СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc484433129)

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ 8](#_Toc484433130)

[1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 9](#_Toc484433131)

[1.1 Система управления базами данных 9](#_Toc484433132)

[1.2 MySQL 10](#_Toc484433133)

[1.3 Spring Framework 12](#_Toc484433134)

[1.4 JavaScript 13](#_Toc484433135)

[1.5 jQuery 14](#_Toc484433136)

[2 ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 16](#_Toc484433137)

[2.1 Анализ предметной области 16](#_Toc484433138)

[2.2 Постановка задачи 16](#_Toc484433139)

[2.3 Обзор существующих аналогов 17](#_Toc484433140)

[2.3.1 Globus.tut.by 17](#_Toc484433141)

[2.3.2 Radzima.org 18](#_Toc484433142)

[2.3.3 Аutotravel.ru 20](#_Toc484433143)

[3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 21](#_Toc484433144)

[3.1 Анализ требований 21](#_Toc484433145)

[3.2 Разработка архитектуры программного комплекса 23](#_Toc484433146)

[3.3 Модель данных 24](#_Toc484433147)

[4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 26](#_Toc484433148)

[4.1 Реализация базы данных 26](#_Toc484433149)

[4.2 Реализация уровня доступа к данным 29](#_Toc484433150)

[4.2 Реализация уровня представления 32](#_Toc484433151)

[4.3 Реализация бизнес-логики 37](#_Toc484433152)

[5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ 40](#_Toc484433153)

[6 ТЕХНИКО - ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ИНФОРМАЦИОННО-ТУРИСТИЧЕСКОГО ПОРТАЛА 45](#_Toc484433154)

[6.1 Характеристика программного продукта 45](#_Toc484433155)

[6.2 Расчет затрат и отпускной цены программного средства 45](#_Toc484433156)

[6.3 Расчет сметы затрат и цены заказного ПО 47](#_Toc484433157)

[6.4 Оценка экономической эффективности применения программного средства у пользователя 52](#_Toc484433158)

[6.5 Расчет экономического эффекта 55](#_Toc484433161)

[6.6 Резюме 57](#_Toc484433162)

[ЗАКЛЮЧЕНИЕ 58](#_Toc484433163)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 59](#_Toc484433164)

[ПРИЛОЖЕНИЕ А 60](#_Toc484433165)

[ПРИЛОЖЕНИЕ B 73](#_Toc484433166)

## ВВЕДЕНИЕ

Мы живем в удивительное время возможностей. Совершить путешествие в соседний город, деревню – сегодня это проще простого. Поэтому так развит сегодня туризм, поэтому так много путешественников. И это здорово!

Каждый из нас в душе путешественник. И в каждом из нас живет непреодолимое желание познавать новые страны, знакомиться с новыми культурами и народами, открывать для себя новые ранее неизведанные места. Почувствовав вкус путешествий, мы уже не можем отказаться от возможности отправиться в ту или иную страну, открыть для себя ее тайны и загадки, окунуться в мир ее богатого культурного и исторического наследия.

Путешествие – это образ жизни. И в то же время путешествие определяет нашу жизнь. Мы с радостью меняем серые рабочие будни на яркие эмоции и впечатления, которые нам дарит захватывающие поездки и экстремальные походы. Мы знакомимся с новыми людьми, заводим друзей, встречаем единомышленников.

Путешествие делает нашу жизнь насыщенной и разнообразной. Новые поездки – это всегда яркие эмоции, незабываемые жизненные моменты, и отличная возможность отдохнуть, а также набраться сил и энергии.

Лишь путешествуя, мы живем. Лишь путешествуя, мы движемся вперед и открываем новые горизонты. Путешествие – это всегда яркая страничка в нашей жизни, наполненная яркими впечатлениями и сладостными воспоминаниями.

В рамках этого дипломного проекта будет рассмотрена разработка информационно-туристического портала, который будет представлять собой удобное средство для отображения достопримечательностей по типам, стилям, эпохам, населённым пунктам, подбор достопримечательностей на основе предпочтений, а также пользовательских отзывов.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности соответствует норме, установленной кафедрой информатики. Цитирования обозначены ссылками на публикации, указанные в «Списке литературы».

## ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей пояснительной записке применяют следующие обозначения и сокращения:

API – Application Programming Interfaces

BLL – Business Logic Layer

DAL – Data Access Layer

SDK – Software Development Kit

БД – база данных

ПО – программное обеспечение

ПП – программный продукт

СУБД – система управления базами данных

POJO – Plain Old Java Object

MVC – Model-View-Controller

DAO – Data Access Object

ООП – объектно-ориентированное программирование.

SQL – structured query language

## 

## 1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

* 1. Система управления базами данных

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.[1].

Можно выделить следующую классификацию СУБД

По модели данных:

– иерархические;

– сетевые;

– реляционные;

– объектно-ориентированные;

– объектно-реляционные;

По степени распределённости:

– локальные СУБД (все части локальной СУБД размещаются на одном компьютере);

– распределённые СУБД (части СУБД могут размещаться на двух и более компьютерах).

По способу доступа к БД:

– Файл-серверные. В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок. Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера. Недостатками являются потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД. На данный момент файл-серверная технология считается устаревшей. Примеры: Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro.

– Клиент-серверные. Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно. Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу. Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Примеры: Oracle, Firebird, Interbase, IBM DB2, Informix, MS SQL Server, Sybase Adaptive Server Enterprise, PostgreSQL, MySQL, Caché, ЛИНТЕР.

– Встраиваемые. Встраиваемая СУБД — СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети. Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить через SQL, либо через специальные программные интерфейсы. Примеры: OpenEdge, SQLite, BerkeleyDB, Firebird Embedded, Microsoft SQL Server Compact, ЛИНТЕР.

Ниже перечислены основные функции СУБД:

– управление данными во внешней памяти (на дисках);

– управление данными в оперативной памяти с использованием дискового кэша;

– журнализация изменений, резервное копирование и восстановление базы данных после сбоев;

– поддержка языков БД (язык определения данных, язык манипулирования данными);

Обычно современная СУБД содержит следующие компоненты:

– ядро, которое отвечает за управление данными во внешней и оперативной памяти, и журнализацию;

– процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;

– подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;

– сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

* 1. MySQL

Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, необходима система управления базой данных, каковой и является ПО MySQL.[2]. Поскольку компьютеры замечательно справляются с обработкой больших объемов данных, управление базами данных играет центральную роль в вычислениях. Реализовано такое управление может быть по-разному - как в виде отдельных утилит, так и в виде кода, входящего в состав других приложений.

MySQL - это система управления реляционными базами данных. В реляционной базе данных данные хранятся не все скопом, а в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных.

MySQL - это ПО с открытым кодом. Применять его и модифицировать может любой желающий. Такое ПО можно получать по Internet и использовать бесплатно. При этом каждый пользователь может изучить исходный код и изменить его в соответствии со своими потребностями. Использование программного обеспечения MySQL регламентируется лицензией GPL (GNU General Public License) в которой указано, что можно и чего нельзя делать с этим программным обеспечением в различных ситуациях.

MySQL является системой клиент-сервер, которая содержит многопоточный SQL-сервер, обеспечивающий поддержку различных вычислительных машин баз данных, а также несколько различных клиентских программ и библиотек, средства администрирования и широкий спектр программных интерфейсов (API). Мы также поставляем сервер MySQL в виде многопоточной библиотеки, которую можно подключить к пользовательскому приложению и получить компактный, более быстрый и легкий в управлении продукт. Доступно также большое количество программного обеспечения для MySQL, в большей части - бесплатного.

MySQL состоит из двух частей: серверной и клиентской.

Сервер MySQL постоянно работает на компьютере. Клиентские программы (например, скрипты PHP) посылают серверу MySQL SQL-запросы через механизм сокетов (то есть при помощи сетевых средств), сервер их обрабатывает и запоминает результат. То есть скрипт (клиент) указывает, какую информацио он хочет получить от сервера баз данных. Затем сервер баз данных посылает ответ (результат) клиенту (скрипту).  
Почему всегда передается не весь результат? Очень просто: дело в том, что размер результирующего набора данных может быть слишком большим, и на его передачу по сети уйдет чересчур много времени. Да и редко когда бывает нужно получать сразу весь вывод запроса (то есть все записи, удовлетворяющие выражению запроса). Например, нам может потребоваться лишь подсчитать, сколько записей удовлетворяет тому или иному условию, или же выбрать из данных только первые 10 записей. Механизм использования сокетов подразумевает технологию клиент-сервер, а это означает, что в системе должна быть запущена специальная программа — MySQL-сервер, которая принимает и обрабатывает запросы от программ. Так как вся работа происходит в действительности на одной машине, накладные расходы по работе с сетевыми средствами незначительны (установка и поддержание соединения с MySQL-сервером обходится довольно дешево).

Cтруктура MySQL трехуровневая: базы данных — таблицы — записи. Базы данных и таблицы MySQL физически представляются файлами с расширениями frm, MYD, MYI. Логически - таблица представляет собой совокупность записей. А записи - это совокупность полей разного типа. Имя базы данных MySQL уникально в пределах системы, а таблицы - в пределах базы данных, поля - в пределах таблицы. Один сервер MySQL может поддерживать сразу несколько баз данных, доступ к которым может разграничиваться логином и паролем. Зная эти логин и пароль, можно работать с конкретной базой данных. Например, можно создать или удалить в ней таблицу, добавить записи и т. д. Обычно имя-идентификатор и пароль назначаются хостинг провайдерами, которые и обеспечивают поддержку MySQL для своих пользователей.

* 1. Spring Framework

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development» (Wrox Press, октябрь 2002 года).

Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первый стабильный релиз 1.0 был выпущен в марте 2004. Spring 2.0 был выпущен в октябре 2006, Spring 2.5 — в ноябре 2007, Spring 3.0 в декабре 2009, и Spring 3.1 в декабре 2011. Текущая версия — 4.3.7.

Несмотря на то, что Spring не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет бо́льшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе.[3]. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

* 1. JavaScript

JavaScript — прототипно-ориентированный сценарный язык программирования. Является диалектом языка ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.[4].

JavaScript является объектно-ориентированным языком, но используемое в языке прототипирование обуславливает отличия в работе с объектами по сравнению с традиционными класс-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, анонимные функции, замыкания — что придаёт языку дополнительную гибкость.

Несмотря на схожий с Си синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

– функции как объекты первого класса;

– автоматическое приведение типов;

– автоматическая сборка мусора;

– анонимные функции.

В языке отсутствуют такие полезные вещи, как:

– модульная система: JavaScript не предоставляет возможности управлять зависимостями и изоляцией областей видимости;

– стандартная библиотека: в частности, отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода/вывода, базовых типов для бинарных данных;

– стандартные интерфейсы к веб-серверам и базам данных;

– система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

* 1. jQuery

JQuery - это кроссплатформенная JavaScript-библиотека, предназначенная для упрощения клиентского сценария HTML. Это бесплатное программное обеспечение с открытым исходным кодом, использующее лицензию MIT. Веб-анализ показывает, что это самая широко распространенная библиотека JavaScript с большим отрывом. Синтаксис jQuery предназначен для упрощения навигации по документу, выбора элементов DOM, создания анимаций, обработки событий и разработки приложений Ajax.[6]. JQuery также предоставляет разработчикам возможности создавать плагины поверх библиотеки JavaScript. Это позволяет разработчикам создавать абстракции для низкоуровневого взаимодействия и анимации, расширенных эффектов и высокоуровневых, тематических виджетов. Модульный подход к библиотеке jQuery позволяет создавать мощные динамические веб-страницы и веб-приложения.

JQuery, по своей сути, является библиотекой манипуляций с объектной моделью документа (DOM). DOM представляет собой древовидное представление всех элементов веб-страницы. JQuery упрощает синтаксис поиска, выбора и управления этими элементами DOM. Например, jQuery можно использовать для поиска элемента в документе с определенным свойством (например, всех элементов с тегом h1), изменения одного или нескольких его атрибутов (например, цвета, видимости) или создания ответа на событие (Например щелчок мышью). JQuery также предоставляет парадигму обработки событий, которая выходит за рамки выбора и манипулирования базовыми элементами DOM. Назначение события и определение функции обратного вызова выполняются за один шаг в одном месте кода. JQuery также ставит своей целью включить другие высокоиспользуемые функции JavaScript (например, затухание и затухание при скрытии элементов, анимацию, управляя свойствами CSS). Принципы разработки с помощью jQuery: Разделение JavaScript и HTML. Библиотека jQuery предоставляет простой синтаксис для добавления обработчиков событий в DOM с помощью JavaScript, вместо добавления атрибутов событий HTML для вызова функций JavaScript. Таким образом, это побуждает разработчиков полностью отделять код JavaScript от HTML-разметки. Краткость и ясность: jQuery способствует краткости и ясности с такими функциями, как цепные функции и сокращенные имена функций. Устранение несовместимости между браузерами. Двигатели JavaScript разных браузеров немного отличаются, поэтому код JavaScript, который работает для одного браузера, может не работать для другого. Как и другие инструменты JavaScript, jQuery обрабатывает все эти кросс-браузерные несоответствия и обеспечивает согласованный интерфейс, который работает в разных браузерах. Расширяемость. Новые события, элементы и методы могут быть легко добавлены, а затем повторно использованы в качестве плагина.

## 2 ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Анализ предметной области

Каждому человек в своей жизни путешествовал. Совершить путешествие в соседний город, деревню – сегодня это проще простого. Поэтому так развит сегодня туризм, туристические сайты, поэтому так много путешественников. Как же выбрать для себя место куда поехать, что посмотреть?

В интернете можно найти десятки сайтов на туристическую тематику. Все они имею схожую функциональность, но часто имеют один недостаток: они не предоставляют достопримечательности на основе предпочтений.

Программное средство, разрабатываемое в рамках рассматриваемого дипломного проекта, предназначено для упорядочения информации обо всех достопримечательностях Беларуси, подбора достопримечательностей на основе предпочтений, вывод ТОП-50 объектов на основе пользовательских отзывов. Основная цель заключается в объединении данных обо всех достопримечательностях Беларуси.

В ходе разработки приложения были проанализированы слабые и сильные места схожих систем, выделены те важные аспекты, которые необходимы пользователям, и все это было реализовано в данном проекте.

2.2 Постановка задачи

В рамках дипломного проекта поставлена цель разработать веб-сайт, который предоставляет пользователям портал для получения информации о достопримечательностях Беларуси, подборе интересующих объектов на основе эпохи, архитектурного стиля.

Разрабатываемый сайт должен удовлетворять следующим требованиям:

* Система должна обладать понятным и удобным интерфейсом, который позволяет пользователю, впервые воспользовавшись сервисом, без особых усилий выполнить необходимые действия.
* Сайт должен одинаково хорошо отображаться на всех популярных на данный период мобильных платформах, с различной конфигурацией программного обеспечения, аппаратного обеспечения и размеров экрана. Также необходимым является и автоматическое растягивание элементов интерфейса, чтобы выглядеть гармонично на устройствах с различными размерами экрана и плотностью пикселей.
* Реализовать удобную систему пользовательской функциональности, авторизацию, регистрацию, роли пользователей, права доступа, управление объектами для администратора
* Архитектура приложения должна быть многослойна и реализовываться с помощью шаблона проектирования MVC что позволит упростить разработку и последующую поддержку системы.
* Доступ к базе данных в приложении должен осуществляться посредством технологии объектно-реляционного отображения, которая связывает базы данных с концепциями объектно-ориентированного программирования, что минимизирует время работы разработчика с бд и уменьшит риск написания непроизводительных и небезопасных sql-запросов

Для достижения поставленной цели необходимо решить следующие задачи:

* С помощью языка программирования Java и фреймфорка Spring Framework, реализовать серверную часть сайта.
* С помощью языка программирования JavaScript и фреймфорка Bootstrap, реализовать клиентскую часть сайта
* С помощью языка запросов SQL и библиотеки Hibernate реализовать работу с данными

2.3 Обзор существующих аналогов

Наиболее известные аналоги программного средства: globus.tut.by, radzima.org, autotravel.ru.

2.3.1 Globus.tut.by

Globus.tut.by представляет собой онлайн-сервис.

На сайте представлена информация о достопримечательностях Беларуси. Есть возможность посмотреть Тот-20, Топ-100 достопримечательностей, присутствует форум.

Главной особенностью сайта является удобная каталогизация объектов по областям, районам, карта достопримечательностей.

На рисунке 2.1 изображена главная страница сайта Globus.tut.by.



Рисунок 2.1 – Globus.tut.by, главная страница.

Можно выделить следующие достоинства данного онлайн-сервиса:

* присутствует форум;
* карта достопримечательностей;
* маршруты путешествий;
* Топ-20, Топ-100 достопримечательностей;
* каталогизация по областям, районам;
* есть выбор интересующих архитектурных стилей, типов.

Однако помимо достоинств онлайн-сервиса можно выделить и следующие недостатки:

* отсутствует поиск по достопримечательностям;
* устаревший интерфейс.

2.3.2 Radzima.org

Белорусский краеведческий ресурс Интернета, созданный в 2002 году. Идея и разработка сайта принадлежат Виктору Калиновскому и Константину Шестовскому.

На настоящий момент география проекта охватывает Брестскую, Гродненскую, Витебскую, Могилевскую, Гомельскую, Минскую, Смоленскую, Брянскую области. Ресурс имеет четыре языковые версии - белорусский, английский, польский и русский.

Проект реализует возможность централизации, систематизацию и своевременного обновления краеведческой информации и представляет собой справочник краеведческих знаний. На сайте много оригинального фотоматериала, можно найти интересные исторические сведения непосредственно о регионах и места бывшего Великого Княжества Литовского. Проект включает как информацию о существующих историко-архитектурные ценности, так и сведения о потерянных достопримечательностях.

Каждый посетитель может стать автором и прислать свой материал, который будет просмотрен и опубликован.

На рисунке 2.2 изображена главная страница сайта Radzima.org.



Рисунок 2.2 – Radzima.org, главная страница.

Достоинства данного сайта:

* карта с достопримечательностями;
* большой каталог достопримечательностей;
* открытость проекта;
* география охватывает Беларусь, Литву, Россию ;
* наличие системы управления пользователями;
* эргономичный дизайн и пользовательский интерфейс;

Недостатки:

* отсутствуют маршруты;
* отсутствует форум;

2.3.3 Аutotravel.ru

На сайте присутствуют данные практически по всем городам России. Все города собраны в алфавитном порядке, что очень облегчает поиск.

Для каждого города собраны наиболее полные данные о гостиницах, местах общественного питания, достопримечательностях. Есть уже готовые маршруты с подробным описанием местности, координатами, точками прибытия, указанием где поесть-поспать.

Если вы путешествуете на автомобиле, то вам в помощь на сайте размещены карты той или иной местности. В каждом городе есть ссылка на карту Гугл с расставленными маркерами объектов.

На рисунке 2.3 изображена главная страница сайта Autotravel.ru.

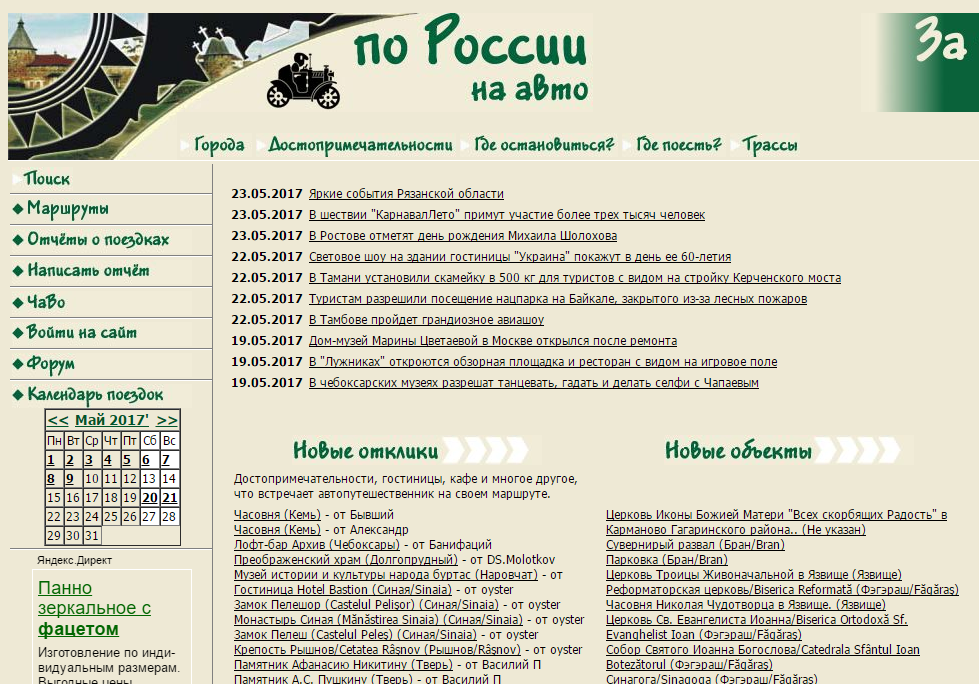


Рисунок 2.3 – Аutotravel.ru, главная страница.

Достоинства:

* города собраны в алфавитном порядке;
* собраны наиболее полные данные о достопримечательностях;
* Google карта достопримечательностей.

Недостатки:

* устаревший дизайн;
* неудобный пользовательский интерфейс.

## 3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

Проектирование является важным этапом разработки программного обеспечения. Любой проект, связанный с созданием программного продукта, требует предварительного проектирования, построения структуры и планирования сроков разработки. Лишь после предварительного утверждения плана разработки и выбора используемых технологий и структуры программного продукта начинается его реализация.

3.1 Анализ требований

В данном проекте будет две роли: пользователь и администратор. Пользователь – это человек, который зашел на сайт и прошёл регистрацию в системе.

Ниже представлены основные функции пользователя:

– возможность регистрации, авторизации и восстановления пароля по средствам электронной почты на сайте;

– просмотр Топ-50 достопримечательностей;

– просмотр объектов определённой эпохи;

– просмотр объектов определённого архитектурного стиля;

– просмотр объектов определённого типа;

– возможность редактирования предпочтительных эпох, стилей, типов;

– просмотр объектов определённого населённого пункта;

– просмотр объектов на основе предпочтений;

– возможность смены пароля;

– просмотр карты достопримечательностей.

Также надо отметить, что функция просмотра объектов на основе предпочтений основывается на основе предпочитаемых стилей, эпох, типов. Пользователь может редактировать предпочитаемые стили, эпохи, типы в кабинете пользователя. Также для выбора определённого населённого пункта можно выбрать его из общего списка населённых пунктов или путём выбора нужной области, затем нужного района.

Доступ пользователя к этим функциям должен быть простым и удобным. В основе реализации графического интерфейса и элементов управления должны лежать современные методики и подходы, использующиеся в разработке приложений. Пользователь должен интуитивно понимать, какая кнопка какую функцию выполняет и куда его приведет то или иное действие.

На рисунке 3.1 изображен сценарий использования приложения пользователем.

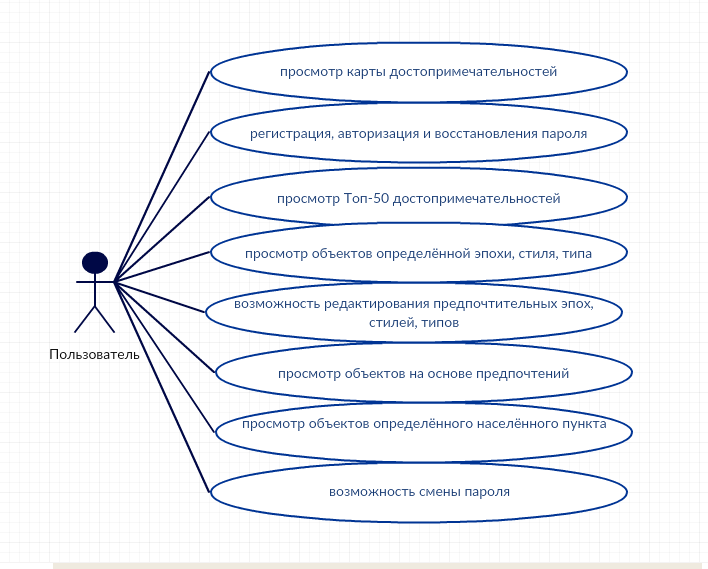


Рисунок 3.1 – Сценарий использования приложения для пользователя

Администратор – это человек, управляющий сайтом, наполнением и редактированием объектов.

Ниже представлены основные функции администратора:

– управление эпохами;

– управление стилями;

– управление типами;

– управление объектами;

– редактирование пользовательских аккаунтов;

– управление отзывами;

– управление статическими страницами сайта.

Также надо отметить, что отзывы модерируются. После того как пользователь оставил отзыв, администратор может его одобрить или удалить.

На рисунке 3.2 изображен сценарий использования приложения администратором.

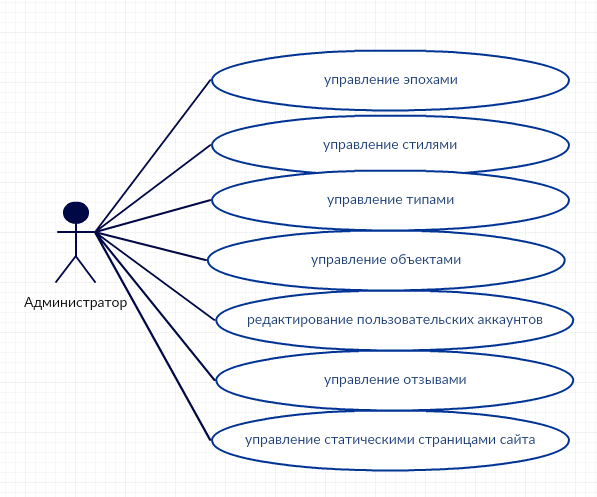


Рисунок 3.2 – Сценарий использования приложения для администратора

Разработка дипломного проекта предполагает реализацию всего выше описанного функционала.

3.2 Разработка архитектуры программного комплекса

Разрабатываемое приложение построено с использованием трехуровневой архитектуры. Концептуальная схема архитектуры системы отображена на рисунке 3.3.

Все уровни приложения взаимодействуют друг с другом посредством четко определенных интерфейсов и классов. Это помогает создать гибкую систему, позволяющую менять реализацию на уровнях без изменений, связанных с этим уровнем других уровней. Более низкий слой при данном проектировании не может напрямую обращаться к более высокому слою.

На рисунке 3.3 изображена концептуальная схема архитектуры приложения.

*Уровень представления*

*Уровень бизнес-логики*

*Удаленный сервер*

*Уровень доступа к данным*

Рисунок 3.3 – Концептуальная схема архитектуры приложения

Уровень представления. Взаимодействие пользователя с системой происходит непосредственно через данный уровень. Уровень представления содержит пользовательский интерфейс, состоящий из графических элементов, элементов управления и форм, отражающих и изменяющих состояние бизнес-модели приложения посредством взаимодействия с уровнем бизнес-логики.

Уровень бизнес-логики. На данном уровне реализуются бизнес-сущности системы, а также методы и классы для работы с ними. Бизнес-сущности описывают состояния объектов. Для изменения состояния бизнес-модели реализуется набор методов и классов, которые оперируют бизнес-сущностями.

Уровень доступа к данным. Задача данного уровня – инкапсуляция работы с базой данных. На данном уровне описываются сущности БД, которые непосредственно проецируются на ее структуру. Для обеспечения сохранения и получения состояния бизнес-модели реализуется набор классов, через которые производится взаимодействие с БД.

3.3 Модель данных

Чтобы приступить к непосредственному проектированию базы данных, необходимо выделить минимальный набор основных сущностей и указать типы связи между ними.

В рассматриваемой предметной области можно выделить следующие основные сущности:

Отзывы – содержит отзывы к достопримечательностям.

Пользователь – содержит основную информацию о пользователе, хранит данные его профиля, настройки системы и данные для авторизации.

Стиль – содержит данные об определённом архитектурном стиле.

Тип – содержит данные об определённом типе.

Эпоха – содержит данные об определённой эпохе.

Роль – есть пользователь и администратор.

А также сущности: Населённый пункт, Область, Район.

Оставшиеся сущности будут представлены на схеме базы данных.

На рис. 3.4 представлена схема спроектированной базы данных.

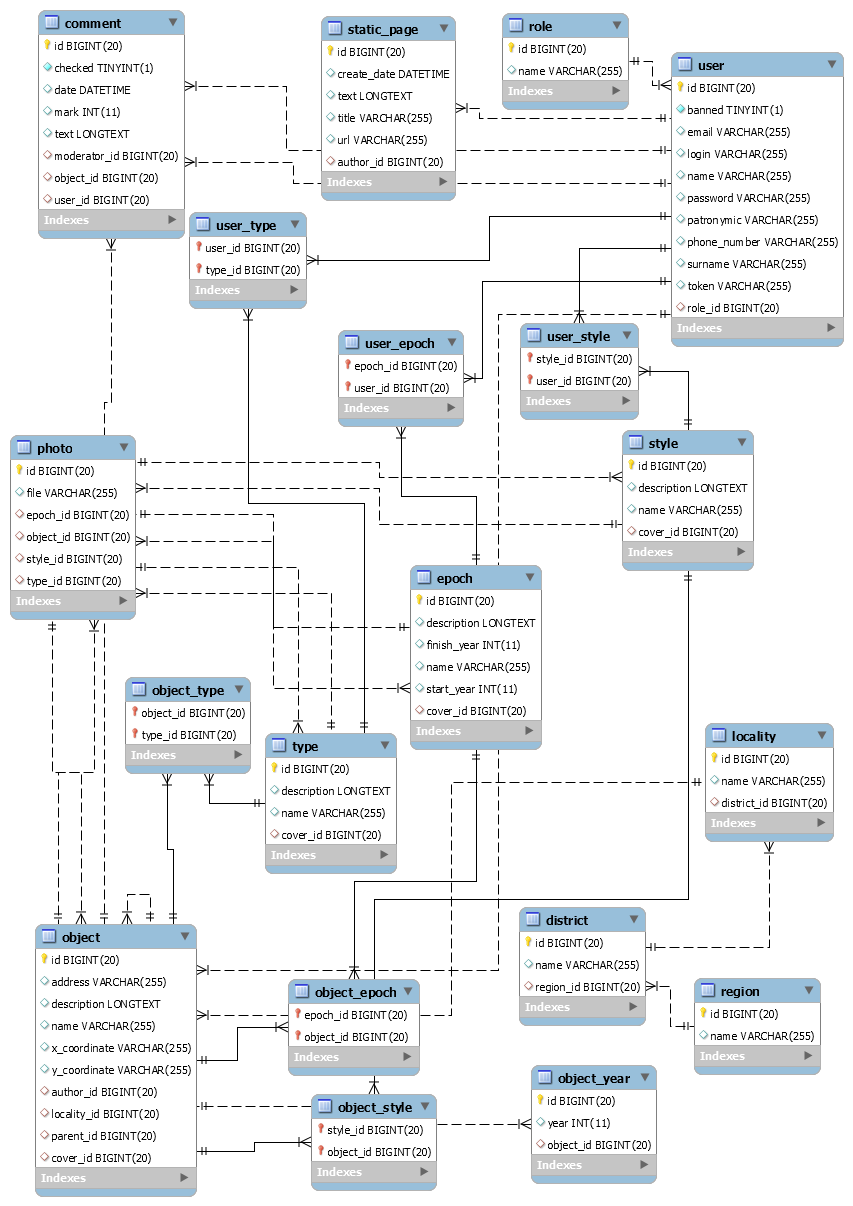


Рисунок 3.4 – Схема базы данных проекта

Детальное описание каждой сущности будет рассмотрено в следующем разделе.

## 4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

4.1 Реализация базы данных

Рассмотрим на физическом уровне таблицы, являющиеся хранилищами для основных бизнес-сущностей приложения, таких как:

- Comment

- Object

- Style

- Type

- Epoch

- User

Рассмотрим описание данных таблицы Comment на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS `comment` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`checked` tinyint(1) NOT NULL DEFAULT '0',

`date` datetime DEFAULT NULL,

`mark` int(11) DEFAULT NULL,

`text` longtext,

`moderator\_id` bigint(20) DEFAULT NULL,

`object\_id` bigint(20) DEFAULT NULL,

`user\_id` bigint(20) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `FK\_bsfc1m8kh6w94ir2xe8i8yfic` (`moderator\_id`),

KEY `FK\_7oolb0mu1ayeskgpq0t87m7oh` (`object\_id`),

KEY `FK\_mxoojfj9tmy8088avf57mpm02` (`user\_id`),

CONSTRAINT `FK\_7oolb0mu1ayeskgpq0t87m7oh` FOREIGN KEY (`object\_id`) REFERENCES `object` (`id`),

CONSTRAINT `FK\_bsfc1m8kh6w94ir2xe8i8yfic` FOREIGN KEY (`moderator\_id`) REFERENCES `user` (`id`),

CONSTRAINT `FK\_mxoojfj9tmy8088avf57mpm02` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

)

Краткое описание полей таблицы:

Text – содержит текст комментария.

Mark – содержит пользовательскую оценку объекта от 1 до 5.

Checked – так как комментарии модерируются, то по умолчанию false, комментарий недоступен другим пользователям, после одобрения администратора true, другие пользователи могут видеть данный комментарий.

Data – содержит дату, когда оставлен комментарий.

Рассмотрим описание данных таблицы Object на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS `object` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`address` varchar(255) DEFAULT NULL,

`description` longtext,

`name` varchar(255) DEFAULT NULL,

`x\_coordinate` varchar(255) DEFAULT NULL,

`y\_coordinate` varchar(255) DEFAULT NULL,

`author\_id` bigint(20) DEFAULT NULL,

`locality\_id` bigint(20) DEFAULT NULL,

`parent\_id` bigint(20) DEFAULT NULL,

`cover\_id` bigint(20) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `FK\_m17ecx43lmgjfu91m9wyfq5rm` (`author\_id`),

KEY `FK\_mipy2h9s36636omvuyay75cah` (`locality\_id`),

KEY `FK\_kv1nch3x7cf96bo8s7ihpk6ra` (`parent\_id`),

KEY `FK\_9smxcckjnl7qrawc83s45r7sc` (`cover\_id`),

CONSTRAINT `FK\_9smxcckjnl7qrawc83s45r7sc` FOREIGN KEY (`cover\_id`) REFERENCES `photo` (`id`),

CONSTRAINT `FK\_kv1nch3x7cf96bo8s7ihpk6ra` FOREIGN KEY (`parent\_id`) REFERENCES `object` (`id`),

CONSTRAINT `FK\_m17ecx43lmgjfu91m9wyfq5rm` FOREIGN KEY (`author\_id`) REFERENCES `user` (`id`),

CONSTRAINT `FK\_mipy2h9s36636omvuyay75cah` FOREIGN KEY (`locality\_id`) REFERENCES `locality` (`id`)

)

Краткое описание полей таблицы:

Name – содержит название объекта.

Description – содержит описание объекта.

Address – содержит адрес объекта.

X, Y coordinate – содержит координаты объекта.

Рассмотрим описание данных таблицы Style, Epoch, Type на DDL-языке SQL, так как таблицы идентичны, то рассмотрим на примере модели Style:

CREATE TABLE IF NOT EXISTS `style` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`description` longtext,

`name` varchar(255) DEFAULT NULL,

`cover\_id` bigint(20) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `FK\_alw2qol5avcga14twy3nrs639` (`cover\_id`),

CONSTRAINT `FK\_alw2qol5avcga14twy3nrs639` FOREIGN KEY (`cover\_id`) REFERENCES `photo` (`id`)

)

Краткое описание полей таблицы:

Name – содержит название стиля.

Description – содержит описание стиля.

Рассмотрим описание данных таблицы User на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS `user` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`banned` tinyint(1) NOT NULL DEFAULT '0',

`email` varchar(255) DEFAULT NULL,

`login` varchar(255) DEFAULT NULL,

`name` varchar(255) DEFAULT NULL,

`password` varchar(255) DEFAULT NULL,

`patronymic` varchar(255) DEFAULT NULL,

`phone\_number` varchar(255) DEFAULT NULL,

`surname` varchar(255) DEFAULT NULL,

`token` varchar(255) DEFAULT NULL,

`role\_id` bigint(20) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `FK\_qleu8ddawkdltal07p8e6hgva` (`role\_id`),

CONSTRAINT `FK\_qleu8ddawkdltal07p8e6hgva` FOREIGN KEY (`role\_id`) REFERENCES `role` (`id`)

)

Краткое описание полей таблицы:

Name – содержит имя пользователя.

Surname – хранит фамилию пользователя.

Login – содержит логин пользователя.

Password – хранит пароль в зашифрованном виде.

Phone\_number – номер телефона пользователя.

4.2 Реализация уровня доступа к данным

Для доступа к БД мы используем Hibernate. Необходимо настроить фреймворк Spring для работы с источником данных, чтобы обеспечить доступ объектов DAO к базе данных. Независимо от используемой формы поддержки DAO необходимо настроить ссылку на источник данных. Можно использовать XML и создать некоторые конфигурационные файлы.

Содержимое конфигурационного файла:

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans:beans xmlns="http://www.springframework.org/schema/security"  
 xmlns:beans="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  
 http://www.springframework.org/schema/security  
 http://www.springframework.org/schema/security/spring-security.xsd"**>  
  
 <**http pattern="/admin/\*\*" auto-config="false" use-expressions="true"**>  
 <**csrf disabled="true"**/>  
 <**access-denied-handler error-page="/denied"** />  
 <**intercept-url pattern="/admin/login" access="isAnonymous()"**/>  
 <**intercept-url pattern="/admin/\*\*" access="hasRole('ROLE\_ADMIN')"**/>  
 <**form-login login-page="/admin/login" login-processing-url="/admin/login"  
 default-target-url="/admin" authentication-failure-url="/admin/login?error=1"  
 username-parameter="login" password-parameter="password"** />  
 <**logout logout-url="/admin/logout" logout-success-url="/admin/login?logout"** />  
 <**remember-me key="xzvCOBOV7bNRaAe" token-validity-seconds="31556926"**/>  
 </**http**>  
  
 <**http auto-config="true"**>  
 <**csrf disabled="true"**/>  
 <**intercept-url pattern="/denied/\*" access="permitAll"** />  
 <**access-denied-handler error-page="/denied"** />  
 <**intercept-url pattern="/login" access="isAnonymous()"**/>  
 <**intercept-url pattern="/registration" access="isAnonymous()"**/>  
 <**intercept-url pattern="/restore" access="isAnonymous()"**/>  
 <**intercept-url pattern="/profile" access="hasRole('ROLE\_ADMIN') || hasRole('ROLE\_USER')"**/>  
 <**intercept-url pattern="/settings" access="hasRole('ROLE\_ADMIN') || hasRole('ROLE\_USER')"**/>  
 <**intercept-url pattern="/preferences" access="hasRole('ROLE\_ADMIN') || hasRole('ROLE\_USER')"**/>  
 <**intercept-url pattern="/\*\*" access="permitAll"**/>  
 <**form-login login-page="/login" login-processing-url="/login"  
 default-target-url="/" authentication-failure-url="/login?error=1"  
 username-parameter="login" password-parameter="password"** />  
 <**logout logout-url="/logout" logout-success-url="/"** />  
 <**remember-me key="eAdBJv6V3v6eOaB" token-validity-seconds="31556926"**/>  
 </**http**>  
  
 <**authentication-manager alias="authenticationManager"**>  
 <**authentication-provider ref="customAuthenticationProvider"**/>  
 </**authentication-manager**>  
  
 <**beans:bean id="encoder" class="org.springframework.security.authentication.encoding.Md5PasswordEncoder"** />  
</**beans:beans**>

Создадим POJO – класс Object.java, который будет отображать сущность Объект.

Расcмотрим реализацию модели Object:

@Entity

@Table(name = **"object"**)  
**public class** Object **extends** BaseEntity {  
 @Column(name = **"name"**)  
 **private** String **name**;  
  
 @Column(name = **"description"**, columnDefinition=**"LONGTEXT"**)  
 **private** String **description**;  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 **private** Locality **locality**;  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 **private** Object **parent**;  
  
 @OneToMany(mappedBy = **"parent"**, fetch = FetchType.***LAZY***, cascade= CascadeType.***REMOVE***, orphanRemoval = **true**)  
 @OrderBy(**"name ASC"**)  
 **private** Set<Object> **childObjects** = **new** HashSet<>();  
  
 @OneToMany(mappedBy = **"object"**, fetch = FetchType.***LAZY***, cascade=CascadeType.***REMOVE***, orphanRemoval = **true**)  
 @OrderBy(**"id ASC"**)  
 **private** Set<Photo> **photos** = **new** HashSet<>();  
  
 @OneToMany(mappedBy = **"object"**, fetch = FetchType.***LAZY***, cascade= CascadeType.***ALL***, orphanRemoval = **true**)  
 @OrderBy(**"id ASC"**)  
 **private** Set<ObjectYear> **yearList** = **new** HashSet<>();  
  
 @Column(name = **"x\_coordinate"**)  
 **private** String **xCoordinate**;  
  
 @Column(name = **"y\_coordinate"**)  
 **private** String **yCoordinate**;  
  
 @Column(name = **"address"**)  
 **private** String **address**;  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 **private** User **author**;  
  
 @ManyToMany(fetch = FetchType.***LAZY***, cascade ={CascadeType.***MERGE***, CascadeType.***PERSIST***, CascadeType.***REFRESH***})  
 @OrderBy(**"name ASC"**)  
 @JoinTable(name = **"object\_epoch"**, joinColumns = @JoinColumn(name = **"object\_id"**), inverseJoinColumns = @JoinColumn(name = **"epoch\_id"**))  
 **private** Set<Epoch> **epochList** = **new** HashSet<>();  
  
 @ManyToMany(fetch = FetchType.***LAZY***, cascade ={CascadeType.***MERGE***, CascadeType.***PERSIST***, CascadeType.***REFRESH***})  
 @OrderBy(**"name ASC"**)  
 @JoinTable(name = **"object\_type"**, joinColumns = @JoinColumn(name = **"object\_id"**), inverseJoinColumns = @JoinColumn(name = **"type\_id"**))  
 **private** Set<Type> **typeList** = **new** HashSet<>();  
  
 @ManyToMany(fetch = FetchType.***LAZY***, cascade ={CascadeType.***MERGE***, CascadeType.***PERSIST***, CascadeType.***REFRESH***})  
 @OrderBy(**"name ASC"**)  
 @JoinTable(name = **"object\_style"**, joinColumns = @JoinColumn(name = **"object\_id"**), inverseJoinColumns = @JoinColumn(name = **"style\_id"**))  
 **private** Set<Style> **styleList** = **new** HashSet<>();  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 **private** Photo **cover**;

Рассмотрим некоторые аннотации в данной модели. @Entity говорит о том что этот объект будет обрабатываться hibernate, @Table(name= «object») , что он будет храниться в таблице object. @Column - аннотация для указания имени столбца, с которым связано поле класса.

@ManyToOne - аннотация указывает, что поле указывает на другой класс, который связан с текущим классом связью многие-к-одному. Здесь также указывается правило каскада cascade= {CascadeType.REFRESH} - в упрощенном виде оно гласит "что сделали с тобой, то сделай и со связью". В нашем случае единственная операция, которая будет передаваться на класс - это перечитывание. Все остальные операции не будут распространяться на связанный класс. Запись fetch=FetchType.LAZY указывает, что загрузка данного поля будет только в случае обращения к данному полю. Пока программа этого не делает, поле будет пустым.

* 1. Реализация уровня представления

Слой представления приложения обеспечивает отображение страниц и взаимодействие с пользователем. Слой представления – это часть приложения, которую видит пользователь и с которой он взаимодействует, поэтому этот слой должен удовлетворять многим требованиям. К ним относятся общие факторы, такие как удобство использования, дизайн и возможность взаимодействия с пользователем. Неудовлетворительное взаимодействие с пользователем будет иметь чрезвычайно негативные последствия для приложения, удачного во всех остальных отношениях. При проектировании приложения важно с самого начала обеспечить привлекательный и интуитивный интерфейс, поскольку на взаимодействие с пользователем оказывают влияние многие аспекты архитектуры приложения.

Один из важных моментов в реализации приложения, от которого во многом зависит успех программного продукта - дизайн. При подготовке дизайна моего приложения, были поставлены цели следовать основным тенденциям в области дизайна, но при этом, сделать упор на удобство пользователя, простоту интерфейса и интуитивно понятную навигацию. Учитывая особенности целевой аудитории было решено выбрать нейтральные цвета. В дизайн-концепции продемонстрировано цветовое оформление информационно-туристического портала, см. рисунок 4.1, расположение блоков, данных и других элементов.

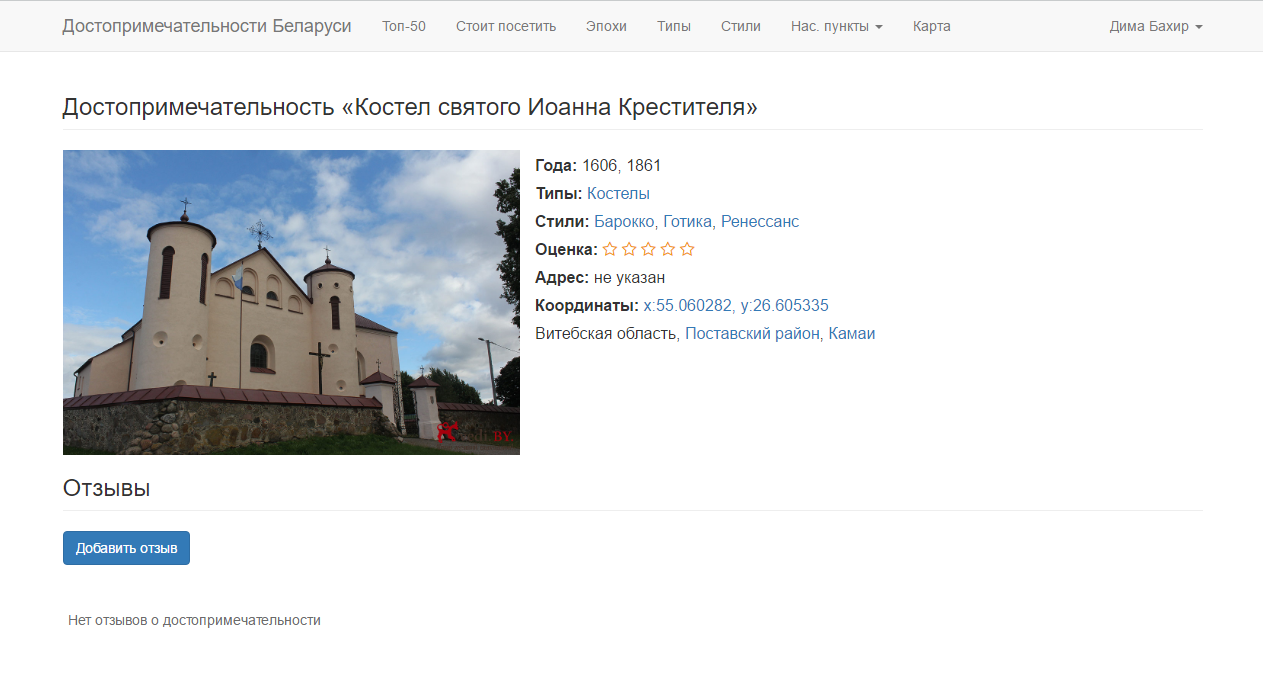


Рисунок 4.1 – Страница информации о достопримечательности.

Все веб-страницы разработанной системы написаны с использованием технологии Java Server Pages.

Технология проектирования Java Server Pages (JSP) - это одна из технологий J2EE, которая представляет собой расширение технологии сервлетов для упрощения работы с Web-содержимым. Страницы JSP позволяет легко разделить Web-содержимое на статическую и динамическую часть, допускающую многократное использование ранее определенных компонентов. Разработчики Java Server Pages могут использовать компоненты JavaBeans и создавать собственные библиотеки нестандартных тегов, которые инкапсулируют сложные динамические функциональные средства.

Спецификация Java Server Pages наследует и расширяет спецификацию сервлетов. Как и сервлеты, компоненты JSP относятся к компонентам Web и располагаются в Web-контейнере. Страницы JSP не зависят от конкретной реализации Web-контейнера, что обеспечивает возможность их повторного использования.

В дополнение к классам и интерфейсам для программирования сервлетов (пакеты javax.servlet и javax.servlet/http), в пакетах javax.servlet.jsp и javax.servlet.jsp.target содержатся классы и интерфейсы, относящиеся к программированию Java Server Pages.

Так выглядит код представление страницы объекта:

<%@**page pageEncoding**="**UTF-8**" %>  
<%@ **taglib prefix**="**spring**" **uri**="**http://www.springframework.org/tags**" %>  
<%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %>  
<%@ **taglib prefix**="**security**" **uri**="**http://www.springframework.org/security/tags**" %>  
<%@ **taglib prefix**="**fn**" **uri**="**http://java.sun.com/jsp/jstl/functions**" %>  
<%@ **taglib prefix**="**fmt**" **uri**="**http://java.sun.com/jsp/jstl/fmt**" %>  
<**security:authorize access="isAnonymous()" var="isAnonymous"**/>  
<**script**>  
 **function** *showRatingForm*() {  
 **$**(**'#rating\_form'**).**slideDown**();  
 **$**(**'#addButton'**).attr(**'onclick'**, **'sendComment();'**);  
 }  
 **function** *sendComment*() {  
 **var** form = **$**(**'#commentForm'**);  
 form.**parsley**().validate();  
 **if** (!form.**parsley**().isValid()){  
 **return**;  
 }  
 **var** rate = **$**(**'input[name="rating"]:checked'**).val();  
 **var** text = **$**(**'#addComment'**).val();  
 **$**.post(**'/sendComment'**, {**objectId**: **${**object.id**}**, **rate**: rate, **text**: text}, **function** (d) {  
 **$**(**'#addButton'**).remove();  
 **if**(d) {  
 *message*(**'rating\_form'**, **'Отзыв успешно добавлен и будет опубликован после проверки администрацией.'**, **'info m-b-0'**, 1, 1);  
 } **else** {  
 *message*(**'rating\_form'**, **'Произошла ошибка.'**, **'info m-b-0'**, 1, 1);  
 }  
 });  
 }  
</**script**>  
<**div class="col-md-12"**>  
  
 <**div class="row"**>  
 <**div class="col-lg-5"**>  
 <**a href="/object/${**object.id**}/gallery"**>  
 <**img style="width**:100%**"  
 src="**<**c:if test="${**!**empty** object.cover.file**}"**>**/photo?name=${**object.cover.file**}**</**c:if**><**c:if test="${empty** object.cover.file**}"**>**/resources/images/noimage.jpg**</**c:if**>**"  
 alt=""**>  
 </**a**>  
 </**div**>  
 <**div style="font-size**: 16**px**;**margin-bottom**:6**px"**>  
 <**b**>Типы:</**b**> **${**fn:length(object.typeList) == 0 ? **'не указаны'** : **''}** <**c:forEach var="t" items="${**object.typeList**}" varStatus="i"**>  
 <**a href="/type/${**t.id**}/objects"**>**${**t.name**}**</**a**>**${**(fn:length(object.typeList) - 1) > i.index ? **','** : **''}** </**c:forEach**>  
 </**div**>  
 <**div style="font-size**: 16**px**;**margin-bottom**:6**px"**>  
 <**b**>Стили:</**b**> **${**fn:length(object.styleList) == 0 ? **'не указаны'** : **''}** <**c:forEach var="s" items="${**object.styleList**}" varStatus="i"**>  
 <**a href="/style/${**s.id**}/objects"**>**${**s.name**}**</**a**>**${**(fn:length(object.styleList) - 1) > i.index ? **','** : **''}** </**c:forEach**>  
 </**div**>  
 <**div style="font-size**: 16**px**;**margin-bottom**:6**px"**><**b**>Оценка:</**b**>  
 <**span class="star-rating"**>  
 <**c:forEach begin="0" end="4" varStatus="i"**>  
 <**c:set var="num" value="${**i.index + 1**}"** />  
 <**c:if test="${**object.rating > num && object.rating < (num + 1)**}"**>  
 <**i class="fa fa-star-half-full" style="color**:**#f39c15**;**"**></**i**>  
 </**c:if**>  
 <**c:if test="${**(object.rating > num && object.rating >= (num + 1)) || object.rating == num**}"**>  
 <**i class="fa fa-star" style="color**:**#f39c15**;**"**></**i**>  
 </**c:if**>  
 <**c:if test="${**object.rating < num**}"**>  
 <**i class="fa fa-star-o" style="color**:**#f39c15**;**"**></**i**>  
 **${**fn:length(object.childObjects) > 0 ? **'<hr>'** : **''}** <**div**>  
 <**c:forEach var="e" items="${**object.childObjects**}"**>  
 <**div class="col-md-3" style="margin-left**: -15**px**;**"**>  
 <**div class="row" style="margin-bottom**: 15**px**;**"**>  
 <**div class="col-lg-12"**>  
 <**a href="/object/${**e.id**}"**>  
 <**img style="width**:100%;**"  
 src="**<**c:if test="${**!**empty** e.cover.file**}"**>**/photo?name=${**e.cover.file**}**</**c:if**><**c:if test="${empty** e.cover.file**}"**>**/resources/images/noimage.jpg**</**c:if**>**"  
 alt=""**>  
 </**a**>  
 <**div class="object-title" style="margin-bottom**:3**px**;**margin-top**:4**px"**><**a  
 href="/object/${**e.id**}"**><**b**>**${**e.name**}**</**b**></**a**></**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
 </**c:forEach**>  
 </**div**>  
</**div**>  
<**script**>  
 **var *logID*** = **'log'**,  
 ***log*** = **$**(**'<div id="'**+***logID***+**'"></div>'**);  
 **$**(**'body'**).append(***log***);  
 **$**(**'[type\*="radio"]'**).change(**function** () {  
 **var** me = **$**(**this**);  
 ***log***.html(me.attr(**'value'**));  
 });  
</**script**>  
  
<**style**>  
:**root** {  
 **font-size**: 1.7**em**;  
 **font-family**: **Helvetica**, **arial**, **sans-serif**;  
}  
</**style**>  
  
  
<**div class="col-md-12" style="margin-bottom**:10**px**;**margin-top**:10**px"**>  
 <**h3 class="page-header" style="margin-top**: 10**px**;**"**>Отзывы</**h3**>  
 <**c:choose**>  
 <**c:when test="${**commentedStatus == 3**}"**>  
 <**div class="no-info"**>  
 Чтобы оставить отзыв, <**a href="/login"**>войдите</**a**> или <**a href="/registration"**>зарегистрируйтесь</**a**>  
 </**div**>  
 </**c:when**>  
 <**c:when test="${**commentedStatus == 1**}"**>  
 <**div class="alert alert-info alert-styled-left alert-arrow-left alert-bordered"**>  
 <**span class="alrt-msg"**>Ваш отзыв будет опубликован после проверки администрацией.</**span**>  
 </**div**>  
 </**c:when**>  
 <**c:when test="${**commentedStatus == 2**}"**></**c:when**>  
 <**c:otherwise**>  
 <**form id="commentForm"**>  
 <**div class="form-group"**>  
 <**textarea class="form-control" name="addComment" id="addComment" rows="5" style="resize**:**vertical**;**" required**></**textarea**>  
 </**div**>  
 </**form**>  
 </**div**>  
  
 <**div class="form-group"**>  
 <**button class="btn btn-primary" onclick="***showRatingForm*();**" id="addButton"**>Добавить отзыв</**button**>  
 </**div**>  
 </**c:otherwise**>  
 </**c:choose**>  
</**div**>  
  
<**div class="col-md-12"**>  
 <**c:if test="${**fn:length(comments) == 0**}"**>  
 <**div class="no-info" style="padding-left**:5**px"**>  
 Нет отзывов о достопримечательности  
 </**div**>  
 </**c:if**>  
 <**c:forEach var="c" items="${**comments**}"**>  
 <**div class="panel panel-default"**>  
 <**div class="panel-heading"**>  
 <**strong**>**${**c.user.fio**}**</**strong**>  
 <**c:forEach begin="0" end="4" varStatus="i"**>  
 <**c:set var="num" value="${**i.index + 1**}"** />  
 <**c:if test="${**c.mark > num || c.mark == num**}"**>  
 <**i class="fa fa-star" style="color**:**#f39c15**;**"**></**i**>  
 </**c:if**>  
 <**c:if test="${**c.mark < num**}"**>  
 <**i class="fa fa-star-o" style="color**:**#f39c15**;**"**></**i**>  
 </**c:if**>  
 </**c:forEach**>  
 </**div**>  
 <**div class="panel-body"**>  
 <**div class="text-muted" style="font-size**:13**px**;**margin-bottom**:10**px"**>  
  
 <**fmt:formatDate value="${**c.date**}" pattern="dd.MM.yyyy в HH:mm:ss" var="createDate"** />  
 **${**createDate**}** </**div**>  
 **${**c.text**}** </**div**>  
 </**div**>  
 </**c:forEach**>  
</**div**>

* 1. Реализация бизнес-логики

Основной уровень приложения, представляющий бизнес-логику, достаточно объемен и его рассмотрение будет представлено поверхностно, описывая основные подходы к реализации того или иного функционала, используемого в приложении.

Бизнес-логика построена на основе сервисов (содержит интерфейсы, в которых описано что делать с данными. Также содержит практические реализации этих интерфейсов.). Рассмотрим на примере сущности Comment.

Одной из основных концепций Spring Data является абстракция Repository. Эта абстракция репозитория представляет собой оболочку вокруг базового интерфейса JPA EntityManager и предлагает более широкие возможности для доступа к данным.

Для начала для Comment создаём Repository:

**public interface** CommentRepository **extends** JpaRepository<Comment, Long> {  
 List<Comment> findByCheckedIsTrue();  
  
 List<Comment> findByCheckedIsFalse();  
  
 List<Comment> findByObjectIdAndUserId(Long objectId, Long userId);  
  
 List<Comment> findByObjectIdAndCheckedIsTrueOrderByDateDesc(Long objectId);  
}

Был создан интерфейс с именем CommentRepository, унаследованный от JpaRepository.

JpaRepository — это интерфейс фреймворка Spring Data предоставляющий набор стандартных методов JPA для работы с БД.

Напрямую использовать Repositories для получение данных на пользовательский интерфейс не принято и считается плохим тоном, для этого были придуманы Services.

Создаётся пакет service и в нем сервис который будет предоставлять конечные данные для пользовательского интерфейса.

Создаем новый интерфейс CommentService:

**public interface** CommentService {  
 List<Comment> findChecked();  
  
 List<Comment> findNotChecked();  
  
 Integer getNotCheckedCount();  
  
 Comment findOne(Long id);  
  
 **void** delete(Long id);  
  
 Boolean check(Long id);  
  
 Boolean send(Long objectId, Integer rate, String text, Long userId);  
  
 Integer checkCommented(Long objectId);  
  
 List<Comment> getComments(Long objectId);  
}

Далее реализуем интерфейс в классе CommentServiceImpl

@Service  
**public class** CommentServiceImpl **implements** CommentService {  
  
 @Autowired  
 **private** CommentRepository **commentRepository**;  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @Autowired  
 **private** ObjectRepository **objectRepository**;  
  
 **public** List<Comment> findChecked() {  
 **return commentRepository**.findByCheckedIsTrue();  
 }  
  
 **public** List<Comment> findNotChecked() {  
 **return commentRepository**.findByCheckedIsFalse();  
 }  
  
 **public** Integer getNotCheckedCount() {  
 **return commentRepository**.getNotCheckedCount();  
 }  
  
 **public** Comment findOne(Long id) {  
 **return commentRepository**.findOne(id);  
 }  
  
 **public void** delete(Long id) {  
 **commentRepository**.delete(id);  
 }  
  
 **public** Boolean check(Long id) {  
 Comment comment = **commentRepository**.findOne(id);  
 comment.setChecked(**true**);  
 comment.setModerator(**userService**.getUser());  
 **return commentRepository**.save(comment) != **null**;  
 }  
  
 **public** Boolean send(Long objectId, Integer rate, String text, Long userId) {  
 Comment c = **new** Comment();  
 c.setDate(**new** Date());  
 c.setChecked(**false**);  
 c.setMark(rate);  
 c.setObject(**objectRepository**.findOne(objectId));  
 c.setUser(**userService**.findOne(userId));  
 c.setText(text);  
 **return commentRepository**.save(c) != **null**;  
 }  
  
 **public** List<Comment> getComments(Long objectId) {  
 **return commentRepository**.findByObjectIdAndCheckedIsTrueOrderByDateDesc(objectId);  
 }  
  
 **public** Integer checkCommented(Long objectId) {  
 **if**(!**userService**.isAuthentificated())  
 **return** 3;  
 List<Comment> comments = **commentRepository**.findByObjectIdAndUserId(objectId, **userService**.getUser().getId());  
 **if**(!comments.isEmpty()) {  
 Comment c = comments.get(0);  
 **if**(!c.getChecked()) {  
 **return** 1;  
 } **else** {  
 **return** 2;  
 }  
 }  
 **return** 0;  
 }  
}

И так, в реализации CommentService есть метод getNotCheckedCount(), в стандартных средствах Spring Data нет такой возможности, поэтому мы напишем свой кастомный метод.

Для этого в интерфейс CommentRepository прописываем следующий метод:

**public interface** CommentRepository **extends** JpaRepository<Comment, Long> {  
  
 @Query(**"select count(f) from Comment f where f.checked = 0"**)  
 Integer getNotCheckedCount();  
  
}

Была использована аннотацию @Query которая позволяет создать SQL запрос.

Остальная бизнес логика в проекте реализовывается аналогичным образом.

## 5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

При помощи тест кейсов было проведено тестирование сайта.

Тест-кейс — это такое описание проверки работы системы, которое может выполнить любой человек из команды, будь то тестировщик, разработчик, аналитик или даже бизнес-заказчик.[10].

Стандартные атрибуты тест-кейса:

Номер — уникальный идентификатор тест-кейса.[11]. Его удобно использовать для одинакового понимания, о какой проверке идет речь (например, дать ссылку в баге).

Тестируемая функциональность — краткое описание сути проверки. Должно быть кратко, но емко.

Последовательность действий — описание действий, необходимых для проверки (например, создание элемента).

Ожидаемый результат (ОР) — сама проверка: что мы ожидаем получить после выполнения шагов ("Элемент создан").

Тестирование было произведено на популярных веб-браузерах: Opera, Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox.

В таблице 5.1 представлены тест-кейсы для пользователя. В таблице 5.2 представлены тест-кейсы для администратора.

Таблица 5.1 – Набор тест-кейсов для роли «Пользователь»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № тест-кейса | Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Полученый результат |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Регистрация пользователя | 1. Войти на сайт  2. Нажать «Регистрация»  3. Ввести все данные и нажать кнопку «Зарегистрироваться» | 1. Отображение главной страницы  2. Отображение формы регистрации  3. Отображение страницы настроек, появление сообщения «Регистрация завершена. Теперь вы можете указать  свои предпочтения» | Тест пройден успешно |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 2 | Авторизация пользователя | 1. Войти на сайт  2. Нажать «Вход»  3. Ввести все данные и нажать на кнопку «Войти» | 1. Отображение главной страницы  2. Отображение страницы авторизации  3. Отображение главной страницы, в левом верхнем угле отображается имя и фамилия пользователя. | Тест пройден успешно |
| 3 | Подбор достопримечательностей на основе предпочтений | 1. Авторизироваться на сайте  2. В левом верхнем угле нажить на своё имя и фамилию  3. Выбрать пункт «настройки»  4. Выбрать интересующие эпохи, типы, стили и нажать кнопку «сохранить»  5. Перейти в раздел «Стоит посетить» | 1. Отображение главной страницы, в левом верхнем угле отображается имя и фамилия пользователя.  2. Раскрывается подменю.  3. Отображение страницы с настройкой предпочтений пользователя  4. Предпочтения сохранятся, появится сообщение «Предпочтения успешно сохранены»  5. Отображение страницы «Стоит посетить» где содержится список достопримечательностей на основе предпочтений пользователя | Тест пройден успешно |

Завершение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 4 | Добавление отзыва на достопримечательность | 1. Авторизироваться на сайте  2. Перейти в раздел «Эпохи»  3. Выбрать эпоху и нажать на кнопку «Показать все объекты»  4.Выбрать объект и нажать на название  5.Нажать кнопку «оставить отзыв»  6. Написать отзыв, поставить оценку и нажать кнопку «отправить» | 1. Отображение главной страницы, в левом верхнем угле отображается имя и фамилия пользователя.  2. Отображение страницы «Эпохи» 3. Отображение страницы с объектами данной эпохи  4. Отображение страницы объекта  5. Отображение окна отзыва  6. Отзыв отправлен и появляется сообщение «Отзыв успешно добавлен и будет опубликован после проверки администрацией.» | Тест пройден успешно |

Таблица 5.2 – Набор тест-кейсов для роли «Администратор»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № тест-кейса | Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Полученый результат |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Добавление стиля | 1. Войти на администраторскую панель.  2. Заполнить форму авторизации и нажать кнопку «Вход» | 1. Отображение формы авторизации  2. Отображение главной страницы администраторской панели  3. Отображение страницы стилей | Тест пройден успешно |

Продолжение таблицы 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Добавление стиля | 3. Нажать на раздел «Стили»  4. Нажать кнопку «Создать стиль»  5. Заполнить форму и нажать кнопку «Сохранить» | 4. Отображение страницы добавления стилей  5. Отображение страницы стилей и появляется сообщение «Стиль успешно создан.» | Тест пройден успешно |
| 2 | Удаление стиля | 1. Войти на администраторскую панель.  2. Заполнить форму авторизации и нажать кнопку «Вход»  3. Нажать на раздел «Стили»  4. Выбрать интересующий стиль и нажать кнопку «Действия», далее нажать кнопку «Удалить»  5.Нажать кнопку «Удалить» | 1. Отображение формы авторизации  2. Отображение главной страницы администраторской панели  3. Отображение страницы стилей  4. Отображение всплывающего окна с сообщением «Вы действительно хотите удалить этот стиль?»  5. Стиль удален и появляется сообщение «Стиль успешно удален» | Тест пройден успешно |
| 3 | Редактирование стиля | 1. Войти на администраторскую панель.  2. Заполнить форму авторизации и нажать кнопку «Вход»  3. Нажать на раздел «Стили»  4. Выбрать интересующий стиль и нажать кнопку «Действия», далее нажать | 1. Отображение формы авторизации  2. Отображение главной страницы администраторской панели  3. Отображение страницы стилей  4. Отображение страницы редактирования стилей  5. Стиль отредактирован и появляется | Тест пройден успешно |

Завершение таблицы 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 3 | Редактирование стиля | кнопку «Редактировать»  5.Редактировать данный стиль и нажать кнопку «Сохранить» | сообщение «Стиль успешно отредактирован» | Тест пройден успешно |
| 4 | Добавление объекта | 1. Войти на администраторскую панель.  2. Заполнить форму авторизации и нажать кнопку «Вход»  3. Нажать на раздел «Объекты»  4. Нажать кнопку «Создать объект»  5. Заполнить форму и нажать кнопку «Сохранить» | 1. Отображение формы авторизации  2. Отображение главной страницы администраторской панели  3. Отображение страницы объектов  4. Отображение страницы добавления объектов  5. Отображение страницы объектов и появляется сообщение «Объект успешно создан.» | Тест пройден успешно |

## 6 ТЕХНИКО - ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ИНФОРМАЦИОННО-ТУРИСТИЧЕСКОГО ПОРТАЛА

6.1 Характеристика программного продукта

Программное средство, разрабатываемое в рамках рассматриваемого дипломного проекта, предназначено для упорядочения информации обо всех достопримечательностях Беларуси. Основная цель заключается в объединении данных обо всех достопримечательностях Беларуси.

Главной особенностью проекта является адаптация под основные виды устройств, удобная каталогизация достопримечательностей по типам, стилям, эпохам, населённым пунктам, подбор достопримечательностей на основе предпочтений, вывод ТОП-50 объектов на основе пользовательских отзывов. Пользование проектом проходит путем выбора интересующих нас стилей, эпох, типов, населённых пунктов и вывод всех объектов данного типа.

Воспользоваться программным средством смогут обычные пользователи которые заинтересованы в получении информации о достопримечательностях Беларуси.

6.2 Расчет затрат и отпускной цены программного средства

Объем ПО (строки исходного кода, LOC) считается по формуле (6.1).

 (6.1)

Перечень и объем функций программного модуля

Таблица 6.1 – Перечень и объем функций программного модуля

|  |  |  |  |
| --- | --- | --- | --- |
| № функции | Наименование (содержание) функции | Объем функции (LОС) | |
| По каталогу Vi | УточненныйVyi |
| 1 | 2 | 3 | 4 |

Продолжение таблицы 6.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 101 | Организация ввода информации | 150 | 150 |
| 102 | Контроль, предварительная об- работка и ввод информации | 450 | 450 |
| 201 | Генерация структуры базы данных | 4300 | 3276 |
| 207 | Манипулирование данными | 9550 | 5645 |
| 208 | Организация поиска и поиск в базе данных | 5480 | 3178 |
| 210 | Загрузка базы данных | 2780 | 1239 |
|  | Итого | 22710 | 13938 |

Среда разработки ПО (IntelliJ IDEA, MySQL 5.4), в связи с использованием более совершенных средств автоматизации, а также фреймворков, готовых библиотек функции 201, 207, 208, 210 были уменьшены и уточненный объем ПО (Vyi) составил 13938 LOC вместо 22710 LOC.

Трудоемкость разработки ПО

Категории сложности ПО – 2-я категория, Vi = 13938 LOC;  
Нормативная трудоемкость ПО (Tн) – 333 чел./дн.;

Коэффициент сложности (Kc) – 0,12;

Коэффициент, учитывающий степень использования при разработке

ПО стандартных модулей (Kт) – 0,8;

Коэффициент новизны разрабатываемого ПО (Kн) – 0,7;

Общая трудоемкость находится из формулы (6.2):

Tо = Tн⋅Kс⋅Kт⋅Kн = 333⋅1,12⋅0,8⋅0,7 = 209 чел./дн.; (6.2)

Численность исполнителей проекта (Чp) считается по формуле (6.3)

, (6.3)

где Tp− срок разработки проекта (лет);

Эффективный фонд времени работы одного работника (Фэф) считается по формуле (4.4):

Фэф = Дг-Дп-Дв-До = 236 дн.; (6.4)

Исходя из формулы (4.5) срок разработки установлен 4 мес. (Tp = 0,33 г.):

(6.5)

6.3 Расчет сметы затрат и цены заказного ПО

Расчет основной заработной платы исполнителей

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле (6.6):

Зоi = ∑ni=1 \* Зci \* Фрi \* К, (6.6)

где n− количество исполнителей, занятых разработкой конкретного ПО;

Зci −среднедневная заработная плата i-го исполнителя (*д.е.*);

Фрi − плановый фонд рабочего времени i-го исполнителя (дн.);

К − коэффициент премирования (1.2).

Расчет основной заработной платы представлен в табл. 6.2.

Таблица 6.2 – Расчет основной заработной платы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Средне-дневная заработная плата, руб. | Плановый фонд рабочего времени, дн. | Коэф-фици-ент преми-рования | Основная заработ-ная плата, руб. |
| Ведущий разработчик | 5.6 | 98 | 1.2 | 658.56 |
| Разработчик | 4.1 | 98 | 1.2 | 482.16 |
| Разработчик | 4.1 | 98 | 1.2 | 482.16 |
| Итого с премией (20%), Зо | - | - | - | 1622.88 |

Дополнительная заработная плата исполнителей проекта

Дополнительная заработная плата включает выплаты, рассчитывается по формуле (6.7), предусмотренные законодательством от труда и определяется по нормативу в процентах от основной заработной платы

 (6.7)

где НД – норматив дополнительной заработной платы (20%).

Дополнительная заработная плата составит:

Зд = 1622.88 ·20/100 = 324.58 руб.

Отчисления в фонд социальной защиты населения и на обязательное страхование

Отчисления в фонд социальной защиты населения и на обязательное страхование определяются в соответствии с действующими законодательными актами по формуле (6.8):

 (6.8)

где НСЗ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34 +0,6%).

Зсз = (1622.88 +324.58) ·34.6 / 100 = 673.82 руб.

Расходы по статье «Машинное время»

Расходы по статье «Машинное время» (РМ) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле (6.9):

Рм = Цм \* Тч \* Ср, (6.9)

где Цм – цена одного машино-часа;

Тч – количество часов работы в день;

Ср – длительность проекта.

Стоимость машино-часа на предприятии составляет 0.5 руб.. Разработка проекта займет 98 дней. Определим затраты по статье “Машинное время”:

Рм = 0.5·8·98 = 393 руб.

Затраты по статье «Накладные расходы»

Затраты по статье «Накладные расходы» (РН), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды (РН), определяются по формуле (6.10)

 (6.10)

где НРН – норматив накладных расходов (50%).

Накладные расходы составят:

Рн = 1622.88 · 0.5 = 811.44 руб.

Общая сумма расходов по всем статьям сметы (Сп) на ПО рассчитывается по формуле (6.11):

 (6.11)

Сn =1622.88 + 324.58 + 673.82 + 393 + 811.44 = 3825.72 руб.

Прибыль ПС рассчитывается по формуле (6.12):

, (6.12)

где Ппс – прибыль от реализации ПС заказчику (руб.);

УР – уровень рентабельности ПС (25%);

СП – себестоимость ПС (руб.).

Ппс = 3825.72 · 25/100 = 956.43 руб.

Прогнозируемая отпускная цена ПС вычисляется по формуле (6.13):

. (6.13)

Цп = 3825.72 + 956.43 = 4782.13 руб.

Налог на добавленную стоимость (НДСi) вычисляется по формуле (6.14):

НДС= ЦП \* НДС : 100, (6.14)

где Hдc− норматив НДС (%).

НДС= 4782.13 руб. \* 20% : 100% = 956.43 руб.

Прогнозируемая отпускная цена (Цo) вычисляется по формуле (6.15):

Цo = Цп+ НДС (6.15)

Цo = 4782.13 + 956.43 = 5738.56 руб.

Затраты на освоение ПО:

Организация-разработчик учавствует в освоении ПО и несет соответствующие затраты. Норма затраты на освоение ( Но = 10%) от себестоимости в расчете на 3 месяца вычисляются по формуле (6.16) :

(6.16)

= 382.57 руб

Кроме того, организация-разработчик осуществляет затраты на сопровождение ПС (РС), которые определяются по нормативу (НС) формула (6.17), где Нс – норматив расходов на сопровождение и адаптацию (20%).

 (6.17)

где Рс – расходы на сопровождение и адаптацию ПС в целом по организации (руб.);

СР – смета расходов в целом по организации без расходов на сопровождение и адаптацию (руб.).

Рс = 3825.72 · 20 / 100 = 765.14 руб.

6.4 Оценка экономической эффективности применения программного средства у пользователя

Расчет экономии основных видов ресурсов в связи   
с использованием нового программного средства представлен в таблице 6.3.

Таблица 6.3 – Общие капитальные вложения пользователя

|  |  |
| --- | --- |
| Затраты | Значения |
| Затраты на приобретение по отпускной цене |  |
| Затраты на освоение |  |
| Затраты на сопровождение | 765.14 Br |
| Затраты на доукомплектование ВТ | 975.00 Br |
| Затраты на пополнение оборотных средств | 75.00 Br |
| Капитальные затраты | 7936.27 Br |

Исходные данные для расчета экономии ресурсов в связи с применением нового программного средства

Таблица 6.4 – Экономия ресурсов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование показателей | Обозна-  чения | Единицы измерения | Значение показателя | | Наименование источника информации |
| в базовом варианте | в новом варианте |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1. Капитальные вло-жения, включая зат-раты пользователя на приобретение | Kпp | руб. |  |  | Договор за-казчика с раз-работчиком |

Продолжение таблицы 6.4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 2. Затраты на сопро-вождение ПО | Кс | руб. |  | 765.14 | Договор за-казчика с раз-работчиком |
| 3. Время простоя сер-виса, обусловленное ПО, в день | П1,П2 | мин | 50 | 4 | Расчетные данные поль-зователя и паспорт ПО |
| 4. Стоимость одного часа простоя | Сп | руб. | 20 | 20 | Расчетные данные поль-зователя и паспорт ПО |
| 5. Среднемесячная ЗП одного програм-миста | Зсм | руб. | 700 | 700 | Расчетные  данные поль-зователя |
| 6. Коэффициент на-числений на зарплату | Кнз |  | 1,2 | 1.2 | Рассчитывает-ся по данным пользователя |
| 7. Среднемесячное количество рабочих дней | Д р | день |  | 20.2 | Принято для расчета |
| 8. Количество типо-вых задач, решаемых за год | Зт1 ,Зт2 | задача | 1700 | 1700 | План пользо-вателя |
| 9. Объем выполня-емых работ | А1, А2 | задача | 1700 | 1700 | План пользо-вателя |
| 10. Средняя трудоем-кость работ на задачу | Tc1,  Тс2 | Человеко-часов | 2 | 0.2 | Рассчитывает-ся по данным пользователя |
| 11. Количество часов работы ты в день | Tч | ч | 8 | 8 | Принято для расчета |
| 12. Ставка налога на прибыль | Hп | % |  | 18 |  |

Экономия затрат на заработную плату в расчете на 1 задачу (Cзe) определяется по формуле (6.18):

, (6.18)

где Зcм− среднемесячная заработная плата одного программиста (руб.);

Tc1, Tc2− снижение трудоемкости работ в расчете на 1 задачу (человеко-часов);

Tч− количество часов работы в день (ч);

Дp− среднемесячное количество рабочих дней.

Сзе = 700\*(2-0.2)/(8\*20.2) = 7.80 руб

Экономия заработной платы при использовании нового ПО (руб.) вычисляется по формуле (6.19):

Сз=СзеА2, (6.19)

где Сз− экономия заработной платы;

А2− количество типовых задач, решаемых за год (задач).

Сз = 7.80\*1700 = 13260 руб.

Экономия с учетом начисления на зарплату (Сн):

*Cн= Cз****ּ*** 1,5 *=* 13260 \*1,2 *=* 15912 руб.

Экономия за счет сокращения простоев сервиса (Сс) определяется по формуле (6.20):

, (6.13)

где Дрг – плановый фонд работы сервиса (дней).

Сс = (50-4)\* 20.2\*20 / 60 = 309.7 руб.

Общая готовая экономия текущих затрат

Общая готовая экономия текущих затрат связанных с использованием нового ПО (Со), рассчитывается по формуле (6.21):

Co= Cн + Cc (6.21)

Co= 15912 + 309.7 = 16221.7 руб.

* 1. Расчет экономического эффекта

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль − дополнительная прибыль, остающаяся в его распоряжении (ΔПЧ), которая определяется по формуле (6.22):

, (6.22)

где Hп− ставка налога на прибыль (%).

ΔПЧ = 16221.7 – (16221.7 \* 0.18) = 13301.79 руб.

В процессе использования нового ПО чистая прибыль в конечном итоге  
возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени − расчетному году (за расчетный год принят 2017-й год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α. В данном примере используются коэффициенты: 2017 г. – 1, 2018-й – 0,8696, 2019-й – 0,7561, 2020 г. – 0,6575. Все рассчитанные данные экономического эффекта сводятся в следующую таблицу.

Расчет экономического эффекта от использования нового программного средства приведён в таблице 6.5

Таблица 6.5 – экономический эффект от использования нового программного средства

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Единицы  измерения | Годы | | | |
| 2017 | 2018 | 2019 | 2020 |
| *Результаты* |  |  |  |  |  |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 13301.79 | 13301.79 | 13301.79 |
| То же с учетом фактора времени | руб. |  | 11567.24 | 10057.48 | 8745.93 |
| *Затраты* |  |  |  |  |  |
| Приобретение ПО (Кпр) | руб. |  |  |  |  |
| Освоение ПО (Koc) | руб. |  |  |  |  |
| Сопровождение (Кс) | руб. | 765.14 |  |  |  |
| Доукомплектование ВТ техническими средствами (Kтc) | руб. | 975.00 |  |  |  |
| Пополнение оборотных средств (Koб) | руб. | 75.00 |  |  |  |
| Всего затрат | руб. | 6503.7 |  |  |  |
| То же с учетом фактора времени | руб. | 6503.7 |  |  |  |

Продолжение таблицы 6.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| *Экономический эффект* |  |  |  |  |  |
| Превышение результата над затратами | руб. | -6503.7 | 11567.24 | 10057.48 | 8745.93 |
| То же с нарастающим итогом | руб. | -6503.7 | 5063.54 | 15121.02 | 23866.95 |
| Коэффициент приведе-ния | единицы | 1 | 0,8696 | 0,7561 | 0,6575 |

6.6 Резюме

Таким образом мы провели технико-экономическое обоснование разрабатываемого проекта, составив смету затрат и рассчитав прогнозируемую прибыль, и показали экономическую целесообразность его создания.

Реализация проекта ПО позволит заказчику снизить трудоемкость решения задач, сократить простои сервиса.

Как видно из таблицы вложения окупят себя уже через год использования программного обеспечения.

Проект представляется эффективным и полезным для заказчика.

**ЗАКЛЮЧЕНИЕ**

В рамках данного дипломного проекта было разработан информационно-туристический портал. Перед разработкой программного средства были изучены теоретические основы разработки веб-приложений, существующие подходы и технологии, для эффективной разработки.

Также еще до непосредственного старта разработки приложения был произведен качественный анализ предметной области. Была продумана и описана архитектура будущего приложения. Приложение было реализовано согласно разработанной архитектуре. Также было проведено тестирование приложения, что помогло выявить ошибки реализации приложения и их исправить.

Разработанное приложение несет практическую пользу, являясь качественным сервисом, облегчающим поиск достопримечательностей, за счёт удобной каталогизации, карты объектов.

Основными достоинствами разработанной платформы являются:

-Подбор достопримечательностей на основе предпочитаемых стилей, эпох, типов.

-Отображение координат объектов на карте Google

-Подбор достопримечательностей на основе рейтинга пользователей

К дальнейшим направлениям развития разработки следует отнести следующие пункты:

-Отображение координат объектов на популярных картах OpenStreetMaps, Yandex

-Создание форума пользователей для обмена информацией о достопримечательностях

-Усложнение методов подбора достопримечательностей на основе предпочтений и отзывов пользователей

Исходя из списка возможностей программного средства, можно сделать вывод, что реализация программного средства соответствует поставленным задачам.

Так же, в ходе выполнения работы были получены новые знания и структурированы знания, полученные за время обучении.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] https://ru.wikipedia.org/wiki/Система\_управления\_базами\_данных- Интернет энциклопедия Википедия, обзор СУБД

[2] В. Васвани. MySQL: использование и администрирование — М.: «Питер», 2011. — 368 с.

[3] Кларенс Хо, Роб Харроп. Spring 3 для профессионалов — М.: «Вильямс», 2012. — 880 с.

[4] Основы JavaScript [Электронный ресурс]. – Режим доступа: <https://learn.javascript.ru/first-steps>

[5] Герберт Шилдт. Java 8. Полное руководство, 9-е издание — М.: «Вильямс», 2015. — 1376 с.

[6] Адам Фримен. jQuery для профессионалов — М.: «Вильямс», 2012. — 960 с.

[7] Spring Data JPA [Электронный ресурс]. – Режим доступа: <http://devcolibri.com/4149>

[8] Аннотации в Hibernate [Электронный ресурс]. – Режим доступа: http://www.javaportal.ru/java/articles/hibernate\_annotations.html

[9] Кузнецов С. Д. Основы баз данных. — 2-е изд. — М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. — 484 с.

[10] Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд — М.: «Вильямс», 2010. — 464 с.

[11] Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. — Киев: ДиаСофт, 2001. — 544 с.

[12] Палицын В.А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. – Мн.: БГУИР, 2006. – 76 с.

**ПРИЛОЖЕНИЕ А**

**Исходный код котроллеров District, Epoch, Object, User**

**package** com.db.tourist.models;  
  
**import** javax.persistence.\*;  
**import** java.util.HashSet;  
**import** java.util.Set;

@Controller  
**public class** DistrictController {  
  
 @Autowired  
 **private** DistrictService **districtService**;  
  
 @Autowired  
 **private** RegionService **regionService**;  
  
 @ModelAttribute(**"regionList"**)  
 **public** Map populateRegions()  
 {  
 List<Region> regionList = **regionService**.findAll();  
 Map<String, String> map = **new** LinkedHashMap<String, String>();  
 **for**(Region r : regionList) {  
 map.put(Objects.*toString*(r.getId()), r.getName());  
 }  
 **return** map;  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/district/getDistricts"**, produces = {**"application/json; charset=UTF-8"**}, method = RequestMethod.***GET***)  
 **public** String getCustomers(@RequestParam(**"regionId"**) Long regionId) **throws** JSONException {  
 List<District> districts = **districtService**.findByRegionId(regionId);  
 JSONArray array = **new** JSONArray();  
 **for**(District u: districts) {  
 JSONObject item = **new** JSONObject();  
 item.put(**"name"**, u.getName());  
 item.put(**"id"**, u.getId());  
 array.put(item);  
 }  
 **return** array.toString();  
 }  
  
 @RequestMapping(value = **"/admin/district"**, method = RequestMethod.***GET***)  
 **public** ModelAndView list() {  
 View view = **new** View(**"district/list"**, **true**);  
 view.addObject(**"title"**, **"Районы"**);  
 view.addObject(**"districts"**, **districtService**.findAll());  
 **return** view;  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/district/delete"**, method = RequestMethod.***POST***)  
 **public void** delete(@RequestParam(**"id"**) Long id) {  
 **districtService**.delete(id);  
 }  
  
 @RequestMapping(value = **"/admin/district/add"**, method = RequestMethod.***GET***)  
 **public** ModelAndView add() {  
 View view = **new** View(**"district/edit"**, **true**);  
 view.addObject(**"title"**, **"Создание района"**);  
 view.addObject(**"district"**, **new** District());  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/district/add"**, method = RequestMethod.***POST***)  
 **public** String add(@ModelAttribute(**"district"**) District district, RedirectAttributes redirectAttributes) {  
 District d = **districtService**.save(district);  
 **if**(d != **null**) {  
 redirectAttributes.addFlashAttribute(**"success"**, **"Район "**+d.getName()+**" успешно создан"**);  
 }  
 **return "redirect:/admin/district"**;  
 }  
  
 @RequestMapping(value = **"/admin/district/edit/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView edit(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"district/edit"**, **true**);  
 view.addObject(**"title"**, **"Редактирование района"**);  
 view.addObject(**"district"**, **districtService**.findOne(id));  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/district/edit/{id}"**, method = RequestMethod.***POST***)  
 **public** String edit(District district, RedirectAttributes redirectAttributes) {  
 **if**(**districtService**.save(district) != **null**) {  
 redirectAttributes.addFlashAttribute(**"success"**, **"Район успешно отредактирован"**);  
 }  
 **return "redirect:/admin/district"**;  
 }  
}

@Controller  
**public class** EpochController {  
  
 @Autowired  
 **private** EpochService **epochService**;  
  
 @Autowired  
 **private** PhotoService **photoService**;  
  
 @Autowired  
 **private** ObjectService **objectService**;  
  
 @Transactional  
 @RequestMapping(value = **"/epoch/{epochId}/objects"**, method = RequestMethod.***GET***)  
 **public** ModelAndView objects(@PathVariable(**"epochId"**) Long id) {  
 View view = **new** View(**"objects"**);  
 Epoch e = **epochService**.findOne(id);  
 view.addObject(**"title"**, **"Достопримечательности эпохи «"** + e.getName() + **"»"**);  
 view.addObject(**"objects"**, **objectService**.setRatings(e.getObjectList()));  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/epochs"**, method = RequestMethod.***GET***)  
 **public** ModelAndView epochs() {  
 View view = **new** View(**"epochs"**);  
 view.addObject(**"title"**, **"Эпохи"**);  
 view.addObject(**"epochs"**, **epochService**.findAllByOrderByNameAsc());  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/epoch/{epochId}/gallery"**, method = RequestMethod.***GET***)  
 **public** ModelAndView epochs(@PathVariable(**"epochId"**) Long epochId) {  
 View view = **new** View(**"gallery"**);  
 Epoch epoch = **epochService**.findOne(epochId);  
 view.addObject(**"title"**, **"Фотоальбом эпохи «"** + epoch.getName() + **"»"**);  
 view.addObject(**"object"**, epoch);  
 **return** view;  
 }  
  
 @RequestMapping(value = {**"/admin"**, **"/admin/epoch"**}, method = RequestMethod.***GET***)  
 **public** ModelAndView list() {  
 View view = **new** View(**"epoch/list"**, **true**);  
 view.addObject(**"title"**, **"Эпохи"**);  
 view.addObject(**"epochs"**, **epochService**.findAll());  
 **return** view;  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/epoch/delete"**, method = RequestMethod.***POST***)  
 **public void** delete(@RequestParam(**"id"**) Long id) {  
 **epochService**.delete(id);  
 }  
  
 @RequestMapping(value = **"/admin/epoch/add"**, method = RequestMethod.***GET***)  
 **public** ModelAndView add() {  
 View view = **new** View(**"epoch/edit"**, **true**);  
 view.addObject(**"title"**, **"Создание эпохи"**);  
 view.addObject(**"epoch"**, **new** Epoch());  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/epoch/add"**, method = RequestMethod.***POST***)  
 **public** String add(@ModelAttribute(**"epoch"**) Epoch epoch, RedirectAttributes redirectAttributes) {  
 Epoch e = **epochService**.save(epoch);  
 **if**(e != **null**) {  
 redirectAttributes.addFlashAttribute(**"success"**, **"Эпоха успешно создана. Теперь вы можете добавить фотографии"**);  
 **return "redirect:/admin/epoch/photo/"** + e.getId();  
 }  
 **return "redirect:/admin/epoch"**;  
 }  
  
 @RequestMapping(value = **"/admin/epoch/edit/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView edit(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"epoch/edit"**, **true**);  
 view.addObject(**"title"**, **"Редактирование эпохи"**);  
 view.addObject(**"epoch"**, **epochService**.findOne(id));  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/epoch/edit/{id}"**, method = RequestMethod.***POST***)  
 **public** String edit(Epoch epoch, RedirectAttributes redirectAttributes) {  
 Epoch e = **epochService**.findOne(epoch.getId());  
 e.setName(epoch.getName());  
 e.setStartYear(epoch.getStartYear());  
 e.setFinishYear(epoch.getFinishYear());  
 e.setDescription(epoch.getDescription());  
 **if**(**epochService**.save(e) != **null**) {  
 redirectAttributes.addFlashAttribute(**"success"**, **"Эпоха успешно отредактирована"**);  
 }  
 **return "redirect:/admin/epoch"**;  
 }  
  
 @RequestMapping(value = **"/admin/epoch/photo/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView photos(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"photo"**, **true**);  
 Epoch epoch = **epochService**.findOne(id);  
 **if**(epoch != **null**) {  
 view.addObject(**"title"**, **"Фотоальбом эпохи «"** + epoch.getName() + **"»"**);  
 view.addObject(**"object"**, epoch);  
 view.addObject(**"objectName"**, **"epoch"**);  
 view.addObject(**"backBtnText"**, **"К списку эпох"**);  
 view.addObject(**"objectTitle"**, **"эпоху"**);  
 }  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/epoch/setCover"**, method = RequestMethod.***POST***)  
 **public** String setCover(@RequestParam(**"epochId"**) Long epochId, @RequestParam(**"coverId"**) Long coverId, RedirectAttributes ra) {  
 Epoch epoch = **epochService**.findOne(epochId);  
 epoch.setCover(**photoService**.findOne(coverId));  
 **epochService**.save(epoch);  
 **return "redirect:/admin/epoch/photo/"** + epoch.getId();  
 }  
  
 @RequestMapping(value = **"/admin/epoch/upload"**, method = RequestMethod.***POST***)  
 **public** Boolean saveFile(@ModelAttribute UploadedFile uploadedFile, @RequestParam(**"objectId"**) Long objectId) {  
 **return epochService**.uploadPhoto(uploadedFile, objectId);  
 }  
  
}

@Controller  
**public class** ObjectController {  
  
 @Autowired  
 **private** ObjectService **objectService**;  
  
 @Autowired  
 **private** EpochService **epochService**;  
  
 @Autowired  
 **private** StyleService **styleService**;  
  
 @Autowired  
 **private** TypeService **typeService**;  
  
 @Autowired  
 **private** RegionService **regionService**;  
  
 @Autowired  
 **private** DistrictService **districtService**;  
  
 @Autowired  
 **private** LocalityService **localityService**;  
  
 @Autowired  
 **private** PhotoService **photoService**;  
  
 @Autowired  
 **private** CommentService **commentService**;  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @RequestMapping(value = **"/object/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView object(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"object"**);  
 Object object = **objectService**.findOne(id);  
 **if**(object != **null**) {  
 view.addObject(**"title"**, **"Достопримечательность «"** + object.getName() + **"»"**);  
 view.addObject(**"commentedStatus"**, **commentService**.checkCommented(id));  
 view.addObject(**"object"**, object);  
 view.addObject(**"comments"**, **commentService**.getComments(id));  
 }  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/object"**, method = RequestMethod.***GET***)  
 **public** ModelAndView list() {  
 View view = **new** View(**"object/list"**, **true**);  
 view.addObject(**"title"**, **"Объекты"**);  
 List<Object> objects = **objectService**.findAll();  
 view.addObject(**"objects"**, objects);  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/object/{objectId}/gallery"**, method = RequestMethod.***GET***)  
 **public** ModelAndView epochs(@PathVariable(**"objectId"**) Long objectId) {  
 View view = **new** View(**"gallery"**);  
 Object object = **objectService**.findOne(objectId);  
 view.addObject(**"title"**, **"Фотоальбом достопримечательности «"** + object.getName() + **"»"**);  
 view.addObject(**"object"**, object);  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/object/childs/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView list(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"object/list"**, **true**);  
 Object object = **objectService**.findOne(id);  
 view.addObject(**"title"**, **"Дочерние объекты «"** + object.getName() + **"»"**);  
 view.addObject(**"objects"**, **objectService**.setRatings(object.getChildObjects()));  
 view.addObject(**"childs"**, **true**);  
 **return** view;  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/object/delete"**, method = RequestMethod.***POST***)  
 **public void** delete(@RequestParam(**"id"**) Long id) {  
 **objectService**.delete(id);  
 }  
  
 @RequestMapping(value = **"/admin/object/add"**, method = RequestMethod.***GET***)  
 **public** ModelAndView add() {  
 View view = **new** View(**"object/edit"**, **true**);  
 view.addObject(**"title"**, **"Создание объекта"**);  
 view.addObject(**"object"**, **new** Object());  
 view.addObject(**"epochs"**, **epochService**.findAll());  
 view.addObject(**"styles"**, **styleService**.findAll());  
 view.addObject(**"types"**, **typeService**.findAll());  
 view.addObject(**"regions"**, **regionService**.findAll());  
 view.addObject(**"districts"**, **districtService**.findAll());  
 view.addObject(**"localities"**, **localityService**.findAll());  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/object/add"**, method = RequestMethod.***POST***)  
 **public** String add(@ModelAttribute(**"object"**) Object object,  
 @RequestParam(value = **"epochs"**, required = **false**) List<Long> epochs,  
 @RequestParam(value = **"types"**, required = **false**) List<Long> types,  
 @RequestParam(value = **"styles"**, required = **false**) List<Long> styles,  
 @RequestParam(value = **"years"**, required = **false**) List<Integer> years,  
 RedirectAttributes redirectAttributes) {  
 Object o = **objectService**.save(object, epochs, types, styles, years);  
 **if**(o != **null**) {  
 redirectAttributes.addFlashAttribute(**"success"**, **"Объект успешно создан. Теперь вы можете добавить фотографии"**);  
 **return "redirect:/admin/object/photo/"** + o.getId();  
 }  
 **return "redirect:/admin/object"**;  
 }  
  
 @RequestMapping(value = **"/admin/object/edit/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView edit(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"object/edit"**, **true**);  
 view.addObject(**"title"**, **"Редактирование объекта"**);  
 Object o = **objectService**.findOne(id);  
 view.addObject(**"object"**, o);  
 view.addObject(**"epochs"**, **epochService**.findAll());  
 view.addObject(**"styles"**, **styleService**.findAll());  
 view.addObject(**"types"**, **typeService**.findAll());  
 view.addObject(**"regions"**, **regionService**.findAll());  
 **if**(o.getLocality() != **null**) {  
 view.addObject(**"districts"**, **districtService**.findByRegionId(o.getLocality().getDistrict().getRegion().getId()));  
 view.addObject(**"localities"**, **localityService**.findByDistrictId(o.getLocality().getDistrict().getId()));  
 view.addObject(**"parents"**, **objectService**.findByLocalityId(o.getLocality().getId()));  
 }  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/object/edit/{id}"**, method = RequestMethod.***POST***)  
 **public** String edit(Object object,  
 @RequestParam(value = **"epochs"**, required = **false**) List<Long> epochs,  
 @RequestParam(value = **"types"**, required = **false**) List<Long> types,  
 @RequestParam(value = **"styles"**, required = **false**) List<Long> styles,  
 @RequestParam(value = **"years"**, required = **false**) List<Integer> years,  
 @PathVariable(**"id"**) Long id, RedirectAttributes redirectAttributes) {  
 Object o = **objectService**.findOne(id);  
 object.setCover(o.getCover());  
 object.setAuthor(o.getAuthor());  
 **if**(**objectService**.save(object, epochs, types, styles, years) != **null**) {  
 redirectAttributes.addFlashAttribute(**"success"**, **"Объект успешно отредактирован"**);  
 }  
 **return "redirect:/admin/object"**;  
 }  
  
 @RequestMapping(value = **"/admin/object/photo/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView photos(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"photo"**, **true**);  
 Object object = **objectService**.findOne(id);  
 **if**(object != **null**) {  
 view.addObject(**"title"**, **"Фотоальбом объекта «"** + object.getName() + **"»"**);  
 view.addObject(**"object"**, object);  
 view.addObject(**"objectName"**, **"object"**);  
 view.addObject(**"backBtnText"**, **"К списку объектов"**);  
 view.addObject(**"objectTitle"**, **"объект"**);  
 }  
 **return** view;  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/object/upload"**, method = RequestMethod.***POST***)  
 **public** Boolean saveFile(@ModelAttribute UploadedFile uploadedFile, @RequestParam(**"objectId"**) Long objectId) {  
 **return objectService**.uploadPhoto(uploadedFile, objectId);  
 }  
  
 @RequestMapping(value = **"/admin/object/setCover"**, method = RequestMethod.***POST***)  
 **public** String setCover(@RequestParam(**"objectId"**) Long objectId, @RequestParam(**"coverId"**) Long coverId, RedirectAttributes ra) {  
 Object object = **objectService**.findOne(objectId);  
 object.setCover(**photoService**.findOne(coverId));  
 **objectService**.save(object);  
 **return "redirect:/admin/object/photo/"** + object.getId();  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/object/getObjects"**, produces = {**"application/json; charset=UTF-8"**}, method = RequestMethod.***GET***)  
 **public** String getCustomers(@RequestParam(**"localityId"**) Long localityId) **throws** JSONException {  
 List<Object> objects = **objectService**.findByLocalityId(localityId);  
 JSONArray array = **new** JSONArray();  
 **for**(Object u: objects) {  
 JSONObject item = **new** JSONObject();  
 item.put(**"name"**, u.getName());  
 item.put(**"id"**, u.getId());  
 array.put(item);  
 }  
 **return** array.toString();  
 }  
}

@Controller  
**public class** UserController {  
  
 @Autowired  
 **private** UserValidator **userValidator**;  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @Autowired  
 **private** PhotoService **photoService**;  
  
 @Autowired  
 **private** ObjectService **objectService**;  
  
 @Autowired  
 **private** Captcha **captcha**;  
  
 @RequestMapping(value = **"/photo"**, method = RequestMethod.***GET***)  
 @ResponseBody  
 **public void** displayPhoto(String name, HttpServletResponse response) {  
 **photoService**.displayPhoto(response, name);  
 }  
  
 @RequestMapping(value = **"/captcha"**, method = RequestMethod.***GET***)  
 **public void** captcha(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **captcha**.captcha(req, resp);  
 }  
  
 @Transactional  
 @RequestMapping(value = **"/"**, method = RequestMethod.***GET***)  
 **public** ModelAndView home() {  
 View view = **new** View(**"main"**);  
 view.addObject(**"title"**, **"Главная"**);  
 **if**(**userService**.isAuthentificated()) {  
 view.addObject(**"objects"**, **objectService**.findRelevant(20));  
 } **else** {  
 view.addObject(**"objects"**, **objectService**.getRandomTop());  
 }  
 view.addObject(**"noTitle"**, **true**);  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/map"**, method = RequestMethod.***GET***)  
 **public** ModelAndView maps(String xCoordinate, String yCoordinate) {  
 View view = **new** View(**"map"**);  
 view.addObject(**"title"**, **"Карта"**);  
 view.addObject(**"xCoordinate"**, xCoordinate);  
 view.addObject(**"yCoordinate"**, yCoordinate);  
 view.addObject(**"objects"**, **objectService**.findAll());  
 **return** view;  
 }  
  
 @Transactional  
 @RequestMapping(value = **"/preferences"**, method = RequestMethod.***GET***)  
 **public** ModelAndView preferences() {  
 View view = **new** View(**"preferences"**);  
 view.addObject(**"title"**, **"Стоит посетить"**);  
 view.addObject(**"objects"**, **objectService**.findRelevant(**null**));  
 **return** view;  
 }  
  
 @Transactional  
 @RequestMapping(value = **"/top"**, method = RequestMethod.***GET***)  
 **public** ModelAndView top() {  
 View view = **new** View(**"top"**);  
 view.addObject(**"title"**, **"Топ-50"**);  
 view.addObject(**"objects"**, **objectService**.getTopList());  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/login"**, method = RequestMethod.***GET***)  
 **public** ModelAndView login(String error) {  
 View view = **new** View(**"login"**);  
 view.addObject(**"title"**, **"Вход"**);  
 **if** (error != **null**)  
 view.addObject(**"error"**, **true**);  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/denied"**, method = RequestMethod.***GET***)  
 **public** String denied() {  
 **return "redirect:/"**;  
 }  
  
 @RequestMapping(value = **"/registration"**, method = RequestMethod.***GET***)  
 **public** ModelAndView registration(Model model) {  
 View view = **new** View(**"registration"**);  
 view.addObject(**"title"**, **"Регистрация"**);  
 view.addObject(**"user"**, model.asMap().get(**"user"**) != **null** ? (User)model.asMap().get(**"user"**) : **new** User());  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/registration"**, method = RequestMethod.***POST***)  
 **public** ModelAndView registration(@RequestParam(**"captcha"**) String captchaReq, User user, BindingResult result,  
 HttpServletRequest request,  
 RedirectAttributes redirectAttributes) {  
 **if** (!captchaReq.equals(**captcha**.getGeneratedKey(request))) {  
 redirectAttributes.addFlashAttribute(**"error"**, **"Введен неверный код с изображения"**);  
 redirectAttributes.addFlashAttribute(**"user"**, user);  
 } **else** {  
 **userValidator**.validate(user, result, **false**);  
 **if** (result.hasErrors()) {  
 View view = **new** View(**"registration"**);  
 view.addObject(**"title"**, **"Регистрация"**);  
 **return** view;  
 }  
 User registered = **userService**.create(user);  
 **if**(registered != **null**) {  
 **userService**.authentication(registered);  
 redirectAttributes.addFlashAttribute(**"success"**, **"Регистрация завершена. Теперь вы можете указать свои предпочтения"**);  
 **return new** ModelAndView(**"redirect:/settings"**);  
 } **else** {  
 redirectAttributes.addFlashAttribute(**"error"**, **"При регистрации произошла ошибка"**);  
 redirectAttributes.addFlashAttribute(**"user"**, user);  
 }  
 }  
 **return new** ModelAndView(**"redirect:/registration"**);  
 }  
  
 @RequestMapping(value = **"/restore"**, method = RequestMethod.***GET***)  
 **public** ModelAndView restore() {  
 View view = **new** View(**"restore/requestForm"**);  
 view.addObject(**"title"**, **"Восстановление доступа"**);  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/restore"**, method = RequestMethod.***POST***)  
 **public** ModelAndView restore(@RequestParam(**"email"**) String email,  
 HttpServletRequest request,  
 RedirectAttributes redirectAttributes) {  
 User user = **userService**.findByEmail(email);  
 **if**(user != **null**) {  
 **if**(!user.getBanned()) {  
 **if** (**userService**.sendRestoreRequest(user, request)) {  
 redirectAttributes.addFlashAttribute(**"info"**, **"Ссылка для восстановления доступа отправлена на <b>"** + user.getEmail() + **"</b>"**);  
 } **else** {  
 redirectAttributes.addFlashAttribute(**"error"**, **"Произошла ошибка"**);  
 }  
 } **else** {  
 redirectAttributes.addFlashAttribute(**"error"**, **"Ваш аккаунт заблокирован"**);  
 }  
 } **else** {  
 redirectAttributes.addFlashAttribute(**"error"**, **"Пользователь с указанным email не найден"**);  
 }  
  
 **return new** ModelAndView(**"redirect:/restore"**);  
 }  
  
 @RequestMapping(value = **"/profile"**, method = RequestMethod.***GET***)  
 **public** ModelAndView profile() {  
 View view = **new** View(**"profile"**);  
 view.addObject(**"title"**, **"Просмотр профиля"**);  
 view.addObject(**"user"**, **userService**.findOne(**userService**.getUser().getId()));  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/profile/{login}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView profile(@PathVariable(**"login"**) String login) {  
 View view = **new** View(**"profile"**);  
 view.addObject(**"title"**, **"Просмотр профиля"**);  
 view.addObject(**"user"**, **userService**.findByLogin(login));  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/restore/confirm/{token}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView confirmRestore(@PathVariable(**"token"**) String token) {  
 View view = **new** View(**"restore/setPassword"**);  
 view.addObject(**"title"**, **"Восстановление доступа"**);  
 User user = **userService**.findByToken(token);  
 **if**(user != **null**) {  
 view.addObject(**"token"**, token);  
 } **else** {  
 view.addObject(**"error"**, **"Ссылка для восстановления не верна, либо просрочена"**);  
 view.addObject(**"hideForm"**, **true**);  
 }  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/restore/confirm/{token}"**, method = RequestMethod.***POST***)  
 **public** String confirmRestore(@PathVariable(**"token"**) String token,  
 @RequestParam(**"password"**) String password,  
 RedirectAttributes redirectAttributes) {  
 User user = **userService**.findByToken(token);  
 **if**(user != **null**) {  
 **if**(**userService**.changePassword(user, password, **true**)) {  
 **userService**.authentication(user);  
 redirectAttributes.addFlashAttribute(**"success"**, **"Пароль успешно изменен"**);  
 **return "redirect:/"**;  
 } **else** {  
 redirectAttributes.addFlashAttribute(**"error"**, **"Старый и новый пароли совпадают, придумайте новый пароль"**);  
 **return "redirect:/restore/confirm/"** + token;  
 }  
 }  
 **return "redirect:/restore"**;  
 }  
}

@Controller  
**public class** UsersController {  
  
 @Autowired  
 **private** UserValidator **userValidator**;  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @RequestMapping(value = **"/admin/users"**, method = RequestMethod.***GET***)  
 **public** ModelAndView list() {  
 View view = **new** View(**"users/list"**, **true**);  
 view.addObject(**"title"**, **"Управление пользователями"**);  
 view.addObject(**"users"**, **userService**.findAll());  
 view.addObject(**"user"**, **userService**.getUser());  
 **return** view;  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/users/delete"**, method = RequestMethod.***POST***)  
 **public void** delete(@RequestParam(**"id"**) Long id) {  
 **userService**.delete(id);  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/users/activate"**, method = RequestMethod.***POST***)  
 **public** Boolean activate(@RequestParam(**"id"**) Long id) {  
 User user = **userService**.findOne(id);  
 **if**(user != **null**) {  
 **return userService**.save(user) != **null**;  
 } **else** {  
 **return false**;  
 }  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/users/lock"**, method = RequestMethod.***POST***)  
 **public** Boolean lock(@RequestParam(**"id"**) Long id) {  
 **return userService**.lockUser(id, **true**);  
 }  
  
 @ResponseBody  
 @RequestMapping(value = **"/admin/users/unlock"**, method = RequestMethod.***POST***)  
 **public** Boolean unlock(@RequestParam(**"id"**) Long id) {  
 **return userService**.lockUser(id, **false**);  
 }  
  
 @RequestMapping(value = **"/admin/users/edit/{id}"**, method = RequestMethod.***GET***)  
 **public** ModelAndView edit(@PathVariable(**"id"**) Long id) {  
 View view = **new** View(**"users/edit"**, **true**);  
 view.addObject(**"title"**, **"Редактирование пользователя"**);  
 view.addObject(**"user"**, **userService**.findOne(id));  
 **return** view;  
 }  
  
 @RequestMapping(value = **"/admin/users/edit/{id}"**, method = RequestMethod.***POST***)  
 **public** ModelAndView edit(User user, BindingResult result, RedirectAttributes redirectAttributes) {  
  
 **userValidator**.validate(user, result, **true**);

**if** (result.hasErrors()) {  
 **return new** View(**"users/edit"**, **true**);  
 }  
  
 User updatedUser = **userService**.update(user);  
 **if**(updatedUser != **null**) {  
 redirectAttributes.addFlashAttribute(**"success"**, **"Пользователь "** + updatedUser.getFio() + **" успешно отредактирован"**);  
 }  
 **return new** ModelAndView(**"redirect:/admin/users"**);  
 }  
}

**ПРИЛОЖЕНИЕ B**

**Исходный код сервисов Comment, Photo**

**package** com.db.tourist.services.Impl;  
  
**import** com.db.tourist.models.Comment;  
**import** com.db.tourist.repositories.CommentRepository;  
**import** com.db.tourist.repositories.ObjectRepository;  
**import** com.db.tourist.services.CommentService;  
**import** com.db.tourist.services.UserService;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.Date;  
**import** java.util.List;  
  
@Service  
**public class** CommentServiceImpl **implements** CommentService {  
  
 @Autowired  
 **private** CommentRepository **commentRepository**;  
  
 @Autowired  
 **private** UserService **userService**;  
  
 @Autowired  
 **private** ObjectRepository **objectRepository**;  
  
 **public** List<Comment> findChecked() {  
 **return commentRepository**.findByCheckedIsTrue();  
 }  
  
 **public** List<Comment> findNotChecked() {  
 **return commentRepository**.findByCheckedIsFalse();  
 }  
  
 **public** Integer getNotCheckedCount() {  
 **return commentRepository**.getNotCheckedCount();  
 }  
  
 **public** Comment findOne(Long id) {  
 **return commentRepository**.findOne(id);  
 }  
  
 **public void** delete(Long id) {  
 **commentRepository**.delete(id);  
 }  
  
 **public** Boolean check(Long id) {  
 Comment comment = **commentRepository**.findOne(id);  
 comment.setChecked(**true**);  
 comment.setModerator(**userService**.getUser());  
 **return commentRepository**.save(comment) != **null**;  
 }  
  
 **public** Boolean send(Long objectId, Integer rate, String text, Long userId) {  
 Comment c = **new** Comment();  
 c.setDate(**new** Date());  
 c.setChecked(**false**);  
 c.setMark(rate);  
 c.setObject(**objectRepository**.findOne(objectId));  
 c.setUser(**userService**.findOne(userId));  
 c.setText(text);  
 **return commentRepository**.save(c) != **null**;  
 }  
  
 **public** List<Comment> getComments(Long objectId) {  
 **return commentRepository**.findByObjectIdAndCheckedIsTrueOrderByDateDesc(objectId);  
 }  
  
 **public** Integer checkCommented(Long objectId) {  
 **if**(!**userService**.isAuthentificated())  
 **return** 3;  
 List<Comment> comments = **commentRepository**.findByObjectIdAndUserId(objectId, **userService**.getUser().getId());  
 **if**(!comments.isEmpty()) {  
 Comment c = comments.get(0);  
 **if**(!c.getChecked()) {  
 **return** 1;  
 } **else** {  
 **return** 2;  
 }  
 }  
 **return** 0;  
 }  
}

@Service  
**public class** PhotoServiceImpl **implements** PhotoService {  
 @Autowired  
 **private** FileHelper **fileHelper**;  
  
 @Autowired  
 **private** PhotoRepository **photoRepository**;  
  
 @Autowired  
 **private** ObjectRepository **objectRepository**;  
  
 @Autowired  
 **private** EpochRepository **epochRepository**;  
  
 @Autowired  
 **private** StyleRepository **styleRepository**;  
  
 @Autowired  
 **private** TypeRepository **typeRepository**;  
  
 @Transactional  
 **public** Boolean deletePhoto(String photo) {  
 **if**(**fileHelper**.delete(photo)) {  
 Photo p = **photoRepository**.findByFile(photo);  
 **if**(p != **null**) {  
 **if**(p.getObject() != **null**) {  
 Object o = p.getObject();  
 **if**(o.getCover() != **null** && o.getCover().getId().equals(p.getId())) {  
 o.setCover(**null**);  
 **objectRepository**.save(o);  
 }  
 }  
 **if**(p.getEpoch() != **null**) {  
 Epoch o = p.getEpoch();  
 **if**(o.getCover() != **null** && o.getCover().getId().equals(p.getId())) {  
 o.setCover(**null**);  
 **epochRepository**.save(o);  
 }  
 }  
 **if**(p.getType() != **null**) {  
 Type o = p.getType();  
 **if**(o.getCover() != **null** && o.getCover().getId().equals(p.getId())) {  
 o.setCover(**null**);  
 **typeRepository**.save(o);  
 }  
 }  
 **if**(p.getStyle() != **null**) {  
 Style o = p.getStyle();  
 **if**(o.getCover() != **null** && o.getCover().getId().equals(p.getId())) {  
 o.setCover(**null**);  
 **styleRepository**.save(o);  
 }  
 }  
 **photoRepository**.delete(p);  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 **public void** displayPhoto(HttpServletResponse response, String photo) {  
 **try** {  
 **fileHelper**.getFile(response, photo);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **public** Photo findOne(Long id) {  
 **return photoRepository**.findOne(id);  
 }  
}

@Service  
**public class** UserServiceImpl **implements** UserService {  
  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 @Autowired  
 **private** Md5PasswordEncoder **encoder**;  
  
 @Autowired  
 **private** EmailSender **emailSender**;  
  
 @Autowired  
 **private** RoleRepository **roleRepository**;  
  
 @Value(**"${app.siteName}"**)  
 **private** String **siteName**;  
  
 **public** User getUser() {  
 **return** ((UserPrincipal) SecurityContextHolder.*getContext*().getAuthentication().getPrincipal()).getUser();  
 }  
  
 **public** User update(User user) {  
 User updUser = **userRepository**.findOne(user.getId());  
 **if**(updUser != **null**) {  
 updUser.setName(user.getName());  
 updUser.setSurname(user.getSurname());  
 updUser.setPatronymic(user.getPatronymic());  
 updUser.setPhoneNumber(user.getPhoneNumber());  
 updUser.setEmail(user.getEmail());  
 updUser.setLogin(user.getLogin());  
 **if**(user.getRole() != **null**) {  
 updUser.setRole(user.getRole());  
 }  
 **return userRepository**.save(updUser);  
 }  
 **return null**;  
 }  
  
 **public** UserPrincipal makeUserPrincipal(User user) {  
 Set<GrantedAuthority> grantedAuthorities = **new** HashSet<>();  
 grantedAuthorities.add(**new** SimpleGrantedAuthority(user.getRole().getName()));  
 **return new** UserPrincipal(user.getLogin(), user.getPassword(), grantedAuthorities, user);  
 }  
  
 @Override  
 **public** User create(User user) {  
 user.setRole(**roleRepository**.findByName(**"ROLE\_USER"**));  
 user.setPassword(**encoder**.encodePassword(user.getPassword(), **null**));  
 **return userRepository**.save(user);  
 }  
  
 **public** Boolean sendRestoreRequest(User user, HttpServletRequest request) {  
 user.setToken(UUID.*randomUUID*().toString());  
 **userRepository**.save(user);  
  
 String url = request.getRequestURL().toString();  
 String domain = url.substring(0, url.length() - request.getRequestURI().length())  
 + request.getContextPath() + **"/"**;  
 String lnk = domain + **"restore/confirm/"** + user.getToken();  
 String text = **"Уважаемый, "** + user.getName() + **" "** + user.getPatronymic() + **".<br/>"** +  
 **"Для восстановления доступа к Вашему профилю на "** + **siteName** + **" перейдите по ссылке "** +  
 **"<a href=\""** + lnk + **"\">"**+lnk+**"</a>"**;  
 **try** {  
 **emailSender**.sendMail(**siteName**, user.getEmail(), **"Восстановление доступа"**, text);  
 } **catch** (MessagingException e) {  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **public** User findByEmail(String email) {  
 **return userRepository**.findByEmail(email);  
 }  
  
 **public** User findByToken(String token) {  
 **return userRepository**.findByToken(token);  
 }  
  
 **public** User findByLogin(String login) {  
 **return userRepository**.findByLogin(login);  
 }  
  
 **public** Integer changePassword(String oldPassowrd, String password) {  
 User user = getUser();  
 **if**(oldPassowrd != **null**) {  
 **if** (!**encoder**.isPasswordValid(user.getPassword(), oldPassowrd, **null**)) {  
 **return** 1;  
 }  
 }  
 **return** changePassword(user, password, **false**) ? 0 : 2;  
 }  
  
 **public** Boolean changePassword(User user, String password, Boolean resetToken) {  
 *//Если новый и старый пароли совпадают* **if** (**encoder**.isPasswordValid(user.getPassword(), password, **null**)) {  
 **return false**;  
 }  
 user.setPassword(**encoder**.encodePassword(password, **null**));  
 **if**(resetToken != **null** && resetToken){  
 user.setToken(**null**);  
 }  
 **return userRepository**.save(user) != **null**;  
 }  
  
 **public** User save(User user) {  
 **return userRepository**.save(user);  
 }  
  
 **public** List<User> findAll() {  
 **return userRepository**.findAll();  
 }  
  
 **public void** delete(Long userId) {  
 **userRepository**.delete(userId);  
 }  
  
 **public** User findOne(Long userId) {  
 **return userRepository**.findOne(userId);  
 }  
  
 **public** Boolean lockUser(Long id, Boolean lockStatus) {  
 User user = **userRepository**.findOne(id);  
 **if**(user != **null** && !user.getBanned().equals(lockStatus)) {  
 user.setBanned(lockStatus);  
 **return userRepository**.save(user) != **null**;  
 } **else** {  
 **return false**;  
 }  
 }  
  
 **public void** authentication(User user) {  
 UserPrincipal principal = makeUserPrincipal(user);  
 Authentication newAuth = **new** UsernamePasswordAuthenticationToken(principal, user.getPassword(), principal.getAuthorities());  
 SecurityContextHolder.*getContext*().setAuthentication(newAuth);  
 }  
  
 **public** Boolean isAuthentificated() {  
 SecurityContext context = SecurityContextHolder.*getContext*();  
 **if** (context == **null**)  
 **return false**;  
  
 Authentication authentication = context.getAuthentication();  
 **if** (authentication == **null**)  
 **return false**;  
  
 **for** (GrantedAuthority auth : authentication.getAuthorities()) {  
 **if** (auth.getAuthority().equals(**"ROLE\_ADMIN"**) || auth.getAuthority().equals(**"ROLE\_USER"**))  
 **return true**;  
 }  
 **return false**;  
 }  
}