Бюджетное образовательное учреждение г. Омска «Лицей №149»

Создание информационного ресурса на примере хранилища файлов

(практико-ориентированный)

Выполнил: ученик 10^4 класса
Прокофьев Максим
Руководитель: Ловыгина Надежда Васильевна, кандидат технических наук, доцент

Омск, 2022

ОГЛАВЛЕНИЕ

Введение……...…….……4

1. Общая информация о веб-приложениях……...…..……5

1.1 Понятие веб-приложения……..5

1.2 Структура веб-приложений……...5

1.3 Устройство серверной части веб-приложения……6

1.3.1 Веб-сервер……..6

1.3.2 База данных……6

1.4 Устройство клиентской части приложения……..7

1.5 Общение сервера с клиентом……8

2. Создание своего веб-приложения…….……...….9

2.1 Требования к приложению……9

2.2 Структура приложения……..9

2.3 Разработка серверной части приложения……9

2.3.1 Выбор веб-сервера……9

2.3.2 Выбор базы данных……10

2.3.3 Модели базы данных……...10

2.3.3.1 Структура таблицы «Users»……...10

2.3.3.2 Структура таблица «Photos»……..11

2.3.4 Механизм авторизации пользователей……..11

2.3.4.1 JWT……..11

2.3.4.2 Регистрация пользователей……...12

2.3.4.3 Авторизация пользователей……..12

2.3.5 Механизм обработки фотографий……...…..13

2.3.5.1 Загрузка фото на сервер…….....13

2.3.5.2 Получение всех фото……..13

2.3.5.3 Получение конкретного фото……13

2.4 Разработка клиентской части……..14

2.4.1 Выбор фреймворка……..14

2.4.2 Создание дизайна сайта……..14

Заключение……...15

Список литературы и Интернет-источников……16

ВВЕДЕНИЕ

Актуальность темы:

В последнее время у класса, в котором я учусь, появляется все больше совместных фотографий. Их становится настолько много, что нужно целое хранилище, чтобы вместить их все.

Я решил создать сайт-хранилище для всех этих файлов. На нем мои одноклассники смогут зарегистрироваться и просматривать фотографии, скачивать себе на устройство или загружать на сайт, чтобы не потерять их. Для похожих целей, несомненно, существуют облачные хранилища файлов, но, в отличие от них, хранение файлов на собственном сайте является бесплатным, а также при создании своего сайта я получу больше опыта в этой сфере, что является для меня очень важным, так как я хочу и дальше развиваться в сфере создания сайтов.

Цель:

Разработка веб-приложения, позволяющего загружать фотографии, хранить их и просматривать в любое время.

Для достижения своей цели я поставил следующие **задачи**:

1. Собрать и структурировать информацию о веб-приложениях и их создании
2. Определить требования для моего сайта
3. Составить структуру приложения
4. Разработать макет дизайна
5. Реализовать серверную и пользовательскую части веб-приложения
6. Запустить сайт в сети «Интернет»

1. ОБЩАЯ ИНФОРМАЦИЯ О ВЕБ-ПРИЛОЖЕНИЯХ

1.1 Понятие веб-приложения

Веб-приложение — клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными службами. Веб-приложения стали широко использоваться в конце 1990-х — начале 2000-х годов.

Существенное преимущество построения веб-приложений для поддержки стандартных функций браузера заключается в том, что функции должны выполняться независимо от операционной системы данного клиента. Вместо того, чтобы писать различные версии для различных операционных систем, приложение создаётся один раз для произвольно выбранной платформы и на ней разворачивается.

1.2 Структура веб-приложений

Веб-приложения можно разделить на несколько типов, в зависимости от разных сочетаний его основных составляющих:

Backend (бэкенд или серверная часть приложения) — работает на удаленном сервере и обрабатывает все запросы пользователя (браузера). При каждом переходе по ссылке браузер отправляет запрос на сервер. Сервер в свою очередь обрабатывает полученный запрос и формирует ответ, который отправляется на клиентскую часть приложения и обрабатывается непосредственно в ней. Бэкенд может быть написан на разных языках программирования: PHP, Python, Ruby, C# и других.

Frontend (фронтенд или клиентская часть приложения) — выполняется в браузере пользователя. Фронтенд это графический интерфейс, то, что вы видите на странице. Графический интерфейс отображается в браузере. Пользователь взаимодействует с веб-приложением именно через браузер, переходя по ссылкам и нажимая кнопки. Эта часть написана на языке программирования Javascript. Приложение может состоять только из клиентской части, если не требуется хранить данные пользователя дольше одной сессии. Это могут быть, например, фоторедакторы или простые игры.

В некоторых случаях веб-приложение может быть создано без использования веб сервера и серверной части в целом, например, когда вся информация обрабатывается прямо в браузере у пользователя. Также, если у приложения присутствует серверная часть, и оно должно обрабатывать какие-либо данные, то важной частью для его создания является база данных:

База данных (БД, или система управления базами данных, СУБД) — программное обеспечение на сервере, занимающееся хранением данных и их выдачей в нужный момент. База данных располагается на сервере, это может быть как сервер, на котором работает вся серверная часть приложения, так и на удаленный от него сервер. Серверная часть веб-приложения обращается к базе данных, извлекая данные, которые необходимы для формирования страницы, запрошенной пользователем.

1.3 Устройство серверной части веб-приложения

1.3.1 Веб-сервер

Серверная часть веб-приложения может быть написана на различных языках программирования, например Ruby, PHP, Java, Python, Javascript и других. Javascript — это язык программирования, изначально предназначенный для написания графического интерфейса на клиентской части, но на нем можно написать и серверную часть, благодаря существованию Node.js — платформы, превращающей Javascript в язык общего назначения посредством трансляции его в машинный код.

Все веб-приложения должны запускаться на веб-сервере. Веб-сервер — это сервер, который принимает HTTP (HyperText Transfer Protocol — «Протокол передачи гипертекста») запросы, обрабатывает их и выдает клиенту HTTP ответ, содержащий, какую-либо информацию. Самыми популярными веб-серверами на данный момент являются Apache, Nginx, Cloudflare. Node.js также может являться веб-сервером благодаря фреймворку ExpressJS, созданного специально для создания веб-приложений на платформе Node.js.

1.3.2 База данных

База данных — это совокупность данных, которые хранятся в соответствии со схемой данных. Они используются для постоянного хранения каких-либо данных на сервере. Базы данных классифицируются по модели хранения данных. Наиболее частыми видами являются иерархические и реляционные базы данных.

Иерархическая база данных — база данных, которая использует модель данных с древовидной структурой, состоящей из данных различных уровней. Например, иерархической базой данных является практически любая файловая система, которая состоит из корневого каталога и имеет иерархию в виде подкаталогов и файлов.

Реляционная база данных — база данных, которая использует модель данных, в которой различные объекты могут быть взаимосвязаны друг с другом. Это достаточно понятный, табличный способ представления данных, в котором каждая строка имеет свой уникальный идентификатор, благодаря которому можно устанавливать взаимосвязь между данными.

1.4 Устройство клиентской части веб-приложения

Клиентская часть веб-приложения — это скрипты, написанные на языке программирования Javascript. Эти скрипты выполняются в браузере пользователя, фактически от них зависит то, что сможет увидеть и сделать пользователь внутри приложения.

В классическом варианте создания сайтов клиентская часть используется 3 составляющие:

1. HTML (HyperText Markup Language — «язык гипертекстовой разметки») — страницы, содержащие контент, показывающийся пользователю
2. CSS (Cascading Style Sheets — «каскадные таблицы стилей») стили, которые описывают внешний вид веб-страницы
3. Javascript скрипты — программный код, который позволяет управлять поведением веб-страницы и ее содержанием.

Однако, концепция веб-приложения предполагает динамичное изменение контента, что становится возможно благодаря DOM (Document Object Model — «объектная модель документа»). DOM — программный интерфейс, который позволяет программам и скриптам получать доступ к содержимому HTML документов и изменять их содержимое, структуру и оформление во время выполнения программы. Для упрощения создания веб-приложения с использованием DOM были созданы Javascript библиотеки (*Фреймворки*), содержащие наборы классов и функций, обеспечивающих динамическое изменение отображения контента на веб-странице. Наиболее популярными фреймворками для создания веб-приложений на данный момент являются Angular, React, Vue.js. Все они различаются между собой, но глобально предоставляют примерно одинаковые возможности.

1.5 Общение сервера с клиентом

Для полноценной работы веб-приложения клиент должен обмениваться с сервером информацией, например при перезагрузке страницы или нажатии на кнопку. Общение клиента и сервера происходит по протоколу HTTP, основой которого является отправка клиентом запросов на сервер и получение различных ответов.

Современные веб-приложения в основном используют протокол HTTPS (HyperText Transfer Protocol Secure), который расширяет возможности протокола HTTP и является более защищенным благодаря поддержке шифрования. Использование зашифрованного канала данных считается хорошим тоном веб-разработки, независимо от важности передаваемых данных.

HTTP запросы бывают разных типов, но на практике с основном используется только GET (получить) и POST (отправить).

GET-запрос — метод, предназначенный для чтения данных с сервера. Например, при загрузке страницы клиент отправляет на сервер запрос на получение HTML документа, расположенного по указанному адресу

POST-запрос — метод, с помощью которого данные отправляются с клиента на сервер. Чаще всего с его помощью отправляются данные различных форм.

Все HTTP запросы имеют определенную структуру и состоят из двух частей — заголовка и тела запроса. В заголовке запроса описывается конечный URL-адрес, на который придет запрос, его тип и различная техническая информация. В теле запроса могут находиться различные параметры, данные, если это POST запрос, или тело запроса может быть пустым, если это GET запрос.

Когда сервер получает запрос, он должен его обработать каким-либо образом, а затем отправить ответ, в котором находится ответ с какими-либо данными и код ответа. Наиболее часто встречающимися кодами являются 200 — если все хорошо, 404 — если страница не найдена и 500 — если сервер не смог обработать запрос и произошла ошибка.

Теперь, когда у нас есть знания о том, что такое веб-приложение, какова структура серверной и клиентской частей, что такое веб-сервер и база данных, как сервер взаимодействует с клиентом, можно определить задачи для своего веб-приложения и начать его разработку.

2. СОЗДАНИЕ СВОЕГО ВЕБ-ПРИЛОЖЕНИЯ

2.1 Требования к приложению

1. Возможность доступа к каталогу файлов только определенных людей, после подтверждения регистрации администратором
2. Интуитивно понятный интерфейс для удобной навигации по сайту
3. Возможность просмотра фотографий в исходном разрешении без сжатия
4. Возможность скачать фотографии с сайта или поделиться ими в социальных сетях
5. Возможность загрузки на сайт целой группы фотографий одновременно

2.2 Структура приложения

Мое веб-приложение будет состоять из 3 основных частей:

Главная страница: страница, на которой будет располагаться каталог файлов с возможность их сортировки по дате создание, ссылки на страницу выхода из аккаунта, на окно добавления новых файлов (только для модераторов и администраторов) и ссылка на панель администрации (только для администраторов).

Страница авторизации пользователей: страница, на которую будут попадать все новоприбывшие пользователи. На ней пользователь может войти в аккаунт либо подать заявку на регистрацию нового аккаунта.

Панель администрации: страница, доступ к которой будут иметь только администраторы сайта. На этой панели администратор сможет получать доступ к списку пользователей и заявок на регистрацию на сайте, принимать или отклонять заявки на регистрацию и изменять роли уже существующих пользователей (назначать модераторами, администраторами и т.д.).

2.3 Разработка серверной части приложения

2.3.1 Выбор веб-сервера

Для выполнения своей цели я решил выбрать аппаратную платформу Node.js и для нее фреймворк веб-приложений Express.js Такой выбор я решил сделать потому, что в таком случае и клиентская часть приложения и серверная будут написаны на одном и том же языке программирования Javascript, что достаточно удобно, а также потому что ранее я уже

работал с платформой Node.js и имею опыт в разработке приложений для этой платформы. Также большим плюсом в сторону выбора Node.js является то, что на данный момент доступно около миллиона различных библиотек, работающих в Node.js.

2.3.2 Выбор базы данных

В данном проекте база данных требуется для двух вещей: чтобы хранить данные пользователей для их авторизации на сервере, и чтобы хранить информацию о загруженных фотографиях. В таком случае иерархическая база данных не требуется, так как все данные можно разместить в виде таблиц. Следовательно, нужно выбирать из реляционных баз данных. Я решил выбрать SQLite, так как это достаточно быстрая СУБД, которая позволяет хранить все данные в одном файле и не требует установки отдельного сервера базы данных.

В отличие от MySQL и других реляционных СУБД, SQLite встраивается прямо в приложение и работает как его составная часть. Для эффективного управления базой данных SQLite в Node.js существует *Sequelize*. Sequelize это ORM — технология программирования, которая связывает базы данных с концепциями объектно ориентированного программирования, создавая виртуальную объектную базу данных. С помощью этого фреймворка становится возможным обращаться к данным из файла базы данных с помощью языка Javascript как к обычному объекту языка.

2.3.3 Модели базы данных

Модель базы данных — то же, что и схема базы данных, то есть описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных.

Между таблицами внутри базы данных могут быть различные взаимосвязи. Например, один пользователь может иметь несколько загруженных фото, а одно фото может иметь только одного автора. Однако, для моих задач, такие взаимоотношения необязательны

Для хранения всех данных моего приложения достаточно двух таблиц. Я назвал их Users (Пользователи) и Photos (Фотографии).

2.3.3.1 Структура таблицы «Users»:

Id: уникальный идентификатор, который автоматически вносится в базу данных при появлении новой записи

nick: псевдоним пользователя, который он указал при регистрации

email: электронная почта, которую пользователь указал при регистрации

password: хэш пароля, который пользователь указал при регистрации и который используется для проверки правильного пароля при входе пользователя в аккаунт

comment: комментарий, который может написать пользователь для регистрации. Он может быть полезен, когда администратор решает вопрос о принятии заявки на регистрацию

role: роль пользователя на сайте

2.3.3.2 Структура таблицы «Photos»:

Id: уникальный идентификатор, который автоматически вносится в базу данных при появлении новой записи

Hash: хэш фотографии, посчитанный по алгоритму md5. Файлы, загруженные на сервер, получают название, состоящее из данного хэша. Это позволяет осуществить загрузку на сервер только уникальных файлов и уведомлять пользователя о том, что фото, которое он пытается загрузить, уже есть на сервере

Ext: расширение файла, загруженного на сервер, это позволяет избежать изменения расширения при загрузке, из-за которого файл будет невозможно прочитать

Name: название исходного файла

2.3.4 Механизм авторизации пользователей

2.3.4.1 JWT

JWT (JSON Web Token) — стандарт для создания токенов доступа, использующийся для передачи данных для идентификации в веб-приложениях. Такой токен состоит из заголовка и полезной нагрузки, закодированных алгоритмом Base64.

В заголовке токена указывается информация для описания токена: тип токена, алгоритм, используемый для его подписи или шифрования и тип содержимого.

В секции полезной нагрузки указывается пользовательская информация. Сервер определяет, какая именно информация будет предоставлена внутри токена

После того, как сервер отправил токен на клиентскую сторону, клиент при каждом запросе отправляет токен на сервер для его проверки и обновления. Если токен становится невалидным, то пользователь должен заново пройти процесс аутентификации. Главным преимуществом этой технологии является то, что серверу не нужно хранить информацию

обо всех выданных пользовательских сессиях, единственная задача сервера — это проверка токена на валидность.

2.3.4.2 Регистрация пользователей

Пользователь открывает страницу регистрации и вводит данные в специальную форму. После нажатия кнопки "Подать заявку" из данных пользователя формируется JSON (JSON — текстовый формат обмена данными, основанный на Javascript) объект, состоящий из введенной электронной почты, пароля, псевдонима и комментария. Данный объект отправляется на сервер с помощью HTTP запроса типа POST на сервер.

На серверной стороне из объекта получаются все нужные данные, после чего в базе данных происходит поиск записи, содержащей указанный адрес электронной почты. Если такая запись обнаруживается, то сервер отправляет HTTP ответ, в котором пишется, что пользователь с такой электронной почтой уже существует и пользователю предлагается войти в его аккаунт. Если такая запись не обнаружена, значит пользователь еще не подавал заявку на регистрацию, запускается функция регистрации пользователя.

В функции регистрации пользователя генерируется уникальный идентификатор. Пароль хэшируется, а затем шифруется с помощью специальной библиотеки для Node.js — bcrypt. Она позволяет шифровать пароль собственным алгоритмом, а также сравнивать 2 пароля, что используется при входе пользователя в аккаунт. В базе данных создается запись, в которую передаются все полученные ранее данные и присваивается роль пользователя GUEST (Гость).

После этого отправляется сообщение на клиент о том, что заявка на регистрацию подана успешно и JWT токен, содержащий такую информацию о пользователе, как его идентификатор, адрес электронной почты, псевдоним и роль на сервере

2.3.4.3 Авторизация пользователей

В форме авторизации пользователь указывает адрес электронной почты, на который он регистрировал аккаунт и пароль от его аккаунта. Формируется объект с указанными данными и с помощью HTTP запроса отправляется на сервер. На сервере из запроса получаются необходимые данных, затем в базе данных ищется запись, соответствующая переданному адресу электронной почты. Если такая запись не находится, то сервер возвращает ответ, в котором сообщается, что пользователь с таким электронным адресом не найден. Если запись находится, то переданный пароль сравнивается с паролем, указанным при входе.

Если пароли не совпадают, то сервер возвращает ответ, в котором сказано, что указан неверный пароль. В ином случае сервер генерирует JWT токен, содержащий информацию о пользователе и имеющий ограниченное время жизни. Данный токен отправляется сервером на сторону клиента, после чего клиент сохраняет его в локальное хранилище и при каждом запросе, требующем авторизации, отправляет его на сервер, где он проверяется на валидность с помощью промежуточного обработчика.

2.3.5 Механизм обработки фотографий

2.3.5.1 Загрузка фото на сервер

Из HTTP запроса типа POST сервер получает файл, либо список файлов и их данные. После этого в цикле каждый файл по очереди сохраняется с именем, состоящим из его md5 хэша и исходного расширения, при этом проверяется, есть ли уже в базе данных аналогичный файл. Если такой файл есть, то он не записывается в базу данных, а на клиент отправляется ответ, в котором говорится, что такой файл уже существует. Если такого файла еще нет, то он сохраняется в локальном хранилище на диске и в базу данных вносится запись, в которой указано его название, хэш, расширение, дата создания файла и идентификатор пользователя, загрузившего данный файл. После чего на клиент отправляется ответ, который содержит информацию о том, сколько файлов было загружено успешно.

2.3.5.2 Получение всех фото

На клиентской части фотографии отображаются в виде каталога с постраничным выводом. Следовательно, когда клиент отправляется запрос о получении фото, в нем указывается страница, которая показывается пользователю в данный момент и лимит фотографий на одной странице. Также пользователь может группировать файлы по дате их создания, если эта группировка включена, то информация о ней тоже поступает в запросе.

Сервер выбирает из базы данных записи, удовлетворяющие критериям и формирует на их основе ответ, включающий в себя названия файлов, хранящихся на сервере, который отправляет клиенту. Клиент, в свою очередь, для отображения файлов в каталоге, обращается к фотографиям по прямой ссылке, состоящей из адреса сайта и названия файла на сервере.

2.3.5.3 Получение конкретного фото

Для получения конкретной фотографии клиент отправляет POST запрос, в теле которого содержится идентификатор целевой фотографии. Сервер получает из тела запроса этот

идентификатор, после чего выполняет поиск в базе данных и возвращает данные об этой фотографии в своем ответе клиенту.

2.4 Разработка клиентской части

2.4.1 Выбор фреймворка

В качестве основы клиентской части моего веб-приложения я решил выбрать *React*. React — это Javascript библиотека для разработки пользовательских интерфейсов внутри веб-приложений.

Я решил выбрать именно этот фреймворк, так как он использует DOM, благодаря которому веб страница быстрее загружается и становится более отзывчивой. Также, при работе с React можно использовать написанные компоненты по несколько раз в разных частях кода или даже проектах.

React использует JSX — расширение синтаксиса для создания объектов Javascript с синтаксисом HTML, такой код впоследствии компилируется в обычный Javascript код

2.4.2 Создание дизайна сайта

Для создания макета дизайна своего приложения я решил использовать онлайн-сервис, который называется *Figma.* Figma позволяет создавать как простые прототипы интерфейсов, так и детально проработанные дизайны мобильных приложений, веб-сайтов и т.д.

Главным плюсом использования Figma для веб-разработчика является возможность сразу видеть CSS стили, которыми обладает каждый объект, созданный в Figma. Это позволяет во много раз упростить создание дизайна уже в программном коде.

Благодаря использованию описанных мною выше технологий, я реализовал серверную часть веб-приложения при помощи веб-сервера Express и базы данных SQLite. Затем я создал макет дизайна при помощи Figma и реализовал клиентскую часть на фреймворке React.

ЗАКЛЮЧЕНИЕ

Я ознакомился с информацией о веб-приложениях. Узнал, что такое веб-сервер и база данных, как сервер взаимодействует с клиентом и какие протоколы для этого использует. Применив изученные технологии, я создал свое веб-приложение, которое позволяет хранить фотографии на сервере и показывать их определенному кругу лиц.

Теперь у меня и моих одноклассников будет доступ веб-сайту, на котором можно будет удобно просматривать совместные фотографии, загружать их туда и не бояться, что эти фотографии могут попасть в руки посторонним лицам.

Я считаю, что данное веб-приложение в дальнейшем можно развивать в различных направлениях. Например, добавить комментарии к фотографиям и отметки «нравится» Также можно будет купить виртуальный выделенный сервер для того, чтобы веб-приложение было запущено на нем, это позволит сделать его работу постоянной и более быстрой.

СПИСОК ЛИТЕРАТУРЫ И ИНТЕРНЕТ-ИСТОЧНИКОВ

1. Бэнкс Алекс, Порселло Ева. GraphQL: язык запросов для современных веб-приложений. — СПб.: «Питер», 2019. — С. 240.

2. Бэнкс Алекс, Порселло Ева. React и Redux: функциональная веб-разработка. — СПб.: «Питер», 2018. — С. 336.

3. Кирупа Чиннатамби. Изучаем React. — СПб.: «Питер», 2019. — С. 368

4. Мардан Азат. React быстро. Веб-приложения на React, JSX, Redux и GraphQL. — СПб.: «Питер», 2019. — С. 560

5. Марко Беллиньясо. Разработка Web-приложений в среде ASP.NET 2.0: задача — проект — решение = ASP.NET 2.0 Website Programming: Problem — Design — Solution. — М.: «Диалектика», 2007. — С. 640

6. Олищук Андрей Владимирович. Разработка Web-приложений на PHP 5. Профессиональная работа. — М.: «Вильямс», 2006. — С. 352

7. Томас Марк Тиленс. React в действии. — СПб.: «Питер», 2019. — С. 368.

8. Как работают веб-приложения: [Электронный ресурс] // https://habr.com/ru/post/450282/

9. Express.js [Электронный ресурс] // https://expressjs.com/ru/

10. React [Электронный ресурс] // https://ru.reactjs.org/