СОЗДАНИЕ ПРИЛОЖЕНИЯ ДЛЯ АВТОМАТИЧЕСКОГО СБОРА ID СТРАНИЦ В СОЦИАЛЬНОЙ СЕТИ

Программа: Разработчик

Специализация: Веб-разработка на Java

Мельник Андрей Викторович

г. Псков

2024 год

**АННОТАЦИЯ**

С каждым годом социальные сети приобретают всё большую значимость в жизни людей. Действительно, на заре своего появления социальные сети использовались только для общения, сейчас же на базе социальных сетей распространяются товары и услуги, строится бизнес, а профессия “блогер” больше не воспринимается как что-то нелепое. Не обошли социальные сети и магазины, количество которых с каждым днём только увеличивается, отчего слова “онлайн-магазин на базе социальной сети” также перестали быть чем-то новым.

Задача данной работы заключалась в создании удобного сайта, который поможет владельцам онлайн-магазинов на базе социальной сети ВКонтакте собирать имена и id страниц комментаторов из-под поста и формировать списки заказов, например, заказы к 8 марта, к новому году, ко дню рождения и так далее; позволит редактировать уже сформированные списки заказчиков, а также делать общую рассылку всем участникам списка.

В ходе работы был разработан сайт “GetId”, который удовлетворяет всем вышеуказанным требованиям, а именно пользователь имеет возможность создать учётную запись с личным кабинетом, в котором будут храниться все списки. Пользователь также может удалять списки, редактировать их: удалять или добавлять новых участников, - и делать рассылки всем участникам списка.

Таким образом, поставленная задача была успешно выполнена.

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc162107985)

[**ОСНОВНАЯ ЧАСТЬ** 5](#_Toc162107986)

[1. Постановка задачи 5](#_Toc162107987)

[2. API или программный интерфейс приложения 5](#_Toc162107988)

[2.1. Типы API 6](#_Toc162107989)

[2.2. Классификация API по сценариям использования 7](#_Toc162107990)

[2.3. Протоколы API 8](#_Toc162107991)

[2.4. Vk API 10](#_Toc162107992)

[3. Стек технологий 11](#_Toc162107993)

[3.1. Стек технологий сайта GetId 12](#_Toc162107994)

[4. Базы данных 17](#_Toc162107995)

[5. Сессия и авторизация 21](#_Toc162107996)

[6. Безопасность сайта 25](#_Toc162107997)

[7. Интерфейс сайта 30](#_Toc162107998)

[8. Анализ программного кода 35](#_Toc162107999)

[**ЗАКЛЮЧЕНИЕ** 38](#_Toc162108000)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ** 39](#_Toc162108001)

[**ПРИЛОЖЕНИЕ А** 40](#_Toc162108002)

[**ПРИЛОЖЕНИЕ Б** 41](#_Toc162108003)

[**ПРИЛОЖЕНИЕ В** 42](#_Toc162108004)

[**ПРИЛОЖЕНИЕ Г** 44](#_Toc162108005)

[**ПРИЛОЖЕНИЕ Д** 46](#_Toc162108006)

**ВВЕДЕНИЕ**

В современном мире нельзя недооценивать влияние интернета на повседневную жизнь. Действительно, мессенджеры заменяют нам общение в реальной жизни, видео-площадки - походы в кинотеатры, а возможность заказать еду - походы в кафе. Не стоит забывать и о покупках в интернете. Сейчас существует сотни онлайн-магазинов разной направленности: от одежды до книг, от еды до бытовой химии, от товаров для животных до покупки этих самых животных онлайн. Поскольку конкуренция таких магазинов достаточно велика, продавцы заботятся не только о качестве продаваемых товаров, но и об удобстве и интуитивной понятности сайтов.

Однако не все магазины имеют свой собственный сайт, некоторые из них расположены на популярных интернет-площадках, например, ВКонтакте, и если интерфейс подобных площадок всё также остаётся удобным для покупателя, то далеко не всегда он удобен для продавца. Например, зарегистрировав группу по производству ароматизированных свечей в социальной сети ВКонтакте, продавец может выставлять фотографию своего изделия, а покупатели, желающие приобрести свечу с запахом корицы или лаванды, оставляют комментарии под фотографией. Если таких комментариев наберётся не более десяти, продавец может самостоятельно собрать нужные ему данные и разослать сообщения с полезной информацией вручную, однако, что делать, если в комментариях собралось десятки или сотни человек?

**ОСНОВНАЯ ЧАСТЬ**

1. Постановка задачи

Задача данной работы заключается в создании удобного сайта, который поможет организатору группы собирать имена и id страниц комментаторов из-под поста в социальной сети Вконтакте и формировать списки различного назначения: например, заказы к 8 марта, к новому году, ко дню рождения и так далее; позволит редактировать уже сформированные списки, а также делать общую рассылку всем участникам списка.

2. API или программный интерфейс приложения

Для создания сайта, отвечающего всем перечисленным в постановке задачи требованиям, необходимо познакомиться с понятием API (англ. Application programming interface). Программный интерфейс приложения, интерфейс прикладного программирования, или же API - это набор функций, который позволяет одному программному обеспечению обмениваться данными с другим. Например, когда пользователь открывает приложение ВКонтакте на телефоне, оно посылает запрос на сервер, который, в свою очередь, считывая запрос, отправляет необходимые данные назад в телефон. Чтобы лучше раскрыть суть данного термина, можно привести следующий пример: представим, что пользователь приложения - это посетитель ресторана, само приложение - меню, а сервер - это кухня. Посетитель выбрал самое вкусное блюдо в меню, но как же повар узнает, что именно нужно приготовить? Посредником, который отдаст новый заказ на кухню, а затем вернёт ответ сервера, то есть готовое блюдо, посетителю - это официант, и в этом примере он является тем самым связующим звеном - API. Стоит ещё раз отметить, что любой API состоит из вызова функций, которые требуют от программного обеспечения выполнения определённых действий. Описание функций, как правило, содержится в документации.

Прикладной программный интерфейс упрощает и ускоряет разработку программного обеспечения, поскольку позволяет добавлять функционал к старым приложениям или писать новые программы с использованием уже существующих технологий, а благодаря документации разработчику не нужно знать, как именно работает та или иная составляющая API, достаточно знать, какой функционал она предоставляет. Подводя итоги, можно отметить, что API является связующим звеном двух систем, скрывающим детали реализации и сложности некоторых деталей от разработчика.

2.1. Типы API

Прикладной программный интерфейс можно разделить на три категории: приватный, партнёрский и публичный. Раскроем суть каждого термина подробнее.

Приватный API. Этот тип API используется внутри компаний и создан для того, чтобы разработчики могли интегрировать более новые системы и приложения в уже существующие или, наоборот, создавать новые продукты, используя уже готовые внутренние наработки. Даже если итоговый продукт становится публичным, приватный API доступен только разработчикам внутри компании - такой подход позволяет контролировать использование API и уберегает программы от злонамеренного вмешательства.

Партнёрский API, как и приватный, не даёт общего доступа к использованию ресурсов компании, но используется в случае сотрудничества компаний, когда одна сторона предоставляет своё программное обеспечение второй для достижения общих целей. Нельзя не отметить, что использование партнёрского API является своего рода рекламой для стороны, предоставляющей своё прикладное программное обеспечение партнёру.

Наконец, публичный API доступен для любых разработчиков, однако делится на два типа: открытый и коммерческий. Из названия понятно, что открытый API может быть использован без каких-либо ограничений, соглашений и дополнительных лицензий, а любая документация, относящаяся к открытому типу публичного API также находится в свободном доступе. Владельцы коммерческого открытого API взимают плату за использование их технологий, но, как правило, предлагают пробную бесплатную версию функционала.

2.2. Классификация API по сценариям использования

API можно классифицировать не только по параметру приватности, но и по системам, для которых они спроектированы. Выделим следующие типы: API для баз данных, для операционных систем, удалённые и web API. Рассмотрим подробнее каждый из представленных типов.

API для баз данных позволяет установить связь между приложением и системой управления базами данных. Разработчики общаются с базой данных посредством запросов для доступа к данным и таблицам, и такие сервисы, как The Drupal 7 Database API позволяют пользователям писать единые запросы для разных баз данных, таких как Oracle или MongoDB.

API для операционных систем определяет, как приложения используют ресурсы и различные сервисы ОС. Стоит отметить, что каждая из множества операционных систем, например, Windows, Linux или MacOS, содержит свой набор API и соответствующую документацию.

Удалённые API определяют правила взаимодействия для приложений, которые находятся на удалённых машинах. Другими словами, программное обеспечение запрашивает доступ к ресурсам, расположенным на другом устройстве. С появлением сетей, вроде Интернета, API стали основываться на web стандартах. Java Database Connectivity API - один из примеров удалённого прикладного программного интерфейса.

Наконец, рассмотрим понятие web API. Этот тип API предоставляет функциональную передачу между двумя web-based системами, которые представляют собой клиент-серверную архитектуру. Такой прикладной программный интерфейс главным образом доставляет запросы от web-приложения и ответы от сервера, используя HTTP-протокол. Разработчики могут использовать этот тип API, чтобы расширить функционал своего приложения или сайта. Например, социальная сеть ВКонтакте позволяет магазинам, интернет-площадки которых находится не только на отдельном сайте, но также и в группе во ВКонтакте, связать эти два ресурса между собой посредством добавления чата сообщества прямиком внутрь сайта магазина.

2.3. Протоколы API

Для того, чтобы использовать различные типы API, необходимо следовать специальным протоколам, которые содержат различные правила, указывающие на доступные типы данных и команды. Наиболее известными протоколами являются: REST, SOAP, RPC. Рассмотрим каждый из них по порядку.

Термин REST был введён в 2000 году, и не является протоколом, а скорее представляет собой архитектуру построения модели, состоящую из шести постулатов:

* клиент-серверная архитектура: интерфейс отделён от серверной части, - такой подход обеспечивает гибкость приложения и позволяет различным компонентам развиваться независимо друг от друга;
* отсутствие состояния означает, что клиентские данные не сохраняются на сервере между запросами;
* универсальный интерфейс предполагает наличие единого пути взаимодействия с сервером, независимо от устройства или типа приложения;
* кэшируемость позволяет сохранять ответы сервера, поэтому ответ по протоколу REST API должен содержать переменную, которая сообщает о том, могут эти данные быть закэшированы или нет;
* архитектура системы представляет собой слои, причём слои ничего не знают друг о друге. Между клиентом и сервером может быть много промежуточных уровней, что позволяет улучшить доступность системы за счёт распределения нагрузки и предоставления общих кэшей.

REST позволяет использовать данные как ресурсы, каждый из которых представляется в виде уникального URL, запросив который, можно получить доступ к необходимым данным. API, основанные на REST-архитектуре, используют HTTP запросы, такие как get, post, connect и другие, чтобы работать с ресурсами. Крупные бренды, вроде Google или Netflix используют REST API, чтобы повысить узнаваемость своего бренда.

Протокол SOAP или протокол простого доступа к объектам - надёжный протокол, являющийся одним из типов web API, который используется с конца 1990-х годов. Он является расширяемым, независимым и нейтральным, то есть способным работать с различными протоколами связи (такими как HTTP, SMTP, TCP и другими). Спецификация этого протокола включает в себя:

* модель обработки, которая указывает, как именно стоит обрабатывать сообщения SOAP;
* правила привязки протокола, содержащие правила использования SOAP с базовыми протоколами, такими как HTTP;
* конструктор сообщений, выстраивающий структуру сообщений SOAP.

Стоит отметить, что этот протокол первым стандартизировал использование сетевых подключений приложениями. Однако SOAP содержит строгие правила и жёсткие стандарты, которые, порой, являются ресурсоёмкими, поэтому многие разработчики выбирают REST, нежели SOAP.

Наконец, поговорим о протоколе RPC или о протоколе удалённого вызова процедур. Эти протоколы являются самыми старыми и самыми простыми типами API. Главной целью RPC является организация распределённых вычислений. Наибольшая эффективность достигается в приложениях с малым временем ответа и малым объёмом передаваемых данных. RPC можно разделить на две группы: XML-RPC и JSON-RPC, - которые, как ясно из названия, используют форматы XML и JSON для передачи данных соответственно. Однако RPC-протоколы очень сложно поддерживать или обновлять, поскольку из-за плотной связи RPC API разработчику нужно хорошо понимать, как изменение может повлиять на другие составляющие этого протокола.

Таким образом, наиболее предпочитаемым стандартом для построения приложений, которые связаны по сети, является архитектура REST, поскольку она содержит в себе все стандарты, лежащие в основе всемирной паутины и является более простой, нежели web сервисы, основанные на SOAP. И в отличие от RCP, слабосвязанная многоуровневая архитектура позволяет разработчикам легко вносить изменения.

2.4. Vk API

API ВКонтакте - это специальный интерфейс, который позволяет получать информацию из баз данных сайта vk.com, используя http-запросы к серверу. Удобство использования API, как уже было сказано ранее, заключается в том, что разработчику не нужно знать устройство базы: из каких таблиц и полей она состоит, как эти таблицы связаны между собой, - достаточно только осуществлять уже готовые API-запросы. Синтаксис запросов и документацию можно посмотреть непосредственно на сайте vk.com в разделе “Для разработчиков”.

Все запросы имеют следующий формат:

<https://api.vk.com/method/wall.getComments?owner_id=277517137&post_id=238> 3&v=5.52.

Рассмотрим все его составляющие по порядку:

* http:// - протокол соединения;
* api.vk.com/method - адрес API-сервиса;
* wall.getComments - метод API Вконтакте, позволяющий получить комментарии из-под поста на стене пользователя или сообщества;
* ?owner\_id=277517137&post\_id=2383 - параметры запроса. Полный список можно посмотреть в описании метода. Здесь разработчик сообщает серверу, что хотел бы получить комментарии из-под записи на стене пользователя с id страницы 277517137, id поста 2383;
* &v=5.52 - версия API.

Сервер вернёт JSON-объект с данными, которые мы запросили. В случае ошибки, разработчик также получит соответствующее сообщение. Стоит пояснить, что формат JSON - это текстовый формат представления данных, особенностью которого является представление информации в виде объектов языка JavaScript. Первоначально, как следует из определения, этот формат создавался для языка JavaScript, однако позже, благодаря удобству представления данных в таком виде, во многие языки были добавлены библиотеки для работы с этим форматом данных.

В данном случае, когда под постом находится всего один комментарий, причём оставленный самим пользователем, ответ сервера будет выглядеть следующим образом:

{"response":[{"first\_name":"Таня","last\_name":"Фёдорова",id: 277517137}]}

Удобство работы с API заключается ещё и в том, что в документации заранее обговорён формат ответа, что значительно упрощает процесс разработки.

3. Стек технологий

Архитектуру сайта, представленного в данной курсовой работе, можно разделить на две составляющие: frontend или клиентскую составляющую, то есть ту часть веб-сайта, с которой взаимодействует пользователь, и backend или серверную составляющую - ту часть веб-сайта, которая отвечает за функционирование и обеспечение данными клиентской части. Иными словами, при разработке был применён принцип разделения ответственности между внешним представлением и внутренней реализацией. И для каждой стороны используется разный стек технологий. Раскроем термин подробнее, а заодно обозначим важные моменты при подборе его составляющих.

Итак, стек технологий - это набор инструментов, которые используются при создании проекта, включающий в себя язык программирования, системы управления базами данных, фреймворки, компиляторы и так далее. Безопасность, масштабируемость и поддержка приложений, а также стоимость - те важные критерии, на которые нужно обратить внимание при выборе технологического стека. Безопасность любого приложения, безусловно, важна, поскольку злоумышленники будут пытаться взломать приложение и получить ценнейший ресурс двадцать первого века - данные, любую информацию. Большинство популярных фреймворков содержат необходимые компоненты безопасности, подключение и работа которых описаны в документации. Масштабируемость проекта означает, что выбранный разработчиком стек технологий в любой момент позволит расширить функционал без особых сложностей, а поддержка приложений в стеке обеспечит и хорошую техническую поддержку разрабатываемого проекта. Стоимость составляющих стека технологий сильно влияет и на стоимость конечного продукта, поэтому важно выбирать не самый дорогой фреймворк из всех представленных, а соотносить цену и качество предоставляемых возможностей.

3.1. Стек технологий сайта GetId

На стороне frontend в стек технологий вошли следующие компоненты: язык гипертекстовой разметки, сокращённо html, каскадные таблицы стилей, или css, шаблонизатор handlebars, и язык программирования JavaScript, отвечающий за интерактивную часть сайта. Для разработки backend был выбран Nodejs и фреймворк Express, а также база данных MongoDB.

Связка html-css-js - первое, с чем знакомится начинающий разработчик сайтов. HTML используется для описания разметки страницы, определяет положение элементов, которые затем увидит пользователь. CSS используется для стилизации документа, позволяя применять различные стили к конкретным элементам страницы с помощью так называемых селекторов. Пример использования HTML-разметки и применения CSS-стилей к элементам разметки можно увидеть в Приложении А и Приложении Б. JavaScript позволяет добавить динамики на веб-страницу, управлять элементами и контролировать взаимодействие с интерфейсом, например, передавать сигнал с кнопки, которую нажал пользователь, на backend. Отдельно стоит выделить шаблонизатор handlebars, который значительно уменьшает количество html-кода. Первое, что необходимо сделать при работе с handlebars - разметить html страницу на составляющие, то есть сделать макет, как показано на рисунке 1.

{{> head }}

<body>

{{> navbar }}

{{{ body }}}

{{> footer }}

</body>

<]/html[>|

*Рисунок 1 - Макет*

Здесь head - это так называемая голова html - документа (рисунок 2). Здесь подключаются все стили и мета-теги.

<!DOCTYPE html>

<html lang = "ru">

<head>

cmeta charset = "utf-8">

^meta name="viewport" content="width=device-width, initial-scale=1.0"|>|

<link rel = "stylesheet" href = "/general.css">

<link href = "https://fonts.googleapis.com/css2?family=Kaushan+Script&family=Montserrat:wght@400;700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="[https://cdnjs.cloudflare.com/аjax/libs/font-awesome/4.7.0/css/font-awesome■min.css](https://cdnjs.cloudflare.com/%d0%b0jax/libs/font-awesome/4.7.0/css/font-awesome%e2%96%a0min.css)">

<title>{{ title }}</title>

</head>

*Рисунок 2 - Head*

Далее navbar - меню приложения (рисунок 3), которое, если разработчик не укажет иного, будет присутствовать на каждой странице сайта.

<header class = ,rheader">

<div class = "container">

<div class = "header inner">

| <div class = "header logo"xa href = "/">GetId</ax/div>

<nav class = "nav">

class = "nav link" href = "/getList"^5bicTpbiH список</а>

{{#if isAuth}}

<a class = "nav link" href = "/add" Сформировать списокс/а>

<a class = "nav link" href = "/userLists">Мои списки</а>

| <a class = "nav link" href = "/auth/logout">BuiiTH</a>

{{else}}

| <a class = "nav link" href = "/auth/login">BoSTH</a>

{{/if}}

</nav>

</div>

</div>

</header>

*Рисунок 3 - Navbar*

Body или тело - это основная часть сайта, с которой взаимодействует пользователь. Для каждой страницы она прописывается в отдельных файлах, и это единственная часть, которую разработчику нельзя полностью шаблонизировать, а нужно постоянно добавлять новые элементы или же полностью менять концепцию страницы. Наконец, footer (рисунок 4), это та часть html-страницы, которая подгружается последней, то есть после того, как остальные части уже готовы к работе, поэтому в footer выносится подключение всех файлов, написанных на JavaScript, а также, по желанию заказчика, контакты владельца страницы.

<script src="/addFields.js"></script>

<script src=''/deleteFields. j s" x/script >

<script src="/login.js”></script>

*Рисунок 4 - Footer*

Изначально JavaScript был создан исключительно для работы с клиентской частью сайтов, однако вскоре было решено расширить возможности языка, так и появилась серверная платформа Nodejs, которая превращает JavaScript из узкоспециализированного языка в язык общего назначения, позволяя писать код, предназначенный для выполнения на сервере, на языке JavaScript. В чём же заключаются преимущества Nodejs? Во-первых, стала возможна full-stack разработка с использованием одного языка программирования и, соответственно, код клиентской и серверной частей стало легче согласовывать и поддерживать, собственные библиотеки JavaScript стали доступны и на серверной части. Во-вторых, код, написанный на JavaScript, является более компактным, нежели тот же код, написанный на языке C, а высокая производительность языка позволяет использовать его там, где скорость играет не последнюю роль. Также высокую скорость выполнения обеспечивает возможность Nodejs преобразовывать JavaScript в машинный код, а не интерпретировать его. В качестве третьего преимущества можно выделить open source, или открытое программное обеспечение. Это означает, что исходный код проекта сделали доступным для возможности модификации и доработки сторонними разработчиками, из чего можно сделать вывод, что Nodejs постоянно развивается и совершенствуется. Отдельным, четвёртым положительным качеством Nodejs стоит отметить менеджер пакетов npm, с помощью которого любой нужный пакет устанавливается и автоматически сохраняется в список зависимостей для разработки при вводе в строку одной команды, тогда как при его отсутствии установка пакета занимала в разы больше времени.

Expressjs - это фреймворк для Nodejs, который предоставляет большой функционал для веб-приложений:

* добавление дополнительных миддлвэйров, или промежуточного программного обеспечения, для обработки запросов;
* набор общих настроек веб-приложения, таких как порт для соединения и шаблонов для отображения ответа;
* написание обработчиков для запросов с разными HTTP-методами и URL-адресами;
* предоставление функционала для защиты приложений.

4. Базы данных

На начальном этапе создания проекта данные личного аккаунта пользователя хранились в json формате, то есть в качестве баз данных использовалась файловая система, однако такой метод имеет ряд существенных минусов. Во-первых, скорость доступа к информации при большом объёме данных. Во-вторых, в файловой системе нельзя настроить взаимосвязь между таблицами, которая в значительной степени облегчает доступ к необходимой информации, а также помогает связать данные в единую систему.

Итак, базы данных - это упорядоченные наборы данных, которые хранятся в электронном виде, позволяют легко изменять, обрабатывать или связывать между собой информацию. Управление базами данных осуществляется через систему управления базами данных, сокращённо СУБД, которая является высокоуровневым программным обеспечением, работающим с низкоуровневыми структурами. Проще говоря, СУБД необходима для того, чтобы сохранять, контролировать и извлекать данные из базы, а также осуществлять запросы. Все базы данных можно разделить на так называемые модели, которые различаются способами обработки данных, и каждая СУБД представляет собой реализацию одной из моделей баз данных. На данный момент самой популярной является реляционная модель, однако существуют задачи, решение которых она не может предоставить. Именно поэтому начинает набирать популярность так называемая NoSQL, или нереляционная, модель, которая не использует строгую структуризацию данных и даёт большую свободу действий при обработке информации, нежели реляционная модель.

Реляционная модель является табличным способом представления данных. Каждый столбец содержит атрибут - элементы информации, например, имя или должность. Ячейки таблицы содержат значения атрибутов: Иванов Иван Иванович, если атрибутом является имя; менеджер/строитель/уборщик - в тех случаях, когда атрибутом является должность. Такая система позволяет установить взаимосвязь между элементами данных. Каждая строка таблицы представлена уникальным набором связанных значений, которые относятся к одному конкретному объекту. Первичный ключ - это уникальный идентификатор, которым может быть помечена строка таблицы, а строки нескольких таблиц связаны между собой внешними ключами. Заканчивая описание реляционной модели, можно вспомнить некоторые популярные реляционные СУБД, такие как SQLite, MySQL или же PostgreSQL.

Нереляционная, или NoSQL- модель, в отличие от реляционной, не использует табличный способ представления данных. Базы данных, основанные на NoSQL моделях, предполагают создание уникальной системы хранения информации, оптимизированной под конкретные требования хранимых данных. Например, данные могут храниться в формате .JSON, в виде графов или же парами ключ-значение. В зависимости от выбранного формата хранения, доступ к данным осуществляется по-разному, то есть от формата хранения данных зависит формат запроса.

При создании сайта GetId была задействована нереляционная модель с форматом хранения данных в виде .JSON документов. Стоит отметить, что термин NoSQL применяется в отношении нереляционных баз данных, он также означает те хранилища данных, которые не используют язык запросов SQL, однако нереляционные базы данных поддерживают запросы, которые совместимы с SQL. На данный момент, самой популярной нереляционной СУБД является MongoDB, которая позволяет группировать коллекции данных с другими базами данных.

Концепция сайта GetId заключается в сборе комментариев и формировании списков. Всю эту процедуру можно описать как работу с документами. MongoDB является документоориентированной СУБД, а значит как нельзя кстати подходит для работы с иерархическими структурами данных. Существенным минусом MongoDB считается отсутствие связей между документами и коллекциями. Отчасти эта проблема решается с помощью так называемых [Database Reference](http://docs.mongodb.org/manual/reference/database-references/) (ссылки на коллекции), но в случае, когда связи между данными необходимы, лучше воспользоваться другой СУБД. Структура хранения данных сайта GetId не предусматривает большого количества взаимосвязей, а сама MongoDB быстрее осуществляет чтение и запись, нежели традиционные SQL-системы, что делает эту СУБД идеальной для проекта.

Первым шагом является вызов команды “npm install mongoose” в командной строке VisualStudio Code, которая позволяет установить пакет mongoose, необходимый для работы с MongoDB. В первой строке (рисунок 5) из библиотеки mongoose вызывается класс Schema, с помощью которого в дальнейшем будет создан объект пользователя. Начиная с третьей строки, создаётся объект userSchema, описывающий конкретного пользователя. Каждый пользователь имеет уникальные поля: name и password, которые заполняются в момент регистрации; userData - это поле, содержащее в себе массив items, где будут храниться сформированные в будущем списки, а также в userData хранится уникальный id каждого списка. Поле id не прописано явно, поскольку оно задаётся автоматически при записи в MongoDB. Каждое поле объекта содержит в себе свойство required, которое означает, что это свойство необходимо для создания модели. Действительно, личный аккаунт пользователя невозможен без логики, то есть без имени, и пароля, а сформированному списку необходимо название, title, и сами пользователи, vkUsers.

const {Schema, model} = require('mongoose');

const userSchema = new Schema({ name: {

type: String, required: true, }, password: { type: string, required: true, b userData: { items: [{ title: { type: string, required: true, b vkUsers: {

type: Array, required: true, b }b b groupToken: { type: string, } });

*Рисунок 5 - Создание схемы пользователя для базы данных*

На рисунок 6 показано, как вся эта информация записывается в базу данных.

id: Objectld("60995al430f88a0d2c245200")

userData: Object

v items: Array

0: Object

«■ vkUsers: Array

v 0: Object

first\_name: "Никита"

lastname:"Березовский"

profile\_id: 138301832

1: Object

first\_name: "Таня"

lastname:"Федорова"

profileid:277517137

\_id: Objectld("609aad5482e01b00153ab5ce")

title:"8 марта”

name:"Irina"

password: "$2a$10$ "

*Рисунок 6 - Модель внутри базы данных*

Как мы видим, при регистрации пользователь указал имя Irina, ввёл пароль, но в MongoDB он представлен в зашифрованном виде. О методах шифрования пароля будет написано в главе 6 «Безопасность сайта». После регистрации, пользователь сформировал список с названием «8 марта», а комментарии под постом оставили двое пользователей.

5. Сессия и авторизация

Функционал сайта GetId делится на две большие категории: для авторизованных пользователей и для тех, кто предпочёл пропустить регистрацию. Это означает, что недоступный неавторизованному пользователю функционал скрыт паролем. Но как при переходе с одной страницы на другую дать понять сайту, что пользователь является авторизованным? Здесь появляется понятие сессии - механизма временного хранения и передачи информации между скриптами. Чтобы начать работу с сессиями в Nodejs, необходимо установить пакет “express-session”, который является частью фреймворка Express, затем нужно подключить пакет

const session = require('express-session');

и приступать к созданию сессии (рисунок 7)

50 app.use(session({

secret: keys.SESSION SECRET,

52 resave: false,

saveUninitialized: false,

54 store: store,

55 }));

*Рисунок 7 - Создание сессии*

Поле secret в 51 строке рисунка 7 передаётся строка, на основе которой осуществляется шифрование сессии. Resave - параметр, который принимаем булево значение, и указывает, нужно ли пересохранять сессию, saveUninitialized также является булевым значением, с помощью которого можно решить, будут ли попадать в хранилище пустые сессии. Наконец, store - это экземпляр хранилища для хранения сессии. Для его использования необходимо подключить библиотеку “connect-mongodb-session”:

const MongoStore = require('connect-mongodb-session')(session);

Затем нужно создать объект store (рисунок 8)

const store = new MongoStore({

34 collection: ’sessions',

35 uri: keys.MONGODB JJRI,

36 })

*Рисунок 8 - Создание хранилища сессии*

Здесь collection - это название коллекции, а в поле uri передаётся уникальная ссылка на кластер MongoDB. Таким образом, мы получаем коллекцию sessions в базе данных, в которой теперь хранятся сессии пользователей, как показано на рисунке 9:

> \_id: "DW7oyR\_my2CfE6KK7te\_8PTCtDFGKE5e" exp ires: 2621- ©5 - 27T14:51:55.209+60:90

v session: Object > cookie: Object csrfSecret: "RuSo7fzDPaZr0t544qLFyZN3"

> flash:Object v user: Object v userData: Object v items: Array v ©: Object

v vkUsers: Array v0: Object firstname: "Никита” last\_name:"Березовский" profile\_id: 138301832 vl: Object

first\_name:"Таня" la st\_name: " Федорова " profileid: 277517137 \_id: Objectld("609aad5482e01b00153ab5ce") title: "8 марта”

\_id: Objectld("60995al430f88a0d2c24520©") name:"Irina"

password: "$2a$10$tAi4gsaGWblKp/wGoQHBeO4IGAsbYzWKr4vySwEitCWBVUdXuOTWy" \_\_v: 9

isAuthenticated: true

sid: null

*Рисунок 9 - Пример хранения сессии пользователя*

Каждая сессия обладает специальным уникальным ключом - токеном, с помощью которого сайт «знает», что это всё тот же пользователь, и не запрашивает авторизацию снова. Однако есть и другая сторона - токен является уязвимым местом для CSRF-атак, но об этом поговорим в главе 6 «Безопасность сайта». В поле expires хранится длительность текущей сессии и время, по прошествии которого сессия истекает, однако срок хранения текущей сессии составляет две недели, поэтому, чтобы не допускать утечек информации или же перегрузки базы данных, стоит удалять сессию после её завершения (рисунок 10):

1. router.get('/logout', async(req, res) =>{
2. req.session.destroy( ()=>{
3. res.redirect('/auth/login');
4. });
5. });

*Рисунок 10 - Удаление сессии*

Прежде, чем говорить об авторизации, необходимо разобрать три термина, которые часто путают: авторизация, аутентификация и идентификация. Итак, идентификация - это процесс запроса логина - таким образом можно определить, что за пользователь хочет воспользоваться услугами сайта. Далее происходит процесс аутентификации, который состоит в подтверждении пользователем своей личности путём ввода верного пароля. Наконец, происходит процесс авторизации - процесс принятия решения о том, какие действия разрешены данному пользователю. Когда пользователь вводит логин и пароль от своего аккаунта, на серверной части приложение происходит попадание в метод post (рисунок 11)

router.post('/login'э loginValidators, async (req, res) =>{ try {

const(name, password} = req.body;

const errors = validationResult(req);

if (I errors.isEmpty()){

req.flash('regError', errors.array()[0].msg);

return res.status(422).redirect('/auth/login')

}

const candidate = await User.findOne({name});

req.session.user = candidate;

req.session.isAuthenticated = true;

req.session.saveferror => {

if(error){

throw error;

}

res.redirect('/userLists');

})

} catch (error) { console.log(error);

}

});

*Рисунок 11 - Метод post для авторизации*

2 module.exports = function(req, res, next){  
if(!req.session.isAuthenticated){

1. return res.redirect('/auth/login');
2. }

next();

s }

*Рисунок 12 - Middleware для проверки авторизации*

Поскольку пользователь может не просто перейти на главную страницу сайта, не будучи авторизованным, но попытаться вручную написать путь до нужной ему страницы, например “<http://localhost:3000/userLists>”, и в том случае, если значение переменной isAuthenticated будет false, пользователя перенаправит на страницу авторизации. Конечно, существуют и более сложные пути обхода защищённых таким образом страниц сайта, поэтому необходимо усиливать безопасность сайта. Об этом мы поговорим в следующей главе.

6. Безопасность сайта

Безопасность web-сайта - это обеспечение сохранности данных, то есть невозможность их получения для посторонних лиц.

Обеспечение защиты сайта стоит начать с шифрования пароля. Для этого необходимо установить пакет “bcryptjs”. Пароль необходимо шифровать на моменте регистрации пользователя, которая на стороне сервера происходит в методе post(‘/register’).

1. х/ router.post('/register', registervalidators, async (req, res) => {
2. v try{
3. const{name, password} = req.body;
4. const errors = validationResult(req);
5. x/ if(!errors.isEmpty()){

req.flash('regError', errors.array()[0].msg);

61 return res.status(422).redirect('/auth/login’)

1. }

const hashpassword = await bcrypt.hash(password, 10)

1. xz const user = new User({
2. | name, password: hashpassword, lists: {items: []}
3. })j
4. await user.save();
5. res.redirect('/auth/login');
6. xz }catch(error){
7. console.log(error);
8. }
9. })

*Рисунок 13 - Регистрация пользователя и шифрование пароля*

На рисунка 13 в 56 строке происходит получение имени нового пользователя и пароля. В строке 63 создаётся переменная hashPassword, значением которой и будет зашифрованный пароль. В функцию bcrypt.hash() передаётся пароль, введённый пользователем, а также параметр salt, который может быть как строкой, так и числом, на основе которого произойдёт шифрование пароля. Чем длиннее будет введённая строка или, соответственно, чем больше будет введённое число, тем сложнее будет взломать пароль, однако не стоит забывать тот факт, что от размера строки или величины числа зависит и скорость шифрования, поэтому не стоит вписывать слишком большие значения. Оптимальная длина строки 10-12 символов. Как уже было показано ранее, результатом работы функции является зашифрованный пароль пользователя, который хранится в базе данных, и в случае её взлома, злоумышленникам будет непросто расшифровать пароль.

Следующим пунктом защиты является безопасность сессии. Как уже было сказано ранее, в сессию передаются необходимые пользовательские данные, что означает, что они хранятся не только внутри базы данных пользователя. Злоумышленники могут перехватывать сессии и таким образом получать доступ к данным пользователя, выполнять действия от его лица: отправлять сообщения, переводить деньги со счёта на счёт или менять пароль. Для защиты сессии необходимо проверять токен. В этом разработчику поможет пакет “csurf”, при добавлении которого, приложение получает вспомогательную функцию, которая проверяет, что полученный токен действителен. Напишем middleware (рисунок 14), где в строке 3 определим переменную csrf.

1 module.exports = function(req, res, next) { res.locals.isAuth = req.session.isAuthenticated; res.locals.csrf = req.csrfToken();

1. next();}

*Рисунок 14 - Middleware для обработки текущей сессии*

После чего созданную переменную необходимо прописать внутри каждой форме, отправляющей post-запрос на сервер (рисунок 15).

<form class = "inputGroup" id = "pasteVkLink" action = "/add” method = "post">

I <input id = "title" type = "text" class = "inputField" name = "title" placeholder = "Название списка" required value=”{{data.title}}">

<input id = "link" type = "text" class = "inputField" name = "link" placeholder = "Ссылка на пост" required value="{{data.link}}">

<input type = "hidden" name = "\_csrf" value = ”{{csrf}}">

<button class = "submitButton" id = "buttonVk">C<j>opMMpoBaTb</button>

</form>

*Рисунок 15 - Применение переменной csrf*

В строке 16, помимо основных данных с формы, мы также передаём токен данной сессии, однако делаем это в скрытом формате. Теперь пользователь может быть уверен в том, что его данные при переходе между страницами не будут перехвачены.

Ещё одним важным этапом защиты приложение является валидация форм, то есть проверка корректности введённых данных. Например, если приложение требует от пользователя ссылку, а получает иные данные, при дальнейшей передаче информации на сервер возникнет проблема, и приложение начнёт работать некорректно. Злоумышленники также могут воспользоваться этой уязвимостью, и совершить серию неверных вводов, что в дальнейшем может серьёзно повлиять на работу приложения в целом. Существуют два вида валидации: со стороны клиентской и серверной частей. Наиболее лёгкой в использовании является валидация первого типа, но в противовес этому факту также существуют довольно простые способы обхода проверок на стороне frontend. Поэтому гораздо безопаснее проводить валидацию одновременно на стороне клиентской и серверной частей.

Для проверки корректности ввода данных достаточно грамотно поработать с html-формами, расставив атрибут “required” для полей ввода и указав тип вводимых данных. Тогда, вводя текст в поле с числовым типом, вы получите ошибку ввода формы. Однако на серверной стороне не всё так просто. Для начала необходимо подключить пакет “connect-flash”, который поможет разделить ошибки на категории, а затем отправить их на frontend:

req.flash('regError', errors.array()[0].msg);

Здесь “regError” - это категория ошибок регистрации, а errors.array()[0].msg хранит в себе сообщения об ошибках, связанных с процессом регистрации. Например, “Пароли не совпадают” или “Имя не может быть короче двух символов”.

Также в важную роль в валидации приложения играет пакет “express-validator”, компоненты которого в дальнейшем помогут проверят данные, пришедшие на сервер, на корректность. Рассмотрим работу с этим пакетом на примере регистрации (рисунок 16).

exports.registerValidators = [

body('name').customfasync (value, {req}) => {

try {

const user = await User.findOne({name: value});

if(user){

return Promise.reject(’Пользователь уже существует');

} ' '

} catch (error) { console.log(error);

1 ~

})

.isLength({min: 2})

.withMessage('Имя не может быть менее 2 символов')

.trim(),

body('password',’Пароль не может быть менее 6 символов')

.isLength({min: 6, max: 56})

.isAlphanumeric()

.trim(),

body('confirm').customfasync (value,{req}) > {

if(value !== req.body.password){

throw new Error('Пароли не совпадают');

}

return true;

})

.trim(),

В

*Рисунок 16 - Валидация регистрации*

В строке 38 вызовы функции custom означает, что проверка будет написана программистом вручную. Как можно видеть, функция проверяет, существует ли пользователь с таким именем или нет. В том случае, если введённое имя не нашлось в коллекции пользователей, происходит следующая проверка - проверка длины имени, которое не может быть короче двух символов или быть пустой строкой. Аналогичные действия проводятся с паролем, но также добавляется проверка, что пароль состоит из латинских букв и цифр.

На рисунке 13 можно видеть, что данный массив с набором различных проверок передаётся в метод post для регистрации и затем с помощью метода req.flash('regError', errors.array()[0].msg) попадает в класс ошибок регистрации, сообщения о которых передаются на frontend.

На протяжении написания программы используются некоторые секретные значения, такие как ссылка для подключения к кластеру базы данных MongoDB, а также ключ шифрования сессии. Нельзя забывать о защите этих важных составляющих, поэтому в корне приложения создаём папку “keys”, в которой будет хранится три файла: index.js, keys.dev.js и keys.prod.js. Как ясно из названия, keys.dev.js используется во время разработки, поэтому внутри этого файла можно задать значения переменных MONGO\_URI и SESSION\_SECRET, а файл keys.prod.js содержит в себе уже скрытые от посторонних глаз переменные (рисунок 16а):

module.exports = {

MONGODBURI: process.env.MONGODB JJ RI, SESSIONSECRET: process.env.SESSIONSECRET

*Рисунок 16а - Сокрытие данных*

В файле index.js происходит проверка. Если на данный момент сайт находится в продакшене, переменным присваиваются скрытые значения. Если же разработчик запустил сайт для отладки на локальном хосте, используются прямые значения переменных (рисунок 16б):

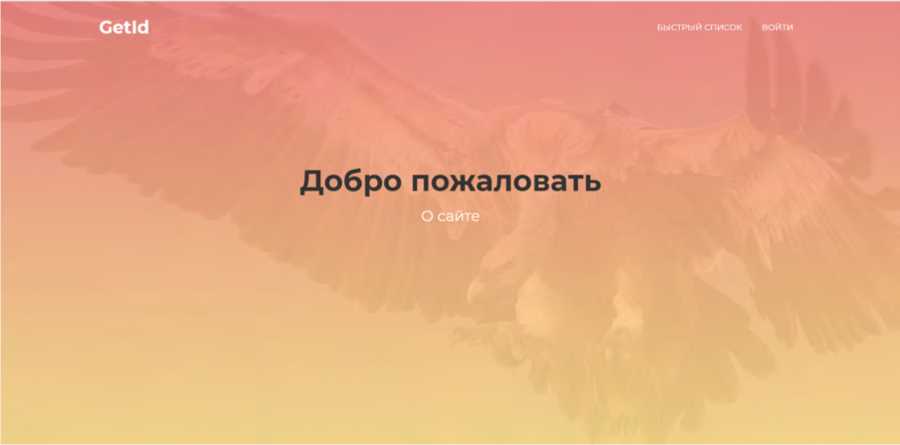
1 if(process.env.NODEENV === 'production'){ module.exports = require(’./keys.prod');

1. } else{
2. module.exports = require(’./keys.dev');
3. 1

*Рисунок 16б - Распределение файлов*

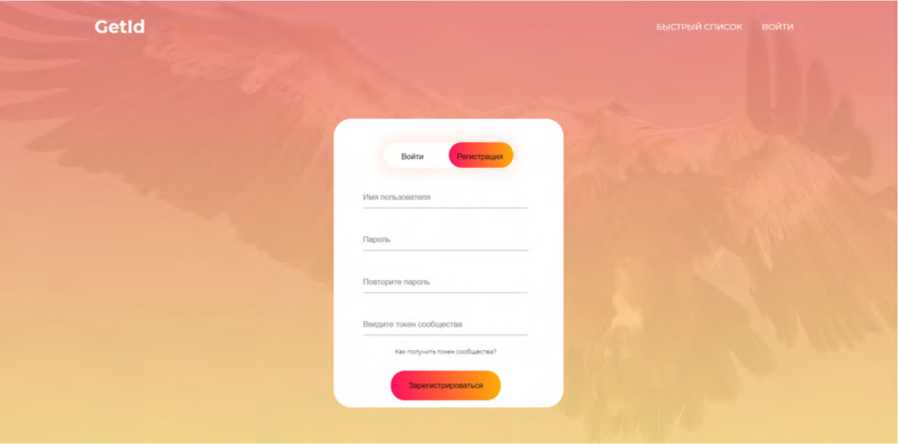
7. Интерфейс сайта

Рассмотрим интерфейс сайта GetId. На главной странице сайта (рисунок 17) есть всего три раздела: “О сайте”, “Быстрый список” и “Войти”. В том случае, если пользователь не хочет регистрироваться в приложении, но хочет собрать список пользователей, он может воспользоваться “Быстрым списком” и скачать файл с именами пользователей, оставивших комментарии под постом сообщества.



*Рисунок 17 - Главная страница сайта*

На странице регистрации (рисунок 18) пользователю обязательно нужно указать имя и пароль. Одним из плюсов сайта GetId является отсутствие запроса email или номера телефона, что означает, что пользователь не будет получать spam-рассылку от самого сайта, и в случае взлома баз данных сайта злоумышленники также не смогут воспользоваться личными данными пользователя или продать их на сторонние сайты. Однако у такого относительно безопасностно способа регистрации есть свой минус - имя каждого пользователя должно быть уникальным, поэтому в случае, если на сайте уже зарегистрирован пользователь с таким именем, вам придётся придумать что-то другое.



*Рисунок 18 - Регистрация или вход на сайт*

Также хочется отметить, что на сайте существует функция отправки сообщения, но воспользоваться ею можно только в том случае, если пользователь указал токен сообщества при регистрации. Ссылка на подробную инструкцию получения токена сообщества находится внутри формы регистрации (рисунок 19).



**Для того, чтобы получить токен сообщества, необходимо выполнить следующие шаги**

Открыть раздел «Управление сообществом» («Управление страницей», если у Вас публичная страница)

Выбрать вкладку «Работа с API» Нажать «Создать ключ доступа»

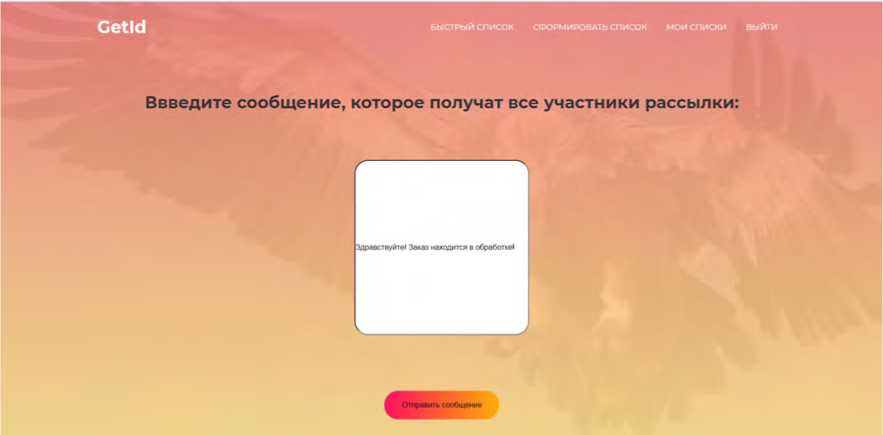
*Рисунок 19 - Инструкция к получению токена сообщества*

После регистрации пользователь может сформировать список, указав его название и ссылку на пост (рисунок 20).



*Рисунок 20 - Список желающих заказать цветы к 8 марта*

Поскольку при регистрации пользователь указал токен сообщества, кнопка “написать сообщение” доступна, и при нажатии на неё можно выполнить общую рассылку сообщения (рисунок 21, рисунок 22)

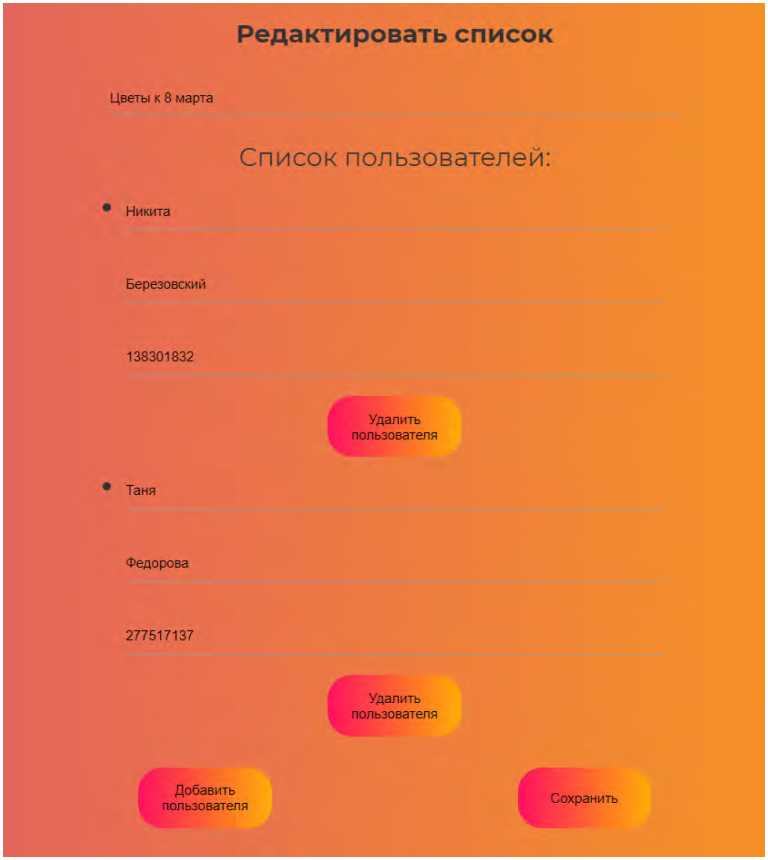


*Рисунок 21 - Отправка сообщения участникам списка*

**Свечи ручной работы 19:17**

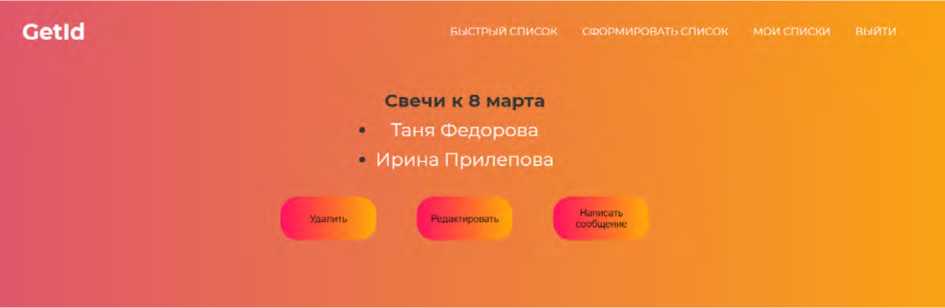
**Здравствуйте! Заказ находится в обработке!**

*Рисунок 22 - Сообщение успешно получено*

Также пользователь может редактировать список: добавлять и удалять участников списка, менять название списка (рисунок 23):

*Рисунок 23 - Редактирование списка пользователей*

После редактирования список успешно был изменён (рисунок 24).



*Рисунок 24 - Обновлённый список*

Наконец, пользователь может удалить сам список (рисунок 24a).



*Рисунок 24a - Список удалён*

Как можно видеть по меню сверху, вкладка “Быстрый список” также доступна и для зарегистрированных пользователей. На рисунке 25 можно увидеть, что будет, если пользователь попытается перейти с помощью адресной строки на несуществующую страницу сайта:



*Рисунок 25 - Обработка 404 ошибки*

8. Анализ программного кода

Рассмотрим операции со списками покупателей, которые формирует пользователь сайта. При переходе на страницу “Мои списки” вызывается метод get, с помощью которого загружается и отображается всё содержимое страницы (рисунок 26):

1. router-get(*'Г ,* auth, async (req, res) => {

const user = await req.user.populate('userData.items');

1. const lists = user.userData.items.map(l => ({
2. ...l.\_doc
3. }));
4. res.render('userLists’, {
5. title: 'My Lists',
6. lists

и });

15

1. });

*Рисунок 26 - Метод GET*

Вывод списков пользователей на экран условно можно разбить на два этапа. Сначала в переменную user считываются все списки, которые лежат в модели пользователя, но кроме списков, модель пользователя также хранит в себе логин и пароль, и чтобы не отдавать эти данные на на frontend, существует метод populate, забирающий необходимые поля модели. Однако не всё так просто, ведь сам массив items хранит в себе много вспомогательной информации, которую получает при попадании в MongoDB. Здесь начинается переход ко второму этапу - очистка массива от лишней информации путём вызова функции \_doc для каждого элемента массива. После чего готовые списки передаются на frontend. В Приложение В представлен html-код отображения списков на странице. Здесь стоит сказать о том, что Handlebars предлагает удобный функционал для отображения информации. Например, условие {{#if}} {{else}} {{/if}} проверяет выполнение необходимых условий прямо во время загрузки необходимой информации на клиентскую часть. То есть если каким-то образом через все проверки серверной части смогла пройти ошибка, условие “if” станет ещё одной преградой для отображения некорректной информации. Также можно воспользоваться циклом {{#each}} {{/each}}для перебора и вывода каждого элемент списка на экран.

Следующий пункт - редактирование информации. Каждый список имеет уникальный id, который получает при попадании в базу данных. Чтобы приложение знало, какой именно список нужно отредактировать, необходимо передать id на страницу редактирования конкретного списка. При нажатии на кнопку <a href = "/userLists/{{\_id}}/edit?allow=true"><button class = "submitButton" id = "buttonVk">Редактировать</button></a> отправляется post запрос на сервер (рисунок 27)

router.get('/:id/edit’, auth, async (req, res) => {  
if(freq.query.allow){

return res.redirect(’/');

}

let list = await req.user.findList!d(req.params.id);

list = list.doc;

res.render('list-edit', {

title: 'Редактировать ${list.title}', list,

});

});

*Рисунок 27 - POST запрос редактирования списка*

В имеющихся у пользователя списках находится нужный и отправляется обратно на frontend конечному пользователю. Когда пользователь заканчивает редактирование списка и нажимает кнопку “Сохранить”, снова вызывается метод post, внутри которого происходит сохранение изменений (рисунок 28)

31 router.post('/edit', auth, async (req, res) => {

32 await req.user.update(req.body);

res.redirect('/userLists');

34 })

*Рисунок 28 - Сохранение внесённых изменений* Код функции update в Приложении Г. В том случае, если пользователь удалил все элементы списка, вызывается функция удаления, код которой представлен в Приложении Д. Если же список не пуст, необходимо заново сформировать ячейку массива items, который содержит в себе все списки пользователя. Для чего нужно формировать новый объект, если можно просто присвоить старому объекту значения нового объекта, пришедшего из формы? Дело в том, что из формы приходит не объект, а всего лишь набор массивов, содержащий в себе имена, фамилии и id страниц всех участников списка. Из этих массивов нужно заново сформировать ячейку массива items, после чего заменить изменяемую ячейку на новую. Функция редактирования использует функцию подсчёта количества слов, которая также представлена в приложении Г. Когда пользователь удаляет участника списка, форма возвращает пустую ячейку в массиве, которая учитывается при итоговом составлении объекта. Поэтому функция подсчёта слов нужна для того, чтобы убрать в полученном массиве все пустоты и затем вернуть точное количество имен, содержащихся в списке. В конце вызывается метод save(), который сохраняет изменения в базе данных.

**ЗАКЛЮЧЕНИЕ**

Таким образом, поставленная задача по созданию удобного сайта, который поможет продавцам собирать имена и id страниц комментаторов из-под поста в социальной сети Вконтакте и формировать списки заказов,была успешно выполнена.

На данный момент один аккаунт может делать рассылку не более чем от лица одного магазина, поскольку токен сообщества запрашивается единожды - при регистрации. В дальнейшем планируется расширение функционала таким образом, чтобы токены разных сообществ могли закрепляться за одним аккаунтом.

Также планируется добавить рандомайзер для проведения розыгрышей внутри сайта. При этом отправлять сообщение победителю можно будет по тому же принципу, по которому осуществляется рассылка сообщений группе людей. Конечно, на просторах интернета существует множество сайтов со встроенным рандомайзером, однако стоит отметить тот факт, что большинство из них являются платными, и абсолютно все требуют email при регистрации или иные способы связи с пользователем, что в дальнейшем используется для рассылки спама и рекламы, а в случае взлома сайта может оказаться в руках злоумышленников. В противовес всем этим минусам, сайт GetId является бесплатным и, как уже говорилось ранее, не требует при регистрации ни номера телефона пользователя, ни email.

Наконец, в будущем сайт GetId будет работать не только с социальной сетью Вконтакте, но и, например, YouTube.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ**

1. Документация по Vk API [Электронный ресурс]: Знакомство с API Вконтакте. URL: <https://vk.com/dev/first_guide>. (дата обращения 25.10.2020).
2. Учебник языка JavaScript [Электронный ресурс]: Современный учебник JavaScript. URL: <https://learn.javascript.ru/>. (дата обращения 17.09.2020).
3. Документация по фреймворку Express [Электронный ресурс]: Express/Node introduction.

URL:https://developer.mozilla.org/ru/docs/Learn/Server-side/Express\_Nodej s/Introduction. (дата обращения 28.11.2020).

1. Хабр [Электронный ресурс]: Токен авторизации на примере JSON WEB Token. URL: <https://learn.javascript.ru/>. (дата обращения 24.02.2021).
2. Хабр [Электронный ресурс]: Когда стоит и не стоит использовать MongoDB. URL: <https://habr.com/ru/company/latera/blog/280196/>. (дата обращения 12.01.2021).

|  |  |
| --- | --- |
| 6. Learn HTML [электронный ресурс]: обучающий <https://www.codecademy.com/learn/learn-html>. (дата | курс. URL: обращения |

10.10.2020).

|  |  |
| --- | --- |
| 7. Learn CSS [электронный ресурс]: обучающий <https://www.codecademy.com/learn/learn-css>. (дата | курс. URL: обращения |

20.10.2020).

**ПРИЛОЖЕНИЕ А**

**Пример HTML разметки**

<div class = intro>

<М>Добро пожаловать на GetId!</h1>

<113>11риятного пов1ьзования!</113>

</div>

**ПРИЛОЖЕНИЕ Б**

**Пример CSS стилей**

.intro

width: 100%;

top: 35%;

position: absolute;

text-align: center;

**ПРИЛОЖЕНИЕ В**

**Анализ программного кода. Пример файла userList.hbs**

<div class = "dataSet" id="usersLists">

{{#if lists.length}}

{{#each lists}}

<div class = "bodyForList">

<div class = "dataList">

<h4>{{title}}</h4>

<ul>

{{#each vkUsers}}

<li>

<a href = "[http://vk.com/id{{profile\_id}}](http://vk.com/id%7b%7bprofile_id%7d%7d)"> {{first\_name}}

{{last\_name}}</a>

</li>

{{/each}}

</ul>

<div class = "listsButtons">

<form action = "/userLists/remove" method = "POST">

<input type = "hidden">

<input type = "hidden" name = "\_csrf" value = "{{@root.csrf}}">

<button type = "submit" class = "submitButton remove" name = "id" value="{{\_id}}">Удалить</button>

</form>

<a href = "/userLists/{{\_id}}/edit?allow=true"><button class =

"submitButton" id = "buttonVk">Редактировать</button></a>

{{#if @root.isToken}}

<a href = "/writeMessage/{{\_id}}/edit?allow=true"><button class = "submitButton" id = "buttonVk">Написать сообщение</button></a>

{{/if}}

</div>

</div>

</div>

{{/each}}

{{else}}

<РГС>Списков нет</pre>

{{/if}}

</div>

**ПРИЛОЖЕНИЕ Г**

**Анализ программного кода. Функция редактирования списка**

function wordsCount(names){

str = names.toString();

str = str.replace(/[\s.,%]/g, ' ');

str = str.replace(/(A\s\*)|(\s\*$)/gi,"");

str = str.replace(/[ ]{2,}/gi," ");

str = str.replace(/\n /,"\n");

return str.split(' ').length;

}

userSchema.methods.update = function(list){

if(list.first\_name == null){

this.removeFromLists(list.id);

return;

}

let clonedItems = [...this.userData.items];

let objUser;

let vkUsers = new Array();

let index = 0;

let nameCounter = wordsCount(list.first\_name);

if(nameCounter == 1){

objUser = {

first\_name: list.first\_name,

last\_name: list.last\_name,

profile\_id: list.profile\_id,

}

vkUsers[index] = objUser;

}else{

for(let user = 0; user < nameCounter; user++){

objUser = {

first\_name: list.first\_name[user], last\_name: list.last\_name[user], profile\_id: list.profile\_id[user],

}

vkUsers[index] = objUser;

index+=1;

for(let item in clonedItems){

if (clonedItems[item].\_id.toString() === list.id.toString()){ clonedItems[item].title = list.title;

clonedItems[item].vkUsers = vkUsers;

const newList = {items: clonedItems}; this.userData = newList;

return this.save();

**ПРИЛОЖЕНИЕ Д**

**Анализ программного кода. Функция удаления списка**

userSchema.methods.removeFromLists = function(id){

let clonedItems = [...this.userData.items];

let iterator = 0;

let arr = new Array();

for(item in clonedItems){

if (clonedItems[item].\_id.toString() != id.toString()){ arr[iterator] = clonedItems[item];

iterator +=1;

clonedItems = arr;

const newList = {items: clonedItems};

this.userData = newList;

return this.save();

}