Содержание

Введение

Хронологическое представление информации позволяет структурировать события и увидеть их в четкой временной последовательности, что особенно важно в таких сферах, как история, управление проектами и личные записи.

Создание цифрового ресурса, который объединяет события и позволяет пользователям добавлять, редактировать и управлять временными последовательностями, предоставляет новые возможности для управления и анализа информации. Это может быть полезно для историков, исследователей, проектных менеджеров и индивидуальных пользователей, стремящихся упорядочить данные о событиях и связанных с ними файлах.

Целью данной разработки является оптимизация процесса создания и редактирования хронологий событий по различным тематическим направлениям, а также повышение удобства управления событиями, хранения мультимедийной информации и персонализированного сохранения данных.

Основными задачами являются:

- Провести обзор и анализ аналогичных решений, представленных на рынке, для выявления их преимуществ и недостатков в контексте заявленных ключевых особенностей.
- Определить набор функций, которые обеспечат удобное создание и управление хронологиями событий с учетом мультимедийного контента (фото, видео, документы).
- Разработать безопасный механизм авторизации и регистрации пользователей, обеспечивающий доступ к персонализированному профилю и защите пользовательских данных.
- Создать интерфейс для удобного добавления, просмотра и редактирования событий с возможностью указания точной даты и времени.

- Разработать систему хранения хронологий в профиле пользователя для обеспечения быстрого доступа к сохраненным данным.
- Реализовать полноэкранный режим просмотра хронологии с доступом к каждому событию и его материалам.
- Разработать функционал для изменения последовательности событий, добавления новых и редактирования существующих записей.

Исследовательская часть

1. Обзор существующих решений для создания хронологий событий

Для того, чтобы понять необходимость предлагаемой разработки, нужно проанализировать аналогичные решения из уже существующих. Рассмотрим 3 аналогичных решения для создания хронологий событий:

1. Journey (рис.1):

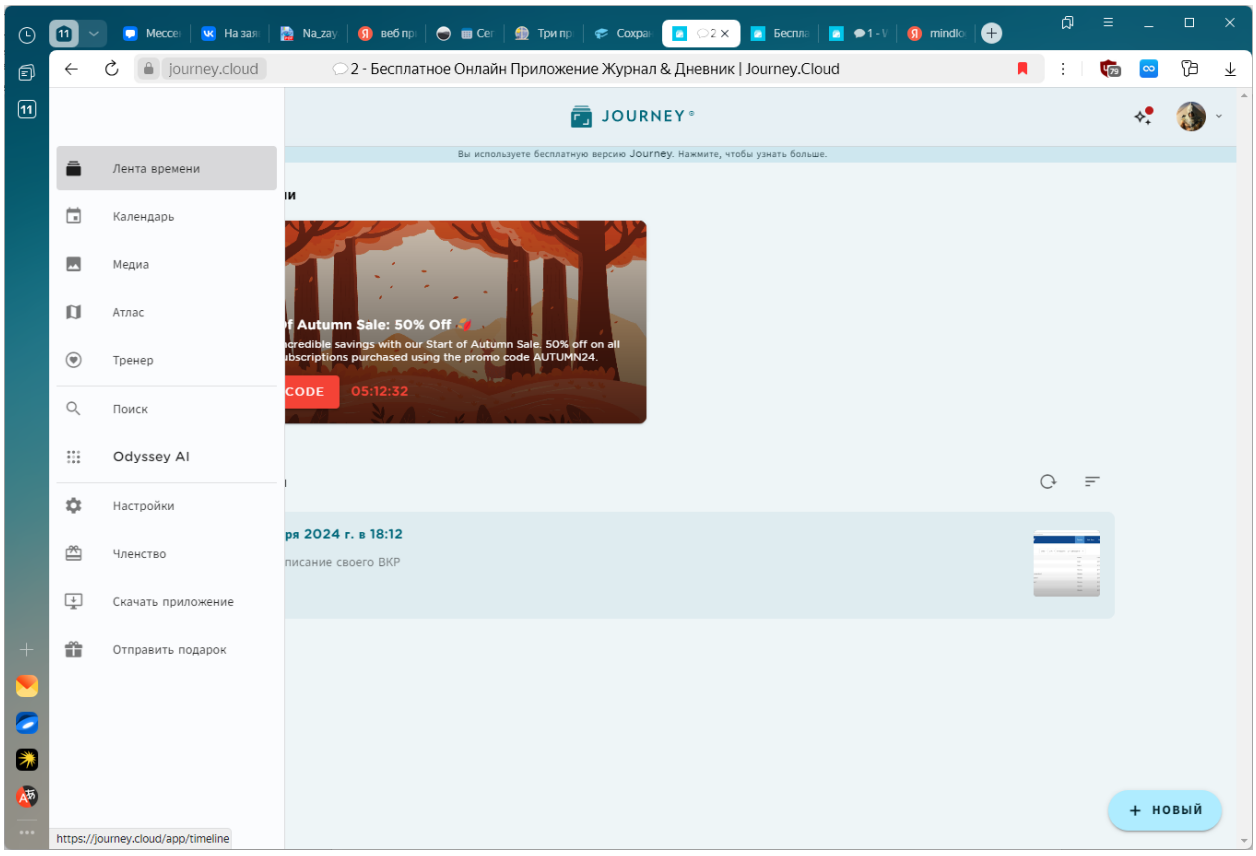

Рисунок 1 – общий вид Journey

Journey — это кроссплатформенное приложение для ведения дневника (Android, iOS, Mac, ПК, Linux, веб-версия, электронная почта), которое

предназначено для того, чтобы ваши воспоминания оставались личными и сохранялись навсегда. Это отмеченное наградами приложение для ведения дневника, которое в 2015, 2016, 2017, 2018 и 2023 годах было выбрано редакторами Google Play Store. В заметках редакторов Apple также упоминается Journey как «приложение для ведения дневника, позволяющее сохранять личные воспоминания».

Однако, у данного аналога есть существенные минусы:

1. Отсутствие представления данных в виде хронологической ленты событий.
2. Нет разделения хронологий событий по различным предметным областям, то есть все события идут одним потоком.
3. Возможность создания событий только на текущую дату и время.
4. Для хранения данных нужно подключать собственное облачное хранилище или покупать подписку на сервис.

2. MindLog (рис.2):

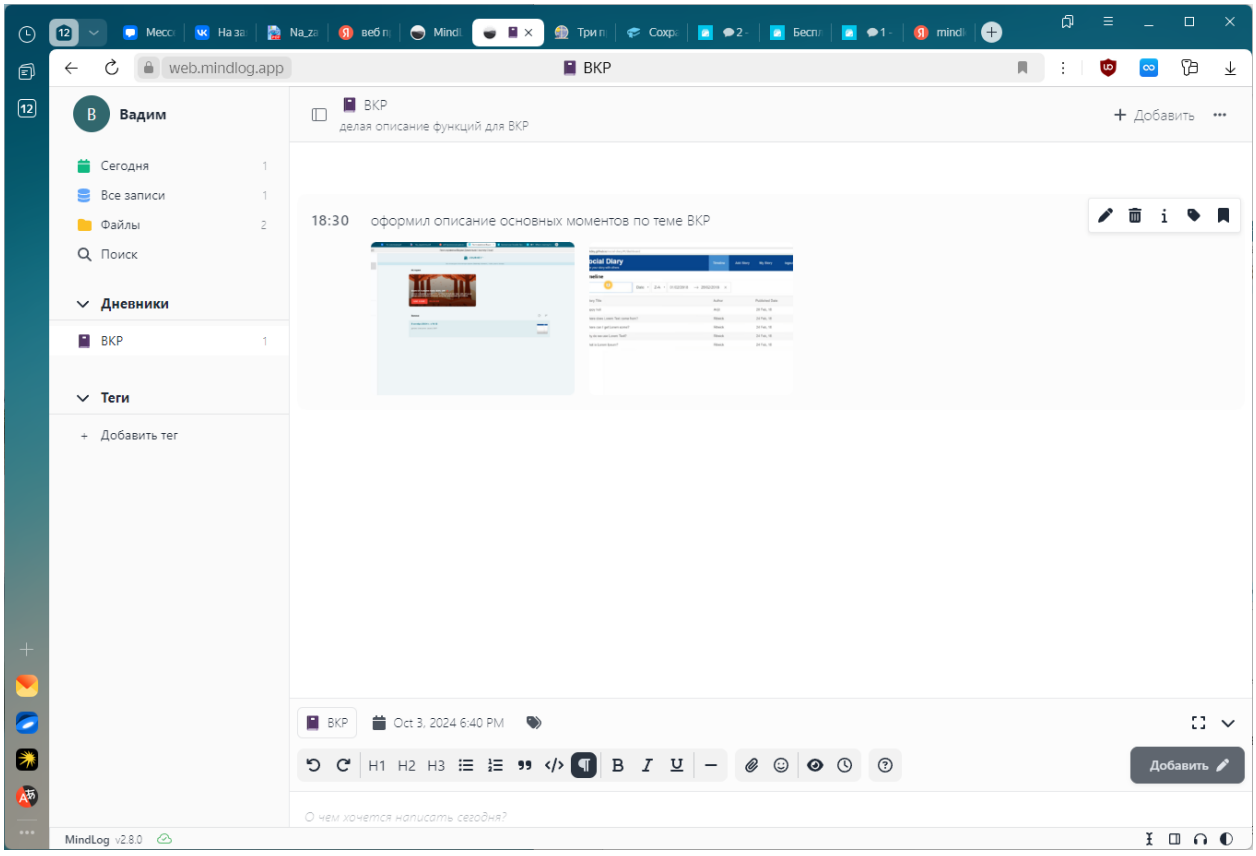

Рисунок 2 – Общий вид MindLog

MindLog — это персональный дневник онлайн-формата для хранения записей и файлов. С помощью MindLog пользователи хранят память о прошедших днях, личные и рабочие заметки, файлы. Делать записи можно как непосредственно в браузере, так и на мобильных устройствах. Данные шифруются с помощью алгоритма AES.

Особенности MindLog:

- хранение записей в онлайн-дневнике;
- прикрепление файлов и тегов к записям;
- текстовый редактор с поддержкой Markdown;
- доступ к заметкам без доступа к интернету;
- непрерывная лента записей пользователя;

Однако, у данного аналога есть существенные минусы:

1. Отсутствие представления данных в виде хронологической ленты событий.
2. Нет разделения хронологий событий по различным предметным областям, то есть все события идут одним потоком.
3. Нет возможности прикреплять к событиям документы.
3. Picnote (рис.3):

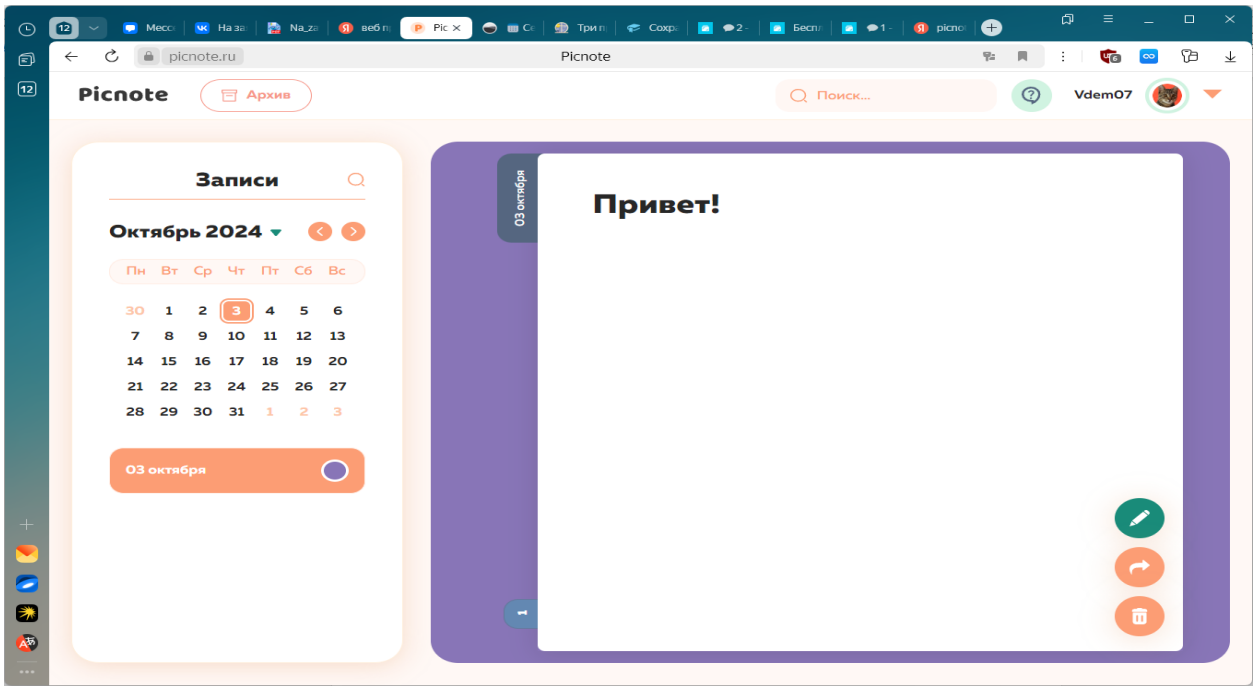

Рисунок 3 – Общий вид Picnote

Picnote — это веб-сайт, который позволяет вести личный дневник в онлайне. Здесь можно записывать свои мысли и заметки.

Некоторые особенности сервиса:

- Креативное оформление. Можно выбирать свой фон и шрифты для каждой записи, украшать странички дневника рисунками и стикерами.
- Удобный календарь. Можно просматривать уже сделанные записи, перемещаясь между ними в один-два клика.
- Загрузка изображений. Можно добавлять свои картинки или искать готовые по ключевым словам.
- Поиск по записям. Записи можно находить с помощью текстового поиска или меток.
- Деление записями. Можно открыть доступ к записи, получив публичную ссылку.
- Чтобы начать вести дневник в Picnote, нужно зарегистрироваться на сайте, ввести почту, придумать никнейм и пароль, затем создать новую запись и записать свои мысли.

Однако, у данного аналога есть существенные минусы:

1. Отсутствие представления данных в виде хронологической ленты событий.
2. Нет разделения хронологий событий по различным предметным областям, то есть все события идут одним потоком.
3. Нет возможности прикреплять к событиям документы.

2. Критерии сравнительного анализа существующих решений для создания хронологий событий

Для оценки аналогичных решений по созданию и управлению хронологией событий были выбраны следующие критерии:

1. Поддержка хронологической ленты событий: Возможность визуализировать события в виде последовательной ленты, что позволяет

пользователям видеть события в порядке их следования и возвращаться к нужным моментам.

2. Разделение хронологий по предметным областям Наличие возможности создавать разные категории или темы хронологий для удобной организации событий (например, личная жизнь, история компании, проект и т.д.), чтобы события не смешивались в одном потоке.
3. Поддержка прикрепления документов к событиям: Возможность добавлять к событиям не только фотографии и видео, но и документы, такие как PDF-файлы, текстовые файлы или другие формы документации.
4. Гибкость выбора даты и времени для события: Поддержка указания произвольной даты и времени события, а не только текущей даты, для возможности добавления исторических данных или будущих событий.
5. Условия хранения данных и доступ к ним: Условия хранения данных, включая необходимость дополнительного облачного хранилища или подписки для сохранения и синхронизации хронологий, что может ограничивать доступность приложения для пользователя.
6. Поддержка мультимедийного контента: Наличие возможности добавления фотографий, видео, а также различных типов файлов для обеспечения полноты представления событий, учитывая мультимедийные материалы.

На основе обзора и анализа существующих решений для создания хронологий событий, составим сравнительную таблицу:

Критерий/решение	Picnote	MindLog	Journey
Поддержка хронологической ленты событий	-	-	-
Разделение хронологий по предметным областям	-	+	-

Поддержка прикрепления документов	-	-	-
Гибкость выбора даты и времени для события	+	+	-
Условия хранения данных и доступ к ним	+	+	-
Поддержка мультимедийного контента	-	-	+

Сравнение существующих хронологических ресурсов, таких как Journey, Mindlog и Picnote, показывает, что большинство из них ориентированы на узкие задачи, такие как ведение личных записей или отслеживание настроения, и не обеспечивают достаточной гибкости и функциональности для более широкого использования. Эти приложения ограничены отсутствием хронологической структуры, недостаточной поддержкой мультимедийных данных и невозможностью адаптировать информацию для профессиональных целей. Кроме того, они не предлагают структурированного подхода к организации событий и часто лишены средств для редактирования или настройки хронологий. Эти ограничения подчеркивают необходимость разработки нового ресурса, способного устранить выявленные недостатки и удовлетворить как личные, так и профессиональные потребности пользователей.

Предлагаемый ресурс объединяет функции, которых недостает большинству аналогов, и включает возможности для создания хронологий, которые можно легко адаптировать как для личных, так и для профессиональных задач. В отличие от рассматриваемых аналогов, он предлагает комплексное решение для организации событий, поддерживая мультимедийные вложения и индивидуальные профили пользователей с возможностью редактирования. Такой подход позволяет использовать

разрабатываемую платформу как универсальный инструмент для хранения, анализа и структурирования временных данных, подходящий для широкого круга пользователей

3. Анализ процесса разработки информационной системы

На начальном этапе разработки информационной системы необходимо собрать и зафиксировать требования к функционалу приложения. Основные требования включают:

1. Возможность регистрации и авторизации пользователей для создания персонализированных профилей и защиты данных.
2. Создание и управление хронологиями событий, которые могут охватывать различные тематические направления (события из жизни, история организаций и т. д.).
3. Возможность добавления мультимедийной информации к событиям, включая тексты, изображения, видео и документы.
4. Функционал для просмотра хронологий на полный экран и работы с отдельными событиями и файлами.
5. Редактирование хронологий и их элементов, обеспечение возможности изменения и обновления данных.

Информационная система, предназначенная для создания и редактирования хронологий событий, представляет собой приложение архитектуры клиент-сервер. Этот подход обеспечивает распределение задач между клиентской и серверной частью системы, а также позволяет обеспечить масштабируемость, гибкость и безопасность приложения. Ниже приводится описание ключевых аспектов системы с точки зрения клиент-серверной архитектуры.

Компоненты системы:

1. Клиентская часть (Frontend): клиентская часть представляет собой интерфейс, с которым взаимодействует пользователь. Она отвечает за отображение данных, отправку запросов к серверу и обработку ответов.

Пользовательский интерфейс (UI) включает в себя все элементы управления, которые позволяют пользователям регистрироваться, авторизовываться, создавать, просматривать и редактировать хронологии событий.

2. Серверная часть (Backend): серверная часть обеспечивает основную логику приложения, обработку запросов от клиентов, управление данными и взаимодействие с базой данных. Сервер принимает запросы от клиентской части (например, создание нового события), обрабатывает их и отправляет ответы обратно клиенту.
3. База данных: база данных используется для хранения всех данных, связанных с системой, включая информацию о пользователях, хронологиях событий и мультимедийных материалах.

Общая схема клиентской и серверной части и их взаимодействие представлены на рисунке 4:

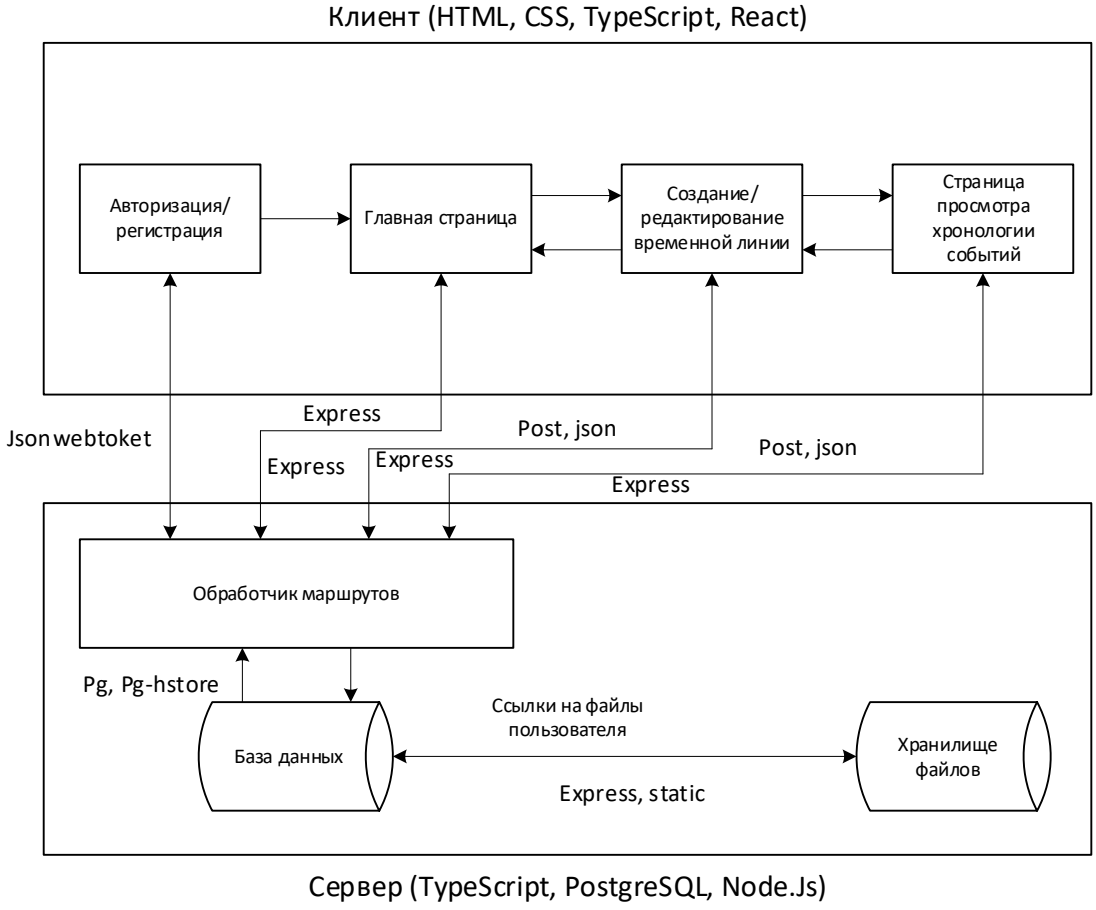

Рисунок 4 – Общая схема разработки

Взаимодействие клиента и сервера:

1. Клиент отправляет запрос: например, при создании новой хронологии событий пользователь вводит данные (описание, даты, изображения и пр.) и нажимает кнопку «Создать». Клиентское приложение формирует запрос и отправляет его на сервер через API.
2. Сервер обрабатывает запрос: сервер принимает запрос, проверяет данные на валидность, применяет бизнес-логику (например, проверку авторизации пользователя), выполняет необходимые операции с базой данных (создание новой записи) и формирует ответ.
3. Сервер отправляет ответ: после выполнения операций сервер отправляет ответ клиенту (например, подтверждение успешного создания хронологии или сообщение об ошибке).
4. Клиент отображает результат: клиентское приложение получает ответ от сервера и обновляет интерфейс в соответствии с действиями пользователя (например, добавляет новую хронологию в список).

4. Анализ необходимых средств реализации

Для реализации информационной системы по созданию и редактированию хронологий событий в рамках клиент-серверной архитектуры необходимо использовать различные технологии и инструменты, обеспечивающие функциональность, производительность и удобство работы с приложением. Рассмотрим средства и технологии, которые могут быть применены для разработки данной системы.

Средства и технологии для клиентской части:

- Фреймворки для разработки пользовательского интерфейса: React: популярная библиотека для построения пользовательских интерфейсов с компонентной архитектурой и широким сообществом поддержки. Vue.js: более легковесный, интуитивно понятный фреймворк с гибкой

архитектурой. Angular: мощный фреймворк, подходящий для крупных проектов с сложной логикой.

- HTML, CSS и препроцессоры CSS (например, Sass или Less): для структурирования разметки страниц и стилизации интерфейса.
- JavaScript и TypeScript: JavaScript для интерактивного поведения страниц. TypeScript (по необходимости) для типизации кода и снижения количества ошибок.
- Фреймворки для стилизации: Bootstrap, Material-UI, Tailwind CSS для ускорения создания UI-компонентов и адаптивной верстки.

Средства и технологии для серверной части:

- Языки программирования и фреймворки: Node.js с Express: легковесный и производительный стек для разработки серверной части с использованием JavaScript. Django (Python): мощный фреймворк с высокой степенью интеграции и встроенной системой администрирования. Spring Boot (Java): подходит для более сложных и корпоративных решений.
- Системы аутентификации и авторизации: Использование JWT (JSON Web Token) для авторизации и хранения токенов сеансов.

5. Выбор средств реализации

Для реализации веб-приложения для создания хронологий событий потребуется использование современных технологий и средств разработки клиент-серверных приложений. Такой подход обеспечит удобство работы с данными, безопасность и масштабируемость системы, а также предоставит пользователям простой и интуитивно понятный интерфейс. Ниже рассмотрим выбранные средства реализации проекта.

1. React (для фронтенда)

React — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. С её помощью создают веб-приложения, которые изменяют отображение без перезагрузки страницы.

Благодаря этому сайты быстро реагируют на действия пользователя, например, заполнение форм, применение фильтров, добавление товаров в корзину и так далее.

Некоторые особенности React:

- Однонаправленная передача данных. Свойства передаются от родительских компонентов к дочерним. Компоненты получают свойства как множество неизменяемых значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через callback-функции.
- Виртуальный DOM. React создаёт кэш-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM браузера.
- JSX. Это расширение синтаксиса JavaScript, которое позволяет использовать HTML-подобный синтаксис для описания структуры интерфейса.
- Методы жизненного цикла. Позволяют разработчику запускать код на разных стадиях жизненного цикла компонента.
- React Hooks. Хуки позволяют использовать состояние и другие возможности React без написания классов.

Причины выбора:

- Компонентный подход: React позволяет строить интерфейс с использованием отдельных компонентов, что упрощает поддержку и повторное использование кода.
- Virtual DOM: снижает нагрузку на браузер за счет обновления только изменившихся частей страницы, что улучшает производительность.
- Большое сообщество: богатая экосистема, множество сторонних библиотек и ресурсов для быстрого решения сложных задач.

- Совместимость: легко интегрируется с другими библиотеками и инструментами, такими как Redux и Context API для управления состоянием приложения.

2. PostgreSQL (СУБД)

PostgreSQL — это объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом. Она помогает хранить и организовывать информацию.

Особенности PostgreSQL:

- Поддержка стандарта SQL. PostgreSQL поддерживает большинство стандартов SQL.
- Гибкость. Пользователи могут создавать функции, операторы, типы данных и индексные методы.
- Соответствие ACID. PostgreSQL выполняет все четыре свойства ACID — набора требований к транзакциям.
- Поддержка JSON и XML. PostgreSQL позволяет хранить и управлять данными в форматах JSON и XML.
- Масштабируемость. PostgreSQL поддерживает горизонтальное и вертикальное масштабирование.
- Надежность. PostgreSQL быстро восстанавливается после сбоев благодаря журналу изменений транзакций (WAL).
- PostgreSQL используют для хранения данных веб-приложений, аналитических приложений, геоинформационных и корпоративных систем.

Причины выбора:

- Надежность и безопасность: PostgreSQL обеспечивает высокую степень надежности данных и строгий контроль за транзакциями.
- Расширяемость: Возможность добавлять собственные функции и использовать встроенные расширения для оптимизации запросов и обработки данных.

- Сложные запросы: Поддержка вложенных запросов, триггеров и хранимых процедур, что позволяет легко манипулировать сложными данными.
- Активное развитие: Широкая документация и сообщество, что упрощает разработку и устранение проблем.

3. Sequelize (ORM для PostgreSQL)

Sequelize — это инструмент для организации взаимодействия между платформой Node.js и реляционными базами данных без использования специального языка запросов SQL.

Он относится к объектно-реляционным сопоставителям (ORM) и связывает базы данных (Postgres, MySQL, MariaDB, SQLite и Microsoft SQL Server) с объектами JavaScript, создавая виртуальную объектную базу данных. Некоторые особенности Sequelize:

- поддержка создания, обновления, удаления сущностей;
- поддержка вложенных сортировок, сложных условий, LEFT JOIN, лимитов, подзапросов, кастомных запросов;
- защита от SQL-инъекций и отмена транзакций;
- возможность определять пользовательские геттеры и сеттеры для атрибутов моделей;
- определение виртуальных атрибутов — они не существуют в исходной таблице базы данных, Sequelize создаёт их автоматически.

Причины выбора:

- Удобное описание моделей: позволяет описывать структуры базы данных в виде моделей JavaScript, облегчая управление данными.
- Миграции: обеспечивает инструменты для контроля версий схем базы данных.
- Ассоциации: легко настраивает связи между таблицами и сложные отношения.

- Совместимость: хорошо сочетается с другими инструментами Node.js, такими как Express, и поддерживает масштабируемые архитектуры.

4. JWT (JSON Web Token) для авторизации

JSON Web Token (JWT) — это открытый стандарт для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.

Формат JWT состоит из трёх частей, которые разделены точкой:

1. Header (заголовок) — информация о токене, тип токена и алгоритм шифрования.
2. Payload (полезные данные) — данные, которые нужно передать в токене. Например, имя пользователя, его роль, истекает ли токен. Эти данные представлены в виде JSON-объекта.
3. Signature (подпись) — подпись токена, которая позволяет проверить, что токен не был изменён.

Особенности JWT:

- Отсутствие необходимости хранения сессий на сервере. Когда пользователь отправляет запрос с токеном, приложению достаточно проверить его подпись и извлечь данные из полезной нагрузки. Это упрощает масштабирование приложения, так как серверу не нужно управлять сессиями.
- Делегирование выдачи и валидации токенов. Приложение не обязано самостоятельно заниматься генерацией и проверкой токенов — эту задачу можно передать отдельному сервису аутентификации. Это позволяет упростить архитектуру приложения и повысить безопасность, так как управление токенами централизовано.
- Поддержка единого входа (SSO). Используя отдельный сервис аутентификации, можно организовать единую точку входа для нескольких приложений или сервисов. После прохождения аутентификации пользователь получает токен, который может быть

использован для доступа ко всем доверенным сервисам без повторного ввода учётных данных.

- Гибкость полезной нагрузки. В полезную нагрузку JWT можно включать любые данные о пользователе, что позволяет приложению работать более эффективно. Например, в токен можно включить информацию о ролях пользователя или других атрибутах, чтобы не обращаться к базе данных для их получения при каждом запросе.

Причины выбора:

- Безопасность: токены шифруются и подписываются, что позволяет защитить данные от несанкционированного доступа.
- Масштабируемость: подходит для распределенных систем, так как не требует проверки данных сессии на сервере.
- Гибкость: можно использовать для авторизации, хранения пользовательских данных и других целей.

5. Bcrypt (для хеширования паролей)

Bcrypt — адаптивная криптографическая хеш-функция формирования ключа, используемая для защищённого хранения паролей.

Некоторые особенности алгоритма bcrypt:

- Использование соли (salting). Перед хэшированием к каждому паролю добавляется случайное значение — соль. Она хранится вместе с хэшем и делает его уникальным для каждого пароля. Соление предотвращает атаки с помощью радужных таблиц и гарантирует, что даже идентичные пароли имеют разные хэши.
- Фактор затрат (work factor). Этот фактор регулирует количество итераций пароля, выполняемых перед генерацией хэша. Преднамеренное замедление процесса хэширования помогает защитить от атак методом перебора и по словарю.

- Поддержка паролей переменной длины. Bcrypt может обрабатывать пароли длиной до 72 байт. Если пароль превышает этот предел, он усекается, что обеспечивает совместимость в разных системах.

Причины выбора:

- Защита паролей: bcrypt применяет соли для хеширования паролей, делая их защищенными от атак перебора.
- Легкость в использовании: Простая интеграция с Node.js-приложениями и поддержка асинхронных операций.

6. CORS (для междоменного обмена)

CORS (Cross-Origin Resource Sharing) — это стандарт, позволяющий предоставлять веб-страницам доступ к объектам сторонних интернет-ресурсов. Сторонним считается любой интернет-ресурс, который отличается от запрашиваемого протоколом, доменом или портом.

Особенности CORS:

- Обеспечивает безопасность и не даёт вредоносным сайтам красть данные, которые принадлежат другим источникам.
- Позволяет серверам указывать сторонние источники, которые имеют право запрашивать у них ресурсы.
- При настройке веб-сайта механизм CORS позволяет выборочно блокировать различные категории доступа пользователя к ресурсам (запись, вставку или считывание).
- Для настройки CORS используются специальные заголовки запроса, например: Access-Control-Allow-Origin (указывает, откуда на сервер разрешены запросы), Access-Control-Allow-Methods (указывает, какие HTTP-методы разрешены для запросов на сервер) и другие.

Причины выбора:

- Безопасность: позволяет контролировать, какие домены могут делать запросы, и предотвращает атаки межсайтового сценария.

- Настройка запросов: позволяет гибко настраивать допустимые методы и заголовки HTTP-запросов.

7. Express (для серверной части)

Express — это быстрый, гибкий и минималистичный веб-фреймворк для приложений Node.js, который облегчает создание веб-приложений и API.

Некоторые особенности Express:

- Поддержка промежуточного программного обеспечения (middleware). С его помощью можно обрабатывать различные задачи, например, обрабатывать запросы, проводить аутентификацию и валидацию данных.
- Маршрутизация. Express предоставляет мощную систему маршрутизации, которая позволяет разработчикам определять, как приложение отвечает на разные HTTP-запросы (GET, POST, PUT, DELETE и др.).
- Поддержка механизмов создания шаблонов. Фреймворк поддерживает различные механизмы создания шаблонов, такие как Pug, EJS и Handlebars. Это позволяет разработчикам легко генерировать динамические HTML-страницы.
- Встроенные методы и утилиты для обработки HTTP-запросов и ответов. С их помощью можно отправлять ответы, устанавливать заголовки и обрабатывать ответы на ошибки.
- Интеграция с базами данных. Express не привязан к конкретной базе данных, что позволяет разработчикам использовать предпочитаемые системы баз данных, будь то реляционные базы данных, такие как MySQL и PostgreSQL, или базы данных NoSQL, например, MongoDB.
- Управление сессиями пользователей. Express предоставляет инструменты для обработки сессий пользователей, что упрощает реализацию аутентификации пользователя и поддержание данных сессий.

- Безопасность. Фреймворк включает встроенные функции безопасности, которые помогают защитить приложение от распространённых уязвимостей, таких как межсайтовый скриптинг (XSS) и подделка межсайтового запроса (CSRF).
- Масштабируемость. Express разработан с учётом производительности, он лёгкий и позволяет легко интегрировать кластеризацию и балансировку нагрузки для обработки высокой нагрузки трафика.

Причины выбора:

- Минималистичность: Простая структура и быстрая настройка для обработки запросов.
- Поддержка middleware: Широкие возможности для расширения функционала с помощью промежуточных обработчиков.
- Легкость интеграции: Быстрая настройка и совместимость с другими библиотеками Node.js, такими как JWT и CORS.

8. express-fileupload (для работы с файлами)

Express-fileupload — это простое промежуточное ПО для загрузки файлов в Express. Оно анализирует запросы multipart/form-data, извлекает файлы, если они доступны, и делает их доступными через свойство req.files.

Особенности Express-fileupload:

- позволяет настраивать поведение с помощью различных опций, например, указывать максимальный размер файла, разрешённые типы файлов и каталог назначения для загруженных файлов;
- предоставляет функцию mv(), которая используется для сохранения загруженного файла. В качестве параметров она принимает путь загрузки и функцию обработки ошибок.
- даёт доступ к загруженным файлам внутри POST-запроса с помощью свойства req.files.

Причины выбора:

- Легкость использования: простой интерфейс для загрузки файлов, который требует минимальной конфигурации.
- Гибкость: поддержка различных форматов файлов и возможность настройки правил обработки файлов (ограничение размера, фильтры по MIME-типу и т.д.).
- Интеграция: легко интегрируется с Express и другими инструментами для работы с файлами и данными.

Выбор этих технологий обеспечивает удобство разработки и поддержки веб-приложения, масштабируемость, безопасность и возможность дальнейшего расширения функциональности. Такой стек инструментов и средств позволяет эффективно реализовать функционал системы, учитывая ключевые особенности и требования к хронологическому ресурсу.

Конструкторская часть

Технологическая часть

Заключение

В ходе выполнения научно-исследовательской работы был проведен всесторонний анализ, охватывающий существующие решения, выбор подходящих технологий и проектирование клиент-серверной архитектуры для веб-приложения, позволяющего создавать и управлять хронологией событий.

Анализ существующих приложений показал, что на рынке отсутствуют решения, которые бы в полной мере удовлетворяли потребности пользователей в создании интерактивной и настраиваемой хронологии событий. Основными недостатками аналогов являются ограниченная функциональность, отсутствие гибкости в организации данных и недостаточная поддержка мультимедиа. Эти результаты обосновали необходимость создания нового приложения, способного устранить выявленные недостатки.

Разработанная архитектура приложения была спроектирована по принципу клиент-серверного взаимодействия. В рамках этой архитектуры клиентская часть обеспечивает удобный интерфейс для взаимодействия пользователей с системой, а серверная часть обрабатывает запросы, управляет данными и обеспечивает их защиту. Подобный подход позволяет разделить задачи обработки данных и отображения информации, что положительно сказывается на производительности и масштабируемости системы.

В ходе исследования также был определен набор инструментов, которые обеспечивают надежность работы системы, простоту ее сопровождения и возможность дальнейшего расширения функционала. Выбранные решения позволяют эффективно управлять данными, обеспечивать безопасность и создавать интуитивно понятный пользовательский интерфейс.

Таким образом, проведенная работа позволила разработать концепцию инновационного и универсального веб-приложения, которое превосходит существующие аналоги по функциональности, удобству использования и поддержке мультимедиа. Результаты исследования создают прочную основу для успешной реализации и внедрения системы.