

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Формализация задачи	4
1.2 Формализация данных	4
1.3 Типы пользователей	5
1.4 Базы данных и системы управления базами данных	6
1.5 Хранение данных о рабочих программах дисциплины	7
1.5.1 Классификация баз данных по способу хранения	7
1.5.2 Выбор модели хранения данных для решения задачи	8
1.5.3 Обзор СУБД с построчным хранением	9
1.5.4 Выбор СУБД для решения задачи	10
1.6 Кэширование данных	10
1.6.1 Проблемы кэширования данных	11
1.6.2 Обзор in-memory NoSQL СУБД	11
1.6.3 Выбор СУБД для решения задачи	13
1.7 Формализация данных	13
1.7.1 База данных рабочих программ дисциплин	13
1.7.2 База данных кэшируемой информации	14
2 Конструкторская часть	15
2.1 Проектирование отношений сущностей	15
2.2 Проектирование базы данных рабочих программ дисциплин	15
2.3 Проектирование базы данных кэширования	20
3 Технологическая часть	21
3.1 Архитектура приложения	21
3.2 Средства реализации	21
3.3 Детали реализации	22
3.4 Взаимодействие с приложением	34
4 Исследовательская часть	37
4.1 Постановка эксперимента	37

4.1.1	Цель эксперимента	37
4.1.2	Описание эксперимента	37
4.1.3	Результат эксперимента	38
	Заключение	42
	Литература	43

Введение

Рабочая программа дисциплины – программа освоения учебного материала, соответствующая требованиям государственного образовательного стандарта высшего профессионального образования и учитывающая специфику подготовки студентов по избранному направлению или специальности. Разрабатывается для каждой дисциплины учебного плана всех реализуемых в университете основных образовательных программ [1].

Хранение, обработка и анализ информации, находящейся в рабочей программы дисциплины может пригодиться для различных систем, например, системы управления обучения (англ. Learning Management System (LMS) [2]). Такой интерфейс может предоставить пользователю системы (в данном случае преподавателю) получать и редактировать информацию о рабочей программы дисциплины в режиме онлайн, например, в личном кабинете пользователя.

Рабочая программа дисциплины обычно представлена в виде документа в формате Microsoft Word [3], что накладывает ограничения на автоматизированную программную обработку и анализ информации, предоставленной в рабочей программы дисциплины.

Цель работы – реализовать программное обеспечение для хранения, редактирования и удаления данных о рабочих программах дисциплин.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- проанализировать варианты представления данных и выбрать подходящий вариант для решения задачи;
- проанализировать системы управления базами данных и выбрать подходящую систему для хранения данных;
- спроектировать базу данных, описать ее сущности и связи;
- реализовать интерфейс для доступа к базе данных;
- реализовать программное обеспечение, которое позволит получить доступ к данным по средствам REST API [4].

1 Аналитическая часть

В данном разделе описана структура рабочей программы дисциплины. Представлен анализ способов хранения данных и систем управления базами данных, оптимальных для решения поставленной задачи. Описаны проблемы кэшированных данных и представлены методы их решения.

1.1 Формализация задачи

Необходимо спроектировать и реализовать базу данных для онлайн-мониторинга состояния трасс и подъемников горнолыжного курорта. Также необходимо разработать интерфейс, позволяющий работать с данной базой для получения и изменения хранящейся в ней информации и мониторинга очередей к подъемникам в онлайн-режиме. Реализовать, как минимум, три вида ролей – пользователь, сотрудник лыжного патруля и администратор.

1.2 Формализация данных

База данных должна хранить информацию о:

- трассах;
- подъемниках;
- связях трасс и подъемников (на одном подъемнике можно добраться до нескольких трасс, и до одной трассы можно добраться на нескольких подъемниках);
- турникетах;
- проездных картах;
- считываниях карт на турникетах подъемников;
- сообщениях о происшествиях;

- пользователях;
- группах пользователей.

В таблице 1.1 приведены категории и сведения о данных.

Таблица 1.1: Категории и сведения о данных

Категория	Сведения
Трассы	ID трассы, название трассы, уровень сложности, открытость/закрытость.
Подъемники	ID подъемника, название подъемника, открытость/закрытость, количество мест, время подъема, время в очереди.
Связи трасс и подъемников	ID записи, ID подъемника, ID трассы.
Турникеты	ID турникета, ID подъемника, открытость/закрытость.
Проездные карты	ID карты, дата и время активации, тип.
Считывания карт на турникетах подъемников	ID записи, ID турникета, ID карты, дата и время считывания.
Сообщения о происшествиях	ID сообщения, ID отправителя, ID прочитавшего, текст сообщения.
Пользователи	ID пользователя, ID карты, email (логин), пароль, ID группы пользователей.
Группы пользователей	ID группы пользователей, права доступа.

1.3 Типы пользователей

В соответствии с поставленной задачей необходимо разработать приложение с возможностью аутентификации пользователей, что делит их, прежде всего, на авторизованных и неавторизованных. для управления приложением необходима ролевая модель: авторизованный (обычный) пользователь, сотрудник лыжного патруля и администратор.

Для каждого типа пользователя предусмотрен свой набор функций:

- неавторизованный пользователь:
 - регистрация,
 - аутентификация,

- просмотр информации о состоянии трасс и подъемников,
- просмотр информации о связях трасс и подъемников;
- авторизованный пользователь:
 - выход,
 - просмотр информации о состоянии трасс и подъемников,
 - просмотр информации о связях трасс и подъемников,
 - отправка сообщений о происшествиях;
- сотрудник лыжного патруля:
 - выход,
 - просмотр и изменение информации о состоянии трасс и подъемников,
 - просмотр и изменение информации о связях трасс и подъемников,
 - просмотр сообщений о происшествиях;
- администратор:
 - выход,
 - просмотр и изменение всей информации, доступной в базе данных, в том числе права доступа групп и отдельных пользователей.

1.4 Базы данных и системы управления базами данных

В задаче разбора и хранения информации рабочей программы дисциплины важную роль имеет выбор модели хранения данных. Для персистентного хранения данных используются базы данных [7]. Для управления этими базами данных используется системы управления данных – СУБД [8]. Система управления базами данных – это совокупность программных

и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

1.5 Хранение данных о рабочих программах дисциплины

Система, разрабатываемая в рамках курсового проекта, предполагает собой приложение, которое является микросервисом [9] одной большой системы – системы управления обучения.

Предполагается, что доступ к разрабатываемому приложению будем иметь лишь только «ядро» этой системы. При этом, только у одного типа пользователя системы есть доступ к данным, хранящимся в приложении – преподавателю. Состояние гонки (англ. Race condition [10]) можно исключить - каждый преподаватель работает только с информацией из файлов, которые он самостоятельно загрузил в базу данных.

Для хранения данных о рабочей программы дисциплины необходимо использовать строго структурированную и типизированную базу данных, потому что вся информация, предоставленная в файлах программы имеет чётко выраженную структуру, которая не будет меняться от дисциплины к дисциплине.

1.5.1 Классификация баз данных по способу хранения

Базы данных, по способу хранения, делятся на две группы – строковые и колоночные. Каждый из этих типов служит для выполнения для определенного рода задач.

Строковые базы данных

Строковыми базами данных называются такие базы данных, записи которых в памяти представляются построчно. Строковые баз данных используются в транзакционных системах (англ. OLTP [11]). Для таких систем

характерно большое количество коротких транзакций с операциями вставки, обновления и удаления данных - INSERT, UPDATE, DELETE.

Основной упор в системах OLTP делается на очень быструю обработку запросов, поддержание целостности данных в средах с множественным доступом и эффективность, которая измеряется количеством транзакций в секунду.

Схемой, используемой для хранения транзакционных баз данных, является модель сущностей, которая включает в себя запросы, обращающиеся к отдельным записям. Так же, в OLTP-системах есть подробные и текущие данные.

Колоночные базы данных

Колоночными базами данных называются базы данных, записи которых в памяти представляются по столбцам. Колоночные базы данных используются в аналитических системах (англ. OLAP [12]). OLAP характеризуется низким объемом транзакций, а запросы часто сложны и включают в себя агрегацию. Время отклика для таких систем является мерой эффективности.

OLAP-системы широко используются методами интеллектуального анализа данных. В таких базах есть агрегированные, исторические данные, хранящиеся в многомерных схемах.

1.5.2 Выбор модели хранения данных для решения задачи

Для решения задачи строчное хранение данных преобладает над колоночным хранением по нескольким причинам:

- задача предполагает постоянное добавление и изменение данных;
- задача предполагает быструю отзывчивость на запросы пользователя;
- задача не предполагает выполнения аналитических запросов;

1.5.3 Обзор СУБД с построчным хранением

В данном подразделе буду рассмотрены популярные построчные СУБД, которые могут быть использованы для реализации хранения в разрабатываемом программном продукте.

PostgreSQL

PostgreSQL [13] – это свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных [14].

PostgreSQL предоставляет транзакции со свойствами атомарности, согласованности, изоляции, долговечности (ACID [15]), автоматически обновляемые представления, материализованные представления, триггеры, внешние ключи и хранимые процедуры. Данная СУБД предназначена для обработки ряда рабочих нагрузок, от отдельных компьютеров до хранилищ данных или веб-сервисов с множеством одновременных пользователей.

Рассматриваемая СУБД управляет параллелизмом с помощью технологии управления многоверсионным параллелизмом (англ. MVCC [16]). Эта технология дает каждой транзакции «снимок» текущего состояния базы данных, позволяя вносить изменения, не затрагивая другие транзакции. Это в значительной степени устраняет необходимость в блокировках чтения (англ. read lock [17]) и гарантирует, что база данных поддерживает принципы ACID.

Oracle Database

Oracle Database [18] – объектно-реляционная система управления базами данных компании Oracle [19]. На данный момент, рассматриваемая СУБД является самой популярной в мире. [20]

Все транзакции Oracle Database соответствуют обладают свойствами ACID, поддерживает триггеры, внешние ключи и хранимые процедуры. Данная СУБД подходит для разнообразных рабочих нагрузок и может ис-

пользоваться практически в любых задачах. Особенностью Oracle Database является быстрая работа с большими массивами данных.

Oracle Database может использовать один или более методов параллелизма. Сюда входят механизмы блокировки для гарантии монопольного использования таблицы одной транзакцией, методы временных меток, которые разрешают сериализацию транзакций и планирование транзакций на основе проверки достоверности.

MySQL

MySQL [21] – свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle.

Рассматриваемая СУБД имеет два основных движка хранения данных: InnoDB [22] и myISAM [23]. Движок InnoDB полностью полностью совместим с принципами ACID, в отличие от движка myISAM. СУБД MySQL подходит для использования при разработке веб-приложений, что объясняется очень тесной интеграцией с популярными языками PHP [24] и Perl [25].

Реализация параллелизма в СУБД MySQL реализовано с помощью механизма блокировок, который обеспечивает одновременный доступ к данным.

1.5.4 Выбор СУБД для решения задачи

Для решения задачи была выбрана СУБД PostgreSQL, потому что данная СУБД имеет поддержку языка `python3u` [26], который упрощает процесс интеграции базы данных в разрабатываемое приложение. Кроме того, PostgreSQL проста в развертывании.

1.6 Кэширование данных

Для ускорения быстродействия разрабатываемого приложения, можно прибегнуть к кэшированию данных. Для кэширования данных можно ис-

пользовать NoSQL [27] in-memory базы данных. Такие базы данных хранят данные в оперативной памяти, что обеспечивает более быстрый доступ к данным.

1.6.1 Проблемы кэширования данных

Синхронизация данных

Приложение пишет в кэш, и в базу данных, которые между собой никак не синхронизируются. Таким образом возникает несогласованность данных. Например, в случае разрабатываемого приложения, возможна ситуация, когда данные удаляются из хранилища и их нужно удалить из кэша. Эту проблему можно решить установкой триггеров в базе данных хранения рабочих программ дисциплин, которые будут срабатывать на изменение / удаление данных и синхронизировать актуальные данные в кэше.

Проблема «холодного старта»

Когда кэш только развертывается, он пуст и в нем нет никаких данных. Все запросы идут напрямую в базу данных, и только спустя какое-то время кэш будет «разогрет» и будет работать в полную силу. Эту проблему можно решить, выбрав СУБД с журналированием всех операций: при перезагрузке можно восстановить предыдущее состояние кэша с помощью журнала событий, который хранится на диске. При этом, при перезапуске кэша, нужно синхронизировать данные с хранилищем: возможно, какие-то данные находящиеся в кэше перестали быть актуальными за время его перезагрузки.

1.6.2 Обзор in-memory NoSQL СУБД

Tarantool

Tarantool [28] – это платформа in-memory вычислений с гибкой схемой

хранения данных для эффективного создания высоконагруженных приложений. Включает себя базу данных и сервер приложений на языке программирования Lua [29].

Tarantool обладает высокой скоростью работы по сравнению с традиционными СУБД. При этом, в рассматриваемой платформе для транзакций реализованы свойства ACID, репликация master-slave [30] и master-master [31], как и в традиционных СУБД.

Для хранения данных используется кортежи (англ. tuple) данных. Кортеж – это массив не типизированных данных. Кортежи объединяются в спейсы (англ. space), аналоги таблицы из реляционной модели хранения данных. Спейс – коллекция кортежей, кортеж – коллекция полей.

В рассматриваемой СУБД реализованы два движка хранения данных: memtx [32] и vinyl [32]. Первый хранит все данные в оперативной памяти, а второй на диске. Для каждого спейса можно задавать различный движок хранения данных.

Каждый спейс должен быть проиндексирован первичным ключом. Кроме того, поддерживается неограниченное количество вторичных ключей. Каждый из ключей может быть составным.

В Tarantool реализован механизм «снимков» текущего состояния хранилища и журналирования всех операций, что позволяет восстановить состояние базы данных после ее перезагрузки.

Redis

Redis [33] – резидентная система управления базами данных класса NoSQL с открытым исходным кодом. Основной структурой данных, с которой работает Redis является структура типа «ключ-значение». Данная СУБД используется как для хранения данных, так и для реализации кэшей и брокеров сообщений.

Redis хранит данные в оперативной памяти и снабжена механизмом «снимков» и журналирования, что обеспечивает постоянное хранение данных. Предоставляются операции для реализации механизма обмена сообщениями в шаблоне «издатель-подписчик»: с его помощью приложения могут создавать программные каналы, подписываться на них и помещать

в эти каналы сообщения, которые будут получены всеми подписчиками. Существует поддержка репликации данных типа master-slave, транзакций и пакетной обработки команд.

Все данные Redis хранит в виде словаря, в котором ключи связаны со своими значениями. Ключевое отличие Redis от других хранилищ данных заключается в том, что значения этих ключей не ограничиваются строками. Поддерживаются следующие абстрактные типы данных:

- строки;
- списки;
- множества;
- хеш-таблицы;
- упорядоченные множества.

Тип данных значения определяет, какие операции доступны для него; поддерживаются высокоуровневые операции: например, объединение, разность или сортировка наборов.

1.6.3 Выбор СУБД для решения задачи

Для кэширования данных была выбрана СУБД Tarantool, так как она проста в развертывании и переносимости, и имеет подходящие коннекторы для базы данных PostgreSQL.

1.7 Формализация данных

1.7.1 База данных рабочих программ дисциплин

База данных рабочих программ дисциплин должна хранить непосредственно информацию о дисциплинах. Информация, которая должна храниться в базе данных описана в разделе ???. Каждая рабочая дисциплина

должна обладать уникальным идентификатором, чтобы её можно было однозначно идентифицировать.

1.7.2 База данных кэшируемой информации

База данных кэшируемых значений должна хранить значения без дополнительной обработки. Данные должны быть актуальны и синхронизированы с основным хранилищем: кэш должен обновляться после каждой транзакции. Кроме того, нужно ограничить размер кэша и добавить вытеснение из него, например, с помощью политики вытеснения LRU [34] (Last Recently Used).

Вывод

В данном разделе:

- рассмотрена структура рабочей программы дисциплины и выявлены её наиболее интересные части;
- проанализированы способы хранения информации для система и выбраны оптимальные способы для решения поставленной задачи;
- проведен анализ СУБД, используемых для решения задачи и также выбраны оптимальные информационные системы;
- рассмотрена проблема актуальности кэшируемых данных и предложено ее решение;
- формализованны данные, используемые в системе.

2 Конструкторская часть

В данном разделе представлены этапы проектирования выделенных в предыдущем разделе баз данных, нужных для решения задачи.

2.1 Проектирование отношений сущностей

На рисунке 2.1 представлена схема сущностей, необходимых для реализации приложения.

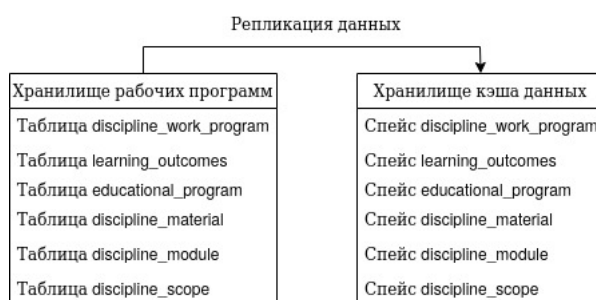


Рис. 2.1: Схема сущностей приложения

2.2 Проектирование базы данных рабочих программ дисциплин

База данных рабочих программ дисциплин будет реализована с использованием СУБД PostgreSQL. В базе данных будет существовать 6 сущностей и 7 таблиц, одна из которых является развязочной. ER-диаграмма сущностей этой базы данных представлена на рисунке 2.2.

Поля таблицы `discipline_work_program` означают:

- `id` – уникальный идентификатор рабочей программы дисциплин; будет использоваться чтобы однозначно идентифицировать рабочую программу в системе;
- `name` – название дисциплины;

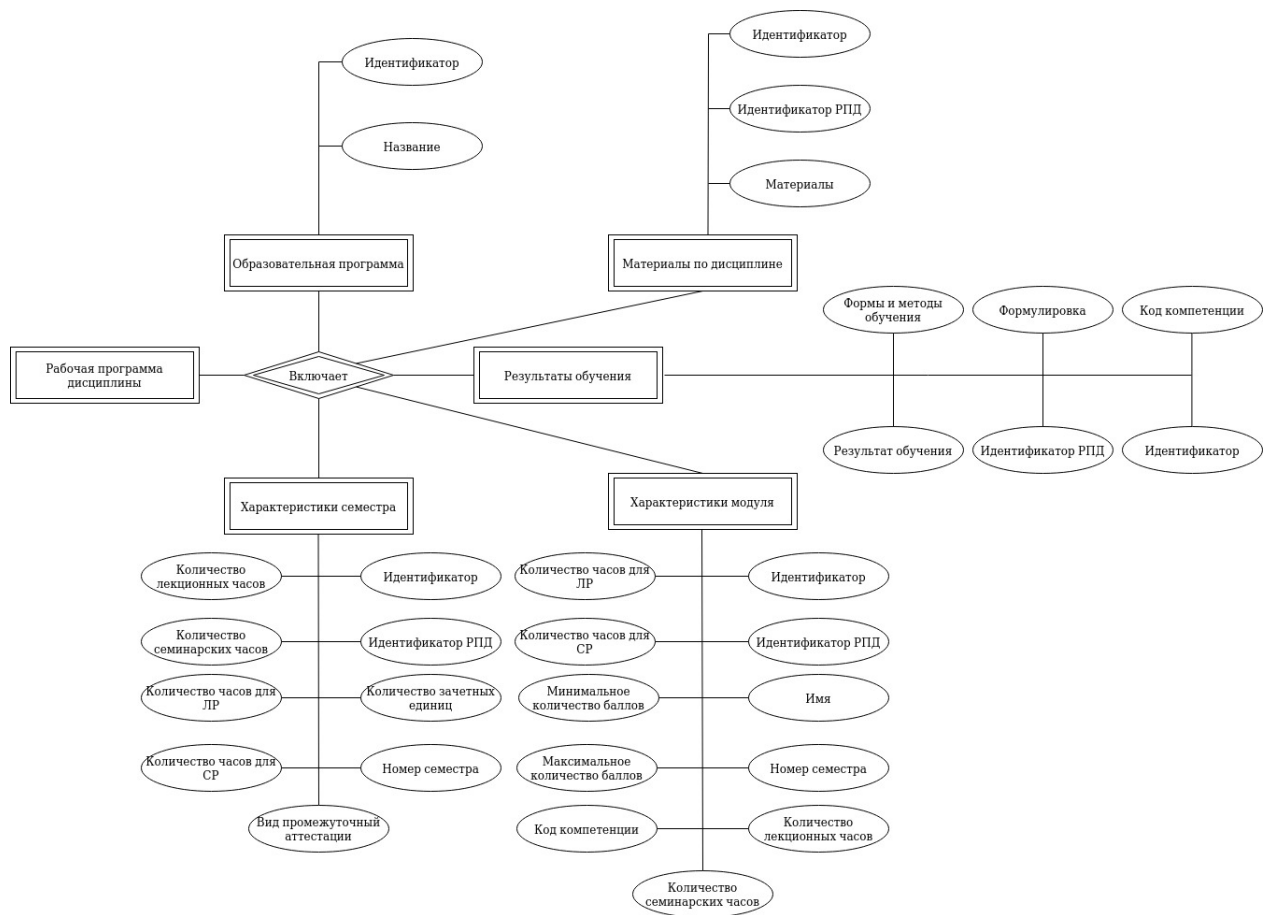


Рис. 2.2: ER-диаграмма сущностей базы данных рабочих программ дисциплин в нотации Чена

- **author** – ФИО автора рабочей программы дисциплины;
- **competency** – компетенция рабочей программы.

Данная таблица является ключевой и имеет связи с другими таблицами с отношением один-ко-многим и многие-ко-многим. Такая схема хранения хранить получить всю информацию о дисциплине, зная лишь ее уникальный идентификатор. При этом, такая схема хранения является достаточно гибкой.

Таблица **educational_program** хранит информацию о образовательной программе:

- **id** – уникальный идентификатор образовательной программы;
- **name** – имя образовательной программы.

Данная таблица и таблица `discipline_work_program` имеет отношение многие-ко-многим. Например, дисциплина «Объектно ориентированное программирование» преподается на образовательной программе «Программная инженерия» и «Информационная аналитика и политические технологии».

Таблица `learning_outcomes` содержит информацию о результатах обучения по данной дисциплине:

- `id` – уникальный идентификатор таблицы;
- `discipline_id` – внешний ключ для таблицы `discipline_work_program`;
- `competency_code` – код компетенции;
- `formulation` – формулировка компетенции;
- `results` – результаты обучения;
- `forms_and_methods` – формы и методы обучения.

Эта таблица имеет связь с таблицей `discipline_work_program` с отношением один-ко-многим. У дисциплины для каждого направления подготовки должны быть различные результаты обучения.

Таблица `discipline_scope` содержит информацию о объеме дисциплины для каждого семестра:

- `id` – уникальный идентификатор таблицы;
- `discipline_id` – внешний ключ для таблицы `discipline_work_program`;
- `semester_number` – номер семестра;
- `credit_units` – количество зачетных единиц;
- `total_hours` – общее количество часов;
- `lectures_hours` – количество часов, выделенных для проведения лекций;

- `seminars_hours` – количество часов, выделенных на семинарские занятия;
- `laboratory_work_hours` – количество часов, выделенное на лабораторные работы;
- `independent_work_hours` – количество часов, выделенное на самостоятельную работу студентом;
- `certification_type` – вид промежуточной аттестации – экзамен или зачет.

Эта таблица имеет связь с таблицей `discipline_work_program` с отношением один-ко-многим. Дисциплина может преподаваться несколько семестров.

Таблица `discipline_module` содержит информацию о содержании дисциплины для каждого модуля учебной дисциплины:

- `id` – уникальный идентификатор таблицы;
- `discipline_id` – внешний ключ для таблицы `discipline_work_program`;
- `semester_number` – номер семестра;
- `name` – название модуля;
- `credit_units` – количество зачетных единиц;
- `total_hours` – общее количество часов;
- `lectures_hours` – количество часов, выделенных для проведения лекций;
- `seminars_hours` – количество часов, выделенных на семинарские занятия;
- `laboratory_work_hours` – количество часов, выделенное на лабораторные работы;

- `independent_work_hours` – количество часов, выделенное на самостоятельную работу студентом;
- `min_scores` – минимальное количество баллов, которое нужно набрать обучающемуся для закрытия этого модуля;
- `max_scores` – максимальное количество баллов, которое можно набрать в течении этого модуля;
- `competency_codes` – компетенции, закрепленные за темой.

Эта таблица имеет связь с таблицей `discipline_work_program` с отношением один-ко-многим. Дисциплина чаще всего имеет несколько модулей.

Таблица `discipline_material` содержит сведения о материалах, необходимых для освоения дисциплины:

- `id` – уникальный идентификатор таблицы;
- `discipline_id` – внешний ключ для таблицы `discipline_work_program`;
- `material_id` – литература, необходимая для освоения дисциплины;

Эта таблица имеет связь с таблицей `discipline_work_program` с отношением один-ко-многим. Для освоения дисциплины, обычно, необходимо более чем один источник информации.

Кроме того, для каждой таблицы будет реализован триггер, срабатывающий после обновления или удаления данных из таблиц. Этот триггер будет посылать сигнал базе данных кэширования, с помощью языка `plpython3u`, о необходимости обновить или удалить информацию из кэша. С помощью таких триггеров можно решить проблему синхронизации данных в хранилище и кэше.

2.3 Проектирование базы данных кэширования

База данных кэширования будет реализована с помощью использования СУБД Tarantool. В базе данных будут полностью продублированы таблицы (в виде спейсов) из хранилища рабочих программ дисциплин. Первичным ключом будет являться поле с уникальным идентификатором этих таблиц (`id`). Кроме того, для спейсов хранящих поле `discipline_id` будет добавлен вторичный ключ по этому полю, для удобного и быстрого сбора нужных данных по заданной дисциплине.

При запросе данных у приложения, будет проводиться проверка, присутствует ли запись в кэше. Если запись присутствует, запрос к базе данных рабочих программ дисциплин производиться не будет и будут возвращены данные из кэша. В противном случае, будет произведен запрос к базе данных хранящую информацию о дисциплинах.

Все спейсы будут созданы на основе движка `memtx`, хранящего все данные в оперативной памяти. Персистентность данных будет обеспечивается при помощи ведения журнала транзакция и системы «снимков» текущего состояния кэша. Эти технологии помогут решить проблему «холодного» старта базы данных кэширования.

Вывод

В данном разделе были представлены этапы проектирования баз данных и рассмотрены особенности используемых СУБД на архитектурном уровне.

3 Технологическая часть

В данном разделе представлены архитектура приложения, средства разработки программного обеспечения, детали реализации и способы взаимодействия с программным продуктом.

3.1 Архитектура приложения

Предполагается, что разрабатываемое приложение является микросервисом одного большего сервиса (LMS). Доступ к данным, хранящимся в приложении будет получен с помощью REST API.

Серверная часть коммуницирует с базами данных при помощи специализированных коннекторов, позволяющих делать запросы к базе данных на языке программирования, который используется для разработки приложения.

Общая схема архитектура приложения представлена на рисунке 3.1.

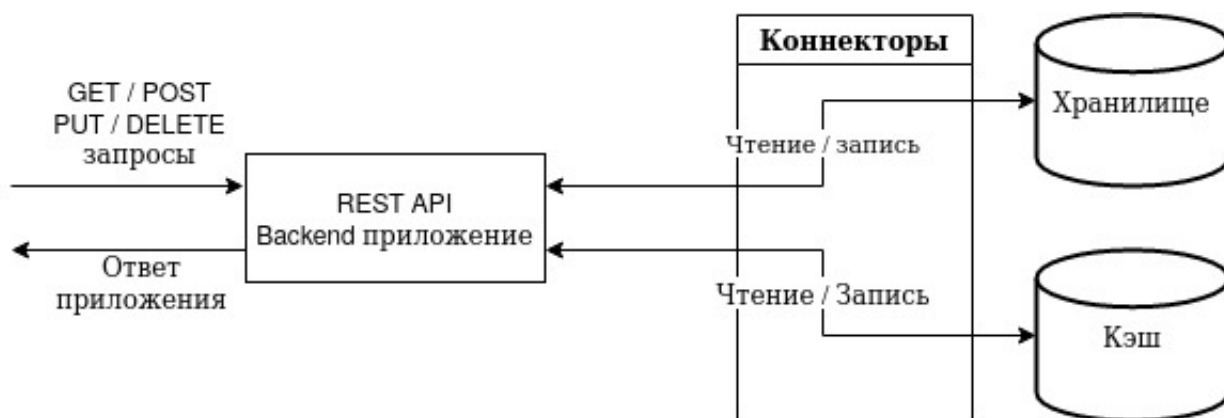


Рис. 3.1: Схема архитектуры приложения

3.2 Средства реализации

Для разработки серверной части был выбран язык программирования Python [35]. Данный выбор обусловлен простотой языка и разворачиванием

REST-приложений на нём. Также в Python имеется очень тесная интеграция с СУБД PostgreSQL, которая будет использоваться для хранения данных о рабочей программы дисциплины. Язык Python имеет коннекторы для платформы in-memory вычислений Tarantool. Для реализации REST API был выбран фреймворк Flask [36].

Для коммуникации серверной части приложения с базами данных были использованы коннекторы: python-tarantool [37] для Tarantool и psycopg2 [38] для PostgreSQL.

Для упаковки приложения в готовый продукт была выбрана система контейнеризации Docker [39]. С помощью Docker, можно создать изолированную среду для программного обеспечения, которое можно будет развернуть на различных операционных системах без дополнительного вмешательства для обеспечения совместимости.

Тестирование программного продукта производилось с помощью фреймворка pytest [40]. Данный фреймворк позволяет писать как модульные, так и функциональные тесты. Для тестирования ПО был реализован ряд функциональных тестов.

3.3 Детали реализации

В листингах 3.1 – 3.3 представлены листинги взаимодействия клиента с сервером и обработка запросов клиента, взаимодействия приложения с базами данных и кэширования данных.

```
1 import logging
2 import time
3
4 from flask import Flask, request
5 import psycopg2
6
7 import db.models as m
8 from db.cache.cache import CacheLRU
9 from db.utils import Utils
10
11 from services.controller import Controller
12 from services.handler import RequestHandler
13 from services.document_parser import DocumentParser
14
15 app = Flask(__name__)
```

```

16 app.config['JSON_AS_ASCII'] = False
17
18 # Waiting for database initialization
19 time.sleep(1)
20 cache = CacheLRU()
21 controller = Controller()
22
23
24 @app.route("/rpd/save", methods=["POST"])
25 def upload_from_file():
26     logging.info(f"/rpd/save router called")
27     repo_psql = controller.discipline_work_program_repo_psql
28     filename = request.get_json()["filename"]
29
30     try:
31         parser = DocumentParser(filename)
32         model = parser.get_discipline_program()
33         model.id = repo_psql.save(model)
34         model = Utils.save_discipline_fields(model, controller.psql_repos)
35     except Exception as err:
36         logging.error(err)
37         return RequestHandler.error_response(500, err)
38
39     return RequestHandler.success_response(data=model)
40
41
42 @app.route("/rpd/<id>", methods=["GET"])
43 def get_dpw_by_id(id=None):
44     logging.info(f"/dpw/{id} (GET) router called")
45     repos = {
46         "storage": controller.psql_repos,
47         "cache": controller.tarantool_repos,
48     }
49
50     try:
51         model = cache.get_by_primary(int(id), "discipline_work_program", repos
52         )
53         model = Utils.collect_discipline_fields(model, cache, repos)
54     except Exception as err:
55         logging.error(err)
56         return RequestHandler.error_response(500, err)
57
58     return RequestHandler.success_response(data=model)
59
60 @app.route("/rpd/<id>", methods=["DELETE"])
61 def remove_dpw_by_id(id=None):
62     logging.info(f"/rpd/{id} (DELETE) router called")

```

```

63 repo_psql = controller.discipline_work_program_repo_psql
64 repo_tarantool = controller.discipline_work_program_repo_tarantool
65 repos = {
66     "storage": controller.psql_repos,
67     "cache": controller.tarantool_repos,
68 }
69
70 try:
71     model = cache.get_by_primary(int(id), "discipline_work_program", repos
72 )
73     Utils.remove_discipline_fields(model, cache, repos)
74     repo_psql.remove(model.id)
75 except Exception as err:
76     logging.error(err)
77     return RequestHandler.error_response(500, err)
78
79 return RequestHandler.success_response(
80     message=f"Work program of discipline with id = {id} successfully
81     deleted")
82
83 @app.route("/rpd/<id>", methods=["PUT"])
84 def edit_dpw_by_id(id=None):
85     logging.info(f"/rpd/{id} (PUT) router called")
86     repo_psql = controller.discipline_work_program_repo_psql
87     repo_taranntool = controller.discipline_work_program_repo_tarantool
88
89 try:
90     model = repo_psql.edit(id=int(id), fields=request.get_json())
91 except Exception as err:
92     logging.error(err)
93     return RequestHandler.error_response(500, err)
94
95 return RequestHandler.success_response(
96     message=f"Work program of discipline with id = {id} successfully
97     changed")
98
99 @app.route("/cache/clear", methods=["PUT"])
100 def clear_cache():
101     logging.info(f"Clear cache router called")
102     cache_repos = controller.tarantool_repos
103
104 try:
105     cache.clear(cache_repos, cache_repos["discipline_work_program"].
106     connection)
107 except Exception as err:
108     logging.error(err)

```



```

107     return RequestHandler.error_response(500, err)
108
109     return RequestHandler.success_response(message=f"Cache successfully
110         cleared")
111
112 @app.route("/cache/size", methods=["GET"])
113 def cache_size():
114     logging.info(f"Get cache size router called")
115     cache_repos = controller.tarantool_repos
116
117     try:
118         size = cache.get_cache_size(cache_repos["discipline_work_program"].
119             connection)
119     except Exception as err:
120         logging.error(err)
121         return RequestHandler.error_response(500, err)
122
123     return RequestHandler.success_response(message=f"Cache size is {size}")
124
125
126 @app.route("/cache/<id>", methods=["DELETE"])
127 def remove_from_cache(id=None):
128     logging.info(f"Remove from cache with id = {id} router called")
129     space_name = request.get_json()["space_name"]
130
131     try:
132         cache.remove(int(id), space_name, controller.tarantool_repos[
133             space_name])
133     except Exception as err:
134         logging.error(err)
135         return RequestHandler.error_response(500, err)
136
137     return RequestHandler.success_response(message=f"Cache successfully
138         cleared")
139
140 @app.errorhandler(404)
141 def page_not_found(error):
142     return RequestHandler.error_response(404, "Invalid URL!")
143
144 if __name__ == "__main__":
145     app.run(host='0.0.0.0')

```

Листинг 3.1: Листинг взаимодействия клиента с сервером

```

1 from datetime import datetime
2 from heapq import heappush as insert_queue, heappop as extract_maximum
3

```

```

4 from db.utils import Utils
5
6 class CacheLRU():
7     def __init__(self, max_size=100):
8         self.max_size = max_size
9         self.current_size = None
10        self.time_queue = []
11
12    def get_by_primary(self, key, space_name, repos):
13        current_time = datetime.timestamp(datetime.now())
14        cache_repo = repos["cache"][space_name]
15        storage_repo = repos["storage"][space_name]
16
17        if self.current_size is None:
18            self.current_size = self.get_cache_size(cache_repo.connection)
19
20        cached_object = cache_repo.get_by_id(key)
21        if cached_object is not None:
22            insert_queue(self.time_queue, (current_time, key, space_name))
23            return cached_object
24
25        if self.current_size >= self.max_size:
26            min_key, cached_space_name = extract_maximum(self.time_queue)[1:]
27            cache_repo.remove(min_key)
28            self.decrement_cache_size(cache_repo.connection)
29
30        obj = storage_repo.get_by_id(key)
31        cache_repo.save(obj)
32        self.increment_cache_size(cache_repo.connection)
33        insert_queue(self.time_queue, (current_time, key, space_name))
34
35        return obj
36
37    def get_by_filter(self, space_name, key, index, repos):
38        current_time = datetime.timestamp(datetime.now())
39        cache_repo = repos["cache"][space_name]
40        storage_repo = repos["storage"][space_name]
41
42        if self.current_size is None:
43            self.current_size = self.get_cache_size(cache_repo.connection)
44
45        cached_objects, primary_keys = cache_repo.get_by_filter(index, key)
46        if cached_objects is not None:
47            for obj, primary_key in zip(cached_objects, primary_keys):
48                insert_queue(self.time_queue, (current_time, primary_key,
49space_name))
50
51        total_cnt = storage_repo.get_objects_count_by_filter(index, key)

```

```

51     objects_left = total_cnt if cached_objects is None else total_cnt -
len(cached_objects)
52     if objects_left == 0:
53         return cached_objects
54
55     if objects_left == total_cnt:
56         primary_keys = [-1] # Full-scan confirmed
57
58     while self.current_size + objects_left >= self.max_size:
59         min_key, space_name = extract_maximum(self.time_queue)[1:]
60         repos["cache"][space_name].remove(min_key)
61         self.decrement_cache_size(cache_repo.connection)
62
63     filter_str = Utils.get_noncached_filter_string(len(primary_keys),
index)
64     objects, primary_keys = storage_repo.get_by_filter(filter_str, tuple(
map(int, [key] + primary_keys)))
65
66     for obj, primary_key in zip(objects, primary_keys):
67         cache_repo.save(obj)
68         self.increment_cache_size(cache_repo.connection)
69         insert_queue(self.time_queue, (datetime.timestamp(datetime.now()),
primary_key, space_name))
70
71     return objects
72
73 def insert(self, key, obj, repo):
74     current_time = datetime.timestamp(datetime.now())
75
76     if self.current_size is None:
77         self.current_size = self.get_cache_size(repo.connection)
78
79     if self.current_size >= self.max_size:
80         key = extract_maximum(self.time_queue)[-1]
81         repo.remove(key)
82
83     insert_queue(self.time_queue, (current_time, key, repo._meta["
space_name"])))
84     self.increment_cache_size(repo.connection)
85     repo.save(obj)
86
87 def remove(self, key, space_name, repo):
88     if self.current_size is None:
89         self.current_size = self.get_cache_size(repo.connection)
90
91     if repo.remove(key) is not None:
92         self.time_queue = list(filter(lambda x: x[1] != key or x[2] !=
space_name, self.time_queue))

```

```

93         self.decrement_cache_size(repo.connection)
94
95     def update(self, key, repo):
96         obj = self.remove(key, repo)
97         self.insert(key, obj, repo)
98
99     def clear(self, repos, connection):
100         for key in repos:
101             space_name = repos[key]._meta["space_name"]
102             connection.call(f"box.space.{space_name}:truncate", ())
103
104             self.current_size = 0
105             connection.space("cache_size").replace((1, 0))
106
107     def increment_cache_size(self, connection):
108         self.current_size += 1
109         connection.space("cache_size").replace((1, self.current_size))
110
111     def decrement_cache_size(self, connection):
112         self.current_size -= 1
113         connection.space("cache_size").replace((1, self.current_size))
114
115     def get_cache_size(self, connection):
116         return connection.space("cache_size").select()[0][1]

```

Листинг 3.2: Листинг модуля кэширования данных с политикой
вытеснения LRU

```

1 import logging
2 from db.utils import Utils
3 from db.repos.abstract import AbstractRepo
4 import db.models as models
5
6 class DisciplineWorkProgramRepoTarantool(AbstractRepo):
7     def __init__(self, connection):
8         self.connection = connection
9         self._meta = {
10             "space_name": "discipline_work_program",
11             "field_names": {"id": 1, "name": 2, "author": 3, "competency": 4}
12         }
13
14         self.space = connection.space(self._meta['space_name'])
15
16     def save(self, model):
17         if not isinstance(model, models.DisciplineWorkProgram):
18             logging.error("Trying to save DisciplineWorkProgram object of
19                 invalid type")
20             raise TypeError("Expected object is instance of
21                 DisciplineWorkProgram")

```

```

20
21     self.space.insert((model.id, model.name, model.author, model.
competency))
22
23 def get_by_id(self, model_id):
24     obj = self.space.select(model_id)
25     if len(obj) == 0:
26         return None
27
28     return models.DisciplineWorkProgram(*obj[0])
29
30 def get_by_filter(self, index, key):
31     raw_objects = self.space.select(key, index=index)
32     if len(obj) == 0:
33         return None, None
34
35     models_list = list()
36     primary_keys = list()
37     for obj in raw_objects:
38         model = models.DisciplineWorkProgram(*obj)
39         models_list.append(model)
40         primary_keys.append(model.id)
41
42     return models_list, primary_keys
43
44
45 def get_all(self):
46     raw_objects = self.space.select()
47     if len(raw_objects) == 0:
48         return None
49
50     models_list = list()
51     for obj in raw_objects:
52         models_list.append(models.DisciplineWorkProgram(*obj))
53
54     return models_list
55
56 def remove(self, id):
57     obj = self.space.delete(id)
58     if len(obj) == 0:
59         return None
60
61     return obj
62
63 def edit(self, *args, **kwargs):
64     obj_id = kwargs['id']
65     updated_args = Utils.get_tarantool_update_args(kwargs['fields'], self.
_meta['field_names'])

```

```

66
67     return self.space.update(obj_id, updated_args)[0]
68
69
70 class LearningOutcomesRepoTarantool(AbstractRepo):
71     def __init__(self, connection):
72         self.connection = connection
73         self._meta = {
74             "space_name": "learning_outcomes",
75             "field_names": {
76                 "id": 1,
77                 "discipline_id": 2,
78                 "competency_code": 3,
79                 "formulation": 4,
80                 "results": 5,
81                 "forms_and_methods": 6,
82             }
83         }
84
85         self.space = connection.space(self._meta['space_name'])
86
87     def save(self, model):
88         if not isinstance(model, models.LearningOutcomes):
89             logging.error("Trying to save LeraningOutcomes object of invalid
90                             type")
91             raise TypeError("Expected object is instance of LearningOutcomes")
92
93         self.space.insert(tuple(model.__dict__.values()))
94
95     def get_by_filter(self, index, key):
96         raw_objects = self.space.select(key, index=index)
97         if len(raw_objects) == 0:
98             return None, None
99
100         models_list = list()
101         primary_keys = list()
102         for obj in raw_objects:
103             model = models.LearningOutcomes(*obj)
104             models_list.append(model)
105             primary_keys.append(model.id)
106
107         return models_list, primary_keys
108
109     def remove(self, id):
110         obj = self.space.delete(id)
111         if len(obj) == 0:
112             return None

```

```

113     return obj
114
115
116 class DisciplineScopeRepoTarantool(AbstractRepo):
117     def __init__(self, connection):
118         self.connection = connection
119         self._meta = {
120             "space_name": "discipline_scope_semester",
121             "field_names": {
122                 "id": 1,
123                 "discipline_id": 2,
124                 "semester_number": 3,
125                 "credit_units": 4,
126                 "total_hours": 5,
127                 "lectures_hours": 6,
128                 "laboratory_work_hours": 7,
129                 "independent_work_hours": 8,
130                 "certification_type": 9
131             }
132         }
133
134         self.space = connection.space(self._meta['space_name'])
135
136     def save(self, model):
137         if not isinstance(model, models.DisciplineScope):
138             logging.error("Trying to save DisciplineScope object of invalid type")
139             raise TypeError("Expected object is instance of DisciplineScope")
140
141         self.space.insert(tuple(model.__dict__.values()))
142
143     def get_by_filter(self, index, key):
144         raw_objects = self.space.select(key, index=index)
145         if len(raw_objects) == 0:
146             return None, None
147
148         models_list = list()
149         primary_keys = list()
150         for obj in raw_objects:
151             model = models.DisciplineScope(*obj)
152             models_list.append(model)
153             primary_keys.append(model.id)
154
155         return models_list, primary_keys
156
157     def remove(self, id):
158         obj = self.space.delete(id)
159         if len(obj) == 0:

```

```

160         return None
161
162     return obj
163
164
165 class DisciplineModuleRepoTarantool(AbstractRepo):
166     def __init__(self, connection):
167         self.connection = connection
168         self._meta = {
169             "space_name": "discipline_module",
170             "field_names": {
171                 "id": 1,
172                 "discipline_id": 2,
173                 "name": 3,
174                 "semester_number": 4,
175                 "lectures_hours": 5,
176                 "seminars_hours": 6,
177                 "laboratory_work_hours": 7,
178                 "independent_work_hours": 8,
179                 "min_scores": 9,
180                 "max_scores": 10,
181                 "competency_codes": 11
182             }
183         }
184
185         self.space = connection.space(self._meta['space_name'])
186
187     def save(self, model):
188         if not isinstance(model, models.DisciplineModule):
189             logging.error("Trying to save DisciplineModule object of invalid
190                             type")
191             raise TypeError("Expected object is instance of DisciplineModule")
192
193         self.space.insert(tuple(model.__dict__.values()))
194
195     def get_by_filter(self, index, key):
196         raw_objects = self.space.select(key, index=index)
197         if len(raw_objects) == 0:
198             return None, None
199
200         models_list = list()
201         primary_keys = list()
202         for obj in raw_objects:
203             model = models.DisciplineModule(*obj)
204             models_list.append(model)
205             primary_keys.append(model.id)
206
207         return models_list, primary_keys

```



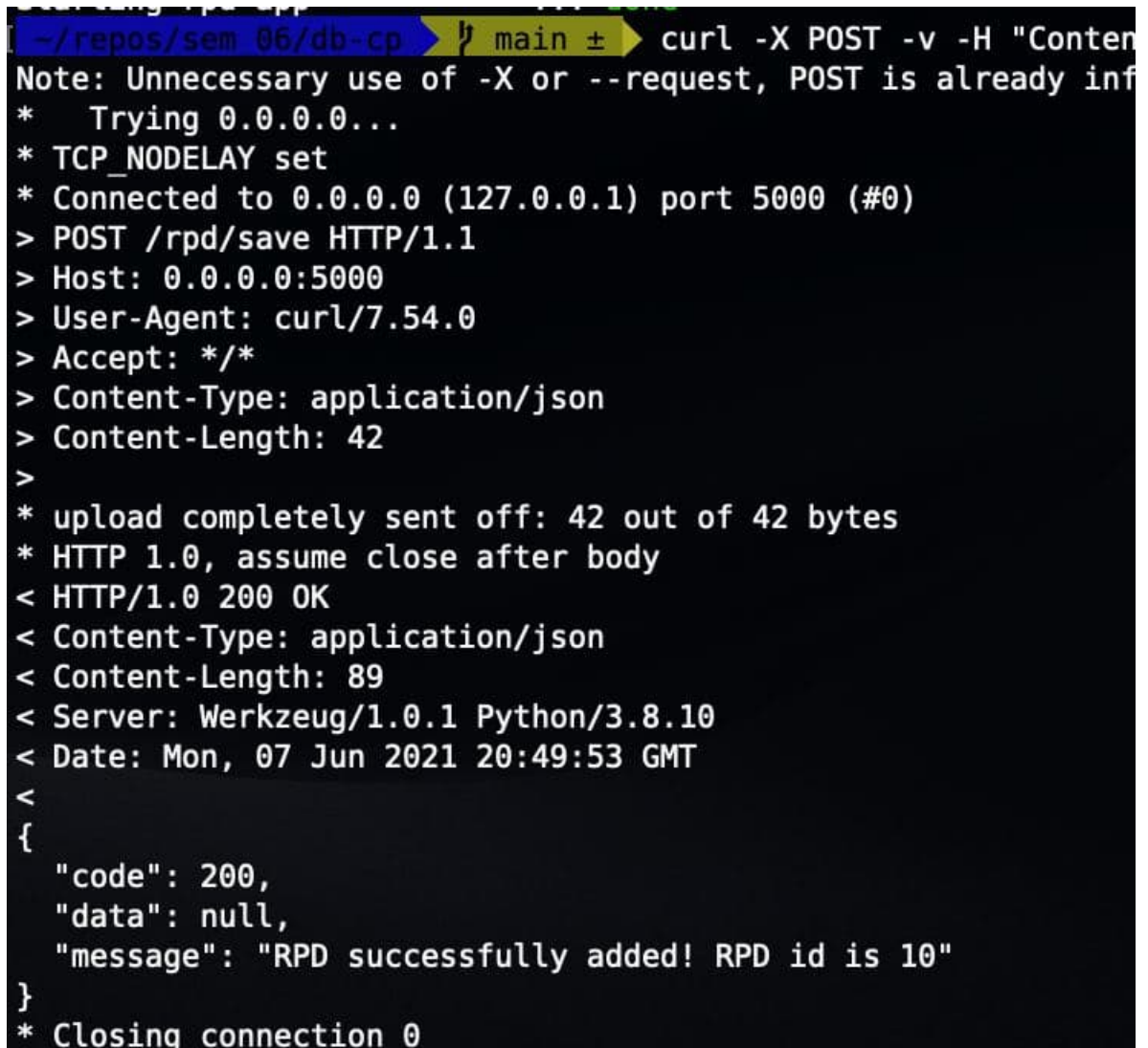
```

207
208 def remove(self, id):
209     obj = self.space.delete(id)
210     if len(obj) == 0:
211         return None
212
213     return obj
214
215
216 class DisciplineMaterialRepoTarantool(AbstractRepo):
217     def __init__(self, connection):
218         self.connection = connection
219         self._meta = {
220             "space_name": "discipline_material",
221             "field_names": {"id": 1, "discipline_id": 2, "materials": 3}
222         }
223
224         self.space = connection.space(self._meta['space_name'])
225
226     def save(self, model):
227         if not isinstance(model, models.DisciplineMaterial):
228             logging.error("Trying to save DisciplineMaterial object of invalid
229                             type")
230             raise TypeError("Expected object is instance of DisciplineMaterial")
231
232         self.space.insert(tuple(model.__dict__.values()))
233
234     def get_by_filter(self, index, key):
235         raw_objects = self.space.select(key, index=index)
236         if len(raw_objects) == 0:
237             return None, None
238
239         models_list = list()
240         primary_keys = list()
241         for obj in raw_objects:
242             model = models.DisciplineMaterial(*obj)
243             models_list.append(model)
244             primary_keys.append(model.id)
245
246         return models_list, primary_keys
247
248     def remove(self, id):
249         obj = self.space.delete(id)
250         if len(obj) == 0:
251             return None
252
253         return obj

```

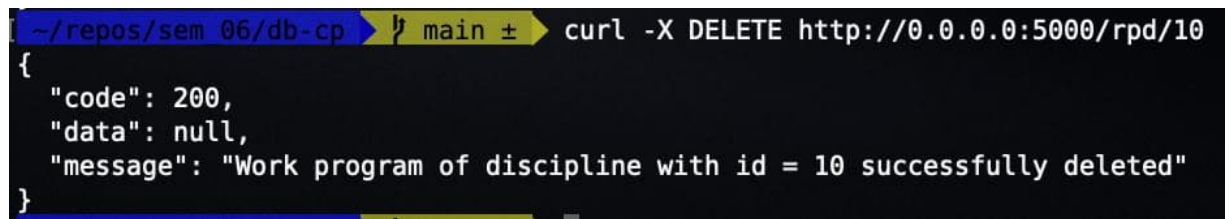
3.4 Взаимодействие с приложением

На рисунках 3.2 – 3.3 представлены примеры запросов к приложению и его ответы.



```
~/repos/sem 06/db-cp main ± curl -X POST -v -H "Content-Type: application/json" http://127.0.0.1:5000/rpd/save HTTP/1.1
Note: Unnecessary use of -X or --request, POST is already inferred
* Trying 0.0.0.0...
* TCP_NODELAY set
* Connected to 0.0.0.0 (127.0.0.1) port 5000 (#0)
> POST /rpd/save HTTP/1.1
> Host: 0.0.0.0:5000
> User-Agent: curl/7.54.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 42
>
* upload completely sent off: 42 out of 42 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 89
< Server: Werkzeug/1.0.1 Python/3.8.10
< Date: Mon, 07 Jun 2021 20:49:53 GMT
<
{
  "code": 200,
  "data": null,
  "message": "RPD successfully added! RPD id is 10"
}
* Closing connection 0
```

Рис. 3.2: Пример сохранения информации о дисциплине посредством http запроса

A terminal window with a dark background. The prompt is `~/repos/sem 06/db-cp` and the command is `curl -X DELETE http://0.0.0.0:5000/rpd/10`. The output is a JSON object: `{ "code": 200, "data": null, "message": "Work program of discipline with id = 10 successfully deleted" }`.

```
~/repos/sem 06/db-cp ➤ main ± curl -X DELETE http://0.0.0.0:5000/rpd/10
{
  "code": 200,
  "data": null,
  "message": "Work program of discipline with id = 10 successfully deleted"
}
```

Рис. 3.3: Пример удаления информации о дисциплине посредством http запроса

Вывод

В данном разделе были представлена архитектура и средства реализации программного обеспечения, листинги ключевых компонентов системы и пример возвращаемых данных системой.

```

[ ~/repos/sem 06/db-cp main ± curl -X GET http://0.0.0.0:5000/rpd/10
{
  "code": 200,
  "data": {
    "author": "Исаев А.Л.",
    "competency": "СУОС3++",
    "discipline_material": [
      {
        "discipline_id": 10,
        "id": 89,
        "material": "Информатика. Конспект лекций: учебное пособие / Исаев
7038-4540-0."
      },
      {
        "discipline_id": 10,
        "id": 90,
        "material": "Объектно-ориентированное программирование: учебник для
//ebooks.bmstu.ru/catalog/97/book1033.html. - ISBN 978-5-7038-3921-8."
      },
      {
        "discipline_id": 10,
        "id": 91,
        "material": "Дополнительные учебные материалы"
      },
      {
        "discipline_id": 10,
        "id": 92,
        "material": "Информатика и программирование. Основы информатики: уч
      },
      {
        "discipline_id": 10,
        "id": 93,
        "material": "Информатика и программирование. Алгоритмизация и прогр
      },
      {
        "discipline_id": 10,
        "id": 94,
        "material": "Информатика. Конспект практических занятий: учебно-мет
      },
      {
        "discipline_id": 10,
        "id": 95,
        "material": "Введение в визуальное программирование на языке C в ср
      },
      {
        "discipline_id": 10,
        "id": 96,
        "material": "Обработка нечисловых типов данных в среде MS VS C++: у
bmstu.ru/catalog/97/book1618.htm.l - ISBN 978-5-7038-4638-4."
      },
    ]
  }
}

```

Рис. 3.4: Пример получения информации о дисциплине посредством http запроса

4 Исследовательская часть

В данном разделе представлена постановка эксперимента по сравнению занимаемого времени для получения данных из хранилища с использованием и без использования кэширования.

4.1 Постановка эксперимента

В данном подразделе представлены цель, описание и результаты эксперимента.

4.1.1 Цель эксперимента

Целью эксперимента является сравнение времени, требуемого для получения данных о рабочей программе дисциплины с и без использованием кэширования данных.

4.1.2 Описание эксперимента

Сравнить занимаемое время можно при помощи отключения реализованного механизма кэширования. Для этого будет достаточно отключить базу данных, хранящую данные о кэшировании и каждый раз выполнять запрос напрямую к базе данных хранящую информацию о рабочих программах дисциплин.

Для проведения будут использоваться кэши разных размеров, а также разное количество запрашиваемых рабочих программ дисциплин. Будут произведены операции, для того чтобы запрашиваемые РПД могли оказаться в кэше.

В поставленном эксперименте одна рабочая дисциплина состоит из 32 единиц (таблиц).

4.1.3 Результат эксперимента

В таблицах 4.1 - 4.3 представлены результаты поставленного эксперимента.

Таблица 4.1: Результаты сравнения времени, необходимого для получения данных без кэширования и с кэшированием (размер кэша - 100 единиц)

Количество РПД	Время без кэширования, мс	Время с кэшированием, мс
1	62012	4554
5	366500	340250
10	751340	702250
25	2657210	2584172
100	9750742	9209781

Таблица 4.2: Результаты сравнения времени, необходимого для получения данных без кэширования и с кэшированием (размер кэша - 1000 единиц)

Количество РПД	Время без кэширования, мс	Время с кэшированием, мс
1	70233	4301
5	398213	28231
10	720304	50300
25	2011763	175680
100	9542401	7001307

Таблица 4.3: Результаты сравнения времени, необходимого для получения данных без кэширования и с кэшированием (размер кэша - 5000 единиц)

Количество РПД	Время без кэширования, мс	Время с кэшированием, мс
1	90213	6521
5	360041	24664
10	519212	40227
25	1881132	109710
100	7796774	754991

На рисунках 4.1 - 4.3 представлены графики зависимости количество запрашиваемых РПД от времени, при разных размерах кэша.

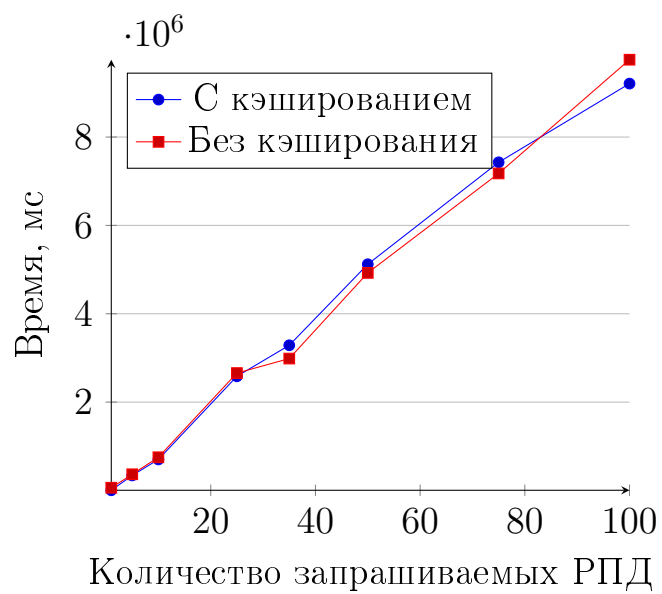


Рис. 4.1: Зависимость времени от количества запрашиваемых РПД (размера кэша 100 элементов)

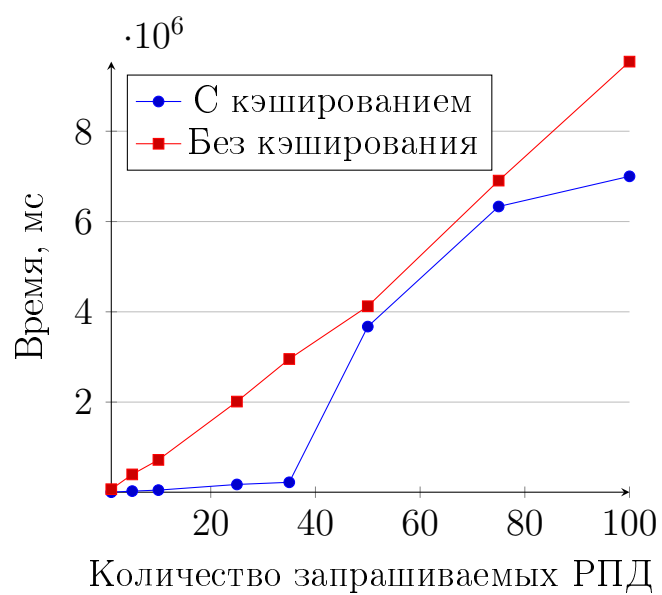


Рис. 4.2: Зависимость времени от количества запрашиваемых РПД (размера кэша 1000 элементов)

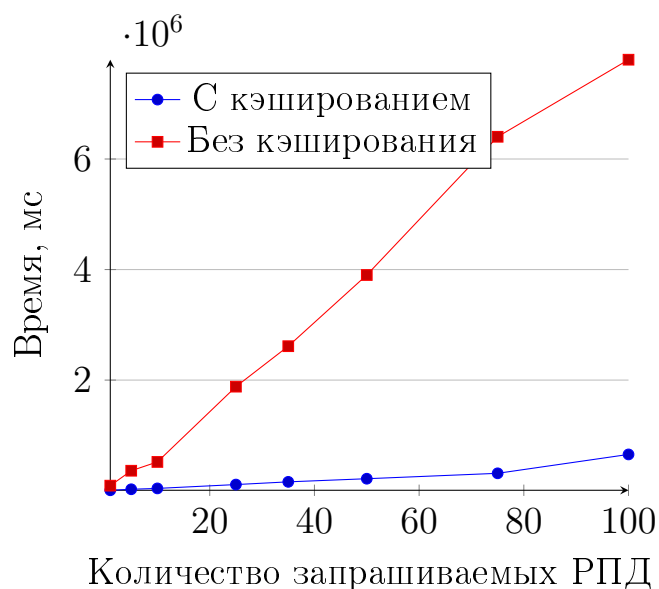


Рис. 4.3: Зависимость времени от количества запрашиваемых РПД (размера кэша 5000 элементов)

Вывод

В результате сравнения времени, необходимого для получения данных о рабочих программах дисциплин, кэширование данных показало неоднозначные результаты:

- приложение с кэшированием данных всегда работает быстрее;
- при запросах РПД превышающих максимальный размер кэша, приложение с кэшированием выигрывает по времени у обычного приложения без кэширования в среднем в 1.05 раза (вся эффективность кэширования практически нивелируется);
- если количество РПД в запросе не превышает максимальный размер кэша, выигрыш по времени в среднем составляет 12 раз (при условии, что выборочные данные находятся в кэше).

Эффективность кэширования данных в разработанном приложении полностью зависит от размера хранимых данных. Например, если есть возможность выделить кэш с максимальным размером хотя бы 30-50% от максимально возможного размера хранимых данных, это обеспечит эффектив-

ность по времени максимум в 12 раз – вряд ли кто-то будет запрашивать более 30% данных расположенных в хранилище.

Такой выигрыш по времени можно обеспечить только при условии что выборочные данные находятся в кэше, что, конечно, нереалистично. Можно сделать предположение, что хотя бы 20% (для этого нужно выбрать подходящую политику вытеснения из кэша) выборочных данных находятся в кэше – даже в таком случае доступ к данным будет ускорен в 2.4 раза.

Заключение

Во время выполнения курсового проекта было реализовано программное обеспечение для хранения, редактирования и удаления данных о рабочих программах дисциплин.

В ходе выполнения поставленной задачи были получены знания в области проектирования баз данных и кэширования данных. Были изучены типы хранения данных и типы СУБД. Поиск подходящего решения для поставленной задачи позволил повысить навыки поиска и анализа информации.

В результате проведенной работы было разработано программное обеспечение, демонстрирующее ускорение отклика приложения с помощью внедрения кэширования данных.

В ходе выполнения экспериментально-исследовательской части было установлено, что приложение с кэшированием данных всегда работает быстрее, чем без. Но, при этом, очень многое зависит от максимального размера кэша: при оптимальном максимальном размере приложение с кэшированием данных может иметь выигрыш по времени в 12 раз.

Литература

- [1] Положение о порядке разработки и утверждение рабочей программы дисциплины [Электронный ресурс]. Режим доступа: [https://www.volgmed.ru/uploads/files/2010-11/1180-polozhenie_o_poryadke_razrabotki_i_utverzhdeniya_rabochej_programmy_uchebnoj_discipliny_\(kursa\).doc](https://www.volgmed.ru/uploads/files/2010-11/1180-polozhenie_o_poryadke_razrabotki_i_utverzhdeniya_rabochej_programmy_uchebnoj_discipliny_(kursa).doc) (дата обращения: 07.06.2021).
- [2] Learning Management System (LMS) — HSE [Электронный ресурс]. Режим доступа: https://www.hse.ru/en/studyspravka/lms_student/ (дата обращения: 07.06.2021).
- [3] Microsoft Word - Word Processing Software | Microsoft 365 [Электронный ресурс]. Режим доступа: <https://www.microsoft.com/en-us/microsoft-365/word> (дата обращения: 07.06.2021).
- [4] What is a REST API? - Red Hat [Электронный ресурс]. Режим доступа: <https://www.redhat.com/en/topics/api/what-is-a-rest-api> (дата обращения: 07.06.2021).
- [5] Образовательные стандарты | МГТУ им. Н. Э. Баумана [Электронный ресурс]. Режим доступа: <https://bmstu.ru/plain/eduStandarts/> (дата обращения: 07.06.2021).
- [6] МГТУ им. Н. Э. Баумана – Официальный сайт [Электронный ресурс]. Режим доступа: <https://bmstu.ru/> (дата обращения: 07.06.2021).
- [7] Что такое база данных | Oracle Россия и СНГ [Электронный ресурс]. Режим доступа: <https://www.oracle.com/ru/database/what-is-database/> (дата обращения: 07.06.2021).
- [8] Что такое СУБД - RU-CENTER [Электронный ресурс]. Режим доступа: https://www.nic.ru/help/что-такое-sbd_8580.html (дата обращения: 07.06.2021).
- [9] Что такое микросервисная архитектура: простое объяснение | MCS Mail.ru [Электронный ресурс]. Режим доступа: <https://mcs.mail.ru/blog/prostym-jazykom-o-mikroservisnoj-arhitekture> (дата обращения: 07.06.2021).

- [10] Race conditions and deadlocks - Microsoft Docs [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/en-us/troubleshoot/dotnet/visual-basic/race-conditions-deadlocks> (дата обращения: 07.06.2021).
- [11] What is OLTP? | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/cloud/learn/oltp> (дата обращения: 07.06.2021).
- [12] What is OLAP? | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/cloud/learn/olap> (дата обращения: 07.06.2021).
- [13] PostgreSQL: Документация. [Электронный ресурс]. Режим доступа: <https://postgrespro.ru/docs/postgresql/> (дата обращения: 07.06.2021).
- [14] PostgreSQL: вчера, сегодня, завтра [Электронный ресурс]. Режим доступа: <https://postgrespro.ru/blog/media/17768> (дата обращения: 07.06.2021).
- [15] Транзакции, ACID, CAP | GeekBrains [Электронный ресурс]. Режим доступа: https://gb.ru/posts/acid_cap_transactions (дата обращения: 07.06.2021).
- [16] Documentation: 12: 13.1. Introduction - PostgreSQL [Электронный ресурс]. Режим доступа: <https://www.postgresql.org/docs/12/mvcc-intro.html> (дата обращения: 07.06.2021).
- [17] Применение блокировок чтения/записи | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/docs/ru/aix/7.2?topic=programming-using-readwrite-locks> (дата обращения: 07.06.2021).
- [18] SQL Language | Oracle [Электронный ресурс]. Режим доступа: <https://www.oracle.com/database/technologies/appdev/sql.html> (дата обращения: 07.06.2021).
- [19] Oracle | Integrated Cloud Applications and Platform Services [Электронный ресурс]. Режим доступа: <https://www.oracle.com/index.html> (дата обращения: 07.06.2021).

- [20] DB-Engines Ranking [Электронный ресурс]. Режим доступа: <https://db-engines.com/en/ranking> (дата обращения: 07.06.2021).
- [21] MySQL Database Service is a fully managed database service to deploy cloud-native applications. [Электронный ресурс]. Режим доступа: <https://www.mysql.com/> (дата обращения: 07.06.2021).
- [22] MySQL Reference Manual 8.0: The InnoDB Storage Engine [Электронный ресурс]. Режим доступа: <https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html> (дата обращения: 07.06.2021).
- [23] MySQL Reference Manual 16.2: The MyISAM Storage Engine [Электронный ресурс]. Режим доступа: <https://dev.mysql.com/doc/refman/8.0/en/myisam-storage-engine.html> (дата обращения: 07.06.2021).
- [24] PHP: Hypertext Preprocessor [Электронный ресурс]. Режим доступа: <https://www.php.net/> (дата обращения: 07.06.2021).
- [25] The Perl Programming Language [Электронный ресурс]. Режим доступа: <https://www.perl.org/> (дата обращения: 07.06.2021).
- [26] PostgreSQL: Документация: 9.6: 44.1. Python 2 и Python 3. [Электронный ресурс]. Режим доступа: <https://postgrespro.ru/docs/postgresql/9.6/plpython-python23> (дата обращения: 07.06.2021).
- [27] Что такое NoSQL? | Amazon AWS [Электронный ресурс]. Режим доступа: <https://aws.amazon.com/ru/nosql/> (дата обращения: 07.06.2021).
- [28] Tarantool – Платформа In-memory вычислений [Электронный ресурс]. Режим доступа: <https://www.tarantool.io/ru/> (дата обращения: 07.06.2021).
- [29] The Programming Language Lua [Электронный ресурс]. Режим доступа: <http://www.lua.org/> (дата обращения: 07.06.2021).
- [30] Tech Confronts Its Use of the Labels «Master» and «Slave» [Электронный ресурс]. Режим доступа: <https://www.wired.com/story/tech-confronts-use-labels-master-slave/> (дата обращения: 07.06.2021).

- [31] How To Set Up MySQL Master-Master Replication [Электронный ресурс]. Режим доступа: <https://www.digitalocean.com/community/tutorials/how-to-set-up-mysql-master-master-replication> (дата обращения: 07.06.2021).
- [32] Движки базы данных | Tarantool [Электронный ресурс]. Режим доступа: <https://www.tarantool.io/ru/doc/latest/book/box/engines/> (дата обращения: 07.06.2021).
- [33] Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker [Электронный ресурс]. Режим доступа: <https://redis.io/> (дата обращения: 07.06.2021).
- [34] LRU, метод вытеснения из кэша | Habr [Электронный ресурс]. Режим доступа: <https://habr.com/ru/post/136758/> (дата обращения: 07.06.2021).
- [35] The official home of the Python Programming Language. [Электронный ресурс]. Режим доступа: <https://www.python.org/> (дата обращения: 07.06.2021).
- [36] Welcome to Flask — Flask Documentation (2.0.x) [Электронный ресурс]. Режим доступа: <https://flask.palletsprojects.com/en/2.0.x/> (дата обращения: 07.06.2021).
- [37] Python driver for Tarantool - GitHub [Электронный ресурс]. Режим доступа: <https://github.com/tarantool/tarantool-python> (дата обращения: 07.06.2021).
- [38] Psycopg – PostgreSQL database adapter for Python [Электронный ресурс]. Режим доступа: <https://www.psycopg.org/docs/> (дата обращения: 07.06.2021).
- [39] Docker: Empowering App Development for Developers [Электронный ресурс]. Режим доступа: <https://www.docker.com/> (дата обращения: 07.06.2021).

[40] pytest: helps you write better programs — pytest documentation [Электронный ресурс]. Режим доступа: <https://docs.pytest.org/en/6.2.x/> (дата обращения: 07.06.2021).