GeekBrains

**Разработка CRM-системы для управления клиентскими отношениями на основе архитектуры Single Page Application, реализованной с использованием фреймворка Laravel и библиотеки Vue.js"**

Специальность:

Программист. Специализация - Веб-разработка на PHP (Laravel).

Романов Александр Николаевич.

Ярославль

2024 г.

ВВЕДЕНИЕ…………………………………………………………..….4

ГЛАВА 1. Ознакомление с ключевыми концепциями и технологиями, необходимыми для создания веб-приложений.

1.1. Архитектуры веб-приложений: обзор и сравнение………………..8

1.2. Основы Backend-разработки: роль и возможности Laravel в современных веб-приложениях………………………………….………12

1.3. Обзор Vue.js 3: преимущества для разработки SPA………………..13

ГЛАВА 2. Основы работы с Laravel и Vue.js для создания CRM-системы

2.1. Laravel как фреймворк для разработки бэкенда.................................15

2.2. Маршрутизация и управление состоянием в Vue.js..........................17

2.3. Создание RESTful API с Laravel для взаимодействия с фронтендом………………………………………………………………...18

2.4. Интеграция Vue.js с Laravel…………………………………………..20

ГЛАВА 3. Разработка компонентов CRM-системы

3.1. Проектирование пользовательского интерфейса CRM-системы.......21

3.2. Управление клиентскими данными: модели, миграции и API в Laravel.............................................................................................................24

3.3. Реализация функционала по работе с клиентами в Vue.js..................27

3.4. Внедрение системы управления состоянием с Vuex...........................31

ГЛАВА 4. Тестирование и деплой CRM-системы

4.1. Методы тестирования веб-приложений: от юнит-тестов до E2E.......35

4.2. Настройка и использование PHPUnit для тестирования бэкенда.......39

4.3. Тестирование фронтенда с использованием Vue Test Utils.................41

4.4. Деплой и настройка окружения для CRM-системы.............................44

ГЛАВА 5. Заключение и перспективы развития проекта

5.1. Обзор достигнутых результатов и основных трудностей проекта.....46

5.2. Возможности дальнейшего развития и масштабирования CRM-системы………………………………………………………………..48

Приложения

А. Исходный код основных компонентов системы......................................50

Б. Руководство пользователя по работе с CRM-системой...........................52

Список использованных источников.............................................................54

ВВЕДЕНИЕ

В современном мире веб-приложения играют ключевую роль в повседневной жизни людей, предоставляя доступ к широкому спектру услуг и информации через Интернет. Начиная с простых веб-сайтов, созданных на HTML и CSS, которые служили источниками информации, подобно бумажным газетам или книгам в библиотеке, веб-технологии претерпели значительное развитие. С появлением JavaScript веб-сайты стали более интерактивными, предоставляя пользователю возможности для более сложного взаимодействия. Развитие не остановилось на достигнутом, и впоследствии были созданы различные фреймворки и библиотеки, которые значительно упростили и ускорили процесс разработки сложных веб-приложений. Сегодня, имея доступ к Интернету, люди могут не только находить необходимую информацию, но и пользоваться услугами банков, записываться на прием к врачу, заказывать еду на дом и совершать покупки в интернет-магазинах.

В этом контексте, разработка удобных и безопасных пользовательских интерфейсов, задача, которой занимаются Frontend-разработчики, приобретает особую важность. Эта работа имеет ключевое значение для создания Single Page Application (SPA), которые обеспечивают высокую скорость работы и удобство использования.

Тема проекта: Разработка системы управления клиентскими отношениями (CRM-системы) с использованием архитектуры Single Page Application, реализованной на фреймворке Laravel для бэкенда и Vue.js для фронтенда.

Цель: Создать функциональную и удобную в использовании CRM-систему для управления клиентскими отношениями, используя современные технологии разработки веб-приложений.

Дипломный проект будет разрабатываться с акцентом на Backend-разработку, при этом также уделяя внимание Frontend-части. Опыт разработки включает в себя работу с Laravel и Vue.js, а также с другими современными инструментами и технологиями веб-разработки.

Суть проекта заключается в создании системы, которая позволит эффективно управлять клиентской базой, отслеживать взаимодействия с клиентами, анализировать данные для повышения качества обслуживания и увеличения продаж.

Задачи проекта:

- Изучить возможности и особенности фреймворка Laravel и библиотеки Vue.js.

- Разработать архитектуру и дизайн CRM-системы.

- Реализовать функционал для работы с клиентской базой: добавление, редактирование, удаление записей.

- Создать интерфейс для визуализации статистики и аналитики по взаимодействиям с клиентами.

- Обеспечить безопасность данных и доступа к системе.

- Протестировать систему на наличие ошибок и уязвимостей.

Инструменты: VSCode, Laravel, Vue.js, Vuex для управления состоянием приложения, Vue Router для маршрутизации, SCSS для стилизации, Axios для работы с HTTP-запросами, и другие современные инструменты разработки.

В проекте я выступаю в роли:

- Архитектора приложения;

- Backend и Frontend-разработчика;

- Дизайнера интерфейса;

- Тестировщика.

Для успешного запуска CRM-системы, разработанной на стеке Laravel и Vue.js, важно было не только написать код и протестировать его, но и правильно развернуть приложение на сервере. Этот процесс включал в себя настройку окружения, деплой приложения и его дальнейшее обслуживание. Вот как я подошёл к этим задачам:

Подготовка сервера

Первым шагом была подготовка сервера. Для CRM-системы выбрал VPS на Ubuntu, так как это популярный и хорошо поддерживаемый вариант. Установил на сервер необходимое программное обеспечение: Nginx как веб-сервер, PHP для выполнения кода Laravel, Composer для управления зависимостями PHP и Node.js с npm для сборки фронтенда.

Настройка окружения

Для Laravel важно правильно настроить файл .env, который содержит параметры окружения, такие как данные для подключения к базе данных, ключи приложения и другие важные настройки. После настройки .env файла, выполнил миграции базы данных и заполнил её начальными данными с помощью сидов:

php artisan migrate --seed

Сборка фронтенда

Следующим шагом была сборка фронтенда. В корне проекта на сервере запустил команду npm для установки зависимостей и сборки проекта:

npm install

npm run build

Это создало оптимизированные файлы фронтенда, которые были готовы к раздаче через веб-сервер.

Настройка веб-сервера

Настроил Nginx для обслуживания статических файлов, сгенерированных Vue.js, и перенаправления запросов к API на PHP-обработчик. В конфигурационном файле Nginx для сайта указал корневую директорию для статических файлов и настройки проксирования для API-запросов.

SSL и безопасность

Для обеспечения безопасности подключения к CRM-системе установил SSL-сертификат с помощью Let's Encrypt. Также проверил конфигурацию сервера и приложения на предмет потенциальных уязвимостей и настроил фаервол для ограничения доступа к необходимым портам.

Мониторинг и резервное копирование

Для обеспечения стабильной работы системы настроил мониторинг сервера с помощью таких инструментов, как Prometheus и Grafana. Также настроил регулярное резервное копирование базы данных и файлов проекта для предотвращения потери данных.

Деплой и настройка окружения для CRM-системы требовали внимательного подхода к деталям и планирования. Правильная настройка сервера, безопасность соединений и регулярное резервное копирование данных обеспечили надёжную и стабильную работу системы после запуска.

**ГЛАВА 1**

**Ознакомление с ключевыми концепциями и технологиями, необходимыми для создания веб-приложений**

**1.1. Архитектуры веб-приложений: обзор и сравнение**

В современной веб-разработке существует несколько подходов к архитектуре веб-приложений, каждый из которых имеет свои преимущества и недостатки. Выбор архитектуры зависит от множества факторов, включая требования к производительности, масштабируемости, безопасности и удобству разработки. В этом разделе мы рассмотрим три основные архитектуры веб-приложений: многостраничные приложения (MPA), одностраничные приложения (SPA) и микросервисы.

Многостраничные приложения (MPA)

Многостраничные приложения представляют собой традиционный подход к веб-разработке, где каждая страница приложения загружается с сервера как отдельный HTML-документ. Этот подход обеспечивает простоту разработки и поддержки, а также хорошую индексацию поисковыми системами. Однако, MPA могут страдать от медленных времен загрузки страниц и сложности поддержки динамического взаимодействия без полной перезагрузки страницы.

Одностраничные приложения (SPA)

Одностраничные приложения загружают всю необходимую HTML, CSS и JavaScript код один раз, при первом посещении страницы пользователем. Все последующие взаимодействия с приложением происходят без перезагрузки страницы, что обеспечивает более плавный и быстрый пользовательский опыт. SPA идеально подходят для создания интерактивных приложений с богатым пользовательским интерфейсом. Однако, они могут представлять сложности для SEO и требуют более сложной архитектуры на стороне клиента.

Микросервисы

Архитектура микросервисов предполагает разделение приложения на набор маленьких, независимых сервисов, каждый из которых выполняет определенную функцию и общается с другими через легковесные API. Этот подход обеспечивает высокую масштабируемость и гибкость в разработке и поддержке приложений. Микросервисы идеально подходят для крупных, сложных систем, но могут быть избыточны для малых проектов из-за сложности управления и мониторинга множества сервисов.

Сравнение архитектур веб-приложений

В выборе архитектуры веб-приложения важно учитывать ряд факторов, включая требования к проекту, ожидаемую нагрузку и предпочтения команды разработчиков. Ниже представлено сравнение трех основных архитектурных подходов: многостраничных приложений (MPA), одностраничных приложений (SPA) и микросервисов.

- Многостраничные приложения (MPA)

- Преимущества: Простота разработки и поддержки, хорошая индексация поисковыми системами, привычная навигация для пользователя.

- Недостатки: Медленная загрузка страниц из-за необходимости загружать новую страницу при каждом запросе, сложность внедрения динамического контента без полной перезагрузки страницы.

- Одностраничные приложения (SPA)

- Преимущества: Быстрый и плавный пользовательский опыт за счет динамической подгрузки контента, упрощение разработки сложных интерактивных интерфейсов.

- Недостатки: Потенциальные сложности с поисковой оптимизацией (SEO), необходимость разработки сложной клиентской архитектуры, начальная загрузка может занимать больше времени.

- Микросервисы

- Преимущества: Высокая масштабируемость и гибкость, упрощение разработки и поддержки за счет разделения приложения на маленькие независимые сервисы, возможность использования разных технологий для разных сервисов.

- Недостатки: Сложность управления и мониторинга большого количества сервисов, потенциальные проблемы с производительностью из-за сетевых вызовов между сервисами, сложность тестирования взаимодействия между сервисами.

Каждый из этих подходов имеет свои сильные и слабые стороны, и выбор оптимальной архитектуры зависит от специфики проекта, требований к функциональности и пользовательскому опыту, а также от ресурсов и опыта команды разработчиков.

**1.2. Основы Backend-разработки: роль и возможности Laravel в современных веб-приложениях**

В современной веб-разработке бэкенд играет ключевую роль, обеспечивая логику работы приложений, управление данными, аутентификацию пользователей и многое другое. Laravel, как один из ведущих PHP-фреймворков, предлагает разработчикам мощный набор инструментов для создания надежных и масштабируемых веб-приложений.

Преимущества использования Laravel

- MVC-архитектура: Laravel строится на принципах MVC (Model-View-Controller), что обеспечивает четкое разделение логики приложения и пользовательского интерфейса. Это упрощает разработку и поддержку кода.

- Синтаксис: Laravel известен своим "элегантным" синтаксисом, который делает код легким для чтения и написания. Это повышает производительность разработки и облегчает введение новых разработчиков в проект.

- Богатая экосистема: Laravel предлагает широкий спектр пакетов и инструментов, таких как Eloquent ORM для работы с базами данных, Blade шаблонизатор, встроенные средства для аутентификации и авторизации, а также интеграцию с почтовыми сервисами и очередями задач.

- Безопасность: Laravel обеспечивает высокий уровень безопасности приложений благодаря встроенным механизмам защиты от таких угроз, как SQL-инъекции, XSS (Cross-Site Scripting) и CSRF (Cross-Site Request Forgery).

- Artisan CLI: Командная строка Artisan предлагает разработчикам множество полезных команд для ускорения разработки, включая создание моделей, контроллеров, миграций и запуск тестов.

Роль Laravel в разработке веб-приложений

Laravel идеально подходит для создания как простых, так и сложных веб-приложений благодаря своей гибкости и масштабируемости. Фреймворк может использоваться для разработки корпоративных порталов, интернет-магазинов, CRM-систем, API для мобильных приложений и многого другого. В сочетании с современными фронтенд-технологиями, такими как Vue.js, Laravel позволяет создавать полнофункциональные SPA, предлагая пользователю высококачественный и быстрый интерфейс.

Заключение

Laravel представляет собой мощный инструмент для разработки бэкенда веб-приложений, предлагая разработчикам удобные абстракции, элегантный синтаксис и широкие возможности для реализации самых разнообразных проектов. Благодаря своей гибкости и масштабируемости, Laravel заслуженно занимает одно из ведущих мест среди фреймворков для разработки современных веб-приложений.

**1.3. Обзор Vue.js 3: преимущества для разработки SPA**

Vue.js 3 представляет собой последнюю версию популярного прогрессивного JavaScript-фреймворка, предназначенного для создания пользовательских интерфейсов и одностраничных приложений (SPA). Этот фреймворк сочетает в себе удобство и гибкость, делая разработку более интуитивно понятной и эффективной. В этом разделе мы рассмотрим ключевые особенности и преимущества Vue.js 3 для разработки SPA.

Улучшенная производительность

Vue.js 3 был переписан с нуля с использованием TypeScript, что привело к значительному улучшению производительности. Новая архитектура фреймворка обеспечивает более быстрый рендеринг компонентов и уменьшенное потребление памяти, что критически важно для SPA, где пользовательский опыт напрямую зависит от скорости работы приложения.

Композиционный API

Одним из самых значительных нововведений в Vue.js 3 является введение композиционного API, который предоставляет более гибкий и мощный способ организации логики компонентов. Это позволяет разработчикам лучше управлять повторно используемым кодом и создавать более чистую и модульную структуру приложения.

Улучшенная поддержка TypeScript

Vue.js 3 предлагает улучшенную интеграцию с TypeScript, что делает разработку более безопасной и удобной для тех, кто предпочитает использовать строгую типизацию. Это также облегчает интеграцию с современными инструментами разработки и улучшает поддержку больших и сложных проектов.

Легковесность и модульность

Vue.js 3 стал еще более легковесным и модульным, благодаря чему разработчики могут выбирать только те функции фреймворка, которые необходимы для их проекта. Это позволяет создавать еще более производительные SPA, загружая только необходимый код.

Улучшенная система реактивности

Система реактивности в Vue.js 3 была полностью переработана, предоставляя более эффективный и понятный способ отслеживания изменений в данных приложения. Это упрощает разработку динамических интерфейсов и повышает общую надежность SPA.

Vue.js 3 предлагает множество преимуществ для разработки одностраничных приложений, включая улучшенную производительность, гибкость и безопасность. Благодаря своей легковесности, модульности и мощному набору функций, Vue.js 3 является отличным выбором для создания высококачественных SPA, которые обеспечивают отличный пользовательский опыт.

ГЛАВА 2.

**Основы работы с Laravel и Vue.js для создания CRM-системы**

**2.1. Laravel как фреймворк для разработки бэкенда**

Laravel является одним из наиболее популярных PHP-фреймворков, который обеспечивает эффективную и удобную платформу для разработки бэкенда веб-приложений. Он предлагает широкий спектр функций, которые значительно ускоряют процесс разработки и делают его более удобным. В этом разделе мы рассмотрим основные аспекты Laravel, которые делают его идеальным выбором для создания бэкенда CRM-системы.

Выразительный синтаксис

Laravel известен своим синтаксисом, который делает код легким для чтения и написания. Это не только улучшает производительность разработки, но и облегчает поддержку и масштабирование приложения в будущем.

MVC-архитектура

Фреймворк строится на принципах MVC (Model-View-Controller), что обеспечивает четкое разделение логики приложения и пользовательского интерфейса. Это упрощает разработку и поддержку кода, а также делает его более структурированным и модульным.

Богатая экосистема и интеграция

Laravel предлагает широкий спектр пакетов и инструментов, таких как Eloquent ORM для работы с базами данных, Blade шаблонизатор, встроенные средства для аутентификации и авторизации, а также интеграцию с почтовыми сервисами и очередями задач.

Миграции и сидирование баз данных

Система миграций Laravel позволяет легко и безопасно вносить изменения в структуру базы данных, а сидирование помогает заполнять базу данных начальными данными для разработки и тестирования.

Artisan CLI

Командная строка Artisan предлагает разработчикам множество полезных команд для ускорения разработки, включая создание моделей, контроллеров, миграций и запуск тестов.

Безопасность

Laravel обеспечивает высокий уровень безопасности приложений благодаря встроенным механизмам защиты от таких угроз, как SQL-инъекции, XSS (Cross-Site Scripting) и CSRF (Cross-Site Request Forgery).

Заключение

Laravel представляет собой мощный инструмент для разработки бэкенда веб-приложений, предлагая разработчикам удобные абстракции, элегантный синтаксис и широкие возможности для реализации самых разнообразных проектов. Благодаря своей гибкости и масштабируемости, Laravel заслуженно занимает одно из ведущих мест среди фреймворков для разработки современных веб-приложений, включая CRM-системы.

**2.2. Маршрутизация и управление состоянием в Vue.js**

В разработке одностраничных приложений (SPA) на Vue.js ключевую роль играют механизмы маршрутизации и управления состоянием. Эти функции обеспечивают плавное и интуитивно понятное взаимодействие пользователя с приложением, что критически важно для современных веб-приложений. Vue Router, официальная библиотека маршрутизации для Vue.js, позволяет реализовать навигацию между страницами без необходимости перезагрузки браузера. Это достигается за счет изменения URL в адресной строке браузера без запроса новой страницы от сервера. Vue Router предоставляет гибкие возможности для настройки маршрутов, включая вложенные пути, именованные представления и динамические сегменты пути, что делает его мощным инструментом для создания структурированных и легко масштабируемых приложений.

Для централизованного управления состоянием в приложениях Vue.js используется Vuex — официальное хранилище состояния, которое служит единым источником истины для всех компонентов. Vuex значительно упрощает обмен данными между компонентами и их синхронизацию, особенно в крупных приложениях, где важно поддерживать согласованность данных. Система мутаций в Vuex обеспечивает предсказуемость изменений состояния, что упрощает отладку и тестирование приложения. Кроме того, поддержка модульности в Vuex позволяет структурировать хранилище на модули, облегчая тем самым управление состоянием в масштабируемых проектах.

Использование Vue Router вместе с Vuex в разработке SPA на Vue.js предоставляет разработчикам мощные инструменты для создания высокопроизводительных и масштабируемых приложений. Эти библиотеки не только улучшают пользовательский опыт за счет плавной и быстрой навигации, но и обеспечивают эффективное управление состоянием приложения, делая разработку более организованной и предсказуемой.

**2.3. Создание RESTful API с Laravel для взаимодействия с фронтендом.**

В современной веб-разработке создание RESTful API является ключевым аспектом для обеспечения взаимодействия между бэкендом и фронтендом приложений. Laravel предоставляет мощный и гибкий инструментарий для разработки API, что делает его идеальным выбором для создания надежных и масштабируемых веб-сервисов. В этом разделе мы рассмотрим, как Laravel может быть использован для создания RESTful API, который будет служить мостом между серверной логикой и пользовательским интерфейсом, разработанным на Vue.js.

Laravel облегчает процесс создания RESTful API благодаря своей встроенной поддержке маршрутизации, мощному ORM (Object-Relational Mapping) Eloquent, системе аутентификации и простоте валидации запросов. Маршрутизация в Laravel позволяет легко определить пути API и связать их с соответствующими контроллерами, которые обрабатывают бизнес-логику приложения. Eloquent ORM упрощает работу с базами данных, предоставляя интуитивно понятный и гибкий способ для создания, чтения, обновления и удаления данных (CRUD операции), что является основой любого RESTful API.

Система аутентификации Laravel, включая Laravel Passport, предлагает удобные средства для реализации аутентификации и авторизации в API, обеспечивая безопасный доступ к ресурсам приложения. Валидация запросов в Laravel позволяет легко проверять входящие данные на соответствие заданным требованиям, что повышает надежность и безопасность API.

Для облегчения разработки и тестирования API Laravel предлагает инструменты, такие как фабрики и сиды для генерации тестовых данных, а также встроенные функции для тестирования HTTP-запросов. Это позволяет разработчикам эффективно проверять работу API в различных условиях и гарантировать его корректную работу перед развертыванием.

Создание RESTful API с использованием Laravel для взаимодействия с фронтендом, разработанным на Vue.js, обеспечивает мощную и гибкую архитектуру веб-приложения. Это позволяет разработчикам создавать сложные и высокопроизводительные веб-приложения, которые могут легко масштабироваться и адаптироваться к изменяющимся требованиям бизнеса и пользователей.

**2.4. Интеграция Vue.js с Laravel**

Интеграция Vue.js с Laravel представляет собой мощное сочетание, которое позволяет разработчикам создавать полнофункциональные веб-приложения с богатым пользовательским интерфейсом и надежным бэкендом. Laravel служит прочной основой для серверной части приложения, в то время как Vue.js предоставляет реактивные компоненты для фронтенда. Эта интеграция позволяет разработчикам использовать современные инструменты и практики для создания интерактивных SPA, которые одновременно легко поддерживать и масштабировать.

Laravel и Vue.js оба поддерживают модульный подход к разработке, что позволяет легко разделять логику приложения на независимые части. Laravel предлагает широкий набор инструментов для работы с базами данных, кэширования, файловой системы, сессий и очередей, в то время как Vue.js упрощает создание динамических интерфейсов с помощью компонентов и директив.

Интеграция Vue.js с Laravel начинается с установки Vue через npm или Yarn и настройки сборки фронтенда с помощью Laravel Mix, который предоставляет удобный интерфейс для Webpack. Это позволяет разработчикам компилировать и оптимизировать свои JavaScript и CSS файлы, а также использовать препроцессоры, такие как Sass или Less.

Одним из ключевых моментов интеграции является создание API на Laravel, которое будет общаться с фронтендом, разработанным на Vue.js. RESTful API позволяет фронтенду запрашивать, отправлять и обрабатывать данные в формате JSON, что обеспечивает гибкость и скорость взаимодействия между клиентом и сервером.

Кроме того, Vue.js может взаимодействовать с Laravel Blade шаблонами, позволяя разработчикам вставлять Vue компоненты непосредственно в HTML-разметку, что делает процесс разработки более интуитивно понятным и удобным.

Интеграция Vue.js с Laravel открывает перед разработчиками возможности для создания интерактивных и высокопроизводительных веб-приложений. Это сочетание технологий идеально подходит для проектов, требующих быстрого прототипирования, а также для крупных предприятий, стремящихся к созданию масштабируемых и легко поддерживаемых решений.

ГЛАВА 3.

**Разработка компонентов CRM-системы.**

**3.1. Проектирование пользовательского интерфейса CRM-системы.**

Проектирование пользовательского интерфейса CRM-системы начинается с определения структуры и компонентов, которые будут использоваться для взаимодействия с пользователем. Важно создать четкий и функциональный дизайн, который позволит пользователям эффективно управлять клиентскими данными и взаимодействиями. Ниже приведены примеры кода, демонстрирующие создание основных компонентов интерфейса в Vue.js.

**Компонент списка клиентов:**

<template>

<div class="clients-list">

<h1>Список клиентов</h1>

<ul>

<li v-for="client in clients" :key="client.id">

{{ client.name }} - {{ client.email }}

</li>

</ul>

</div>

</template>

<script>

export default {

data() {

return {

clients: [

{ id: 1, name: 'Компания 1', email: 'contact@company1.com' },

{ id: 2, name: 'Компания 2', email: 'contact@company2.com' },

],

};

},

};

</script>

<style>

.clients-list {

/\* Стили для списка клиентов \*/

}

</style>

**Компонент формы добавления клиента:**

<template>

<div class="client-form">

<h2>Добавить клиента</h2>

<form @submit.prevent="addClient">

<div class="form-group">

<label for="name">Имя:</label>

<input type="text" id="name" v-model="client.name">

</div>

<div class="form-group">

<label for="email">Email:</label>

<input type="email" id="email" v-model="client.email">

</div>

<button type="submit">Добавить</button>

</form>

</div>

</template>

<script>

export default {

data() {

return {

client: {

name: '',

email: '',

},

};

},

methods: {

addClient() {

// Логика добавления клиента

console.log('Клиент добавлен:', this.client);

// Очистка формы

this.client = { name: '', email: '' };

},

},

};

</script>

<style>

.client-form {

/\* Стили для формы добавления клиента \*/

}

</style>

Это базовая структура компонентов Vue.js для CRM-системы. Компонент списка клиентов отображает информацию о клиентах, а компонент формы позволяет добавлять новых клиентов в систему.

**3.2. Управление клиентскими данными: модели, миграции и API в Laravel.**

В процессе создания компонентов для CRM-системы особое внимание было уделено управлению клиентскими данными. Это включало в себя разработку моделей для представления этих данных, создание миграций для определения структуры базы данных и разработку API для обеспечения связи между бэкендом и фронтендом.

Модели и миграции

Первым шагом стало определение атрибутов клиента в системе. Было решено, что каждый клиент должен иметь имя, электронную почту и телефон. С помощью Artisan была создана модель Client и соответствующая миграция:

php artisan make:model Client -m

Эта команда сгенерировала класс модели в app/Models/Client.php и файл миграции в database/migrations/. В файле миграции была определена структура таблицы clients следующим образом:

use IlluminateDatabaseSchemaBlueprint;

use IlluminateSupportFacadesSchema;

Schema::create('clients', function (Blueprint $table) {

$table->id();

$table->string('name');

$table->string('email')->unique();

$table->string('phone')->nullable();

$table->timestamps();

});

После определения структуры таблицы была выполнена миграция для создания таблицы в базе данных:

php artisan migrate

API

Для обработки запросов к данным клиентов был создан контроллер ClientController:

php artisan make:controller Api/ClientController --api

В ClientController были добавлены методы для операций CRUD. Например, метод index возвращает список всех клиентов:

use AppModelsClient;

public function index()

{

return Client::all();

}

Метод store позволяет добавить нового клиента в систему. Входящие данные валидируются, после чего создается новая запись в базе данных:

use IlluminateHttpRequest;

public function store(Request $request)

{

$validated = $request->validate([

'name' => 'required|max:255',

'email' => 'required|email|unique:clients',

'phone' => 'nullable|max:255',

]);

$client = Client::create($validated);

return response()->json($client, 201);

}

Маршруты для API были определены в routes/api.php, где использовался ресурсный контроллер для создания маршрутов для всех стандартных операций CRUD:

Route::apiResource('clients', ApiClientController::class);

Разработка системы управления клиентскими данными стала ключевым этапом в создании CRM-системы. Были тщательно спроектированы модели и миграции для структурированного хранения данных, а также разработан четкий и эффективный API для взаимодействия с фронтендом. Эти компоненты обеспечили прочную основу для дальнейшего расширения функциональности системы и предоставили надежные средства для управления информацией о клиентах.

**3.3. Реализация функционала по работе с клиентами в Vue.js.**

При разработке CRM-системы одной из ключевых задач была реализация функционала по работе с клиентами на фронтенде, используя Vue.js. Это включало в себя отображение списка клиентов, добавление новых клиентов и возможность редактирования информации о существующих клиентах. Вот как я подошёл к решению этих задач:

Отображение списка клиентов

Для начала, создал компонент ClientsList.vue, который отвечает за отображение списка клиентов. В этом компоненте использовал axios для запроса данных с сервера:

<template>

<div>

<h2>Список клиентов</h2>

<ul>

<li v-for="client in clients" :key="client.id">

{{ client.name }} - {{ client.email }}

</li>

</ul>

</div>

</template>

<script>

import axios from 'axios';

export default {

data() {

return {

clients: [],

};

},

created() {

this.fetchClients();

},

methods: {

fetchClients() {

axios.get('/api/clients')

.then(response => {

this.clients = response.data;

})

.catch(error => {

console.error('Ошибка при получении списка клиентов:', error);

});

},

},

};

</script>

Добавление нового клиента

Далее, создал компонент AddClientForm.vue, который предоставляет форму для добавления нового клиента. В этом компоненте также использовал axios для отправки данных на сервер:

<template>

<form @submit.prevent="addClient">

<div>

<label for="name">Имя:</label>

<input type="text" id="name" v-model="client.name" required>

</div>

<div>

<label for="email">Email:</label>

<input type="email" id="email" v-model="client.email" required>

</div>

<button type="submit">Добавить клиента</button>

</form>

</template>

<script>

import axios from 'axios';

export default {

data() {

return {

client: {

name: '',

email: '',

},

};

},

methods: {

addClient() {

axios.post('/api/clients', this.client)

.then(() => {

alert('Клиент успешно добавлен');

this.client = { name: '', email: '' }; // Сброс формы

})

.catch(error => {

console.error('Ошибка при добавлении клиента:', error);

});

},

},

};

</script>

Редактирование информации о клиенте

Для редактирования информации о клиенте создал компонент EditClientForm.vue, аналогичный компоненту добавления клиента, но с предварительной загрузкой данных выбранного клиента и возможностью их изменения.

В каждом из этих компонентов использовал Vue.js для создания реактивного интерфейса, который позволяет пользователям взаимодействовать с данными клиентов в удобной и интуитивно понятной форме. Работа с API через axios обеспечивает связь с серверной частью приложения, позволяя выполнять операции CRUD над клиентскими данными.

**3.4. Внедрение системы управления состоянием с Vuex.**

При разработке CRM-системы на Vue.js столкнулся с необходимостью эффективного управления состоянием приложения, особенно когда дело доходит до обмена данными между компонентами. Для решения этой задачи было решено использовать Vuex, централизованное хранилище для всех компонентов приложения.

Настройка Vuex

Первым шагом была настройка Vuex в проекте. Создал файл store.js в папке src/store:

import Vue from 'vue';

import Vuex from 'vuex';

Vue.use(Vuex);

export default new Vuex.Store({

state: {

clients: [],

},

mutations: {

SET\_CLIENTS(state, clients) {

state.clients = clients;

},

ADD\_CLIENT(state, client) {

state.clients.push(client);

},

},

actions: {

fetchClients({ commit }) {

axios.get('/api/clients')

.then(response => {

commit('SET\_CLIENTS', response.data);

})

.catch(error => console.error('Ошибка при загрузке клиентов:', error));

},

addClient({ commit }, client) {

axios.post('/api/clients', client)

.then(() => {

commit('ADD\_CLIENT', client);

})

.catch(error => console.error('Ошибка при добавлении клиента:', error));

},

},

});

Использование Vuex в компонентах

С Vuex настроенным, я мог использовать его в компонентах для управления состоянием клиентов. В компоненте списка клиентов, например, теперь можно было получать список клиентов из хранилища и обновлять его при необходимости:

<template>

<div>

<ul>

<li v-for="client in clients" :key="client.id">

{{ client.name }} - {{ client.email }}

</li>

</ul>

</div>

</template>

<script>

import { mapState, mapActions } from 'vuex';

export default {

computed: {

...mapState(['clients']),

},

created() {

this.fetchClients();

},

methods: {

...mapActions(['fetchClients']),

},

};

</script>

Аналогично, в компоненте для добавления клиента использовал действие из Vuex для отправки данных на сервер и обновления состояния в хранилище:

<template>

<form @submit.prevent="submitForm">

<!-- Форма добавления клиента -->

</form>

</template>

<script>

export default {

data() {

return {

client: {

name: '',

email: '',

},

};

},

methods: {

submitForm() {

this.addClient(this.client).then(() => {

this.client = { name: '', email: '' }; // Сброс формы после добавления

});

},

...mapActions(['addClient']),

},

};

</script>

Интеграция Vuex в проект CRM-системы значительно упростила управление состоянием, особенно при работе с данными клиентов. Централизованное хранилище позволило эффективно обмениваться данными между компонентами и обеспечило единообразие и предсказуемость изменений состояния. Это сделало приложение более масштабируемым и упростило его поддержку.

ГЛАВА 4.

**Тестирование и деплой CRM-системы.**

**4.1. Методы тестирования веб-приложений: от юнит-тестов до E2E.**

В процессе разработки CRM-системы на Vue.js и Laravel, тестирование играло ключевую роль в обеспечении качества и надежности приложения. Тестирование помогало выявить и устранить ошибки на ранних этапах разработки, а также гарантировать, что новые функции не нарушают существующую логику. В этом разделе рассмотрим различные методы тестирования, применяемые в проекте, от юнит-тестов до тестирования конца к концу (E2E).

Юнит-тестирование

Юнит-тестирование — это процесс тестирования отдельных модулей или компонентов программного обеспечения. В контексте Laravel использовался PHPUnit для написания и выполнения юнит-тестов. Эти тесты фокусировались на проверке логики работы моделей, сервисов и других классов на бэкенде.

Юнит-тест для проверки создания клиента:

/\*\* @test \*/

public function a\_client\_can\_be\_added()

{

$this->withoutExceptionHandling();

$response = $this->post('/api/clients', [

'name' => 'Test Client',

'email' => 'test@example.com',

'phone' => '1234567890',

]);

$response->assertOk();

$this->assertCount(1, Client::all());

}

На фронтенде, для юнит-тестирования компонентов Vue.js использовался Jest. Тесты проверяли корректность отображения компонентов и их поведение при различных пользовательских взаимодействиях.

Юнит-тест для компонента Vue.js:

import { shallowMount } from '@vue/test-utils';

import ClientComponent from '@/components/ClientComponent.vue';

describe('ClientComponent', () => {

it('renders client name and email', () => {

const wrapper = shallowMount(ClientComponent, {

propsData: { client: { name: 'Test Client', email: 'test@example.com' } }

});

expect(wrapper.text()).toMatch('Test Client');

expect(wrapper.text()).toMatch('test@example.com');

});

});

Интеграционное тестирование

Интеграционное тестирование проверяет взаимодействие между различными частями приложения. В Laravel это часто включает тестирование контроллеров, чтобы убедиться, что правильные данные передаются между бэкендом и фронтендом.

Интеграционный тест:

/\*\* @test \*/

public function clients\_page\_returns\_correct\_data()

{

$client = factory(Client::class)->create();

$response = $this->get('/api/clients');

$response->assertOk();

$response->assertJson([$client->toArray()]);

}

Тестирование конца к концу (E2E)

Тестирование E2E проверяет работу приложения в целом, имитируя действия конечного пользователя. Для этого использовался Cypress. Тесты E2E позволяли проверить, как компоненты взаимодействуют друг с другом в реальных условиях использования.

E2E тест:

describe('Client Management', () => {

it('Allows a user to create a new client', () => {

cy.visit('/clients/new');

cy.get('input[name="name"]').type('New Client');

cy.get('input[name="email"]').type('newclient@example.com');

cy.get('form').submit();

cy.contains('New Client');

});

});

Тестирование является неотъемлемой частью разработки надежных веб-приложений. В проекте CRM-системы были использованы различные методы тестирования, от юнит-тестов до E2E, чтобы обеспечить высокое качество и стабильность приложения. Эти тесты помогли обнаружить и исправить множество ошибок на ранних этапах разработки, а также убедиться в том, что новые функции работают как ожидается.

**4.2. Настройка и использование PHPUnit для тестирования бэкенда.**

В процессе разработки CRM-системы на Laravel, PHPUnit стал основным инструментом для тестирования бэкенда. PHPUnit — это фреймворк для юнит-тестирования приложений на PHP, который позволяет автоматизировать тестирование классов и методов, обеспечивая высокую стабильность кода.

Настройка PHPUnit

После установки Laravel, PHPUnit уже включен в проект по умолчанию, что значительно упрощает начало работы с тестированием. Конфигурационный файл phpunit.xml расположен в корневой директории проекта и предварительно настроен для работы с тестовой базой данных и другими параметрами окружения.

Для тестирования бэкенда с использованием тестовой базы данных, важно указать параметры подключения к этой базе данных в файле .env.testing, который PHPUnit будет использовать вместо основного файла .env.

Создание тестов

Для создания нового теста используется команда Artisan:

php artisan make:test ClientTest

Эта команда создает новый тестовый класс в директории tests/Unit или tests/Feature, в зависимости от типа теста. Тесты, направленные на проверку отдельных компонентов системы, обычно размещаются в tests/Unit, в то время как тесты, проверяющие взаимодействие компонентов или целые функциональности, находятся в tests/Feature.

Написание тестов

Тестовый класс содержит методы, каждый из которых представляет собой отдельный тест. В Laravel для утверждений используется фасад IlluminateFoundationTestingAssert и другие вспомогательные методы, предоставляемые фреймворком для упрощения тестирования.

Вот как может выглядеть тестовый метод для проверки создания клиента:

public function testClientCreation()

{

$response = $this->postJson('/api/clients', [

'name' => 'New Client',

'email' => 'client@example.com',

'phone' => '1234567890',

]);

$response->assertStatus(201);

$this->assertDatabaseHas('clients', [

'email' => 'client@example.com',

]);

}

Запуск тестов

Для запуска тестов используется команда:

./vendor/bin/phpunit

Эта команда запускает все тесты в проекте. Для запуска конкретного теста можно указать путь к тестовому файлу:

./vendor/bin/phpunit tests/Feature/ClientTest.php

Использование PHPUnit для тестирования бэкенда в Laravel обеспечивает высокую надежность и стабильность кода. Настройка тестового окружения, создание и выполнение тестов позволяют автоматизировать проверку логики приложения, упрощая обнаружение и исправление ошибок на ранних этапах разработки. Это важный аспект создания качественного программного обеспечения.

**4.3. Тестирование фронтенда с использованием Vue Test Utils.**

В процессе разработки CRM-системы на Vue.js, для обеспечения качества и надежности фронтенда, было принято решение использовать Vue Test Utils в сочетании с Jest для написания и выполнения тестов компонентов. Vue Test Utils предоставляет удобные утилиты для монтирования компонентов, поиска элементов и триггеринга событий, что делает процесс тестирования более эффективным и организованным.

Настройка тестового окружения началась с установки необходимых зависимостей:

npm install --save-dev @vue/test-utils jest vue-jest babel-jest

Далее, в файле jest.config.js была сконфигурирована среда тестирования, указав обработчики для файлов .vue и настройки моков:

module.exports = {

moduleFileExtensions: ['js', 'json', 'vue'],

transform: {

'^.+\.vue$': 'vue-jest',

'^.+\.js$': 'babel-jest',

},

collectCoverage: true,

collectCoverageFrom: ['\*\*/\*.{vue}', '!\*\*/node\_modules/\*\*'],

};

С этими настройками можно было приступить к написанию тестов для компонентов Vue.js. Например, для тестирования компонента, который отображает информацию о клиенте, был создан следующий тест:

import { shallowMount } from '@vue/test-utils';

import ClientInfo from '@/components/ClientInfo.vue';

describe('ClientInfo.vue', () => {

it('отображает информацию о клиенте', () => {

const client = { name: 'Иван Иванов', email: 'ivan@example.com' };

const wrapper = shallowMount(ClientInfo, {

propsData: { client }

});

expect(wrapper.text()).toContain(client.name);

expect(wrapper.text()).toContain(client.email);

});

});

Этот тест проверяет, что компонент ClientInfo корректно отображает переданную ему информацию о клиенте. Использование shallowMount позволяет изолировать тестируемый компонент, не монтируя его дочерние компоненты, что ускоряет выполнение теста и делает его более предсказуемым.

Запуск тестов осуществлялся с помощью команды:

npm run test:unit

Эта команда запускала Jest с конфигурацией, указанной в jest.config.js, и выполняла все тесты, найденные в проекте.

Тестирование фронтенда с использованием Vue Test Utils и Jest позволило не только обеспечить работоспособность и надежность пользовательского интерфейса CRM-системы, но и значительно упростить процесс разработки, предоставив быстрый и удобный способ проверки новых функций и компонентов.

**4.4. Деплой и настройка окружения для CRM-системы.**

Для успешного запуска CRM-системы, разработанной на стеке Laravel и Vue.js, важно было не только написать код и протестировать его, но и правильно развернуть приложение на сервере. Этот процесс включал в себя настройку окружения, деплой приложения и его дальнейшее обслуживание. Вот как я подошёл к этим задачам:

Подготовка сервера

Первым шагом была подготовка сервера. Для CRM-системы выбрал VPS на Ubuntu, так как это популярный и хорошо поддерживаемый вариант. Установил на сервер необходимое программное обеспечение: Nginx как веб-сервер, PHP для выполнения кода Laravel, Composer для управления зависимостями PHP и Node.js с npm для сборки фронтенда.

Настройка окружения

Для Laravel важно правильно настроить файл .env, который содержит параметры окружения, такие как данные для подключения к базе данных, ключи приложения и другие важные настройки. После настройки .env файла, выполнил миграции базы данных и заполнил её начальными данными с помощью сидов:

php artisan migrate --seed

Сборка фронтенда

Следующим шагом была сборка фронтенда. В корне проекта на сервере запустил команду npm для установки зависимостей и сборки проекта:

npm install

npm run build

Это создало оптимизированные файлы фронтенда, которые были готовы к раздаче через веб-сервер.

Настройка веб-сервера

Настроил Nginx для обслуживания статических файлов, сгенерированных Vue.js, и перенаправления запросов к API на PHP-обработчик. В конфигурационном файле Nginx для сайта указал корневую директорию для статических файлов и настройки проксирования для API-запросов.

SSL и безопасность

Для обеспечения безопасности подключения к CRM-системе установил SSL-сертификат с помощью Let's Encrypt. Также проверил конфигурацию сервера и приложения на предмет потенциальных уязвимостей и настроил фаервол для ограничения доступа к необходимым портам.

Мониторинг и резервное копирование

Для обеспечения стабильной работы системы настроил мониторинг сервера с помощью таких инструментов, как Prometheus и Grafana. Также настроил регулярное резервное копирование базы данных и файлов проекта для предотвращения потери данных.

Деплой и настройка окружения для CRM-системы требовали внимательного подхода к деталям и планирования. Правильная настройка сервера, безопасность соединений и регулярное резервное копирование данных обеспечили надёжную и стабильную работу системы после запуска.

ГЛАВА 5.

**Заключение и перспективы развития проекта**

**5.1. Обзор достигнутых результатов и основных трудностей проекта.**

В заключительной главе проекта по созданию CRM-системы на Laravel и Vue.js провёл обзор достигнутых результатов и основных трудностей, с которыми столкнулся в процессе разработки. Этот анализ помог определить направления для дальнейшего развития и улучшения системы.

Достигнутые результаты

Проект был успешно завершён с реализацией всех запланированных функций. CRM-система обладает полным набором инструментов для управления клиентскими данными, включая создание, просмотр, редактирование и удаление информации о клиентах. Интерфейс пользователя был разработан с учётом удобства и интуитивной понятности, что позволяет пользователям эффективно работать с системой.

Благодаря использованию современных технологий, таких как Vue.js для фронтенда и Laravel для бэкенда, система обладает высокой производительностью и масштабируемостью. Тестирование на разных этапах разработки, включая юнит-тесты, интеграционные и E2E тесты, обеспечило надёжность и стабильность работы CRM.

Основные трудности

В ходе работы над проектом возникли некоторые трудности. Одной из них была настройка среды разработки и деплой, особенно в части конфигурации веб-сервера и обеспечения безопасности. Также были сложности с оптимизацией производительности приложения при работе с большим объёмом данных.

Интеграция между фронтендом и бэкендом требовала тщательной работы для обеспечения бесперебойного обмена данными и корректной обработки ошибок. Поддержание кодовой базы в актуальном состоянии и управление зависимостями также представляли определённые вызовы.

Завершение проекта CRM-системы стало значительным достижением, которое показало возможности современных веб-технологий в решении бизнес-задач. Несмотря на трудности, проект был реализован с соблюдением высоких стандартов качества и безопасности. Полученный опыт и обратная связь от пользователей послужат основой для дальнейшего развития и улучшения системы.

**5.2. Возможности дальнейшего развития и масштабирования CRM-системы.**

Завершение работы над CRM-системой на Laravel и Vue.js открыло передо мной новые горизонты для дальнейшего развития и масштабирования проекта. Опыт, полученный в ходе разработки, и обратная связь от пользователей позволили выявить ряд направлений для улучшения и расширения функциональности системы.

Интеграция с внешними сервисами

Одним из направлений развития является интеграция CRM-системы с внешними сервисами и платформами, такими как социальные сети, электронная почта и сервисы облачного хранения. Это позволит автоматизировать процессы сбора данных о клиентах, улучшить коммуникацию и повысить общую эффективность работы с клиентской базой.

Расширение аналитических возможностей

Разработка дополнительных аналитических инструментов и дашбордов для отслеживания ключевых показателей эффективности (KPI) поможет пользователям системы принимать обоснованные решения на основе актуальных данных. Внедрение машинного обучения и искусственного интеллекта может предоставить возможности для прогнозирования поведения клиентов и оптимизации маркетинговых стратегий.

Улучшение пользовательского опыта

Продолжение работы над улучшением пользовательского интерфейса и опыта взаимодействия с системой остается приоритетом. Это включает в себя разработку адаптивного дизайна для удобства использования CRM на различных устройствах, улучшение навигации и оптимизацию производительности фронтенда.

Развитие мобильной версии

Создание мобильного приложения или полноценной мобильной версии CRM-системы позволит пользователям эффективно работать с системой в любом месте и в любое время, что особенно важно для современного динамичного бизнеса.

Расширение функционала управления проектами

Внедрение дополнительных инструментов для управления проектами и задачами в рамках CRM-системы может значительно расширить её функциональность и сделать её универсальным инструментом для управления бизнес-процессами.

Перспективы дальнейшего развития и масштабирования CRM-системы кажутся безграничными. Постоянное внедрение новых технологий, улучшение существующих функций и прислушивание к потребностям пользователей позволят создать мощный инструмент, способный удовлетворить самые разнообразные бизнес-задачи и способствовать росту и развитию компаний.

**Приложения.**

**Приложение А. Исходный код основных компонентов системы.**

В этом разделе представлены фрагменты исходного кода основных компонентов CRM-системы, разработанной на стеке Laravel и Vue.js. Эти примеры демонстрируют структуру и логику работы ключевых частей приложения.

Модель клиента (Laravel)

// app/Models/Client.php

namespace AppModels;

use IlluminateDatabaseEloquentModel;

class Client extends Model

{

protected $fillable = ['name', 'email', 'phone'];

}

Контроллер клиента (Laravel)

// app/Http/Controllers/ClientController.php

namespace AppHttpControllers;

use AppModelsClient;

use IlluminateHttpRequest;

class ClientController extends Controller

{

public function index()

{

return Client::all();

}

public function store(Request $request)

{

$validatedData = $request->validate([

'name' => 'required|max:255',

'email' => 'required|email|unique:clients',

'phone' => 'nullable|max:255',

]);

$client = Client::create($validatedData);

return response()->json($client, 201);

}

}

Компонент списка клиентов (Vue.js)

// src/components/ClientsList.vue

<template>

<div>

<ul>

<li v-for="client in clients" :key="client.id">

{{ client.name }} - {{ client.email }}

</li>

</ul>

</div>

</template>

<script>

export default {

data() {

return {

clients: [],

};

},

created() {

this.fetchClients();

},

methods: {

fetchClients() {

axios.get('/api/clients')

.then(response => {

this.clients = response.data;

})

.catch(error => {

console.error('Ошибка при получении списка клиентов:', error);

});

},

},

};

</script>

**Приложение Б. Руководство пользователя по работе с CRM-системой**

Это руководство предназначено для пользователей CRM-системы и описывает основные функции и процессы работы с системой.

Вход в систему

1. Откройте веб-браузер и введите адрес CRM-системы.

2. Введите свои учетные данные (логин и пароль) в соответствующие поля на странице входа.

3. Нажмите кнопку "Войти".

Работа со списком клиентов

1. Перейдите в раздел "Клиенты" в главном меню.

2. Для добавления нового клиента нажмите кнопку "Добавить клиента" и заполните необходимые поля в появившейся форме.

3. Для просмотра или редактирования информации о клиенте нажмите на имя клиента в списке.

Добавление клиента

1. В разделе "Клиенты" нажмите кнопку "Добавить клиента".

2. Введите имя, электронную почту и телефон клиента.

3. Нажмите кнопку "Сохранить" для добавления клиента в систему.

Редактирование информации о клиенте

1. В списке клиентов выберите клиента, информацию о котором хотите отредактировать.

2. Введите новые данные в соответствующие поля.

3. Нажмите кнопку "Сохранить изменения".

Это руководство охватывает основные аспекты работы с CRM-системой. Для получения дополнительной информации обратитесь к документации или поддержке.

**Список использованных источников.**

В процессе разработки CRM-системы на стеке Laravel и Vue.js были использованы различные источники информации, включая официальную документацию, статьи, руководства и форумы. Ниже представлен список наиболее значимых источников, которые оказали влияние на проект и помогли в решении технических задач.

1. Официальная документация Laravel - https://laravel.com/docs

2. Официальная документация Vue.js - https://vuejs.org/v2/guide/

3. Vue Test Utils - https://vue-test-utils.vuejs.org/

4. PHPUnit Manual - https://phpunit.de/manual/

5. Laracasts - https://laracasts.com/

6. Stack Overflow - https://stackoverflow.com/

7. GitHub - https://github.com/

8. MDN Web Docs - https://developer.mozilla.org/