Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем
Направление подготовки 09.04.04 – Программная инженерия
Направленность (профиль) образовательной программы Управление разработкой программного обеспечения

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой
_____ А.В. Бушманов
«____»_____2021 г.

МАГИСТЕРСКАЯ РАБОТА

на тему: Разработка веб-приложения для реализации дифференцированных каталогов данных с применением полнотекстового поиска

Исполнитель студент группы 957-ом	_____ (подпись, дата)	А.Ю. Манвелян
Руководитель доцент, канд. техн. наук	_____ (подпись, дата)	Т.А. Галаган
Руководитель научного содержания программы магистратуры профессор, д-р техн. наук	_____ (подпись, дата)	И.Е. Еремин
Нормоконтроль инженер кафедры	_____ (подпись, дата)	В.Н. Адаменко
Рецензент доцент, канд. физ.-мат. наук	_____ (подпись, дата)	Д.В. Фомин

Благовещенск 2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем

УТВЕРЖДАЮ
Зав. кафедрой
_____ А.В. Бушманов
«____» _____ 2021 г,

З А Д А Н И Е

К магистерской диссертации студента Манвеляна Артема Юрьевича

1. Тема магистерской диссертации: Разработка веб-приложения для реализации дифференцированных каталогов данных с применением полнотекстового поиска.

(утверждено приказом от 01.03.2021 №412-уч)

2. Срок сдачи студентом законченной работы (проекта): _____

3. Исходные данные к магистерской диссертации: отчет о прохождении преддипломной практики, ГОСТы, научные публикации, дополнительная литература.

4. Содержание магистерской диссертации (перечень подлежащих разработке вопросов): анализ современных решений для эффективной и полнофункциональной веб-разработки, описание алгоритмического и программного обеспечения при разработке приложения для реализации дифференцированных каталогов данных, разработка и тестирование веб-приложения для реализации дифференцированных каталогов данных.

5. Перечень материалов приложения: А – руководство пользователя.

6. Дата выдачи задания: 25.02.2021

Руководитель магистерской диссертации: Галаган Татьяна Алексеевна, доцент, канд. техн. наук.

Задание принял к исполнению (25.02.2021): _____

РЕФЕРАТ

Магистерская диссертация содержит 97 с., 48 рисунков, 1 таблицу, 1 приложение, 44 источника.

ВЕБ-ФРЕЙМВОРКИ, ВЕБ-ПРИЛОЖЕНИЕ, ДИФФЕРЕНЦИРОВАННЫЙ КАТАЛОГ, ПОЛНОТЕКСТОВЫЙ ПОИСК, LARAVEL, PHP, БАЗА ДАННЫХ, VUE.JS, АРХИТЕКТУРА, API, PORTO, VUEX

Предметом магистерской диссертации является исследование построения приложений с использованием современных веб-технологий (фреймворков Laravel и Vue.js), изучение и применение систем полнотекстового поиска.

Целью магистерской диссертации является разработка с учетом требований веб-приложения (инструмента), предоставляющего возможность создавать дифференцированные каталоги данных и применять технологию полнотекстового поиска.

Выполнение работы выполняется в несколько стадий. Первой стадией является анализ современных решений для эффективной и полнофункциональной веб-разработки, где рассматриваются современные веб-фреймворки и эффективность применения систем полнотекстового поиска. Вторая стадия – описание программного обеспечения и архитектурного решения применяемого при разработке приложения для реализации дифференцированных каталогов данных. На третьей стадии происходит описание разработки и тестирования разработанного веб-приложения, а также написание руководства пользователя.

В результате было разработано и находится на стадии тестирования веб-приложение для реализации дифференцированных каталогов данных.

СОДЕРЖАНИЕ

Введение 8
1 Анализ современных решений для полнофункциональной веб-разработки 11
1.1 Актуальность веб-фреймворков для разработки интернет-приложений 11
1.2 Особенности разработки веб-приложений на фреймворке Laravel 15
1.3 Эффективность применения систем полнотекстового поиска 17
2 Алгоритмическое и программное обеспечение при разработке приложения для реализации дифференцированных каталогов данных 23
2.1 Выбор архитектуры при разработке программного обеспечения 23
2.2 Обоснование выбора средств разработки 32
2.2.1 Выбор серверного и клиентского фреймворков 32
2.2.2 Выбор типа базы данных и СУБД 36
2.2.3 Выбор системы полнотекстового поиска 39
2.3 Характеристика выбранного программно-технического обеспечения 40
2.3.1 Характеристика выбранного технического обеспечения 40
2.3.2 Характеристика выбранного программного обеспечения 43
3 Разработка и тестирование веб-приложения для реализации дифференцированных каталогов данных 49
3.1 Этапы разработки программного обеспечения 49
3.1.1 Описание требований 49
3.1.2 Выбор модели жизненного цикла 51
3.1.3 Реализация архитектуры 53
3.1.4 Реализация работы с данными 60
3.1.5 Реализация работы с внутренними и внешними интерфейсами 62
3.1.6 Декомпозиция работы с основным функционалом 69
3.2 Тестирование программного обеспечения 72
3.3 Применение разработанного приложения 75
Заключение 78
Библиографические ссылки 79

Библиографический список 81
Приложение А 85

НОРМАТИВНЫЕ ССЫЛКИ

В настоящей магистерской диссертации использованы ссылки на следующие стандарты и нормативные документы:

ГОСТ 19.601-78 ЕСПД Единая система программной документации

ГОСТ 19.004-80 ЕСПД Термины и определения

ГОСТ 19.103-77 ЕСПД Стадии разработки

ГОСТ 19.103-77 ЕСПД Обозначения программ и программных документов

ГОСТ 19.104-78 ЕСПД Основные надписи

ГОСТ 19.105-78 ЕСПД Общие требования к программным документам

ГОСТ 19.106-78 ЕСПД Требования к программным документам, выполненным печатным способом

ГОСТ 19.506-79 ЕСПД Описание языка. Требования к содержанию и оформлению

ГОСТ 19.401-78 ЕСПД Текст программы. Требования к содержанию и оформлению

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ

API – application programming interface (программный интерфейс приложения);

JSON – javascript object notation (текстовый формат обмена данными);

MVC – model-view-controller (модель-представление-контроллер);

ORM – object-relational mapping (объектно-реляционное отображение);

XP – extreme programming (экстремальное программирование);

ПО – программное обеспечение;

СУБД – система управления базами данных.

ВВЕДЕНИЕ

В наши дни во всем мире наблюдается стремительно усиливающийся рост объёма и скорости возникновения различной информации, увеличение количества научных, научно-популярных и публицистических статей, фотоаппараты и видеокамеры стали широко распространены, а на компьютерах пользователей стали хранится тысячи терабайт дифференцированных данных. В связи с этим существует проблема хранения, организации и структуризации таких данных, необходимости удобного поиска среди них и публикации в глобальной сети, которая и рассмотрена в работе.

Исходя из поставленной проблемы была сформулирована цель работы, которая заключается в разработке с учетом требований веб-приложения (инструмента), предоставляющего возможность создавать дифференцированные каталоги данных и применять технологию полнотекстового поиска.

В качестве задач были выделены несколько основных:

- сформировать требования к разрабатываемому ПО;

- рассмотреть основные паттерны при разработке современных веб-приложений и внедрить необходимые из них;

- реализовать серверную часть приложения с применением фреймворка Laravel;

- выполнить проект и разработать API приложения;

- применить технологию полнотекстового поиска;

- реализовать базовый шаблон для представления данных с помощью клиентского фреймворка Vue.js.

Предметом работы является исследование построения приложений с использованием современных веб-технологий (серверного фреймворка Laravel и клиентского – Vue.js), изучение и применение систем полнотекстового поиска.

Первая глава магистерской работы содержит в себе анализ современных решений для эффективной и полнофункциональной веб-разработки. Здесь рассматривается актуальность использования современных веб-фреймворков для

разработки интернет-приложений, особенности разработки приложений с помощью серверного PHP-фреймворка Laravel, его интеграция с клиентским JavaScript фреймворком Vue.js, а также описывается эффективность применения систем полнотекстового поиска.

Во второй главе описывается алгоритмическое и программное обеспечение при разработке веб-приложения для реализации дифференцированных каталогов данных с применением полнотекстового поиска. Представляется архитектурное решения, применяемое при разработке, также происходит обоснование выбора средства разработки и соответственно характеристика выбранного программного обеспечения.

В третье главе рассматривается процесс разработки, а именно описание требований, модели жизненного цикла, а также показана реализации выбранной архитектуры, работа с данными и интерфейсами в приложении. Далее рассказывается о процессе тестирования разработанного веб-приложения и его применение. Также было написано руководство пользователя, с которым можно ознакомиться в приложении А.

Научная новизна исследования заключается в применении совокупности технологий (фреймворков Laravel и Vue.js), ранее не использовавшихся для подобных приложений, кроме того, применение технологии полнотекстового поиска, использование которой в алгоритмах работы приложения повышает релевантность поиска в дифференцированных каталогах данных.

Практическая значимость магистерской диссертации заключается:

- в предоставлении пользователям разработанного в рамках диссертационной работы более простого и при этом бесплатного варианта приложения для каталогизации данных с возможностью полнотекстового поиска;

- в возможности для сторонних разработчиков быстрого и удобного расширения данного открытого приложения за счет применения в нем модульной архитектуры и возможности внешней авторизации.

Апробация результатов диссертационного исследования отображена в следующих научных публикациях:

- Манвелян А.Ю., Научный руководитель – Галаган Т.А. Проектирование веб-приложения для реализации каталогов данных с применением полнотекстового поиска // Молодежь XXI века: шаг в будущее: материалы XXI региональной научно-практической конференции (20 мая 2020г., Благовещенск). – Изд-во Дальневост. гос. аграр. ун-та, 2020. – С. 122-123;

- Манвелян, А.Ю., Галаган Т.А. Применение фреймворка Laravel при разработке веб-приложения для управления дифференцированными каталогами данных // Системный анализ в медицине: матер. XIV межд.научн.конф. (САМ – 2020) (15-16 октября 2020г., Благовещенск). – Благовещенск: Изд-во ДНЦ ФПД, 2020. – С. 58-60;

- Манвелян А.Ю., Научный руководитель – Галаган Т.А. Разработка web-приложения для реализации каталогов данных // Молодежь XXI века: шаг в будущее: материалы XXII региональной научно-практической конференции (20 мая 2021г., Благовещенск). – Благовещенск: Изд-во БГПУ, 2021. – С. 766-767;

- Манвелян, А.Ю., Галаган Т.А. Применение хранилища состояния Vuex при разработке пользовательских интерфейсов с помощью Vue.js на основе фреймворка Laravel // «Современные инновации в науке и технике» (МТО-57): материалы 11-й Всероссийской научно-технической конференции (15-16 апреля 2021 г., Курск). – С. 166-170.

Также результаты диссертационного исследования были представлены на следующих научных конференциях:

- XXI региональная научно-практическая конференция «Молодежь XXI века: шаг в будущее» – 2020, 20.05.21;

- XXX научная конференция АмГУ «День науки» – 2020, 15.04.2021.

1 АНАЛИЗ СОВРЕМЕННЫХ РЕШЕНИЙ ДЛЯ ПОЛНОФУНКЦИОНАЛЬНОЙ ВЕБ-РАЗРАБОТКИ

1.1 Актуальность веб-фреймворков для разработки интернет-приложений

Веб-фреймворки внесли большой вклад в профессиональную разработку и в данный момент практически незаменимы в практически любой области веб-программирования для относительно быстрого и менее затратного процесса написания приложений.

В 90-ые годы 20 века, во время расцвета сайтов и веб-индустрии в целом, приложения писались с нуля, каждый писал по-своему, что приводило к проблеме отсутствия единого стандарта стиля, только непосредственно сам разработчик мог быстро изменить или развернуть их. Веб-фреймворки решили данную проблему, привнеся некоторые правила написания и структурирования кода, а также предоставляя набор готовых библиотек. Так, ближе к концу 90-ых, сложности, связанные с написанием кода приложений, были нивелированы с появлением общепринятого подхода к разработке веб-приложений. В это же время появляются языки, используемые исключительно для интернет-разработок. Их разнообразие в данный момент позволяет выбрать наиболее подходящий в зависимости от потребностей.

Существует два основных типа веб-фреймворков:

- back-end фреймворки, работающие на серверной стороне приложения и отвечающие за отдельные, но очень важные части приложения, без которых оно не сможет нормально работать (API, бизнес-логика, планировщики задач и др.);

- front-end фреймворки, работающие на клиентской стороне, то есть в браузере и отвечающие за внешний вид приложения, отображение данных в наиболее подходящем для этого виде.

Свойства и архитектурные особенности серверных фреймворков не дают возможности создавать сложные динамические веб-интерфейсы. С помощью них можно создавать простые статические веб-страницы, ограниченные в

функционале и валидации формы, также они могут формировать выходные данные для API и отвечать за авторизацию. Их главное цель состоит в управлении приложением на стороне сервера, где они отвечают за критически важные части, без которых приложение не сможет нормально функционировать. Ниже представлены несколько наиболее известных и популярных веб-фреймворков и языки программирования, на которых с ними необходимо работать:

- Django – Python;
- Laravel – PHP;
- Express.js – JavaScript;
- Ruby on Rails – Ruby;
- .NET Core – C#;
- Spring Framework – Java.

В отличие от серверных, клиентские фреймворки не взаимодействуют с логикой приложения. Данный тип фреймворков работает на стороне клиенте в браузере. Они используются для создания многофункциональных динамических интерфейсов, а также позволяют создавать различные анимации и так называемые одностраничные приложения (Single-page application). Стоит отметить, что клиентские фреймворки отличаются по функциональности и области применения. Некоторые весят десятки килобайт и предназначены слабонагруженных приложений, а некоторые могут вести десятки мегабайт и содержат в себе инструменты для создания практически любого приложения. Вот наиболее популярные и используемые:

- Angular;
- React.js;
- Svelte;
- Backbone.js;
- Vue.js.

Представленные клиентские фреймворки используют JavaScript в качестве языка программирования.

Существуют фреймворки, используя которые можно создать как серверную, так и клиентскую часть приложения, так называемые full-stack фреймворки, как, например, Meteor или Nuxt.js. Серверная и клиентская части работают на одном языке JavaScript, и для них используется один и тот же код. Обычно такие фреймворки работают в режиме реактивности и имеют двустороннюю, что позволяет на ходу менять данные, так, к примеру, если что-то меняется в одной части интерфейса, то изменения происходят и в других [1].

Рассмотрим основные отличительные особенности веб-фреймворков, которые делают их быстрыми, удобными и полнофункциональными при разработке (рисунок 1.1).

Рисунок 1.1 – Особенности веб-фреймворков

Технология шаблонов является набором различных методов и программного кода, реализованных для удобного и быстрого создания веб-страниц. Обычно каждый фреймворк имеет свой шаблонизатор, которые используется для представления HTML контента на странице.

Механизм роутинга представляет собой сопоставление путей в браузере с функционалом на сервере, там самым это упрощает работу программиста, создает красивые пути для пользователя и улучшает индексирование веб-приложения в поисковых системам.

Веб-кэширование позволяет сохранять состояния (хэши) страниц приложения, чтобы избежать сильной нагрузки на сервер улучшить пользовательский опыт при работе с сайтом, так как страницы грузятся именно их кэша.

Scaffolding технология позволяет сгенерировать структуру, типичные части или даже готовый проект, который нужно будет доработать под нужды разрабатываемого приложения, что экономит время и увеличивает скорость разработки, а также стандартизирует кодовую базу [2].

Все фреймворки содержат в себе огромное количество готовых библиотек и решений для разработки, призванных облегчить и ускорить разработку. Также существует множество обучающих сайтов и документаций, которые обеспечивают быстрое введение в технологию.

Большая часть веб-фреймворков частично или полностью поддерживает модель MVC, что позволяет строить стандартизированные, хорошо структурируемые и расширяемые интернет-приложения. Схема модели MVC представлена на рисунке 1.2.

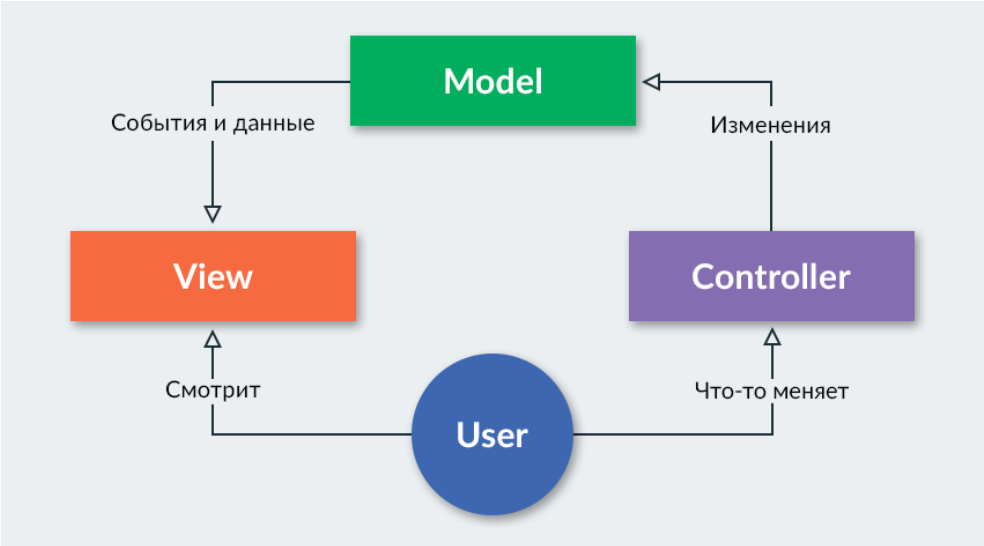

Рисунок 1.2 – Схема архитектурной модели MVC

В целом большинства фреймворков предлагают реализацию или основаны на модульной архитектуре, что позволяет добавлять необходимый функционал в виде отдельных модулей по мере необходимости. Существует множество отдельных энтузиастов и тематических сообществ, пакеты расширения для популярных фреймворков, что также является одним из главных преимуществ веб-фреймворков, что делает их достаточно актуальным выбором по сравнению затратами по созданию приложений с нуля, и конечно же все актуальные фреймворков часто обновляются, что говорит о их популярности и востребованности в сообществе, а вместе с их свободностью делает изучение и использование хотя бы одного серверного и клиентского фреймворка больше необходимостью, чем приятным дополнением [3].

1.2 Особенности разработки веб-приложений на фреймворке Laravel

Разработка приложений с использованием серверного фреймворка Laravel обычно содержит в себе следующие этапы:

- формирование главных целей и задач разрабатываемого проекта;

- формирование необходимого окружения для решения необходимой задачи. Обычно архитектура строится по принципу «от основного к второстепенному». Работа с базой данных ведется с использованием Eloquent ORM, которая предоставляет объектно-ориентированный интерфейс для написания sql-запросов;

- создание интерфейса и связывание его с серверными роутами;

- наполнение приложения необходимым тематическими данными;

- тестирование работоспособности и отказоустойчивости, а также и исправление ошибок;

- развертывание на хостинге, настройка и запуск приложения;

- сопровождение и поддержка, доработка программного обеспечения.

Перечень основных преимуществ и недостатков

Разработка на Laravel достаточно похожа на создание веб-ресурсов с использованием фреймворка Symphony, большую часть которого включает в себя Laravel и имеет ряд достоинств:

- большой набор функционала «из коробки»;

- возможность разрабатывать масштабируемые веб-приложения независимо от сложности и выбранной отрасли;

- простая и удобная реализация различных, обычно достаточно, функциональных возможностей;

- предоставление механизмов тестирования и быстрого обновления фреймворка за счет поддержки различных версий интернет-ресурса;

- простая и понятная система пакетов, реализуемая с помощью пакетного менеджера Composer, что позволяет с легкостью находить и подключать любые необходимые компоненты для разрабатываемой системы;

- надежная защита базы данных от SQL, CSRF, XSS;

- все изменения в PHP и направлений развития веб-сайтов учитываются в обновлениях исходного кода данного фреймворка и во всех последующих версиях, но также сохраняется обратная совместимость;

К недостаткам разработки сайта на данном фреймворке можно отнести следующее:

- разработка занимает больше времени в основном из-за более высокого порога вхождения, особенно для новичков;

- рассчитано больше для профессиональных разработчиков.

Также фреймворк предоставляет различные инструменты для улучшения приложения, соблюдения безопасности при обработке данных и упрощения жизни разработчикам, как, например: мощную систему валидации данных, защиту от подделки межсайтовых запросов. Также содержит в себе удобный интерфейс для построения запросов и модуль для работы с моделями базы данных, который имеет защиту от различный SQL-атак. Кроме стандартной авторизации и аутентификации в фреймворке в виде отдельных пакетов представлена возможность API аутентификация с помощью Laravel Passport или Laravel Sanctum, где первый предоставляет полную реализацию сервера OAuth2 [4], а второй – интерфейс JWT токенов для связи клиентской и серверной части.

Говоря о эффективной разработке интерфейсов, стоит отметить, что Laravel поставляется с JavaScript фреймворком Vue.js, который удобно интегрирован и позволяет эффективнее разрабатывать клиентскую часть по сравнению с другими клиентскими решениями. Vue.js предоставляет механизм составных компонентов, который тесно взаимодействует с blade шаблонами, предоставляемыми Laravel, позволяет передавать в них данные напрямую из кода, что позволяет пользователю реактивно вносить изменения в приложении, переключаться между компонентами, которые будут перерисовываться на ходу без необходимости перезагружать страницу. Так. Например, вы можете создать компоненты для загрузки различных данных, легко переключаться между ними и предоставить пользователю лучший опыт по работе с интерфейсом, всплывающими подсказками и отсутствием перезагрузок. Также Vue.js позволяет создавать SPA на базе фреймворка Laravel, совместим с различными хранилищами состояний, такими как Vuex и Redux, которые помогают справляться с обработкой потоков данных в сложных приложениях. Тем самым можно утверждать, что использование связки Vue.js и Laravel в значительной степени увеличат эффективность разработки приложений, а также облегчат и ускорят её.

В итоге разработка сайта на Laravel – это свобода творчества и возможность реализовать практически любой интернет-проект, независимо от его направленности и сложности, что объясняется наличием большого выбора дополнительных пакетов, а также способностью усовершенствования под высокие нагрузки.

1.3 Эффективность применения систем полнотекстового поиска

Большинство современных СУБД предоставляют поиск заданных шаблонов в столбце, но не обеспечивают релевантность поиска или делают это с низкой эффективность. В то же время современные поисковые системы предлагают такие функции по умолчанию и часто являются приоритетным выбором.

Так, многие проекты, которые начинались с стандартного поиска реляционной базы данных, затем, по мере увеличение объема данных или количества пользователей, перешли на систему полнотекстового поиска, потому что таким

образом улучшался пользовательский и удовлетворялись потребности более широкого круга пользователей. В других случаях это связано с тем, что первоначальный выбор поиска в базе данных был сделан потому, что команда была более знакома с архитектурой базы данных и её особенностей, но из-за снижения производительности была вынуждена переключиться на полнотекстовый поиск. Также возможно использование вместе обоих технологий, либо для разных частей приложение, либо проводя данные через обе технологии для того, чтобы получить преимущества обеих и предоставлять пользователю более релевантные поисковые данные. Сравним системы полнотекстового поиска и поиск в реляционных базах данных.

Сравнение полнотекстового поиска и поиска в реляционных баз данных

Полнотекстовые системы лучше подходят для быстрого поиска в больших объемах неструктурированного, частично структурированного или структурированного текста по определенному слову или словам. Они предоставляют возможности расширенного текстового поиска и систему ранжирования по релевантности, чтобы выдать результаты по соответствию их «нечеткому» поисковому запросу, например при в опечатках или схожих по значению словах. Системы полнотекстового поиска также предоставляют механизмы обработки естественного языка для понимания требований и нужд пользователя и имеют алгоритмы рекомендаций, что предоставляет пользователю лучший персонализированный опыт использования приложения.

Рассматривая же реляционные базы данных, стоит сказать, что они достаточно эффективно справляются с хранением и обработкой структурированных данных, поддерживают поиск нескольких типов записей для заданных полей и эффективны для быстрого обновления определенных отдельных записей. Но релевантность результатов, получаемых из базы данных, не будет иметь такого же качества обработки в системах полнотекстового поиска. Также, если администратор базы данных не знает, какие запросы чаще всего отправляют пользователи, то производительность используемой СУБД будет довольно невысокой, что ухудшит опыт пользователя при взаимодействии с приложением.

Использование и принцип работы систем полнотекстового поиска

Системы полнотекстового поиска отлично подходят для быстрого и эффективного поиска в больших объемах текстовой информации. Сюда могут входить неструктурированные данные, такие как текстовые документы форматов: doc, docx, odt, pdf, txt и другие, а также частично структурированные, как, например HTML страницы, XML, JSON, YAML файлы, которые структурированы по определённым правилам и могут содержать большое количества текста. Поисковые системы также могут классифицировать данную информацию на основе определенных значений в приведенных данных, например, по сортировке (по алфавиту, ценовому диапазону, региону, цвету, размеру, типу файла, автору). Возможности полнотекстового поиска также включают поддержку базового поиска, по ключевым словам, логическим операторам, предварительно обрабатывать фразы естественного языка и другой различный функционал.

Помимо всего полнотекстовые поисковые системы умеют ранжировать данные по релевантности для определения наилучшего соответствия запросу. Здесь берется во внимание частота запросов поиска по документу, их частота в целом и близость внутри документа.

Системы полнотекстового поиска используют индекс для выполнения запросов. Наиболее часто используемым является инвертированный индекс, который считает каждый термин в каждом документе и указывает их расположение, то есть какие файлы содержат этот термин и где эти термины встречаются в документах. Для каждых данных может быть отдельный индекс, или все может содержаться в одном индексе. Также обычно есть возможность обработки нетекстовых полей, таких как, например числовой диапазон или дата. Но эти возможности обычно должны быть найдены в сами документы и обычно не вычисляются поисковой системой в режиме реального времени.

Помимо индексации данных, большинство полнотекстовых поисковых систем могут сохранять и извлекать данные в их изначальном виде. Такая возможность нужна, чтобы заполнить список результатов поиска фактическими

данными из необходимых документов, так как это дает пользователям лучшее представление о документе, прежде чем непосредственно открыть его.

Стоит отметить, поисковые системы ограничены в своей способности быстро и безопасно обрабатывать транзакционные обновления как это делает реляционная СУБД и им необходимо адаптироваться к изменениям, попадающим в индекс поиска, что позволяет показывать пользователю сложную разветвленную информацию об объекте.

Также система полнотекстового поиска может быть более эффективным инструментом, если данные из таблиц будут преобразованы в единый формат, наиболее подходящий для полнотекстовых операций. Это наиболее полезно, когда есть только одна или несколько таблиц и ограничения в обработке транзакций.

Полнотекстовые поисковые системы, чтобы влиять на релевантность пользовательских запросов, кроме текста предоставленного в индексе, также могут полагаться на нейронные сети. Действия пользователей агрегируются и передаются сохраняются в системе, чтобы обеспечить более релевантные результаты и улучшить взаимодействие с пользователем.

Рисунок 1.3 – Агрегирование различных источников данных

Еще одна отличительная черта поисковых систем – возможность индексировать данные из множества различных источников (рисунок 1.3), как файловые

системы, веб-сервера, CRM-системы, баз данных и многих других источников информации. Запрос ищет информацию во любой из этих систем или во всех с учетом предоставленных прав доступа.

Исходя из вышесказанного можно выделить основные особенности полнотекстового поиска:

- почти мгновенный поиск в большом количестве документов, содержащих несколько терминов. Поиск включает в себя эффективную категоризацию результатов поиска на основе значений выбранных полей;

- гибкие инструменты запросов с возможностью распределения для поиска наиболее релевантных документов;

- самообучение системы в реальном времени на основе предыдущих запросов и поведения пользователей с повышением релевантности поиска;

- возможности для добавления, удаления или обновления документов;

- кроме индексации и поиска имеется возможность хранения данных для более визуально понятного представления их пользователю.

Когда использовать в приложении систему полнотекстового поиска

Требования к приложению, которые предполагают выбор системы полнотекстового поиска вместо или наряду с реляционной базой данных:

- приложение будет индексировать большой объем преимущественно текстовой информации, а запросы пользователей трудно предугадать заранее, и они часто могут меняться;

- запросы могут охватывать несколько различных источников данных, как файловая система, различные API, CRM, CMS, базы данных и др;

- потенциально приложение должно справляться с обработкой большого количества запросов;

- приложению необходимо поддерживать специфичные запросы к данным;

- оптимальная релевантность не может быть достигнута с помощью поиска в СУБД.

Итоги

Технологии полнотекстового поиска отлично подходят для быстрого поиска и обработки больших объемов данных, но не так сильны в обработке типов записей или транзакций. Эффективность, предлагаемая современной поисковой системой, более чем достаточна в большинстве ситуаций и часто является единственным решением для приложений, обрабатывающих большие объёмы данных.

Многие проекты имеют тенденцию полностью переводить с реляционной базы данных на систему полнотекстового поиска, если СУБД больше не удовлетворяет потребностям приложения. Но также более простым и не менее эффективным способом является совмещение двух технологий и частичный перенес данных в поисковые системы для поддержки конкретных поисковых потребностей проекта. Для этого база данных должна быть проиндексирована, чтобы внутри неё можно было искать в системе полнотекстового поиска. Данные будут использоваться одновременно в обеих системах, чтобы охватить преимущества каждой из этих собственных систем.

2 АЛГОРИТМИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПРИ РАЗРАБОТКЕ ПРИЛОЖЕНИЯ ДЛЯ РЕАЛИЗАЦИИ ДИФФЕРЕНЦИРОВАННЫХ КАТАЛОГОВ ДАННЫХ

2.1 Выбор архитектуры при разработке программного обеспечения

Общий взгляд на архитектуру

Данное ПО является более простым, но в то же время бесплатным и открытым аналогом систем для создания и автоматизации создания дифференцированных каталогов данных, основанное на современных технологиям, отвечающим последний стандартам разработки. Ниже можно увидеть общую схему работы всех компонентов веб-приложения.

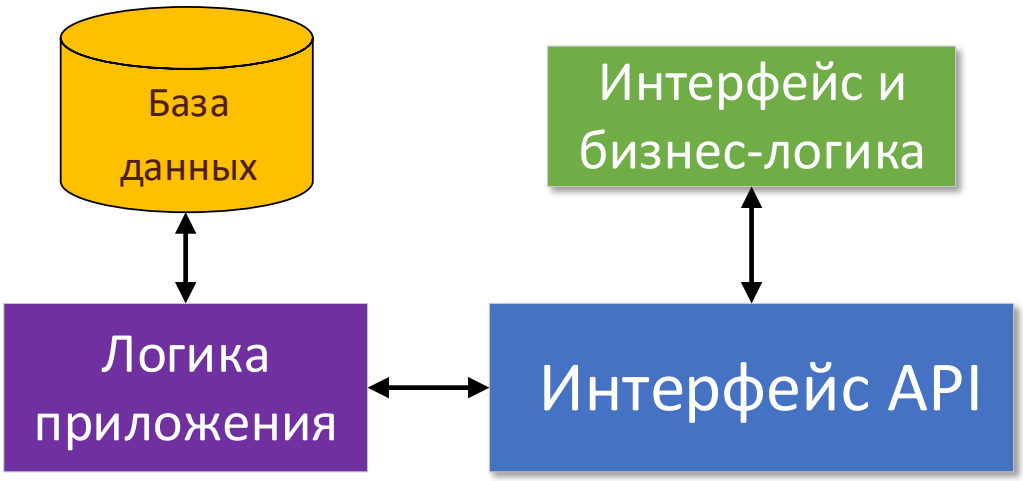

Рисунок 2.1 – Наиболее общая схема работы веб-приложении для реализации каталогов данных

Как видно из рисунка выше приложение состоит из 3 основных частей, а также взаимодействует с СУБД посредством серверной части.

Приложения делится на модули и использует в каждом модель MVC с разделением на нескольких слоёв, таких как: механизмы для работы с данными(model), контроллеры управления, интерпретирующие действия пользователя и оповещающие модель о необходимости изменений (Controller), и механизм вывода на клиентскую часть с использованием API(View).

Данный подход позволяет разграничить работу системы, что улучшить читаемость кода и безопасность кода. Также архитектура полностью отделяет серверную и клиентскую часть, предоставляя только API для их взаимодействия.

Такое проектирование позволит в дальнейшем гибко взаимодействовать с системой, создавая различные виды клиентов, как, например, мобильные приложения. В целом подобный подход позволяет с лёгкостью расширять и модернизировать проект, что является хорошим преимуществом.

Архитектурный паттерн серверный части

При разработке приложений одними из основных принципов являются простота, понятность и масштабируемость кода. Так, например, один из самых известных принципов проектирования KISS (keep it simple and straightforward) гласит, что простота кода – превыше всего, потому что простой код – наиболее понятный, из чего в дальнейшем складывается и скорость масштабирования и поддержки проекта. Разделение кода на меньшие части один из основных способов упростить читаемость и понимание кода. Это может проявляться в банальном вынесении частей в отдельные функции или же в соблюдении принципа единственной ответственности из акронима SOLID, который гласит, что для каждого класса должно быть определено единственное назначение, а все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.

При разработке относительно сложного программного обеспечения часто используется разделение на слабосвязанные модули, что дает возможность строить большие, комплексные приложения, которые в свою очередь будут просты в поддержке и разработке распределёнными командами. В большинстве случаев применяется объектно-ориентированное программирование с внедрением правил предметно-ориентированного проектирования (Domain-driven design), принципов DRY, GRASP, а также ранее упомянутых SOLID и KISS.

Для разработки в структуру фреймворка Laravel был внедрен архитектурный паттерн Porto – один из современных архитектурных паттернов программного обеспечения, состоящий из принципов и шаблонов, помогающих строить

код подобных многомодульных систем так, чтобы его можно было легко поддерживать и использовать повторно.

Porto дает возможность создать масштабируемое монолитное приложение, которые при необходимости можно легко разделить на несколько микросервисов для обеспечения возможности повторного использования бизнес-логики (функций приложения).

Паттерн наследует концепции архитектур DDD (Domain Driven Design), Modular, Micro Kernel, MVC (Model View Controller), Layered и ADR (Action Domain Responder).

Также Porto соответствует списку фундаментальных принципов проектирования SOLID, OOP, LIFT, DRY, CoC, GRASP, Generalization, High Cohesion и Low Coupling.

Porto состоит из слоев «корабль» и «контейнеры» (аналог модулей), абстрагируясь на 3 уровня кода, которые можно рассмотреть на рисунке 2.2.

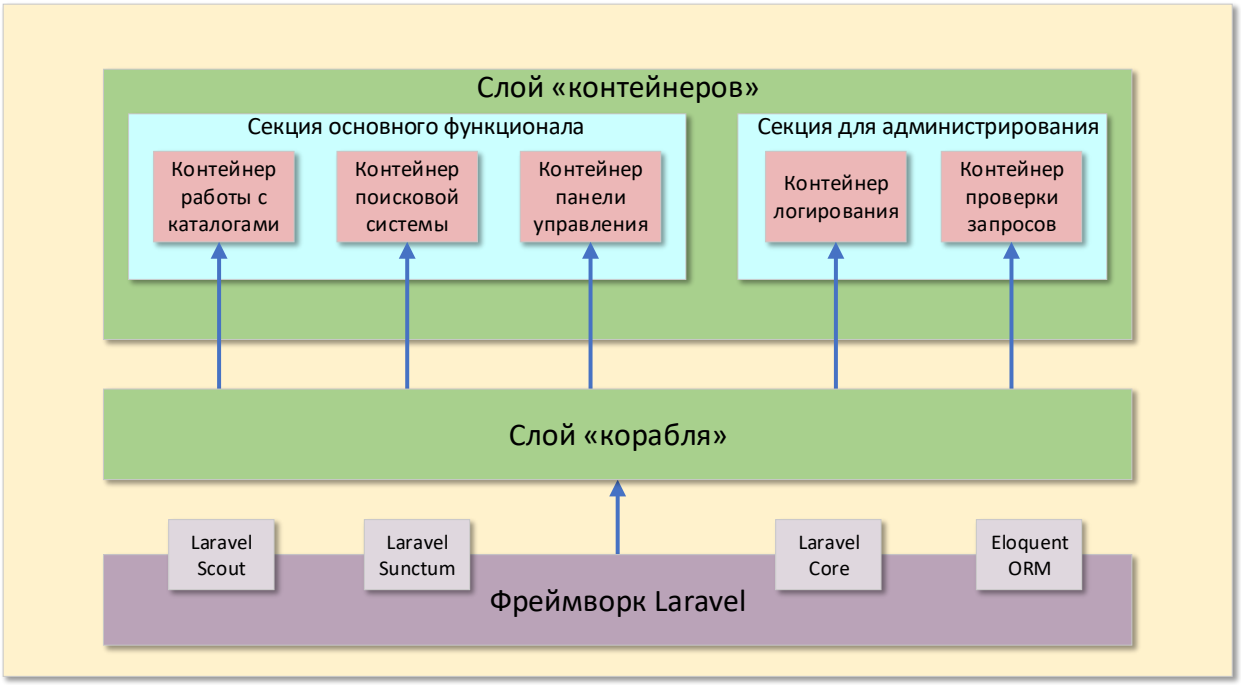

Рисунок 2.2 – Слои приложения

Низкоуровневый код

Код фреймворка (реализует базовые операции, такие как чтение файлов с диска или взаимодействие с базой данных).

Код среднего уровня

Общий код приложения (реализует функциональные возможности, обслуживающие код высокого уровня. И он полагается на код низкого уровня для работы). Располагается в слое «корабля».

Код высокого уровня

Код бизнес-логики (инкапсулирует сложную логику и для работы полагается на код среднего уровня). Располагается в слое «контейнеров».

Рассмотрим более подробнее каждый слой паттерна.

а) Слой «корабля»

Уровень «Корабль» содержит родительские классы (классы, расширяемые каждым отдельным компонентом) и некоторый служебный код. Родительские классы наследуют такие основные классы фреймворка как [5]:

1) Request, который предоставляет объектно-ориентированный способ взаимодействия с HTTP-запросом, а также для получения входных данных, файлов cookie и файлов, которые были отправлены с запросом.

2) Controller, который сгруппировывает связанную логику обработки похожих запросов в один класс, например, контроллер пользователя может обрабатывать все входящие запросы, относящиеся к пользователям.

3) Response, который обеспечивает различные методы для построения ответов HTTP

4) Model, который взаимодействует с базой данных, посредством ORM, Помимо получения записей из таблицы базы данных, модели Eloquent также позволяют вставлять, обновлять и удалять записи из таблицы.

5) Exception, который обрабатывает все исключения и позволяет регистрировать пользовательские.

Помимо описанных, существует множество других сущностей, от которых также абстрактно наследуются слой «корабля», тем самым создавая прослойку, позволяющую в дальнейшем легко обновлять сам фреймворк и дополнять сущности новыми возможностями (например, добавление функции к классу базовой модели делает его доступным в каждой модели контейнеров). Также на данном

слое находятся общие сущности, содержащие многоразовые функции и классы, которые могут использоваться каждым контейнером, как например транспортеры (Data Transfer Object), которые используются для передачи пользовательских данных (поступающих из запросов, команд) из одного места в другое (из действия в задачу, из контроллера в действия и др.) или трансформеры (Responses Transformers) классы, отвечающие за преобразование моделей в массивы.

Родительские классы слоя «Корабль» предоставляют полный контроль над компонентами контейнера (например, добавление функции к классу базовой модели делает его доступным в каждой модели контейнеров).

Уровень «Корабль» также играет важную роль в отделении кода приложения от кода используемого фреймворка, что облегчает обновление фреймворка, не затрагивая код приложения.

б) Слой контейнеров

Porto разрешает сложность приложения, разбивая его на более мелкие управляемые контейнеры. Контейнеры Porto похожи по своей природе на модули (из модульной архитектуры) и домены (из архитектуры DDD), представляющие собой части предметной области, к которой применяется разрабатываемое программное обеспечение. Каждый контейнер реализует модель MVC (рис 2.3), являющуюся основной парадигмой, на которой строится фреймворк Laravel.

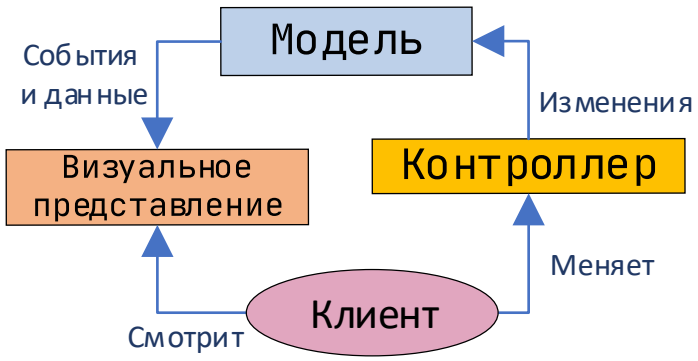

Рисунок 2.3 – Модель MVC

Контейнер делится на механизмы для работы с данными(model), контроллеры управления, интерпретирующие действия пользователя и оповещающие

модель о необходимости изменений (Controller), и механизм вывода на клиентскую часть с использованием API(View). Каждый из базисов, в свою очередь, также может разделятся на более мелкие компоненты.

Слой контейнеров – это место, где находится бизнес-логика приложения. Контейнеры могут зависеть от других контейнеров, а также от составных частей других контейнеров и общих классов. Также возможны другие формы связи, такие как прослушивание событий, запускаемых другими контейнерами.

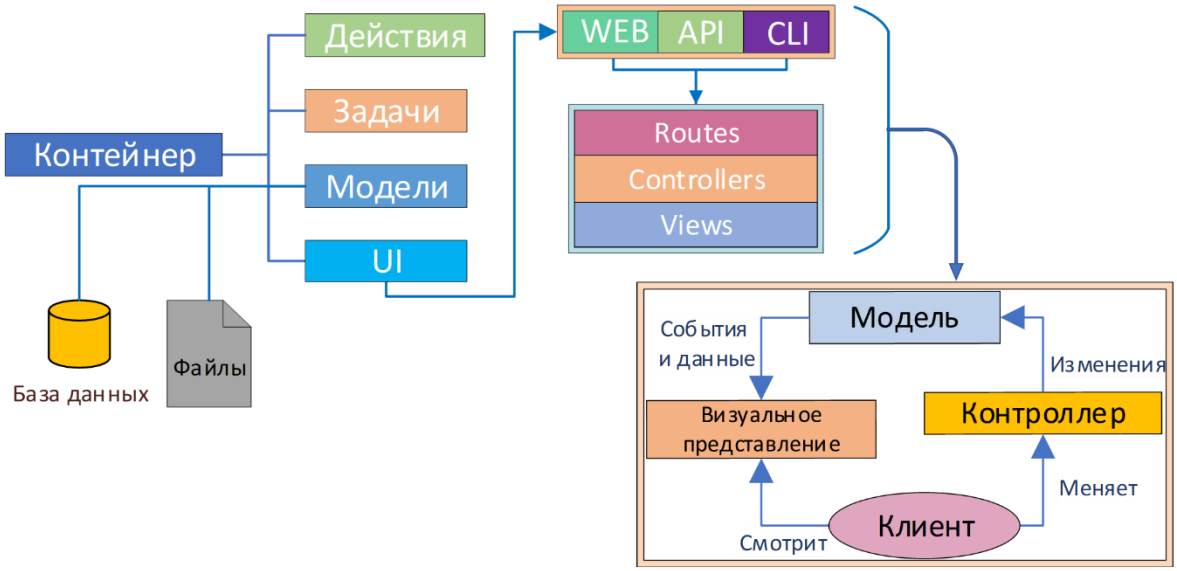

Рисунок 2.4 – Структура и логика работы контейнера

На уровне контейнера есть набор компонентов с предопределенными обязанностями (рисунок 2.4). Porto определяет список компонентов с набором рекомендаций, которым нужно следовать при их использовании, чтобы архитектура оставалась расширяемой и хорошо поддерживаемой. В целом компоненты обеспечивают согласованность и упрощают сопровождение кода, поскольку заранее известно, где находится каждый фрагмент кода.

На основе вышеописанного можно выделить основными преимущества использования Porto:

- модульность и возможность повторного использования;

- правила и рекомендации Porto минимизируют и определяют направления зависимостей между контейнерами, чтобы избежать циклических ссылок между

ними. Это позволяет группировать связанные контейнеры в разделы для того, чтобы повторно использовать их вместе в разных проектах;

- масштабируемость;

- паттерн структурирован таким образом, чтобы обеспечить разделение кода и согласованность. Он имеет организованную кодовую базу и четкий рабочий процесс разработки с предопределенными направлениями потока данных и зависимостей;

- возможность тестирования и отладки;

- соблюдение принципа единой ответственности за счет наличия единственной функции для каждого класса приводит к «тонким» классам, что упрощает тестируемость;

- адаптивность;

- возможность приспособить приложение к будущим изменениям с минимальными усилиями, так как пользовательский интерфейс отделен от бизнес-логики приложения и отделен друг от друга в каждом контейнере;

- расширяемость и гибкость;

- строение архитектуры принимает во внимание будущий рост и гарантирует, что код останется обслуживаемым независимо от размера проекта. Это достигается за счет модульной структуры, разделения задач и организованной связи между компоненты;

- простая обновляемость;

- легко выполнить обновления фреймворка благодаря полному разделению между приложением и кодом фреймворка на уровне корабля;

- основываясь на выделенных преимуществах, можно сделать вывод о том, что данный паттерн проектирования является отличным решением для создания модульной, хорошо масштабируемой и легко поддерживаемой архитектуры при разработке приложения для реализации каталогов данных.

Архитектурный паттерн клиентской части

Vue.js определен как ViewModel слой шаблона MVVM (рисунок 2.5). Он соединяет модель и представление в двустороннее связывание данных. Текущие

DOM-изменения и форматированный вывод абстрагируются в директивах и фильтрах [6]. Цель такого подхода в предоставлении преимущества быстрых связываний данных и сложных представлений компонентов с API как простые, так и понятные.

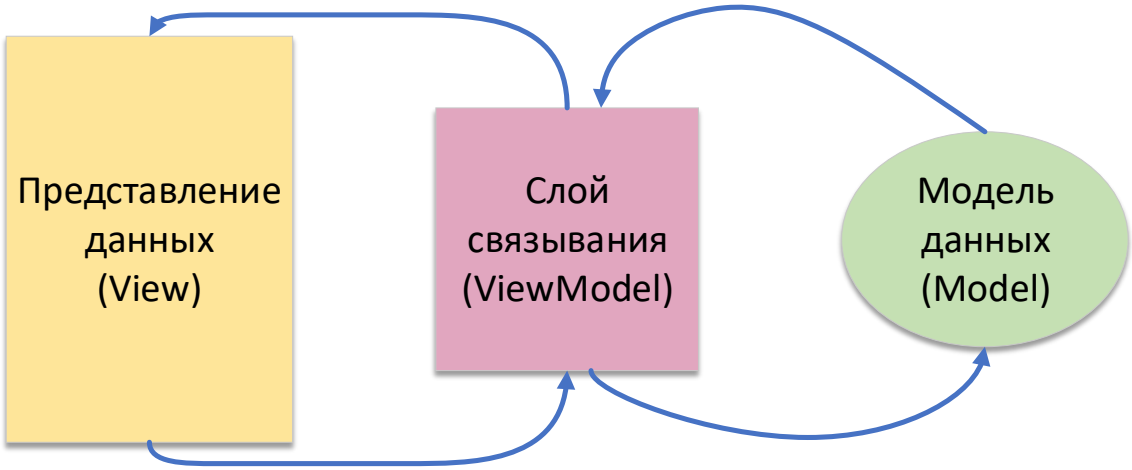

Рисунок 2.5 – Общий принцип работы клиентской части приложения

Каждый экземпляр Vue при создании проходит через последовательность шагов инициализации – настраивает наблюдение за данными, компилирует шаблон, монтирует экземпляр в DOM, обновляет DOM при изменении данных. DOM – это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, а также изменять содержимое, структуру и оформление таких документов. Проект базируется на vue-cli – инструменте для быстрого создания проектов под Vue.js. Такие проекты уже имеют некоторую готовую начальную структуру, установленную конфигурацию, в частности, конфигурацию для Webpack, а также ряд базовых файлов.

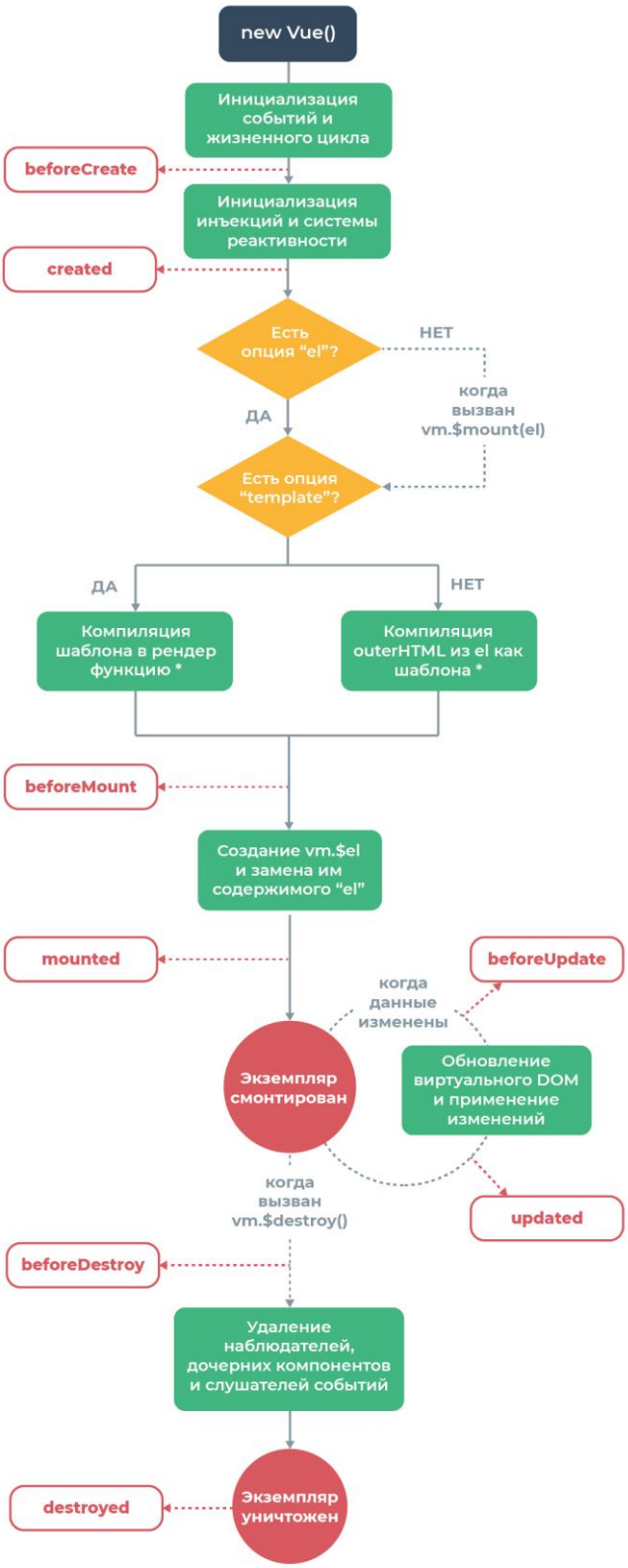

Рисунок 2.6 – Схема жизненного цикла экземпляра Vue

Выше можно увидеть блок-схему, описывающую процесс создания экземпляра Vue с описанием хуков жизненного цикла [7], который в дальнейшем управляет всеми клиентскими процессами в приложении.

2.2 Обоснование выбора средств разработки

2.2.1 Выбор серверного и клиентского фреймворков

Программное обеспечение для разработки должно было удовлетворять следующим основным параметрам:

- простота в изучении нового функционала. Чтобы разработчик мог за минимально потраченное количество времени освоить новый функционал и писать качественный код;

- простота в написании кода. Код должен быть интуитивно понятным и простым;

- скорость работы;

- скорость написание кода;

- безопасность. Среди наших клиентов много банков, а для них безопасность главнее всего;

- масштабируемость. Чтобы легко можно было писать проекты различных размеров;

- функциональность. Из коробки должны быть доступны решения многих типичных задач;

При рассмотрении серверных решений для разработки приложения для реализации каталогов данных был выбор между три основными конкурентами:

- PHP и фреймворк Laravel;

- Python и фреймворк Django;

- Ruby и фреймворк Ruby on Rails.

Сравним основные аспекты, на которых делался выбор технологии.

Сообщество

У всех решений есть отличное сообщество и официальные документации, но в целом изучать Django и Laravel намного проще, чем Ruby on Rails из-за особенностей языка и не сильной популярности в настоящее время.

Безопасность

Все сравниваемые фреймворки имеют высокий уровень безопасности. Все имеют middleware для обслуживания входящих запросов и их проверки, пока они

ещё не дошли до кода. Все три также предоставляют токены csrf для форм и защиту от различных атак.

Время, необходимое для создания приложения

Если существует идеальное понимание фреймворка, то создание приложения в Ruby on Rails является самым быстрым, поскольку оно спроектировано таким образом, чтобы разработчикам не приходилось тратить время на базовые вещи, и они могли сосредоточиться на основной части. приложения. В целом на Laravel нужно написать больше строк кода, чем на Django и Ruby on Rails, но в тоже время порог вхождения в Laravel ниже и предоставляется обширное количество сторонних пакетов, расширяющий функционал и добавляющий разнообразные скаффолдинг решения.

Плюсы и минусы

Общие вещи во всех фреймворках

- все - MVC (Django также называется MTV, но изменилось только название, концепция та же);

- все фреймворки ориентированы на удобочитаемость и простоту кода и распространения файлов;

- все делают используют ORM. Хотя есть возможность выполнять необработанные запросы в исключительных случаях;

- все автоматически создают таблицы в базе данных из файлов миграции.

- все фреймворки имеют простые и безопасные системы маршрутизации. Веб-страницы отображаются динамически;

- у всех есть свои собственные системы шаблонов, и каждая система шаблонов богата фильтрами и предопределенными функциями.

Django

Django имеет очень мощную библиотеку со следующими функциями:

- встроенная административная панель, декораторы и общие классы представлений;

- автоматически генерируемые формы для моделей;

- мощный механизм кэширования;

- поддержка классов промежуточного программного обеспечения, которые могут вмешиваться на различных этапах обработки запросов и выполнять настраиваемые функции;

- внутренняя система диспетчеризации;

- мультиязычность, которая включает в себя перевод собственных компонентов Django на множество языков;

- система сериализации, которая манипулирует JSON-представлениями экземпляров модели Django;

- интерфейс встроенной среды модульного тестирования Python;

- расширяемая система аутентификации;

- динамический административный интерфейс;

- инструменты для создания каналов синдикации RSS и Atom;

- режим многосайтовости, который позволяет запускать несколько веб-сайтов, каждый со своим собственным контентом и приложениями;

- инструменты для создания Google Sitemaps;

- встроенные средства защиты от подделки межсайтовых запросов, межсайтовых сценариев, SQL-инъекций, взлома паролей и других типичных веб-атак, большинство из которых включены по умолчанию.

Laravel

- в последние несколько лет Laravel развивался быстрее, чем Django и Ruby on Rails;

- Composer является мощным пакетным менеджером для PHP и предоставляет удобную систему упаковки и зависимостей;

- роуты - самый простой в управлении и абстрактный способ маршрутизации. В Laravel 7 добавлена поддержка API и каналов Websocket;

- ORM - еще одна система, позволяющая абстрагироваться и автоматизировать работу с моделями и взаимосвязь базы данных с нашим приложением;

- миграции - гораздо более элегантный способ версионирования базы данных;

- предоставление очередей для удобного отложенного запуска действий;

- внутренняя поддержка Redis;

- внедрение зависимостей - простое тестирование и автоматизация загрузки зависимостей.

Rails

- Ruby on Rails включает инструменты, которые «из коробки» упрощают общие задачи разработки, такие как скаффординг, которые могут автоматически создавать некоторые модели и представления, необходимые для базового вебсайта. Также включены WEBrick, простой веб-сервер Ruby, который распространяется вместе с Ruby, и Rake, система сборки, распространяемая как gem. Вместе с Ruby on Rails эти инструменты обеспечивают базовую среду разработки;

- Active Record играет важную роль. Это модель, являющаяся прослойкой ответственной за представление бизнес-логики и данных. Active Record упрощает создание и применение объектов, данные которых требуют хранения в базе данных;

- автоматическая маршрутизация. Автоматически определяются некоторые общие функции таблиц базы данных, такие как создание, редактирование и отображение. Это означает, что не нужно тратить время на простую работу, и можно сосредоточится на бизнес логике;

- командная строка. С помощью командной строки можно выполнять множество операций, например использовать Rake. Rake – автономная утилита Ruby, которая заменяет утилиту make Unix и использует Rakefile и. rake файлы для создания списка задач;

- модуль ActiveModelHelper содержит вспомогательные методы для быстрого создания форм из объектов, которые следуют соглашениям Active Model, начиная с моделей Active Record.

В целом все представленные решения удовлетворяют необходимым требованиям в той или иной степени и являются хорошими фреймворками, но Python в большинстве тестов оказался медленнее последней версии PHP и JIT компилятором, которая имеет самый быстрый интерпретатор в мире на сегодняшний момент [8], а функциональность Ruby on Rails и количество доступных решений

оказалось на порядок ниже, чем у Laravel. Также Laravel предоставляет решения по созданию модульной API системы, которые не удалось найти в других фреймворках.

В результате рассмотрения вышеописанных серверных языков, фреймворков и архитектурных паттернов, которые они реализуют, при разработке приложения для создания дифференцированных каталогов был выбран серверный веб-фреймворк Laravel, который будет подробнее рассмотрен в следующей главе.

Для реализации же клиентской части приложения нет каких-то особых предпочтений, так как клиентская часть взаимодействует с API и может быть написана на чем угодно, что удобнее разработчикам.

Для создания стандартного решения был выбран JavaScript-фреймворка - Vue.js, который по умолчанию интегрирован с Laravel и довольно просто в изучении. Инструмент является довольно молодым и включает в себя сильные стороны самых популярных долгоиграющих клиентских фреймворков React и Angular, которые тоже рассматривались как конкуренты для разработки стандартной клиентской части, но из-за сложностей в изучении и интеграции выбор сделан в пользу вышесказанного Vue.js.

2.2.2 Выбор типа базы данных и СУБД

При выборе базы данных для приложения были рассмотрены два основных варианта:

- SQL;
- NoSql.

Реляционные базы данных используют структурированный язык запросов (Structured Query Language, SQL) для определения и обработки данных. С одной стороны, это открывает большие возможности для разработки [9]: SQL один из наиболее гибких и распространённых языков запросов, так что его выбор позволяет минимизировать ряд рисков, и будет особенно кстати, если предстоит работа с комплексными запросами. С другой стороны, в SQL есть ряд ограничений. Построение запросов на этом языке обязывает предопределять структуру данных и, что при определённых запросах может быть губительным для всей системы.

Нереляционные базы данных, в свою очередь, предлагают динамическую структуру данных, которые могут храниться несколькими способами: ориентированно по колонкам, документо-ориентированно, в виде графов или на основе пар «ключ-значение». Такая гибкость означает следующее:

- вы можете создавать документы, не задавая их структуру заранее;
- каждый документ может обладать собственной структурой;
- у каждой базы данных может быть собственный синтаксис;
- вы можете добавлять поля прямо во время работы с данными.

Масштабируемость

В большинстве случаев SQL базы данных вертикально масштабируемые, то есть вы можете увеличивать нагрузку на отдельно взятый сервер, наращивая мощность центральных процессоров, объёмы ОЗУ или системы хранения данных. А NoSQL базы данных горизонтально масштабируемы. Это означает, что вы можете увеличивать трафик, распределяя его или добавляя больше серверов к вашей СУБД. Всё равно, что добавлять больше этажей к вашему зданию, либо добавлять больше зданий на улицу. Во втором случае, система может стать куда больше и мощнее, делая выбор NoSQL базы данных предпочитаемым для больших или постоянно меняющихся структур данных.

Структура

В реляционных СУБД данные представлены в виде таблиц, в то время как в нереляционных – в виде документов, пар «ключ-значение», графов или wide-column хранилищ. Это делает SQL базы данных лучшим выбором для приложений, которые предполагают транзакции с несколькими записями – как, например, система учётных записей – или для устаревших систем, которые были построены для реляционных структур.

В число СУБД для SQL баз данных входят MySQL, Oracle, PostgreSQL и Microsoft SQL Server. Для работы с NoSQL подойдут MongoDB, BigTable, Redis, RavenDB Cassandra, HBase, Neo4j и CouchDB.

В качестве выбора для приложения рассматривались MySQL и MongoDB. Рассмотрим основные преимущества двух баз данных.

MySQL

- проверено временем. MySQL – крайне развитая СУБД, что означает наличие большого сообщества вокруг неё, множество примеров и высокую надёжность;

- совместимость. MySQL доступна на всех основных платформах, включая Linux, Windows, Mac, BSD и Solaris. Также у неё есть библиотеки для языков вроде Node.js, Ruby, C#, C++, Java, Perl, Python и PHP;

- окупаемость. Это СУБД с открытым исходным кодом, находящаяся в свободном доступе;

- реплицируемость. Базу данных MySQL можно распределять между несколькими узлами, таким образом уменьшая нагрузку и улучшая масштабируемость и доступность приложения;

- шардинг. В то время как шардинг невозможен на большинстве SQL баз данных, MySQL является исключением.

MongoDB

- динамическая схема. Как упоминалось выше, эта СУБД позволяет гибко работать со схемой данных без необходимости изменять сами данные [10];

- масштабируемость. MongoDB горизонтально масштабируема, что позволяет легко уменьшить нагрузку на сервера при больших объёмах данных;

- удобство в управлении. СУБД не нуждается в отдельном администраторе базы данных. Благодаря достаточному удобству в использовании, ей легко могут пользоваться как разработчики, так и системные администраторы;

- скорость. Высокая производительность при выполнении простых запросов;

- гибкость. В MongoDB можно без вреда для существующих данных, их структуры и производительности СУБД добавлять поля или колонки.

В итоге выбор был сделан в пользу MySQL 8.0 и выше, которая имеет тип поля JSON, что позволяет сохранять JSON объекты в базе данных [11] и полностью удовлетворяет потребностям приложения, так как является самой

доступной и простой базой данных, которую можно будет использовать на любом хостинге.

2.2.3 Выбор системы полнотекстового поиска

Были рассмотрены три системы полнотекстового поиска для внедрения в приложение:

- Elasticsearch;
- Algolia;
- MeiliSearch.

В качестве основного инструмента для поисковой системы был выбран MeiliSearch, так как предоставляет готовый драйвер для пакета Laravel Scout [12], который позволяет легко внедрить поиск в модели Laravel, а также является легковесным и достаточно быстрым, чтобы у пользователей был позитивный опыт при работе с интерфейсом приложения. Сравним MeiliSearch с другими представленными альтернативами для выявления особенностей каждого из них.

MeiliSearch и Elasticsearch

Elasticsearch был разработан как серверная поисковая система и обычно используется для создания поисковых панелей для конечных пользователей [13].

В отличие от Elasticsearch, который является общей поисковой системой, MeiliSearch фокусируется на предоставлении определенных функций.

Elasticsearch может выполнять поиск по огромным объемам данных и выполнять анализ текста. Чтобы сделать его эффективным для поиска в приложении надо больше времени на его изучение, чтобы иметь возможность настраивать и адаптировать его в соответствии с вашими потребностями.

MeiliSearch предназначен для предоставления конечным пользователям эффективного мгновенного поиска.

Также Elasticsearch иногда может работать слишком медленно, если необходимо обеспечить мгновенный поиск. В большинстве случаев он значительно медленнее возвращает результаты поиска по сравнению с MeiliSearch.

MeiliSearch и Algolia

MeiliSearch был вдохновлен продуктом Algolia и лежащими в его основе алгоритмами. Он предоставляет аналогичные функции и достигает того же уровня актуальности так же быстро как Algolia. Но в отличие от Algolia, MeiliSearch имеет открытый исходный код и написан на Rust, современном языке программирования, который обеспечивает быструю работу. Rust также обеспечивает переносимость и гибкость, что делает легким его развертывание на любом устройстве.

Подводя итог, можно выделить, что MeiliSearch - идеальный выбор для разрабатываемого приложения, так как предоставляет простой и легкий инструмент для поиска, устойчивой к опечаткам, имеющий возможность поиска по префиксу и являющийся интуитивно понятным для пользователей и мгновенно возвращающим результаты с почти идеальной релевантностью.

2.3 Характеристика выбранного программно-технического обеспечения

2.3.1 Характеристика выбранного технического обеспечения

Аппаратное обеспечение

Система ориентирована на любое клиентское аппаратное обеспечение способное приемлемо работать с браузером Internet Explorer 11 и новее. Для серверного аппаратного обеспечения рекомендуется использовать:

- двухъядерный процессор с архитектурой x86;
- один гигабайт ОЗУ (в топ числе для нормальной работы composer);
- предпочтительно твердотельные накопители и размер выделенного пространства от 20 гигайбайт, а также обычные жесткие диски для создания резервных копий;
- рекомендуемая мощность источника питания от 400 Вт;
- хорошо проветриваемый корпус и кулер охлаждение для процессора с рассеиваемой мощностью не ниже, чем заявленное TDP процессора.

Выбранные аппаратные характеристики необходимы для полноценной работы приложения со средней загрузкой.

Среда разработки и эксплуатации

Приложение строится на основе серверного PHP-фреймворка Laravel и клиентского JavaScript-фреймворка – Vue.js.

Для работы данной связки технологий необходимо следующее:

- любая операционная система, в приоритете Linux;
- PHP> = 7.3;
- BCMath PHP Extension;
- Ctype PHP Extension;
- Fileinfo PHP Extension;
- JSON PHP Extension;
- Mbstring PHP Extension;
- OpenSSL PHP Extension;
- PDO PHP Extension;
- Tokenizer PHP Extension;
- XML PHP Extension;
- Node.js> = 12.15;
- npm> = 5.0;

Всем этим требованиям удовлетворяет Laravel Homestead - официальный предварительно упакованный пакет программного обеспечения для создания и конфигурирования виртуальной среды разработки Vagrant, который предоставляет удобную среду разработки, не требующую установки PHP, веб-сервера и любого другого серверного программного обеспечения на аппаратном обеспечении для разработки. Также данная среда содержит все необходимые вспомогательные средства для разработки, такие как:

- Git – система контроля версий. Все версии проекта будут также удаленно хранится на веб-сервисе для хостинга IT-проектов GitHub;

- Composer – пакетный менеджер уровня приложений для языка программирования PHP, который предоставляет средства по управлению зависимостями в PHP-приложении. Соответственно все зависимости фреймворка и проекта

удобно регулируются в одном файле, что позволяет избежать путаницы и ошибок;

- различные СУБД для создания и управления базами данных, что также является неотъемлемой частью проекта, так как предполагается хранение большого количества различной информации;

- таким образом выбранное решение для среды разработки является лучшим и самым актуальным вариантов на данный момент.

После завершения разработки планируется разворачивание приложения на рабочем окружении(деплой), в котором производится работа с реальными пользователями и актуальными данными. Для этого будет использоваться Ubuntu 20.04 Server вместе с системой контейнеризации приложения Docker – процессом адаптации приложения и его компонентов для работы в облегченных средах, называемых контейнерами [14]. Это изолируемые одноразовые среды, которые используются для разработки, тестирования и развертывания приложений в производственной среде. Структуру работы Docker можно увидеть на рисунке 2.7.

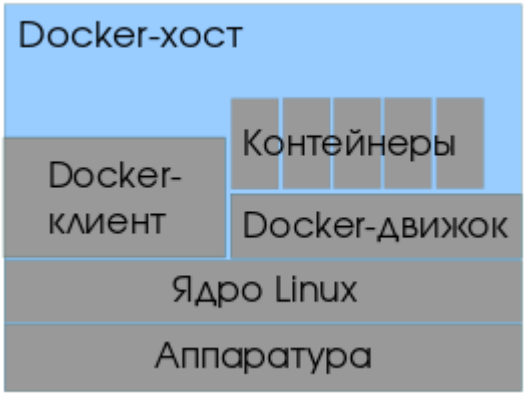

Рисунок 2.7 – Структура работы Docker

Для большой надежности приложение и база данных будут хранится в отдельных контейнерах, что обеспечит бесперебойность работы и защиту данных, так как если что-то случится с приложением, всё что понадобиться это перезагрузить контейнер. Для управления несколькими контейнерами приложения будет использоваться Docker Compose, необходимый для одновременного управления несколькими контейнерами, входящими в состав приложения. В конечном

итоге на development сервере будет работать приложение и база данных, каждый в отдельном защищенном контейнере, что позволит легко управлять приложением, его версиями и данными.

2.3.2 Характеристика выбранного программного обеспечения

Рассмотрим выбранный программный комплекс.

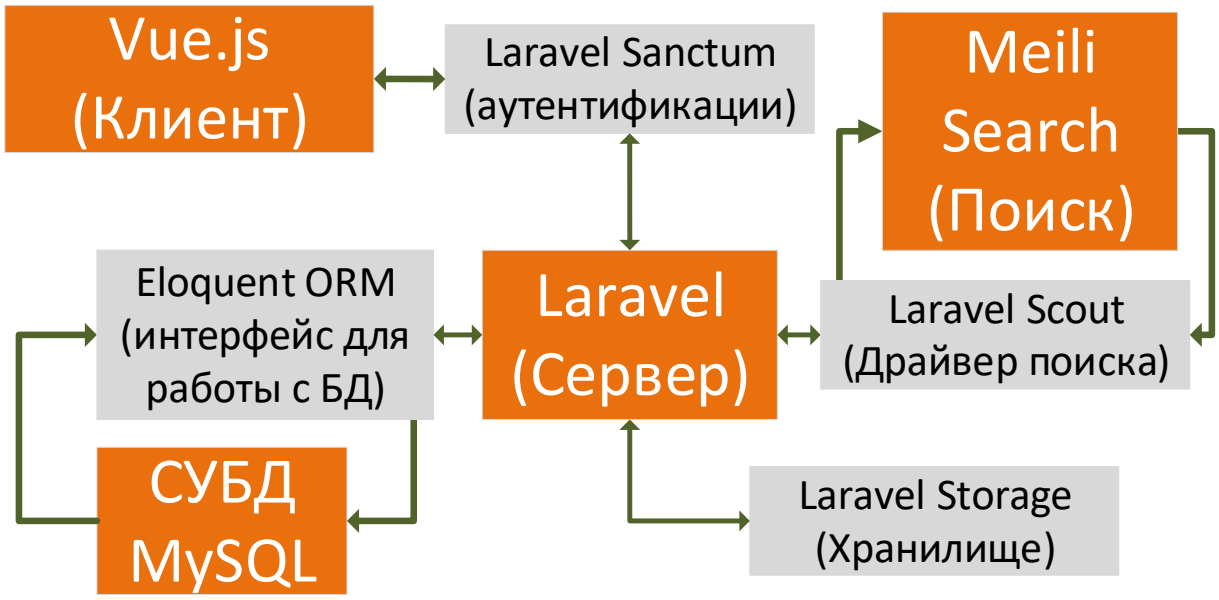

Рисунок 2.8 – Диаграмма компонентов ПО

На диаграмме выше оранжевым цветом отмечены основные компоненты для разработки приложения, а серым – дополнения к фреймворку, которые упрощают работу и предоставляют дополнительные возможности. Laravel взаимодействует с поисковым движком MeiliSearch, клиентом написанным на Vue.js, базой данных и своим внутренним хранилищем, образуя глобальный интерфейс работы программного обеспечения.

Laravel – бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC. Laravel выпущен под лицензией MIT. Фреймворк предоставляет все современные возможности по разработки приложений на PHP, масштабированию и поддержке:

- MVC-архитектура;

- широкий и разнообразный функционал, включающий в себя шаблонизатор blade, кэширование, OAuth авторизация, тестирование, мультиязычность и другое;

- возможность создавать масштабные интернет-проекты, независимо от сложности и направленности, в том числе и многоуровневые веб-сайты;

- простая и понятная система пакетов, что позволяет находить почти любые необходимые компоненты для разрабатываемой системы и освобождает от их написания с нуля;

- надежная защита базы данных от SQL, CSRF, XSS;

- все изменения в PHP и направлений развития веб-сайтов обязательно учитываются в обновлениях исходного кода фреймворка и во всех последующих версиях;

- возможность масштабирования проекта;

- открытый код и большое комьюнити;

- повышенная производительность и кеширование;

- миграции баз данных;

- объектно-ориентированные библиотеки;

- встроенные PHPUnit-тесты из коробки;

- мультиязычность;

- работа с ошибками и исключениями;

- система отложенных задач.

В целом фреймворк отвечает всем показателям качества, может работать на любой операционной систему, поддерживающий необходимые требования, а также обеспечивает надежность и эффективность работы приложения за счет встроенных систем оптимизации и безопасности.

Vue.js, как базовое решение для клиентской части, определен как ViewModel слой шаблона MVVM. Он создает двустороннее связывание между моделью и представлением. Изменения дерева DOM и форматированный вывод абстрагируются в директивах и фильтрах (рисунок 2.9). Цель такого

подхода в предоставлении преимущества быстрых связываний данных и сложных представлений компонентов с API как простые, так и понятные.

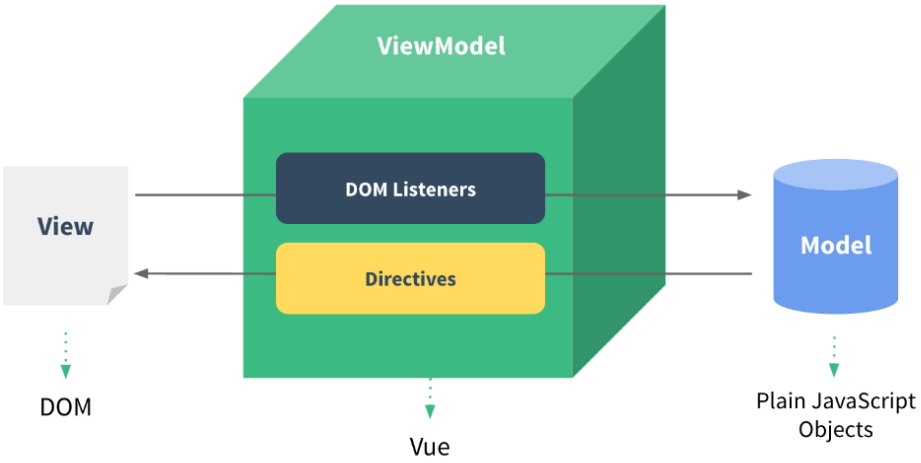

Рисунок 2.9 – Визуализация логики алгоритма работы клиентской архитектуры приложения основанного на Vue.js

Каждый экземпляр Vue инициализируется по шагам – настраивает наблюдение за данными, компилирует шаблон, монтирует экземпляр в DOM, обновляет DOM при изменении данных. DOM – это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, а также изменять содержимое, структуру и оформление таких документов.

Рассмотрим основные преимущества фреймворка:

- может работать как самостоятельный фреймворк, так и как библиотека для сторонних проектов;

- обеспечивает хорошее быстродействие и малый объём файлов;

- обладает двусторонней реактивностью, что позволяет удобно манипулировать данными;

- обеспечивает работу с помощью однофайловых компонентов, что достаточно удобно для разветвлённой архитектуры;

- является хорошо расширяемым с помощью отдельных плагинов;

- обладает широкой документацией на разных языках, в том числе и русском;

- может быть без проблем внедрён в разрабатываемый проект;

- достаточно низкий порог вхождения.

В качестве основного хранилища информации была выбрана база данных MySQL. Опишем её основные характеристики:

- полностью многопоточное использование ядерных нитей. Это означает, что пакет может легко использовать много CPUs, если они есть;

- интерфейсы для языков C, C++, Eiffel, Java, Perl, PHP, Python и Tcl;

- работает на многих различных платформах;

- много типов столбцов: целые со знаком или без него, FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATE TIME, TIMESTAMP, YEAR, SET и ENUM [15];

- очень быстрые объединения, использующие оптимизированное однопроходное объединение многих таблиц;

- полная поддержка операторов и функций в частях запроса SELECT и WHERE;

- SQL-функции выполнены через хорошо оптимизированную библиотеку классов и должны выполняться с такой скоростью, с какой только возможно! Обычно не имеется никакого распределения памяти вообще после инициализации запроса;

- полная поддержка предложений SQL GROUP BY и ORDER BY. Поддержка групповых функций (COUNT (), COUNT (DISTINCT ...), AVG (), STD (), SUM (), MAX () и MIN ());

- поддержка LEFT OUTER JOIN и RIGHT OUTER JOIN с синтаксисами ANSI SQL и ODBC;

- привилегии и система паролей, которая является очень гибкой и безопасной, и позволяет проверку, основанную на имени хоста. Пароли безопасны потому, что вся передача пароля шифрована, когда Вы соединяетесь с сервером;

- очень быстрые дисковые таблицы B-tree с индексным сжатием;

- можно иметь до 32 индексов на таблицу. Каждый индекс может состоять от 1 до 16 столбцов или частей столбцов;

- записи фиксированной и переменной длины;

- таблицы в памяти, которые используются как временные таблицы;

- поддержка поистине огромных объемов данных. Известен случай использования MySQL на 60000 таблиц, хранящих около 5000000000 строк;

- все столбцы имеют значения по умолчанию. Вы можете использовать вызов INSERT, чтобы вставить подмножество столбцов таблицы. Те столбцы, которым явно не заданы значения, будут автоматически установлены к их значениям по умолчанию;

- очень быстрая поточно-безопасная система управления памятью;

- сервер умеет выдавать сообщения об ошибках и диагностику на разных языках.

Также выбранная СУБД имеет широкую интеграцию с моделями Laravel и обеспечивает простой интерфейс запросов, что также послужило при её выборе.

Характеризуя выбранную поисковую систему, хочется отметить очень быструю работу и хорошо описанную структуру, реализованную с помощью эффективных алгоритмов.

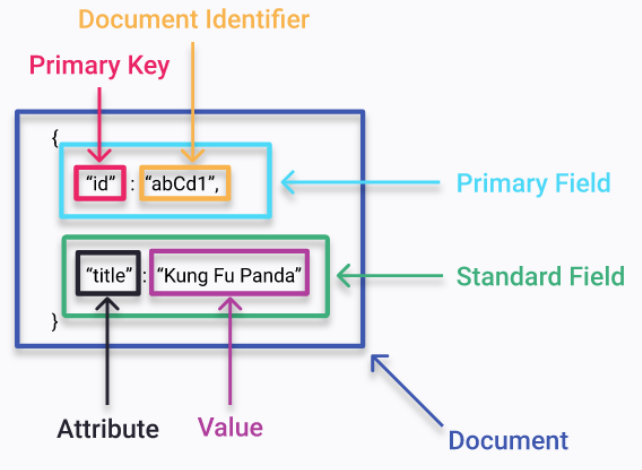

Рисунок 2.10 – Структура документа MeiliSearch

Документ, рассмотренный на рисунке 2.10, является объектом, состоящим из одного или нескольких полей. Каждое поле состоит из атрибута и связанного с ним значения. Документы функционируют как контейнеры для организации

данных и являются основными блоками базы данных MeiliSearch [16]. Для поиска документа их необходимо сначала добавить в индекс. Образец документа может выглядеть так:

```
{
  "id": "1564seq12ss",
  "title": "Кунг-фу Панда",
  "genre": "Анимационный фильм",
  "release-year": 2008,
  "cast": [{"Jack Black": "Po"}, {"Jackie Chan": "Monkey"}]
}
```

Рисунок 2.11 – Образец документа MeiliSearch

Документы представлены в виде JSON объектов типа пар ключ-значение. Документы имеют не более 1000 полей, так как ранжирование может больше не быть эффективным из-за этого, что приведет к неопределенному поведению. Кроме того, каждый документ должен иметь как минимум одно поле, содержащее первичный ключ и уникальный идентификатор.

3 РАЗРАБОТКА И ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ РЕАЛИЗАЦИИ ДИФФЕРЕНЦИРОВАННЫХ КАТАЛОГОВ ДАННЫХ

3.1 Этапы разработки программного обеспечения

3.1.1 Описание требований

Функциональные требования

Для достижения цели работы веб-приложение должно реализовывать следующий основной функционал:

- возможность создания дифференцированных каталогов данных с использованием полнотекстового поиска;

- возможность скачивания и комментирования и удаленного получения данных;

- предоставление внешнего API для создания различных клиентов, например, мобильное приложение;

- возможность использования защищённой внешней авторизации для пользователей приложения;

- возможность упрощенного создания различных клиентских приложений и шаблонов.

Описание работы интерфейса и требования к его взаимодействию с пользователем

Не менее важное значение чем функционал при работе с приложением имеет качество интерфейса пользователя, в том числе его графический дизайн и юзабилити – наличие комплекса свойств интерфейса, обеспечивающих комфортную работу пользователя.

В качестве требований к графическому дизайну веб-приложения, предъявляемых на стадии его разработки отнесены:

- привлекательность дизайн-решения для пользователей, прозрачная навигация и целевая ориентация в приложении;

- быстрота обучения при работе с веб-приложением;

- уникальность и запоминаемость;

- гибкость дизайн-решения.

Привлекательность дизайн-решения наряду с информационной ценностью заставит пользователя задержаться на странице и внимательнее отнестись к увиденному. Правильно спроектированный дизайн облегчает восприятие информационного материала.

Уникальность и запоминаемость графического интерфейса пользователя позволяет идентифицировать ресурс как знакомый и формирует к нему доверительное отношение. Кроме того, интерфейс должен удовлетворять определенным эргономическим требованиям, т.е. обеспечивать комфортность действий и ускорить адаптацию пользователя к информационной среде. Информационный материал должен быть подан таким образом, чтобы он был достаточно удобен для восприятия.

Под гибкостью дизайн-решения в работе понимается обеспечение быстрой и качественной его адаптации к возможным изменениям на уровне выполняемых функций и платформы.

В качестве наиболее важных вопросов, которые необходимо учитывать при проектировании и разработке интерфейса можно отнести следующие:

- цветовое решение;
- шрифты;
- графика;
- представление текстовой информации;
- дизайн страницы как единого объекта представления информации.

Также обеспечение юзабилити приложения должно достигаться посредством рационального использования:

- управляющих элементов;
- навигации;
- изложения информационного материала с точки зрения юзабилити;
- наличии функциональных элементов сайта для обеспечения требований по юзабилити.

Интерфейс в приложении строится с использованием JavaScript фреймворка – Vue.js, который обеспечивает качество, быстроту выполнения всех вышеперечисленных требований работы интерфейса. В дополнении, за счет того, что фреймворк мало весит экономится время загрузки страниц, что улучшает опыт использования UI и UX, идет плюс к поисковому ранжированию и конверсии. Пользователю проще ориентироваться в интерфейсе, который моментально реагирует на его действия. Данное техническое решение позволяет избавиться от некоторых рутинных вещей и сосредоточиться на проектировании и создании интерфейса.

3.1.2 Выбор модели жизненного цикла

В качестве модели жизненного цикла была выбрала спиральная стратегия, которая подразумевает разработку в виде последовательности версий с нечетко определёнными требованиями, уточняющимися в течении всей разработки.

Спиральная модель обычно применяется при разработке инновационных креативных проектов. В начале работы над проектом у заказчика и разработчика нет полного понимая готового решения и полной уверенности в успешности разрабатываемого решения, из-за чего требования не могут быть полностью определены. На основании этого разработка приложения происходит по частям, с возможностью корректировки требований или отказа от дальнейшей разработки. Разработка проекта может быть остановлена на любой стадии.

Данная модель отлично вписывается в процесс разработки приложения для реализации дифференцированных каталогов данных, так как приложение не имеет чёткий границ в объёме функциональности, которая может быть внедрена во время разработки.

При выборе спиральной модели жизненного цикла чаще всего используется методология быстрой разработки приложений RAD (Rapid Application Development) и экстремальное программирование.

RAD разработка обычно содержит в себе три основных элемента:

- команду разработчиков до 10 человек;
- хорошо описанный график работ, обычно длящийся до 6 месяцев;

- циклическую разработку, в которой функционал добавляется по мере взаимодействия с заказчиком. Также методология RAD предполагает широкое использование различных сторонних вендорный решений для ускорения и упрощения разработки.

Методология экстремального программирования ориентирована на разработку небольших приложений с несформированными и часто изменяющимися требованиями.

Главными чертами XP методологии являются:

- частый выпуск мажорных и минорных версий, когда в RAD модели это занимает обычно минимум 2 месяца;

- Общение с заказчиком происходит на протяжении всего цикла разработки;

- упор на максимально простое архитектурное решение и минималистический графический интерфейс, в угоду времени и скорости разработки. Чем проще интерфейс, тем более качественный опыт получит пользователь при работе с приложением;

- применение одинакового стиля кода, а также ведение общедоступной истории развития системы в системе контроля версий, как, например, Git;

- непрерывное и пересекающееся проектирование, разработка, интеграция и тестирование системы.

Представленные методологии разработки в первую очередь используются для максимального уменьшения сроков и цены создаваемых приложений, а также для улучшения их качества за счет постоянной обратной связи с заказчиком и учетом мнений.

В результате рассмотрения двух методологий, предпочтение было отдано XP методологии, так как именно она предполагает частую смену версий и модификаций, простое проектирование и минималистичный дизайн и также освобождает от строго графика выполнения работ.

3.1.3 Реализация архитектуры

Как говорилось ранее в пункте 2.1 архитектура делится на слой корабля и слой контейнеров.

При реализации слоя корабля использовано абстрагирование и все основные компоненты фреймворка расширяются сущностями слоя корабля.

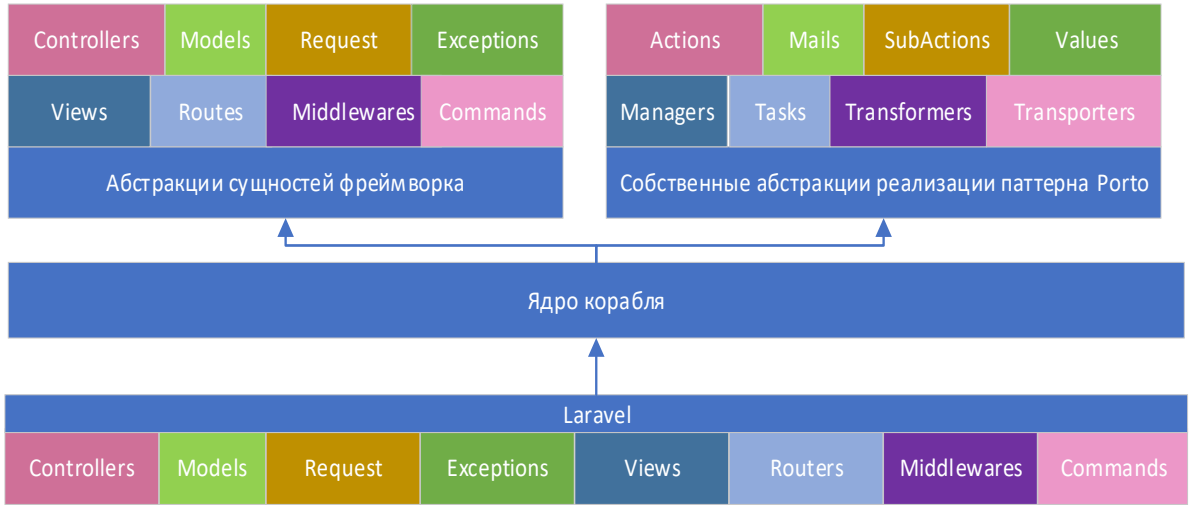

Рисунок 3.1 – Реализация слоя «корабля»

Как видно из рисунка выше слой корабля наследует основные компоненты фреймворка, что делает его независимым от версий Laravel, и предоставляет их слою контейнеров с доработками, как например, функции сериализации объектов, улучшенный механизм входных запросов и другие. Также в ядре «корабля» созданы свои абстракции, которые предоставляют бизнес логики дополнительный удобный функционал, необходимый для эффективной модульной разработки.

Рассмотрим подробнее реализацию основных компонентов, предоставляемых слоем «корабля», которые являются основополагающими при разработке приложения.

Routes (маршруты)

Сущность маршруты является первым звеном в цепочке получения HTTP-запросов. Также маршруты отвечают за перенаправление всех входящих HTTP-запросов на методы соответствующего контроллера.

Файлы маршрутов содержат конечные точки, которые представляют собой шаблоны URL-адресов, которые однозначно идентифицируют входящие HTTP-запросы, дополняют их проверками типа Middleware, если это необходимо.

Правила реализации маршрутов:

- в каждом контейнере может существовать три типа маршрутов: маршруты API, веб-маршруты и маршруты CLI. Каждый тип маршрутов содержится в своей папке и отделен от другого;

- папка веб-маршрутов содержит только веб-точки доступные из браузера; папка маршрутов API содержит только конечные точки API, которые доступны через вызовы из любых клиентов, подключенных к приложению; CLI маршруты же доступны также в браузере и через API и вызывают консольные команды Artisan;

- каждый контейнер обязательно должен содержать хотя бы один маршрута. В свою очередь каждый файл маршрута должен содержать одну конечную точку, что однозначно закрепляет за файлом маршрут, предоставляя удобства при разработке.

Controllers (Управляющие, C в MVC)

Контроллеры наследуются от класса LaravelBaseController и отвечают за обработку запроса, передачи данных запроса в Actions (Действия) и построение ответа. Соответственно проверка и ответ должны происходить в отдельных классах, но запускаться контроллером, что инкапсулирует всю реализацию, делая контроллер «тонким» звеном обслуживания. Концепция контроллеров взята из стандартной архитектуры фреймворка, но с ограниченными и предопределенными обязанностями.

Правила реализации контроллеров:

Контроллерам не должны знать о бизнес-логики, соответственно все детали реализации необходимо содержать в отдельных классах. Контроллер выполняет только следующие работы:

- чтение данных запроса;

- вызов действия;

- создание ответа.

Данные запроса должны передаваться исключительно Actions (Действиям), которые будут описаны ниже. Уровень контроллера помогает сделать действие повторно используемым в разных интерфейсах (веб-интерфейс и API), поскольку он не создает ответа (response), и это уменьшает количество дублирования кода.

Requests (Запросы)

Запросы наследуются от класса LaravelRequest и служат для передачи данных в приложение. Запросы - лучшее место для применения различных проверок, поскольку правила будут происходить в каждым запросе. Запросы также могут проверять авторизацию, например есть ли у пользователя доступ к какому-либо маршруту.

Правила реализации запросов:

Запросы приходят только в контроллеры и там остаются, а в котроллер в действие передает копию данных запроса. Они автоматически проверяют, соответствуют ли данные запроса правилам проверки, и если нет, то будет отправлен соответствующий ответ.

Actions (Действия)

Действия наследуются от класса Action, реализованным в ядре «корабля» и представляют собой варианты различной бизнес-логики, которая может выполниться по запросу с определёнными данными разделяясь на поддействия (subactions) и задачи (tasks).

Действия принимают структуры данных в качестве входных данных, производят какие-либо манипуляции с ними ими в соответствии с бизнес-логикой и возвращают в контроллер ответ. Также действия не занимаются сбором данных, а только используют предоставленные контроллером. Действия находятся в одноименной папке в контейнере и должны называться по тому, какое действие они производят, что позволит лишь взглянув на классы понять, что делает тот или иной контейнер.

Правила реализации действий:

- каждое действие отвечает за выполнение одной части бизнес-логики, так соответственно можно в дальнейшем легко расширять программный код новыми действиями, не изменяя действующие решение;

- действие может передавать данные между задачами, тем самым разделяя бизнес-логику на мелкие элементы, что также улучшает читаемость и расширяемость;

- действие может вызывать несколько задач, которые в свою очередь могут вызывать задачи из других контейнеров;

- действия должны вернуть ответ контроллеру;

- действие не должно вызывать другое действие, тем самым при разработке не создается ненужных зависимостей;

- каждое действие имеет только одну функцию run(), которая распределяет данные по задачам инкапсулируя их;

- в действиях обрабатываются все возможные исключения, которые возникают во всех вложенных поддействиях и подзадачах.

Tasks (Задачи)

Классы задач наследуются от класса Task, реализованным в ядре «корабля» и содержат общую бизнес-логику для нескольких действий в разных контейнерах. Каждая задача отвечает за небольшую часть бизнес-логики. Задачи можно повторно использовать как в действиях, поддействиях и любых других будущих частях логики.

Главный смысл задач состоит в том, что всякий раз, когда необходимо повторно использовать фрагмент кода в действиях, то его необходимо вынести в отдельную задачу. При этом необязательно создавать для всего задачи, так как это может плодить много файлов. Сначала можно написать всю бизнес логику контейнера в действии, а потом перенести повторяющиеся части в отдельные задачи, что улучшит читабельность и масштабируемость в дальнейшем.

Правила реализации задач:

- каждая задача имеет единственную ответственность и соответствует принципу первого правила акронима SOLID;

- задача получает и возвращает данные, но не ответ контроллеру;

- задачи вызываются только из Действия, так как это приводет к появлению недокументированных функций;

- задача не должна вызывать другую задачу, так как это нарушит иерархичность и соответственно читаемость кода. По этой же причина она не должна вызывать и действие;

- задачи также как и действия имеют единственную функцию run(). Однако при необходимости у них может быть больше функций с явными именами;

- задача может принимать любые данные в параметрах, кроме объекта запроса, чтобы не нарушать общей архитектуры.

Model (Модель, M в MVC)

Модели наследуются от класса LaravelEloquentModel и обеспечивают абстракцию на базу данных, файловую систему или любое другое хранилище данных. Модели отвечают за то, как следует обрабатывать данные и следят, чтобы данные поступали в нужное хранилище.

Правила реализации моделей:

- модель не должна содержать никакую бизнес-логику. Она описывает только данные, описывающие её саму (отношения с другими моделями, скрытые поля, имя таблицы, заполняемые атрибуты и др.);

- один контейнер может содержать любое количество моделей и зависеть от моделей других контейнеров.

Views (Визуальное представление, V в MVC)

Представления содержат HTML-код, обслуживаемый приложением.

Их основная цель - отделить логику приложения от вывода данных.

Правила реализации представлений:

- представления можно вызывать только контроллеров;

- представления также, как и действия должны быть разделены на несколько файлов и папок в зависимости от того, что они отображают.

Transformers (Трансформеры выходных данных)

Трансформаторы – это классы, отвечающие за преобразование моделей в массивы. Являются аналогом Views для API ответов в формате JSON, наследуются от класса TransformerAbstract.

Трансформеры преобразуют модель или коллекцию моделей в форматированный сериализованный вид, необходимый для API.

Правила реализации трансформеров:

Все ответы API необходимо форматировать с помощью трансформаторов, чтобы стандартизировать запросы. Также каждая модель, используемая в API запросах, должна иметь трансформатор.

Exceptions (Исключения)

Исключения – это ответ, который следует ожидать при ошибках выполнения. Должен быть четко определен и описан. Наследуются от класса HttpException.

Правила реализации исключений:

- существует возможность описать локальные исключения для каждого контейнера, в котором выделить необходимые исключения для задач, подзадач, моделей или любого класса, а также общие – для слоя «корабля», которые потом можно использовать из любого контейнера;

- верхний уровень при вызове всегда должен обработать все ожидаемые исключения от вызываемого класса. Так одно из главных правил архитектуры, то действия обрабатывают все исключения, чтобы они не просачивались в компоненты верхнего уровня, например, контроллеры и не вызывали неожиданное поведение. Называть же исключения желательно наиболее конкретно и понятно, чтобы с виду было сразу понятно, что оно обрабатывает.

На основании вышесказанного рассмотрим упрощённую схему с порядком взаимодействия компонентов по этапам на уровне контейнера (рисунок 3.2).

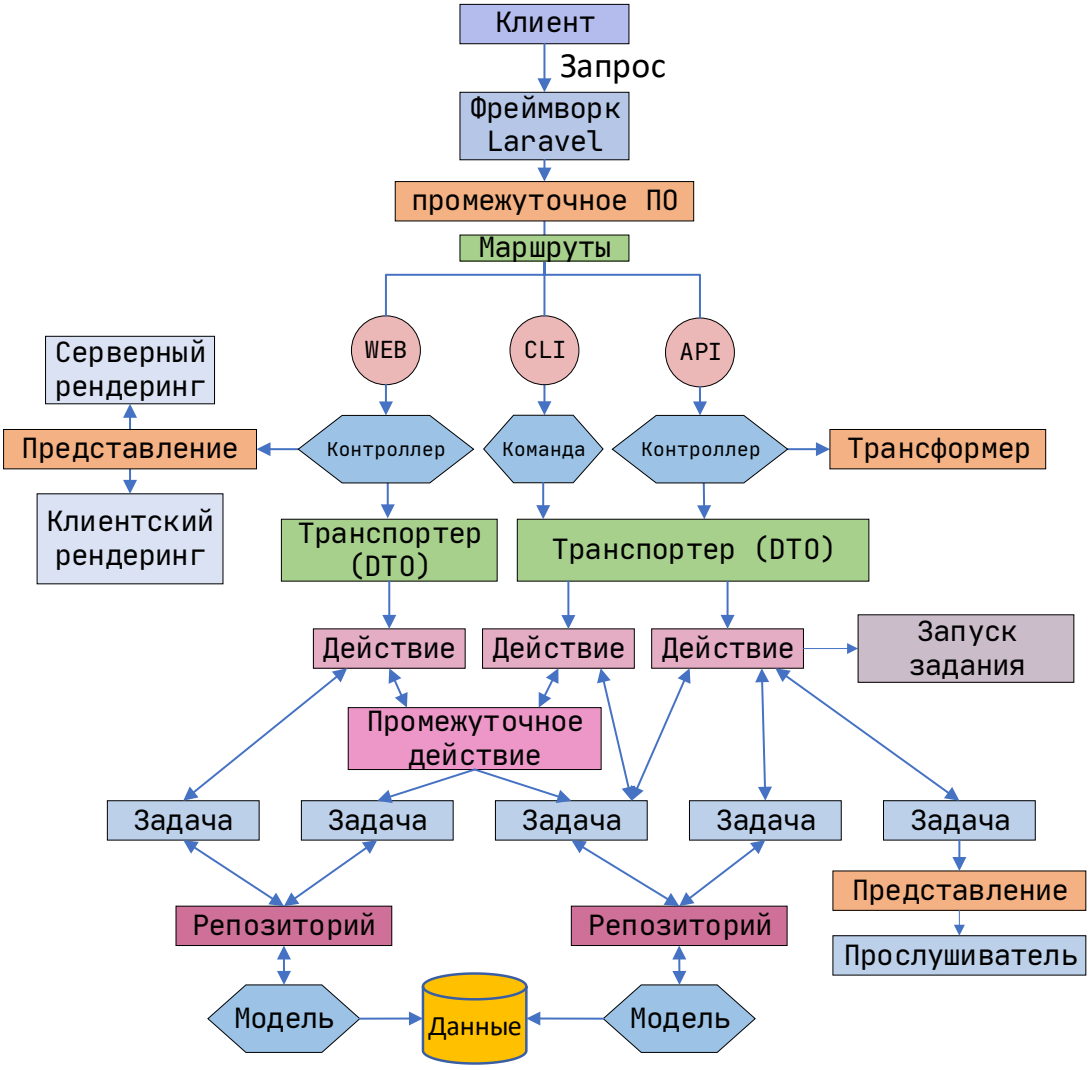

Рисунок 3.2 – Схема взаимодействия основных компонентов

1) Route (конечный маршрут) вызывается пользователем для совершения какого-либо действия;

2) Route вызывает промежуточное программное обеспечение (Middleware). Входящие данные валидируются, а также проверяется аутентификация и авторизация;

3) Route вызывает свою функцию управления в класс управления (Controller);

4) Controller вызывает действие (Action) и передает ему все данные из транспортера (DTO);

5) Action выполняет бизнес-логику, вызывая отдельные задачи (Tasks);

6) Tasks выполняют единственную задачу, тем самым принимая на себя единственную ответственность;

7) Action возвращает ответ в Controller;

8) Controller строит ответ и отправляет его обратно пользователю в виде представления или JSON данных.

3.1.4 Реализация работы с данными

Программное обеспечение для реализации каталогов данных должно получать на вход данные любого типа, хранить их, предоставляет возможность управлять и отдавать пользователям их в удобном виде.

Логическая модель данных

Рассмотрим схему обработки данных приложением (рисунок 3.3).

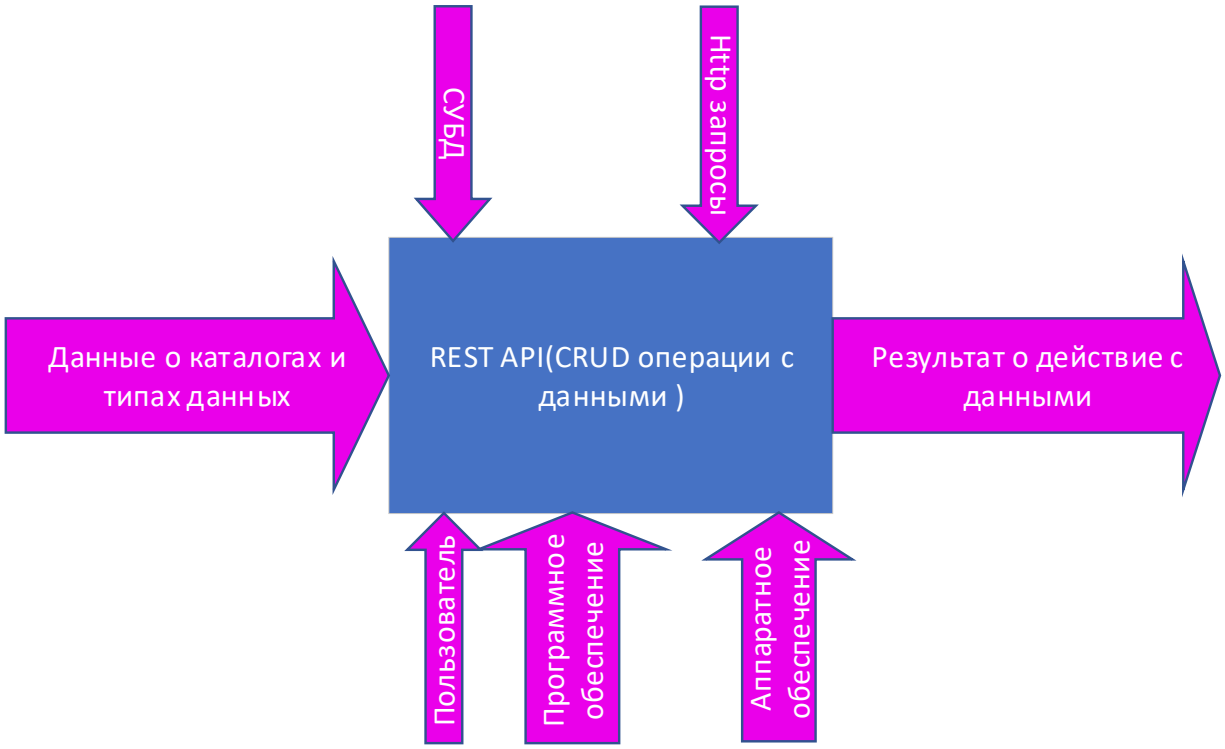

Рисунок 3.3 – Функциональная схема IDEF0 работы с данными в приложении

Как видно из схемы на вход пользователь подает некоторые данных, или запрашивает их изменение, удаление, дальше запрос проходит через API системы, где происходят необходимые операции с данными и возвращается результат в виде визуального уведомления.

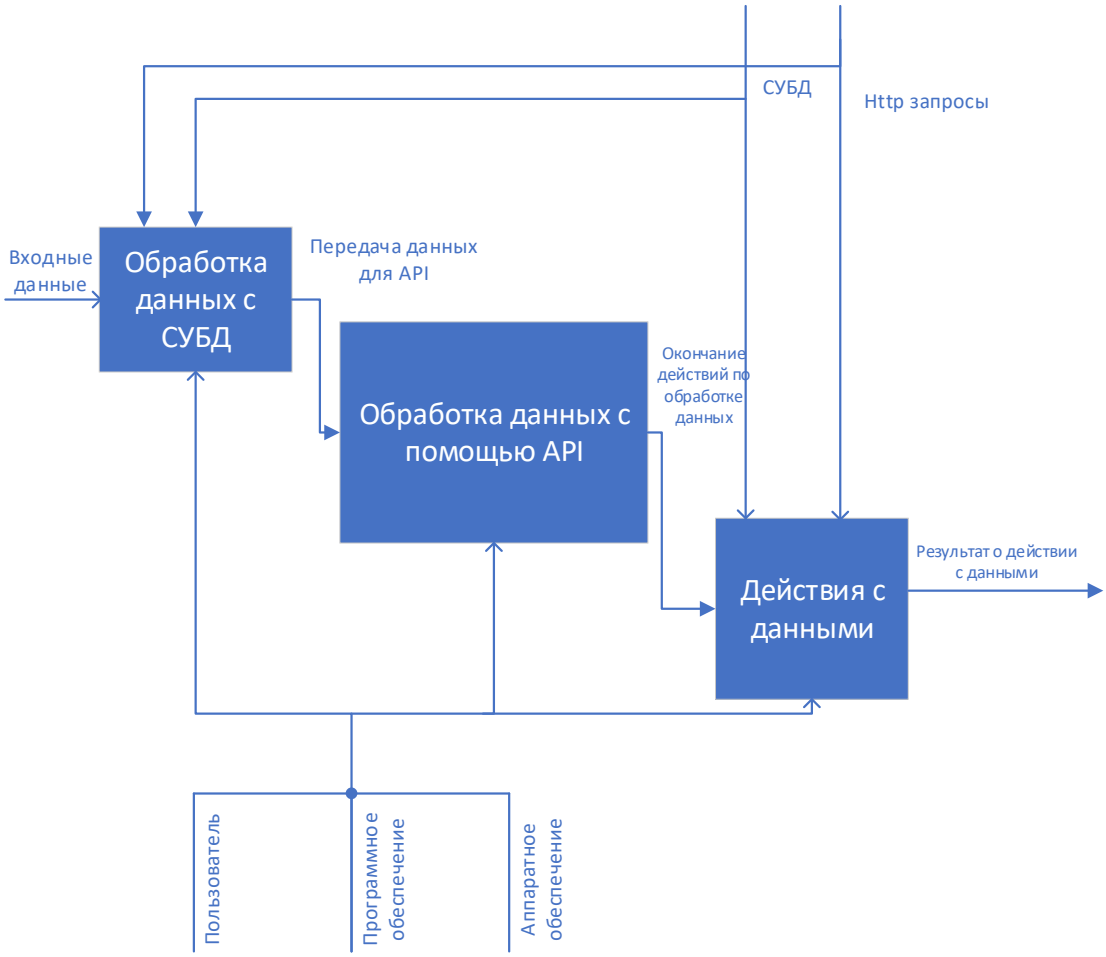

Рисунок 3.4 – Декомпозиция функциональной схемы IDEF0 работы с данными в приложении

API работает через протокол https для защиты передаваемых данных, а также требует JWT-токена или сессионные cookies для проверки авторизации.

Получение, целостность, хранение и утилизация данных

На сервере необходима установка SSL сертификата для шифрования сетевого трафика с использованием формата TLS последней версии. Это позволит отправлять и получать данные с гарантией целостности. Структура Laravel подразумевает корень сайта в папке /public, соответственно необходима настройка сервера таким образом, чтобы остальные папки на этом уровне не были доступны в браузере. Таким образом хранить данные необходимо в отдельной папке storage на сервере, доступ к которой в фреймворке осуществляется с помощью специального модуля FlyStorage, которая предоставляет единый интерфейс для взаимодействия со многими типами файловых систем.

Модель вложенного множества

Для создания иерархический деревьев в базе данных используется реализация модели вложенного множества [17].

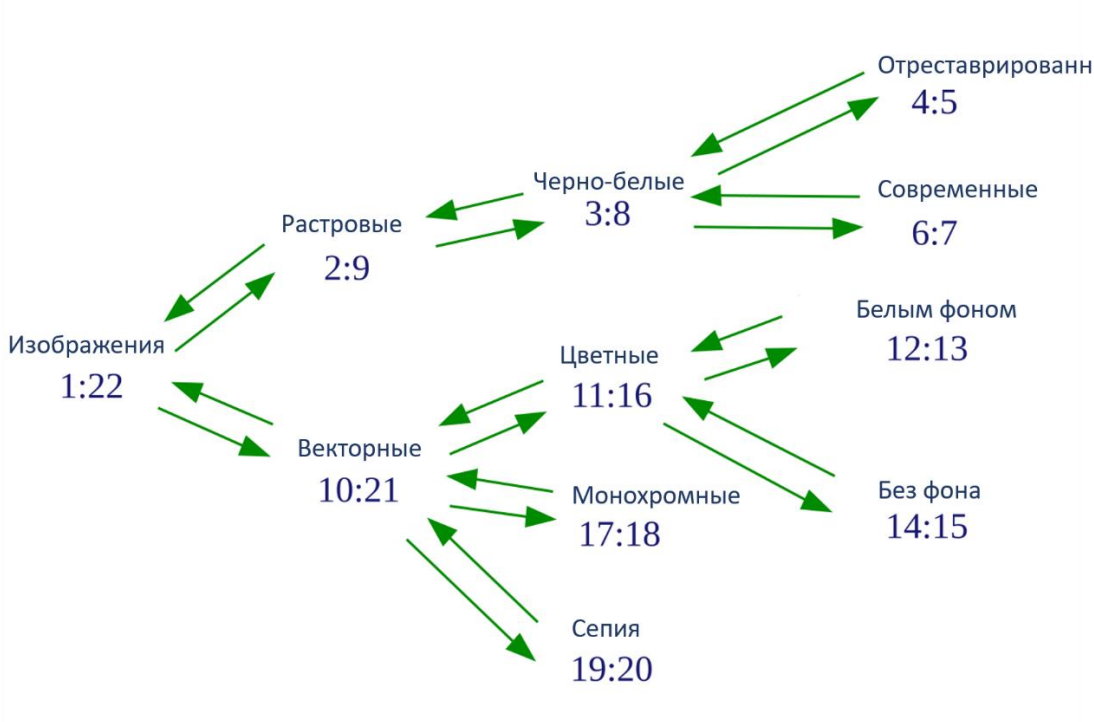

Рисунок 3.5 – Пример модели вложенного множества

Модель вложенного множества (рисунок 3.5) состоит в том, чтобы пронумеровать узлы в соответствии с обходом дерева, который дважды посещает каждый узел. При обоих посещениях назначаются номера в порядке посещения. Обход оставляет два числа для каждого узла, которые хранятся в виде двух атрибутов. Запрос становится недорогим: принадлежность к иерархии можно проверить, сравнив эти числа. Обновление узла требует перенумерации дерева и поэтому является дорогостоящим. Улучшения, в которых вместо целых чисел используются рациональные числа, помогут избежать перенумерации и поэтому узлы обновляются быстрее, хоть и намного сложнее.

3.1.5 Реализация работы с внутренними и внешними интерфейсами

Пользовательские интерфейсы

Веб-интерфейс разрабатывается с помощью клиентский компонентов, работающий на основе библиотеки Vue.js. Архитектура строится на основе шаблона архитектуры MVVM (Model-View-ViewModel). Для хранения моделей на

клиенте использовалось Vuex – это реализация Flux архитектуры, паттерн управления состоянием и библиотека для приложений на Vue.js, которая позволяет отделять данные от логики, тем самым делая разработку интерфейса более удобной, сам интерфейс перестает быть перегруженным вызовами API. Vuex служит централизованным хранилищем данных для всех компонентов приложения с правилами, гарантирующими, что состояние может быть изменено только предсказуемым образом. Vuex позволяет вынести всё общее состояние приложения из компонентов и управлять им глобально. При этом дерево компонентов становится одним большим «представлением», и любой компонент может получить доступ к состоянию приложения или вызывать действия для изменения состояния, независимо от того, где они находятся в дереве. Рассмотрим ниже схему работы с состоянием во Vuex.

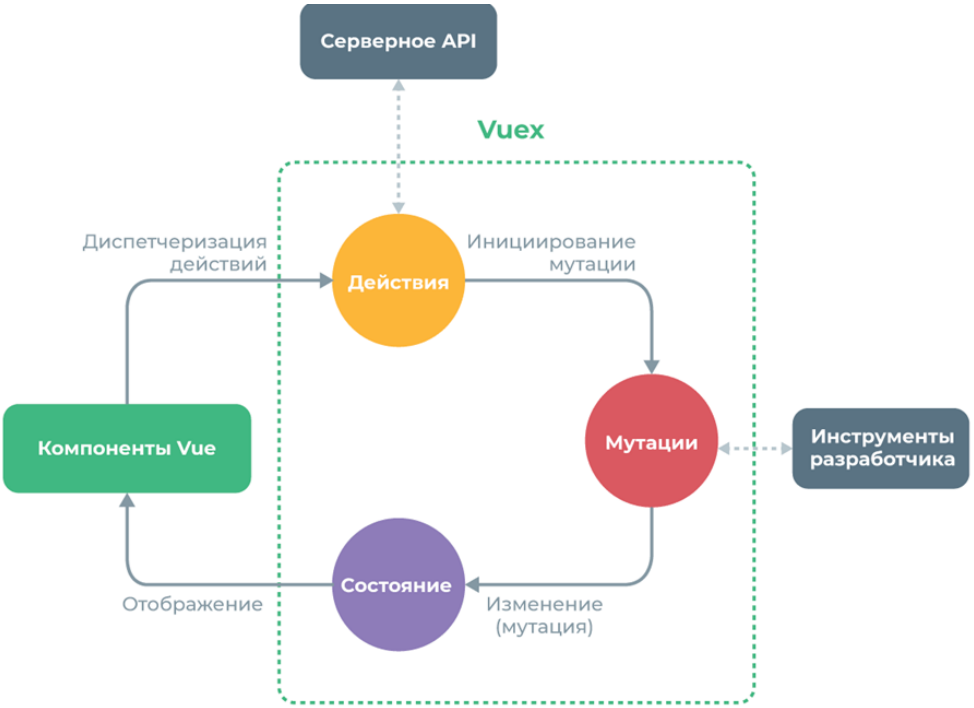

Рисунок 3.6 – Схема работы с состоянием во Vuex

Хранилище имеет 4 объекта:

- state (состояние) – хранит данные;

- getter (вызовы отображения) – позволяет получить данные из state;

- mutation (мутации) – позволяет изменять данные в state (только синхронно, для избегания конфликта транзакций);

- action (дейсвтия) – позволяет получать и изменять данные, поддерживает асинхронные вызовы.

Vuex не накладывает каких-то значительных ограничений на структуру кода, требует соблюдения нескольких правил:

- глобальное состояние приложения должно содержаться в глобальном хранилище;

- единственным механизмом изменения этого состояния являются мутации, являющиеся синхронными транзакциями;

- асинхронные операции инкапсулируются в действия или их комбинации.

Таким образом, чётко определяя и разделяя концепции, возникающие при управлении состоянием, и требуя соблюдения определённых правил, описанных выше, которые поддерживают независимость между представлениями и состояниями, получается лучше структурированный код, что облегчает поддержку.

Резюмируя, можно сказать, что Vuex хранилище содержит и реактивно обновляет состояние приложения на клиенте, полученное с помощью API предоставленное фреймворком Laravel или же напрямую из контроллера через шаблонизатор Blade.

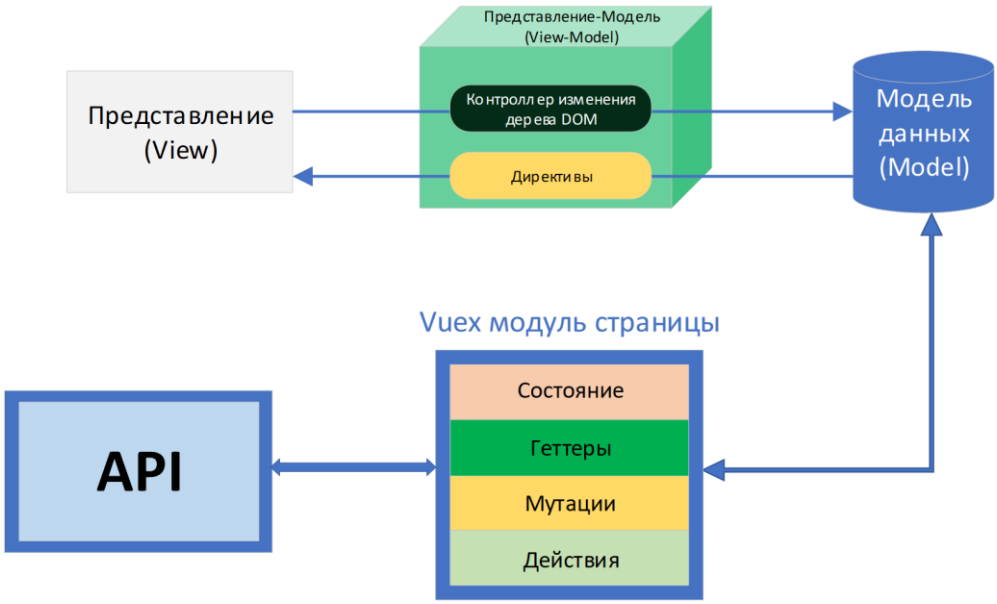

Рисунок 3.7 – Логика работы клиентской части

Таким образом модули Vuex являются единственным местом в клиенской архитектуре, где хранятся и обрабатываются данные, что позволяет

компонентам Vue.js заботиться исключено о бизнес логике, тем самым упрощая разработку и масштабирование пользовательских интерфейсов.

Vue работает только на «уровне представления» и не используется для промежуточного программного обеспечения и бэкэнда, он может легко интегрироваться с другими библиотеками.

Нет каких-то особых указаний по реализации шаблона интерфейса приложения, рекомендуется создания компонентной системы по методологии БЭМ (Блок, Элемент, Модификатор) – компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая «Copy-Paste». Также, альтернативой, возможно использование таких фреймворком для создания интерфейсов как Bootstrap, TailWind CSS и других. Они обычно предлагают свою структуру, которая также хорошо документирована.

Интерфейсы рекомендуется делать адаптивными и использованием стандартной сетки для всех основных видов устройств: смартфоны, планшеты, ПК. Работа с картинками – одна из самых главных проблем при работе с адаптивным дизайном. Рекомендуемый выбор – использование разных версий изображений под разные экраны, что увеличить скорость работы и поможет избежать проблем в дальнейшем.

Для всех вводимых данных в формах и других структурах данных необходимо делать проверку типов во избежании проблем валидации на сервере.

Интерфейс REST API

API представляет собой промежуточное звено между СУБД и клиентской частью приложения [18]. Оно принадлежит серверной части системы и обеспечивает взаимодействие компонентов с базой данных и объединяет их в единую систему.

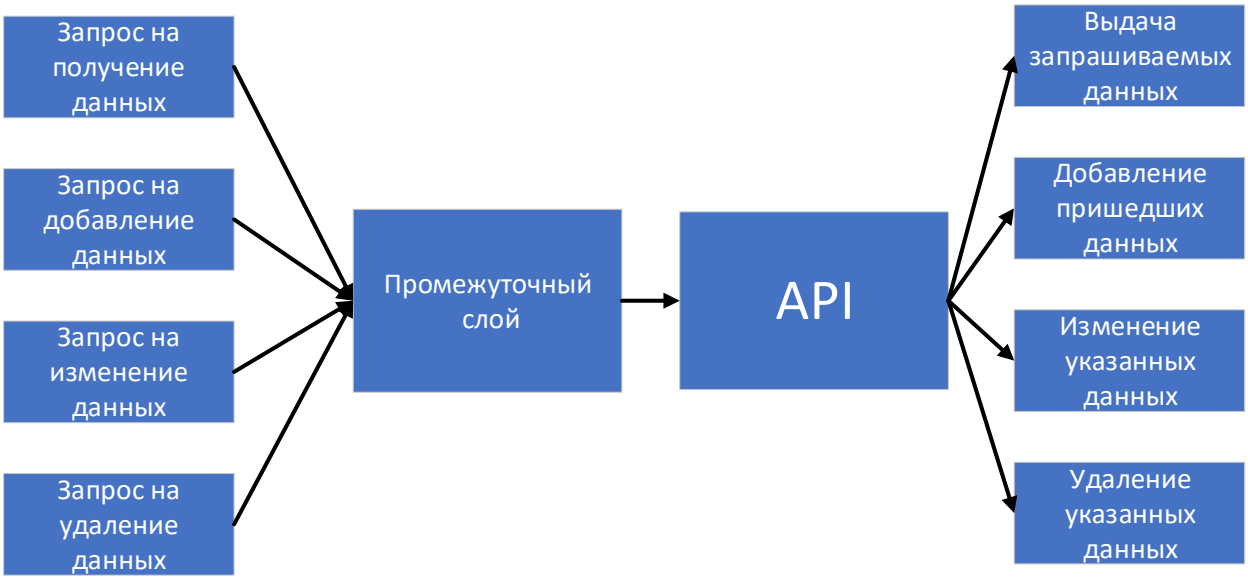

Рисунок 3.8 – Схема работы API системы

Как видно из рисунка 3.8 API получает внешний запрос на одно из действий, таких как получение, добавление, обновление и удаление. Далее запрос проходит через промежуточный слой, который является авторизацией для проверки подлинности пользователя, так, например это может быть токен авторизации. После проверки пользователя API выполняет соответствующий запрос и возвращает ответ с кодом успешного выполнения или ошибки.

Основные функции REST API контроллера модели каталогов представлены на табл. 3.1.

Таблица 3.1 – Основные функции API контроллера модели каталогов

Запрос	URI	Действие
GET	api/catalogs	Получить всё
POST	api/catalogs	Добавить запись
GET	api/catalogs /{catalog}	Получить конкретный элемент
PUT/PATCH	api/catalogs/{catalog}	Обновить конкретный элемент
DELETE	api/catalogs/{catalog}	Удалить конкретный элемент

Используя данные URI, можно получать данные из базы данных. Вместо фигурных скобок принимается объект модели. Так, например GET запрос, представленный на рисунке 3.9, возвращает JSON коллекцию, составленную с помощью технологии API Resource и возвращающую все элементы модели catalog.

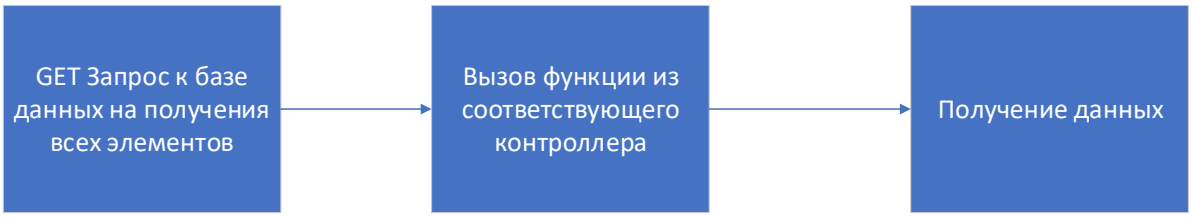

Рисунок 3.9 – Схема GET запроса получения данных с API

API используется для работы со всеми серверными компонентами и базой данных.

Интерфейс полнотекстовой поисковой системы

В качестве технологии полнотекстового поиска был выбран MeiliSearch, более подробно с которым можно ознакомиться во 2 главе. Он является масштабируемой полнотекстовой поисковой и аналитической системой с открытым исходным кодом. MeiliSearch позволяет хранить большие объемы данных, проводить среди них быстрый поиск и аналитику.

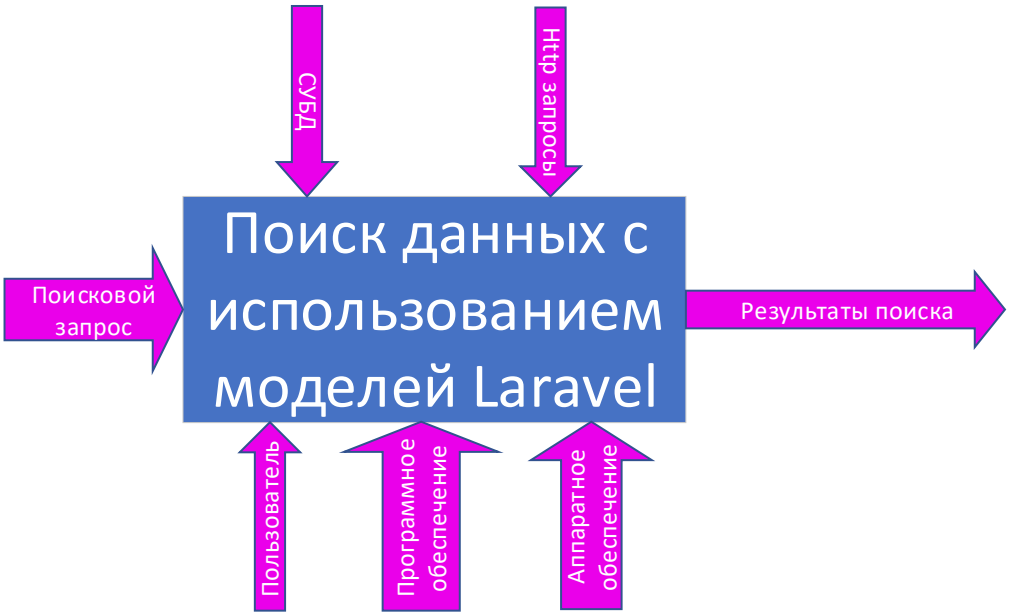

Рисунок 3.10 – Функциональная схема IDEF0 контейнера полнотекстового поиска

Контейнер работает с использованием поисковой системы MeiliSearch [19], использует базу данных и инкапсулирует внутреннюю работу (рисунок 3.10). Поиск ведётся не по именам документов, а по их содержимому, всему или существенной части. Технология заранее формируют для поиска так называемый

полнотекстовый(инвертированный) индекс - словарь, в котором перечислены все слова и указано, в каких местах они встречаются. При наличии такого индекса достаточно осуществить поиск нужных слов в нём и тогда сразу же будет получен список документов, в которых они встречаются. В результате пользователь получает ответ ввиде JSON объекта с результатами поиска.

Ниже можно подробнее рассмотреть декомпозицию функциональной схемы IDEF0 компонента полнотекстового поиска.

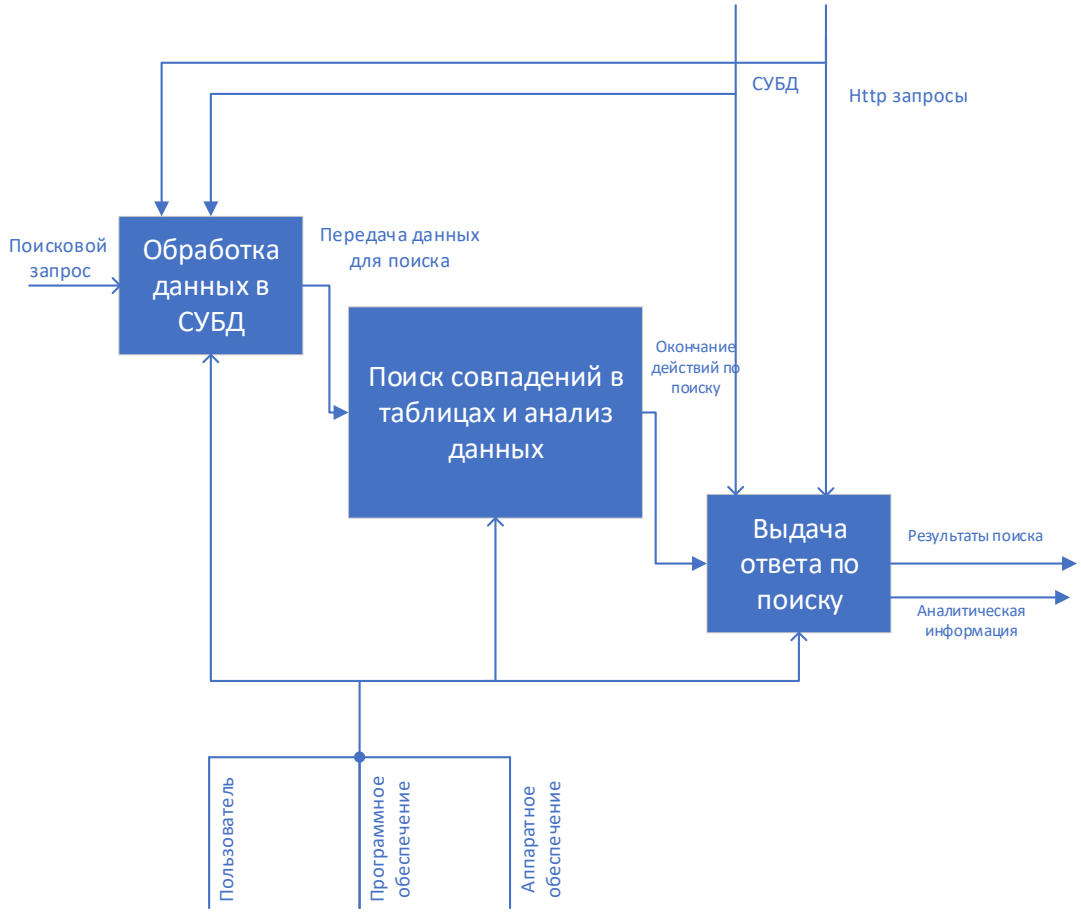

Рисунок 3.11 – Декомпозиция функциональной схемы IDEF0 компонента полнотекстового поиска

Интерфейс Oauth2

Так как приложение имеет свой API, то возникает вопрос его защиты от несанкционированного доступа. В ПО должен быть внедрён модуль OAuth2, который обеспечивает внешнюю авторизацию и соответственно защищает API так, чтобы только авторизованные модулем пользователи имели доступ к нему. За

реализацию подмодуля отвечает пакет Laravel Passport [20], созданный на основе сервера League OAuth2, схема которого представлена ниже.

Рисунок 3.12 – Абстрактное описание работы протокола OAuth2

League OAuth2 умеет создавать и хранить клиентов, коды авторизации, выдающиеся приложениям, если конечный пользователь разрешим им доступ [21]. На основании этих кодов создаются токены доступа и обновления, которые могут как общими, так и персональными.

Интерфейс позволяет создавать новых клиентов, подтверждать их токены, обновлять, создавать персональные токены.

Связка API и пакет Laravel Passport даёт хорошо спроектированный и надёжно защищенный от внешних атак интерфейс внешней авторизации.

3.1.6 Декомпозиция работы с основным функционалом

Приложение работает как многопользовательское клиент-серверное приложение, имеющее дифференцированные серверные и клиентские компоненты.

Были разработаны диаграммы прецедентов, закрытой и открытой частей приложения. Ниже представлена диаграмма прецедентов, показывающая

наиболее основные процессы при администрировании и наполнении приложения данными.

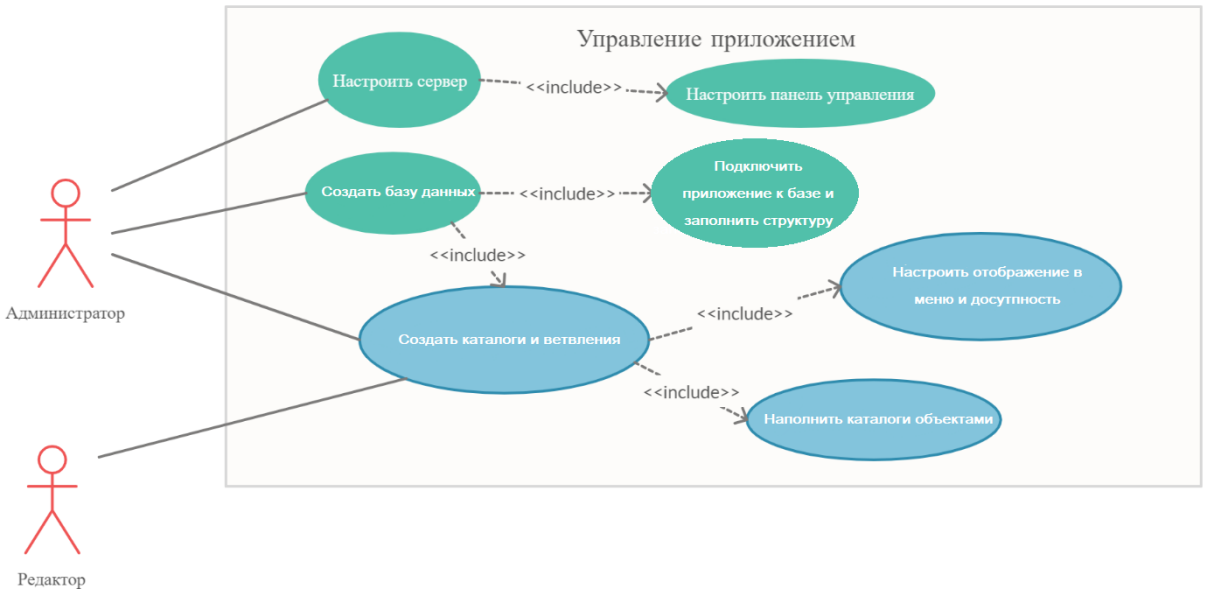

Рисунок 3.13 – Диаграмма прецедентов при управлении приложением редактором

Администратор занимается действиями, необходимым для корректной работы приложения, а то есть настраивает сервер и панель управления, создает проект, будь то библиотека или галерея и настраивает его нужным образом, возможно изменяет клиентские модули с точки зрения дизайна. Администратор также может создавать новых пользователей и настраивать их роли.

Редактор же после создания и настройки проекта может приступить к заполнению проекта данными (это может делать и администратор). Необходимо заполнить структуру проекта и создать объекты структуры, то есть сам контент. После чего приложения будет создано и готово к работе. Также можно вывести популярные объекты и визуальные улучшения клиентской части в виде баннеров и карусели.

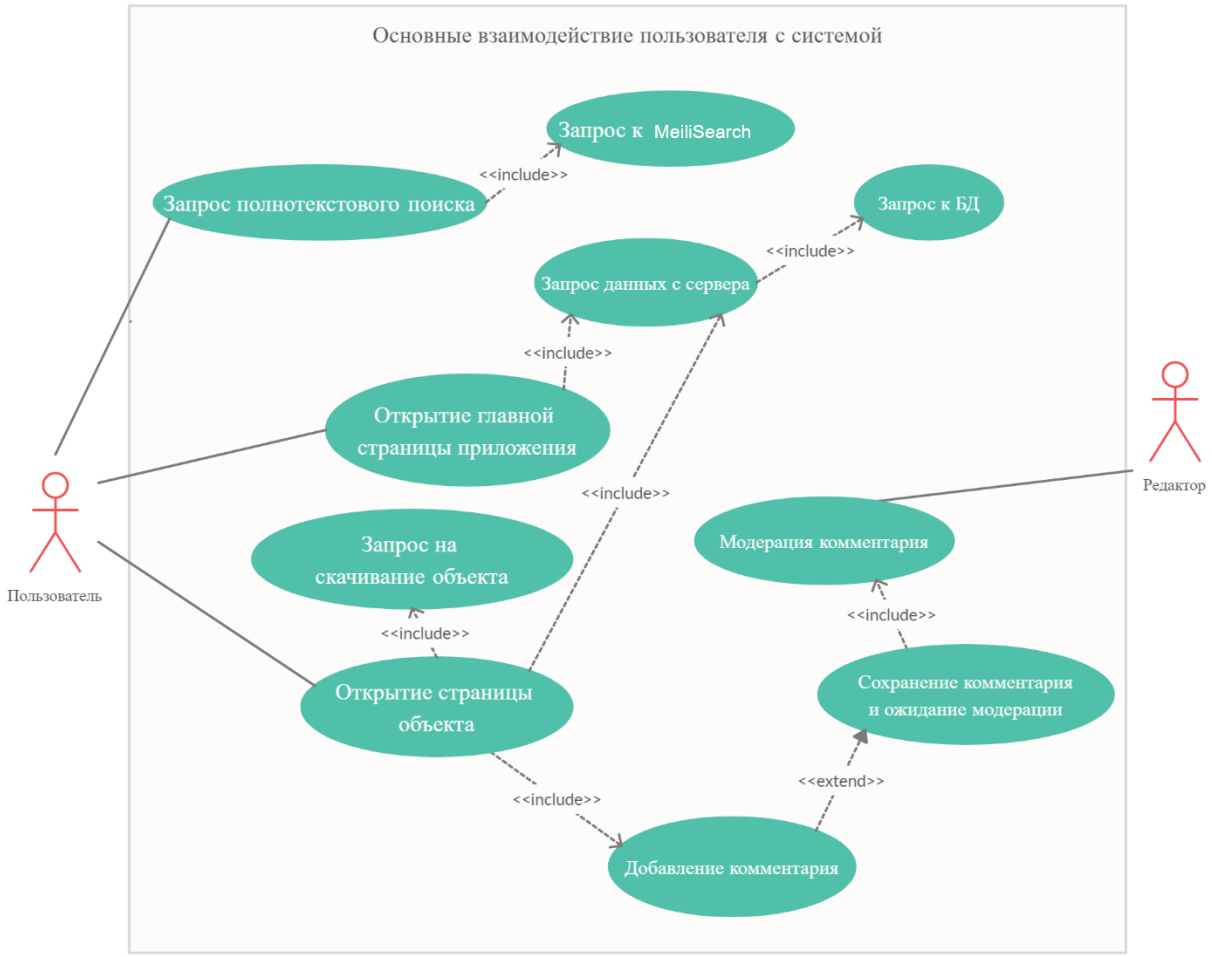

Рисунок 3.14 – Диаграмма прецедентов с точки зрения конечного пользователя

Исходя из диаграммы выше видно, что пользователю доступны страницы, созданные для в виде клиентских модулей Vue.js, их может быть много, но, если рассматривать фундаментальные это главная страница и страница просмотра объекта, а также модуль полнотекстового поиска, представленный в виде MeiliSearch. При открытии главной страницы пользователь получает все данные, пришедшие с сервера, то же происходит и на других страницах, в частности на странице объекта, можно ещё скачать объект и, например, добавить к нему комментарий, который потом пройдя модерацию, будет отображен. Запросы поиска проходят через сервер поисковой системы, индексируются и возвращаются ввиде результатов поиска, представленных в формате JSON. Также ниже можно более подробнее рассмотреть диаграмму деятельности при поисковом запросе на сервер.

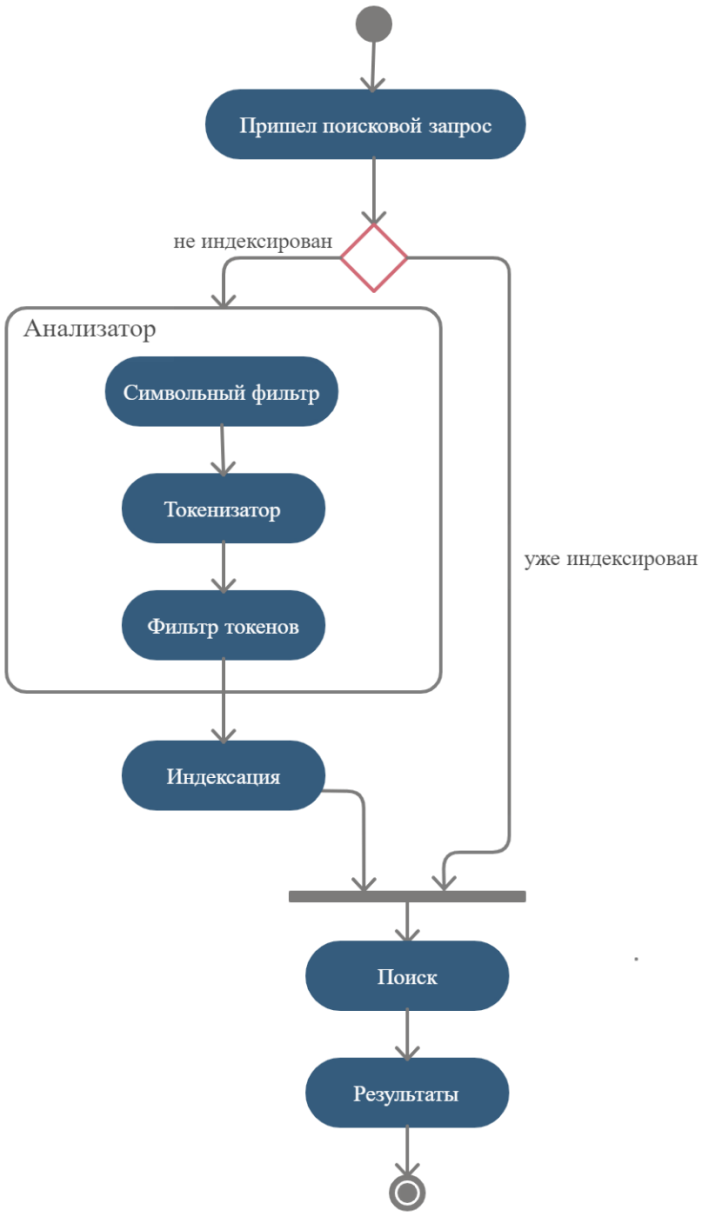

Рисунок 3.15 – Диаграмма деятельности при поисковом запросе

При запросе проверяется был ли ранее индексирован фрагмент и в зависимости от этого принимается решение, соответственно сразу провести поиск в таблице индиктов и выдать результат или же сначала запрос пройдет через символьный анализатор и разобьётся на лексемы, которые потом про индексируются.

3.2 Тестирование программного обеспечения

Тестирование систем – важный этап производства ПО, направленный на детальное исследование программного кода и выявление ошибок в работе

систем. Одна из главных целей тестирования – проверка соответствия работоспособности системы в целом или ее отдельных модулей ожиданиям заказчика.

В качестве инструмента для тестирования был выбран Laravel Dusk, который представляет простой в использовании API для автоматизации браузера и тестирования. Его целью является предоставление правильного способа тестирования различных интерактивных функций, таких как клик по кнопке или ссылке, заполнение формы, тестирование API, Drag and Drop и др. Также по Laravel Dusk является самостоятельным решением и не требует, чтобы были установлены, например JDK или Selenium. По умолчанию Dusk использует ChromeDriver для запуска тестов в браузере.

Laravel Dusk предоставляет возможность функционального тестирования, которое включает проверку функциональности приложения, корректности выполнения задач, так и проверки приложение по модулям или наборам модулей, то есть используя юнит-тестирование. Цель модульного тестирования – изолировать отдельные части программы и показать, что по отдельности эти части работоспособны [22].

Были протестированы такие части ПО как форма входа, API системы, то есть доступ ко всех записям с базы данных, а соответственно и CRUD функции, ссылки на главной странице и побочных и др.

На рисунках 3.16 и 3.17 представленных ниже показаны примеры двух тестов:

- тест входа в панель управления с помощью учетной записи администратора и дальнейшее перенаправление;

- тест получения данных из API программы, проверка всех предоставленных путей доступа к данным.

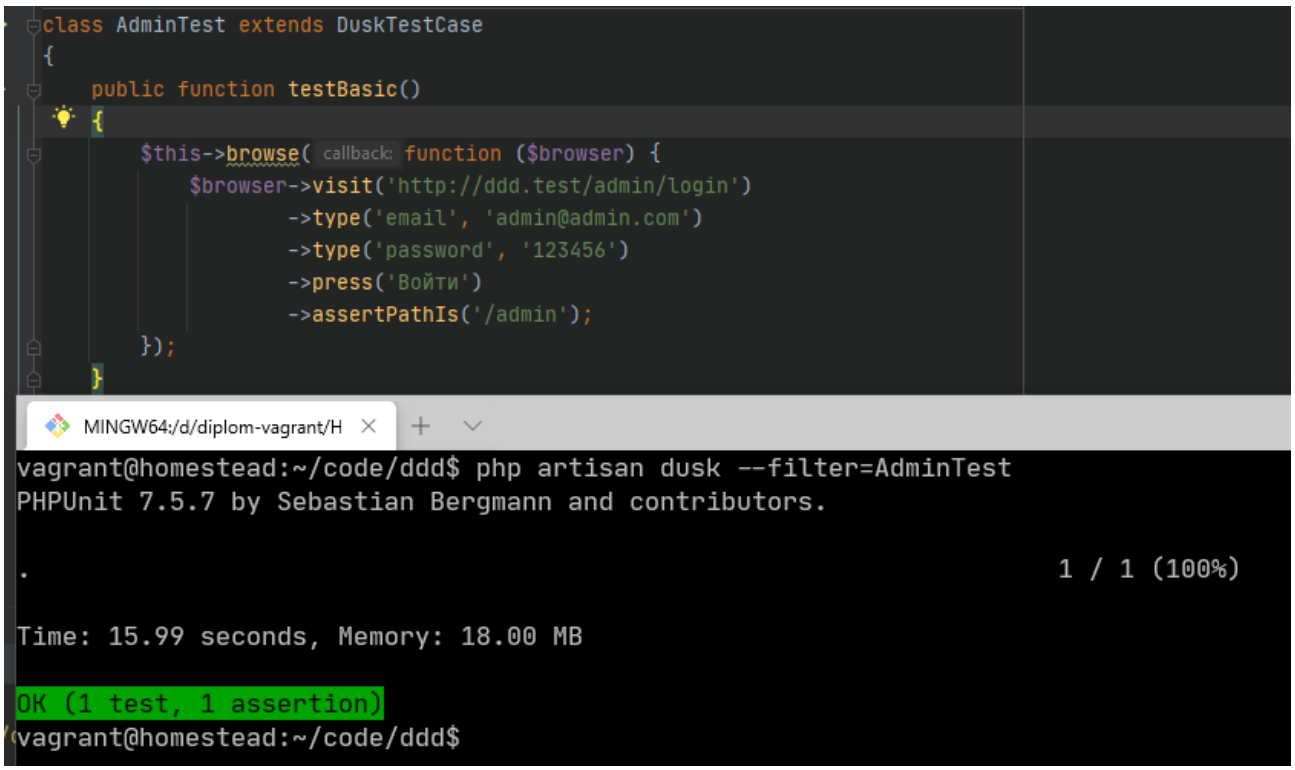

Рисунок 3.16 – Тест формы входа в панель управления

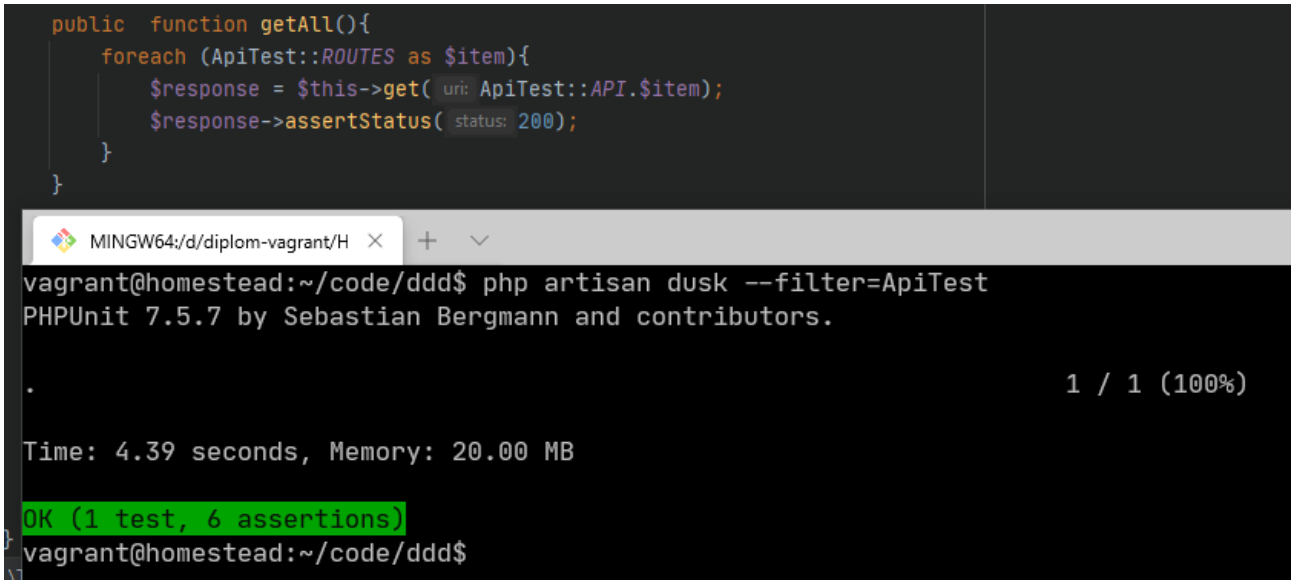

Рисунок 3.17 – Тест получения данных из API

На рисунках можно увидеть, что тесты выполнились успешно, также за какое время выполнились тесты, какое количество памяти было использовано и сколько действий было выполнено.

Не всегда тесты заканчивались успешно, периодически выдавая ошибки. На рисунке ниже рассмотрим ошибку теста, вызванную неправильным вызовом одного из API запросов.

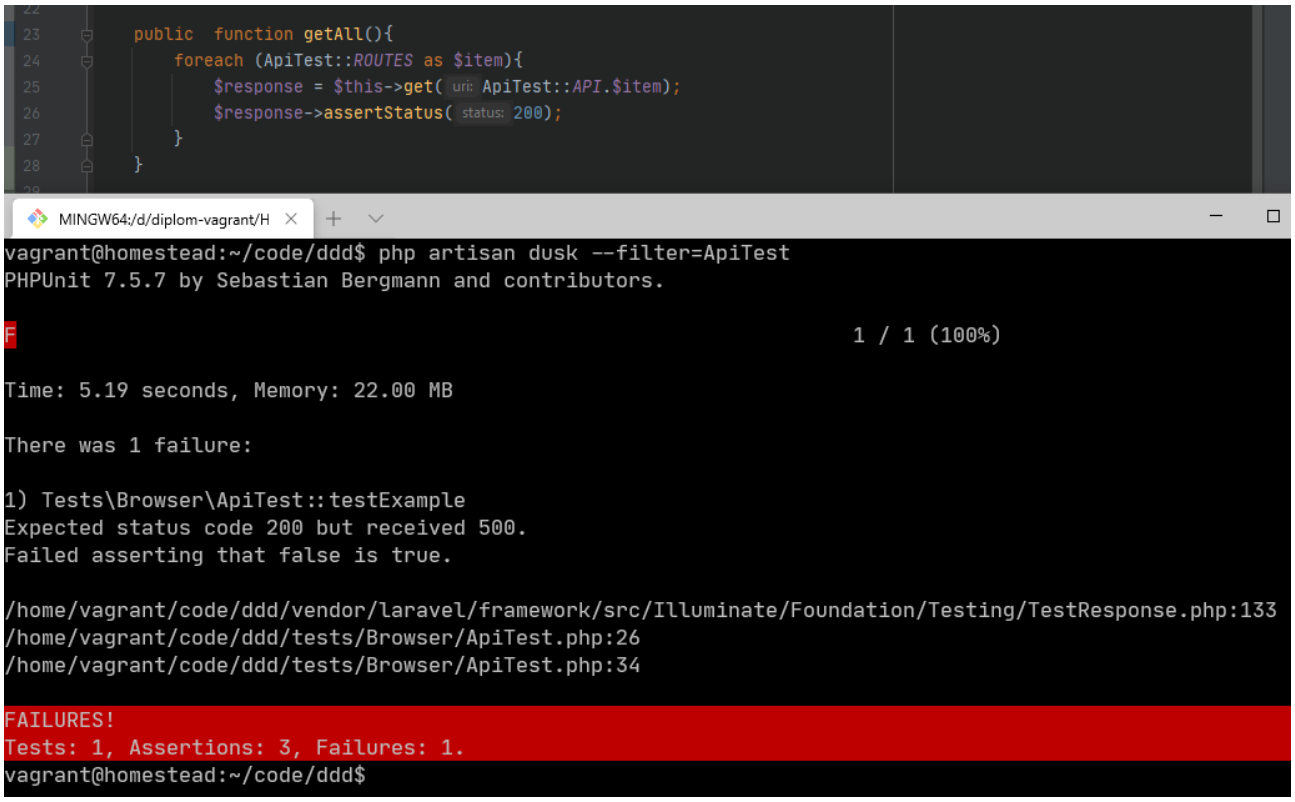

Рисунок 3.18 – Ошибка в тесте функции GET для API

Видно, что тест прошел только 3 API запроса и дальше выдал ошибку, которые удалось таким способом отладить.

Тестирование является неотъемлемой частью любой серьезной разработки, которое, позволяет быстро находить и исправлять ошибки в работе логики приложений и бизнес-логики, а также элементов приложения в браузере, как например неправильные ссылки или неработающие кнопки. Также подобные тесты были написаны и для всех остальных частей приложения, что позволяет быстро отлаживать функционал.

3.3 Применение разработанного приложения

Был проанализирован рынок программных продуктов для быстрой каталогизации больших объёмов различных данных, было выявлено, что не так много расширяемых, хорошо описанных приложений. Большинство из них устарели и не соответствуют современным стандартам скорости и простоты работы.

Так, например, для создания университетских онлайн-библиотек статей и книг часто используются системы, сделанные внутри вуза и являющиеся морально устаревшими в настоящее время или же, платформа Web-ИРБИС 64. Она предназначена для осуществления доступа пользователей к электронным каталогам и другим библиографическим базам данных. Система представляет собой типовое интегрированное решение в области автоматизации библиотечных технологий и предназначена для использования в библиотеках любого типа и профиля для использования в качестве одной из основных компонент библиотечных Интернет-серверов и Интернет-комплексов. Нельзя не отметить её достаточно обширные возможности, соответствие стандартам и долгую историю разработки, но при этом система имеет невзрачный устаревший интерфейс и медленную скорость работы, а также высокий порог вхождения, что является существенными недостатками. Также существуют и другие решения, по типу продуктов компании «ОБС», например, АБИС «Руслан-Нео», которые являются не менее обширными и полнофункциональными. Но все данные продукты являются коммерческими и не дешевыми, а также представляют собой ПО для создания и автоматизации библиотек. Подобный функционал могут в той или иной степени предоставить CMS системы, как, например, Wordpress, 1С-Битрикс, OpenCart и другие. Некоторые достаточно дорогие, как например 1С-Битрикс, а также предоставляют в большинстве своём устаревшую кодовую базу и отсутствие каких-либо архитектурных подходов и безопасности. Другие же, в свою очередь, или слишком универсальные и по умолчанию предоставляют только небольшой базовый функционал, или узконаправленные, как OpenCart для разработки интернет-магазинов или WordPress для новостных сайтов или блогов. Также для полноценной работы CMS обычно требуют установки сторонних плагинов и тем, которые часто также платные и не всегда хорошо обновляются.

В работе разработано и тестируется открытое приложение, основанное на современных фреймворках Laravel и Vue.js, с помощью которого можно создать дифференцированные каталоги с возможностью хранить данные различных форматов с возможностью удобного редактирования и дополнения, а также

использования полнотекстового поиска. Наличие модульной архитектуры и внешней доверенной авторизации дает удобство расширения для других разработчиков. Хотя приложение и не претендует соревноваться в возможностях с другими именитыми коммерческими конкурентами, но предоставляет более простой и бесплатный вариант подобных решений, который будет в большой степени интересен обычным пользователям, а также организациям, которым необходимо каталогизировать различную информацию, как например, сборники статей или фотогалереи. Ознакомиться с руководством пользователя можно в приложении А.

Также стоит отметить, что аналогичных открытых систем, основанных на фреймворках, применяемых в работе выявлено не было, что делает приложение актуальным и даже конкурентноспособным в некоторых аспектах по сравнению с другими коммерческими решениями. Из этого можно сделать, что работа является актуальной.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы был произведён анализ предметной области, по итогу которого были рассмотрены: актуальность использования веб-фреймворков при разработке современных веб-приложений, особенности разработки с использованием фреймворка Laravel и его интеграции с клиентским решением – Vue.js, а также применимость и эффективность использования систем полнотекстового поиска.

Далее было описано алгоритмическое и программное обеспечение, использованное при разработке программного обеспечения, а именно описано применяемое архитектурное решение, обоснован выбор средств разработки по сравнению с аналогами, а также описаны и выполнены этапы разработки и тестирования, результатом которых стало описание архитектуры и основных алгоритмов работы с данными и интерфейсами. Были представлены результаты разработки программных компонентов и пользовательского интерфейса, в том числе написание руководства пользователя, показаны способы тестирования, использованные при разработке и представлены UML-диаграммы декомпозиции работы с основным функционалом программного обеспечения.

Было выявлено, что не существует аналогичных открытых приложений, построенных на технологиях Laravel и Vue.js, а сторонние проекты хоть и являются более обширными, но в то же время многие из них платные, узконаправленные и не позволяют манипулировать различными типами данных, что делает приложение актуальным и даже конкурентноспособным в некоторых аспектах по сравнению с другими коммерческими решениями, а выбранную тематика оправдывает свою актуальность и практическую значимость.

В результате было разработано, тестируется и продолжает дорабатываться полнофункциональное открытое веб-приложение для создания и предоставления дифференцированных каталогов данных с возможностью полнотекстового поиска по ним, с высокими возможностями масштабирования, которое не требовательно к ресурсам и работает по умолчанию с минимальными настройками.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1 Tproger.ru: Введение в веб-фреймворки [Электронный ресурс]. URL: https://tproger.ru/translations/ web-frameworks-how-to-get-started/

2 Разработка веб-приложения "Система тестирования" на основе фреймворка Laravel [Электронный ресурс] : бакалавр. работа / А. Ю. Манвелян ; рук. работы Т. А. Галаган ; АмГУ, ФМиИ, Каф. ИиУС. - Благовещенск : [б. и.], 2019. - 94 с. - Б. ц.

3 Там же. С. 15.

4 Laravel.com: Официальная документация фреймворка Laravel. Laravel Passport [Электронный ресурс]. URL: https://laravel.com/docs/8.0

5 Github.com: Архтектурный Паттерн Porto [Электронный ресурс]. URL: https://habr.com/ru/post/334126/

6 Vuejs.org: Официальная документация фреймворка Vue.js [Электронный ресурс]. URL: https://ru.vuejs.org/v2/guide/

7 Hanchett, E. Vue.js in Action / E. Hanchett, B. Listwon – 2-е изд., Manning Publications, 2018. – 375 с.

8 Php.net: Официальная документация языка программирования PHP [Электронный ресурс]. URL: https://www.php.net/manual/ru/

9 Tproger.ru: Особенности SQL и NoSQL [Электронный ресурс]. URL: https://tproger.ru/translations/sql-vs-nosql/

10 Mkdev.me: Введение в MongoDB [Электронный ресурс]. URL: https://mkdev.me/posts/vvedenie-v-mongodb

11 Opennet.ru: Основные возможности MySQL [Электронный ресурс]. URL: https://www.opennet.ru/docs/RUS/mysqlcli/glava01.html

12 Laravel.com: Официальная документация фреймворка Laravel. Laravel Scout [Электронный ресурс]. URL: https://laravel.com/docs/8.x/scout

13 Alexeykalina.github.io: Elasticsearch find it [Электронный ресурс]. URL: https://alexeykalina.github.io/technologies/elasticsearch-fulltextsearch.html

14 Wolff, E. A Practical Guide to Continuous Delivery / E. Wolff – 2-е изд., Manning Publications, 2017. – 288 с.

15 Opennet.ru: Основные возможности MySQL [Электронный ресурс]. URL: https://www.opennet.ru/docs/RUS/mysqlcli/glava01.html

16 Ruhighload.com: Полнотекстовый поиск [Электронный ресурс]. URL: https://ruhighload.com/Полнотекстовый+поиск

17 Wikipedia.org: Nested set model [Электронный ресурс]. URL: wikipedia.org/wiki/Nested_set_model

18 Habr.com: что такое API [Электронный ресурс]. URL: https:// https://habr.com/ru/post/464261

19 Platform.sh: Deploy Meilisearch on Platform.sh [Электронный ресурс]. URL: https://platform.sh/marketplace/templates/meilisearch/

20 Laravel.com: Официальная документация фреймворка Laravel. Laravel Passport [Электронный ресурс]. URL: https://laravel.com/docs/8.0

21 Oauth.net: Официальная документация протокола авторизации OAuth 2.0 [Электронный ресурс]. URL: https://oauth.net/2/

22 Laravel.com: Официальная документация фреймворка Laravel. Laravel Dusk [Электронный ресурс]. URL: https://laravel.com/docs/8.x/dusk

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Дронов, В. Laravel. Быстрая разработка современных динамических Web-сайтов на PHP, MySQL, HTML и CSS / В. Дронов. – Изд-во БХВ-Петербург, 2018. – 688 с.

2 Котеров, Д.В, PHP 7 / Д.В. Котеров. – Изд-во БХВ-Петербург, 2016. – 1088 с.

3 Манвелян, А.Ю., Галаган Т.А. Применение фреймворка Laravel при разработке веб-приложения для управления дифференцированными каталогами данных // Системный анализ в медицине: матер. XIV межд.научн.конф. (САМ – 2020) (15-16 октября 2020г., Благовещенск). – Благовещенск: Изд-во ДНЦ ФПД, 2020. – С. 58-60

4 Манвелян, А.Ю., Галаган Т.А. Разработка веб-приложения «Система тестирования» на основе фрейморка Lavarel // Вестник АмГУ. Серия «Естественные и экономические науки», Благовещенск: АмГУ, 2019. – Выпуск 87. – С. 49-53

5 Манвелян А.Ю., Научный руководитель – Галаган Т.А. Проектирование веб-приложения для реализации каталогов данных с применением полнотекстового поиска // Молодежь XXI века: шаг в будущее: материалы XXI региональной научно-практической конференции (20 мая 2020г., Благовещенск). – Изд-во Дальневост. гос. аграр. ун-та, 2020. – С. 122-123

6 Манвелян А.Ю., Научный руководитель – Галаган Т.А. Разработка web-приложения для реализации каталогов данных // Молодежь XXI века: шаг в будущее: материалы XXII региональной научно-практической конференции (20 мая 2021г., Благовещенск). – Благовещенск: Изд-во БГПУ, 2021. – С. 766-767

7 Манвелян, А.Ю., Галаган Т.А. Применение хранилища состояния Vuex при разработке пользовательских интерфейсов с помощью Vue.js на основе фреймворка Laravel // «Современные инновации в науке и технике» (МТО-57): материалы 11-й Всероссийской научно-технической конференции (15-16 апреля 2021 г., Курск). – С. 166-170

8 Разработка веб-приложения "Система тестирования" на основе фреймворка Laravel [Электронный ресурс] : бакалавр. работа / А. Ю. Манвелян ; рук. работы Т. А. Галаган ; АмГУ, ФМиИ, Каф. ИиУС. - Благовещенск : [б. и.], 2019. - 94 с. - Б. ц.

9 Орлов, С.А. Технологии разработки программного обеспечения //С.А. Орлов. – учеб. СПб.: Питер, 2012. – 608 с.

10 Петров, В.И. Информационные системы // В.И. Петров. – СПб.: Питер, 2002. – 688 с.

11 Эванс, Э. Предметно-ориентированное проектирование (DDD) / Эванс Эрик. – М.: Вильямс, 2018. – 448 с.

12 Alexeykalina.github.io: Elasticsearch find it [Электронный ресурс]. URL: https://alexeykalina.github.io/technologies/elasticsearch-fulltextsearch.html

13 Aws.amazon.com: Документная база данных [Электронный ресурс]. URL: https://aws.amazon.com/ru/nosql/document

14 Bignerdranch.com: Flux-архитектураThe Flux pattern in VueJS [Электронный ресурс]. – Режим доступа https://www.bignerdranch.com/blog/the-flux-pattern-in-vuejs-building-a-simple-store

15 Codex.so: Эффективный поиск на сайте с помощью Elasticsearch [Электронный ресурс]. URL: https://codex.so/elastic-search

16 Developer.mozilla.Org: Документация к языку программирования JavaScript [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/

17 Digitalocean.com: Введение в OAuth [Электронный ресурс].URL: https://www.digitalocean.com/community/tutorials/oauth-2-ru

18 Github.com: Архтектурный Паттерн Porto [Электронный ресурс]. URL: https://habr.com/ru/post/334126/

19 Habr.com: Архитектура REST [Электронный ресурс]. URL: https://habr.com/ru/post/38730/

20 Habr.com: что такое API [Электронный ресурс]. URL: https:// https://habr.com/ru/post/464261

21 Habr.com: Паттерн Domain Driven Design [Электронный ресурс]. URL: https://habr.com/ru/post/334126/

22 Habr.com: Vuex: структурирование больших проектов и работа с модулями [Электронный ресурс]. – Режим доступа https://habr.com/ru/company/ruvds/blog/420357/

23 Laravel.com: Официальная документация фреймворка Laravel. Laravel Passport [Электронный ресурс]. URL: https://laravel.com/docs/8.0

24 Laravel.com: Официальная документация фреймворка Laravel. Laravel Scout [Электронный ресурс]. URL: https://laravel.com/docs/8.x/scout

25 Laravel.com: Официальная документация фреймворка Laravel. Laravel Sunctum [Электронный ресурс]. URL: https://laravel.com/docs/8.x/ sanctum

26 Laravel.com: Официальная документация фреймворка Laravel. Laravel Dusk [Электронный ресурс]. URL: https://laravel.com/docs/8.x/dusk

27 Mkdev.me: Введение в MongoDB [Электронный ресурс]. URL: https://mkdev.me/posts/vvedenie-v-mongodb

28 Meilisearch.com: Особенности Meilisearch [Электронный ресурс]. URL: https://docs.meilisearch.com/

29 Oauth.net: Официальная документация протокола авторизации OAuth 2.0 [Электронный ресурс]. URL: https://oauth.net/2/

30 Opennet.ru: Основные возможности MySQL [Электронный ресурс]. URL: https://www.opennet.ru/docs/RUS/mysqlcli/glava01.html

31 Php.net: Официальная документация языка программирования PHP [Электронный ресурс]. URL: https://www.php.net/manual/ru/

32 Ruhighload.com: Полнотекстовый поиск [Электронный ресурс]. URL: https://ruhighload.com/Полнотекстовый+поиск

33 Tproger.ru: Введение в веб-фреймворки [Электронный ресурс]. URL: https://tproger.ru/translations/web-frameworks-how-to-get-started/

34 Tproger.ru: Особенности SQL и NoSQL [Электронный ресурс]. URL: https://tproger.ru/translations/sql-vs-nosql/

35 Vuejs.org: Официальная документация фреймворка Vue.js [Электронный ресурс]. URL: https://ru.vuejs.org/v2/guide/

36 Ru.vuejs.org: Управление состоянием приложения [Электронный ресурс]. – Режим доступа: https://ru.vuejs.org/v2/guide/state-management.html

37 Vuex.vuejs.org: Использование констант для обозначения типов объектов хранилища Vuex [Электронный ресурс]. – Режим доступа: https://vuex.vuejs.org/ru/guide/mutations.html#использование-констант-для-обозначения-типов-мутации

38 Web-creator.ru: MongoDB - документо-ориентированная база данных (NoSQL) [Электронный ресурс]. URL: https://web-creator.ru/articles/mongo_db

39 Hanchett, E. Vue.js in Action / E. Hanchett, B. Listwon – 2-е изд., Manning Publications, 2018. – 375 с.

40 Stauffer, M. Laravel: Up & Running / M. Stauffer. – 2-е изд., – М.: 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, 2019. – 454 с.

41 Wolff, E. A Practical Guide to Continuous Delivery / E. Wolff – 2-е изд., Manning Publications, 2017. – 288 с.

42 Blog.andrewray.me: Flux for Beginners [Электронный ресурс]. – Режим доступа: https://blog.andrewray.me/flux-for-beginners/

43 Platform.sh: Deploy Meilisearch on Platform.sh [Электронный ресурс]. URL: https://platform.sh/marketplace/templates/meilisearch/

44 Wikipedia.org: Nested set model [Электронный ресурс]. URL: wikipedia.org/wiki/Nested_set_model

ПРИЛОЖЕНИЕ А

Руководство пользователя

На основании тестирования было составлено руководство пользователя для работы с приложением.

Установка

Для установки системы необходимо предварительно настроить веб-сервер, на корень «/public», скопировать туда файлы закинув их архивом или же получив их с хранилища проектов GitHub. Далее в конфиге «.env» необходимо настроить подключение к базе данных. Далее нужно выполнить команду artisan install команду на сервере для установки решения и перейти в браузере по вашему доменному имени, где уже следовать дальнейшим инструкциям.

Форма входа

Для доступа к панели управления как администратор или редактор необходимо авторизоваться в форме входе, введя email и пароль (рисунок 1).

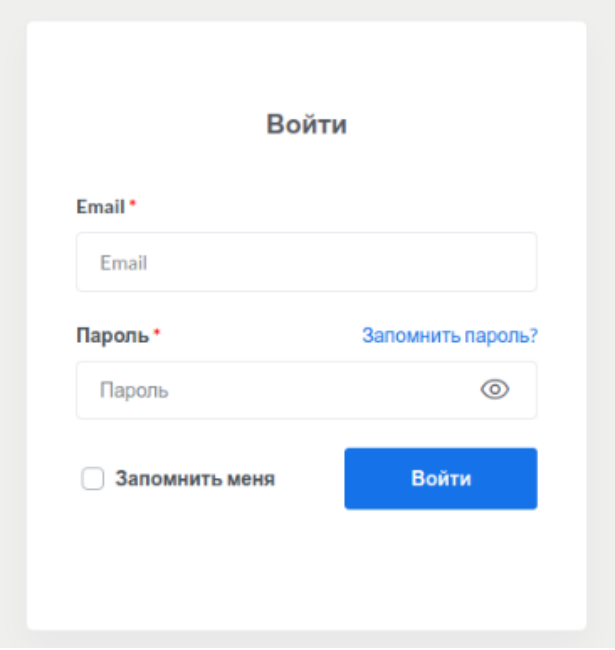

Рисунок 1 – Форма входа в панель управления

Так же чтобы войти как обычный пользователь нужно сначала зарегистрироваться, введя необходимые данные на форме (рисунок 2).

Продолжение ПРИЛОЖЕНИЯ А

Руководство пользователя

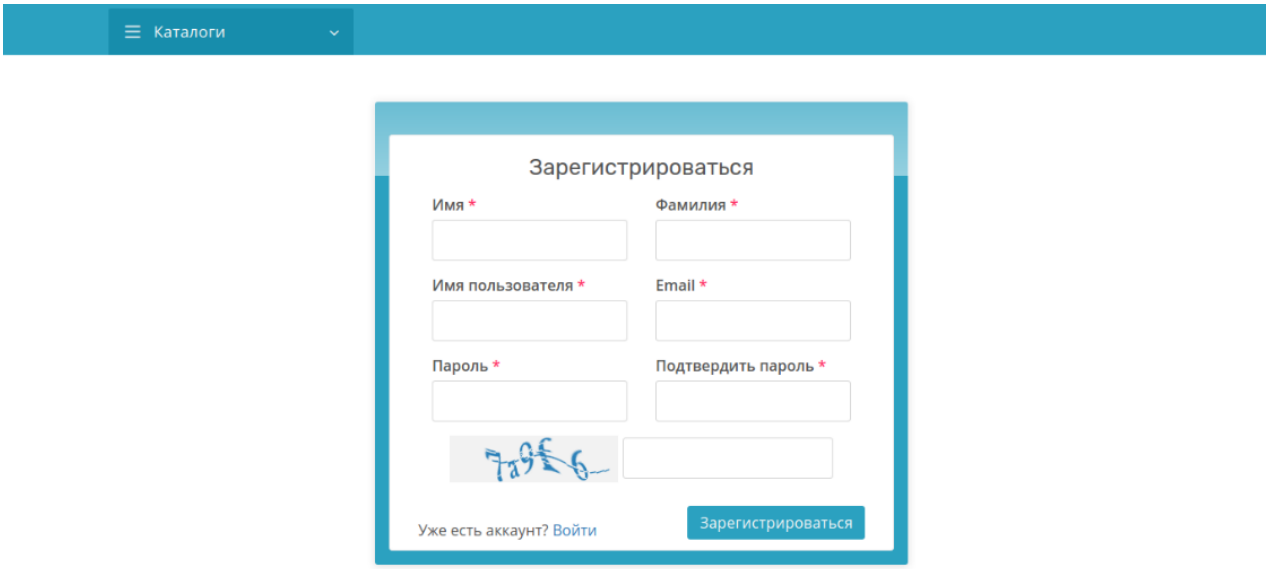

Рисунок 2 – Форма регистрации пользователя

Далее необходимо войти с помощью созданной учетной записи (рисунок 3), чтобы оценивать, комментировать, оставлять отзывы и др.

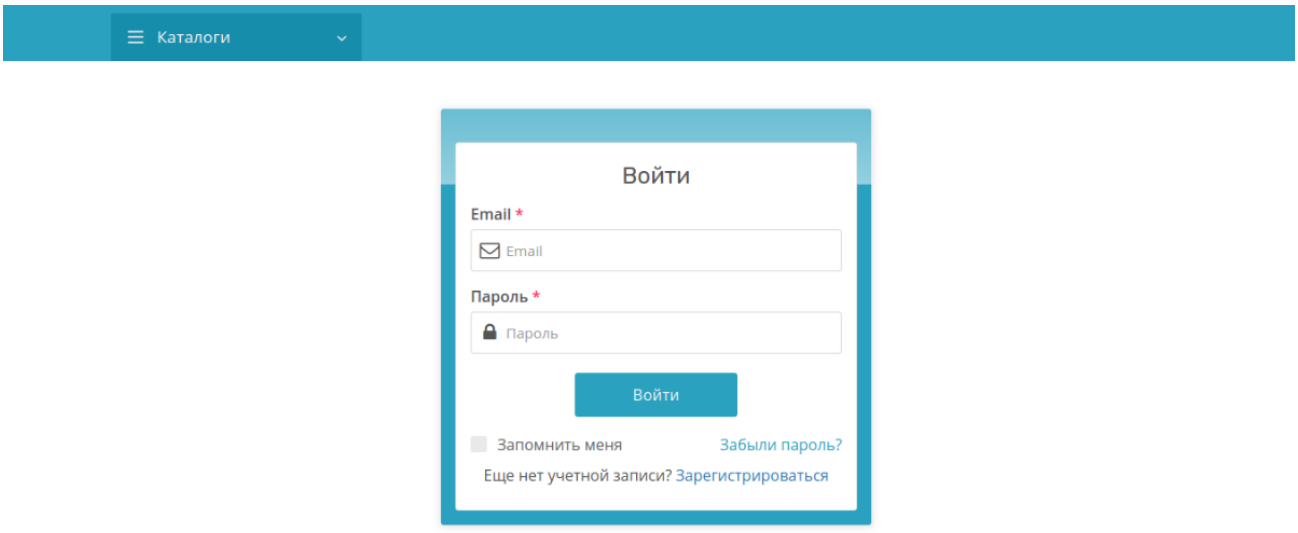

Рисунок 3 – Форма входа пользователя

После входа пользователь получает доступ к личному кабинету (рисунок 4), где может совершать различные действия, как, например, публиковать свои материалы или изменить данные от учетной записи.

Продолжение ПРИЛОЖЕНИЯ А

Руководство пользователя

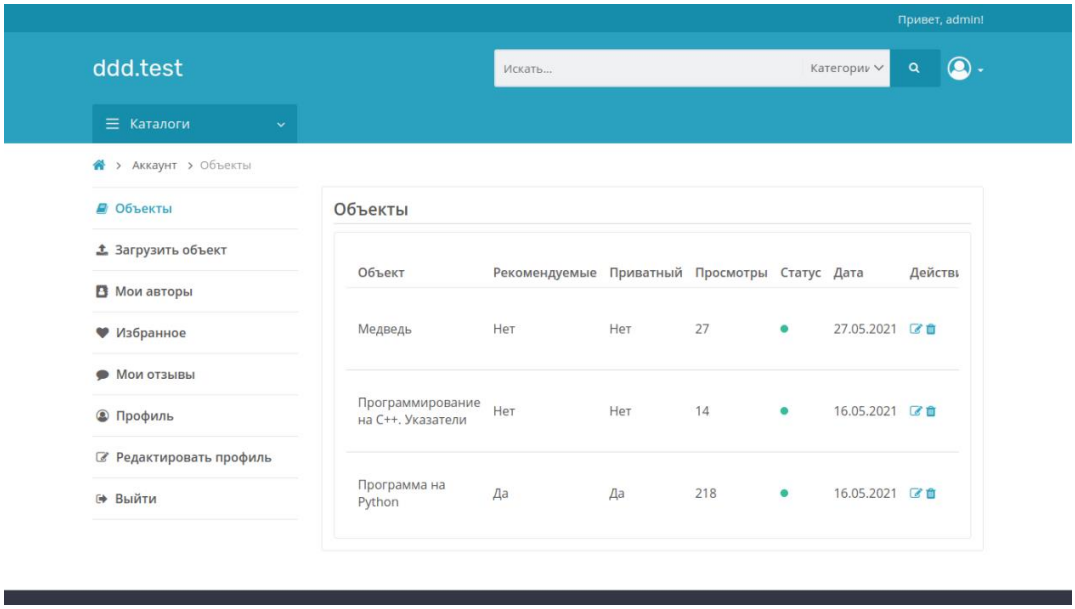

Рисунок 4 – Личный кабинет пользователя

Панель администрирования

Войдя как администратор или пользователь панели администрирования на главной странице (рисунок 5) можно увидеть общую статистику по объектам и пользователям по количеству, времени и верификации. Также можно увидеть последние загруженные объекты и информации о них.

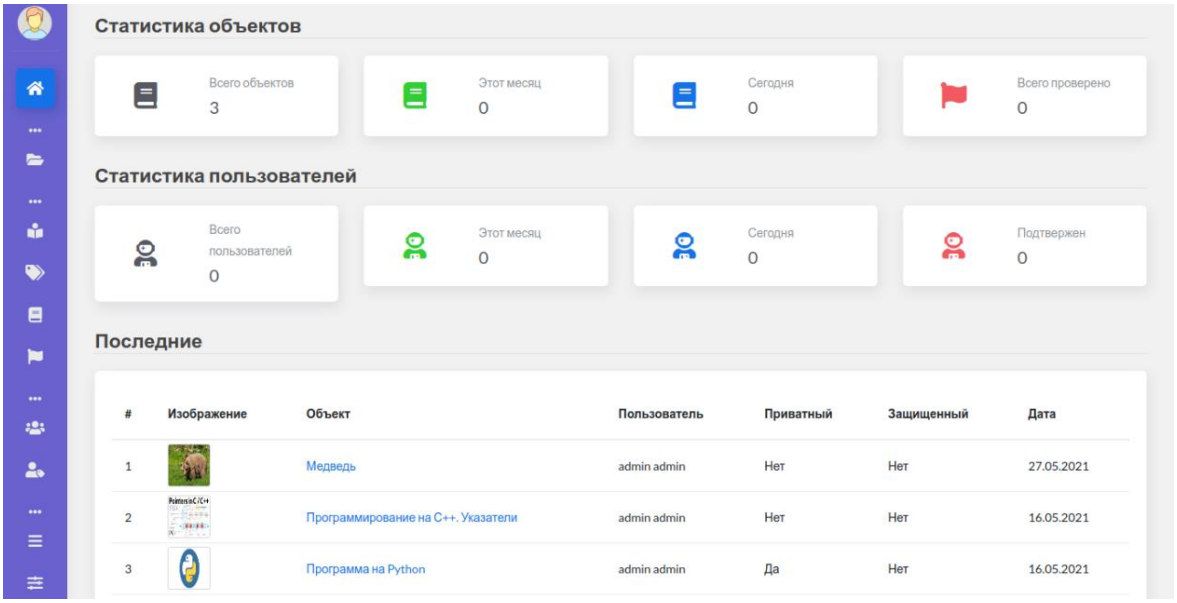

Рисунок 5 – Главный экран панели управления

Руководство пользователя

Также в панели имеются другие настройки приложения, как например, создание и редактирование пользователей и ролей (рисунок 6).

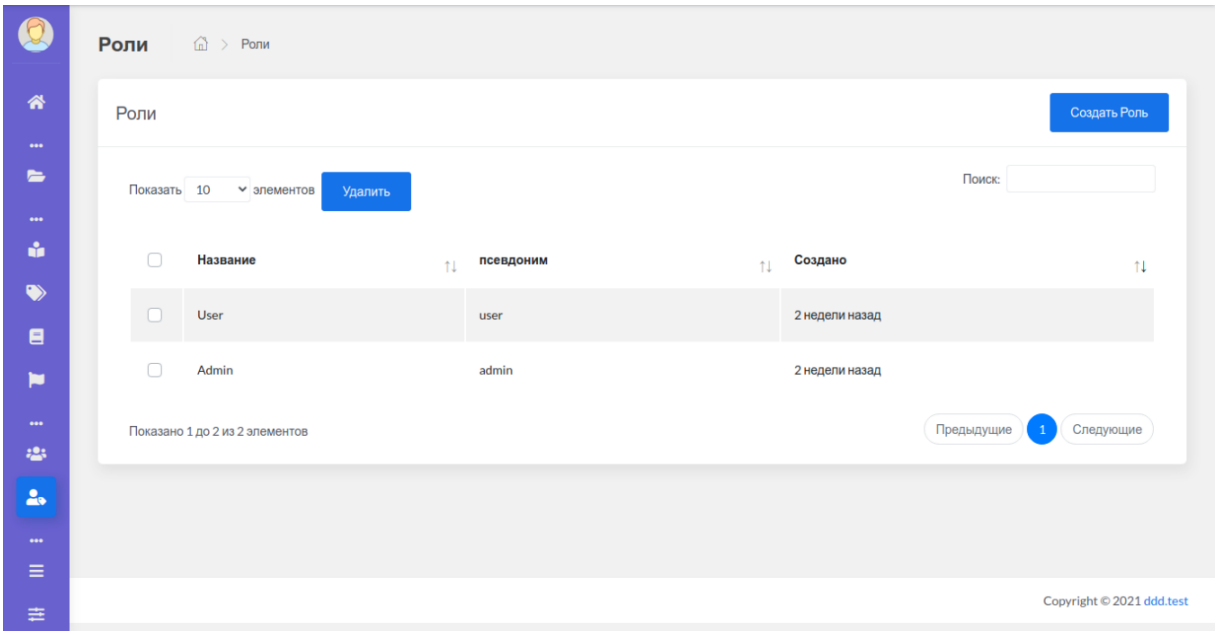

Рисунок 6 – Экран редактирования ролей системы

На странице создания каталогов (рисунок 7) можно реализовать дифференцированные разветвленные каталоги данных, которые потом вывести в меню визуального решения или же через API.

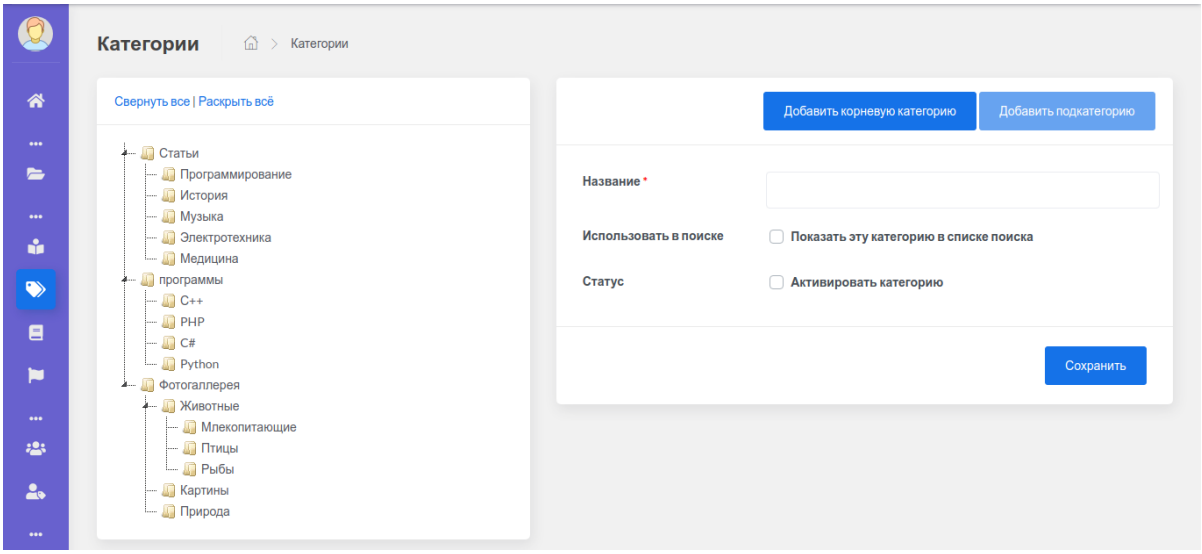

Рисунок 7 – Экран создания дифференцированных каталогов данных

Руководство пользователя

Каждый каталог может содержать неограниченное число ветвлений, в каждом из которых могут находится неограниченное количество объектов. Также есть возможность указать индексировать ли эту категорию в поиске и активировать ли её в целом на сайте в API. К каждому ответвлению созданных каталогов можно добавлять объекты различных данных, которые потом выводить на клиентской части приложения.

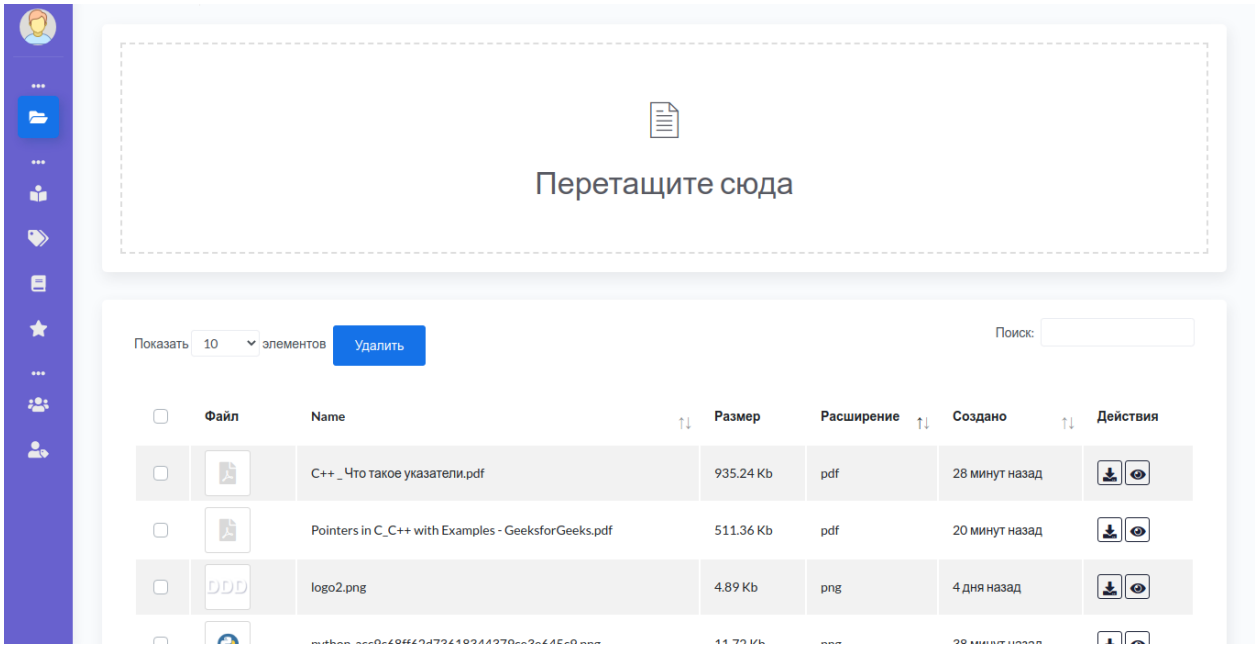

Рисунок 8 – Экран глобального добавления файлов

На рисунке 8 изображена страница, где можно глобально добавлять множество файлов, которые потом использовать в каталоге. Также данные можно добавлять непосредственно при создании объектов, так также может быть удобно.

Руководство пользователя

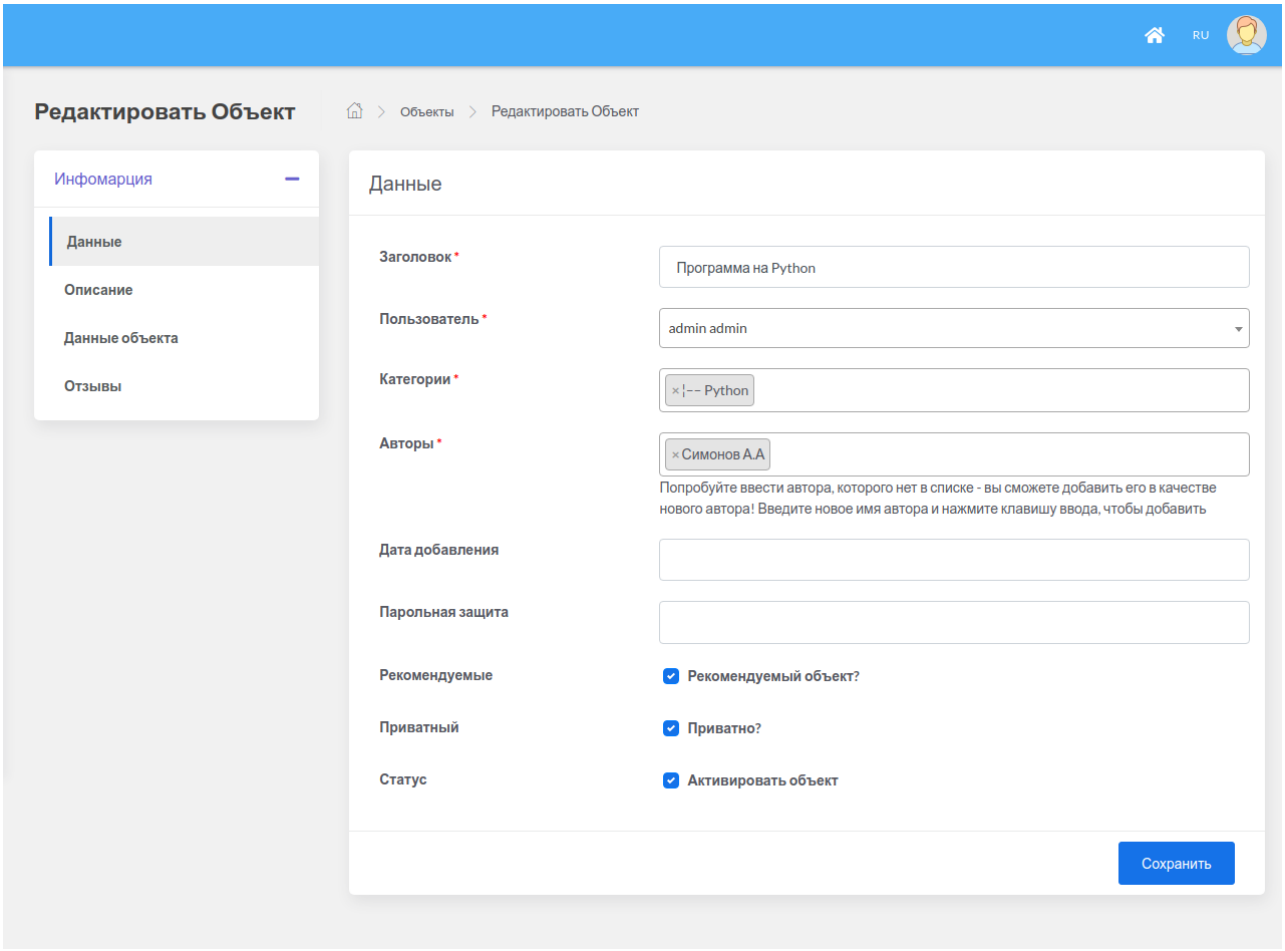

Рисунок 9 – Экран добавления и редактирования объектов

На данном экране можно заполнить различные данные об объекте, добавить к нему категорию и автора объекта (фотографии, видео, статьи, скрипта и др.), а также другие данные и некоторые настройки, как доступность и статус объекта.

Во вкладке «Данные объекта» (рисунок 10) можно добавить картинку для превью и файлы самого объекта.

Продолжение ПРИЛОЖЕНИЯ А

Руководство пользователя

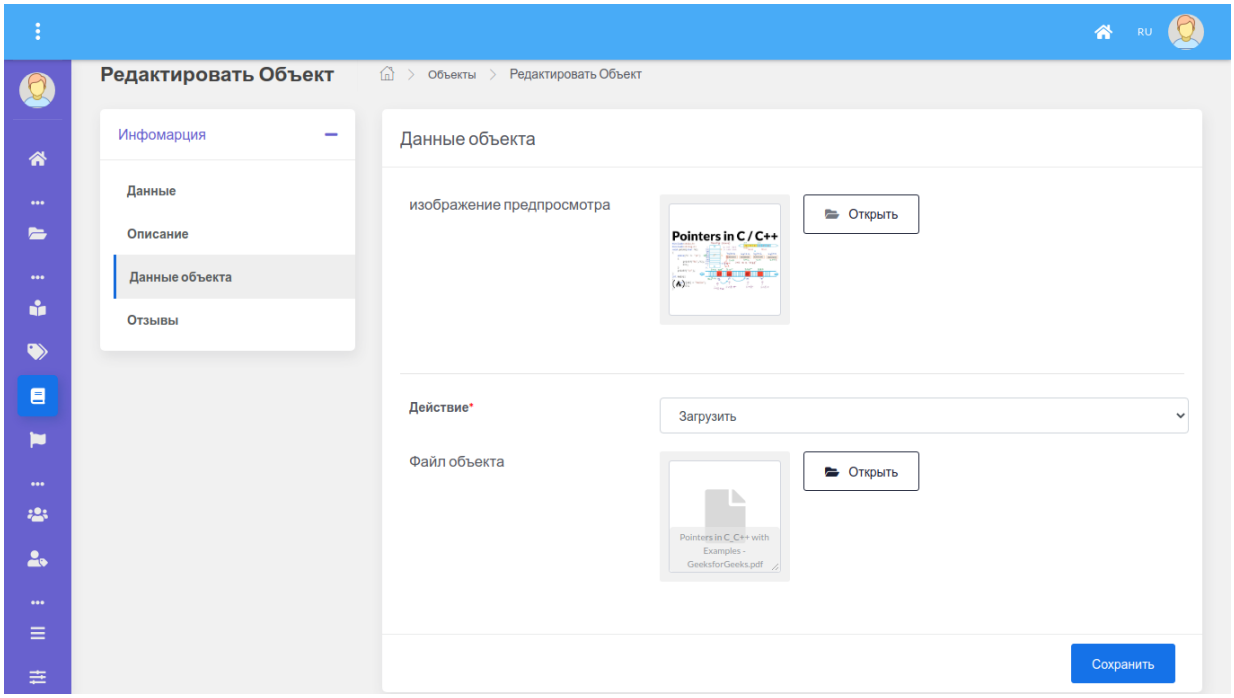

Рисунок 10 – Экран добавления файлов к объекту

Авторы для объектов являются самостоятельной сущностью и создаются отдельно (рисунок 11), чтобы избежать повторений и вести статистику.

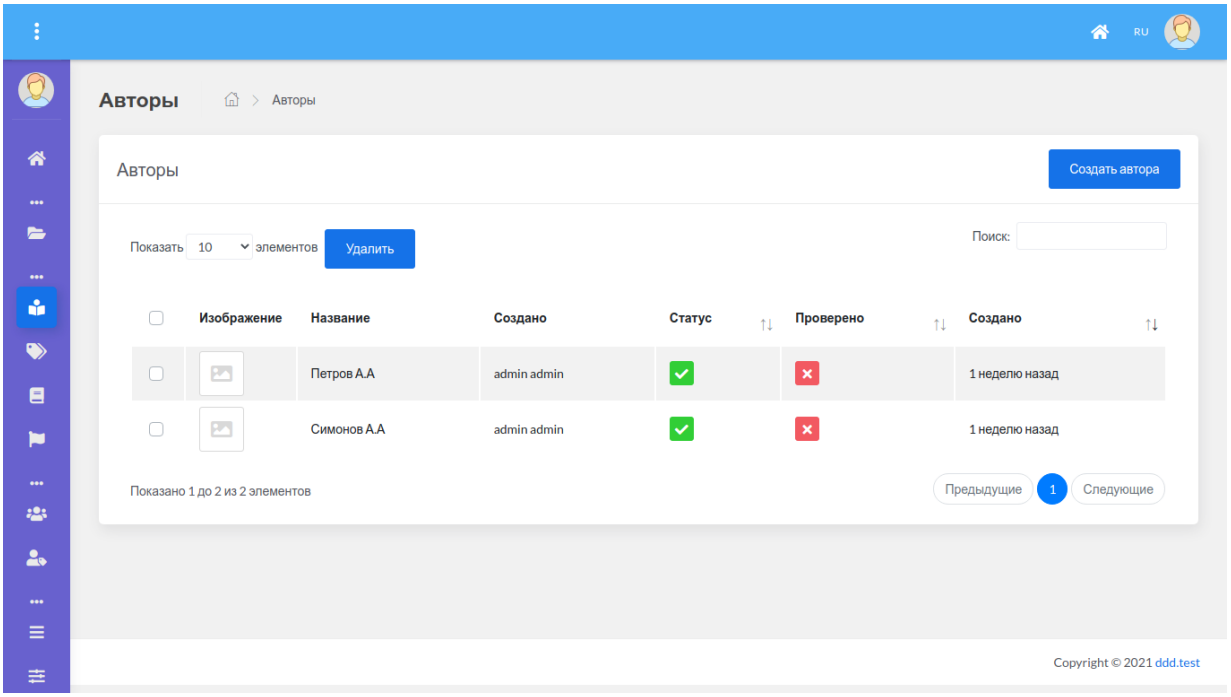

Рисунок 11 – Экран добавления и редактирования авторов

Продолжение ПРИЛОЖЕНИЯ А

Руководство пользователя

Клиентское приложение

Рассмотрим клиентское приложение

Главная страница

Главная страница (рисунок 12) состоит из баннера, представления каталогов, списка популярных объектов и формы для поиска с применением фильтрации. Возможно расширение блоков на главной странице.

Можно как ввести поисковой запрос, который выдаст ответ с помощью полнотекстового поиска, так и добавить дополнительные фильтры категорий при поиске.

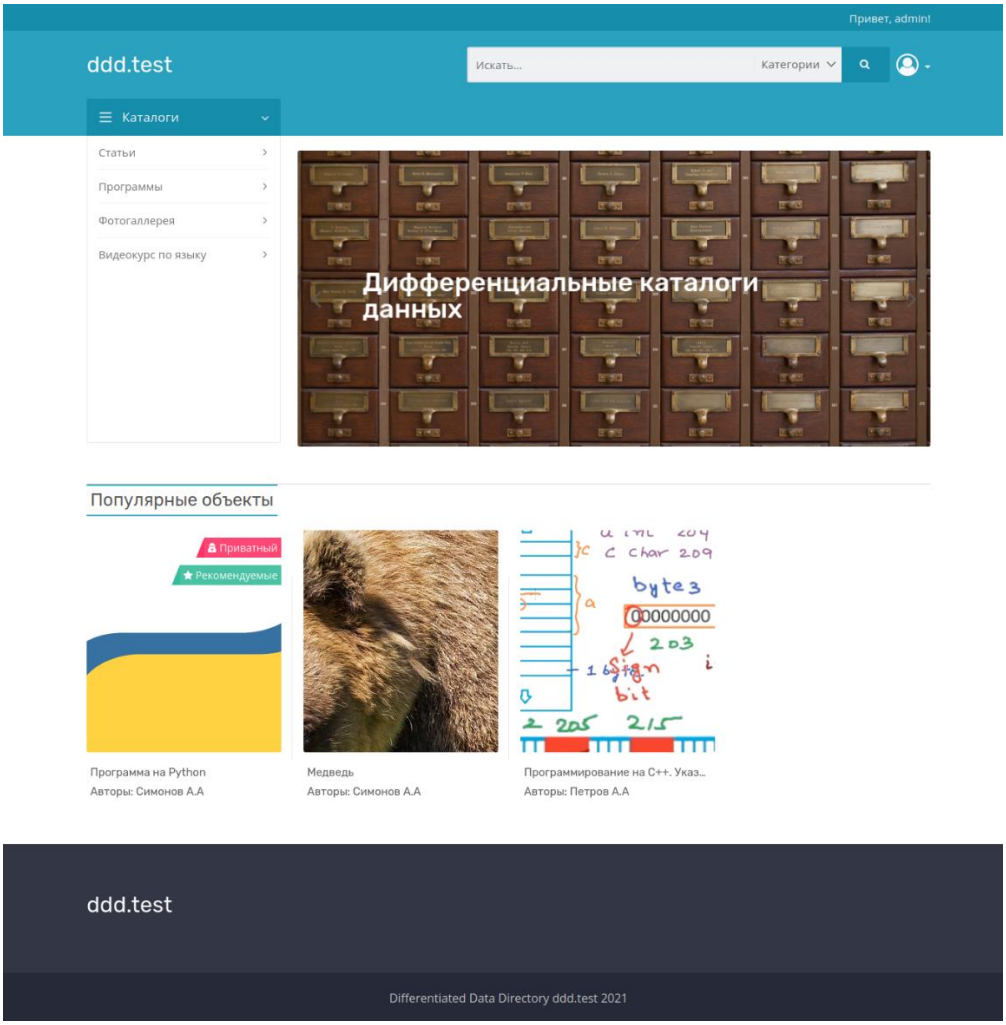

Рисунок 12 – Экран главной страницы клиентского приложения

Руководство пользователя

Пользователь может оценить объекты с помощью системы оценок(звезды), также добавить в избранное, если необходимо и заказать объект или купить онлайн, если такое предусмотрено.

Объекты могут постоянно обновляются по мере их добавления или изменения, также предусмотрена возможность ручной сортировки, если это необходимо, что позволяет вывести контент в любом порядке в нужных количествах в панели управления.

Созданные каталоги и подкаталоги выводятся в меню. Каждый элемент меню имеет ссылку на нужную ветвь каталога, перейдя по которой можно увидеть объекты, находящиеся в ней. Пользователю также доступно меню личного кабинета.

На рисунке ниже можно увидеть пример полнотекстового поискового запроса.

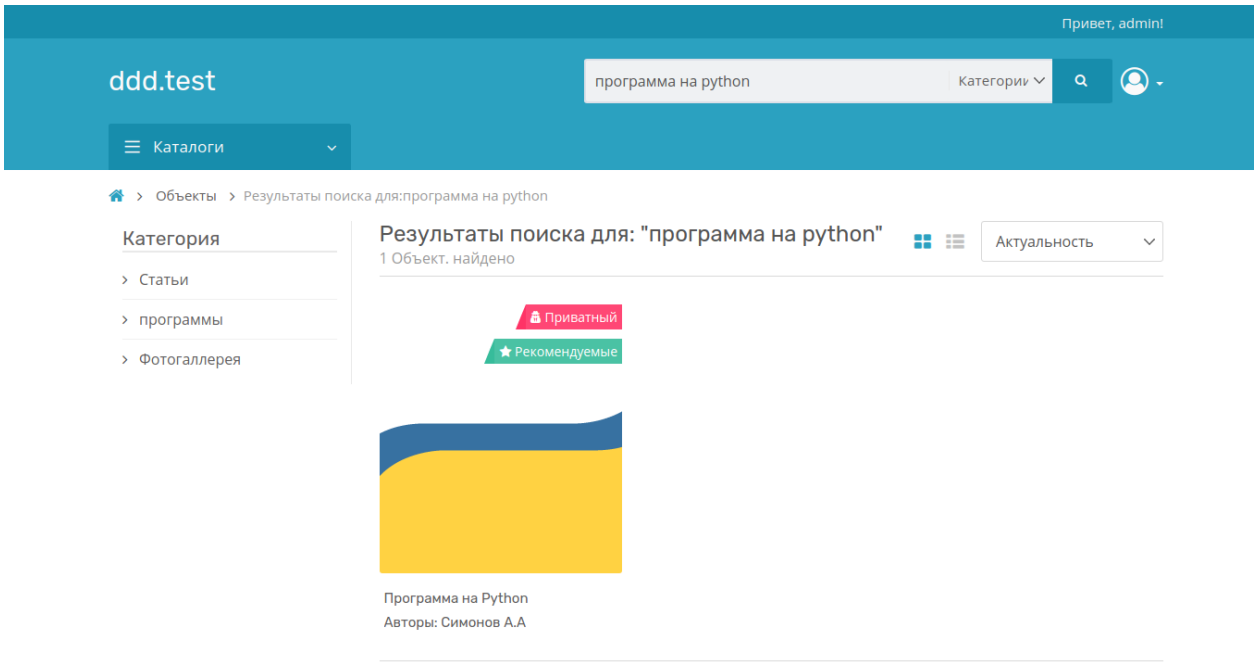

Рисунок 13 – Пример экрана результатов поиска

Продолжение ПРИЛОЖЕНИЯ А

Руководство пользователя

Детальная страница

Нажав на какой-либо объект, пользователь может увидеть страницу детального просмотра. Рассмотрим несколько примеров детальных страниц объектов различных типов с их описанием, возможностями по оценке, комментированию, скачиванию и представлению данных.

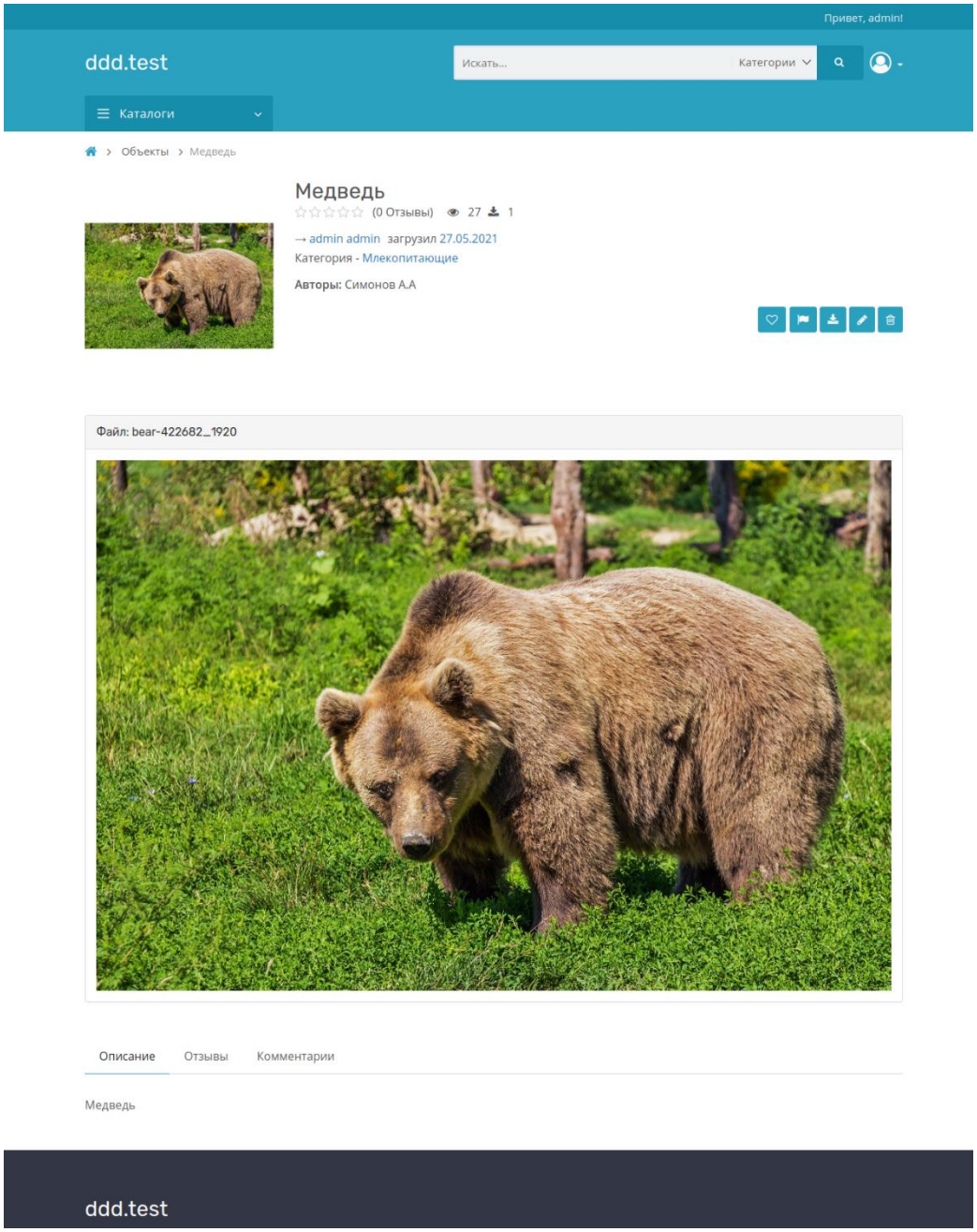

Рисунок 14 – Экран детальной страницы объекта типа «изображение»

Продолжение ПРИЛОЖЕНИЯ А

Руководство пользователя

Выше представлен пример детальной страницы изображения в формате png.

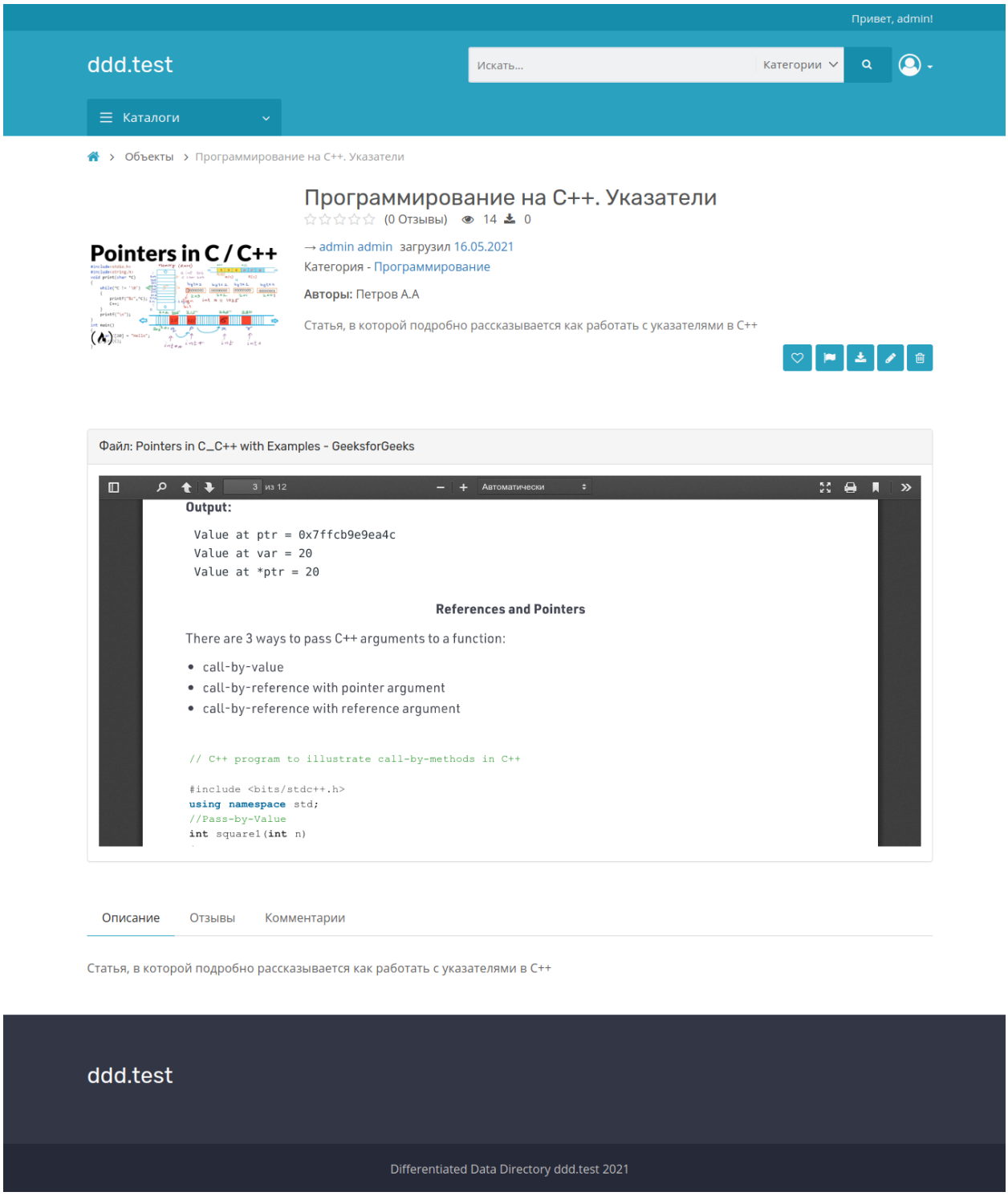

Рисунок 15 – Экран детальной страницы объекта типа «статья»

Продолжение ПРИЛОЖЕНИЯ А

Руководство пользователя

На рисунке 15 представлен пример детальной страницы документа в формате pdf в удобном для пользователя виде.

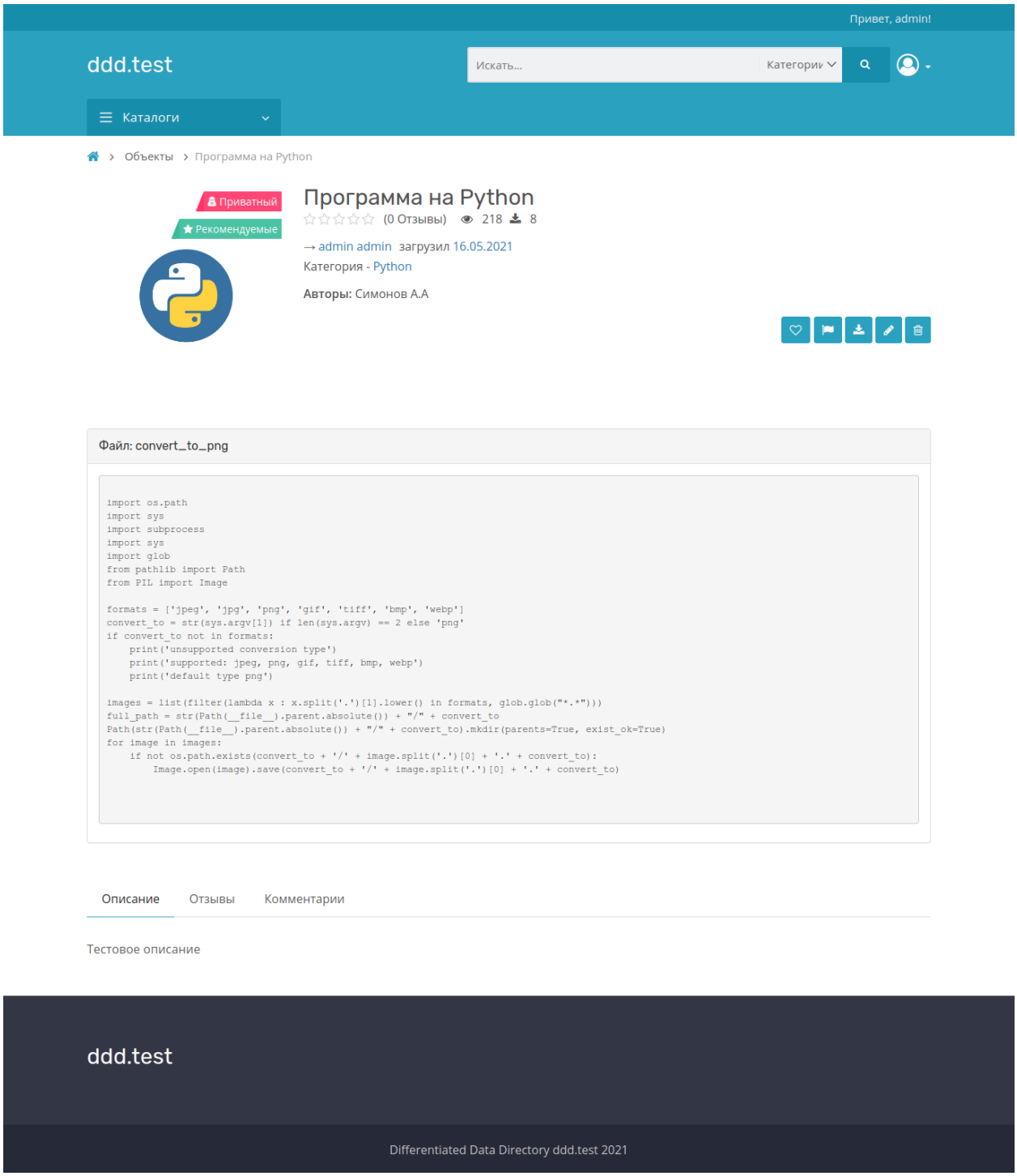

Рисунок 16 – Экран детальной страницы объекта типа «программный код»

Как видно из рисунков, детальные страницы объектов (изображение, статья, скрипта с программным кодом) соответствуют своим типам. На данной странице можно увидеть обложку объекта, поставить оценку, поделиться, добавить в избранное, скачать, а также отредактировать или удалить, нажав на соответствующие кнопки. Также можно оставить комментарий, отзыв и прочитать краткое и полное описание, узнать статистику по просмотрам и скачиваниям. Так, например, статья в формате .pdf выводится в удобном окне просмотра, фотография подстраивается под размер экрана, а программный код выводится в специальных тегах, предназначенных для этого.