Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Разработка сервера системы мобильного маркетинга

Курсовая работа студента 445 группы

Зарубина Михаила Сергеевича

Научный руководитель Вячеслав Алексеевич Кириллин
кандидат ф.-м.н.

Санкт-Петербург
2010

Оглавление

Введение

О предметной области:

Проект SmartKupon (c) - (www.smartkupon.ru) – это новый рекламный инструмент, в основе которого лежит привлечение клиентов в заведения, посредством предоставления им специальных дисконтных условий и уникальных предложений. Одной из основных составляющих системы являются купоны различных заведений города, позволяющие получить скидки, бонусы, которые не предоставляются в других поощрительных акциях этих заведений. Иначе говоря - это система мобильного маркетинга, использующая классическую архитектуру: клиент - сервер.

Клиентское приложение SmartKupon - это программа в мобильном телефоне, которая доступна для установки совершенно бесплатно каждому владельцу практически любого современного мобильного устройства. На данный момент она реализована для основных современных мобильных платформы: J2ME, iPhone, Android, Windows Mobile..

Серверная часть – это комплекс, состоящий из нескольких компонент, схематически изображенных на рис.1.

Помимо самих самих клиентских приложений не менее важным компонентом системы SmartKupon является устройство-приложение валидатор. Оно предназначено для верификации купонов клиента. Если говорить простым языком, то это устройство по функциональности похоже на валидатор, используемый в общественном транспорте (автобус, трамвай, метро), только проверяет он не проездные документы, а купоны и карты со скидками.

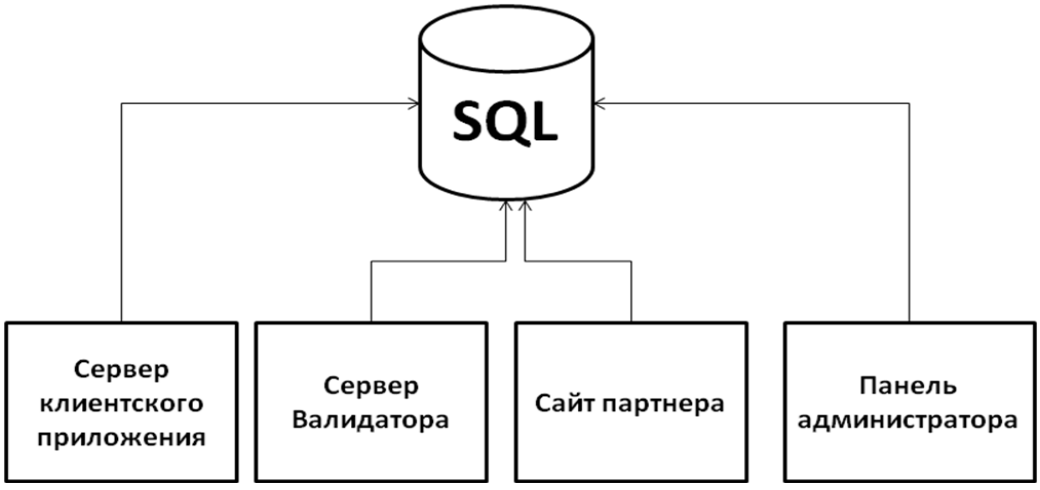

Рис.1

Система SmartKupon изначально состояла из серверной части, клиентского приложения и валидатора, но функциональность этих составляющих была минимальная. В частности, не было таких важных возможностей, как:

1) Механизм выдачи персонализированного контента. Пример: выдача картинок мобильному клиенту, специально сжатых сервером под размер экрана конкретного мобильного телефона
2) Персонализированный учет статистики.
3) Таргетированная выдача контента. Пример: выдавать пользователю наиболее интересные лично ему предложения, акции, купоны и не отображать непривлекательные для данного пользователя купоны.
4) Механизм контроля и обеспечения непрерывной работы сервера.
5) Система, позволяющая отслеживать ошибки приложений клиента и просматривать историю всех операций каждого пользователя в отдельности.
6) Механизм предотвращения ошибок и защита от мошенничества в алгоритме валидации

Цель работы: Разработать и интегрировать в серверную часть системы функциональность, описанную в пунктах 1) – 6)

1. Обзор средств и подходов

Используемые технологии:

Серверная часть системы SmartKupon реализована с применением следующих технологий:

- СУБД – MySQL
- Система для проектирования базы данных MySQL Workbench
- Остальные компоненты – написаны на языке Java, с применение библиотек Java EE, основное взаимодействие идет через HTTP Servlet
- Контейнером сервлетов является Apache TomCat
- Взаимодействие с базой данной осуществляется посредствам драйвера JDBC
- Пользовательской веб-интерфейс реализован через JSP, с применением технологии JSTL
- Логи сервера ведутся с помощью библиотеки log4j
- Взаимодействия сервера с клиентом осуществляется через протокол основанный на технологии XML

Уточнение задач:

1) Механизм выдачи персонализированного контента.

   Пример: выдача картинок мобильному клиенту, специально сжатых сервером под размер экрана конкретного мобильного телефона

   Для того чтобы реализовать данную возможность, сервер должен “уметь” следующее:

   a. Различать клиентов, и хранить данные о состоянии приложения состояние клиента
   b. Определять характеристики мобильного устройства (например, разрешение экрана)
   c. Сжимать изображения JPEG компрессором
   d. Хранить сжатые изображения

2) Персонализированный учет статистики.

   В панели администратора должна быть возможность увидеть: где, кто и когда воспользовался услугой от SmartKupon, т.е. предъявил купон. Для этого, как минимум, нужна следующая функциональность:

   a. Хранить уникальные сущности клиентов
   b. Хранить уникальные сущности сотрудников партнера

   c. Хранить список всех предъявленных купонов каждым клиентом

3) Таргетированная выдача контента.

   Пример: выдавать пользователю наиболее интересные лично ему предложения, акции, купоны.

   Что нужно со стороны сервера:

   a. Хранить статистику для каждого пользователя по всем операциям над купонами: просмотр, взятие, предъявление, удаление.
   b. На основе собранной статистики выдавать клиенту купоны

4) Механизм контроля и обеспечения непрерывной работы сервера.

   Эта задача требует дополнительного исследования, но очевидно, что нужно следить за uptime временем сервера.

5) Система, позволяющая отслеживать ошибки приложений клиента и просматривать историю всех операций каждого пользователя в отдельности.

   Что нужно со стороны сервера:

   a. Хранить уникальные сущности клиентов
   b. Принимать логи с мобильных устройств и сохранять их на сервере
   c. Организовать удобный доступ к любому событию, произошедшему в приложении на любом мобильном устройстве

6) Механизм предотвращения ошибок и защита от мошенничества в алгоритме валидации.

   Эта задача требует дополнительного исследования, но, скорее всего, придется переделывать алгоритм валидации. Основные требования к алгоритму:

   a. Код валидации должен быть коротким, 4-5 знаков
   b. Вероятность ввести не тот код должна быть минимальный < 1%

Из всего вышеперечисленного можно сделать вывод, что переработке подлежат все компоненты сервера, т.е. база данных, сервер мобильного приложения, сервер валидатора, партнерский сайт и панель администратора.

2. Описание решения

В этом разделе будет подробно описана каждая задача, и приведен алгоритм ее решения.

2.1) Механизм выдачи персонализированного контента.

Под этими словами скрывается несколько задач:

a. Выдача картинок нужного разрешения для мобильного клиента, так, чтобы ему не приходилось ее масштабировать.
b. Оправка произвольных сообщений клиентам:
    i. Сообщения от администрации
    ii. Сообщения о начислении балов

Итак, первая часть задачи подробнее:

Так как существует довольно много различный разрешений экранов у мобильных устройств, сначала было решено хранить на сервере только один размер картинки для купона, а масштабировать непосредственно в мобильном приложении клиента. Но у данного метода сразу было выявлено несколько недостатков:

a. Неэкономное использования канала GPRS/EDGE. Для телефонов с небольшим разрешением экрана, удобнее было бы показывать картинки в 2-3 раза меньше исходных по размеру. А при данном подходе пользователь тратит в 2-3 раза больше денег на мобильный интернет. Так же увеличивается время ожидания загрузки картинки с сервера
b. На клиентском устройстве тратится довольно много времени на масштабирование картинки, особо это ощутимо на Java телефонах с маломощным процессором.
c. На слабых Java телефонах порой настолько мало памяти, что они просто не с состоянии решить задачу масштабирования картинок в режиме реального времени. И приложение просто закрывается с ошибкой OutOfMemory.

Поэтому было решено перенести нагрузку по масштабированию картинок на сервер. Заодно было решено найти более универсальный подход. Т.е. если дизайнер рисует купон с разрешением 800x480, то после сжатия до 176x220 или более мелкого разрешения картинки становятся очень плохого качества, а если на них располагался текст, то прочесть его не удается. Поэтому было решено выяснить, какие разрешения являются наиболее распространенными. Собранные статистические данные наглядно отражены на рис.2

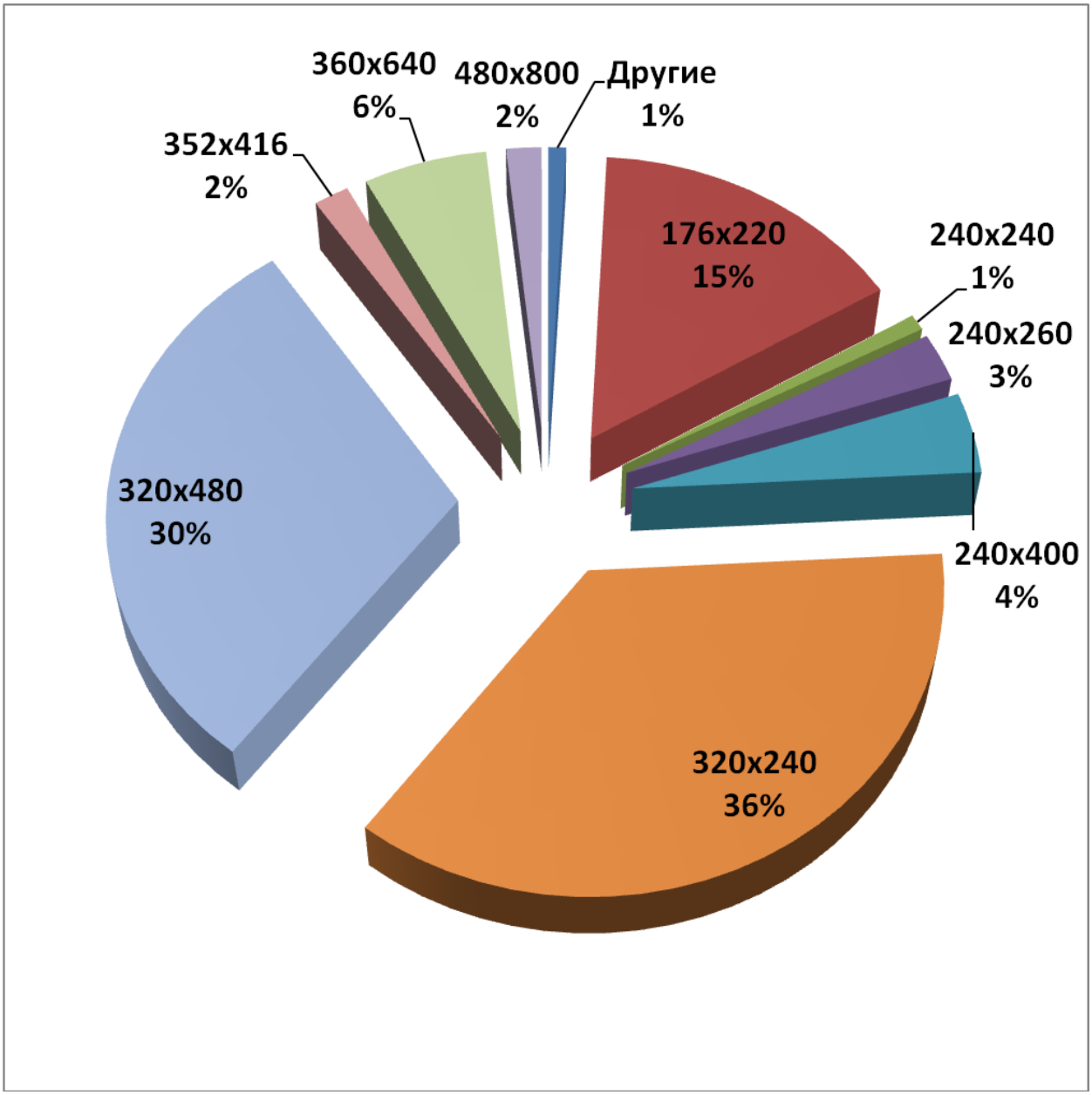

Рис. 2

Очевидно, что разрешения 320x240, 320x480 и 176x220 являются наиболее распространенными. А 800x480 является на данный момент самым большим разрешением для мобильных телефонов. Поэтому было принято решение, что дизайнер будет рисовать картинки для четырех экрнанов 800x480, 320x240, 320x480, 176x220. А для остальных экранов сервер будет автоматически масштабировать нужную картинку, выбирая наиболее подходящую из четырех.

Для хранения всех картинок была разработана таблица CouponImageBinary (см. рис3).

В качестве алгоритма сжатия изображений был выбран алгоритм, заимствованный у программы с открытым исходным кодом - Fast Image Resizer for Java.

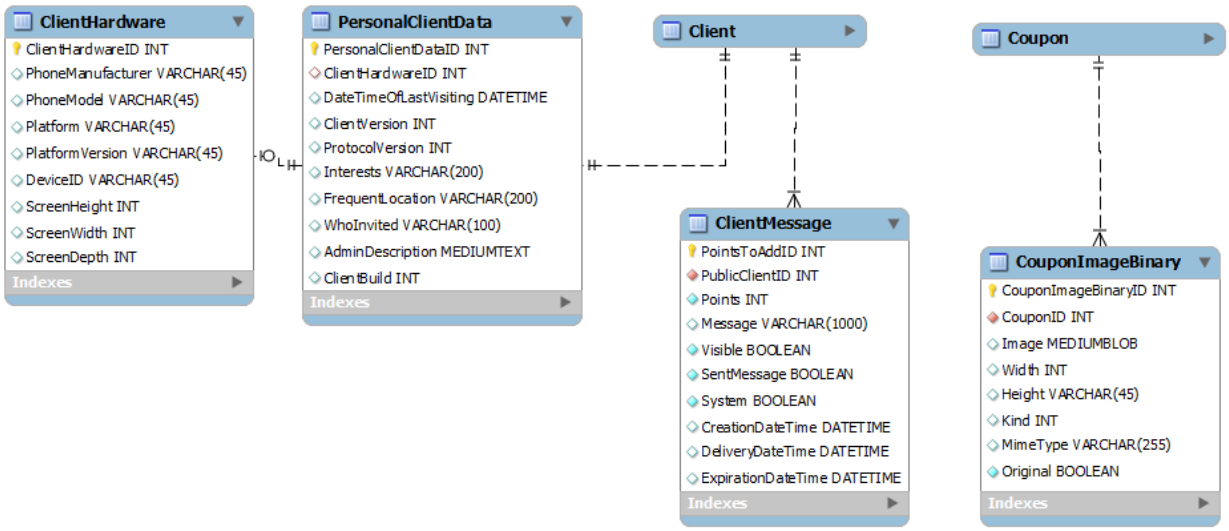

Рис. 3

Еще одним важным моментом является выяснение, какая картинка нужна клиенту. Если клиент будет напрямую отправлять серверу свое разрешения экрана, то это не значит, что именно такого размера ему нужна картинка, т.к. на некоторых телефонах приложение не может работать в полноэкранном режиме, а сверху и снизу отображаются системные панели, причем на разных телефонах они разные по высоте. Из этого был сделан вывод, что клиент должен сообщать серверу разрешение области, на которой реально можно отобразить картинку. Для этого в сервере клиентского приложения был добавлен специальный запрос в протокол (см. рис.4)

```
<?xml version="1.0" encoding="UTF-8"?>
<request kind="frontside" protocol-version="4" type="getimage" client-
id="4B37Ab68C8F259CC58E5986E6181B875" coupon-id="18">
    <image-area width="300" height="300"/>
</request>

<?xml version="1.0" encoding="UTF-8"?>
<response protocol-version="3" type="getimage">
    <image coupon-id="18" kind="frontside">[BASE64_IMAGE]
    </image>
</response>
```

Рис.4

Вторая часть задачи - оправка произвольных сообщений клиентам:

Для хранения сообщений в БД была добавлена таблица ClientMessage (см. Рис. 3). А в панель администратора специальная форма (см. Рис. 5).

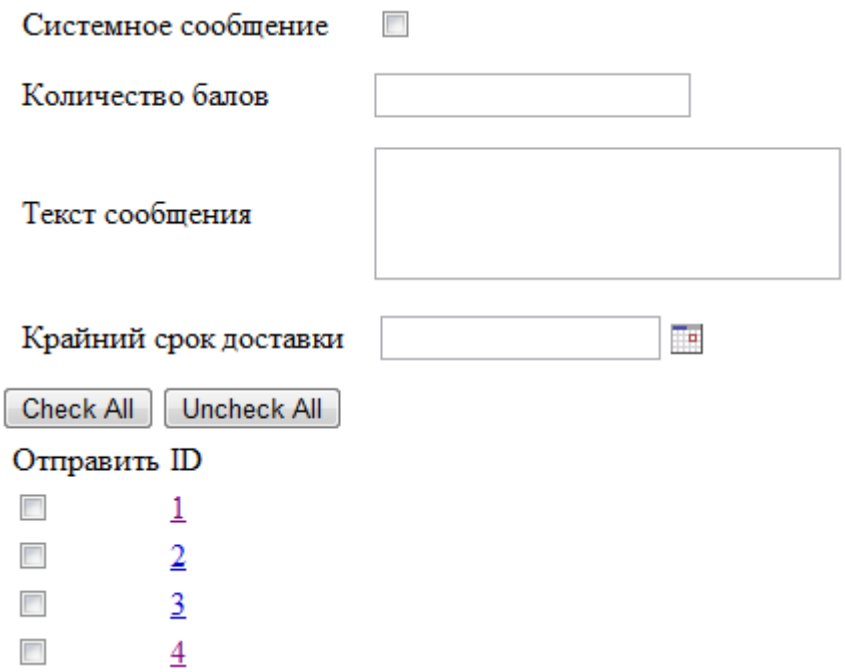

Рис. 5

2.2) Персонализированный учет статистики.

Изначально был сделан просто в виде счетчиков для купона: число скачиваний, предъявлений, удалений (см. Табл. 1). Но со временем оказалось, что этих данных недостаточно, и хочется иметь более подробную статистику для каждого клиента и купона в виде: где, кто, когда и что использовал. Для этого в базе данных была создана таблица ClientCoupon (см. Рис. 6). И в панели администратора добавлена соответствующая форма (см. Табл. 2).

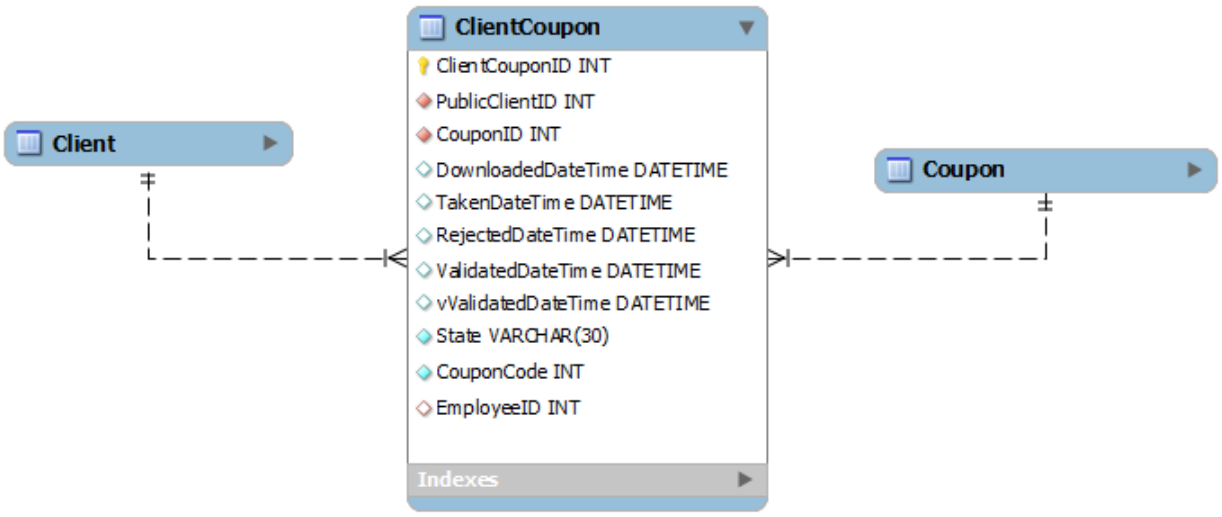

Рис. 6

Партнер	Купон	Скачали	Взяли	Предъявили
Coca Bar	Чашка кофе или чайничек чая	315	227	14
Пиццерия Челентано	Напиток в подарок	164	69	7
Конюшенный Двор	Бесплатный вход	188	89	5
Мини-отель Водограй	Скидка 30%	126	31	4
Посольство красоты	Скидка 10% на первое посещение	123	31	3
Sushi Land	Бутылка вина в подарок	115	31	3

Табл. 1

Партнер	Купон	Клиент	Время предъявления	Сотрудник партнера
Кофейня "Манго"	Десерт от Шеф-кондитера.	45	2010-05-12 15:03:12.0	Иванов
Шаром покати	Скидка 50%	9	2010-05-12 14:55:33.0	Петров
Господин Оформитель	Подарки от Господина Оформителя!	9	2010-05-04 16:39:10.0	Сидоров
Пиццерия Челентано	Напиток в подарок	63	2010-05-02 14:54:57.0	Иванов
CoffeeNoa	Вторая чашка кофе бесплатно!	9	2010-04-28 17:20:31.0	Иванов
Трибунал Бар	10% скидка	74	2010-04-23 22:08:17.0	Петров
Ресторан "Дворянское Гнездо"	Обед в подарок	13	2010-04-23 18:14:55.0	Сидоров

Табл. 2

2.3) Таргетированная выдача контента.

В рамках данной курсовой работы было решено реализовать один из самых простых алгоритмов, основанный на всей предыдущей истории операций клиента. Т.е. учитываются следующие действия: просмотр купона, взятие купона себе, предъявление купона, удаление купона и переворачивание купона.

Алгоритм сбора предпочтений клиента:

Константы

(значения констант не имеют научного обоснования, были взяты те значения, при которых достигается удовлетворяющий нас результат)

CATEGORY_START_PRIORITY = 50;

COUPON_START_PRIORITY = 0

CATEGORY_TAKE_PRIORITY_INCREMENT = 0.05

CATEGORY_LOOK_PRIORITY_INCREMENT = 0.025

COUPON_DECLINE_PRIORITY = -10

CATEGORY_DECLINE_PRIORITY_DECREMENT = 0.05

MERGE_INTERVAL = 5

Необходимые данные:

1. Категории:

   Примеры категорий: Ресторан, Кафе, Бар, Фастфуд

   Тип заведения: Дешёвый, Недорогой, Нормальный, Недешёвый, Дорогой

   И другие…
2. Для каждой категории для каждого купона для каждого клиента надо хранить число – приоритет этой категории.

   Предположительный вариант исполнения:

   Таблица категорий, таблица купонов, таблица клиентов – уже есть

   Таблица приоритетов:

PriorityID	Priority	CouponID	ClientID	CategoryID
int	float	int	int	int

3. Начальный приоритет: CategoryPriority = CATEGORY_START_PRIORITY
4. Для каждого купона хранится его приоритет.

   AprioriCouponPriority – float

5. Начальный приоритет: AprioriCouponPriority = COUPON_START_PRIORITY

Алгоритм построения списка купонов

1. Вычисления приоритета купона для каждого клиента: CouponPriority = [Sum(i=1 to n) (CategoryPriority_i)] / n + AprioriCouponPriority, где n – количество категорий у данного купона
2. Сортируем по убыванию по CouponPriority массив всех купонов (переменная AllCoupons)
3. Генерируем массив случайной выборки купонов (алгоритм генерации ниже) (получаем переменные RandomCoupons – выборка случайных купонов, и NormalCoupons – оставшиеся купоны)
4. Объединяем массивы NormalCoupons и RandomCoupons по правилу – вначале MERGE_INTERVAL купонов из NormalCoupons, потом 1 купон из RandomCoupons

Алгоритм построения случайной выборки

1. Вычисляем длину массивы случайно выбранных купонов (RandomCoupons): AllCoupons.Length / MERGE_INTERVAL
2. Вычисляем длину массива оставшихся купонов (NormalCoupons): AllCoupons.Length – RandomCoupons.Length
3. Генерируем RandomCoupons.Length различных случайных чисел в промежутке от 0 до AllCoupons.Length. Т.е. каждому купону сопоставим случайное число.
4. Элементы с этими номерами переводим в массив RandomCoupons, в полученном порядке
5. Остальные купоны переводим в массив NormalCoupons, не меняя их порядка

Реакции на действия клиента

1. Клиент посмотрел заднюю сторону купона:
   Для всех категорий, которые есть у купона, CategoryPriority *= (1 + CATEGORY_LOOK_PRIORITY_INCREMENT)
2. Клиент взял купон себе
   Для всех категорий, которые есть у купона, CategoryPriority *= (1 + CATEGORY_TAKE_PRIORITY_INCREMENT)
3. Клиент нажал на купоне "Мне это не интересно"
   Для всех категорий, которые есть у купона, CategoryPriority *= (1 – CATEGORY_DECLINE_PRIORITY_DECREMENT)
   AprioriCouponPriority -= DECLINE_COUPON_PRIORITY

Резюме:

Данный алгоритм неплохо справляется с поставленной задачей, но имеет видимые недостатки: большой объём расходуемой памяти при большом количестве купонов или большом количестве одновременно подключившихся клиентов.

Так же этот алгоритм не учитывает такой важной вещи как местоположение клиента. Но этого пока не требовалось, так как в системе нет механизма получения местоположения клиента. А как только такой механизм появится, расширить текущий алгоритм будет не сложно.

2.4) Механизм контроля и обеспечения непрерывной работы сервера.

Вечная проблема всех серверов – это борьба за непрерывную и бесперебойную работу в режиме 24/7 или иначе100% uptime. А, как известно, 100% uptime никто гарантировать не может, но есть много способов приблизиться к нему. Система SmartKupon дислоцируется на удаленном сервере, поэтому нет непосредственного контроля и способов предотвращения таких ситуаций, как отключение электропитания или интернета. Но если произошла какая-то подобная неприятная ситуация, например, зависла виртуальная машина (машина, на которой расположено программное обеспечение системы SmartKupon), то необходимо об этом оповестить системного администратора и программистов в кратчайшие сроки, чтобы они предприняли какие-то меры. Именно об этой технологии и пойдет речь в данном разделе курсовой.

Понятно, что критерием работоспособности сервера является корректность ответов на запросы. При разработке системы SmartKupon были выявлены две основные ситуации, при которых сервер считается нерабочим:

2.1) Зависание одной из компонент сервера

2.2) Невозможность подключиться к базе данных

При зависании какой-то части сервер просто перестает реагировать на определенные виды запросов. Существует довольно таки много способов обнаружения этой ситуации. А что делать, когда не удается подключиться к базе данных, и как это проверять автоматически? Первое, что приходит на ум – это делать проверку на самом сервере, и если не удалось подключиться, то отправлять оповещение разработчикам. Какие минусы у данного подхода:

a. Невозможность сверх быстрого оповещения нужных людей в любое время суток, при использовании доступных вариантов (e-mail или Instance Messager).

b. Если отказала одна компонента (подключение к БД), то велика вероятность, что и другая может не работать (SMTP сервер)

Поэтому приходим к выводу, что надо наблюдать за сервером другими способами извне. Было проведено исследование средств, позволяющих справиться с данной задачей, т.е. регулярно обращаться к серверу и в случае его недоступности отправлять всевозможные оповещения (e-mail, sms) администратору и программистам.

Были найдены следующие сервисы, позволяющие это делать: host-tracker.com, siteuptime.com, binokl.info, alertra.com. Но выбор пал на host-tracker.com, т.к. он лучше всего справляется с задачей отправки смс на телефоны российских мобильных операторов.

Для проверки доступности базы данных в каждую компоненту сервера был добавлен обработчик запроса вида: <request type="smart-ping">. На что сервер должен проделать следующие операции:

a. Подключиться к БД
b. Взять из определенного поля таблицы известное значение. (В нашем случае было добавлено булевское поле ServerWorking=true)
c. Отправить в ответ значение, полученное из БД

Т.е. в случае корректной работы должно всегда отправляться true, а любой другой ответ считается неверным. И если сервис host-tracker.com получает неверный ответ, например false, то всем программистам и администратору, отвечающим за серверную часть, приходит смс, что случились какие-то проблемы. Этот метод позволят обнаружить и устранить проблему в первые 5-10 минут после ее возникновения.

2.5) Система, позволяющая отслеживать ошибки приложений клиента и просматривать историю всех операций каждого пользователя в отдельности.

При разработке программного обеспечения возникает очень много ситуаций, когда невозможно полностью протестировать продукт и отдать пользователю приложение со стопроцентной гарантией, что в нем нет ошибок. А если учитывать специфику разработки приложений под мобильные платформы, то таких ситуаций становится на порядок больше. Т.к. число моделей мобильных телефонов исчисляется сотнями, а новые модели появляются каждый месяц, физически невозможно в рамках небольшой компании проверять приложение на всех устройствах. Единственным доступным вариантом остается вести подробный журнал действий пользователя каждого мобильного телефона, и в случае поведения приложения, отклоняющегося от нормы, сообщать об этих

отклонениях разработчикам. Проще говоря, вести подробные логи, и сообщения обо всех ошибках (exception, error) отправлять программистам. Но т.к. количество пользователей приложения исчисляется сотнями и тысячами, то необходимо упорядочить всю накопленную информацию, чтобы с ней было удобно работать.

Со стороны клиентского приложения надо просто аккуратно и подробно записывать все логи, и периодически отправлять их на сервер (например, по требованию сервера, или при возникновении исключения на клиентском приложении).

Со стороны сервера необходимо упорядочивать полученные логи, обеспечивать быстрый доступ к ним, выводить их в удобочитаемом виде для разработчиков, и своевременно оповещать программистов о случившихся ошибках в клиентских приложениях. Для этого были разработаны следующие части:

1) В базу данных были добавлены необходимые таблицы ClientError и ClientHistory (см. рис. 7)
2) В протокол сервера клиентского приложения добавлены соответствующие запросы (см. пример на рис. 8)
3) В панель администратора добавлен интерфейс для удобного и быстрого отображения нужной истории (см. пример на рис. 9 и рис. 10)
4) В сервер клиентского приложения добавлен модуль оповещения (по e-mail) разработчиков об ошибках.

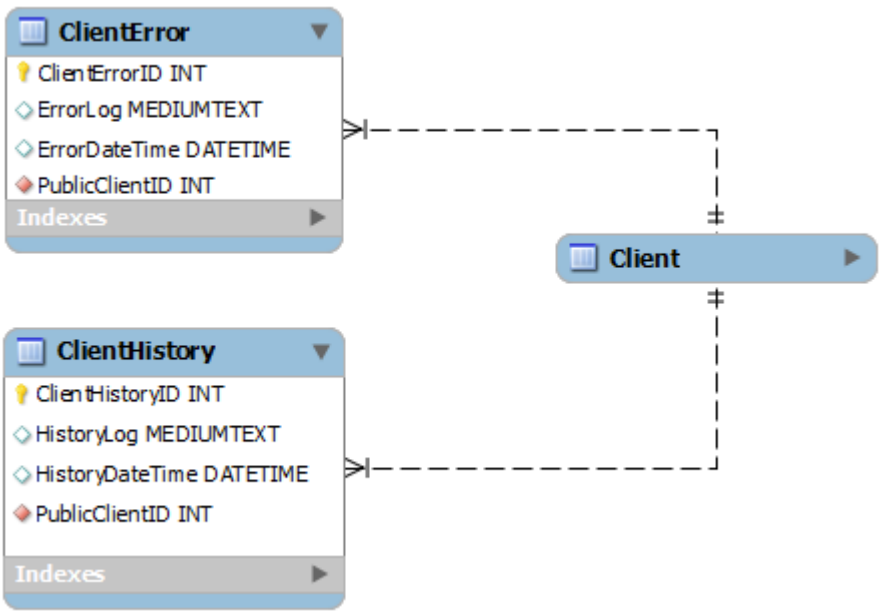

Рис.7

```
<?xml version="1.0" encoding="UTF-8"?>
<request protocol-version="4" type="log" timestamp="1268219669" client-
id="4B37Ab68C8F259CC58E5986E6181B875">
   <log class="XPathQuery" timestamp="1268199523" level="error" module="low level">
       <![CDATA[Nodes were nil.]]>
   </log>
   <log class="SKCache" timestamp="1268199527" level="verbose" module="persistent storage">
       <![CDATA[Image was found. Hash = 2_3]]>
   </log>
</request>
```

Рис. 8

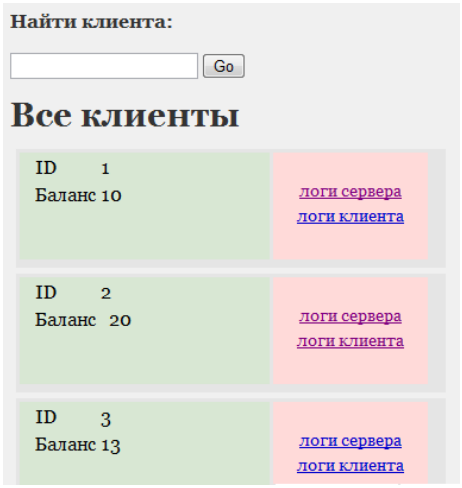

Рис. 9

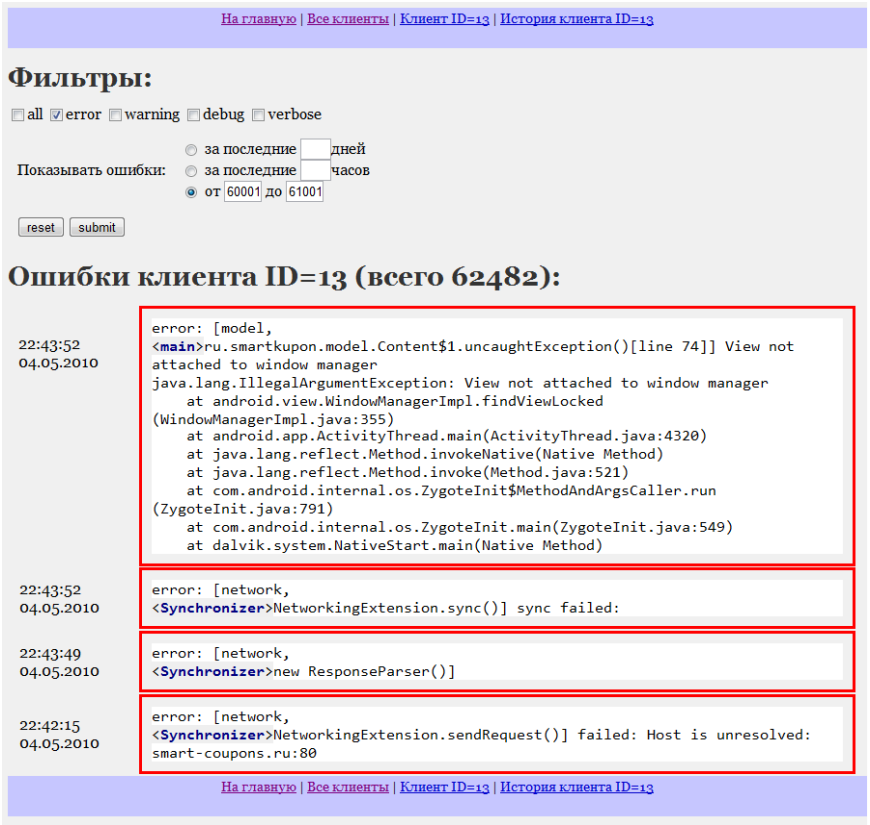

Рис. 10

Как работает эта система в комплексе.

Во время использования мобильного приложения все основные функции записываются в журнал лога. Сообщения в логах имеют несколько уровней важности, например: error, warning, debug, verbose. Если происходит какая-то внештатная ситуация, то в лог записывается сообщение с уровнем важности error, и диспетчер логов понимает, что нужно отправить всю историю на сервер. Если отправить логи на сервер сразу не удалось, то они сохраняются в память и будут отосланы при следующем подключении к сети интернет. Если логи были успешно отправлены, то на этом миссия клиентского приложения заканчивается, и сервер начинает обрабатывать логи. Все сообщения сохраняются в соответствующих таблицах в БД ClientHistory и ClientError (см. рис.7). Если в логах были найдены сообщения приоритета error, то сервер оповестит нужного разработчика по e-mail. Как это происходит: сервер знает, какой разработчик отвечает за какую мобильную платформу и знает его e-mail. (Пример: J2ME – Иванов, iPhone - Петров). При получении лога с приоритетом error текст данного сообщения вставляется в письмо и добавляется ссылка на веб-интерфейс панели администратора, вида

http://web-site.ru/admin/clienthistory.jsp?ClientID=2&Period=Count&Count=100

Пройдя по ссылке (см. пример рис.10), разработчик может подробно ознакомится со всеми действиями пользователя и понять, из-за чего произошла такая ошибка.

2.6) Механизм предотвращения ошибок и защита от мошенничества в алгоритме валидации.

Клиент, предъявляя в заведении купон, должен быть уверен, что бонусы от этого купона будут зачислены на его счет. А партнер хочет быть уверен, что ему предъявляют настоящий купон, а не что-то похожее на него (скриншот например).

По сути ключ или код купона – это число. Понятно, что чем из большего числа символов состоит число, тем ключ становится надежнее и уникальнее. Однако приходится учитывать способность валидатора быстро считать этот ключ. Валидатор представляет собой устройство, основанное на мобильном телефоне с камерой. Считывание кода происходит при помощи камеры. К сожалению, не всегда можно воспользоваться этой удобной функцией. Например, на экране телефона клиента трещина, или любой другой дефект, мешающий считать драгоценные биты информации. При этом человек довольно-таки легко может понять, что написано на экране. Получается ситуация, часто происходящая в супермаркетах: штрих код (см. рис 11) не может быть считан устройством, поэтому приходится его вводить вручную.

Рис. 11

Вводить такое число (см. рис. 13) непозволительно долго, тем более при вводе этого числа можно еще и ошибиться. Но почти любой человек в состоянии ввести 4-5 цифр без ошибок с первого раза. Поэтому в рамках данной курсовой будет представлен алгоритм валидации с использованием коротких чисел.

Описание алгоритма валидации:

1) В БД у каждой сущности Partner есть таблица PartnerCode, в которую изначально складываются 8999 перемешенных чисел с 1000 по 9999.

2) При взятии купона клиенту выдается одно из этих чисел, при этом оно сразу убирается из таблицы PartnerCode, т.е. получается, что у каждого клиента есть уникальный код для каждого купона, а, значит, колизий не возникнет.

3) Если увеличится число клиентов, одновременно взявших данный купон, но еще не воспользовавшихся им, скажем до 1000, то вероятность свалидировать правильный номер случайно возрастает, в таком случае мы просто переходим на 5тизначный алгоритм валидации. Т.е. просто генерируем новые списки от 10000 до 99999. Этого нам будет достаточно всегда при текущих масштабах системы. (до 100 тыс. пользователей)

4) После валидирования купона на 2х устройствах, т.е. клиентское приложении и валидатор, это число возвращается обратно в таблицу PartnerCode, но встает в конец очереди, и скорее всего он уже выдан не будет, т.к. при увеличении числа клиентов произойдет переход на 5ти-значный алгоритм, но при интенсивном использовании приложения малым количеством клиентов эти списки будут прокручиваться по нескольку раз, не переходя на 5ти-значный алгоритм валидации.

5) При валидировании купона на сервере происходит сопоставление времени предъявления купона и его валидации. При этом время корректируется сервером. Если разность между временем предъявления и валидации велика (более суток), то сервер сообщает об ошибке и просит заново ввести код.

Как можно оценить вероятность ввода неверного числа?

Предположим, что 2000 клиентов взяли один купон. Если рассматривать 5ти-значные коды, то получается P(вероятность)=2000/89999 =0.022 ~ 2%

Если учесть, что максимум 50 человек в день может прийти в заведение по данному купону, а это можно проверить с учетом коррекции времени, то получается что P_1=P/(2000/50)= 0.022/40 ~ 0.06%

Характеристики:

- 4-5 циферный код. (Т.е. быстрый ввод данных)
- Вероятность случайно ввести чужой код 0.06% . (Довольно таки приемлемая для данной системы)

Недостатки:

Нельзя использовать в качестве платежной системы, для сравнения у кредитных карт вероятность правильного ввода номера чужой кредитной карты ~0,0001%

Заключение

Целью работы было разработка и интеграция в серверную часть системы недостающей функциональности:

1) Механизм выдачи персонализированного контента.
2) Персонализированный учет статистики.
3) Таргетированная выдача контента
4) Механизм контроля и обеспечения непрерывной работы сервера.
5) Система, позволяющая отслеживать ошибки приложений клиента и просматривать историю всех операций каждого пользователя в отдельности.
6) Механизм предотвращения ошибок и защита от мошенничества в алгоритме валидации

Все задачи были реализованы и успешно интегрированы в рабочую систему.

Планы на будущее:

- Адаптировать систему таргетированной выдачи контента с учетом местоположения клиента.
- Улучшить алгоритм валидации.

Список литературы

1. Apache Tomcat Documentation - http://tomcat.apache.org/tomcat-5.5-doc/index.html
2. MySQL Documentation - http://dev.mysql.com/doc/
3. OReilly Head First Servlets and JSP - http://www.torrentdownloads.net/searches/OReilly.Head.First.Servlets.and.JSP.2nd.Edition.Mar.2008.pdf
4. Spring Documentation http://www.springsource.org/documentation