Оглавление

Введение

Бизнес-модель подписки – это бизнес-модель, в которой клиент должен регулярно (с определенной периодичностью) платить фиксированную сумму, чтобы получить доступ к продукту [1].

По состоянию на 2021 год 55% людей в мире имеют подписку на услуги более чем одного сервиса из «большой тройки» (Netflix, Amazon Prime, Hulu) [2]. И это только сервисы видеостриминга. Около 400 млн. людей имеют подписку на сервисы аудиостриминга [3].

Пандемия COVID-19 в 2020 году положительно повлияла на развитие бизнес-модели подписки: количество поисковых запросов о подписках на видеостриминговые сервисы в среднем выросло на 102% [4], а количество пользователей, приобретших подписки на хотя бы один такой сервис, выросло в среднем на 20% [5].

Несмотря на то, что пандемия отступает, бизнес-модель подписки только развивает свою популярность, приобретенную в период пандемии. По прогнозам, потраченные пользователями в 2021 году 4054 млн. долларов на подписки вырастут до 7813 млн. долларов в 2025 году [6].

Чтобы иметь полный контроль над денежными потоками в приложениях с моделью подписок, бизнесу нужны соответствующие инструменты, чтобы анализировать, планировать и прогнозировать движения средств.

Учесть течение финансов можно при помощи метрики MRR (Monthly Reccuring Revenue) [7]. Существуют инструменты, которые позволяют вычислять значение данной метрики для заданного набора данных. Основной их недостаток в том, что они коммерческие и доступны не каждому бизнесу, желающему провести анализ и последующий прогноз финансовых потоков.

Цель работы – реализовать программное обеспечение для определения MRR приложения по заданному набору данных.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- определить методы вычисления MRR;
- проанализировать варианты представления данных для аналитики и выбрать подходящий вариант для решения задачи;

- проанализировать системы управления базами данных и выбрать подходящую систему для хранения данных;
- спроектировать базу данных, описать ее сущности, связи;
- реализовать интерфейс для доступа к базе данных;
- реализовать программное обеспечение, которое предоставит пользователю доступ к построенным отчетам в графической и табличной форме.

1 Аналитическая часть

В данном разделе описана метрика MRR, постановка задачи построения MRR отчета. Представлен анализ способов хранения данных и систем управления базами данных, оптимальных для решения поставленной задачи. Приведен анализ существующих решений.

1.1 Формализация задачи

Принятие решений на основе данных является обязательным условием при развитии бизнеса. Однако подход основанный на данных сильно зависит от самих данных, правильности их сбора и обработки. Зачастую, бизнес не умеет собирать нужные данные правильно, также он сталкивается с проблемами их обработки.

Ключевые данные, необходимые для принятия решений, это данные о продажах, а именно, кто, когда, сколько платит. При этом, в современном мире данные о платежах обычно имеются в любом бизнесе.

В качестве примера будут рассматриваться приложения с бизнес-моделью подписки, которые имеют отличительную особенность – рекуррентные платежи. Вести учет движения средств в такой модели, используя стандартный бухгалтерский подход, неправильно, так как он не учитывает будущие поступления, кроме того бухгалтерский подход нормирует все поступления на сутки, а следовательно при оплате клиентом месячной подписки, например, с 20 января по 19 февраля получится вклад от услуги в оба месяца, при этом вклад в каждый месяц будет рассчитан пропорционально числу дней в месяце. При этом клиент понимает, что подписка была оформлена в январе, а следующий платеж будет совершен в феврале. Для учета таких платежей придумали отчет MRR, в котором платежи нормируются на месяц начала действия оплаченного периода, таким образом, оплаченный период с 20 января по 19 февраля целиком будет относиться к январю.

1.2 Метрика MRR

Monthly Recurring Revenue – метрика для принятия решений, отображающая регулярную месячную выручку. Применяется в SaaS-приложениях [8] с бизнес-моделью подписок [7].

1.2.1 Подготовка данных для MRR отчета

Для того, чтобы построить MRR отчет, в исходных данных обязательно должна быть следующая информация:

- `customer_id` – идентификатор клиента, уникальное значение, определяющее каждого отдельного клиента;
- `paid_amount` – сумма, заплаченная клиентом;
- `paid_plan` – тип тарифного плана (месячный, трехмесячный, полугодовой);
- `period_start` – дата выставления счета (начало периода);

В таблице 1.1 приведен пример начальной выгрузки данных.

Таблица 1.1 – Начальная выгрузка данных

customer_id	**period_start**	**paid_plan**	**paid_amount**
1	01.06.2019	annually	90
2	01.06.2019	annually	90
3	01.06.2019	monthly	10
3	01.08.2019	monthly	10
4	01.06.2019	monthly	10
5	01.06.2019	monthly	10
6	01.06.2019	monthly	10
7	01.06.2019	annually	90
8	01.06.2019	monthly	10
9	01.06.2019	monthly	10
10	01.06.2019	monthly	10
11	01.06.2019	monthly	10
12	01.06.2019	annually	90
13	01.06.2019	annually	90
14	01.06.2019	monthly	10

Для построения отчета необходимо вычислить дату окончания оплаченного периода. Для этого необходимо правильно учитывать количество дней в месяцах, фактор високосного года и т.п.

В таблице 1.2 приведен пример доработанной выгрузки данных.

Таблица 1.2 – Доработанная выгрузка данных

customer_id	**period_start**	**paid_plan**	**paid_amount**	**period_end**
1	01.06.2019	annually	90	31.05.2020
2	01.06.2019	annually	90	31.05.2020
3	01.06.2019	monthly	10	30.06.2019
3	01.08.2019	monthly	10	31.08.2019
4	01.06.2019	monthly	10	30.06.2019
5	01.06.2019	monthly	10	30.06.2019
6	01.06.2019	monthly	10	30.06.2019
7	01.06.2019	annually	90	31.05.2020
8	01.06.2019	monthly	10	30.06.2019
9	01.06.2019	monthly	10	30.06.2019
10	01.06.2019	monthly	10	30.06.2019
11	01.06.2019	monthly	10	30.06.2019
12	01.06.2019	annually	90	31.05.2020
13	01.06.2019	annually	90	31.05.2020
14	01.06.2019	monthly	10	30.06.2019

После вычисления даты окончания периода необходимо создать промежуточный набор данных, в котором будет учтено распределение уплаченных средств согласно периодам (нормированным на начало месяца) в которые они попадают. Фактически этот набор данных может представлять из себя таблицу, в которой строки – идентификаторы клиентов, столбцы – периоды, в ячейках которых указывается сумма, уплаченная клиентом за этот период. Сумма вычисляется в зависимости от того, какой тарифный план использует клиент.

В таблице 1.3 приведен пример таблицы распределения уплаченных средств согласно периодам.

Таблица 1.3 – Распределение уплаченных средств согласно периодам

	06.2019	07.2019	08.2019	09.2019	10.2019	11.2019	12.2019	01.2020	02.2020	03.2020	04.2020	05.2020
1	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
2	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
3	10	0	0	0	0	0	0	0	0	0	0	0
3	0	0	10	0	0	0	0	0	0	0	0	0
4	10	0	0	0	0	0	0	0	0	0	0	0
5	10	0	0	0	0	0	0	0	0	0	0	0
6	10	0	0	0	0	0	0	0	0	0	0	0
7	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
8	10	0	0	0	0	0	0	0	0	0	0	0
9	10	0	0	0	0	0	0	0	0	0	0	0
10	10	0	0	0	0	0	0	0	0	0	0	0
11	10	0	0	0	0	0	0	0	0	0	0	0
12	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
13	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
14	10	0	0	0	0	0	0	0	0	0	0	0

Важно понимать, что в полученном наборе данных могут быть клиенты, которые платили несколько раз (например, месячная подписка с разницей в месяц). Такие клиенты будут иметь несколько строчек в таблице. В таблице 1.3 есть такой случай – это клиент с идентификатором `3`.

Теперь необходимо свести этот отчет к каждому уникальному клиенту, чтобы считать динамику MRR (`new`, `old`, `reactivation`, `expansion`, `contraction`, `churn`. Подробнее о каждом критерии будет сказано далее).

В таблице 1.4 приведен пример нормированной по клиентам таблицы распределения уплаченных средств согласно периодам.

Таблица 1.4 – Нормированное распределение уплаченных средств согласно периодам

	06.2019	07.2019	08.2019	09.2019	10.2019	11.2019	12.2019	01.2020	02.2020	03.2020	04.2020	05.2020
1	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
2	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
3	10	0	10	0	0	0	0	0	0	0	0	0
4	10	0	0	0	0	0	0	0	0	0	0	0
5	10	0	0	0	0	0	0	0	0	0	0	0
6	10	0	0	0	0	0	0	0	0	0	0	0
7	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
8	10	0	0	0	0	0	0	0	0	0	0	0
9	10	0	0	0	0	0	0	0	0	0	0	0
10	10	0	0	0	0	0	0	0	0	0	0	0
11	10	0	0	0	0	0	0	0	0	0	0	0
12	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
13	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
14	10	0	0	0	0	0	0	0	0	0	0	0

Каждая строка полученной таблицы – это клиент, а каждый столбец – сумма поступлений от клиента в этот период.

1.2.2 Построение отчета о динамике MRR

Для того, чтобы считать динамику MRR, необходимо ввести компоненты метрики:

- `New` – поступления от клиента, который еще ни разу не оплачивал подписку в указанный промежуток. Если ориентироваться по таблице 1.4, то это значит, что левее выбранного периода поступлений нет;
- `Old` – поступления от клиента в выбранном периоде равны поступлениям в предыдущем периоде;
- `Reactivation` – поступления от клиента в выбранном периоде не равны нулю, при этом в прошлом периоде они были равны нулю. Также были поступления до предыдущего периода;
- `Expansion` – поступления от клиента в выбранном периоде больше поступлений в предыдущем периоде, причем поступления в предыдущем периоде не равны нулю;
- `Contraction` – поступления от клиента в выбранном периоде меньше поступлений в предыдущем периоде;
- `Churn` – поступления от клиента в выбранном периоде равны нулю, а в предыдущем периоде больше нуля.

Итоговый MRR за выбранный период можно расчитать по формуле 1.1:

$$MRR_i = New_i + Old_i + Reactivation_i + Expansion_i - Contraction_i - Churn_i \tag{1.1}$$

Произведя вычисления по каждому периоду, можно получить результат, представленный в таблице 1.5.

Таблица 1.5 – Результат вычисления MRR

	06.2019	07.2019	08.2019	09.2019	10.2019	11.2019	12.2019	01.2020	02.2020	03.2020	04.2020	05.2020
New	127.5	0	0	0	0	0	0	0	0	0	0	0
Old	0	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5
Expansion	0	0	0	0	0	0	0	0	0	0	0	0
Reactivation	0	0	10	0	0	0	0	0	0	0	0	0
Contraction	0	0	0	0	0	0	0	0	0	0	0	0
Churn	0	-90	0	-10	0	0	0	0	0	0	0	0
MRR	127.5	-52.5	47.5	27.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5	37.5

1.3 Базы данных и системы управления базами данных

В задаче построения MRR отчета важную роль играет выбор модели хранения данных, которая будет использоваться для хранения данных о пользователях и анализируемых транзакций.

Для персистентного хранения данных используются базы данных [9]. Для управления базами данных используются системы управления базами данных (сокращенно СУБД) [9]. Система управления базами данных – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

1.4 Хранение данных о пользователе

Система, разрабатываемая в рамках курсового проекта, предполагает собой приложение, доступное всем желающим. Поэтому необходимо предусмотреть наличие нескольких пользователей, каждый из которых будет иметь возможность загружать свои наборы данных для аналитики.

1.4.1 Классификация баз данных по способу обработки

По способу обработки базы данных делятся на две группы – реляционные и нереляционные базы данных. Каждый из двух типов служит для выполнения определенного рода задач.

Реляционные базы данных (SQL)

Данные реляционных баз хранятся в виде таблиц и строк, таблицы могут иметь связи с другими таблицами через внешние ключи, таким образом образуя некие отношения.

Реляционные базы данных используют язык SQL [10]. Структура таких баз данных позволяет связывать информацию из разных таблиц с помощью внешних ключей (или индексов), которые используются для уникальной идентификации любого атомарного фрагмента данных в этой таблице. Другие таблицы могут ссылаться на этот внешний ключ, чтобы создать связь между частями данных и частью, на которую указывает внешний ключ.

SQL используют универсальный язык структурированных запросов для определения и обработки данных. Это накладывает определенные ограничения: прежде чем начать обработку, данные надо разместить внутри таблиц и описать.

Нереляционные базы данных (NoSQL)

Данные нереляционных баз данных не имеют общего формата. Они могут представляться в виде документов (Mongo [11], Tarantool [12]), пар ключ-значение (Redis [13]), графовых представляниях (Neo4j [14]).

Динамические схемы для неструктурированных данных позволяют:

- ориентировать информацию на столбцы или документы;
- основывать ее на графике;
- организовывать в виде хранилища KeyValue;
- создавать документы без предварительного определения их структуры, использовать разный синтаксис;
- добавлять поля непосредственно в процессе обработки.

1.4.2 Выбор модели хранения данных для решения задачи

Для решения задачи NoSQL базы данных выглядят лучше в контексте работы по нескольким причинам:

- задача не предполагает наличие отношений. В системе будет присутствовать одна коллекция пользователей;

- задача предполагает отслеживание наборов данных (ссылок на наборы), которыми располагает пользователь , причем этот набор постоянно изменяется по размеру, следовательно присутствует элемент динамичности схемы. Следует рассматривать документоориентированные СУБД.

1.4.3 Обзор NoSQL СУБД

Существует большое количество NoSQL СУБД. В данном подразделе будут рассмотрены популярные некоммерческие NoSQL документоориентированные СУБД, которые могут быть использованы в проекте.

Couchbase

Couchbase (Couchbase Server) [15] – система управления базами данных класса NoSQL, предоставляет сходные с Apache CouchDB средства для создания документоориентированных баз данных в сочетании с Membase-подобными хранилищами в формате «ключ – значение». Благодаря поддержке стандартного протокола memcached [16], система остаётся совместимой с большим числом унаследованных приложений и может выступать в роли прозрачной замены ряда других NoSQL-систем. Исходный код системы распространяется под лицензией Apache.

Кроме возможности хранения данных в формате «ключ – значение», Couchbase позволяет использовать концепцию документоориентированного хранилища, в котором в качестве единицы хранения данных выступает документ, который имеет уникальный идентификатор, версию и содержит произвольный набор именованных полей в формате «ключ – значение». Используемая модель данных позволяет определять документы в формате JSON, снимая с разработчика необходимость определения схемы хранения. Запросы и индексация данных могут выполняться в соответствии с парадигмой MapReduce. Для организации псевдоструктурированного набора

данных из произвольных документов предлагается концепция формирования представлений (view).

MongoDB

MongoDB [11] – документоориентированная система управления базами данных, не требующая описания схемы таблиц.

MongoDB реализует подход к построению баз данных без таблиц, схем, SQL-запросов, внешних ключей. В отличие от реляционных баз данных, MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность. Способ хранения данных в MongoDB похож на JSON [17]. Для хранения в MongoDB применяется формат BSON (binary JSON) [18]. BSON позволяет работать с данными быстрее, но данные в JSON-формате занимают меньше места, чем в формате BSON.

1.4.4 Выбор СУБД для решения задачи

Для решения задачи была выбрана СУБД MongoDB, потому что она, на мой взгляд, более проста в эксплуатации, а также поддерживает мультидокументные ACID [19] транзакции с изоляцией снапшотов.

1.5 Хранение данных о транзакциях

Каждый пользователь системы может добавить новую выгрузку данных для построения по ней MRR отчета. Хранить данные о выгрузках вместе с пользователем в документоориентированной базе данных не является оптимальным, так как при выгрузке будет получаться ненужная

информация о пользователе. Хранить в другой коллекции так же не будет оптимальным, так как будет затруднена агрегация данных.

Для хранения данных о транзакциях необходимо использовать независимую от базы данных пользователей базу данных, причем данная база данных должна быть строго структурированной для унифицированной обработки данных.

1.5.1 Классификация баз данных по способу хранения

По способу хранения базы данных делятся на две группы – строковые и колоночные базы данных. Каждый из двух типов служит для выполнения определенного рода задач.

Строковые базы данных (OLTP системы)

Строковыми базами данных называются базы данных, записи которых в памяти представляются построчно. Строковые базы данных используются в транзакционных системах (OLTP) [20]. Для таких систем характерно большое количество коротких транзакций с операциями вставки, обновления и удаления (INSERT, UPDATE, DELETE). Основной упор в системах OLTP делается на очень быструю обработку запросов, поддержание целостности данных в средах с множественным доступом и эффективность, измеряемую количеством транзакций в секунду. В базе данных OLTP есть подробные и текущие данные, а схемой, используемой для хранения транзакционных баз данных, является модель сущностей. Она включает в себя запросы, обращающиеся к отдельным записям, таким как обновление данных клиента в базе данных компании.

Колоночные базы данных (OLAP системы)

Колоночными базами данных называются базы данных, записи которых в памяти представляются по столбцам. Колоночные базы данных используются в аналитических системах (OLAP) [21]. OLAP характеризуется относительно низким объемом транзакций. Запросы часто очень сложны

и включают агрегацию. Для систем OLAP время отклика является мерой эффективности. OLAP приложения широко используются методами интеллектуального анализа данных. В базе данных OLAP есть агрегированные, исторические данные, хранящиеся в многомерных схемах. Иногда нужно получить доступ к большому объему данных в управленческих записях, например, какова была прибыль компании в прошлом году.

1.5.2 Выбор модели хранения данных для решения задачи

Для решения задачи колоночное хранение преобладает над строковым по нескольким причинам:

- задача не предполагает постоянное добавление и изменение данных. Данные заносятся в базу данных единожды;
- задача предполагает выполнение аналитики, следовательно колоночное хранение в приоритете.

1.5.3 Обзор колоночных СУБД

Существует большое количество колоночных СУБД. В данном подразделе будут рассмотрены популярные некоммерческие колоночные СУБД, которые могут быть использованы в проекте.

Greenplum Community Edition

Greenplum [22] – это технология больших данных , основанная на архитектуре MPP (massive parallel processing) и PostgresSQL [23] с открытым исходным кодом.

Greenplum использует методы массовой параллельной обработки (MPP). Каждый компьютерный кластер состоит из главного узла, резервного главного узла и узлов сегмента. Все данные находятся на узлах сегмента, а информация каталога хранится в главных узлах. Узлы сегментов запускают

один или несколько сегментов, которые представляют собой измененные экземпляры базы данных PostgreSQL и которым назначается идентификатор содержимого. Для каждой таблицы данные распределяются между узлами сегмента на основе ключей столбцов распределения, указанных пользователем на языке определения данных. Для каждого идентификатора содержимого сегмента существует как основной, так и зеркальный сегменты, которые не работают на одном физическом хосте. Когда запрос поступает на главный узел, он анализируется, планируется и отправляется всем сегментам для выполнения плана запроса и либо возврата запрошенных данных, либо вставки результата запроса в таблицу базы данных. Язык структурированных запросов используется для представления запросов системе. Семантика транзакции соответствует ACID.

ClickHouse

ClickHouse [24] – это колоночная аналитическая СУБД с открытым кодом, позволяющая выполнять аналитические запросы в режиме реального времени на структурированных больших данных, разрабатываемая компанией Яндекс.

ClickHouse использует собственный диалект SQL близкий к стандартному, но содержащий различные расширения: массивы и вложенные структуры данных, функции высшего порядка, вероятностные структуры, функции для работы с URI, возможность для работы с внешними key-value хранилищами («словарями»), специализированные агрегатные функции, функциональности для семплирования, приблизительных вычислений, возможность создания хранимых представлений с агрегацией, наполнения таблицы из потока сообщений Apache Kafka и т. д.

Однако при этом имеются и ограничения – отсутствие транзакций, отсутствие точечных UPDATE/DELETE (пакетный UPDATE/DELETE был введен в июне 2018 года), ограниченная поддержка синтаксиса JOIN, строгие типы с необходимостью явного приведения, для некоторых операций промежуточные данные должны помещаться в оперативную память, отсутствие оконных функций, отсутствие полноценного оптимизатора запросов, точечного чтения, присутствие ограничений в реализации некоторых функций, связанных со спецификой использования ClickHouse в Яндексе,

и т. д.

Система оптимизирована для хранения данных на жестких дисках (используются преимущества линейного чтения, сжатия данных). Для обеспечения отказоустойчивости и масштабируемости ClickHouse может быть развернут на кластере (для координации процесса репликации используется Apache ZooKeeper). Для работы с базой данных существует консольный клиент, веб-клиент, HTTP интерфейс, ODBC и JDBC-драйверы, а также готовые библиотеки для интеграции со многими популярными языками программирования и библиотеками.

1.5.4 Выбор СУБД для решения задачи

Для решения задачи была выбрана СУБД ClickHouse, потому что она, имеет различные типы таблиц, которые значительно упростят обработку данных в курсовой работе [25]. Помимо этого, в тестах на производительность ClickHouse показал более высокий результат, нежели Greenplum [26].

1.6 Хранение данных о вычисленном MRR

Для того, чтобы повторно не вычислять MRR периода, который уже был вычислен, можно прибегнуть к кэшированию данных. Для кэширования можно прибегнуть к использованию NoSQL in-memory баз данных. Такие базы данных хранят данные в оперативной памяти, что обеспечивает более быстрый доступ, нежели если бы данные хранились на диске.

1.6.1 Обзор in-memory NoSQL СУБД

Redis

Redis [13] – резидентная система управления базами данных класса NoSQL с открытым исходным кодом, работающая со структурами данных типа «ключ – значение». Используется как для баз данных, так и для реализации кэшей, брокеров сообщений.

Хранит базу данных в оперативной памяти, снабжена механизмами снимков и журналирования для обеспечения постоянного хранения (на дисках, твердотельных накопителях). Также предоставляет операции для реализации механизма обмена сообщениями в шаблоне «издатель-подписчик»: с его помощью приложения могут создавать каналы, подписываться на них и помещать в каналы сообщения, которые будут получены всеми подписчиками (как IRC-чат). Поддерживает репликацию данных с основных узлов на несколько подчинённых (англ. master – slave replication). Также поддерживает транзакции и пакетную обработку команд (выполнение пакета команд, получение пакета результатов).

Все данные Redis хранит в виде словаря, в котором ключи связаны со своими значениями. Одно из ключевых отличий 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 (команды) доступны для него; поддерживаются такие высокоуровневые операции, как объединение и разность наборов, сортировка наборов.

Tarantool

Tarantool [12] – это платформа in-memory вычислений с гибкой схемой данных для эффективного создания высоконагруженных приложений. Включает в себя базу данных и сервер приложений на Lua.

Обладает высокой скоростью работы по сравнению с традиционными СУБД, обладая теми же свойствами: персистентности, транзакционности ACID, репликации master-slave, master-master.

Для хранения данных используются таплы (кортежи). Это массив с данными, которые не типизированы. Кортежи или таплы объединяются в спейсы. Спейс – это аналог из мира SQL, таблица. Спейс это коллекция таплов, а тапл это коллекция полей.

Кортежи организованы в пространства (space или таблицы). Для каждого пространства указывается технология хранения (memtx или vinyl).

Пространство должно быть проиндексировано первичным ключом. Также поддерживаются неограниченное количество вторичных ключей.

Ключ может состоять из одного и более полей.

1.6.2 Выбор СУБД для решения задачи

Для кэширования данных выбрана СУБД Tarantool, потому как она проста в развертывании, а также имеет исчерпывающую документацию на русском языке.

1.7 Формализация данных

В данном подразделе описаны требования к данным, содержащимся в базах данных, используемых при разработке приложения.

1.7.1 База данных пользователей

База данных пользователей должна хранить информацию о пользователях системы. Каждый пользователь должен обладать уникальным идентификатором, чтоы можно было однозначно идентифицировать пользователя.

1.7.2 База данных транзакций

База данных транзакций должна хранить информацию о каждой проведенной транзакции, причем неважно однозначно идентифицировать транзакцию, потому как они будут выбираться в промежутке между какими-то датами. Для того, чтобы использовать одну таблицу для транзакций всех пользователей, транзакцию можно помечать специальным признаком, который позволит в выборке брать транзакции, принадлежащие только текущему пользователю.

Помимо этого база должна хранить промежуточные таблицы распределения уплаченных средств по периодам.

1.7.3 База данных кэшируемого результата

База данных кэшируемого результата должна хранить конечный результат для прямого визуализирования, без дополнительной обработки.

1.8 Анализ существующих решений

В качестве существующих решений для анализа выбраны сервисы Pabbly Subscriptions Billing [27], SaaS Metrics [8], ChartMogul [28], Databox [29] и Baremetrics [30].

В таблице 1.6 представлен сравнительный анализ существующих решений.

Таблица 1.6 – Анализ существующих решений

Сервис	**Цена**	**Аналитика продаж**	**Аналитика подписок**	**Безлимитный расчет MRR**
Pabbly Subscriptions Billing	От $19 в месяц	Да	Да	Да
SaaS Metrics	Свяжитесь с отделом продаж	Нет	Нет	Нет
ChartMogul	От $125 в месяц	Да	Да	Нет
Databox	От $59 в месяц	Да	Да	Нет
Baremetrics	От $50 в месяц	Да	Да	Нет

Вывод

В данном разделе была рассмотрена метрика MRR, задача построения MRR отчета. Были проанализированы способы хранения информации для компонентов системы, а также выбраны оптимальные способы для решения потавленной задачи. Был проведен анализ СУБД, используемых для решения задачи и также выбраны оптимальные информационные системы. Также были формализованы данные, используемые в системе.

2 Конструкторская часть

В данном разделе представлены этапы проектирования баз данных.

2.1 Проектирование базы данных пользователей

База данных пользователей будет реализована с использованием СУБД MongoDB. В базе данных будет существовать одна коллекция, хранящая в себе документы с информацией о пользователях. Структура документа, хранящего информацию о пользователе, представлена на рисунке 2.1.

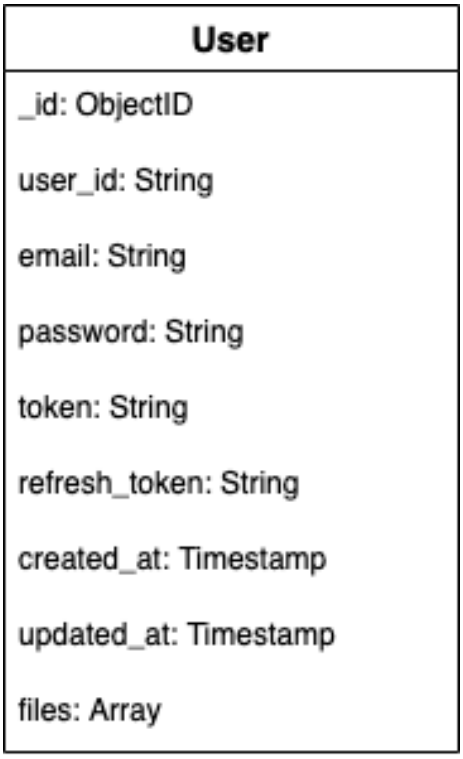

Рисунок 2.1 – Структура документа `User`

Поля документа означают:

- `_id` – служебное поле MongoDB, являющееся уникальным идентификатором документа. Не может повторяться в пределах одной коллекции;
- `user_id` – уникальный идентификатор пользователя, который будет использоваться для идентификации записей в таблице транзакций.

Также будет осуществляться для генерации JWT токенов для JWT-авторизации [31];

- `email` – электронная почта пользователя, используется для авторизации и регистрации;
- `password` – пароль пользователя, хранящийся в шифрованном виде. Шифрование производится при помощи функции bcrypt [32];
- `token` – JWT-токен пользователя;
- `created_at` – время создания JWT-токена;
- `updated_at` – время обновления JWT-токена;
- `files` – массив, хранящий названия файлов, загруженных пользователем, в которых хранятся данные для аналитики;

2.2 Проектирование базы данных транзакций

База данных транзакций будет реализована с использованием СУБД ClickHouse. В базе данных будет существовать одна таблица, хранящая в себе информацию обо всех транзакциях, и произвольное количество таблиц, хранящих в себе нормированное распределение уплаченных средств за выбранный промежуток времени. Таблицы распределения средств будут создаваться каждый раз, когда пользователь выбирает период для определенного файла, за исключением случая, когда ранее уже был выбран данный период для данного файла. Структура таблиц, хранящих информацию о транзакциях и о распределении уплаченных средств, представлена на рисунке 2.2.

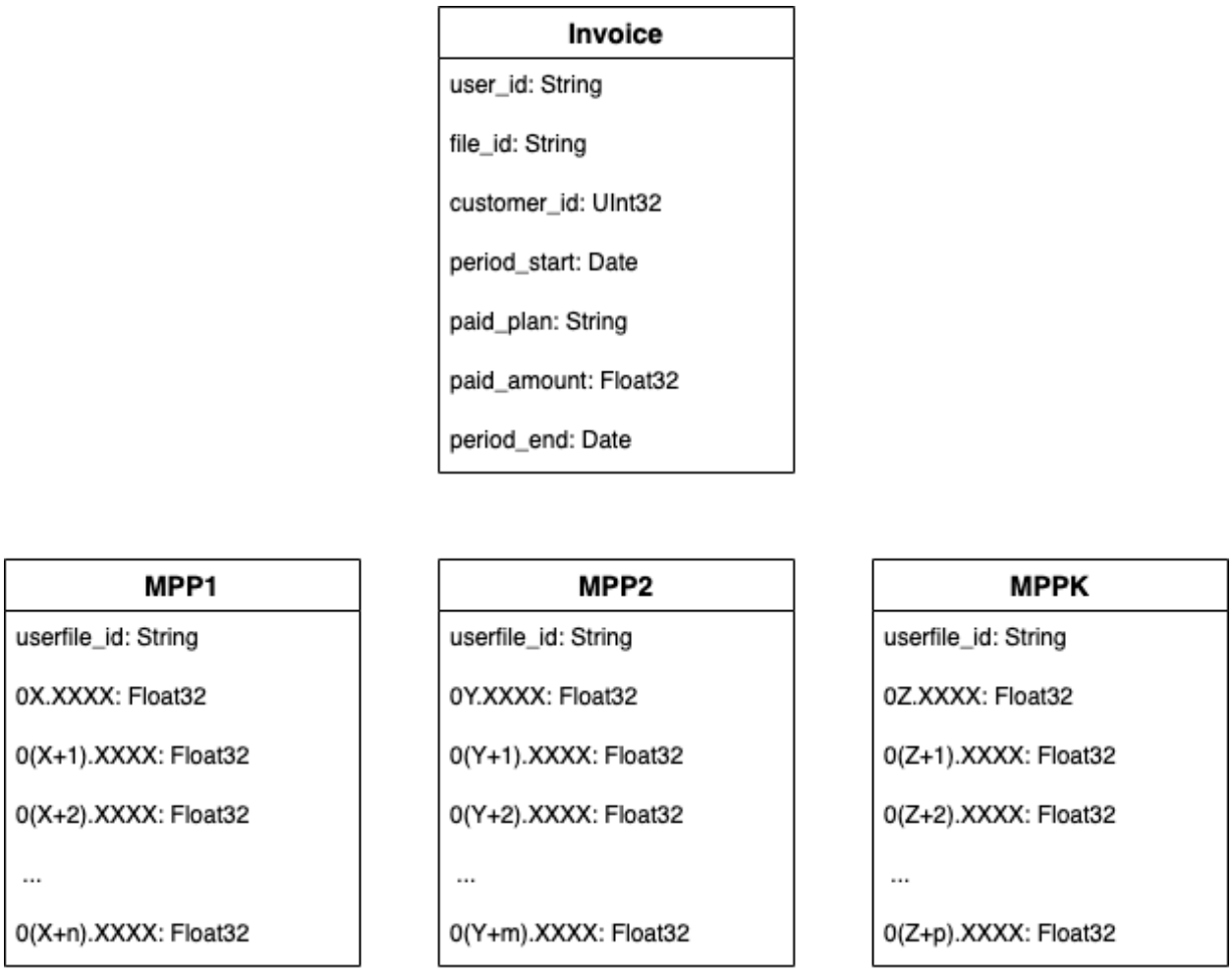

Рисунок 2.2 – Структура таблиц `Invoice` и `MPP`

Поля таблицы `Invoice` означают:

- `user_id` – уникальный идентификатор пользователя, который загрузил данные;
- `file_id` – название файла, из которого были загружены данные;
- `customer_id` – уникальный идентификатор клиента, произведшего транзакцию;
- `period_start` – начало расчетного периода;
- `paid_plan` – тарифный план;
- `paid_amount` – уплаченная сумма за весь расчетный период;
- `period_end` – конец расчетного периода;

Поля таблицы $\texttt{MPP}_i$ означают:

- `userfile_id` – уникальный идентификатор, необходимый для нормирования записей таблицы;
- `0X.XXXX` – месяц из выбранного периода. Количество столбцов таблицы зависит от выбранного пользователем аналитического периода.

СУБД ClickHouse позволяет создавать таблицы с разными особенностями обработки и хранения данных за счет применения определенных методов и триггеров к данным. Каждая таблица имеет свой «движок», каждый из которых имеет свои особенности [25].

Таблица `Invoice` использует движок `Memory`, позволяющий производить CRUD операции над записями таблицы, при этом хранение данных происходит в оперативной памяти, что обеспечивает быстрое выполнение запросов, но ограничивает в персистентности данных. Для исправления данного недостатка можно использовать движок `Buffer`, который хранит данные в оперативной памяти, но переодически сбрасывает их на диск, таким образом имея две копии в один момент времени. В случае аварийного отказа базы данных, потеря данных не произойдет.

Таблица `MPP` использует движок `SummingMergeTree` из семейства движков `MergeTree`. Данный движок позволяет производить необходимую нормировку по клиентам, сохраняя в каждый момент времени только одну запись с заданным ключом условия. В данной работе таким ключом является поле `userfile_id`. Обнаружив несколько записей с заданным ключом, движок будет суммировать числовые поля записей, имеющих одинаковый ключ. Таким образом без дополнительной обработки можно получить нормированное распределение уплаченных средств.

2.3 Проектирование базы данных кэширования

База данных кэширования будет реализована с использованием СУБД Tarantool. В базе данных будет существовать одна таблица (спейс), хранящая в себе информацию о расчитанном MRR для заданного периода.

Первичным ключом будет являться поле, содержащее в себе уникальный идентификатор пользователя, имя файла и заданный период. При повторном запросе данных сначала будет производиться проверка, присутствует ли запись в кэше. В случае, если запись присутствует, запрос к базе данных транзакций производиться не будет и будут возвращены данные из кэша. В противном случае будет произведен запрос к базе данных с транзакциями. Структура спейса, хранящего кэшируюмую информацию, представлена на рисунке 2.3.

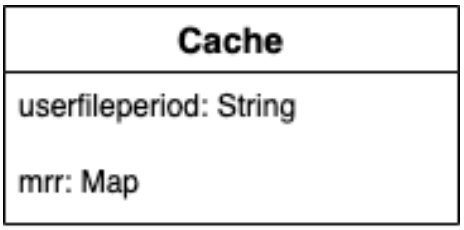

Рисунок 2.3 – Структура спейса `Cache`

Поля спейса `Cache` означают:

- `userfileperiod` – уникальный идентификатор, являющийся первичным ключом;
- `mrr` – набор пар «ключ – значение», в которых ключом является компонент MRR (New, Old, Reactivation, Exapnsion, Contraction, Churn, Total), а значением – его непосредственное значение.

Спейс `Cache` создан на основе движка `memtx`, хранящего все данные в оперативной памяти [33]. Персистентность данных обеспечивается при помощи ведения журнала транзакций, хранящегося на диске. При аварийном завершении работы базы данных и последующем перезапуске транзакции, записанные в журнал, применяются заново, и база данных становится доступна для использования в том же состоянии, в котором была до момента отказа.

Вывод

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

3 Технологическая часть

В данном разделе представлены архитектура приложения, средства разработки программного обеспечения, детали реализации и пользовательский интерфейс.

3.1 Архитектура приложения

Пользователь взаимодействует с системой посредством Web-приложения, построенного с использованием паттерна SPA (Single Page Application) [34].

Пользовательская часть приложения коммуницирует с серверной частью при помощи реализованного RPC API на стороне сервера [35] [36].

Серверная часть коммуницирует с базами данных при помощи специализированных коннекторов, позволяющих делать запросы к базе данных из языка программирования.

Вся обработка данных производится на сервере, чтобы не загружать клиентское приложение и не вызывать задержек в обработке.

Общая схема архитектуры приложения представлена на рисунке 3.1.

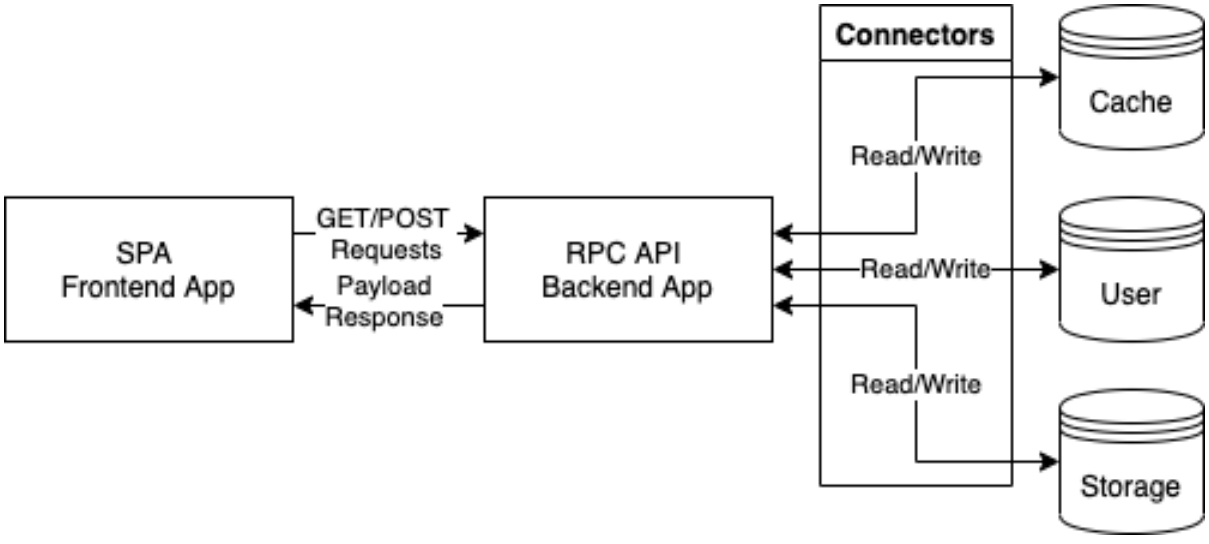

Рисунок 3.1 – Схема архитектуры приложения

3.2 Средства реализации

Для разработки пользовательского интерфейся был выбран JavaScript фреймворк `Vue.js 3` [37] [38]. Vue.js позволяет создавать отзывчивые SPA приложения, основанные на компонентах. Компоненты впоследствии можно переиспользовать без дублирования кода. Помимо этого, при помощи специальных библиотек: `vuex` и `vue-router` можно оптимально управлять данными внутри приложения и имитировать многостраничное приложение соответственно [39] [40].

Для разработки серверной части приложения был выбран язык `Go` [41]. Выбор обусловлен тем, что данный язык является компилируемым, следовательно в процессе разработке возникнет меньше промежуточных ошибок. Также Go предоставляет нативную поддержку многопоточности, что позволяет повысить производительность приложения. В качестве фреймворка для реализации API был выбран `gin` [42]. gin предоставляет наибольшую производительность среди существующих Go фреймворков для реализации REST/RPC API [43].

Для коммуницирования серверной части приложения с базами данных были использованы официальные коннекторы: `go-tarantool` для Tarantool, `dbr` для ClickHouse и `mongo-go-driver` для MongoDB [44] [45] [46].

Для упаковки приложения в готовый продукт была выбрана система контейнеризации `Docker` [47]. Docker позволяет создать изолированную среду для программного обеспечения, которое можно будет разворачивать на различных операционных системах без дополнительно вмешательства для обеспечения совместимости.

3.3 Детали реализации

В листингах 3.1 – 3.5 представлены листинги реализации взаимодействия клиента с сервером, обработка запроса с клиента на сервере, подключение и инициализация базы данных пользователей, транзакций, кэшированных данных соответственно.

Листинг 3.1 – Пример реализации взаимодействия клиента с сервером

```
  async uploadData(context, data) {
    const response = await fetch("http://localhost:8081/files/upload", {
      method: "POST",
      body: data,
      headers: {
        token: localStorage.getItem("token"),
      },
    });
    const responseData = await response.json();

    if (!response.ok) {
      const error = new Error(responseData.message || "Failed to fill data");
      throw error;
    }

    context.commit("setFile", responseData.filename);
  },
```

Листинг 3.2 – Пример реализации обработки запроса клиента на сервере

```
func SaveFile(c *gin.Context) {
    userID, ie := c.Get("userID")
    if !ie {
        c.AbortWithStatusJSON(http.StatusInternalServerError, gin.H{
            "message": "Unable to determine logged in user",
        })
        return
    }

    file, err := c.FormFile("file")
    if err != nil {
        c.AbortWithStatusJSON(http.StatusBadRequest, gin.H{
            "message": "No file is received",
        })
        return
    }

    fext := filepath.Ext(file.Filename)
    if fext != ".csv" {
        c.AbortWithStatusJSON(http.StatusBadRequest, gin.H{
            "message": "Wrong file format. Please provide CSV file",
        })
        return
    }

    dir := fmt.Sprintf("static/%v", userID)
    fname := fmt.Sprintf("%v%v", uuid.New(), fext)
    fpath := fmt.Sprintf("%v/%v", dir, fname)
```

```
29
30      if _, err = os.Stat(dir); os.IsNotExist(err) {
31          os.Mkdir(dir, 0777)
32      }
33
34      err = c.SaveUploadedFile(file, fpath)
35      if err != nil {
36          c.AbortWithStatusJSON(http.StatusInternalServerError, gin.H{
37              "message": "Unable to save the file",
38          })
39          return
40      }
41
42      err = utils.UpdateFiles(fname, userID.(string))
43      if err != nil {
44          c.AbortWithStatusJSON(http.StatusInternalServerError, gin.H{
45              "message": "Unable to update user in db",
46          })
47          return
48      }
49
50      csvFile, err := os.Open(fpath)
51      if err != nil {
52          c.AbortWithStatusJSON(http.StatusInternalServerError, gin.H{
53              "message": "Failed to find data with given id",
54          })
55          return
56      }
57      defer csvFile.Close()
58
59      invoices := []*models.Invoice{}
60
61      err = gocsv.UnmarshalFile(csvFile, &invoices)
62      if err != nil {
63          c.AbortWithStatusJSON(http.StatusBadRequest, gin.H{
64              "message": "Failed to parse given CSV file",
65          })
66          return
67      }
68
69      err = utils.UploadFile(userID.(string), fname, invoices)
70      if err != nil {
71          c.AbortWithStatusJSON(http.StatusBadRequest, gin.H{
72              "message": "Failed to upload data to database",
73          })
74          return
75      }
76
```

```
    c.JSON(http.StatusOK, gin.H{
        "message": "File uploaded",
        "filename": fname,
    })
}
```

Листинг 3.3 – Инициализация базы данных пользователей

```
func ConnectDB() {
    client, err := mongo.NewClient(options.Client().ApplyURI(os.Getenv("MONGODB_URL")))
    if err != nil {
        panic("Failed to create MongoDB client")
    }

    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    err = client.Connect(ctx)
    if err != nil {
        panic("Failed to connect to MongoDB")
    }

    err = client.Ping(ctx, readpref.Primary())
    if err != nil {
        panic("Failed to ping MongoDB")
    }

    UserCol = client.Database("mrr").Collection("user")
    DB = client
}
```

Листинг 3.4 – Инициализация базы данных транзакций

```
func ConnectDB() {
    conn, err := dbr.Open("clickhouse", os.Getenv("CLICKHOUSE_URL"), nil)
    if err != nil {
        panic("Failed to connect to Clickhouse")
    }

    err = initDB(conn)
    if err != nil {
        panic("Failed to create Clickhouse database")
    }

    err = initTable(conn)
    if err != nil {
        panic("Failed to create Clickhouse table")
    }

```

```
    DB = conn
}

func initDB(conn *dbr.Connection) error {
    sess := conn.NewSession(nil)
    _, err := sess.Exec(`
    CREATE DATABASE IF NOT EXISTS mrr
    `)

    return err
}

func initTable(conn *dbr.Connection) error {
    sess := conn.NewSession(nil)
    _, err := sess.Exec(`
    CREATE TABLE IF NOT EXISTS mrr.storage(
        user_id String,
        file_id String,
        customer_id UInt32,
        period_start Date,
        paid_plan String,
        paid_amount Float32,
        period_end Date
    )
    ENGINE = Memory
    `)

    return err
}
```

Листинг 3.5 – Инициализация базы данных кэширования

```
func ConnectDB() {
    opts := tarantool.Opts{User: "guest"}
    conn, err := tarantool.Connect(os.Getenv("TARANTOOL_URL"), opts)
    if err != nil {
        panic("Failed to connect to Tarantool")
    }

    err = initSpace(conn)
    if err != nil {
        fmt.Println(err)
        panic("Failed to create Tarantool space")
    }

    DB = conn
}

func initSpace(conn *tarantool.Connection) error {
```

```
    _, err := conn.Eval("return␣box.schema.space.create('cache')", []interface{}{})
    if err != nil {
        errString := err.Error()
        if !(strings.Contains(errString, "unsupported") || strings.Contains(errString,
            "exists")) {
            return err
        }
    }

    _, err = conn.Eval(`return␣box.space.cache:format({
␣␣␣␣␣␣␣␣{name␣=␣'userfileperiod',␣type␣=␣'string',␣is_nullable␣=␣false},
␣␣␣␣␣␣␣␣{name␣=␣'mrr',␣type␣=␣'map',␣is_nullable␣=␣false},
␣␣␣␣})`, []interface{}{})
    if err != nil {
        return err
    }

    _, err = conn.Eval("return␣box.space.cache:create_index('pk',␣{parts␣=␣
        {'userfileperiod'}})", []interface{}{})
    if err != nil {
        errString := err.Error()
        if !strings.Contains(errString, "exists") {
            return err
        }
    }

    return nil
}
```

3.4 Пользовательский интерфейс

На рисунке 3.2 представлена главная страница Web-приложения. Главную страницу может просматривать любой пользователь.

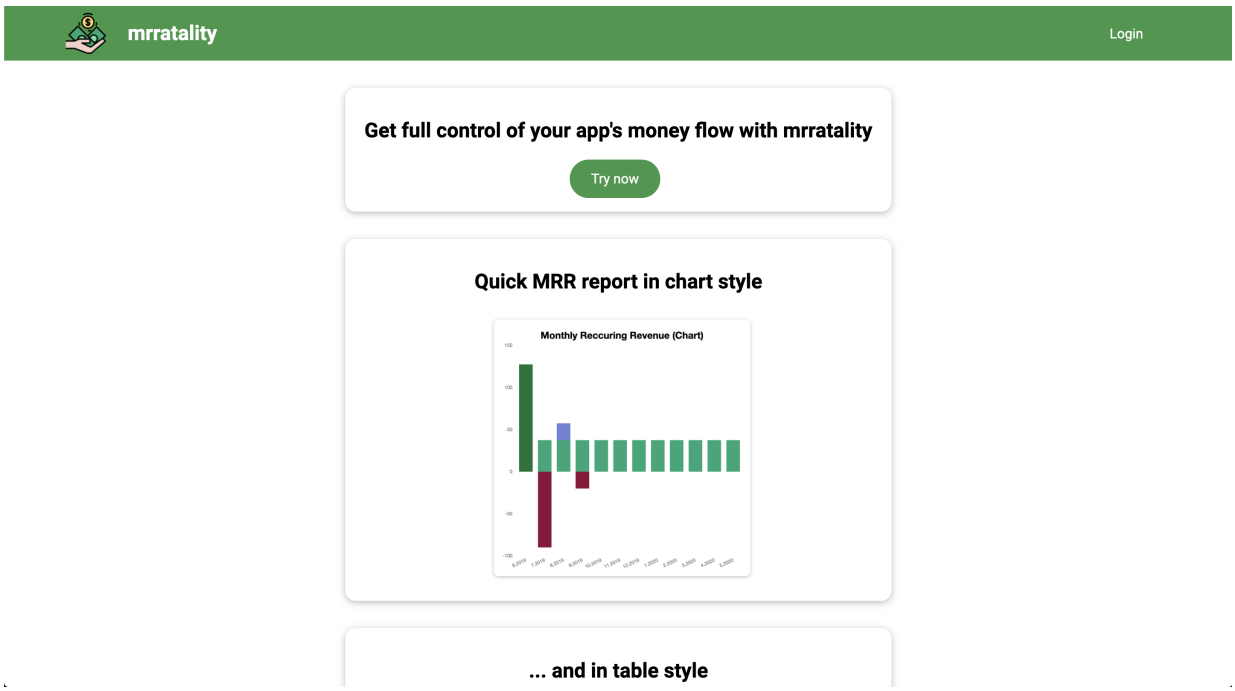

Рисунок 3.2 – Главная страница приложения

На рисунке 3.3 представлена страница входа. Страницу входа может просматривать любой неавторизованный пользователь. При попытке авторизованного пользователя посетить страницу входа, пользователь перенаправляется на главную страницу.

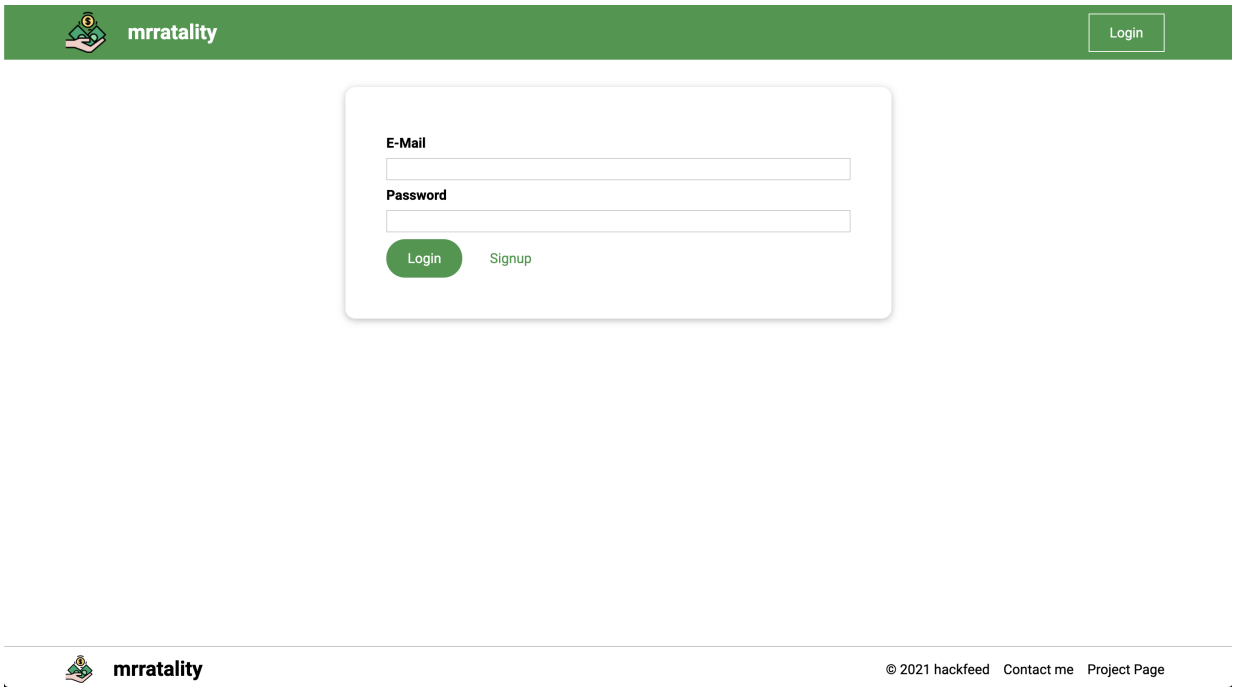

Рисунок 3.3 – Страница входа приложения

На рисунках 3.4 – 3.5 представлен контроль над корректностью даных для входа в приложение. Если в базе данных пользователей не найден поль-

зователь с соответствующим адресом электронной почты, то будет выдано соответствующее сообщение. Аналогично в случае, если пользователь указал неверный пароль.

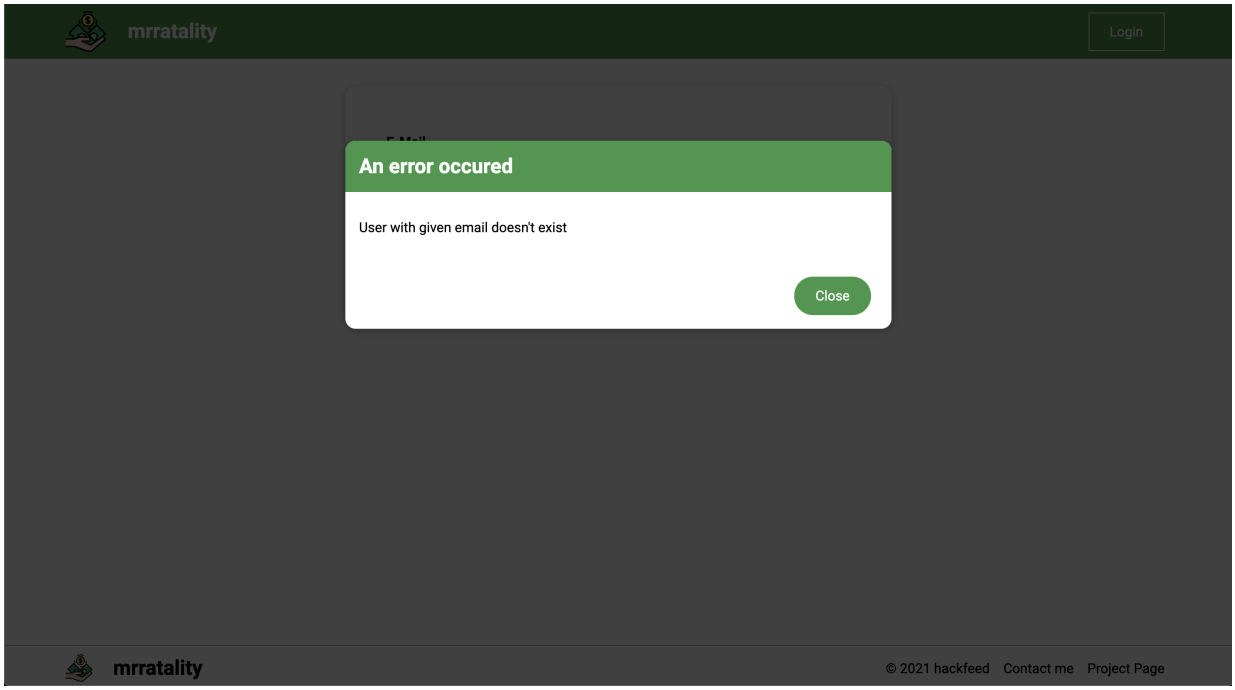

Рисунок 3.4 – Ошибка входа из-за несуществующего адреса электронной почты

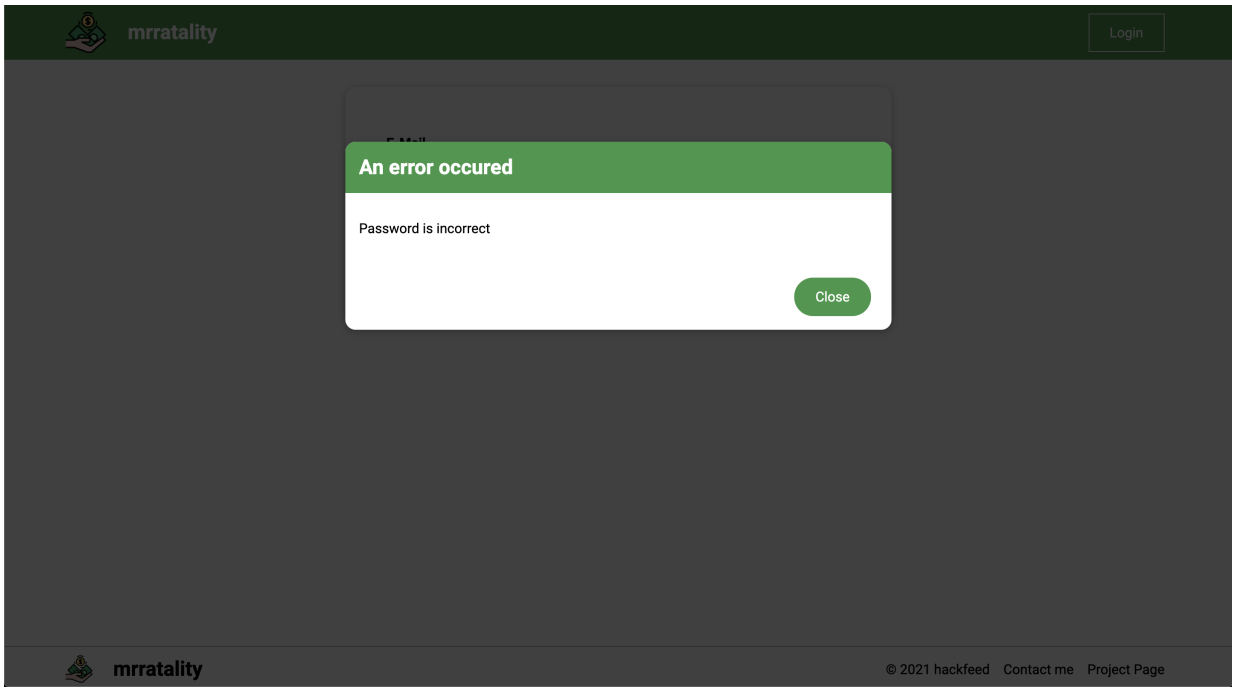

Рисунок 3.5 – Ошибка входа из-за неверного пароля

На рисунке 3.6 представлен вид личного кабинета пользователя, в котором он может загружать новые файлы, удалять ненужные и выбирать

файлы для аналитики.

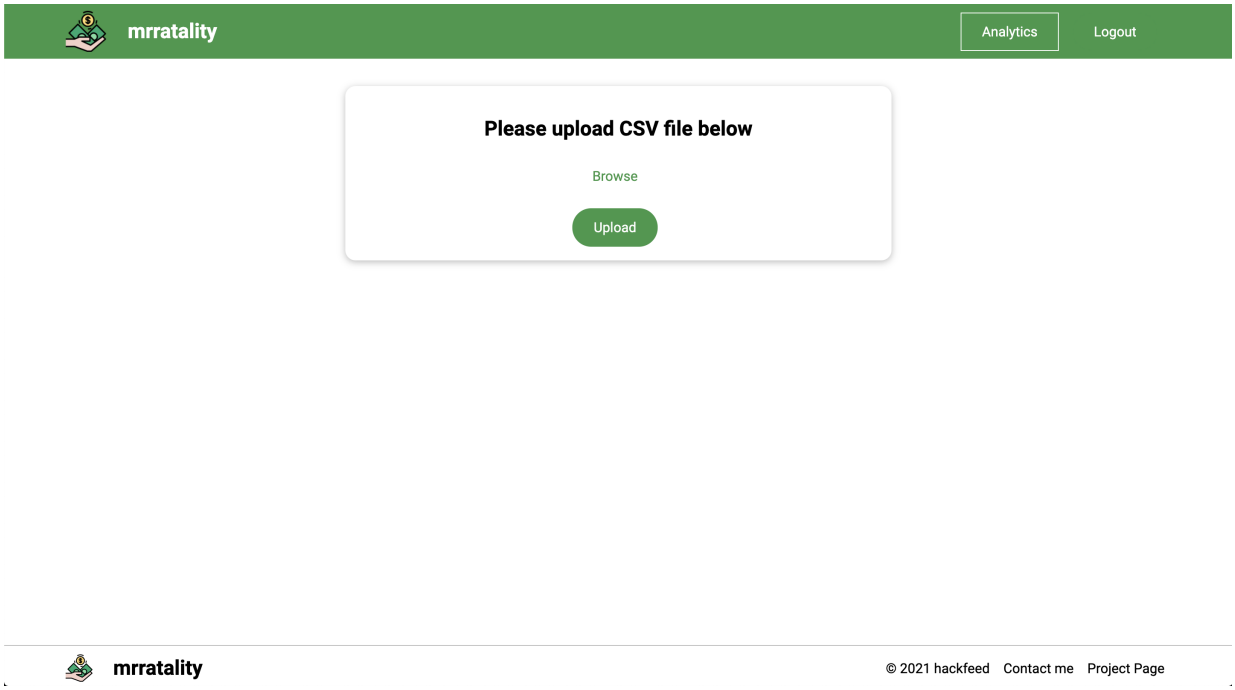

Рисунок 3.6 – Личный кабинет пользователя

На рисунке 3.7 представлен вид личного кабинета пользователя, который загрузил несколько файлов в систему. Пользователь может ориентироваться в файлах по дате загрузки.

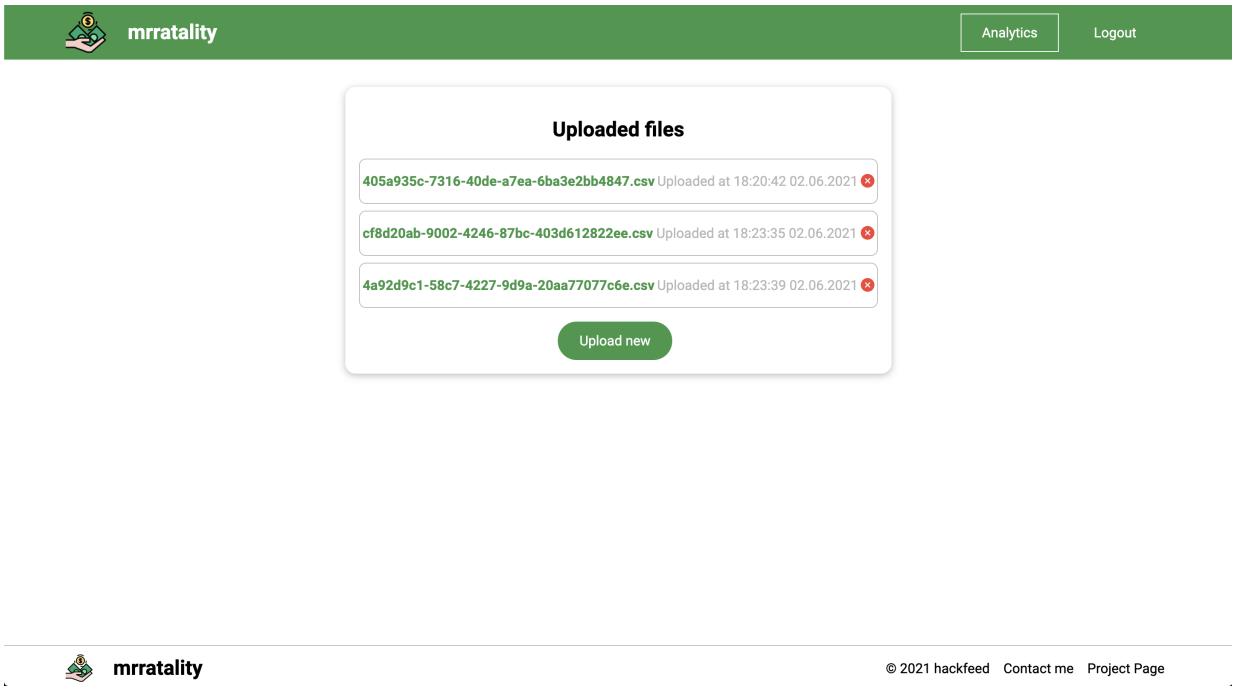

Рисунок 3.7 – Личный кабинет пользователя с загрженными файлами

На рисунке 3.8 представлена страница для выбора периода для анали-

тики выбранного файла.

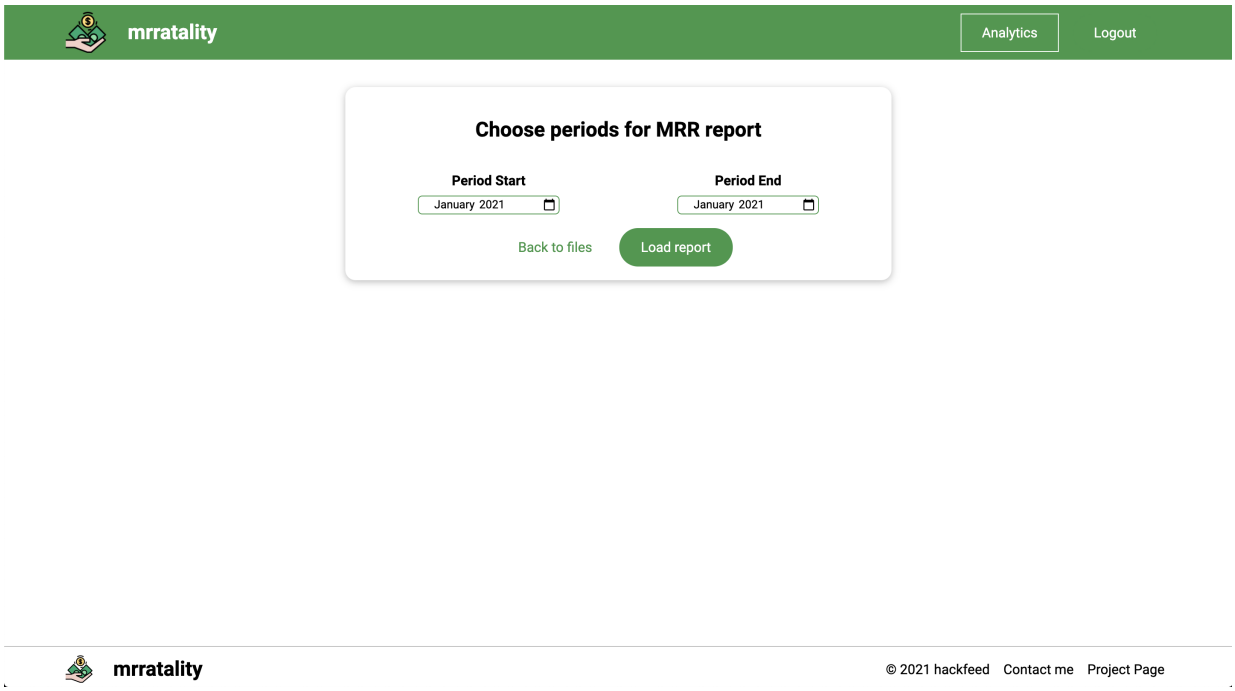

Рисунок 3.8 – Выбор периода для аналитики

На рисунках 3.9 – 3.10 представлен контроль ошибок на данном этапе: пользователь оповещается об ошибках в случаях, если пользователь выбрал окончание периода, дата которого раньше начала периода, и если в выбранный период не было данных.

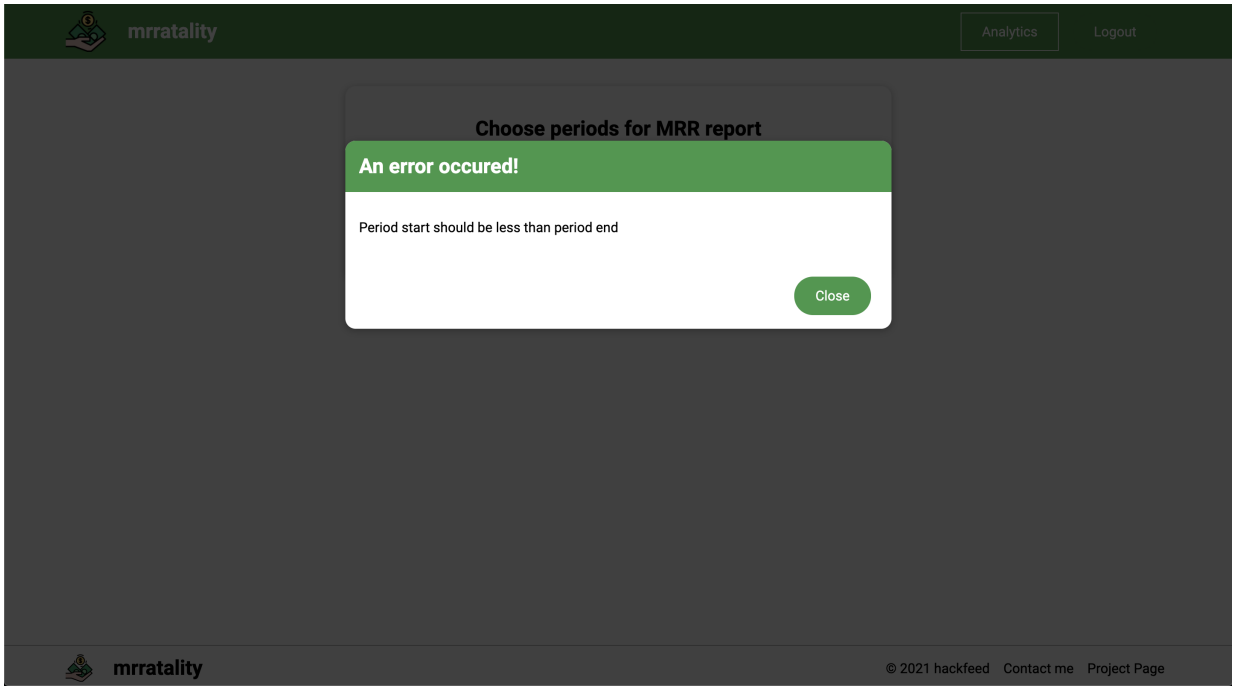

Рисунок 3.9 – Ошибка аналитики из-за неправильных границ периода

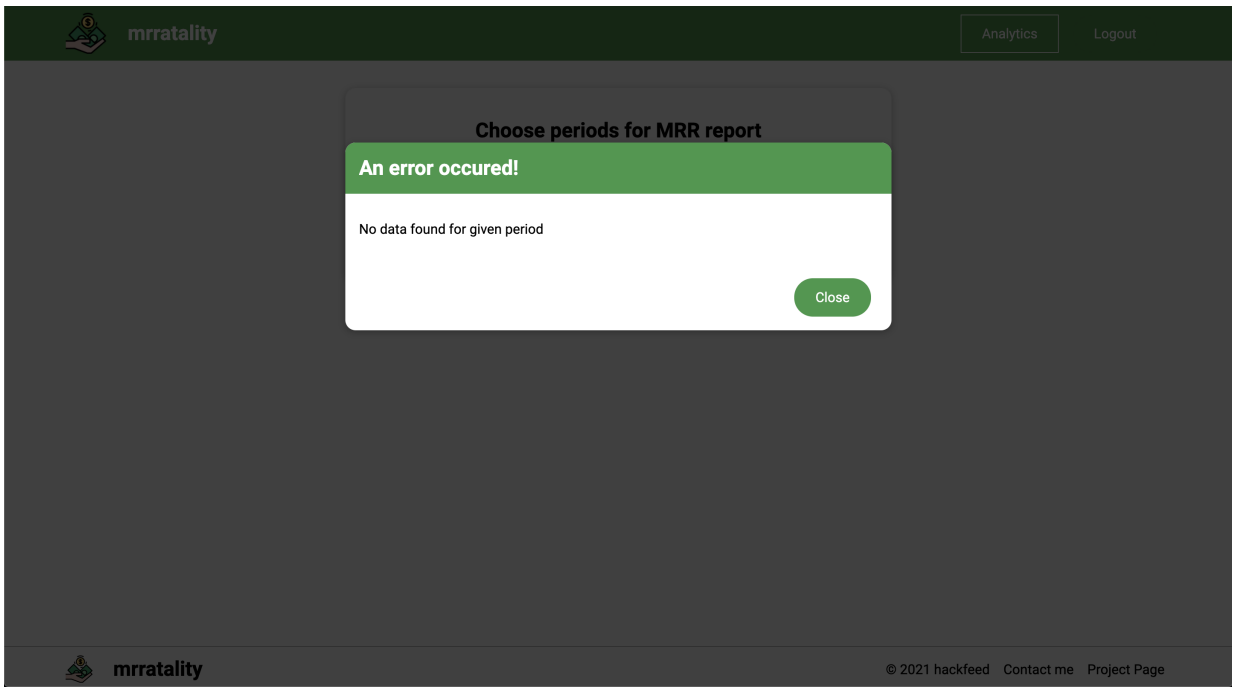

Рисунок 3.10 – Ошибка аналитики из-за отсутствия данных в выбранный период

На рисунках 3.11 – 3.12 представлены результаты построения MRR отчета в виде гистограммы и в виде таблицы. Гистограмма является интерактивной: при наведении курсора мыши можно увидеть дополнительные сведения.

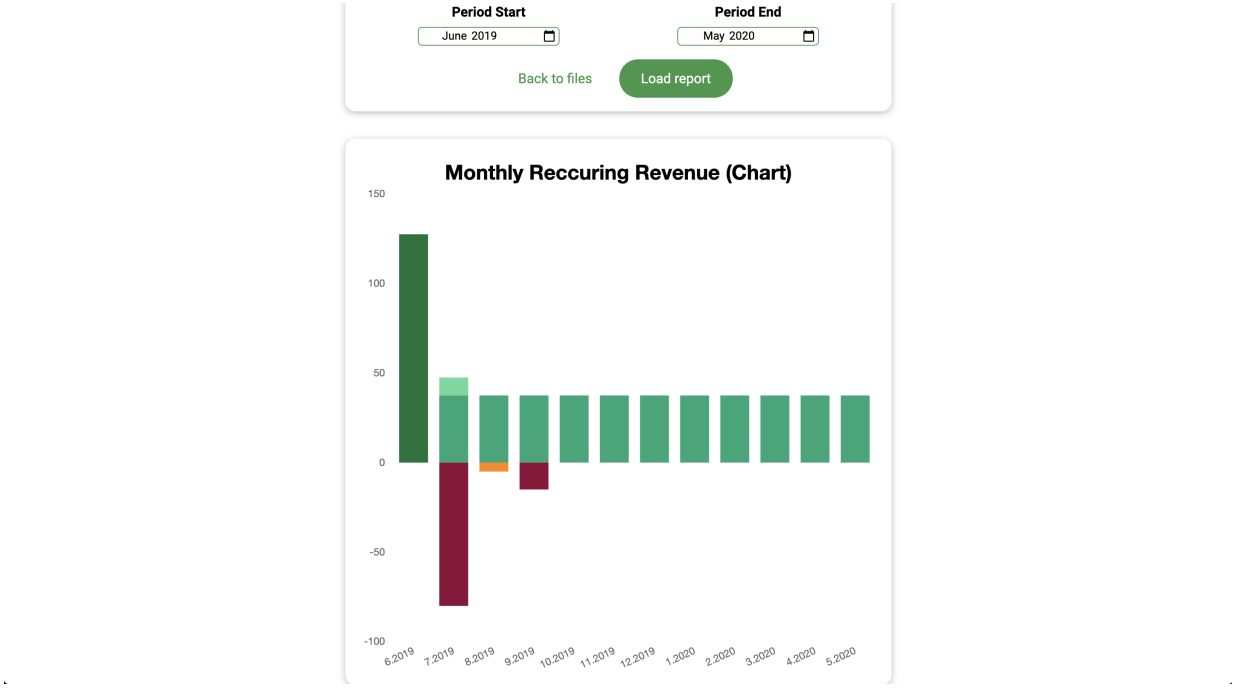

Рисунок 3.11 – MRR отчет в виде гистограммы

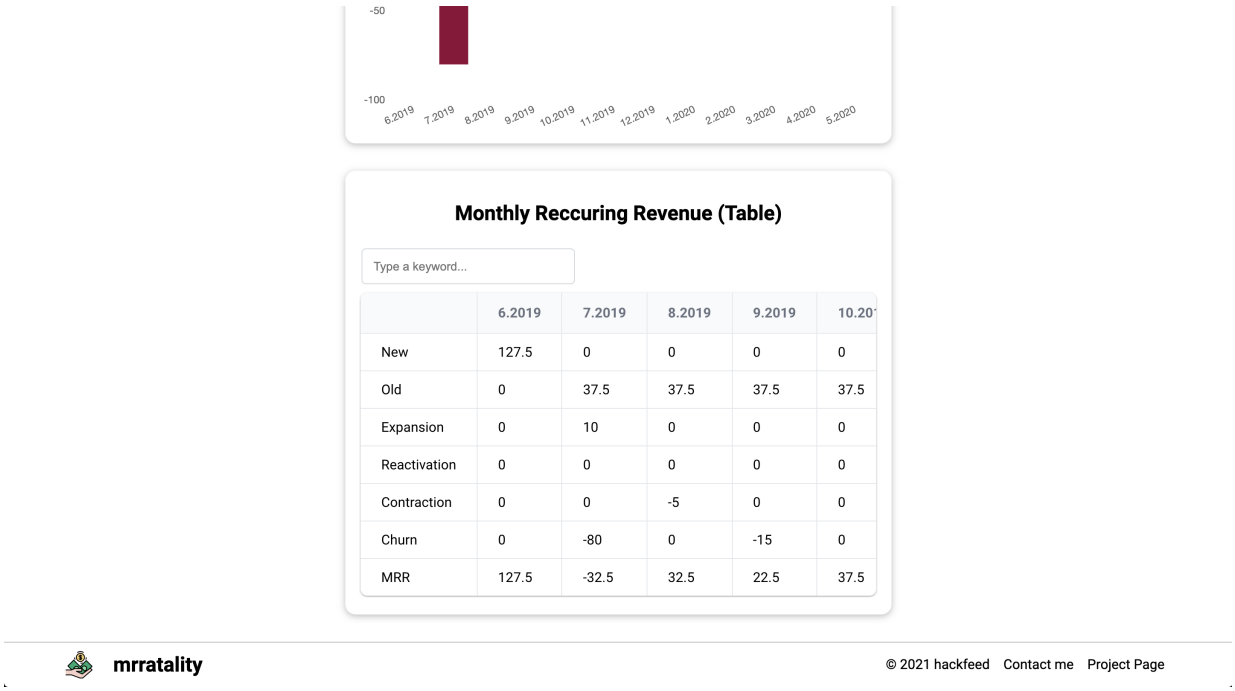

Рисунок 3.12 – MRR отчет в виде таблицы

Вывод

В данном разделе были представлены средства реализации программного обеспечения, листинги ключевых компонентов системы а также представлен пользовательский интерфейс приложения.

4 Исследовательская часть

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

4.1 Постановка эксперимента

В данном подразделе представлены цель, описание и результаты эксперимента.

4.1.1 Цель эксперимента

Целью эксперимента является сравнение времени, требуемого для получения итоговых данных для построения MRR отчета с и без использования кэширования.

4.1.2 Описание эксперимента

Сравнить занимаемое время можно при помощи отключения реализованного механизма кэширования.

Для этого нужно отключить базу данных, хранящую данные о кэшировании, и каждый раз выполнять запрос напрямую к базе данных с транзакциями.

Для проведения эксперимента будут использоваться данные, по которым можно построить годовой MRR отчет (12 месяцев). Тестирование будет проводиться на периодах 36 месяцев, 24 месяца, 12 месяцев, 6 месяцев, 3 месяца, 1 месяц.

4.1.3 Результат эксперимента

В таблице 4.1 представлены результаты поставленного эксперимента.

Таблица 4.1 – Результаты сравнения времени, необходимого для получениях данных без кэширования и с кэшированием

Количество месяцев	**Время без кэширования, мс**	**Время с кэшированием, мс**
1	576.69	3.65
3	736.40	2.85
6	647.37	2.48
12	647.01	3.08
24	628.51	2.54
36	661.81	2.90

Вывод

В результате сравнения времени, необходимого для получения данных для построения MRR отчета, алгоритм с кэшированием выигрывает у алгоритма без кэширование примерно в 250 раз. Данный результат достигается за счет того, что в приложении база данных с кэшем хранит уже готовые для отчета данные, не требующие дополнительнойо бработки, в то время как база данных транзакций в конечном выиде содержит лишь нормированное распределение уплаченных клиентом средств. Несмотря на тот факт, что в приложении в СУБД ClickHouse используется движок Memory, хранящий данные в оперативной памяти, алгоритм с кэшированием все равно выигрывает за счет асимптотической сложности поиска по индексу $O(1)$, что означает, что при любом объеме выборки будет получен одинаковый результат затрачиваемого времени.

Заключение

Во время выполнения курсового проекта было реализовано программное обеспечение, реализующее построение MRR отчета по заданным данным.

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

В результате проведенной работы было разработано программное обеспечение, демонстрирующее возможности разных способов хранения данных и особенности существующих СУБД.

В ходе выполнения эксперементально-исследовательской части было установленно, что алгоритм построения отчета, использующий кэширование, справляется с задачей в 250 раз быстрее алгоритма без кэширования, что дает пользователю более приятный опыт использования приложения.

Литература

[1] Subscription business model - Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Subscription_business_model (дата обращения: 28.05.2021).

[2] 55% of People Subscribe to More Than One of the "Big Three" Streaming Services [Электронный ресурс]. Режим доступа: https://www.cordcuttersnews.com/55-of-people-subscribe-to-more-than-one-of-the-big-three-streaming (дата обращения: 28.05.2021).

[3] Global streaming music subscribers 2020 | Statista [Электронный ресурс]. Режим доступа: https://www.statista.com/statistics/669113/number-music-streaming-subscribers/ (дата обращения: 28.05.2021).

[4] Global SVoD subscription search growth 2020 | Statista [Электронный ресурс]. Режим доступа: https://www.statista.com/statistics/253945/free-streaming-tv-market-leaders-in-the-us/ (дата обращения: 28.05.2021).

[5] Online media and dating services subscriptions since covid 19 by brand | Statista [Электронный ресурс]. Режим доступа: https://www.statista.com/statistics/1116270/subscription-to-online-media-and-dating-services-since-covid-19-by (дата обращения: 28.05.2021).

[6] Subscription Management in 2021 And Beyond | by SoftClouds [Электронный ресурс]. Режим доступа: https://softclouds.medium.com/subscription-management-in-2021-beyond-f823428cf4b4 (дата обращения: 28.05.2021).

[7] MRR – что за зверь, и как его считать [Электронный ресурс]. Режим доступа: https://khanin.info/blog/106 (дата обращения: 28.05.2021).

[8] Software as a service – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Software_as_a_service (дата обращения: 01.06.2021).

[9] Database – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Database (дата обращения: 01.06.2021).

[10] SQL – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/SQL (дата обращения: 01.06.2021).

[11] The most popular database for modern apps | MongoDB [Электронный ресурс]. Режим доступа: https://www.mongodb.com/ (дата обращения: 01.06.2021).

[12] Tarantool – платформа in-memory вычислений [Электронный ресурс]. Режим доступа: https://www.tarantool.io/ru/ (дата обращения: 01.06.2021).

[13] Redis [Электронный ресурс]. Режим доступа: https://redis.io/ (дата обращения: 01.06.2021).

[14] Graph Database Platform | Graph Database Management System | Neo4j [Электронный ресурс]. Режим доступа: https://neo4j.com/ (дата обращения: 01.06.2021).

[15] Couchbase: Best NoSQL Cloud Database Service [Электронный ресурс]. Режим доступа: https://www.couchbase.com/ (дата обращения: 01.06.2021).

[16] memcached - a distributed memory object caching system [Электронный ресурс]. Режим доступа: https://memcached.org/ (дата обращения: 01.06.2021).

[17] JSON – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/JSON (дата обращения: 01.06.2021).

[18] BSON – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/BSON (дата обращения: 01.06.2021).

[19] ACID – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/ACID (дата обращения: 01.06.2021).

[20] Online transaction processing – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Online_transaction_processing (дата обращения: 01.06.2021).

[21] Online analytical processing – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Online_analytical_processing (дата обращения: 01.06.2021).

[22] Greenplum Database [Электронный ресурс]. Режим доступа: https://greenplum.org/ (дата обращения: 01.06.2021).

[23] PostgreSQL: The world's most advanced open source database [Электронный ресурс]. Режим доступа: https://www.postgresql.org/ (дата обращения: 01.06.2021).

[24] ClickHouse - fast open-source OLAP DBMS [Электронный ресурс]. Режим доступа: https://clickhouse.tech/ (дата обращения: 01.06.2021).

[25] Введение | Документация ClickHouse [Электронный ресурс]. Режим доступа: https://clickhouse.tech/docs/ru/engines/table-engines/ (дата обращения: 01.06.2021).

[26] Performance comparison of analytical DBMS [Электронный ресурс]. Режим доступа: https://clickhouse.tech/benchmark/dbms/ (дата обращения: 01.06.2021).

[27] Pabbly Subscriptions - Subscription Billing And Management Software [Электронный ресурс]. Режим доступа: https://www.pabbly.com/subscriptions/ (дата обращения: 01.06.2021).

[28] ChartMogul | Subscription Analytics Platform [Электронный ресурс]. Режим доступа: https://chartmogul.com/ (дата обращения: 01.06.2021).

[29] Business Analytics Platform And KPI Dashboards | Databox [Электронный ресурс]. Режим доступа: https://databox.com/ (дата обращения: 01.06.2021).

[30] Baremetrics: Subscription And Insights for Stripe, Braintree, Recurly And more! [Электронный ресурс]. Режим доступа: https://baremetrics.com/ (дата обращения: 01.06.2021).

[31] JSON Web Token – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/JSON_Web_Token (дата обращения: 01.06.2021).

[32] bcrypt – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Bcrypt (дата обращения: 02.06.2021).

[33] Движки базы данных | Tarantool [Электронный ресурс]. Режим доступа: https://www.tarantool.io/ru/doc/latest/book/box/engines/ (дата обращения: 02.06.2021).

[34] Single-page application – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Single-page_application (дата обращения: 02.06.2021).

[35] Remote procedure call – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Remote_procedure_call (дата обращения: 02.06.2021).

[36] API – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/API (дата обращения: 02.06.2021).

[37] JavaScript – Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/JavaScript (дата обращения: 02.06.2021).

[38] Vue.js [Электронный ресурс]. Режим доступа: https://vuejs.org/ (дата обращения: 02.06.2021).

[39] What is Vuex? | Vuex [Электронный ресурс]. Режим доступа: https://vuex.vuejs.org/ (дата обращения: 02.06.2021).

[40] Vue Router [Электронный ресурс]. Режим доступа: https://router.vuejs.org/ (дата обращения: 02.06.2021).

[41] The Go Programming Language [Электронный ресурс]. Режим доступа: https://golang.org/ (дата обращения: 02.06.2021).

[42] gin-gonic/gin: Gin is a HTTP web framework written in Go (Golang). It features a Martini-like API with much better performance – up to 40 times faster. If you need smashing performance, get yourself some Gin. [Электронный ресурс]. Режим доступа: https://github.com/gin-gonic/gin (дата обращения: 02.06.2021).

[43] gin-gonic/gin: Gin is a HTTP web framework written in Go (Golang). It features a Martini-like API with much better performance – up to 40 times faster. If you need smashing performance, get yourself some Gin. [Электронный ресурс]. Режим доступа: https://github.com/gin-gonic/gin#benchmarks (дата обращения: 02.06.2021).

[44] tarantool/go-tarantool: Tarantool 1.6+ client for Go language [Электронный ресурс]. Режим доступа: https://github.com/tarantool/go-tarantool (дата обращения: 02.06.2021).

[45] mailru/dbr: Additions to Go database/sql for super fast performance and convenience. (fork of gocraft/dbr) [Электронный ресурс]. Режим доступа: https://github.com/mailru/dbr (дата обращения: 02.06.2021).

[46] mongodb/mongo-go-driver: The Go driver for MongoDB [Электронный ресурс]. Режим доступа: https://github.com/mongodb/mongo-go-driver (дата обращения: 02.06.2021).

[47] Empowering App Development for Developers | Docker [Электронный ресурс]. Режим доступа: https://www.docker.com/ (дата обращения: 02.06.2021).