Вступление

В современном мире информационных технологий все большее значение приобретает разработка качественных и удобных веб-приложений для различных целей. Одним из таких приложений является личный кабинет пользователя, который позволяет осуществлять авторизацию, просмотр и редактирование персональных данных, получение информации о статусе и результатах обучения, общение с преподавателями и другими пользователями и т.д. Личный кабинет является неотъемлемой частью любого образовательного портала или платформы дистанционного обучения.

Однако создание личного кабинета требует не только разработки клиентской части приложения, которая отвечает за взаимодействие с пользователем и отображение интерфейса, но и разработки серверной части приложения, которая отвечает за обработку запросов пользователя, хранение и обработку данных, реализацию самой логики работы приложения и обеспечение безопасности, надежности работы приложения. Серверная часть приложения является сердцем и мозгом личного кабинета пользователя.

Проблема, которую необходимо было решить в ходе этой работы, заключалась в разработке и внедрении надежной, безопасной и масштабируемой серверной архитектуры для системы личных учетных записей. Это включало в себя выбор подходящих технологий, проектирование моделей данных и схем, разработку серверных API и сервисов, а также интеграцию различных компонентов системы.

Краткий анализ других работы в этой области показывает, что на рынке доступно множество аналогичных систем и решений. Однако каждая система имеет свои уникальные особенности и требования, и универсального решения не существует. Таким образом, дизайн и внедрение серверной части системы личного кабинета должны были быть адаптированы к конкретным потребностям и ограничениям колледжа и его студентов.

Объектом данной работы является разработка серверной части системы личного кабинета Нижнетагильского Машиностроительного Техникума. Это включает в себя все процессы, компоненты и взаимодействия, которые необходимы для бесперебойного и безопасного функционирования системы.

Предметом этой работы является разработка и внедрение надежной, безопасной и масштабируемой серверной архитектуры для системы личных учетных записей. Это включает в себя выбор подходящих технологий, проектирование моделей данных и схем, разработку серверных API и сервисов, а также интеграцию различных компонентов системы.

Целью данной работы является разработка серверной архитектуры для системы личного кабинета Нижнетагильского Машиностроительного Техникума, которая была бы надежной, безопасной и масштабируемой. Архитектура должна быть способна обрабатывать большое количество пользователей и данных, обеспечивать быстрый и эффективный доступ к образовательным ресурсам и обеспечивать конфиденциальность и безопасность пользовательских данных.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ существующих решений по созданию серверной части личных кабинетов пользователей;

- выбрать оптимальный стек технологий для разработки серверной части личного кабинета НМТ;

- спроектировать архитектуру серверной части личного кабинета НМТ;

- реализовать функционал серверной части личного кабинета НМТ;

- провести тестирование и отладку серверной части личного кабинета НМТ.

Теоретическая значимость работы заключается в том, что она способствует расширению и углублению знаний в области разработки серверных приложений для личных кабинетов пользователей. В работе рассматриваются основные принципы и подходы к проектированию и реализации серверной части личного кабинета, а также анализируются существующие технологии и инструменты для разработки серверных приложений на языке программирования JavaScript. Результаты работы могут быть использованы для дальнейшего изучения и совершенствования методов и средств разработки серверных приложений, а также для обмена опытом и знаниями с другими разработчиками.

Практическая значимость данного исследования заключается в том, что оно обеспечит более эффективную и надежную серверную инфраструктуру для личного кабинета Нижнетагильского Машиностроительного Техникума. Это улучшит пользовательский интерфейс личного кабинета, сделает его более безопасным и надежным, а также повысит эффективность административных задач Нижнетагильского Машиностроительного Техникума. Развитая инфраструктура также может быть использована в качестве шаблона для других образовательных учреждений, имеющих аналогичные потребности.

Теоретической базой для написания работы послужили следующие источники:

- Кантор И. Современный учебник JavaScript [Электронный ресурс]. – Режим доступа: https://learn.javascript.ru/. – Дата доступа: 01.03.2021.

- Макконнелл С. Совершенный код: мастер-класс / пер. с англ. – СПб.: Питер, 2019. – 896 с.

- Флэнаган Д. JavaScript: подробное руководство / пер. с англ. – 6-е изд., испр. – М.: Вильямс, 2011. – 1080 с.

- Шилдт Г., Холмс Б., Кузнецов А., Кузнецов А., Левин И., Попов А., Тарасевич А., Тарасевич А., Тарасевич А., Тарасевич А., Тарасевич А., Тарасевич А., Тарасевич А., Тарасевич А Java: полное руководство / пер. с англ.. - 11-е изд.. - М.: Вильямс, 2019.. - 1488 c..

- Эльдер Дж., Гудрич Майкл Типпи Джей Algorithms Unlocked / пер.. - М.: Вильямс, 2017.. - 240 c..

Теоретическая часть

Анализ требований:

Для проведения анализа полученных требований, в первую очередь, необходимо, чтобы каждое из выявленных в предыдущих пунктах являлось:

- Единственным;

- Проверяемым;

- Недвусмысленным;

- Обслуживаемым

- Выполнимым

- Актуальным

- Завершенным

- Последовательным

- Атомарным

Также требования, разрабатываемые к продукту, должны соответствовать 6 показателям качества:

1. Функциональность

- Возможность авторизации пользователей в веб-приложении.

- Разграничение доступа к информации и функционалу веб-приложения с помощью использования следующих ролей: администратор, преподаватель, родитель, студент

- Возможность просмотра всех пользователей своей личной информации об аккаунте.

- Возможность просмотра расписания занятий всеми типами пользователей.

- Возможность добавления расписания занятий администратором.

- Возможность просмотра полученных уведомлений студентами, родителями и преподавателями.

- Возможность отправки уведомлений преподавателями и студентами для выбранных пользователей.

- Наличие возможности администратором просмотра всех пользователей веб-приложения.

- Возможность просмотра личной зачетной книжки студентом и родителем (просмотра зачетной книжки своего ребенка) по семестрам и годам обучения.

- Возможность администратора и преподавателя вносить изменения в зачетную книжку студента.

- Возможность администратора добавления дисциплин.

- Возможность администратора добавления/изменение/удаление полей зачетной книжки (указанием типа экзамена/зачета, количества часов, рассчитанного на обучение, даты сдачи дисциплины, закрепление определенного преподавателя за дисциплиной и другое)

- Возможность администратора добавления/удаления учебных семестров

- Возможность всех типов пользователей загружать файлы на сервер

- Возможность пользователей создавать/удалять виртуальные папки на сервере для структуирования файлов на сервере и личного удобства.

- Разграничение прав доступа к папкам и файлам (см. ниже подраздел Правила Разграничения Файлов и Папок)

- Возможность авторизации через active directory.

2) Переносимость:

- Возможность работы серверной части веб-приложения на различных OC

- Минимальные требования для работы сервера: 1 ядро процессора на чистоте 1 ГГц, 1 Гб оперативной памяти, широкоформатное подключение к сети Интернет.

3) Надежность:

- Быстрая скорость восстановления работы системы после сбоя

4) Удобство сопряжения:

- Наличие возможности исправления ошибок, обнаруженных при использовании продукта

- Возможность расширения имеющегося функционала в рамках нужд Нижнетагильского Машиностроительного Техникума

- Возможность подключения к существующему серверу Active Directory для удобного внедрения серверной части в общую образовательную экосистему.

Правила Разграничения Файлов и Папок по Ролям

Изначально в системе будет присутствовать корневая директория - папка «Задание», которая видна каждому пользователю и каждый может скачивать файлы, содержащие в ней и её папках. Только администраторы и преподавателю имеют возможность загружать и удалять файлы и папки из данной корневой директории.

Таким образом, если преподавателю необходимо выложить материал для студентов, то ему необходимо создать папку в папке «Задания» в соответствии со своей дисциплиной, после чего уже в неё загружать задания.

Стоит отметить, что папки и файлы, созданные одним преподавателем в корневой директории не могут быть редактированы/добавлены/удалены другими иными пользователями (в том числе другими администраторами, преподавателями).

Нужно провести некоторое разграничение: преподаватели также имеют свою личную папку, доступ к которой никто не может получить, кроме самих преподавателей.

Также как и преподаватели, студенты имеют личные папки, в которые могут добавлять файлы/папки и удалять их. Любой преподаватель и администратор может просматривать и скачивать файлы студентов, но по аналогии с папки «Задание» - преподаватели не имеют возможности удалять или добавлять файлы в личные папки студентов.

Благодаря такой системе студенты и преподаватели смогут обмениваться файлами без проблем с безопасностью: никто не может редактировать чужие файлы, что обеспечивает прозрачность системы. (см. рисунок 1)

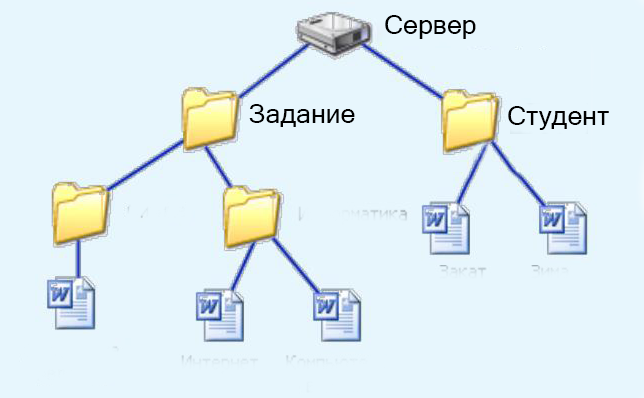


Рисунок 1 – визуализация виртуальной файловой системы.

Для удобства представления была разработана таблица в полной мере отражающая разграничение прав в файловой системе. (см. рисунок 2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Студент | Преподаватель | Администратор | Родитель |
| Просмотр/загрузка файлов/папок в папке «Задания» | ○ | ○ | ○ | ○ |
| Редактирование/добавление/удаление файлов/папок в папке «Задания» | × | ○ | ○ | × |
| Просмотр/загрузка файлов/папок из личной папки студента | ● | ○ | ○ | ● |
| Редактирование/добавление/удаление файлов/папок в личной папке студента | ● | × | × | ● |
| Просмотр/загрузка файлов/папок в личной папке преподавателя | × | ● | × | × |
| Редактирование/добавление/удаление файлов/папок в личной папке преподавателя | × | ● | × | × |
| Просмотр/загрузка файлов/папок в подпапке «Задания» | ○ | ○ | ○ | ○ |
| Редактирование/добавление/удаление файлов/папок в подпапке «Задания» | × | ● | ● | × |

Рисунок 2 – визуализация разграничения прав для файлов и папок.

Где:

× - Никто из представленной группы не имеет доступа.

○ – Доступ имеют все из представленной группы.

● – Никто из представленной группы не имеет доступа, кроме пользователя, зарегистрированного за файлом/папкой (например, если пользователь загружает файл, этот файл автоматически подписывается под него)

Спецификация требований:

Для фиксации установленных требований была разработана диаграмма вариантов использования. Данная диаграмма создается для наглядного представления функционального назначения.

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 4 – диаграмма вариантов использования.

Серверная часть:

В контексте разработки веб-приложений серверная часть отвечает за обработку входящих запросов от клиентов и генерацию ответов. Серверная часть обычно состоит из набора технологий, протоколов и служб, которые работают вместе для обеспечения необходимой функциональности. Эти технологии могут включать серверные языки программирования, такие как PHP, Python, Ruby или JavaScript, а также веб-серверы, такие как Apache, Nginx и IIS.

Серверная часть веб-приложения может быть разбита на различные уровни, каждый со своими собственными обязанностями. Самый нижний уровень - это сетевой уровень, который отвечает за обработку входящих запросов и отправку ответов обратно клиенту. Прикладной уровень - это уровень, который находится поверх сетевого уровня и отвечает за обработку запросов и генерацию ответов.

Фреймворки — это программные платформы, которые определяют структуру программной системы и облегчают разработку и объединение разных компонентов большого программного проекта. Фреймворки отличаются от библиотек тем, что они диктуют правила построения архитектуры приложения и используют инверсию управления. Фреймворки могут быть разных видов и предназначения, например веб-фреймворки, игровые фреймворки или научные фреймворки.

Fastify — это веб-фреймворк для Node.js, который сфокусирован на предоставлении лучшего опыта разработчика с минимальными накладными расходами и мощной архитектурой плагинов. Он вдохновлен Hapi и Express и является одним из самых быстрых веб-фреймворков на рынке . Fastify поддерживает валидацию и сериализацию данных с помощью JSON Schema, логирование с помощью Pino, расширяемость с помощью хуков, плагинов и декораторов. Fastify также имеет большое сообщество и экосистему модулей для различных задач.

Fastify построен поверх Node.js и использует свой встроенный HTTP-сервер. Он разработан таким образом, чтобы быть модульным и легко расширяемым, с системой плагинов, которая позволяет разработчикам добавлять функциональность по мере необходимости. Fastify также разработан таким образом, чтобы его было легко тестировать, благодаря встроенной поддержке модульного тестирования и интеграционного тестирования.

JWT-токены — это открытый стандарт (RFC 7519), который определяет компактный и самодостаточный способ безопасной передачи информации между сторонами в виде JSON-объекта. Эта информация может быть проверена и доверена, потому что она цифрово подписана. JWT-токены могут быть подписаны с помощью секрета (с алгоритмом HMAC) или пары открытого/закрытого ключа с использованием RSA или ECDSA.

JWT-токены полезны для таких сценариев, как:

- Авторизация: это самый распространенный сценарий использования JWT. После входа пользователя каждый последующий запрос будет содержать JWT, позволяющий пользователю получать доступ к маршрутам, сервисам и ресурсам, разрешенным этим токеном. Единый вход - это функция, которая широко использует JWT в настоящее время, потому что он имеет небольшой объем и может быть легко использован в разных доменах.

- Обмен информацией: JWT-токены - это хороший способ безопасной передачи информации между сторонами. Поскольку JWT-токены могут быть подписаны - например, с использованием пары открытого/закрытого ключа - вы можете быть уверены, что отправители являются теми, за кого они себя выдают. Кроме того, поскольку подпись рассчитывается с использованием заголовка и полезной нагрузки, вы также можете проверить, что содержимое не было изменено.

Структура JWT-токена состоит из трех частей, разделенных точками (.), которые являются:

- Заголовок

- Полезная нагрузка

- Подпись

Заголовок обычно состоит из двух частей: типа токена (JWT) и используемого алгоритма подписи (например HMAC SHA256 или RSA).

Полезная нагрузка содержит утверждения (claims), которые являются заявлениями об объекте (обычно пользователе) и дополнительных данных. Существуют три типа утверждений: зарегистрированные (registered), публичные (public) и приватные (private).

Подпись гарантирует целостность и подлинность токена. Она рассчитывается по формуле:

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

где secret - это ключ для генерации хеша.

Протоколы HTTP/HTTPS

Протокол HTTP является основой передачи данных во Всемирной паутине. Он определяет, как форматируются и передаются сообщения, а также определяет действия, которые веб-серверы и браузеры должны предпринимать в ответ на различные команды.

HTTPS - это защищенная версия HTTP, которая использует шифрование SSL / TLS для защиты передаваемых данных. Протокол HTTPS необходим для обеспечения безопасности и конфиденциальности веб-транзакций, особенно в отношении конфиденциальных данных, таких как пароли, номера кредитных карт и личная информация.

Стандарт REST API - это способ создания и использования API (application programming interface), который следует определенным принципам и ограничениям. REST (representational state transfer) означает, что при запросе ресурса с помощью REST API сервер возвращает текущее состояние ресурса в стандартизированном представлении. Например, сервер может вернуть данные в формате JSON или XML.

Стандарт REST API имеет следующие основные характеристики:

- Основан на HTTP. REST API использует стандартные HTTP-методы для выполнения операций над ресурсами, таких как GET, POST, PUT, PATCH и DELETE. Каждый ресурс имеет свой уникальный URI (uniform resource identifier), который идентифицирует его в сети.

- Без сохранения состояния (stateless). Каждый запрос к REST API должен содержать всю необходимую информацию для его обработки. Сервер не хранит информацию о состоянии клиента или предыдущих запросах. Это улучшает производительность и масштабируемость системы.

- Единообразный интерфейс (uniform interface). Все запросы и ответы к REST API должны следовать одному и тому же формату и соглашениям. Это упрощает разработку и интеграцию разных компонентов системы.

- Кэшируемость (cacheable). Ответы от сервера могут быть кэшированы на стороне клиента или промежуточных серверов для повышения эффективности и снижения нагрузки на сервер. Ответы должны содержать информацию о том, можно ли их кэшировать или нет.

- Слоистая архитектура (layered system). Взаимодействие между клиентом и сервером может проходить через разные слои или компоненты системы, такие как балансировщики нагрузки, прокси-серверы или шлюзы. Клиент не знает о структуре системы за пределами своего непосредственного контакта с сервером.

Одним из книжных источников: "RESTful Web APIs" авторства Leonard Richardson, Mike Amundsen и Sam Ruby (O'Reilly Media, 2013).

Стандарт OpenAPI (OAS) определяет стандартный, независимый от языка программирования интерфейс для описания HTTP API, который позволяет как людям, так и компьютерам обнаруживать и понимать возможности сервиса без доступа к исходному коду, дополнительной документации или анализа сетевого трафика. Стандарт OpenAPI ранее назывался спецификацией Swagger и был частью фреймворка Swagger до 2016 года, когда он стал отдельным проектом под эгидой OpenAPI Initiative - совместного проекта Linux Foundation. Стандарт OpenAPI может использоваться для генерации документации по API, кода для серверов и клиентов на разных языках программирования, тестовых инструментов и многих других сценариев. Последняя версия стандарта OpenAPI - 3.1.0 - была выпущена в феврале 2021 года.

1.[OpenAPI Specification v3.1.0 | Introduction, Definitions, & More](https://spec.openapis.org/oas/latest.html)

PostgreSQL — это мощная, открытая объектно-реляционная система управления базами данных с более чем 35-летней историей активной разработки, которая заслужила себе прочную репутацию надежности, функциональности и производительности. PostgreSQL поддерживает как реляционные, так и нереляционные типы данных и совместим с множеством языков программирования. PostgreSQL можно скачать в виде готовых пакетов или установщиков для разных платформ, а также в виде архива исходного кода, если вы хотите самостоятельно его скомпилировать. PostgreSQL также имеет большое сообщество разработчиков и пользователей, которые предлагают дополнительное программное обеспечение, интерфейсы и расширения для PostgreSQL.

PostgreSQL известен своей надежностью, масштабируемостью и робастностью. Он особенно хорошо подходит для веб-приложений, которым требуется высокая доступность и производительность.

1.[PostgreSQL: The world's most advanced open source database](https://www.postgresql.org/)

Зачем разделять веб-приложения на клиентские часть и серверная часть?

- Большая гибкость и масштабируемость. Клиентская часть отвечает за отображение интерфейса и взаимодействие с пользователем, а серверная часть - за обработку запросов и доступ к данным. Это позволяет разрабатывать и обновлять каждую часть независимо от другой, а также использовать разные технологии и языки программирования для каждой части .

- Большая безопасность и надежность. Серверная часть скрывает от клиента детали реализации бизнес-логики и доступа к базам данных, что уменьшает риск несанкционированного доступа или манипуляции с данными . Кроме того, серверная часть может выполнять проверку и валидацию данных, полученных от клиента, чтобы предотвратить ошибки или атаки.

- Большая производительность и эффективность. Клиентская часть может выполнять некоторые операции на стороне пользователя (например, валидация формы или фильтрация данных), что уменьшает нагрузку на сервер и сеть . Серверная часть может оптимизировать обработку запросов и доступ к данным, используя кэширование, индексирование или распределенные системы.

Одним из книжных источников по данной теме является "Web Development with Node and Express: Leveraging the JavaScript Stack" авторства Ethan Brown (O'Reilly Media, 2014). В главе 3 этой книги автор подробно объясняет разницу между клиентской и серверной частями веб-приложения, а также приводит примеры кода на Node.js для создания простого веб-сервера.

Active Directory — это служба каталогов, которая хранит информацию об объектах в сети и делает эту информацию доступной для пользователей и администраторов. Active Directory использует структурированное хранилище данных в качестве основы для логической иерархической организации информации о каталогах. Эти объекты обычно включают общие ресурсы, такие как серверы, тома, принтеры, а также учетные записи пользователей и компьютеров в сети.

Сервер с active directory — это сервер, на котором установлена роль Active Directory Domain Services (AD DS). AD DS предоставляет методы для хранения данных каталогов и предоставления этих данных в сети. Сервер с AD DS называется контроллером домена (domain controller). Контроллер домена отвечает за аутентификацию и авторизацию пользователей и устройств в домене. Домен — это логическая группа компьютеров и других объектов под единым управлением.