АННОТАЦИЯ

Выпускная квалификационная работа на тему «Разработка методов оптимизации взаимодействия веб-сайта с базой данных» содержит 71 страниц текста, рисунков – 9, использованных источников – 38.

В современном мире трудно переоценить значимость баз данных. Базы данных прочно укрепили свои позиции повсеместно в промышленных, образовательных, здравоохранительных, правоохранительных и в других общественно важных структурах, а также в сферах бизнеса. Также, базы данных активно используются в веб-сайтах и от оптимальности взаимодействия веб-сайтов с базой данных зависит скорость работы первых. В свою очередь, скорость работы веб-сайтов влияет на комфорт их использования пользователями.

Ключевые слова: оптимизация, кеширвоание, Redis, брокер очередей, RabbitMQ, AMQP

Предмет исследования. Методы оптимизации работы веб-сайта с базой данных.

Объект исследования. Оптимизация взаимодействия веб-сайта с базой данных.

Цель работы. Разработать веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных**.**

Метод исследования. При исследовании применялся теоретический анализ литературы и материалов сети Интернет, методы структурного и объектно-ориентированного программирования.

Результаты работы. В работе разработаны веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных путём использования кеширования программных алгоритмов и синхронного выполнения CRUD-операция с базой данных.

Работа имеет теоретическое и практическое значение, т.к. базы данных активно используются в веб-сайтах и от оптимальности взаимодействия веб-сайтов с базой данных зависит скорость работы первых. В свою очередь, скорость работы веб-сайтов влияет на комфорт их использования пользователями.

ANNOTATION

The final qualifying work on the topic "Development of methods for searching websites with a database" contains 71 pages of text, figures – 9, used sources - 38.

In some cases, it is difficult to overestimate the sensitivity of databases. Databases have firmly established their positions in the field of industry, education, health, health care and other social and structural issues, as well as in the interests of business. In addition, database data is heavily used in websites and is different from other versions of database websites that depend on the speed of the former. In turn, the speed of the websites showed the convenience of their use by users.

Keywords: optimization, caching, Redis, queue broker, RabbitMQ, AMQP

Subject of study. Website performance optimization.

Object of study. Website flavor optimization with database.

Objective. Develop web libraries that implement methods of interaction between a website and a database.

Research method. Using the materials, a theoretical analysis of the literature and the Internet, methods of structural and object-oriented programming were prepared.

Work results. In the work, web libraries have been developed that implement methods for interacting websites with a database, using the use of caching software algorithms and synchronous execution of CRUD operations with the database.

The work has theoretical and practical significance, because. Database data is heavily used in websites and is distinguished by the unique effects of websites with a database that depend on the speed of the former. In turn, the speed of the websites showed the convenience of their use by users.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 9
ГЛАВА 1. ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ О ВЗАИМОДЕЙСТВИИ ВЕБ-САЙТА С БАЗОЙ ДАННЫХ 12
1.1. Описание реляционных баз данных 12
3.1. Описание операций взаимодействия с базой данных 17
ГЛАВА 2. Анализ МЕТОДОВ ОПТМИЗАЦИИ ВЗАИМОДЕЙСТВИЯ С БАЗОЙ ДАННЫХ 21
2.1. Метод кеширования результатов запросов к базе данных 21
2.2. Метод синхронного выполнения операций с базой данных 24
ГЛАВА 3. РАЗРАБОТКА ВЕБ-БИБЛИОТЕК 29
3.1. Инструменты разработки 29
3.1.1. Язык программирования PHP 29
3.1.2. Хранилище данных Redis 30
3.1.3. Протокола обмена сообщениями AMQP и брокера сообщений RabbitMQ 32
3.2. Постановка задачи 42
3.3. Разработка веб-библиотеки для кеширования работы программных алгоритмов 44
3.4. Разработка веб-библиотеки для синхронного выполнения операций с базой данных 49
3.5. Результат использования разработанных веб-библиотек 62
ЗАКЛЮЧЕНИЕ 66
СПИСОК ЛИТЕРАТУРЫ 68

ВВЕДЕНИЕ

Обоснование выбора темы и ее актуальность

В современном мире трудно переоценить значимость баз данных. Базы данных прочно укрепили свои позиции повсеместно в промышленных, образовательных, здравоохранительных, правоохранительных и в других общественно важных структурах, а также в сферах бизнеса. Также, базы данных активно используются в веб-сайтах и от оптимальности взаимодействия веб-сайтов с базой данных зависит скорость работы первых. В свою очередь, скорость работы веб-сайтов влияет на комфорт их использования пользователями.

Высокую нагрузку на базу данных вызывают операции создания, чтения, изменения и удаления данных. Эти операции можно объединить одной аббревиатурой — CRUD операции (Create, Read, Update, Delete).

Высокая нагрузка на базу данных, которая может возникать при выполнении CRUD-операции, увеличивает нагрузку на веб-сервер, что отрицательным образом сказывается на скорости работы веб-сайта в целом. Высокая нагрузка на веб-сервер потребует дополнительных затрат на его поддержку. А низкая скорость работы веб-сайта отрицательным образом скажется на комфорте использования сайта пользователями. Пользователей, в свою очередь, не будет устраивать скорость работы веб-сайта, в следствии чего они будут выбирать более производительные сайты конкурентов.

Таким образом, проблема оптимизации взаимодействия веб-сайта с базой данных очень актуальна. Особенно сильно это проблема актуально для веб-сайтов, занимающихся коммерческой деятельностью, например, интернет магазины. Ведь уровень комфорта использования интернет-магазинов напрямую влияет на количество активных пользователей, а соответственно, от этого зависит прибыль. Для решения проблемы оптимизации взаимодействия веб-сайта с базой данных, в рамках выпускной квалификационной работы, будут разработаны веб-

библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных.

Предмет исследования

Методы оптимизации работы веб-сайта с базой данных.

Объект исследования

Оптимизация взаимодействия веб-сайта с базой данных.

Цель работы

Разработать веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных.

Основные задачи исследования

1. Изучить операции взаимодействия веб-сайта с базой данных;

2. Проанализировать методы оптимизации взаимодействия веб-сайта с базой данных;

3. Разработать веб-библиотеки, реализующие методы оптимизации взаимодействия веб-сайта с базой данных.

Структура работы

Работа состоит из введения, трёх глав, заключения и списка источников.

Во введении рассматривается актуальность работы, ставится цель и формулируются задачи, необходимые для достижения поставленной цели.

Первая глава представляет собой описание операций взаимодействия с базой данных.

Во второй главе описываются методы оптимизации взаимодействия с базой данных.

В третьей главе описывается разработка веб-библиотек реализующих методы оптимизации веб-сайта с базой данных и результаты их использования.

В заключении делаются выводы по проделанной работе.

В конце работы приводится список использованных источников

ГЛАВА 1. ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ О ВЗАИМОДЕЙСТВИИ ВЕБ-САЙТА С БАЗОЙ ДАННЫХ

1.1. Описание реляционных баз данных

Реляционные базы данных — это средство эффективного хранения информации. База данных обеспечивает:

1. Защиту данных от потери;

2. Защиту данных от повреждений,

3. Позволяет экономно использовать человеческие и технические ресурсы;

4. Позволяет производить поиск информации, с помощью механизмов, которые отвечают оптимальным требованиям к производительности.

При проектировании и разработке баз данных, используют системы управления базами данных (СУБД).

Реляционные базы данных приобрели свою популярность благодаря внедрению реляционных моделей в СУЬД, что положительным образом сказалось на удобстве при работе с данными. Понятие «Система управления базами данных» разработал англичанин Эдгар Кодд. Реляционные модели характеризуются:

1. Простотой;

2. Табличной формой;

3. Использованием формальной математики и реляционных вычислений при обработке данных [22].

Хранение данных, в реляционных базах данных представлены в виде таблиц, которые, в свою очередь, состоят из строк и столбцов. Поля таблиц содержат собственное наименование. Столбцы таблицы могут содержать такие данные как даты, строки, числа и другие типы данных.

В реляционных базах данных, отношения между таблицами могут быть следующих типов:

1. Один к одному;

2. Многие ко многим;

3. Один ко многим.

В Данные, в реляционных базах данных, представлены в виде двумерного массива и имеют следующие особенности:

1. Любая таблица является обособленной частью данных;

2. Столбцы таблиц умеют уникальные название в пределах таблицы;

3. В таблицах отсутствуют одинаковые строки;

4. Данные в одном столбце имеют одинаковый тип;

5. Столбцы и строки могут иметь произвольный порядок.

Как правило, базы данных используют для моделирования предметных областей реального мира.

Модель данных — концептуальное описание предметной области. Модель данных содержит определение сущностей и атрибутов сущности: например, сущность «студент» содержит такие атрибуты, как, имя, курс, группа.

Предметная область — это часть реального мира, которую необходимо представить и смоделировать в виде базы данных.

Схема базы данных содержит описание модели данных, которую использует база данных.

Таким образом, базу данных можно представить в виде схемы базы данных и модели данных.

Механизм базы данных — это специальные средства, предназначенные для физического манипулирования данными: хранением их на диске и извлечением по запросу. SQL Server использует клиент-серверную архитектуру и предназначен для создания систем, от средних до больших. Он прекрасно масштабируется и

может поддерживать несколько тысяч пользователей, работающих с важными приложениями.

Реляционная модель основывается на математических принципах из теории множеств и логики предикатов. Реляционная модель определяет способ представления структур данных, методы зашиты целостности данных, а также операции манипулирования данными.

Кроме реляционной модели хранения и манипулирования данными, существуют также такие модели, как:

1. Иерархическая;

2. Сетевая;

3. Звездообразная модели данных.

Можно сформулировать основные принципы реляционных баз данных:

1. Данные упорядочены в виде строк и столбцов.

2. Значения данных имеют скалярный тип, что означает — для любой строки и столбца существует только одно значение.

3. Принцип замыкания, который означает, что все операции выполняются над целым отношением, и результатом выполнения этих операций также является целое отношение.

Принцип замыкания позволяет использовать результаты одной операции в качестве исходных данных для выполнения другой операции.

Рассмотрим термины, используемые в реляционной теории.

Сущность — это нечто, о чем нужно хранить информацию в разрабатываемой системе.

Атрибуты сущности — это записи об определенных параметрах каждой из сущностей.

Домен — это набор всех допустимых значений, которые может содержать атрибут.

Модель данных должна определять связи между сущностями. Связи представляют собой соотношения между сущностями.

Рассмотрим основные принципы нормализации реляционных баз данных. Принципы нормализации позволяют контролировать оптимальность структуры базы данных. Нормальные формы определяют строгость правил, которым подчиняется структура базы данных. Рассмотрим шесть нормальных форм. Каждая следующая нормальная форма расширяет предыдущую.

Реляционная модель позволяет связать отношения через атрибуты. Полностью нормальная модель данных подразумевает устранение избыточности. Для устранения избыточности, данные разбиваются на несколько отношений. Разбивать отношения на несколько нужно таким образом, чтобы при обратном соединении разделённых отношений получалась структура и данные исходного отношения.

Содержимое отношения — это неупорядоченное множество, которое состоит из нуля или более кортежей. Каждый элемент множества кортежей должен быть уникален. В таком случае для любого отношения должна существовать комбинация атрибутов, однозначно определяющая каждый кортеж. Такой набор из одного или более атрибутов называют ключом-кандидатом.

Ключом-кандидатом может являться любой атрибут. Каждый ключ-кандидат должен однозначно определять каждый кортеж любого специфического множества кортежей, а также для всех возможных кортежей в любой момент времени. Обратный логика тоже верен: два кортежа с одинаковыми значениями ключа-кандидата должны представлять одну и ту же сущность.

Первая нормальная форма. Отношение находится в первой нормальной форме, в том случает, если домены, являются скалярными величинами.

Вторая нормальная форма. Отношение находится во второй нормальной форме, если:

1. Отношение находится в первой нормальной форме;

2. Все атрибуты отношения зависят от полного набора атрибутов ключа-кандидата.

Третья нормальная форма. Отношение находится в третьей нормальной форме если:

1. Отношение находится во второй нормальной форме;

2. Все не ключевые атрибуты являются независимы.

Нормальная форма Бойса-Кодда. Нормальная форма Бойса-Кодда рассматривается как вариант третьей нормальной формы. Нормальная форма Бойса-Кодда имеет дело с отношениями, для которых существует несколько ключей-кандидатов. Для применения нормализации по Бойсу-Кодду, нужно выполнение следующих условий:

1. Отношение должно иметь больше двух ключей-кандидатов;

2. Более двух ключей-кандидатов должны являться составными;

3. Ключи-кандидаты должны содержать перекрывающиеся атрибуты.

Четвертая нормальная форма. Четвертая нормальная форма следует следующему принципу независимые повторяющиеся группы данных не следует размещать в одном и том же отношении. Четвёртая нормальная форма состоит в разделении многозначных зависимостей на разные отношения. Можно сказать, что отношение находится в четвертой нормальной форме, если:

1. Оно находится в нормальной форме Бойса-Кодда;

2. Все многозначные зависимости являются функциональными зависимостями от ключей-кандидатов.

Пятая нормальная форма. Пятая нормальная форма имеет дело в зависимости соединения. Зависимости соединения руководствуются принципу: «Если сущность 1 зависит от сущности 2, сущность 2 зависит от сущности 3, а сущность 3 в свою очередь зависит от сущности 1, то все три сущности обязательно должны входить в один и тот же кортеж».

Ранее был рассмотрен процесс нормализации модели данных. Суть этого процесса — в анализе сущностей предметной области для моделирования отношений, охватывающих все относящиеся к ним данные. Но кроме отношений, у модели данных есть такая часть, как — связи между отношениями и ограничения, налагаемые на эти связи.

Сущности, которые связаны между собой, называются «участниками». Размерностью связи называется число связанных участников. Связи между сущностями могут быть:

1. Двойные связи;
2. Унарные связи;
3. Тройные связи.

Связи можно классифицировать одним из трех возможных способов: как «полные» или «частичные», «необязательные» или «обязательные», а также в терминах «слабых» и «обычных» сущностей.

3.1. Описание операций взаимодействия с базой данных

CRUD — акроним, который обозначает четыре базовые операции, используемые при взаимодействии с базой данных:

1. Создание (англ. create),
2. Чтение (англ. read),
3. Модификация (англ. update),
4. Удаление (англ. delete).

Акроним CRUD введён Джеймсом Мартином (англ. James Martin) в 1983 году как классификация функций по манипуляции данными [1].

В системах, реализующих доступ к базе данных с помощью архитектурного стиль взаимодействия REST-API, CRUD функции реализуются через типы запросов POST, GET, PUT и DELETE, соответственно [14].

Рассмотрим подробнее процесс создания новой записи. Функция «Create» предназначена для добавления новых строк в таблицу. Это можно сделать с помощью команды INSERT INTO — ключевого слова, за которым следует название таблицы. Далее указываются имена столбцов и значения, которые нужно вставить (листинг 1). Функция «INSERT INTO» добавит новые строки в таблицу, и каждой созданной записи будет присвоен свой уникальный идентификатор.

Листинг 1 — Примеры запросов на добавление записи

```
INSERT INTO table_name
VALUES (value_1, value_2, value_3, -//-);
INSERT INTO table_name (column_1, column_2, column_3,-//-)
VALUES (value_1, value_2, value_3, -//-);
```

Функция «Read», позволяет извлекать определенные записи из базы данных и считывать их значения. Это можно сделать с помощью команды SELECT — ключевого слова, за которым следует название таблицы. Ниже приведен пример такого запроса (листинг 2).

Листинг 2 — Пример запроса вывода данных

```
SELECT * FROM table_name
```

Этот запрос не внесет никаких изменений в таблицу, а отобразит все существующие записи в этой таблице. Также можно указать критерий поиска записей посредством добавления секции WHERE (листинг 3).

Листинг 3 — Пример запроса вывода данных с условием

```
SELECT * FROM table_name
WHERE NAME LIKE ‘%a’
```

Обновление «Update» — данная операция позволяет изменять существующих записей в базе данных. Это можно сделать с помощью команды UPDATE. При выполнении операции UPDATE необходимо определить таблицу и столбцы, которые следует обновить (листинг 4). Также необходимо указать условие, по которому будут выбираться строки для обновления. В противном случае будут обновлены все указанные столбы таблицы.

Листинг 4 — Примеры запроса на обновление

```
UPDATE table_name
SET column1 = value_1, column_2 = value_2, -//-
WHERE ID = 3;
```

Операция «Delete» используется для удаления записи из таблицы. Это можно сделать с помощью команды DELETE (листинг 5). В SQL реализована возможность удаления, как одной конкретной записи, посредством указания соответствующего, так и удаление всех записей, содержащихся в таблице.

Листинг 5 — Пример запроса на удаление

```
DELETE FROM table_name
WHERE ID = 3;
```

Благодаря функциям чтения, создания, обновления и удаления удается организовать простое и правильное взаимодействие с хранимыми данными. Также позволяя разграничивать доступ пользователей по группам. Для одной группы разрешать только чтение записей, а другой предоставлять доступ к созданию и обновлению, удалению записей. Операции CRUD являются минимально необходимыми как для взаимодействия пользователей с системой, так и для разработчиков системы.

CRUD-операции требовательны к ресурсам веб-сервера, особенно это касается баз данных с большим количеством таблиц и строк в таблицах.

Операция «Read» способна спровоцировать высокую нагрузку на веб-сервер, в случае большого количества запросов к базе данных, например, по посещении страниц веб-сайта пользователями.

Операции «Create», «Update» и «Delete», кроме выполнения своих непосредственных функций, инициируют пересоздание индексов в таблицах баз данных, которые используются для более быстрого выполнения «Read» операций, что также создаёт дополнительную нагрузку на веб-сервер.

ГЛАВА 2. АНАЛИЗ МЕТОДОВ ОПТМИЗАЦИИ ВЗАИМОДЕЙСТВИЯ С БАЗОЙ ДАННЫХ

2.1. Метод кеширования результатов запросов к базе данных

Кеш — это высокоскоростной уровень хранения данных, как правило, в пределах ограниченного времени. Кэширование позволяет эффективно повторно использовать ранее полученные данные.

Как правило, кеш хранятся на устройства, которые позволяют получать данные с минимальной задержкой. Примером такого устройства может служить ОЗУ (оперативное запоминающее устройство). Основная цель, ради которой используют кеш — ускорение процесса получения, предварительно сохранённых, данных. Использование кеша избавляет от необходимости обращаться к менее быстрому базовому уровню хранения данных.

В кэше, как правило, хранится только необходимый набор данных, в течении ограниченного промежутка времени. Кеширование повышает скорость получения данных благодаря высоким показателям скорости обработки запросов в ОЗУ. Для обеспечения аналогичной скорости доступа к данным с помощью традиционных реляционных баз данных на базе жестких дисков, требуются дополнительные ресурсы, использование которых приводит к повышения расходов, но всё равно не позволяет достигнуть такой высокой скорости доступа к данным, какую может обеспечить кеш, хранимый в ОЗУ.

Кэш используется на разных технологических уровнях, включая операционные системы, сетевые уровни, в том числе сети доставки контента (CDN) и систем доменных имён (DNS), интернет-приложения и базы данных.

Использование кеширования позволяет сократить задержки при выполнении операций ввода-вывода для большинства приложений, с большой нагрузкой на чтение, например игровых ресурсов, мультимедийных порталов, социальных сетей.

Кешировать можно запросы и ответы к API, HTML файлы, JavaScript, результаты запросов к системам управления базами данных, произвольные вычисления.

При реализации кеширования необходимо контролировать достоверность данных, которые хранятся в кеше. Для удаления из кэша неактуальных данных применяются такие механизмы, как TTL (время жизни). А также механизм тегированного кеша, при котором кеш можно пометить определённым тегом и удалять неактуальный кеш при достижении произвольного события. Для обеспечения высокой скорости доступности к данным из кеша, можно использовать сервисы, позволяющие сохранять кеш в оперативной памяти, например Redis [38].

Преимущества использования кеширования:

1. Повышение производительности приложений;
2. Сокращение затрат на поддержку базы данных;
3. Снижение нагрузки на серверную часть;
4. Высокая пропускная способность операций чтения — количество операций ввода-вывода в секунду.

Для большинства веб-сайтов стоит кешировать:

- Изображения;
- CSS Стили;
- Javascript скрипты;
- Результаты запросов к базе данных;
- Работу сложных программных алгоритмов;
- HTML-страницы.

Не рекомендуется кешировать такие данные, как:

- Конфиденциальные данные;
- Часто изменяемый контент;

• Контент, зависимый от пользователя.

Бывают ситуации, когда использовать кеширование невозможно или оно не должно использоваться из-за принципа создания контента (например, если контент генерируется автоматически для каждого пользователя) или из-за содержимого контента (например, конфиденциальная информция). Еще одна проблема, с которой сталкиваются многие администраторы при настройке кеширования, — это ситуация, когда более старые версии вашего контента находятся в кэше и еще не устарели, даже если уже появились новые версии.

Эти распространенные проблемы могут отрицательным образом повлиять на скорость использования кэша и актуальность контента. Подобные проблемы помогают устранить разработка стратегии кеширования. Стратегия кэширования всегда зависит от ситуации, но есть общие рекомендации.

• Необходимо устанавливать отдельные директории для изображений, CSS-стилей и общего контента. Такой подход позволит легче ссылаться на них с любой страницы сайта;

• Необходимо использовать одинаковые URL-адреса для ссылок на одинаковые элементы. Поскольку кэш отталкивается как от хоста, так и от адреса к запрошенному контенту.

• Необходимо использовать CSS-спрайты для таких элементов как значки и навигация. Такой подход уменьшают количество запросов, необходимых для отрисовки сайта, и позволяют кешировать единственный спрайт в течение длительного промежутка времени.

• Внешние ресурсы необходимо хранить локально. Например, если используются javascript-скрипты и другие внешние ресурсы, их необходимо размещать на сервере, на котором расположен веб-сайт.

Грамотно составленная политика кеширования может оказать значительное положительное влияние на сайт. Кэширование позволяет сократить расходы, связанные с одновременным обслуживанием одного и того же контента. Сервер

сможет обрабатывать большее количество трафика с помощью того же аппаратного обеспечения. Эффективное кэширование может значительно повысить производительность. Самое главное — кэширование может улучшить пользовательский опыт использования веб-сайта, благодаря чему посетители будут возвращаться на сайт.

2.2. Метод синхронного выполнения операций с базой данных

Синхронного выполнения операций с базой данных, таких как добавление, изменение, удаление данных, можно добиться использовав брокеры сообщений.

Брокер сообщений — это отдельный сервис, который отвечает за хранение и доставку данных от сервисов-отправителей к сервисам-получателям с помощью модели Producer / Consumer.

Брокер сообщений — это тип архитектуры, в которой элементы системы взаимодействуют друг с другом с помощью посредника. Благодаря его работе снимается нагрузка с веб-сервисов, так как им не приходится пересылать сообщения: брокер сообщений берет на себя всю работу, по пересылке сообщений.

Брокер сообщений использует две основные сущности: producer (издатель сообщений) и consumer (потребитель/подписчик).

Producer (издатель сообщений) публикует информацию в виде сообщений, сгруппированных по какому-либо атрибуту;

Consumer (потребитель/подписчик) подписывается на потоки сообщений с определенными атрибутами и обрабатывает их.

Одна сущность занимается созданием сообщений и отправкой их другой сущности-потребителю. В процессе отправки сообщений, есть ещё один сервис, предназначенный для хранения сообщений, полученных от отправителя.

Возможны несколько вариантов передачи и получения сообщений:

1. Сообщение отправляется напрямую от отправителя к получателю;

2. Схема публикации/подписки.

В первом случае, когда сообщение отправляется напрямую от отправителя к получателю, каждое сообщение используется лишь однократно. Схему данного варианта передачи и получения сообщений можно увидеть на рисунке 1.

Рисунок 1 — Схема передачи сообщений напрямую от отправителя к получателю

Во втором случае, когда сообщения передаются по схеме публикации/подписки, отправитель не знает своих получателей и просто публикует сообщения в определённую тему. Потребители получают сообщений той темы, на которую они подписаны. На основе данной системы может быть выстроена работа по распределением задач между подписчиками. В подобных системах логика работы выстроена следующим образом. В одну и ту же тему публикуются сообщения для разных потребителей. Каждый потребитель видит уникальный атрибут своего сообщения и забирает его для исполнения. Схему данного варианта обмена сообщениями можно увидеть на рисунке 2.

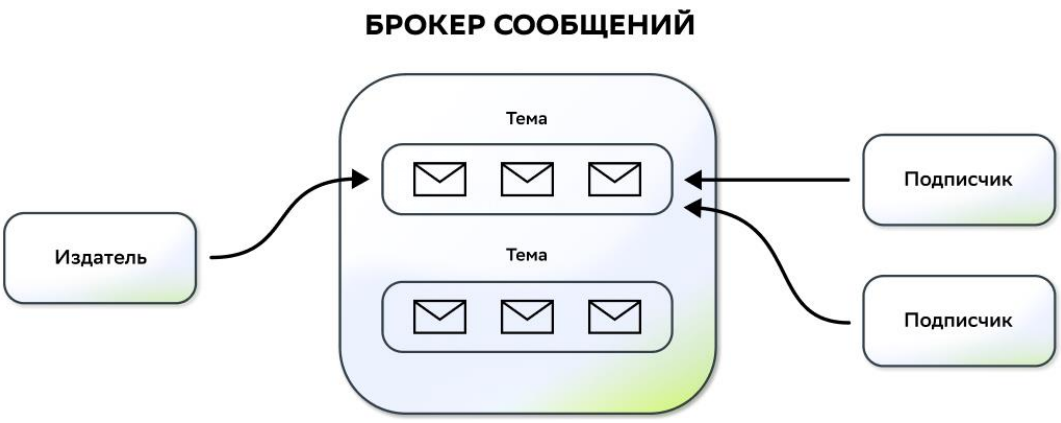

Рисунок 2 — Передача сообщений по схеме публикации/подписки

Группирующим сообщения атрибутом выступает очередь, которая нужна, чтобы разделять потоки сообщений. Таким образом, получатели могут подписываться только на те группы сообщений, которые их интересуют.

Очередь можно рассматривать как канал связи между автором и читателем сообщений. Авторы помещают сообщения в очередь, после чего сообщения передаются читателям, которые подписаны на эту очередь. Один читатель получает по одному сообщению, после чего оно становится недоступным для других читателей.

Под сообщением подразумевается единица информации, состоящая из тела сообщения и метаданных брокера. Тело представляет из себя набор байт определенного формата.

Получатель обязательно должен знать формат сообщения, иметь возможность обработать полученное сообщение.

Большинство брокеров сообщений работает по описанному выше принципу, основанном на AMQP (Advanced Message Queuing Protocol) — протоколе, описывающий стандарт отказоустойчивого обмена сообщений с использованием очередей.

Данный подход обеспечивает несколько важных преимуществ:

1. Масштабируемость;
2. Слабая связанность;
3. Эластичность.

Масштабируемость. Если сообщения появляются в очереди быстрее, чем читатель успевает их обрабатывать мы можем запустить несколько читателей и подписать их на одну очередь.

Слабая связанность гарантируется за счет асинхронной передачи сообщений, что позволяет отправителю передать данные и продолжить работать,

не дожидаясь ответа от получателя. Получатель, в свою очередь, обрабатывает сообщения, когда ему удобно, а не в момент отправки сообщения.

Эластичность. Наличие между приложениями посредника в виде очереди, помогает справляться с большим потоком данных в пиковые нагрузки. Очередь, в этом случае, выступает в роле буфера, в котором сообщения будут накапливаться и по мере возможности считываться получателем [10].

Брокеры сообщений используют для:

1. Для организации взаимодействия между разрозненными сервисами, даже если один из сервисов не работает в данный момент. То есть отправитель может отправлять сообщения, даже если получатель недоступен в данный момент;
2. Для увеличения производительности системы в целом за счёт асинхронной обработки задач;
3. Для обеспечения доставки сообщений.

Когда брокеры сообщений могут быть полезны:

1. Если система выполняет действия, которые требуют много времени на выполнение и потребляют большое количество ресурсов, но не требуют немедленного результата;
2. Если система отличается разветвлённой структурой, например, микросервисная архитектура, то для их взаимодействия между сервисами можно использовать брокер сообщений, который в этом случае будет выступать в роли связывающей сущности;
3. В мобильных приложениях, в которых используются push-уведомлений. Если множество смартфонов, на которых установлено мобильное приложение, подписаны на определённую тему сообщений, то смартфон сможет выводить уведомления у пользователей, считывая сообщений из брокера сообщений;

Брокеров сообщений существует большое множество. Например:

1. Apache ActiveMQ;

2. Apache Kafka;
3. Apache Qpid;
4. RabbitMQ.

Брокеры сообщений обладают определёнными возможностями. Например, одни используются для создания инфраструктуры между распределёнными частями приложения. Другие используются для интернета вещей, и работают на основе легковесного протокола MQTT. Подобные брокеры используются для сбора статистики, температуры и прочих показателей с распределённых датчиков, установленных на определённых устройствах

Небольшое время задержки при передаче сообщения по сети, а также возможность двунаправленной связи в реальном времени позволяют использовать брокеры сообщений, например, в управлении робототехническими устройствами в реальном времени. Такой принцип управления позволяет сократить задержки до десятков миллисекунд, что допустимо для низкоскоростных устройств.

ГЛАВА 3. РАЗРАБОТКА ВЕБ-БИБЛИОТЕК

3.1. Инструменты разработки

3.1.1. Язык программирования PHP

PHP (Hypertext Preprocessor) — это интерпретируемый язык программирования общего назначения с открытым исходным кодом. PHP специально сконструирован для веб-разработки и его код может внедряться непосредственно в HTML [9]. Главная область применения PHP — это написание скриптов, которые выполняются на веб-сервере.

PHP является веб-ориентированным языком программирования с динамической типизацией. Этот язык программирования можно сочетать с HTML кодом. Как правило, программы, написанные на языке программирования PHP, выполняются на веб-сервере, а результат отправляется браузеру в виде HTML разметки. Синтаксис языка программирования PHP схож с синтаксисом языка Си.

Основные возможности языка программирования PHP:

1. Получение параметров POST- и GET-запросов, а также переменных окружения веб-сервера;
2. Взаимодействие с системами управления базами данных (Cloudscape и Apache Derby, Microsoft SQL Server, Sybase, ODBC, mSQL, Informix, Ovrimos SQL, Lotus Notes, DB++, DBM, dBase, DBX, FrontBase, FilePro, MySQL, MySQLi, SQLite, PostgreSQL, Oracle Database (OCI8), IBM DB2, Ingres II, SESAM, Firebird и InterBase, Paradox File Access, MaxDB, интерфейс PDO, Redis);
3. Работа с HTTP-авторизацией;
4. Передача HTTP-заголовков;
5. Работа с сессиями и cookies;
6. Работа с удалёнными и локальными файлами;
7. Обработка загружаемых на сервер файлов;
8. Создание и взаимодействие с API;

9. Создание приложение с графическим интерфейсом пользователя при использовании фреймворка Qt Designer;

10. Создание консольных приложений.

3.1.2. Хранилище данных Redis

Redis (Remote DIctionary Server) — это нереляционное хранилище структур данных в памяти с открытым исходным кодом.

Redis позволяет работать с такими структурами данных, как:

1. Строки;
2. Хэши;
3. Наборы;
4. отсортированные наборы с запросами диапазона.

Redis позволяет выполнять, с перечисленными структурами данных, такие атомарные операции, как конкатенация строк, добавление элемента в список, увеличение значения в хэше, объединения и разности множеств, вычисление пересечения или получение элемента с наивысшим рейтингом в отсортированном наборе [31].

Максимальная производительность в Redis достигается тем, что Redis работает с данными в оперативной памяти. Для сохранности данных, Redis может периодически сохранять данные на постоянное запоминающие устройство, что обезопасит данные в случае обесточивания оперативных запоминающих устройств.

Redis может быть установлен на такие операционные системы, как: Linux, Windows, macOS.

Отличие Redis от реляционных СУБД:

1. Данные хранятся в оперативной памяти. Благодаря этому Redis выигрывает в производительности у реляционных СУБД;

2. Отсутствует язык SQL;

3. Данные хранятся не в виде таблиц, а в виде строк, списков, хешей, множеств, в том числе отсортированных.

В роли чего можно использовать Redis:

1. Как хранилище пользовательских сессий;

2. Как брокер сообщений;

3. Как СУБД для небольших приложений, блогов;

4. Для кэширования данных из основного хранилища, что значительно снижает нагрузку на реляционную базу данных. В качестве инструмента для кеширования Redis будет использован при разработке программного обеспечения для кеширования программных алгоритмов в рамках текущей работы;

5. Для хранения «быстрых» данных — когда важны скорость и критичны задержки передачи (аналитика и анализ данных, финансовые и торговые сервисы).

Основные команды для управления данными в Redis:

1. HSET — сохраняет значение по ключу;

2. HGET — получение значения по ключу (для определённого поля);

3. HGETALL — получение всех пар «ключ-значение»;

4. HKEYS и HVALS — получение всех ключей и соответствующих им значений.

С помощью подобных команд можно управлять данными непосредственно из командной строки операционной системы, что используется крайне редко. Зачастую, манипуляции с данными происходят с помощью языков программирования, для которых существуют готовые библиотеки, позволяющие взаимодействовать с Redis.

Язык программирования PHP имеет встроенные средства для взаимодействия с Redis и не требует установки сторонних библиотек. В рамках выпускной квалификационной работы Redis использовался при разработке

программного обеспечения для кеширования результаты работы программных алгоритмов.

3.1.3. Протокола обмена сообщениями AMQP и брокера сообщений RabbitMQ

AMQP (Advanced Message Queuing Protocol) — открытый протокол для передачи сообщений между компонентами системы. Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений [15]. Протокол AMQP вводит три понятия:

1. Exchange (обменник или точка обмена) — в неё отправляются сообщения. Обменник распределяет сообщение в одну или несколько очередей. Он маршрутизирует сообщения в очередь на основе созданных связей (binding) между ним и очередью.

2. Queue (очередь) — структура данных на диске или в оперативной памяти, которая хранит ссылки на сообщения и отдает копии сообщений consumers (потребителям). Одна очередь может использоваться несколькими потребителями.

3. Binding (привязка) — правило, которое сообщает точке обмена в какую из очередей эти сообщения должны попадать. Обменник и очередь могут быть связаны несколькими привязками.

Протокол AMQP работает поверх протокола TCP/IP.

RabbitMQ — это реализация AMQP с открытым исходным кодом. Сервер написан на Erlang и поддерживает несколько клиентов, таких как: Python, Ruby, .NET, Java, JMS, C, PHP, ActionScript, XMPP, STOMP. Он используется для хранения и пересылки сообщений в распределенной системе и хорошо работает с точки зрения простоты использования, масштабируемости и высокой

доступности. RabbitMQ маршрутизирует сообщения по всем базовым принципам протокола AMQP. Отправитель передает сообщение брокеру, а тот доставляет его получателю. RabbitMQ реализует и дополняет протокол AMQP.

Основная идея модели обмена сообщениями в RabbitMQ заключается в следующем: producer (отправитель сообщений) отправляет сообщения не на напрямую в очередь, а в точку обмена (exchange). Довольно часто отправитель не знает, в какую очередь будет доставлено сообщение. С одной стороны, точка обмена получает сообщения от отправителя, а с другой — отправляет их в очереди. Точка обмена должен точно знать, что делать с полученным сообщением. Нужно ли передать сообщение в определённую очередь, или сообщение необходимо передать в несколько очередей, или вовсе, необходимо игнорировать полученное сообщение [21].

Процессы, которые происходят в RabbitMQ можно описать следующим образом:

1. Отправитель отправляет сообщение в определённую точку обмена;
2. Точка обмена, получив сообщение, передаёт его в одну или несколько очередей;
3. Очередь хранит ссылку на это сообщение. Само сообщение хранится в оперативной памяти или на диске;
4. Как только получатель готов считать сообщение из очереди, сервер отправляет его получателю;
5. Получатель получает, обрабатывает сообщение и отправляет RabbitMQ подтверждение об обработке сообщения;
6. RabbitMQ, получив подтверждение, удаляет копию сообщения из очереди.

Для взаимодействия PHP и RabbitMQ использовалась сторонняя PHP-библиотека php-amqplib/php-amqplib. Примеры использования библиотеки php-

amqplib/php-amqplib для взаимодействия с брокером сообщений RabbitMQ будут рассмотрены ниже.

RPC. Процесс RPC (remote procedure call) лежит в основе практически всех взаимодействий с ядром RabbitMQ. Например, начальные обсуждения условий клиента с RabbitMQ, демонстрирует определённый процесс RPC. Как только эта последовательность завершится, RabbitMQ будет готов принимать запросы от клиента (рис. 3).

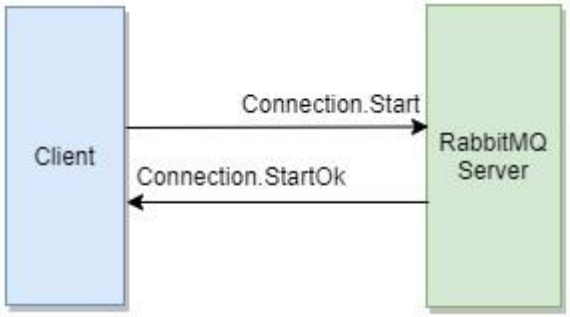

Рисунок 3 — Схема подключения клиента к RabbitMQ

Также в спецификации AMQP и клиент, и сервер могут вызывать команды. Это означает, что клиент ожидает взаимодействие с сервером. Команды — это классы и методы.

Подключение и каналы. Для такого обмена информацией между клиентом и сервером используются каналы. Каналы создаются в рамках определенного подключения. Каждый канал изолирован от других каналов. В синхронном случае невозможно выполнять следующую команду, пока не получен ответ (рис. 4).

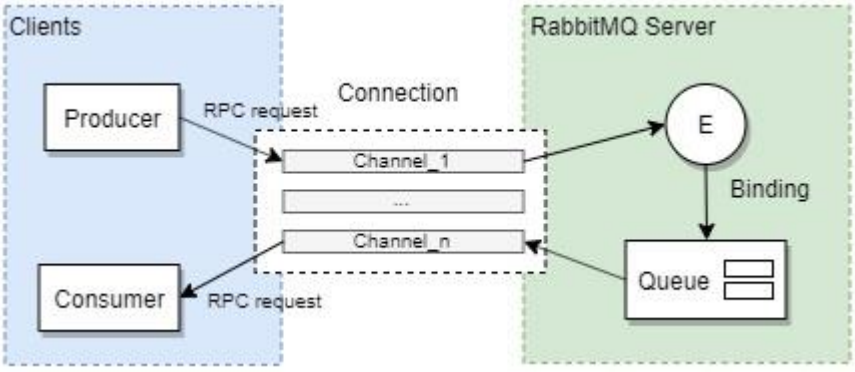

Рисунок 4 — Схема взаимодействия клиента к RabbitMQ

Простой пример создания подключения и канала при помощи AMQPStreamConnection можно увидеть в листинге 6.

Листинг 6 — пример создания подключения и канала при помощи AMQPStreamConnection

```
/**
     * Метод создания подключения с брокером сообщений
     * @return AMQPChannel
     */
    private static function getChannel():AMQPChannel
    {
        $host = 'localhost';
        $user = 'guest';
        $password = 'guest';
        $vhost = '/';
        $port = 5672;
        $connection = new AMQPStreamConnection($host,
$port, $user, $password, $vhost);

        $shutdown = function($connection, $channel)
        {
            if(!is_null($connection)){
                $connection->close();
            }
            if(!is_null($channel)){
                $channel->close();
            }
        };
        register_shutdown_function($shutdown, $connection,
$connection->channel());
        return $connection->channel();
```

```
    }
```

Не рекомендуется создавать новое подключение для каждой операции, так как это приводит к большим затратам ресурсов веб-сервера. Каналы также должны быть постоянными, но многие ошибки протокола приводят к закрытию канала, поэтому срок службы канала может быть короче, чем у соединения.

Exchange — обменник или точка обмена. Exchange распределяет сообщение между очередями. Маршрутизация сообщений между очередями происходит на основе созданных связей (bindings) между Exchange и очередью.

Рассмотрим виды exchange.

Direct exchange — следует использовать в случае, когда сообщения необходимо доставить в конкретные очереди. Сообщению задаётся определённый ключ маршрутизации, после чего оно передаётся в обменник. Обменник, в свою очередь, передаёт сообщение во все очереди, которые связаны с этим обменником соответствующим ключом маршрутизации. Ключ маршрутизации — это строка. Поиск соответствия ключа маршрутизации и очереди происходит при помощи сравнения строк на равенство друг другу. Графическое представление потока сообщений для direct exchange можно увидеть на рисунке 5.

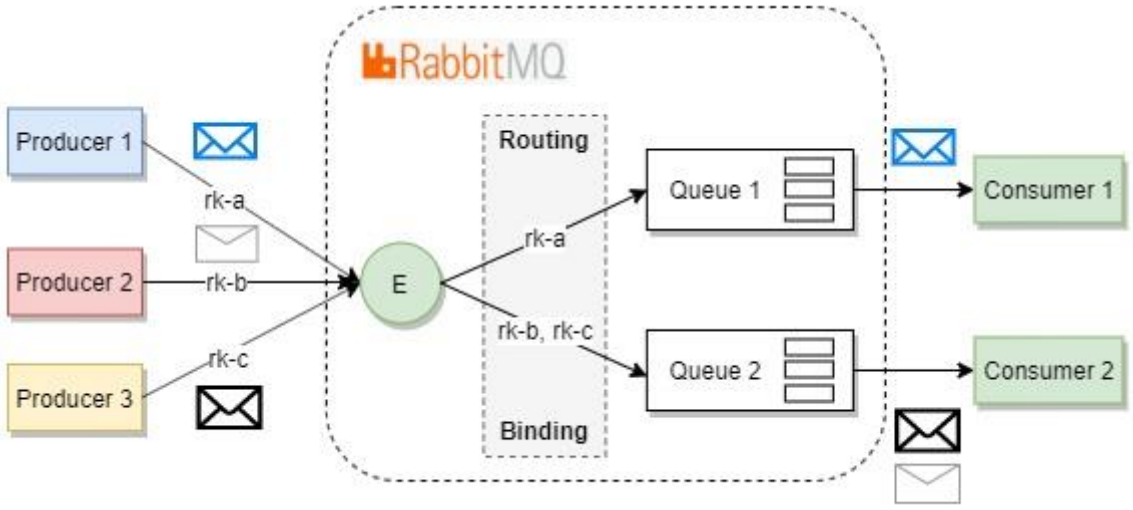

Рисунок 5 — Графическое представление потока сообщений в direct exchange

Topic exchange – аналогично direct exchange дает возможность осуществления выборочной маршрутизации путем сравнения ключа маршрутизации. Но, в данном случае, ключ задается по шаблону. При создании шаблона используются 0 или более слов (буквы AZ и az и цифры 0-9), разделенных точкой, а также символы * и #.

- * — может быть заменен на ровно 1 слово;
- # — может быть заменен на 0 или более слов.

Графическое представление потока сообщений для topic exchange можно увидеть на рисунке 6.

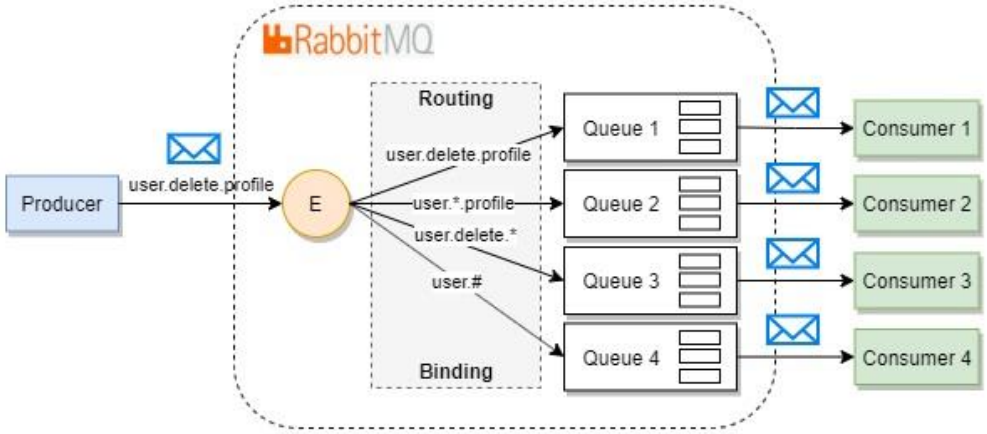

Рисунок 6 — Графическое представление потока сообщений в topic exchange

Fanout exchange – все сообщения доставляются во все очереди даже если в сообщении задан ключ маршрутизации.

Особенности:

- RabbitMQ не работает с ключами маршрутизации и шаблонами что положительно влияет на производительность. Это самый быстрый exchange;
- Все потребители должны иметь возможность обрабатывать все сообщения.

Графическое представление потока сообщений в Fanout exchange можно увидеть на рисунке 7.

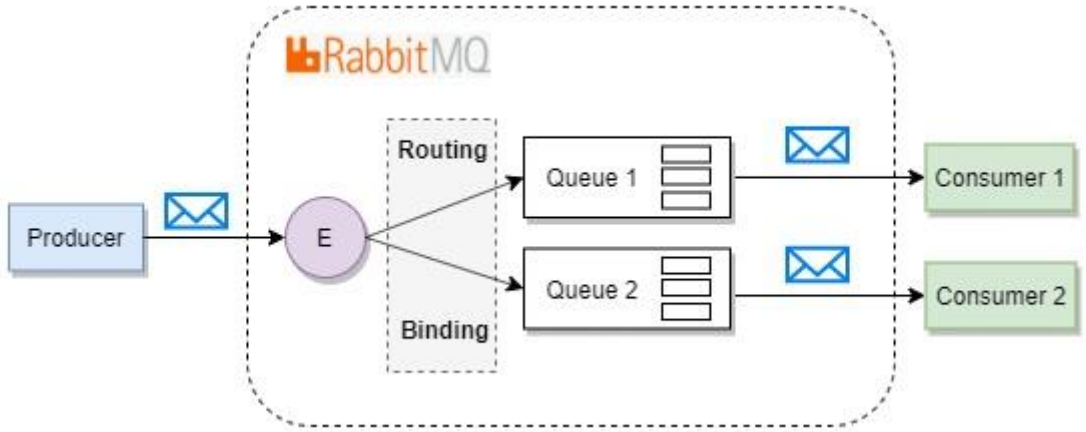

Рисунок 7 — Графическое представление потока сообщений в fanout exchange

Рассмотрим подробнее очереди (Queue) и привязки к очередям (Bindings).

Queue (очередь) — структура данных на диске или в оперативной памяти, которая хранит ссылки на сообщения и предоставляет их копии потребителям.

Binding (привязка) — правило, сообщающее обменнику, в какую из очередей должны попадать сообщения.

Если очередь создается с установленным параметром autoDelete, то очередь имеет возможность автоматически удалять себя. Такие очереди обычно создаются при подключении первого клиента и удаляются после отключения всех клиентов.

Если очередь создана с параметром exclusive, то такая очередь позволяет подключиться только одному потребителю и удаляется при закрытии канала. Пока канал не закрыт, клиент может отключиться/подключиться, но только в пределах одного соединения. Если установлена опция exclusive, то опция autoDelete не действует.

Если очередь создана с установленным параметром durable, то такая очередь сохраняет свое состояние и восстанавливается после перезапуска сервера/брокера. Эта очередь будет существовать до тех пор, пока не будет вызвана команда queue_delete.

Создание очереди. Очередь создается с помощью синхронного RPC-запроса к серверу. Запрос делается с помощью метода queue_declare. Пример создания очереди при помощи queue_declare можно увидеть в листинге 7.

Листинг 7. Пример создания очереди при помощи queue_declare

```
 // ...
 queue_declare (
    queue,
    durable,
    exclusive,
    autoDelete,
    arguments
);
// ...
```

Параметры метода queue_declare:

• queue — название очереди, которую мы хотим создать. Название должно быть уникальным и не может совпадать с системным именем очереди

• durable — если true, то очередь будет сохранять свое состояние и восстанавливается после перезапуска сервера/брокера

• exclusive — если true, то очередь будет разрешать подключаться только одному потребителю

• autoDelete — если true, то очередь обретает способность автоматически удалять себя

• arguments — необязательные аргументы.

Разберём подробнее параметр arguments:

• x-message-ttl(x-message-time-to-live) — позволяет установить время истечения срока действия сообщения в миллисекундах. Если очередь создается с установленным аргументом x-message-ttl, то очередь автоматически исключает сообщения с истекшим сроком действия. Установка значения аргумента x-message-ttl указывает максимальный возраст для всех сообщений в данной очереди. Создание такой очереди помогает предотвратить получение устаревшей информации. Это можно использовать в системах реального времени. Установка

аргумента x-message-ttl для очереди, для которой установлен обменник отклоненных сообщений, приведет к тому, что отклоненные сообщения в этой очереди будут иметь дату истечения срока действия;

- x-expires — устанавливает значение в миллисекундах, после которого очередь удаляется. Очередь может истечь, только если у нее нет подписчиков. Если подписчики подключены к очереди, она может быть автоматически удалена только тогда, когда все подписчики вызывают Basic.Cancel или отключаются. Очередь может истечь только в том случае, если к ней не был сделан запрос Basic.Get. В противном случае текущее значение параметра времени жизни сбрасывается до нуля, и очередь больше не будет автоматически удаляться. Также нет гарантии того, как быстро будет удалена очередь после истечения срока ее существования;
- x-max-length — устанавливает максимальное количество сообщений в очереди. Если количество сообщений в очереди начинает превышать максимальное количество, то самые старые начинают удаляться;
- x-max-lenght-bytes — устанавливает максимально допустимый общий размер полезной нагрузки сообщений в очереди. При превышении установленного значения (переполнение очереди при публикации следующего сообщения) самые старые сообщения будут удаляться;
- x-overflow — этот аргумент используется для настройки поведения при переполнении очереди. Доступны два значения: drop-head (по умолчанию) и reject-publish. Если вы выберете drop-head, то самые старые сообщения будут удалены. Если выбрать reject-publish, то прием сообщений будет приостановлен;
- x-dead-letter-exchange — указывает exchange, на который отправляются отклонённые сообщения, которые не были повторно поставлены в очередь;
- x-dead-letter-routing-key — указывает необязательный ключ маршрутизации для отклоненных сообщений;
- x-max-priority — включает сортировку по приоритету в очереди с максимальным значением приоритета 255 (версии RabbitMQ 3.5.0 и выше). Число

указывает максимальный приоритет, который будет поддерживать очередь. Если аргумент не установлен, очередь не будет поддерживать приоритет сообщений.;

• x-queue-mode — позволяет перевести очередь в ленивый режим. В этом режиме на диске будет храниться как можно больше сообщений. Использование оперативной памяти будет минимальным. Если он не установлен, очередь будет хранить сообщения в памяти, чтобы доставлять сообщения как можно быстрее;

Повторный вызов queue_declare с теми же параметрами вернет полезную информацию об этой очереди. Например, общее количество сообщений, ожидающих в этой очереди, и общее количество потребителей, подписавшихся на нее.

После простоя очереди в течение 10 секунд она переходит в спящий режим, что приводит к значительному уменьшению требуемой для этой очереди памяти.

Создание Binding. Привязка создается с помощью синхронного запроса RPC к серверу. Запрос делается с помощью метода queue_bind, вызываемого с параметрами:

• название очереди;

• название точки обмена;

• другие параметры.

Пример создания привязки при помощи RabbitMQ.Client можно увидеть в листинге 8.

Листинг 8. Пример создания привязки при помощи queue_bind

```
 //...
queue_bind (
    queue,
    exchange,
    routingKey,
    arguments
);
//...
```

Параметры метода queue_bind:

- queue — название очереди;
- exchange — название обменника;
- routingKey — ключ маршрутизации;
- arguments — необязательные аргументы.

В рамках выпускной квалификационной работы, брокер сообщений RabbiqMQ использовался при разработке программного обеспечения для обмена сообщениями по протоколу AMQP.

3.2. Постановка задачи

Одни из самых ресурсоёмких операций с базой данных — это CRUD операции со строками таблиц. Под аббревиатурой CRUD скрываются такие операции как:

1. Создания записей (Create);
2. Чтение записей (Read);
3. Изменение записей (Update);
4. Удаление записей (Delete).

CRUD операции соответствуют следующие HTTP-методы:

1. Создать запись — метод POST;
2. Прочитать записать — метод GET;
3. Изменить запись — метод PUT;
4. Удалить запись — метод DELETE.

Зачастую, на веб-сайтах, CRUD-операции инициируют пользователи. Например, при посещении страниц веб-сайта, на которых присутствует динамически-изменяемая информация, хранящаяся в базе данных, пользователи инициируют операцию чтения данных. На крупных веб-сайтах, особенно в интернет-магазинах, где количество товаров исчисляется десятками тысяч и тысячами активных пользователей, операция чтения данных способна вызвать серьёзную нагрузку на сервер, в результате чего скорость загрузки страниц веб-

сайта будет на неудовлетворительном уровне, а поддержка веб-сервера будет требовать больших финансовых затрат. Пользователей, в свою очередь, не будет устраивать скорость работы веб-сайта, в следствии чего будет уменьшаться количество активных пользователей, а следовательно, прибыль от работы интернет-магазина будет падать.

Для снижения нагрузки на базу данных, которую создают большое количество запросов на чтение записей, можно воспользоваться кешированием запросов к базе данных. Но простых в использовании веб-библиотек, предоставляющих возможность кешировать запросы к базе данных, найдено не было. Поэтому было решено разработать веб-библиотеку, которая позволит кешировать не только запросы к базе данных, а также и кешировать работу программных алгоритмов. А храниться кеш будет в оперативной памяти сервера, что положительно скажется на скорости сохранения кеша и получения кеша.

Что касается операций создания, изменения, удаления записей. Данные операции также инициируют пользователи веб-сайтов. Подобные операции могут вызываться, например, при отправке каких-либо заявок с сайта, оформление заказа, добавление и удаление товаров из корзины интернет-магазина. Когда тысячи пользователей инициируют подобные операции в один момент времени — это отрицательным образом сказывается на нагрузке как на базу данных, так и в целом на веб-сервер. В следствии чего, как и в случае запросов на чтение записей из базы данных, скорость работы сайта падает, количество активных пользователей уменьшается, финансовые затраты на поддержку веб-сервера увеличиваются.

Для снижения нагрузки на базу данных, которую вызывают операции создания, изменения, удаления записей, подойдёт отложенная обработка этих операций путём добавления их в очередь задач и дальнейшего выполнения задач, находящихся в очереди. Готовых, простых в использовании решений, позволяющих реализовать механизм добавления задач в очередь и механизм чтения задач из очереди, не было найдено. Поэтому, было решено разработать

веб-библиотеку, которая позволит добавлять задачи в очередь, забирать задачи из очереди и их исполнять. Также разрабатываемая веб-библиотека будет отличаться высокой масштабируемостью. Под масштабируемостью подразумевается лёгкое создание произвольного количества очередей, а также создание произвольного количества обработчиков очередей. Для одной очереди может быть создано несколько обработчиков, что позволит ускорить обработку задач.

В итоге, была поставлена задача по разработке веб-библиотек:

1. Веб-библиотека для кеширования работы программных алгоритмов;
2. Веб-библиотека для добавления задач в очередь и выполнение этих задач.

Веб-библиотеки должны отличаться простотой в использовании, высокой производительностью и масштабируемостью.

3.3. Разработка веб-библиотеки для кеширования работы программных алгоритмов

При разработке веб-библиотеки для кеширования программных алгоритмов использовались следующие инструменты:

1. Язык программирования PHP;
2. Нереляционное хранилище данных Redis;
3. Система контроля версия Git;
4. IDE PhpStorm.

У языка программирования php уже есть встроенные инструменты для взаимодействия с Redis. На основе этих инструментом и будет разработана веб-библиотека.

Веб-библиотека представляет из себя PHP-класс, который предоставляет инструменты взаимодействия с кешем, хранящемся в Redis.

Опишем методы класса. Самый основной метод — это конструктор класса, в который передаются параметры, от которых зависит куда или от куда будет получен кеш. Программный код конструктора класса отражён в листинге 9.

Листинг 9 — Конструктор класса веб-библиотеки для кеширования программных алгоритмов

```
/**
     * @param array $keyData - массив с данными, из которых нужно сделать ключ кеша
     * @param int $ttl - время жизни кеша
     * @param string $tag - теги кеша, можно использовать для удаления кеша по тегу
     */
    public function __construct(array $keyData, int $ttl = 3600, string $tag = '')
    {
        $this->redis = new \Redis();
        $this->redis->pconnect('localhost');
        $keyData[] = $ttl;
        if(!empty($tag))
        {
            $keyData[] = $tag;
        }
        $this->key = md5(serialize($keyData));
        if(!empty($tag))
        {
            $this->key = "TAG_" . $tag . ":" . $this->key;
        }
        $this->ttl = $ttl;
    }
```

В конструктор передаётся три параметра:

1. $keyData;

2. $ ttl;

3. $tag.

Параметр $keyData — это массив с данными, от которых зависит идентификатор сохраняемого кеша. В дальнейшем, по этому идентификатору можно будет получить сохранённые в кеше данные. В качестве этого параметра, например, может выступать условие выборки записей из базы данных. Чтобы преобразовать параметр $keyData в идентификатор кеша, этот параметр сериализуется и хешируется с помощью алгоритма md5.

Параметр $ttl представляет из себя целочисленное значение и необходим для задания времени жизни кеша в секундах. Параметр является необязательным и по умолчанию его значение равно 3600 секунд.

Параметр $tag представляет из себя строку и необходим для поддержки механизма тегированного кеша. Одним тегом можно объединить множество закешированных данных. Затем, по тегу, можно либо получить или удалить данные, объединённые одним тегом.

Для получения закешированных данных реализован метод getCache. Программный код этого метода можно увидеть в листинге 10.

Листинг 10 — Метод для получения закешированных данных

```
/**
     * Получение закешированных данных. Если данных нет, то
вернёт false
     * @return bool|array
     */
    public function getCache()
    {
        if($this->isExists())
        {
```

```
            return     unserialize(self::$redis->get($this-
>key));
        }

        return false;
    }
```

В случае, если метод getCache не вернул результат, вызывается метод setCache, который предназначен для сохранения данных в кеш. В данный метод необходимо передать один параметр-массив, в котором должны находится сохраняемые в кеш данный. Программный код методе setCache представлен в листинге 11.

Листинг 11 — Метод для сохранения данных в кеш

```
/**
     * @param array $data - массив с данными, которые нужно
сохранить в кеш
     */
    public function setCache(array $data)
    {
        self::$redis->set($this->key,     serialize($data),
$this->ttl);
    }
```

Для удаления кеша, объединённого одним тегом, реализован метод clearCacheByTag. В этот метод необходимо передать один параметр, являющийся тегом кеша. Программный код метода clearCacheByTag представлен в листинге 12.

Листинг 12 — Метод для очистки кеша, объединённого общим тегом

```
/**
     * Метод очищает кеш по тегу
     * @param string $tag - тег кеша
     * @return void
     */
    public function clearCacheByTag(string $tag): void
    {
        self::$redis->del($this->getKeysByTag($tag));
    }
```

Также, можно получить все закешированные данные, объединённые одним тегом. Для этого реализован метод getCacheByTag. Программная реализация этого метода представлена в листинге 13.

Листинг 13 — Метод получения данных, объединённых общим тегом

```
/**
     * Метод очищает кеш по тегу
     * @param string $tag - тег кеша
     * @return array
     */
    public function getCacheByTag(string $tag): array
    {
        $arKeys = $this->getKeysByTag($tag);
        $arResult = [];
        foreach ($arKeys as $key)
        {
            if($data = self::$redis->get($key));
            {
                $arResult[] = $data;
            }
```

```
        }
        return $arResult;
    }
```

Таким образом, была разработана веб-библиотека для кеширования работы программных алгоритмов.

Данная веб-библиотека обладает следующими возможностями:

1. Сохранение произвольных данных в кеш. Кеш сохраняется в оперативной памяти. Для хранения кеша в оперативной памяти использовался инструмент Redis;

2. Получение данных из кеша по ключу;

3. Задание времени жизни кеша;

4. Задание кешу произвольного тега для получения или удаление кеша по тегу.

3.4. Разработка веб-библиотеки для синхронного выполнения операций с базой данных

При разработке веб-библиотеки использовались следующие инструменты:

1. Язык программирования php;
2. Composer — это менеджер пакетов для php;
3. Система контроля версия Git;
4. IDE PhpStorm.

Язык программирования PHP не имеет инструментов, реализующих стандарт отказоустойчивого обмена сообщениями посредством очередей по протоколу AMQP. Для этого была использована сторонняя веб-библиотека php-amqplib, которая была установлена с помощью менеджера пакетов Composer. На основе php-amqplib и была разработана веб-библиотека для синхронного выполнения операций с базой данных.

Библиотека представляет из себя три класса:

1. Connector — класс для взаимодействием с брокером очередей (установка соединения, отправка и получение сообщений);

2. aMessageProcessing — абстрактный класс для обработки получаемых сообщений. Этот класс обеспечивает высокий уровень масштабируемости, при создании новых классов-обработчиков сообщений, за счёт наследования этого класса.

Рассмотрим основные методы класса Connector. Как было сказано ранее, данных класс обеспечивает соединение с брокером сообщений, а также предназначен для передачи сообщений и получения сообщений, с последующей его передачей в соответствующий класс-обработчик.

Метод getMappingRoutingKeyClass. Данный метод возвращает php-массив соответствий ключа маршрутизации и класса-обработчика сообщений. Ключ маршрутизации передаётся вместе с сообщением и необходим для передачи сообщения в нужный обработчик. Таким образом, при создании нового класса-обработчика, необходимо предусмотреть ключ маршрутизации для этого класса и записать это соответствие в метод getMappingRoutingKeyClass. Пример реализации этого метода можно увидеть в листинге 14.

Листинг 14 — Метод получения соответствий ключа маршрутизации и класса-обработчика сообщений

```
  /**
     *  Метод  возвращает  массив  соответствий  ключей
маршрутизации сообщений и классов для обработки сообщений
     * @return array
     */
    private                 static                 function
getMappingRoutingKeyClass():array
    {
```

```
        return [
            Storage::ROUTING_KEY => Storage::class,
        ];
    }
```

Метод getChannel. Данный метод предназначен для установления соединения с брокером сообщений. Внутри этого метода прописываются доступы к брокеру сообщений (хост, логин, пароль и порт). Данный метод возвращает экземпляр класса AMQPChannel, который реализован в веб-библиотеке php-amqplib. Программный код этого метода можно увидеть в листинге 15.

Листинг 15 — Метод инициализации соединения с брокером сообщений

```
/**
     * Метод создания подключения с брокером сообщений
     * @return AMQPChannel
     */
    private static function getChannel():AMQPChannel
    {
        $host = 'localhost';
        $user = 'guest';
        $password = 'guest';
        $vhost = '/';
        $port = 5672;
        $connection = new AMQPStreamConnection($host,
$port, $user, $password, $vhost);
        $shutdown = function($connection, $channel)
        {
            if(!is_null($connection)){
                $connection->close();
            }
```

```
            if(!is_null($channel)){
                $channel->close();
            }
        };
        register_shutdown_function($shutdown,  $connection,
$connection->channel());
        return $connection->channel();
    }
```

Метод initRabbitConfig. Данный метод предназначен для воссоздания инфраструктуры в брокере очередей, а именно, создаётся обменник(exchange), очереди(queue) и очередям задаются ключи маршрутизации(binding). К одной очереди можно привязать несколько ключей маршрутизации. Данные для создания инфраструктуры берутся из классов, список которых возвращает метод getMappingRoutingKeyClass (листинг 14). Программный код метода initRabbitConfig можно увидеть в листинге 16.

Листинг 16 — Метод создания инфраструктуры в брокере сообщений

```
/**
     * Метод создания инфраструктуры в брокере сообщений
     */
    private static function initRabbitConfig()
    {
        $channel = self::getChannel();
        $exchange = self::getExchange();
        if(!empty($exchange))
        {
            $channel->exchange_declare($exchange,
AMQPExchangeType::DIRECT, false, true, false);
            /**
```

```
             * @var aMessageProcessing $CLASS
             */
            $arRk = self::getMappingRoutingKeyClass();
            foreach ($arRk as $CLASS)
            {
                $queue = $CLASS::getQueueName();
                $routingKey = $CLASS::getRoutingKey();
                if(!empty($queue) && !empty($routingKey))
                {
                    $channel->queue_declare($queue,  false,
true, false, false);
                    $channel->queue_bind($queue, $exchange,
$routingKey);
                }
            }
            $channel->getConnection()->close();
            $channel->close();
        }
    }
```

Метод processMessage. Данный метод отвечает за передачу сообщения соответствующему классу-обработчику сообщений. Метод требует два обязательных аргумента:

1. $routing_key — ключ маршрутизации, для сопоставления сообщения и класса, который должен обработать сообщение;

2. $message — непосредственно само полученное сообщение.

Внутри метода определяется необходимый класс-обработчик, которому и передаётся полученное сообщение. Программный код метода processMessage можно увидеть в листинге 17.

Листинг 17 — Метод обработки сообщения

```
/**
     * Метод для обработки сообщения
     * @param string $routing_key
     * @param string $message
     * @return array
     */
    private static function processMessage(string
$routing_key, string $message):array
    {
        $arResult = ['success' => false];
        /**
         * @var $obj aMessageProcessing
         */
        $className                                    =
self::getClassNameByRoutingKey($routing_key);
        if(!empty($className))
        {
            $obj = new $className($message);
            $obj->runWork();
            $arResult = ['success' => $obj->getResult()-
>isSuccess(),      'errors'      =>      $obj->getResult()-
>getErrorMessages()];
        }
        else
        {
            $arResult['errors'][]   =   'Некорректный   ключ
маршрутизации:' . $routing_key;
        }
        return $arResult;
```

```
    }
```

Метод runWorkConsumer. Данные метод необходим для запуска обработки сообщений из очереди. В метод необходимо передать один параметр — это идентификатор очереди в брокере сообщений. Программный код метода можно увидеть в листинге 18.

Листинг 18 — Метод прослушивания очереди сообщений

```
/**
     * Метод запускает Consumer для прослушивания очереди
$queue
     * @param string $queue - название очереди, которую
нужно прослушивать
     */
    public static function runWorkConsumer(string $queue)
    {
        self::initRabbitConfig();

        /** @var \PhpAmqpLib\Channel\AMQPChannel $channel
*/
        $channel = self::getChannel();
        $channel->basic_consume($queue, '', false, false,
false, false,

'\Igrik\Vkr\AMQP\Connector::process_message_callback');
        $timeout = 0;
        while ($channel->is_consuming()) {
            $channel->wait(null, false, $timeout);
        }
    }
```

Сам метод необходимо вызывать в скрипте, который должен непрерывно работать на веб-сервере и автоматически перезапускаться в случае его остановки.

Таким образом реализован класс Connector, который предназначен для создания соединения с брокером сообщений, а также реализует передачу сообщений в необходимый класс-обработчик.

Рассмотрим абстрактный класс aMessageProcessing. Этот класс должны наследовать все классы-обработчики сообщений. В нём реализована основная логика по обработке сообщений, что позволяет с лёгкостью масштабировать веб-библиотеку, добавляя новые классы-обработчики сообщений.

Классы-обработчики сообщений, которые наследую абстрактный класс aMessageProcessing обязательно должны описать следующие методы:

1. getRoutingKey — метод возвращающие ключ маршрутизации. Результат этого метода будет использован в классе Connector, для сопоставления классов-обработчиков и получаемых из брокера очередей сообщений;

2. getQueueName — метод возвращающий идентификатор очереди в брокере сообщений. Необходим для автоматического создания инфраструктуры в брокере сообщений с помощью метода initRabbitConfig класса Connector (листинг 18).

3. runProcessMessage — основной метод класса-обработчика сообщений, в котором должна быть реализована обработка полученного сообщения.

Работа с классами-обработчиками сообщений начинается с конструктора абстрактного класса aMessageProcessing, в который передаётся сообщение. А сам экземпляр класса-обработчика и обработка полученного сообщения инициируется в методе processMessage класса Connector (листинг 17).

Таким образом реализован абстрактный php-класс aMessageProcessing, который предназначен для создания классов-обработчиков сообщений, которые, в свою очередь, будут заниматься обработкой сообщений.

Полный программный код абстрактного класса aMessageProcessing можно увидеть в листинге 19.

Листинг 19 — Программный код абстрактного класса aMessageProcessing

```
  <?php

namespace Igrik\Vkr\AMQP;

use Igrik\Vkr\Result;

abstract class aMessageProcessing
{

    const ERROR_CODE_EMPTY_BODY = "EMPTY_BODY";
    const ERROR_CODE_NOT_ACCEPT_VALUE = "NOT_ACCEPT_VALUE";
    const       ERROR_CODE_NOT_ISSET_REQUIRE_FIELD        =
"NOT_ISSET_REQUIRE_FIELD";
    const          ERROR_CODE_ELEMENT_NOT_FOUND          =
"ELEMENT_NOT_FOUND";
    const ERROR_CODE_OTHER_ERROR = "OTHER_ERROR";
    const ERROR_CODE_BITRIX = "BTRX_ELEMENT_NOT_FOUND";
    const         ERROR_CODE_EMPTY_TITLE_STORAGE         =
"EMPTY_TITLE_STORAGE";
    const          ERROR_CODE_EMPTY_EXTERNAL_ID          =
"EMPTY_EXTERNAL_ID";
    const ERROR_CODE = [
        self::ERROR_CODE_EMPTY_BODY => [
            "MESSAGE" => "Сообщение пустое"
        ],
        self::ERROR_CODE_NOT_ISSET_REQUIRE_FIELD => [
```

```
            "MESSAGE"    =>    "Не    передан    обязательный
параметр:"
        ],
        self::ERROR_CODE_NOT_ACCEPT_VALUE => [
            "MESSAGE" => "Передано недопустимое значение:"
        ],
        self::ERROR_CODE_ELEMENT_NOT_FOUND => [
            "MESSAGE" => "Элемент не найден:"
        ],
        self::ERROR_CODE_OTHER_ERROR => [
            "MESSAGE" => ""
        ],
        self::ERROR_CODE_BITRIX => [
            "MESSAGE" => "Ошибка Битрикса:"
        ],
        self::ERROR_CODE_EMPTY_TITLE_STORAGE => [
            "MESSAGE" => "Пустое поле «Название склада»"
        ],
        self::ERROR_CODE_EMPTY_EXTERNAL_ID => [
            "MESSAGE" => "Пустое поле «Внешний код»"
        ]
    ];

    protected Result $result;
    protected array $message;

    final function __construct(string $message)
    {
        $this->setMessageData($message);
        $this->initResult();
```

```
    }

    private final function setMessageData(string $message)
    {
        $this->message    =    (array)json_decode($message,
JSON_UNESCAPED_UNICODE);
    }

    protected final function getMessageData(): array
    {
        return $this->message;
    }

    private final function initResult()
    {

        $this->result = new Result();
        return $this->result;
    }

    public final function getResult(): Result
    {
        return $this->result;
    }

    protected  final  function  addError(string  $errorCode,
string $addMessage = '')
    {
        $messageError                =                $this-
>getMessageErrorByMessageCode($errorCode) . $addMessage;
```

```
        $this->result->addError($messageError, $errorCode);
    }

    protected final function
getMessageErrorByMessageCode(string $errorCode = ''):
string
    {
        if (isset(self::ERROR_CODE[$errorCode])) {
            return self::ERROR_CODE[$errorCode]['MESSAGE'];
        }

        return 'Неизвестная ошибка';
    }

    protected final function checkRequireInArray(array
$array, array $requireFields, string $addText = '')
    {
        foreach ($requireFields as $field) {
            $fieldCode = $field['FIELD_CODE'];
            if (!isset($array[$fieldCode])) {
                $this-
>addError(self::ERROR_CODE_NOT_ISSET_REQUIRE_FIELD,
$fieldCode . "::" . $addText);
            } elseif (!empty($field['ACCEPT_VALUES'])) {
                if (!in_array($array[$fieldCode],
$field['ACCEPT_VALUES'])) {
                    $this-
>addError(self::ERROR_CODE_NOT_ACCEPT_VALUE, $fieldCode .
"=" . $array[$fieldCode] . "::" . $addText);
                }
```

```
            }
        }
    }

    protected function editMessage(){}

    public final function runWork(): bool
    {
        $this->runProcessMessage();
        return $this->getResult()->isSuccess();
    }

    final protected function
getRequireFieldsByConfig(string $configCode)
    {
        return $this-
>getRequireFieldsConfigs()[$configCode];
    }

    protected abstract function checkRequireFields(): bool;

    protected abstract function getRequireFieldsConfigs():
array;

    protected abstract function runProcessMessage();

    abstract static function getRoutingKey():string;

    abstract static function getQueueName():string;
}
```

3.5. Результат использования разработанных веб-библиотек

Разработанные веб-библиотеки были использованы в реальном проекте, который является интернет-магазином. Характеристики интернет-магазина:

1. Более 40 000 товаров;

2. Более 10 000 активных пользователей в сутки;

3. Содержит информацию о наличии и ценах товаров в 63 розничных магазинах по всех России;

4. Есть интеграция с системой учёта 1С.

Также, интернет-магазин имеет интеграцию с системой учёта 1С, из которой приходит информация о товарах, ценах, остатках по всем розничным магазинам. Данные из 1С передавались по HTTP протоколу в XML-формате, что отрицательным образом сказывалось на скорости обновления данных на сайте, ведь очередная порция данных могла быть отправлена из 1С только после того, как был получен ответ от сайта на обработку предыдущей порции данных. Для оптимизации процесса передачи данных из системы учёта 1С на сайт была использована разработанная веб-библиотека для синхронного выполнения операция с базой данных, что позволило использовать брокер сообщений в процессе получения данных из 1С сайтом.

После применения веб-библиотеки для кеширования программных алгоритмов, скорость загрузки страниц интернет-магазина уменьшилась с 2-3 секунд, до десятых долей секунды. А количество запросов к базе данных, при посещении страниц пользователями, было уменьшено с 2-3 тысяч, до нескольких сотен, что положительным образом сказалось на нагрузке на базу данных и на веб-сервер в целом.

Применение разработанных веб-библиотек позволило достичь следующих результатов:

1. Среднее количество запросов к базе данных, при посещении страниц сайта пользователями, удалось уменьшить с 2800 запросов, до 250-300 запросов, что существенно уменьшило общую нагрузка как на базу данных, так и на веб-сервер в целом;

2. Среднее время формирования веб-страниц сайта сервером удалось уменьшить с 3 секунд, до 0.4 секунд, что существенно ускорило время загрузки веб-страниц у пользователей и положительно сказалось на пользовательском опыте использовании интернет-магазина;

3. Средняя нагрузка на процессор веб-сервер уменьшилась на 30%;

4. Средний показатель потребляемой оперативной памяти удалось уменьшить на 10% и составил 34ГБ, вместо 38ГБ.

Данные о нагрузке на веб-сервер до использования веб-библиотек (рис. 8) и после (рис. 9), удалось получить благодаря системе мониторинга веб-сервера Zabbix, которая позволила получить графики нагрузки на веб-сервер.

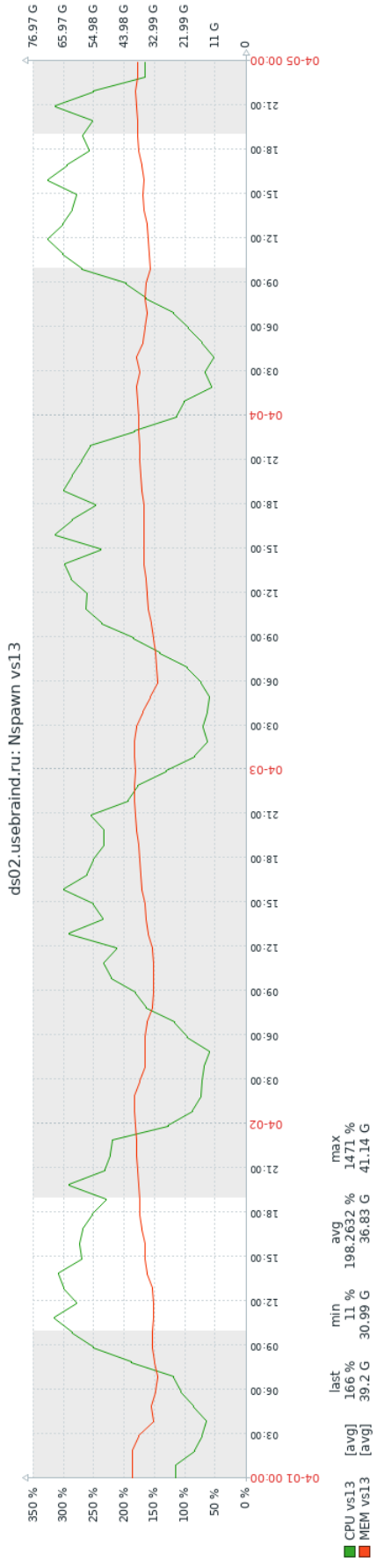

Рисунок 8 — График нагрузки на веб-сервер до использования веб-библиотек

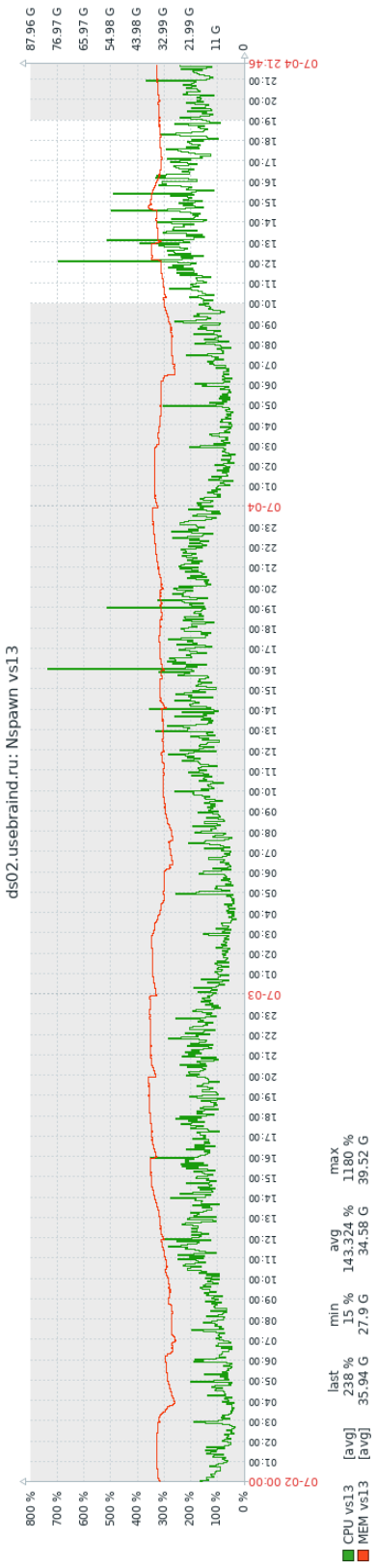

Рисунок 9 — График нагрузки на веб-сервер после использования веб-библиотек

ЗАКЛЮЧЕНИЕ

В данной работе рассматривалась проблема оптимизации взаимодействия веб-сайта с базой данных. Были изучены CRUD-операции взаимодействия с базой данных. Проанализированы методы оптимизации взаимодействия с базой данных средствами кешировани, а также синхронного выполнения операция CRUD-операций с базой данных. Результатом работы стала разработка веб-библиотек, реализующих метода оптимизации взаимодействия веб-сайта с базой данных. На языке PHP написаны:

- веб-библиотека для кеширования работы программных алгоритмов;
- веб-библиотека для синхронного выполнения CRUD-операций с базой данных.

Разработанные веб-библиотеки были использованы на реальном проекте, являющимся интернет-магазином, и позволили существенно уменьшить нагрузку на веб-сервер при использовании веб-сайта пользователями, а именно:

1. Удалось уменьшить среднее количество запросов к базе данных, при посещении страниц сайта пользователями с 2800 запросов, до 250-300 запросов, что существенно уменьшило общую нагрузка как на базу данных, так и на веб-сервер в целом;
2. Удалось уменьшить среднее время формирования веб-страниц сайта сервером с 3 секунд, до 0.4 секунд, что существенно ускорило время загрузки веб-страниц у пользователей и положительно сказалось на пользовательском опыте использовании интернет-магазина;
3. Удалось уменьшить среднюю нагрузку на процессор веб-сервера на 30%;
4. Удалось уменьшить средний показатель потребляемой оперативной на 10% и составил 34ГБ, вместо 38ГБ.

Программный код разработанных веб-библиотек был выложен в публичный доступ в репозиторий, который находится по ссылке https://github.com/IgorTyutyunov/vkr.

В будущем планируется развивать разработанные веб-библиотеки, а именно:

1. Разработать веб-интерфейс для более удобного управления очередями брокером сообщений и создания классов для обработки сообщений.
2. Разработать собственную ORM систему для взаимодействия с базой данных, которая позволит упростить написание SQL-запросов, а также позволит оптимизировать их перед непосредственным выполнением;
3. Разработать инструмент для анализа веб-сайта на предмет неэффективных запросов к базе данных, а также на предмет неоптимальных программных алгоритмов;
4. Разработать инструмент для анализа скорости загрузки веб-страниц и количества запросов к базе данных. Этот инструмент поможет разработчикам в оптимизации программных алгоритмов;

СПИСОК ЛИТЕРАТУРЫ

1. HTTP Methods [Электронный ресурс]. – Режим доступа: https://restfulapi.net/http-methods/. – Дата доступа:25.01.2022.

2. RabbitMQ tutorial - "Hello World!" — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-one-php.html. - Дата доступа:25.01.2022.

3. RabbitMQ tutorial - Publish/Subscribe — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-three-php.html. - Дата доступа:25.01.2022.

4. RabbitMQ tutorial - Reliable Publishing with Publisher Confirms — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-seven-php.html. - Дата доступа:25.01.2022.

5. RabbitMQ tutorial - Remote procedure call (RPC) — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-six-php.html. - Дата доступа:25.01.2022.

6. RabbitMQ tutorial - Routing — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-four-php.html. - Дата доступа:25.01.2022.

7. RabbitMQ tutorial - Topics — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-five-php.html. - Дата доступа:25.01.2022.

8. RabbitMQ tutorial - Work Queues — RabbitMQ: [Электронный ресурс]. - Режим доступа: https://www.rabbitmq.com/tutorials/tutorial-two-php.html. - Дата доступа:25.01.2022.

9. RabbitMQ. Часть 1. Introduction. Erlang, AMQP [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/488654/– Дата доступа: 25.01.2022.

10. Redis [Электронный ресурс]. – Режим доступа: https://evilinside.ru/redis/– Дата доступа: 25.01.2022.

11. Strong typing for event-driven microservice architecture / Н.С. Герасимов, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Компьютерные инструменты в образовании. – 2019. – № 1. – С. 43-53.

12. Анализ производительности redis в mysql для веб-кэширования / А.К. Зарипов, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Вестник магистратуры. – 2021. – № 6. – С. 52-53.

13. Архитектурные приемы при разработке программного обеспечения, зависимого от интерфейса пользователя / О.В. Игнатьева, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Инженерный вестник Дона. – 2022. – № 2. – С. 10-19.

14. Асинхронное взаимодействие. Брокеры сообщений [Электронный ресурс]. – Режим доступа: https://itnan.ru/post.php?c=1&p=534858. – Дата доступа: 25.01.2022.

15. Басов А.С. Сравнение современных СУБД / А.С. Басов // Вестник науки. – 2020. – № 7. – С. 50-54.

16. Бумажная промышленность: kafka против rabbitmq. сравнительное исследование двух отраслевых эталонных реализаций publish/subscribe / К.С. Москвичева, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Форум молодёжной науки. – 2020. – № 4. – С. 3-17.

17. Васильева К.Н. Реляционные базы данных / К.Н. Васильева, К.Н. Васильева // Colloquium-journal. – 2020. – № 2. – С. 22-23.

18. Глотов И.Н. Защищённая СУБД с сохранением порядка / И.Н. Глотов, С.В. Овсянников, В.Н. Тренькаев // Прикладная дискретная математика. – 2014. – № 7. – С. 81-82.

19. Использование СУБД Redis в качестве промежуточного хранилища данных для PostgreSQL / О.И. Рубин, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // StudNet. – 2020. – № 9. – С. 1646-1650.

20. Исследование производительности субд при работе с кластерными базами данных на основе эргономического анализа / Е.А. Елисеева, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // StudNet. – 2022. – № 4. – С. 2888-2909.

21. Ковега Д.Н. Распределенная отказоустойчивая СУБД / Д.Н. Ковега, В. А. Крищенко // Машиностроение и компьютерные технологии. – 2012. – № 3. – С. 1-7.

22. Мейер М. Теория реляционных баз данных / М. Мейер. – Москва: Мир, 1987. – 608 с.

23. Методы повышения производительности современных веб-приложений / В.Н. Гридин, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Известия Южного федерального университета. Технические науки. – 2020. – № 2. – С. 193-200.

24. Модель затрат для оптимизации аналитических запросов в гетерогенных системах / П.А. Курапов, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // International Journal of Open Information Technologies. – 2022. – № 4. – С. 61-70.

25. Обобщенная схема скрытого компактного хранения данных различных пользователей в общей открытой базе / В.А. Романьков, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Известия Иркутского государственного университета. Серия: Математика. – 2022. – № 40. – С. 63-77.

26. Особенности добавления механизма рабочего процесса в приложениях на базе микросервисной архитектуры, управляемой событиями / Р. Дандан, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Международный журнал гуманитарных и естественных наук. – 2022. – № 2. – С. 26-31.

27. Парсинг телеграм-каналов как элемент системы автоматизированного анализа информации, полученной из сети интернет / И.И. Карабак, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Прикаспийский журнал: управление и высокие технологии. – 2022. – № 1. – С. 9-17.

28. Парсинг телеграм-каналов как элемент системы автоматизированного анализа информации, полученной из сети интернет / И.И. Карабак, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Прикаспийский журнал: управление и высокие технологии. – 2022. – № 9. – С. 9-17.

29. Подсистема распределенного решения оптимизационных задач / Д.В. Заруба, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Известия Южного федерального университета. Технические науки. – 2019. – № 19. – С. 1-12.

30. Применение асинхронного обмена информацией в веб-приложениях / А.В. Скрыпников, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Международный журнал гуманитарных и естественных наук. – 2021. – № 12. – С. 105-108.

31. Работа с rabbitmq php [Электронный ресурс]. – Режим доступа: https://biznessrussia.ru/dom-i-dacha/rabota-s-rabbitmq-php.html– Дата доступа: 25.01.2022.

32. Разработка программно-аппаратного комплекса сбора и хранения данных термометрии / Ш.А. Оцоков, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Инженерный вестник Дона. – 2022. – № 2. – С. 1-9.

33. Разработка распределенной системы обмена уведомлениями на основе микросервисной архитектуры / К.Н. Цебренко, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Международный журнал гуманитарных и естественных наук. – 2021. – № 1. – С. 119-122.

34. Реализация протокола обмена данными между программными агентами в облачной инфраструктуре в географически распределенных центрах обработки данных / Н.Ю. Самохин, Э.А. Челышев, Д.В. Шибитов, М.В. Раскатова // Научно-технический вестник информационных технологий, механики и оптики. – 2019. – № 6. – С. 1086-1092.

35. Система мониторинга Zabbix / В.А. Чистяков, Б.Е. Мыктыбаев, А.Б. Жанбеков, В.С. Котяшев // Science Time. – 2015. – № 1. – С. 836-839.

36. Сравнение протоколов передачи данных в интернете вещей / Т.И. Курмаев, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Международный научно-исследовательский журнал. – 2022. – № 1. – С. 45-47.

37. Тенденции развития, риски и перспективы баз больших данных / В.Л. Плескач, В.Н. Краснощок, Я.В. Криволапов, Л.Н. Скачек // Colloquium-journal. – 2022. – № 1. – С. 39-42.

38. Что такое PHP? [Электронный ресурс]. – Режим доступа: https://www.php.net/manual/ru/intro-whatis.php. – Дата доступа: 25.01.2022.