**СОДЕРЖАНИЕ**

[Введение 8](#_Toc9214986)

[1 Анализ литературных источников и существующих аналогов 10](#_Toc9214987)

[1.1 Веб-сервисы и особенности их применения 10](#_Toc9214988)

[1.2 Обзор существующих программ 13](#_Toc9214989)

[2 Моделирование предметной области и разработка функциональных требований 20](#_Toc9214990)

[2.1 Постановка задачи и функциональные требования проекта 20](#_Toc9214991)

[2.2 Моделирование предметной области 21](#_Toc9214992)

[3 Проектирование веб-сервиса 24](#_Toc9214993)

[3.1 Общая структура приложения 24](#_Toc9214994)

[3.2 Основные модули приложения 26](#_Toc9214995)

[3.3 Проектирование схемы базы данных 29](#_Toc9214996)

[3.4 Распределенное хранилище данных 32](#_Toc9214997)

[4 Разработка веб-сервиса 38](#_Toc9214998)

[4.1 Выбор и обоснование технологий реализации 38](#_Toc9214999)

[4.2 Сторонние библиотеки 50](#_Toc9215000)

[4.2 Алгоритмы работы веб-сервиса 52](#_Toc9215001)

[4.3 Особенности реализации веб-сервиса 57](#_Toc9215002)

[5 Тестирование веб-сервиса 59](#_Toc9215003)

[6 Руководство пользователя 61](#_Toc9215004)

[6.1 Системные требования и установка веб-сервиса 61](#_Toc9215005)

[6.2 Описание пользовательского интерфейса 62](#_Toc9215006)

[6.3 Регистрация 63](#_Toc9215007)

[6.4 Аутентификация 64](#_Toc9215008)

[6.5 Управление вопросами 65](#_Toc9215009)

[6.6 Управление ответами 67](#_Toc9215010)

[6.7 Создание категории 68](#_Toc9215011)

[6.8 Поиск возможного ответа 70](#_Toc9215012)

[7 Технико-экономическое обоснование эффективности разработки и реализации веб-сервиса вопросов и ответов 71](#_Toc9215013)

[7.1 Характеристика программного продукта 71](#_Toc9215014)

[7.2 Расчёт сметы затрат и цены веб-сервиса 71](#_Toc9215015)

[7.3 Расчёт экономического эффекта от применения веб-сервиса у пользователя 81](#_Toc9215016)

[7.4 Вывод по технико-экономическому обоснованию 86](#_Toc9215017)

[Заключение 88](#_Toc9215018)

[Список использованных источников 89](#_Toc9215019)

[Приложение А 91](#_Toc9215020)

[Приложение Б 93](#_Toc9215021)

# **ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ**

ПС – Программное средство;

XML – Extensible Markup Language;

HTML – Hypertext Markup Language;

SOAP – Simple Object Access Protocol;

WSDL – Web Services Description Language;

UDDI – Universal Description, Discovery and Integrationn;

HTTP – HyperText Transfer Protocol;

БД – База данных;

СУБД – Система управления базами данных;

ПП – Программный продукт;

ПО – Программное обеспечение;

MVC – Model View Controller;

ОС – Операционная система;

MD5 – Message Digest 5.

# **ВВЕДЕНИЕ**

В современном мире почти у каждого человека есть доступ в Интернет. Все пользователи сегодня ищут информацию в глобальной сети. Наиболее популярной во всем мире поисковой системой является Google, которая ежедневно обрабатывает миллионы запросов пользователей.

Как только у человека появляется какой-то вопрос, он обращается к поисковым системам, чтобы найти ответ на свой вопрос. Но в поисковых системах довольно много бесполезной информации, на просмотр которой приходится тратить большое количество времени. Более того, если веб-сайт закрыт от индексации, либо страницы с нужными вопросами еще не были проиндексированы при помощи поискового робота, то значительно много информации будет недоступно пользователю в момент поиска.

Не всегда целесообразно полагаться на поисковые системы, так как они часто полагаются на предыдущие запросы пользователя. И если пользователь задает совершенно новый вопрос, поисковая система не может предложить решение проблемы, а чаще всего просто выводится большое количество нерелеватной информации, на прочтение которой пользователи тратят большое количество времени. В итоге пользователи переходят на специализированные сервисы вопросов и ответов, но и они имеют ряд недостатков.

Довольно популярны сервиса вопросов и ответов, где пользователи задают вопросы, а другие на них отвечают. Но при этом каждый раз, когда вы хотите получить ответ, приходится ждать, пока пользователи ответят вам. Но если носителей информации с какими-либо специфическими знаниями нет в сети, вы можете довольно долго ждать ответ. Более того, чтобы ваш вопрос был замечен, часто необходимо, чтобы он был проиндексирован поисковыми роботами. Только после этого потенциальные носители знаний смогут помочь пользователю в решении его вопроса.

Более того, современные сервисы довольно узконаправлены, для примера система вопросов ответов для программистов. Поэтому необходимо иметь учетную запись на каждом сервисе и помнить логины и пароли от каждого, когда вы хотите задать вопрос на определенном сервисе. Пользователям проще иметь один аккаунт для поиска ответов на свои вопросы. Так как существует большое количество небольших сервисов, количество посетителей на большей части из них небольшое. Поэтому вам придется ждать ответ на свой вопрос после его задания много времени.

Если тема вопроса интересна небольшому количеству пользователей, ответ можно ждать около одного месяца. Поисковые системы ранжируют сервисы с низкой популярностью ниже остальных, поэтому в некоторых случаях носители нужной вам информации не смогут найти такой сервис вопросов и ответов.

Ежедневно на подобных сервисах пользователи задают тысячи повторяющихся вопросов. Существующие аналогичные сервисы не анализируют вопрос пользователя перед его публикацией. Соответственно, в таких довольно много повторяющихся пар ответов/вопросов.

Число данных растет каждый день, поэтому необходимо средство для структурирования и хранения больших объемов ежедневно создаваемых вопросов и ответов.

Существующие решения в полной мере не удовлетворяют запросам пользователей. Они не предлагают функционал для поиска только что заданного по уже существующим парам вопросов и ответов.

Все данные пользователя должны храниться в веб-сервисе, чтобы пользователь мог получить доступ к базе знаний с любого устройства, поддерживающего выход в интернет.

Целью данного дипломного проекта является разработка веб-сервиса вопросов и ответов для быстрого автоматического поиска ответов на вопросы пользователя. После задания вопроса, система должна осуществить поиск по уже существующим вопросам и предложить готовый ответ. Если ответ действительно верный, пользователь может не публиковать свой вопрос, а просто сообщить об этом системе. С каждой новой парой вопрос-ответ точность системы должна только повышаться.

Поэтому тема проекта является актуальной.

# **1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И СУЩЕСТВУЮЩИХ АНАЛОГОВ**

## **1.1 Веб-сервисы и особенности их применения**

Для реализации приложения было решено использовать веб-сервис. Веб-сервис – идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами, а также HTML-документ сайта, отображаемый браузером пользователя.

Веб-сервисы [1] могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (SOAP, XML-RPC) и соглашениях (REST). Веб-служба является единицей модульности при использовании сервис-ориентированной архитектуры приложения.

Веб-сервисами называют услуги, оказываемые в Интернете. В этом употреблении термин требует уточнения, идёт ли речь о поиске, веб-почте, хранении документов, файлов, закладок и т. п. Такими веб-сервисами можно пользоваться независимо от компьютера, браузера или места доступа в Интернет.

Когда служба разработана, исполнитель регистрирует её в каталоге, где её могут найти потенциальные заказчики. Заказчик, найдя в каталоге подходящую службу, импортирует оттуда её WSDL-спецификацию и разрабатывает в соответствии с ней своё программное обеспечение. WSDL описывает формат запросов и ответов, которыми обмениваются заказчик и исполнитель в процессе работы. Для обеспечения взаимодействия используются следующие стандарты: XML, SOAP, WSDL, UDDI, JSON.

XML это расширяемый язык разметки, предназначенный для хранения и передачи структурированных данных. SOAP это протокол обмена сообщениями на базе XML.

WSDL это язык описания внешних интерфейсов веб-службы на базе XML.

UDDI это универсальный интерфейс распознавания, описания и интеграции (Universal Discovery, Description and Integration). Каталог веб-служб и сведений о компаниях, предоставляющих веб-службы во всеобщее пользование или конкретным компаниям. Пока UDDI существуют, однако, только в небольших фирменных сетях и ещё не нашли широкого распространения в открытом интернете. JSON более эффективный язык разметки, ставший массовым в 2010х годах.

Существуют средства автоматизации разработки веб-служб, разделяющиеся на две основные группы. При разработке снизу-вверх сначала пишутся имплементирующие классы, а из их исходного текста генерируются WSDL-файлы, документирующие службу. Недостатком этого метода является подверженность Java-классов частым изменениям. При подходе сверху-вниз сначала подготавливается WSDL, а из него генерируется скелет Java-класса, имплементирующего службу. Этот путь считается более трудным, зато приводит к более чистым и лучше защищенным от изменений решениям. Пока формат сообщений, которыми обмениваются заказчик и исполнитель, не меняется, изменения в каждом из них не нарушают взаимодействия. Эта техника называется иногда «contract first», так как исходной точкой является WSDL («договор» между заказчиком и исполнителем).

Веб-службы обеспечивают взаимодействие программных систем независимо от платформы. Например, Windows-C#-клиент может обмениваться данными с Java-сервером, работающим под Linux. Веб-службы основаны на базе открытых стандартов и протоколов. Благодаря использованию XML достигается простота разработки и отладки веб-служб. Использование интернет-протокола обеспечивает HTTP-взаимодействие программных систем через межсетевой экран. Это значительное преимущество, по сравнению с такими технологиями, как CORBA, DCOM или Java RMI. С другой стороны, веб-службы не привязаны намертво к HTTP – могут использоваться и другие протоколы.

Меньшая производительность и больший размер сетевого трафика по сравнению с технологиями RMI, CORBA, DCOM за счёт использования текстовых XML-сообщений. Однако на некоторых веб-серверах возможна настройка сжатия сетевого трафика. Аспекты безопасности. Ответственные веб-службы должны использовать кодирование, возможно — требовать аутентификации пользователя. Достаточно ли здесь применения HTTPS, или предпочтительны такие решения, как XML Signature, XML Encryption или SAML, должно быть решено разработчиком.

Прежде всего, веб-сервисы [2] (или веб-службы) – это технология. Как и любая другая технология, они имеют довольно четко очерченную среду применения.

Если посмотреть на веб-сервисы в разрезе стека сетевых протоколов, мы увидим, что это, в классическом случае, не что иное, как еще одна надстройка поверх протокола HTTP.

С другой стороны, если гипотетически разделить Интернет на несколько слоев, мы сможем выделить, как минимум, два концептуальных типа приложений – вычислительные узлы, которые реализуют нетривиальные функции и прикладные веб-ресурсы. При этом вторые, зачастую заинтересованы в услугах первых.

Но и сам Интернет разнороден, т. е. различные приложения на различных узлах сети функционируют на разных аппаратно-программных платформах, и используют различные технологии и языки.

Чтобы связать все это и предоставить возможность одним приложениям обмениваться данными с другими, и были придуманы веб-сервисы.

По сути, веб-сервисы – это реализация абсолютно четких интерфейсов обмена данными между различными приложениями, которые написаны не только на разных языках, но и распределены на разных узлах сети.

Именно с появлением веб-сервисов развилась идея SOA — сервис-ориентированной архитектуры веб-приложений (Service Oriented Architecture).

На самом деле, SOAP произошел от XML-RPC и является следующей ступенью его развития. В то время как REST – это концепция, в основе которой лежит скорее архитектурный стиль, нежели новая технология, основанный на теории манипуляции объектами CRUD (Create Read Update Delete) в контексте концепций WWW.

Безусловно, существуют и иные протоколы, но, поскольку они не получили широкого распространения, мы остановимся в этом кратком обзоре на двух основных – SOAP и REST. XML-RPC ввиду того, что является несколько «устаревшим», мы рассматривать подробно не будем.

Нас в первую очередь интересуют вопросы создания новых веб-служб, а не реализация клиентов к существующим (как правило поставщики веб-сервисов поставляют пакеты с функциями API и документацией, посему вопрос построения клиентов к существующим веб-службам менее интересен с точки зрения автора).

SOAP более применим в сложных архитектурах, где взаимодействие с объектами выходит за рамки теории CRUD, а вот в тех приложениях, которые не покидают рамки данной теории, вполне применимым может оказаться именно REST ввиду своей простоты и прозрачности. Действительно, если любым объектам вашего сервиса не нужны более сложные взаимоотношения, кроме: «Создать», «Прочитать», «Изменить», «Удалить» (как правило – в 99% случаев этого достаточно), возможно, именно REST станет правильным выбором. Кроме того, REST по сравнению с SOAP, может оказаться и более производительным, так как не требует затрат на разбор сложных XML команд на сервере (выполняются обычные HTTP запросы – PUT, GET, POST, DELETE). Хотя SOAP, в свою очередь, более надежен и безопасен.

Уже невозможно представить, как предыдущие поколения жили без интернета. Чтобы оплатить квитанцию, нужно было простоять долгое время в очереди в банке. Для перевода текста приходилось доставать тяжелый словарь, а иногда и вовсе идти за ним в библиотеку. Список покупок был ограничен тем, что завозили в магазины города. Сейчас же можно и счета оплатить, и перевод сделать, и купить, что только ни захочется, - и все это с помощью интернета и многочисленных онлайн сервисов.

Онлайн сервисы [3] – это сайты, предоставляющие услуги различного характера. Их первостепенная задача – облегчить работу и значительно сэкономить время. Подобные сайты успешно заменяют компьютерные программы, которые зачастую стоят дорого и требуют специальной настройки. С их помощью в интернете можно делать почти все – проводить денежные операции, узнавать последние новости, общаться, хранить информацию, редактировать ее, пересылать и публиковать. Кстати, нередко в интернете встречаются полностью бесплатные онлайн сервисы, которые не требуют оплат за свои услуги!

## **1.2 Обзор существующих программ**

На текущий момент существует множество решений, позволяющих пользователям размещать вопросы и искать на них ответы.

К наиболее популярным из них относятся: Ответы Mail.Ru, Quora, Toster, ASKfm, Stack Overflow, Askee, TheQuestion.

**1.2.1** Ответы Mail.ru

Сервис вопросов и ответов, разработанный компанией Mail.ru [4]. Доступен для бесплатного использования всем пользователя интернета. Запуская «Ответы», компания называла их социальным поисковиком, способным компенсировать неэффективность машинного поиска и неточности в формулировке запроса. На момент открытия у Mail.ru было 20 миллионов активных учётных записей и несколько миллионов активных пользователей сервисов Блоги@Mail.Ru и Фото@Mail.Ru. Благодаря этой аудитории планировалось сделать из сервиса «крупнейшую базу знаний в Рунете». На рисунке 1.1 показано добавление нового вопроса.

В основу сервиса с самого начала была заложена соревновательная модель, когда наиболее активные и «полезные» пользователи оказывались выше в рейтинге участников.

Ключевая метрика – КПД – это отношение лучших ответов к общему числу ответов (лучший ответ выбирается автором вопроса или по результатам голосования). От него зависит число полученных за ответ баллов: чем выше КПД участника, тем больше баллов начисляется. За вопросы и нарушение правил система вычитает баллы. С достижением определённого числа баллов пользователю присваивается новый статус (от «новичка» до «высшего разума»), расширяющий лимит на число вопросов, ответов и голосований за один день.

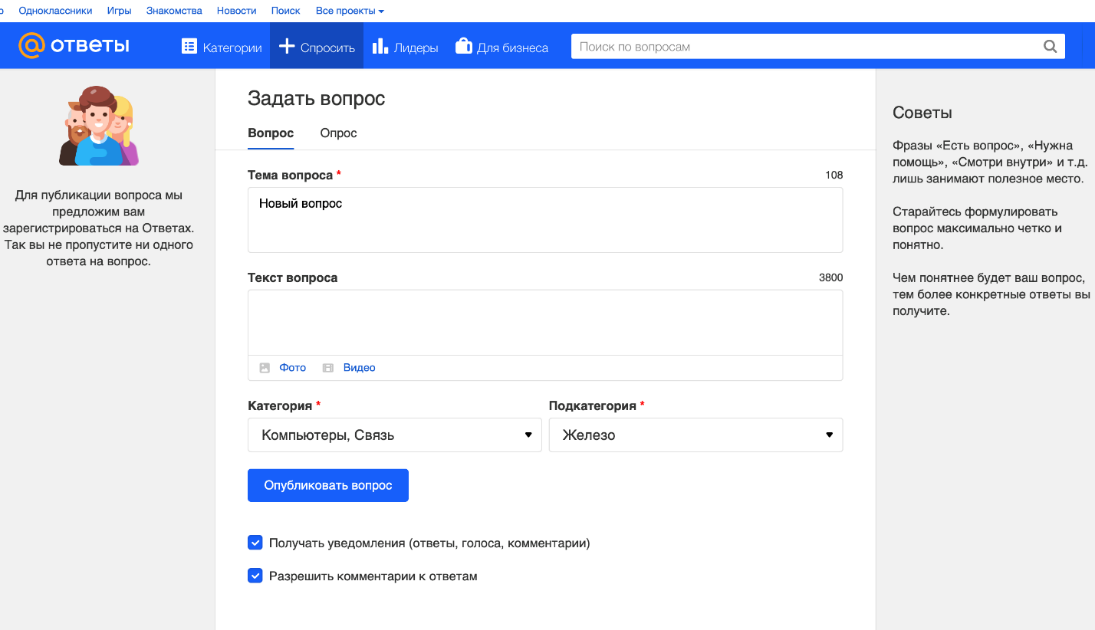


Рисунок 1.1 – Добавление нового вопроса на Ответы Mail.ru

**1.2.2** Quora

Quora – это сайт вопросов и ответов [5], где вопросы задаются, отвечаются, редактируются и организуются сообществом пользователей в форме мнений. Его издатель, Quora Inc. Сервис впервые стал доступен для общественности 21 июня 2010 года. Пользователи могут сотрудничать, редактируя вопросы и предлагая изменения ответов, представленные другими пользователями. Главная страница сервиса показана на рисунке 1.2.

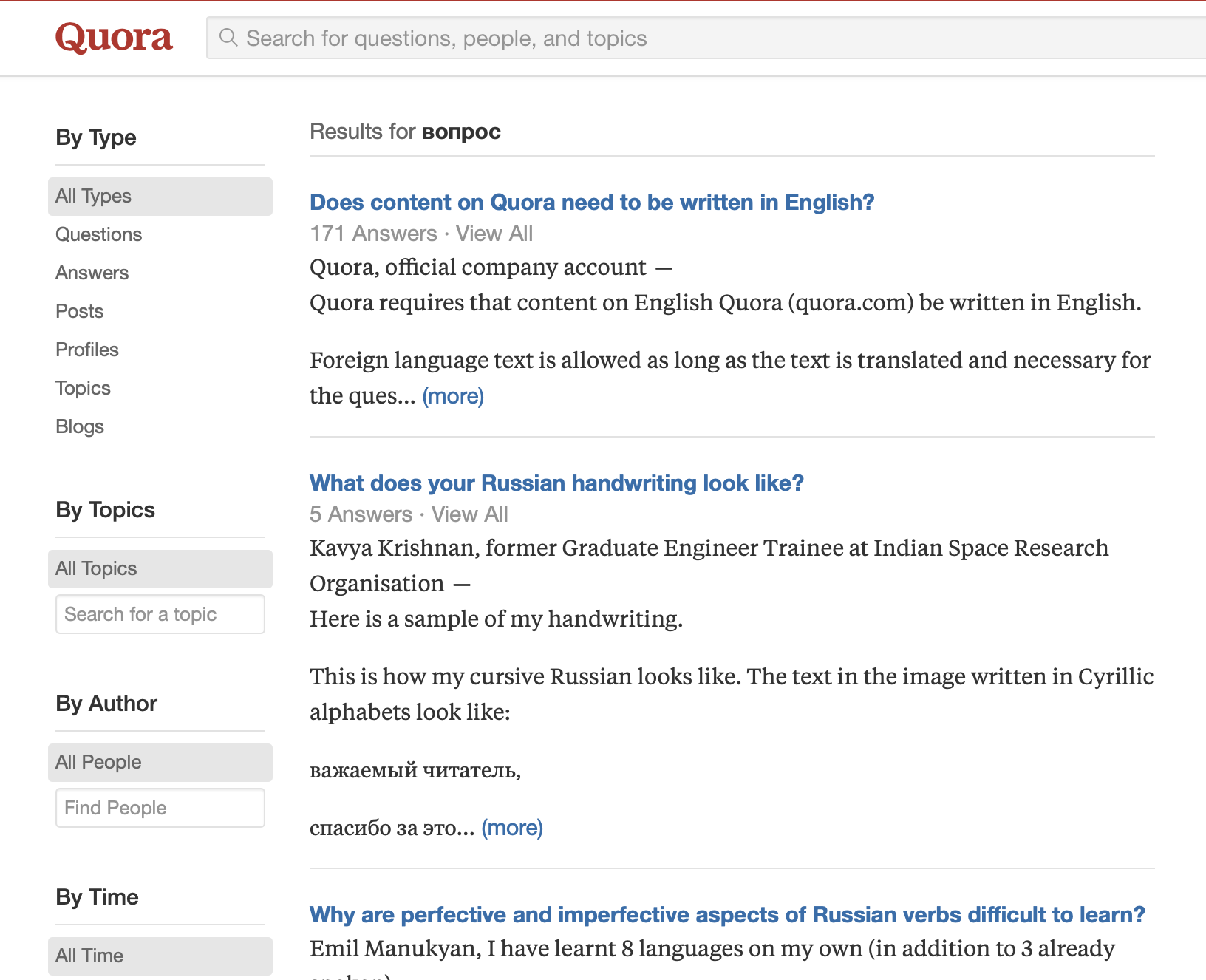
Quora требует, чтобы пользователи регистрировались с полной формой своих настоящих имен, а не с псевдонимом в интернете. Хотя проверка имен не требуется, ложные имена могут быть сообщены сообществом. Это было сделано с явным намерением добавить достоверности ответам. Пользователи также могут войти в систему со своими учетными записями Google или Facebook, используя протокол OpenID. Они могут повышать или понижать ответы и предлагать изменения для существующих ответов, предоставляемых другими пользователями. Сообщество Quora включает в себя некоторых известных людей, таких как Джимми Уэйлс, Ричард А. Мюллер, Джастин Трюдо, Барак Обама, Хиллари Клинтон и покойный Адриан Ламо, а также многие нынешние и бывшие профессиональные спортивные личности.

Рисунок 1.2 – Сервис вопросов и ответов Quora

Пользователи с определенной активностью на сайте имеют возможность писать свои ответы анонимно, но не по умолчанию. Посетители, не желающие входить в систему или использовать куки, вынуждены были прибегать к обходным путям для использования сайта.

**1.2.3** Toster

Toster – сервис вопросов и ответов для IT специалистов [6]. На «Тостере» можно получить ответ на вопрос по любой теме IT от участников сообщества, хорошо разбирающихся в этой теме. Главная страница показана на рисунке 1.3.

Всё взаимодействие на «Тостере» строится при помощи двух типов публикаций: вопросов и ответов на них. На конкретный вопрос каждый участник сообщества может дать только один ответ.

Получить ответ можно двумя способами: самому задать вопрос, либо найти уже существующий и подписаться на него. В обоих случаях вам на почту или в уведомления на сайте будут приходить уведомления о новых ответах. Участники сообщества отмечают хорошие ответы кнопкой «Нравится», и те поднимаются вверх в списке ответов. Автор вопроса может также пометить ответ «решением»: это будет означать, что ответ ему лично пригодился.

Уточнить вопрос или ответ можно в комментариях. Если в переписке появляется много дополнительной информации, хорошим тоном считается отредактировать свой вопрос или ответ и дополнить их этой информацией.

Все вопросы на «Тостере» привязываются к тегам, обозначающим предметную область, к которой вопрос относится. На любой тег можно подписаться и получать уведомления о вопросах, появляющихся по этой теме.

В разделе «Все теги» находится список всех существующих на сайте тем, среди которых можно найти интересующие и подписаться. Также любой тег можно найти с помощью глобального поиска по сайту.

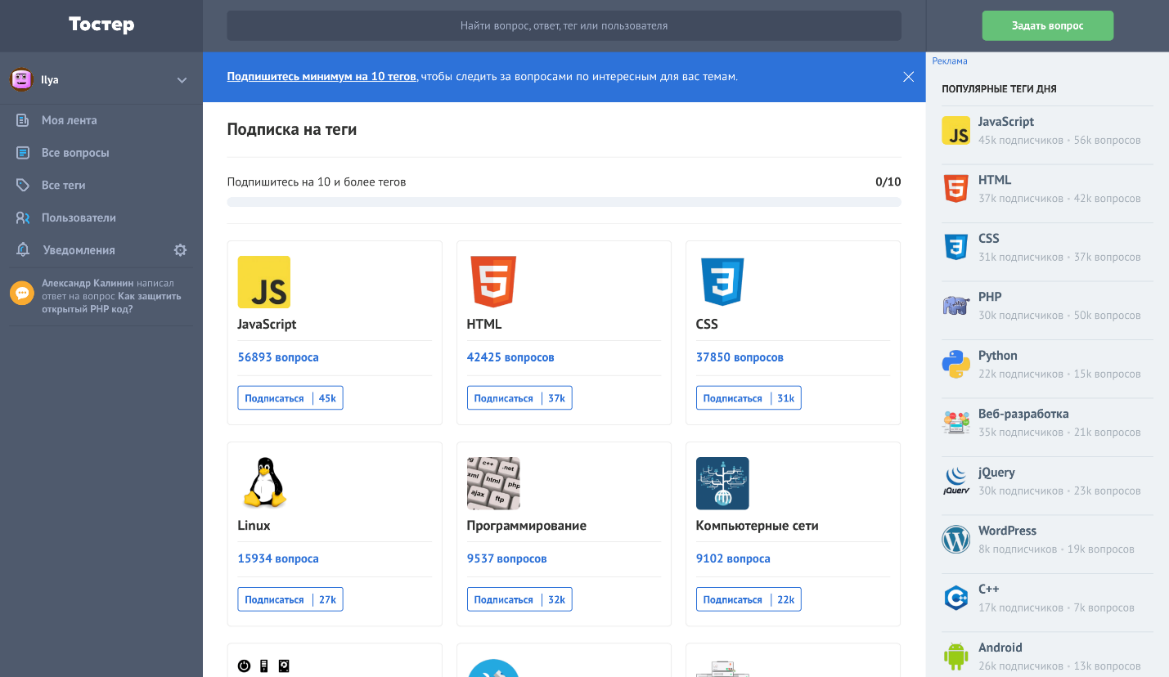


Рисунок 1.3 – Сервис вопросов и ответов Toster

Как только в тегах, на которые вы подписаны, добавляются новые вопросы, они появляются в разделе «Моя лента». Кроме того, вы можете получать еженедельный дайджест самых интересных вопросов по вашим тегам. Помогайте другим и зарабатывайте репутацию

Всякий раз, когда кто-то подписывается на ваш вопрос, или кому-то нравится ваш ответ, или кто-то отмечает ваш ответ решением, вы зарабатываете дополнительную репутацию. Репутация зарабатывается исключительно по тем тегам, к которым привязан вопрос, который вы задаёте или на который отвечаете. Репутация на «Тостере» называется «вкладом».

Свой вклад по каждому тегу можно увидеть в своём профиле. Видны только те теги, на которые вы подписаны. На странице каждого тега можно увидеть всех участников, внёсших наибольший вклад в этот тег. Чтобы попасть в этот список, нужно быть подписанным на данный тег.

**1.2.4** ASKfm

ASKfm – сеть вопросов и ответов, запущенная в июне 2010 года [7]. Сайт создан в Латвии как конкурент Formspring. После регистрации пользователь заполняет свою анкету и может начать, как от своего имени, так и анонимно задавать и отвечать на вопросы других пользователей. ASKfm интегрирована с другими социальными сетями, такими как Facebook, Twitter, ВКонтакте и Tumblr, которые повлияли на широкое распространение этой сети. Главная страница показана на рисунке 1.4.

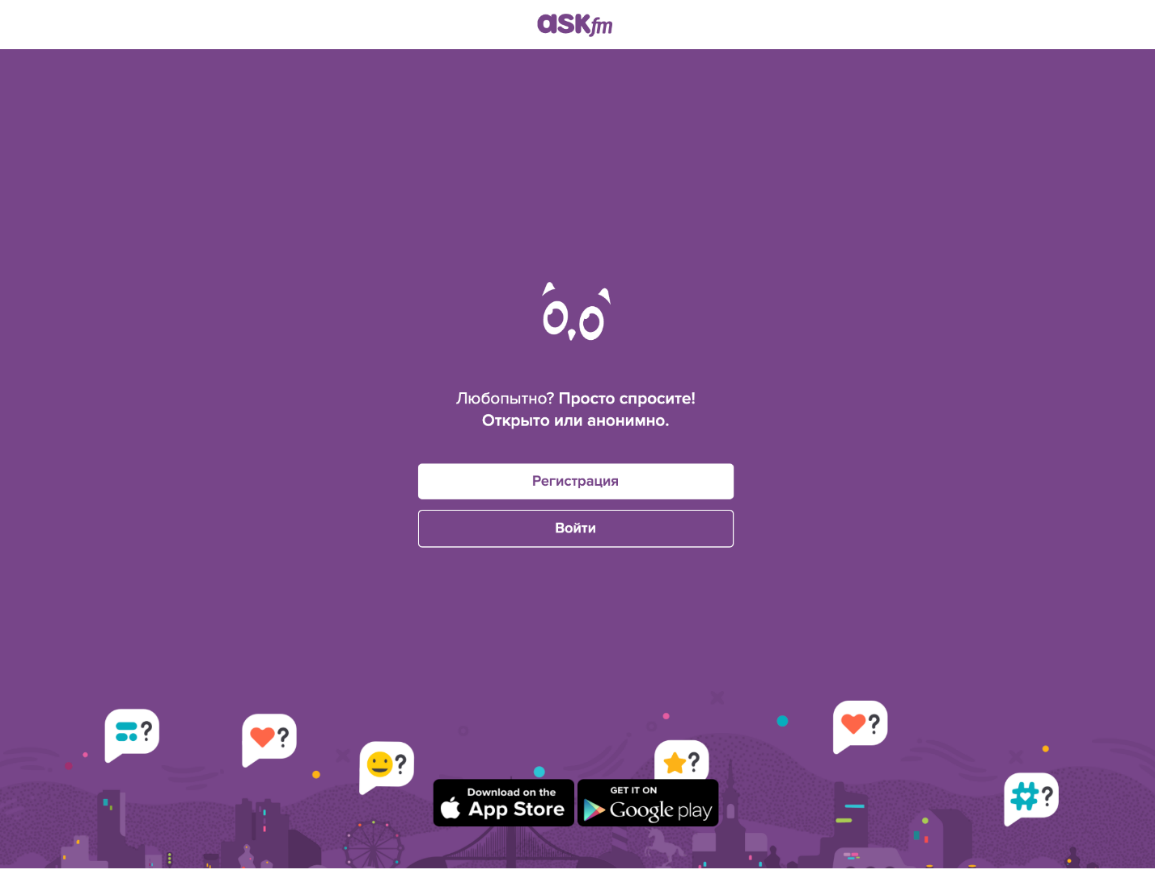


Рисунок 1.4 – Главная страница сервиса ASKfm

Сеть основана в Латвии в июне 2010 года в качестве конкурента Formspring. С тех пор сервис обогнал конкурента и является очень популярной сетью в России и некоторых других странах. Занимая 148 место в мире по версии Alexa Internet. При этом с 3 марта 2010 года уже развивался другой аналогичный проект для русскоязычных пользователей: Спрашивай.ру.

**1.2.5** Stack Overflow

Stack Overflow – популярная система вопросов и ответов [8] о программировании, разработанная Джоэлем Спольски и Джеффом Этвудом в 2008 году. Является частью Stack Exchange Network. Как и в других системах подобного рода, Stack Overflow предоставляет возможность оценивать вопросы и ответы, что поднимает или понижает репутацию зарегистрированных пользователей (вариант игрофикации). Проект создан с использованием ASP.NET 4 и ASP.NET MVC. Главная страница показана на рисунке 1.5.

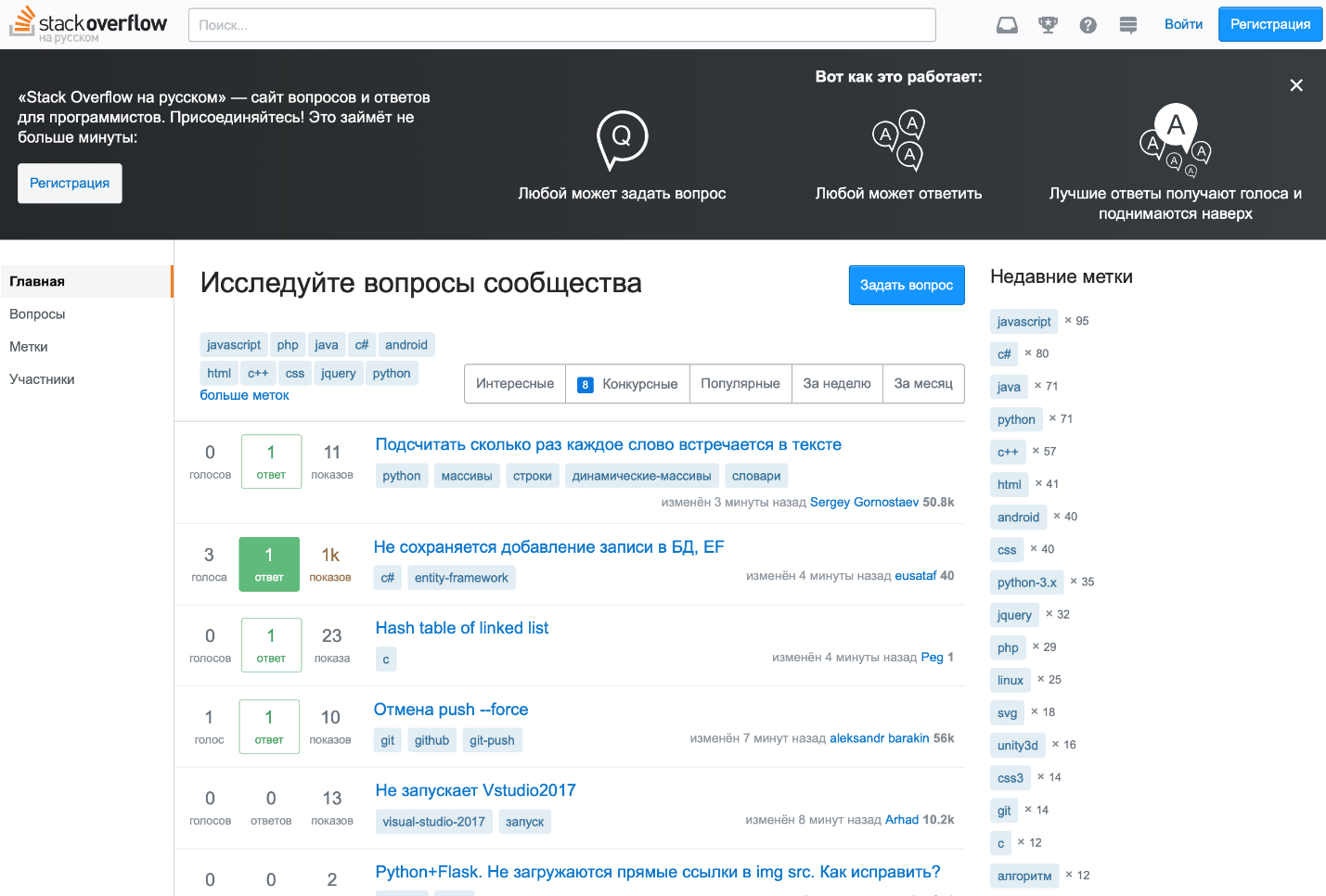


Рисунок 1.5 – Главная страница сервиса Stack Overflow

В течение долгого времени Stack Overflow был ресурсом, использующим исключительно английский язык. Сегодня, помимо англоязычного сообщества Stack Overflow, в сеть Stack Exchange Network входят четыре раздела Stack Overflow на языках отличных от английского.

**1.2.6** Askee

Askee.ru – сервис вопросов и ответов [9], где вы можете не только задавать интересующие вас вопросы, но давать ответы пользователям сайта. Список вопросов показан на рисунке 1.6.

Всегда есть люди ищущие ответы на свои вопросы, а где-то совсем рядом есть всезнайки, которые помогут решить самые сложные проблемы.

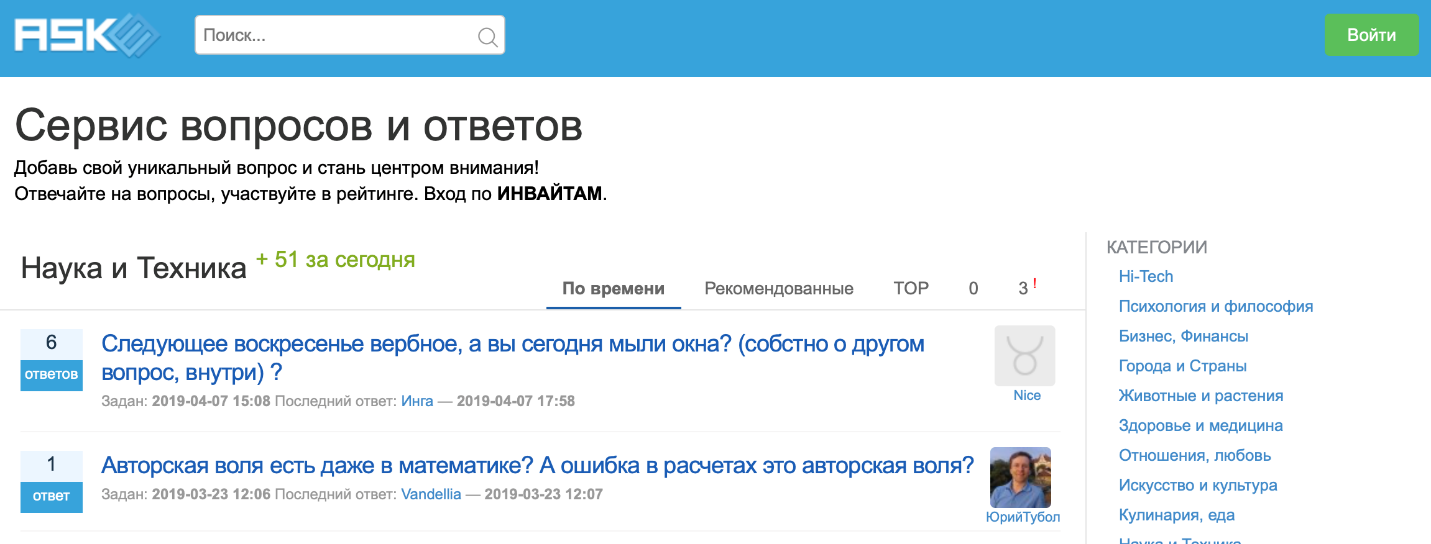


Рисунок 1.6 – Список вопросов на Askee

Суть данного сервиса предельно проста: у человека есть интересующий его вопрос, он задает его на этом сервисе, а другие участники отвечают.

Вы можете спрашивать и общаться, жаловаться и советоваться, решать жизненно важные вопросы и давать исчерпывающие ответы, приносящие реальную пользу посетителям сайта, тем самым помогая друг другу находить ответы на простые и сложные жизненные вопросы.

**1.2.7** TheQuestion

TheQuestion – российский сервис вопросов и ответов, который строит открытое сообщество людей, которые готовы критически осмысливать окружающую действительность, помогать друг другу, делиться разными взглядами, подвергать сомнению свои собственные убеждения, узнавать новое и меняться к лучшему [10]. Каждый человек может задать вопрос, мы находим тех, кто даст самый лучший ответ.

В одном вопросе должен содержаться только один вопрос. В вопросе не может содержаться утверждение, если это не общеизвестный факт.

Вопросы, затрагивающие обстоятельства личной жизни конкретных людей, не принимаются, если не представляют общественного интереса.

В ответе могут быть ссылки на статьи, книги или фильмы по теме, но ответ на вопрос не может состоять из одной ссылки с комментарием к ней или из текста, написанного не автором ответа. Для задания вопроса необходимо зарегистрироваться, как показано на рисунке 1.7.

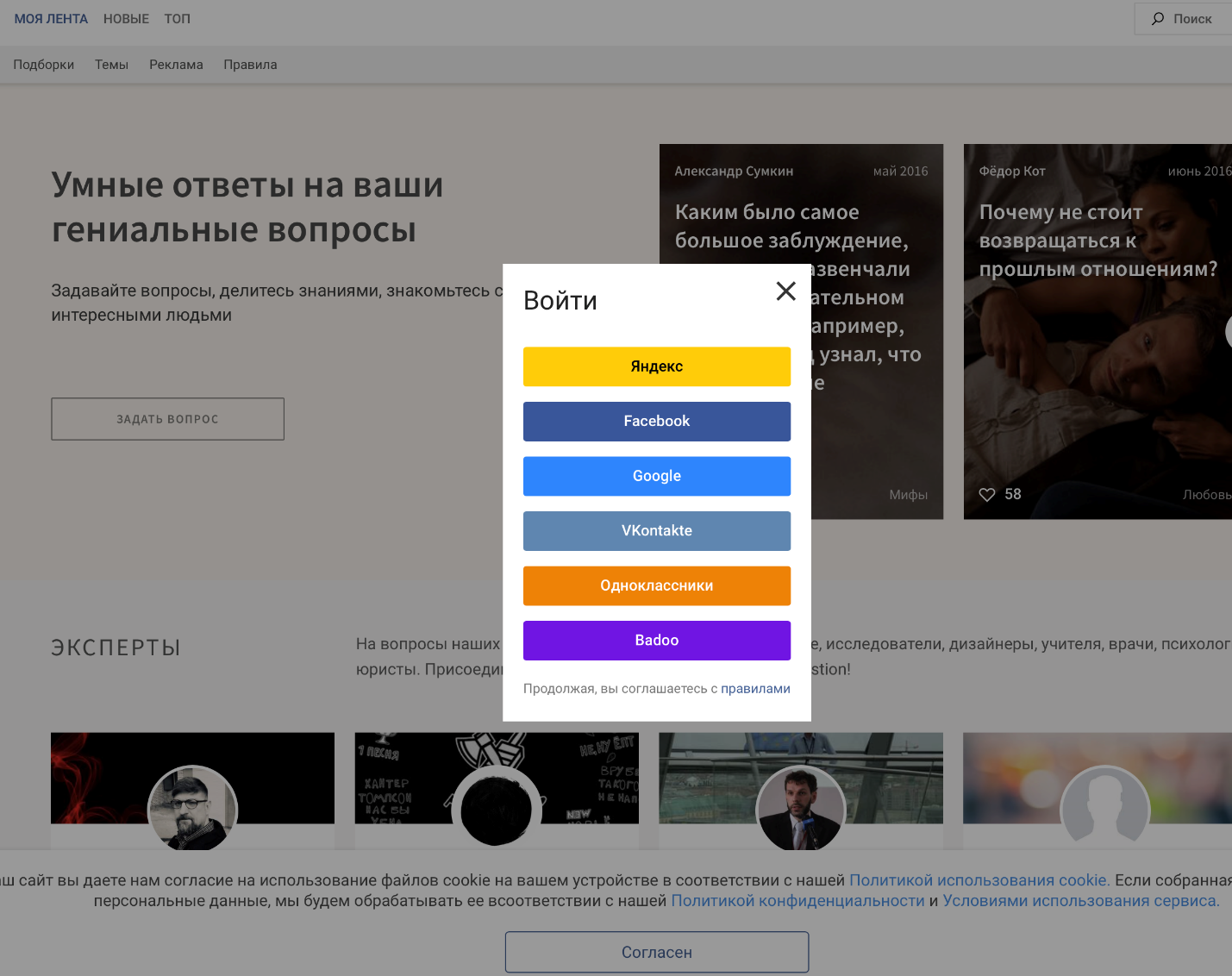


Рисунок 1.7 – Регистрация пользователя

На вопросы пользователей отвечают актеры, ученые, исследователи, дизайнеры, учителя, врачи, психологи, юристы. Это сервис, с помощью которого каждый может задать вопрос и писать ответы на вопросы по тем областям знаний, в которых он или она разбирается лучше всего.

Большинство сервисов вопросов и ответов должны реализовывать следующие задачи:

– возможность задать вопрос и получить на него через некоторое время ответ;

– доступ к существующей базе вопросов и ответов;

– поиск вопросов по категориям;

– возможность проголосовать за лучший ответ на свой вопрос.

В соответствии с темой дипломного проекта, наибольший интерес в су­ществующем ПО будет представлять функциональность быстрого поиска ответов на основе уже готовых пар вопросов и ответов.

Существующие на рынке решения являются неполными. Часто они не предлагают готовый ответ сразу после того, как пользователь ввёл свой вопрос.

# **2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## **2.1 Постановка задачи и функциональные требования проекта**

При проектировании приложения планируется реализация большого количества функциональных возможностей, призванных обеспечить пользователя удобным инструментом для поиска ответов на свои вопросы. Требования к ПО можно условно разделить на три группы:

* авторизация и управление ролями пользователей;
* работа с вопросами;
* поиск подходящего ответа на запрос пользователя.

К первой группе относятся следующие функциональные возможности:

* пользователь должен иметь доступ к своей учетной при помощи любого устройства, подключенного к сети интернет;
* гибкое управление процессом регистрации и авторизации, в том числе возможность создать учетные записи для модераторов и администраторов;
* модуль авторизации, дающий пользователям определенные привилегии только если они имеют соответствующую роль.

Вторая группа функциональных возможностей включают функции, с которыми пользователь сталкивается в процессе работы с вопросами. В таким функция относится:

* сохранение и индексация вопроса в хранилище ключ-значение;
* возможность задать категорию задаваемому вопросу;
* возможность для модераторов и администраторов редактировать, удалять вопросы;
* возможность добавлять ответы к заданным вопросам;
* возможность голосовать за лучший ответ для любого посетителя сайта.

К третьей группе функциональных возможностей относится:

* поиск по базе вопросов и получение возможного ответа на запрос пользователя;
* поиск подходящего ответа на основе голосов пользователей, отданных за ответы в похожем вопросе;
* осуществление поиска по похожим вопросам после создания самого вопроса.

Основными требованиями к разрабатываемому приложению являются высокая скорость выдачи ответа на запрос пользователя и высокое качество графического пользовательского интерфейса.

Приложение должно поддерживать один язык – русский, так как предназначено для русскоязычной аудитории. Веб-сервис должен поддерживать многопользовательский режим, позволяя работать в приложении одновременно нескольким людям.

С точки зрения аппаратных требований, приложение должно работать на современных Linux дистрибутивах, таких как Debian, Ubuntu после установки необходимых библиотек.

Архитектура веб-сервиса должна поддерживать возможность масштабирования для обработки большого количества запросов пользователей за единицу времени. В процессе разработки не должны быть использованы механизмы, препятствующие процессу масштабирования приложения.

Так как в приложении будут присутствовать личные данные пользователей, необходимо обеспечить их защиту и не показывать такие сведения как электронная почта любым пользователям интернета. Пользователи должны видеть только имена пользователей других участников веб-сервиса. Необходимо предусмотреть возможность ввода имени пользователя на этапе регистрации.

Для восстановления доступа к аккаунту необходима возможность сброса пароля пользователя при помощи электронной почты, указанной при регистрации в веб-сервисе.

Просмотр всех вопросов и ответов должен быть доступен всем пользователям без регистрации и аутентификации на веб-сервисе. Доступ к профилю любого пользователя с общей информацией, такой как количество вопросов и ответов, сами вопросы и ответы должен быть возможен без регистрации. Управление ролями пользователей, категориями должен осуществлять только владелец-администратор веб-сервиса.

## **2.2 Моделирование предметной области**

Предметная область – множество всех сущностей, связей между ними и свойства, которыми обладает проектируемая область. Проектирование предметной области является важным предварительным этапом разработки информационных систем. Продумывание сущностей системы и способов использования приложения позволяет создать надежный и качественный продукт.

Ниже перечислены некоторые из возможных претендентов системы, с которыми пользователям предстоит взаимодействовать:

* Разрабатываемое программное средство должно обеспечивать работу с сущностью вопрос, должен быть реализован функционал добавления и удаления вопросов, так как в некоторых случаях пользователи могут оставлять не совсем уместные вопросы и у администрации веб-сервиса должна быть возможность их удалить.
* Приложение должно поддерживать работу с сущностью ответ, так как пользователи должны иметь возможность оставить свой ответ на определенный вопрос, в случае, если на него уже не был дан правильный ответ.
* После добавления ответа, у любого авторизированного пользователя веб-сервиса должна появится возможность проголосовать за понравившийся ответ, это позволит в будущем пользователям, сталкивающимся с аналогичной проблемой, получить ответ моментально после задания аналогичного вопроса на сервисе.
* Так как вопросов на веб-сервисе может быть большое количество, необходимо их сгруппировать в категории – своеобразные тематические контейнеры, в которых будут находиться вопросы схожей тематики. Таким образом, если пользователь не может четко сформулировать вопрос, он сможет поискать похожие вопросы в категориях.
* Ввиду большого количества вопросов на сервисе, контролировать весь сервис одному человеку может быть трудно. Поэтому необходимо выделить сущность роль, а также возможность выдать роль определенному пользователю веб-сервиса. Выдавать и забирать роли должен только администратор сервиса.

В ходе моделирования были выделены следующие роли: гость, пользователь, модератор, администратор. Далее рассмотрим подробнее способы получения ролей в приложении.

Получение первой роли администратор должно осуществляться при помощи консоли управления веб-сервисом. Данная роль дает полный контроль над веб-сервисом, поэтому получение роли администратора не должно быть простым для нежелательных пользователей. После того, как был назначен первый администратор веб-сервиса, у данного администратора должна появится возможность назначения других пользователей в качестве администраторов веб-сервиса. Только администраторы должны иметь возможность управлять категориями, так как это довольно критичный момент в процессе использования веб-сервиса.

Незарегистрированный пользователь должен иметь роль гость. Данная роль позволяет посетителю веб-сервиса просматривать вопросы, ответы, категории, профили пользователь. Без регистрации доступна большая часть функционала для просмотра ресурсов. Регистрация позволяет быть участником веб-сервиса и взаимодействовать с ресурсами.

После прохождения регистрации, посетителю веб-сервиса автоматически должна присваиваться роль пользователь. Основное отличие незарегистрированного пользователя от гостя в возможности задавать вопросы, добавлять ответы, голосовать, то есть участвовать в жизни веб-сервиса.

Роль модератор должна давать возможность управлять всеми вопросами и ответами на веб-сервисе, поэтому назначение данной роли аккаунту пользователя должно осуществляться администратором веб-сервиса. Роль модератора необходима для осуществления контроля за действиями пользователь и, при необходимости, удаления нежелательного содержимого.

Для лучшего представления того, как пользователи будут использовать приложения, была создана диаграмма вариантов использования, которая представлена на рисунке 2.1.

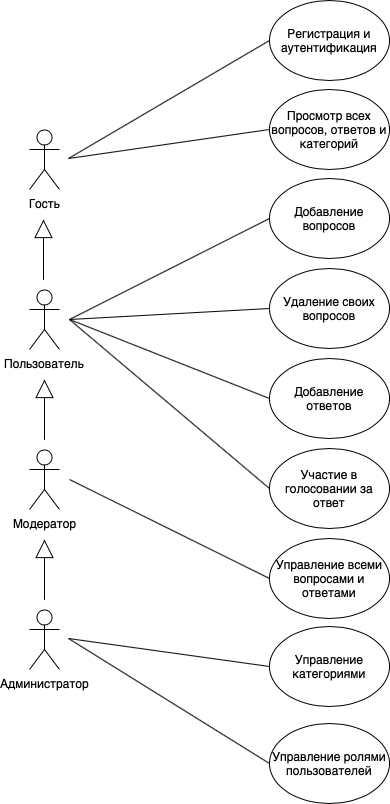


Рисунок 2.1 – Диаграмма вариантов использования приложения

# **3 ПРОЕКТИРОВАНИЕ ВЕБ-СЕРВИСА**

## **3.1 Общая структура приложения**

Для обеспечения высокой надёжности, удобства в развитии и сопровождении программного продукта, он должен иметь чёткую и понятную структуру. Различные части (модули, слои) приложения должны выполнять строго определённые функции; каждый слой приложения должно взаимодействовать с другими слоями как можно меньше.

При разработки данного приложения использовалась известная трёхуровневая архитектура. В структуре приложения выделяется также три уровня как показано на рисунке 3.1:

1. Уровень графического пользовательского интерфейса. Данный модуль отвечает за отображение информации и взаимодействие пользователя с приложением.
2. Уровень бизнес-логики. На этом уровне реализована основная логика приложения. Для разрабатываемого приложения к этому модулю относится вся логика по работе с вопросами, ответами и ролями, логика по управлению категориями, по аутентификации пользователя и др.
3. Уровень доступа к данным. Данный уровень реализует функциональность, необходимую для получения данных из базы данных и обновления информации в ней. Также к этому уровню относят функционал по преобразованию данных, если это требуется.

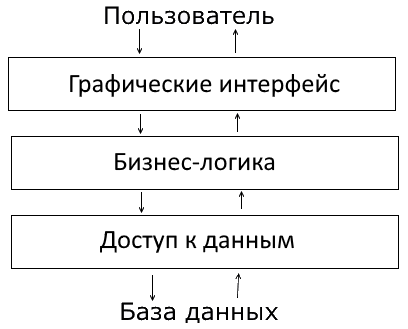


Рисунок 3.1 – Трёхуровневая архитектура приложения

Хранилище ключ-значение было выделено в отдельный компонент веб-сервиса. Данное хранилище представляет собой приложение, которое будет использовать в качестве хранилища базу данных либо хранение данных в памяти. В данном слое реализуется функционал для доступа к базе ключ-значение. Вынесение данного функционала в отдельное приложение позволяет абстрагироваться от используемой базы данных.

Общая структура веб-сервиса представлена на рисунке 3.2.

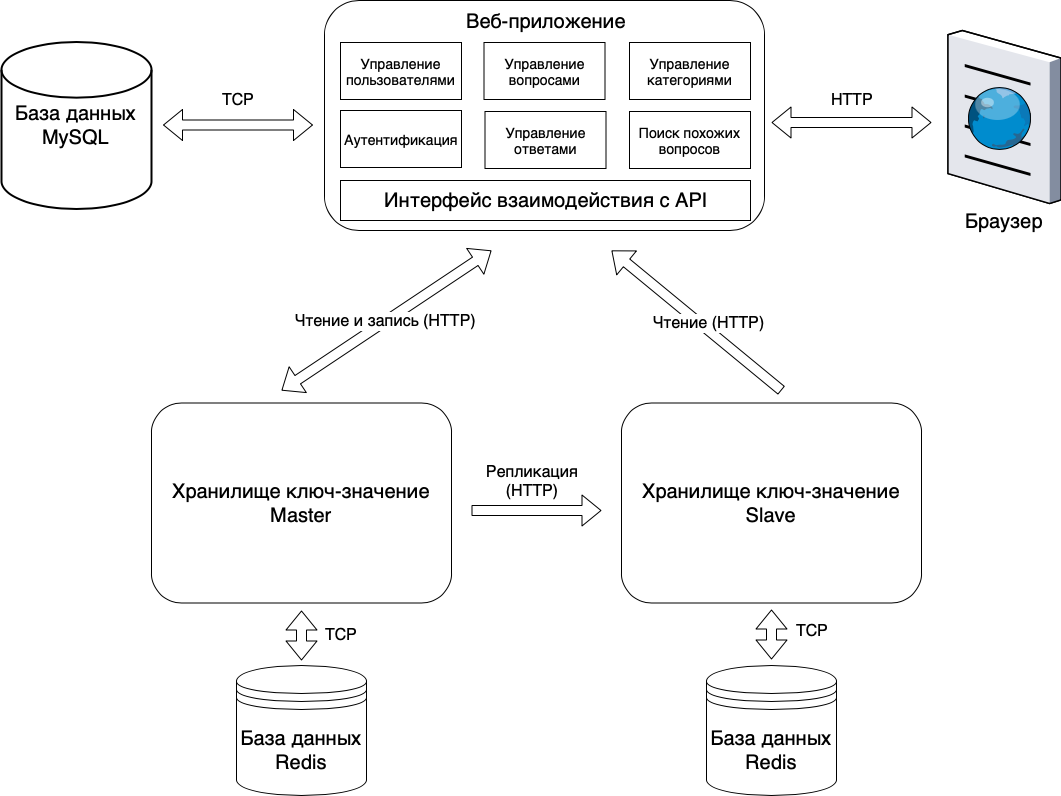


Рисунок 3.2 – Архитектура веб-сервиса

Для реализации основного приложения была выбрана модель MVC, так как данный шаблон проектирования реализован в большинстве веб-приложений и является практически стандартом для современного веб-приложения. Схема MVC представлена на рисунке 3.3.

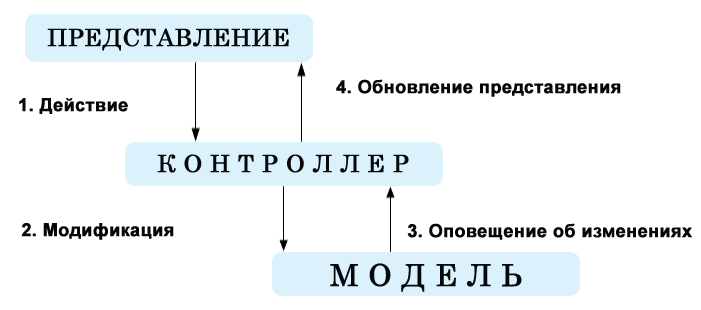


Рисунок 3.3 – Схема MVC

Организация доступа к БД в виде отдельного приложения позволяет обеспечить возможность масштабирования средства для быстрого поиска ответов на вопросы независимо от основного приложения. При этом, хранилище ключ-значение должно иметь возможность реплицировать свои данные. Тип репликации будет синхронный, так как в момент записи вопроса скорость записи вопроса не критична настолько, насколько в момент чтения.

Синхронная репликация позволяет быть уверенным в том, что данные на базах-репликах были успешно обновлены или созданы еще в момент создания вопроса. В случае появления ошибок записи, они будут выявлены еще на этапе задания вопроса.

Хранилище ключ-значение должно иметь возможность использовать или собственное хранилище данных в памяти, или использовать стороннюю базу ключ-значение. Данное приложение является оберткой над логикой хранения самих данных, таким образом в будущем можно будет заменить само хранилище на совершенно другое, не изменяя при этом никакой логики взаимодействия с хранилищем в основном приложении.

Функция репликации из собственной базы продолжит работать после изменения хранилища, так как она тоже должна быть абстрагирована от типа хранилища данных, используемое в настоящий момент. Взаимодействие между базой и основным приложением должно осуществляться по протоколу HTTP.

## **3.2 Основные модули приложения**

Всё приложение разбивается на отдельные модули, что позволяет обеспечить гибкую структуру программы. В частности, при таком подходе допускается изменение и расширение любого из модулей без изменения остальных. Каждый логически выделенный блок программы выполняет определенные функции. Кроме того, каждый блок программы так или иначе связан с остальными блоками, чтобы обеспечить работоспособность всего приложения в целом. Связь, как правило, реализуется посредством обмена данными между блоками. Часть модулей и связей между ними представлена на рисунке 3.4.

Модуль отображения данных. При помощи этого модуля пользователь взаимодействует с приложением, так как этот блок ответственен за отображение работы всего приложения. Блок выводит информацию о вопросах и ответах, предоставляет интерфейс для аутентификации пользователя, интерфейс управления ролями, интерфейс для отображения найденного ответа.

Модуль управления синхронизацией вопросов с хранилищем ключ-значение. В этом блоке сосредоточена вся логика по работе синхронизации вопросов. Данный модуль использует ряд других блоков, обеспечивающих доступ к хранилищу, доступ к серверу облачного хранения данных, проверку уникальности загружаемого вопроса.



Рисунок 3.3 – Основные модули веб-сервиса

Модуль для взаимодействия модулей. Степень повторного использования можно повысить через разбиение системы на множество блоков, однако при этом возникает множество взаимосвязей между этими блоками, которое приводит к обратному эффекту. Для исключения этой проблемы нужно инкапсулировать взаимодействие между блоками в блок–посредник. Этот посредник будет управлять взаимодействием группы блоков. Он перенаправляет запросы и ответы от одного блока к другому. Это сделано для того, чтобы централизовать работу приложения в одном месте и избавиться от жестких зависимостей между блоками, чтобы изменение одного блока приложения не влияло на остальные.

Модуль взаимодействия веб-приложения с хранилищем ключ-значение. Данный модуль должен обеспечивать выбор правильного модуля, необходимого для выполнения определенной операции в хранилище. Это позволит абстрагироваться от операций над хранилищем. Благодаря данному блоку, остальная часть приложения ничего не знает о конкретном действии, которое будет произведено над хранилищем.

Модуль для подготовки текста вопроса к последующему хешированию. В данном модуле необходим функционал для разбиения вопроса на определенные логические части, которые будут отправлены в хранилище на хранение отдельно друг от друга.

Модуль для обработки слова. В данном модуле будет реализован функционал для подготовки слова к хранению в базе. Будет реализован механизм для получения MD5 хеша из слова, механизм для взаимодействия с блоком, отвечающим за взаимодействие с хранилищем. Функционал для обновления состояния хранилища.

Модуль для управления процессом сохранения вопроса. В данном модуле необходимо реализовать вызов блоков для разбиения вопроса на слова и обеспечивать взаимодействие с блоком сохранения конкретного ответа. Данный блок является агрегирующим и позволяет абстрагироваться от процесса подготовки вопроса к хранению, а также абстрагироваться от логики сохранения слова из вопроса в хранилище.

Модуль поиска вопроса. В данном модуле будет реализована логика для поиска похожего вопроса. В модуль будет передаваться текст вопроса. Модуль будет разбирать вопрос на составные части и производить поиск в хранилище. Работа по разбиению вопроса на составные части и работа с хранилищем будет выполняться в других модулям, поэтому данный модуль ничего не знает о конкретных реализациях взаимодействия с хранилищем и о логике разбора вопроса на составные части.

Модуль управления голосованием. Для определения лучшего ответа, необходимо определенным образом вычислить наиболее полезный ответ. Данный модуль будет отвечать за возможность голосования у аутентифицированных пользователей. Модуль должен осуществлять проверку, не проголосовал ли уже пользователь за определенный ответ. Это поможет исключить накрутку голосов одним пользователем. Модуль должен следить за уникальностью пар голос-пользователь, а также обеспечивать вызов модуля работы с базой данных для сохранения голоса пользователя веб-сервиса.

Модуль взаимодействия с хранилищем. Так как хранилище для быстрого поиска вопросов будет располагаться отдельно от основного приложения, его реализация и веб-интерфейс могут изменяться со временем. Поэтому логика взаимодействия с ним при помощи сети должна быть реализована в виде отдельного модуля. Только данный модуль во всем приложении должен знать детали взаимодействия с сетевым хранилищем. При изменении деталей реализации сетевого хранилища, изменения потребуется внести только в данный модуль. Это уменьшит вероятность ошибки во время будущей поддержки приложения, так как изменения потребуется внести всего в одном месте приложения.

Модуль управления ролями. В данном модуле будет реализована логика работы с пользовательскими ролями. Модуль должен обеспечивать возможность задания роли определенному пользователю, а также возможность забрать определенную роль и выдать другие привилегии.

Модуль управления доступом. В данном модуле, в отличии от модуля управления ролями, должно происходить конкретное задание действий, которые может выполнять пользователь с определенной ролью.

Модуль предоставления доступа к данным из базы данных. Ни одно большое приложение не может обойтись без хранения информации в базе данных, но кроме хранения данных необходимо реализовать возможность взаимодействия управляющих блоков разрабатываемого приложения с данными хранящимися в базе данных. Этот блок предоставляет возможность сохранять, удалять, обновлять и считывать данные из базы данных.

Модуль аутентификации пользователей. Модуль предназначен для того, чтобы дать пользователю возможность аутентифицироваться и зарегистрироваться в веб-сервисе. Если у пользователя имеется учетная запись, он сможет пройти аутентификацию со своими учетными данными. Если пользователь забыл свой пароль от сервиса, он сможет восстановить его при помощи данного модуля.

## **3.3 Проектирование схемы базы данных**

Для хранения информации о вопросах, ответах, категориях, голосах, пользователях и ролей в приложении предполагается использование базы данных. На основе анализа предметной области были выделены сущности, в полной мере её описывают: вопрос, ответ, категория, пользователь, голос, роль. Для каждой сущности были определены наиболее подходящие поля.

Сущность вопрос должна содержать поля для хранения самого содержимого вопроса и время добавления вопроса. Сущность ответ должна содержать количество голосов, содержимое ответа пользователя и время ответа. Категория должна включать название и время добавления категории. Роль должна содержать название роли. Сущность голос не должна содержать полей, кроме связей с другими таблицами. Сущность пользователь должна содержать поля для хранения адреса электронный почты, хешированного пароля, имени пользователя, а также времени и даты регистрации аккаунта на веб-сервисе.

Для полей сущностей были выбраны типы данных. Для хранения времени добавления сущности было принято использовать колонку типа timestamp. Вопросы и ответы содержат текстовую информацию, поэтому для хранения данной информации был выбран тип text. В сущносте пользователь для хранения адреса электронной почты, хешированного пароля, имени пользователи был выбран тип text. Для хранения даты и времени регистрации был выбран тип timestamp.

Рассмотрим связи между основными сущностями. Ответы относятся к вопросу, поэтому в ответе должна храниться информация о вопросе, к которому относится ответ. Вопросы и ответы принадлежат конкретным пользователям, поэтому в данных сущностях необходимо хранить информацию об авторах. Роль и пользователь должны быть связаны при помощи связи многие ко многим. В вопросе должна содержаться информация об категории, в которую был добавлен вопрос. В сущносте голос должна содержаться информация о пользователя, который проголосовал, ответ, на который оставлен определенный голос.

В таблице users хранится информация о пользователях приложения, которые могут как зарегистрироваться, так и аутентифицироваться в приложении.

Рассмотрим колонки данной таблицы:

* email – электронный ящик пользователя приложения, он необходим для восстановления доступа к аккаунту, а также для идентификации пользователя в приложении;
* encrypted\_password – хеш-сумма, вычисленная на основе пароля пользователя, необходимо чтобы не хранить пароль пользователя в открытом виде;
* reset\_password\_token – случайно сгенерированное значение, которое будет использоваться в качестве ключа для восстановления доступа к учетной записе, если пользователь забыл свой пароль;
* reset\_password\_token\_sent\_at – время отправления ключа восстановления доступа на почту пользователя, необходимо для создания механизма истечения срока действия ключа;
* created\_at – дата и время регистрации пользователя;
* updated\_at – дата и время последнего обновления пользователя, например, смены пароля.

Таблица roles содержит информацию о ролях пользователей. Роли из данной таблицы могут быть применены к определенным пользователям при помощи промежуточной таблицы users\_roles. Если в определенный момент ни у одного из пользователей нет определенной роли, запись из данной таблицы можно удалить и создать заново, когда определенный пользователь получит необходимую роль. Основные колонки таблицы:

* name – название роли, которая может быть присвоена определенному пользователю;
* resource\_type – название класса сущности, к которой будет применена данная роль;
* resource\_id – идентификатор сущности, к которой будет применена данная роль в связке с колонкой resource\_type;
* created\_at – дата и время создания роли;
* updated\_at – дата и время последнего обновления роли.

Таблица users\_roles необходима для сопоставления пользователя и его ролей. Поле с ролью можно разместить прямо в таблице с пользователями, но в таком случае теряется возможность задать пользователю несколько ролей. Основные колонки:

* user\_id – идентификатор пользователя;
* role\_id – идентификатор роли из таблицы roles.

Таблица categories хранит информацию обо всех категориях вопросов, добавленных администратором. Основные колонки:

* name – название категории, которое задается администратором в момент её создания;
* created\_at – дата и время создания категории;
* updated\_at – дата и время последнего обновления категории.

Таблица questions содержит информацию о вопросах, которые задают пользователи приложения. Вопросы из данной колонки могут быть использованы для вывода списков вопросов на сайте. Основные колонки:

* description – описание вопроса, в котором содержится вопрос, который пользователь задает при помощи приложения;
* user\_id ­– идентификатор пользователя, задавшего вопрос, необходим для определения автора вопроса;
* category\_id – идентификатор категории, в которую был добавлен данный вопрос;
* created\_at – дата и время создания вопроса.

Таблица answers содержит информацию об ответах на запросы пользователя приложения. Ответы на вопрос могут оставлять любые пользователи, в том числе автор вопроса, чтобы иметь возможность ответить на определенный ответ от другого участника приложения. Основные колонки:

* content – содержимое ответа пользователя, которое может добавить любой из участников веб-сервиса;
* user\_id – идентификатор пользователя, который добавил данный ответ на определенный вопрос;
* votes\_count – количество голосов за данный ответ, необходимо для обеспечения более быстрого доступа к данному счетчику со стороны веб-сервиса при поиске похожих ответов;
* created\_at – дата и время создания ответа;
* updated\_at – дата и время последнего обновления ответа.

Таблица votes служит для хранения информации о голосах пользователей за определенные ответы. Колонки таблицы:

* user\_id – идентификатор пользователя, который добавил данный голос на определенный ответ;
* answer\_id – идентификатор ответа, к которому был добавлен голос.

Итоговая схема базы данных представлена на рисунке 3.4.

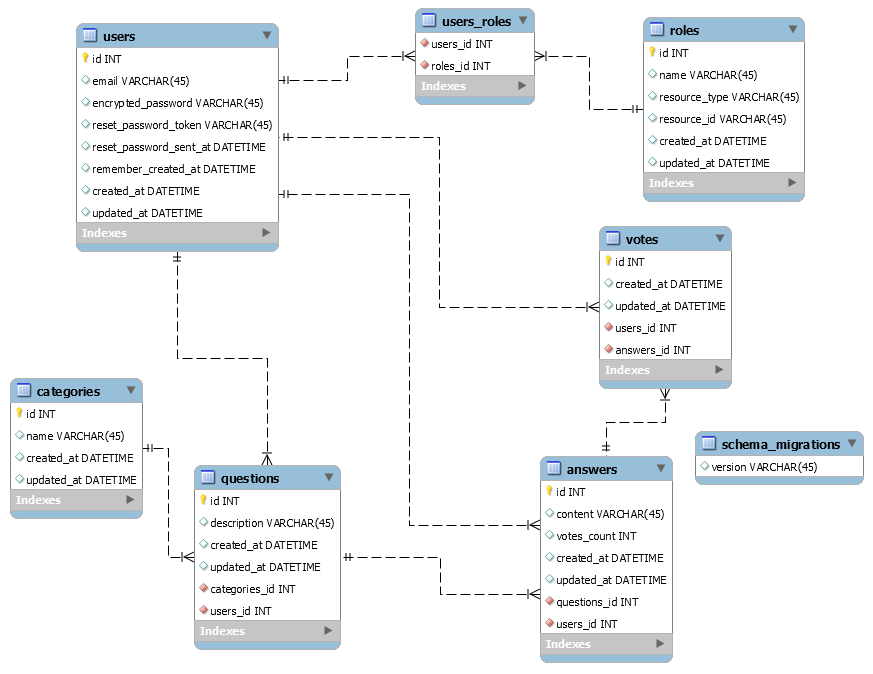


Рисунок 3.4 – Схема базы данных приложения

## **3.4 Распределенное хранилище данных**

Для реализации части базы данных было предложено использовать распределенное хранилище. Для обработки и хранения большого количества пользовательских дан­ных (в данном случае базы вопросов пользователей) недостаточно просто одного сервера. Для того, чтобы обеспечить пользователям доступ к миллионам записей без задержек и сбоев удобно использовать технологию распределенного хра­нилища данных.

Традиционно распределенное хранилище – это, по сути, централизованная система. Он хранит данные на нескольких независимых устройствах, используя масштабную структуру системы и несколько серверов хранения для совместной нагрузки на хранилище, и используя серверы местоположения для поиска и хранения информации [11].

На рынке существует много распределенных ключ-значение хранилищ. Функционал многих из них схож, но есть и отличия в их работе.

Memcached – это удобное высокопроизводительное хранилище данных в памяти [12]. Продуманное масштабируемое решение с открытым исходным кодом обеспечивает время отклика на уровне долей миллисекунды, что позволяет использовать его в качестве кэша или хранилища сессий. Memcached широко применяется для поддержки рекламных технологий, площадок интернет-коммерции, игровых, мобильных и интернет-приложений, а также других приложений, работающих в режиме реального времени.

В отличие от баз данных, хранящих данные на дисках или твердотельных накопителях, Memcached сохраняет данные в оперативной памяти. Поскольку Memcached, как и другие хранилища данных типа «ключ-значение» в памяти, не нуждается в доступе к диску, это исключает задержки, связанные с поиском, и обеспечивает доступ к данным за микросекунды. Кроме того, хранилище Memcached является распределенным, поэтому его можно просто масштабировать путем добавления новых узлов. Многопоточность Memcached позволяет быстро наращивать вычислительную мощность. Благодаря высокой скорости, масштабируемости, простоте, эффективности управления памятью и поддержке API для большинства распространенных языков программирования Memcached широко применяется для создания масштабного кэша с высокой производительностью.

Все данные Memcached хранятся в основной памяти сервера. В отличие от баз данных (таких как PostgreSQL, Cassandra и MongoDB, в которых данные хранятся преимущественно на дисках или твердотельных накопителях), хранилища данных в памяти не совершают постоянных циклических обращений к диску. Благодаря этому многократно увеличивается количество выполняемых операций и сокращается время отклика. Это обеспечивает исключительное быстродействие: время чтения и записи измеряется долями миллисекунды, и хранилище может выполнять миллионы операций в секунду.

Простота и универсальность делают Memcached мощным и очень удобным решением для разработки приложений. При работе с Memcached разработчики получают в распоряжение множество готовых клиентов с открытым исходным кодом. Поддерживаемые языки программирования включают Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, Go и многие другие.

Распределенная многопоточная архитектура упрощает масштабирование Memcached. Данные можно распределить между несколькими узлами, что позволяет наращивать ресурсы путем добавления в кластер новых узлов. Благодаря многопоточности Memcached также может использовать несколько ядер в одном узле. Это позволяет просто наращивать вычислительную мощность. С помощью Memcached можно создавать высокомасштабируемые распределенные решения для кэширования, обеспечивающие стабильную высокую производительность.

Memcached – качественный проект с открытым исходным кодом и активным сообществом поддержки. Использование Memcached для повышения производительности поддерживается в таких приложениях, как интернет-магазины, блоги. Поскольку Memcached базируется на открытых стандартах, поддерживает открытые форматы данных и имеет множество клиентов, это исключает привязку к определенному поставщику или технологии.

Memcached – прекрасное решение для реализации высокопроизводительного кэша в памяти, позволяющего сократить задержку при доступе к данным, повысить пропускную способность и снизить нагрузку на серверные системы. Время доступа к объектам в кэше Memcached составляет менее миллисекунды. Кроме того, такой кэш можно просто и экономично масштабировать при повышении нагрузки. Memcached широко применяется для кэширования сессий, веб-страниц, API, результатов запросов к базе данных, а также изображений, файлов и метаданных.

Многие разработчики используют Memcached как хранилище данных в памяти и средство управления данными для сессий приложений, работающих в масштабе Интернета, которым не требуется постоянное хранение. Memcached обеспечивает задержку на уровне долей миллисекунды и поддерживает работу в масштабах, необходимых для управления такими данными сессий, как профили пользователей, учетные данные и состояние сессий.

Redis (расшифровывается как Remote Dictionary Server) – это быстрое хранилище данных типа «ключ‑значение» в памяти с открытым исходным кодом для использования в качестве базы данных, кэша, брокера сообщений или очереди [13]. Проект возник, когда Сальваторе Санфилиппо, первоначальный разработчик Redis, пытался улучшить масштабируемость стартапа в Италии. Redis обеспечивает время отклика на уровне долей миллисекунды и позволяет приложениям, работающим в режиме реального времени, выполнять миллионы запросов в секунду. Такие приложения востребованы в сфере игр, рекламных технологий, финансовых сервисов, здравоохранения и IoT. Redis широко применяется для кэширования, управления сеансами, разработки игр, создания таблиц лидеров, аналитики в режиме реального времени, работы с геопространственными данными, поддержки служб такси, чатов и сервисов обмена сообщениями, потоковой передачи мультимедиа и приложений с отправкой сообщений по модели «издатель – подписчик».

Все данные в Redis хранятся в памяти, а не на дисках или твердотельных накопителях, как в других базах данных. Поскольку Redis, как и другие хранилища данных в памяти, не нуждается в доступе к диску, это исключает задержки, связанные с поиском, и обеспечивает доступ к данным за микросекунды. В число возможностей Redis входит поддержка разнообразных структур данных, обеспечение высокой доступности, работа с геопространственными данными, создание скриптов Lua, проведение транзакций, постоянное хранение данных на диске и поддержка кластеров. Все это упрощает создание приложений, работающих в режиме реального времени в масштабе всего Интернета.

Как Redis, так и MemCached представляют собой хранилища данных в памяти с открытым исходным кодом. Высокопроизводительный сервис кэширования с распределенной памятью Memcached отличает простота, а Redis обладает широкими функциональными возможностями, которые позволяют эффективно использовать хранилище для разнообразных целей. Они используются с реляционными базами данных или базами данных на основе пар «ключ – значение» для повышения производительности.

Redis 5, а теперь уже Redis 5.0.3, – это последняя общедоступная версия Redis с открытым исходным кодом. С момента первого выпуска в 2009 г. система Redis с открытым исходным кодом превратилась из технологии кэширования в простое в использовании и быстрое хранилище данных в памяти с универсальными структурами данных и временем отклика на уровне долей миллисекунды. Главной вехой для Redis стал выпуск версии 5.0, в которую вошел целый ряд улучшений и усовершенствований. Основным нововведением стало внедрение функции Streams – первой совершенно новой структуры данных в Redis после HyperLogLog. В этом выпуске также добавлены команды для структур данных Sorted Set и новые возможности для API модуля.

Все данные Redis находятся в основной памяти сервера, в отличие от таких баз данных, как PostgreSQL, Cassandra, MongoDB и других, которые большую часть данных хранят на магнитных дисках или SSD‑накопителях. По сравнению с традиционными дисковыми базами данных, требующими циклического обращения к диску для большинства операций, хранилища данных в памяти, такие как Redis, свободны от этого ограничения. Благодаря этому многократно увеличивается количество выполняемых операций и сокращается время отклика. В результате обеспечивается чрезвычайно высокая производительность. Операции чтения или записи в среднем занимают менее миллисекунды, скорость работы достигает миллионов операций в секунду.

В отличие от упрощенных хранилищ на основе пар «ключ – значение», которые поддерживают ограниченный набор структур данных, Redis поддерживает огромное разнообразие структур данных, позволяющее удовлетворить потребности разнообразных приложений.

Типы данных Redis включают:

* строки – текстовые или двоичные данные размером до 512 МБ;
* списки – коллекции строк, упорядоченные в порядке добавления;
* множества – неупорядоченные коллекции строк с возможностью пересечения, объединения и сравнения с другими типами множеств;
* сортированные множества – множества, упорядоченные по значению;
* хеш‑таблицы – структуры данных для хранения списков полей и значений;
* битовые массивы – тип данных, который дает возможность выполнять операции на уровне битов;
* структуры HyperLogLog – вероятностные структуры данных, служащие для оценки количества уникальных элементов в наборе данных.

Redis упрощает код, позволяя писать меньше строк для хранения, использования данных и организации доступа к данным в приложениях. К примеру, если приложение содержит данные, хранящиеся в хэш‑таблице, и требуется сохранить эти данные в хранилище, можно просто использовать структуру данных хэш‑таблицы Redis. Решение подобной задачи с использованием хранилища данных, не поддерживающего структуры хэш‑таблиц, потребует написания серьезного объема кода для преобразования данных из одного формата в другой. Redis уже оснащен встроенными структурами данных и предоставляет множество возможностей их комбинирования и взаимодействия с данными клиента. Разработчикам под Redis доступны более ста клиентов с открытым исходным кодом. Поддерживаемые языки программирования включают Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go и многие другие.

В Redis применяется архитектура узлов «ведущий‑подчиненный» и поддерживается асинхронная репликация, при которой данные могут копироваться на несколько подчиненных серверов. Это обеспечивает как улучшенные характеристики чтения (так как запросы могут быть распределены между серверами), так и ускоренное восстановление в случае сбоя основного сервера. Для обеспечения постоянного хранения Redis поддерживает снимки состояния на момент времени (копирование наборов данных Redis на диск).

Redis предлагает архитектуру «ведущий‑подчиненный» с одним ведущим узлом или с кластерной топологией. Это позволяет создавать высокодоступные решения, обеспечивающие стабильную производительность и надежность. Если требуется настроить размер кластера, доступны различные варианты вертикального и горизонтального масштабирования. В результате можно наращивать кластер в соответствии с потребностями.

Redis – проект с открытым исходным кодом, поддерживаемый активным сообществом. Поскольку Redis базируется на открытых стандартах, поддерживает открытые форматы данных и имеет множество клиентов, отсутствует вероятность блокировки поставщиком или технологического тупика.

Облачное хранилище данных — это модель онлайн-хранилища, в кото­ром данные хранятся на многочисленных распределённых в сети серверах, предоставляемых в пользование клиентам, в основном, третьей стороной [14]. В противовес модели хранения данных на собственных выделенных серверах, приобретаемых или арендуемых специально для подобных целей, количество или какая-либо внутренняя структура серверов клиенту, в общем случае, не видна. Данные хранятся, а равно и обрабатываются, в так называемом облаке, которое представляет собой, с точки зрения клиента, один большой виртуаль­ный сервер. Физически же такие серверы могут располагаться удалённо друг от друга географически, вплоть до расположения на разных континентах.

Другими словами, это своеобразный онлайн-сервис, предоставляющий возможность хранить файлы на удалённом сервере. То есть приложение мо­жет загрузить данные в любое онлайн-хранилище и в будущем использовать их прямо из сервера. С точки зрения клиента, все операции происходят в од­ном месте, так называемом «облаке». Однако на самом деле, удалённый сер­вер чаще всего располагается в разных местах, а иногда и на разных конти­нентах. Но это нисколько не затрудняет работу облачных сервисов, так как скорость работы зависит от клиента. А точнее, от скорости интернет-соедине­ния у клиента.

Ключевой особенностью облачного хранения данных является то, что хоть облачное хранилище как правило включает в себя большое количество серверов, с точки зрения клиента все они представляют собой единое целое за счёт высокого уровня виртуализации.

Среди преимуществ облачных хранилищ стоит отметить высокую надёжность за счёт распределения и резервного копирования данных. Все за­гружаемые данные сразу копируются на несколько независимых серверов. Даже при выходе из строя некоторых серверов данные не пострадают и будут доступны, а системы внутреннего контроля хранилища автоматически рас­пределят недостающие копии данных по другим рабочим серверам.

Облачная инфраструктура хранения обеспечивает хорошее масштабирование. Серверы и система хранения допускают изменение размера без всяких последствий для пользователей. Облачная архитектура хранения поддерживает автономные вычисления.

# **4 РАЗРАБОТКА ВЕБ-СЕРВИСА**

## **4.1 Выбор и обоснование технологий реализации**

В качестве стека технологий был выбран: язык программирования Ruby, фреймворк Ruby on Rails, фреймворк Sinatra, база данных Redis, база данных MySQL. Ниже приведено описание каждой из технологий.

В качестве языка программирования был выбран язык Ruby. Ruby – динамический, рефлективный, интерпретируемый высокоуровневый язык программирования [15]. Язык обладает независимой от операционной системы реализацией многопоточности, сильной динамической типизацией, сборщиком мусора и многими другими возможностями. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу – к Smalltalk. Также некоторые черты языка взяты из Python, Lisp.

Кроссплатформенная реализация интерпретатора языка является полностью свободной. Сейчас Ruby входит в большинство дистрибутивов ОС Linux, поставляется вместе с Mac OS X, доступен пользователям других операционных систем. Одним из основных приложений, связанных с Ruby, продолжает оставаться Ruby on Rails, который продолжает активно развиваться, но использование Ruby значительно шире – на нём разрабатывается большое количество приложений различного назначения, кроме того, он используется в качестве скриптового языка для автоматизации и настройки приложений и написания административных утилит, в частности, в ОС Linux.

Приоритетом является удобство и минимизация затрат труда программиста при разработке программы, освобождение программиста от рутинной работы, которую компьютер может выполнять быстрее и качественнее. Особое внимание, в частности, уделено будничным рутинным занятиям (обработка текстов, администрирование), и для них язык настроен особенно хорошо. В противовес машинно-ориентированным языкам, работающим быстрее, Ruby это язык, наиболее близкий к человеку. Любая работа с компьютером выполняется людьми и для людей, и необходимо заботиться в первую очередь о затрачиваемых усилиях людей.

Программа должна вести себя так, как ожидает программист. Но в контексте Ruby это означает наименьшее удивление не при знакомстве с языком, а при его основательном изучении. Сам Мацумото утверждает, что целью разработки была минимизация неожиданностей при программировании для него, но после распространения языка он с удивлением узнал, что мышление программистов похоже, и для многих из них принцип «наименьшей неожиданности» совпал с его принципом.

Избыточность допустима, если она удобна. Ruby унаследовал идеологию языка программирования Perl в части предоставления программисту возможностей достижения одного и того же результата несколькими различными способами. Люди различны, и им для свободы необходима возможность выбирать.

Наличие в языке динамических средств, вплоть до самомодификации программы во время исполнения, дают возможности, которые очень полезны для эффективного программирования. Снижение производительности, на которое приходится пойти ради них, в большинстве случаев вполне допустимо.

Ruby полностью объектно-ориентированный язык. В нём все данные являются объектами, в отличие от многих других языков, где существуют примитивные типы. Каждая функция – метод. Ruby использует вызов по соиспользованию, хотя в сообществе Ruby часто говорят, что он использует вызов по ссылке. Для программиста, привыкшего к распространённым гибридным языкам программирования, некоторые эффекты такого решения могут показаться неожиданными.

Ruby не поддерживает множественное наследование, но вместо него есть мощный механизм примесей.

Ruby является мультипарадигменным языком: он поддерживает процедурный стиль (определение функций и переменных вне классов), объектно-ориентированный (всё – объект), функциональный (анонимные функции, замыкания, возврат значения всеми инструкциями, возврат функцией последнего вычисленного значения). Он поддерживает рефлексию, метапрограммирование, информацию о типах переменных на стадии выполнения (см. динамическая идентификация типа данных).

Основные возможности ruby:

* имеет лаконичный и простой синтаксис;
* позволяет обрабатывать исключения в стиле Java и Python;
* позволяет переопределять операторы, которые на самом деле являются методами;
* полностью объектно-ориентированный язык программирования, все данные в ruby являются объектами;
* не поддерживает множественное наследование;
* содержит автоматический сборщик мусора;
* поддерживает замыкания с полной привязкой к переменным;
* имеет независимую от ОС поддержку многопоточности. [12]

В качестве фреймворка для основного приложения был выбран Rails.

Ruby on Rails – фреймворк, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных [16]. Является открытым программным обеспечением и распространяется под лицензией MIT.

Базируется на следующих принципах разработки приложений. Первый это максимальное использование механизмов повторного использования, позволяющих минимизировать дублирование кода в приложениях. По умолчанию используются соглашения по конфигурации, типичные для большинства приложений – явная спецификация конфигурации требуется только в нестандартных случаях.

Фреймворк, в отличии от CMS (системы управления контентом), которую может развернуть и настроить даже не программист, требует проектирования и разработки квалифицированными специалистами. Но зато на нём проще и быстрее создавать проекты, которые отличаются своим функционалом от полностью типового сайта.

Основным преимуществом языка программирования Ruby и фреймворка Ruby on Rails является скорость разработки. На практике скорость разработки проектов на RoR выше на 30-40 процентов по отношению к любому другому языку программирования или фреймворку. Такой прирост скорости разработки объясняется обширным набором готовых к работе штатных инструментов RoR, возможностью использовать готовые решения других разработчиков, ну и, конечно, удобством программирования на Ruby.

Кроме того, в отличие от других фреймворков, в составе RoR есть отличные средства автоматизированного тестирования.

Так же следует отметить, что Ruby on Rails обеспечивает лучшую безопасность проекта. При использовании инструментов RoR исключены SQL-инъекции и XSS-атаки, все входные параметры экранируется по умолчанию, выводимые переменные в шаблонах также экранируются.

Конвенции написания программного кода на базе Rails позволяют писать действительно понятный программный код, который впоследствии может быть без проблем модифицирован в адекватные сроки, а соблюдение этих конвенций делает программный код сопровождаемым не только изначальным разработчиком, но и любым другим специалистом или любой другой командой. Отсутствие привязки к изначальному разработчику – это очень важная составляющая проекта, разрабатываемого заказчиком силами внешней компании, а не собственными силами.

В Rails разработке [17] обычно используется TDD-подход, что делает создаваемые решения более стабильными и сопровождаемыми. Функциональность самого фреймворка также покрыта автоматическими тестами, что делает его использование действительно надёжным – есть уверенность, что ничего не сломается. Для бизнес-систем эта составляющая крайне важна – от стабильности работы приложения часто зависит эффективность работы бизнеса в целом.

Основными компонентами приложений на Ruby on Rails являются модель, представление и контроллер. Ruby on Rails использует REST-стиль построения веб-приложений.

Модель предоставляет остальным компонентам приложения объектно-ориентированное отображение данных (таких как каталог продуктов или список заказов). Объекты модели могут осуществлять загрузку и сохранение данных в реляционной базе данных, а также реализуют бизнес-логику.

Для хранения объектов модели в реляционной СУБД по умолчанию в Rails 3 использована библиотека ActiveRecord. Конкурирующий аналог – DataMapper. Существуют плагины для работы с не реляционными базами данных, например, Mongoid для работы с MongoDB.

Представление создаёт пользовательский интерфейс с использованием полученных от контроллера данных. Представление также передает запросы пользователя на манипуляцию данными в контроллер (как правило, представление не изменяет непосредственно модель).

В Ruby on Rails представление описывается при помощи шаблонов ERB – файлов HTML с дополнительными включениями фрагментов кода Ruby (Embedded Ruby или ERb). Вывод, сгенерированный встроенным кодом Ruby, включается в текст шаблона, после чего получившаяся страница HTML возвращается пользователю. Кроме ERB возможно использовать ещё около 20 шаблонизаторов, в том числе Haml.

Контроллер в Rails – это набор логики, запускаемой после получения HTTP-запроса сервером. Контроллер отвечает за вызов методов модели и запускает формирование представления.

Соответствие интернет-адреса с действием контроллера (маршрут) задается в файле конфигурации. Открытые методы контроллера являются так называемыми действиями. Действия часто соответствует отдельному представлению. Например, по запросу пользователя admin/index будет вызван метод index класса AdminController и затем использовано представление index.html.erb из директории views/admin.

Предпочтительным методом интеграции с веб-серверами является проксирование – использование веб-сервера в качестве прокси-сервера перед сервером приложения. Особняком стоят модули Phusion Passenger для интеграции с серверами Apache и nginx.

Ruby on Rails использует интерфейс Rack [18], что позволяет использовать менее распространённые механизмы (FCGI, CGI, SCGI). Ruby on Rails может работать с Apache, Lighttpd или любым другим веб-сервером, поддерживающим FastCGI. Для разработки и отладки используется веб-сервер Puma (ранее WEBrick, встроенный в Ruby, или Mongrel). В качестве сервера базы данных поддерживаются MySQL, PostgreSQL, Firebird, DB2, Oracle и Microsoft SQL Server. Также поддерживается встраиваемая база данных SQLite.

Для Windows существует дистрибутив Instant Rails с настроенной и готовой к работе сразу после установки рабочей средой для разработки Rails-приложений, которая включает в себя сервер Apache и СУБД MySQL, а также дистрибутив RubyInstaller, включающий последние версии Ruby и инструменты разработчика. Для платформ Windows, Linux, Mac OS X имеется комплексный установщик BitNami RubyStack, включающий в себя все необходимое для разработки в среде Rails, включая Ruby, RubyGems, Ruby on Rails, MySQL, Apache, Mongrel и Subversion.

Помимо этого, сайты BitNami.org и JumpBox.com бесплатно предлагают образы VMware с готовой Linux-средой для развертывания RoR-приложений. Эти образы можно подключить к своему серверу виртуальных машин или развернуть в предлагаемой облачной среде.

Для разработки AJAX-приложений в RoR (до версии 5.1) по умолчанию использовался javascript-фреймворк jQuery, однако вместо него можно использовать и другие библиотеки. В ранних версиях Ruby on Rails (до 3.1), js-фреймворком по умолчанию был Prototype.

В качестве репозитория плагинов Ruby on Rails использует экосистему пакетов RubyGems. Некоторые плагины со временем были включены в базовую поставку Rails, например, Sass и CoffeeScript; другие же, хотя и не были включены в базовую поставку, являются стандартом де-факто для большинства разработчиков (например, средство модульного тестирования RSpec).

Начиная с 3-й версии Rails наблюдается тенденция вынесения части функциональных возможностей в отдельные модули, отчасти из-за их более быстрого развития, чем сам Rails, отчасти для облегчения фреймворка.

Некоторые модули: Devise (для аутентификации), CanCan (для авторизации), Kaminari (для разделения записей, извлекаемых из базы данных, или элементов массива по страницам), Faker (для случайной генерации тестовых наборов данных в веб-приложениях), friendly\_id (позволяет создавать человеко-понятные веб-адреса), Active Admin (для создания панелей администрирования), CommunityEngine (для создания социальных сетей).

В качестве базы данных была выбрана MySQL.

База данных - это специально разработанное хранилище для различных типов данных [19]. Каждая база данных, имеет определённую модель (реляционная, документно-ориентированная), которая обеспечивает удобный доступ к данным. Системы управления базами данных (СУБД) - специальные приложения (или библиотеки) для управления базами данных различных размеров и форм.

СУБД должна обеспечивать реляционную модель работы с данными. Сама модель подразумевает определенный тип связи между сущностями из разных таблиц. Чтобы хранить и работать с данными, такой тип СУБД должен иметь определенную структуру (таблицы). В таблицах каждый столбец может содержать данные разного типа. Каждая запись состоит из множества атрибутов (столбцов) и имеет уникальный ключ, хранящейся в той же таблице - все эти данные взаимосвязаны между собой, как описано в реляционной модели.

Отношения в базах данных можно рассматривать как математическое множество, содержащее в себе число атрибутов, которые суммарно представляют собой базу данных и информацию, хранящуюся в ней (фраза для тех, кто понимает, что такое математическое множество).

При создании структуры таблицы каждое поле записи должно иметь заранее описанный тип (например, строка, целочисленное значение). Все СУБД имеют в своем составе различные типы данных, которые не всегда взаимозаменяемы. При работе с СУБД всегда приходится сталкиваться с подобными ограничениями.

Рассмотрим три основных свобоно распространяемых СУБД. SQLite – очень мощная встраиваемая система управления. MySQL – самая популярная и распространённая СУБД. PostgreSQL - наиболее продвинутая СУБД.

SQLite легко встраиваемая в приложения база данных. Так как это система базируется на файлах, то она предоставляет довольно широкий набор инструментов для работы с ней, по сравнению с сетевыми СУБД. При работе с этой СУБД обращения происходят напрямую к файлам (в этих файлах хранятся данные), вместо портов и сокетов в сетевых СУБД. Именно поэтому SQLite очень быстая, а также мощная благодаря технологиям обслуживающих библиотек.

Рассмотрим преимущества SQLite. Файловая структура - вся база данных состоит из одного файла, поэтому её очень легко переносить на разные машины.

Используемые стандарты - хотя может показаться, что эта СУБД примитивная, но она использует SQL. Некоторые особенности опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), но основные все-таки поддерживаются.

Отличная при разработке и тестировании - в процессе разработки приложений часто появляется необходимость масштабирования. SQLite предлагает всё что необходимо для этих целей, так как состоит всего из одного файла и библиотеки написанной на языке C.

Один из основных недостатков SQLite это отсутствие системы пользователей - более крупные СУБД включают в свой состав системы управления правами доступа пользователей. Обычно применения этой функции не так критично, так как эта СУБД используется в небольших приложениях. Также, отсутствие возможности увеличения производительности - опять, исходя из проектирования, довольно сложно выжать что-то более производительное из этой СУБД.

MySQL – это самая распространенная полноценная серверная СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб приложениями. Обучиться использованию этой СУБД довольно просто, так как на просторах интернета вы легко найдете большее количество информации [20].

Несмотря на то, что в ней не реализован весь SQL функционал, MySQL предлагает довольно много инструментов для разработки приложений. Так как это серверная СУБД, приложения для доступа к данным, в отличии от SQLite работают со службами MySQL.

Рассмотрим преимущества MySQL. Простота в работе – установить MySQL довольно просто. Дополнительные приложения, например, GUI, позволяет довольно легко работать с БД

Богатый функционал – MySQL поддерживает большинство функционала SQL.

Безопасность – большое количество функций обеспечивающих безопасность, которые поддерживается по умолчанию.

Масштабируемость – MySQL легко работает с большими объемами данных и легко масштабируется

Скорость – упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

Рассмотрим недостатки MySQL. Известные ограничения - по задумке в MySQL заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях.

Проблемы с надежностью - из-за некоторых способов обработки данных MySQL (связи, транзакции, аудиты) иногда уступает другим СУБД по надежности.

Медленная разработка – хотя MySQL технически открытое ПО, существуют жалобы на процесс разработки. Стоит заметить, что существуют другие довольно успешные СУБД созданные на базе MySQL, например MariaDB.

PostgreSQL является самым профессиональным из всех трех рассмотренных нами СУБД. Она свободно распространяемая и максимально соответствует стандартам SQL. PostgreSQL или Postgres стараются полностью применять ANSI/ISO SQL стандарты своевременно с выходом новых версий.

От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Например, полная поддержка надежных транзакций, т.е. атомарность, последовательность, изоляционность, прочность (Atomicity, Consistency, Isolation, Durability (ACID).) Благодаря мощным технологиям Postgre очень производительна. Параллельность достигнута не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом (MVCC), что также обеспечивает соответствие ACID. PostgreSQL очень легко расширять своими процедурами, которые называются хранимые процедуры. Эти функции упрощают использование постоянно повторяемых операций.

Хотя PostgreSQL и не может похвастаться большой популярностью в отличии от MySQL, существует довольно большое число приложений облегчающих работу с PostgreSQL, несмотря на всю мощность функционала. Сейчас довольно легко установить эту СУБД используя стандартные менеджеры пакетов операционных систем.

Рассмотрим преимущества PostgreSQL. Открытое ПО соответствующее стандарту SQL - PostgreSQL - бесплатное ПО с открытым исходным кодом. Эта СУБД является очень мощной системой.

Большое сообщество - существует довольно большое сообщество в котором вы запросто найдёте ответы на свои вопросы

Большое количество дополнений - несмотря на огромное количество встроенных функций, существует очень много дополнений, позволяющих разрабатывать данные для этой СУБД и управлять ими.

Расширения – существует возможность расширения функционала за счет сохранения своих процедур.

Объектность – PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого.

Рассмотрим недостатки PostgreSQL. Производительность - при простых операциях чтения PostgreSQL может значительно замедлить сервер и быть медленнее своих конкурентов, таких как MySQL

Популярность – по своей природе, популярностью эта СУБД похвастаться не может, хотя и присутствует довольно большое сообщество.

Хостинг – в силу выше перечисленных факторов иногда довольно сложно найти хостинг с поддержкой этой СУБД.

MySQL поддерживает SQL (структурированный язык запросов) и может применяться в качестве SQL-сервера. Это означает, что общаться с сервером можно на языке SQL: клиент посылает серверу запрос, тот его обрабатывает и отдает клиенту только те данные, которые были получены в результате этого запроса. Тем самым клиенту не требуется выкачивать данные и производить вычисления, как, например, в Microsoft Access.

Кроме того, MySQL - это ПО с открытым кодом, т.е. его можно свободно изучать и изменять. Пакет распространяется на условиях GPL (General Public License), его можно бесплатно загрузить из Интернета (http://www.mysql.com) для некоммерческого применения.

С появлением Интернет-технологий, позволяющих создавать динамичные Web-страницы, необычайно возрос спрос и на СУБД, которые наиболее полно подходили бы для этого по быстродействию, надежности и стабильности. И здесь хорошо проявил себя пакет MySQL, который получился быстрым, простым и надежным, но, правда, за счет ухудшения функциональности (сразу оговоримся, что разработчики MySQL обещают добавить недостающие функции уже в ближайших версиях программы).

По большому счету, отсутствие некоторых функций, которые были принесены в жертву быстродействию и надежности, не создает больших хлопот пользователям (хотя иногда некий дискомфорт и имеет место). Для работы с полноценной корпоративной базой данных MySQL недотягивает, но с повседневными задачами MySQL справляется довольно хорошо.

Транзакции - позволяют объединить несколько SQL-запросов в одну единицу работы и в случае сбоя любого из запросов, входящего в эту единицу, выполнить откат, чтобы вернуть данные в исходное состояние. Поясним на примере.

Необходимо снять деньги с одного счета и положить на другой. Для этого нужно выполнить два SQL-запроса: первый - снять деньги с одного счета, второй – зачислить их на другой счет. Если не применять транзакции, то в случае сбоя при выполнении второго запроса деньги будут сняты со счета, но не будут зачислены на другой счет. Применение транзакций позволяет сделать откат, как если бы деньги вообще не снимались со счета.

Заметим, что при помощи команды LOCK TABLES в MySQL можно эмулировать транзакцию. Эта команда блокирует таблицу на время выполнения запросов, и тем самым обеспечивается целостность данных, но откат все равно нельзя сделать.

Триггеры – служат для автоматизации контроля за состоянием и работой базы данных. Триггер хранится в базе и срабатывает, когда происходит определенное событие. Возьмем тот же пример с переводом денег: в случае сбоя при выполнении второго запроса сработает триггер, который выполнит откат либо пошлет сообщение администратору базы данных.

Хранимые процедуры - это несколько SQL-команд, которые хранятся в базе данных под неким именем и в совокупности выполняют некую функцию. При помощи хранимых процедур можно расширить синтаксис SQL так, что он будет похож на обычный язык программирования (например, Oracle PL/SQL). В нашем примере с переводом денег два SQL-запроса можно было бы сохранить под одним именем, а потом вызвать эту процедуру, передав ей в качестве параметров два номера счета и сумму денег. Тогда оба запроса выполнялись бы в одной транзакции.

Вложенные запросы - позволяют подставлять значения в условия отбора динамически, по результатам выполнения другого запроса. По мнению автора, если без всего вышеперечисленного еще можно как-то обойтись, то отсутствие вложенных запросов иногда очень портит жизнь.

MySQL – свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

MySQL имеет двойное лицензирование [21]. MySQL может распространяться в соответствии с условиями лицензии GPL. Однако по условиям GPL, если какая-либо программа использует библиотеки (или включает в себя другой GPL-код) MySQL, то она тоже должна распространяться по лицензии GPL. Это может расходиться с планами разработчиков, не желающих открывать исходные тексты своих программ. Для таких случаев предусмотрена коммерческая лицензия, которая также обеспечивает качественную сервисную поддержку. Для свободного программного обеспечения Oracle предоставляет отдельное исключение из правил, явным образом разрешающее использование и распространение MySQL вместе с ПО, распространяемым под лицензией из определённого Oracle списка.

MySQL портирована на большое количество платформ: AIX, BSDi, FreeBSD, HP-UX, Linux, macOS, NetBSD, OpenBSD, OS/2 Warp, SGI IRIX, Solaris, SunOS, SCO OpenServer, UnixWare, Tru64, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Server 2003, WinCE, Windows Vista, Windows 7 и Windows 10. Существует также порт MySQL к OpenVMS. Важно отметить, что на официальном сайте СУБД для свободной загрузки предоставляются не только исходные коды, но и откомпилированные и оптимизированные под конкретные операционные системы готовые исполняемые модули СУБД MySQL. [18]

Взаимодействие между основным и вспомогательным приложением было решено осуществлять при помощи протокола HTTP.

HTTP – протокол прикладного уровня передачи данных изначально – в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных. Основой HTTP является технология клиент-сервер, то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, а также поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов. HTTP используется также в качестве транспорта для других протоколов прикладного уровня, таких как SOAP, XML-RPC, WebDAV.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку (в частности, для этого используется HTTP-заголовок). Именно благодаря возможности указания способа кодирования сообщения, клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

HTTP – протокол прикладного уровня; аналогичными ему являются FTP и SMTP [22]. Обмен сообщениями идёт по обыкновенной схеме запрос-ответ. Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами запрос-ответ. Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, куки на стороне клиента, сессии на стороне сервера). Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

В качестве фреймворка для хранилища ключ-значения был выбран Sinatra [23]. Основные преимущества данного фреймворка:

* небольшой фрейморк для небольших проектов;
* прост в изучении и применении для решения узкоспециализированных задач;
* работает со многими библиотеками языка Ruby [24].

В качестве хранилища ключ-значение для вспомогательного приложения был выбран Redis.

Redis – это быстрое хранилище данных типа ключ‑значение в памяти с открытым исходным кодом для использования в качестве базы данных, кэша, брокера сообщений или очереди. Проект возник, когда Сальваторе Санфилиппо, первоначальный разработчик Redis, пытался улучшить масштабируемость стартапа в Италии. Redis обеспечивает время отклика на уровне долей миллисекунды и позволяет приложениям, работающим в режиме реального времени, выполнять миллионы запросов в секунду. Такие приложения востребованы в сфере игр, рекламных технологий, финансовых сервисов, здравоохранения и IoT. Redis широко применяется для кэширования, управления сеансами, разработки игр, создания таблиц лидеров, аналитики в режиме реального времени, работы с геопространственными данными, поддержки служб такси, чатов и сервисов обмена сообщениями, потоковой передачи мультимедиа и приложений с отправкой сообщений по модели издатель – подписчик (Pub/Sub).

Все данные в Redis хранятся в памяти, а не на дисках или твердотельных накопителях, как в других базах данных. Поскольку Redis, как и другие хранилища данных в памяти, не нуждается в доступе к диску, это исключает задержки, связанные с поиском, и обеспечивает доступ к данным за микросекунды. В число возможностей Redis входит поддержка разнообразных структур данных, обеспечение высокой доступности, работа с геопространственными данными, создание скриптов Lua, проведение транзакций, постоянное хранение данных на диске и поддержка кластеров. Все это упрощает создание приложений, работающих в режиме реального времени в масштабе всего Интернета.

Как Redis, так и MemCached представляют собой хранилища данных в памяти с открытым исходным кодом. Высокопроизводительный сервис кэширования с распределенной памятью Memcached отличает простота, а Redis обладает широкими функциональными возможностями, которые позволяют эффективно использовать хранилище для разнообразных целей. Подробное сравнение функций, которое поможет вам принять решение, см. по ссылке Сравнение Redis и Memcached. Они используются с реляционными базами данных или базами данных на основе пар ключ – значение, такими как MySQL, Postgres, Aurora, Oracle, SQL Server, DynamoDB и многими другими, для повышения производительности.

Все данные Redis находятся в основной памяти сервера, в отличие от таких баз данных, как PostgreSQL, Cassandra, MongoDB и других, которые большую часть данных хранят на магнитных дисках или SSD‑накопителях. По сравнению с традиционными дисковыми базами данных, требующими циклического обращения к диску для большинства операций, хранилища данных в памяти, такие как Redis, свободны от этого ограничения. Благодаря этому многократно увеличивается количество выполняемых операций и сокращается время отклика. В результате обеспечивается чрезвычайно высокая производительность. Операции чтения или записи в среднем занимают менее миллисекунды, скорость работы достигает миллионов операций в секунду.

Redis упрощает код, позволяя писать меньше строк для хранения, использования данных и организации доступа к данным в приложениях. К примеру, если приложение содержит данные, хранящиеся в хэш‑таблице, и требуется сохранить эти данные в хранилище, можно просто использовать структуру данных хэш‑таблицы Redis. Решение подобной задачи с использованием хранилища данных, не поддерживающего структуры хэш‑таблиц, потребует написания серьезного объема кода для преобразования данных из одного формата в другой. Redis уже оснащен встроенными структурами данных и предоставляет множество возможностей их комбинирования и взаимодействия с данными клиента. Разработчикам под Redis доступны более ста клиентов с открытым исходным кодом. Поддерживаемые языки программирования включают Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go и многие другие.

В Redis применяется архитектура узлов ведущий‑подчиненный и поддерживается асинхронная репликация, при которой данные могут копироваться на несколько подчиненных серверов. Это обеспечивает как улучшенные характеристики чтения (так как запросы могут быть распределены между серверами), так и ускоренное восстановление в случае сбоя основного сервера. Для обеспечения постоянного хранения Redis поддерживает снимки состояния на момент времени (копирование наборов данных Redis на диск).

Redis прекрасно подходит для организации высокодоступного кэша в памяти, который уменьшает задержку доступа, увеличивает пропускную способность и снижает нагрузку на реляционную базу данных или базу данных NoSQL и на приложение. Redis может обеспечить доступ к часто запрашиваемым данным с задержкой в доли миллисекунды и позволяет с легкостью выполнять масштабирование, справляясь с повышением нагрузок без дорогостоящего наращивания мощности БД на уровне сервера. Типовые примеры использования Redis – это кэширование результатов запросов к базе данных, долговременных сессий, веб‑страниц или часто используемых объектов, таких как изображения, файлы и метаданные.

Redis поддерживает системы издатель – подписчик с заданными шаблонами и различные структуры данных, такие как списки, сортированные множества и хэш‑таблицы. Это позволяет использовать Redis для создания высокопроизводительных комнат чата, лент комментариев, работающих в режиме реального времени, лент новостей в социальных сетях и систем взаимодействия серверов. Структура данных список сервиса Redis позволяет легко создавать упрощенные очереди. Списки Redis обеспечивают выполнение элементарных операций, а также возможности блокировки, поэтому подходят для различных приложений, в которых требуется надежный брокер сообщений или циклический список.

Redis как хранилище данных в памяти с высокой доступностью и долговременным хранением широко применяется для хранения данных сессий в приложениях, работающих в масштабе всего Интернета, а также для управления такими данными. Redis обеспечивает задержку на уровне долей миллисекунды, масштабируемость и отказоустойчивость, необходимые для управления такими данными сессий, как профили пользователей, учетные данные, состояние сессий и индивидуальные пользовательские настройки. [3]

## **4.2 Сторонние библиотеки**

При разработке на языке программирования Ruby принято придерживаться компонентного подхода. Компонентом в терминах ruby называют гем. Для управления пакетами стандартно в Rails используется RubyGems – система управления пакетами для языка программирования Ruby, который предоставляет стандартный формат для программ и библиотек Ruby (в самодостаточном формате), инструменты, предназначенные для простого управления установкой гемов и сервер для их распространения.

В ходе разработки веб-сервиса были использованы следующие Ruby библиотеки, значительно ускоряющие разработку:

1. annotate необходим для добавления аннотаций в моделях приложения на Rails, создает наглядность базы данных при разработке приложения.

2. bootsnap библиотека необходима для оптимизации работы Rails приложения, а также для его более быстрого старта.

3. cancancan библиотека используется для обеспечения безопасности. При помощи неё организуется авторизация пользователь на определенные действия в приложении. Позволяет задать роли в одном файле ability и не дублировать их в моделях, контроллерах и представлениях, оставляя их код чистым, храня логику управления правами в одном месте. Состоит из двух частей: библиотека авторизации, которая определяет права доступа пользователя к определенным объектам и вспомогательные (так называемые helper) методы для контроллеров, которые помогают держать в контроллере минимум кода.

4. coffee-rails это адаптер для Rails, который позволяет компилировать файлы coffee и преобразовывать их в javascript, который поддерживается всеми современными браузерами.

5. devise это гибкое решение аутентификации для Rails на основе Warden. Основные его преимущества: основа на Rack (высокая совместимость с Ruby приложениями), целостное MVC решение для аутентификации, позволяет иметь несколько моделей аутентифицированных одновременно, полностью состоит из модулей и позволяет отключать ненужные или подключать необходимые модули. Состоит из 10 модулей.

6. devise-bootstap-views библиотека является дополнением к библиотеке devise, позволяет добавить bootstrap на страницы регистрации и авторизации пользователей. Это необходимо для создания современного адаптивного дизайна.

7. font-awesome-rails добавляет поддержку иконок из библиотеки, называющейся font awesome.

8. jbuilder это язык для описания JSON структур, которые приложение сможет отдавать в удобном для программиста виде.

9. jquery-rails добавляет поддержку jQuery в Rails приложение. Используется в паре с механизмом обработки вспомогательных файлов.

10. less-rails добавляет в Rails приложение поддержку синтаксиса less. Используется в паре с библиотекой bootstrap

11. mysql2 это адаптер, который необходим для работы Rails приложения с базой данных MySQL. Осуществляет подключение к базе данных с необходимыми конфигурационными параметрами.

12. puma веб-сервер, который стандартно поставляется вместе с новыми версиями Rails приложений.

13. recaptcha библиотека необходима для подключения к сервису Recaptcha от компании Google. Используется на этапе регистрации для защиты от роботов.

14. redis библиотека необходима для работы с хранилищем Redis. Обычно используется в качестве кеша, для работы фоновых задач в Rails.

15. rest-client библиотека необходима для связи с отдельным приложением ключ-значение. Используется как на стороне приложения, так и на стороне ключ-значение хранилища, обеспечивая их связь.

16. rolify это пакет, позволяющий добавить управление ролями в веб- приложение. Имеет функциональность проверки ролей для пользователей и не реализует элементы авторизации.

17. russian библиотека необходима для интеграции русского языка в приложение Rails.

18. slim библиотека, добавляющая поддержку языка slim. Данный язык используется для разметки представлений в Rails приложениях.

19. twitter-bootstrap-rails необходим для интеграции фреймворка bootstrap в файлопровод Rails приложения. Bootstrap – это набор инструментов от Twitter предназначенных для начала разработки веб-приложений и сайтов. Он включает базовые CSS и HTML для типографии, форм, кнопок, таблиц, сеток и навигации.

20. byebug средство для удобной отладки приложений на Ruby. Позволяет создавать надежные приложения.

21. turbolinks библиотека для создания эффекта одностраничных приложений. Позволяет в существующее приложения добавить возможность работать с приложением без перезагрузки страниц в браузере. Является библиотекой javascript, которая осуществляет работу по сохранению истории, изменяет принципы рендеринга страниц.

22. uglifier библиотека, необходимая для компиляции всех дополнений в веб-приложении.

23. sinatra небольшой фреймворк для быстрого создания небольших приложений. Позволяет создавать гибкие приложения на Ruby.

24. foreman необходим для управления процессами, такими как ruby, webpack на локальной машине разработки и на хостинге.

25. connection\_pool библиотека используется для ограничения одновременного обращения к таким ресурсам, как redis. [21]

Данные сторонние библиотеки доступны для скачиваний бесплатно через стандартный менеджер пакетов языка Ruby.

## **4.2 Алгоритмы работы веб-сервиса**

В основе веб-сервиса лежат следующие алгоритмы работы: алгоритм аутентификации, алгоритм репликации данных, алгоритм добавления нового вопроса, алгоритм подбора ответа, алгоритм взаимодействия с базой данных, алгоритм ранжирования ответа.

Алгоритм аутентификации:

Для осуществления аутентификации в приложении используется библиотека devise. После ввода формы аутентификации, пользовательские данные поступают в модуль для аутентификации devise. Происходит хеширование введенного пользователем пароля и далее происходит поиск пары почта-пароль среди всех пользовательских записей в базе данных. Если пользователь ввел правильным только один из параметров (электронную почту или пароль), то модуль вернет ошибку, в которой не будет указано конкретно поле, в котором ошибся пользователь. Это необходимо для того, чтобы не сообщить злоумышленнику информацию о зарегистрированных пользователях.

Алгоритм репликации данных:

Для осуществления репликации данных используется вспомогательное приложение. После того, как на основную базу данных (master) приходит запрос на управление какой-либо сущностью, например, добавление или удаление определенной записи. Сначала необходимая операция выполняется на локальной базе данных. Далее, в случае успешного выполнения операции, происходит HTTP запрос на приложение-реплику с запросом на выполнение определенной операции.

Далее master сервер получает ответ от реплики в синхронном режиме. Это позволяет быть уверенным в том, что сервер репликации онлайн и успешно выполняет приходящие запросы. Если операция выполнена успешно, master сервер возвращает результат выполнения своей локальной операции. В случае ошибок master сервер уведомляет источник запроса об ошибках.

Добавление нового вопроса:

После добавления вопроса пользователем в базу данных, его необходимо разбить на части, проиндексировать и сохранить в удобном для дальнейшей работы виде. Вопрос необходимо обработать следующим образом.

Сначала необходимо убедиться в том, что пользователь ввел все параметры вопроса правильно. Далее, если все параметры верные, необходимо сохранить вопрос в реляционную базу данных и получить уникальный идентификатор вопроса, который будет использоваться в дальнейшем в ходе работы с хранилищем ключ-значение.

После получения идентификатора вопроса, текст вопроса необходимо разбить на слова и удалить все символы, кроме букв и цифр из входной строки. Для каждого слова из полученного массива слов необходимо рассчитать хеш-сумму.

Алгоритм подсчёта хеш-суммы одновременно должен быть несложным в вычислениях, быстрым, компактным и обеспечивать уникальную идентификацию файла на основе его содержимого. В качестве такого алгоритма был выбрал MD5.

Далее, после того как был получен массив из хешей, необходимо для каждого хеша произвести выборку записей из хранилища ключ-значение.

Необходимо проверить, было ли уже добавлено определенное слово в хранилище. Если да, то новый идентификатор вопроса необходимо просто дописать в полученную выборку и после записать данное значение в базу ключ-значение. Если слово новое для системы, необходимо создать массив и произвести запись в хранилище. Данную операцию необходимо произвести для каждого слова из вопроса.

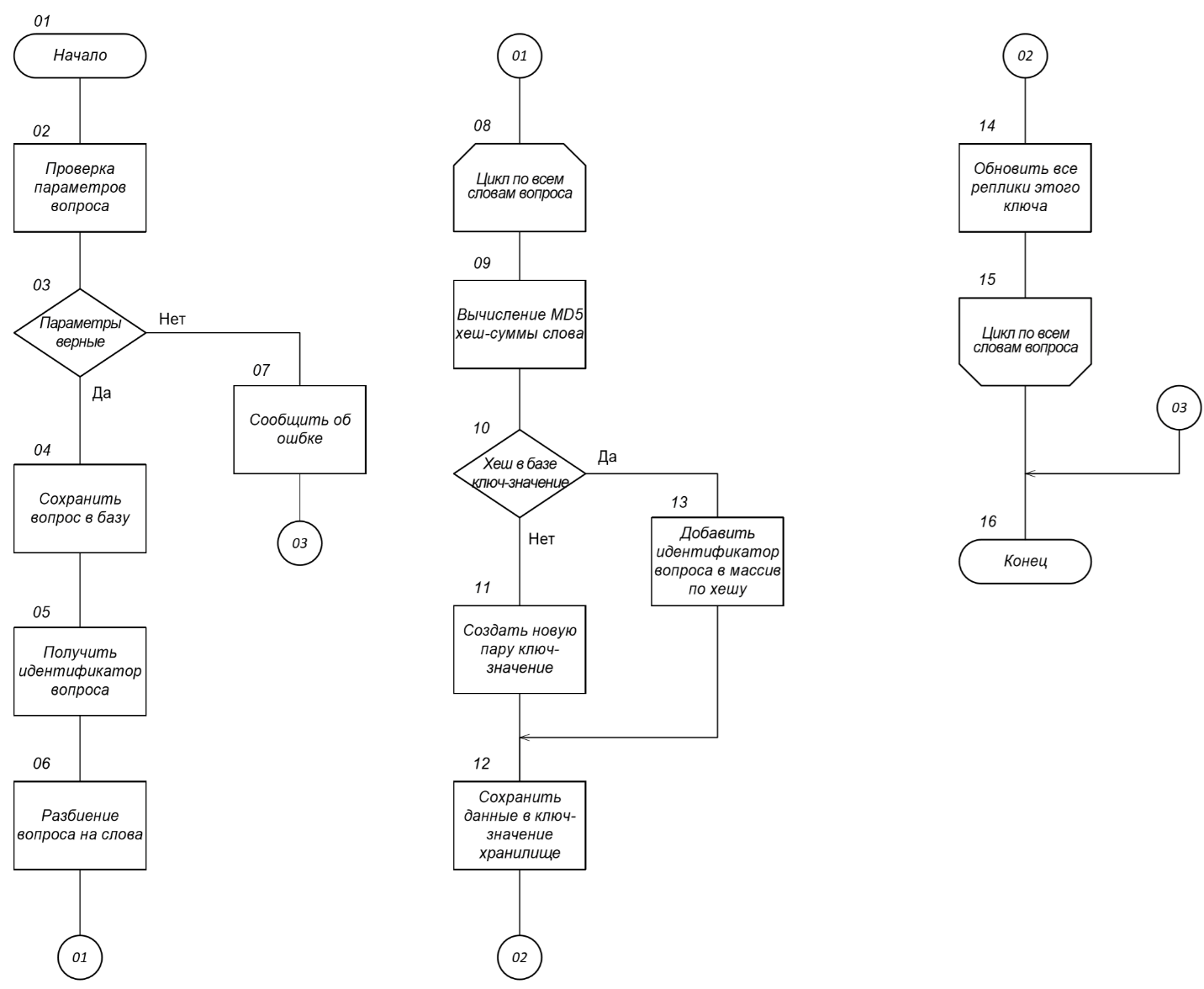
После сохранения определенных данных в хранилище ключ-значение происходит синхронная репликация данных на второстепенные сервера. Тип репликации выбран синхронный, так как это позволяет быть уверенным на момент записи, что данные доставлены и сохранены во всех репликах. Алгоритм добавления вопроса представлен на рисунке 4.1.

Рисунок 4.1 – Алгоритм добавления нового вопроса

Алгоритм подбора ответа:

После задания вопроса, у пользователя появляется возможность мгновенно найти ответ на свой вопрос, если он уже был задан и отвечен на сервисе.

Сначала необходимо разбить исходный вопрос на слова и сформировать хеши аналогично тому, как это происходит в процессе записи вопроса. После происходит выборка всех идентификаторов вопросов по данным словам. Далее все идентификаторы вопросов группируются. В каждой группе производится подсчет количества идентификаторов. Выбирается идентификатор с максимальным количеством вхождений в группе. Алгоритм поиска похожего ответа представлен на рисунке 4.2.

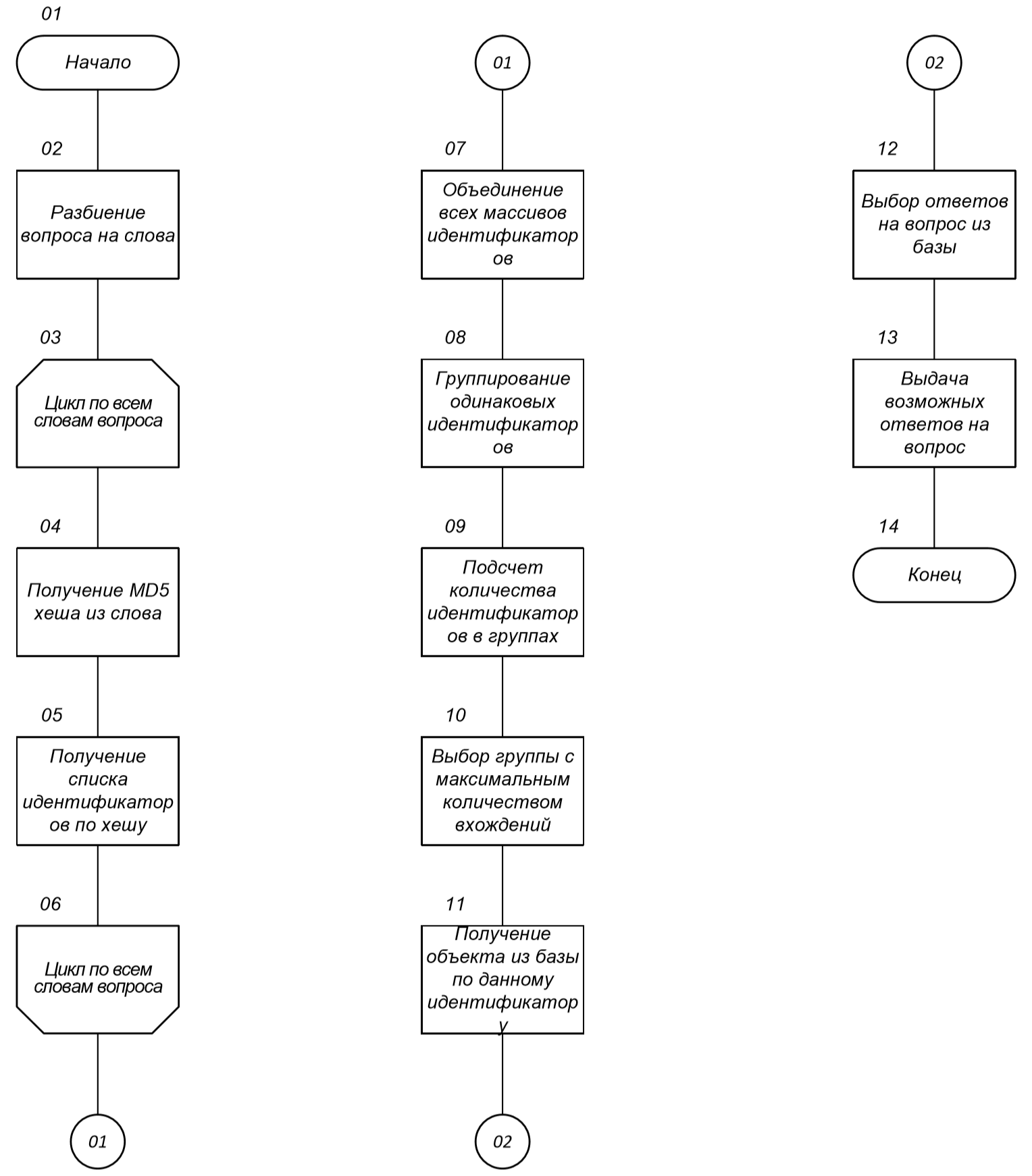


Рисунок 4.2 – Алгоритм поиска похожего ответа

После получения идентификатора похожего ответа, необходимо определить подходящий ответ. Приложение сначала попытается выбрать ответ с максимальным количеством голосов от автора. Если автор не отменил ни один из вопросов, будет произведена попытка выбрать ответ с максимальным количеством голосов. Если на вопрос еще никто не отвечал, алгоритм в итоге вернет пустой ответ.

Алгоритм взаимодействия с базой данных:

За взаимодействие с базой данных частично отвечает модуль ActiveRecord, часть стандартной поставки Rails фреймворка.

На этапе запуска приложения любым из способов происходит процесс установления соединения с базой данных. Далее, если приложению требуется записать некоторый объем данных, оно использует уже установленное соединение. Алгоритм представлен на рисунке 4.3.

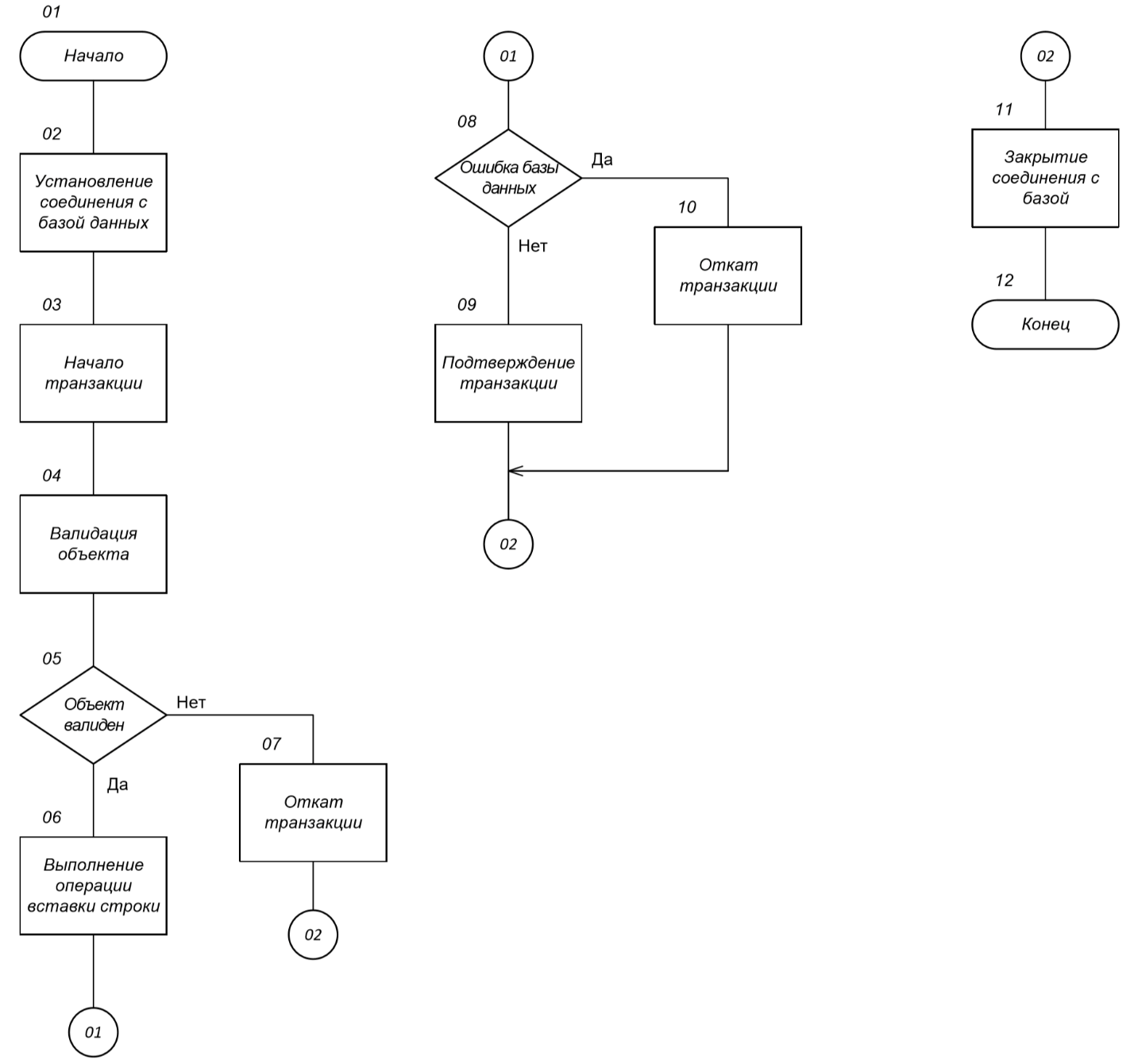


Рисунок 4.3 – Алгоритм взаимодействия с базой данных

Перед поступлением данных в базу данных, их необходимо проверить на правильность сначала на уровне приложения. Для этого входные данные сначала проверяются на соответствие правилам из модели приложения. Если данные успешно проходят проверку, они передаются дальше. Если в процессе проверки правильности данных возникли ошибки, ошибки выводятся в виде исключений.

Так как используется транзакционная база данных MySQL с движком InnoDB с включенным режимом autocommit, который предполагает выполнение всех операций в транзакциях, необходимо начать выполнение транзакции на уровне базы данных.

Далее необходимо произвести сохранение данных в таблицу базы данных. Название таблицы соответствует названию модели в приложении. Если база данных вернула успешный ответ, то есть сохранение прошло без проблем, необходимо завершить транзакцию и вернуть пользователю ответ. В случае, если в процессе сохранение информации в базе данных возникли ошибки, происходит откат транзакции, то есть все внесенные изменения в рамках текущей транзакции отбрасываются.

Алгоритм ранжирования ответа:

При выводе ответов пользователей на определенный вопрос необходимо правильно отсортировать ответы, чтобы наиболее полезные ответы находились в самом верху страницы с ответом.

Если на вопрос имеются ответы, отмеченные автором вопроса, это означает, что данные вопросы в полной мере решили проблему автора. Данные ответы сортируются по убыванию количества голосов и выводятся в списке вопросов на первых местах.

Далее, если на данный вопрос имеются ответы, которые автор по определенным причинам не отметил, то они сортируются по убыванию количества голосов и выводятся под отмеченными автором ответами. Сортировка по количеству ответов происходит на стороне базы данных, так как в таблице с ответами для этого имеется колонка с количеством голосов пользователей.

При поиске похожего вопроса, а соответственно происходит подбор похожего ответа, происходит поиск возможного ответа по данному алгоритму, но за одним исключением. При поиске похожего вопроса происходит выборка только первого ответа. То есть это может быть, как ответ, отмеченный автором с максимальным количеством голосов, так и ответ, не отмеченный автором, но с наибольшим количеством голосов. Если ни один из ответов автор не отметил, поиск будет осуществляться на основе мнения сообщества веб-сервиса. В данном случае будет выведен ответ с максимальным количеством голосов.

## **4.3 Особенности реализации веб-сервиса**

Для подсчета количества голосов у ответа, количества вопросов, заданных пользователем и количества его ответов, необходимо каждый раз делать запрос в базу данных для выборки данной информации.

В Rails имеется механизм, позволяющий создать кеширующие колонки в базе данных, которые будут автоматически обновляться при обновлении или добавлении соответствующих связанных сущностей.

При помощи Rails можно создать специальное кеширующее поле, которое позволит нам не вычислять значение счетчика динамически каждый раз при обращении к профилю пользователя, а брать как есть из дополнительной колонки в таблице пользователей. При этом Rails обеспечивает корректность данного счетчика.

Для легкости управления файлами JavaScript и CSS было решено использовать Rails asset pipeline для компиляции ассетов в production окружении [25]. В процессе разработки приложения компиляция осуществляется на лету для обеспечения мгновенной перезагрузки веб-приложения в локальном браузере. Но компиляция ассетов занимает довольно много ресурсов процессора, поэтому общепринятой практикой является прекомпиляция данных файлов в рабочем окружении. Процесс компиляции происходит только один раз во время загрузки приложения. Затем данные файлы могут раздаваться при помощи content delivery network сети. Данный подход позволяет ускорить загрузку ассетов, уменьшить расстояние от пользователя до необходимых файлов, а также не нагружать основное приложение компиляцией. В рабочих окружениях ассеты не изменяются в процессе работы, поэтому версия на момент загрузки приложения является актуальной до нового релиза.

Первой особенностью asset pipeline является соединение ассетов, что может уменьшить количество запросов, необходимых браузеру для отображения страницы. Браузеры ограничены в количестве запросов, которые они могут выполнить параллельно, поэтому меньшее количество запросов может означать более быструю загрузку вашего приложения.

Sprockets соединяет все JavaScript файлы в один главный файл .js и все CSS файлы в один главный файл .css. Можно настроить эту стратегию, сгруппировав файлы любым способом. В production, Rails вставляет метку SHA256 в каждое имя файла, таким образом файл кэшируется браузером. Кэш можно сделать недействительным, изменив эту метку, что происходит автоматически каждый раз, когда изменяется содержимое файла.

Второй особенностью asset pipeline является минимизация или сжатие ассетов. Для файлов CSS это выполняется путем удаления пробелов и комментариев. Для JavaScript могут быть применены более сложные процессы. Можно выбирать из набора встроенных опций или определить свои.

Третьей особенностью asset pipeline является то, что он позволяет писать ассеты на языке более высокого уровня с дальнейшей компиляцией до фактического ассета. Поддерживаемые языки по умолчанию включают Sass для CSS, CoffeeScript для JavaScript и ERB для обоих.

# **5 ТЕСТИРОВАНИЕ ВЕБ-СЕРВИСА**

Было проведено тестирования веб-сервиса с точки зрения пользовательского интерфейса, возможности взлома приложения, а также проверок входных данных на уровне базы данных.

В ходе проверки работы интерфейса приложения при помощи браузеров Safari, Firefox были выявлены некоторые недочеты в дизайне приложения для мобильных устройств. Они были устранены путем использования элементов адаптивной верстки фреймворка Bootstrap.

Для проведения тестирования приложения с точки зрения безопасности была использована программа Postman, позволяющая делать прямые HTTP запросы к приложению и тем самым определить уязвимости. Подробные результаты тестирования приведены в таблице 5.1.

Для тестирования безопасности приложения на уровне исходного кода была проведена проверка при помощи библиотеки brakeman. Данная библиотека ищет уязвимости в исходном коде приложения на любых стадиях разработки и позволяет указать разработчику на возможные слабые места приложения.

Таблица 5.1 – Тестирование безопасности веб-сервиса

|  |  |  |  |
| --- | --- | --- | --- |
| Запрос | Ответ | Ожидаемый ответ | Пояснение |
| GET /users | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 403 forbidden | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 403 forbidden | Гостевой пользователь не получил доступ к странице со списком пользователей. |
| POST /questions | Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 422 unprocessable | Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 422 unprocessable | Гостевой пользователь не может добавить вопрос. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Запрос | Ответ | Ожидаемый ответ | Пояснение |
| GET /questions/1 | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 200 OK | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 200 OK | Любой пользователь может получить доступ к ответу. |
| POST /categories | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 422 unprocessable | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 422 unprocessable | Пользователи, кроме администратора, не могут добавлять новые категории. |
| DELETE /questions/72 | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 422 unprocessable | Server: Cowboy  Connection: keep-alive  Content-Type: text/html  Cache-Control: no-cache  Transfer-Encoding: chunked  Status: 422 unprocessable | Гость не может удалить вопрос. |

Была проверена регистрация и авторизация (сервис recaptcha) на предмет уязвимостей. Таким образом, приложение безопасно для дальнейшей эксплуатации и использования.

# **6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Руководство пользователя включает системные требования к устройству, необходимые для установки и запуска приложения, а также подробную инструкцию по использованию приложения.

## **6.1 Системные требования и установка веб-сервиса**

Веб-сервис не предъявляет никаких особых требований к компьютеру пользователя. Для того, чтобы воспользоваться приложением подойдет любой современный браузер на любой современной популярной операционной системе.

Для разворачивания самого приложения необходима любая из сред Linux, наличие установленного Ruby в системе версии 2.5 и новее.

Дополнительные зависимости, которые были использованы в веб-сервисе:

– mysql версии 5.7;

– redis версии 5.0.

Для разворачивания приложения была выбрана платформа heroku. Сначала необходимо создать три Ruby приложения на данной платформе. Одно из них необходимо для основного приложения, два остальных являются хранилищами ключ-значение.

Для основного приложения необходимо добавить плагины MySQL и Redis. Данные операции можно выполнить при помощи графического интерфейса платформы heroku. После добавления плагинов необходимо установить переменные окружения REDISCLOUD\_URL и DATABASE\_URL для баз данных Redis и MySQL соответственно. Далее необходимо создать базу данных в MySQL, которая будет использоваться в приложении. Название для базы данных можно найти в конфигурационных файлах приложения.

Далее необходимо установить командную строку от heroku для локальной операционной системы. Для загрузки приложения на хостинг heroku необходимо добавить код в репозиторий проекта на heroku и выполнить git команду для загрузки кода в сервис heroku. После загрузки heroku в автоматическом режиме определит Rails приложение и выполнит все необходимые шаги для разворачивания приложения. После загрузки приложения будет проведена установка зависимостей при помощи менеджеров пакетов bundler и yarn. Далее будет проведена миграция базы данных MySQL, компиляция ассетов. После успешного выполнения всех автоматических шагов будет запущен бесплатный контейнер с приложением. В бесплатном режиме контейнер работает в течении 30 минут с последнего запроса. Далее контейнер уходит в режим сна. Как только приходит новый запрос в приложение, находящийся в режиме сна, веб-сервис запускается и приступает к обработке входящих пользовательских запросов. Heroku поддерживает интеграцию с современными системами continuous integration, поэтому непрерывную интеграцию приложения для его активной разработки можно развернуть достаточно быстро. После установки основного приложения необходимо при помощи консоли сервиса heroku создать пользователя администратора, а также назначить ему необходимую роль.

Рассмотрим разворачивание хранилища ключ-значения. Сначала добавим для обоих приложений плагин Redis. Данный плагин будет использоваться в качестве хранилища. Далее в оба приложения необходимо добавить переменную окружения REDIS\_URL с адресом только что созданной базы данных. В master приложение хранилища необходимо добавить переменную окружения SLAVE\_URL с адресом реплики. Данный адрес будет использоваться в приложении для осуществления репликации. После создания и конфигурации обоих приложений на платформе необходимо добавить код хранилища в оба репозитория и произвести его загрузку при помощи системы контроля версий git. После выполнения данных шагов веб-сервис будет готов к использованию.

## **6.2 Описание пользовательского интерфейса**

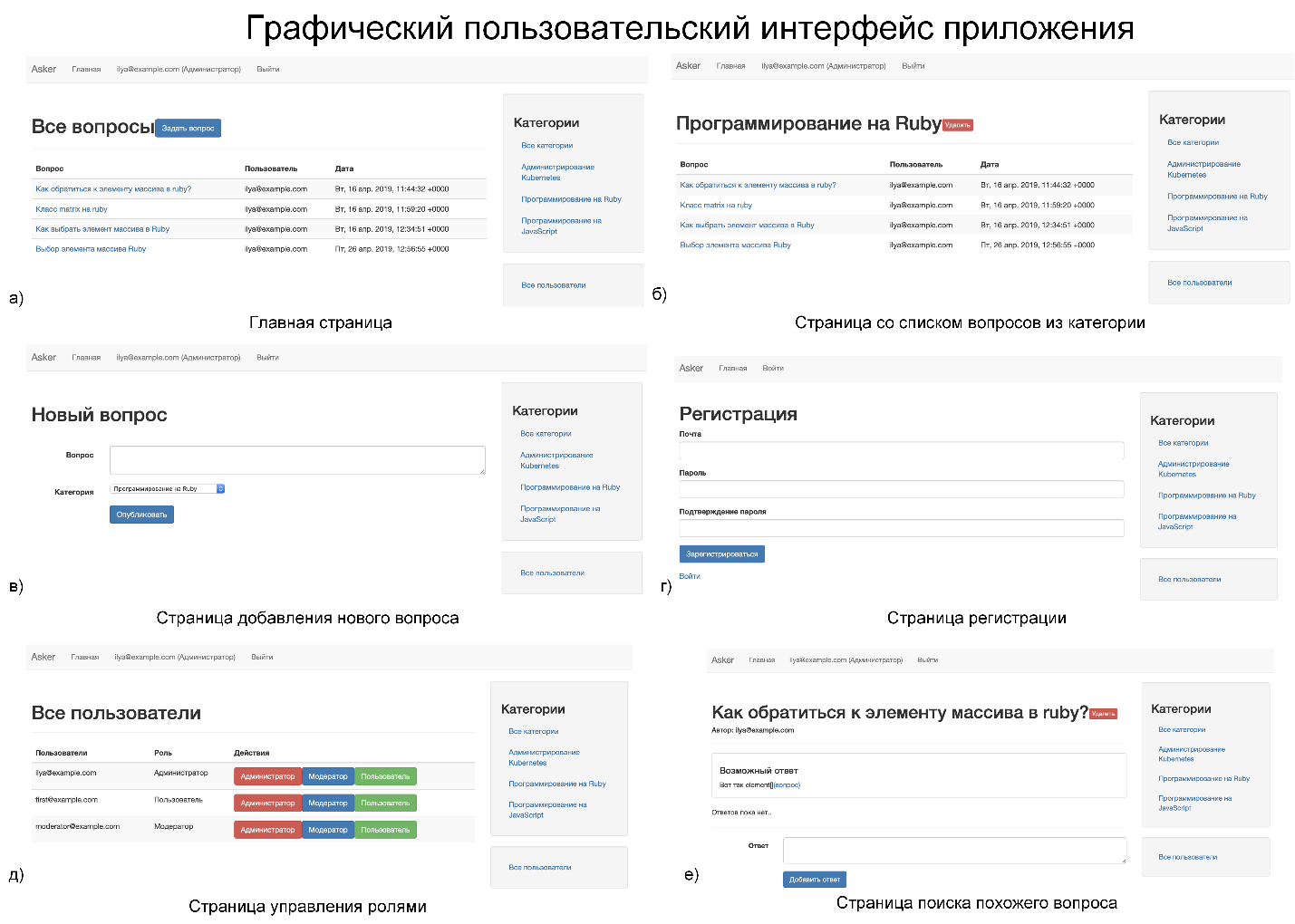
Пользовательский интерфейс веб-сервиса представлен на рисунке 6.1.

Рисунок 6.1 – Графический пользовательский интерфейс

Графический пользовательский интерфейс веб-сервиса предполагает следующие экранные формы:

* главная страница приложения показана на рисунке 6.1(а), на которой расположен список всех вопросов с сортировкой по дате добавления, список всех категорий, а также ссылка на список пользователей;
* страница категории показан на рисунке 6.1(б), на которой расположены вопросы из выбранной категории;
* страница добавления нового вопроса показана на рисунке 6.1(в), при помощи которой пользователь может добавить свой вопрос;
* страница регистрации показан на рисунке 6.1(г), на которой гости веб-сервиса могут создать новую учетную запись;
* страница управления ролями пользователей показа на рисунке 6.1(д), на которой администраторы веб-сервиса могут управлять ролями пользователей;
* страница поиска похожего вопроса показана на рисунке 6.1(е), на которой любой посетитель веб-сервиса может найти похожий возможный ответ на вопрос, представленный на странице.

## **6.3 Регистрация**

Для регистрации необходимо нажать кнопку «‎Зарегистрироваться» как показано на рисунке 6.2.

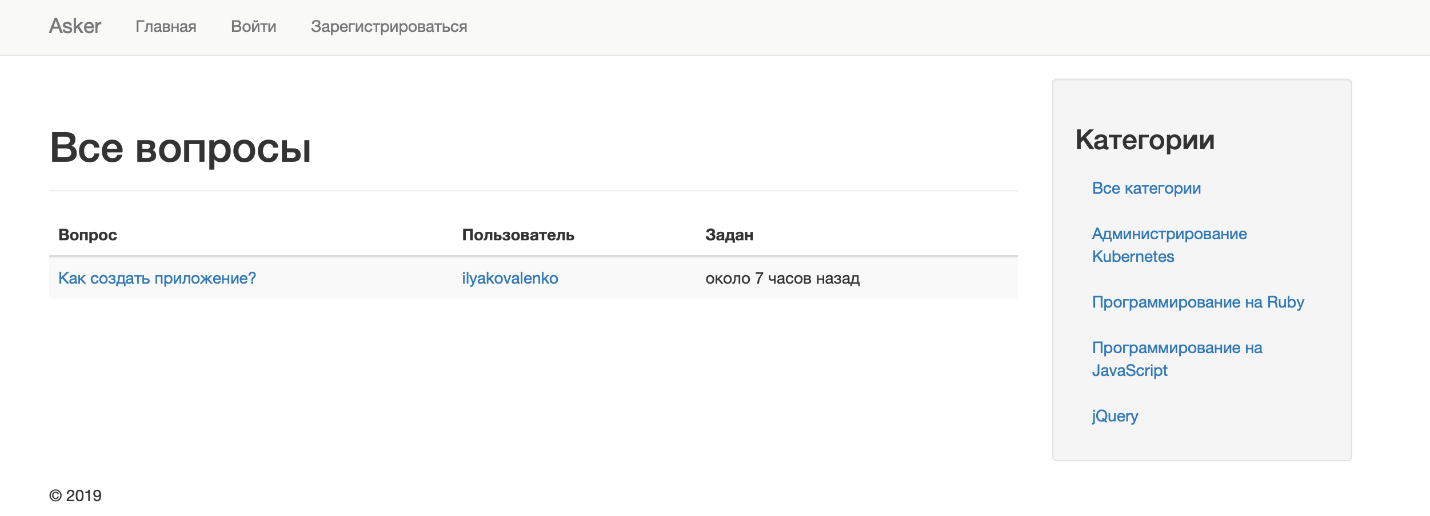


Рисунок 6.2 – Кнопка для прохождения регистрации

Далее необходимо ввести необходимые пользовательские данные в форму для регистрации. Необходимо осуществить ввод адреса электронной почты, логина для участия на сайте, а также пароль. Пароль необходимо ввести два раза, чтобы избежать случайных ошибок и опечаток при его вводе. Ограничение на длину пароля минимально 6 символов. Данная мера необходима, чтобы пользователи не выбирали очень простые пароли, а использовали пароли, которые проблематично подобрать при помощи автоматических программ. Для защиты сервиса от спама, на форме регистрации имеется защита от сервиса recaptcha. Форма регистрации показана на рисунке 6.3.

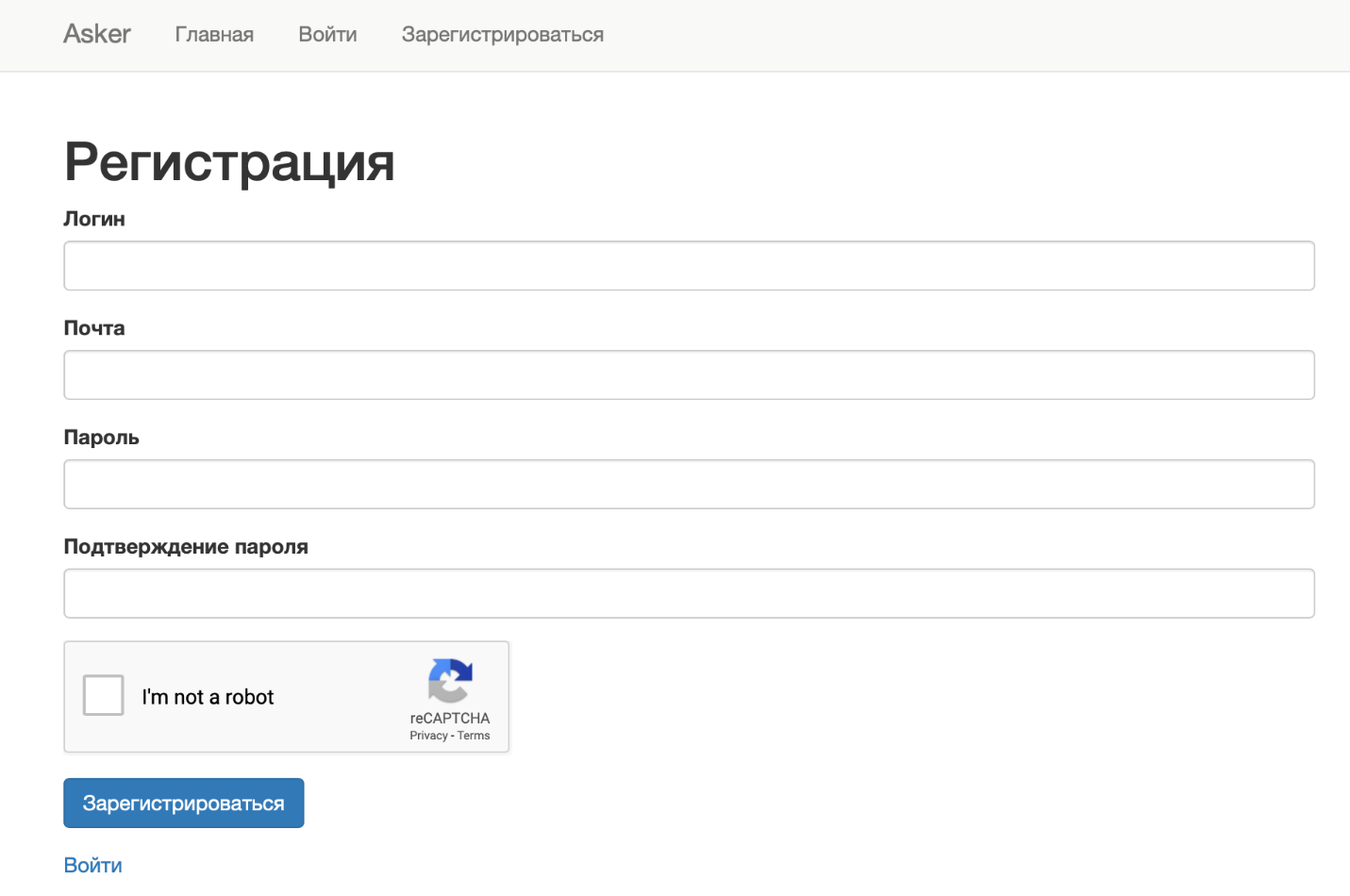


Рисунок 6.3 – Форма регистрации в веб-сервисе

## **6.4 Аутентификация**

После прохождения регистрации вы можете аутентифицироваться в веб-сервисе для дальнейшей работы над вопросами и ответами.

Аутентификация пользователей осуществляется при помощи электронной почты пользователя и пароля. Кнопка «‎Войти» показана на рисунке 6.4.

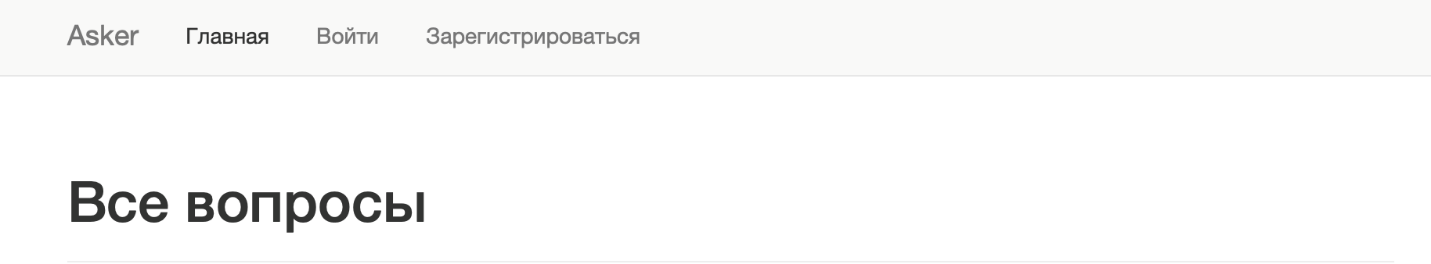


Рисунок 6.4 – Кнопка для аутентификации

После открытия страницы с аутентификацией, у пользователя появляется форма, в которую необходимо ввести логин и пароль. Если пользователь пользуется личным компьютером, он может воспользоваться функцией «‎Запомнить меня». В таком случае пользователь получит возможность быть несколько дольше аутентифицированным в веб-сервисе. Форма аутентификации представлена на рисунке 6.5.

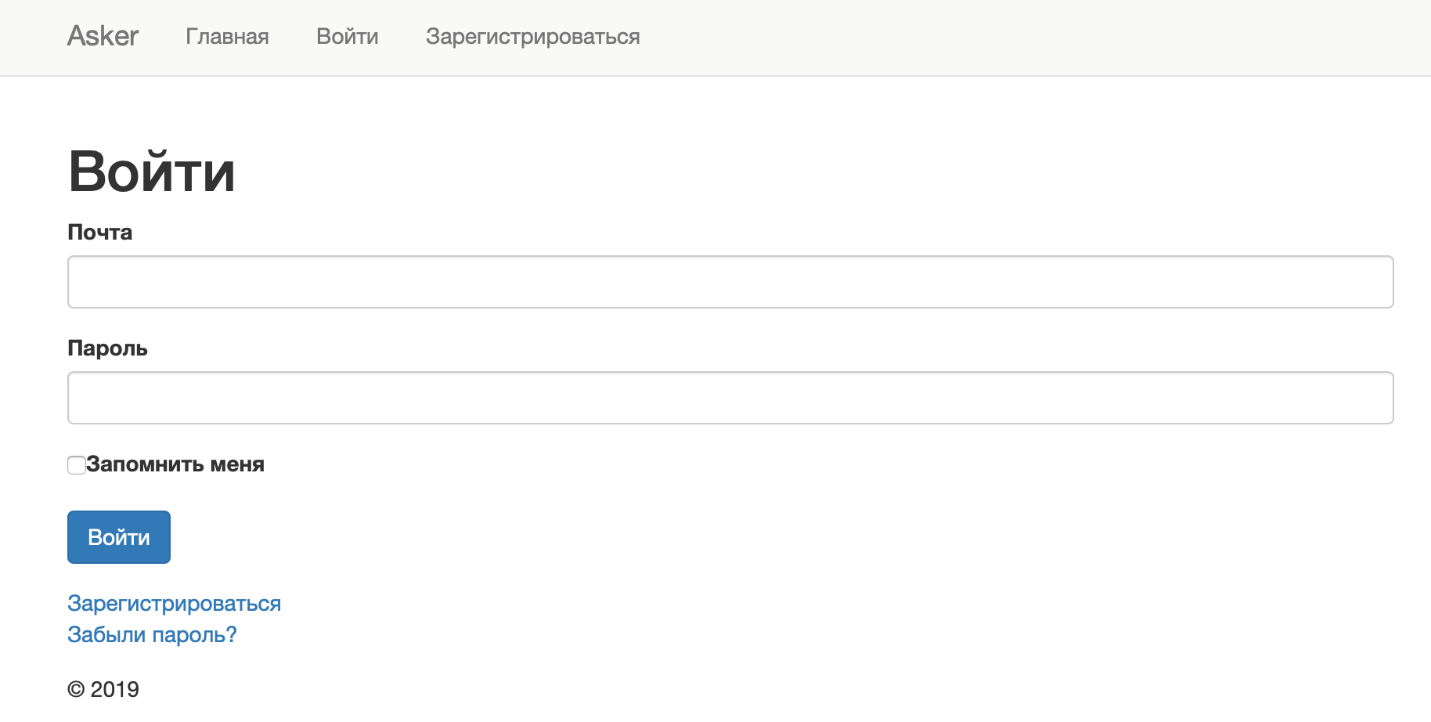


Рисунок 6.5 – Форма аутентификации

## **6.5 Управление вопросами**

Для управления вопроса сначала необходимо пройти регистрацию и аутентификацию на веб-сервисе. Для добавления нового вопроса необходимо нажать на кнопку «‎Задать вопрос» как показано на рисунке 6.6.

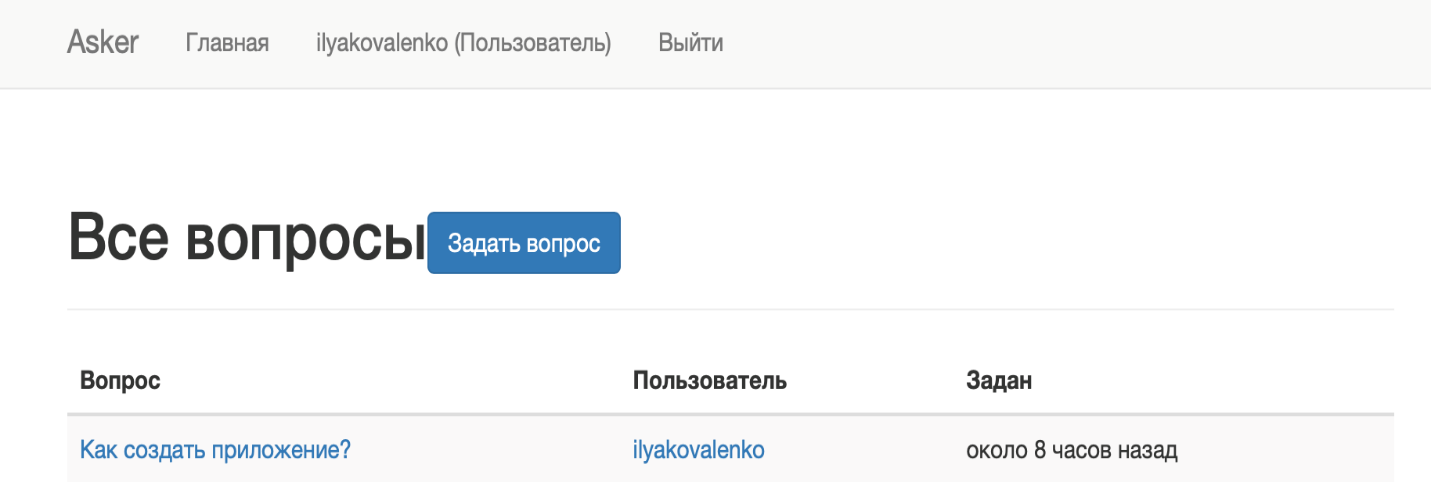


Рисунок 6.6 – Кнопка добавления нового вопроса

После нажатия на кнопку добавления нового вопроса открывается форма для добавления нового вопроса как показано на рисунке 6.7.

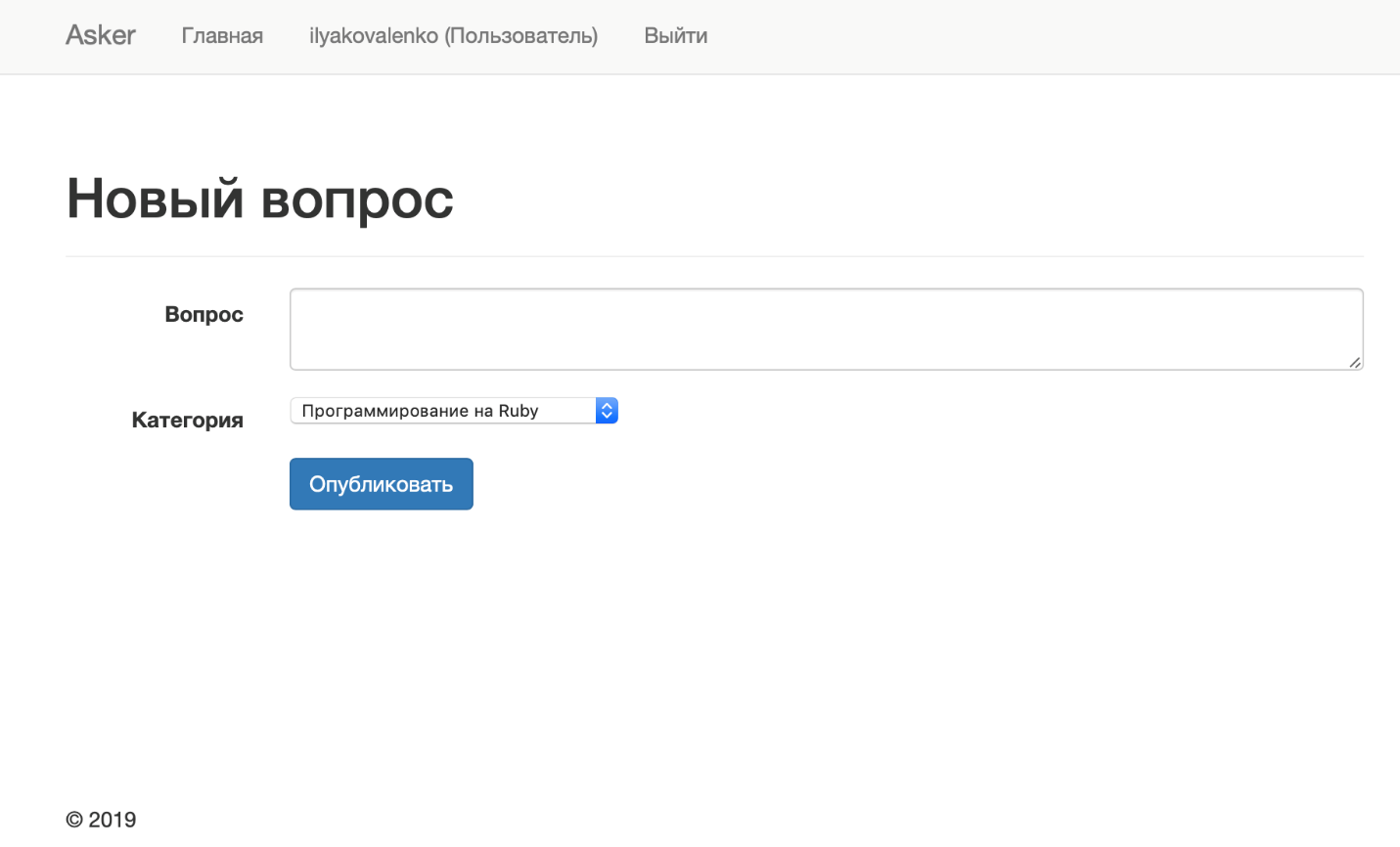


Рисунок 6.7 – Форма для добавления нового вопроса

После открытия формы добавления нового вопроса, вы можете заполнить форму с вопросом (задать свой вопрос) и выбрать категорию. Категории необходимы для того, чтобы классифицировать вопросы по определенной тематике. Это делает поиск вопросов определенной тематики проще.

Если заданный вопрос более не актуален либо был опубликован по ошибке, вы сможете удалить его при помощи кнопки «Удалить‎» как показано на рисунке 6.8.

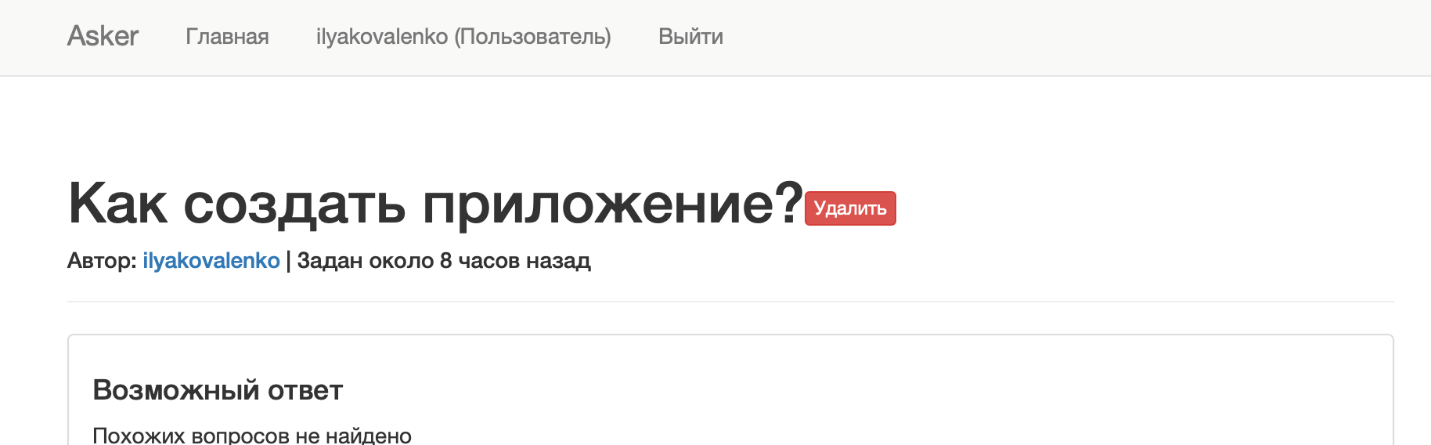


Рисунок 6.8 – Кнопка для удаления заданного вопроса

## **6.6 Управление ответами**

Когда некоторый пользователь задал свой вопрос, любой аутентифицированный посетитель веб-сервиса может написать ответ на определенный вопрос. Форма для добавления нового ответа представлена на рисунке 6.9.



Рисунок 6.9 – Форма для добавления ответа

Добавленный ответ выглядит как показано на рисунке 6.10.

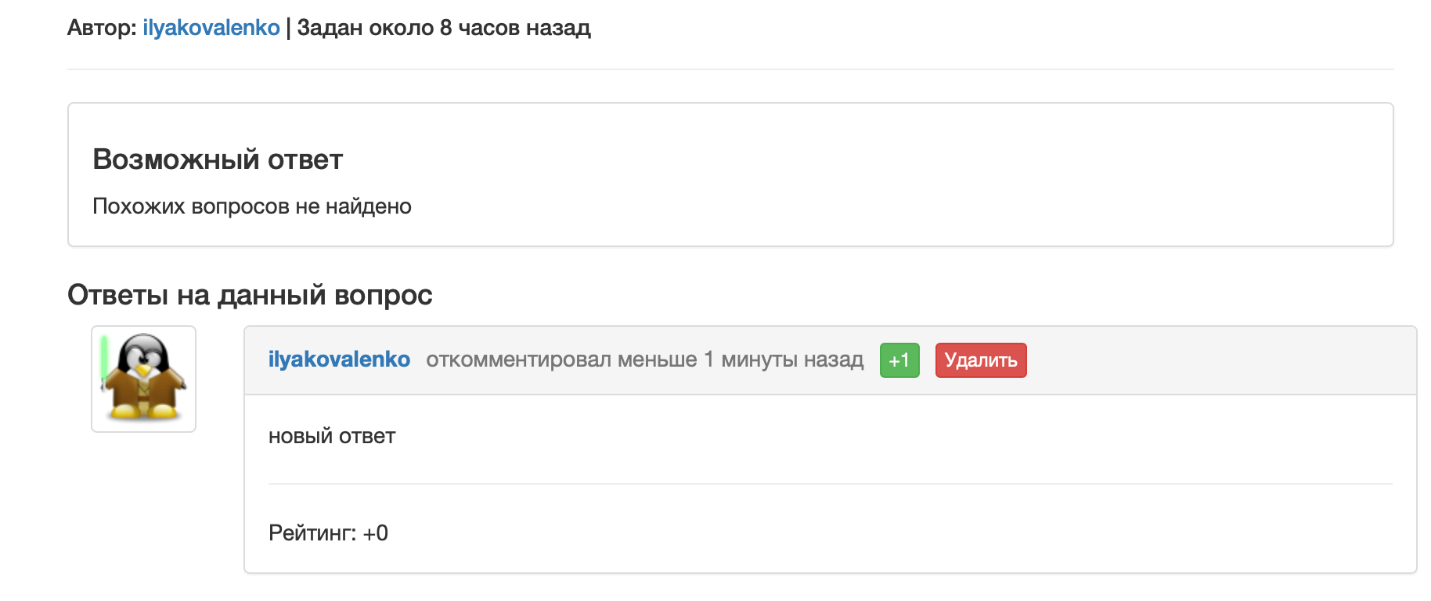


Рисунок 6.10 – Добавленный ответ

Для голосования за лучший ответ необходимо воспользоваться кнопкой «‎+1» как показано на рисунке 6.11.

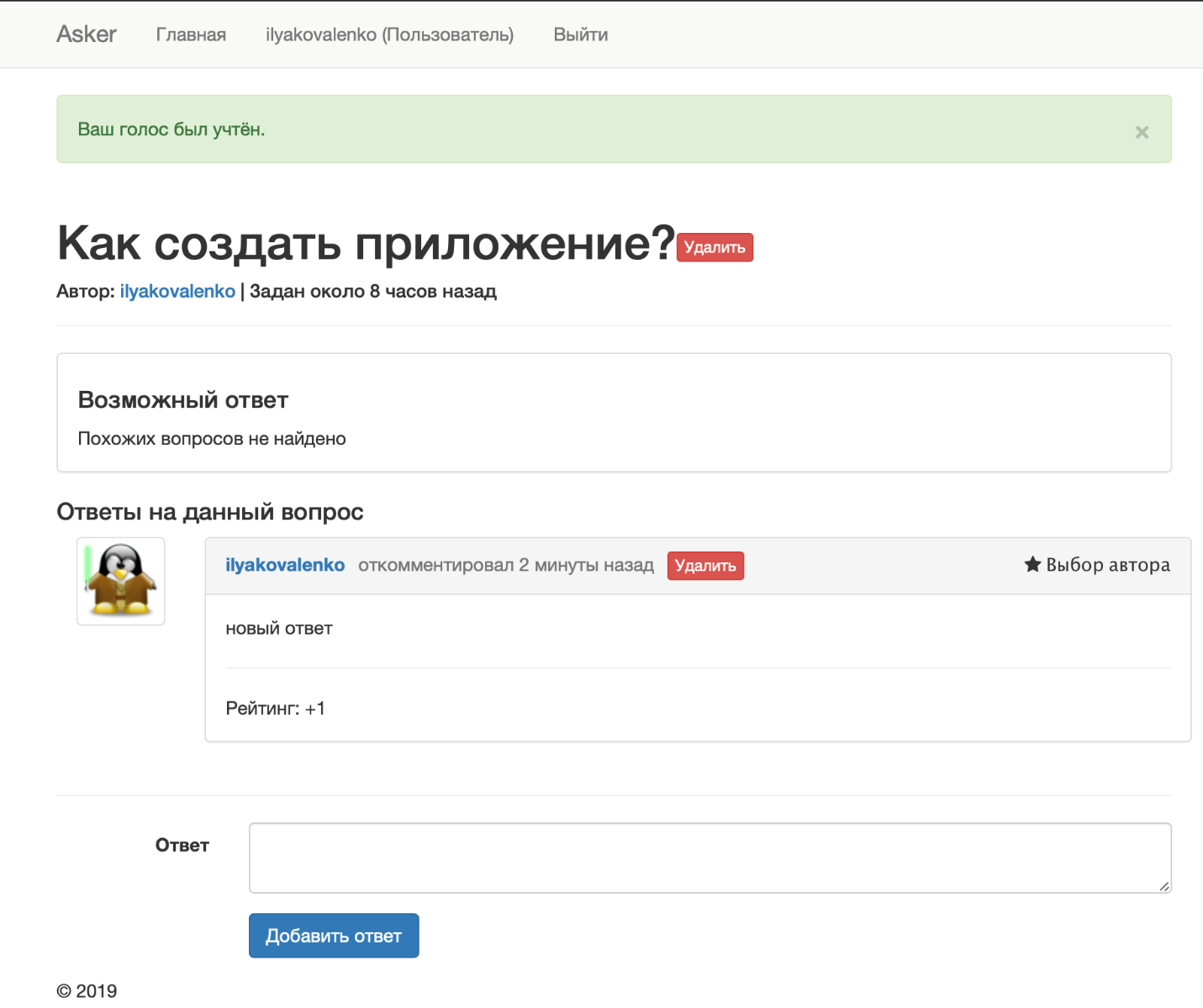


Рисунок 6.11 – Ответ с рейтингом

Если автор вопроса нажимает кнопку «‎+1», то ответ считается выбором автора. Такие ответы выводятся в самом верху всего списка ответов и являются показателем качества ответа.

## **6.7 Создание категории**

Категории позволяют разложить вопросы по определенным тематикам, а также позволяют структурировать веб-сервис, разбивая его по определенным разделам. Возможность создавать новые категории доступна пользователям с ролью администратор.

Для добавления новой категории нужно перейти во вкладку «‎Все категории» и воспользоваться кнопкой «‎Создать категорию» как показано на рисунке 6.12.

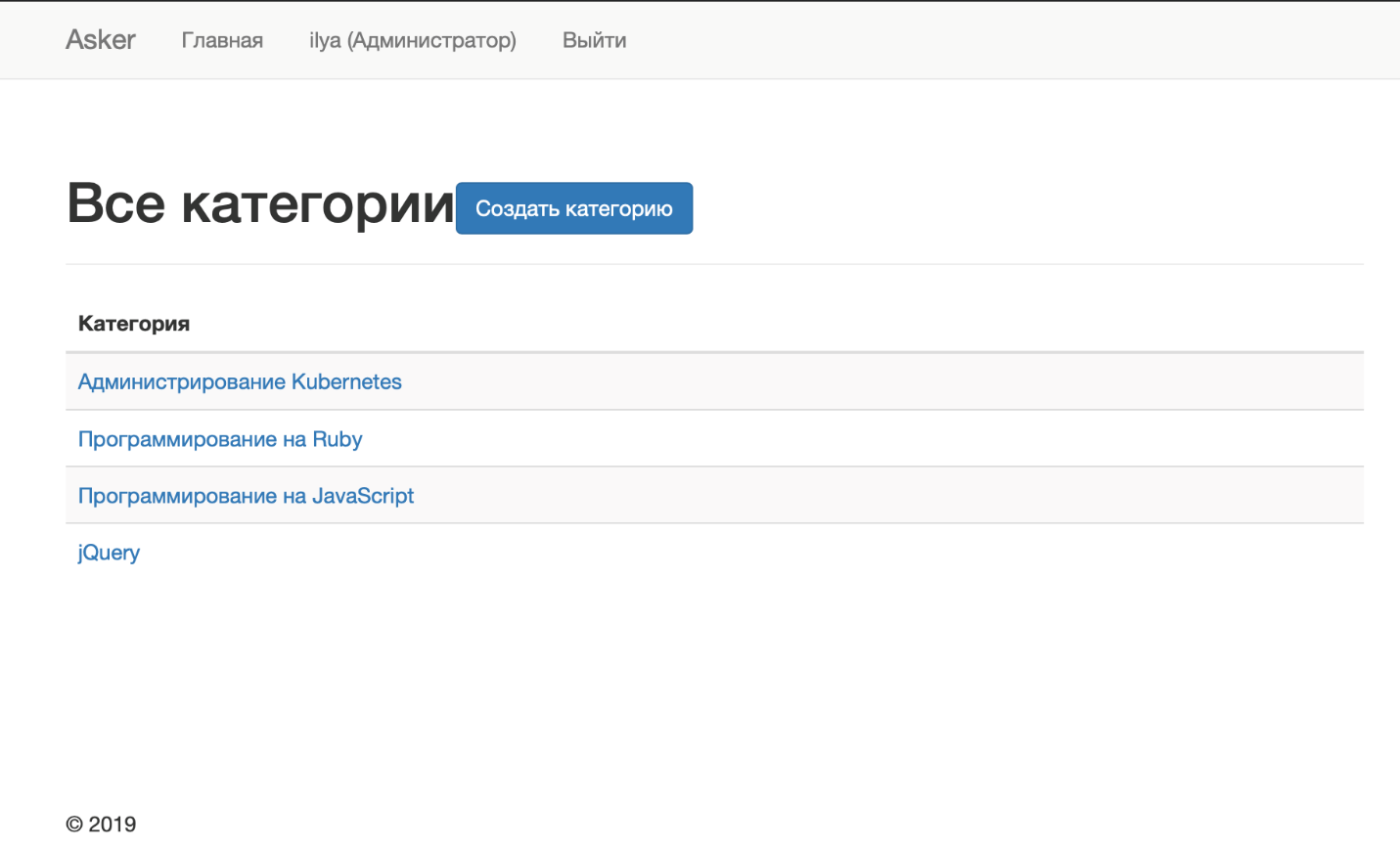


Рисунок 6.12 – Кнопка добавления новой категории

После нажатия по кнопку добавления открывается форма добавления новой категории как показано на рисунке 6.13.

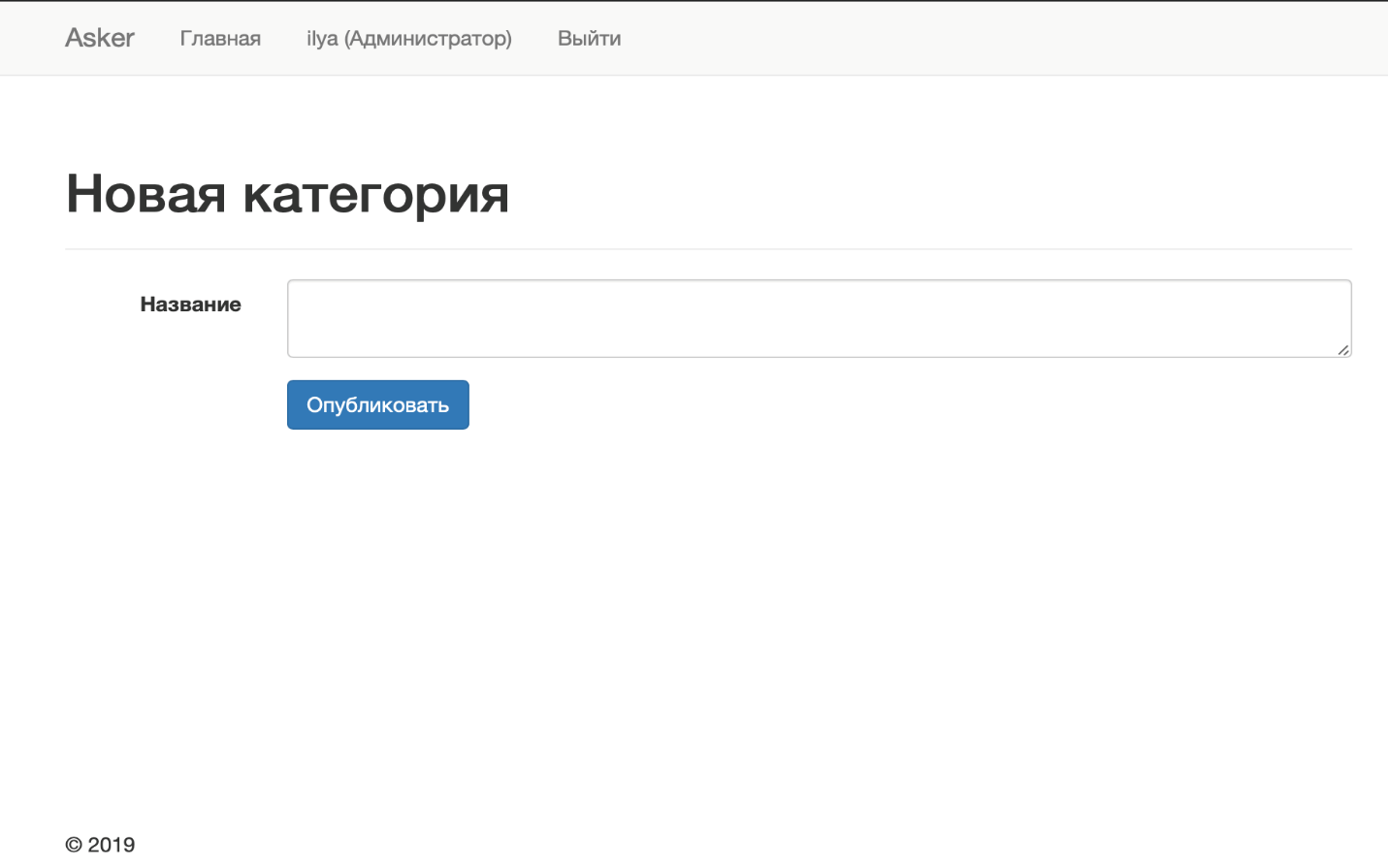


Рисунок 6.13 – Форма добавления новой категории

После заполнения формы добавления новой категории, администратор попадает на страницу с новой категорией как показано на рисунке 6.14.

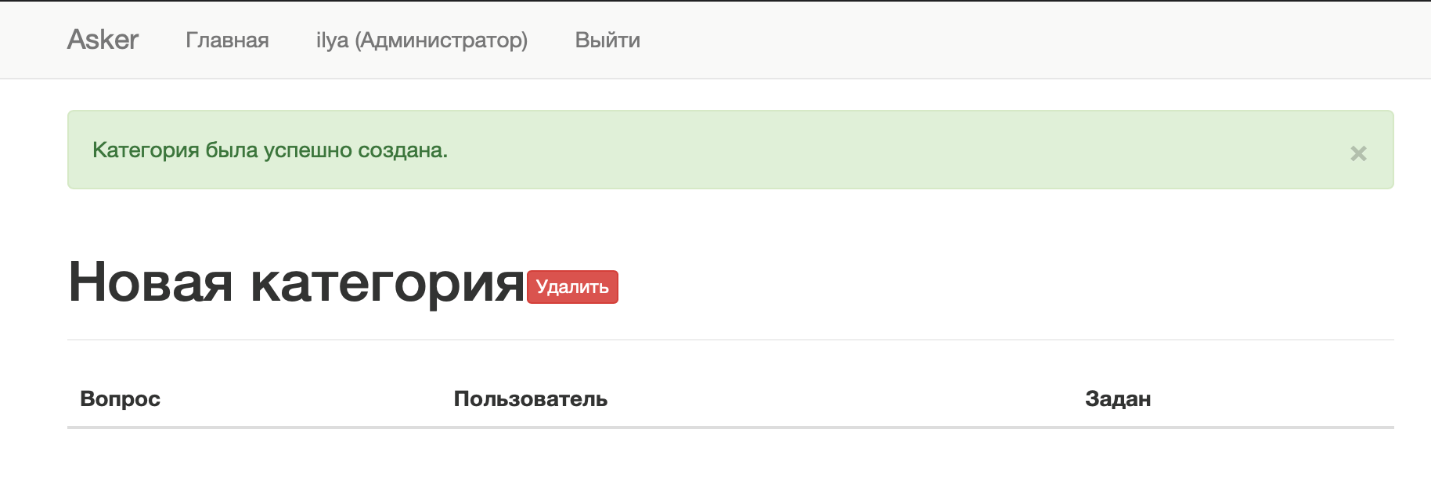


Рисунок 6.14 – Добавленная категория

## **6.8 Поиск возможного ответа**

Если на веб-сервисе уже был добавлен похожий вопрос и на него был дан ответ, то на странице с вопросом будет показан возможный ответ, как это показано на рисунке 6.15.

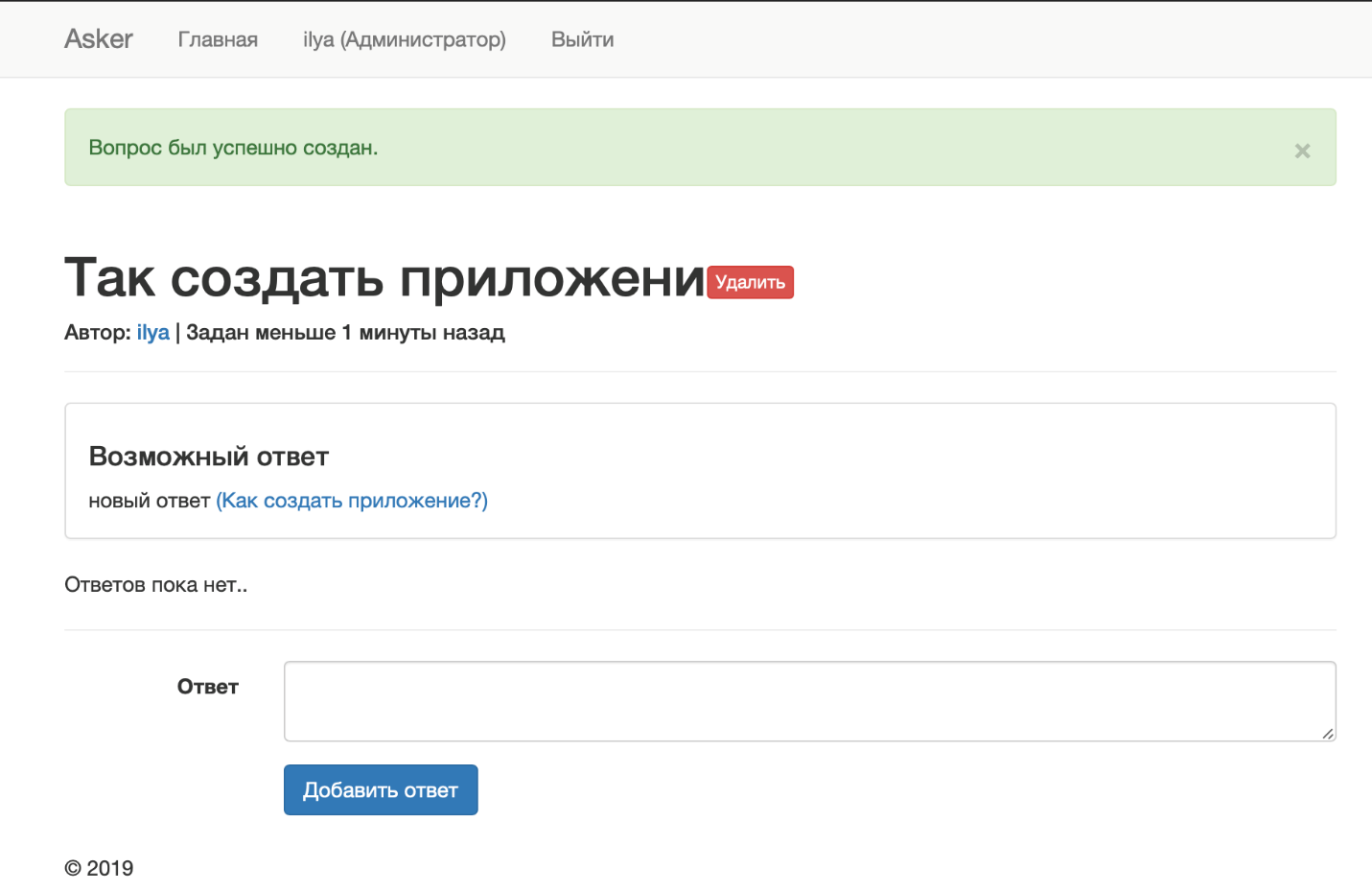


Рисунок 6.15 – Возможный ответ на вопрос

# **7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ВЕБ-СЕРВИСА ВОПРОСОВ И ОТВЕТОВ**

## **7.1 Характеристика программного продукта**

Разработанный в дипломном проекте вес-сервис позволяет пользователю быстро искать ответы на свои вопросы, если подобный вопрос уже задавался на веб-сервисе, а также дает возможность голосовать за лучший ответ. Если подобного ответа на сервисе нет, другие участники могут на него ответить.

Приложение предназначено для использования на веб-серверах с ОС семейства Linux. Пользоваться самим приложением могут пользователи любых современных ОС, при помощи любого браузера.

Основные функции разработанного приложения:

* авторизация и управление ролями пользователей;
* работа с вопросами;
* поиск подходящего ответа на запрос пользователя.
* гибкое управление процессом регистрации и авторизации, в том числе возможность создать учетные записи для модераторов и администраторов;
* система ролей, дающая пользователям определенные привилегии только если они имеют соответствующую роль.
* возможность задать категорию задаваемому вопросу;
* быстрый поиск по базе и получение ответа на запрос пользователя;
* поиск подходящего ответа на основе голосов пользователей, отданных за ответы в похожем вопросе;
* осуществление поиска по похожим вопросам после создания самого вопроса.

Веб-сервис будет размещен в сети Интернет и будет доступен всем желающим. Разработчик получит прибыль от размещения платных вопросов, от закрепления своих вопросов в самом верху популярных разделов. При размещении вопроса в самом верху страницы, его увидит большее количество посетителей сайта.

## **7.2 Расчёт сметы затрат и цены веб-сервиса**

Разрабатываемый программный продукт относится к второй категории сложности [26], поскольку относится к ПО с свойством переносимости. Программный продукт является ПО общего назначения и относится к категории новизны Б (Кн = 0,9). При расчете сметы затрат будут использоваться данные, приведенные в таблице 7.1. Они отражают текущую финансовую ситуацию.

Таблица 7.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателей | Буквенные обозначения | Единицы измерения | Количество |
| Группа сложности | - | - | 2 |
| Коэффициент новизны |  | - | 0,9 |
| Коэффициент, использования стандартных модулей |  | - | 0,7 |
| Дополнительный коэффициент сложности |  | - | 0,12 |
| Установленная плановая продолжительность разработки |  | лет | 1 |
| Количество дней в году |  | дней | 365 |
| Количество праздничных дней |  | дней | 9 |
| Количество выходных дней |  | дней | 104 |
| Количество дней отпуска |  | дней | 24 |
| Тарифная ставка 1-го разряда |  | руб. | 35,5 |
| Продолжительность рабочего дня |  | часов | 8 |
| Установленный фонд рабочего времени |  | дней | 228 |
| Норматив дополнительной заработной платы |  | % | 20 |
| Ставка отчислений в ФСЗН |  | % | 35 |
| Ставка отчислений на обязательное социальное страхование |  | % | 6 |
| Норма расхода материалов от основной заработной платы |  | % | 3 |
| Цена одного машинного часа |  | руб. | 0,45 |
| Норматив расхода машинного времени |  | ч. / 100 строк кода | 12 |
| Норматив расходов на командировки |  | % | 15 |
| Норматив прочих затрат |  | % | 20 |
| Норматив накладных расходов |  | % | 50 |
| Уровень рентабельности |  | % | 20 |
| Ставка налога на добавленную стоимость | НДС | % | 20 |
| Норматив расходов на освоение |  | % | 10 |
| Норматив расходов на сопровождение |  | % | 20 |
| Ставка налога на прибыль |  | % | 18 |

Отправной точкой для расчёта плановой сметы затрат на разработку ПО, требуется определить общий объем программного продукта (V\_о). В качестве единицы измерения примем количество строк исходного кода (Lines of Code, LOC). Прогнозируемый общий объём ПО определяется по каталогу функций. Каталог функций данного веб-сервиса представлен в таблице 7.2.

Таблица 7.2 – Каталог функций веб-сервиса

|  |  |  |
| --- | --- | --- |
| Код функции | Наименование (содержание) функции | Объем функций (LOC) |
| 101 | Организация ввода информации | 500 |
| 102 | Контроль, предварительная обработка информации | 590 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 2320 |
| 203 | Формирование баз данных | 1020 |
| 204 | Обработка наборов и записей базы данных | 7300 |
| 207 | Манипулирование данными | 9750 |
| 405 | Система настройки ПО | 250 |
| 506 | Обработка ошибочных и сбойных ситуаций | 300 |
| 507 | Обеспечение интерфейса между компонентами | 2990 |
| 703 | Расчет показателей | 2060 |
| 707 | Графический вывод результатов | 10302 |
| – | Общий объём (VО) | 37482 |

На основе общего объёма и категории сложности программного обеспечения определяется нормативная трудоёмкость, которая, в данном случае, для = 37482 и второй категории сложности, составит = 847 человеко-дней.

Наличие интерактивного интерфейса позволяет применить к объёму ПО коэффициент Кс, который определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.1) |

где – коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;

n – количество учитываемых характеристик.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Исходя из нормативной трудоёмкости можно определить общую трудоёмкость, то по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.2) |

где Кс – дополнительный коэффициент сложности;

Кт – коэффициент использования типовых программ и модулей;

Кн – коэффициент новизны.

Подставив значения в формулу (7.2), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Имея общую трудоёмкость, определяется численность исполнителей проекта, либо срок его разработки. Данный проект делался на заказ, при этом заранее было определено, что работа будет выполнена двумя людьми.

Для определения срока разработки проекта, необходимо рассчитать эффективный фонд времени одного работника ():

|  |  |  |
| --- | --- | --- |
|  |  | (7.3) |

где Дг – количество дней в году;

Дп – количество праздничных дней в году;

Дв ­– количество выходных дней в году;

До – количество дней отпуска.

Таким образом, по формуле (7.3) фонд эффективного времени составит:

Срок разработки проекта (Тр) определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.4) |

где Чр – численность исполнителей проекта;

То – общая трудоемкость разработки проекта, человеко-дней;

Фэф – эффективный фонд времени работы одного работника.

Подставив значения в формулу (7.4), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Месячная тарифная ставка исполнителя () определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.5) |

где Тм1 – месячная тарифная ставка первого разряда, руб.;

Тк – тарифный коэффициент.

В данном случае имеется два работника – инженера-программиста II-й категории (тарифный разряд – 12, тарифный коэффициент – 2,84).

Месячная тарифная ставка, определённая по формуле (7.5) составит:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Исходя из месячной тарифной ставки рассчитывается часовая тарифная ставка ():

|  |  |  |
| --- | --- | --- |
|  |  | (7.6) |

где Фр – среднемесячная норма рабочего времени, ч.

При подстановке значений в формулу (7.6), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Основная заработная плата исполнителей рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.7) |

где Тч*i* – часовая тарифная ставка *i*-го исполнителя, руб.;

Тч – количество часов работы в день, ч;

Фп – плановый фонд рабочего времени *i*-го исполнителя;

К – коэффициент премирования, принятый равным 1,4.

Учитывая число разработчиков n = 2, определим основную заработную плату по формуле (7.7):

|  |  |  |
| --- | --- | --- |
|  |  |  |

Дополнительная заработная плата () включает в себя оплаты отпусков и другие выплаты, предусмотренные законодательством, и определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.8) |

где Нд – норматив дополнительной заработной платы (15%).

Тогда получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Отчисления в фонды социальной защиты населения и социального страхования определяются по следующим формулам:

|  |  |  |
| --- | --- | --- |
|  |  | (7.9) |
|  |  | (7.10) |

где Нсз – норматив отчислений в фонд социальной защиты населения (35%);

Нсс – норматив отчислений в фонд социального страхования (6%).

По формулам (7.9) и (7.10) получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Расходы по статье «Материалы» отражают расходы на бумагу, тонер и прочие вещи, необходимые для разработки ПО, и определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.11) |

где Нмз – норма расхода материалов от основной заработной платы (3%).

Подставив значения в формулу (7.11), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Расходы по статье «Машинное время» включает оплату машинного времени для разработки и отладки ПО и определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.12) |

где Цм*i* – цена одного машино-часа, руб.;

*V*о*i* – общий объем ПО (строк исходного кода);

Нмв – норматив расхода машинного времени на отладку 100 строк исходного кода, машино-часов.

В современных условиях разработки используется понижающий коэффициент 0,4. Подставляя значения в формулу (7.12), с учётом понижающего коэффициента получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Расходы по статье «Научные командировки» включают затраты на командировочные выезды и определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.13) |

где Нрнк – норматив расходов на командировки по организации (15%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Расходы по статье «Прочие затраты» включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Расходы по данной статье определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.14) |

где Нпз – норматив прочих затрат в целом по организации (20%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Затраты по статье «Накладные расходы» связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств. Определяются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.15) |

где Нрн – норматив накладных расходов в целом по организации.

Общая сумма расходов по смете () определяется как сумма выше рассчитанных показателей:

|  |  |  |
| --- | --- | --- |
|  |  | (7.16) |

Подставив рассчитанные ранее значения в формулу (7.16), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Прогнозируемая прибыль от создаваемого ПО определяется как:

|  |  |  |
| --- | --- | --- |
|  |  | (7.17) |

где Сп*i* – себестоимость ПО, руб.;

Урп*i* – уровень рентабельности ПО (20%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

На основе прогнозируемой прибыли определяется прогнозируемая цена ПО без налогов:

|  |  |  |
| --- | --- | --- |
|  |  | (7.18) |

Подставляя значения в формулу (7.18), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

При расчёте отпускной цены дополнительно учитывается налог на добавочную стоимость (НДС):

|  |  |  |
| --- | --- | --- |
|  |  | (7.19) |

где Ндс – норматив налога на добавленную стоимость (20%).

По формуле (7.19) налог на добавочную стоимость равен:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Таким образом, с учётом НДС отпускная цена рассчитывается как:

|  |  |  |
| --- | --- | --- |
|  |  | (7.20) |

Подставив значения в формулу (7.20), рассчитаем отпускную цену программного обеспечения:

|  |  |  |
| --- | --- | --- |
|  |  |  |

В дополнение к выше рассчитанным параметрам, определяются расходы на освоение () и сопровождение () ПО:

|  |  |  |
| --- | --- | --- |
|  |  | (7.21) |
|  |  |  |
|  |  | (7.22) |

где Но – норматив расходов на освоение ПО (10%);

Нс – норматив расходов на сопровождение ПО (20%).

Используя формулы (7.21) и (7.22), определим значения расходов на освоение (), а также сопровождение () программного обеспечения:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Все выше рассчитанные показатели сведены в таблицу 7.3.

Таблица 7.3 – Расчёт себестоимости и отпускной цены

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статьи | Норматив | Формула расчёта | Значение, руб. |
| Основная ЗП | – |  |  |
| Дополнительная ЗП | Нд = 20% |  |  |

Продолжение таблицы 7.3

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статьи | Норматив | Формула расчёта | Значение, руб. |
| Отчисления в фонд соцзащиты | Нсз = 35% |  |  |
| Отчисления в фонд социального страхования | Нсс = 6% |  |  |
| Машинное время | Нмв = 12 ч |  |  |
| Научные командировки | Ннк = 15% |  |  |
| Прочие затраты | Нпз = 20% |  |  |
| Накладные расходы | Нрн = 50% |  |  |
| Общая сумма по смете | – |  |  |
| Прогнозируемая прибыль | Урп = 20% |  |  |
| Прогнозируемая цена без налогов | – |  |  |
| НДС | Ндс = 20% |  |  |
| Отпускная цена | – |  |  |
| Освоение ПО | Но = 10% |  |  |
| Сопровождение ПО | Нс = 20% |  |  |

Учитывая налог на прибыль, можно рассчитать итоговую сумму, которая останется разработчикам и будет является их экономическим эффектом:

|  |  |  |
| --- | --- | --- |
|  |  | (7.23) |

где ΔПч – чистая прибыль;

Ппс – прогнозируемая прибыль;

Нп – норматив налога на прибыль (18%).

Подставив значения в формулу (7.23), определим чистую прибыль:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Чистая прибыль от реализации ПО (ΔПч = 1364.84 рублей) остается организации-разработчику и представляет собой экономический эффект от создания новой веб-системы.

## **7.3 Расчёт экономического эффекта от применения веб-сервиса у пользователя**

Для определения экономического эффекта от использования нового программного обеспечения у потребителя необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового программного обеспечения (расходы на заработную плату с начислениями, материалы, машинное время) с расходами по соответствующим статьям при использовании прежнего варианта программного обеспечения. При сравнении базового и нового вариантов ПО в качестве экономического эффекта будет выступать общая экономия всех видов ресурсов относительно базового варианта. При этом создание нового ПО окажется экономически целесообразным лишь в том случае, если все капитальные затраты окупятся за счет получаемой экономии в ближайшие 2–3 года.

Таблица 7.4 – Исходные данные для определения экономического эффекта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование показателей | Обо­значе­ние | Единицы из­мерения | Значение показателя | |
| в базовом варианте | в новом варианте |
| Капитальные вложения, включая затраты поль­зователя на приобрете­ние ПO | Кпр | руб. | - |  |
| Затраты на освоение ПО | Кос | руб. | - |  |
| Затраты на сопровожде­ние ПО | Кс | руб. | - |  |
| Затраты на укомплекто­вание ВТ техническими средствами в связи с внедрением нового ПО | Ктс | руб. | - | 639 |
| Среднемесячная ЗП одного программиста | Зсм | руб. | 280 | 280 |

Продолжение таблицы 7.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование показателей | Обо­значе­ние | Единицы из­мерения | Значение показателя | |
| в базовом варианте | в новом варианте |
| Затраты на пополнение оборотных средств в связи с эксплуатацией нового ПО | Коб | руб. | - | 200 |
| Коэффициент начисле­ний на зарплату | Кнз |  | 1,5 | 1,5 |
| Среднемесячное коли­чество рабочих дней | Др | день | - | 20,3 |
| Количество типовых за­дач, решаемых за год | Зт1, Зт2 | задача | 1 800 | 1 800 |
| Объем выполняемых работ за год | А1, А2 | задача | 1 800 | 1 800 |
| Средняя трудоемкость работ | Тс1, Тс2 | чел.-час на задачу | 6 | 0,87 |
| Средний расход машин­ного времени | Мв1, Мв2 | маш.-час на задачу | 4 | 0,5 |
| Цена 1-го машино-часа работы ЭВМ | Цм | руб. | 0,45 | 0,45 |
| Количество часов ра­боты в день | Тч | ч | 8 | 8 |
| Ставка налога на при­быль | Нп | % | 18 | 18 |

Общие капитальные вложения заказчика (потребителя) рассчитываются по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.24) |

где Кпр – затраты пользователя на приобретение программного обеспечения по отпускной цене у разработчика с учетом стоимости услуг по эксплуатации, руб.;

Кос – затраты пользователя на освоение эксплуатирования веб-системы, руб.;

Кс – затраты пользователя на оплату услуг по сопровождению программного обеспечения, руб.;

Ктс – затраты на доукомплектование ВТ техническими средствами в связи с внедрением нового ПО, руб.;

Коб – затраты на пополнение оборотных средств в связи с использованием нового ПО, руб.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия затрат на заработную плату () при использовании нового ПО в расчете на объем выполненных работ определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.25) |

где Сзе – экономия затрат на заработную плату при решении задач c использованием нового ПО на 1 задачу, руб.;

А2 – объем выполненных работ с использованием нового ПО (задач).

Экономия затрат на заработную плату рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.26) |

где Зсм – среднемесячная заработная плата одного программиста, руб.;

Тс1, Тс2 – трудоемкости работ в расчете на 1 задачу, человеко-часов;

Тч – количество часов работы в день, ч;

Др – среднемесячное количество рабочих дней.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Тогда, подставляя значения в формулу (7.25), рассчитаем экономию затрат на заработную плату:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия с учетом начисления на зарплату вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.27) |

где Кнз – коэффициент начислений на зарплату (1,5).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Экономия затрат на оплату машинного времени () в расчете на выполненный объем работ в результате применения нового ПО:

|  |  |  |
| --- | --- | --- |
|  |  | (7.28) |

где Сме – экономия затрат на оплату машинного времени в расчёте на 1 задачу с использованием нового ПО.

Экономия затрат на оплату машинного времени в расчете на 1 задачу определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.29) |
|  |  |  |

где Цм – цена одного машино-часа работы ЭВМ;

Мв1, Мв2 – средний расход машинного времени при применении соответственно базового и нового ПО.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Таким образом, по формуле (7.28) определим экономию затрат на оплату машинного времени в расчете на выполненный объем работ:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Общая годовая экономия текущих затрат, связанных с использованием нового ПО является важным фактором и вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.30) |

Подставляя ранее полученные значения в формулу (7.30), получим:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Для заказчика в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении, которая определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.31) |

где Нп – ставка налога на прибыль (18%).

|  |  |  |
| --- | --- | --- |
|  |  |  |

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты.

Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2019 год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования αt, который рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.32) |
|  |  |  |

где *E* – норматив приведения разновременных затрат и результатов (c учётом безрисковой ставки процента по валютным депозитам, уровня инфляции, роста спроса и стабильности дохода примем *E* = 24%);

*t* – номер года, результаты и затраты которого приводятся к расчётному (2019 год – 1, 2020 год – 2, 2021 год – 3, 2022 год – 4);

*tp* – номер расчётного года (2019).

Таким образом, получим следующие значения коэффициентов дисконтирования:

2019 год: 

2020 год: 

2021 год: 

2022 год: 

Сведем данные расчета экономического эффекта в таблицу 7.5.

Таблица 7.5 – Расчет экономического эффекта от использования нового ПС

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | Годы | | | |
| 2019 | 2020 | 2021 | 2022 |
| *Результаты:* | | | | | |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 21903,84 | 21903,84 | 21903,84 |
| То же с учетом фактора времени | руб. |  | 17665,45 | 14246.26 | 11488.56 |
| *Затраты:* | | | | | |

Продолжение таблицы 7.5

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | | | Годы | | | | | | |
| 2019 | | 2020 | 2021 | | 2022 | |
| Приобретение ПО (Кпр) | руб. | | | 11981,06 | |  |  | |  | |
| Освоение ПО (Кос) | руб. | | | 832,02 | |  |  | |  | |
| Сопровождение (Кс) | руб. | | | 1664,04 | |  |  | |  | |
| Доукомплектование ВТ техническими средствами (Ктс) | руб. | | | 639 | |  |  | |  | |
| Пополнение оборотных средств (Коб) | руб. | | | 200 | |  |  | |  | |
| Всего затрат | руб. | | | 15316,12 | |  |  | |  | |
| То же с учетом фактора времени | руб. | | | 15316,12 | |  |  | |  | |
| *Затраты:* | | | | | | | | | | |
| Превышение результатов над затратами | | руб. | -15316,12 | | 17665,45 | | | 14246,26 | | 11488,56 |
| То же нарастающим итогом | | руб. | -15316,12 | | 2349,33 | | | 16595,59 | | 28084,15 |
| Коэффициент приведения | | ед. | 1 | | 0,8065 | | | 0,6504 | | 0,5245 |

## **7.4 Вывод по технико-экономическому обоснованию**

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

Среднегодовая величина чистой прибыли:

Чистый дисконтированный доход за четыре года производства продукции составит 23297,16 тыс. руб.

Все инвестиции окупаются на второй год использования программного продукта.

Рентабельность инвестиций в приобретение программного продукта:

Таким образом было произведено технико-экономическое обоснование разрабатываемого проекта, составлена смета затрат и рассчитана прогнозируемая прибыль, и показана экономическая целесообразность разработки.

Полученные результаты свидетельствуют об эффективности разработки и внедрения в эксплуатацию веб-системы контроля над документацией пользователя.

Таким образом разработка и реализация приложения является экономически целесообразной.

# **ЗАКЛЮЧЕНИЕ**

В данном дипломном проекте был спроектирован и реализован веб-сервис для работы с вопросами и ответами. Приложение предназначено для быстрого поиска ответа на заданный вопрос, если похожая пара вопрос-ответ уже существует в системе и ответ на нее был дан. В этом случае сразу после создания такого вопроса, пользователю будет предложен ответ на вопрос. Поиск похожих вопросов осуществляется быстро при помощи масштабируемого хранилища ключ-значение.

Приложения позволяет зарегистрироваться и аутентифицироваться посетителям веб-сервиса. Администраторы приложения имеют возможность назначать роли модератора и администратора другим участникам сообщества.

Пользователи могут удалить свой вопрос или ответ, если посчитают его не слишком полезным для веб-сервиса. Модераторы могут управлять всеми вопросами и ответами, расположенными на веб-сервисе. Это позволяет удалять нежелательный контент из приложения.

Приложение поддерживает многопользовательский режим доступа, поэтому в приложении одновременно могут работать большое количество людей. Поддерживается русский язык интерфейса.

Пользователи могут определять полезность ответов при помощи оценок других пользователь, при помощи значков выбора автора.

Кроме того, приложение является и удобным средством для поиска информации по интересующей пользователя теме, так как все вопросы структурированы по категориям. Категории позволяют группировать темы с похожей близкой тематикой.

Регистрация в приложении защищена при помощи анти-спам системы, поэтому появление большого количества нежелательной информации маловероятно. Для восстановления доступа к аккаунту предусмотрен сброс пароля при помощи адреса электронной почты.

Веб-сервис доступен для использования во всех современных браузеров. Веб-приложение предназначено для установки на всех популярных дистрибутивах Linux, таких как Debian, Ubuntu.

Приложение вместе со всей вспомогательной инфраструктурой было размещено на платформе heroku и доступно всем пользователям Интернета. Веб-сервис готов к горизонтальному масштабированию. Возможна реализация механизма автоматического разворачивания новых серверов с приложением при определенных условиях.

Все цели дипломного проектирования достигнуты в полном объёме. Приложение имеет направление для дальнейшего развития и расширения функциональных возможностей. В частности, в ближайшие планы входит интеграция Kubernetes для добавления возможности автоматически масштабироваться в случае возникновения большого количества посетителей веб-сервиса.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Дергачев А.М. Проблемы эффективного использования сетевых сервисов / Научно-технический вестник – СПбГУ, ИТМО, 2011, с. 83-87
2. Веб-сервисы в теории и на практике. [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/46374
3. Онлайн сервисы и их виды. [Электронный ресурс]. – Режим доступа: http://book-science.ru/applied/onlajn-servisy-i-ih-vidy.html
4. [Ответы@Mail.Ru](mailto:Ответы@Mail.Ru). [Электронный ресурс]. – Режим доступа: <https://help.mail.ru>
5. Quora. [Электронный ресурс]. – Режим доступа: https://www.quora.com/about
6. Toster. [Электронный ресурс]. – Режим доступа: https://toster.ru/help/about
7. ASKfm. [Электронный ресурс]. – Режим доступа: https://about.ask.fm/about
8. Stack Overflow. [Электронный ресурс]. – Режим доступа: http://highscalability.com/stack-overflow-architecture
9. О сервисе Askee. [Электронный ресурс]. – Режим доступа: https://mirfactov.com/askee-otlichnyiy-servis-voprosov-i-otvetov
10. О сервисе TheQuestion. [Электронный ресурс]. – Режим доступа: https://thequestion.ru/rules
11. Распределенное хранение данных. [Электронный ресурс]. – Режим доступа: https://thedbit.com/wiki/2023.html
12. Memcached. [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/memcached
13. Redis. [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/redis
14. Архитектуры облачных систем обработки. [Электронный ресурс]. – Режим доступа: http://www.seagate.com/ru/ru/tech-insights/cloud-compute-and-cloud-storage-architecture-master-ti
15. Д. Флэнаган, Ю. Мацумото. Язык программирования Ruby – СПбГУ, 2011, 496 с.
16. Хэнссон Д. Х., Томас Д. Гибкая разработка веб-приложений в среде Rails. – СПб., 2008 – 720 с.
17. Преимущества Ruby on Rails. [Электронный ресурс]. – Режим доступа: <https://makeagency.ru/blog/item/pochemu-ruby-on-rails>
18. Почему стоит выбрать Ruby on Rails. [Электронный ресурс]. – Режим доступа: <https://web-creator.ru/articles/why_ruby_on_rails>
19. SQLite vs MySQL vs PostgreSQL: сравнение систем управления базами данных. [Электронный ресурс]. – Режим доступа: http://devacademy.ru/posts/sqlite-vs-mysql-vs-postgresql
20. MySQL: Особенности и сферы применения. [Электронный ресурс]. – Режим доступа: <https://www.bytemag.ru/articles/detail.php?ID=6547>
21. Документация MySQL. [Электронный ресурс]. – Режим доступа: https://dev.mysql.com/doc
22. Протокол передачи гипертекста HTTP. [Электронный ресурс]. – Режим доступа: <https://www.lib.ru/WEBMASTER/rfc2068>
23. Sinatra. [Электронный ресурс]. – Режим доступа: <https://leavinsprogramming.blogspot.com/2012/05/pros-and-cons-of-using-sinatra.html>
24. Документация библиотек Ruby. [Электронный ресурс]. – Режим доступа: <https://www.rubydoc.info>
25. Rails Asset Pipeline. [Электронный ресурс]. – Режим доступа: https://rusrails.ru/asset-pipeline
26. Палицын В.А. Экономическое обоснование проектов программного обеспечения — Минск, БГУИР, 2006. — 76 с.

# **ПРИЛОЖЕНИЕ А**

(обязательное)

Исходный текст класса QuestionSearch

class QuestionSearch

attr\_reader :exclude\_question\_id, :description

def initialize(exclude\_question\_id, description)

@exclude\_question\_id = exclude\_question\_id

@description = description

end

def similar\_answer

return if (question = search\_question).blank?

answer = question.answers.order("votes\_count DESC").where(is\_author: true).first

return answer if answer.present?

question.answers.order("votes\_count DESC").first

end

def search\_question

return if similar\_question\_id.blank?

Question.where(id: similar\_question\_id).first

end

def similar\_question\_id

return if ordered\_marlin\_questions.blank?

@similar\_question\_id ||= ordered\_marlin\_questions[0]

end

def ordered\_marlin\_questions

@ordered\_marlin\_questions ||= marlin\_question\_ids.max\_by { |\_key, value| value }

end

def marlin\_question\_ids

words.map { |word| get\_ids(word) }.flatten.each\_with\_object(Hash.new(0)) do |element, hash|

next if exclude\_question\_id.to\_i == element.to\_i

hash[element] += 1

end

end

private

def get\_ids(word)

WordProcessor.new(word).get\_ids

end

def words

QuestionDescriptionCrop.call(description)

end

end

Исходный текст класса MarlinManager

class MarlinManager

attr\_reader :key

def initialize(key)

@key = key

End

class << self

def ping

new("check").ping

end

end

def ping

MarlinActions::Ping.new(key).perform

end

def read\_key

MarlinActions::Get.new(key).perform

end

def put\_key(value)

MarlinActions::Put.new(key).perform(value)

end

def delete\_key

MarlinActions::Delete.new(key).perform

end

end

# **ПРИЛОЖЕНИЕ Б**

(обязательное)

Исходный текст классов Marlin

require "redis"

require "connection\_pool"

REDIS ||= ConnectionPool.new(size: 15) { Redis.new }

module Marlin

module Persisters

class RedisStorage < Marlin::Persister

def read(key)

REDIS.with do |redis|

redis.get(key.to\_s)

end

end

def write(key, value)

REDIS.with do |redis|

redis.set(key.to\_s, value)

end

end

def delete(key)

REDIS.with do |redis|

redis.del(key.to\_s)

end

end

def flushall

REDIS.with do |redis|

redis.flushall

end

end

end

end

end

module Marlin

module Persisters

class Memory < Marlin::Persister

@@storage = {}

def read(key)

@@storage[key]

end

def write(key, value)

@@storage[key] = value

end

def delete(key)

@@storage.delete(key)

end

end

end

end

require "sinatra/base"

module Marlin

class App < Sinatra::Base

set :root, File.dirname(\_\_FILE\_\_)

KEYS\_ROUTE = "/keys/:key"

get "/" do

"Marlin Key-Value Storage (#{ENV['SLAVE\_URL'] ? 'Master' : 'Slave' })"

end

get "/ping" do

"OK"

end

put KEYS\_ROUTE do

Marlin::Actions::SaveKey.new(params[:key]).call(request.body.read)

end

delete KEYS\_ROUTE do

Marlin::Actions::DeleteKey.new(params[:key]).call

end

get KEYS\_ROUTE do

Marlin::Actions::ReadKey.new(params[:key]).call

end

get "/flushall" do

Marlin::Actions::FlushAll.new(nil).call

end

require File.join(root, "/config/initializers/autoloader.rb")

end

end

require 'rest-client'

module Marlin

module Actions

class ReplicateKey < Marlin::Action

def call(action)

value = @persister.read(@key)

return unless ENV['SLAVE\_URL']

RestClient.put(key\_route, value) if action == :save

RestClient.delete(key\_route) if action == :delete

RestClient.get(flush\_route) if action == :flushall

end

private

def key\_route

[ENV['SLAVE\_URL'], "/keys/", @key].join

end

def flush\_route

[ENV['SLAVE\_URL'], "/flushall"].join

end

end

end

end

AUTOLOAD\_DIRS = %w(

action.rb

persister.rb

actions/\*.rb

persisters/\*.rb

)

AUTOLOAD\_DIRS.each do |autoload\_dir|

Dir[File.join(Marlin::App.root, autoload\_dir)].each do |file|

next if file.include?("initializers/autoloader")

require file

end

end

module Marlin

class Persister

protected

def read(key)

raise NotImplementedError

end

def write(key, value)

raise NotImplementedError

end

def flushall

raise NotImplementedError

end

end

end

module Marlin

class Action

def initialize(key, persister = Marlin::Persisters::RedisStorage.new)

@key = key

@persister = persister

end

protected

def call

raise NotImplementedError

end

end

end

Исходный текст модуля управления аутентификацией

# frozen\_string\_literal: true

class Devise::RegistrationsController < DeviseController

prepend\_before\_action :require\_no\_authentication, only: [:new, :create, :cancel]

prepend\_before\_action :authenticate\_scope!, only: [:edit, :update, :destroy]

prepend\_before\_action :set\_minimum\_password\_length, only: [:new, :edit]

# GET /resource/sign\_up

def new

build\_resource

yield resource if block\_given?

respond\_with resource

end

# POST /resource

def create

build\_resource(sign\_up\_params)

resource.save

yield resource if block\_given?

if resource.persisted?

if resource.active\_for\_authentication?

set\_flash\_message! :notice, :signed\_up

sign\_up(resource\_name, resource)

respond\_with resource, location: after\_sign\_up\_path\_for(resource)

else

set\_flash\_message! :notice, :"signed\_up\_but\_#{resource.inactive\_message}"

expire\_data\_after\_sign\_in!

respond\_with resource, location: after\_inactive\_sign\_up\_path\_for(resource)

end

else

clean\_up\_passwords resource

set\_minimum\_password\_length

respond\_with resource

end

end

# GET /resource/edit

def edit

render :edit

end

# PUT /resource

# We need to use a copy of the resource because we don't want to change

# the current user in place.

def update

self.resource = resource\_class.to\_adapter.get!(send(:"current\_#{resource\_name}").to\_key)

prev\_unconfirmed\_email = resource.unconfirmed\_email if resource.respond\_to?(:unconfirmed\_email)

resource\_updated = update\_resource(resource, account\_update\_params)

yield resource if block\_given?

if resource\_updated

set\_flash\_message\_for\_update(resource, prev\_unconfirmed\_email)

bypass\_sign\_in resource, scope: resource\_name if sign\_in\_after\_change\_password?

respond\_with resource, location: after\_update\_path\_for(resource)

else

clean\_up\_passwords resource

set\_minimum\_password\_length

respond\_with resource

end

end

# DELETE /resource

def destroy

resource.destroy

Devise.sign\_out\_all\_scopes ? sign\_out : sign\_out(resource\_name)

set\_flash\_message! :notice, :destroyed

yield resource if block\_given?

respond\_with\_navigational(resource){ redirect\_to after\_sign\_out\_path\_for(resource\_name) }

end

# GET /resource/cancel

# Forces the session data which is usually expired after sign

# in to be expired now. This is useful if the user wants to

# cancel oauth signing in/up in the middle of the process,

# removing all OAuth session data.

def cancel

expire\_data\_after\_sign\_in!

redirect\_to new\_registration\_path(resource\_name)

end

protected

def update\_needs\_confirmation?(resource, previous)

resource.respond\_to?(:pending\_reconfirmation?) &&

resource.pending\_reconfirmation? &&

previous != resource.unconfirmed\_email

end

# By default we want to require a password checks on update.

# You can overwrite this method in your own RegistrationsController.

def update\_resource(resource, params)

resource.update\_with\_password(params)

end

# Build a devise resource passing in the session. Useful to move

# temporary session data to the newly created user.

def build\_resource(hash = {})

self.resource = resource\_class.new\_with\_session(hash, session)

end

# Signs in a user on sign up. You can overwrite this method in your own

# RegistrationsController.

def sign\_up(resource\_name, resource)

sign\_in(resource\_name, resource)

end

# The path used after sign up. You need to overwrite this method

# in your own RegistrationsController.

def after\_sign\_up\_path\_for(resource)

after\_sign\_in\_path\_for(resource) if is\_navigational\_format?

end

# The path used after sign up for inactive accounts. You need to overwrite

# this method in your own RegistrationsController.

def after\_inactive\_sign\_up\_path\_for(resource)

scope = Devise::Mapping.find\_scope!(resource)

router\_name = Devise.mappings[scope].router\_name

context = router\_name ? send(router\_name) : self

context.respond\_to?(:root\_path) ? context.root\_path : "/"

end

# The default url to be used after updating a resource. You need to overwrite

# this method in your own RegistrationsController.

def after\_update\_path\_for(resource)

sign\_in\_after\_change\_password? ? signed\_in\_root\_path(resource) : new\_session\_path(resource\_name)

end

# Authenticates the current scope and gets the current resource from the session.

def authenticate\_scope!

send(:"authenticate\_#{resource\_name}!", force: true)

self.resource = send(:"current\_#{resource\_name}")

end

def sign\_up\_params

devise\_parameter\_sanitizer.sanitize(:sign\_up)

end

def account\_update\_params

devise\_parameter\_sanitizer.sanitize(:account\_update)

end

def translation\_scope

'devise.registrations'

end

private

def set\_flash\_message\_for\_update(resource, prev\_unconfirmed\_email)

return unless is\_flashing\_format?

flash\_key = if update\_needs\_confirmation?(resource, prev\_unconfirmed\_email)

:update\_needs\_confirmation

elsif sign\_in\_after\_change\_password?

:updated

else

:updated\_but\_not\_signed\_in

end

set\_flash\_message :notice, flash\_key

end

def sign\_in\_after\_change\_password?

return true if account\_update\_params[:password].blank?

Devise.sign\_in\_after\_change\_password

end

end

# frozen\_string\_literal: true

class Devise::SessionsController < DeviseController

prepend\_before\_action :require\_no\_authentication, only: [:new, :create]

prepend\_before\_action :allow\_params\_authentication!, only: :create

prepend\_before\_action :verify\_signed\_out\_user, only: :destroy

prepend\_before\_action(only: [:create, :destroy]) { request.env["devise.skip\_timeout"] = true }

# GET /resource/sign\_in

def new

self.resource = resource\_class.new(sign\_in\_params)

clean\_up\_passwords(resource)

yield resource if block\_given?

respond\_with(resource, serialize\_options(resource))

end

# POST /resource/sign\_in

def create

self.resource = warden.authenticate!(auth\_options)

set\_flash\_message!(:notice, :signed\_in)

sign\_in(resource\_name, resource)

yield resource if block\_given?

respond\_with resource, location: after\_sign\_in\_path\_for(resource)

end

# DELETE /resource/sign\_out

def destroy

signed\_out = (Devise.sign\_out\_all\_scopes ? sign\_out : sign\_out(resource\_name))

set\_flash\_message! :notice, :signed\_out if signed\_out

yield if block\_given?

respond\_to\_on\_destroy

end

protected

def sign\_in\_params

devise\_parameter\_sanitizer.sanitize(:sign\_in)

end

def serialize\_options(resource)

methods = resource\_class.authentication\_keys.dup

methods = methods.keys if methods.is\_a?(Hash)

methods << :password if resource.respond\_to?(:password)

{ methods: methods, only: [:password] }

end

def auth\_options

{ scope: resource\_name, recall: "#{controller\_path}#new" }

end

def translation\_scope

'devise.sessions'

end

private

# Check if there is no signed in user before doing the sign out.

#

# If there is no signed in user, it will set the flash message and redirect

# to the after\_sign\_out path.

def verify\_signed\_out\_user

if all\_signed\_out?

set\_flash\_message! :notice, :already\_signed\_out

respond\_to\_on\_destroy

end

end

def all\_signed\_out?

users = Devise.mappings.keys.map { |s| warden.user(scope: s, run\_callbacks: false) }

users.all?(&:blank?)

end

def respond\_to\_on\_destroy

# We actually need to hardcode this as Rails default responder doesn't

# support returning empty response on GET request

respond\_to do |format|

format.all { head :no\_content }

format.any(\*navigational\_formats) { redirect\_to after\_sign\_out\_path\_for(resource\_name) }

end

end

end

# frozen\_string\_literal: true

class Devise::PasswordsController < DeviseController

prepend\_before\_action :require\_no\_authentication

# Render the #edit only if coming from a reset password email link

append\_before\_action :assert\_reset\_token\_passed, only: :edit

# GET /resource/password/new

def new

self.resource = resource\_class.new

end

# POST /resource/password

def create

self.resource = resource\_class.send\_reset\_password\_instructions(resource\_params)

yield resource if block\_given?

if successfully\_sent?(resource)

respond\_with({}, location: after\_sending\_reset\_password\_instructions\_path\_for(resource\_name))

else

respond\_with(resource)

end

end

# GET /resource/password/edit?reset\_password\_token=abcdef

def edit

self.resource = resource\_class.new

set\_minimum\_password\_length

resource.reset\_password\_token = params[:reset\_password\_token]

end

# PUT /resource/password

def update

self.resource = resource\_class.reset\_password\_by\_token(resource\_params)

yield resource if block\_given?

if resource.errors.empty?

resource.unlock\_access! if unlockable?(resource)

if Devise.sign\_in\_after\_reset\_password

flash\_message = resource.active\_for\_authentication? ? :updated : :updated\_not\_active

set\_flash\_message!(:notice, flash\_message)

resource.after\_database\_authentication

sign\_in(resource\_name, resource)

else

set\_flash\_message!(:notice, :updated\_not\_active)

end

respond\_with resource, location: after\_resetting\_password\_path\_for(resource)

else

set\_minimum\_password\_length

respond\_with resource

end

end

protected

def after\_resetting\_password\_path\_for(resource)

Devise.sign\_in\_after\_reset\_password ? after\_sign\_in\_path\_for(resource) : new\_session\_path(resource\_name)

end

# The path used after sending reset password instructions

def after\_sending\_reset\_password\_instructions\_path\_for(resource\_name)

new\_session\_path(resource\_name) if is\_navigational\_format?

end

# Check if a reset\_password\_token is provided in the request

def assert\_reset\_token\_passed

if params[:reset\_password\_token].blank?

set\_flash\_message(:alert, :no\_token)

redirect\_to new\_session\_path(resource\_name)

end

end

# Check if proper Lockable module methods are present & unlock strategy

# allows to unlock resource on password reset

def unlockable?(resource)

resource.respond\_to?(:unlock\_access!) &&

resource.respond\_to?(:unlock\_strategy\_enabled?) &&

resource.unlock\_strategy\_enabled?(:email)

end

def translation\_scope

'devise.passwords'

end

end

# frozen\_string\_literal: true

class Devise::ConfirmationsController < DeviseController

# GET /resource/confirmation/new

def new

self.resource = resource\_class.new

end

# POST /resource/confirmation

def create

self.resource = resource\_class.send\_confirmation\_instructions(resource\_params)

yield resource if block\_given?

if successfully\_sent?(resource)

respond\_with({}, location: after\_resending\_confirmation\_instructions\_path\_for(resource\_name))

else

respond\_with(resource)

end

end

# GET /resource/confirmation?confirmation\_token=abcdef

def show

self.resource = resource\_class.confirm\_by\_token(params[:confirmation\_token])

yield resource if block\_given?

if resource.errors.empty?

set\_flash\_message!(:notice, :confirmed)

respond\_with\_navigational(resource){ redirect\_to after\_confirmation\_path\_for(resource\_name, resource) }

else

respond\_with\_navigational(resource.errors, status: :unprocessable\_entity){ render :new }

end

end

protected

# The path used after resending confirmation instructions.

def after\_resending\_confirmation\_instructions\_path\_for(resource\_name)

is\_navigational\_format? ? new\_session\_path(resource\_name) : '/'

end

# The path used after confirmation.

def after\_confirmation\_path\_for(resource\_name, resource)

if signed\_in?(resource\_name)

signed\_in\_root\_path(resource)

else

new\_session\_path(resource\_name)

end

end

def translation\_scope

'devise.confirmations'

end

end

# frozen\_string\_literal: true

if defined?(ActionMailer)

class Devise::Mailer < Devise.parent\_mailer.constantize

include Devise::Mailers::Helpers

def confirmation\_instructions(record, token, opts={})

@token = token

devise\_mail(record, :confirmation\_instructions, opts)

end

def reset\_password\_instructions(record, token, opts={})

@token = token

devise\_mail(record, :reset\_password\_instructions, opts)

end

def unlock\_instructions(record, token, opts={})

@token = token

devise\_mail(record, :unlock\_instructions, opts)

end

def email\_changed(record, opts={})

devise\_mail(record, :email\_changed, opts)

end

def password\_change(record, opts={})

devise\_mail(record, :password\_change, opts)

end

end

end

# frozen\_string\_literal: true

require 'devise/strategies/base'

module Devise

module Strategies

# This strategy should be used as basis for authentication strategies. It retrieves

# parameters both from params or from http authorization headers. See database\_authenticatable

# for an example.

class Authenticatable < Base

attr\_accessor :authentication\_hash, :authentication\_type, :password

def store?

super && !mapping.to.skip\_session\_storage.include?(authentication\_type)

end

def valid?

valid\_for\_params\_auth? || valid\_for\_http\_auth?

end

# Override and set to false for things like OmniAuth that technically

# run through Authentication (user\_set) very often, which would normally

# reset CSRF data in the session

def clean\_up\_csrf?

true

end

private

# Receives a resource and check if it is valid by calling valid\_for\_authentication?

# A block that will be triggered while validating can be optionally

# given as parameter. Check Devise::Models::Authenticatable.valid\_for\_authentication?

# for more information.

#

# In case the resource can't be validated, it will fail with the given

# unauthenticated\_message.

def validate(resource, &block)

result = resource && resource.valid\_for\_authentication?(&block)

if result

true

else

if resource

fail!(resource.unauthenticated\_message)

end

false

end

end

# Get values from params and set in the resource.

def remember\_me(resource)

resource.remember\_me = remember\_me? if resource.respond\_to?(:remember\_me=)

end

# Should this resource be marked to be remembered?

def remember\_me?

valid\_params? && Devise::TRUE\_VALUES.include?(params\_auth\_hash[:remember\_me])

end

# Check if this is a valid strategy for http authentication by:

#

# \* Validating if the model allows http authentication;

# \* If any of the authorization headers were sent;

# \* If all authentication keys are present;

#

def valid\_for\_http\_auth?

http\_authenticatable? && request.authorization && with\_authentication\_hash(:http\_auth, http\_auth\_hash)

end

# Check if this is a valid strategy for params authentication by:

#

# \* Validating if the model allows params authentication;

# \* If the request hits the sessions controller through POST;

# \* If the params[scope] returns a hash with credentials;

# \* If all authentication keys are present;

#

def valid\_for\_params\_auth?

params\_authenticatable? && valid\_params\_request? &&

valid\_params? && with\_authentication\_hash(:params\_auth, params\_auth\_hash)

end

# Check if the model accepts this strategy as http authenticatable.

def http\_authenticatable?

mapping.to.http\_authenticatable?(authenticatable\_name)

end

# Check if the model accepts this strategy as params authenticatable.

def params\_authenticatable?

mapping.to.params\_authenticatable?(authenticatable\_name)

end

# Extract the appropriate subhash for authentication from params.

def params\_auth\_hash

params[scope]

end

# Extract a hash with attributes:values from the http params.

def http\_auth\_hash

keys = [http\_authentication\_key, :password]

Hash[\*keys.zip(decode\_credentials).flatten]

end

# By default, a request is valid if the controller set the proper env variable.

def valid\_params\_request?

!!env["devise.allow\_params\_authentication"]

end

# If the request is valid, finally check if params\_auth\_hash returns a hash.

def valid\_params?

params\_auth\_hash.is\_a?(Hash)

end

# Note: unlike `Model.valid\_password?`, this method does not actually

# ensure that the password in the params matches the password stored in

# the database. It only checks if the password is \*present\*. Do not rely

# on this method for validating that a given password is correct.

def valid\_password?

password.present?

end

# Helper to decode credentials from HTTP.

def decode\_credentials

return [] unless request.authorization && request.authorization =~ /^Basic (.\*)/mi

Base64.decode64($1).split(/:/, 2)

end

# Sets the authentication hash and the password from params\_auth\_hash or http\_auth\_hash.

def with\_authentication\_hash(auth\_type, auth\_values)

self.authentication\_hash, self.authentication\_type = {}, auth\_type

self.password = auth\_values[:password]

parse\_authentication\_key\_values(auth\_values, authentication\_keys) &&

parse\_authentication\_key\_values(request\_values, request\_keys)

end

def authentication\_keys

@authentication\_keys ||= mapping.to.authentication\_keys

end

def http\_authentication\_key

@http\_authentication\_key ||= mapping.to.http\_authentication\_key || case authentication\_keys

when Array then authentication\_keys.first

when Hash then authentication\_keys.keys.first

end

end

def request\_keys

@request\_keys ||= mapping.to.request\_keys

end

def request\_values

keys = request\_keys.respond\_to?(:keys) ? request\_keys.keys : request\_keys

values = keys.map { |k| self.request.send(k) }

Hash[keys.zip(values)]

end

def parse\_authentication\_key\_values(hash, keys)

keys.each do |key, enforce|

value = hash[key].presence

if value

self.authentication\_hash[key] = value

else

return false unless enforce == false

end

end

true

end

# Holds the authenticatable name for this class. Devise::Strategies::DatabaseAuthenticatable

# becomes simply :database.

def authenticatable\_name

@authenticatable\_name ||=

ActiveSupport::Inflector.underscore(self.class.name.split("::").last).

sub("\_authenticatable", "").to\_sym

end

end

end

end

# frozen\_string\_literal: true

require 'devise/strategies/authenticatable'

module Devise

module Strategies

# Remember the user through the remember token. This strategy is responsible

# to verify whether there is a cookie with the remember token, and to

# recreate the user from this cookie if it exists. Must be called \*before\*

# authenticatable.

class Rememberable < Authenticatable

# A valid strategy for rememberable needs a remember token in the cookies.

def valid?

@remember\_cookie = nil

remember\_cookie.present?

end

# To authenticate a user we deserialize the cookie and attempt finding

# the record in the database. If the attempt fails, we pass to another

# strategy handle the authentication.

def authenticate!

resource = mapping.to.serialize\_from\_cookie(\*remember\_cookie)

unless resource

cookies.delete(remember\_key)

return pass

end

if validate(resource)

remember\_me(resource) if extend\_remember\_me?(resource)

resource.after\_remembered

success!(resource)

end

end

# No need to clean up the CSRF when using rememberable.

# In fact, cleaning it up here would be a bug because

# rememberable is triggered on GET requests which means

# we would render a page on first access with all csrf

# tokens expired.

def clean\_up\_csrf?

false

end

private

def extend\_remember\_me?(resource)

resource.respond\_to?(:extend\_remember\_period) && resource.extend\_remember\_period

end

def remember\_me?

true

end

def remember\_key

mapping.to.rememberable\_options.fetch(:key, "remember\_#{scope}\_token")

end

def remember\_cookie

@remember\_cookie ||= cookies.signed[remember\_key]

end

end

end

end

Warden::Strategies.add(:rememberable, Devise::Strategies::Rememberable)

# frozen\_string\_literal: true

require 'devise/strategies/authenticatable'

module Devise

module Strategies

# Default strategy for signing in a user, based on their email and password in the database.

class DatabaseAuthenticatable < Authenticatable

def authenticate!

resource = password.present? && mapping.to.find\_for\_database\_authentication(authentication\_hash)

hashed = false

if validate(resource){ hashed = true; resource.valid\_password?(password) }

remember\_me(resource)

resource.after\_database\_authentication

success!(resource)

end

# In paranoid mode, hash the password even when a resource doesn't exist for the given authentication key.

# This is necessary to prevent enumeration attacks - e.g. the request is faster when a resource doesn't

# exist in the database if the password hashing algorithm is not called.

mapping.to.new.password = password if !hashed && Devise.paranoid

unless resource

Devise.paranoid ? fail(:invalid) : fail(:not\_found\_in\_database)

end

end

end

end

end

Warden::Strategies.add(:database\_authenticatable, Devise::Strategies::DatabaseAuthenticatable)

# frozen\_string\_literal: true

require 'active\_model/version'

require 'devise/hooks/activatable'

require 'devise/hooks/csrf\_cleaner'

module Devise

module Models

# Authenticatable module. Holds common settings for authentication.

#

# == Options

#

# Authenticatable adds the following options to devise\_for:

#

# \* +authentication\_keys+: parameters used for authentication. By default [:email].

#

# \* +http\_authentication\_key+: map the username passed via HTTP Auth to this parameter. Defaults to

# the first element in +authentication\_keys+.

#

# \* +request\_keys+: parameters from the request object used for authentication.

# By specifying a symbol (which should be a request method), it will automatically be

# passed to find\_for\_authentication method and considered in your model lookup.

#

# For instance, if you set :request\_keys to [:subdomain], :subdomain will be considered

# as key on authentication. This can also be a hash where the value is a boolean specifying

# if the value is required or not.

#

# \* +http\_authenticatable+: if this model allows http authentication. By default false.

# It also accepts an array specifying the strategies that should allow http.

#

# \* +params\_authenticatable+: if this model allows authentication through request params. By default true.

# It also accepts an array specifying the strategies that should allow params authentication.

#

# \* +skip\_session\_storage+: By default Devise will store the user in session.

# By default is set to skip\_session\_storage: [:http\_auth].

#

# == active\_for\_authentication?

#

# After authenticating a user and in each request, Devise checks if your model is active by

# calling model.active\_for\_authentication?. This method is overwritten by other devise modules. For instance,

# :confirmable overwrites .active\_for\_authentication? to only return true if your model was confirmed.

#

# You can overwrite this method yourself, but if you do, don't forget to call super:

#

# def active\_for\_authentication?

# super && special\_condition\_is\_valid?

# end

#

# Whenever active\_for\_authentication? returns false, Devise asks the reason why your model is inactive using

# the inactive\_message method. You can overwrite it as well:

#

# def inactive\_message

# special\_condition\_is\_valid? ? super : :special\_condition\_is\_not\_valid

# end

#

module Authenticatable

extend ActiveSupport::Concern

BLACKLIST\_FOR\_SERIALIZATION = [:encrypted\_password, :reset\_password\_token, :reset\_password\_sent\_at,

:remember\_created\_at, :sign\_in\_count, :current\_sign\_in\_at, :last\_sign\_in\_at, :current\_sign\_in\_ip,

:last\_sign\_in\_ip, :password\_salt, :confirmation\_token, :confirmed\_at, :confirmation\_sent\_at,

:remember\_token, :unconfirmed\_email, :failed\_attempts, :unlock\_token, :locked\_at]

included do

class\_attribute :devise\_modules, instance\_writer: false

self.devise\_modules ||= []

before\_validation :downcase\_keys

before\_validation :strip\_whitespace

end

def self.required\_fields(klass)

[]

end

# Check if the current object is valid for authentication. This method and

# find\_for\_authentication are the methods used in a Warden::Strategy to check

# if a model should be signed in or not.

#

# However, you should not overwrite this method, you should overwrite active\_for\_authentication?

# and inactive\_message instead.

def valid\_for\_authentication?

block\_given? ? yield : true

end

def unauthenticated\_message

:invalid

end

def active\_for\_authentication?

true

end

def inactive\_message

:inactive

end

def authenticatable\_salt

end

# Redefine serializable\_hash in models for more secure defaults.

# By default, it removes from the serializable model all attributes that

# are \*not\* accessible. You can remove this default by using :force\_except

# and passing a new list of attributes you want to exempt. All attributes

# given to :except will simply add names to exempt to Devise internal list.

def serializable\_hash(options = nil)

options = options.try(:dup) || {}

options[:except] = Array(options[:except])

if options[:force\_except]

options[:except].concat Array(options[:force\_except])

else

options[:except].concat BLACKLIST\_FOR\_SERIALIZATION

end

super(options)

end

# Redefine inspect using serializable\_hash, to ensure we don't accidentally

# leak passwords into exceptions.

def inspect

inspection = serializable\_hash.collect do |k,v|

"#{k}: #{respond\_to?(:attribute\_for\_inspect) ? attribute\_for\_inspect(k) : v.inspect}"

end

"#<#{self.class} #{inspection.join(", ")}>"

end

protected

def devise\_mailer

Devise.mailer

end

# This is an internal method called every time Devise needs

# to send a notification/mail. This can be overridden if you

# need to customize the e-mail delivery logic. For instance,

# if you are using a queue to deliver e-mails (active job, delayed

# job, sidekiq, resque, etc), you must add the delivery to the queue

# just after the transaction was committed. To achieve this,

# you can override send\_devise\_notification to store the

# deliveries until the after\_commit callback is triggered.

#

# The following example uses Active Job's `deliver\_later` :

#

# class User

# devise :database\_authenticatable, :confirmable

#

# after\_commit :send\_pending\_devise\_notifications

#

# protected

#

# def send\_devise\_notification(notification, \*args)

# # If the record is new or changed then delay the

# # delivery until the after\_commit callback otherwise

# # send now because after\_commit will not be called.

# if new\_record? || changed?

# pending\_devise\_notifications << [notification, args]

# else

# render\_and\_send\_devise\_message(notification, \*args)

# end

# end

#

# private

#

# def send\_pending\_devise\_notifications

# pending\_devise\_notifications.each do |notification, args|

# render\_and\_send\_devise\_message(notification, \*args)

# end

#

# # Empty the pending notifications array because the

# # after\_commit hook can be called multiple times which

# # could cause multiple emails to be sent.

# pending\_devise\_notifications.clear

# end

#

# def pending\_devise\_notifications

# @pending\_devise\_notifications ||= []

# end

#

# def render\_and\_send\_devise\_message(notification, \*args)

# message = devise\_mailer.send(notification, self, \*args)

#

# # Deliver later with Active Job's `deliver\_later`

# if message.respond\_to?(:deliver\_later)

# message.deliver\_later

# # Remove once we move to Rails 4.2+ only, as `deliver` is deprecated.

# elsif message.respond\_to?(:deliver\_now)

# message.deliver\_now

# else

# message.deliver

# end

# end

#

# end

#

def send\_devise\_notification(notification, \*args)

message = devise\_mailer.send(notification, self, \*args)

# Remove once we move to Rails 4.2+ only.

if message.respond\_to?(:deliver\_now)

message.deliver\_now

else

message.deliver

end

end

def downcase\_keys

self.class.case\_insensitive\_keys.each { |k| apply\_to\_attribute\_or\_variable(k, :downcase) }

end

def strip\_whitespace

self.class.strip\_whitespace\_keys.each { |k| apply\_to\_attribute\_or\_variable(k, :strip) }

end

def apply\_to\_attribute\_or\_variable(attr, method)

if self[attr]

self[attr] = self[attr].try(method)

# Use respond\_to? here to avoid a regression where globally

# configured strip\_whitespace\_keys or case\_insensitive\_keys were

# attempting to strip or downcase when a model didn't have the

# globally configured key.

elsif respond\_to?(attr) && respond\_to?("#{attr}=")

new\_value = send(attr).try(method)

send("#{attr}=", new\_value)

end

end

module ClassMethods

Devise::Models.config(self, :authentication\_keys, :request\_keys, :strip\_whitespace\_keys,

:case\_insensitive\_keys, :http\_authenticatable, :params\_authenticatable, :skip\_session\_storage,

:http\_authentication\_key)

def serialize\_into\_session(record)

[record.to\_key, record.authenticatable\_salt]

end

def serialize\_from\_session(key, salt)

record = to\_adapter.get(key)

record if record && record.authenticatable\_salt == salt

end

def params\_authenticatable?(strategy)

params\_authenticatable.is\_a?(Array) ?

params\_authenticatable.include?(strategy) : params\_authenticatable

end

def http\_authenticatable?(strategy)

http\_authenticatable.is\_a?(Array) ?

http\_authenticatable.include?(strategy) : http\_authenticatable

end

# Find first record based on conditions given (ie by the sign in form).

# This method is always called during an authentication process but

# it may be wrapped as well. For instance, database authenticatable

# provides a `find\_for\_database\_authentication` that wraps a call to

# this method. This allows you to customize both database authenticatable

# or the whole authenticate stack by customize `find\_for\_authentication.`

#

# Overwrite to add customized conditions, create a join, or maybe use a

# namedscope to filter records while authenticating.

# Example:

#

# def self.find\_for\_authentication(tainted\_conditions)

# find\_first\_by\_auth\_conditions(tainted\_conditions, active: true)

# end

#

# Finally, notice that Devise also queries for users in other scenarios

# besides authentication, for example when retrieving a user to send

# an e-mail for password reset. In such cases, find\_for\_authentication

# is not called.

def find\_for\_authentication(tainted\_conditions)

find\_first\_by\_auth\_conditions(tainted\_conditions)

end

def find\_first\_by\_auth\_conditions(tainted\_conditions, opts={})

to\_adapter.find\_first(devise\_parameter\_filter.filter(tainted\_conditions).merge(opts))

end

# Find or initialize a record setting an error if it can't be found.

def find\_or\_initialize\_with\_error\_by(attribute, value, error=:invalid) #:nodoc:

find\_or\_initialize\_with\_errors([attribute], { attribute => value }, error)

end

# Find or initialize a record with group of attributes based on a list of required attributes.

def find\_or\_initialize\_with\_errors(required\_attributes, attributes, error=:invalid) #:nodoc:

attributes.try(:permit!)

attributes = attributes.to\_h.with\_indifferent\_access

.slice(\*required\_attributes)

.delete\_if { |key, value| value.blank? }

if attributes.size == required\_attributes.size

record = find\_first\_by\_auth\_conditions(attributes) and return record

end

new(devise\_parameter\_filter.filter(attributes)).tap do |record|

required\_attributes.each do |key|

record.errors.add(key, attributes[key].blank? ? :blank : error)

end

end

end

protected

def devise\_parameter\_filter

@devise\_parameter\_filter ||= Devise::ParameterFilter.new(case\_insensitive\_keys, strip\_whitespace\_keys)

end

end

end

end

end

# frozen\_string\_literal: true

require 'devise/omniauth'

module Devise

module Models

module Omniauthable

extend ActiveSupport::Concern

def self.required\_fields(klass)

[]

end

module ClassMethods

Devise::Models.config(self, :omniauth\_providers)

end

end

end

end

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | | *Наименование* | | | | *Дополнительные сведения* | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР ДП 1–40 01 01 01 053 ПЗ | | | | | *Пояснительная записка* | | | | 122 с. | |
|  | | | | | Отзыв руководителя | | | |  | |
|  | | | | | Рецензия | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.551005-01 СА | | | | | Алгоритм добавления нового | | | | Формат А1 | |
|  | | | | | вопроса. | | | |  | |
|  | | | | | Схема алгоритма | | | |  | |
| ГУИР.551005-02 СА | | | | | Алгоритм взаимодействия с базой | | | | Формат А1 | |
|  | | | | | данных. | | | |  | |
|  | | | | | Схема алгоритма | | | |  | |
| ГУИР.551005-03 СА | | | | | Алгоритм подбора ответа. | | | | Формат А1 | |
|  | | | | | Схема алгоритма | | | |  | |
| ГУИР.551005-01 ПЛ | | | | | Модель базы данных приложения. | | | | Формат А1 | |
|  | | | | | Плакат | | | |  | |
| ГУИР.551005-02 ПЛ | | | | | Графический пользовательский | | | | Формат А1 | |
|  | | | | | интерфейс приложения. | | | |  | |
|  | | | | | Плакат | | | |  | |
| ГУИР.551005-03 ПЛ | | | | | Укрупненная архитектура | | | | Формат А1 | |
|  | | | | | программного средства. | | | |  | |
|  | | | | | Плакат | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | *БГУИР ДП 1-40 01 01 01 053 Д1* | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | *Веб-сервис вопросов и ответов на базе платформы Ruby on Rails.*  *Ведомость дипломного  проекта* | *Литера* | | | *Лист* | *Листов* |
| Разраб. | | Коваленко И.А. |  | 25.05 |  | *Т* |  | *122* | *122* |
| Пров. | | Парамонов А.И. |  | 28.05 | *Кафедра ПОИТ*  *гр. 551005* | | | | |
| Т.контр. | | Парамонов А.И. |  | 28.05 |
| Н.контр. | | Марина И.М. |  | 28.05 |
| Утв. | |  |  |  |