**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ**

**И ИНФОРМАТИКИ**

**Кафедра многопроцессорных систем и сетей**

**РАЗРАБОТКА ПРОГРАММНОЙ СИСТЕМЫ ДЛЯ УПРАВЛЕНИЯ ЦВЕТОЧНЫМИ ЗАКАЗАМИ С ИСПОЛЬЗОВАНИЕМ ФРЕЙМВОРКОВ SPRING И ANGULARJS**

Курсовая работа

Павловой Маргариты Валерьевны

студентки 4 курса,

специальность «информатика»

Научный руководитель:

доцент

Рафеенко Е.Д.

Минск, 2015

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 4](#_Toc443394073)

[1 АЛГОРИТМЫ ПОИСКА ПОХОЖИХ ИЗОБРАЖЕНИЙ 8](#_Toc443394074)

[1.1 Алгоритм поиска похожих изображений на основе цвета и текстуры 8](#_Toc443394075)

[1.2 Метод поиска похожих изображений с использованием индекса качества изображения 10](#_Toc443394076)

[2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 14](#_Toc443394077)

[2.1 Структура проекта 14](#_Toc443394078)

[2.2 Технические возможности созданного приложения 18](#_Toc443394079)

[ЗАКЛЮЧЕНИЕ 24](#_Toc443394080)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc443394081)

**РЕФЕРАТ**

Курсовой проект, 19 страниц, 4 рисунка, 4 источника.

**Ключевые слова**:SPRING, SPRING SECURITY, HIBERNATE, ANGULARJS, КОНТЕЙНЕР, ФРЕЙМВОРК, ФАБРИКА, КОМПОНЕНТА, МОДУЛЬ, ДИРЕКТИВА, NEXMO, GOOGLE MAPS.

**Объект исследования:** технологии для разработки WEB-приложений.

**Цель работы:** разработка программной системы для управления цветочными заказами с использованием фреймворков Spring и AngularJS.

**Методы работы:** исследование эффективных решений для создания WEB-приложения.

**В результате** проведенной работы были исследованы технологии для разработки WEB-приложения.

# ВВЕДЕНИЕ

Роль цифровых данных значительно возросла в последние 10 лет в связи с увеличивающимся распространением цифровой техники. Возрастает количество и доступность цифровых фотоаппаратов, сканеров, настольных издательских систем, настольных баз данных. Постоянно увеличивается пропускная способность каналов сети Интернет. Все это приводит к значительному росту количества обрабатываемых данных, однако алгоритмы обработки и поиска не успевают обеспечить адекватную реакцию на столь значительное увеличение объема доступных цифровых материалов. Особенно сильно это ощущается в области средств и алгоритмов поиска в мультимедийных базах данных.

На данный момент поиск в большей части мультимедийных баз производится с применением принципов и алгоритмов, разработанных несколько десятков лет назад для другого типа данных, что приводит к неудовлетворительным результатам поиска.

Большая часть имеющихся на данный момент алгоритмов не учитывает особенности восприятия человеком изображений, и базируется на разметке баз изображений текстовыми тэгами и набором атрибутов, по которым в дальнейшем можно производить поиск с использованием классических алгоритмов поиска строки.

Восприятие человеком информации является трудноформализуемой задачей, и результаты поиска с использованием существующих алгоритмов поиска слабо коррелируют с тем, что хотел бы видеть пользователь, осуществляющий поиск.

Зачастую, с увеличением размера базы изображений традиционные алгоритмы исчерпывают свои возможности. Например, для ускорения поиска можно просматривать уменьшенные копии изображений. Поиск изображения среди сотен уменьшенных копий еще является посильной задачей для пользователя, однако увеличение порядка размера базы до тысяч изображений делает данную задачу практически невозможной. Поэтому при существенном расширении базы данных поиск методом полного перебора изображений теряет свою эффективность.

Одной из распространенных стратегий поиска является индексация по ключевым словам и атрибутам. Данный подход имеет свои преимущества, однако выдвигает ряд требований, исполнение которых порой становится чрезвычайно затруднительным.

Первой причиной является, собственно необходимость разметки базы изображений ключевыми словами, которая является весьма затратной процедурой. Второй причиной является то, что некоторые визуальные аспекты изображения в принципе являются трудноописуемыми или неописуемыми вообще. Далее, пользователю может быть неизвестно, какие аспекты внесены в индекс, а какие нет. Стоит отметить, что некоторые классы изображений (например, текстуры) вообще с большим трудом поддается разметке ключевыми словами. Третьей причиной является неоднозначность восприятия изображения и, как следствие из этого, поиск по базе изображений, размеченной тэгами одним человеком для другого значительно менее эффективен.

Изложенные выше соображения привели к росту исследований в области контекстного поиска изображений. Контекстный поиск изображений имеет ряд других названий, среди которых «поиск по содержимому», «поиск по образцу», «поиск по эскизу», «поиск по методу подобия». Можно заметить, что контекстный поиск является альтернативой традиционным методам поиска и может применяться совместно с ними.

Поиск изображений похожих на заданное изображение находит применение во многих областях, включая архитектуру, телевидение, средства мультимедиа, графический дизайн, историю искусств, криминологию, геологию, медицину.

Дизайнер может потребовать найти изображения, похожие на заданное, для использования в качестве альтернативы или на замену.

В криминологии данные алгоритмы востребованы при поиске фотографии по фотороботу. Также для правоохранительных органов актуальным является поиск фрагмента или целого изображения в записи камер наблюдения. Как правило, при этом используется фотография грубого разрешения.

Архитектор может использовать данный метод поиска для отбора решений, которые являются удачными для данной местности, а также для поиска аналогов задумываемого или исполненного строения.

Данная задача также является актуальной и в домашней сфере. Увеличившаяся доступность цифровой аппаратуры привела к значительному росту домашних видео- и фотоархивов.

Контекстный поиск призван качественно улучшить результаты поиска в цифровых библиотеках. Во всех упомянутых областях совершенствование технологии формирования изображения является важной и общепризнанной проблемой.

За последние 10 лет в свете усиления интереса к данной проблеме появилось несколько систем контекстного поиска изображений. Стоит упомянуть наиболее известные системы, базирующиеся на поиске с использованием ключевых слов:

1. HotBot (http://hotbot.lycos.com)

2. NBCi (http://www.snap.com)

3. Yahoo! Image Search (http://search.yahoo.com/images)

4. Lycos multimedia searcher (http://multimedia.lycos.com)

5. AltaVista Images Search Center (http://www.altavista.com/image)

6. Google Image Search (http://images.google.com)

7. PICSearch (http://www.picsearch.com/)

Наиболее старой и известной системой является система контекстного поиска QBIC (Query By Image Content), разработанная компанией IBM. Некоторые исследователи отмечают, что данная система достаточно успешна в работе с цветом и текстурой, однако недостаточно точна при работе с формой изображения.

Большинство данных систем являются результатом исследований и скорее полигоном для испытания идей исследователя, нежели законченным продуктом. Преимущественно этим обосновывается то, что алгоритмы, лежащие в основе данных систем, обыгрывают один или два аспекта контекстного поиска. Большая часть этих систем использует цветовые и текстурные особенности, небольшая часть использует пространственные особенности, т.е. местоположение (в некоторых случаях распределение) особенности на картинке.

Поиски по цветовым особенностям, как правило, дают неплохие результаты, несколько хуже обстоит дело с поиском по текстурным особенностям. Исключением являются случаи присутствия в базе некой превалирующей текстуры. Поиск по форме изображения на данный момент дает плохие результаты в большинстве данных систем.

Таким образов, в данной работе будут рассмотрены алгоритмы поиска подобных изображений, выполнен их анализ и модификация для улучшений качества решения.

1. АЛГОРИТМЫ ПОИСКА ПОХОЖИХ ИЗОБРАЖЕНИЙ НА ОСНОВЕ ЦВЕТА И ТЕКСТУРЫ

# 1.1 Описание

**Одним из методо поиска похожих изображений является извлечение из каждого изображения несколько ключевых характеристик, которые будут использоваться для вычисления подобия между изображениями. Эти характеристики описывают содержимое изображения, и поэтому они должны быть тщательно выбраны в соответствии с контекстом. Таким образов, в данном разделе будет рассмотрен поиск похожих изображений в терминах цвета и текстуры.**

**Цветовая характеристика представлена с помощью среднего цвета**  и матрицы ковариации . Пусть

**где – число пикселей изображения, – цвет -ого пикселя в канале . Предполагая использование оригинального RGB (Red, Green, Blue) цветового пространства, матрица ковариаций будет:**

**Текстурная характеристика извлечена из обобщенной матрицы смежности. Предположим, изображение содержит столбцов и   
строк и** **уровней серого цвета. Пусть представляют столбцы, – строки, и – множество уровней серого цвета. Смежная матрица – матрица размерности , где**

**Т.е. количество смежных пар серого уровня и , для которых расстояние Пары пикселей и имеют расстояние и ориентацию**

**Обобщенная смежная матрица это расширение смежной матрицы для мультиспектральных изображений, т.е. изображений, представленных n цветовыми каналами. Пусть и два цветовых канала. Обобщенная ковариационная матрица будет:**

**Например, в случае цветного изображения, закодированного в трех каналах (RGB), имеется шесть смежных матриц: (RR), (GG), (BB), которые так же, как и смежная матрица серого уровня, вычисляются по одному каналу, и (RG), (RB), (GB), которые учитывают корреляцию между каналами.**

**Система поиска основана на мере сходства между заданным изображением и изображением базы данных .**

**Для цветовой характеристики, основанной на матрице ковариаций, мера расстояний Бхаттачарйя используется в следующем виде:**

**Где и вектора средних цветовых характеристик, и – ковариационные матрицы изображений и , - определитель матрицы.**

**Расстояние Евклида используется как расстояние между текстурными характеристиками**

**Объедим два расстояния в функцию согласующегося подобия:**

**Где**  и веса факторов, которые должны быть выбраны экспериментально.

# 1.2 Метод поиска похожих изображений с использованием индекса качества изображения

**Поиск похожих изображений является важным предметом исследований для мультимедийного управления базами данных. Эта статья используется модель индекса качества для поиска аналогичных изображений с цифровых баз данных. Чтобы ускорить извлечение, модель индекса качества разделяется на 3 фактора: потеря корреляции, искажение яркости и контрастности искажений.**

**В обработке изображений наиболее часто используемыми измерениями для оценки разницы между двумя изображениями являются среднеквадратичная ошибка (MSE), пиковое соотношение сигнала к шуму (PSNR), средняя абсолютная ошибка (MAE) и так далее. Эти методы измерения обычно легкие и имеют более низкую сложность вычислений. Тем не менее, в соответствии с экспериментами, большинство из методов не могут быть точно применены в строгих условиях тестирования или в искажении изображения окружающей средой. Кроме того, эти способы требуют человеческого зрения для измерения качества. Поэтому был предложен другой способ измерения качества изображения – индекс универсального качества изображения (индекс качества), чтобы оценить разницы между двумя изображениями.**

**Из-за быстрого продвижения в информации технология позволила получить доступ к большому количеству изображений в одно мгновение из любого уголка мира. Тем не менее из-за огромного количества доступных изображений, найти правильный образ из большого числа очень трудно. В начале ключевые слова были использованы для поиска требуемых изображений. Каждое изображение в образе базы данных связано с аннотацией ключевых слов. Но такие методы стали бесполезны и неэффективны, так как все больше и больше изображений поступают из Интернета и многие из них не включают ключевые слова. К тому же каждое изображение включает различные ключевые слова. Таким образом, некоторые исследователи разработали другие методы поиска подобных изображений. Один популярный метод поиска изображений по содержанию (CBIR), которых извлекает характеристики такие как цвет, текстура, форма, время, расстояние.**

**В данном алгоритме примем концепцию показателя качества для представления характеристики изображения для поиска похожих образов. Чтобы ускорить извлечение, модель индекса качества разделяется на 3 фактора: потеря корреляции, искажение яркости и контрастности искажений.**

**В 2003 году был предложен метод на основе CDESSO, который характеризовал сложность цвета, цветовые различия между соседними пикселями для 24-битного полноцветного изображения. Во-первых, был использован метод алгоритм K среднего, чтобы уменьшить цветовое пространство изображения. Он разделял все пиксели изображения на 64 кластера. Каждый кластер имел свой собственный бункер для записи разницы между соседними пикселями. Во-вторых, каждый пиксель в изображении был вставлен в ближайший кластер. Затем было просканировано уменьшенное изображение в спиральном порядке и вычислили разность между каждыми двумя соседними пикселями. Разность добавляется к соответствующему кластеру текущего пикселя. Было использовано 64 кластера, называемые цветовой гистограммой, как характеристики исследуемого изображения. С точки зрения поиска изображений, гистограмма заданного изображения сравнивается с гистограммой существующих изображений, используя Евклидово расстояние. По их экспериментальным результатам видно, что метод обеспечивает не только высокую степень точности, но и не изменяет своего качества при вращении изображения. Способ поиска изображения CDESSO, а также другие цветовые методы, основанные на CBIR, используют Евклидово расстояние, чтобы оценить разницу между двумя изображениями. Однако измерения, такие как MSE, PSNR и Евклидово расстояние, не могут быть точно применены в строгих условиях тестирования или при искажении изображения окружающей средой. Поэтому был предложен математически определенный универсальный указатель качества изображения.**

**Пусть оригинальное изображение, где - i-ый пиксель изображения , и – тестовое изображение. Расстояние между изображениями и измеряется с помощью:**

**Где – среднее изображения , определенное** ,  **– среднее изображения , определенное** , – дисперсия **,**  – дисперсия **,**  – корреляция между изображением  **и изображением , определяемая**

Диапазон Q от -1 до 1. Если значение Q близко к 1, тогда можно сказать, что оригинальное изображение и тестируемое изображение схожи. Если значение Q равно -1, тогда оригинальное изображение и тестируемое изображение абсолютны различны.

Предположим, что изображение А представляет собой цифровой образ размера . Большинство цифровых изображений представлены в цветовом пространстве RGB. Таким образом, каждый пиксель может быть интерпретирован в 3 измерениях: красном, зеленом и синем. Однако пространство RGB не очень удобно для анализа изображений. Следовательно, первый шаг предложенного метода состоит в преобразовании цветового 3-размерного пространства в 1-размерное серое пространство. Пусть  **есть изображение А в 1-размерном сером пространстве. Пусть В – тестовое изображение. Первым шагом в поиске изображений, похожих на В заключается в преобразовании изображения В в 1-размерное серое пространство . Следующим шагом является использование уравнения (1) для расчета индекса качества между изображениями и . Изображение с большим показателем индекса качества – изображение, которое наиболее схоже с тестовым. Поскольку диапазон Q от -1 до 1, то когда Q равно 1 можно сказать, что изображения и одинаковы. В модели индекса качества значение Q вычисляется с помощью среднего, дисперсии и корреляции между и . Среднее и дисперсия могут быть вычислены независимо. Но корреляцию нужно вычислять динамически, что требует больше времени. Поэтому формулу показателя качества (1) можно преобразовать. Новая формула может отфильтровать несовпадающие изображения заранее, чтобы ускорить поиск схожих изображений. В соответствии с определением модели, индекс качества является комбинацией трех факторов: потери корреляции, искажение яркости и контраст искажений. Измененное уравнение индекса качества:**

**Где**  – коэффициент корреляции, – искажение

яркости, – контраст искажения. Диапазон от -1 до 1; диапазон

от 0 до 1; диапазон от 0 до 1. Можно заметить, что только значение касается каждого пикселя  **и** . Величины  **, ,** . Пусть . Тогда индекс качества может быть представлен . Зная диапазон значений , любое значение может уменьшить значение Q, за исключением . Следовательно, можно заранее определить порог Т, чтобы отфильтровать маловероятные изображения, для которых . Другими словами, сравнивать необходимо те изображения, для которых .

# 1.3 Метод поиска похожих изображений с использованием индекса качества изображения

**Хэш-функции (или функции свертки) представляют собой функции преобразования, которые позволяют получить «отпечаток» фиксированной длинны для исходных данных. Классические криптографические хэш-алгоритмы (напр. MD5 или SHA-1) работают таким образом, чтобы для различных исходных данных, как мало бы они не отличались друг от друга, в результате получались максимально отличные хэш-значения. В отличие от классических хэш-функций, перцептуальные хэш-алгоритмы генерируют хеши, предназначенные для сравнения исходных данных. Хэш-значчения таких алгоритмов тем ближе, чем более схожи были исходные данные.**

**Таким образом, задача сравнения изображений сводится к вычислению хэш-значений этих изображений, и вычислению расстояния Хэмминга между ними. Чем меньше расстояние Хэмминга, тем более похожи данные изображения, расстояние 0, к примеру означает, что изображения, скорее всего, являются полностью идентичными. И напротив, чем более высокая дистанция, тем больше изображения отличаются друг от друга (к примеру, дистанция = 10 и больше для хэша размером 64 бита означает, что это, скорее всего, разные картинки).**



**Существуют различные алгоритмы вычисления перцептуальных хешей, которые различаются между собой чувствительностью к определенным типам искажений: изменению размера изображения, изменению соотношения сторон, цветовых характеристик (яркость, контраст, гамма), наложения водяных знаков и т.д.**

**Первый рассматриваемый хеш-алгоритм называется Average Hash. Данный хэш-алгоритм является очень быстрым, не чувствителен к масштабированию исходного изображения, сжатию или растяжению, изменению яркости или контрастности. Данный алгоритм основан на среднем значении, и, как следствие, является чувствительным к операциям, изменяющим среднее значение (например, изменение уровней или цветового баланса изображения).**

**Для построения aHash выполняются следующие шаги:**

**1. Уменьшение размера изображения. Вначале исходное изображение уменьшается до размера 8х8 пикселей, что даст хэш размером 64 бита (в приведенном примере используется блок размером 16х16). Размер изображения влияет на точность сравнения, и скорость работы алгоритма. Чем больше изображение, тем более высокая точность сравнения будет получена, но требует больше времени для расчета.**

**Масштабирование может выполняться без соблюдения пропорций, таким образом, полученный хэш будет соответствовать всем вариантам изображения с любым отношением сторон.**



**Если рассматривать изображение как дискретный сигнал, то в таком сигнале высокие частоты обеспечивают детализацию изображения, а низкие частоты показывают его структуру. Уменьшение размера изображения удаляет высокие частоты, и, таким образом, полученное изображение состоит преимущественно из низких частот, сохраняя общую структуру изображения.**

**2. Перевод в градации серого. Данный шаг позволяет в 3 раза уменьшить размер хэша, за счет уменьшения количества компонент с 3-х значений RGB до одного уровня серого. В примере данный шаг реализован вместе со следующим шагом.**



**3. Вычисление среднего значения. Далее вычисляется среднее значение по всем точкам изображения:**

**4. Упрощение изображения, при котором каждый пиксель заменяется значением 0, если он меньше рассчитанного на 3-м шаге среднего значения, и 1 если больше.**

**Из данного изображения получается цепочка бит (считыванием изображения построчно), из которой и строится хэш-значение:**

**Как было сказано выше, данный хэш не чувствителен к изменению масштаба изображения, изменению яркости или контрастности, но является чувствительным к изменению цветового баланса изображения изображения, так как это влияет на получаемое среднее значение. Данный недостаток исправляется следующим алгоритмом pHash.**

# 1.4 Метод поиска похожих изображений с использованием индекса качества изображения

**pHash во многом повторяет шаги aHash, но при этом добавляет еще один этап, на котором выполняется дискретное косинусное преобразование (DCT), которое позволяет разделить изображение на части разной степени «важности» (на гармоники дискретного сигнала), влияющие на качество изображения (данное преобразовании используется при кодировании изображений в формате JPEG).**

**Первые два шага алгоритма идентичны aHash, за тем исключением, что размер изображения выбирается больше (напр. 32х32). В данном случае данный шаг предназначен не для удаления высоких частот (что будет сделано позже), а для упрощения алгоритма DCT.**

**Следующим шагом изображение переводится в градации серого, и выполняется DCT-преобразование, которое разбивает изображение на набор базовых частот (в pHash этот алгоритм работает на блоке 32х32, в отличие от JPEG, который, как правило, использует блоки 8х8). Для 2D матриц данное преобразование выглядит следующим образом:**

**Где**

**M, N — размер входной матрицы**

**f(i, j) — значение матрицы (интенсивность пикселя) в строке i и колонке j**

**F(u, v) — DCT-коэффициент в строке k1 и колонке k2 матрицы DCT. Данные коэффициенты могут рассматриваться как весовые коэффициенты базисных функций. Например, для матрицы с размером 8×8 элементов существует 64 базовые функции, что продемонстрировано на изображении:**

**Выходной массив DCT содержит целые числа в диапазоне [-1024, 1023]. Для большинства изображений значимыми являются низкие частоты, которые будут расположены в левом верхнем углу DCT-матрицы. После расчета DCT-матрица сокращается, отбрасыванием незначимых (высоких) частот, в результате чего получается блок меньшего размера (8х8 или 16х16), и так же, как и в случае aHash-алгоритма, вычисляется среднее значение:**

**Дальнейшие шаги полностью соответствуют алгоритму aHash: значения получившийся матрицы сокращаются до значений 1 или 0 в зависимости от значения каждого пикселя (больше или меньше среднего значения), и на основе получившегося изображения строится хэш (в примере оба шага объединены в один):**

**Как и в aHash, значения pHash можно сравнивать между собой с помощью алгоритма расстояния Хэмминга. Такой вариант уже выдержит гамма-коррекцию или изменение гистограммы изображения.**

1. ****ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ****

# ****2.1 Структура проекта****

Разрабатываемое приложение предназначено для управления цветочными заказами. Для хранения информации будет использоваться база данных MySQL, которая будет содержать таблицы, изображенные на рисунке 2.1:

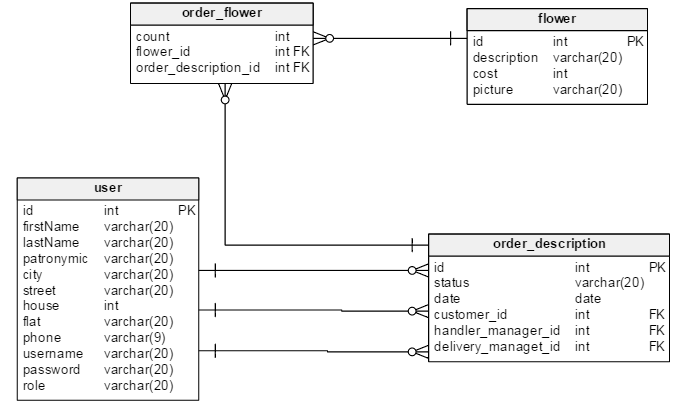
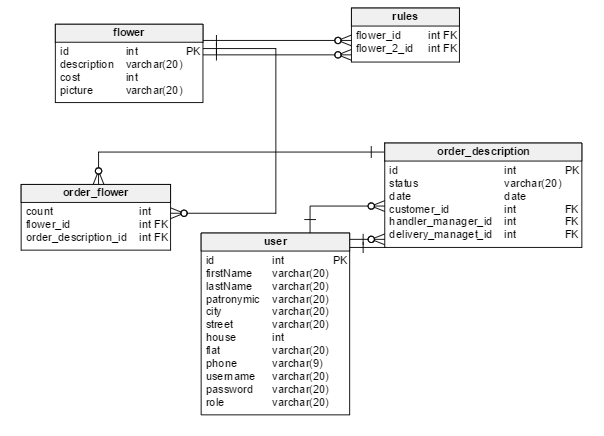


Рисунок 2.1 – Структура базы данных

Работа над созданием приложения начинается с создания каркаса приложения. Каркас включает в себя следующие этапы:

1. Подключение Hibernate и настройка соответствующих параметров [3].

Это демонстрирует следующий фрагмент кода:

<session-factory>

<property name="hibernate.connection.driver\_class"> com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url"> jdbc:mysql://localhost:3306/course\_work</property>

<property name="hibernate.connection.password"> 1234567 </property>

<property name="hibernate.connection.username"> root</property>

<property name="hibernate.dialect"> org.hibernate.dialect.MySQLDialect</property>

<property name="show\_sql">true</property>

<property name="hbm2ddl.auto">create</property>

<mapping class="bsu.model.Flower" />

<mapping class="bsu.model.Order" />

<mapping class="bsu.model.User" />

<mapping class="bsu.model.Rules" />

<mapping class="bsu.model.OrderElement" />

</session-factory>

Для работы с информацией, хранимой в базе данных, в приложении должны быть классы-сущности, которые отображаются на таблицы. Также необходима настройка классов модели для отображения в базу данных. Это возможно благодаря аннотациям:

@Entity

@Table(name = "user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

@Column(columnDefinition = "INT unsigned")

private Long id;

@Column(nullable = false)

private String firstName;

}

1. Настройка Spring Security.

Приложения включает несколько ролей пользователей: администратор, менеджер обработки заказа, менеджер доставки заказа и пользователь. Для того чтобы обезопасить действия пользователя и сервер от несанкционированного доступа, используется Spring Security. Следующий фрагмент кода демонстрирует работу Spring Security:

<http auto-config="true" use-expressions="true">

<access-denied-handler error-page="/#/login" />

<form-login

login-page="/#/login"

authentication-failure-url="/#/login?error"

default-target-url="/#/login?success"

always-use-default-target="true"/>

<logout invalidate-session="true"

logout-success-url="/#/login?logout"

delete-cookies="JSESSIONID, userInfo"/>

</http>

<authentication-manager>

<authentication-provider user-service-ref="customUserDetailsService">

</authentication-provider>

</authentication-manager>

1. Настройка Spring.

Spring настройка – конфигурационные файлы. Использование фреймворка Spring:

<context:component-scan base-package="bsu.service.impl"/>

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/course\_work"/>

<property name="username" value="root"/>

<property name="password" value="pavlovamarisha"/>

</bean>

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="persistenceUnitName" value="myPersistenceUnit"/>

<property name="packagesToScan" value="bsu.model"/>

<property name="jpaVendorAdapter">

<bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">

<property name="databasePlatform" value="org.hibernate.dialect.MySQL5InnoDBDialect"/>

<property name="showSql" value="false"/>

<property name="generateDdl" value="true"/>

</bean>

</property>

<property name="persistenceProvider">

<bean class="org.hibernate.jpa.HibernatePersistenceProvider"></bean>

</property>

</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory"/>

</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>

<jpa:repositories base-package="bsu.repository"/>

<bean id="customUserDetailsService"

class="bsu.service.impl.CustomUserDetailService">

</bean>

<task:annotation-driven />

1. Настройка AngularJS.

Настройка AngularJS включает в себя создание статической страницы:

<html>

<head> </head>

<body ng-app="OrderFlower">

<header>

</header>

<div class="content" id="content" ng-view></div>

</body>

</html>

А также создание модуля и контроллера:

var app = angular.module("OrderFlower", ['ngRoute, 'ngCookies']);

app.controller("authorizationController", function ($scope, $http, $location, $rootScope, $cookieStore) {});

1. Настройка Nexmo.

Для того чтобы использовать сервис Nexmo в Java, достаточно создать клиента, создать сообщение и указать клиенту послать сообщение. В программе это выглядит следующим образом:

NexmoSmsClient client = null;  
try {  
 client = new NexmoSmsClient(*API\_KEY*, *API\_SECRET*);  
} catch (Exception e) {  
 e.printStackTrace();  
}  
Random rand = new Random();  
int max = 999999;  
int min = 100000;  
int randomNum = rand.nextInt((max - min) + 1) + min;  
TextMessage message = new TextMessage(*SMS\_FROM*, phone, new Integer(randomNum).toString());  
SmsSubmissionResult[] results = null;  
try {  
 results = client.submitMessage(message);  
} catch (Exception e) {  
 e.printStackTrace();  
}

1. Подключение Google Maps.

Для использования карт необходимо указать на html место, в которое будет помещена карта:

<div id="map" style="height: 350px;"></div>  
<div id="class" ng-repeat="marker in markers | orderBy : 'title'">  
 <a href="#" ng-click="openInfoWindow($event, marker)">{{marker.title}}</a>  
</div>

И с помощью JavaScript инициализировать карту:

var mapOptions = {  
 zoom: 10,  
 center: new google.maps.LatLng(53.55, 27.33),  
 mapTypeId: google.maps.MapTypeId.TERRAIN  
};  
  
$scope.map = new google.maps.Map(document.getElementById('map'), mapOptions);  
$scope.markers = [];

var marker = new google.maps.Marker({  
 map: $scope.map,  
 position: new google.maps.LatLng(53, 27)  
});  
$scope.markers.push(marker);

# ****2.2 Технические возможности созданного приложения****

Созданное приложение для заказа цветов позволяет осуществлять следующие действия:

* Просматривать информацию о цветах и букетах, вводить необходимое количество экземпляров и делать заказ (рисунок 2.2)
* Создавать собственный вариант букета, путем указания наименования цветка и количества экземпляров. Перед оформлением заказа может быть показано сообщение о том, что некоторые из выбранных для букета цветов не сочетаются друг с другом. Это сообщение не препятствует оформлению заказа, а лишь рекомендует вам изменить свой выбор. Данный случай представлен на рисунке 2.3.
* Запрет на оформление заказа для неавторизированного пользователя
* Возможность создания нового пользователя с ролью «Пользователь». Клиенту необходимо заполнить информацию о себе, ввести адрес проживания, который будет отображен на карте, добавить номер телефона и информацию для авторизации (рисунок 2.4). Для того чтобы убедиться, что информацию вводит реальный человек, добавлена функция проверки по номеру телефон. Идея этой функции в том, что на указанный номер посылается шестизначный код, который пользователь должен ввести в окошко, представленное на рисунке 2.5
* При авторизации клиента, имеющего роль «Менеджер», появляется возможность просматривать никем не занятые заказы и заказы, ответственным за которые является авторизованный клиент. Данная возможность изображена на рисунке 2.6.

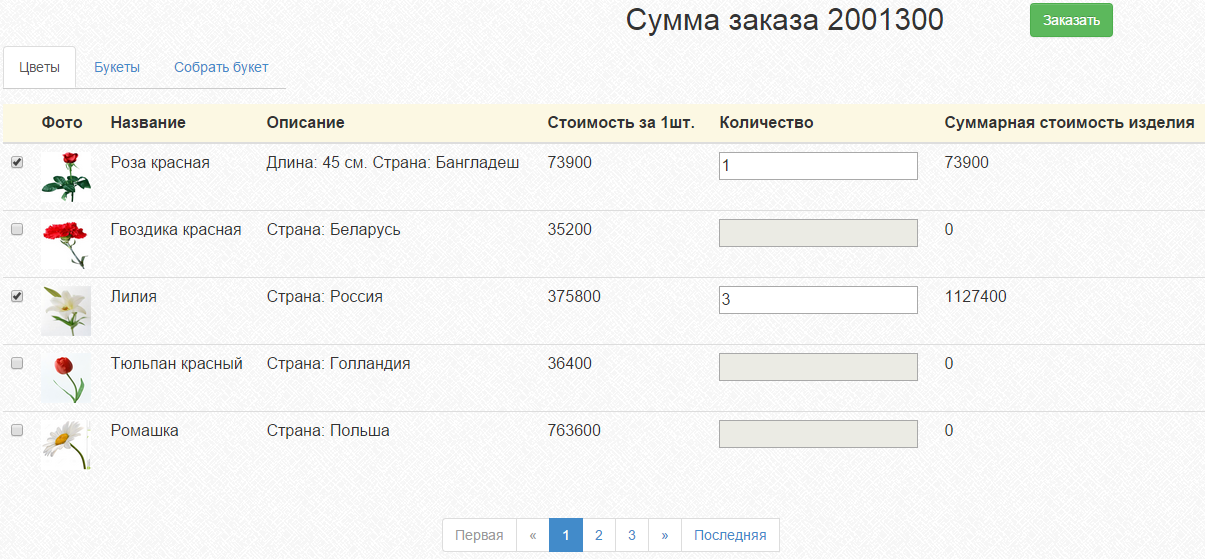


Рисунок 2.2 – Просмотр информации о цветах. Указание необходимого числа экземпляров цветка для заказа

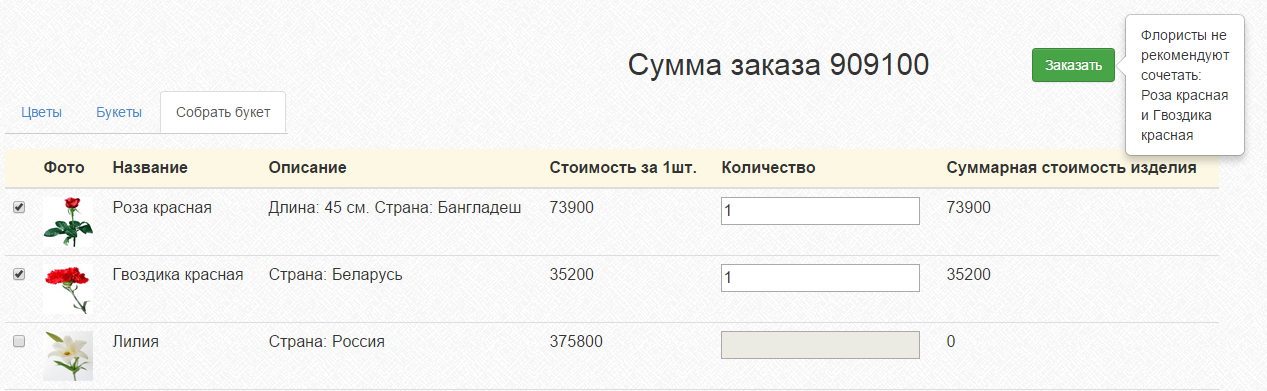
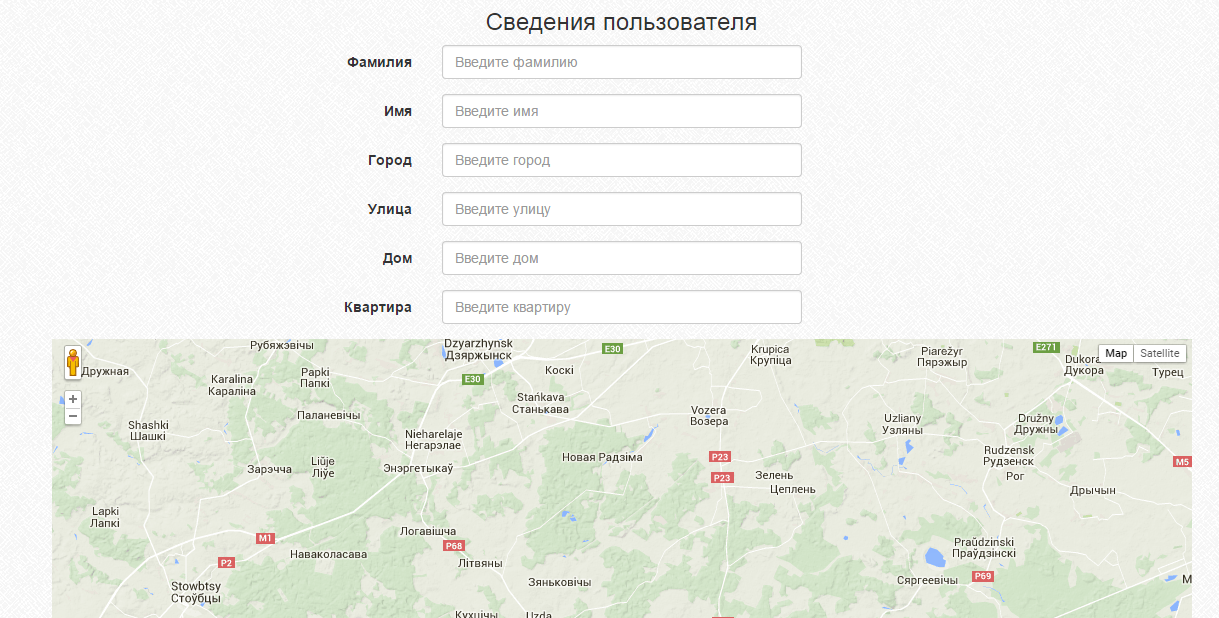
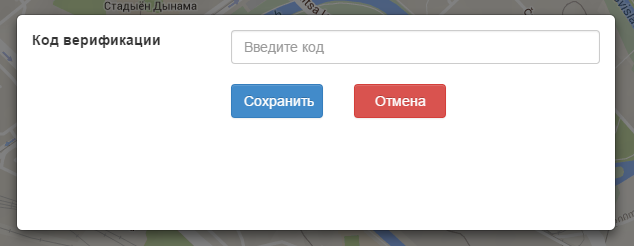


Рисунок 2.3 – Сообщение о не сочетаемости цветов



2.4 – Форма для заполнения при создании нового клиента



2.5 – Форма для ввода кода верификации

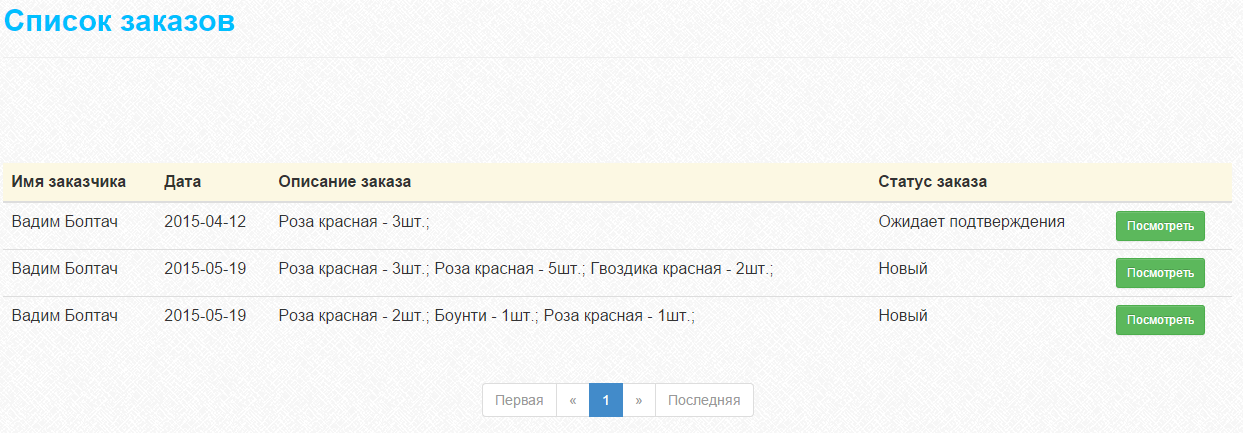


Рисунок 2.6 – Список заказов для авторизированного клиента

* Авторизированный с ролью «Менеджер» клиент имеет возможность просматривать список пользователей (рисунок 2.7) и смотреть информацию о них (рисунок 2.8). Также он обладает возможностью создавать пользователя с ролью «Менеджер». Менеджер может изменять статус заказа (рисунок 2.9).
* Каждый авторизированный пользователь может просматривать свои заказы. Но редактировать статус заказа клиент с ролью «Пользователь» не может. Этот вариант представлен на рисунке 2.10.

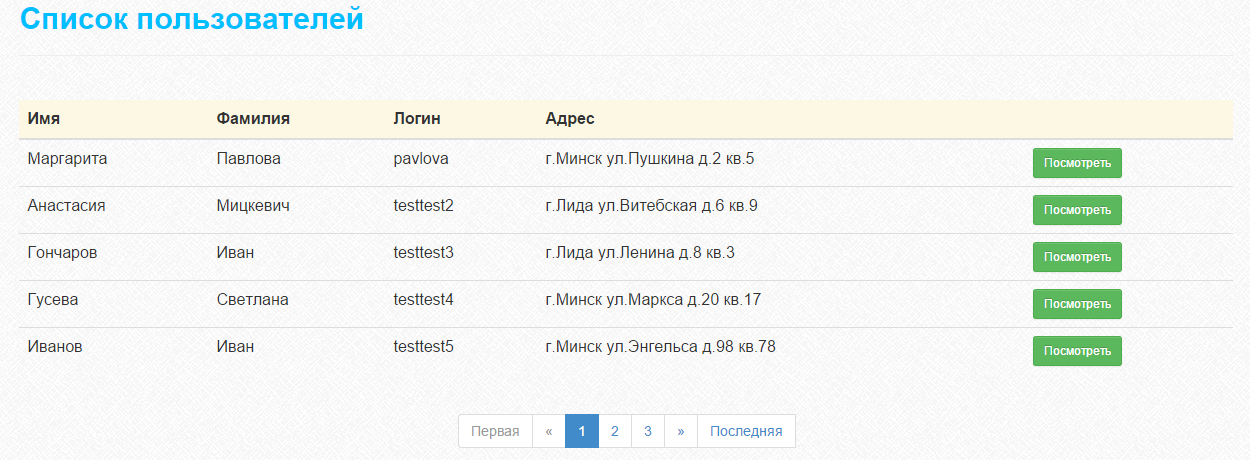


Рисунок 2.7 – Список пользователей

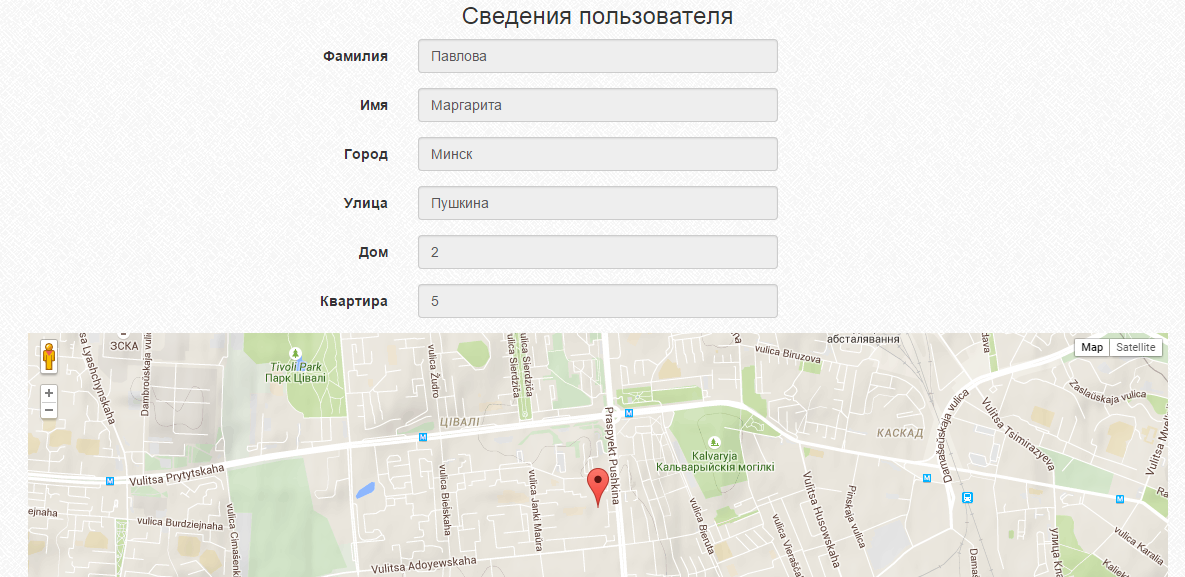


Рисунок 2.8 – Информация о пользователе

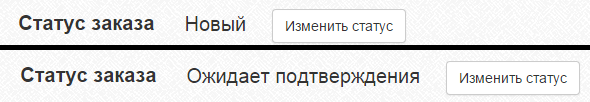


Рисунок 2.9 – Изменение статуса заказа

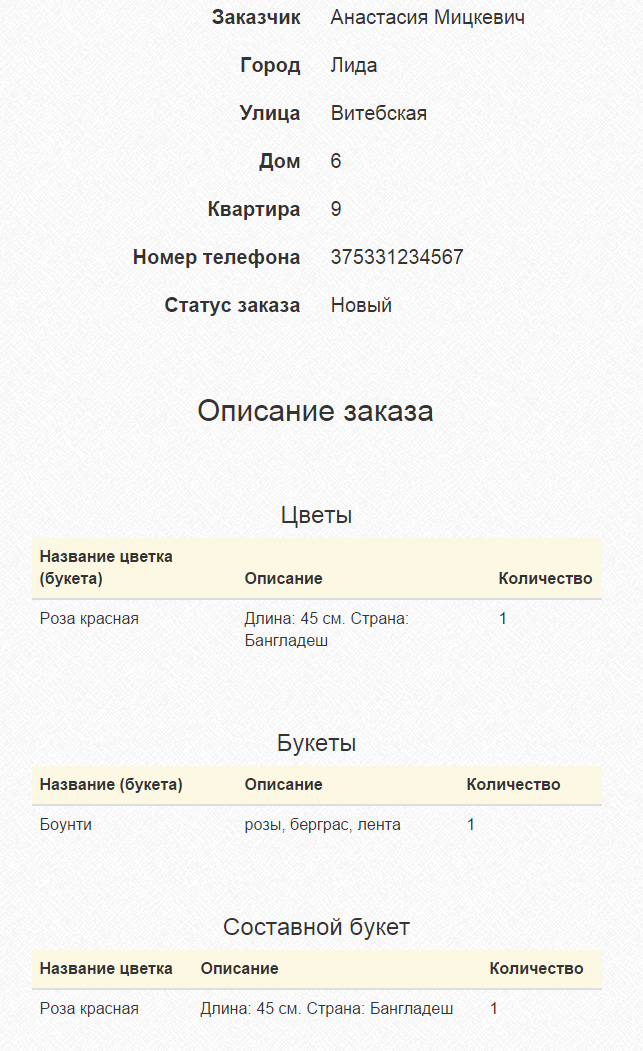


Рисунок 2.10 – Просмотр информации о заказе

Для каждого авторизированного пользователя представлена своя панель навигации (рисунок 2.11 и рисунок 2.12). Но для каждого пользователя в правой верхней части панели отображается его роль и логин, нажав на который можно редактировать свой профиль.



Рисунок 2.11 – Навигационная панель для клиента, авторизированного с ролью «Менеджер»



Рисунок 2.12 – Навигационная панель для клиента, авторизированного с ролью «Пользователь»

# ЗАКЛЮЧЕНИЕ

В результате проведенной работы были изучены методы создания WEB-приложений. Вследствие чего для проектирования WEB-приложения были выбраны технологии: фремворки Sping и AngularJS. Также были добавлены сервисы Nexmo и Google Maps.

В процессе проектирования WEB-приложения был реализован следующий набор функционала:

* Просмотр ассортимента цветов и букетов, составление собственного букета
* Вход в приложение
* Создание нового пользователя с разными ролями
* Проверка клиента на подлинность
* Оформление заказа
* Продвижение заказа от одного менеджера к другому путем изменения статуса заказа
* Просмотр личных заказов
* Просмотр доступных заказов в зависимости от роли пользователя
* Изменение собственного профиля
* Возможность просмотра профилей пользователей для менеджера
* Указание местоположения на карте

Используемый сервис Nexmo позволяет проводить проверку клиента на стадии создания аккаунта. Это позволяет выявить людей, которые ввели неверные данные и которые впоследствии могли бы нанести вред приложению. Сервис Google Maps позволяет курьерам с помощью карт узнавать местоположение клиента, что позволяет ускорить доставку цветов.

Реализованное приложение является актуальным решением проблемы возрастающего интереса со стороны пользователей к быстрому оформлению заказов на покупку цветов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Spring [Electronic resource]. – Mode of access : <http://www.spring.io>. – Date of access : 15.10.2014.
2. AngularJS [Electronic resource] / AngularJS – Superheroic JavaScript MVW Framework. – Mode of access : <https://www.angularjs.com>. – Date of access : 9.11.2014.
3. Hibernate [Electronic resource] / Hibernate. Everything data. - Hibernate. – Mode of access : <http://hibernate.org/>. – Date of access : 13.10.2014.
4. Walls C. Spring in action / C. Walls. – 3 edition. – New York : Manning Publications Co., 2011. – 400 p.
5. Nexmo [Electronic resource] / Nexmo – APIs for SMS, Voice and Phone Verification. - Nexmo. – Mode of access : <http://nexmo.org/>. – Date of access : 21.03.2015.
6. Google Maps [Electronic resource] / Google Maps API – Google Developers . – Google Maps. – Mode of access : <https://developers.google.com/maps>. – Date of access : 25.03.2015.