# Содержание

[Содержание 1](#_Toc419227581)

[Введение 2](#_Toc419227582)

[Определение требований 3](#_Toc419227583)

[Предметная область 3](#_Toc419227584)

[Описание бизнес-процессов 3](#_Toc419227585)

[Постановка задачи 4](#_Toc419227586)

[Требования к функционалу 5](#_Toc419227587)

[Требования к реализации 7](#_Toc419227588)

[Сравнение с аналогами 9](#_Toc419227589)

[Архитектура разрабатываемой системы 12](#_Toc419227590)

[Программный интерфейс для ввода данных 13](#_Toc419227591)

[Взаимодейтсвие с программным интерфейсом 15](#_Toc419227592)

[Модуль построения отчетов 16](#_Toc419227593)

[Процесс генерации отчетов 18](#_Toc419227594)

[Приложение для просмотра отчетов 19](#_Toc419227595)

[Проектное решение 21](#_Toc419227596)

[Требования к аппаратной части 21](#_Toc419227597)

[Выбор аппаратного решения 22](#_Toc419227598)

[Преимущества перед другими решениями 23](#_Toc419227599)

[Недостатки перед другими решениями 25](#_Toc419227600)

[Выбор платформы 26](#_Toc419227601)

[Выбор программного решения 30](#_Toc419227602)

[Программная реализация 39](#_Toc419227603)

[Программный интерфейс 39](#_Toc419227604)

[Модуль генерации отчетов 43](#_Toc419227605)

[Приложение для просмотра отчетов 48](#_Toc419227606)

[Руководство пользователя 61](#_Toc419227607)

# Введение

И для большого супермаркета, и для маленькой торговой точки сегодня как никогда актуальна автоматизация торговли, она стала традиционным инструментом современного бизнеса, поскольку сокращает затраты времени и, в конечном итоге, денег. Автоматизация торговли не только приводит к сокращению расходов, но и существенно упрощает процесс ведения бизнеса. Однако зачастую по мере роста у розничных компаний возникают сложности с получением качественной аналитики. Как правило, это связано с тем, что данные хранятся в не удобном для анализа виде, при этом объем накопленных данных с годами постоянно растет. Кроме того, в компаниях, имеющих несколько точек продаж, могут одновременно использоваться разные учетные системы, интеграция между которыми часто слабо развита. Для качественного управления бизнесом и принятия решений этого недостаточно. Очевидно, что для этих задач требуется использование информационного хранилища данных, в котором вся информация доступна «по нажатию кнопки».

Одним из вариантов решения этой проблемы является агрегация данных из точек продаж в одном месте. Однако далеко не все компании готовы создавать для этого собственную инфраструктуру и заниматься ее поддержкой. В особенности это относится к малому бизнесу, для которого подобные решения далеко не всегда экономически целесообразны.

Альтернативой построению собственной инфраструктуры является использование сторонних решений для агрегации данных. Не требуя проведения объемных работ по интеграции и предлагая свои услуги в формате ежемесячной подписки, они могут предоставить владельцам предприятий ценную аналитическую информацию за небольшие деньги.

# Определение требований

## Предметная область

Рассматриваемая предметная область - бизнес-аналитика для предприятий розничной торговли. Потенциальные клиенты разрабатываемого решения – малый и средний бизнес, владеющий одной или несколькими точками продаж, оснащенными системами автоматизации торговли и электронного учета. Рассматриваемое в рамках данного дипломного проекта решение – система сбора, хранения и обработки данных о продажах.

## Описание бизнес-процессов

Данные обо всех проводимых транзакциях попадают в систему учета компании. Информация о транзакциях содержит время проведения сделки, список товаров, сумму покупки, номер терминала или аккаунт продавца, реализовавшего товар, а при наличии у компании программ лояльности – номер скидочной карты или идентификатор покупателя.



Либо сразу после проведения транзакции, либо с заданной периодичностью, данные отправляются из системы учета на веб-сервер системы бизнес-аналитики. После обработки данных, клиент с помощью веб-приложения получает доступ к аналитической информации и отчетам.

## Постановка задачи

Разработать систему автоматизации анализа данных о продажах и построения отчетов с учетом специфики розничной торговли. Основная задача системы – оценка эффективности работы предприятия и помощь в оптимизации продаж.

На основе данных о продажах/транзакциях (количеству и стоимости проданных покупателю товаров), получаемых от систем учета предприятия, POS-систем в точках продаж, либо вручную введенных данных, система должна предоставлять аналитическую информацию, которая может помочь предприятиям в принятии более эффективных решений и повышении уровня продаж.

Система должна поддерживать обработку данных, получаемых из точек продаж, в реальном времени, тем самым предоставляя актуальную информацию, недоступную при ручной обработке данных. Эта информация позволит руководителям предприятий быстрее принимать необходимые решения. Система также должна быть способна анализировать большие объемы информации со многих точек продаж одновременно, избавляя предприятия от необходимости проводить объемные вычисления самостоятельно, а также предоставляя возможность анализировать показатели в глобальном масштабе.

Система должна позволять расчитывать типовые показатели, такие как:

* Общая выручка за месяц;
* Средняя выручка с покупателя;
* Популярные товары;
* Продавцы, реализовавшие больше всего товаров
* Товары, приносящие наибольшую выручку;
* Покупатели, приносящие наибольшую выручку;

Кроме расчета типовых показателей, система должна предоставлять возможность расчитывать сравнительные и агрегированные показатели для двух и более наборов данных:

* Общая выручка с нескольких точек продаж;
* Сравнительная выручка по точкам продаж;
* Изменение уровня продаж;
* Популярные пары товаров;
* Структура продаж;

Необходимо предусмотреть возможность вывода данных в текстовом (табличном) и графическом (в виде графиков) виде.

## Требования к функционалу

* Загрузка и выгрузка данных в формате CSV;
* Генерация отчетов по заданным промежуткам времени;
* Визуализация данных;
* Поддержка аккаунтов пользователей;
* Хранение данных в обезличенном виде;
* Рассчет стоимости использования в зависимости от нагрузки.

### Загрузка и выгрузка данных в формате CSV

Должна быть предусмотрена возможность загружать исходные данные и получать готовые отчеты в формате CSV. Ввод и вывод данных должны быть реализованы как в виде графического интерфейса, так и в виде программного интерфейса (API) для возможности интеграции с другими системами и автоматизации процесса загрущки данных.

### Генерация отчетов по заданным промежуткам времени

Пользователь должен иметь возможность просматривать отчеты за выбранный промежуток времени.

### Визуализация данных

Графический интерфейс системы должен предоставлять возможность визуализации данных в виде графиков и диаграм. Визуализация должна происходить на стороне клиентского приложения, без обращения к серверу.

### Поддержка аккаунтов пользователей

Доступ к системе должен осуществляться только после авторизации пользователя по логину и паролю. Пользователи могут иметь доступ только к отчетам, относящимся к их организации. Пользователи должны иметь возможность изменить пароль. Также необходимо предоставить возможность регистрации новых пользователей для выбранной организации.

Кроме того, пользователь должен иметь возможность управления ключами доступа в программному интерфейсу на случай, если используемых ключ был скомпроментирован.

### Хранение данных в обезличенном виде

Необходимо предусмотреть поддержку обработки данных в обезличенном виде и возможность дальнейшей расшифровки таких данных клиентом. Данные для расшифровки должны храниться на стороне клиентского приложения и не отправляться на сервер.

Пользователь должен иметь возможность загружать список пар ключ-значение в формате CSV, с помощью которых разыменованные наименования товаров, названия точек продаж и имена продавцов и покупателей могли бы быть отображены в исходном виде.

### Рассчет стоимости использования в зависимости от нагрузки

Система должна автоматически измерять объем загруженных пользователем данных и на его основе рассчитывать стоимость использования системы за каждый месяц работы.

## Требования к реализации

* Платформонезависимость;
* Возможность масштабирования системы в зависимости от текущей нагрузки;
* Система не должна требовать от клиента проведения работ по интеграции, покупки оборудования и других предварительных расходов;
* Возможность интеграции с другими системами;
* Клиентское приложение должно работать без установки дополнительного программного обеспечения.

### Платформонезависимость

Система должна работать на любой современной операционной системе, включая Windows Server 2014 и Linux (Debian, Fedora, Arch Linux, Ubuntu Server и др. ), а также на облчаных платформах Microsoft Azure и Amazon Web Services. Также необходимо предусмотреть возможность работы системы на всех популярных веб-серверах (IIS, Apache, Nginx).

### Возможность масштабирования системы в зависимости от текущей нагрузки

Архитектура системы должна быть построена так, чтобы нагрузка могла быть распределена между несколькими серверами. При этом добавление новых серверов должно увеличивать мощность системы линейно и выход из строя любого из серверов не должен сказываться на работоспособности системы в целом. Подключение новых серверов не должно требовать остановки системы. Система должна автоматически подключать и освобождать используемые вычислительные мощности в зависимости от текущей загрузки.

### Система не должна требовать от клиента проведения работ по интеграции, покупки оборудования и других предварительных расходов

Для начала пользования системой не должно требоваться каких-либо работ и предварительных расходов со стороны клиента, за исключением случаев, когда требуется произвести инеграцию системы в существующую инфраструктуру компании.

### Возможность интеграции с другими системами

Система должна иметь возможность интеграции с другими системами при помощи программных интерфейсов. Программный интерфейс должен работать по протоколу HTTP для удаленного доступа к системе, а также поддерживать формат передач данных JSON. Необходимо предусмотреть меры по защите конфиденциальности передаваемых данных и защите от неправомерного доступа к программному интерфейсу. Программный интерфейс системы должен предоставлять все возможности, доступные через графический интерфейс.

### Клиентское приложение должно работать без установки дополнительного программного обеспечения

Пользователь должен иметь доступ к системе с любого устройства, имеющего выход в интернет, без установки какого-либо программного обеспечения. Минимальный требуемый набор операционных систем включает в себя Windows версии XP и старше, Mac OS X любой версии, iOS и Android.

## Сравнение с аналогами

На сегодняшний день рынок систем бизнес-аналитики представлен главным образом «коробочными» решениями, требующими создания и поддержки соответствующей инфраструктуры. Системы, работающие по «облачной» схеме, чаще всего ориентированы на интеграцию с интернет-магазинами. Таким образом, рассматриваемая в данном дипломном проекте система закрывает пустующую нишу рынка, образовавшуюся лишь недавно благодаря развитию информационных технологий и интернета.

Главными недостатками существующих решений являются:

* Необходимость создания и поддержки собственной инфраструктуры.
* Высокая стоимость программного обеспечения и работ по внедрению.

Разрабатываемая в данном дипломном проекте система лишена этих недостатков.

В качестве параметров для сравнения систем бизнес-аналитики можно выделить:

* Тип решения – «коробочное», либо «облачное» программное обеспечение.
* Модель лицензионных отчислений – единоразово при покупке, либо помесячная оплата (аренда).
* Необходимость построения собственной инфраструктуры для работы системы.
* Необходимость внесения изменений в существующую инфраструктуру для интеграции.
* Поддержка клиентским приложением мобильных платформ.
* Возможность интеграции с CRM-системами.

Лидерами рынка систем бизнес-аналитики в России являются три продукта: 1C Предприятие, Norbit SPA BI4Retail и WebFocus. Проведем сравнение этих систем и разрабатываемого решения по параметрам, выбранным выше.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **1С:Предприятие** | **Norbit SPA BI4Retail** | **WebFocus** | **Разрабатываемая система** |
| Тип ПО | коробочное | коробочное | коробочное | облачное |
| Модель использования | покупка | покупка | покупка | аренда |
| Необходимость построения инфраструктуры | да | да | да | нет |
| Необходимость изменения инфраструктуры | нет | нет | да | нет |
| Поддержка мобильных платформ | да | нет | да | да |
| Интеграция с CRM-системами | да | нет | нет | нет |

Как видно из таблицы, разрабатываемое решение обладаем следующими преимуществами:

* Отсутствие необходимости проведения работ по построению новой или изменению существующей инфраструктуры.
* Возможность работы с системой с мобильных платформ.
* Отсутствие предварительных расходов на покупку программного обеспечения.

# Архитектура разрабатываемой системы

Разрабатываемую систему было решено разделить на три изолированных друг от друга компонента:

1. Программный интерфейс для ввода данных
2. Модуль построения отчетов
3. Приложение для просмотра отчетов

Преимущества изоляции компонентов системы:

* **Отказоустойчивость**. Выход из строя компонента системы не сказывается на работе остальных компонентов.
* **Распределение нагрузки**. Загруженность одного из компонентов не влияет на скорость работы других частей системы.
* **Масштабируемость.** Компоненты могут быть масштабируемы независимо друг от друга, с учетом текуших потребностей системы.

Полный цикл от получения данных из точек продаж до просмотра готовых отчетов состоит из следующих этапов:

1. Данные из точек продаж отправляются на программный интерфейс.
2. Программный интерфейс размещает данные во временном хранилище.
3. Модуль для построения отчетов загружает данные из временного хранилища для обработки.
4. Готовые отчеты сохраняются в базу данных с отчетами.
5. Приложение для просмотра отчетов запрашивает необходимые отчеты из базы данных.



## Программный интерфейс для ввода данных

Данные со всех точек продаж в автоматическом режиме поступают на программный интерфейс. В задачи программного интерфейса входит:

* Аутентификация пользователя (приложения)
* Балансировка нагрузки между обработчиками
* Поддержка достаточной пропускной способности
* Трансляция полученных данных в необходимый формат для дальнейшей обработки
* Сохранение данных в промежуточное хранилище
* Сохранение статистики использованных ресурсов для биллинга

### Аутентификация пользователя

Все поступающие на программный интерфейс запросы должны проходить аутентификацию. Программный интерфейс должен самостоятельно определять, к какой именно организации относится клиент, отправивший запрос.

### Балансировка нагрузки между обработчиками

Программный интерфейс должен автоматически распределять нагрузку между имеющимеся обработчиками запросов с целью недопущения их перегрузки. Если какой-либо из обработчиков превысил допустимую нагрузку, последующие запросы должны быть распределены между остальными обработчиками до тех пор, пока перегруженный обработчик не закончит обработку данных.

### Поддержка достаточной пропускной способности

Программный интерфейс должен обеспечивать достаточную пропускную способность приема данных. Специфика розничной торговли предполагает неравномерность нагрузки в разное время суток и время года. Число запросов может изменяться на порядки в течении одного дня или в периоды праздников и распродаж. Приоритетной задачей интерфейса является получение данных от клиентов. Эту задачу допустимо решать в ущерб быстроты их обработки.

### Трансляция полученных данных в необходимый формат для дальнейшей обработки

Формат передачи данных между клиентом и программным интерфейсом может отличаться от формата, который требуется модулю построения отчетов. Программный интерфейс должен проводить трансляцию данных в необходимый формат.

### Сохранение данных в промежуточное хранилище

После трансляции, данные для построения отчетов должны быть сохранены в промежуточное хранилище, к которому будет обращаться модуль построения отчетов. Промежуточное хранилище должно отвечать требованиям быстродействия и одновременного доступа большого числа одновременно работающих с ним обработчиков программного интерфейса и модуля построения отчетов. Кроме количественных показателей производительности, это также требует от хранилища данных поддержку идемпотентности операций, то есть одни и те же данные, отправленные в хранилище несколькими обработчиками, не должны быть сохранены более, чем один раз.

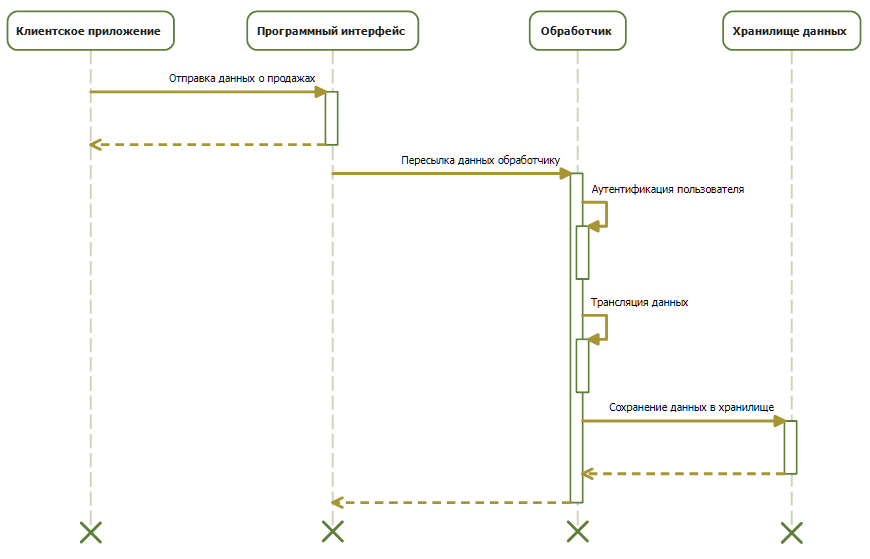
### Сохранение статистики использованных ресурсов для биллинга

Количественные показатели ресурсов, задействованных в обработке запросов, поступающих на программный интерфейс, должны быть сохранены для дальнейшей обработки системой биллинга.

## Взаимодейтсвие с программным интерфейсом

На диаграмме последовательности ниже показан процесс получения данных для обработки программным интерфейсом. Так как наибольший приоритет в работе программного интерфейса отдается пропускной способности, аутентификация пользователя и обработка полученных данных происходят уже после того, как завершается запрос от клиентского приложения. Это позволяет минимизировать время подключения пользователя к интерфейсу, тем самым повысив количество обрабатываемых запросов на единицу времени.

В случае, если аутентификация пользователя не удалась, обработчик завершает работу и данные не отправляются на дальнейшую обработку.



## Модуль построения отчетов

После того, как данные из точек продаж поступили в систему, они отправляются в модуль построения отчетов для обработки. Функции модуля:

* Проверка с заданной периодичностью наличия в хранилище новых данных
* Загрузка всех необходимых для построения отчетов данных
* Обработка полученных данных и рассчет заданных показателей
* Сохранение полученных результатов во временное хранилище
* Загрузка результатов в базу данных готовых отчетов

### Проверка с заданной периодичностью наличия в хранилище новых данных

С заданной периодичностью модуль построения отчетов проверяет, не поступили ли в хранилище новые данные. Если новые данные обнаружены, происходит их загрузка.

### Загрузка всех необходимых для построения отчетов данных

Модуль загружает из временного хранилища все необходимые для построения отчетов за заданный период данные и отправляет их на обработку. При наличии в модуле нескольких обработчиков, нагрузка распределяет между ними.

### Обработка полученных данных и рассчет заданных показателей

Выполняется обработка полученных данных и построение всех необходимых отчетов.

### Сохранение полученных результатов во временное хранилище

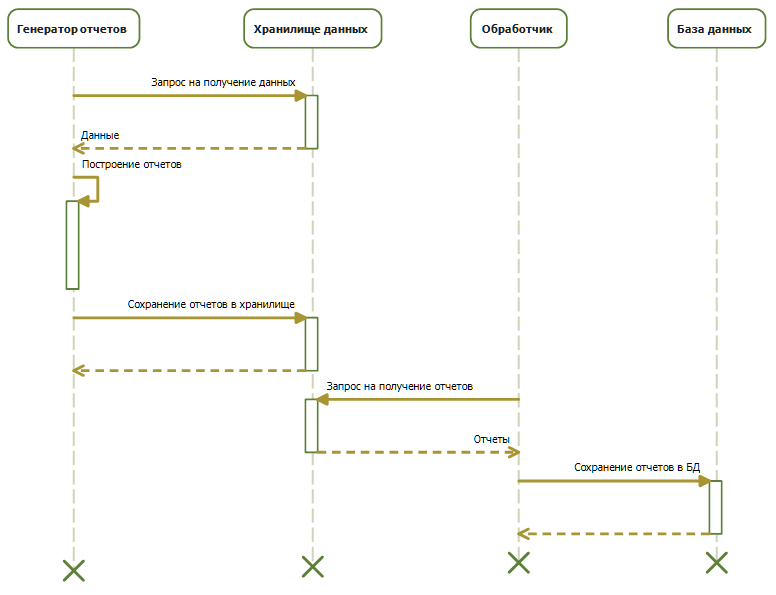
Результаты рассчетов сохраняются обработчиками во временное хранилище.

### Загрузка результатов в базу данных готовых отчетов

После того, как все обработчики закончили проводить рассчеты, полученные данные отправляются из временного хранилища в базу данных, с которой работает приложение для просмотра отчетов.

## Процесс генерации отчетов

На диаграмме последовательностей показан процесс обработки полученных через программный интерфейс данных, их обработка и сохранение результатов во временное хранилище, из которого их позже переносит в базу данных отчетов обработчик.



Данная архитектура позволяет запускать одновременно несколько генераторов отчетов, работающих в параллельном режиме. Использование хранилища данных для временного размещения в нем результатов работы генераторов отчетов позволяет снять лишнюю нагрузку на базу данных и изолировать компоненты системы друг от друга. В случае, если база данных по какой-либо причине оказывается недоступна, обработчик просто отложит задачу переноса отчетов из хранилища в базу на более позднее время, тем самым не прерывая работу генераторов отчетов.

## Приложение для просмотра отчетов

После того, как отчеты попадают в базу данных, они становятся доступны для просмотра через веб-приложение. Основные функции приложения:

* Авторизация пользователей
* Вывод отчетов за выбранный месяц
* Загрузка данных для обработки
* Управление данными для расшифровки

### Авторизация пользователей

Перед тем, как получить доступ к системе, все пользователи должны авторизироваться с помощью логина и пароля. Авторизированному пользователю доступны отчеты только той компании, на которую был зарегистрирован его аккаунт.

### Вывод отчетов

Пользователь имеет возможность вывести отчеты за текущий или любой прошедший месяц и год.

### Загрузка данных для обработки

Кроме загрузки данных напрямую с использованием программного интерфейса, система должна позволять загружать данные вручную. Эта функция реализуется приложением для просмотра отчетов, которое должно поддерживать загрузку CSV-файлов, их трансформацию в необходимый формат и отправку на программный интерфейс для дальнейшей обработки.

### Управление данными для расшифровки

Так как все данные должны храниться в системе в обезличенном виде, в приложении для просмотра отчетов необходимо реализовать функционал, позволяющий расшифровывать данные для удобства просмотра. Вся необходимая для этого информация должна храниться на стороне клиента и не отправляться на сервер.

# Проектное решение

## Требования к аппаратной части

Одним из ключевых требований к системе является платформонезависимость и переносимость программного обеспечения. Так как нельзя однозначно сказать, какие объемы данных должна будет обрабатывать система в долгосрочной перспективе, программное обеспечение должно поддерживать масштабирование. Немаловажной является также является стоимость эксплуатации системы.

Исходя из этих требований, мы получаем следующие критерии выбора аппаратной части:

* Поддержка широкого спектра операционных систем
* Максимальная дешевизна компонентов системы
* Возможность наращивания вычислительных мощностей системы
* Низкая стоимость эксплуатации

### Поддержка широкого спектра операционных систем

Долгий жизненный срок программного обеспечения предполагает смену программных платформ, на которых работает это ПО. Это может быть продиктовано сменой требований организации к набору используемого ПО, наличием в штате сотрудников с необходимыми для обслуживания систем навыками, финансовыми вопросами и т.д.

При выборе аппаратной платформы для системы необходимо учитывать возможность быстрой смены любого программного обеспечения и поддержку как можно более широкого спектра поддерживаемых операционных систем как на текущий момент, так и в долгосрочной перспективе.

### Максимальная дешевизна компонентов системы

Невозможность точно предсказать, насколько финансово успешным будет проект, выдвигает требование к максимальной дешевизне аппаратного решения, достаточного для вывода системы на рынок и последующей эксплуатации.

Аппаратное решение должно быть максимально дешевым. Стоимость замены компонентов платформы, как в случае выхода из строя, так и в случае необходимости повышения производительности системы, должны быть минимальны.

### Возможность наращивания вычислительных мощностей системы

Аппаратное решение должно позволять проводить замену компонентов с целью наращивания вычислительных мощностей системы. Кроме того, решение должно поддерживать возможность объединения нескольких независимых узлов системы в кластеры, тем самым позволяя увеличивать производительность, даже когда будет достигнут предел увеличения производительности отдельных узлов.

## Выбор аппаратного решения

Исходя из заданных требований, наиболее разумным видится использование в качестве аппаратного решения платформы облачных вычислений.

Облачные вычисления - концепция, подразумевающая обеспечение доступа по требованию к пулу вычислительных ресурсов, которые могут быть оперативно предоставлены и освобождены с минимальными затратами.

## Преимущества перед другими решениями

### Отсутствие предварительных расходов

Использование облачной платформы позволяет провести запуск системы без каких-либо начальных расходов. В отличие от использования собственных серверов, при использовании облачных серверов нет необходимости в покупке или аренде оборудования. Оплата облачных вычислительных мощностей производится по факту их использования, а не по предоплате.

Кроме отсутствия предварительных расходов на аппаратную часть системы, в случае использования платного программного обеспечения, облачные решения также не требуют покупки лицензий. Вместо этого оплата лицензионных отчислений включена в стоимость использования облачных ресурсов и также оплачивается почасово, по факту использования. При наличии собственных лицензий на используемое ПО, провайдеры облачных услуг позволяют использовать эти лицензии на облачных серверах, вычитая из стоимости аренды ресурсов стоимость лицензионных отчислений.

### Простота масштабирования системы

Облачные платформы позволяют легко и быстро масштабировать систему, как в сторону увеличения вычислительных мощностей, так и в сторону их уменьшения. Масштабирование может производиться в автоматическом режиме, в зависимости от текущей нагрузки.

Таким образом отпадает необходимость проводить рассчеты и делать прогнозы вычислительных мощностей, которые понадобятся для стабильной работы системы. Возможность масштабирования “вниз” также может снизить расходы в случае, если рассчет необходимых вычислительных мощностей оказался завышенным. В случае покупки или аренды классических серверов, вернуть деньги за уже приобретенные мощности будет невозможно. А в случае заниженного прогноза требуемых мощностей, либо резко возросшей нагрузки на систему, оперативное увеличение доступных мощностей может быть затруднено или вовсе невозможно, тем самым снижая стабильность работы системы.

### Широкая поддержка программного обеспечения

Использование технологий виртуализации позволяет провайдерам облачных услуг предлагать на выбор практически все существующие на сегодняшний день операционные системы.

Кроме того, большинство провайдеров облачных услуг предоставляют предустановленные конфигурации программного обеспечения, заточенные под конкретные задачи, что снимает необходимость проводить настройку ПО самостоятельно.

### Низкая стоимость эксплуатации и обслуживания

Управление и использование облачных мощностей требует гораздо меньших усилий, так как отсутствует необходимость в обслуживании аппаратной части системы. Все работы по обслуживанию, замене и починке оборудования производятся самим провайдером облачных услуг и не требуют наличия в штате соответствующего персонала.

Настройка, масштабирование и обслуживание системы в облачной среде не требует специальных знаний и может проводиться специалистами, имеющими лишь базовые знания в области системного администрирования.

Возможность масштабирования как “вверх”, так и “вниз” позволяет оплачивать только то количество вычислительных мощностей, которое требуется для работы системы в данный момент. Масштабирование системы может выполняться в полностью автоматическом режиме. Это особенно ценно в случае автоматизации работы розничной торговли, когда основная нагрузка системы приходится на светлое время суток, а ночью практически отсутствует. Кроме того, в периоды праздников и распродаж, когда нагрузка на систему резко возрастает, автоматическое масштабирование гарантирует, что система будет доступна и сможет обслуживать клиентов.

## Недостатки перед другими решениями

### Высокая стоимость эксплуатации в долгосрочной перспективе

Как правило, в долгосрочной перспективе стоимость аренды вычислительных мощностей может превышать стоимость покупки и обслуживания собственных аппаратных решений. При использовании платного программного обеспечения, стоимость его аренды через несколько лет выходит дороже покупки лицензий.

Разница в цене между облачными и реальными вычислительными мощностями становится все более заметной по мере роста производительности системы. В этом случае имеет смысл заранее спроектировать систему так, чтобы возможно было осуществить перенос отдельных частей или всей системы целиком на другую аппаратную платформу, без необходимости внесения изменений в программное обеспечение системы.

### Отсутствие доступа к аппаратной части

В некоторых ситуациях, преимущество облачных платформ, заключающееся в отсутствии необходимости обслуживания аппаратной части системы, может быть недостатком. При использовании облачных серверов, выбор используемой аппаратной платформы как правило недоступен или сильно ограничен. В особенности это касается систем ввода-вывода и графических подсистем.

## Выбор платформы

Облачные вычисления предоставляются широким рядом компаний и отличаются по цене и предлагаемым услугам. Модели осблуживания провайдеров облачных услуг можно условно разделить на две категории: инфтраструктура как услуга и платформа как услуга.

### Инфраструктура как услуга

Инфраструктура как услуга (IaaS, Infrastructure-as-a-Service) – модель, при которой пользователю предоставляется возможность самостоятельного управления ресурсами. Пользователь может самостоятельно устанавливать и осблуживать программное обеспечение, включая операционные системы и прикладное ПО. Как правило, пользователю предоставляется ограниченный доступ к настройке аппаратной инфраструктуры – системам хранения данных, сетевым сервисам. Пользователь может развертывать и управлять кластерами виртуальных серверов, организовывать репликацию и резервное копирование данных, устанавливать и настраивать программное обеспечение.

Крупнейшим представителем данной категории является компания Amazon с ее облачной платформой Amazon Web Services (сокращенно AWS). Кроме непосредственно продажи виртуальных серверов, AWS предоставляет доступ ко множеству программных и инфраструктурных решений:

* Amazon Simple Storage Service (Amazon S3) – географически распределенная система хранения данных, имеющая высокую масштабируемость, надежность и скорость работы.
* Amazon CloudFront – система доставки данных (CDN, Content Delivery Network), позволяющая передавать данные конечным пользователям с минимальной задержкой и высокой скоростью, вне зависимости от их географического местоположения.
* Amazon Simple Queue Service (Amazon SQS) – сервис очередей сообщений, позволяющий с высокой надежностью передавать данные между распределенными компонентами системы.
* Amazon Kinesis – сервис буферизации данных, поддерживающий тысячи одновременных подключений, и позволяющий распределить нагрузку на компоненты системы.
* Amazon Elastic Beanstalk – утилита для простого управления кластерами серверов любого размера.
* Amazon Elastic Load Balancing (ELB) – балансировщик нагрузки, позволяющий автоматически добавлять и убирать вычислительные мощности, а также временно отключать от кластера перегруженные или неисправные узлы.
* Amazon Elastic MapReduce (EMR) – кластер серверов Apache Hadoop, интегрированный с другими сервисами платформы.
* AWS Data Pipeline – сервис для выполнения задач по перемещению данных между хранилищами и базами данных.

Как и у большинства других облачных провайдеров, оплата вычислительных и инфраструктурных мощностей производится по факту их использования. Тарификация виртуальных серверов является почасовой и зависит от выбранных аппаратных характеристик. Дополнительно, пользователь оплачивает входящий и исходящий трафик и стоимость хранения данных. В случае, если предполагаемые требования к объему вычислительных мощностей известны заранее, клиент может сделать предоплату по более низкой цене, чем при почасовой аренде.

### Платформа как услуга

Платформа как услуга (PaaS, Platform-as-a-Service) – модель, при которой клиенту предоставляется возможность использования готовой инфраструктуры для размещения программного обеспечения. Контроль и обслуживание инфраструктуры, в том числе сети, серверов, операционных систем, систем хранения данных и других компонентов осуществляется провайдером и как правило недоступен для конфигурации пользователем, за исключением выбора необходимой вычислительной мощности, пропускной способности каналов связи и объема систем хранения данных.

Многие провайдеры, работающие по этой модели, предоставляют доступ к дополнительным сервисам, таким как реляционные и нереляционные базы данных как услуга, серверам с различным предустановленным и неконфигурируемым программным обеспечением, автоматически масштабируемым системам хранения данных, системам анализа данных и другим программным комплексам.

Одним из крупнейших представителей данной категории является облачный сервис Microsoft Azure. Ключевыми компонентами платформы являются:

* Azure Web Sites – сервис для размещения веб-приложений, поддерживающий множество современных языков программирования и фреймворков, включая .NET, Node.js, PHP, Python, Java и другие.
* Azure Virtual Machines – инфраструктурный сервис, схожий по возможностям с IaaS-платформами, предоставляющий виртуальные сервера, работающие на Windows Server 2008/2012 или одном из нескольких дистрибутивов Linux.
* SQL Azure – сервис, основанный на Microsoft SQL Server, обслуживание которого лежит на провайдере, а оплата производится только по потребленным ресурсам.
* Azure CDN – географически распределенная сеть доставки данных.
* HDInsight Hadoop – кластер Apache Hadoop, обслуживанием которого занимается провайдер услуги, а клиенту предоставляется доступ лишь к выполнению прикладных задач.

Исходя из требований к аппаратному решению, наиболее подходящим вариантом является платформа Amazon Web Services. Преимуществами этой платформы для разрабатываемой системы являются:

* Наличие множества готовых сервисов, позволяющих не заниматься созданием и обслуживанием собственных инфраструктурных решений, таких как системы хранения и передачи данных.
* Возможность настройки инфраструктуры исходя из требований и особенностей разработки и эксплуатации системы.
* Доступность широкого рядя программного обеспечения и возможность его настройки под нужды проекта.

## Выбор программного решения

### Язык программирования

Исходя из требования, что клиентское приложение должно работать на компьютерах пользователей без установки какого-либо дополнительного программного обеспечения, а также то, что приложение должно работать как на персональных компьютерах, так и на мобильных устройствах, единственным доступным решением остается разработка веб-приложения.

Существует множество различных платформ и языков программирования, позволяющих разрабатывать веб-приложения. Все они предоставляют примерно одинаковые возможности, поэтому на выбор используемого инструмента в большей степени влияют экономические и организационные факторы, такие как доступность и навыки разработчиков, жизненный цикл и перспективность выбранной технологии, стоимость технической поддержки проекта.

В качестве основы разрабатываемой системы была выбрана платформа Node.js, а в качестве языка программирования – JavaScript. К достоинствам этого выбора можно отнести:

* **Открытость**: Node.js является полностью открытой и бесплатной платформой.
* **Мультиплатформенность**: Node.js работает на всех современных операционных системах, включая Windows, Linux и FreeBSD. Кроме того, платформа поддерживается всеми популярными веб-серверами – Apache, Nginx, IIS, а также может работать и без использования сторонних веб-серверов.
* **Универсальность**: использование языка JavaScript позволяет писать на одном и том же языке как серверную, так и клиентскую часть приложения. Кроме уменьшения числа разработчиков и требований к их навыкам, это также позволяет переиспользовать код между серверной и клиентской частями системы. Кроме того, плафторма Amazon Web Services позволяет использовать Node.js для написания модулей к системе AWS Lambda – вспомогательному сервису, упрощающему интеграцию между различными компонентами AWS.
* **Популярность**: по статистике крупнейшего репозитория открытых проектов GitHub, JavaScript является самым популярным языком разработки на сегодняшний день - около 15% всех проектов написаны на JavaScript. Пакетный менеджер Node.js – NPM является на сегодняшний день крупнейшим ресурсом открытого ПО для разработчиков, и его популярность продолжает расти.

JavaScript – прототипно-ориентированный язык программирования со слабой динамической системой типизации, автоматическим управлением памяти и поддержкой функциональной и императивной парадигм программирования. Изначально являясь языком программирования, использование которого ограничено разработкой приложений для веб-браузеров, с появлением платформы Node.js он превратился в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с системой ввода-вывода, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода.

Благодаря неблокирующей работе системы ввода-вывода, Node.js является хорошим выбором для построения программных интерфейсов (API). Разрабатываемое клиентское приложение построено по «одностраничной» схеме, при которой все необходимые данные приложение получает используя программный интерфейс. Таким образом, сочетание данной схемы разработки и использование Node.js позволяет достичь высокой производительности веб-приложения.

Используемой версией JavaScript был выбран стандарт ECMAScript 6 (ES6). На текущий момент, стандарт ES6 все еще находится на стадии черновика и не был окончательно утвержден (планируемая дата выхода окончательной редакции – июнь 2015 года). Тем не менее, было решено использовать самую свежую версию стандарта с тем, чтобы обеспечить как можно более долгий жизненный цикл разрабатываемой системы. Затраты на доработку кода в случае каких-либо изменений в окончательной редакции стандарта не должны быть существенны, особенно на фоне остальных приемуществ.

Так как далеко не все веб-браузеры имеют поддержку стандарта ES6, исходный код приложения перед публикацей транслируется в наиболее распространенную версию стандарта - ECMAScript 5.

### Операционная система

Как было сказано выше, Node.js работает на всех современных операционных системах, а приложения, написанные на нем, являются платформонезависимыми (при условии, что не используются сторонние модули, работающие не на всех платформах). При таких условиях, при выборе операционной системы на первый план выходят требования к простоте конфигурации и сопровождения, а также надежность и стоимость эксплуатации.

Как будет показано далее, для повышения стабильности работы системы было решено использовать “горизонтальный” вариант масштабирования системы – одновременный запуск нескольких серверов с распределением нагрузки между ними. Использование коммерческих операционных систем при горизонтальном масштабировании может быть экономически затратно, поэтому выбор был сделан в пользу дистрибутива Linux, предоставляемого Amazon. К достоинствам этого варианта можно отнести:

* **Отсутствие необходимости в конфигурации и администрировании системы**: дистрибутив постявляется со всем необходимым предустановленным программным обеспечением и оптимальными настройками для работы Node.js. Дистрибутив активно поддерживается разработчиком, своевременно и в автоматическом режиме получает обновления.
* **Интеграция с другими сервисами**: дистрибутив тесно интегрирован со всеми сервисами AWS, включая ElasticBeanstalk – платформу для автоматического развертывания веб-приложений с поддержкой горизонтального масштабирования.

### Хранилище данных

В качестве хранилища данных было решено использовать сервис Amazon Simple Storage Service (Amazon S3). S3 – веб-служба, предоставляющая возможность для хранения и получения любого объёма данных, в любое время из любой точки сети. На такой выбор повлияли следующие характеристики этого сервиса:

* **Отказоустойчивость и высокая скорость работы**. Сервис поддерживает тысячи одновременных подключений и может сохранять необходимую производительность практически при любых нагрузках на систему. Заявленная гарантированная производительность системы составляет до 300 запросов в секунду, что более чем достаточно для работы разрабатываемой системы при любых нагрузках.
* **Простая интеграция с другими сервисами**. S3 тесно интегрирован со всеми сервисами платформы, включая EMR, используемый для построения отчетов, и Data Pipeline, ответственный за перенос данных из хранилища в базу данных.
* **Простота администрирования**. Сервис не требует какой-либо работы по администрированию и поддержке.

### База данных

Для хранения готовых отчетов, данных компаний, пользователей и биллинга используется СУБД MySQL.

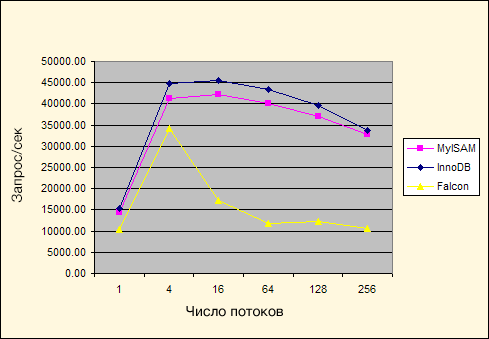
MySQL – реляционная система управления базами данных (СУБД), разрабатываемая компанией Oracle.

MySQL портирована на большое количество платформ: FreeBSD, GNU/Linux, Mac OS X, NetBSD, OpenBSD, Solaris, SunOS, Windows, Windows Server, WinCE и многие другие. В размках разрабатываемой системы выбор был сделан в пользу Linux.

Выбор данной СУБД продиктован следующими факторами:

* **Простота администрирования**. База данных работает на основе сервиса Amazon RDS, позволяющего легко масштабировать сервер СУБД, автоматически выполнять резервное копирование данных и проводить обновление программного обеспечения.
* **Дешевизна эксплуатации**. MySQL является открытым проектом и не требует каких-либо финансовых затрат на лицензионные отчисления.
* **Простота интеграции**. Сервис Data Pipeline поддерживает работу с MySQL без необходимости написания собственных модулей для загрузки данных.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. В нашем случае выбор был сделан в пользу InnoDB, которая имеет более высокую производительность при выборке данных, чем MyISAM.



На графике выше показана сравнительная производительность типов таблиц MyISAM, InnoDB и Falcon на запросах выборки по ключевому значению. Как видно из графика, производительность InnoDB остается стабильно высокой и независимо от нагрузки показывает лучший результат, чем у других типов таблиц.

### Генератор отчетов

Для обработки полученных данных, подсчета показателей и генерации отчетов используется Apache Hadoop – платформа для выполнения распределенных программ, работающих на кластере серверов. Преимуществами такого подхода является:

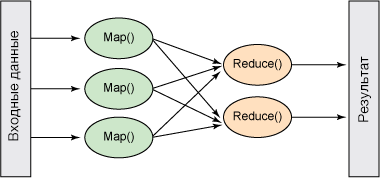
* **Высокая масштабируемость**. Платформа может практически неограниченно масштабироваться и обрабатывать любые объемы данных.
* **Простота работы**. В качестве инструмента для обработки данных используется язык запросов, схожий с SQL. Это позволяет начать работу с системой и приступить к созданию отчетов без какой-либо специальной подготовки.

Apache Hadoop разработан в рамках парадигмы MapReduce, согласно которой приложение разделяется на большое количество элементарных заданий, выполняемых на узлах кластера и сводимых в конечный результат.

Преимущество MapReduce заключается в том, что он позволяет распределенно производить операции предварительной обработки и свертки. Операции предварительной обработки работают независимо друг от друга и могут производиться параллельно. Аналогично, множество рабочих узлов могут осуществлять свертку - для этого необходимо только чтобы все результаты предварительной обработки с одним конкретным значением ключа обрабатывались одним рабочим узлом в один момент времени. Хотя этот процесс может быть менее эффективным по сравнению с более последовательными алгоритмами, MapReduce может быть применен к большим объёмам данных, которые могут обрабатываться большим количеством серверов.

Таким образом, процесс обработки данных состоит из двух этапов:

1. **Шаг Map**: предварительная обработка данных. Главный узел кластера разделяет входные данные на части и распределяет их между остальными узлами. Операции предварительно обработки работают независимо друг от друга и могут производиться параллельно.
2. **Шаг Reduce**: свертка обработанных данных. Главный узел кластера получает ответы от рабочих узлов и на их основе формирует результат вычисления. Аналогично операции map, reduce может производиться параллельно мнодеством узлов кластера.



Для простоты развертывания и управления кластером Hadoop используется сервис Amazon EMR. Сервис использует специальную сборку Hadoop, включающую модули для интеграции с другими сервисами платформы. Кроме всего прочего, это позволяет использовать хранилище S3 в качестве распределенной файловой системы кластера, без необходимости перемещать куда-либо данные, которые были загружены программным интерфейсом системы.

За работу шага Map ответственнен фреймворк Apache Pig, использующий высокоуровневый язык программирования Pig Latin. Использование Pig позволяет проводить предварительную обработку данных без необходимости написания низкоуровневого кода для управления работой кластера.

На шаге Reduce используется фреймфорк Apache Hive, который предоставляет язык запросов HiveSQL, схожий с SQL. Использование этого фреймворка также позволяет абстрагироваться от низкоуровневых деталей операции свертки и сосредоточиться непосредственно на обработке данных.

# Программная реализация

## Программный интерфейс

### Принцип работы

Внешним элементом программного интерфейса, с которым взаимодействует пользовательское приложение в точках продаж является веб-сервис Amazon Kinesis. Сервис служит буфером, позволяющим получать данные одновременно от сотен клиентов и распределять их между обработчиками.

В данной реализации в качестве обработчиков выступает приложение, работающее на сервисе AWS Lambda. Приложение представляет собой модуль Node.js, принимающий на вход данные в формате JSON, полученные программным интерфейсом.



Обработчик вызывается автоматически при поступлении данных в буфер. Каждый обработчик получает за раз не более 10 запросов на загрузку данных. Количество используемых обработчиков автоматически изменяется, чтобы оперативно обрабатывать поступающие данные. После того, как обработчик заканчивает работу, модуль отключается и тарификация используемых ресурсов останавливается, что позволяет не нести расходы за простои.

Тарификация пользователей системы производится по факту использованных вычислительных ресурсов. Для упрощения рассчетов, общий объем вычислений приравнивается к количеству загруженных через программный интерфейс записей и продажах товаров.

### Формат данных

Клиентские приложения взаимодействуют с программным интерфейсом по протоколу HTTP. Все данные передаются в сериализованном виде в формате JSON. Корректный формат входных данных имеет следующий вид:

|  |  |
| --- | --- |
| sales | Массив записей о приобретенных товарах. |
| check | Уникальный идентификатор покупки. Каждая покупка может включать в себя несколько записей о приобретенных товарах. |
| customer | Уникальный идентификатор покупателя (может отсутствовать). |
| item | Уникальный идентификатор товара. |
| location | Уникальный идентификатор точки продажи. |
| price | Цена реализованного товара (с учетом скидок). |
| staff | Уникальный идентификатор продавца, реализовавшего данный товар (может отсутствовать). |
| time | Время совершения покупки. |

{

"sales": [

{

"check": "check1",

"customer": "customer1",

"item": "item1",

"location": "location1",

"price": 120.5,

"staff": "staff1",

"time": "2015-04-27T15:19:07.294Z"

},

{

"check": "check1",

"customer": "customer1",

"item": "item2",

"location": "location1",

"price": 200,

"staff": "staff1",

"time": "2015-04-27T15:19:07.294Z"

},

{

"check": "check2",

"item": "item2",

"location": "location2",

"price": 200,

"staff": "staff2",

"time": "2015-04-27T16:20:05.344Z"

},

]

}

Обработчик проверяет корректность полученных данных и преобразовывает их в формат CSV, который затем сохраняется в хранилище данных S3. Хранилище настроено таким образом, чтобы данные, загруженные более недели назад, автоматически удалялись. Это позволяет не хранить данные, для которых уже были сгенерированы все необходимые отчеты. Данные в хранилище имеют аналогичную структуру, но вместо формата JSON используется формат CSV:

1,location1,"2015-04-27T15:19:07.294Z",check1,staff1,customer1,item1,120.5

1,location1,"2015-04-27T15:19:07.294Z",check1,staff1,customer1,item2,200

1,location2,"2015-04-27T16:20:05.344Z",check2,staff2,,item2,200

### Тарификация

После обработки полученных через программный интерфейс данных, обработчик подсчитывает количество записей о приобретенных товарах и отправляет эту информацию вместе с уникальным идентификатором компании, на которую зарегистрирован текущий пользователь, в очередь сообщений для дальнейшей обработки системой биллинга. Формат сообщений имеет следующий вид:

{

"uuid": "aa74268e-a51b-4ed0-bec5-5ffba2f6015a",

"company": 1,

"sales": 3,

"time": "2015-04-27T21:35:16.129Z"

}

|  |  |
| --- | --- |
| uuid | Уникальный идентификатор сообщения. |
| company | Уникальный идентификатор компании. |
| sales | Число полученных записей о продажах товаров. |
| time | Время обращения к программному интерфейсу. |

Система биллинга раз в сутки загружает сообщения, сохраненные в очереди и подсчитывает общее число записей, которое были загружены в систему каждой компанией. Эта информация затем сохраняется в базе данных приложения для просмотра отчетов и становится доступна для просмотра пользователями.

## Модуль генерации отчетов

### Принцип работы

С заданной периодичностью сервис Amazon EMR развертывает кластер виртуальных серверов под управлением Apache Hadoop. В задачи модуля входит:

1. Загрузка данных из хранилища
2. Распределение данных между узлами-обработчками
3. Рассчет показателей и сохранение данных в хранилище
4. Перенос готовых отчетов из хранилища в базу данных



### Загрузка данных из хранилища

В качестве одного из параметров, кластер получает интервал времени, для которого требуется рассчитать необходимые показатели и сгенерировать отчет. Перед тем, как данные могут быть отправлены узламикластера, они должны быть загружены в распределенную файловую систему (Hadoop Distributed File System, HDFS). Благодаря тому, что данные уже находятся в хранилище S3, которая имеет интеграцию с Hadoop, нам не требуется предварительно перемещать их куда-либо для того чтобы главный узел кластера имел к ним доступ.

Формат CSV, в котором хранятся данные, также поддерживается платформой и не требует предварительной обработки. Для загрузки данных используется следующий Pig-скрипт:

sales =

LOAD '$INPUT/sales' USING CSVExcelStorage(',', 'NO\_MULTILINE', 'UNIX', 'SKIP\_INPUT\_HEADER', 'SKIP\_OUTPUT\_HEADER')

AS (

company: int,

location: chararray,

time: datetime,

check: chararray,

staff: chararray,

customer: chararray,

item: chararray,

price: float

);

### Распределение данных между узлами-обработчками

После предварительной загрузки данных, необходимо выделить показатель, исходя из которого они будут распределятся между узлами кластера. В качестве такого показателя был выбран уникальный идентификатор покупки (чека). Таким образом, все записи, относящиеся к одной покупке, всегда будут находиться только на одном узле кластера.

Для удобства обработки, данные дополнительно разбиваются по трем категориям:

* *checks\_by\_location*: Данные о компании, точки продажи, продавце и покупателе, относящиеся к каждой покупке.
* *checks\_totals*: Общая сумма каждой покупки.
* *items\_by\_check*: Данные о товарах, относящихся к каждой покупке.

Дальнейшее распределение данных между узлами происходит автоматически, под управлением Apache Pig.

### Рассчет показателей и сохранение данных

На этапе вычислений и свертки данных используется фреймворк Apache Hive. Перед тем, как данные, которые были подготовлены на предыдущем этапе, могут быть использованы, необходимо создать виртуальные таблицы Hive, основанные на данных, полученных от Pig:

DROP TABLE IF EXISTS checks\_by\_location;

CREATE EXTERNAL TABLE checks\_by\_location

(

check string, company int, location string, customer string, staff string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION ${INPUT}/checks\_by\_location/';

Для сохранения результатов вычислений также необходимо создать таблицы необходимого формата. При создании таблиц укажем, что данные должны быть сохранены в формате CSV:

DROP TABLE IF EXISTS report\_item\_pair;

CREATE TABLE report\_item\_pair

(

company int,

location string,

item1 string,

item2 string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '${OUTPUT}/report-item-pair';

Для построения запросов используется язык HiveSQL, очень похожий на SQL. К примеру, для рассчета общего числа продаж каждого из товаров можно использовать следующий запрос:

INSERT OVERWRITE TABLE report\_item\_popular

SELECT checks\_by\_location.company, checks\_by\_location.location, items\_by\_check.item, COUNT(items\_by\_check.item)

FROM items\_by\_check

JOIN checks\_by\_location ON items\_by\_check.check = checks\_by\_location.check

GROUP BY checks\_by\_location.company, checks\_by\_location.location, items\_by\_check.item;

В общей сложности, на данном этапе происходит рассчет следующих показателей:

* *report\_customer\_average*: Средняя выручка с каждой покупки.
* *report\_customer\_top*: Рейтинг покупателей, приносящих максимальный доход.
* *report\_item\_pair*: Популярные пары товаров.
* *report\_item\_popular*: Рейтинг популярности товаров.
* *report\_item\_top*: Товары, приносящие наибольший доход.
* *report\_staff\_top*: Рейтинг продавцов по общей выручке.
* *report\_sales\_total*: Общая выручка с разбивкой по точкам продаж.

По мере выполнения вычислений, полученные данные сохраняются в хранилище в формате CSV, откуда они затем будут перемещены в базу данных приложения для просмотра отчетов.

### Перенос отчетов из хранилища в базу данных

После завершения работы кластера и сохранения отчетов, запускается дополнительный обработчик, который считывает результаты вычислений из хранилища и выполняет SQL-скрипты для вставки данных в базу отчетов.

В случае, если база данных недоступна, обработчик откладывает свою работу на 10 минут до тех пор, пока операция вставки не завершится успешно.

При наличии в базе данных записей с ключевым значением, совпадающим с переносимыми данными, вместо вставки происходит обновление данных. Это необходимо для случаев, когда построение отчетов было выполнено дважды для одно и того же набора данных, к примеру при перессчете агрегированных показателей в течение одного дня.

Пример SQL-запроса для сохранения данных, учитывающего возможность совпадения ключевых полей:

INSERT INTO `report\_customer\_average` (company\_id, location\_key, date, value) VALUES(?, ?, ?, ?) ON DUPLICATE KEY UPDATE value = values(value);

Управлением процесса переноса управляет сервис Amazon Data Pipeline, скрипт для работы которого можно найти в приложении к данному дипломному проекту.

## Приложение для просмотра отчетов

### Принцип работы приложения

Приложение для просмотра отчетов является веб-приложением, построенным по одностраничному принципу: загрузка с сервера выполняется один раз, после чего все необходимые данные приложение получает без перезагрузки страницы. Это позволяет повысить скорость работы и уменьшить время загрузки приложения.



Приложение работает на кластере из веб-серверов nginx, который управляется балансировщиком нагрузки. В зависимости от числа активных клиентов, балансировщик автоматически создает или отключает виртуальные сервера и равномерно распределяет нагрузку между ними.

Статические файлы клиентского приложения размещаются на серверах сети доставки данных для обеспечения более быстрой загрузки и снижения нагрузки на веб-сервера приложения.

Серверная часть веб-приложения построена на основе фреймворка Express.js и имеет следующую структуру:

* Страница входа в систему. Перед использованием системы все пользователи должны пройти аутентификацию. В случае, если пользователь пытается открыть какую-либо страницу или получить данные через программный интерфейс, не будучи аутентифицированным, он автоматически перенаправляется на страницу входа в систему.
* Страница приложения для просмотра отчетов. Основной раздел сайта, на котором располагается клиентское приложение, с помощью которого пользователь взаимодействует с системой.
* Программный интерфейс. Веб-сервис, через который клиентское приложение получает данные отчетов, биллинга и управляет аккаунтом пользователя.

Аутентификация пользователя осуществляется с помощью браузерных cookies, хранящих зашифрованный уникальный идентификатор текущей сессии пользователя. Так как приложение работает на кластере серверов, не имеющих связи друг с другом, для хранения сессий пользователей используется внешнее хранилище на основе СУБД Redis.

Redis (от REmote DIctionary Server) - сетевое журналируемое хранилище данных типа «ключ-значение». Redis является нереляционной высокопроизводительной СУБД, оптимизированной для работы в кластере. Все данные Redis хранит в виде словаря, в котором ключи связаны со своими значениями. Redis поддерживает такие высокоуровневые операции, как объединение и разность наборов, а также их сортировку. Высокая производительность Redis обуславливается тем, что все данные хранятся в оперативной памяти.

Redis поддерживает репликацию типа master-slave. Репликация помогает защитить данные, копируя их на другие сервера. Репликация также может быть использована для увеличения производительности, так как запросы на чтение могут обслуживаться slave-узлами.

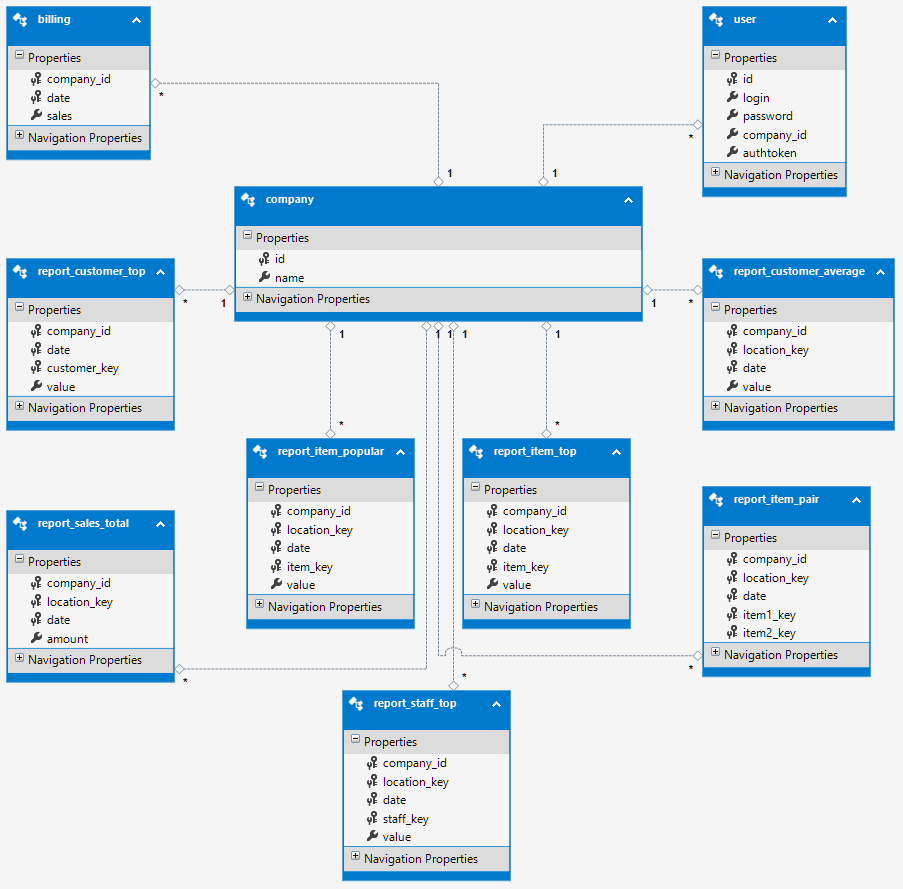
Упрощенная схема приложения изображена на рисунке ниже:



Кластером веб-серверов Node.js управляет балансировщик нагрузки. Для повышения производительности приложения, балансировщик автоматически направляет все запросы одного пользователя на один и тот же сервер. Это позволяет хранить данные о сессии пользователя в памяти сервера, не обращаясь каждый раз к хранилищу Redis.

### Структура базы данных

База данных приложения имеет следующую структуру:



### Описание таблиц

Таблица **company** содержит записи о компаниях, зарегистрированных в системе.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| id | int | Уникальный идентификатор |
| name | varchar(255) | Название компании |

В таблице **user** хранятся записи о пользователях системы.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| id | int | Уникальный идентификатор |
| login | varchar(32) | Логин пользователя |
| password | varachar(255) | Захешированный пароль |
| company\_id | int | ID компании, на которую зарегистрирован пользователь |
| authtoken | varchar(255) | Уникальный ключ для авторизации на программном интерфейсе системы |

Таблица **billing** содержит записи о количестве запросов, обработанных системой для каждой компании, исходя из которых рассчитывается стоимость использования системы.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | date | День |
| sales | int | Число запросов |

Таблица **report\_customer\_top** содержит отчеты о покупателях, приносящих наибольшую выручку.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | datetime | Дата |
| customer\_key | varchar(255) | Уникальный идентификатор покупателя |
| value | decimal(10,4) | Общая выручка |

Таблица **report\_customer\_average** содержит отчеты о средней выручке с каждой покупки.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | datetime | Дата |
| location\_key | varchar(255) | Уникальный идентификатор точки продажи |
| value | decimal(10,4) | Средняя выручка |

Таблица **report\_item\_pair** содержит отчеты о популярных парах товаров.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | datetime | Дата |
| location\_key | varchar(255) | Уникальный идентификатор точки продажи |
| item1\_key | varchar (255) | Первый товар пары |
| item2\_key | varchar (255) | Второй товар пары |

Таблица **report\_item\_popular** содержит отчеты о количестве проданных товаров.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | datetime | Дата |
| location\_key | varchar(255) | Уникальный идентификатор точки продажи |
| item\_key | varchar (255) | Уникальный идентификатор товара |
| value | int | Количество проданных единиц товара |

Таблица **report\_item\_top** содержит отчеты о выручке с каждого проданного товара.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | datetime | Дата |
| location\_key | varchar(255) | Уникальный идентификатор точки продажи |
| item\_key | varchar (255) | Уникальный идентификатор товара |
| value | decimal(10,4) | Общая выручка с товара |

Таблица **report\_sales\_total** содержит отчеты об общей выручке с каждой из точек продаж.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | datetime | Дата |
| location\_key | varchar(255) | Уникальный идентификатор точки продажи |
| amount | decimal(10,4) | Общая выручка с точки продажи |

Таблица **report\_staff\_top** содержит отчеты о выручке, которую принес каждый из продавцов.

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Назвначение |
| company\_id | int | ID компании |
| date | datetime | Дата |
| location\_key | varchar(255) | Уникальный идентификатор точки продажи |
| staff\_key | varchar (255) | Уникальный идентификатор продавца |
| value | decimal(10,4) | Общая выручка |

### Программный интерфейс

Взаимодействие с программным интерфейсом приложения происходит по протоколу HTTP, все данные передаются в формате JSON. Аутентификация пользователя производится с помощью cookie, которую пользователь получает после успешного входа в систему. Программный интерфейс предоставляет следующие методы:

|  |  |  |
| --- | --- | --- |
| HTTP-метод | URL | Операция |
| GET | /api/profile | Получение данных об аккаунте пользователя. |
| POST | /api/profile/change-password | Запрос на смену пароля пользователя. |
| POST | /api/profile/sign-up | Запрос на регистрацию нового пользователя. |
| POST | /api/profile/create-company | Запрос на регистрацию новой организации. |
| POST | /api/profile/generate-auth-token | Запрос на создание нового ключа доступа к программному интерфейсу. |
| GET | /api/report/customer-average | Отчет о среднем чеке. |
| GET | /api/report/customer-top | Отчет о постоянных покупателях. |
| GET | /api/report/item-pair | Отчет о популярных парах товаров. |
| GET | /api/report/item-popular | Отчет о популярных товарах. |
| GET | /api/report/item-top | Отчет о прибыльных товарах. |
| GET | /api/report/sales-total | Отчет о продажах. |
| GET | /api/report/staff-top | Отчет о лучших продавцах. |
| POST | /api/report | Отправка данных о продажах. |

Пример выполнения запроса на получение отчета о покупателях, которые принесли наибольшую выручку с 8 по 9 мая 2015 года:

GET /api/report/customer-top?date=2015-05-08&days=2

[

{

"customer\_key": "customer1",

"date": "2015-05-08",

"value": 1764

},

{

"customer\_key": "customer9",

"date": "2015-05-08",

"value": 2866.25

},

{

"customer\_key": "customer8",

"date": "2015-05-08",

"value": 1085.75

},

{

"customer\_key": "customer5",

"date": "2015-05-09",

"value": 1479.5

},

{

"customer\_key": "customer6",

"date": "2015-05-09",

"value": 3308

}

]

|  |  |
| --- | --- |
| Поле | Назначение |
| customer\_key | Уникальный идентификатор покупателя. |
| date | День, к которому относится текущая запись. |
| value | Выручка с текущего покупателя за день. |

Наименования товаров, названия точек продаж и имена покупателей и продавцов хранятся в базе данных в обезличенном виде. Получив от программного интерфейса данные отчета, клиентское приложение самостоятельно сопоставляет идентификатор разыменованной записи и отображает ее пользователю в исходном виде. Таким образом сохраняется конфиденциальность данных. Схема работы этого алгоритма показана ниже:



### Клиентское приложение

В основе клиентского приложения лежит JavaScript-фреймворк AngularJS, разрабатываемый Google. Фреймворк построен вокруг шаблона проектирования MVC (Model-View-Controller).

Шаблон MVC – это схема разработки, с помощью которой данные приложения, пользовательский интерфейс и взаимодействие с пользователем могут быть разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные:



### Подготовка кода перед публикацией

Так как клиентское приложение написано с использованием JavaScript версии ES6, которая поддерживается не всеми распространенными веб-браузерами, перед публикацией кода над ним выполняется трансляция в JavaScript версии ES5. Пример такой трансляции представлен ниже.

Готовый код подвергается минификации, компрессии по алгоритму Deflate и размещается на серверах сети доставки данных (Content delivery network, CDN) Amazon CloudFront, что позволяет в несколько раз уменьшить время загрузки страницы приложения по сравнению с размещением клиентского кода непосредственно на сервере с приложением и без дополнительной обработки.



Исходный код:

window.app.Store = class Store {

constructor() {

this.\_storage = null;

}

clear() {

this.\_storage = null;

return Promise.resolve();

}

read() {

var self = this;

return new Promise(resolve => {

resolve(self.\_storage);

});

}

};

Транслированный код:

window.app.Store = (function () {

function Store() {

\_classCallCheck(this, Store);

this.\_storage = null;

}

\_createClass(Store, {

clear: {

value: function clear() {

this.\_storage = null;

return Promise.resolve();

}

},

read: {

value: function read() {

var self = this;

return new Promise(function (resolve) {

resolve(self.\_storage);

});

}

}

}); return Store;

})();

Минифицированный код:

window.app.Store=function(){function Store(){\_classCallCheck(this,Store),this.\_storage=null}return \_createClass(Store,{clear:{value:function(){return this.\_storage=null,Promise.resolve()}},read:{value:function(){var self=this;return new Promise(function(resolve){resolve(self.\_storage)})}},write:{value:function(value){var self=this;return new Promise(function(resolve){self.\_storage=value,resolve()})}}}),Store}()

### Особенности разработки графического интерфейса

Графический интрефейс приложения построен с использованием языка размертки HTML и каскадных таблиц стилей CSS. Как и в случае с кодом JavaScript, конечный код разметки страниц и стилей получается методом трансляции.

В случае с HTML, исходным языком разметки служит язык шаблонов Jade, который является основным языком шаблонов фреймворка Express. Пример шаблона Jade:

extends layout\_base

block head

title Ошибка сервера

block body

.container: .row: div(class='col-md-4 col-md-offset-4'): div(class='login-panel panel panel-default')

.panel-heading: h3.panel-title Ошибка сервера

.panel-body

div(class="alert alert-danger")

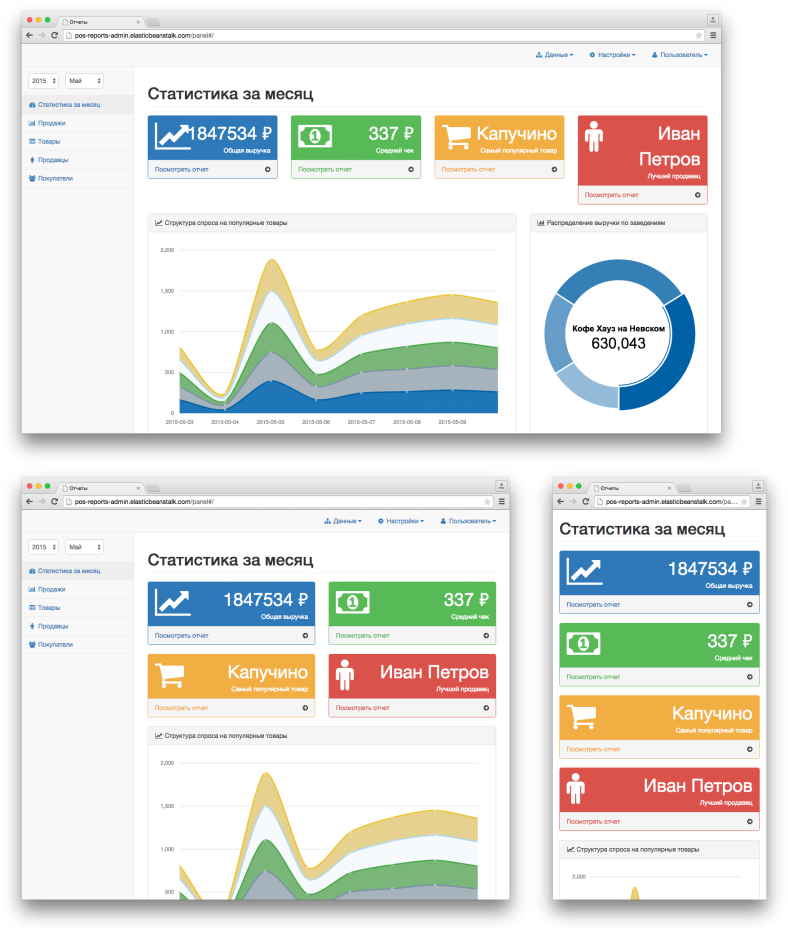
p

if error

= error

button(class="btn btn-primary btn-lg btn-block",onclick="window.location='/'"): | OK

В качестве препроцессора таблиц CSS в приложении используется язык LESS, который является подмножеством диалекта CSS.

Использование препроцессоров JavaScript, HTML и CSS позволяет заметно сократить время разработки приложения за счет более удобного синтаксиса и расширенных возможностей по сравнению со стандартными средствами разработки веб-приложений.

Одним из требований к клиентскому приложению является поддержка мобильных платформ. Как правило, современные мобильные платформы способны работать с любыми веб-приложениями без каких-либо доработок. Однако для обеспечения удобства просмотра приложения на маленьких экранах мобильных телефонов и планшетов, в разработке пользовательского интерфейса использованы методы так называемой адаптивной верстки. При изменении размеров окна веб-браузера, приложение автоматически перемещает элементы графического интерфейса для лучшего их отображения при текущих размерах экрана.

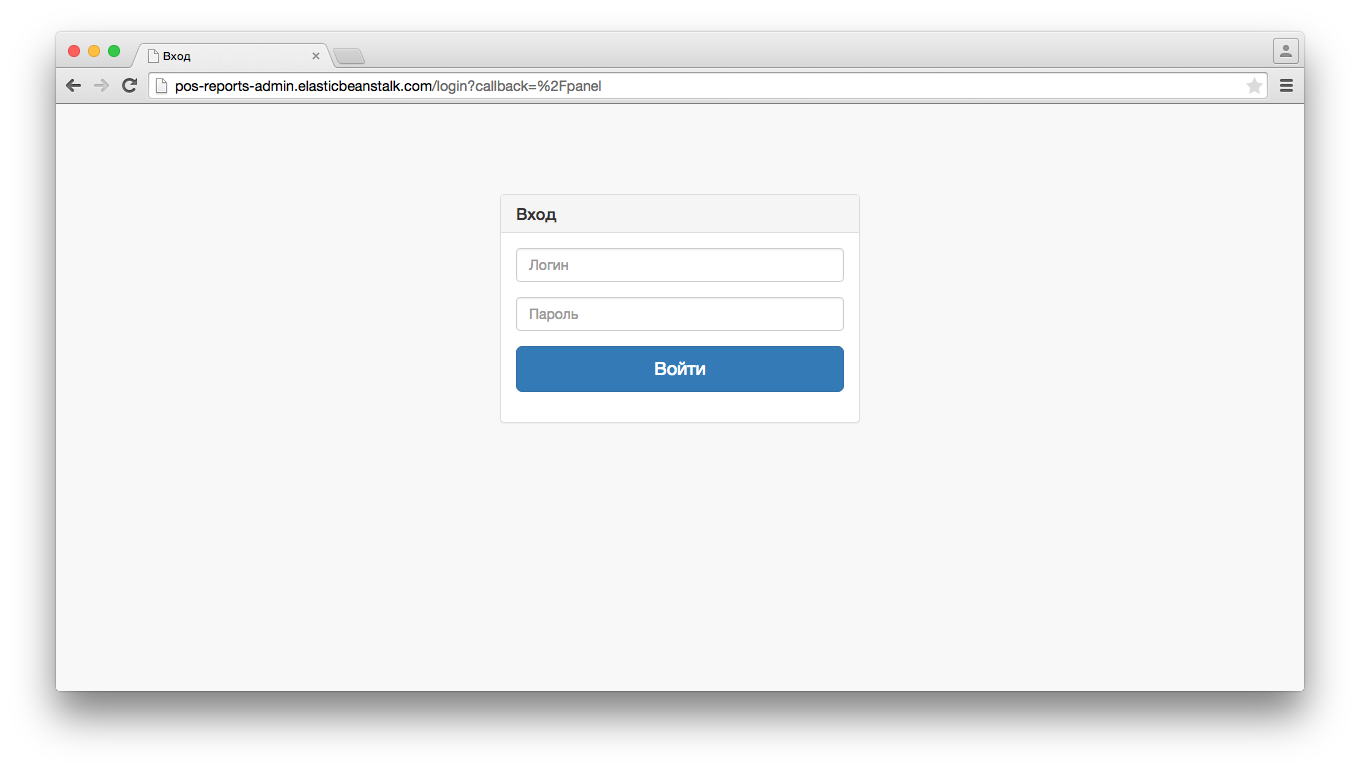
Таким образом, одно и то же приложение может без каких-либо ограничений может работать как на персональных компьютерах, так и на планшетах и смартфонах.

# Руководство пользователя

Для доступа к приложению для просмотра отчетов, пользователь должен перейти на веб-сайт, на котором располагается приложение.

### Страница входа в систему

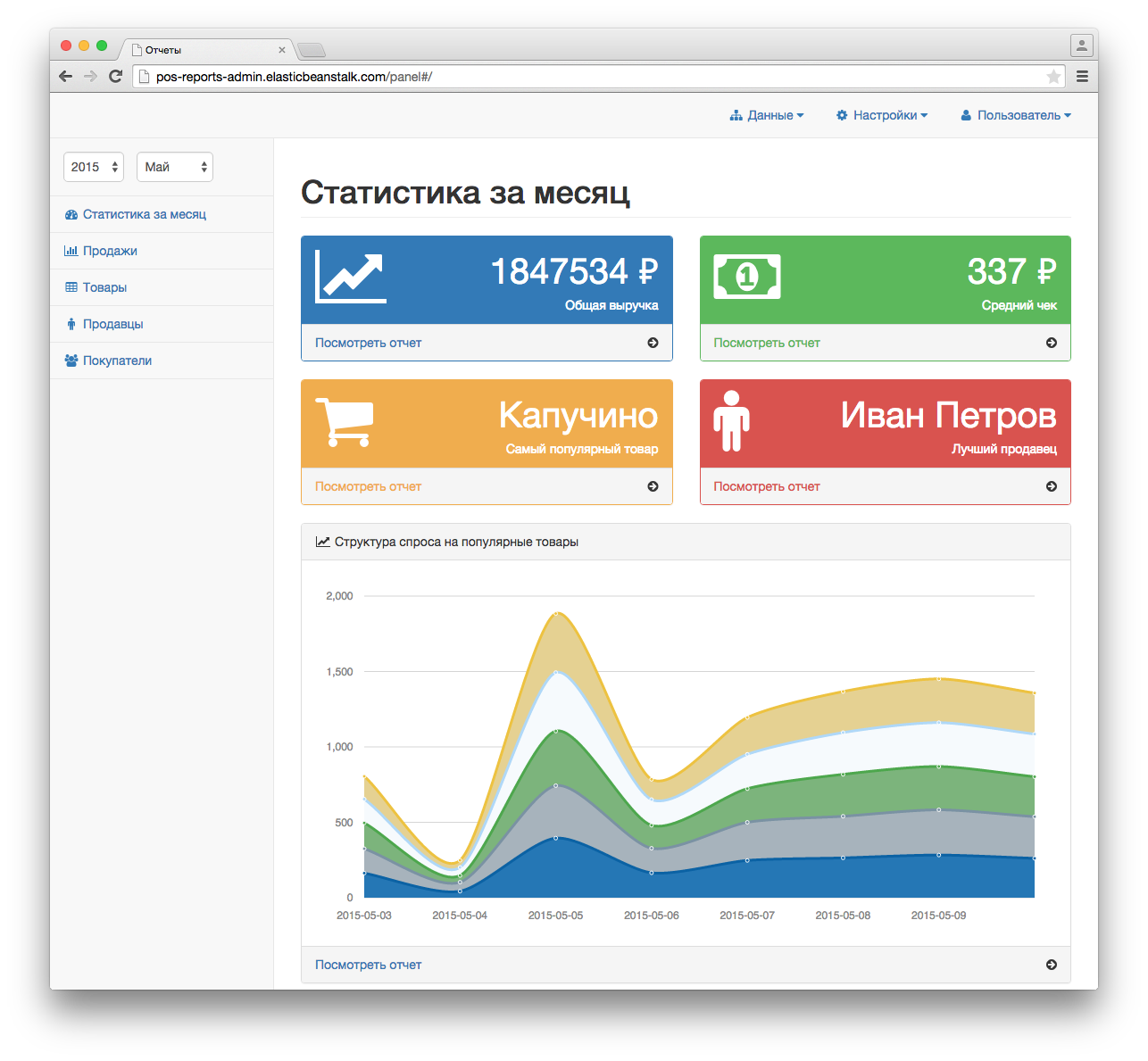
Для входа в систему пользователь должен ввести свой логин и пароль.



После успешного входа, пользователь перенаправляется на главную страницу просмотра отчетов.

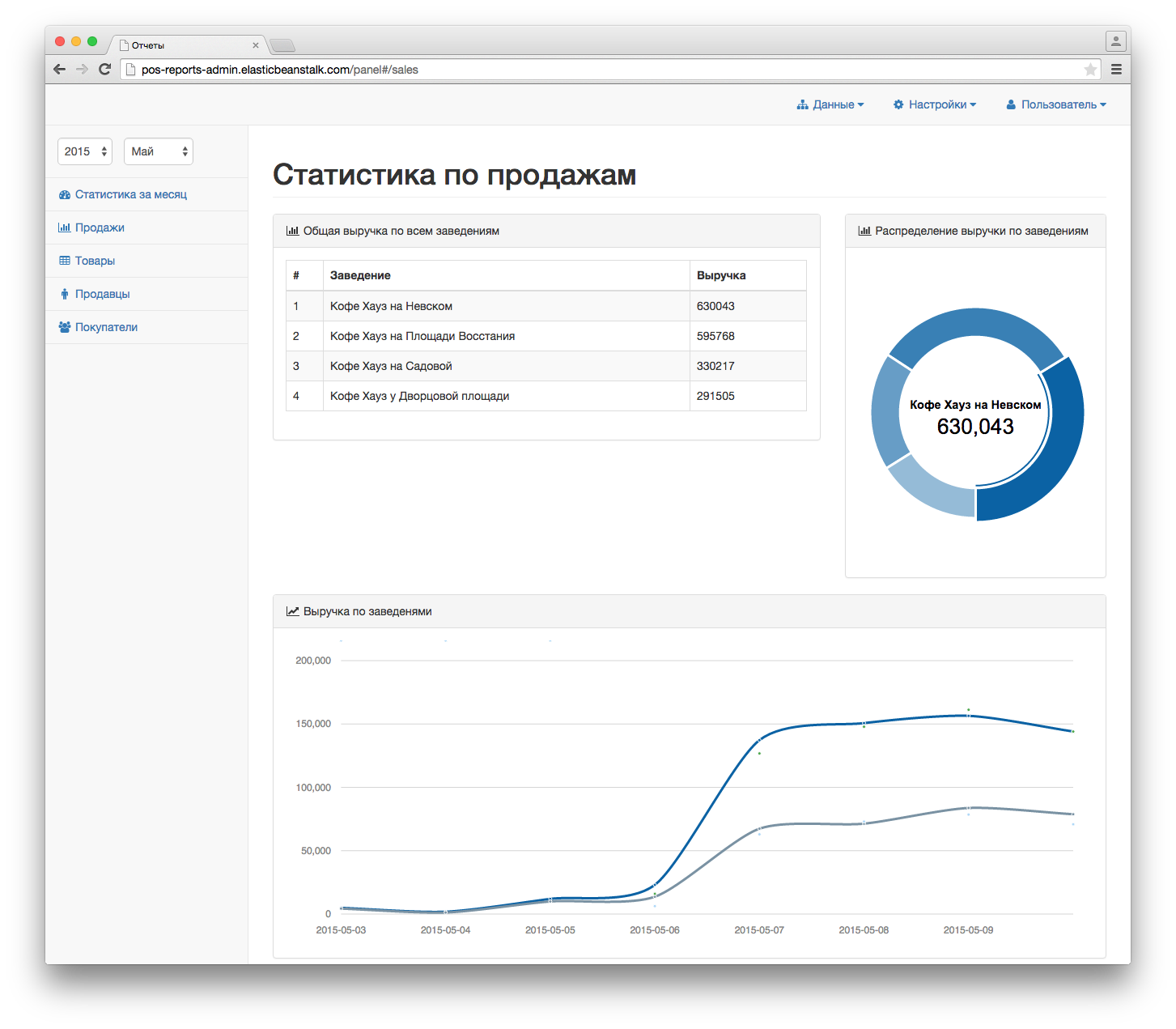
### Главная страница

На главной странице выводится общая статистика работы организации за выбранный месяц: общая выручка, размер среднего чека, самый популярный товар и лучший продавец. Ниже располагается график изменения структуры спроса на самые популярные товары. Внизу каждого информационного блока есть ссылка для перехода на страницу с детальным отчетом.



Используя форму в левом верхнем углу, пользователь может выбрать месяц и год за который нужно вывести отчет. Ниже формы располагается главное меню, через которое можно перейти к подробным отчетам. В правом верхнем углу располагается дополнительное меню, с помощью которого можно перейти на экраны загрузки и управления данными, регистрации новых пользователей, профилю текущего пользователя, либо выйти из системы.

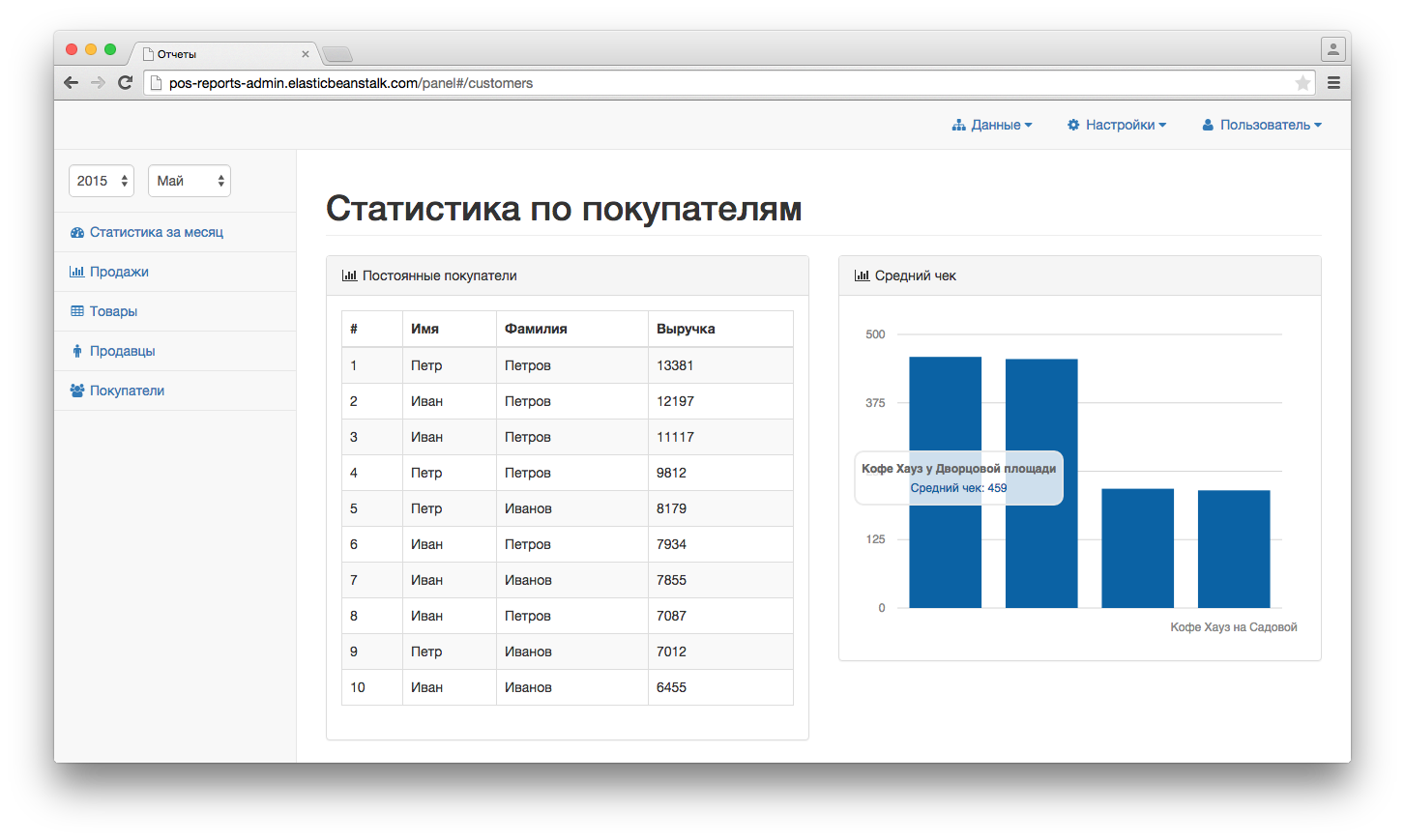
### Статистика продаж



Страница статистики продаж содержит отчет по общей выручке с разбивкой по отдельным заведениям, а также показывает с помощью круговой диаграммы распределение выручки между заведениями. При наведении указателя на диаграмму можно увидеть подробную информацию по каждому из заведений.

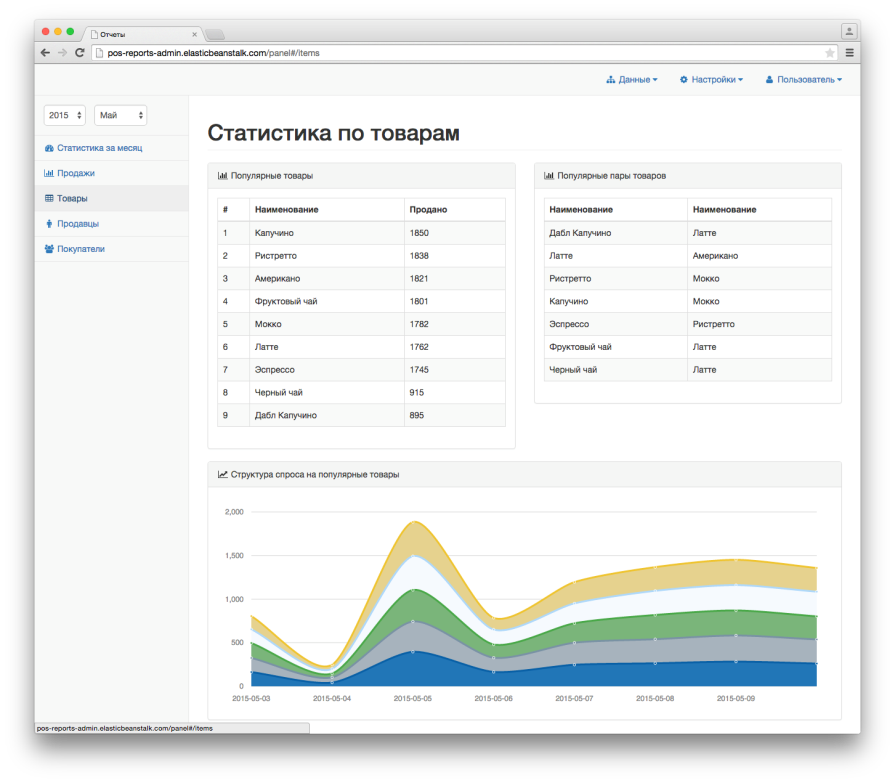
Внизу страницы располагается график изменения выручки заведений в течение месяца.

### Статистика по покупателям

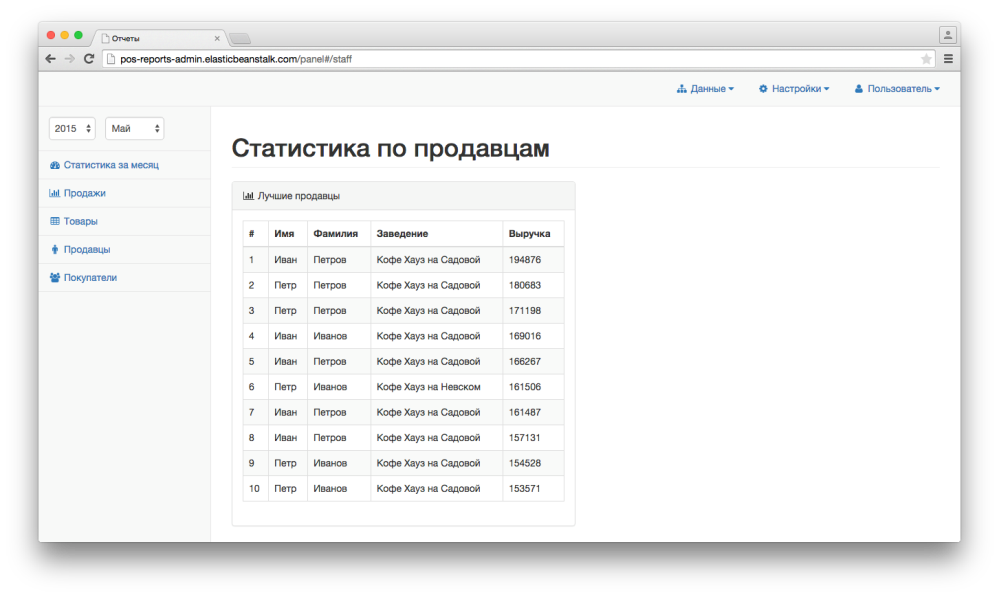
Страница статистики по покупателям содержит рейтинг из 10 покупателей по общей выручке. В правой части страницы находится диаграмма, показывающая размер среднего чека в каждой из точек продаж. При наведении указателя на диаграмму можно увидеть точный размер среднего чека в каждом заведении.

### Статистика по товарам

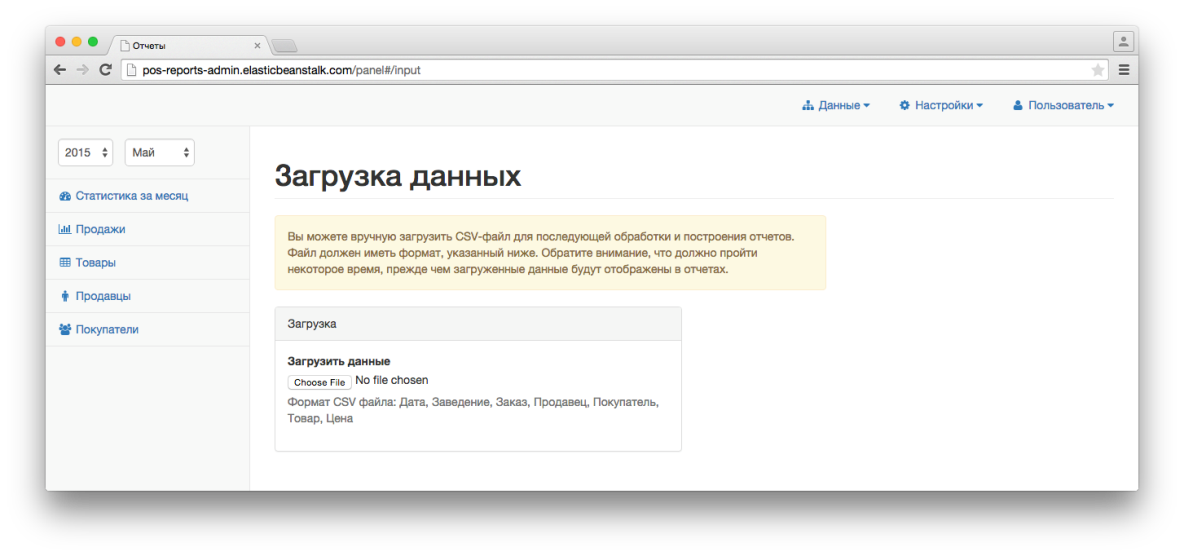
Страница статистика по товарам содержит отчеты о самых популярных товарах и о товарах, которые чаще всего покупают в паре. Кроме того, в нижней части страницы находится график изменения структуры спроса на самые популярные товары за текущий месяц. При наведении указателя на график можно посмотреть точное количество каждого из проданных товаров в любой из дней.



Статистика по продавцам

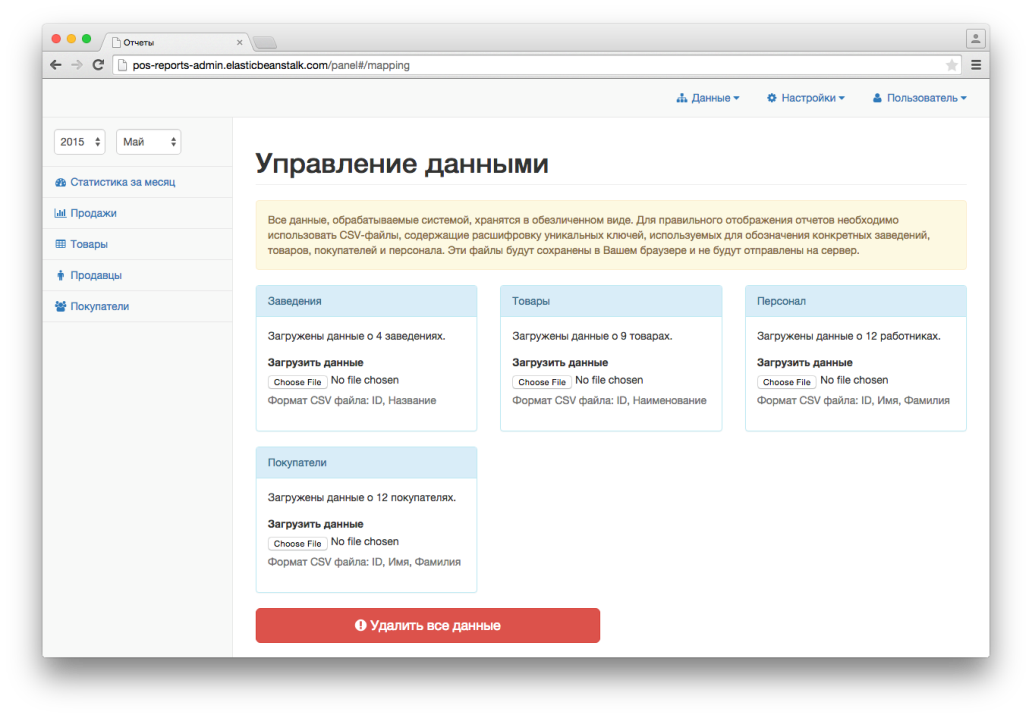
На странице статистики по продавцам выводится рейтинг продавцов среди всех заведений компании, отсортированный по общей выручке.

### Загрузка данных



На странице загрузки данных пользователь может отправить на обработку данные о продажах в формате CSV.

### Управление данными

Страница управления данными позволяет загрузить в приложение CSV-файлы для расшифровки названий заведений, наименований товаров и имен покупателей и продавцов. 

При желании пользователь может удалить все загруженные файлы расшифровки. Вся загруженная информация сохраняется в браузере пользователя и не отправляется на сервер, тем самым сохраняя конфиденциальность данных. Примеры корректных файлов для расшифровки данных приведены ниже.

Данные о заведениях:

location1,"Кафе на Садовой"

location2,"Кафе на Невском"

location3,"Кафе на Площади Восстания"

location4,"Кафе у Дворцовой площади"

Данные о товарах:

item1,"Дабл Капучино"

item2,"Латте"

item3,"Американо"

item4,"Ристретто"

item5,"Мокко"

item6,"Капучино"

item7,"Эспрессо"

item8,"Фруктовый чай"

item9,"Черный чай"

Данные о продавцах:

staff1,"Иван","Иванов"

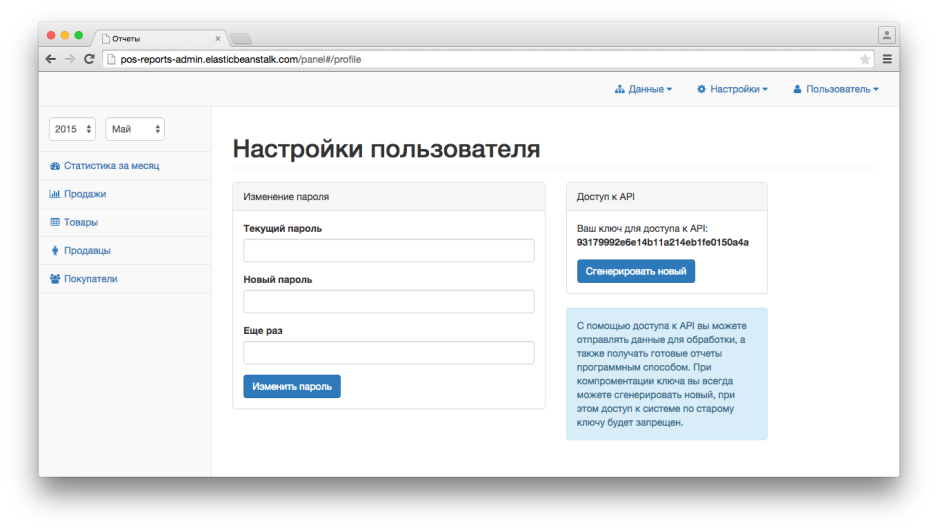
staff2,"Петр","Петров"

Данные о покупателях:

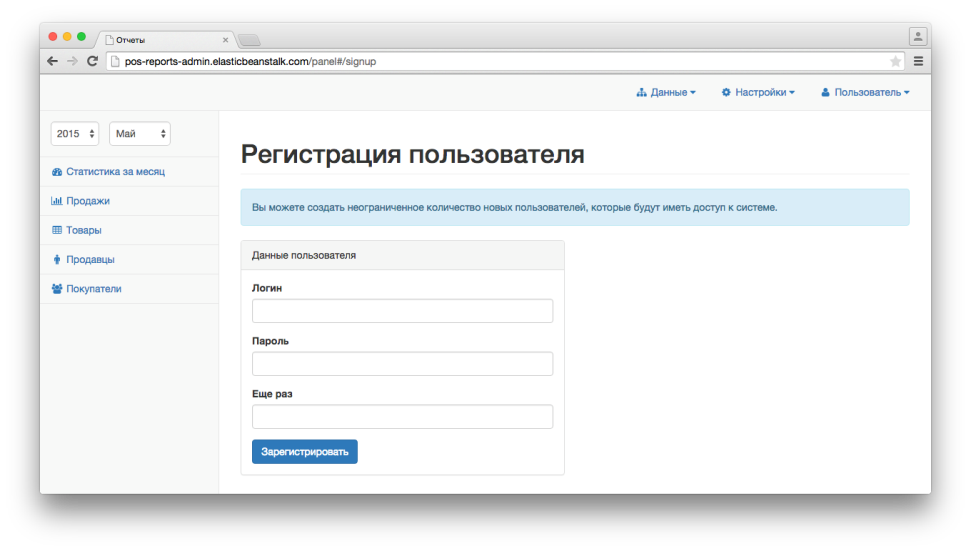
customer1,"Иван","Иванов"

customer2,"Петр","Петров"

Настройки пользователя

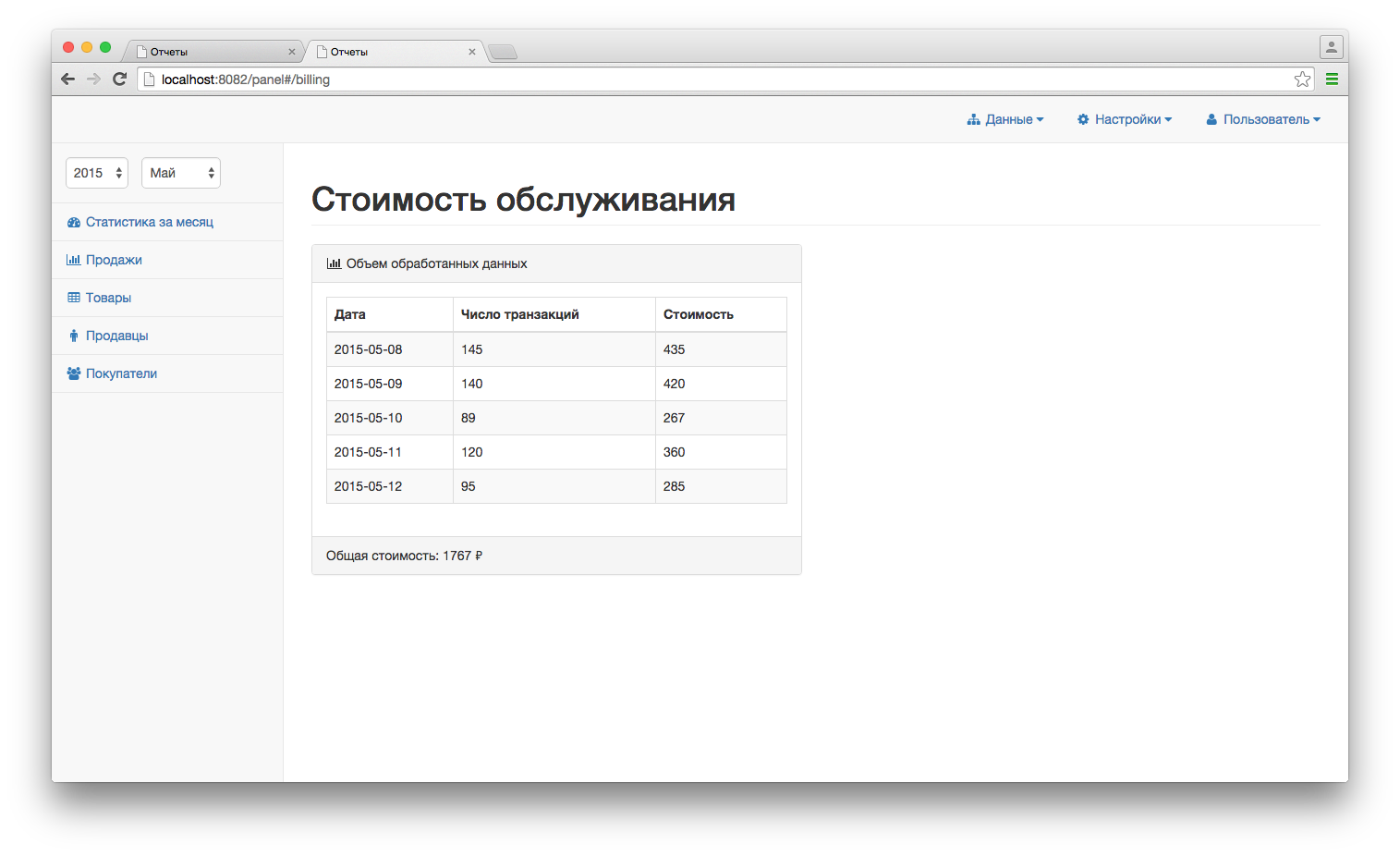


Страница настроек пользователя позволяет сменить пароль для входа в систему, а также сгенерировать новый ключ для доступа к программному интерфейсу системы. После того, как новый ключ сгенерирован, все старые ключи пользователя становятся недействительны.

Регистрация нового пользователя

Страница регистрации нового пользователя позволяет создавать новые аккаунты для доступа к системе. Для регистрации нового пользователя необходимо придумать уникальный логин и пароль не короче 6 символов.

### Стоимость обслуживания



На данном экране пользователь может посмотреть статистику использования системы и стоимость месячного обслуживания. Стоимость обслуживания рассчитывается исходя из объема загруженных в систему данных.

# Вопросы экономики и организации производства

Авав

# Безопасность и санитарно-гигиенические условия труда на рабочем месте пользователя ПЭВМ

вывы

# Заключение

В рамках дипломной работы была разработана автоматизированная система генерации отчетов для предприятий розничной торговли, удовлетворяющая поставленным требованиям.

Система позволяет руководителям предприятий оперативно получать в одном месте аналитическую информацию на основе данных из нескольких точек. Внедрение системы не требует серьезных затрат на построение новой или замену существующей информационной инфраструктуры.

В разработке системы был использованы мощные и современные технологии и свободное ПО, такие как Node.js, Apache Hadoop, облачные вычисления на платформе Amazon Web Service и множество других.

Дальнейшее развитие проекта может быть по следующим направлениям:

* Интеграция с платформами интернет-торговли.
* Использование технологий машинного обучения (платформа Amazon Machine Learning, Azure Machine Learning или собственная разработка) для построения прогнозов и предоставления рекомендаций для руководителей предприятий.
* Сбор дополнительной информации, к примеру анализ поведения пользователей интернет-магазинов.
* Создание новых видов отчетов.

# Список использованных источников

1. Пауэрс Ш., Изучаем Node.js – издательство Питер, 2013.
2. Кантелон М., Хартер М., Головайчук T., Райлих Н., Node.js в действии – издательство Питер, 2014.
3. Закас Н., JavaScript для профессиональных веб-разработчиков – издательство Питер, 2015.
4. Trevor Burnham, Async JavaScript – издательство The Pragmatic Programmers, 2012.
5. Шмитт К., Симпсон К., HTML5. Рецепты программирования – издательство Питер, 2012.
6. Васвани В., MySQL: использование и администрирование – издательство Питер, 2011.
7. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д., Приемы объектно-ориентированного проектирования – издательство Питер, 2015.
8. Маклафлин Б., Поллайс Г., Уэст Д., Объектно-ориентированный анализ и проектирование – издательство Питер, 2013.
9. Фримен Э., Сьерра К., Бейтс Б., Паттерны проектирования – издательство Питер, 2014.
10. Уайт Т., Hadoop. Подробное руководство – издательство Питер, 2013.
11. Пайлон Д., Питмен Н., UML 2 для программистов – издательство Питер, 2012.
12. James Murty, Programming Amazon Web Services – издательство O'Reilly Media, 2012.
13. Jurg van Vliet, Flavia Paganelli, Programming Amazon EC2 - издательство O'Reilly Media, 2011.

# Приложение 1

Полный исходный код приложения доступен по адресу http://github.com/artema/pos-reports

## Конфигурация кластера Apache Hadoop

{

"objects": [

{

"id": "ActivityId",

"schedule": {

"ref": "DefaultSchedule"

},

"name": "POSReports Hive job",

"maximumRetries": "2",

"runsOn": {

"ref": "ReportsClusterId"

},

"postStepCommand": "/home/hadoop/bin/emrfs delete --time 2 -u days",

"dependsOn": {

"ref": "ReportsActivityId"

},

"type": "EmrActivity",

"step": "s3://us-east-1.elasticmapreduce/libs/script-runner/script-runner.jar,s3://us-east-1.elasticmapreduce/libs/hive/hive-script,--base-path,s3://us-east-1.elasticmapreduce/libs/hive/,--hive-versions,latest,--run-hive-script,--args,-f,#{myPOSHIVESCRIPT},-d,INPUT=#{myPOSTRANSFORMEDDATA},-d,OUTPUT=#{myPOSOUTPUTDATA}"

},

{

"resourceRole": "DataPipelineDefaultResourceRole",

"id": "Default",

"failureAndRerunMode": "CASCADE",

"scheduleType": "cron",

"schedule": {

"ref": "DefaultSchedule"

},

"name": "Default",

"pipelineLogUri": "#{myPOSLOGS}",

"role": "DataPipelineDefaultRole"

},

{

"id": "ReportsDataNodeId",

"precondition": {

"ref": "ReportsExistsPreconditionId"

},

"schedule": {

"ref": "DefaultSchedule"

},

"filePath": "#{myPOSINPUTDATA}",

"name": "POSReports S3",

"type": "S3DataNode"

},

{

"terminateAfter": "1 Hour",

"id": "ReportsClusterId",

"amiVersion": "3.2.1",

"keyPair": "#{myPOSKEYPAIR}",

"schedule": {

"ref": "DefaultSchedule"

},

"masterInstanceType": "m1.medium",

"bootstrapAction": "s3://elasticmapreduce/bootstrap-actions/configure-hadoop,-e,fs.s3.consistent=true,-e, fs.s3.consistent.metadata.tableName=EmrFSMetadata, -e, fs.s3.consistent.retryCount=5, -e, fs.s3.consistent.retryPeriodSeconds=10, -e, fs.s3.enableServerSideEncryption=true",

"coreInstanceType": "m1.medium",

"enableDebugging": "true",

"name": "POSReports Cluster",

"coreInstanceCount": "1",

"type": "EmrCluster"

},

{

"id": "ReportsActivityId",

"input": {

"ref": "ReportsDataNodeId"

},

"schedule": {

"ref": "DefaultSchedule"

},

"name": "POSReports Pig job",

"maximumRetries": "2",

"runsOn": {

"ref": "ReportsClusterId"

},

"type": "EmrActivity",

"preStepCommand": "/home/hadoop/bin/emrfs sync #{myPOSINPUTDATA}",

"step": "s3://us-east-1.elasticmapreduce/libs/script-runner/script-runner.jar,s3://us-east-1.elasticmapreduce/libs/pig/pig-script,--base-path,s3://us-east-1.elasticmapreduce/libs/pig/,--pig-versions,latest,--run-pig-script,--args,-f,#{myPOSPIGSCRIPT},-p,INPUT=#{myPOSINPUTDATA},-p,OUTPUT=#{myPOSTRANSFORMEDDATA}"

},

{

"id": "ReportsExistsPreconditionId",

"name": "POSReportsExistsPrecondition",

"maximumRetries": "10",

"role": "DataPipelineDefaultRole",

"s3Prefix": "#{myPOSINPUTDATA}",

"type": "S3PrefixNotEmpty"

},

{

"id": "DefaultSchedule",

"name": "Every 1 day",

"startAt": "FIRST\_ACTIVATION\_DATE\_TIME",

"type": "Schedule",

"period": "1 days"

}

],

"parameters": [

{

"id": "myPOSKEYPAIR",

"description": "EMR Instances Key-Pair",

"type": "String"

},

{

"id": "myPOSOUTPUTDATA",

"description": "S3 Output Data Path",

"type": "AWS::S3::ObjectKey"

},

{

"id": "myPOSTRANSFORMEDDATA",

"description": "S3 Transformed Data Path",

"type": "AWS::S3::ObjectKey"

},

{

"id": "myPOSPIGSCRIPT",

"description": "Apache Pig Script Path",

"type": "AWS::S3::ObjectKey"

},

{

"id": "myPOSHIVESCRIPT",

"description": "Apache Hive Script Path",

"type": "AWS::S3::ObjectKey"

},

{

"id": "myPOSINPUTDATA",

"description": "S3 Input Data Path",

"type": "AWS::S3::ObjectKey"

},

{

"id": "myPOSLOGS",

"description": "S3 Logs Path",

"type": "AWS::S3::ObjectKey"

}

],

"values": {

"myPOSINPUTDATA": "s3://pos-reports/input/#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')}",

"myPOSTRANSFORMEDDATA": "s3://pos-reports/data/#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')}",

"myPOSOUTPUTDATA": "s3://pos-reports/reports/#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')}",

"myPOSKEYPAIR": "eu-west-1",

"myPOSPIGSCRIPT": "s3://pos-reports/scripts/reports.pig",

"myPOSHIVESCRIPT": "s3://pos-reports/scripts/reports.q",

"myPOSLOGS": "s3://pos-reports/logs/"

}

}

## Конфигурация обработчика переноса отчетов

{

"objects": [

{

"id": "Default",

"name": "Default",

"failureAndRerunMode": "CASCADE",

"pipelineLogUri": "#{myLogsS3Loc}",

"resourceRole": "DataPipelineDefaultResourceRole",

"role": "DataPipelineDefaultRole",

"schedule": {

"ref": "DefaultSchedule"

},

"scheduleType": "cron"

},

{

"id": "Ec2Instance",

"name": "Ec2Instance",

"actionOnTaskFailure": "terminate",

"instanceType": "t1.micro",

"securityGroups": "#{myEc2RdsSecurityGrps}",

"terminateAfter": "1 Hours",

"type": "Ec2Resource"

},

{

"id": "CSVFormat",

"name": "CSVFormat",

"type": "CSV"

},

{

"id": "DefaultSchedule",

"name": "Every 1 day",

"period": "1 days",

"startAt": "FIRST\_ACTIVATION\_DATE\_TIME",

"type": "Schedule"

},

{

"id": "ActivityCustomerAverage",

"name": "ActivityCustomerAverage",

"input": {

"ref": "InputCustomerAverage"

},

"output": {

"ref": "DestinationCustomerAverage"

},

"runsOn": {

"ref": "Ec2Instance"

},

"type": "CopyActivity"

},

{

"id": "InputCustomerAverage",

"name": "InputCustomerAverage",

"directoryPath": "#{myInputS3Loc}/report-customer-average",

"precondition": {

"ref": "PreconditionInputCustomerAverage"

},

"dataFormat": {

"ref": "CSVFormat"

},

"type": "S3DataNode"

},

{

"id": "PreconditionInputCustomerAverage",

"name": "PreconditionInputCustomerAverage",

"s3Prefix": "#{myInputS3Loc}/report-customer-average",

"maximumRetries": "5",

"role": "DataPipelineDefaultRole",

"type": "S3PrefixNotEmpty"

},

{

"id": "DestinationCustomerAverage",

"name": "DestinationCustomerAverage",

"table": "report-customer-average",

"insertQuery": "INSERT INTO `#{table}` (company\_id, location\_key, date, value) VALUES(?, ?, '#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')} 00:00:00', ?) ON DUPLICATE KEY UPDATE value = values(value);",

"selectQuery": "select \* from #{table}",

"\*password": "#{\*myRDSPassword}",

"connectionString": "#{myRDSConnectStr}",

"username": "#{myRDSUsername}",

"type": "MySqlDataNode"

},

{

"id": "ActivityCustomerTop",

"name": "ActivityCustomerTop",

"input": {

"ref": "InputCustomerTop"

},

"output": {

"ref": "DestinationCustomerTop"

},

"runsOn": {

"ref": "Ec2Instance"

},

"type": "CopyActivity"

},

{

"id": "InputCustomerTop",

"name": "InputCustomerTop",

"directoryPath": "#{myInputS3Loc}/report-customer-top",

"precondition": {

"ref": "PreconditionInputCustomerTop"

},

"dataFormat": {

"ref": "CSVFormat"

},

"type": "S3DataNode"

},

{

"id": "PreconditionInputCustomerTop",

"name": "PreconditionInputCustomerTop",

"s3Prefix": "#{myInputS3Loc}/report-customer-top",

"maximumRetries": "5",

"role": "DataPipelineDefaultRole",

"type": "S3PrefixNotEmpty"

},

{

"id": "DestinationCustomerTop",

"name": "DestinationCustomerTop",

"table": "report-customer-top",

"insertQuery": "INSERT INTO `#{table}` (company\_id, date, customer\_key, value) VALUES(?, '#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')} 00:00:00', ?, ?) ON DUPLICATE KEY UPDATE value = values(value);",

"selectQuery": "select \* from #{table}",

"\*password": "#{\*myRDSPassword}",

"connectionString": "#{myRDSConnectStr}",

"username": "#{myRDSUsername}",

"type": "MySqlDataNode"

},

{

"id": "ActivityItemPair",

"name": "ActivityItemPair",

"input": {

"ref": "InputItemPair"

},

"output": {

"ref": "DestinationItemPair"

},

"runsOn": {

"ref": "Ec2Instance"

},

"type": "CopyActivity"

},

{

"id": "InputItemPair",

"name": "InputItemPair",

"directoryPath": "#{myInputS3Loc}/report-item-pair",

"precondition": {

"ref": "PreconditionInputItemPair"

},

"dataFormat": {

"ref": "CSVFormat"

},

"type": "S3DataNode"

},

{

"id": "PreconditionInputItemPair",

"name": "PreconditionInputItemPair",

"s3Prefix": "#{myInputS3Loc}/report-item-pair",

"maximumRetries": "5",

"role": "DataPipelineDefaultRole",

"type": "S3PrefixNotEmpty"

},

{

"id": "DestinationItemPair",

"name": "DestinationItemPair",

"table": "report-item-pair",

"insertQuery": "INSERT INTO `#{table}` (company\_id, location\_key, date, item1\_key, item2\_key) VALUES(?, ?, '#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')} 00:00:00', ?, ?) ON DUPLICATE KEY UPDATE item1\_key = values(item1\_key);",

"selectQuery": "select \* from #{table}",

"\*password": "#{\*myRDSPassword}",

"connectionString": "#{myRDSConnectStr}",

"username": "#{myRDSUsername}",

"type": "MySqlDataNode"

},

{

"id": "ActivityItemPopular",

"name": "ActivityItemPopular",

"input": {

"ref": "InputItemPopular"

},

"output": {

"ref": "DestinationItemPopular"

},

"runsOn": {

"ref": "Ec2Instance"

},

"type": "CopyActivity"

},

{

"id": "InputItemPopular",

"name": "InputItemPopular",

"directoryPath": "#{myInputS3Loc}/report-item-popular",

"precondition": {

"ref": "PreconditionInputItemPopular"

},

"dataFormat": {

"ref": "CSVFormat"

},

"type": "S3DataNode"

},

{

"id": "PreconditionInputItemPopular",

"name": "PreconditionInputItemPopular",

"s3Prefix": "#{myInputS3Loc}/report-item-popular",

"maximumRetries": "5",

"role": "DataPipelineDefaultRole",

"type": "S3PrefixNotEmpty"

},

{

"id": "DestinationItemPopular",

"name": "DestinationItemPopular",

"table": "report-item-popular",

"insertQuery": "INSERT INTO `#{table}` (company\_id, location\_key, date, item\_key, value) VALUES(?, ?, '#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')} 00:00:00', ?, ?) ON DUPLICATE KEY UPDATE value = values(value);",

"selectQuery": "select \* from #{table}",

"\*password": "#{\*myRDSPassword}",

"connectionString": "#{myRDSConnectStr}",

"username": "#{myRDSUsername}",

"type": "MySqlDataNode"

},

{

"id": "ActivityItemTop",

"name": "ActivityItemTop",

"input": {

"ref": "InputItemTop"

},

"output": {

"ref": "DestinationItemTop"

},

"runsOn": {

"ref": "Ec2Instance"

},

"type": "CopyActivity"

},

{

"id": "InputItemTop",

"name": "InputItemTop",

"directoryPath": "#{myInputS3Loc}/report-item-top",

"precondition": {

"ref": "PreconditionInputItemTop"

},

"dataFormat": {

"ref": "CSVFormat"

},

"type": "S3DataNode"

},

{

"id": "PreconditionInputItemTop",

"name": "PreconditionInputItemTop",

"s3Prefix": "#{myInputS3Loc}/report-item-top",

"maximumRetries": "5",

"role": "DataPipelineDefaultRole",

"type": "S3PrefixNotEmpty"

},

{

"id": "DestinationItemTop",

"name": "DestinationItemTop",

"table": "report-item-top",

"insertQuery": "INSERT INTO `#{table}` (company\_id, location\_key, date, item\_key, value) VALUES(?, ?, '#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')} 00:00:00', ?, ?) ON DUPLICATE KEY UPDATE value = values(value);",

"selectQuery": "select \* from #{table}",

"\*password": "#{\*myRDSPassword}",

"connectionString": "#{myRDSConnectStr}",

"username": "#{myRDSUsername}",

"type": "MySqlDataNode"

},

{

"id": "ActivitySalesTotal",

"name": "ActivitySalesTotal",

"input": {

"ref": "InputSalesTotal"

},

"output": {

"ref": "DestinationSalesTotal"

},

"runsOn": {

"ref": "Ec2Instance"

},

"type": "CopyActivity"

},

{

"id": "InputSalesTotal",

"name": "InputSalesTotal",

"directoryPath": "#{myInputS3Loc}/report-sales-total",

"precondition": {

"ref": "PreconditionInputSalesTotal"

},

"dataFormat": {

"ref": "CSVFormat"

},

"type": "S3DataNode"

},

{

"id": "PreconditionInputSalesTotal",

"name": "PreconditionInputSalesTotal",

"s3Prefix": "#{myInputS3Loc}/report-sales-total",

"maximumRetries": "5",

"role": "DataPipelineDefaultRole",

"type": "S3PrefixNotEmpty"

},

{

"id": "DestinationSalesTotal",

"name": "DestinationSalesTotal",

"table": "report-sales-total",

"insertQuery": "INSERT INTO `#{table}` (company\_id, location\_key, date, amount) VALUES(?, ?, '#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')} 00:00:00', ?) ON DUPLICATE KEY UPDATE amount = values(amount);",

"selectQuery": "select \* from #{table}",

"\*password": "#{\*myRDSPassword}",

"connectionString": "#{myRDSConnectStr}",

"type": "MySqlDataNode",

"username": "#{myRDSUsername}"

},

{

"id": "ActivityStaffTop",

"name": "ActivityStaffTop",

"input": {

"ref": "InputStaffTop"

},

"output": {

"ref": "DestinationStaffTop"

},

"runsOn": {

"ref": "Ec2Instance"

},

"type": "CopyActivity"

},

{

"id": "InputStaffTop",

"name": "InputStaffTop",

"directoryPath": "#{myInputS3Loc}/report-staff-top",

"precondition": {

"ref": "PreconditionInputStaffTop"

},

"dataFormat": {

"ref": "CSVFormat"

},

"type": "S3DataNode"

},

{

"id": "PreconditionInputStaffTop",

"name": "PreconditionInputStaffTop",

"s3Prefix": "#{myInputS3Loc}/report-staff-top",

"maximumRetries": "5",

"role": "DataPipelineDefaultRole",

"type": "S3PrefixNotEmpty"

},

{

"id": "DestinationStaffTop",

"name": "DestinationStaffTop",

"table": "report-staff-top",

"insertQuery": "INSERT INTO `#{table}` (company\_id, location\_key, date, staff\_key, value) VALUES(?, ?, '#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')} 00:00:00', ?, ?) ON DUPLICATE KEY UPDATE value = values(value);",

"selectQuery": "select \* from #{table}",

"\*password": "#{\*myRDSPassword}",

"connectionString": "#{myRDSConnectStr}",

"type": "MySqlDataNode",

"username": "#{myRDSUsername}"

}

],

"parameters": [

{

"description": "RDS MySQL username",

"id": "myRDSUsername",

"type": "String"

},

{

"description": "RDS MySQL password",

"id": "\*myRDSPassword",

"type": "String"

},

{

"description": "RDS MySQL security group(s)",

"helpText": "The names of one or more EC2 security groups that have access to the RDS MySQL cluster.",

"id": "myEc2RdsSecurityGrps",

"isArray": "true",

"optional": "true",

"type": "String",

"watermark": "security group name"

},

{

"description": "RDS MySQL connection string",

"id": "myRDSConnectStr",

"type": "String"

},

{

"description": "Input S3 file path",

"id": "myInputS3Loc",

"type": "AWS::S3::ObjectKey"

},

{

"description": "Logs S3 file path",

"id": "myLogsS3Loc",

"type": "AWS::S3::ObjectKey"

}

],

"values": {

"myInputS3Loc": "s3://pos-reports/reports/#{format(minusDays(@scheduledStartTime,1),'YYYY-MM-dd')}",

"myRDSConnectStr": "jdbc:mysql://pos-db.cddk2mcy7t4j.eu-west-1.rds.amazonaws.com:3306/pos",

"myRDSUsername": "admin",

"\*myRDSPassword": "",

"myLogsS3Loc": "s3://pos-reports/logs/"

}

}

## Lambda-функция для приема данных из Kinesis

var AWS = require('aws-sdk'),

s3 = new AWS.S3(),

async = require('async'),

stringify = require('csv-stringify'),

moment = require('moment'),

uuid = require('node-uuid');

function saveData(type, items) {

var payloads = items.reduce(function(value, item) {

if (!item) {

return value;

}

var payload = value[item.session.id];

if (!payload) {

value[item.session.id] = payload = {

session: item.session,

data: []

};

}

payload.data.push(item.data);

return value;

}, {});

return Object.keys(payloads).map(function(key) {

var payload = payloads[key];

return function(next) {

stringify(payload.data, function(err, output){

if (err) {

return next(err);

}

s3.putObject({

Bucket: 'pos-reports',

ContentType: 'text/csv',

StorageClass: 'REDUCED\_REDUNDANCY',

Key: 'input/' + moment(payload.session.date).format('YYYY-MM-DD') + '/' + type + '/' + payload.session.id + '.csv',

Body: output

}, next);

});

};

});

}

function getSession(sessions, time) {

if (!sessions || sessions.length === 0) {

return null;

}

time = moment(time);

var session = null,

keys = Object.keys(sessions);

if (keys.length > 0) {

session = keys.reduce(function(value, id) {

var s = sessions[id];

if (time.isSame(moment(s.date), 'day')) {

return s;

}

return value;

}, null);

}

if (!session) {

session = {

id: uuid.v4(),

date: time.format('YYYY-MM-DD')

};

sessions[session.id] = session;

}

return session;

}

module.exports = function(entry) {

var tasks = [],

sessions = {};

if (entry.sales) {

console.log('Processing ' + entry.sales.length + ' sale records...');

tasks = tasks.concat(saveData('sales', entry.sales.map(function(payload) {

return {

session: getSession(sessions, payload.time),

data: [

entry.company,

payload.location,

payload.time,

payload.check,

payload.staff,

payload.customer,

payload.item,

payload.price

]

};

})));

}

return function(next) {

console.log('Running ' + tasks.length + ' tasks...');

async.parallel(tasks, next);

};

};

## Программный интерфейс для доступа к отчетам

var moment = require('moment');

var isAuthenticated = function(req, res, next) {

if (req.isAuthenticated()) {

return next();

}

res.status(401).end();

};

var onData = function(res) {

return function(data) {

res.json(data);

};

};

var onError = function(res) {

return function(e) {

console.error(e);

res.status(500).end();

};

};

module.exports = function(app) {

function createReport(name) {

app.get('/api/report/' + name, isAuthenticated, function(req, res) {

var reports = app.get('reports'),

users = app.get('users');

users.getCompany(req.user).then(function(company) {

reports[name.replace('-', '\_')]({

date: req.query.date,

days: req.query.days || 1,

company: company

}).then(onData(res), onError(res));

}, onError(res));

});

}

createReport('billing');

createReport('customer-average');

createReport('customer-top');

createReport('item-pair');

createReport('item-popular');

createReport('item-top');

createReport('sales-total');

createReport('staff-top');

app.post('/api/report', isAuthenticated, function(req, res) {

var reports = app.get('reports'),

users = app.get('users');

users.getCompany(req.user).then(function(company) {

reports.uploadData(company, req.body).then(onData(res), onError(res));

}, onError(res));

});

};

var AWS = require('aws-sdk'),

moment = require('moment');

function formatDate(date, days) {

var result = moment(date);

if (days > 0) {

result.add(days, 'days');

}

return result.format('YYYY-MM-DD 00:00:00');

}

function formatDateShort(date, days) {

var result = moment(date);

if (days > 0) {

result.add(days, 'days');

}

return result.format('YYYY-MM-DD');

}

function Reports(pool) {

this.\_pool = pool;

}

Reports.prototype.customer\_average = function(query) {

return this.\_report(

'SELECT `location\_key`, DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `value` FROM `report-customer-average` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDate(query.date), formatDate(query.date, query.days) ]

);

};

Reports.prototype.customer\_top = function(query) {

return this.\_report(

'SELECT `customer\_key`, DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `value` FROM `report-customer-top` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDate(query.date), formatDate(query.date, query.days) ]

);

};

Reports.prototype.item\_pair = function(query) {

return this.\_report(

'SELECT `location\_key`, DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `item1\_key`, `item2\_key` FROM `report-item-pair` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDate(query.date), formatDate(query.date, query.days) ]

);

};

Reports.prototype.item\_popular = function(query) {

return this.\_report(

'SELECT `location\_key`, DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `item\_key`, `value` FROM `report-item-popular` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDate(query.date), formatDate(query.date, query.days) ]

);

};

Reports.prototype.item\_top = function(query) {

return this.\_report(

'SELECT `location\_key`, DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `item\_key`, `value` FROM `report-item-top` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDate(query.date), formatDate(query.date, query.days) ]

);

};

Reports.prototype.sales\_total = function(query) {

return this.\_report(

'SELECT `location\_key`, DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `amount` as `value` FROM `report-sales-total` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDate(query.date), formatDate(query.date, query.days) ]

);

};

Reports.prototype.staff\_top = function(query) {

return this.\_report(

'SELECT `location\_key`, DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `staff\_key`, `value` FROM `report-staff-top` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDate(query.date), formatDate(query.date, query.days) ]

);

};

Reports.prototype.billing = function(query) {

return this.\_report(

'SELECT DATE\_FORMAT(`date`, "%Y-%m-%d") `date`, `sales` FROM `billing` WHERE `company\_id` = ? AND `date` >= ? AND `date` < ? ORDER BY `date` ASC',

[ query.company.id, formatDateShort(query.date), formatDateShort(query.date, query.days) ]

);

};

Reports.prototype.uploadData = function(company, payload) {

var self = this;

return new Promise(function(resolve, reject) {

var kinesis = new AWS.Kinesis();

kinesis.putRecord({

Data: JSON.stringify({

company: company.id,

sales: payload.sales || []

}),

PartitionKey: 'web',

StreamName: 'pos-input'

}, function(e) {

if (e) {

return reject(e);

}

resolve();

});

});

};

Reports.prototype.\_report = function(query, values) {

var self = this;

return new Promise(function(resolve, reject) {

self.\_pool.getConnection(function(err, connection) {

if (err) {

return reject(err);

}

connection.query(query, values, function(err, rows) {

if (err) {

return reject(err);

}

connection.release();

resolve(rows || []);

});

});

});

};

module.exports = function(pool) {

return new Reports(pool);

};

## Тестовый клиент

var AWS = require('aws-sdk'),

moment = require('moment'),

uuid = require('node-uuid');

if (process.env.NODE\_ENV !== 'production') {

AWS.config.credentials = new AWS.SharedIniFileCredentials({ profile: 'me' });

AWS.config.update({ region: 'eu-west-1' });

}

function run(day, company) {

function getRandom(array) {

return array[Math.round(Math.random() \* (array.length - 1))];

}

var items = [

{ name: 'item1', price: 100.5 },

{ name: 'item2', price: 150 },

{ name: 'item3', price: 290 },

{ name: 'item4', price: 150.75 },

{ name: 'item5', price: 230.5 },

{ name: 'item6', price: 310 },

{ name: 'item7', price: 200.5 },

{ name: 'item8', price: 147 },

{ name: 'item9', price: 240 }

];

var customers = [

"customer1",

"customer2",

"customer3",

"customer4",

"customer5",

"customer6",

"customer7",

"customer8",

"customer9",

"customer10",

"customer11",

"customer12"

];

var staff = [

"staff1",

"staff2",

"staff3",

"staff4",

"staff5",

"staff6",

"staff7",

"staff8",

"staff9",

"staff10",

"staff11",

"staff12"

];

var locations = [

"location1",

"location2",

"location3",

"location4"

];

var kinesis = new AWS.Kinesis();

for (var v = 0; v < 10; v++) {

var sales = [];

for (var i = 0; i < 100; i++) {

var checkId = uuid.v4(),

checkLocation = getRandom(locations),

checkStaff = getRandom(staff),

checkCustomer = getRandom(customers),

checkDate = moment(day)

.add(Math.floor(24 \* Math.random()), 'hours')

.add(Math.floor(60 \* Math.random()), 'minutes')

.add(Math.floor(60 \* Math.random()), 'seconds');

if (Math.random() < 0.7) {

checkCustomer = null;

}

for (var j = 0; j < Math.random() \* 4; j++) {

var item = getRandom(items);

var check = {

time: checkDate.format(),

location: checkLocation,

check: checkId,

staff: checkStaff,

item: item.name,

price: item.price,

customer: checkCustomer

};

sales.push(check);

}

}

kinesis.putRecord({

Data: JSON.stringify({

company: company,

sales: sales

}),

PartitionKey: 'test',

StreamName: 'pos-input'

}, function(err) {

if (err) {

console.error(err, err.stack);

}

else {

console.log('Complete');

}

});

}

}

run('2015-05-15', 1);

# Приложение 2

## Скрипт создания БД

CREATE TABLE `billing` (

`company\_id` int(11) NOT NULL,

`date` date NOT NULL,

`sales` int(11) NOT NULL,

PRIMARY KEY (`company\_id`,`date`),

CONSTRAINT `fk\_billing\_company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `company` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=latin1;

CREATE TABLE `report-customer-average` (

`company\_id` int(11) NOT NULL,

`location\_key` varchar(255) NOT NULL,

`date` datetime NOT NULL,

`value` decimal(10,4) DEFAULT NULL,

PRIMARY KEY (`company\_id`,`location\_key`,`date`),

CONSTRAINT `fk-report-customer-average-company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `report-customer-top` (

`company\_id` int(11) NOT NULL,

`date` datetime NOT NULL,

`customer\_key` varchar(255) NOT NULL,

`value` decimal(10,4) DEFAULT NULL,

PRIMARY KEY (`company\_id`,`date`,`customer\_key`),

CONSTRAINT `fk-report-customer-top-company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `report-item-pair` (

`company\_id` int(11) NOT NULL,

`location\_key` varchar(255) NOT NULL,

`date` datetime NOT NULL,

`item1\_key` varchar(255) NOT NULL,

`item2\_key` varchar(255) NOT NULL,

PRIMARY KEY (`company\_id`,`location\_key`,`date`,`item1\_key`,`item2\_key`),

KEY `ix-report-item-pair` (`date`,`company\_id`) USING BTREE,

CONSTRAINT `report-item-pair-company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `report-item-popular` (

`company\_id` int(11) NOT NULL,

`location\_key` varchar(255) NOT NULL,

`date` datetime NOT NULL,

`item\_key` varchar(255) NOT NULL,

`value` int(255) DEFAULT NULL,

PRIMARY KEY (`company\_id`,`location\_key`,`date`,`item\_key`),

CONSTRAINT `fk-report-item-popular-company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `report-item-top` (

`company\_id` int(11) NOT NULL,

`location\_key` varchar(255) NOT NULL,

`date` datetime NOT NULL,

`item\_key` varchar(255) NOT NULL,

`value` decimal(10,4) DEFAULT NULL,

PRIMARY KEY (`company\_id`,`location\_key`,`date`,`item\_key`),

CONSTRAINT `fk-report-item-top-company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `report-sales-total` (

`company\_id` int(11) NOT NULL,

`location\_key` varchar(255) NOT NULL,

`date` datetime NOT NULL,

`amount` decimal(10,4) DEFAULT NULL,

PRIMARY KEY (`company\_id`,`location\_key`,`date`),

CONSTRAINT `report-sale-total-company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `report-staff-top` (

`company\_id` int(11) NOT NULL,

`location\_key` varchar(255) NOT NULL,

`date` datetime NOT NULL,

`staff\_key` varchar(255) NOT NULL,

`value` decimal(10,4) DEFAULT NULL,

PRIMARY KEY (`company\_id`,`location\_key`,`date`,`staff\_key`),

CONSTRAINT `report-staff-top-company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `user` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`login` varchar(255) NOT NULL,

`password` varchar(255) NOT NULL,

`company\_id` int(11) NOT NULL,

`authtoken` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `ix\_user\_company` (`company\_id`),

KEY `ix\_user\_username` (`login`),

CONSTRAINT `fk\_user\_company` FOREIGN KEY (`company\_id`) REFERENCES `company` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=latin1;

## Скрипт шага Map в Apache Hadoop

REGISTER file:/home/hadoop/pig/lib/piggybank.jar

DEFINE CSVExcelStorage org.apache.pig.piggybank.storage.CSVExcelStorage;

sales =

LOAD '$INPUT/sales' USING CSVExcelStorage(',', 'NO\_MULTILINE', 'UNIX', 'SKIP\_INPUT\_HEADER', 'SKIP\_OUTPUT\_HEADER')

AS (

company: int,

location: chararray,

time: datetime,

check: chararray,

staff: chararray,

customer: chararray,

item: chararray,

price: float

);

sales\_by\_check = GROUP sales BY check;

checks\_by\_location = FOREACH sales\_by\_check GENERATE

group as check,

BagToTuple(sales.company).$0 as company,

BagToTuple(sales.location).$0 as location,

BagToTuple(sales.customer).$0 as customer,

BagToTuple(sales.staff).$0 as staff;

checks\_totals = FOREACH sales\_by\_check GENERATE

group as check,

MAX(sales.time) as time,

SUM(sales.price) as total;

items\_by\_check = FOREACH sales GENERATE

check,

item,

price;

STORE checks\_by\_location INTO '${OUTPUT}/checks\_by\_location';

STORE checks\_totals INTO '${OUTPUT}/checks\_totals';

STORE items\_by\_check INTO '${OUTPUT}/items\_by\_check';

## Скрипт шага Reduce в Apache Hadoop

DROP TABLE IF EXISTS checks\_by\_location;

CREATE EXTERNAL TABLE checks\_by\_location

(

check string, company int, location string, customer string, staff string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '${INPUT}/checks\_by\_location/';

DROP TABLE IF EXISTS checks\_totals;

CREATE EXTERNAL TABLE checks\_totals

(

check string, time string, total float

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '${INPUT}/checks\_totals/';

DROP TABLE IF EXISTS items\_by\_check;

CREATE EXTERNAL TABLE items\_by\_check

(

check string, item string, price float

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '${INPUT}/items\_by\_check/';

DROP TABLE IF EXISTS report\_customer\_average;

CREATE TABLE report\_customer\_average

(

company int,

location string,

value float

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE

LOCATION '${OUTPUT}/report-customer-average';

DROP TABLE IF EXISTS report\_customer\_top;

CREATE TABLE report\_customer\_top

(

company int,

customer string,

value float

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE

LOCATION '${OUTPUT}/report-customer-top';

DROP TABLE IF EXISTS report\_item\_pair;

CREATE TABLE report\_item\_pair

(

company int,

location string,

item1 string,

item2 string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE

LOCATION '${OUTPUT}/report-item-pair';

DROP TABLE IF EXISTS report\_item\_popular;

CREATE TABLE report\_item\_popular

(

company int,

location string,

item string,

count int

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE

LOCATION '${OUTPUT}/report-item-popular';

DROP TABLE IF EXISTS report\_item\_top;

CREATE TABLE report\_item\_top

(

company int,

location string,

item string,

count int

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE

LOCATION '${OUTPUT}/report-item-top';

DROP TABLE IF EXISTS report\_sales\_total;

CREATE TABLE report\_sales\_total

(

company int,

location string,

total float

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE

LOCATION '${OUTPUT}/report-sales-total';

DROP TABLE IF EXISTS report\_staff\_top;

CREATE TABLE report\_staff\_top

(

company int,

location string,

staff string,

total float

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE

LOCATION '${OUTPUT}/report-staff-top';

INSERT OVERWRITE TABLE report\_customer\_average

SELECT checks\_by\_location.company, checks\_by\_location.location, AVG(checks\_totals.total)

FROM checks\_totals

JOIN checks\_by\_location ON checks\_totals.check = checks\_by\_location.check

GROUP BY checks\_by\_location.location, checks\_by\_location.company;

INSERT OVERWRITE TABLE report\_customer\_top

SELECT checks\_by\_location.company, checks\_by\_location.customer, SUM(checks\_totals.total)

FROM checks\_totals

JOIN checks\_by\_location ON checks\_totals.check = checks\_by\_location.check

WHERE checks\_by\_location.customer != ''

GROUP BY checks\_by\_location.location, checks\_by\_location.company, checks\_by\_location.customer;

INSERT OVERWRITE TABLE report\_item\_pair

SELECT l.company as company, l.location as location, i1.item as item1, i2.item as item2

FROM items\_by\_check i1

JOIN items\_by\_check i2 ON i1.check = i2.check

JOIN checks\_by\_location l ON i1.check = l.check

WHERE i1.item != i2.item

GROUP BY l.company, l.location, i1.item, i2.item

HAVING COUNT(i1.item) > 1;

INSERT OVERWRITE TABLE report\_item\_popular

SELECT checks\_by\_location.company, checks\_by\_location.location, items\_by\_check.item, COUNT(items\_by\_check.item)

FROM items\_by\_check

JOIN checks\_by\_location ON items\_by\_check.check = checks\_by\_location.check

GROUP BY checks\_by\_location.company, checks\_by\_location.location, items\_by\_check.item

HAVING COUNT(items\_by\_check.item) > 1;

INSERT OVERWRITE TABLE report\_item\_top

SELECT checks\_by\_location.company, checks\_by\_location.location, items\_by\_check.item, SUM(items\_by\_check.price)

FROM items\_by\_check

JOIN checks\_by\_location ON items\_by\_check.check = checks\_by\_location.check

GROUP BY checks\_by\_location.company, checks\_by\_location.location, items\_by\_check.item;

INSERT OVERWRITE TABLE report\_staff\_top

SELECT checks\_by\_location.company, checks\_by\_location.location, checks\_by\_location.staff, SUM(checks\_totals.total)

FROM checks\_totals

JOIN checks\_by\_location ON checks\_totals.check = checks\_by\_location.check

GROUP BY checks\_by\_location.company, checks\_by\_location.location, checks\_by\_location.staff;

INSERT OVERWRITE TABLE report\_sales\_total

SELECT checks\_by\_location.company, checks\_by\_location.location, SUM(checks\_totals.total)

FROM checks\_totals

JOIN checks\_by\_location ON checks\_totals.check = checks\_by\_location.check

GROUP BY checks\_by\_location.location, checks\_by\_location.company;

# Приложение 3

## Техническое задание

### Введение

Система автоматизации анализа данных о продажах и построения отчетов с учетом специфики розничной торговли.

### Требования к функциональным характеристикам

Система должна представлять собой комплекс программных средств для решения поставленных задач и соответствовать следующим требованиям к функциональности:

* Предоставление программного интерфейса для загрузки данных о продажах в систему.
* Автоматическое построение отчетов по полученным данным с интервалом не реже одного раза в сутки.
* Рассчет показателей на основе данных о продажах: общая выручка за месяц, средняя выручка с покупателя, список популярных товаров, продавцы, реализовавшие больше всего товаров, товары, приносящие наибольшую выручку, покупатели, приносящие наибольшую выручку.
* Построение отчетов и графиков на основе агрегированных показателей: общая выручка с нескольких точек продаж, сравнительная выручка по точкам продаж, изменение уровня продаж, популярные пары товаров, структура продаж.
* Возможность расшифровки разыменованных данных в приложении на машине пользователя.

### Условия эксплуатации

Система должны быть реализована на языке программирования высокого уровня и предоставлять доступ через веб-интерфейс.

Сервер:

* Облачная платформа Amazon Web Services или Microsoft Azure.
* Не более 5 виртуальных серверов с объемом оперативной памяти до 4 гигабайт.

Клиент:

* Персональный компьютер с веб-браузером не старше Internet Explorer 9, Firefox 30 или Chrome 37, либо планшетный компьютер Apple iPad 3 или более новый, Asus Nexus 7 или более новый.

### Требования к информационной и программной совместимости

Клиентская часть системы должны быть разработана на язяке JavaScript.

Серверная часть должна быть разработана на платформе Node.js или аналогичной. В качестве СУБД необходимо использовать MySQL или PostgreSQL.

Операционной системой для сервера может служить Windows Server 2012 или более новая версия, либо дистрибутив Linux не старше 3 лет.

# Приложение 4

## Контрольные примеры

### Загрузка данных в систему

Используя тестовый клиент из приложения 1, можно сгенерировать и отправить на программный интерфейс случайные данные о продажах. Пример полученных данных:

{

"sales": [

{

"check": "e805838b-cd10-494f-aa37-ffaaf463d999",

"customer": null,

"item": "item6",

"location": "location3",

"price": 310,

"staff": "staff5",

"time": "2015-05-15T21:39:27+03:00"

},

{

"check": "e805838b-cd10-494f-aa37-ffaaf463d999",

"customer": null,

"item": "item7",

"location": "location3",

"price": 200.5,

"staff": "staff5",

"time": "2015-05-15T21:39:27+03:00"

},

{

"check": "84ba5edb-b9d8-443e-975d-7ed7b09842aa",

"customer": "customer3",

"item": "item1",

"location": "location1",

"price": 100.5,

"staff": "staff7",

"time": "2015-05-15T02:58:12+03:00"

}

]

}

### Расшифровка разыменованных данных

Для проверки работы функции расшифровки данных в приложении для просмотра отчетов можно воспользоваться следующими файлами:

locations.csv:

location1,"Кофе Хауз на Садовой"

location2,"Кофе Хауз на Невском"

location3,"Кофе Хауз на Площади Восстания"

location4,"Кофе Хауз у Дворцовой площади"

items.csv:

item1,"Дабл Капучино"

item2,"Латте"

item3,"Американо"

item4,"Ристретто"

item5,"Мокко"

item6,"Капучино"

item7,"Эспрессо"

item8,"Фруктовый чай"

item9,"Черный чай"

customers.csv:

customer1,"Иван","Иванов"

customer2,"Петр","Петров"

customer3,"Иван","Петров"

customer4,"Петр","Иванов"

customer5,"Иван","Иванов"

staff.csv:

staff1,"Иван","Иванов"

staff2,"Петр","Петров"

staff3,"Иван","Петров"

staff4,"Петр","Иванов"

staff5,"Иван","Иванов"