиентами 2) Тестирование сбора данных на клиентах по запросам с сервера 3) Тестирование запросов данных с сервера на веб-клиенте 4) Тестирование сбора данных на клиентах по команде веб-клиента

Определив объекты тестирования, мы обеспечили основу для последующих этапов тестирования, которые позволят не только выявить и устранить потенциальные проблемы, но и подтвердить, что разработанный модуль соответствует всем функциональным и техническим требованиям.

4.5. Архитектура тестируемой системы

Архитектура тестируемой системы представляет собой клиент-серверную архитектуру. Архитектура клиент-сервер является одной из фундаментальных основ современных информационных систем, привлекая своей способностью эффективно разделять обработку данных между клиентскими станциями и сервером.

Однако такая архитектура также налагает высокую зависимость от сервера: в случае его отказа весь доступ к функциям системы теряется, что делает критически важным поддержание высокого уровня его отказоустойчивости. Стоимость серверного оборудования может быть существенной, особенно учитывая необходимость обслуживания профессиональными специалистами. Кроме того, масштабирование сервера может оказаться сложной задачей, требующей значительных инвестиций, если потребности превышают возможности текущей конфигурации. Производительность системы в значительной степени зависит от скорости сети, поскольку все операции требуют сетевого обмена данными, что может стать узким местом в производительности, особенно при высоких нагрузках.

4.6. Описание процесса тестирования

В рамках обеспечения качества программного продукта, систематический подход к тестированию занимает центральное место в разработке ПО. Процесс тестирования необходим для верификации функциональности, производительности, и совместимости программного продукта со спецификациями и требованиями пользователя. Для того чтобы процесс тестирования происходил максимально эффективно была разработана таблица сценариев тестирования разработанного продукта. (Табл. 4.3)

Таблица 4.3

Сценарии тестирования разработанного продукта

Название сценария тестирования	Тестовый набор	Ожидаемый результат	Актуальный результат	Результат тестирования
Запрос на сбор истории браузеров	Подключенные к системе клиенты	Запросы будут отправлены подключенным клиентам. Результаты появятся в базе данных	Клиенты получили запрос. Записи с результатами появились в базе.	Тестирование пройдено успешно
Запрос на сбор информации об аппаратном обеспечении	Подключенные к системе клиенты	Запросы будут отправлены подключенным клиентам. Результаты появятся в базе данных	Клиенты получили запрос. Записи с результатами появились в базе.	Тестирование пройдено успешно
Разделение доступа к системе	Учетные данные, не имеющие права администратора. Учетные данные с правами администратора	Доступ к системе будет получен только данными с достаточным уровнем прав	Доступ к системе получен только при использовании данных с правами администратора	Тестирование пройдено успешно
Просмотр истории браузера клиента	Опрошенный ранее клиент	Будут получены адреса недавно посещенных страниц.	Получен набор последних посещенных клиентом страниц.	Тестирование пройдено успешно
Просмотр информации об аппаратном обеспечении	Опрошенный ранее клиент	Будут получены данные о компонентах и их состоянии в порядке убывания даты замера	Получены данные о компонентах в порядке от новых данных к старым	Тестирование пройдено успешно

Разработанная таблица сценариев тестирования демонстрирует планирование и структурирование тестов, включая определение тестового набора, ожидаемых и актуальных результатов, а также итоговых результатов тестирования. Это позволяет не только оценить качество разработанного продукта, но и выявить возможные отклонения и ошибки, требующие корректировки.

Таким образом, тестирование играет критическую роль в обеспечении надежности, безопасности и соответствия программного продукта ожиданиям пользователей и стандартам качества. Систематический подход к тестированию, включающий детально проработанные сценарии, способствует повышению качества конечного продукта и является неотъемлемой частью процесса

ЗАКЛЮЧЕНИЕ

В ходе данной работы была разработана система внедрения политик безопасности для компании "SecureGuard Solutions", что положительно сказалось над затратами на контроль безопасности на предприятии, увеличивая контроль над деятельностью сотрудников в рабочее время. Настоящий проект не только воплощает современные технологии и стандарты в области кибербезопасности, но и создает устойчивую основу для дальнейшего развития и улучшения. "PolicyEnforcer" представляет собой значимый шаг в обеспечении безопасности данных, и его внедрение в практику компаний будет способствовать укреплению их киберсреды.

Таким образом, разработка системы "PolicyEnforcer" подтверждает наши усилия в области кибербезопасности и служит важным вкладом в обеспечение безопасности информационных ресурсов в условиях современных технологических вызовов и угроз.

СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ

1. Sullivan, B., Liu, V. (2016). Web Application Security: A Beginner's Guide.
2. Skeet, J. (2019). C# in Depth.
3. Stefanov, S. (2016). React Up and Running: Building Web Applications.
4. Vespa, R. (2018). SignalR Real-time Application Cookbook.
5. Winand, M. (2016). SQL Performance Explained.
6. Richer, J., Sanso, A. (2015). OAuth 2.0 in Action.
7. Ruby, S., Thomas, D., Hansson, D. H. (2020). Agile Web Development with Rails.
8. Newman, S. (2015). Building Microservices.
9. Stuttard, D., Pinto, M. (2011). The Web Application Hacker's Handbook.
10. Microsoft Documentation. - [Электронный ресурс] URL: <https://learn.microsoft.com/ru-ru/dotnet/>

ПРИЛОЖЕНИЕ 1. ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ

Серверный модуль.

Класс Program

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using Newtonsoft.Json;
using PolicyEnforcer.ServerCore;
using PolicyEnforcer.ServerCore.Database.Context;
using PolicyEnforcer.ServerCore.Hubs;
using System.Reflection;

internal class Program
{
    private static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        builder.Services.AddSignalR(o => {
            o.EnableDetailedErrors = true;
            o.MaximumReceiveMessageSize = 100240;
        }).AddNewtonsoftJsonProtocol(opts =>
            opts.PayloadSerializerSettings.TypeNameHandling = TypeNameHandling.Auto);

        builder.Services.AddAuthorization();
        builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(opts =>
                opts.TokenValidationParameters = new TokenValidationParameters {
                    ValidateIssuer = true,
                    ValidIssuer = AuthHelper.Issuer,
                    ValidateAudience = true,
                    ValidAudience = AuthHelper.Audience,
                    ValidateLifetime = true,
                    IssuerSigningKey = AuthHelper.GetSymmetricSecurityKey(),
                    ValidateIssuerSigningKey = true
                });

        builder.Services.AddDbContext<PolicyEnforcerContext>();
        builder.Services.AddControllers();

        builder.Services.AddSwaggerGen(c => {
            c.SwaggerDoc("v1", new OpenApiInfo { Title = "PolicyEnforcerAPI", Version = "v1" });

            c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme {
                Description = @"JWT Authorization header using the Bearer scheme. \r\n\r\n
                    Enter 'Bearer' [space] and then your token in the text input below.
                    \r\n\r\nExample: 'Bearer 12345abcdef'",
                Name = "Authorization",
                In = ParameterLocation.Header,
                Type = SecuritySchemeType.ApiKey,
                Scheme = "Bearer"
            });

            c.AddSecurityRequirement(new OpenApiSecurityRequirement() {
                {
                    new OpenApiSecurityScheme {
                        Reference = new OpenApiReference {
                            Type = ReferenceType.SecurityScheme,
                            Id = "Bearer"
                        },
                        Scheme = "oauth2",
                        Name = "Bearer",
                        In = ParameterLocation.Header,
                    },
                    new List<string>()
                }
            });

            // Set the comments path for the Swagger JSON and UI.
            var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
            var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
            c.IncludeXmlComments(xmlPath);
        });
    }
}
```

```

var configBuilder = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json");

var config = configBuilder.Build();
var url = config.GetValue<string>("WorkingURL");

builder.WebHost.UseUrls(url);
var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(options =>
    {
        options.SwaggerEndpoint("/swagger/v1/swagger.json", "v1");
    });
}
app.UseDefaultFiles();
app.UseStaticFiles();

app.UseCors(x => x
    .AllowAnyMethod()
    .AllowAnyHeader()
    .SetIsOriginAllowed(origin => true) // allow any origin
    .AllowCredentials());

app.MapControllers();
app.MapHub<DataCollectionHub>("/data");

app.Run();
}
}

```

Класс Dictionaries

```

namespace PolicyEnforcer.ServerCore
{
    public class Dictionaries
    {
        public static Dictionary<int, string> Roles => new()
        {
            { 0, "user" },
            { 1, "admin" }
        };
    }
}

```

Класс AuthHelper

```

using Microsoft.IdentityModel.Tokens;
using System.Security.Cryptography;
using System.Text;

namespace PolicyEnforcer.ServerCore
{
    public class AuthHelper
    {
        public const string Issuer = "PolicyEnforcerServer";
        public const string Audience = "PolicyEnforcerClient";
        const string key = "chuchikmuchik731";
        public static SymmetricSecurityKey GetSymmetricSecurityKey() =>
            new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));

        public static byte[] HashString(string s)
        {
            byte[] bytes = Encoding.UTF8.GetBytes(s);
            return MD5.HashData(bytes);
        }
    }
}

```

Класс DataCollectionHub

```

using Microsoft.AspNetCore.SignalR;
using Newtonsoft.Json;
using PolicyEnforcer.ServerCore.Database.Context;
using PolicyEnforcer.ServerCore.Database.Models;
using PolicyEnforcer.ServerCore.DTO;

```

```

namespace PolicyEnforcer.ServerCore.Hubs
{
    public static class UserHandler
    {
        {
            public static HashSet<string> ConnectedIDs { get; set; } = new();
        }
    }

    public class DataCollectionHub : Hub
    {
        private PolicyEnforcerContext _context;
        public DataCollectionHub(PolicyEnforcerContext dbContext)
        {
            _context = dbContext;
        }
        public override Task OnConnectedAsync()
        {
            UserHandler.ConnectedIDs.Add(Context.ConnectionId);
            return base.OnConnectedAsync();
        }

        public override Task OnDisconnectedAsync(Exception? exception)
        {
            UserHandler.ConnectedIDs.Remove(Context.ConnectionId);

            return base.OnDisconnectedAsync(exception);
        }

        public async Task ReturnHardwareReadings(List<string> readings)
        {
            foreach (var pieceRaw in readings)
            {
                var piece = JsonConvert.DeserializeObject<HardwarePiece>(pieceRaw);
                _context.HardwareInfos.Add(new()
                {
                    DateMeasured = piece.TimeMeasured,
                    MeasurementId = Guid.NewGuid(),
                    InstanceName = piece.InstanceName,
                    Load = piece.Load,
                    Temperature = piece.Temperature,
                    UserId = piece.UserId
                });
            }

            await _context.SaveChangesAsync();
        }

        public async Task ReturnBrowserHistory(List<string> readings)
        {
            foreach (var pieceRaw in readings)
            {
                var piece = JsonConvert.DeserializeObject<BrowserHistory>(pieceRaw);
                _context.BrowserHistories.Add(new()
                {
                    DateVisited = piece.DateVisited,
                    Id = Guid.NewGuid(),
                    UserId = piece.UserId,
                    BrowserName = piece.BrowserName,
                    Url = piece.Url,
                });
            }

            await _context.SaveChangesAsync();
        }
    }
}

```

Класс BrowserHistoryDTO

```

namespace PolicyEnforcer.ServerCore.DTO
{
    public class BrowserHistoryDTO
    {
        {
            public string Url { get; set; }
            public DateTime DateVisited { get; set; }
            public string BrowserName { get; set; }
        }
    }
}

```

Класс HardwareInfoDTO

```
namespace PolicyEnforcer.ServerCore.DTO
{
    public class HardwareInfoDTO
    {
        public string InstanceName { get; set; }
        public double? Temperature { get; set; }
        public double? Load { get; set; }
        public DateTime DateMeasured { get; set; }
    }
}
```

Класс HardwarePiece

```
using PolicyEnforcer.Interfaces;

namespace PolicyEnforcer.ServerCore.DTO
{
    public class HardwarePiece : IHardwarePiece
    {
        public string InstanceName { get; set; }
        public Guid UserID { get; set; }
        public float? Temperature { get; set; }
        public float? Load { get; set; }
        public DateTime TimeMeasured { get; set; }
    }
}
```

Класс TokenDTO

```
namespace PolicyEnforcer.ServerCore.DTO
{
    public class TokenDTO
    {
        public string Token { get; set; }
        public Guid UserID { get; set; }
    }
}
```

Класс UserDTO

```
namespace PolicyEnforcer.ServerCore.DTO
{
    public class UserDTO
    {
        /// <summary>
        /// Логин
        /// </summary>
        public string Login { get; set; }

        /// <summary>
        /// Пароль
        /// </summary>
        public string Password { get; set; }
    }
}
```

Класс UserResponseDTO

```
namespace PolicyEnforcer.ServerCore.DTO
{
    public class UserResponseDTO
    {
        public Guid UserId { get; set; }
        public string Login { get; set; }
        public int AccessLevel { get; set; }
    }
}
```

Класс PolicyEnforcerContext

```
using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using PolicyEnforcer.ServerCore.Database.Models;

namespace PolicyEnforcer.ServerCore.Database.Context;
```



```

public partial class PolicyEnforcerContext : DbContext
{
    public PolicyEnforcerContext()
    {
    }

    public PolicyEnforcerContext(DbContextOptions<PolicyEnforcerContext> options)
        : base(options)
    {
    }

    public virtual DbSet<BrowserHistory> BrowserHistories { get; set; }

    public virtual DbSet<HardwareInfo> HardwareInfos { get; set; }

    public virtual DbSet<User> Users { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        #warning To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid
        scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148.
        For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723263.
        => optionsBuilder.UseSqlServer("Server=.\HOASERVER_DEV;Database=PolicyEnforcer;User
        Id=sa;Password=chuchikmuchik;TrustServerCertificate=true");

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<BrowserHistory>(entity =>
            {
                entity.ToTable("BrowserHistory");

                entity.Property(e => e.Id)
                    .ValueGeneratedNever()
                    .HasColumnName("ID");
                entity.Property(e => e.BrowserName).HasMaxLength(30);
                entity.Property(e => e.DateVisited).HasColumnType("datetime");
                entity.Property(e => e.Url)
                    .HasColumnType("text")
                    .HasColumnName("URL");
                entity.Property(e => e.UserId).HasColumnName("UserID");

                entity.HasOne(d => d.User).WithMany(p => p.BrowserHistories)
                    .HasForeignKey(d => d.UserId)
                    .HasConstraintName("FK_BrowserHistory_Users");
            });

            modelBuilder.Entity<HardwareInfo>(entity =>
            {
                entity.HasKey(e => e.MeasurementId);

                entity.ToTable("HardwareInfo");

                entity.Property(e => e.MeasurementId)
                    .ValueGeneratedNever()
                    .HasColumnName("MeasurementID");
                entity.Property(e => e.DateMeasured).HasColumnType("datetime");
                entity.Property(e => e.InstanceName).HasMaxLength(70);
                entity.Property(e => e.UserId).HasColumnName("UserID");

                entity.HasOne(d => d.User).WithMany(p => p.HardwareInfos)
                    .HasForeignKey(d => d.UserId)
                    .HasConstraintName("FK_HardwareInfo_Users");
            });

            modelBuilder.Entity<User>(entity =>
            {
                entity.Property(e => e.UserId)
                    .ValueGeneratedNever()
                    .HasColumnName("UserID");
                entity.Property(e => e.Login).HasMaxLength(40);
                entity.Property(e => e.Password)
                    .HasMaxLength(16)
                    .IsFixedLength();
            });

            OnModelCreatingPartial(modelBuilder);
        }
    }

```

```

    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}

```

Класс BrowserHistory

```

using System;
using System.Collections.Generic;

namespace PolicyEnforcer.ServerCore.Database.Models;

public partial class BrowserHistory
{
    public Guid Id { get; set; }

    public Guid UserId { get; set; }

    public string Url { get; set; } = null!;

    public DateTime DateVisited { get; set; }

    public string BrowserName { get; set; } = null!;

    public virtual User User { get; set; } = null!;
}

```

Класс HardwareInfo

```

namespace PolicyEnforcer.ServerCore.Database.Models;

public partial class HardwareInfo
{
    public string InstanceName { get; set; } = null!;

    public double? Temperature { get; set; }

    public double? Load { get; set; }

    public DateTime DateMeasured { get; set; }

    public Guid UserId { get; set; }

    public Guid MeasurementId { get; set; }

    public virtual User User { get; set; } = null!;
}

```

Класс User

```

namespace PolicyEnforcer.ServerCore.Database.Models;

public partial class User
{
    public Guid UserId { get; set; }

    public string Login { get; set; } = null!;

    public byte[] Password { get; set; } = null!;

    public int AccessLevel { get; set; }

    public virtual ICollection<BrowserHistory> BrowserHistories { get; set; } = new List<BrowserHistory>();

    public virtual ICollection<HardwareInfo> HardwareInfos { get; set; } = new List<HardwareInfo>();
}

```

Класс AdminContrtoller

```

using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.SignalR;
using PolicyEnforcer.ServerCore.Database.Context;
using PolicyEnforcer.ServerCore.Database.Models;
using PolicyEnforcer.ServerCore.DTO;
using PolicyEnforcer.ServerCore.Hubs;

namespace PolicyEnforcer.ServerCore.Controllers
{

```

```

[Route("api/[controller]")]
[ApiController]
[Authorize(Roles = "admin")]
public class AdminController : ControllerBase
{
    private IHubContext<DataCollectionHub> _dataHub;
    private PolicyEnforcerContext _context;
    public AdminController(IHubContext<DataCollectionHub> dataHub, PolicyEnforcerContext context)
    {
        _dataHub = dataHub;
        _context = context;
    }

    /// <summary>
    /// Изменяет уровень доступа указанного пользователя
    /// </summary>
    /// <param name="userID">идентификатор целевого пользователя</param>
    /// <param name="newAccessLevel">новый уровень доступа</param>
    /// <returns></returns>
    [HttpPost("changeaccesslevel")]
    public IActionResult ChangeUserRole(
        [FromForm] Guid userID,
        [FromForm] int newAccessLevel)
    {
        var target = _context.Users.FirstOrDefault(x => x.UserId == userID);
        if (target is null || newAccessLevel < 0 || newAccessLevel > 1)
        {
            return BadRequest();
        }

        target.AccessLevel = newAccessLevel;
        _context.Users.Update(target);
        _context.SaveChanges();

        return Ok();
    }

    /// <summary>
    /// Возвращает историю посещений сайтов для клиента
    /// </summary>
    /// <param name="userID">клиент-цель</param>
    /// <returns></returns>
    [HttpGet("getbrowserhistory/{userID}")]
    public async Task<IActionResult> GetBrowserHistory(Guid userID)
    {
        var target = _context.BrowserHistories.Where(x => x.UserId == userID).OrderByDescending(x => x.DateVisited).Take(150);

        // Создание конфигурации сопоставления
        var config = new MapperConfiguration(cfg => cfg.CreateMap<BrowserHistory, BrowserHistoryDTO>());
        // Настройка AutoMapper
        var mapper = new Mapper(config);
        // сопоставление
        var result = mapper.Map<List<BrowserHistoryDTO>>(target.ToList());

        return Ok(result);
    }

    /// <summary>
    /// Возвращает список всех пользователей
    /// </summary>
    /// <returns></returns>
    [HttpGet("getusers")]
    public IActionResult GetUsers()
    {
        // Создание конфигурации сопоставления
        var config = new MapperConfiguration(cfg => cfg.CreateMap<User, UserResponseDTO>());
        // Настройка AutoMapper
        var mapper = new Mapper(config);
        // сопоставление
        var users = mapper.Map<List<UserResponseDTO>>(_context.Users.ToList());

        return Ok(users);
    }

    [HttpGet("gethardwarereadings/{userID}")]
    public async Task<IActionResult> GetHardwareReadings(Guid userID)
    {

```

```

var target = _context.HardwareInfos.Where(x => x.UserId == userID).OrderByDescending(x => x.DateMeasured);

// Создание конфигурации сопоставления
var config = new MapperConfiguration(cfg => cfg.CreateMap<HardwareInfo, HardwareInfoDTO>());
// Настройка AutoMapper
var mapper = new Mapper(config);
// сопоставление
var result = mapper.Map<List<HardwareInfoDTO>>(target.ToList());

return Ok(result);
}

/// <summary>
/// Генерирует запросы на сбор данных об истории посещений сайтов подключенных клиентов
/// </summary>
[HttpGet("requestbrowserhistory")]
public async Task<ActionResult> RequestBrowserHistory()
{
    await _dataHub.Clients.All.SendAsync("GetBrowserHistory", 10);
    return Ok();
}

/// <summary>
/// Генерирует запросы на сбор данных об аппаратных компонентах подключенных клиентов
/// </summary>
[HttpGet("requesthardware readings")]
public async Task<ActionResult> RequestHardwareInfo()
{
    await _dataHub.Clients.All.SendAsync("GetHardwareInfo");
    return Ok();
}
}
}

```

Класс UsersController

```

using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using PolicyEnforcer.ServerCore.Database.Context;
using PolicyEnforcer.ServerCore.Database.Models;
using PolicyEnforcer.ServerCore.DTO;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;

namespace PolicyEnforcer.ServerCore.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UsersController : ControllerBase
    {
        private readonly PolicyEnforcerContext _context;
        public UsersController(PolicyEnforcerContext context)
        {
            _context = context;
        }

        /// <summary>
        /// Поиск пользователя по идентификатору
        /// </summary>
        /// <param name="userID">идентификатор пользователя</param>
        /// <returns></returns>
        [Authorize]
        [HttpGet("{userID}")]
        public IActionResult GetUser(Guid userID)
        {
            var user = _context.Users.FirstOrDefault(x => x.UserId == userID);

            if (user is null)
            {
                return BadRequest();
            }

            // Создание конфигурации сопоставления
            var config = new MapperConfiguration(cfg => cfg.CreateMap<User, UserResponseDTO>());
            // Настройка AutoMapper
            var mapper = new Mapper(config);

```

```

// сопоставление
var result = mapper.Map<UserResponseDTO>(user);

return Ok(result);
}

/// <summary>
/// Авторизация в системе
/// </summary>
/// <param name="loginInfo">Данные для входа</param>
/// <returns>токен</returns>
[HttpPost("login")]
public IActionResult Login(
    [FromBody] UserDTO loginInfo)
{
    var hashedPassword = AuthHelper.HashString(loginInfo.Password);
    var user = _context.Users.FirstOrDefault(x => x.Login == loginInfo.Login && x.Password == hashedPassword);

    if (user is null)
    {
        return Unauthorized();
    }

    var claims = new List<Claim> {
        new Claim(ClaimTypes.Name, user.Login),
        new Claim(ClaimTypes.Role, Dictionaries.Roles[user.AccessLevel])
    };

    var jwt = new JwtSecurityToken(
        issuer: AuthHelper.Issuer,
        audience: AuthHelper.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddDays(1),
        signingCredentials: new SigningCredentials(AuthHelper.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));

    return Ok(new TokenDTO { Token = new JwtSecurityTokenHandler().WriteToken(jwt), UserID = user.UserId });
}

/// <summary>
/// Регистрация в системе
/// </summary>
/// <param name="loginInfo">данные для входа</param>
/// <returns>токен</returns>
[HttpPost("register")]
public IActionResult Register(
    [FromBody] UserDTO loginInfo)
{
    var user = _context.Users.FirstOrDefault(x => x.Login == loginInfo.Login);

    if (user is not null)
    {
        return Conflict();
    }

    user = new User()
    {
        Login = loginInfo.Login,
        AccessLevel = 0,
        Password = AuthHelper.HashString(loginInfo.Password),
        UserId = Guid.NewGuid()
    };

    _context.Users.Add(user);
    _context.SaveChanges();

    var claims = new List<Claim> { new Claim(ClaimTypes.Name, user.Login), new Claim(ClaimTypes.Role,
user.AccessLevel.ToString() ) };
    var jwt = new JwtSecurityToken(
        issuer: AuthHelper.Issuer,
        audience: AuthHelper.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddDays(1),
        signingCredentials: new SigningCredentials(AuthHelper.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));

    return Ok(new TokenDTO { Token = new JwtSecurityTokenHandler().WriteToken(jwt), UserID = user.UserId });
}
}
}

```

Сборка PolicyEnforcer.Interfaces
Класс IHardwarePiece

```
using System;

namespace PolicyEnforcer.Interfaces
{
    public interface IHardwarePiece
    {
        string InstanceName { get; set; }
        Guid UserID { get; set; }
        float? Temperature { get; set; }
        float? Load { get; set; }
        DateTime TimeMeasured { get; set; }
    }
}
```

Класс IVisitedURL

```
using System;

namespace PolicyEnforcer.Interfaces
{
    public interface IVisitedURL
    {
        string Url { get; set; }
        Guid UserID { get; set; }
        string BrowserName { get; set; }
        DateTime DateVisited { get; set; }
    }
}
```

Клиентский модуль
Класс Program

```
using PolicyEnforcer.Service.Services;
using PolicyEnforcer.Service.Services.Interfaces;
using System.Runtime.InteropServices;

internal class Program
{
    private static void Main(string[] args)
    {
        var configBuilder = new ConfigurationBuilder()
            .AddJsonFile("appsettings.json");

        var configuration = configBuilder.Build();

        IHost host = Host.CreateDefaultBuilder(args)
            .ConfigureServices(services =>
            {
                services.AddHostedService<ServerConnectionService>();
                services.AddTransient<IHistoryCollectionService, HistoryCollectionService>();
                services.AddTransient<IHardwareMonitoringService, HardwareMonitoringService>();
                services.Configure<PolicyEnforcer.Service.POCO.LoginInfo>(configuration.GetSection("LoginInfo"));
            })
            .ConfigureLogging((context, logging) =>
            {
                logging.ClearProviders();
                logging.AddConfiguration(context.Configuration.GetSection("Logging"));

                // для отладки
                if (context.HostingEnvironment.IsDevelopment())
                {
                    AllocConsole();
                    logging.AddConsole();
                    logging.AddDebug();
                }
            })
            .Build();

        host.Run();
    }
}
```

```
[DllImport("kernel32.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
```

```

static extern bool AllocConsole();
}

Класс ServerConnectionService

using Microsoft.AspNetCore.SignalR.Client;
using Microsoft.Extensions.Options;
using Newtonsoft.Json;
using PolicyEnforcer.Service.Configuration;
using PolicyEnforcer.Service.Models;
using PolicyEnforcer.Service.Services.Interfaces;
using System.Configuration;
using System.Net.Http.Json;
using System.Text;

namespace PolicyEnforcer.Service.Services
{
    public class ServerConnectionService : IServerConnectionService, IHostedService
    {
        private readonly IHardwareMonitoringService _monitoringService;
        private readonly IHistoryCollectionService _historyCollectionService;
        private readonly ILogger<ServerConnectionService> _logger;
        private readonly HttpClient _httpClient;
        private Guid _userID;
        private readonly POCO.LoginInfo _loginInfo;

        HubConnection _connection { get; set; }

        public ServerConnectionService(IHistoryCollectionService historyCollectionsvc, IHardwareMonitoringService monitoringsvc,
            ILogger<ServerConnectionService> logger, IOption<POCO.LoginInfo> loginConfig)
        {
            _monitoringService = monitoringsvc;
            _historyCollectionService = historyCollectionsvc;
            _logger = logger;
            _loginInfo = loginConfig.Value;

            var clientHandler = new HttpClientHandler()
            {
                ServerCertificateCustomValidationCallback = (message, cert, chain, errors) => { return true; }
            };

            _httpClient = new HttpClient(clientHandler);
        }

        public async Task StartAsync(CancellationToken cancellationToken)
        {
            try
            {
                await ConfigureConnection();

                await Task.CompletedTask;
            }
            catch (Exception ex)
            {
                _logger.LogError(ex.Message);
            }
        }

        /// <summary>
        /// Конфигурация подключения к серверу
        /// </summary>
        /// <returns></returns>
        private async Task ConfigureConnection()
        {
            var configFile =
            System.Configuration.ConfigurationManager.OpenExeConfiguration(System.Configuration.ConfigurationUserLevel.None);
            var settings = configFile.AppSettings.Settings;

            var loginInfo = await Login(settings);

            var url = settings["WorkingURL"]; // Адрес сервера хранится в конфиге

            _connection = new HubConnectionBuilder()
                .WithUrl($"{url}/data",
                    options =>
                    {
                        options.UseDefaultCredentials = true;
                        options.HttpMessageHandlerFactory = (msg) =>

```

```

    {
        if (msg is HttpClientHandler clientHandler)
        {
            // bypass SSL certificate
            clientHandler.ServerCertificateCustomValidationCallback +=
                (sender, certificate, chain, sslPolicyErrors) => { return true; };
        }

        return msg;
    };
    options.AccessTokenProvider = () => Task.FromResult("Bearer " + loginInfo.Token);
})
.WithAutomaticReconnect()
.AddNewtonsoftJsonProtocol(opts =>
    opts.PayloadSerializerSettings.TypeNameHandling = Newtonsoft.Json.TypeNameHandling.Auto)
.Build();

await _connection.StartAsync();
_logger.LogInformation($"Connection established at: {DateTimeOffset.Now}");

_connection.On("GetHardwareInfo", GetTemps);
_connection.On<int>("GetBrowserHistory", GetBrowserHistory);

configFile.Save(ConfigurationSaveMode.Modified);
System.Configuration.ConfigurationManager.RefreshSection(configFile.AppSettings.SectionInformation.Name);
}

/// <summary>
/// Авторизация в системе
/// </summary>
/// <param name="settings">конфиг</param>
/// <returns></returns>
private async Task<LoginInfo> Login(KeyValueConfigurationCollection? settings)
{
    var filepath = Path.Combine(Environment.CurrentDirectory, "config.xml");
    if (!File.Exists(filepath))
    {
        var loginInfo = await Register(settings);
        return loginInfo;
    }

    var config = XMLHelper.FromXmlFile<LoginConfig>(filepath);

    var login = config.Login;
    var password = config.Password;

    var token = await RenewToken(new UserDTO() { login = login, password = password });

    return new LoginInfo { Username = login, Password = password, Token = token };
}

/// <summary>
/// Обновление токена
/// </summary>
/// <param name="userinfo">учетные данные</param>
/// <returns></returns>
private async Task<string> RenewToken(UserDTO userinfo)
{
    var message = new HttpRequestMessage
    {
        Method = HttpMethod.Post,
        Content = JsonContent.Create(userinfo),
        RequestUri = new UriBuilder("https://26.85.180.83:6969/api/Users/login").Uri
    };

    var response = _httpClient.Send(message);
    response.EnsureSuccessStatusCode();

    var token = await response.Content.ReadFromJsonAsync<TokenDTO>();
    this._userID = token.UserID;

    return token.Token;
}

/// <summary>
/// Регистрация в системе

```



```

/// </summary>
/// <param name="settings">конфиг</param>
/// <returns></returns>
private async Task<LoginInfo> Register(KeyValueCollection? settings)
{
    var loginInfo = LoginInfo.Generate();

    var content = new UserDTO() { login = loginInfo.Username, password = loginInfo.Password };

    string json = JsonConvert.SerializeObject(content);
    var httpContent = new StringContent(json, Encoding.UTF8, "application/json");

    var message = new HttpRequestMessage
    {
        Method = HttpMethod.Post,
        Content = httpContent,
        RequestUri = new UriBuilder("https://26.85.180.83:6969/api/Users/register").Uri
    };

    var response = _httpClient.Send(message);

    response.EnsureSuccessStatusCode();

    var token = await response.Content.ReadFromJsonAsync<TokenDTO>();

    this._userID = token.UserID;
    loginInfo.Token = token.Token;

    var loginConfig = new LoginConfig
    {
        Login = loginInfo.Username,
        Password = loginInfo.Password,
        Token = token.Token,
    };

    var filepath = Path.Combine(Environment.CurrentDirectory, "config.xml");
    XMLHelper.ToXmlFile(loginConfig, filepath);

    return loginInfo;
}

public async Task StopAsync(Cancellation_token cancellation_token)
{
    _logger.LogInformation($"Service stopped at {DateTime.Now}");
    await Task.CompletedTask;
}

/// <summary>
/// Опрс АО
/// </summary>
public async void GetTemps()
{
    _logger.LogInformation($"Received hardware poll request at {DateTimeOffset.Now}");
    var readings = _monitoringService.PollHardware(_userID);

    await _connection.InvokeAsync("ReturnHardwareReadings", readings);
}

/// <summary>
/// Сбор истории браузера
/// </summary>
/// <param name="batchSize">количество записей в пакете данных</param>
public async void GetBrowserHistory(int batchSize = 10)
{
    _logger.LogInformation($"Received history collection request at {DateTimeOffset.Now}");
    var readings = _historyCollectionService.GetBrowsersHistory(DateTime.Now.AddDays(-2), _userID);

    while (readings.Count > 0)
    {
        var ceiling = readings.Count > batchSize ? batchSize : readings.Count;
        _connection.InvokeAsync("ReturnBrowserHistory", readings.GetRange(0, ceiling));
        readings.RemoveRange(0, ceiling);
    }
}
}
}

```

Класс HistoryCollectionService

```

using Newtonsoft.Json;
using PolicyEnforcer.Service.Extensions;
using PolicyEnforcer.Service.HistoryCollection;
using PolicyEnforcer.Service.Services.Interfaces;
using System.Data.SQLite;

namespace PolicyEnforcer.Service.Services
{
    public class HistoryCollectionService : IHistoryCollectionService
    {
        private static string AppDataLocal => Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
        private static string AppDataRoaming => Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);

        /// <summary>
        /// Собирает историю браузеров
        /// </summary>
        /// <param name="from">дата, после которой отбираются записи</param>
        /// <param name="userID">идентификатор пользователя</param>
        /// <returns>Сериализованный результат</returns>
        public List<string> GetBrowsersHistory(DateTime from, Guid userID)
        {
            var conStrings = GetHistoryFiles();

            var result = new List<string>();
            foreach (var con in conStrings)
            {
                string tempFilePath = Path.Combine(Environment.CurrentDirectory, DateTime.Now.Ticks.ToString());
                File.Copy(con.DBPath, tempFilePath);

                try
                {
                    using var connection = new SQLiteConnection($"Data Source={tempFilePath}", true);
                    using SQLiteCommand command = connection.CreateCommand();

                    connection.Open();

                    const long secondsBetween19701601 = 11644473600;
                    var timestamp = ((DateTimeOffset)from).ToUnixTimeSeconds() + secondsBetween19701601;
                    command.CommandText = $"select * from urls where last_visit_time >= {timestamp * 1000000}";

                    using (SQLiteDataReader reader = command.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            // Структура БД истории посещений в chromium-браузерах унифицирована
                            var url = new VisitedURL(con.BrowserName, reader["url"].ToString(), reader["last_visit_time"].ToString(), userID);
                            result.Add(JsonConvert.SerializeObject(url));
                        }
                    }
                    connection.Close();
                }
                finally
                {
                    SQLiteConnection.ClearAllPools();
                    GC.Collect();
                    GC.WaitForPendingFinalizers();
                    File.Delete(tempFilePath);
                }
            }

            return result;
        }

        /// <summary>
        /// Возвращает лист с моделями браузеров
        /// </summary>
        /// <returns></returns>
        private List<BrowserModel> GetHistoryFiles()
        {
            var result = new List<BrowserModel>();

            result.AddBrowser(AppDataLocal, "Google\\Chrome\\User Data\\Default\\History", "Google Chrome");
            result.AddBrowser(AppDataRoaming, "Opera Software\\Opera GX Stable\\History", "Opera GX");
            result.AddBrowser(AppDataRoaming, "Opera Software\\Opera Stable\\History", "Opera");
            result.AddBrowser(AppDataLocal, "Microsoft\\Edge\\User Data\\Default\\History", "Microsoft Edge");
        }
    }
}

```

```

        return result;
    }
}

```

Класс HardwareMonitoringService

```

using LibreHardwareMonitor.Hardware;
using Newtonsoft.Json;
using PolicyEnforcer.Service.HardwareMonitoring;
using PolicyEnforcer.Service.Services.Interfaces;

namespace PolicyEnforcer.Service.Services
{
    public class HardwareMonitoringService : IHardwareMonitoringService
    {
        private Computer computer;

        public HardwareMonitoringService()
        {
            UpdateComputerInfo();
        }

        /// <summary>
        /// Собирает ответ серверу
        /// </summary>
        /// <param name="userID"></param>
        /// <returns></returns>
        public List<string> PollHardware(Guid userID)
        {
            var result = new List<string>();

            foreach (var hw in computer.Hardware)
            {
                result.AddRange(GetHardwareReadings(hw, userID));
            }

            return result;
        }

        /// <summary>
        /// Опрашивает сенсоры на АО
        /// </summary>
        /// <param name="hw">единица АО</param>
        /// <param name="userID">идентификатор пользователя</param>
        /// <returns>сериализованный результат</returns>
        private List<string> GetHardwareReadings(IHardware hw, Guid userID)
        {
            var result = new List<string>();

            var hwPiece = new HardwarePiece { InstanceName = hw.Name, UserID = userID };
            foreach (var sensor in hw.Sensors)
            {
                if (sensor.SensorType == SensorType.Temperature)
                {
                    hwPiece.Temperature = sensor.Value;
                }
                if (sensor.SensorType == SensorType.Load)
                {
                    hwPiece.Load = sensor.Value;
                }
            }

            hwPiece.TimeMeasured = DateTime.Now;

            result.Add(JsonConvert.SerializeObject(hwPiece));

            foreach (var childHw in hw.SubHardware)
            {
                result.AddRange(GetHardwareReadings(childHw, userID));
            }

            return result;
        }

        /// <summary>
        /// Инициализация
        /// </summary>
        private void UpdateComputerInfo()

```

```

    {
        computer = new Computer
        {
            IsCpuEnabled = true,
            IsGpuEnabled = true,
            IsMemoryEnabled = true,
            IsMotherboardEnabled = true,
            IsStorageEnabled = true,
        };

        var visitor = new UpdateVisitor();

        computer.Open();
        computer.Accept(visitor);
    }
}

```

Класс LoginInfo

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PolicyEnforcer.Service.POCO
{
    public class LoginInfo
    {
        public string Login { get; set; }
        public string Password { get; set; }
        public string Token { get; set; }
        public string UserID { get; set; }
    }
}

```

Класс UserDTO

```

namespace PolicyEnforcer.Service.Models
{
    public class UserDTO
    {
        public string login { get; set; }
        public string password { get; set; }
    }
}

```

Класс LoginInfo

```

using PolicyEnforcer.Service.Logging;

namespace PolicyEnforcer.Service.Models
{
    internal class LoginInfo
    {
        internal static LoginInfo Generate() => new()
        {
            Username = string.Format("Client-{0}-{1}", Environment.MachineName, Environment.UserName),
            Password = LoggingHelper.GeneratePassword(10),
        };

        internal string Username { get; set; }
        internal string Password { get; set; }
        public string? Token { get; internal set; }
    }
}

```

Класс TokenDTO

```

namespace PolicyEnforcer.Service.Models
{
    public class TokenDTO
    {
        public string Token { get; set; }
        public Guid UserID { get; set; }
    }
}

```

Класс LoggingHelper

```
namespace PolicyEnforcer.Service.Logging
{
    public static class LoggingHelper
    {
        public static string GeneratePassword(int length)
        {
            string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+";
            var random = new Random();
            return new string(Enumerable.Repeat(chars, length)
                .Select(s => s[random.Next(s.Length)]).ToArray());
        }
    }
}
```

Класс VisitedURL

```
using PolicyEnforcer.Interfaces;

namespace PolicyEnforcer.Service.HistoryCollection
{
    public class VisitedURL : IVisitedURL
    {
        public VisitedURL(string browserName, string url, string date, Guid userid)
        {
            BrowserName = browserName;
            UserID = userid;
            Url = url;

            // Chromium-браузеры сохраняют дату в формате Unix, с отсчетом от 1601 года
            DateTime dateVisited = new(1601, 1, 1, 0, 0, 0, 0, DateTimeKind.Utc);
            DateVisited = dateVisited.AddSeconds(Double.Parse(date) / 1000000).ToLocalTime();
        }

        public string Url { get; set; }
        public Guid UserID { get; set; }
        public string BrowserName { get; set; }
        public DateTime DateVisited { get; set; }

        public override string ToString()
        {
            return $"Browser: {BrowserName} URL: {Url} Date: {DateVisited}";
        }
    }
}
```

Класс BrowserModel

```
namespace PolicyEnforcer.Service.HistoryCollection
{
    public class BrowserModel
    {
        public string BrowserName { get; set; }
        public string DBPath { get; set; }
    }
}
```

Класс UpdateVisitor

```
using LibreHardwareMonitor.Hardware;

namespace PolicyEnforcer.Service.HardwareMonitoring
{
    public class UpdateVisitor : IVisitor
    {
        public void VisitComputer(IComputer computer)
        {
            computer.Traverse(this);
        }

        public void VisitHardware(IHardware hardware)
        {
            hardware.Update();
            foreach (IHardware subHardware in hardware.SubHardware) subHardware.Accept(this);
        }
    }
}
```

```

        public void VisitParameter(IParameter parameter) { }
        public void VisitSensor(ISensor sensor) { }
    }
}

```

Класс HardwarePiece

```

using PolicyEnforcer.Interfaces;

namespace PolicyEnforcer.Service.HardwareMonitoring
{
    internal class HardwarePiece : IHardwarePiece
    {
        public string InstanceName { get; set; }
        public Guid UserID { get; set; }
        public float? Temperature { get; set; }
        public float? Load { get; set; }
        public DateTime TimeMeasured { get; set; }
    }
}

```

Класс ListExtensions

```

using PolicyEnforcer.Service.HistoryCollection;

namespace PolicyEnforcer.Service.Extensions
{
    public static class ListExtensions
    {
        public static void AddBrowser(this List<BrowserModel> arg, string appData, string historyPath, string browserName)
        {
            string path = Path.Combine(appData, historyPath);
            if (File.Exists(path))
            {
                arg.Add(new BrowserModel { BrowserName = browserName, DBPath = path });
            }
        }
    }
}

```

Класс XMLHelper

```

using System.Text;
using System.Xml.Serialization;
using System.Xml;

namespace PolicyEnforcer.Service.Configuration
{
    internal static class XMLHelper
    {
        public static bool NewLineOnAttributes { get; set; }
        /// <summary>
        /// Serializes an object to an XML string, using the specified namespaces.
        /// </summary>
        public static string ToXml(object obj, XmlSerializerNamespaces ns)
        {
            Type T = obj.GetType();

            var xs = new XmlSerializer(T);
            var ws = new XmlWriterSettings { Indent = true, NewLineOnAttributes = NewLineOnAttributes, OmitXmlDeclaration = true };

            var sb = new StringBuilder();
            using (XmlWriter writer = XmlWriter.Create(sb, ws))
            {
                xs.Serialize(writer, obj, ns);
            }
            return sb.ToString();
        }
        /// <summary>
        /// Serializes an object to an XML string.
        /// </summary>
        public static string ToXml(object obj)
        {
            var ns = new XmlSerializerNamespaces();
            ns.Add("", "");
            return ToXml(obj, ns);
        }
    }
}

```

```

    }

    /// <summary>
    /// Deserializes an object from an XML string.
    /// </summary>
    public static T FromXml<T>(string xml)
    {
        XmlSerializer xs = new XmlSerializer(typeof(T));
        using (StringReader sr = new StringReader(xml))
        {
            return (T)xs.Deserialize(sr);
        }
    }

    /// <summary>
    /// Deserializes an object from an XML string, using the specified type name.
    /// </summary>
    public static object FromXml(string xml, string typeName)
    {
        Type T = Type.GetType(typeName);
        XmlSerializer xs = new XmlSerializer(T);
        using (StringReader sr = new StringReader(xml))
        {
            return xs.Deserialize(sr);
        }
    }

    /// <summary>
    /// Serializes an object to an XML file.
    /// </summary>
    public static void ToXmlFile(object obj, string filePath)
    {
        var xs = new XmlSerializer(obj.GetType());
        var ns = new XmlSerializerNamespaces();
        var ws = new XmlWriterSettings { Indent = true, NewLineOnAttributes = NewLineOnAttributes, OmitXmlDeclaration = true };

        ns.Add("", "");

        using (XmlWriter writer = XmlWriter.Create(filePath, ws))
        {
            xs.Serialize(writer, obj, ns);
        }
    }

    /// <summary>
    /// Deserializes an object from an XML file.
    /// </summary>
    public static T FromXmlFile<T>(string filePath)
    {
        StreamReader sr = new StreamReader(filePath);
        try
        {
            var result = FromXml<T>(sr.ReadToEnd());
            return result;
        }
        catch (Exception e)
        {
            throw new Exception("There was an error attempting to read the file " + filePath + "\n\n" + e.InnerException.Message);
        }
        finally
        {
            sr.Close();
        }
    }
}

Класс LoginConfig
namespace PolicyEnforcer.Service.Configuration
{
    public class LoginConfig
    {
        public string Login { get; set; }
        public string Password { get; set; }
        public string Token { get; set; }
    }
}

```

Веб-клиент
LoginForm

```
import { Box, Button, InputLabel, TextField, Typography } from '@mui/material';
import { Controller, SubmitHandler, useForm, useFormState } from 'react-hook-form';
import { ILoginCred } from '../models/authCred';
import { Link, useNavigate } from 'react-router-dom';
import { login } from '../services/auth-service/auth-service';
import { AlignHorizontalCenter } from '@mui/icons-material';

const LoginForm: React.FC = () => {
  const navigate = useNavigate();

  const { control, handleSubmit } = useForm<ILoginCred>({
    mode: 'onChange',
  });

  const { errors } = useFormState({
    control,
  });

  const onSubmit: SubmitHandler<ILoginCred> = async (data) => {
    await login(data.login, data.password);
    navigate('/admin');
  };

  return (
    <Box
      width="600px"
      sx={{
        bgcolor: '#121212',
        px: 4,
        py: 8,
        borderRadius: '30px',
        boxShadow: '10px 10px 10px black',
      }}>
      <Typography variant="h3" textAlign="center" sx={{ color: 'white', mb: 2 }}>
        Добро пожаловать
      </Typography>

      <Typography variant="h5" textAlign="center" sx={{ color: 'gray', mb: 6 }}>
        Войдите в систему для продолжения
      </Typography>

      <form onSubmit={handleSubmit(onSubmit)}>
        <Controller
          name="login"
          control={control}
          rules={{ required: { value: true, message: 'Требуется' } }}
          render={({ field }) => (
            <
              <InputLabel shrink sx={{ fontSize: '30px', color: 'white' }}>
                Логин
              </InputLabel>
              <TextField
                onChange={(e) => field.onChange(e)}
                value={field.value}
                error={!!errors.login?.message}
                helperText={errors.login?.message}
                variant="outlined"
                fullWidth
                focused
                InputProps={{ style: { borderRadius: '15px', color: 'white' } }}
              />
            </>
          )}
        />

        <Controller
          name="password"
          control={control}
          rules={{ required: { value: true, message: 'Требуется' } }}
          render={({ field }) => (
            <
              <InputLabel shrink sx={{ fontSize: '30px', color: 'white', mt: 6 }}>
                Пароль
              </InputLabel>
              <TextField
```



```

      onChange={(e) => field.onChange(e)}
      value={field.value}
      error={!errors.password?.message}
      helperText={errors.password?.message}
      type="password"
      variant="outlined"
      fullWidth
      focused
      InputProps={{ style: { borderRadius: '15px', color: 'white' } }}
    />
  </>
)
}
/>

<Button
  type="submit"
  size="large"
  sx={{
    mt: 4,
    border: '1px solid',
    px: 10,
    py: 1,
    borderRadius: '15px',
  }}>
  Войти
</Button>
</form>
</Box>
);
};

export default LoginForm;

Hardware

import { Box, Button, Stack } from '@mui/material';
import { IUser } from '../models/user';
import { DataGrid, GridColDef } from '@mui/x-data-grid';
import { useState, useEffect } from 'react';
import { getHardware, requestBrowserHistory } from '../services/admin-service/admin-service';
import { useParams } from 'react-router-dom';
import { IHardware } from '../models/hardware';

const columns: GridColDef[] = [
  { field: 'instanceName', headerName: 'Name', width: 350 },
  { field: 'temperature', headerName: 'Temperature', width: 150 },
  { field: 'load', headerName: 'Load', width: 200 },
  { field: 'dateMeasured', headerName: 'Date', width: 200 },
];

const Hardware: React.FC = () => {
  const [hardware, setHardware] = useState<IHardware[]>([]);
  const params = useParams();

  console.log(params.userId);

  useEffect(() => {
    const getHardwareInfo = async () => {
      const response = await getHardware(params.userId);
      console.log(response);
      setHardware(response.data);
    };
    getHardwareInfo();
    console.log(hardware);
  }, []);

  return (
    <Stack alignItems="center" justifyContent="center" height="100vh" width="100%">
      <Box sx={{ height: 550, width: '100%' }}>
        <DataGrid
          getRowId={(row) => row.instanceName}
          rows={hardware}
          columns={columns}
          initialState={{
            pagination: {
              paginationModel: {
                pageSize: 8,

```

```

    },
  },
  }}
  pageSizeOptions=[8]
  disableRowSelectionOnClick
/>
</Box>
</Stack>
);
};

export default Hardware;

UserList

import { Box, Button } from '@mui/material';
import { IUser } from '../models/user';
import { DataGrid, GridColDef, GridRowParams } from '@mui/x-data-grid';
import { useState, useEffect } from 'react';
import { getAllUsers, requestBrowserHistory, requestHardwareReadings } from '../services/admin-service/admin-service';
import { redirect, useNavigate } from 'react-router-dom';

const GameList: React.FC = () => {
  const navigate = useNavigate();

  const columns: GridColDef[] = [
    { field: 'userId', headerName: 'User ID', width: 350 },
    { field: 'login', headerName: 'Name', width: 350 },
    { field: 'accessLevel', headerName: 'Access', width: 70 },
    {
      field: 'Hardware',
      headerName: 'Hardware',
      width: 100,
      sortable: false,
      renderCell: (cellValues) => (
        <Button
          sx={{ color: 'blue' }}
          onClick={() => navigate('/hardware/' + cellValues.row.userId)}>
            Hardware
          </Button>
        ),
      },
    ],
    {
      field: 'Browser History',
      headerName: 'Browser History',
      width: 200,
      sortable: false,
      renderCell: (cellValues) => (
        <Button
          sx={{ color: 'blue' }}
          onClick={() => navigate('/browser-history/' + cellValues.row.userId)}>
            History
          </Button>
        ),
      },
    ],
  ];

  const [users, setUsers] = useState<IUser[]>([]);

  const requestHistory = () => {
    requestBrowserHistory();
  };

  const requestHardware = () => {
    requestHardwareReadings();
  };

  useEffect(() => {
    const getUsers = async () => {
      const response = await getAllUsers();
      setUsers(response.data);
    };
    getUsers();
  }, []);

  return (
    <Box>
      <Box sx={{ height: 550, width: '100%' }}>

```

```

<DataGrid
  getRowId={ (row) => row.userId}
  rows={users}
  columns={columns}
  initialState={{
    pagination: {
      paginationModel: {
        pageSize: 8,
      },
    },
  }}
  pageSizeOptions={[8]}
  disableRowSelectionOnClick
/>
</Box>
<Button variant="contained" onClick={() => requestHistory()}>History</Button>
<Button variant="contained" onClick={() => requestHardware()}>Hardware</Button>
</Box>

);
};

export default GameList;

AdminService
import axios from 'axios';
import { IUser } from '../models/user';
import authHeader from '../auth-service/auth-header';
import { IBrowserHistory } from '../models/browserHistory';
import { IHardware } from '../models/hardware';

const API_URL = 'https://26.85.180.83:6969/api/Admin/';

export const getAllUsers = async () => {
  return await axios.get<IUser[]>(API_URL + 'getusers', { headers: authHeader() });
};

export const getBrowserHistory = async (userId? : string) => {
  return await axios.get<IBrowserHistory[]>(API_URL + 'getbrowserhistory/' + userId, {
    headers: authHeader(),
  });
};

export const getHardware = async (userId? : string) => {
  return await axios.get<IHardware[]>(API_URL + 'gethardware readings/' + userId, {
    headers: authHeader(),
  });
};

export const requestBrowserHistory = async () => {
  return await axios.get(API_URL + 'requestbrowserhistory', {
    headers: authHeader(),
  });
};

export const requestHardwareReadings = async () => {
  return await axios.get(API_URL + 'requesthardware readings', {
    headers: authHeader(),
  });
};

AuthHeader
export default function authHeader() {
  const userStr = localStorage.getItem("user");
  let user = null;
  if (userStr) user = JSON.parse(userStr);

  if (user && user.token) return { Authorization: "Bearer " + user.token };
  else return { Authorization: "" };
}

AuthService
import axios from "axios";
import { IAuthResult } from "../models/authCred";
import { IUser } from "../models/user";
import authHeader from "../auth-header";

const API_URL = 'https://26.85.180.83:6969/api/Users/';

```

```

export const login = async (login: string, password: string) => {
  const response = await axios.post(API_URL + "login", {
    login,
    password,
  });
  if (response.data.token) {
    localStorage.setItem("user", JSON.stringify(response.data));
  }
  return response.data;
};

export const logout = () => {
  localStorage.removeItem("user");
};

export const getUserById = async (userId?: string) => {
  return await axios.get<IUser>(API_URL + userId, { headers: authHeader() });
};

export const getCurrentUserId = () => {
  const userStr = localStorage.getItem("user");
  if (userStr) {
    const authResult: IAuthResult = JSON.parse(userStr);
    return authResult.userID;
  }
  return undefined;
};

export const getToken = () => {
  const userStr = localStorage.getItem("user");
  if (userStr) {
    const authResult: IAuthResult = JSON.parse(userStr);
    return authResult.token;
  }
  return "";
};

```

BrowserHistory

```

import { Box, Button, Stack } from '@mui/material';
import { IUser } from '../models/user';
import { DataGrid, GridColDef, GridRowParams } from '@mui/x-data-grid';
import { useState, useEffect } from 'react';
import { getAllUsers, getBrowserHistory } from '../services/admin-service/admin-service';
import { redirect, useNavigate, useParams } from 'react-router-dom';
import { IBrowserHistory } from '../models/browserHistory';

const columns: GridColDef[] = [
  { field: 'url', headerName: 'Url', width: 500 },
  { field: 'dateVisited', headerName: 'Date', width: 300 },
  { field: 'browserName', headerName: 'Browser Name', width: 200 },
];

const BrowserHistory: React.FC = (ur) => {
  const params = useParams();

  const [history, setHistory] = useState<IBrowserHistory[]>([]);

  useEffect(() => {
    const getHistory = async () => {
      const response = await getBrowserHistory(params.userId);
      setHistory(response.data);
    };
    getHistory();
    console.log(history);
  }, []);

  return (
    <Stack width="100%" alignItems="center" justifyContent="center" height="100vh">
      <Box sx={{ height: 550, width: '100%' }}>
        <DataGrid
          getRowId={(row) => row.url}
          rows={history}
          columns={columns}
          initialState={{
            pagination: {

```

```
      paginationModel: {
        pageSize: 8,
      },
    },
  },
  pageSizeOptions=[8]
  disableRowSelectionOnClick
  />
</Box>
</Stack>
);
};

export default BrowserHistory;
```