# **ВВЕДЕНИЕ**

В современном мире почти у каждого человека есть персональный компьютер и смартфон, нередко в дополнение к ним есть планшет, «умный» телевизор и «умные» часы. Часто все эти и другие электронные устройства используются для поиска информации в интернете.

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, современные сервисы довольно узконаправлены, для примера система вопросов ответов для программистов. Поэтому необходимо иметь учетную запись на каждом сервисе и искать нужный, когда вы хотите задать вопрос на определенную тему. Из-за существования большого количества небольших сервисов, количество посетителей на них небольшое. Поэтому вам придется ждать ответ довольно много времени.

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

Количество разнообразных электронных устройств с доступом в интернет постоянно растёт и поэтому проблема становится ещё более актуальной.

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

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.

В ходе дипломного проектирования была тщательно изучена предметная область для нахождения оптимального способа поиска поиска ответов на вопросы. Были рассмотрены существующие технологические решения с целью выбора наиболее подходящих инструментов и технологий для достижения цели дипломного проекта. Было выполнено тщательное проектирование приложения, разработаны эффективные алгоритмы поиска и сохранения информации. Было выполнено технико-экономическое обоснование эффективности разработки и реализации приложения.

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

## **1.1 Распределенное хранилище данных**

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. [1]

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. 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. [2]

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. 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;
* хэш‑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. [3]

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. 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.

Большим плюсом данной технологии является стоимость — пользовать платит только за то пространство, которое использует в данный момент, т. к. нет необходимости резервировать большое количество пространства ввиду хорошей масштабируемости [4].

## **1.2 Обзор существующих программ**

Существует немало решений, позволяющих пользователям размещать вопросы и искать на них ответы. Ниже приве­дены некоторые из задач, которые может выполнять типичный сервис вопросов и ответов:

– 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.

1.2.1 Ответы Mail.ru

Сервис вопросов и ответов, разработанный компаниией Mail.ru (см. рисунок 1.1). 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».

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

Ответы@Mail.ru используют сложную систему ранжирования пользователей, учитывающую размер и качество их вклада.

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.

Сервис монетизируется за счёт контекстной рекламы, а также VIP-статуса — платного функционала, расширяющего возможности пользователя. Кроме того, Mail.ru использует свои контентные сервисы как базу для формирования ответов на пользовательские запросы, в частности добавляет в выдачу содержимое «Ответов». В октябре 2014 года вышло приложение для платформы Android.[5]

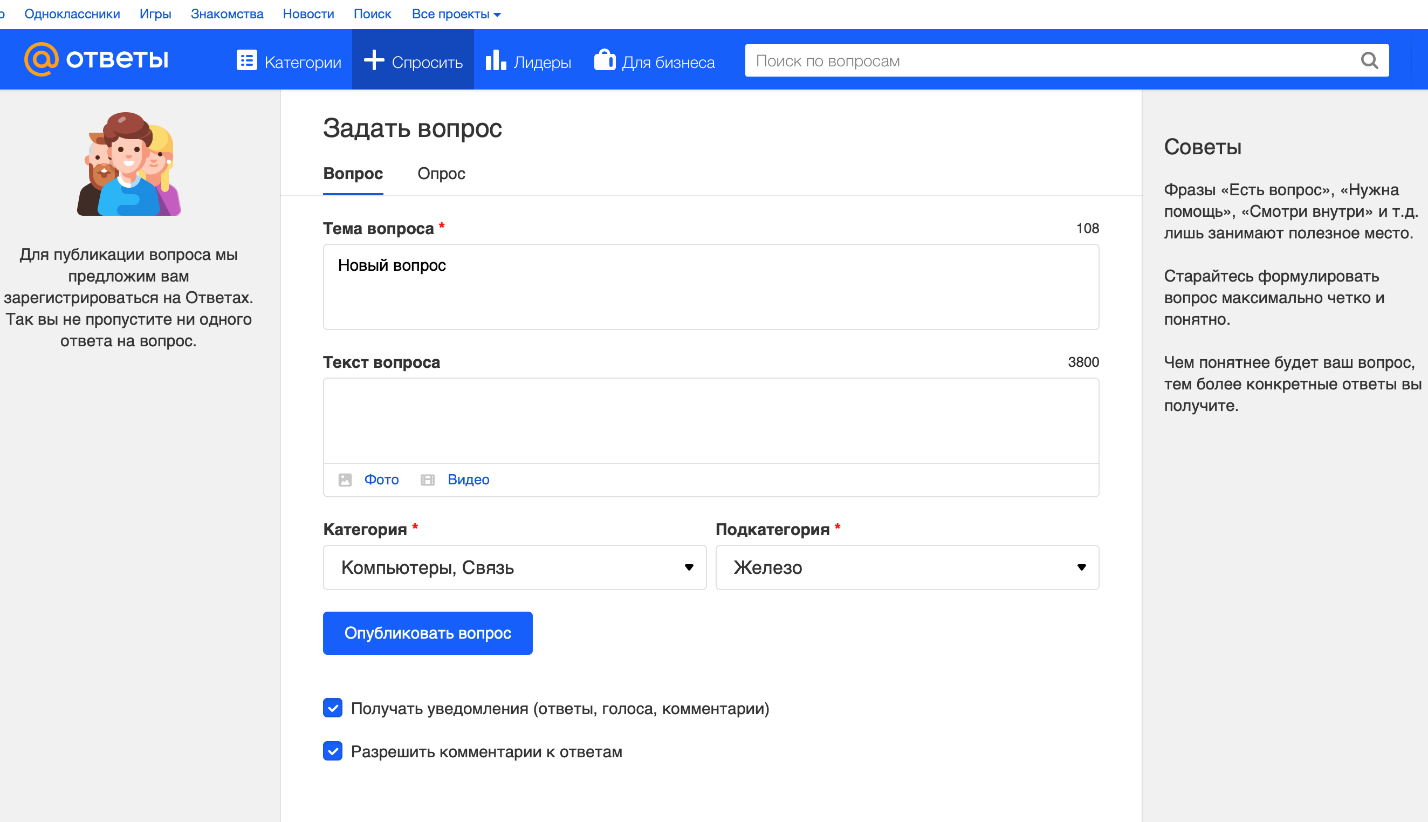


Рисунок 1.1 – Добавление нового вопроса на Ответы Mail.ru

1.2.2 Quora

Quora — это сайт вопросов и ответов, где вопросы задаются, отвечаются, редактируются и организуются сообществом пользователей в форме мнений. Его издатель, Quora Inc. Компания была основана в июне 2009 года, а сайт впервые стал доступен для общественности 21 июня 2010 года. Пользователи могут сотрудничать (см. рисунок 1.2), редактируя вопросы и предлагая изменения ответов, представленные другими пользователями.

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.

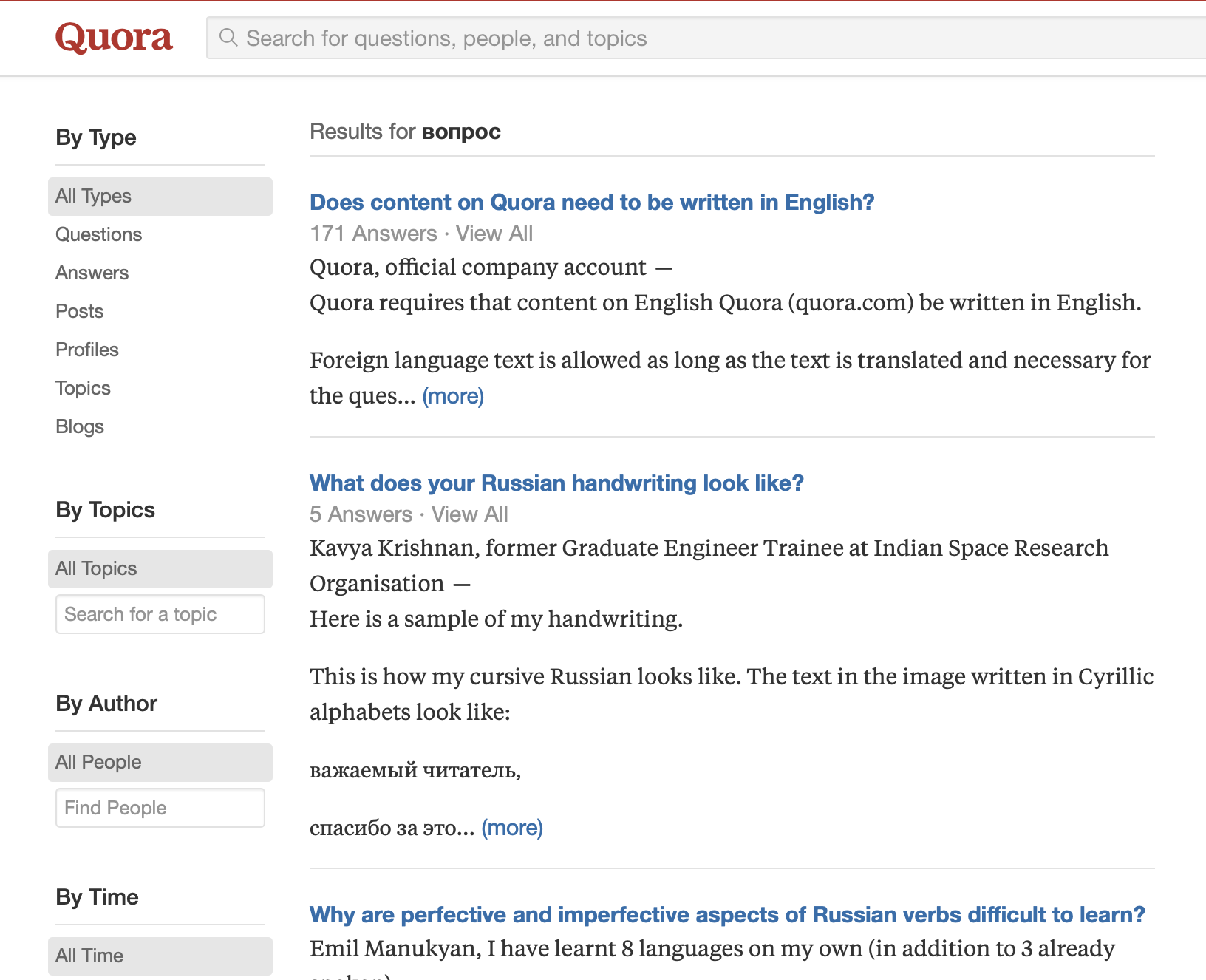


Рисунок 1.2 – Сервис вопросов и ответов Quora

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. [6]

1.2.3 Toster

Toster – сервис вопросов и ответов для IT специалистов. На «Тостере» можно получить ответ на вопрос по любой теме IT от участников сообщества, хорошо разбирающихся в этой теме (см. рисунок 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.

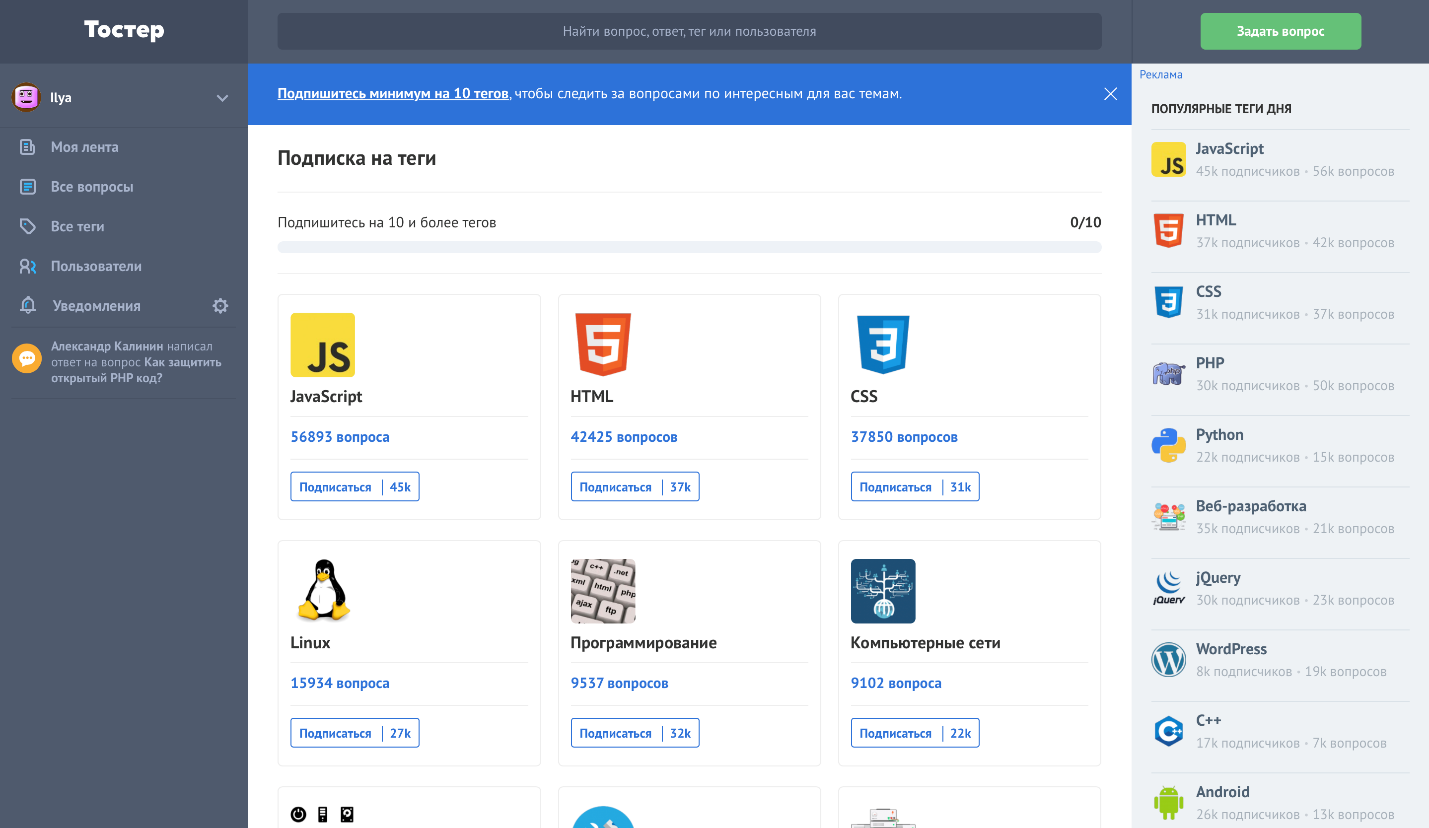
В разделе «Все теги» находится список всех существующих на сайте тем, среди которых можно найти интересующие и подписаться. Также любой тег можно найти с помощью глобального поиска по сайту.

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. [7]

1.2.4 ASKfm

ASKfm – сеть вопросов и ответов, запущенная в июне 2010 года. 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 (см. рисунок 1.4).

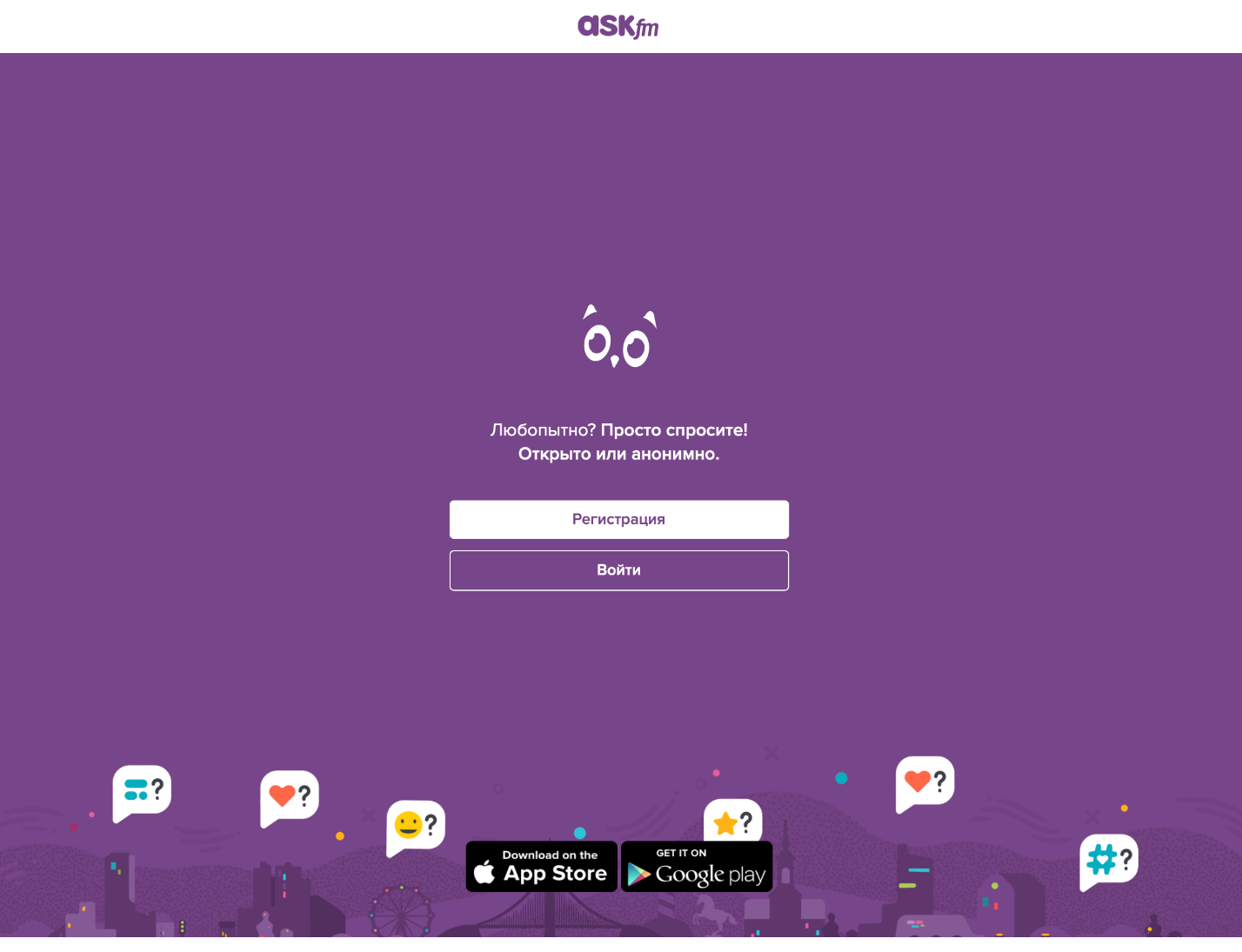


Рисунок 1.4 – Главная страница сервиса ASKfm

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. [8]

1.2.5 Stack Overflow

Stack Overflow — популярная система вопросов и ответов 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 (см. рисунок 1.5) 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.

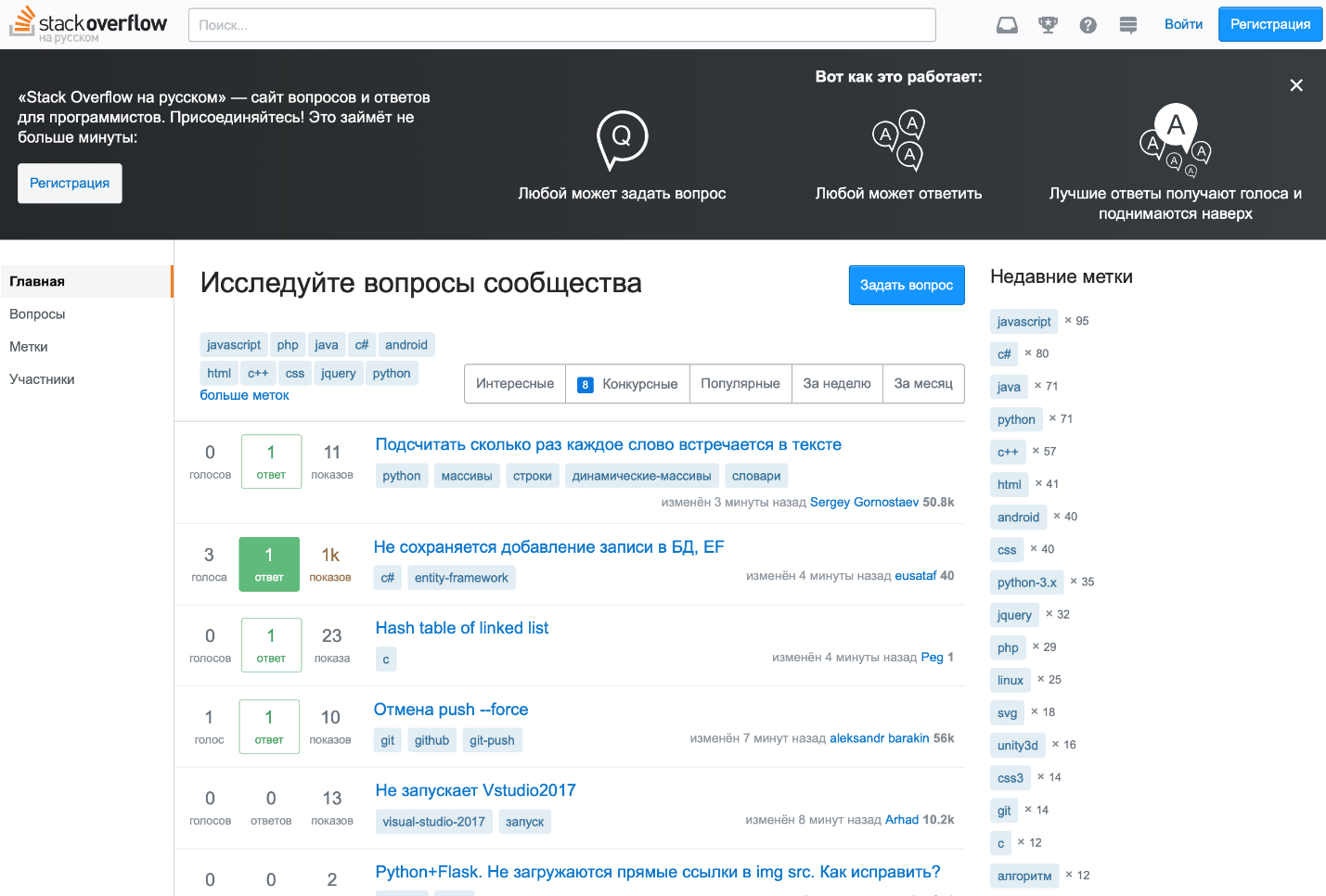


Рисунок 1.5 – Главная страница сервиса Stack Overflow

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. [9]

1.2.6 Askee

Askee.ru – сервис вопросов и ответов, где вы можете не только задавать интересующие вас вопросы, но давать ответы пользователям сайта (см. рисунок 1.6).

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

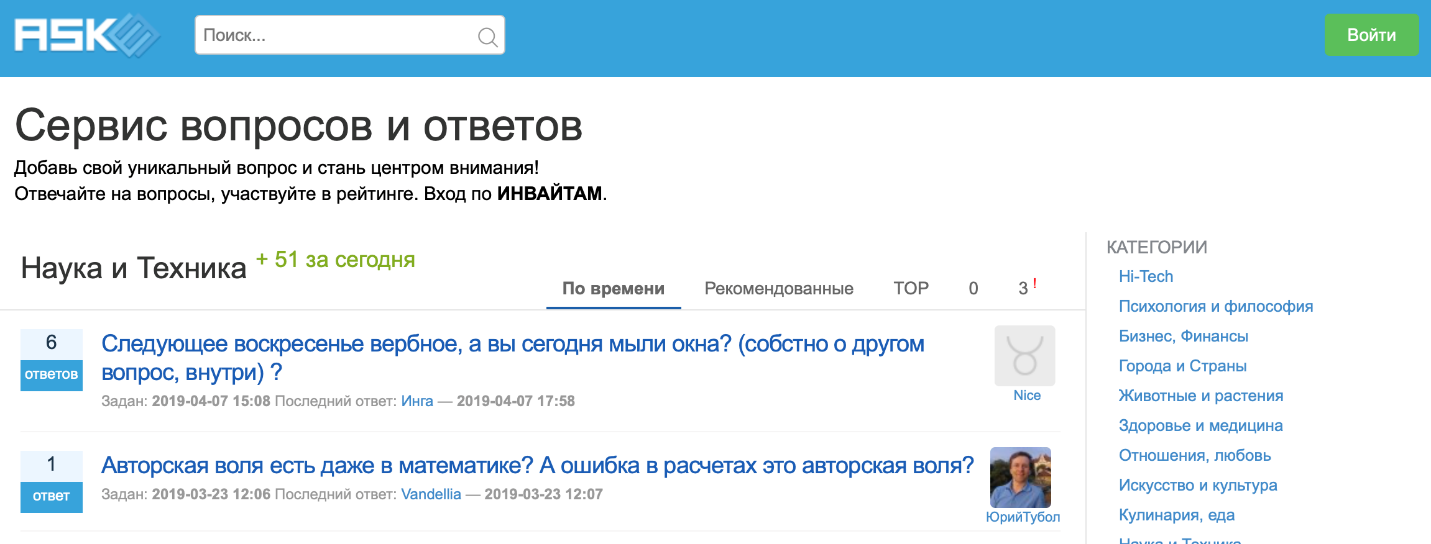


Рисунок 1.6 – Список вопросов на Askee

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. [10]

1.2.7 TheQuestion

TheQuestion – российский сервис вопросов и ответов (см. рисунок 1.7), 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. 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.

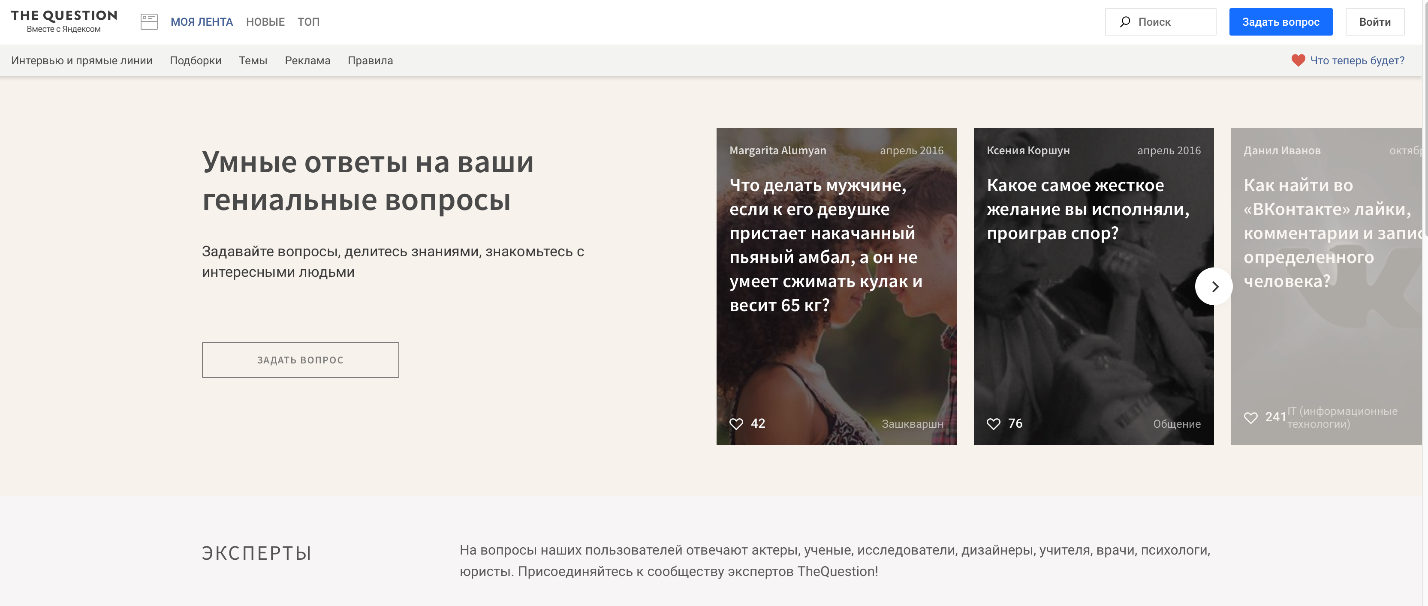


Рисунок 1.7 – Главная страница сервиса TheQuestion

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.

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. [11]

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

## **2.1 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 сущностей системы, с которыми пользователю 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.

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

Далее более подробно рассмотрены функциональные особенности разрабатываемого приложения, общая структура приложения и проектирование графического пользовательского интерфейса.

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

## **2.2 Функциональные особенности разрабатываемого приложения**

При проектировании приложения предусматривалась реализация большого количества функциональных возможностей, призванных обеспечить пользователя удобным инструментом для поиска ответов на свои вопросы. Их 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;
* система ролей, дающая 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:

* 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.

# **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 так называемая трёхуровневая архитектура.

В структуре приложения можно выделить три уровня (см. рисунок 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.

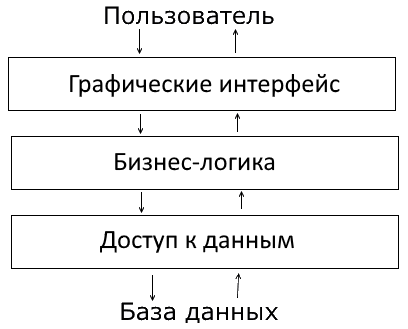


Рисунок 3.1 – Трёхуровневая архитектура приложения

Также в отдельное приложение можно выделить хранилище ключ-значение, который предоставляет функциональность, используемую основным веб-приложением. На этом этапе реализуется функциональность доступа к базе ключ-значение. Вынесение данного функционала в отдельное приложение позволяет абстрагироваться от используемой в настоящий момент базы данных.

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 3.2.

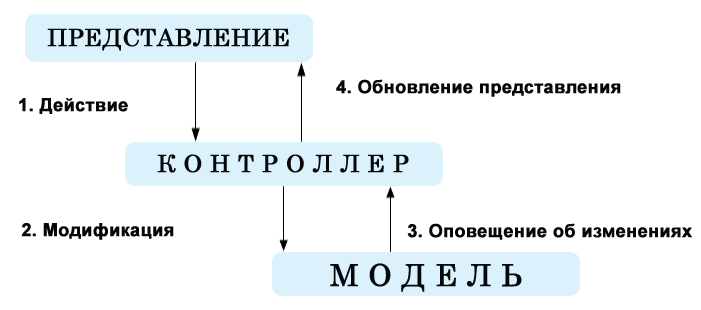


Рисунок 3.2 – Схема MVC

Так как необходимо обеспечить возможность масштабировать средство для быстрого поиска ответов на вопросы отдельно от основного приложения, базу данных с функцией поиска по ключам было решено вынести в отдельное приложение. 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.

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.

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.

Модуле 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 авторизоваться и зарегистрироваться в веб-сервисе. Если у пользователя имеется учетная запись, он сможет пройти аутентификацию со своими учетными данными.

## **3.3 Проектирование схемы базы данных**

Для хранения информации о вопросах, ответах, категориях, голосах, пользователях и ролей приложение будет использовать базу данных. Фрагмент схемы базы данных представлен на рисунке 3.3.

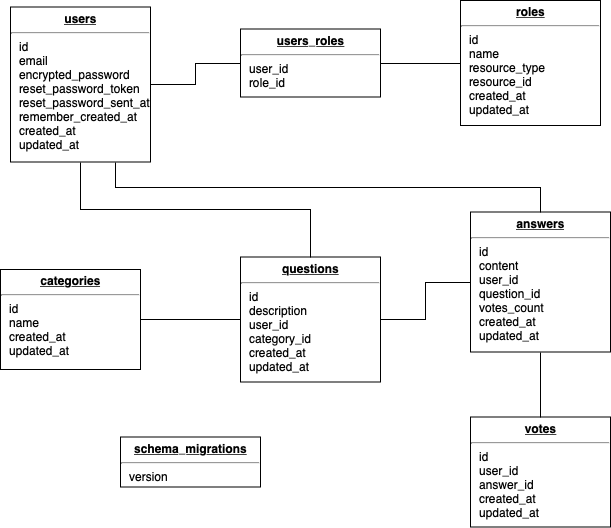


Рисунок 3.3 - Фрагмент схемы базы данных приложения

Ключевыми для приложения являются таблицы users, roles, categories, questions, answers, votes.

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;
* updated\_at – дата и время последнего обновления 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 данный голос;
* created\_at – дата и время создания голоса;
* updated\_at – дата и время последнего обновления голоса.

Таблица schema\_migrations служит для хранения информации о том, применилась ли к данной базе данных определенная миграция. Содержит единственное поле version, которое хранит идентификатор миграции.