



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Категоризация данных для предвыборных кампаний»

Студент ИУ7-64Б
(Группа)

(Подпись, дата)

А. А. Андреев
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Т. Н. Романова
(И. О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка к курсовой работе содержит 67 страниц, 26 иллюстрации, 4 таблицы, 28 источников, 1 приложений.

Ключевые слова: База данных, PostgreSQL, Python, гражданин, мандат, выборы, срез данных.

Объектом разработки является клиент-серверное приложение, предназначенного для получения аналитических сводок по гражданам.

Приложение реализовано на языке программирования Python3 с использованием Flask и Tarantool.

Целью курсовой работы является разработка структуры базы данных, которая содержит информацию о гражданах, участвующих в выборной процедуре по определенному мандату, и их принадлежность к определенной категории. Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ существующих решений;
- определить необходимую функциональность;
- провести анализ существующих СУБД;
- изучить стек технологий, необходимых для создания веб-приложения;
- спроектировать базу данных, необходимую для хранения и структурирования данных;
- реализовать спроектированную базу данных с использованием выбранной СУБД;
- реализовать веб-приложение, обеспечивающее взаимодействие с реализованной базой данных и решающее задачи, поставленные в техническом задании.

В результате проведенной работы исследовательским путем было получено, что приложение с кэшированием работает значительно быстрее, чем без него. Также исследование показало, что внедрение кэширования позволяет оптимизировать скорость работы до 19 раз.

В дальнейшем программа может быть модернизирована за счёт добавления новых сущностей, а также с использованием оптимизации запросов.

Содержание

Введение	6
1 Аналитический раздел	8
1.1 Формализация задачи	8
1.2 Краткий анализ аналогичных решений	10
1.3 Структура паспорта гражданина	10
1.4 Базы данных и системы управления базами данны	11
1.5 Хранение данных гражданина	11
1.5.1 Классификация баз данных по способу хранения	12
1.5.2 Обзор СУБД с построчным хранением	12
1.5.3 Выбор СУБД для решения задачи	14
1.6 Кэширование данных	14
1.6.1 Проблемы кэширования данных	15
1.6.2 Обзор in-memory NoSQL СУБД	16
1.6.3 Формализация данных и выбор СУБД	18
1.7 Вывод	18
2 Конструкторский раздел	20
2.1 Проектирование архитектуры и верхнеуровневой логики	20
2.2 Проектирование базы данных пользователей системы	21
2.3 Проектирование базы данных граждан страны	22
2.4 Проектирование бизнес-логики и работы интерфейса	26
2.5 Проектирование базы данных кэширования	28
3 Технологический раздел	29
3.1 Архитектура WEB-приложения	29
3.2 Средства реализации	29
3.3 Генерация исходных данных	30
3.4 Детали реализации	30
3.5 Взаимодействие с приложением	31
3.5.1 Авторизация	31
3.5.2 Меню интерфейса	32
3.5.3 Дашбоард	33

3.5.4	Страница модератора	35
3.5.5	Страница администратора	36
4	Экспериментальный раздел	40
4.1	Исследование зависимости времени получения данных без ке- ширования и с кэшированием	40
4.2	Вывод	44
	Заключение	45
	Список использованных источников	46
	ПРИЛОЖЕНИЕ А Листинги	48

Введение

Процесс проведения предвыборной-агитационной кампании в Российской Федерации является критически важным для любой политической единицы и структуры: от ее результата зависит дальнейшее положение во власти и место в системе.

Каждое специализированное программное решение было разработано и продолжает модифицироваться специально для конкретной политической единицы. Причем доступ предоставляется только для членов той или иной политической организации.

Интерфейс программных решений позволяет при наличии соответствующего уровня доступа получить нужную выгрузку данных о том или ином мандате.

Все существующие решения не имеют категоризации данных в динамическом режиме, а позволяют совершать операции выгрузки по заранее продуманным шаблонам, что не позволяет получить ответ на конкретный вопрос о том или ином срезе мандата в определенных промежутках времени.

А именно срез данных для проведения кампаний является ключевым для получения мест в органах законодательной и исполнительной власти путем элективных процессов.

Цель данной работы – программное обеспечение (далее ES - Election Statistics) для хранения и категоризации данных электората императивного мандата.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

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

6) провести исследование эффективности.

1 Аналитический раздел

1.1 Формализация задачи

Каждый мандат имеет свои специфические особенности, которые приходится учитывать избирающимся кандидатам: климат, рельеф, этнос, культура и национальные принадлежности. Не всегда кандидат на выборный пост является жителем региона, в котором выбирается, зачастую, это связано с ротацией кадров по уровню квалификации профессиональных навыков. Именно ротация позволяет выровнять баланс между руководящими составами и их командами в субъектах Российской Федерации от муниципальных образований до федеральных субъектов.

Зачастую сформированные годами на территории мандата правила предвыборных кампаний оказываются провальными, хотя выработанные техники укладываются в специфические особенности субъекта. Это связано с тем, что агрегация данных не происходит с достаточным уровнем динамического плюрализма по направлениям. Только явно категоризированные данные могут однозначно показать результат транспарентности в избирательной кампании.

Изменение особенностей электората мандата может быть связано как с его перемещением (динамический/перемещающийся мандат), но в большей части с реакцией текущего руководящего состава на происходящие социально-экономические микро и макро конфликты в обществе.

А именно такие данные можно категоризировать различными технологиями и методами, к тому же только эти данные можно получать в автоматическом или полуавтоматическом режиме, взаимодействуя посредством открытого/полуоткрытого АПИ с федеральными органами власти, крупнейшими рекрутинговыми компаниями и интегрированными некоммерческими организациями в систему.

Самой малой единицей политического процесса нашей страны является гражданин Российской Федерации, поэтому именно его надо брать в качестве минимальной единицы представления данных, именно данные, связанные с ним, нужно агрегировать о удобным для политический технологий представлять.

Для решения социально-экономических микро и макро конфликтов необ-

ходимо представлять гражданина со всей информацией, не запрещенной к хранению и дальнейшему использованию, содержащейся в его паспорте, полных данных о месте его работы и уровне образования. В дальнейшем такую модель можно автоматически расширять путем дополнения данными об интересах пользователя и его увлечениях с помощью АПИ социальных сетей.

Кроме того, имея динамически изменяющийся срез данных об электорате текущее руководство субъекта, пришедшее выборным путем, может выбирать подходящее время настроений для тех или иных серьезных решений.

1.2 Краткий анализ аналогичных решений

Краткий анализ аналогичных решений приведен в Таблице 1.2, из которого видно, что существующие решения не решают точных задач получения быстрого ответа от динамически обновляемой базы для решения политических вопросов.

Таблица 1.2

Название	Применимость	Доступность	Реализация
ЕСУДЕР (Единая Система Управления Данными Единой России)	Политическими технологиями, привязанными к кандидату	Система доступна только члену партии ЕР	WEB-интерфейс в почтовом клиенте
ИИМ ЛДПР (Институт Исследования Мнений)	Общественным деятелям партии	Система доступна только члену партии ЛДПР	Определенная страница с кнопками выбора в ЛК члена партии
ISSEK РОССТАТа	Неавтоматизированная система агрегации собранных в ручном режиме данных	Система доступна только сотрудникам РОССТАТа во внутреннем контуре	WEB-интерфейс с набором таблиц, экспортируемых в Excel без возможности настроек выбора данных

1.3 Структура паспорта гражданина

Основываясь на условии нашей задачи, наиболее корректным способом хранения данных выглядит представление их с привязкой к конкретному гражданину.

К тому же, такая модель позволит в дальнейшем создавать такие объекты как группы граждан и работать с ними, минимизирую скорость аналитического запроса.

Структура паспорта гражданина - это весь набор информации, который хранится в бумажной версии паспорта, и накладывающиеся на них ограничения. В нашем случае, №152-ФЗ запрещает хранить серию и номер паспорта без особого на это разрешения.

1.4 Базы данных и системы управления базами данны

В задаче кластеризации информации социальных данных важную роль имеет выбор модели хранения данных [1]. Для персистентного хранения данных используются базы данных. Для управления этими базами данных используется системы управления данных – СУБД. 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 гражданина

Система, разрабатываемая в рамках курсового проекта, предполагает собой приложение, которое является микро сервисом одной большой системы. Предполагается, что доступ к разрабатываемому приложению будем иметь лишь только «ядро» этой системы [2]. При этом, только у одного типа пользователя системы есть доступ к данным, хранящимся в приложении – Super User (Далее SU). Состояние гонки [3] (англ. Race condition) можно исключить - каждый SU работает только с информацией из файлов, которые он самостоятельно загрузил в базу данных. Для хранения данных о гражданах необходимо использовать строго структурированную и типизированную базу данных, потому что вся информация, предоставленная в файлах программы имеет четко выраженную структуру.

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. анных.

Строковые

Строковыми базами данных называются такие базы данных, записи которых в памяти представляются построчно. Строковые баз данных используются в транзакционных системах (англ. OLTP). Для таких систем характерно большое количество коротких транзакций с операциями вставки, обновления и удаления данных - INSERT, UPDATE, DELETE. Основной упор в системах OLTP [4] 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 данных.

Колоночные

Колоночными базами данных называются базы данных, записи которых в памяти представляются по столбцам. Колоночные базы данных используется в аналитических системах (англ. OLAP). OLAP [5] 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 Обзор СУБД с построчным хранением

Существует множество СУБД с построчным хранением, каждая из них имеет свои преимущества и недостатки. Для того, чтобы принять наиболее

взвешанно решение о выборе конкретной СУБД, надо рассмотреть некоторые из них.

PostgreSQL

PostgreSQL [6] – это свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

PostgreSQL [7] предоставляет транзакции со свойствами атомарности, согласованности, изоляции, долговечности (ACID [8]), автоматически обновляемые представления, материализованные представления, триггеры, внешние ключи и хранимые процедуры. Данная СУБД [9] предназначена для обработки ряда рабочих нагрузок, от отдельных компьютеров до хранилищ данных или веб-сервисов с множеством одновременных пользователей.

Рассматриваемая СУБД [8] управляет параллелизмом с помощью технологии управления многоверсионным параллелизмом (англ. MVCC). Эта технология дает каждой транзакции «снимок» текущего состояния базы 10 данных, позволяя вносить изменения, не затрагивая другие транзакции. Это в значительной степени устраняет необходимость в блокировках чтения (англ. read lock) и гарантирует, что база данных поддерживает принципы ACID [9].

Oracle DB

Oracle Database [10] – объектно-реляционная система управления базами данных компании Oracle. На данный момент, рассматриваемая СУБД является самой популярной в мире.

Все транзакции Oracle Database соответствуют обладают свойствами ACID, поддерживает триггеры, внешние ключи и хранимые процедуры. Данная СУБД подходит для разнообразных рабочих нагрузок и может использоваться практически в любых задачах. Особенностью Oracle Database является быстрая работа с большими массивами данных.

Oracle Database [13] 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 [14] – свободная реляционная система управления базами данных. Разработку и поддержку MySQL [11] осуществляет корпорация Oracle. Рассматриваемая СУБД имеет два основных движка хранения данных: InnoDB и myISAM. Движок InnoDB полностью полностью совместим с принципами ACID, в отличие от движка myISAM [12]. СУБД MySQL [13] подходит для использования при разработке веб-приложений, что объясняется очень тесной интеграцией с популярными языками PHP [14] и Perl [15]. Реализация параллелизма в СУБД MySQL реализовано с помощью механизма блокировок, который обеспечивает одновременный доступ к данным.

1.5.3 Выбор СУБД для решения задачи

Из-за специфики задачи о постоянных аналитических запросах, обновлению данных и необходимой отзывчивости построчное хранение преобладает над колоночным.

Специфика задачи для нас - это то, что количество полей, характеризующих гражданина значительно меньше, чем самих граждан.

Для решения задачи была выбрана СУБД PostgreSQL [20] по причине наличия бесшовной интеграции с некоторыми библиотеками языка программирования Python, которые позволяют реализовывать SQL-запросы.

Для решения задачи также можно было выбрать Java/PHP, которые также имеют библиотеки с бесшовной интеграцией, но в рамках курсового проекта именно Python3 позволит в короткие сроки реализовать MVP продукта, на котором можно провести исследования и в дальнейшем доработки.

1.6 Кэширование данных

В связи с большим объемом хранимых данных, по которым будут проводиться аналитические запросы, уже сейчас необходимо задумать о скорости отклика системы, то есть проведения запросов. Для решения этой задачи существует множество вариантов, самые популярные из них - это надстройка над таблицами и кэширование. Первое - это вполне хорошее решение, но для

корпоративных систем, а второе вполне не сложно в реализации и имеет бесшовную интеграцию с библиотеками языка программирования Python. Для такой операции надо использовать нереляционные базы данных, которые хранятся в оперативной памяти, что ускорит процесс проведения аналитических запросов по большим данным.

1.6.1 Проблемы кэширования данных

В крупных системах происходит сложное кэширование, распределенное по слоям, где для каждого слоя правила кэширования устанавливаются разные, но как правило процесс для всех слоев запускается одновременно, что приводит к неминуемому краху системы кэширования и увеличению времени отклика системы.

Синхронизация данных

Необходимо всегда следить за синхронизацией данных, записанные в хранилище кэширования и основной базой данных, чтобы следовать правилу единственности. Для того, чтобы решить эту задачу, необходимо создать набор триттеров (CRUD), нивелирующих такую проблему.

Проблема холодного старта

Когда кэш только разворачивается, 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 [21] 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

NoSQL СУБД - обозначение широкого класса разнородных систем управления базами данных, появившихся в конце 2000-х — начале 2010-х годов и существенно отличающихся от традиционных реляционных СУБД с доступом к данным средствами языка SQL. Применяется к системам, в которых делается попытка решить проблемы масштабируемости и доступности за счёт полного или частичного отказа от требований атомарности и согласованности данных.

Резидентные СУБД за счёт оптимизаций, возможных в условиях хранения и обработки в байтоадресуемой оперативной памяти, обеспечивают лучшее быстродействие, чем СУБД, работающие с базами данных на устройствах постоянного хранения, как правило, с блочной организацией, и подключаемых по шинным или сетевым интерфейсам. При этом размер резидентной базы данных ограничен ёмкостью оперативной памяти узла. Для ряда резидентных СУБД реализуются техники репликации и сегментирования (англ. sharding), позволяющие работать с единой резидентной базой данных на нескольких узлах. Поскольку оперативная память энергозависима, то используется запись с предварительным журналированием на энергонезависимом устройстве для обеспечения целостности базы данных при внезапной перезагрузке, то есть, работа с резидентной базой не исключает зависимости от производительности подсистемы ввода-вывода (хотя и снижает её).

Широко применяются для приложений, где время отклика имеет решающее значение, в частности, в задачах управления телекоммуникационным оборудованием, для торгов в реальном времени. В качестве эффективных сценариев для применения резидентных баз данных отмечаются аналитика в реальном времени и гибридная транзакционно-аналитическая обработка.

Tarantool

Tarantool [16] – 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 [17].

Tarantool обладает высокой скоростью работы по сравнению с традицион-

ными СУБД. При этом, в рассматриваемой платформе для транзакций реализованы свойства ACID, репликация master-slave [18] и master-master [19], как и в традиционных СУБД.

Для хранения данных используется кортежи (англ. tuple) данных. Кортеж – это массив нетипизированных 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 и vinyl [20]. 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 [21] – 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 доступные для него; поддерживаются высокоуровневые операции: например, объединение, разность или сортировка наборов.

1.6.3 Формализация данных и выбор СУБД

База данных граждан Российской Федерации должна хранить непосредственно информацию о них. Информация, которая должна храниться в базе данных описана в разделе 1.3 Каждый гражданин должен обладать уникальным идентификатором, чтобы его/ее можно было однозначно идентифицировать.

У каждого гражданина имеется свод характеристик, такие как паспортные данные, данные об образовании и другие.

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 Вывод

В данном разделе:

- рассмотрена структура паспорта гражданина;
- 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;
- формализованные данные, используемые в системе.

2 Конструкторский раздел

2.1 Проектирование архитектуры и верхне-уровневой логики

Для того, чтобы грамотно реализовать приложение, в первую очередь необходимо спроектировать архитектуру.

Проектируемая архитектура должна учитывать разделенные базы пользователей и данных системы, а также репликацию данных граждан.

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

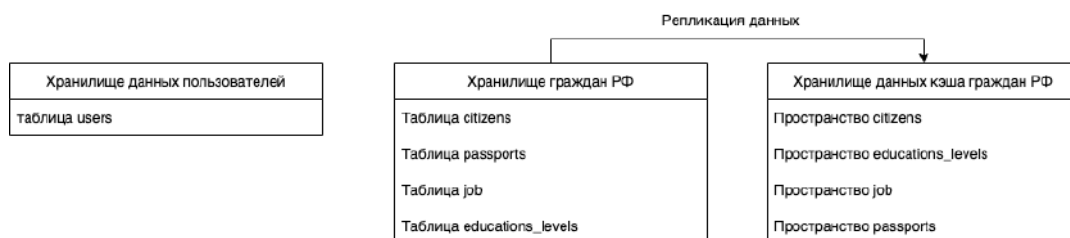


Рисунок 2.1 – Схема сущностей WEB-приложения ES.

Пользователь приложения делится на три уровня доступа, каждый из которых разрешает более высокому уровню выполнять свой набор операций (см. Таблицу ??)

Таблица 2.1 – Возможности от уровня доступа пользователей

Название	Пользователь	Менеджер	Администратор
Просмотр среза данных (Дашборд)	Да	Да	Да
Обновление данных пользователей	Нет	Да	Да
CRUD всех данных	Нет	Нет	Да

Также более наглядно посмотреть роли пользователей и их назначения можно на use-case диаграмме (см. Рисунок 2.2

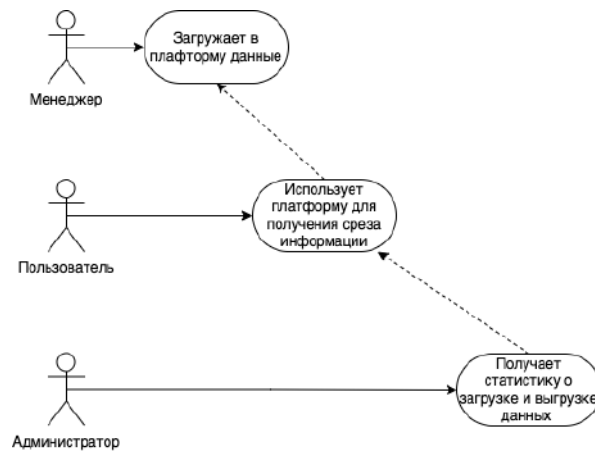


Рисунок 2.2 – use-case диаграмма

2.2 Проектирование базы данных пользователей системы

База данных пользователей системы будет реализована с помощью СУБД PostgreSQL. В базе данных будет существовать одна таблица <users> (см. Рисунок 2.3).

Таблица специально вынесена в отдельную базу данных, чтобы в дальнейшем разделить потоки операций на основную функцию приложения в виде аналитических статистических запросов и сопутствующие в виде хранимых данных для непосредственной работы с личным кабинетом, внутреннем чатом, прикрепленными данными пользователей и тд.

Поля таблица users означают:

- id - уникальных идентификатор пользователя в системе;
- login - имя пользователя в системе, используемое в сопутствие с полем password для авторизации в системе;
- password - пароль пользователя в системе, используемое в сопутствие с полем login для авторизации в системе;
- acslevel - уровень доступа пользователя в системе. Отвечает за описанные роли в разделе 2.1.

Users	
PK	<u>id</u>
	login
	password
	acs_level

Рисунок 2.3 – Таблица users

2.3 Проектирование базы данных граждан страны

База данных пользователей системы будет реализована с помощью СУБД PostgreSQL. В базе данных будет существовать 6 таблиц и 4 сущности (см. Рисунок 2.4).

Поля таблица citizen означают:

- citizenID - уникальных идентификатор гражданина страны в системе;
- educationsLevelsStatusesList - список сущностей полученного образования;
- tempEducationStatus - текущий статус образования;
- jobStatusesList - список сущностей рабочих позиций;
- tempJobStatus - текущий статус рабочей позиции.

Данная таблица citizen являются одной из ключевых, потому что она будет служить для решения второй по важности задачи приложения - демонстрации статистических данных от сложных наборов правил к каждому гражданину.

Поля таблица passports означают:

- passportID - уникальных идентификатор паспорта гражданина страны в системе;
- name - имя;
- surname - фамилия;
- patronymic - отчество;

- registration - регион рождения;
- childrenNumber - количество детей;
- childerIDLnownList - список известных детей;
- isMarriedNow - текущий статус брака;
- marrigesIDList - список браков;
- birthDate - дата рождения.

Данная таблица passports являются ключевой, потому что она будет служить для решения самой главной задачи приложения - получение точного демографического и социального статистического среза.

К примеру, депутат Яценко из небольшого регионального города Х. желает продолжить двигаться по карьерной лестнице и переехать в столицу Ж., где собирается избираться от муниципального округа Б.. Она ничего не знает о месте, а главное людях этого региона, поэтому ей просто необходимо получить срез статистики по точному запросу “Регион Б:Возраст:ПродолжительностьЖизни:Смертность:ЗП:... и тд.”, после чего она, подготовившись к беседе с жителями, проведет стандартную работу с населением и успешно изберется.

Поля таблица EducationsLevels означают:

- EducationLevelD - уникальных идентификатор образования гражданина;
- school - школа;
- college - колледж;
- bacholor - бакалавр;
- magistrty - магистратура;
- aspirantury - аспирантура;

Данное разделение на относительно небольшие таблицы сделано из опыта на основе выполнения лабораторных работ курса по базам данных Гавриловой

Юлии Михайловны: при относительно большом объеме данных (>500 000 строк), которые находятся в режиме постоянного чтения, база данных начинает работать значительно медленнее. Поэтому решено выделять отдельные сущности, по каждой из которых производить статистические запросы.

Поля таблица `EducationsStatuses` означают:

- `EducationLevelID` - неуникальный идентификатор статуса образования;
- `finished` - закончено;
- `unfinished` - не закончено;
- `impregnable` - брошено;

Данная таблица создана для того, чтобы в любой момент изменения наших правил Болонской системы образования изменить: Добавить/Удалить необходимые/ставшие излишними статусы образования.

Поля таблица `Job` означают:

- `JobID` - уникальный идентификатор рабочей позиции конкретного гражданина;
- `company-name` - компания;
- `position-name` - позиция;
- `position-salary` - заработная плата;
- `date-start` - дата начала работы;
- `date-end` - дата окончания работы;
- `job-status` - статус работы;

Данное разделение произведено по тому же правилу, что и с таблицами образования. К примеру, в нашей базе данных 25.000.000 граждан, каждый из которых работал на 2.5 работах, а это означает, что записей о работе уже более 60.000.000, и для простого поиска невозможного/нелогичного для кэширования запроса придется ожидать несколько минут, поэтому выделяем в отдельную таблицу данных о работе. И в будущем, возможно, делим их по

курсам $Job1[1, 10.000.000], \dots, Jonn[10.000.000*n, 10.000.000*(n+1)]$, чтобы их быстро модифицировать и относительно быстро выполнять поиск.

Поля таблица JobStatuses означают:

- JobStatusID - неуникальный идентификатор паспорта гражданина страны в системе;
- finished - имя;
- not finished - фамилия;

Данная таблица необходима по причине, аналогичной причине в статусе образования.

Таблица citizen и passport имеет отношение один-к-одному, так как любой гражданин имеет одни данные от рождения.

Таблица citizen и educationsLevels имеет отношение один-ко-многим, потому что многие граждане имеют несколько различных видов образования.

Таблица citizen и Job имеет отношение один-ко-многим, потому что многие граждане имеют несколько мест работ.

Таблица educationsLevels и JobStatuses имеет отношение многие-ко-многим, для того, чтобы оптимизировать расширение полей и не хранить излишнюю информацию, то есть не занимать лишнюю память.

Кроме того, для каждой таблицы будет реализован триггер, срабатывающий после UD-операций.

Реализация триггера будет выполнена шагами:

- произошла операция UD, на которую сработал триггер;
- триггер посылает сигнал базе данных кэширования, с помощью `rlpython3u`, о необходимости обновить или удалить информацию из кэша.

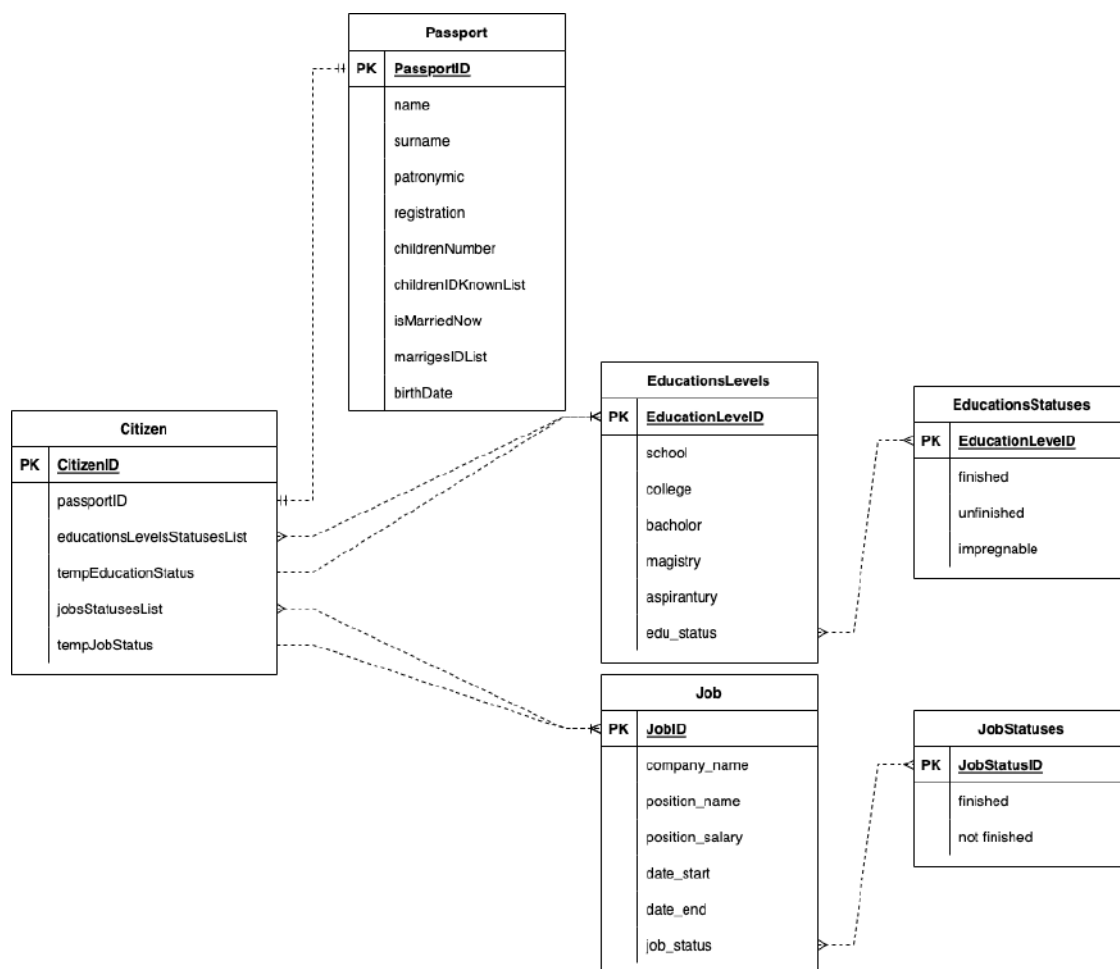


Рисунок 2.4 – ER диаграмма БД ES

2.4 Проектирование бизнес-логики и работы интерфейса

Чтобы правильно реализовать бизнес-логику - необходимо формализованную задачу предвыборных кампаний трансформировать в нормализованный вид.

На Рисунке 2.5 приведена UML-диграмма базового модуля бизнес-логики WEB-приложения ES.

Интерфейс системы может быть "раздут для этого необходимо выделить основные его элементы.

На Рисунке 2.6 приведен пример необходимой реализации интерфейса WEB-приложения ES.

На основе полученного интерфеса необходимо получить связь его элементов: конкретные правила взаимодействия.

На Рисунке 2.7 приведена диаграмма для этого примера WEB-приложения

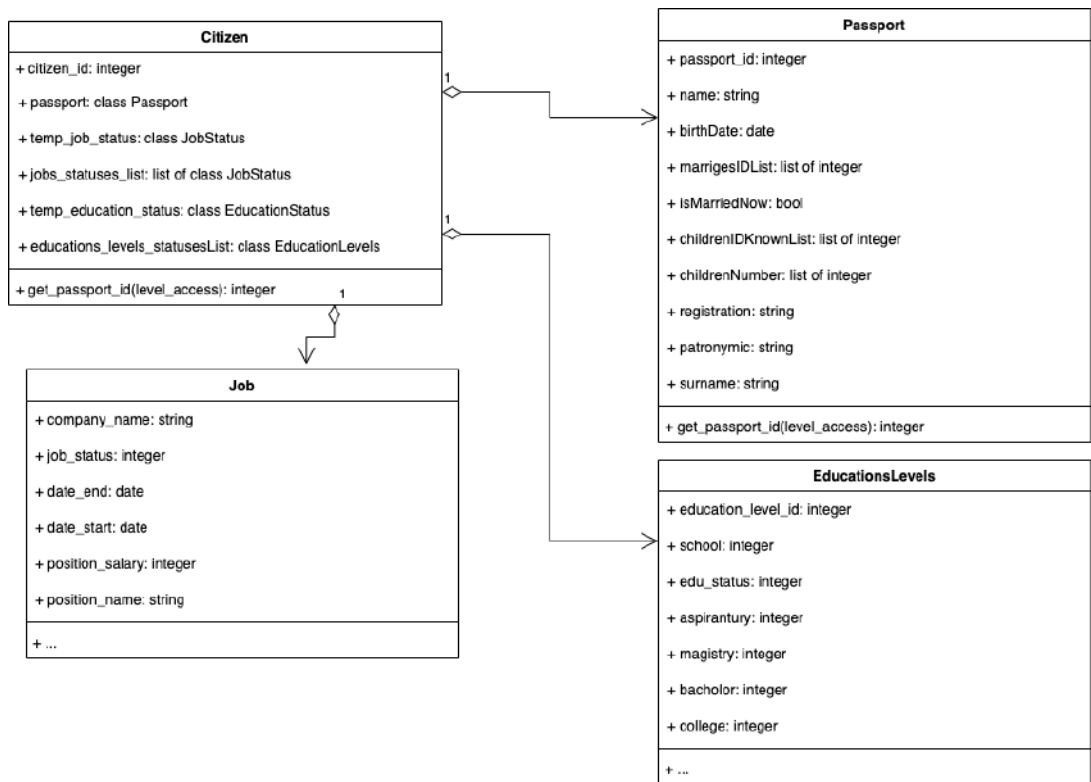


Рисунок 2.5 – UML базового модуля бизнес логики

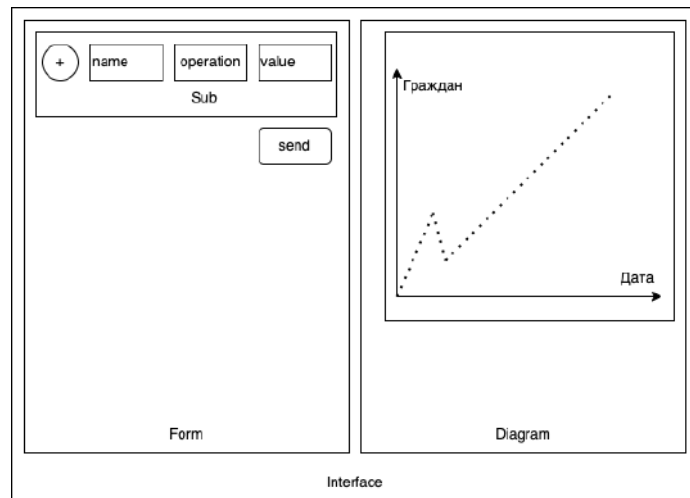


Рисунок 2.6 – Пример необходимой реализации интерфейса

ES.

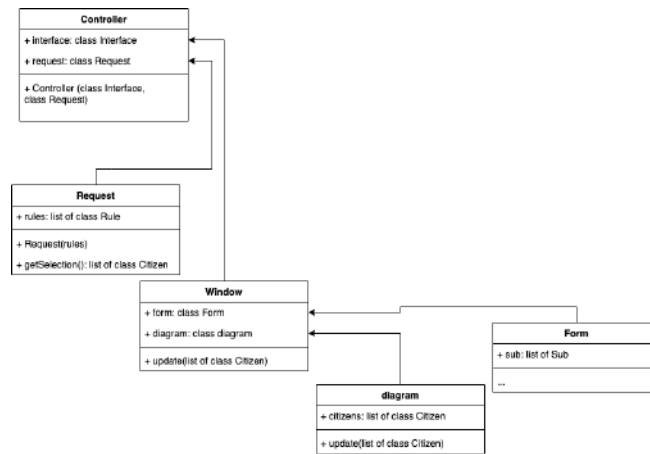


Рисунок 2.7 – UML примера необходимой реализации интерфейса

2.5 Проектирование базы данных кэширования

База данных кэширования будет реализована с помощью использования СУБД Tarantool, в ней будут полностью продублированы таблицы (в виде спейсов) из хранилища ES. Первичным ключом будет являться поле с уникальным идентификатором этих таблиц (id). Кроме того, для спейсов хранящих поле passportID будет добавлен вторичный ключ по этому полю, для удобного и быстрого сбора нужных данных по заданному паспорту.

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

Все пространства будут созданы на основе 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.

3 Технологический раздел

3.1 Архитектура WEB-приложения

Данное приложение является набором из двух микросервисов: один для работы с данными пользователей, другой для работы с данными граждан. Доступ к данным, хранящимся в приложении, получен с помощью REST API. С базами данных взаимодействует серверная часть с помощью специальных коннекторов, позволяющих реализовывать запросы в тексте программы. Общая схема архитектуры приложения представлена на Рисунке 3.1

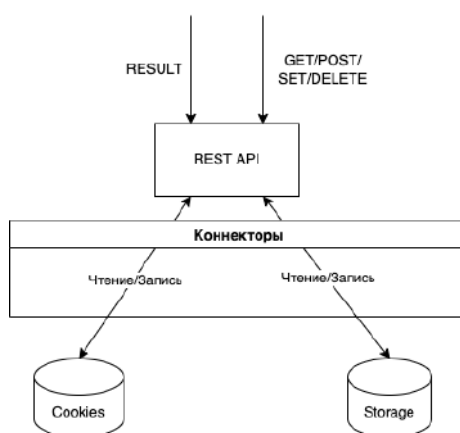


Рисунок 3.1 – Архитектура

3.2 Средства реализации

Для разработки серверной части был выбран язык программирования Python из-за удобной поддержки развертывания REST-приложений, интеграции с СУБД PostgreSQL. Язык Python имеет коннекторы для платформы in-memory вычислений Tarantool. Для реализации REST API был выбран фреймворк Flask.

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

Для упаковки приложения в готовый продукт была выбрана система контейнеризации Docker. С помощью Docker, можно создать изолированную среду для программного обеспечения, которое можно будет развернуть на раз-

личных операционных система без дополнительного вмешательства для обеспечения совместимости.

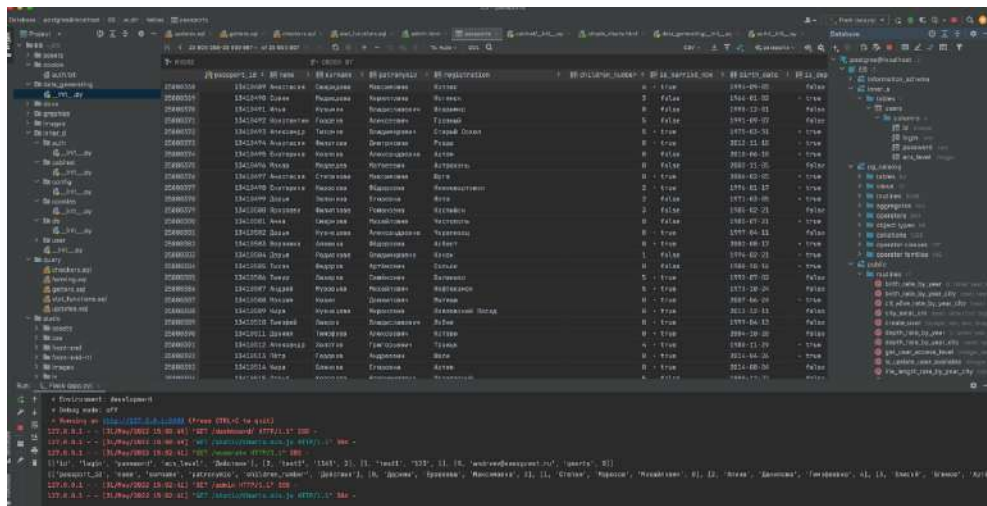
Тестирование программного продукта производилось с помощью фреймворка `pytest`. Данный фреймворк позволяет писать как модульные, так и функциональные тесты. Для тестирования ПО был реализован ряд функциональных тестов.

3.3 Генерация исходных данных

Для решения задачи с помощью собственного модуля генерации было сгенерировано 25.000.000 граждан (см. Рисунок 3.2), такое большое количество записей было сделано не случайно, была цель - подтвердить гипотезы по поводу кэширования и выбранном архитектуры.

Вполне можно было выполнить генерацию одного или двух миллионов граждан, но ради исследования было взято число значительно большее.

Такое решение принято для того, чтобы убедиться в яном преимуществе методов кэширования над его отсутствием.



The screenshot displays a software interface for data generation. It features a sidebar on the left with a tree view of data categories. The main area shows a large table with multiple columns containing generated data, including names, IDs, and other attributes. The table is populated with numerous rows, illustrating the scale of the generated dataset.

Рисунок 3.2 – Объем генерации данных

3.4 Детали реализации

В Приложении А представлено взаимодействие клиента с сервером, обработка его запросов, взаимодействие приложения с базами данных и кэширо-

вание данных, модуль кэширования данных с политикой вытеснения LRU, взаимодействие с Tarantool.

LRU (least recently used) — это алгоритм, при котором вытесняются значения, которые дольше всего не запрашивались. Соответственно, необходимо хранить время последнего запроса к значению. И как только число закэшированных значений превосходит N необходимо вытеснить из кеша значение, которое дольше всего не запрашивалось.

Для реализации этого метода нам понадобятся две структуры данных:

- Хеш-таблица `hashTable`, которая будет хранить непосредственно закэшированные значения.
- Очередь с приоритетами `timeQueue`. Структура, которая поддерживает следующие операции:
 - Добавить пару значение и приоритет `timeQueue.Add(val, priority)`.
 - Извлечь (удалить и вернуть) значение с наименьшим приоритетом `timeQueue.extractMinValue()`.

3.5 Взаимодействие с приложением

3.5.1 Авторизация

С приложением необходимо взаимодействовать через Web-интерфейс, который реализован на голем HTML с готовой библиотекой `EduTem` для вывода графиков в CSS/JS.

У приложения существует возможность выбора в меню приложений страницы сводной аналитики, страницы модерирования или администрирования данных при наличии соответствующего уровня доступа.

На каждой странице, открытой из пункта меню БД отдает нужный набор функций для уровня доступа пользователя.

Первым делом пользователя встречается страница авторизации, где ему необходимо ввести свои данные или зарегистрироваться (см. Рисунок 3.3)

Элемент развернут из надстроки EduTem для bootstrap4 и реализован на стороне БД: функция верификации наличия пользователя в системе и доступ к авторизации.

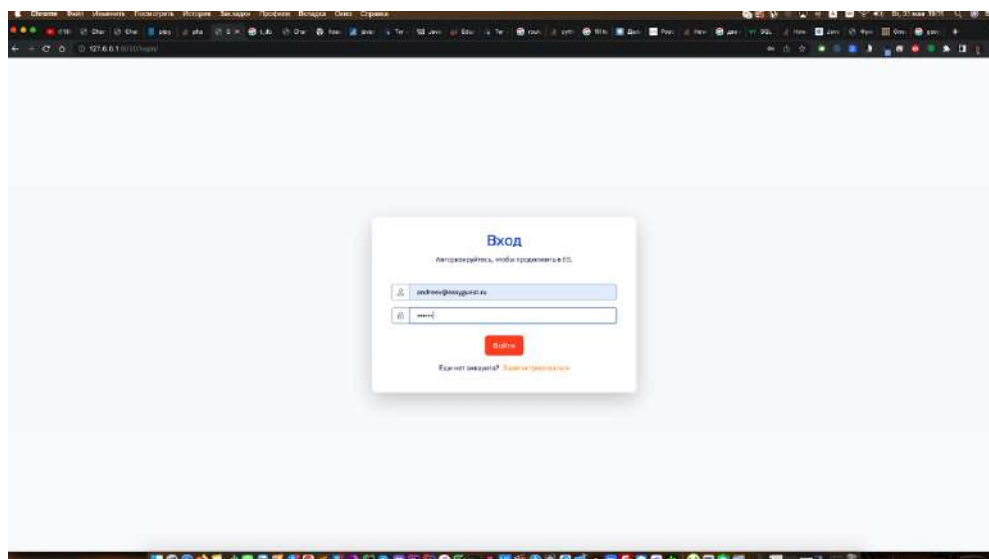


Рисунок 3.3 – Авторизация

3.5.2 Меню интерфейса

В качестве меню интерфейса (см. Рисунок 3.4) пользователю доступны три возможности: Просмотр дашборда, Выход из аккаунта и Управление данными (Доступ открыт только пользователям, чей уровень доступа ≤ 1 (см. Рисунок 3.5))

Изначально, было решение развернуть полноценный bootstrap4 с отлаженным интерфейсом, но по причине конкретной задачи по исследованию кэширования было решено использовать "голый"html/csss.

[Статистика](#) [Выход из аккаунта](#) [Управление данными](#)

Рисунок 3.4 – Меню WEB-приложения ES

	id	login	password	acs_level
1	2	test2	1145	2
2	1	test1	123	1
3	0	andreev@easyguest.ru	qwerty	0

Рисунок 3.5 – Выделенные уровни доступа

3.5.3 Дашбоард

На странице дашборда пользователем всех уровней доступны статистики и возможность их сконфигурировать.

Вывод всей статистики

По умолчанию при загрузке страницы у пользователя появляется вывод всей статистики (см. Рисунок 3.6).

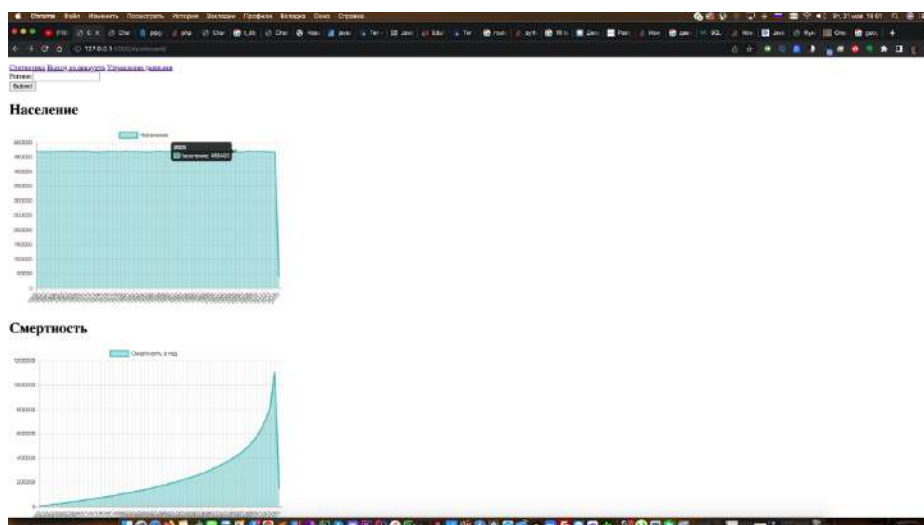


Рисунок 3.6 – Вывод всей статистики

Вывод региона

При вводе региона дашборд обновляется и показывает соответствующую информацию (см. Рисунок 3.7).



Рисунок 3.7 – Вывод статистики по региону

3.5.4 Страница модератора

На странице модератора пользователь с правами модератора может обновлять данные других пользователей и граждан (см. Рисунок 3.8).



Рисунок 3.8 – Страница управления данными модератора

К примеру, модератору необходимо принятому на работу сотруднику выдать новые права, то есть с уровня простого пользователя поднять его до уровня модератора, тогда он выбирает пользователя и изменяет данные (см. Рисунок 3.9), после нажатия обновить, он получает сообщение об успешной операции (см. Рисунок ??).



Рисунок 3.9 – Страница управления данными модератора, модератор решил обновить данные



Рисунок 3.10 – Страница управления данными модератора, модератор успешно обновил данные

3.5.5 Страница администратора

На странице администратора пользователь с правами администратора может обновлять/удалять/добавлять данные других пользователей и граждан (см. Рисунок 3.11).



Рисунок 3.11 – Страница администратора

Так, к примеру, администру звонит на горячую линию пенсионер, который не может разобраться с процессом авторизации и просит ему помочь это сделать. Администратора открывает одноименную панель и вводит данные пользователя (см. Рисунок 3.12), высылая его логин и пароль, к примеру, на почту. Если у него произошла ошибка (пользователь с такими данными существует), то на его экран выведется соответствующее сообщение (см. Рисунок ??), но если же все хорошо и пользователь создан, то администратора также

получит сообщение (см. Рисунок 3.14).



Рисунок 3.12 – Страница управления данными администратора, администратор вводит данные нового пользователя



Рисунок 3.13 – Страница управления данными администратора, пользователь уже существует



Рисунок 3.14 – Страница управления данными администратора, администратор успешно добавил нового пользователя

Если же администратору необходимо удалить данные, то он с легкостью введет название таблицы и ID строки и выполнить удаление, статус об успешном или неуспешном удалении он также получит сообщение (см. Рисунок ??).

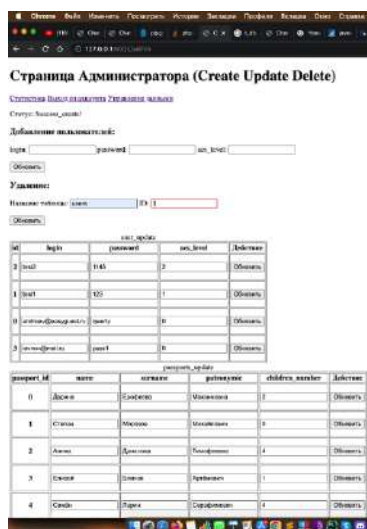


Рисунок 3.15 – Страница управления данными администратора, администратор удаляет запись

При безконфликтном удалении, Администратор получит соответствующее сообщение (см. Рисунок ??).



Рисунок 3.16 – Страница управления данными администратора, администратор успешно удалил запись

4 Экспериментальный раздел

В данном разделе представлена постановка эксперимента по сравнению занимаемого времени для получения данных из хранилища с использованием и без использования кэширования. Для того, чтобы грамотно сравнить время, был принудительно включен механизм кэширования: была отключена база данных, хранящая данные о кэшировании и каждый раз выполняется запрос к базе данных, хранящих информацию о гражданах.

4.1 Исследование зависимости времени получения данных без кэширования и с кэшированием

Задача эксперимента - наблюдение зависимости изменения времени генерации отрисовки сцены от количества компонентов детали. Точные результаты исследования представлены в Таблице ??. Графическое представление таблицы доступны на 4.2.

Таблица 4.1 – Зависимости времени получения данных без кеширования и с кешированием, кэш - 100 ед

РПД, экз.	С кэширование, мс	Без кэширования, мс
1	52012	3554
5	366500	340250
10	733000	680500
25	1622500	1101250
30	2199000	2041500
45	3298500	3062250
60	4398000	4083000
65	4764500	4423250
80	5864000	5854000
95	6963500	7104750
100	7330000	7505000

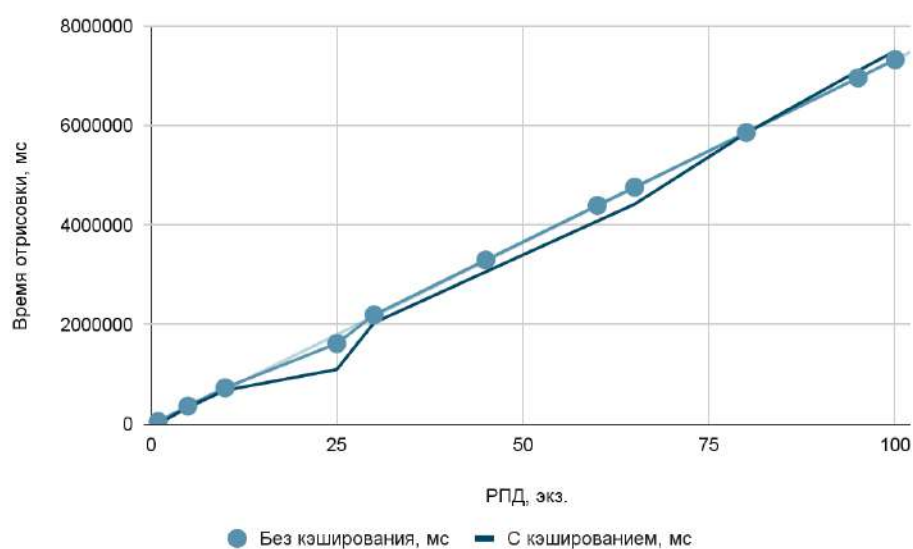


Рисунок 4.1 – График зависимости времени получения данных без кеширования и с кешированием, кэш - 100 ед.

Таблица 4.2 – Зависимости времени получения данных без кеширования и с кешированием, кэш - 1000 ед.

РПД, экз.	С кэширование, мс	Без кэширования, мс
1	52012	3254
5	366500	4100
10	733000	12131
25	1622500	513290
30	2199000	534410
45	3298500	1518912
60	4398000	4083000
65	4764500	4423250
80	5864000	4943250
95	6963500	5834582
100	7330000	6124582

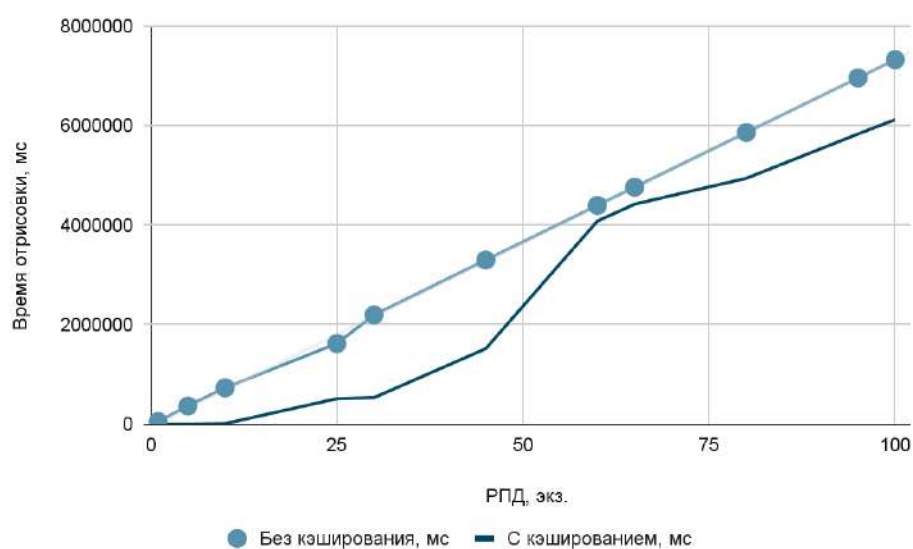


Рисунок 4.2 – График зависимости времени получения данных без кеширования и с кешированием, кэш - 1000 ед.

Таблица 4.3 – Зависимости времени получения данных без кеширования и с кешированием, кэш - 5000 ед.

РПД, экз.	С кэширование, мс	Без кэширования, мс
1	52012	7521
5	366500	10193
10	733000	42227
25	1622500	109710
30	2199000	131652
45	3298500	197478
60	4398000	263304
65	4764500	285246
80	5864000	351072
95	6963500	416898
100	7330000	552840

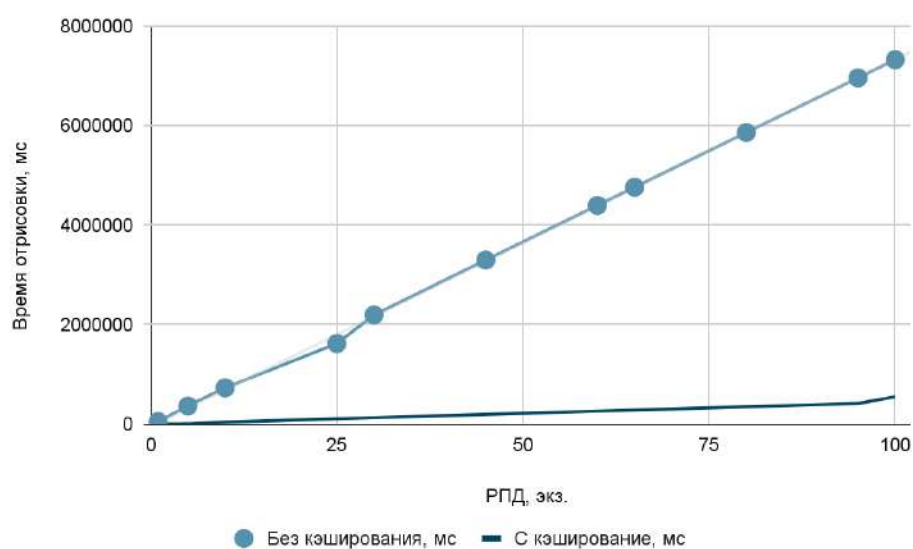


Рисунок 4.3 – График зависимости времени получения данных без кеширования и с кешированием, кэш - 5000 ед.

4.2 Вывод

Исследование оказалось неоднозначным:

- Использование кэширование позволяет работать с данными быстрее;
- Приложение с кэширование выигрывает по времени у приложения без него чуть более, чем в один раз, когда запросы РПД превышают максимальный размер кэша;
- В противном случае, когда кэш значительно больше запроса РПД, выигрыш по времени составляет в среднем 19 раз;

Оказалось, что в разработанном приложении эффективность кэширования зависит от размера хранимых данных. Но хранить весь объем данным с учетом того, что таблица `passports` на 25 млн. записей в кэше весит почти 3 ГБ неправильно и никто на такое не пойдет, поэтому наиболее выгодным решением выглядит хранить хотя бы четверть данных в кэше, что обеспечит ускорение более чем в три раза.

Заключение

В ходе работы над курсовым проектом были решены все поставленные задачи: разработано программное обеспечение для хранения, редактирования и удаления, а главное сбора статистической информации о гражданах страны по конкретному региону, что дало возможность получить дополнительные знания в области проектирования баз данных и кэширования данных, различных СУБД, в том числе их целей и возможностей. В результате проведенной работы исследовательским путем было получено, что приложение с кэшированием работает значительно быстрее, чем без него. Также исследование показало, что внедрение кэширования позволяет оптимизировать скорость работы до 19 раз.

В дальнейшем программа может быть модернизирована за счёт добавления новых сущностей, а также с использованием оптимизации запросов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 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: 11.05.2022).
2. Что такое база данных | Oracle Россия и СНГ [Электронный ресурс]. Режим доступа: <https://www.oracle.com/ru/database/what-is-database/> (дата обращения: 11.05.2022).
3. Что такое СУБД - RU-CENTER [Электронный ресурс]. Режим доступа: https://www.nic.ru/help/chto-takoe-subd_8580.html (дата обращения: 11.05.2022).
4. 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: 11.05.2022).
5. 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: 11.05.2022).
6. What is OLTP? | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/cloud/learn/oltp> (дата обращения: 11.05.2022).
7. What is OLAP? | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/cloud/learn/olap> (дата обращения: 11.05.2022).
8. PostgreSQL: Документация. [Электронный ресурс]. Режим доступа: <https://postgrespro.ru/docs/postgresql/> (дата обращения: 11.05.2022).
9. PostgreSQL: вчера, сегодня, завтра [Электронный ресурс]. Режим доступа: <https://postgrespro.ru/blog/media/17768> (дата обращения: 11.05.2022).
10. Documentation: 12: 13.1. Introduction - PostgreSQL [Электронный ресурс]. Режим доступа: <https://www.postgresql.org/docs/12/mvcc-intro.html> (дата обращения: 11.05.2022). [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: 11.05.2022).
11. Транзакции, ACID, CAP | GeekBrains [Электронный ресурс]. Режим доступа: https://gb.ru/posts/acid_cap_transactions (дата обращения: 11.05.2022).
12. SQL Language | Oracle [Электронный ресурс]. Режим доступа: <https://www.oracle.com/database/technologies/appdev/sql.html> (дата обращения: 11.05.2022).
13. Oracle | Integrated Cloud Applications and Platform Services [Электронный

- ресурс]. Режим доступа: <https://www.oracle.com/index.html> (дата обращения: 11.05.2022).
14. DB-Engines Ranking [Электронный ресурс]. Режим доступа: <https://db-engines.com/en/ranking> (дата обращения: 11.05.2022).
 15. MySQL Database Service is a fully managed database service to deploy cloud-native applications. [Электронный ресурс]. Режим доступа: <https://www.mysql.com/> (дата обращения: 11.05.2022).
 16. 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: 11.05.2022).
 17. 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: 11.05.2022).
 18. PHP: Hypertext Preprocessor [Электронный ресурс]. Режим доступа: <https://www.php.net/> (дата обращения: 11.05.2022).
 19. The Perl Programming Language [Электронный ресурс]. Режим доступа: <https://www.perl.org/> (дата обращения: 11.05.2022).
 20. 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: 11.05.2022).
 21. Что такое NoSQL? | Amazon AWS [Электронный ресурс]. Режим доступа: <https://aws.amazon.com/ru/nosql/> (дата обращения: 11.05.2022).

ПРИЛОЖЕНИЕ А

Листинги

В листинге А.1 приведен код запуска приложения. ,,

Листинг А.1 – App run (app.py)

```
from flask import Flask

def create_app():

    t_app = Flask(__name__)
    t_app.static_folder = 'static'

    t_app.config['SECRET_KEY'] = 'secret-key-goes-here'
    t_app.config['SQLALCHEMY_DATABASE_URI'] =
        'postgresql://localhost/[ES][qwerty]'

    # blueprint for auth routes in our app
    from inner_d.auth import Auth
    Auth.register(t_app, route_base='/')

    # blueprint for auth routes in our app
    from inner_d.cabinet import Cabinet
    Cabinet.register(t_app, route_base='/')

    # blueprint for non-auth parts of app
    from main import main as main_blueprint
    t_app.register_blueprint(main_blueprint)

    return t_app

if __name__ == '__main__':
    app = create_app()
    app.run()
```

В листинге А.2 приведен код взаимодействия клиента и сервера. ,,

Листинг А.2 – Client-Server (package cabinet)

```
import json
from io import StringIO
import graphics
from inner_d import cookies, db, config
from flask import Blueprint, render_template, redirect, url_for,
    request
from flask_classful import FlaskView, route
from inner_d.db import DB
import logging
import psycpg2
import db.models as m
from db.cache.cache import CacheLRU
from db.utils import Utils
import config

class Cabinet(FlaskView):
    _cookies = None
    _db = None
    _cache = None
    def __init__(self):
        self._cookies = cookies.Cookies()
        self._db = DB()
        self._cache = CacheLRU()
    def _get_lab_val(self, arr):
        labels, values = [], []
        for elem in arr:
            labels.append(elem[0])
            values.append(elem[1])
        return labels, values
    @route('/cabinet', methods=['POST'])
    def cabinet(self):
        logging.info(f"- cabinet called")
        if self._cookies.is_auth_opened():
            logging.info(f"- cabinet redirected to dashboard")
            return redirect(url_for('Cabinet:dashboard'))
        cur_t = self._db.conn.cursor()
        usr_login, usr_psd = request.form.get('email'),
            request.form.get('password')
        cur_t.execute(
```



```

        'select id, login, password from inner_s.users where
            login = %s and password = %s',
        (usr_login, usr_psd)
    )
    f_usr = cur_t.fetchall()
    usr_id = f_usr[0][0]
    cur_t.close()

    if len(f_usr) == 1:
        self._cookies.set_auth_opened(usr_id, usr_login)
        logging.info(f"- success login")
        logging.info(f"- cabinet redirected to dashboard")
        return redirect(url_for('Cabinet:dashboard'))
    logging.info(f"- cabinet redirected to login")
    return redirect(url_for('Auth:login'))
@route('/moderate', methods=['POST'])
def moderate_post(self):
    try:
        user_update = request.form.get('user_update')
        user_id, user_login, user_password, user_ac_level =
            request.form.get('id'),
            request.form.get('login'),
            request.form.get('password'),
            request.form.get('acs_level')
        t_db_cur = self._db.conn.cursor()
        t_db_cur.execute(
            "select * from
                is_update_user_available('{usr_id}',
                '{usr_login}',
                '{user_password}');".format(usr_id=user_id,
                usr_login=user_login,
                user_password=user_password))
        if len(t_db_cur.fetchall()) <= 1:
            t_db_cur.execute(
                "select * from update_user('{usr_id}',
                '{usr_login}', '{user_password}',
                '{user_ac_level}');".format(
                    usr_id=user_id, usr_login=user_login,
                    user_password=user_password,
                    user_ac_level=user_ac_level))
        return self.moderate_comp("Success_update!")

```

```

except EOFError:
    print()
    return self.moderate_comp("Error update (Field Error)")
@route('/moderate')
def moderate(self):
    return self.moderate_comp("Data fetched!")
def moderate_comp(self, message):
    if not self._cookies.is_auth_opened():
        return redirect(url_for('Auth:login'))
    usr_id, usr_login = self._cookies.get_auth_cookie()
    t_db_cur = self._db.conn.cursor()
    t_db_cur.execute("select * from
        get_user_access_level('{usr_id}',
        '{usr_login}');".format(usr_id=usr_id,
        usr_login=usr_login))
    lev = t_db_cur.fetchall()[0][0]
    if lev == config.USR_ACS_L:
        return redirect(url_for('Cabinet:dashboard'))
    if lev == config.ADMIN_ACS_L:
        return redirect(url_for('Cabinet:admin'))
    t_db_cur.execute("select * from inner_s.users;")
    users = [{"id", "login", "password", "acs_level",
        "Doing"}]

    for t_l in t_db_cur.fetchall():
        t_l_l = []
        for p in t_l:
            t_l_l.append(p)
        users.append(t_l_l)
    t_db_cur.execute(
        "select passport_id, name, surname, patronymic,
        children_number from w_dir.passports where
        passport_id < 1000;")
    passports = [
        ["passport_id", "name", "surname",
        "patronymic", "children_number", "Doing"]]
    for t_l in t_db_cur.fetchall():
        t_l_l = []
        for p in t_l:
            t_l_l.append(p)
        passports.append(t_l_l)

```

```

print(users)
print(passports)
users_io = StringIO()
json.dump(users, users_io)
passports_io = StringIO()
json.dump(passports, passports_io)
return render_template('main-dark/moderate.html',
    users=users_io, message=message,
    passports=passports_io)
@route('/admin', methods=['POST'])
def admin_post(self):
    try:
        user_update = request.form.get('user_update')
        user_id, user_login, user_password, user_ac_level =
            request.form.get('id'),
            request.form.get('login'),
            request.form.get('password'),
            request.form.get('acs_level')
        if str(user_update) != 'None' and str(user_id) !=
            None:
            t_db_cur = self._db.conn.cursor()
            t_db_cur.execute("select * from
                is_update_user_available('{usr_id}',
                '{usr_login}',
                '{user_password}');".format(usr_id=user_id,
                usr_login=user_login,
                user_password=user_password))
            if len(t_db_cur.fetchall()) <= 1:
                t_db_cur.execute(
                    "select * from update_user('{usr_id}',
                    '{usr_login}', '{user_password}',
                    '{user_ac_level}');".format(usr_id=user_id,
                    usr_login=user_login,
                    user_password=user_password,
                    user_ac_level=user_ac_level))
                return self.admin_comp("Success_update!")
    except EOFError:
        print()
    try:
        create_user = request.form.get('create_user')

```

Usr d-m

```

user_login, user_password, user_ac_level =
    request.form.get('login'), request.form.get('password'),
    request.form.get('acs_level')
    if (str(create_user) != 'None' and str(user_login)
        != 'None'):
        t_db_cur = self._db.conn.cursor()
        t_db_cur.execute(
            "select max(id) from inner_s.users")
        user_id = t_db_cur.fetchall()[0][0] + 1
        t_db_cur.execute(
            "select * from
                is_update_user_available('{usr_id}',
                '{usr_login}',
                '{user_password}');".format(usr_id=user_id,
                usr_login=user_login,
                user_password=user_password))
        if len(t_db_cur.fetchall()) < 1:
            t_db_cur.execute(
                "select * from create_user('{usr_id}',
                    '{usr_login}', '{user_password}',
                    '{user_ac_level}');".
                    format(usr_id=user_id,
                        usr_login=user_login,
                        user_password=user_password,
                        user_ac_level=user_ac_level))
            return self.admin_comp("Success_create!")
except EOFError:
    print()
try:
    delete_data = request.form.get('delete_data')
    table_name_i, id_i = request.form.get('table_name'),
    request.form.get('id')
    if str(delete_data) != 'None' and str(table_name_i)
        != 'None' and str(id_i) != 'None':
        t_db_cur = self._db.conn.cursor()
        t_db_cur.execute(
            "select * from inner_s.{table_name} where
                id='{id}';".format(table_name=table_name_i,
                id=id_i))
        if len(t_db_cur.fetchall()) == 1:
            t_db_cur.execute(

```

```

        "delete from inner_s.{table_name} where
            id='{id}';".format(table_name=table_name_i,
                               id=id_i))
        return self.admin_comp("Success_delete!")
    else:
        return self.admin_comp("Row_doesnt_exist!")
        return self.admin_comp("Error_delete!")
except EOFError:
    print()
    return self.admin_comp("Error update (Field Error)")
@route('/admin')
def admin(self):
    return self.admin_comp("Data fetched!")
def admin_comp(self, message):
    if not self._cookies.is_auth_opened():
        return redirect(url_for('Auth:login'))
    usr_id, usr_login = self._cookies.get_auth_cookie()
    t_db_cur = self._db.conn.cursor()
    t_db_cur.execute("select * from
        get_user_access_level('{usr_id}',
            '{usr_login}');".format(usr_id=usr_id,
            usr_login=usr_login))
    lev = t_db_cur.fetchall()[0][0]
    if lev == config.USR_ACS_L:
        return redirect(url_for('Cabinet:dashboard'))
    if lev == config.MODERATOR_ACS_L:
        return redirect(url_for('Cabinet:moderate'))
    t_db_cur.execute("select * from inner_s.users;")
    users = [["id", "login", "password", "acs_level",
        "Doing"]]
    for t_l in t_db_cur.fetchall():
        t_l_l = []
        for p in t_l:
            t_l_l.append(p)
        users.append(t_l_l)
    t_db_cur.execute("select passport_id, name, surname,
        patronymic, children_number from w_dir.passports
        where passport_id < 1000;")
    # print(t_db_cur.fetchall())
    passports = [
        ["passport_id", "name", "surname",

```

```

        "patronymic", "children_number", "Doing"]]
for t_l in t_db_cur.fetchall():
    t_l_l = []
    for p in t_l:
        t_l_l.append(p)
    passports.append(t_l_l)
print(users)
print(passports)
users_io = StringIO()
json.dump(users, users_io)
passports_io = StringIO()
json.dump(passports, passports_io)
return render_template('main-dark/admin.html',
    users=users_io, message=message,
    passports=passports_io)
def dashboard(self):
    if not self._cookies.is_auth_opened():
        return redirect(url_for('Auth:login'))
    repos = {
        "storage" : config.psql_repos,
        "cache" : config.tarantool_repos
    }
    t_db = DB()
    t_db_cur = t_db.conn.cursor()
    city = request.values.get('city')
    try:
        model = cache.get_birth(str(city), "passports",
            repos)
        Utils.remove_passports_fields(model, cache, repos)
        config.psql_repos.remove(model.city)
        return
        render_template('main-dark/simple_charts.html',
            data=model)
    except Exception as err:
        logging.error(err)
    t_db_cur.execute("select * from
        city_exist_cnt('{}')".format(city))

    # Except process
if request.values.get("city") is None or

```

```

t_db_cur.fetchall()[0][0] == 0:
    city = 'Not stated'
t_db_cur.execute("select * from
cit_alive_rate_by_year_city('{ }');".format(city))
cit_alive_lab, cit_alive_val =
    self._get_lab_val(t_db_cur.fetchall())
cit_alive = graphics.Graphics('People', cit_alive_lab,
cit_alive_val)
t_db_cur.execute("select * from
birth_rate_by_year_city('{ }');".format(city))
b_lab, b_val = self._get_lab_val(t_db_cur.fetchall())
birth = graphics.Graphics('Birth per year', b_lab,
b_val)
t_db_cur.execute("select * from
depth_rate_by_year_city('{ }');".format(city))
b_lab, b_val = self._get_lab_val(t_db_cur.fetchall())
depth = graphics.Graphics('Depth per year', b_lab, b_val)
t_db_cur.execute("select * from
life_length_rate_by_year_city('{ }');".format(city))
life_length_lab, life_length_val =
    self._get_lab_val(t_db_cur.fetchall())
life_length = graphics.Graphics('Life length by year',
life_length_lab, life_length_val)
try:
    model = self._db.get_passports()
    model.city = config.psycopg_repos.save(model)
    model = Utils.save_passport_fields(model,
config.psycopg_repos)
except Exception as err:
    logging.error(err)
return render_template('main-dark/simple_charts.html',
birth=birth, depth=depth, life_length=life_length,
cit_alive=cit_alive)

```

В листинге А.3 приведен код функций set-s. ,,

Листинг А.3 – Checkers

```
create or replace function is_update_user_available(id_i int,
login_i text, password_i text)
returns table
(
    id_t int
) language plpgsql as
$$
begin
    return query select id as id_t from inner_s.users
        where id = id_i or login = login_i or password =
            password_i;
end;
$$;

select * from is_update_user_available(0,
    'andreev@easyguest.ru', 'qwerty')
```


В листинге А.4 приведен код функций get-s. ,,

Листинг А.4 – Getters

```
drop function get_user_access_level (usr_id int, usr_login text);

/* Getting access level */
create or replace function get_user_access_level (usr_id int,
    usr_login text)
    returns table (
        acs_l int,
        u_id int
    )
    language plpgsql
as $$
begin
    return query
        select acs_level as acs_l, id as u_id from
            inner_s.users where id = usr_id and login =
                usr_login;
end;$$;

select * from get_user_access_level(0, 'andreev@easyguest.ru');
```

В листинге А.5 приведен код некоторых функций обновления. ,,

Листинг А.5 – Updating functions

```
create or replace function update_user(id_i int, login_i text,
    password_i text, acs_level_i int)
returns boolean language plpgsql as $$
begin
    update inner_s.users
        SET login = login_i,
            password = password_i,
            acs_level = acs_level_i
        WHERE id = id_i;
    RETURN FOUND;
end;
$$;

select * from update_user(0, 'andreev@easyguest.ru', 'qwerty',
    1);

create or replace function create_user(id_i int, login_i text,
    password_i text, acs_level_i int)
returns boolean language plpgsql as $$
begin
    insert into inner_s.users (id, login, password,
        acs_level) values (id_i, login_i, password_i,
        acs_level_i);
    return FOUND;
end;
$$;
```

В листинге А.6 приведен код некоторых функций статистики. ,,

Листинг А.6 – Statistics functions

```
/* People */
drop function cit_alive_rate_by_year_city(city text);

create or replace function cit_alive_rate_by_year_city (city
    text)
    returns table (
        year_birth decimal,
        cnt bigint
    )
    language plpgsql
as $$
begin

    if city = 'Not stated' then
        return query
            select extract(year from "birth_date") as
                year_birth, count(extract(year from
                    "birth_date")) as cnt from w_dir.passports
                group by year_birth order by year_birth;
    else
        return query
            select extract(year from "birth_date") as
                year_birth, count(extract(year from
                    "birth_date")) as cnt from w_dir.passports
                where "is_depth"=false and registration=city
                group by year_birth order by year_birth;
    end if;
end;$$;

/* Birth */
drop function birth_rate_by_year_city(city text);

create or replace function birth_rate_by_year_city (city text)
    returns table (
        year_birth decimal,
        cnt bigint
    )
    language plpgsql
as $$
```

```

begin
    if city = 'Not stated' then return query select extract(year
        from "birth_date") as year_birth, count(extract(year from
        "birth_date")) as cnt from w_dir.passports group by
        year_birth order by year_birth;
    else return query select extract(year from "birth_date")
        as year_birth, count(extract(year from "birth_date"))
        as cnt from w_dir.passports where registration=city
        group by year_birth order by year_birth;
    end if;
end;$$;

/* Depth */
drop function depth_rate_by_year_city(city text);

create or replace function depth_rate_by_year_city (city text)
    returns table (
        year_depth decimal,
        cnt bigint
    )
    language plpgsql
as $$
begin
    if city = 'Not stated' then
        return query
            select extract(year from "depth_date") as
                year_depth, count(extract(year from
                "depth_date")) as cnt from w_dir.passports where
                is_depth=true group by year_depth order by
                year_depth;
    else
        return query
            select extract(year from "depth_date") as
                year_depth, count(extract(year from
                "depth_date")) as cnt from w_dir.passports where
                is_depth=true and registration=city group by
                year_depth order by year_depth;
    end if;
end;$$;

select * from depth_rate_by_year_city('Not stated');

```

```

/* Life length */
drop function life_length_rate_by_year_city(city text);

create or replace function life_length_rate_by_year_city (city
    text)
    returns table (
        ll numeric,
        cnt bigint
    )
    language plpgsql
as $$
begin
    if city = 'Not stated'
        then
            return query
                select (extract(year from "depth_date") -
                    extract(year from "birth_date")) as ll,
                    count(extract(year from "depth_date") -
                        extract(year from "birth_date")) as cnt from
                    w_dir.passports where is_depth=true group by
                    ll order by ll;
        else
            return query
                select (extract(year from "depth_date") -
                    extract(year from "birth_date")) as ll,
                    count(extract(year from "depth_date") -
                        extract(year from "birth_date")) as cnt from
                    w_dir.passports where is_depth=true and
                    registration=city group by ll order by ll;
        end if;
end;$$;

```

В листинге А.7 приведен код некоторых функций формирования таблиц.

”

Листинг А.7 – Forming functions

```
drop table w_dir.passports;

create table w_dir.passports (
    passport_ID int,
    name text,
    surname text,
    patronymic text,
    registration text,
    children_Number int,
    is_Married_Now bool,
    birth_Date date,
    is_Depth bool,
    depth_Date date,
    primary key (passport_ID)
);
```

В листинге А.8 приведен код взаимодействия с Tarantool. ,,

Листинг А.8 – Working with Tarantool (package tarantool)

```
import logging
from db.utils import Utils
from db.repos.abstract import AbstractRepo
import db.models as models

class passportRepoTarantool(AbstractRepo):

    connection = None
    _meta = None

    space = None

    def __init__(self, connection):
        self.connection = connection
        self._meta = {
            "space_name": "passports",
            "field_names": {"passport_id": 1, "name": 2,
                           "surname": 3, "patronymic": 4,
                           "registration": 5,
                           "children_number": 6,
                           "is_married_now": 7,
                           "birth_date": 8,
                           "is_depth": 9, "depth_date": 10}
        }

        self.space = connection.space(self._meta['space_name'])

    def save(self, model):
        if not isinstance(model, models.passport):
            logging.error("Trying to save passport object of
                           invalid type")
            raise TypeError("Expected object is instance of
                             passport")

        self.space.insert((model.id, model.name, model.surname,
                           model.patronymic, model.registration,
                           model.children_number,
                           model.is_married_now,
```

```

                                model.birth_date, model.is_depth,
                                model.depth_date))

def get_by_id(self, model_id):
    obj = self.space.select(model_id)
    if len(obj) == 0:
        return None

    return models.passport(*obj[0])

def get_by_filter(self, index, key):
    raw_objects = self.space.select(key, index=index)
    if len(obj) == 0:
        return None, None

    models_list = list()
    primary_keys = list()
    for obj in raw_objects:
        model = models.passport(*obj)
        models_list.append(model)
        primary_keys.append(model.id)

    return models_list, primary_keys

def get_all(self):
    raw_objects = self.space.select()
    if len(raw_objects) == 0:
        return None

    models_list = list()
    for obj in raw_objects:
        models_list.append(models.passport(*obj))

    return models_list

def remove(self, id):
    obj = self.space.delete(id)
    if len(obj) == 0:
        return None

    return obj

```



```

def edit(self, *args, **kwargs):
    obj_id = kwargs['id']
    updated_args =
        Utils.get_tarantool_update_args(kwargs['fields'],
            self._meta['field_names'])

    return self.space.update(obj_id, updated_args)[0]

class usersRepoTarantool(AbstractRepo):

    connection = None
    _meta = None

    space = None

    def __init__(self, connection):
        self.connection = connection
        self._meta = {
            "space_name": "users",
            "field_names": {
                "id": 1,
                "login": 2,
                "password": 3,
                "acs_level": 4,
            }
        }

        self.space = connection.space(self._meta['space_name'])

    def save(self, model):
        if not isinstance(model, models.users):
            logging.error("Trying to save users object of
                invalid type")
            raise TypeError("Expected object is instance of
                users")

        self.space.insert(tuple(model.__dict__.values()))

    def get_by_filter(self, index, key):

```

```

raw_objects = self.space.select(key, index=index)
if len(raw_objects) == 0:
    return None, None

models_list = list()
primary_keys = list()
for obj in raw_objects:
    model = models.users(*obj)
    models_list.append(model)
    primary_keys.append(model.id)

return models_list, primary_keys

def remove(self, id):
    obj = self.space.delete(id)
    if len(obj) == 0:
        return None

return obj

```