



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 (ИУ)

КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

***Разработка базы данных для сервиса по почтовой
рассылке персонализированных предложений***

Студент группы ИУ7-62Б

Карпова Е. О.

Руководитель курсовой работы

Кострицкий А. С.

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 53 с., 11 рис., 9 табл., 44 источн., 1 прил.

Ключевые слова: Базы данных, ClickHouse, Apache Kafka, PostgreSQL, SQL, Docker, SMTP, API, Vue, SPA, агрегация данных, колоночная база данных, строковая база данных.

Объектом разработки является база данных для сервиса по почтовой рассылке персонализированных предложений.

В процессе разработки был проведен анализ предметной области и сравнительный анализ предложенного решения относительно существующих. Формализованы варианты использования, хранимые данные и предоставляемые сервисом возможности. Проведен выбор модели данных, базы данных по способу хранения и системы управления базой данных по способу доступа к базе данных. Составлена ER-диаграмма сущностей проектируемой базы данных в нотации Чена и спроектированы сущности базы данных и накладываемые ограничения целостности. Была описана проектируемая ролевая модель на уровне базы данных, определены права доступа к внутренним структурам и обоснован выбор средств реализации базы данных и приложения. Описан интерфейс доступа к базе данных. Также, проведено исследование производительности строковых и колоночных СУБД на агрегационных запросах.

По результатам исследования сформулирован вывод о большей производительности колоночных баз данных по сравнению со строковыми для выполнения агрегационных запросов.

СОДЕРЖАНИЕ

РЕФЕРАТ	2
ОПРЕДЕЛЕНИЯ	5
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	6
ВВЕДЕНИЕ	7
1 Аналитический раздел	8
1.1 Анализ предметной области	8
1.2 Обзор существующих решений	9
1.3 Формализация задачи	10
1.4 Выбор модели данных для базы данных	12
1.5 Выбор базы данных	14
1.6 Выбор системы управления базами данных	15
1.7 Вывод	16
2 Конструкторский раздел	17
2.1 Формализация сущностей системы	17
2.2 Домены	20
2.3 Ролевая модель на уровне базы данных	21
2.4 Разработка табличной функции	22
2.5 Вывод	23
3 Технологический раздел	24
3.1 Выбор СУБД	24
3.2 Выбор средств разработки серверной и клиентской частей ПО	25
3.3 Выбор средств реализации API	26
3.4 Взаимодействие компонентов и безопасность	28
3.5 Web-интерфейс	29
3.6 Тестирование	29
3.7 Вывод	30

4 Исследовательский раздел	31
4.1 Постановка исследования	31
4.2 Результаты исследования	31
4.3 Вывод	33
ЗАКЛЮЧЕНИЕ	34
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	38
ПРИЛОЖЕНИЕ А SQL-скрипты	39
ПРИЛОЖЕНИЕ Б Реализованный web-интерфейс	44
ПРИЛОЖЕНИЕ В Тестирование	47

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

База данных — совокупность данных из определённой предметной области, организованных по определённым правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ [1].

Модель данных — абстрактная модель, определяющая совокупность правил создания и использования данных, описывающая способ представления данных, доступ к ним и связи между ними в той или иной области [2].

Система управления базами данных — это совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия её с прикладными программами [1].

Брокер сообщений — программное обеспечение, которое позволяет приложениям, системам и службам взаимодействовать друг с другом и обмениваться информацией [3].

Топик — логическая категория сообщений [4].

Партиция — физическое хранилище сообщений в кластере Kafka [4].

Движок таблицы — тип таблицы, определяющий способ хранения и получения данных, поддерживаемые виды запросов и иные параметры работы с данными [5].

Аутентификация — это проверка соответствия субъекта и того, за кого он пытается себя выдать с помощью некой уникальной информации [6].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

БД — База Данных.

СУБД — Система Управления Базами Данных.

ACID — Atomicity, Consistency, Isolation, Durability.

API — Application Programming Interface.

MVVM — Model-View-ViewModel.

JWT — JSON Web Token.

АКИТ — Ассоциация Компаний Интернет-Торговли.

HTML — HyperText Markup Manguage.

JSON — JavaScript Object Notation.

SQL — Structured Query Language.

ПО — Программное Обеспечение.

SMTP — Simple Mail Transfer Protocol.

SPA — Single Page Application.

ВВЕДЕНИЕ

В последние годы информационные технологии проникают всё в большее количество сфер общественной жизни, в том числе и сегмент торговли. Большинство современных компаний нацелено на продвижение именно своих интернет-платформ, что связано с уменьшением транзакционных расходов и ростом роли дистанционного взаимодействия [7]. Продвижение онлайн-площадок в сети Интернет при помощи рекламной рассылки является одним из распространённых методов [8].

Целью данной работы является разработка базы данных для сервиса по почтовой рассылке персонализированных предложений.

В ходе работы необходимо решить следующие задачи:

- 1) Провести анализ предметной области и сравнить предложенное решение с существующими.
- 2) Формализовать варианты использования и предоставляемые сервисом возможности и описать информацию, подлежащую хранению в базе данных.
- 3) Произвести выбор модели данных, базы данных по способу хранения и системы управления базой данных по способу доступа к базе данных.
- 4) Составить ER-диаграмму сущностей проектируемой базы данных в нотации Чена и спроектировать сущности базы данных и накладываемые ограничения целостности.
- 5) Описать проектируемую ролевую модель на уровне базы данных и определить права доступа к внутренним структурам.
- 6) Описать выбор средств реализации базы данных и приложения и интерфейс доступа к базе данных.
- 7) Провести сравнительный анализ различных систем управления базой данных для хранения и обработки аналитических данных.

1 Аналитический раздел

1.1 Анализ предметной области

По данным ежегодного отчёта Global Digital 2022, почти 6 из 10 интернет-пользователей трудоспособного возраста (58,4%) покупают что-то в Интернете каждую неделю [9]. Также, результаты проведённого АКИТ e-commerce анализа показывают, что по сравнению с 2021 годом доля российского рынка интернет-торговли выросла к 2022 году на 1065 миллиардов рублей [10].

В связи с этим, интернет-реклама обрела широкое распространение, как обеспечивающая наиболее эффективное продвижение при помощи механизмов таргетинга [8]. Одним из видов такого продвижения является рекламная рассылка, когда пользователям через каналы обмена сообщениями отправляются письма рекламного характера. Особенно эффективной является персонализированная рекомендательная рассылка, содержание которой формируется на основе предпочтений, так как она позволяет наиболее корректно определять возможные интересы конечного пользователя [11].

Однако, распространение рекламы по сетям электросвязи допускается только при условии предварительного согласия абонента или адресата на её получение [12], [13], [14]. Закон о рекламе не определяет порядка и формы получения согласия, поэтому оно может быть выражено в любой форме [15].

Подобная рассылка позволяет стимулировать пользователя к возвращению на сайт [8], подбирать с помощью персонализации актуальные для конкретного пользователя предложения, что увеличивает шанс покупки.

Возможна реализация уведомления пользователей о, например, проведении массовых акций, появлении нового ассортимента, предоставлении персональной акции.х

1.2 Обзор существующих решений

Для сравнения решений были выбраны критерии:

- 1) Местоположение серверов.
- 2) Форма распространения продукта.
- 3) Встроенная аналитика.
- 4) Ценовая политика.

Было проведено сравнение предложенного решения (SO) с тремя конкурентными: Unisender (U), Mindbox (M), Sendsay (S). По информации, предоставляемой компаниями на сайтах, они являются ведущими в данной сфере [16], [17], [18]. Л.д. — личные данные, уст. — устанавливаемый, усл. — условно.

Таблица 1.1 – Таблица сравнения существующих решений

Критерий	Решение			
	U	M	S	SO
Местоположение серверов	РФ	РФ	РФ	РФ
Форма распр.	Облачный	Облачный	Облачный	Уст.
Сегментация	По л.д., по активности с письмами и на сайте	По л.д., по активности на сайте	По л.д., по активности с письмами	По л.д., по активности на сайте на основе аналитики
Ценовая политика	Усл. беспл.	Платно	Усл. беспл.	Бесплатно

Таким образом, предложенное решение, как устанавливаемое, обеспечивает построение независимой от внешних факторов системы и возлагает на пользователя обязанность по выбору защищённой точки хранения, так

как происходит работа с персональными данными. Также, данное решение, в отличие от аналогов, является полностью бесплатным и позволяет агрегировать данные о пользовательской активности на сайте, формируя на их основе статистику и более обширную аналитическую сводку, влияющую впоследствии на правила рассылки.

1.3 Формализация задачи

В рамках работы необходимо разработать веб-приложение, обеспечивающее управление рекламной рассылкой на основе данных, получаемых основной торговой площадкой от клиента, при наличии согласия на обработку этих данных. Обрабатываются любые зарегистрированные в системе события, связанные с действиями клиента на сайте.

Приложение хранит предоставленные личные данные о клиенте и информацию о событиях, связанных с зарегистрированными клиентами.

Разрабатываемый продукт должен представлять собой приложение модели клиент-сервер. Серверная часть приложения отвечает за взаимодействие с хранилищем данных и обработку запросов основной системы, которая подключена к приложению. Клиентская часть представляет собой веб-страницу с пользовательским интерфейсом.

Формализация пользовательских сценариев

Пользователи системы рассылки — таргетологи и администраторы. Клиент основной системы не имеет собственного интерфейса в приложении, но при наличии согласия на обработку персональных данных, запрашиваемого на стороне подключающейся системы, его действия изменяют информацию, хранящуюся в приложении.

На рисунке 1.1 представлены пользовательские сценарии для реализации.

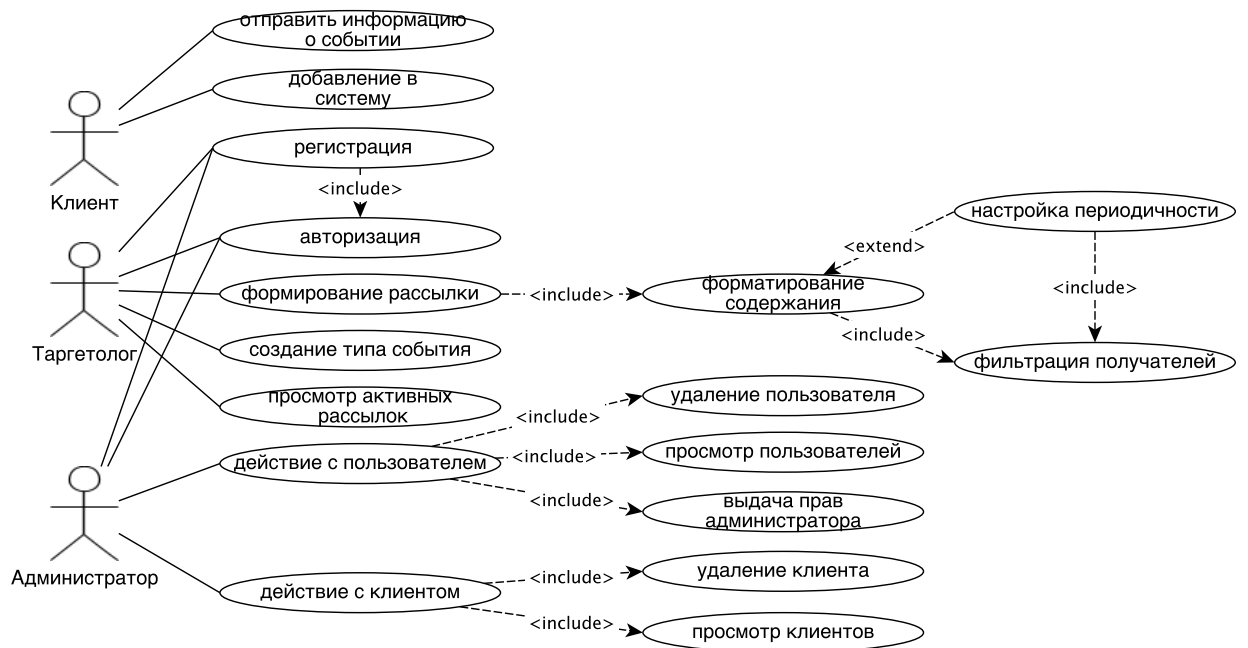


Рисунок 1.1 – Пользовательские сценарии

Таргетологам доступны:

- список незавершённых на момент просмотра рассылок;
- экран формирования письма: с возможностью форматирования текста письма в HTML и предпросмотра результата, настройки периодичности и фильтрации;
- возможность создания типа события для дальнейшей фильтрации.

Администратору доступны функции управления системой верхнего уровня:

- просмотр пользователей и клиентов системы;
- удаление пользователей и клиентов системы;
- выдача прав администратора пользователю.

Формализация данных

Разрабатываемая база данных должна состоять из следующих сущностей:

- 1) Клиент.
- 2) Событие.
- 3) Пользователь.
- 4) Рассылка.
- 5) Расписание.

На рисунке 1.2 представлена ER-диаграмма сущностей в нотации Чена.

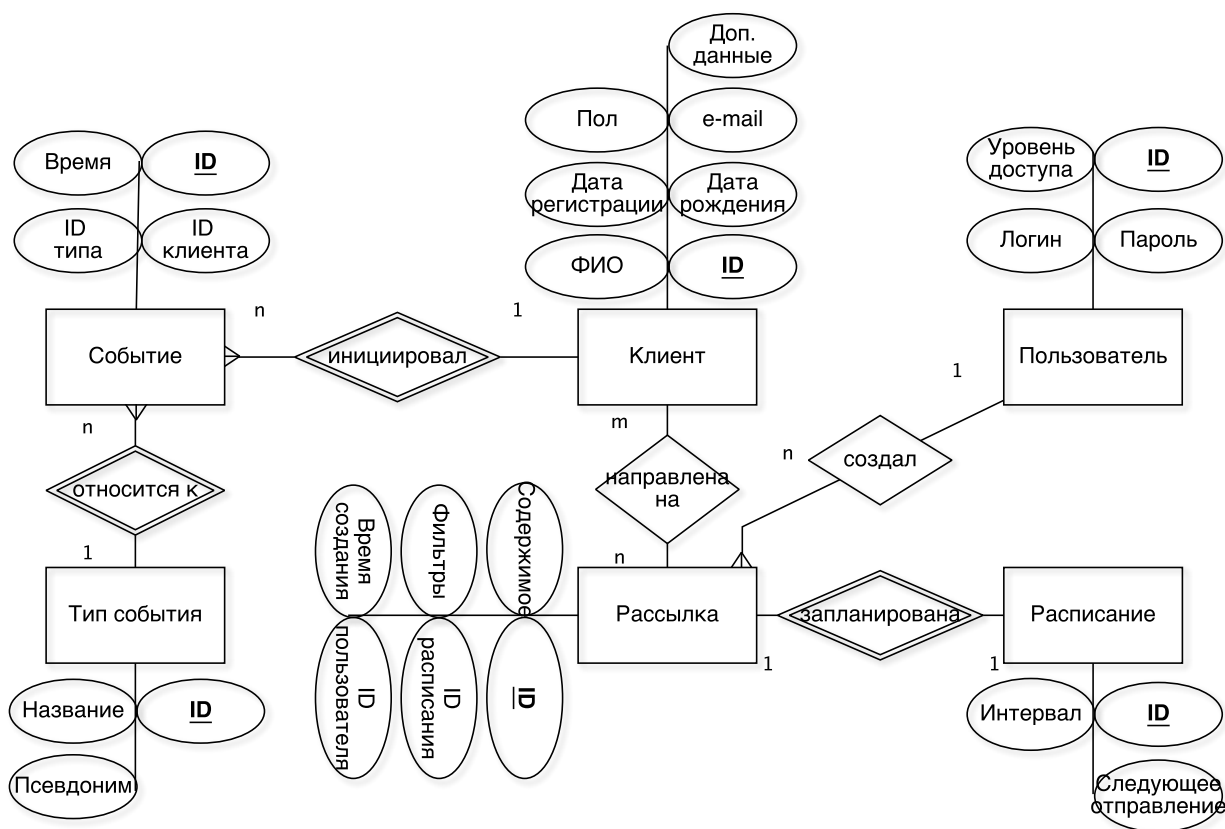


Рисунок 1.2 – ER-диаграмма

1.4 Выбор модели данных для базы данных

Для построения базы данных, как совокупности взаимосвязанных данных некоторой предметной области, необходимо определить базовую модель данных. Модель данных — это абстрактная модель, определяющая совокупность правил создания и использования данных, описывающая способ

представления данных, доступ к ним и связи между ними в той или иной области [2].

Иерархическая модель

Графически такую модель можно представить в виде графа типа «дерево» [19]. В такой модели имеется одна вершина — корень дерева, являющаяся входом в структуру. Каждая вершина, отличная от корня, может иметь только одну исходную вершину и сколько угодно порожденных. В связи с этим является невозможной реализация типа связи «многие ко многим», а так же создание сущности без родительского узла. Направление каждой связи определяется иерархией. Ограничения на тип внутренней структуры отсутствуют, связи задаются адресными указателями, что накладывает обязательство учёта физической организации хранения [20].

Сетевая модель

Сетевая модель графически представима в виде графа типа «сеть» [19]. Входом в такую структуру может являться любая вершина. Для одной вершины нет ограничений на количество исходных и ею порождённых. Между парой вершин может существовать более одной связи, поэтому реализация типа связи «многие ко многим» становится возможной. Однако, подобная реализация значительно усложнит схему проектируемой базы данных [21]. Направление и характер связи в сетевых моделях не predetermined, как в иерархической модели, поэтому направление каждой связи должно быть указано. Ограничения на тип внутренней структуры отсутствуют. Связи задаются адресными указателями, что накладывает обязательство учёта физической организации хранения [20].

Инвертированные списки

Также, существуют системы, построенные на использовании инвертированных списков [19]. В таких базах данные и информация о связях между элементами данных логически и физически сепарированы. Данные хранятся в файлах сложной структуры, а информация о связях сосредотачивается в отдельных блоках — ассоциаторах и сетях связи. В системах, построенных на инвертированных списках, можно реализовывать связь типа «многие ко мно-

гим». Отделение ассоциативной информации от хранимых данных позволяет изменять связи, не изменяя при этом сами данные.

Реляционная модель

Реляционная модель состоит из трех частей: структурной, целостностной и манипуляционной. Структурная часть описывает, из каких объектов состоит реляционная модель. Целостная часть фиксирует базовые требования целостности. Манипуляционная часть описывает способы манипулирования реляционными данными.

В реляционных базах данные хранятся в форме отношений — таблиц, состоящих из записей. Записи имеют линейную структуру [19]. Таблицы могут иметь связи с другими таблицами через внешние ключи, а каждая запись имеет свой уникальный в пределах таблицы идентификатор — первичный ключ. Модель поддерживает связи «один к одному» и «один ко многим». Связь «многие ко многим» реализуется с помощью создания связующей сущности. Данная модель характеризуется низкой трудоёмкостью отображения сложных связей между сущностями [20], наличием строгой регламентации операций над сущностями и правил организации системы хранения, позволяющих избежать дублирования данных [21].

1.5 Выбор базы данных

Базы данных (БД), по способу хранения, делятся на две группы — строковые и колоночные.

Строковые базы данных

Строковые базы данных — это такие базы данных, записи которых в памяти представлены построчно. Для систем, использующих строковые базы, характерно большое количество операций вставки, обновления и удаления данных [22]. Эффективнее будут запросы, обращающихся к отдельным записям. В таких системах мера эффективности — количество транзакций в единицу времени.

Колоночные базы данных

Колоночные базы данных — это такие базы данных, записи которых в

памяти представляются по столбцам [22]. Колоночные базы данных используются в аналитических системах, где запросы сложнее и включают в себя агрегацию данных. В таких системах мера эффективности — время отклика.

1.6 Выбор системы управления базами данных

Система управления базами данных (СУБД) — это приложение, обеспечивающее создание, обновление, удаление, хранение и поиск информации в базе данных. По используемой модели данных СУБД делится на реляционные и нереляционные. Так как выбор уже сделан в пользу реляционной модели, то данные категории рассматриваться не будут. По способу доступа к базе данных СУБД делятся на файл-серверные, клиент-серверные, встраиваемые и прочие.

Файл-серверные СУБД

Файл-серверные СУБД — это СУБД, у которых файлы данных располагаются локально [23]. На компьютере пользователя запускается копия приложения. По каждому запросу к БД из приложения данные из таблиц БД полностью загружаются на компьютер пользователя, независимо от того, сколько на самом деле данных необходимо для выполнения запроса. Каждый пользователь имеет на своем компьютере локальную копию данных, время от времени обновляемых из реальной БД, расположенной на сетевом сервере. Вся тяжесть выполнения запросов к базе данных и управления целостностью базы данных ложится на приложение пользователя.

Клиент-серверные СУБД

Клиент-серверная СУБД — основанная на технологии «клиент-сервер» [24]. Этот механизм позволяет обмениваться информацией между клиентом и сервером с переносом основной нагрузки на сервер. Клиент только организывает доступ пользователя к серверу. При направлении запроса к БД клиент получает только искомые записи из-за ненужности передавать весь массив данных из БД.

Встраиваемые СУБД

Встраиваемые СУБД — это СУБД, которые встраиваются в приложения в виде разделяемой программной библиотеки [25]. Встраиваемые СУБД удобно использовать в легковесных приложениях, использующих СУБД для доступа к данным, компьютерных играх, хранящих данные игрового процесса и самой игры, большом количестве приложений, предназначенных для работы на мобильных устройствах. Встраиваемая СУБД локально хранит данные конкретного приложения и не рассчитана на совместное использование.

1.7 Вывод

В этом разделе был проведён анализ предметной области и обзор существующих решений, обоснована актуальность решаемой продуктом задачи. Также, была формализована задача, пользовательские сценарии и данные.

Была выбрана реляционная модель данных, как наилучшим образом подходящая для построения системы, подразумевающей нетривиальные связи между сущностями. Отсутствие необходимости изменять структуру данных и возможность исключить дублирование также повлияли на данный выбор.

Для хранения аналитических данных была выбрана колоночная база данных, так как для данного вида информации большую долю запросов будут составлять агрегационные, наиболее оптимально выполнимые в колоночных БД. Для хранения иных данных в работе будет использована строковая БД.

В качестве СУБД была выбрана клиент-серверная, так как данный вариант не накладывает ограничений на объём хранимых данных, и позволяет не манипулировать ненужными в рамках запроса данными.

2 Конструкторский раздел

2.1 Формализация сущностей системы

Обозначения в таблицах: PK — первичный ключ, FK — внешний ключ, U — уникальное значение, NN — поле не может принимать неопределённое значение, I — целочисленный тип, B — логический тип, DT — тип временной отметки, SP — тип временного интервала, S — символьный тип, L — логический тип, J — JSON, ID — тип идентификатора.

Таблица Client

В таблице Client содержится информация о клиентах основной системы. Описание полей таблицы Client представлено в таблице 2.1.

Таблица 2.1 — Описание полей таблицы Client

Название	Характеристики					
	PK	FK	U	NN	Тип	Значение
id	+	—	+	+	ID	Идентификатор клиента
name	—	—	—	+	S	Имя клиента
surname	—	—	—	+	S	Фамилия клиента
patronymic	—	—	—	—	S	Отчество клиента
birthDate	—	—	—	+	DT	Дата рождения
registrationDate	—	—	—	+	DT	Дата регистрации
gender	—	—	—	+	S	Пол клиента
email	—	—	+	+	S	Адрес электронной почты
data	—	—	—	—	J	Дополнительная информация

Таблица User

В таблице User содержится информация о пользователях системы рассылки. Описание полей таблицы User представлено в таблице 2.2.

Таблица 2.2 – Описание полей таблицы User

Название	Характеристики					
	PK	FK	U	NN	Тип	Значение
id	+	–	+	+	ID	Идентификатор пользователя
login	–	–	+	+	S	Логин пользователя
password	–	–	–	+	S	Пароль пользователя
isAdmin	–	–	–	+	B	Уровень доступа в системе

Таблица Event

В таблице Event содержится информация о событиях основной системы. Описание полей таблицы Event представлено в таблице 2.3.

Таблица 2.3 – Описание полей таблицы Event

Название	Характеристики					
	PK	FK	U	NN	Тип	Значение
id	+	–	+	+	ID	Идентификатор события
alias	–	–	–	+	S	Идентификатор типа события
clientID	–	+	–	–	ID	Идентификатор инициатора
eventTime	–	–	–	+	DT	Время события

Таблица EventType

В таблице EventType содержится информация о типах событий основной системы. Описание полей таблицы EventType представлено в таблице 2.4.

Таблица 2.4 – Описание полей таблицы EventType

Название	Характеристики					
	PK	FK	U	NN	Тип	Значение
id	+	–	+	+	ID	Идентификатор события
name	–	–	–	+	S	Название события
alias	–	–	+	+	S	Псевдоним события

Таблица Ad

В таблице Ad содержится информация о рекламных записях для рассылки. Описание полей таблицы Ad представлено в таблице 2.5.

Таблица 2.5 – Описание полей таблицы Ad

Название	Характеристики					
	PK	FK	U	NN	Тип	Значение
id	+	–	+	+	ID	Идентификатор рассылки
content	–	–	–	+	S	Содержимое записи
filters	–	–	–	+	J	Фильтры рассылки
createTime	–	–	–	+	DT	Время создания
userID	–	+	–	–	ID	Идентификатор создателя
scheduleID	–	+	+	+	ID	Идентификатор расписания

Таблица Schedule

В таблице Schedule содержится информация о расписаниях для рассылок. Описание полей таблицы Schedule представлено в таблице 2.6.

Таблица 2.6 – Описание полей таблицы Schedule

Название	Характеристики					
	PK	FK	U	NN	Тип	Значение
id	+	–	+	+	ID	Идентификатор расписания
nextTime	–	–	–	+	DT	Время следующей рассылки
span	–	–	–	+	SP	Интервал рассылки
finished	–	–	–	+	B	Активность рассылки
periodic	–	–	–	+	B	Периодичность рассылки

2.2 Домены

Для контроля валидности данных введены следующие домены:

- 1) etType — временная метка с ограничениями: не позже, чем текущий момент времени.
- 2) genderType — перечисление, имеющее 2 возможных значения: male, female.
- 3) bdType — временная метка с ограничениями: не позже, чем текущий момент времени и не ранее, чем 01.01.1900 00:00:00.
- 4) rdType — временная метка с ограничениями: не позже, чем текущий момент времени, не ранее, чем дата рождения.
- 5) loginType — символьный тип с ограничениями: не менее 5 символов.
- 6) aliasType — символьный тип с ограничениями: не менее 5 символов.
- 7) ctType — временная метка с ограничениями: не позже, чем текущий момент времени.

На рисунке 2.1 представлена ER-диаграмма сущностей в нотации Мартина.

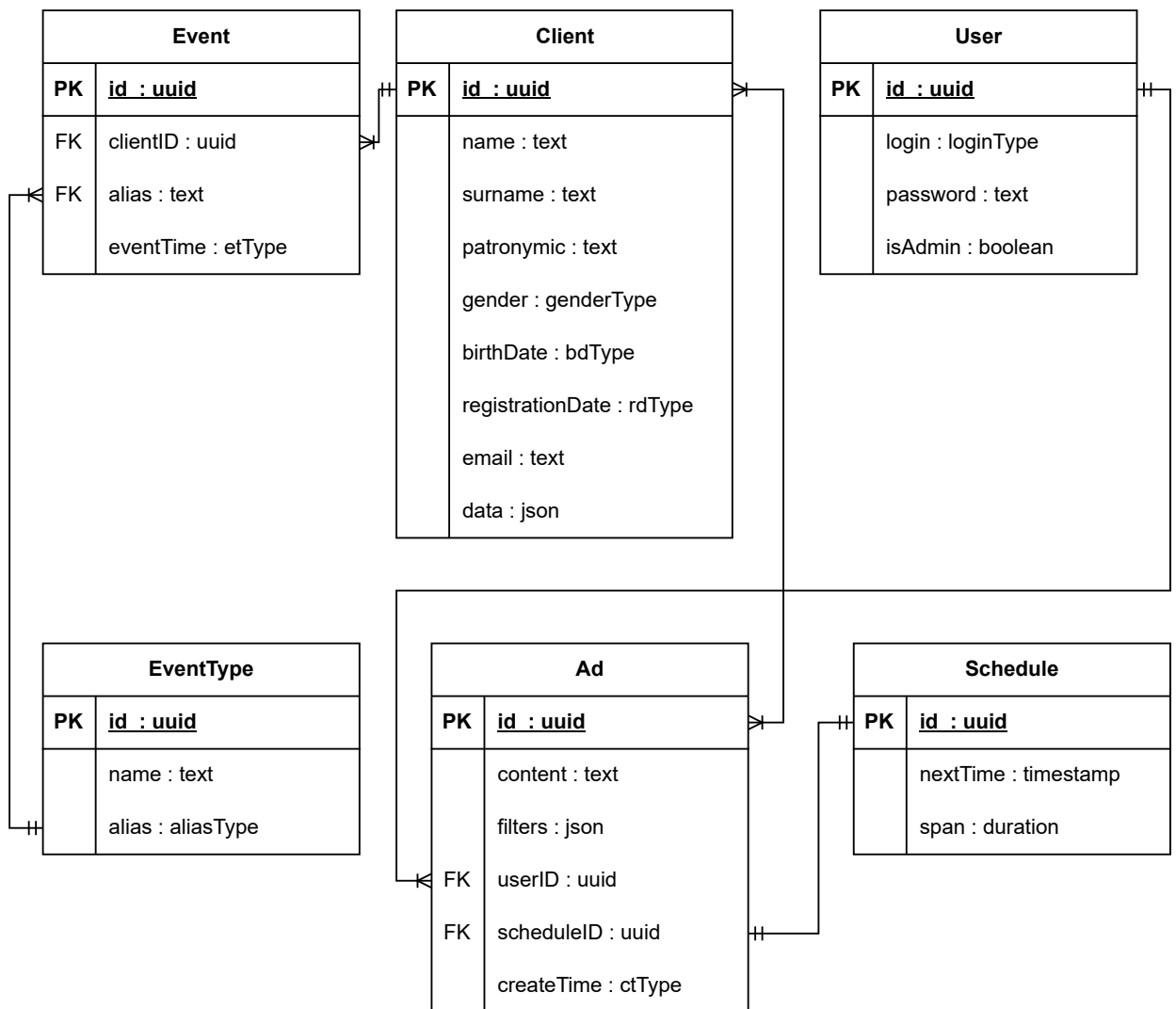


Рисунок 2.1 – ER-диаграмма сущностей в нотации Мартина

2.3 Ролевая модель на уровне базы данных

На уровне взаимодействия с базой данных представлена следующая ролевая модель:

- 1) Client — пользователь основной системы. Обладает правами на:
 - INSERT в таблицу клиентов Client;
 - INSERT в таблицу событий Events.
- 2) Targetologist — таргетолог, координирующий рекламную рассылку. Обладает правами на:
 - SELECT, INSERT в таблицу пользователей User;

- SELECT, INSERT в таблицу рассылок Ad;
- SELECT, INSERT в таблицу типов событий EventType;
- SELECT, INSERT, UPDATE в таблицу расписания Schedule;
- SELECT в таблицу клиентов Client;
- SELECT в таблицу событий Events.

3) Administrator — администратор системы. Обладает правами на:

- SELECT, INSERT, DELETE, UPDATE в таблицу пользователей User;
- SELECT, DELETE в таблицу клиентов Client.

2.4 Разработка табличной функции

Необходимо разработать табличную функцию, реализующую выборку записей, соответствующих записям о рекламной рассылке, которую необходимо отправить в заданный временной интервал. Записи выбираются из таблицы, сформированной объединением таблицы рассылок Ad с таблицей расписания Schedule.

Схема соответствующей табличной функции `get_ads_by_span` представлена на рисунке 2.2.

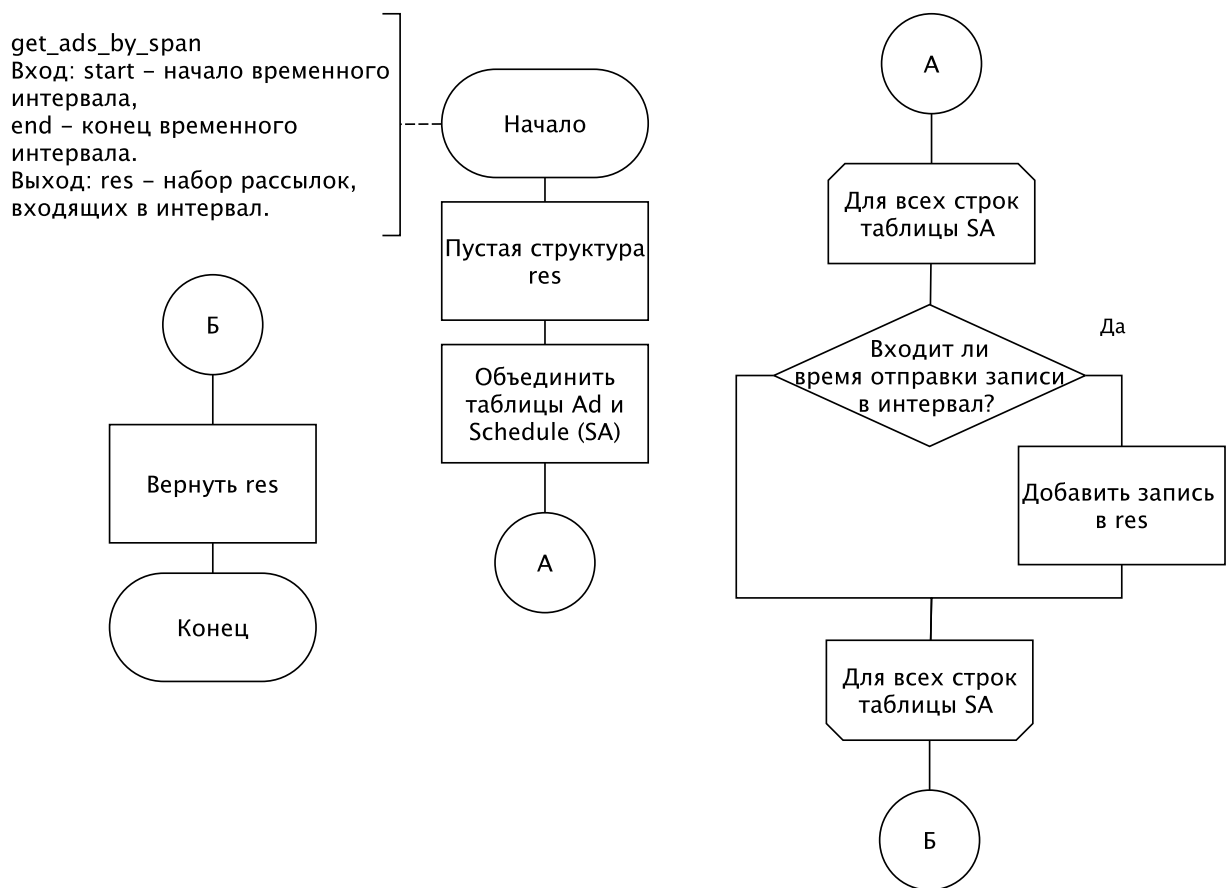


Рисунок 2.2 – Схема табличной функции get_ads_by_span

2.5 Вывод

В данном разделе была проведена формализация сущностей проектируемой системы: на каждую сущность приведена таблица, описывающая её характеристики и необходимые атрибуты. Также, описаны используемые домены, позволяющие поддерживать целостность и непротиворечивость системы, и ролевая модель, определяющая права доступа пользователей к объектам базы данных. Представлена схема алгоритма работы реализуемой табличной функции, осуществляющей выборку записей о рекламной рассылке, которую нужно отправить в заданный интервал времени, реализация которой приведена в листинге Приложения А.

3 Технологический раздел

3.1 Выбор СУБД

Основные данные

В качестве СУБД была выбрана бесплатная система управления базами данных PostgreSQL, поддерживаемая большинством операционных систем. PostgreSQL является реляционной [26] и поддерживает множество различных типов и структур данных, в том числе данные в текстовом формате JSON. Также с её помощью можно создавать пользовательские типы данных. PostgreSQL отвечает требованиям ACID, а также поддерживает различные виды ограничений, за счёт которых обеспечивается целостность данных системы, оконные функции, табличные функции и триггеры. БД Postgres является строковой, что означает высокую скорость выполнения запросов, обращающихся к отдельным записям. Листинги скриптов для работы с Postgres приведены в Приложении А.

Аналитические данные

Для аналитических данных была выбрана колоночная СУБД ClickHouse. Она обеспечивает сжатие данных, автоматическое распараллеливание больших запросов по ресурсам предоставленного сервера и физическую сортировку данных по первичному ключу [27]. ClickHouse поддерживает декларативный язык запросов на основе SQL, включая группировку, оконные функции, подзапросы и объединения. Данный вид БД обеспечивает высокую скорость запросов на агрегацию, что особенно важно для аналитических данных [22]. Листинги скриптов для работа с ClickHouse приведены в Приложении А.

Для таблиц был использован движок MergeTree как наиболее функциональный среди движков ClickHouse [28].

Для таблицы очереди событий был использован движок Kafka [29], который работает с Apache Kafka. Это позволяет организовывать прямую связь топика в Kafka с таблицей ClickHouse, а следовательно: подписываться на потоки данных, организовать отказоустойчивое хранилище, обрабатывать события по мере их появления. При помощи данного движка реализуется массовая вставка данных [30], что обеспечивает большую производительность

и меньшую нагрузку при работе с колоночными СУБД.

Потоковые данные

Для передачи данных о событиях в основной системе, которые можно отнести к потоковым данным, в хранилище ClickHouse был использован Apache Kafka — распределённый программный брокер сообщений с открытым исходным кодом для обработки потоковых данных в реальном времени с высокой пропускной способностью [31]. Также, Kafka был использован для подготовки рекламных сообщений к отправлению. В обоих случаях были созданы соответствующие топика, то есть виртуальные хранилища сообщений схожего содержания, — `events` для добавления событий в хранилище и `ads` для рассылки рекламы. Данные разных топиков обрабатываются параллельно, и внутри каждого топика Kafka автоматически партиционирует данные, что также обеспечивает параллельную обработку. В целом, в Kafka процессы генерирования, отправки и считывания сообщений организованы независимо друг от друга. Использование брокера обеспечивает видимое преимущество в скорости работы с потоками данных.

3.2 Выбор средств разработки серверной и клиентской частей ПО

Для разработки серверной части был выбран язык программирования Golang [32]. Выбор обусловлен наличием в Golang библиотек для тестирования ПО, работы с `http` протоколом [33], взаимодействия с выбранными для хранения данных СУБД [34], работы с JWT-токенами [35], шифрования данных [36], а также иных необходимых для реализации поставленных цели и задач средств.

В качестве SMTP сервера для рассылки почтовой рекламы был выбран Brevo [37]. По данным компании, 99,8% всех отправленных с её платформы электронных писем, доставляются в течение 20 секунд. Также, одной из причин выбора стало то, что сервис предоставляет 300 бесплатных электронных писем в день при использовании бесплатного плана. Для рассылки был использован и подтверждён домен `blurby.lownie.su`.

Для разработки клиентской части был выбран фреймворк Vue.js — фреймворк для создания пользовательских интерфейсов, подходящий для

реализации сложных одностраничных приложений [38]. Во Vue зависимости компонента автоматически отслеживаются при отрисовке, поэтому система отрисовывает только затронутые компоненты при изменении состояния в отличие от React [39]. По производительности Vue сравним с такими популярным фреймворками как React и Angular и является более производительным, чем AngularJS. Также, Vue проще многих других фреймворков по архитектуре.

Был использован MVVM паттерн. Интерфейс перестраивается автоматически при изменении данных за счёт использования Vuex [40]: он служит централизованным хранилищем для всех компонентов приложения, изменяя представляемые пользователю данные предписанным образом.

3.3 Выбор средств реализации API

Для реализации API взаимодействия серверной и клиентской частей продукта был использован Swagger — набор инструментов, созданных на основе спецификации OpenAPI, нужных при разработке, создании, документировании и использовании API [41].

В файле спецификации были описаны пути запросов и соответствующие им методы и форматы запросов и ответов на запрос. При помощи OpenAPI Generator был сгенерирован серверный интерфейс, на основе которого были позже реализованы обработчики запросов [42].

Интерфейс взаимодействия серверной и клиентской частей представлен в таблице 3.1.

Таблица 3.1 – Интерфейс взаимодействия серверной и клиентской частей

Путь	Метод	Описание
/login	post	Вход пользователя в систему
/register	post	Регистрация пользователя
/events	post	Добавление информации о событии
/filter	post	Фильтрация событий
/users	get	Получение информации о всех пользователях
/clients	get	Получение информации о всех клиентах
/client	get	Получение информации о клиенте
/client	post	Регистрация клиента системы
/client	delete	Удаление клиента из системы
/stats	get	Получение статистики системы
/user/id	get	Получение информации о пользователе по ID
/user	delete	Удаление пользователя
/user	get	Получение информации о пользователе по логину
/user	put	Выдача пользователю прав администратора
/event_types	post	Добавление типа события
/event_types	get	Получение типов событий
/ads	post	Создание рекламной рассылки
/ads	get	Получение рекламных рассылок
/users/me	get	Получение информации о текущем пользователе

3.4 Взаимодействие компонентов и безопасность

Взаимодействие серверной части с хранилищем было реализовано при помощи паттерна "Репозиторий". Разделение серверной логики и компонента доступа к данным позволяет без изменения кода подменять реализацию хранилища, обеспечивая удовлетворение обработчиков репозитория в заданному в логике интерфейсу.

Обобщённая схема взаимодействия компонентов изображена на рисунке 3.1.

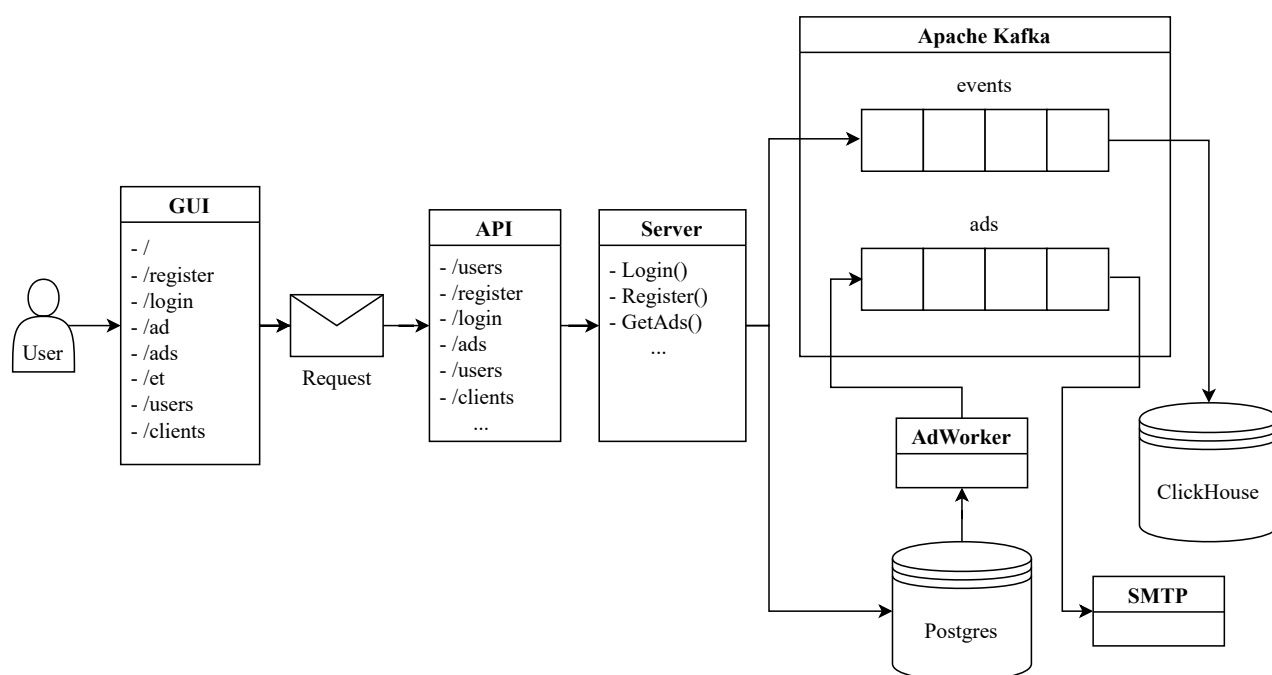


Рисунок 3.1 – Взаимодействие компонентов

Для безопасности хранимых и передаваемых между компонентами данных были предусмотрены определённые средства защиты:

- 1) Производится шифрование пароля с помощью хэш-функции bcrypt, пароль поступает в хранилище уже в зашифрованном виде.
- 2) Ролевая модель ограничивает доступ пользователей к данным хранилища, если производится попытка их получения с подключения, не имеющего соответствующих прав.
- 3) Производится аутентификация пользователя с помощью JWT-токена — средства безопасной передачи информации посредством механизма подписи — назначаемого при успешном входе в систему [43].

3.5 Web-интерфейс

Web-интерфейс организован в формате одностраничного web-приложения или SPA.

Таргетологу доступны:

- 1) домашняя страница с просмотром статистики системы;
- 2) страница формирования рекламной рассылки с настройками фильтрации;
- 3) страница просмотра списка незавершённых рекламных рассылок;
- 4) страница просмотра и создания типов событий, обрабатываемых системой.

Администратору доступны:

- 1) домашняя страница с просмотром статистики системы;
- 2) страница просмотра и редактирования списка всех пользователей системы;
- 3) страница просмотра и редактирования списка всех клиентов системы.

В приложении Б представлен реализованный web-интерфейс.

3.6 Тестирование

Было проведено модульное тестирование функций модуля работы с пользователями системы при помощи стандартной библиотеки testing языка Golang. Модульные тесты обеспечивают полное покрытие функций авторизации, регистрации и входа в систему. В Приложении В приведена реализация одного из тестов для функции входа в систему.

Также, было проведено интеграционное тестирование функции получения информации о клиенте модуля работы с клиентами системы. Для каждой группы интеграционных тестов запускается Docker-контейнер с базой данных, которая при запуске заполняется тестовыми данными. SQL-скрипт для заполнения таблиц тестовыми данными, реализация функции запуска контейнера и реализация одного из тестов приведены в Приложении В.

3.7 Вывод

В данном разделе был описан выбор систем управления базами данных. Были выбраны СУБД Postgres и ClickHouse, как наиболее полно удовлетворяющие поставленным задачам. Для аналитических данных будет использована СУБД ClickHouse, так как колоночные СУБД имеют большую производительность на запросах агрегации, для иных — Postgres. Также, описан выбор средств разработки серверной и клиентской частей ПО, для которых были выбраны Golang и Vue.js соответственно. Приведена таблица, описывающая интерфейс взаимодействия серверной и клиентской частей ПО и представлена общая схема взаимодействия компонентов системы. Описаны методы тестирования и обеспечения безопасности данных.

4 Исследовательский раздел

4.1 Постановка исследования

Целью исследования является сравнение времени выполнения агрегационных запросов с варьирующей сложностью фильтрации данных в СУБД Postgres и ClickHouse. Исследование проводилось при количестве фильтров от 0, когда фильтрации не происходит, до 9. Даже на меньшем диапазоне разница в скорости обработки подобных запросов становится заметной.

Запросы выполнялись одновременно на разных серверах с одинаковыми характеристиками. Технические характеристики устройства, на котором выполнялось исследование [44]:

- операционная система Ubuntu 22.04;
- 4 ГБ оперативной памяти;
- процессор с тактовой частотой 3.3 ГГц.

По результатам исследования была составлена сравнительная таблица и построен график зависимости времени выполнения запроса на агрегацию данных от количества применяемых к данным фильтров.

4.2 Результаты исследования

Результаты замеров времени выполнения (в мс) приведены в таблице 4.1. Количество фильтров равное нулю означает, что фильтрация данных не производилась, то есть были получены все данные, хранимые в таблице. При замерах в таблице событий находилось 100 000 000 событий и было зарегистрировано 1000 клиентов.

Таблица 4.1 – Таблица сравнения времени выполнения агрегационных запросов с варьирующей сложностью фильтрации данных в СУБД Postgres и ClickHouse

Количество фильтров	Время выполнения запроса в Postgres, мс	Время выполнения запроса в ClickHouse, мс
0	29325	385
1	28224	4322
2	51671	7966
3	76943	11947
4	94394	15959
5	117422	20059
6	143793	24127
7	171238	28409
8	193777	32350
9	210269	36201

Зависимость времени выполнения агрегационных запросов от числа фильтров в СУБД Postgres и ClickHouse приведена на рисунке 4.1.

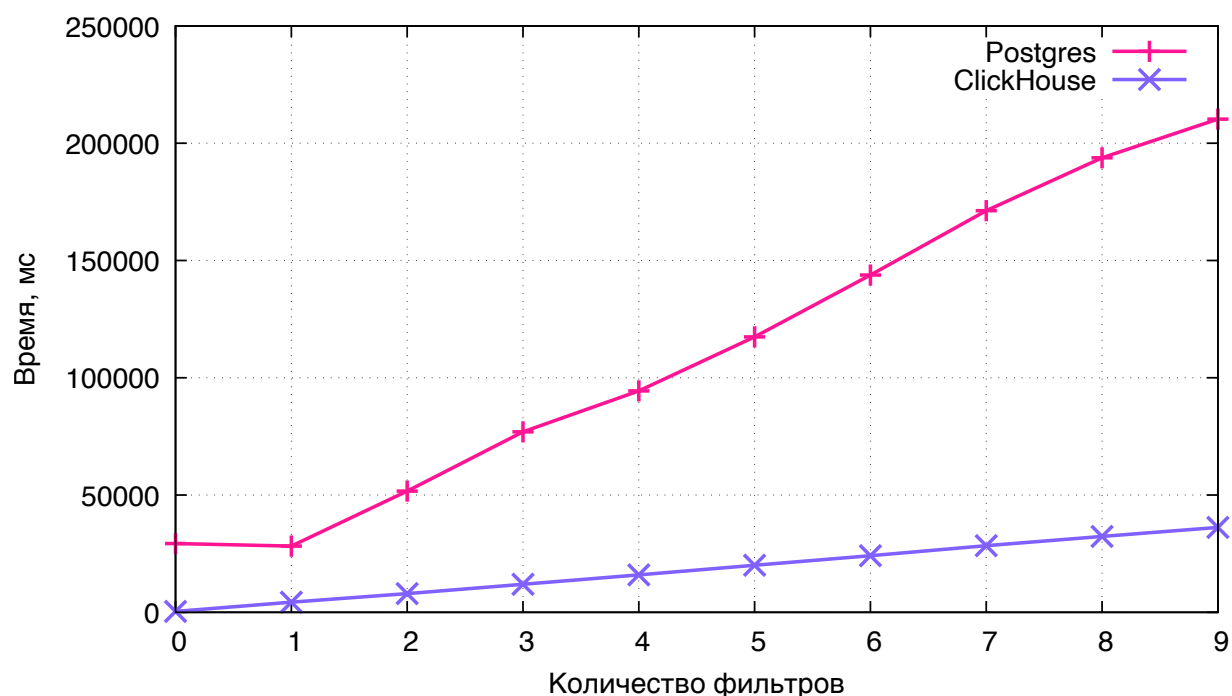


Рисунок 4.1 – Сравнение времени выполнения агрегационных запросов в зависимости от числа фильтров в СУБД Postgres и ClickHouse

4.3 Вывод

В данном разделе было описано проведенное исследование производительности двух используемых в работе СУБД — Postgres и ClickHouse — при выполнении агрегационных запросов. Результаты исследования показывают, что агрегационные запросы более эффективно выполняются в ClickHouse. Без фильтрации запрос в ClickHouse выполняется быстрее запроса в Postgres в 76 раз. При 9 фильтрах это разница составляет 5 раз.

Также, зависимость скорости выполнения запроса от количества фильтров в ClickHouse практически линейная. При этом запрос в Postgres без фильтрации выполняется дольше, чем запрос с одним фильтром, что может быть связано с тем, что в первом случае необходимо было получить большее количество записей.

Данное исследование показывает, что колоночные СУБД обрабатывают агрегационные запросы эффективнее строковых. Это объясняется тем, что агрегация чаще всего производится по значениям одного столбца. Доступ к данным одного столбца оптимизирован в СУБД колоночного типа в связи с особенностями хранения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были решены следующие задачи:

- 1) Был проведён анализ предметной области и сравнение предложенного решения с существующими.
- 2) Были формализованы варианты использования и предоставляемые сервисом возможности и описана информация, подлежащая хранению в базе данных.
- 3) Был проведён выбор модели данных, базы данных по способу хранения и системы управления базой данных по способу доступа к базе данных.
- 4) Была составлена ER-диаграмма сущностей проектируемой базы данных в нотации Чена и спроектированы сущности базы данных и накладываемые ограничения целостности.
- 5) Была описана проектируемая ролевая модель на уровне базы данных и определены права доступа к внутренним структурам.
- 6) Был обоснован выбор средств реализации базы данных и приложения и описан интерфейс доступа к базе данных.
- 7) Был проведён сравнительный анализ различных систем управления базой данных для хранения и обработки аналитических данных.

Таким образом, была достигнута цель работы: была разработана база данных для сервиса по почтовой рассылке персонализированных предложений.

В будущем разработанную программу можно будет усовершенствовать, добавив больше типов фильтрации, возможность анализировать действия клиентов с отправленными письмами и расширив функционал по редактированию шаблона рекламных писем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпова И. П. Базы данных. Учебное пособие. — 2009.
2. Цифровая экономика= модели данных+ большие данные+ архитектура+ приложения? / В. Куприяновский [и др.] // International journal of open information technologies. — 2016. — Т. 4, № 5. — С. 1—13.
3. What is a message broker? [Электронный ресурс]. — Режим доступа: <https://www.ibm.com/topics/message-brokers> (дата обращения: 13.05.2023).
4. Wu H., Shang Z., Wolter K. Performance prediction for the apache kafka messaging system // 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). — IEEE. 2019. — С. 154—161.
5. Движки таблиц ClickHouse [Электронный ресурс]. — Режим доступа: <https://clickhouse.com/docs/ru/engines/table-engines> (дата обращения: 13.05.2023).
6. Иванов В. В., Лубова Е. С., Черкасов Д. Ю. Аутентификация и авторизация // Проблемы современной науки и образования. — 2017. — 2 (84). — С. 31—33.
7. Развитие интернет-торговли в России в условиях цифровизации / М. Кудинова [и др.] // Инновации и инвестиции. — 2022. — № 4. — С. 238—243.
8. Чернявская Е. Ю. Форматы и преимущества рекламы в сети Интернет // Вестник Белгородского университета кооперации, экономики и права. — 2013. — № 2. — С. 375—378.
9. DataReportal. Digital 2022 Global Digital Overview [Электронный ресурс]. — Режим доступа: <https://datareportal.com/reports/digital-2022-global-overview-report> (дата обращения: 13.03.2023).
10. Ассоциация компаний интернет-торговли (АКИТ). Сводные аналитические данные [Электронный ресурс]. — Режим доступа: <http://www.akit.ru/analytics/analyt-data> (дата обращения: 13.03.2023).

11. *Плеханов С. В., Прытков А. Н., Бабаев С. Э.* Персонализация как неотъемлемый аспект деятельности современной организации // Парадигмы управления, экономики и права. — 2020. — 1 (3). — С. 49.
12. Федеральный закон от 13.03.2006 N 38-ФЗ (ред. от 05.12.2022) "О рекламе". — 2022.
13. Федеральный закон от 07.07.2003 N 126-ФЗ (последняя редакция) "О связи". — 2003.
14. Постановление Пленума ВАС РФ от 08.10.2012 N 58 "О некоторых вопросах практики применения арбитражными судами Федерального закона "О рекламе". — 2012.
15. *Акимова И., Гаспарян Э.* Digital-реклама: запретить нельзя узаконить // Конкуренция и право. — 2016. — Т. 6. — С. 30.
16. Unisender. О нас [Электронный ресурс]. — Режим доступа: <https://www.unisender.com/ru/about/> (дата обращения: 13.03.2023).
17. Mindbox. О компании [Электронный ресурс]. — Режим доступа: <https://mindbox.ru/company/> (дата обращения: 13.03.2023).
18. Статистика Sendsay [Электронный ресурс]. — Режим доступа: <https://sendsay.ru/clients> (дата обращения: 13.03.2023).
19. *Дуго С.* Базы данных // Проектирование и создание. Учебно-методический комплекс. М.: Изд. Центр УАОИ. — 2008.
20. *Бородин Д., Строганов Ю.* К задаче составления запросов к базам данных на естественном языке // Новые информационные технологии в автоматизированных системах. — 2016. — № 19. — С. 119—126.
21. *Корягин Д.* МОДЕЛИ БАЗ ДАННЫХ // Аллея науки. — 2020. — Т. 1, № 5. — С. 985—988.
22. *Попов И., Сахнов Г.* Преимущества использования колоночных СУБД в системах бизнес-аналитики // Научные труды Вольного экономического общества России. — 2012. — Т. 164. — С. 339—345.
23. *Дынер А.* Система управления файл-серверными базами данных // Труды молодых ученых Алтайского государственного университета. — 2010. — № 7. — С. 119—120.

24. *Соколов Я. А.* Разработка клиент-серверной СУБД // Старт в науку: актуальные вопросы техники и технологий. — 2019. — С. 17—24.
25. *Копытов А. В., Адаманский А. В.* Реализация шифрования данных во встраиваемой СУБД EJDB // Вестник Новосибирского государственного университета. Серия: Информационные технологии. — 2014. — Т. 12, № 2. — С. 48—54.
26. *Douglas K., Douglas S.* PostgreSQL: a comprehensive guide to building, programming, and administering PostgreSQL databases. — 2003.
27. Отличительные возможности ClickHouse [Электронный ресурс]. — Режим доступа: <https://clickhouse.com/docs/ru/introduction/distinctive-features> (дата обращения: 13.05.2023).
28. Документация движка Merge Tree ClickHouse [Электронный ресурс]. — Режим доступа: <https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/mergetree> (дата обращения: 15.04.2023).
29. Документация движка Kafka ClickHouse [Электронный ресурс]. — Режим доступа: <https://clickhouse.com/docs/ru/engines/table-engines/integrations/kafka> (дата обращения: 15.04.2023).
30. Интеграция Kafka с ClickHouse [Электронный ресурс]. — Режим доступа: <https://clickhouse.com/docs/ru/engines/table-engines/integrations/kafka> (дата обращения: 15.04.2023).
31. Apache Kafka documentation [Электронный ресурс]. — Режим доступа: <https://kafka.apache.org/documentation/> (дата обращения: 13.05.2023).
32. Документация по языку программирования Go [Электронный ресурс]. — Режим доступа: <https://go.dev/doc/> (дата обращения: 13.05.2023).
33. Документация пакета snet/http языка Go [Электронный ресурс]. — Режим доступа: <https://pkg.go.dev/net/http> (дата обращения: 13.05.2023).
34. Документация библиотеки GORM для Golang [Электронный ресурс]. — Режим доступа: <https://gorm.io/docs/> (дата обращения: 13.05.2023).

35. Документация библиотеки jwt-go для Golang [Электронный ресурс]. — Режим доступа: <https://pkg.go.dev/github.com/golang-jwt/jwt> (дата обращения: 15.04.2023).
36. Документация библиотеки bcrypt для Golang [Электронный ресурс]. — Режим доступа: <https://pkg.go.dev/golang.org/x/crypto/bcrypt> (дата обращения: 15.04.2023).
37. Документация по Brevo [Электронный ресурс]. — Режим доступа: https://www.brevo.com/?tap_a=30591-fb13f0&tap_s=267844-e77da3 (дата обращения: 15.04.2023).
38. Руководство по использованию Vue.js [Электронный ресурс]. — Режим доступа: <https://ru.vuejs.org/v2/guide/> (дата обращения: 15.04.2023).
39. Сравнение популярных фреймворков с Vue.js [Электронный ресурс]. — Режим доступа: <https://ru.vuejs.org/v2/guide/comparison.html> (дата обращения: 15.04.2023).
40. Документация по Vuex [Электронный ресурс]. — Режим доступа: <https://vuex.vuejs.org> (дата обращения: 15.04.2023).
41. Документация по Swagger [Электронный ресурс]. — Режим доступа: <https://swagger.io> (дата обращения: 15.04.2023).
42. Документация по OpenAPI Generator [Электронный ресурс]. — Режим доступа: <https://openapi-generator.tech> (дата обращения: 15.04.2023).
43. Документация по JWT-токенам [Электронный ресурс]. — Режим доступа: <https://jwt.io> (дата обращения: 15.04.2023).
44. Сервис по аренде виртуальных серверов [Электронный ресурс]. — Режим доступа: <https://timeweb.cloud> (дата обращения: 15.04.2023).

ПРИЛОЖЕНИЕ А

SQL-скрипты

В листингах А.1 – А.7 приведены листинги SQL-скриптов для создания таблиц, функций, ролей и реализованные запросы для СУБД Postgres и ClickHouse.

Листинг А.1 – SQL-скрипт создания таблиц в СУБД Postgres

```
1 create extension "uuid-oss";
2
3 create table users (
4     uuid uuid default uuid_generate_v4() primary key,
5     login text unique not null,
6     password text not null,
7     is_admin bool default false
8 );
9
10 create type gender as enum (
11     'female',
12     'male'
13 );
14
15 create table clients (
16     uuid uuid default uuid_generate_v4() primary key,
17     name text not null,
18     surname text not null,
19     patronymic text,
20     gender gender not null,
21     birth_date timestamp not null check (birth_date <
22         current_timestamp::timestamp and birth_date >
23         '01.01.1900 00:00:00'::timestamp),
24     registration_date timestamp not null check
25         (registration_date < current_timestamp::timestamp and
26         registration_date > clients.birth_date),
27     email text unique not null check ( email like '%@%.%' ),
28     data json
29 );
30
31 create table event_types (
32     uuid uuid default uuid_generate_v4() primary key,
33     name text not null,
```

```

30     alias text unique not null check ( length(alias) > 5 )
31 );
32
33 create table schedules (
34     uuid uuid default uuid_generate_v4() primary key,
35     next_time timestamp not null,
36     finished boolean default false not null,
37     periodic boolean default false not null,
38     span text not null
39 );
40
41 create table ads (
42     uuid uuid default uuid_generate_v4() primary key,
43     content text not null,
44     filters json ,
45     user_id uuid references users (uuid) on delete set null,
46     schedule_id uuid unique references schedules (uuid) on
         delete cascade,
47     creation_time timestamp not null check (creation_time <
         current_timestamp::timestamp)
48 );
49
50 create table events (
51     uuid uuid default uuid_generate_v4() primary key,
52     time timestamp not null,
53     client_id uuid references clients (uuid) on delete set
         null,
54     alias text references event_types (alias) not null check
         (length(alias) > 5)
55 );

```

Листинг А.2 – SQL-скрипт создания ролей в системе управления базами данных Postgres

```

1 create role admin login password 'password1';
2 create role targetologist login password 'password2';
3 create role client login password 'password3';
4
5 grant insert on clients to client;
6 grant insert on events to client;
7
8 grant insert, select on users to targetologist;
9 grant insert, select on ads to targetologist;

```



```

10 grant insert, select on event_types to targetologist;
11 grant select on events to targetologist;
12 grant insert, select, update on schedules to targetologist;
13 grant select on clients to targetologist;
14
15 grant insert, select, delete, update on users to admin;
16 grant select, delete on clients to admin;

```

Листинг А.3 – SQL-скрипт создания таблицы events в СУБД ClickHouse

```

1 CREATE TABLE coursework.events_queue (
2     id UUID,
3     client_id UUID,
4     alias String,
5     time DateTime('UTC')
6 ) ENGINE = Kafka('127.0.0.1:9092', 'blurby-events',
7     'clickhouse-blurby-events', 'JSONEachRow');
8
9 CREATE TABLE coursework.events
10 (
11     id UUID,
12     client_id UUID,
13     alias String,
14     time DateTime('UTC')
15 ) ENGINE = MergeTree ORDER BY (time);
16
17 CREATE MATERIALIZED VIEW coursework.events_mv TO
18     coursework.events AS
19     SELECT *
20     FROM coursework.events_queue;

```

Листинг А.4 – SQL-скрипт создания ролей в СУБД ClickHouse

```

1 CREATE USER targetologist IDENTIFIED WITH sha256_password BY
2     'password1';
3
4 CREATE ROLE targetologist_role;
5 GRANT SELECT ON coursework.events TO targetologist_role;
6 GRANT SELECT ON coursework.ad_send_times TO targetologist_role;
7
8 GRANT targetologist_role TO targetologist;

```

Листинг А.5 – SQL-скрипт табличной функции в СУБД Postgres

```

1 create or replace function get_ads_by_span(finish timestamp)
    returns table(
2     uuid uuid,
3     content text,
4     filters json,
5     user_id uuid,
6     schedule_id uuid,
7     creation_time timestamp,
8     next_time timestamp,
9     span text,
10    finished boolean,
11    periodic boolean
12 )
13 as $$
14 begin
15     return query (
16         select a.uuid as uuid, a.content, a.filters, a.user_id,
            a.schedule_id, a.creation_time, s.next_time, s.span,
            s.finished, s.periodic
17         from ads a join schedules s on s.uuid = a.schedule_id
18         where s.next_time <= finish and s.finished = false
19     );
20 end;
21 $$
22     language plpgsql;

```

Листинг А.6 – SQL-скрипт запроса на фильтрацию пользователей в СУБД ClickHouse

```

1 select distinct client_id
2     from (
3         select
4             client_id,
5             alias,
6             count(*) as num
7         from coursework.events
8         where time between now() – interval 5 hour and now()
9         group by client_id, alias
10        order by num desc
11        limit 5 by client_id
12    ) where alias = 'alias'

```

Листинг А.7 – SQL-скрипт запроса на фильтрацию пользователей в СУБД Postgres

```
1 select distinct cast sub1.client_id
2   from (
3     select distinct sub.client_id , sub.alias , sub.c ,
4         row_number() over (partition by sub.client_id
5                               order by sub.c desc) as r
6   from (
7     select client_id , alias , count(*) as c from events
8       where time between current_timestamp - '5
9             hours '::interval and current_timestamp
10    group by events.client_id , alias) as sub
11  ) as sub1
12  where alias = 'alias ' and r <= 5
```

ПРИЛОЖЕНИЕ Б

Реализованный web-интерфейс

На рисунках Б.1 – Б.5 представлен web-интерфейса.

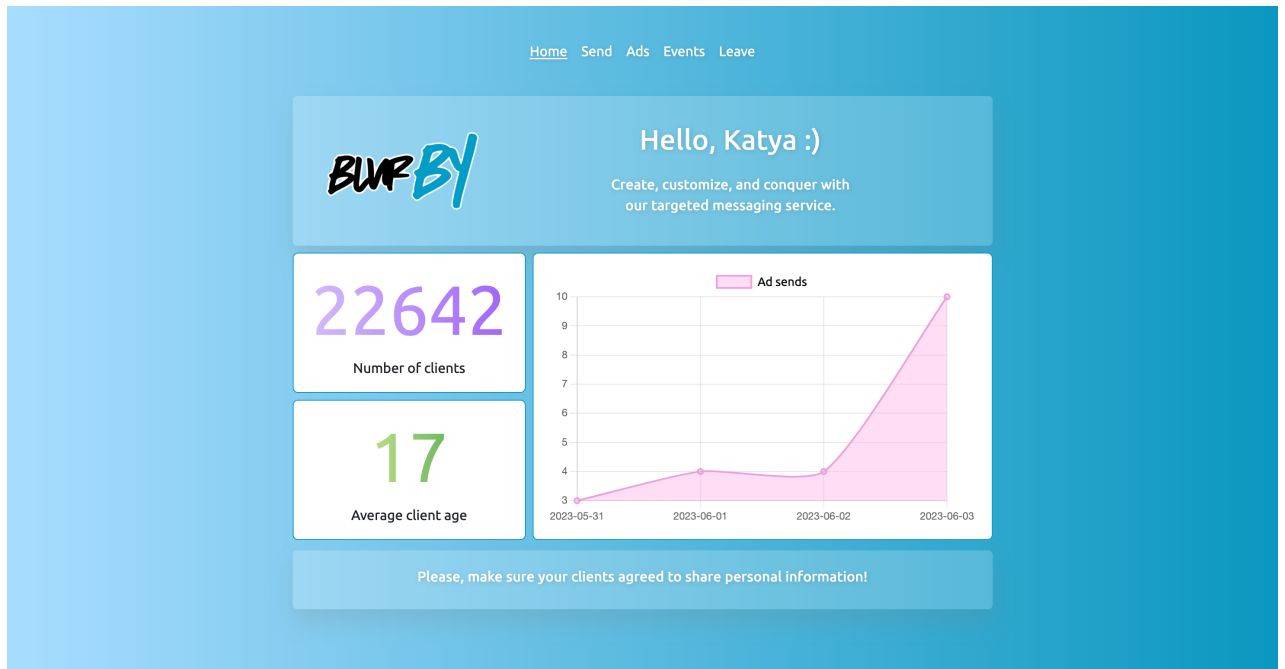


Рисунок Б.1 – Домашняя страница

The screenshot shows the 'Active ads' page. At the top, the navigation bar includes 'Home', 'Send', 'Ads' (which is active), 'Events', and 'Leave'. The page title is 'Active ads'. Below the title, there is a form for creating or editing an ad. The form contains the following sections:

- Content:** A text box with 'Hello, world!' and a label 'informative message' below it.
- Create time:** A field showing '03.06.2023, 18:04:46'.
- Schedule:** A table with the following data:

Finished:	false
Periodic:	true
Span:	1m0s
Next time:	03.06.2023, 18:05:46
- Filters:** A table with the following data:

Type:	by field
Field:	name
Comparison:	=
Value #1:	Lesha

Рисунок Б.2 – Страница просмотра списка незавершённых рассылок

Home Send Ads Events Leave

Create an event type

Alias

Description

Create

Event types


Name:	aaa
Alias:	fzdrT9zg6qvghbt

Name:	aaa
Alias:	9mcwhgf9jvkdvf4

Рисунок Б.3 – Страница просмотра и создания типов событий

Home Users Clients Leave


Clients

 a a a
a@a.ru

Actions ▾

Birth date: 01.02.2006, 17:00:00


Registration date: 31.05.2023, 18:21:07

 a a a
a@a.ru

Actions ▾

Birth date: 01.02.2006, 17:00:00


Registration date: 31.05.2023, 18:21:07

 a a a
a@a.ru

Actions ▾

Birth date: 01.02.2006, 17:00:00

Registration date: 31.05.2023, 18:21:08

 a a a
a@a.ru

Actions ▾

Birth date: 01.02.2006, 17:00:00

Registration date: 31.05.2023, 18:21:08

Рисунок Б.4 – Страница просмотра и редактирования списка клиентов

[Home](#) [Send](#) [Ads](#) [Events](#) [Leave](#)

Create an advertisement

Ad content in HTML

☐ Preview

Schedule preferences

☐ Send periodically

Span (m:h:d):

0

0

0

Filtering preferences

Field filter

Value #1

Add filter

Actions

Create ad

Ready filters

No ready filters

Рисунок Б.5 – Страница формирования рекламной рассылки

ПРИЛОЖЕНИЕ В

Тестирование

В листингах В.1 – В.7 приведены листинги функций необходимых для тестирования разработанного ПО.

Листинг В.1 – Модульный тест для функции входа в систему

```
1 func TestProfile_Enter(t *testing.T) {
2     mc := minimock.NewController(t)
3     password := "password"
4     hash, _ := bcrypt.GenerateFromPassword([]byte(password),
5         bcrypt.DefaultCost)
6     user := &models.User{
7         ID:      uuid.New(),
8         Login:    "uehfkjsyg",
9         Password: string(hash),
10        IsAdmin: false,
11    }
12    type fields struct {
13        userRepo      interfaces.UserRepository
14        dailyBonus     int
15        tokenExpiration time.Duration
16        secretKey      string
17    }
18    type args struct {
19        ctx      context.Context
20        login    string
21        password string
22    }
23    tests := []struct {
24        name      string
25        fields    fields
26        args      args
27        want      string
28        wantErr   bool
29    }{
30        {
31            name: "successful enter",
32            fields: fields{
33                userRepo:
34                    mocks.NewUserRepositoryMock(mc).GetByLoginMock.
35                    Return(user, nil),
```

```

33         dailyBonus:      5,
34         tokenExpiration: time.Hour,
35         secretKey:        "secretkey",
36     },
37     args: args{
38         ctx:      context.Background(),
39         login:     "uehfkjsyg",
40         password: password,
41     },
42     want:        "",
43     wantErr: false,
44 },
45 }
46 for _, tt := range tests {
47     t.Run(tt.name, func(t *testing.T) {
48         p := &Profile{
49             userRepository: tt.fields.userRepo,
50             tokenExpiration: tt.fields.tokenExpiration,
51             secretKey:        tt.fields.secretKey,
52         }
53         _, err := p.Login(tt.args.ctx, tt.args.login,
54             tt.args.password)
55         if (err != nil) != tt.wantErr {
56             t.Errorf("Login() error = %v, wantErr %v", err,
57                 tt.wantErr)
58             return
59         }
60     })
61 }

```

Листинг В.2 – SQL-скрипт заполнения БД тестовыми данными используемыми в интеграционных тестах

```

1 insert into clients (uuid, name, surname, patronymic, gender,
   birth_date, registration_date, email, data) values
2 ('a52b8aea-d751-4933-91bb-691132e3b760', '1', '1', '1', 'male',
   '2006-01-02 15:04:05', '2009-01-02 15:04:05', '1@mail.ru',
   null),
3 ('3f010aca-5008-4aa5-a1a3-a061a876783f', '2', '2', '2', 'male',
   '2007-01-02 15:04:05', '2009-01-02 15:04:05', '2@mail.ru',
   null),
4 ('77dcd288-79b2-4655-9584-cc9b5329665d', '3', '3', '3',

```



```
'female', '2008-01-02 15:04:05', '2009-01-02 15:04:05',  
'3@mail.ru', null);
```

Листинг В.3 – Интеграционный тест для функции получения информации о клиенте

```
1 func TestClient_Get(t *testing.T) {  
2     dbContainer, db, err := containers.SetupTestDatabase()  
3     if err != nil {  
4         t.Fatal(err)  
5     }  
6     defer func() { _ =  
7         dbContainer.Terminate(context.Background()) }()  
8  
9     cr := postgres.NewCR(db, db, db)  
10  
11     user := &models.User{  
12         ID:      uuid.New(),  
13         Login:    "uehfkjsyg",  
14         Password: "r3",  
15         IsAdmin:  true,  
16     }  
17  
18     userNA := &models.User{  
19         ID:      uuid.New(),  
20         Login:    "uehfkjsyg",  
21         Password: "r3",  
22         IsAdmin:  false,  
23     }  
24  
25     id, err := uuid.Parse("77dcd288-79b2-4655-9584-cc9b5329665d")  
26     if err != nil {  
27         t.Errorf("uuid parse: %v", err)  
28         return  
29     }  
30  
31     idNE, err :=  
32         uuid.Parse("77dcd288-79b2-4655-9584-cc9b5329665a")  
33     if err != nil {  
34         t.Errorf("uuid parse: %v", err)  
35         return  
36     }  
37 }
```

Листинг В.4 – Интеграционный тест для функции получения информации о клиенте (продолжение)

```
1
2  bd, err := time.Parse(time.RFC3339, "2008-01-02T15:04:05Z")
3  if err != nil {
4      t.Errorf("bd parse: %v", err)
5      return
6  }
7
8  rd, err := time.Parse(time.RFC3339, "2009-01-02T15:04:05Z")
9  if err != nil {
10     t.Errorf("bd parse: %v", err)
11     return
12 }
13
14 client := &models.Client{
15     ID:          id,
16     Name:        "3",
17     Surname:     "3",
18     Patronymic:  "3",
19     Gender:      "female",
20     BirthDate:   bd,
21     RegistrationDate: rd,
22     Email:       "3@mail.ru",
23     Data:        nil,
24 }
25
26 type fields struct {
27     clientRepository interfaces.ClientRepository
28 }
29 type args struct {
30     ctx context.Context
31     id  uuid.UUID
32 }
33 tests := []struct {
34     name      string
35     fields    fields
36     args      args
37     want      *models.Client
38     wantErr   bool
39 }{
```

```

40     {
41         name:    "successful get test",
42         fields:  fields{cr},
43         args:    args{
44             ctx: mycontext.UserToContext(context.Background(),
45                 user),
46             id:  id,
47         },
48         want:    client,
49         wantErr: false,
50     },
51 }
52 for _, tt := range tests {
53     t.Run(tt.name, func(t *testing.T) {
54         c := &Client{
55             clientRepository: tt.fields.clientRepository,
56         }
57         got, err := c.Get(tt.args.ctx, tt.args.id)
58         if (err != nil) != tt.wantErr {
59             t.Errorf("Get() error = %v, wantErr %v", err,
60                 tt.wantErr)
61             return
62         }
63         if !reflect.DeepEqual(got, tt.want) {
64             t.Errorf("Get() got = %v, want %v", got, tt.want)
65         }
66     })
67 }

```

Листинг В.5 – Функция запуска контейнера для интеграционного тестирования

```

1 func SetupTestDatabase() (testcontainers.Container, *gorm.DB,
2     error) {
3     containerReq := testcontainers.ContainerRequest{
4         Image:    "postgres:latest",
5         ExposedPorts: []string{"5432/tcp"},
6         WaitingFor: wait.ForListeningPort("5432/tcp"),
7         Env: map[string]string{
8             "POSTGRES_DB":    "testdb",
9             "POSTGRES_PASSWORD": "postgres",

```

Листинг В.6 – Функция запуска контейнера для интеграционного тестирования (продолжение)

```
1         "POSTGRES_USER":      "postgres",
2     },
3 }
4
5 dbContainer, err := testcontainers.GenericContainer(
6     context.Background(),
7     testcontainers.GenericContainerRequest{
8         ContainerRequest: containerReq,
9         Started:          true,
10    })
11 if err != nil {
12     return nil, nil, fmt.Errorf("generic container: %w", err)
13 }
14
15 host, err := dbContainer.Host(context.Background())
16 if err != nil {
17     return nil, nil, fmt.Errorf("host: %w", err)
18 }
19
20 port, err := dbContainer.MappedPort(context.Background(),
21     "5432")
22 if err != nil {
23     return nil, nil, fmt.Errorf("port: %w", err)
24 }
25 dsn := fmt.Sprintf("host=%s port=%d user=postgres
26     password=postgres dbname=testdb sslmode=disable", host,
27     port.Int())
28 pureDB, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
29 if err != nil {
30     return nil, nil, fmt.Errorf("gorm open: %w", err)
31 }
32
33 sqlDB, err := pureDB.DB()
34 if err != nil {
35     return nil, nil, fmt.Errorf("get db: %w", err)
36 }
```

Листинг В.7 – Функция запуска контейнера для интеграционного тестирования (продолжение)

```
1   if err = goose.Up(sqlDB ,
2       "../..../deployments/migrations/postgres"); err != nil {
3       return nil, nil, fmt.Errorf("up migrations: %w", err)
4   }
5   text, err :=
6       os.ReadFile("../..../internal/containers/data.sql")
7   if err != nil {
8       return nil, nil, fmt.Errorf("read file: %w", err)
9   }
10  if err := pureDB.Exec(string(text)).Error; err != nil {
11      return nil, nil, fmt.Errorf("exec: %w", err)
12  }
13
14  return dbContainer, pureDB, nil
15 }
```