Оглавление

[Введение 3](#_Toc451102044)

[Цели и задачи работы 4](#_Toc451102045)

[Глава 1. Основные сведения об облачных технологиях 5](#_Toc451102046)

[1.1 Общая информация 5](#_Toc451102047)

[1.2 Модели обслуживания 6](#_Toc451102048)

[1.3 Основные свойства облачных технологий 8](#_Toc451102049)

[1.4 Преимущества и недостатки облачных сервисов 8](#_Toc451102050)

[1.5 Основные поставщики облачных услуг 11](#_Toc451102051)

[Глава 2. Реализация проекта 14](#_Toc451102052)

[2.1 Основная идея и возможности системы 14](#_Toc451102054)

[2.2 Архитектура проекта 15](#_Toc451102055)

[2.3 Использование Google Cloud Storage 16](#_Toc451102056)

[2.4 Основные используемые фреймворки 19](#_Toc451102057)

[2.4.1. Spring 19](#_Toc451102058)

[2.4.2. Hibernate 22](#_Toc451102059)

[2.4.3. Apache Tiles 24](#_Toc451102060)

[2.5 Apache Maven 25](#_Toc451102061)

[Глава 3. Перспективы развития 27](#_Toc451102062)

[Заключение 28](#_Toc451102063)

[Список использованных источников 29](#_Toc451102064)

[Приложения 30](#_Toc451102065)

[Приложение 1. Реестр вариантов использования 30](#_Toc451102066)

[Приложение 2. Схема базы данных. 31](#_Toc451102067)

[Приложение 3. Скриншоты основных страниц 32](#_Toc451102068)

Введение

В настоящее время происходит быстрое развитие облачных технологий. Существует множество различных облачных сервисов, например виртуальные машины, хранилища, базы данных. Они предоставляются различными компаниями, такими, как Google, Microsoft Azure, Amazon. И для большинства из них разработаны готовые API, которые значительно облегчают процесс разработки.

Облачные сервисы дают большие преимущества при их использовании, среди которых можно выделить надежность, доступность, экономичность, масштабируемость.

Их легко подключать и тем самым они помогают быстро получить все необходимые для системы ресурсы. При этом не нужно закупать оборудование для функционирования системы, например компьютеры, жесткие диски и др., что существенно снижает расходы.

Облачные сервисы позволяют не быть привязанным к конкретному физическому устройству или аппаратной платформе и получать доступ к своим данным в любой точке мира и с любого устройства, имеющего доступ в интернет. При этом оплата осуществляется только за реально использованные ресурсы.

Таким образом, в связи с все большим развитием облачных технологий, я решила изучить один из их видов, а именно Cloud Storage, и использовать возможности данного сервиса в разрабатываемом веб-проекте.

Цели и задачи работы

Моей целью было изучение возможностей и принципов работы с Cloud Storage, а также некоторых фреймворков для разработки веб-приложений.

Задача моей работы заключалась в реализации сайта с использованием Google Cloud Storage, а также таких фреймворков, как Spring, Hibernate, Apache Tiles.

Глава 1. Основные сведения об облачных технологиях

* 1. Общая информация

Идея облачных вычислений появилась еще в 1960 году, когда Джон Маккарти высказал предположение, что когда-нибудь компьютерные вычисления будут производиться с помощью «общенародных утилит». Считается, что идеология облачных вычислений получила популярность с 2007 года благодаря быстрому развитию каналов связи и стремительно растущим потребностям пользователей.

Под облачными вычислениями (от англ. cloud computing, также используется термин «облачная (рассеянная) обработка данных») обычно понимается предоставление пользователю компьютерных ресурсов и мощностей в виде интернет-сервиса. Таким образом, вычислительные ресурсы предоставляются пользователю в «чистом» виде, и пользователь может не знать, какие компьютеры обрабатывают его запросы, под управлением какой операционной системы это происходит и т.д.

В настоящее время крупные вычислительные облака состоят из тысяч серверов, размещенных в центрах обработки данных (ЦОД). Они обеспечивают ресурсами десятки тысяч приложений, которые одновременно используют миллионы пользователей. [1]

Основное отличие «облачного» программного решения от обычного в том, что вся информация сохраняется не на пользовательском жестком диске, а на удаленном сервере. Аналогично с производимыми операциями: они нагружают не персональный компьютер или ноутбук, а мощности серверов компании, предоставляющей то или иное приложение. Пользователь же получает лишь результат, отправляемый на монитор через интернет.

Существует большое количество сервисов, среди которых можно выделить сервисы для создания и управления виртуальными машинами, различные хранилища, сервисы для развертывания приложения в облаке, для работы с Big Data и др.

Причины возрастающей популярности облачных технологий понятны: возможности их применения очень разнообразны и позволяют экономить как на обслуживании и персонале, так и на инфраструктуре.

* 1. Модели обслуживания

Инфраструктура как услуга (IaaS, Infrastructure-as-a-Service) предоставляется как возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетей и другими фундаментальными вычислительными ресурсами, например, потребитель может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы, платформенное и прикладное программное обеспечение. Потребитель может контролировать операционные системы, виртуальные системы хранения данных и установленные приложения, а также ограниченный контроль набора доступных сервисов (например, межсетевой экран, DNS). Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, типов используемых операционных систем, систем хранения осуществляется облачным провайдером.[2]

Примерами услуг инфраструктуры служат IBM SmartCloud Enterprise, VMWare, Amazon EC2, Windows Azure, Google Cloud Storage, Parallels Cloud Server и многие другие.

Платформа как услуга (PaaS, Platform-as-a-Service) — модель, когда потребителю предоставляется возможность использования облачной инфраструктуры для размещения базового программного обеспечения для последующего размещения на нём новых или существующих приложений (собственных, разработанных на заказ или приобретённых тиражируемых приложений). В состав таких платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения — системы управления базами данных, связующее программное обеспечение, среды исполнения языков программирования — предоставляемые облачным провайдером. Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения осуществляется облачным провайдером, за исключением разработанных или установленных приложений, а также, по возможности, параметров конфигурации среды (платформы).[2]

Примерами услуг платформы служат IBM SmartCloud Application Services, Amazon Web Services, Windows Azure, Boomi, Cast Iron, Google App Engine и другие.

Программное обеспечение как услуга (SaaS, Software-as-a-Service) — модель, в которой потребителю предоставляется возможность использования прикладного программного обеспечения провайдера, работающего в облачной инфраструктуре и доступного из различных клиентских устройств или посредством тонкого клиента, например, из браузера (например, веб-почта) или интерфейс программы. Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения, или даже индивидуальных возможностей приложения (за исключением ограниченного набора пользовательских настроек конфигурации приложения) осуществляется облачным провайдером.[2]

Примерами SaaS являются Gmail, Google Docs, Netflix, Photoshop.com, Acrobat.com, Intuit QuickBooks Online, IBM LotusLive, Unyte, Salesforce.com, Sugar CRM и WebEx. Значительная часть растущего рынка мобильных приложений также является реализацией SaaS.

* 1. Основные свойства облачных технологий

Национальный Институт стандартов и технологий NIST (National Institute of Standards and Technology, USA) в своем документе “The NIST Definition of Cloud Computing” определяет следующие характеристики облаков:

– возможность в высокой степени автоматизированного самообслуживания системы со стороны провайдера;

– наличие системы Broad Network Access;

– сосредоточенность ресурсов на отдельных площадках для их эффективного распределения;

* быстрая масштабируемость (ресурсы могут неограниченно выделяться и высвобождаться с большой скоростью в зависимости от потребностей);

– управляемый сервис (система управления облаком автоматически контролирует и оптимизирует выделение ресурсов).[1]

* 1. Преимущества и недостатки облачных сервисов

Самым главным недостатком является полная зависимость от поставщика этих услуг. Фактически предприятие (пользователь) оказывается заложником провайдера сервисов и провайдера доступа в сеть Интернет.[1]

Любой владелец бизнеса чувствует себя более уверенно, когда вся необходимая информация физически находится на территориально близком носителе. Когда тот же жесткий диск можно, образно говоря, «потрогать» и убедиться, что данные в безопасности. Тем не менее, эта безопасность весьма условна: в крупном серверном центре, где работают лучшие специалисты в этой области, система защиты и резервного копирования данных всегда лучше, чем в отдельной компании, имеющей иную сферу деятельности.

Второй недостаток «облачных» моделей программного обеспечения — невозможность проконтролировать конфиденциальность в отношении данных «в облаке». [3]

Среди основных преимуществ, связанных с использованием облачных технологий, можно выделить следующие:

Доступность. Доступ к информации, хранящейся на облаке, может получить каждый, кто имеет компьютер, планшет, любое мобильное устройство, подключенное к сети интернет.

Мобильность. У пользователя нет постоянной привязанности к одному рабочему месту. Из любой точки мира менеджеры могут получать отчетность, а руководители — следить за производством.

Экономичность. Одним из важных преимуществ называют уменьшенную затратность. Пользователю не надо покупать дорогостоящие, большие по вычислительной мощности компьютеры и ПО, а также он освобождается от необходимости нанимать специалиста по обслуживанию локальных IT-технологий. При этом плата только за использованные ресурсы.

Арендность. Пользователь получает необходимый пакет услуг только в тот момент, когда он ему нужен, и платит, собственно, только за количество приобретенных функций. При этом услуги легко подключаются.

Гибкость. Все необходимые ресурсы предоставляются провайдером автоматически.

Высокая технологичность. Большие вычислительные мощности, которые предоставляются в распоряжение пользователя, которые можно использовать для хранения, анализа и обработки данных.

Надежность. Некоторые эксперты утверждают, что надежность, которую обеспечивают современные облачные вычисления, гораздо выше, чем надежность локальных ресурсов, аргументируя это тем, что мало предприятий могут себе позволить приобрести и содержать полноценный ЦОД.[4]

Масштабируемость. Использование локальных ресурсов вызывает некоторые трудности, например, в случае необходимости расширения пространства. При использовании облачных технологий происходит оперативное выборочное наращивание мощности.

Скорость досупа. Данные можно размещать на серверах, где большее число пользователей.

Упрощение аппаратного обеспечения. Аппаратное обеспечение может быть сильно упрощено при обработке данных и хранении информации в удаленных центрах данных. Все эти проблемы почти полностью перекладываются на провайдера услуг.

Стандартизация ПО. Подход, связанный с использованием облачных технологий, позволяет стандартизировать ПО, даже если на компьютерах предприятия установлены разные операционный системы (Windows, Linux, MacOS и т.п.). [1]

Также можно перечислить еще следующие преимущества:

* Круглогодичная безотказно устойчивая работа компании.
* Повышение безопасности за счет консолидации вычислительных ресурсов, сведение до минимума «человеческого фактора» и подотчетность пользователей к несанкционированному доступу в систему и скачиванию данных.
* Шифрование данных, криптография и защита данных.
* Использование API. Поставщики предлагают использование готовое API для разных языков программирования для работы с сервисами.
* Снижение временных затрат, например затрат на внедрение и оперативное перераспределение ресурсов.

Google Apps для бизнеса выделяет эти же преимущества, только добавляет, что при использовании их cloud computing компания защищает окружающую среду, объясняя это тем, что службы Apps работают на базе центров обработки данных Google, отличающихся сверхнизким энергопотреблением, поэтому углеродоемкость и энергозатраты при их использовании будут значительно ниже при использовании локальных серверов.

* 1. Основные поставщики облачных услуг

На сегодняшний день существует большое множество поставщиков облачных платформ, хранилищ и ПО.

В настоящий момент основными поставщиками облачной инфраструктуры считаются Amazon, Google и Microsoft. У каждой из компаний имеется целая линейка предоставляемых услуг, среди которых для сравнения выбраны наиболее популярные.

Виртуальны машины

Google App Engine – сервис хостинга сайтов и web-приложений на серверах Google. Бесплатно предоставляется до 1 Гб дискового пространства, 10 Гб входящего трафика в день, 10 Гб исходящего трафика в день, 200 миллионов гигациклов CPU в день и 2 000 операций отправления электронной почты в день. Приложения, разворачиваемые на базе App Engine, должны быть написаны на Python, Java либо Go. Предлагается набор API для сервисов хранилища datastore API (BigTable) аккаунтов Google, набор API для загрузки данных по URL, электронной почты и т.д.

Платформа Google конкурирует с аналогичными сервисами от Amazon, которые предоставляют возможность размещать файлы и веб-приложения, используя свою инфраструктуру. В отличие от многих обычных размещений приложений на виртуальных машинах, таких как Amazon EC2, платформа App Engine тесно интегрирована с приложениями.

Amazon Elastic Compute Cloud (Amazon EC2) – веб-сервис, предоставляющий вычислительные мощности в облаке. Он дает пользователям полный контроль над вычислительными ресурсами, а также доступную среду для работы. Amazon EC2 позволяет пользователям создать Amazon Machine Image (AMI), который будет содержать их приложения, библиотеки, данные и связанные с ними конфигурационные параметры, или использовать заранее настроенные шаблоны образов для работы Amazon S3.

Windows Azure Compute – компонент, реализующий вычисления на платформе Windows Azure, предоставляет среду выполнения на основе ролевой модели.[1]

Хранилища

Google Cloud Storage – сервис хостинга файлов, основанный на IaaS. Все файлы, которые записываются или перезаписываются на серверы, автоматически шифруются по алгоритму AES-128.

Amazon Simple Storage Service (Amazon S3) – веб-служба, предлагаемая Amazon Web Services, предоставляющая возможность для хранения и получения любого объема данных, в любое время из любой точки сети, так называемый файловый хостинг. В марте 2012 года компания Nasuni провела опыт, в течение которого поочередно передавала массивный объем данных (12 Тб) из одного облачного сервиса в другой. В эксперименте участвовали наиболее рейтинговые облака: Amazon S3, Windows Azure и Rackspace. К удивлению исследователей, скорость передачи данных сильно отличалась в зависимости от того, какое облако принимало данные. Самый лучший показатель скорости записи данных оказался у Amazon S3, передача данных из двух других сервисов занимала всего 4–5 часов, в то время как передача данных в Rackspace заняла чуть меньше недели, а в Windows Azure – 40 часов.

Windows Azure Storage – компонент хранилища, предоставляющий масштабируемое хранилище.[1]

Все эти платформы (Windows Azure, Google, Amazon) имеют API, построенное на REST, HTTP и XML, что позволяет разработчикам использовать облачные сервисы с любой операционной системой, устройствами и платформами.

Глава 2. Реализация проекта

1. 1. Основная идея и возможности системы

Основная идея проекта - создание событий, которые уже прошли или еще будут в будущем. События могут быть совершенно разного типа, например концерты, выставки, просто встреча друзей и др.

Вы, как пользователь, добавляете основную информацию о событии: название, тип, описание, даты и теперь другие пользователи смогут его просматривать.

Основная цель создания события - делиться своими фотографии, которые были на нем сделаны. Пользователь, который участвовал в этом событии и у него есть фотографии, может запросто ими поделится и при этом просматривать чужие. Таким образом, данный проект позволяет объединить в одном месте большое количество фотографий события и просматривать их в любое время.

Реестр вариантов использования представлен в приложении 1.

Главная цель создания проекта - изучение на его примере использования облачного хранилища.

Проект подразумевает огромное количество фотографий, при этом заранее неизвестен их объем. Поэтому использование локальных ресурсов не является целесообразным и эффективным.

Проект и база данных размещены на облачной виртуальной машине. Я использовала сервис Google Cloud Compute Engine. Использование облачной виртуальной машины позволяет подключаться к ней из любой точки мира и не быть привязанным к работе конкретного компьютера.

Код проекта доступен по следующей ссылке: <https://bitbucket.org/veronica_haritonova/images_store>

* 1. Архитектура проекта

Система спроектирована на основе шаблона MVC. Model-view-controller — схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

Среди слоев модели можно выделить dao слой для работы с базой данных и сервисный слой, в котором происходит обращение к dao-классам.

Таким образом, архитектура проекта может быть представлена схемой, изображенной на рис.1

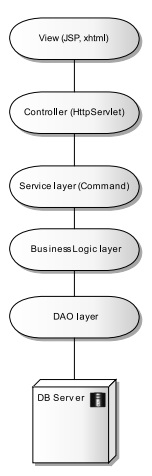


рис.1. Архитектурное проектирование. Взаимодействие слоев.

Схема базы данных представлена в приложении 2.

Серверная часть написана на языке Java с использованием таких фреймворков, как Spring Core, Spring MVC, Spring Security, Hibernate. Данные фреймворки в значительной степени облегчают процесс создания веб-приложения и общения с базой данных.

Для генерации страниц из частей использовался Apache tiles, для оформления - CSS, разметки - HTML, для динамики веб-страниц и отправки запросов - JS, JQuery, Ajax.

Реализована интеграция с Google Cloud Storage для хранения фотографий пользователей.

* 1. Использование Google Cloud Storage

Google Cloud Storage позволяет хранить данные различных форматов и управлять ими. При этом гарантируется высокая надежность, производительность и доступность. Google Cloud Storage можно использовать для широкого круга сценариев, включая обслуживание контента веб-сайта, хранения данных как архива и восстановления при катастрофах и др.[5]

В проекте я использовала API, предоставляемое Google, а именно библиотеку google-api-services-storage. Она содержит готовые классы, необходимые для работы с хранилищем.

Для работы с Google Cloud Storage сначала необходимо получить объект для доступа приложения к данным. Для этого используются следующие методы:

*/\*\**

*\* Creates an authorized Credential object.*

*\** ***@return*** *an authorized Credential object.*

*\** ***@throws*** *IOException*

*\*/*

**public static** GoogleCredential getCredential() **throws** IOException, GeneralSecurityException {

GoogleCredential credential = **new** GoogleCredential.Builder().setTransport(*HTTP\_TRANSPORT*)

.setJsonFactory(***JSON\_FACTORY***)

.setServiceAccountId(***SERVICE\_ACCOUNT\_EMAIL***)

.setServiceAccountScopes(StorageScopes.*all*())

.setServiceAccountPrivateKeyFromP12File(**new** File(***PRIVATE\_KEY\_LOCATION***))

.build();

**return** credential;

}

*/\*\**

*\* Build and return an authorized Drive client service.*

*\** ***@return*** *an authorized Drive client service*

*\** ***@throws*** *IOException*

*\*/*

**public static** Storage getService() **throws** IOException, GeneralSecurityException {

**if** (**null** == *storageService*) {

GoogleCredential credential = *getCredential*();

*storageService* = **new** Storage.Builder(*HTTP\_TRANSPORT*, ***JSON\_FACTORY***, credential)

.setApplicationName(***APPLICATION\_NAME***).build();

}

**return** *storageService*;

}

Объект класса Storage используется для сохранения и удаления изображений. Данный функционал реализован в классе CloudFileServiceImpl, который представлен ниже.

@Service

**public class** CloudFileServiceImpl **implements** ICloudFileService {

**private static** Logger *logger* = Logger.*getLogger*(CloudFileServiceImpl.**class**);

*/\*\**

*\* Uploads data to an object in a bucket.*

*\**

*\** ***@param name*** *the name of the destination object.*

*\** ***@param contentType*** *the type of the data.*

*\** ***@param stream*** *the data*

*\** ***@param bucketName*** *the name of the bucket to create the object in.*

*\*/*

**public void** uploadStream(

String name, String contentType, InputStream stream, String bucketName)

**throws** ServiceException {

**try** {

InputStreamContent contentStream = **new** InputStreamContent(contentType, stream);

StorageObject objectMetadata = **new** StorageObject()

.setName(name)

.setAcl(Arrays.*asList*(

**new** ObjectAccessControl().setEntity(**"allUsers"**).setRole(**"READER"**)));

Storage client = *getService*();

Storage.Objects.Insert insertRequest = client.objects().insert(

bucketName, objectMetadata, contentStream);

insertRequest.execute();

} **catch** (IOException | GeneralSecurityException e) {

*logger*.error(e);

**throw new** ServiceException(e);

}

}

*/\*\**

*\* Deletes an object in a bucket.*

*\**

*\** ***@param path*** *the path to the object to delete.*

*\** ***@param bucketName*** *the bucket the object is contained in.*

*\*/*

**public void** deleteObject(String path, String bucketName)

**throws** ServiceException {

**try** {

Storage client = *getService*();

client.objects().delete(bucketName, path).execute();

} **catch** (IOException | GeneralSecurityException e) {

*logger*.error(e);

**throw new** ServiceException(e);

}

}

}

Управление ресурсами осуществляется через Google Cloud Console, где отображается вся информация об используемых ресурсах и сервисах.

* 1. Основные используемые фреймворки

### 2.4.1. Spring

Spring Framework — универсальный [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) [с открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для [Java](https://ru.wikipedia.org/wiki/Java)-платформы.

Spring Framework обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики. [6]

Spring Framework может быть рассмотрен как коллекция меньших фреймворков. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. Эти фреймворки делятся на структурные элементы типовых комплексных приложений: [Inversion of Control](https://ru.wikipedia.org/wiki/Inversion_of_Control)-контейнер, фреймворк [аспектно-ориентированного программирования](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), фреймворк доступа к данным, фреймворк управления [транзакциями](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), фреймворк [MVC](https://ru.wikipedia.org/wiki/Model-view-controller), фреймворк [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) и [авторизации](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) ([Spring Security](https://ru.wikipedia.org/wiki/Spring_Security)), фреймворк работы с сообщениями и др.[6]

Центральной частью Spring Framework является контейнер [Inversion of Control](https://ru.wikipedia.org/wiki/Inversion_of_Control), который предоставляет средства конфигурирования и управления объектами Java с помощью [отражения](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Контейнер отвечает за управление жизненным циклом объекта: создание объектов, вызов методов инициализации и конфигурирование объектов путем связывания их между собой.

Объекты, создаваемые контейнером, также называются управляемыми объектами (beans). Обычно конфигурирование контейнера осуществляется путем загрузки XML-файлов, содержащих определение bean’ов и предоставляющих информацию, необходимую для создания bean’ов. Ниже представлен пример создания объекта dataSource, использующийся в проекте.

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

<**property name="driverClassName" value="${db.driver}"**/>

<**property name="url" value="${db.path}"**/>

<**property name="username" value="${db.login}"**/>

<**property name="password" value="${db.password}"**/>

</**bean**>

Другим способом создания объектов является использование аннотаций.

Для реализации MVC используются следующие аннотации : @Controller, @Service, @Repository. При этом при конфигурации необходимо указать строчку: <**mvc:annotation-driven**/>

Для работы с пользователями использовался Spring Security. Получение данных о пользователях реализовано в классе AuthUserService:

@Service

**public class** AuthUserService **implements** UserDetailsService {

**private** AuthUserDAOImpl **authUserDAO**;

@Override

**public** UserDetails loadUserByUsername(String s) **throws** UsernameNotFoundException {

AuthUser user = **null**;

**try** {

user = **authUserDAO**.getUserByUserName(s);

**if** (user == **null**) {

**throw new** UsernameNotFoundException(**"User not found"**);

}

} **catch** (DAOException e) {

e.printStackTrace();

}

**return** user;

}

**public** AuthUserDAOImpl getAuthUserDAO() {

**return authUserDAO**;

}

**public void** setAuthUserDAO(AuthUserDAOImpl authUserDAO) {

**this**.**authUserDAO** = authUserDAO;

}

}

Конфигурация данного класса прописана в xml файле:

<**beans:bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"**/>

<**global-method-security secured-annotations="enabled"**/>

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

<**access-denied-handler error-page="/login?accessDenied"**/>

<**form-login**

**login-page="/login"**

**default-target-url="/events"**

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

**username-parameter="login"**

**password-parameter="password"**/>

<**logout logout-success-url="/login?logout"**/>

</**http**>

<**beans:bean id="authUserDAO"**

**class="com.haritonova.imagestore.dao.impl.AuthUserDAOImpl"**>

<**beans:property name="dataSource" ref="dataSource"**/>

</**beans:bean**>

<**beans:bean id="authUserService"**

**class="com.haritonova.imagestore.service.impl.AuthUserService"**>

<**beans:property name="authUserDAO" ref="authUserDAO"**/>

</**beans:bean**>

<**authentication-manager**>

<**authentication-provider user-service-ref='authUserService'**>

<**password-encoder ref="passwordEncoder"**/>

</**authentication-provider**>

</**authentication-manager**>

Для ограничения доступа пользователям к проекту, используется аннотация @Secured.

### 2.4.2. Hibernate

[Hibernate](http://ru.wikipedia.org/wiki/Hibernate_(%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0)) является ORM-решением для языка [Java](http://ru.wikipedia.org/wiki/Java). Данная технология не только заботится о связи Java классов с таблицами базы данных (и типов данных Java в типы данных [SQL](http://ru.wikipedia.org/wiki/Sql)), но также предоставляет средства для автоматического построения запросов и извлечения данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL и [JDBC](http://ru.wikipedia.org/wiki/Jdbc) кода. Hibernate генерирует SQL вызовы и освобождает разработчика от ручной обработки результирующего набора данных и конвертации объектов, сохраняя приложение портируемым во все SQL базы данных.[7]

На мой взгляд, самое главное преимущество Hibernate в том, что он делает код независимым от типа сервера баз данных. При этом он позволяет программисту не думать об устройстве и особенностях использования какого-то конкретного типа базы данных. Можно использовать один и тот же код, как для Oracle, так и я для MySql.

Для мэппинга полей в базе данных и классах я использовала аннотации:

@Entity

@Table(name = **"country"**)

**public class** Country **implements** Serializable {

@Id

@GeneratedValue(generator = **"country\_seq\_gen"**, strategy = GenerationType.***SEQUENCE***)

@SequenceGenerator(name = **"country\_seq\_gen"**, sequenceName = **"country\_seq"**, allocationSize = 2)

@Column(name = **"country\_id"**)

**private** Long **id**;

@Column(name = **"key\_name"**)

@NotEmpty

@NotNull

@Size(max = 50)

**private** String **countryName**;

*//other methods*

}

При этом значительно упрощается получение объекта, например по его id:

@Override

**public** Country get(Long countryId) **throws** DAOException {

**if** (countryId != **null**) {

Session session = **sessionFactory**.getCurrentSession();

**return** session.get(Country.**class**, countryId);

} **else** {

**return null**;

}

}

### 2.4.3. Apache Tiles

Apache Tiles - это фреймворк для построения и использования шаблонов страниц. Первоначально разрабатывался, чтобы упростить разработку интерфейсов пользователей в веб-приложениях, но его использование не ограничено средой JavaEE.   
 Tiles позволяет разработчикам определять фрагменты страниц, которые могут быть собраны в отдельную страницу во время исполнения. Эти фрагменты могут использоваться как включения в основную страницу, чтобы сократить количество повторов общих элементов страниц, или встраивания с другими элементами, чтобы разработать серию шаблонов, которые можно многократно использовать. Данные шаблоны можно использовать повсеместно в приложении.[8]

Конфигурация шаблонов и страниц осуществляется в xml файлах.

Конфигурация шаблона:

<**tiles-definitions**>

<**definition name="defaultTemplate" template="/WEB-INF/pages/layouts/user-layout.jsp"**>

<**put-attribute name="header" value="/WEB-INF/pages/layouts/header.jsp"**/>

<**put-attribute name="menu" value="/WEB-INF/pages/layouts/menu.jsp"**/>

<**put-attribute name="content" value=""**/>

<**put-attribute name="footer" value="/WEB-INF/pages/layouts/footer.jsp"**/>

</**definition**>

<**definition name="loginTemplate" template="/WEB-INF/pages/layouts/login-layout.jsp"**>

<**put-attribute name="header" value="/WEB-INF/pages/layouts/header.jsp"**/>

<**put-attribute name="content" value=""**/>

<**put-attribute name="footer" value="/WEB-INF/pages/layouts/footer.jsp"**/>

</**definition**>

</**tiles-definitions**>

Конфигурация страниц, генерируемых с помощью шаблонов:

<**tiles-definitions**>

<**definition name="login" extends="loginTemplate"**>

<**put-attribute name="title" value="Login"**/>

<**put-attribute name="content" value="/WEB-INF/pages/content/login.jsp"**/>

</**definition**>

<**definition name="successRegister" extends="loginTemplate"**>

<**put-attribute name="title" value="Login"**/>

<**put-attribute name="content" value="/WEB-INF/pages/content/successRegister.jsp"**/>

</**definition**>

<**definition name="\*" extends="defaultTemplate"**>

<**put-attribute name="title" value="Event"**/>

<**put-attribute name="content" value="/WEB-INF/pages/content/{1}.jsp"**/>

</**definition**>

</**tiles-definitions**>

* 1. Apache Maven

В качестве инструмента для сборки проекта я использовала Apache Maven. Maven - это инструмент для сборки Java проекта: компиляции, создания jar или war, тестирования, создания дистрибутива программы, генерации документации.[9]

При использовании Maven главным файлом является pom.xml, в котором прописываются основные зависимости, плагины, конфигурация проекта и информация о нем:

<**dependency**>

<**groupId**>com.google.apis</**groupId**>

<**artifactId**>google-api-services-storage</**artifactId**>

<**version**>v1-rev56-1.21.0</**version**>

</**dependency**>

На мой взгляд, Maven значительно упрощает работу с различными библиотеками. При этом у всех работающих над проектом разработчиков будут необходимые библиотеки с их определенными версиями и не нужно вручную их настраивать и подключать. Нет проблем с версиями, все программисты в команде будут использовать нужные, определенные в pom.xml, версии. Также удобно, когда все зависимости прописаны в одном файле.

При использовании Maven проект имеет строго определенную структуру. Благодаря Maven, разработчикам, переходящим от проекта к проекту, нет необходимости постоянно разбираться в структуре проекта и в процессе его сборки.

При создании нового проекта можно создавать пустой проект, а также можно его создать на основе какого-нибудь стандартного архетипа, например, веб-приложения.

Таким образом, Maven значительно упрощает разработку приложений и делает создание и структурирование проекта более шаблонным.

Глава 3. Перспективы развития

В будущем планируется дальнейшее развитие проекта, а именно:

* Error tracking. Интеграция с одним из инструментов для логирования и сбора информации об ошибках в системе. Представителями таких инструментов являются: Raygun, Takipi, Airbrake, StackHunter, Sentry и др.
* Cloud Database. Экспорт существующей базы данных в облако и хранение там данных. Это обеспечит защиту данных и улучшит их доступность.
* Publish on cloud. Опубликование проекта в облаке и предоставление его конечным пользователям для использования.
* Добавление нового функционала. Планируется добавление подтверждения электронной почты, возможности оценивать фотографии, приглашать присоединиться к событию, создание закрытых событий и др.

### 

Заключение

В ходе работы над курсовым проектом мной были изучены Java-фреймворки для работы с веб-проектом и базами данных такие, как Spring, Hibernate, Apache Tiles. Эти фреймворки использовались для реализации веб-проекта, основной функциональностью которого является хранение и предоставление пользователям фотографий с различных событий.

Для хранения фотографий использовались облачные технологии, а именно Google Cloud Storage. Использование облачного хранилища позволяет значительно уменьшить объем данных, хранящихся на локальном сервере. Также это обеспечивает надежность и доступность данных, упрощает расширение пространства при необходимости.

Мною была изучена общая информация об облачных технологиях, а также преимущества и недостатки их использования. Среди основных недостатков можно выделить следующие: зависимость от поставщика, отсутствие полного контроля над ресурсами. Среди преимуществ - надежность, доступность, экономичность, масштабируемость, простота подключения и использования, мобильность, арендность и др.

В настоящее время облачные технологии постоянно развиваются. Качество и количество предоставляемых поставщиками услуг постоянно растет. Количество встраиваемых компьютеров увеличивается благодаря снижению цен на процессоры и повсеместному распространению Интернета. Растут также и объемы передаваемых данных с последующей их обработкой (часто в режиме реального времени).

Поэтому можно с уверенностью говорить, что в ближайшие годы роль облачных вычислений будет еще больше увеличиваться.

Список использованных источников

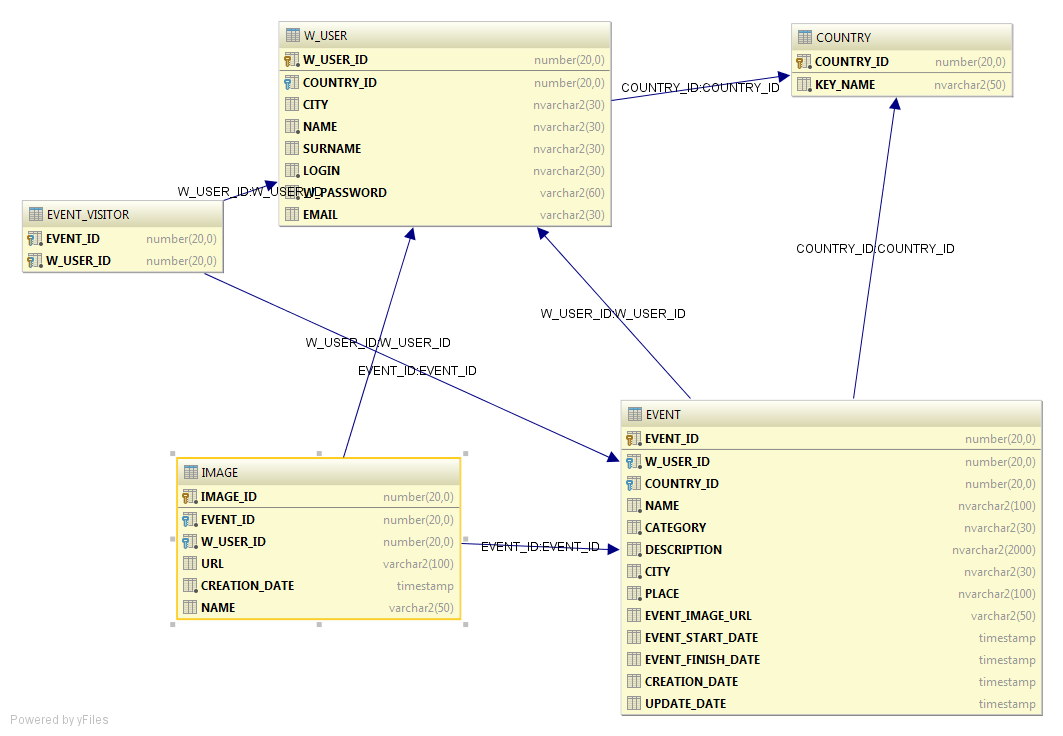
1. Облачные технологии: основные понятия, задачи и тенденции развития // Программные продукты, системы и алгоритмы URL: http://swsys-web.ru/cloud-computing-basic-concepts-problems.html
2. Что такое облака? // Rentacloud URL: http://www.rentacloud.su/whatiscloud
3. История развития облачных технологий. // Gurtam URL: http://blog.gurtam.com/2011/10/history-of-cloud-computing-from-amazon-to-wialon/
4. Бизнес в облаках. Чем полезны облачные технологии для предпринимателя. // Kontur URL: https://kontur.ru/articles/225
5. Google Cloud Storage Documentation // Google Cloud Platform URL: https://cloud.google.com/storage/docs/overview
6. Spring Framework // Википедия URL: https://ru.wikipedia.org/wiki/Spring\_Framework
7. Основы Hibernate // Хабрахабр URL: https://habrahabr.ru/post/29694/
8. Apache Tiles // Apache Tiles URL: https://tiles.apache.org/framework/index.html
9. Руководство по Maven // Apache Maven Project URL: http://www.apache-maven.ru/

Приложения

Приложение 1. Реестр вариантов использования

|  |  |
| --- | --- |
| Guest | Регистрация |
| Guest | Вход |
| User | Просмотр последних прошедших событий |
| User | Просмотр событий в его городе |
| User | Поиск событий по критериям |
| User | Просмотр полной информации о событии |
| User | Участие в событии |
| User | Удаление из участников события |
| User | Просмотр своих фотографий |
| User | Просмотр своих созданных событий |
| User | Просмотр событий, в которых участвует |
| User | Просмотр профиля |
| User | Редактирование личных данных |
| User | Изменение пароля |
| User | Создание событие |
| User | Редактирование события |
| User | Удаление события |
| User | Добавление фотографии |
| User | Удаление фотографии |

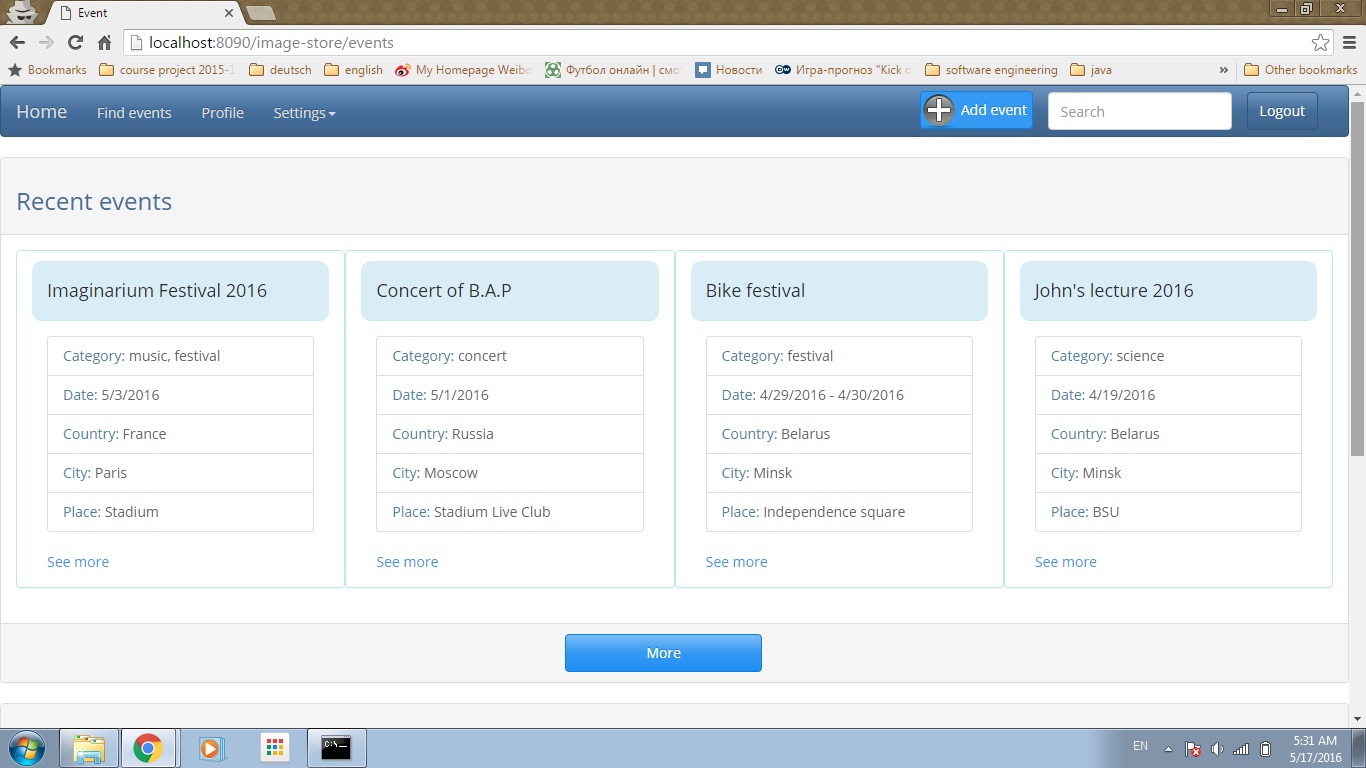
Приложение 2. Схема базы данных.



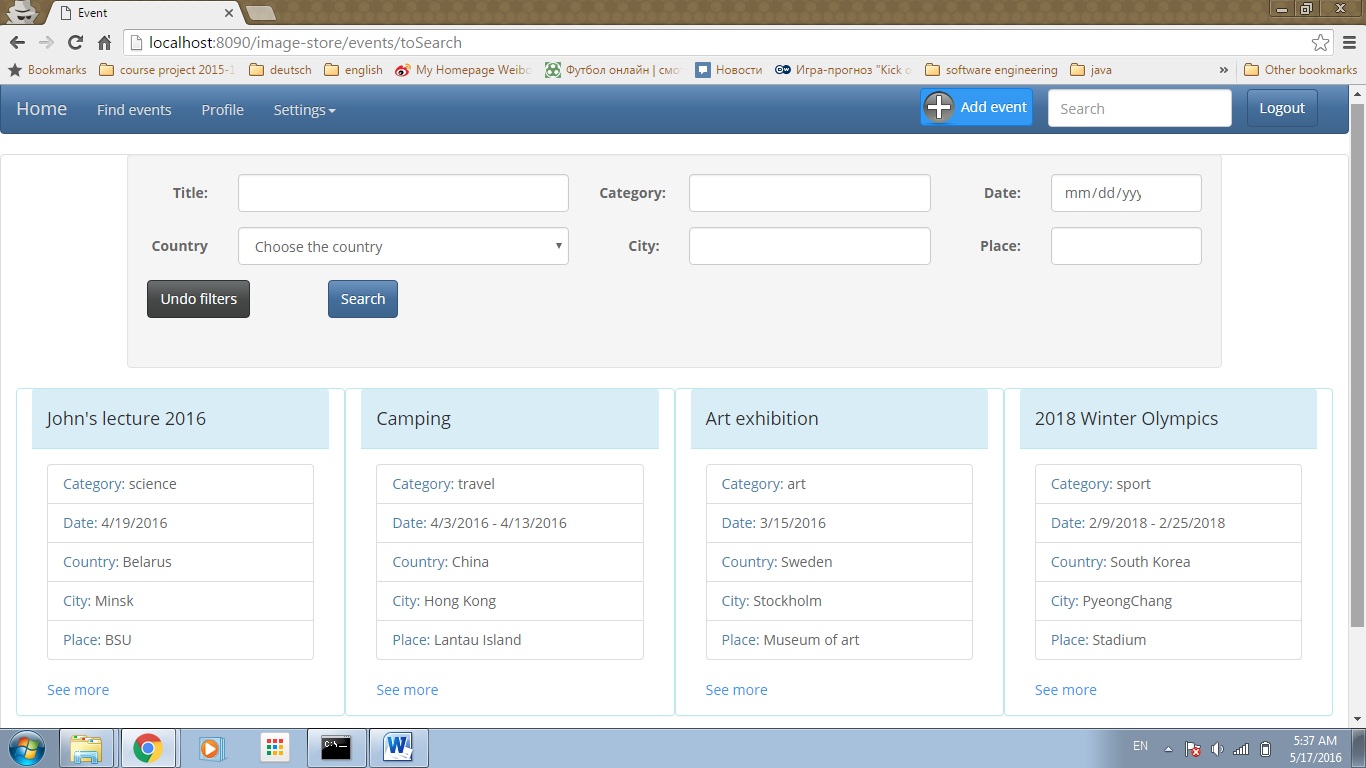
#### 

Приложение 3. Скриншоты основных страниц

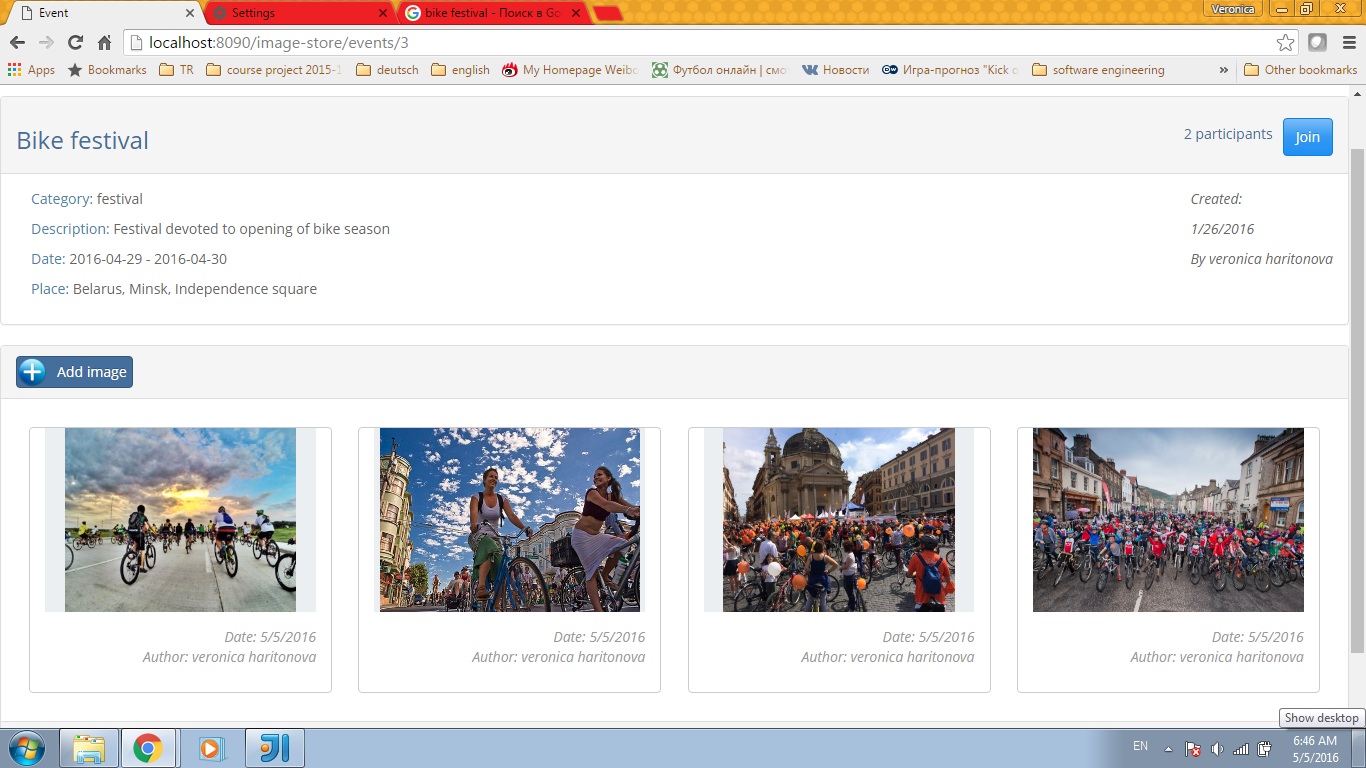
Главная страница:

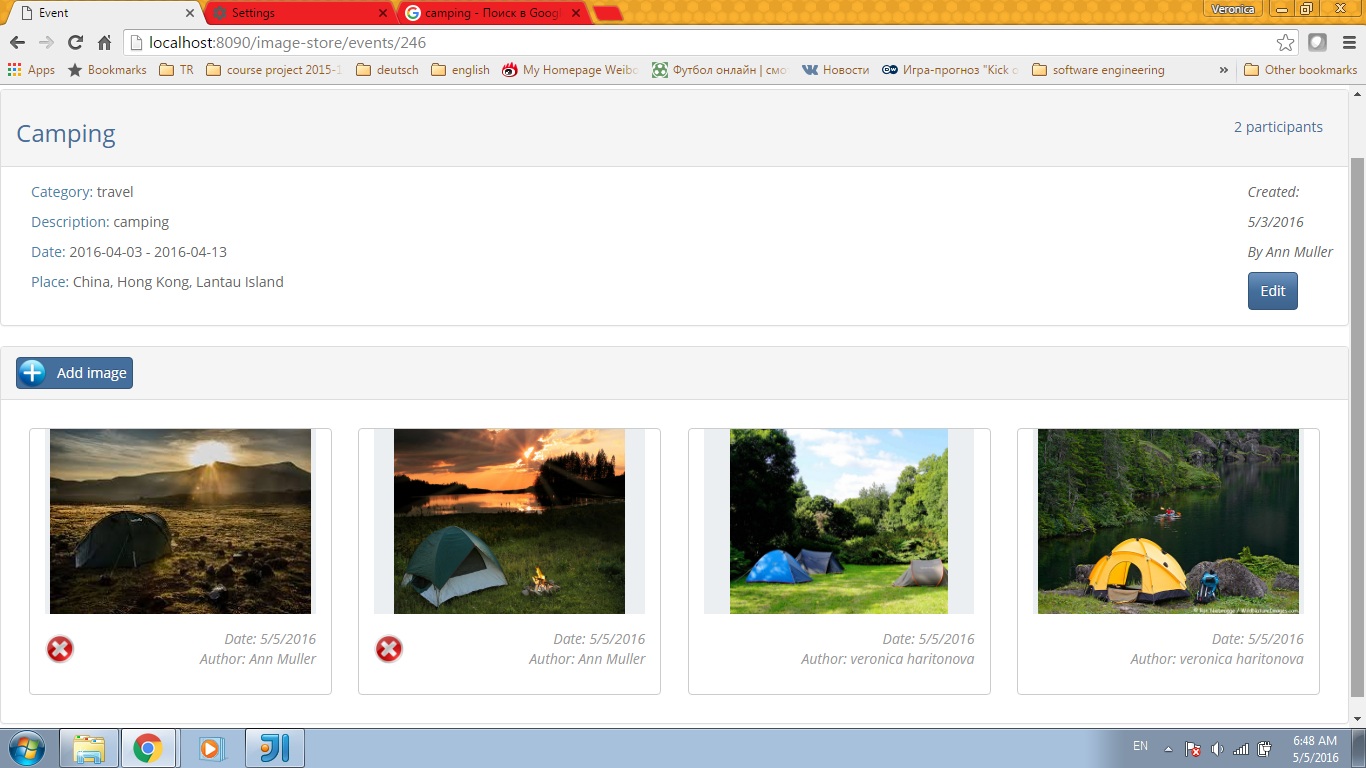


Поиск:

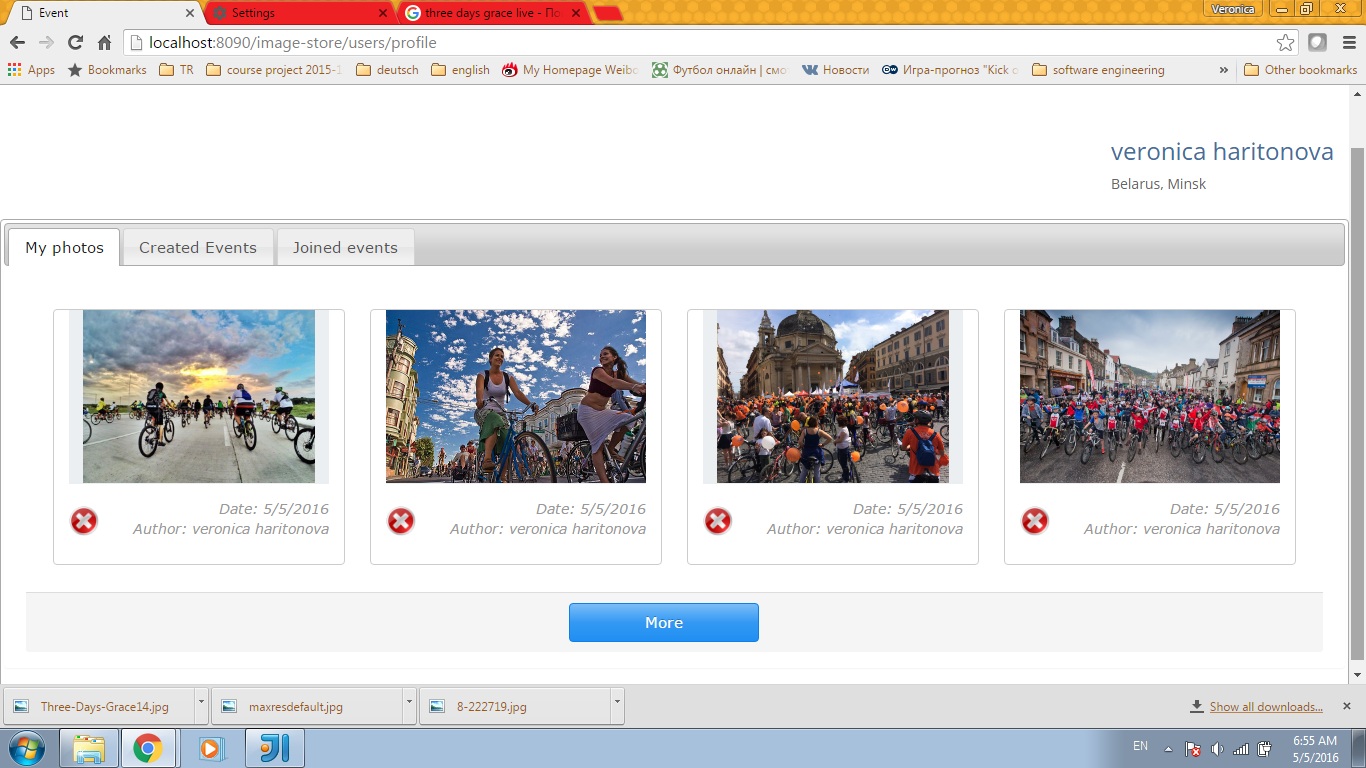


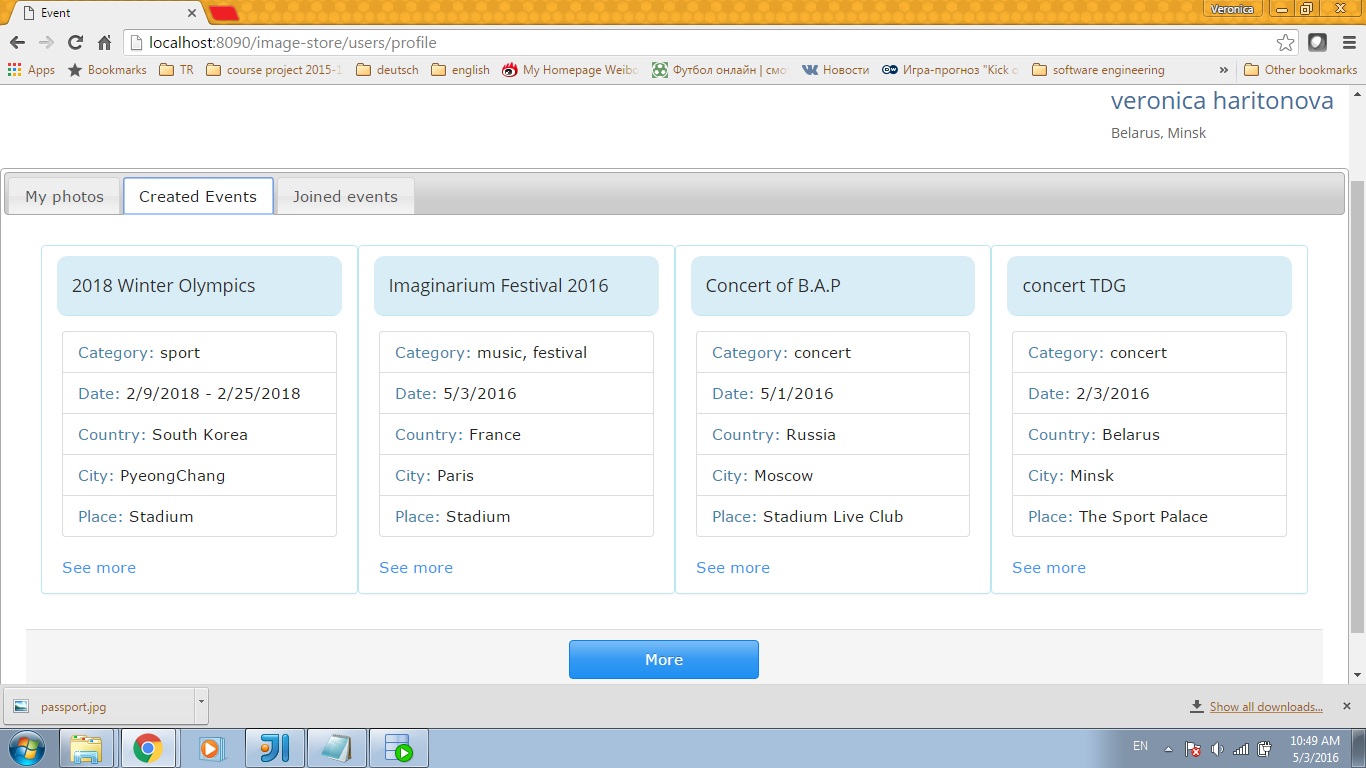
Страницы событий:





Страница пользователя:





Страница создания события:

