Министерство образования Республики Беларусь

ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет «Повышения квалификации и переподготовки кадров»

Кафедра ««Информационные системы и автоматизация производства»»

Допущена к защите

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. Е. Казаков

« » 2020 г.

ДИПЛОМНАЯ РАБОТА

**тема: разработка информационной системы управления аварийно-профилактической группой**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | |  | |
| Специальность | | | 1-40 01 73 «Программное обеспечение информационных систем» | |
| Группа | | | ИС-17 | |
| Исполнитель |  |  |  | Д.Н.Прядкин |
|  |  |  |  | И.О. Фамилия |
| Руководитель |  |  |  | к.т.н., доц. А.С.Дягилев |
|  |  |  |  | учёная степень, должность, И.О. Фамилия |

Витебск 2020

**УО «Витебский государственный технологический университет»**

**Факультет повышения квалификации и переподготовки кадров**

УТВЕРЖДАЮ

Зав. кафедрой

В. Е. Казаков

(ФИО, подпись)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

**ЗАДАНИЕ**

**по подготовке дипломной работы**

1.Тема работы «Разработка информационной системы управления аварийно-профилактической группой»

Утверждена приказом по институту от №

2. Сроки сдачи слушателем выполненной работы

3. Перечень вопросов, которые будут разработаны в дипломной работе или краткое содержание дипломной работы:

* + 1. Понятие об информационной системе управления аварийно-профилактической группой
    2. Понятие информационной системы управления аварийно-профилактической группой и ее применение
    3. Предметная область и требования к информационной системе управления аварийно-профилактической группой
    4. Средства реализации и разработки информационной системы управления аварийно-профилактической группой
    5. Разработка информационной системы управления аварийно-профилактической группой
    6. Проектирование структуры базы данных
    7. Разработка классов
    8. Интерфейс информационной системы управления аварийно-профилактической группой
    9. Тестирование информационной системы управления аварийно-профилактической группой

4. Перечень практических материалов, которые будут разработаны в дипломной работе:

* + 1. информационная система управления аварийно-профилактической группой.

Задание принял для выполнения Прядкин Д.Н.

(фамилия) (дата)

Подпись слушателя

**Реферат**

Дипломная работа 68 c., 12 рис., 18 источника, 3 прил.

**Ключевые слова:** СИСТЕМА УПРАВЛЕНИЯ, PHP, БАЗЫ ДАННЫХ, XHTML, CSS.

**Объект исследования** – информационная система управления аварийно-профилактической группой.

**Предмет исследования** – применение современных информационных технологий для создания информационной системы управления аварийно-профилактической группой.

**Цель работы** – проектирование и разработка информационной системы управления аварийно-профилактической группой.

**Методы исследования и технология разработки**: описательно-аналитический, экспериментальный, объектно-ориентированное программирование, построение баз данных. В качестве системы управления базами данных выбрана СУБД MySQL 8.0. Средствами разработки являются PHP 7.4, ХHTML, CSS, UML. Инструментарий для разработки: IDE PhpStorm 2020.2, Apache 2.4, онлайн ресурс для построения UML-диаграмм Draw.io, Bootstrap 4, MySQL Workbench 8.0 CE.

Теоретическая и практическая значимость заключается в том, что результаты работы можно применять на практике, концепция данного исследования может быть использована в качестве методического пособия при разработке информационных систем.

**Перечень условных обозначений и терминов**

СУБД – система управления базами данных.

MVC (Model View Controller, «модель-представление-контроллер», «модель-вид-контроллер») – схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента.

PHP – рекурсивный акроним словосочетания Hypertext Preprocessor – это распространенный язык программирования общего назначения с открытым исходным кодом. PHP специально сконструирован для веб-разработок и его код может внедряться непосредственно в HTML.

XHTML – Extensible HyperText Markup Language (расширяемый язык разметки гипертекста).

SQL – Structured Query Language (язык структурированных запросов).

CSS (англ. Cascading Style Sheets – каскадные таблицы стилей) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

UML (англ. Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

HTTP – HyperText Transport Protocol (протокол передачи гипертекста).

ООП – объектно-ориентированное программирование.

**Оглавление**

[Введение 7](#_Toc48668442)

[1 Понятие об информационной системе управления аварийно-профилактической группой 9](#_Toc48668443)

[1.1 Понятие системы управления аварийно-профилактической группой и ее применение 9](#_Toc48668444)

[1.2 Предметная область и требования к информационной системе управления аварийно-профилактической группой 11](#_Toc48668445)

[1.3 Средства реализации и разработки информационной системы управления аварийно-профилактической группой 13](#_Toc48668446)

[2.2 Разработка классов 27](#_Toc48668447)

[2.4 Тестирование информационной системы управления аварийно-профилактической группой 62](#_Toc48668448)

[Заключение 66](#_Toc48668449)

[Список использованных источников 67](#_Toc48668450)

[ПРИЛОЖЕНИЕ А 69](#_Toc48668451)

[ПРИЛОЖЕНИЕ В 74](#_Toc48668452)

# 

**Введение**

Современное общество подчиняется концепции современного этапа общественного развития, характеризующего переход к новой форме постиндустриального общества, где доминирующей ценностью, экономической и ресурсной, становится «знание» как таковое. Это формирует так называемое общество знания или информационное общество.

Суть информационного общества заключается в том, что человеческая цивилизация после аграрной и индустриальной стадии развития вступает в новую - информационную, где информация считается наиболее ценным ресурсом, а её доступность является наиболее важной в данной идеологии. Однако информация - лишь инструмент знания, сама по себе знанием она не является. Избыток информации и её доступность не приводит к приращению знания.

В обществе знания важнее всего «научиться учиться», а новые информационные технологии должны способствовать постоянному обновлению личной и профессиональной компетенции. Новые технологии повсеместно ускоряют создание и распространение знаний. Обучение становится ключевой ценностью обществ знания. Важное значение в обществе знания приобретает способность ориентироваться в потоке информации, когнитивные способности, критический ум, позволяющий отличать полезную информацию от бесполезной. В этом обществу помогают информационные ресурсы и электронные библиотеки.

Электронная библиотека — упорядоченная коллекция разнородных электронных документов (в том числе [книг](https://ru.wikipedia.org/wiki/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0), [журналов](https://ru.wikipedia.org/wiki/%D0%96%D1%83%D1%80%D0%BD%D0%B0%D0%BB)), снабжённых средствами навигации и поиска. Может быть [веб-сайтом](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82), где постепенно накапливаются различные [тексты](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82) (чаще литературные, но также научные и любые другие, вплоть до компьютерных программ) и медиафайлы, каждый из которых самодостаточен и в любой момент может быть востребован читателем [5].

**Целью** нашей работы является проектирование и разработка информационной системы управления аварийно-профилактической группой.

Основные **задачи**:

* изучить основные характеристики систем управления работами, определить их задачи и функции;
* обозначить аудиторию и предметную область разрабатываемой системы управления аварийно-профилактической группой, функции, которые она будет выполнять; определить информационную составляющую;
* выбрать программные средства и разработать систему управления аварийно-профилактической группой.

Конечный **результат** работы будет представлять собой сайт управления производственными задачами и поездками выполняемыми аварийно-профилактической группы. Теоретическая и практическая значимость заключается в том, что результаты работы можно применить для планирования и учёта задач в любой сфере деятельности.

**Методы исследования и технология разработки**: описательно-аналитический, экспериментальный, объектно-ориентированное программирование, процедурное программирование, построение баз данных. В качестве системы управления базами данных выбрана СУБД MySQL 8.0. Средствами разработки являются PHP 7.4, ХHTML, CSS, UML. Инструментарий для разработки: IDE PhpStorm 2020.2, Apache 2.4, онлайн ресурс для построения UML-диаграмм Draw.io, Bootstrap 4, MySQL Workbench 8.0 CE.

1 **Понятие об информационной системе управления аварийно-профилактической группой**

**1.1 Понятие системы управления аварийно-профилактической группой и ее применение**

Информационная система управления аварийно-профилактической группой появилась как самостоятельный объект в результате эволюции технологий в различных направлениях человеческой деятельности, вобрав в себя их элементы. Основными из них были следующие:

- электронные архивы;

Для обеспечения компактного хранения больших объемов документов и организации системы их быстрого поиска стали создаваться электронные архивы, например, архивы распорядительных документов предприятия, архивы производственных чертежей и многие другие. На основе созданного архива документов со временем стали появляться целые классы систем, обеспечивающие управление электронным документооборотом предприятия, оптимизацию управленческих процессов, организацию совместной работы специалистов. В системах данного класса появился опыт массовой оцифровки бумажных документов, систематизации и надежного хранения больших объёмов электронных документов.

- интернет;

Отдельным феноменом в данной области стала сеть Интернет. Возможность самостоятельного опубликования материалов в электронном виде, или, как теперь говорят, создания контента привлекла многих авторов. Развитие средств опубликования существенно снизило порог вхождения – уже не требовалось знания, скажем, специальных языков или каких-то технологий, для того чтобы опубликовать в Сети собственное произведение. Появились новые подходы к поиску информации в распределенной сети хранения. В этой области приоритет отдавался поиску по содержимому документа, что стало мощнейшим стимулом технологий морфологического и семантического анализа документов. К настоящему времени пришло осознание, что для эффективного поиска ресурсов в огромном электронном хранилище дополнительно требуются меры систематизации ресурсов по их семантике (своего рода каталогизация – как в библиотеке). Технологии семантического Web’а призваны существенно повысить качество поиска ресурсов.

- книжные проекты в интернете;

По мере развития Интернета стали появляться компании, успешно находившие инновационные решения для работы в крупнейшей сети электронных хранилищ ресурсов – Google, Yandex, Yahoo и другие. Огромное количество доступных и потому иногда трудно отыскиваемых ресурсов стимулировало выделение хранилищ качественного контента. Тут, естественно, взоры обратились к библиотекам, где при комплектовании фонда производится профессиональный отбор литературы.

- электронные книги издательств;

Вслед за интернет-магазинами книг бумажных появились магазины электронных книг. Такой интернет-магазин стал своего рода электронной библиотекой. В этой области важнейшей составляющей стали технологии защиты электронных книг от несанкционированного копирования и использования. Электронные версии печатных книг, доступные для продажи пользователю, могут быть использованы и для печати в традиционном виде. Технологии цифровой печати (print-on-demand) сделали возможным печать книг тиражами любого объёма и, кроме того, практически через любой промежуток времени после первого выхода книги из печати.

- проекты в библиотеках;

Библиотеки тоже стали создавать коллекции электронных ресурсов. В электронную среду они перенесли свои сильные стороны: качественную каталогизацию ресурса, заботу о сохранности цифрового произведения, стандарты в описании ресурсов и обмене данными. Инициатива по созданию таких коллекций часто была связана с проектами по сохранению культурного наследия, обеспечению его доступности самым широким слоям населения [6].

Особое значение электронные библиотеки имеют для организации учебной работы. Самостоятельная работа является неотъемлемой частью учебного процесса, которая требует соответствующего информационно-предметного обеспечения. Имея в своем распоряжении электронный учебник, практикум по курсу, рекомендации по решению типовых задач, библиотеку электронных носителей информации по тематике дисциплины, электронные справочники, перечень вопросов к семинарам и экзаменам, студент может эффективно организовать свою самостоятельную работу с учетом собственных возможностей и потребностей.

**1.2 Предметная область и требования к информационной системе управления аварийно-профилактической группой**

Разрабатываемая система управления аварийно-профилактической группой может использоваться для планирования и учёта процесса выполнения работ и выездов автотранспорта. Данная работа является очень актуальной в связи с острой необходимостью в автоматизации документирования бизнес-процессов организаций. Система позволяет осуществлять одновременный доступ к необходимой информации для различных групп пользователей.

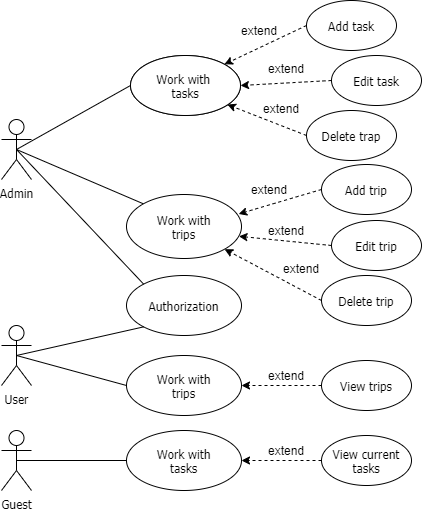
Администратор работает с задачами и учитывает выезды автотранспорта (добавляет, редактирует и удаляет), справочниками (добавляет, редактирует и удаляет локации, транспорт и персонал), также осуществляет управление доступом к информационной системе.

Пользователь может просматривать ведомость выездов автотранспорта.

Диаграмма прецедентов для разрабатываемого веб-сайта приведена на рисунке 1.1.

Таким образом, разрабатываемая система управления аварийно-профилактической группой должна соответствовать следующим техническим требованиям:

* наличие системы авторизации и разграничения прав доступа к содержимому базы данных;
* гибкость и удобство использования, как для пользователей, так и для администратора;
* возможность добавлять, изменять, удалять (для администратора) и просматривать (для пользователя) заявок на выезд автотранспорта для выполнения производственных задач;
* возможность (для администратора) изменять статус задачи в зависимости от этапов её реализации.



**Рисунок 1.1 – Диаграмма прецедентов системы управления аварийно-профилактической группой**

**1.3 Средства реализации и разработки информационной системы управления аварийно-профилактической группой**

Учитывая поставленные выше задачи, в качестве средств реализации и разработки системы управления аварийно-профилактической группой были выбраны PHP, XHTML, CSS. В качестве системы управления базами данных выбрана СУБД MySQL, для разработки диаграмм прецедентов и классов использовался язык UML.

PHP – это строго типизированный объектно-ориентированный язык программирования. Достоинством программ, разработанных на языке Java, является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание [8].

Основными преимуществами языка программирования Java являются:

- Объектно-ориентированность: в Java все является объектом. Дополнение может быть легко расширено, так как он основан на объектной модели.

**-** Платформонезависимость: в отличие от многих других языков, включая C и C++, Java, когда был создан, он не компилировался в платформе конкретной машины, а в независимом от платформы байт-коде. Этот байт код распространяется через интернет и интерпретируется в Java Virtual Machine (JVM), на которой он в настоящее время работает.

**-** Простота: процессы изучения и введение в язык программирования Java остаются простыми. Если Вы понимаете основные концепции объектно-ориентированного программирования, то он будет прост для Вас в освоении.

**-** Безопасность: методы проверки подлинности основаны на шифровании с открытым ключом.

**-** Архитектурно-нейтральность: компилятор генерирует архитектурно-нейтральные объекты формата файла, что делает скомпилированный код исполняемым на многих процессорах, с наличием системе Java Runtime.

**-** Портативность: архитектурно-нейтральный и не имеющий зависимости от реализации аспектов спецификаций — все это делает Java портативным. Компилятор в Java написан на ANSI C с чистой переносимостью, который является подмножеством POSIX.

**-** Прочность: прилагает усилия, чтобы устранить ошибки в различных ситуациях, делая упор в основном на время компиляции, проверку ошибок и проверку во время выполнения.

**-** Многопоточность: функции многопоточности, можно писать программы, которые могут выполнять множество задач одновременно. Введение в язык Java этой конструктивной особенности позволяет разработчикам создавать отлаженные интерактивные приложения.

- Интерпретированность: Java байт-код переводится на лету в машинные инструкции и нигде не сохраняется. Делая процесс более быстрым и аналитическим, поскольку связывание происходит как дополнительное с небольшим весом процесса.

**-** Высокопроизводительность: введение Just-In-Time компилятора, позволило получить высокую производительность.

**-** Распространенность: предназначен для распределенной среды интернета.

**-** Динамичность: программирование на Java считается более динамичным, чем на C или C++, так как он предназначен для адаптации к меняющимся условиям. Программы могут выполнять обширное количество во время обработки информации, которая может быть использована для проверки и разрешения доступа к объектам на время выполнения [15].

XHTML – это основанный на [XML](https://htmlweb.ru/xml/xml1.php) язык разметки гипертекста, максимально приближенный к текущим стандартам [HTML](https://htmlweb.ru/html/bhtml.php). XHTML отличается от HTML строгостью написания кода. Если HTML позволял писать практически любые конструкции и браузер их корректно распознавал, то теперь, с появлением XHTML, это стало невозможным. Последний требует строгого соблюдения всех правил, предъявляемых W3C. Строгие требования к оформлению XHTML- кода позволяют избежать многих ошибок ещё на стадии написания и отладки. XHTML-документ состоит из текста, представляющего собой содержание документа, и тегов, определяющих его структуру и внешний вид при отображении браузером, например, теги сообщают браузеру, какая часть текста является заголовком, где начинается новый абзац, что нужно подчеркнуть и где расположить изображение. XHTML-данные правильно отдавать клиенту именно в формате <application/xhtml+xml>, так как все преимущества, помимо кросс-браузерности (увеличение скорости анализа кода процессором XML, сообщение об ошибках самим браузером и пр.), могут достигаться только в случае, если пользовательский агент поддерживает XHTML и ему сообщается о том, что входящие данные - XHTML-код. Единственное, что надо помнить при отправке XHTML-кода: если браузер понимает XHTML, то только тогда информацию можно отправить как <application/xhtml+xml>; если же нет, то только как <text/html> [10].

CSS (англ. Cascading Style Sheets – каскадные таблицы стилей) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например, к SVG или XUL. CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана), или при выводе устройствами, использующими шрифт Брайля [9].

MySQL – свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle. MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц, благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц. Преимуществами СУБД MySQL можно считать следующие:

- Высокая скорость обработки данных;

- Поддержка SQL, что обеспечивает высокий уровень кроссплатформенности данных и кода, созданных с помощью MySQL. Благодаря этому можно спокойно перенести БД в любую другую современную СУБД, также поддерживающую язык структурированных запросов;

- Система привилегий, что позволяет наделять каждую учетную запись сервера правами на осуществление определенных действий с данными. Причем не только на уровне сервера, БД, но и на уровне отдельных таблиц.

- Хэширование паролей – обеспечивает высокий уровень «противовзломности». Именно поэтому в MySQL восстановить пароль root очень сложно. Так что лучше не забывать его [13].

UML (англ. Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода [8].

Для создания веб-сайта на локальном компьютере будет использоваться сборка на «домашней» (локальной) Windows-машине без необходимости выхода в Интернет. Сборка выглядит следующим образом:

* GlassFish,
* MySQL,
* Java,
* Spring,
* JSF.

GlassFish сервер –  [сервер приложений](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) с открытым исходным кодом, реализующий спецификации [Java EE](https://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition" \o "Java Platform, Enterprise Edition), изначально разработанный [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems" \o "Sun Microsystems). Основными достоинствами GlassFish считается эталонная реализация спецификации Java ЕЕ, что исключает большое количество ошибок, возникающих при написании приложений для Java EE. Он реализует самые последние нововведения в спецификации раньше, чем любые другие серверы приложений на рынке.  Сервер GlassFish является одним из немногих (если не сказать - единственным) свободно-распространяемым Java EE-совместимым сервером приложений, который поддерживает спецификацию Java ЕЕ 7 в наиболее полном объеме [4].

Spring Framework — универсальный [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) [с открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для [Java](https://ru.wikipedia.org/wiki/Java" \o "Java)-платформы. Он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели [Enterprise JavaBeans](https://ru.wikipedia.org/wiki/EJB" \o "EJB). 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 [18].

Spring может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. Эти фреймворки делятся на структурные элементы типовых комплексных приложений [16]. На сегодняшний день Spring разделён на некоторое количество отдельных модулей, что позволяет Вам самим решать, какие из них использовать в Вашем приложении.

Основной контейнер (Core Container)включает в себя Beans, Core, Context и SpEL (expression language).

Beans отвечает за BeanFactory которая является сложной реализацией паттерна Фабрика (GoF).

Модуль Coreобеспечивает ключевые части фреймворка, включая свойства IoC и DI.

Contextпостроен на основе Beans и Core и позволяет получить доступ к любому объекту, который определён в настройках. Ключевым элементом модуля Contextявляется интерфейс ApplicationContext.

Модуль SpEL обеспечивает мощный язык выражений для манипулирования объектами во время исполнения.

КонтейнерData Access/Integrationсостоит из JDBC, ORM, OXM, JMS и модуля Transatcions.

JDBCобеспечиваетабстрактный слой JDBC и избавляет разработчика от необходимости вручную прописывать монотонный код, связанный с соединением с БД.

ORMобеспечивает интеграцию с такими популярными ORM, как Hibernate, JDO, JPA и т.д.

Модуль OXMотвечает за связь Объект/XML – XMLBeans, JAXB и т.д.

МодульJMS (Java Messaging Service)отвечает за создание, передачу и получение сообщений.

Transactionsподдерживает управление транзакциями для классов, которые реализуют определённые методы.

Web слой состоит из Web, Web-MVC, Web-Socket, Web-Portlet. МодульWeb обеспечивает такие функции, как загрузка файлов и т.д. Web-MVCсодержит реализацию Spring MVC для веб-приложений. Web-Socketобеспечивает поддержку связи между клиентом и сервером, используя Web-Socket-ы в веб-приложениях. Web-Portletобеспечивает реализацию MVC в среде портлетов.

Spring также включает в себя ряд других важных модулей, таких как AOP, Aspects, Instrumentation, Messaging и Test. AOPреализует аспекто-ориентированное программирование и позволяет использовать весь арсенал возможностей АОП. Модуль Aspectsобеспечивает интеграцию с AspectJ, которая также является мощным фреймворком АОП. Instrumentationотвечает за поддержку class instrumentation и class loader, которые используются в серверных приложениях. МодульMessaging обеспечивает поддержку STOMP [14].

Для работы приложения были использованы следующие модули:

- Spring Boot Data JPA – для работы с Hibernate

- Spring Boot Web – для реализации MVC, создания веб-контейнера

- Spring Boot Security – для обеспечения безопасности приложения (авторизация, аутентификация)

- Spring Boot Actuator – анализ различных run time данных во время работы приложения (бины)

- Spring Boot AOP – для отображения сообщений действий пользователей

Для создания диаграмм использовался инструмент для UML-диаграмм Creately. Назначение программы - создание схем и диаграмм. Приложение имеет различные панели инструментов для форм, соединителей, соединение объектов создается одним щелчком, изменение размерных стрелок является естественным и быстрым. Возможно выравнивание, установка размеров, группировка, расположение в слоях, использование умных форм для макетов UI, UML и больше. Каждая умная форма была тщательно спроектирована, чтобы соответствовать ее контексту рисунку. Изменение ключевых визуальных особенностей интуитивно в нескольких щелчках. Возможна совместная работа, действующие комментарии и тематики обсуждения. Легко импортирует существующие файлы Visio в Creately.

dbForge Studio for MySQL v8.2 Express - это бесплатный и профессиональный инструмент для разработчиков БД и пользователей MySQL. Он автоматизирует рутинные задачи по разработке и администрированию СУБД MySQL, а также открывает новые возможности для получения лучшего результата. В нем можно разрабатывать SQL скрипты, хранимые процедуры и функции, составлять и выполнять запросы, редактировать данные, осуществлять их экспорт и импорт, управлять пользователями, редактировать объекты БД, работать с проектами БД и многое другое.

Для работы над кодом использовалася IntelliJ IDEA Ultimate Edition - платная версия самой умной среды разработки на основе открытого кода. Основными преимуществами данной версии является умное автодополнение, инструменты для анализа качества кода, удобная навигация, расширенные рефакторинги и форматирование для Java, Groovy, Scala, Clojure и Erlang, интеграция с серверами приложений: Tomcat, JBoss, WebSphere, WebLogic,GlassFish и другими, поддержка популярных фреймворков: Java EE, Spring, Grails, Play, Android, GWT, React, AngularJS и другие, интеграция с автоматизированными инструментами сборки и управления проектом, включая Maven, Gradle, Ant и другими, интеграция с системами управления версиями, включая Git, Subversion, Mercurial и CSV [10].

**Выводы:**

Таким образом, электронные библиотеки в сфере, например, образования включают в себя электронные учебники, методические разработки, которые способствуют повышению эффективности, доступности и качества образования.

Одной из характерных черт современной управления аварийно-профилактической группой должна стать беспрепятственная возможность пользователей обращаться к книгам.

Таким образом, была выделена предметная область и основные требования к разрабатываемому программному продукту:

* система разграничения прав пользователей;
* гибкость и простота загрузки и получения информации;
* соответствие требованиям, предъявляемым к электронным библиотекам.

В соответствии с поставленными задачами были выбраны также средства и инструментарий для проектирования и разработки веб-сайта, с учетом современных тенденций сайтостроения: JAVA, XHTML, CSS, MySQL, а также фреймворк Spring, технология JSF, среда разработки IntelliJ IDEA Ultimate Edition и другие необходимые программные продукты.

**2 Разработка информационной системы управления аварийно-профилактической группой**

**2.1 Проектирование структуры базы данных**

Продуманное проектирование базы данных чрезвычайно важно для правильной работы приложения. MySQL – это система управления реляционными базами данных. Важная особенность реляционных систем, их отличие от одноуровневых баз данных – возможность располагать данные в нескольких таблицах. Взаимосвязанные данные можно хранить в отдельных таблицах и объединять по ключу, общему для обеих таблиц. Ключ – это отношение (relation) между таблицами. Выбор первичного ключа (primary key) – наиболее важное решение, принимаемое при разработке новой базы данных. Самое главное – необходимо гарантировать уникальность выбранного ключа. Если есть вероятность того, что значение некоторого атрибута может совпасть у двух записей, то его нельзя использовать в качестве первичного ключа. Если таблица содержит ключевые поля из другой таблицы, то между ними образуется связь – взаимоотношением внешнего ключа (foreign key), например, «начальник-подчиненный» или «покупатель-покупка». Существуют следующие разновидности отношений (связей) между таблицами:

один-ко-многим. Самый распространенный вид связи. Одной записи таблицы соответствует любое количество (и 0, и 1, и даже еще больше) записей другой таблицы;

многие-ко-многим. Связь "многие-ко-многим" возникает между двумя таблицами, когда в каждой из них может присутствовать несколько ключей другой таблицы;

один-к-одному. При связи "один-к-одному" каждому элементу соответствует один и только один другой элемент.

Представление о взаимоотношениях данных и наиболее эффективном способе их организации называется нормализацией. Нормализация заключается в разделении данных на основе логических взаимоотношений с целью минимизировать дублирование данных.

Проектирование базы данных складывается из нескольких этапов и включает в себя концептуальное (инфологическое), логическое (даталогическое) и физическое проектирование.

Концептуальное (инфологическое) проектирование – это построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Чаще всего концептуальная модель базы данных включает в себя: описание информационных объектов или понятий предметной области и связей между ними и описание ограничений целостности, т.е. требований к допустимым значениям данных и к связям между ними [15].

Опишем предметную область проектируемой нами базы данных. Требуется разработать информационную систему управления аварийно-профилактической группой, для внесения, хранения и предоставления информации. Количество таблиц и связи между ними определяются задачами, которым должен отвечать разрабатываемый проект.

Таблица «author» хранит данные об имени автора, его дату рождения и идентификационный номер. У одного автора может быть несколько книг (связь один-ко-многим).

Таблица «book» предназначена для хранения информации о названии книги, содержания, количестве страниц, номера isbn, идентификационного номера жанра, к которому относится книга, идентификационного номера автора книги, годе издания, обложке, среднем рейтинге, общем количестве проголосовавших и описание.

Таблица «genre» хранит данный о названии жанра и его идентификационном номере.

Таблица «publisher» содержит информацию о названии издательства и его идентификационном номере.

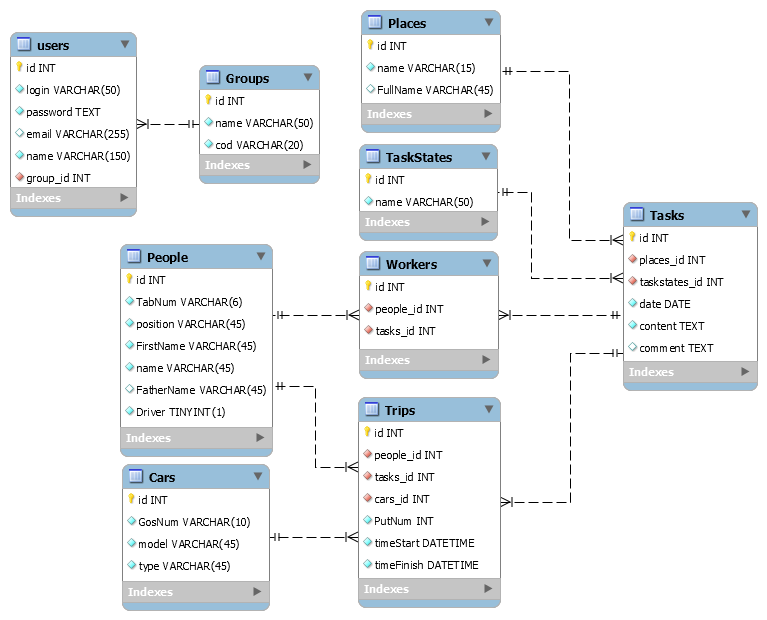
Таблица «votes», содержащая информацию об идентификационном номере книги, количества голосований, логина.

Таблица «user» используется в процессе авторизации, содержит номер, логин и пароль.

Таблица «user\_roles» содержит роли пользователей.

Логическое (даталогическое) проектирование – это создание схемы базы данных на основе конкретной модели данных. Для реляционной модели данных это набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Даталогическая модель разрабатываемой базы данных «wb\_db» представлена на рисунке 2.1.



**Рисунок 2.1 - Даталогическая модель базы данных**

Таблица «author» состоит из следующих полей:

* id, тип – bigint(20), является первичным ключом, имеет свойство AUTO\_INCREMENT, содержит уникальный номер;
* fio, тип – varchar(300);
* birthay, тип – date.

Таблица «book» содержит поля:

* id, тип – bigint(20), является первичным ключом, имеет свойство AUTO\_INCREMENT, содержит уникальный номер;
* name, тип – varchar(45);
* content, тип – longblob;
* page\_count, тип – int(11);
* isbn, тип – varchar(100), содержит уникальный номер;
* genre\_id, тип – bigint(20), является первичным ключом;
* author\_id, тип – bigint(20), является первичным ключом;
* publish\_year, тип – int(11);
* publisher\_id, тип – bigint(20), является первичным ключом;
* image, тип – longblob;
* avg\_rating, тип – int(5);
* total\_vote\_count, тип - bigint(20);
* total\_rating, тип – bigint(20);
* view\_count, тип – bigint(20);
* descr, тип – varchar(500).

Таблица «publisher» состоит из следующих полей:

* id, тип – bigint(20), является первичным ключом, имеет свойство AUTO\_INCREMENT, содержит уникальный номер;
* name, тип – varchar(100).

Таблица «genre» состоит из следующих полей:

* id, тип – bigint(20), является первичным ключом, имеет свойство AUTO\_INCREMENT, содержит уникальный номер;
* name, тип – varchar(100).

Таблица «vote» состоит из следующих полей:

* id, тип – bigint(20), является первичным ключом, имеет свойство AUTO\_INCREMENT, содержит уникальный номер;
* value, тип – int(5);
* book\_id, тип - bigint(20);
* username, тип – varchar(100).

Таблица «user» состоит из следующих полей:

* id, тип – int(11), является первичным ключом, имеет свойство AUTO\_INCREMENT;
* username, тип – varchar(50);
* password, тип - varchar(200);
* enabled, тип – tinyint(1);
* password\_sha256, тип – varchar(100).

Таблица «user\_roles» состоит из следующих полей:

* id, тип – int(11), является первичным ключом, имеет свойство AUTO\_INCREMENT;
* username, тип – varchar(50);
* role, тип - varchar(50).

Например, таблицы «author» и «book» связаны между собой по внешнему ключу через полe «id» таблицы «author» и «author\_id» таблицы «book», связь «один-ко-многим» – одному «id» может соответствовать несколько «author\_id» таблицы «book». Таблицы «genre» и «book» связаны между собой по внешнему ключу через полe «id» таблицы «genre» и «genre\_id» таблицы «book», связь «один-ко-многим» – одному «id» может соответствовать несколько «genre\_id» таблицы «book». Таблицы «publisher» и «book» связаны между собой по внешнему ключу через полe «id» таблицы «publisher» и «publisher\_id» таблицы «book», связь «один-ко-многим» – одному «id» может соответствовать несколько «publisher\_id» таблицы «book».

Необходимо отметить, что для работы с базой данных и связи Java классов с таблицами базы данных используется технология Hibernate, которая не только заботится о связи Java классов с таблицами базы данных (и типов данных Java в типы данных [SQL](http://ru.wikipedia.org/wiki/Sql)), но также предоставляет средства для автоматического построения запросов и извлечения данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL и [JDBC](http://ru.wikipedia.org/wiki/Jdbc) кода. Hibernate генерирует SQL вызовы и освобождает разработчика от ручной обработки результирующего набора данных и конвертации объектов, сохраняя приложение портируемым во все SQL базы данных [16].

В проекте использованы следующие аннотации:

@Entity — Указывает, что данный бин (класс) является сущностью.

@Table — указывает на имя таблицы, которая будет отображаться в этой сущности.

@Column — указывает на имя колонки, которая отображается в свойство сущности.

@Id — идентификационный номер колонки.

@GeneratedValue — указывает, что данное свойство будет создаваться согласно указанной стратегии.

@Lob — указание на большие объекты.

@JoinColumn  — применяется когда внешний ключ находится в одной из сущностей. Используется для получения готового объекта, а не его идентификационного номера.

@ManyToOne — указывает на связь многие к одному.

@OneToMany — указывает на связь [один ко многим](http://javastudy.ru/hibernate/hibernate-one-to-many/).

Программный код создания базы данных представлен в Приложении А.

**2.2 Разработка классов**

При разработке нашего программного продукта мы используем объектно-ориентированное программирование на языке JAVA. Объектно-ориентированные программы более просты и мобильны, их легче модифицировать и сопровождать, чем их «традиционных» собратьев.

Стратегию ООП лучше всего описать как смещение приоритетов в процессе программирования от функциональности приложения к структурам данных. Это позволяет программисту моделировать в создаваемых приложениях реальные объекты и ситуации.

Специфика ООП заметно повышает эффективность труда программистов и позволяет им создавать более мощные, масштабируемые и эффективные приложения. Объектно-ориентированное программирование основано на следующих принципах:

Инкапсуляция – это механизм, объединяющий данные и обрабатывающий их код как единое целое. Инкапсуляцией называется включение различных мелких элементов в более крупный объект, в результате чего программист работает непосредственно с этим объектом. Это приводит к упрощению программы, поскольку из нее исключаются второстепенные детали.

Полиморфизм позволяет использовать одни и те же имена для похожих, но технически разных задач. Главным в полиморфизме является то, что он позволяет манипулировать объектами путем создания стандартных интерфейсов для схожих действий. Полиморфизм значительно облегчает написание сложных программ.

Наследование позволяет одному объекту приобретать свойства другого объекта, при наследовании точная копия дополняется уникальными свойствами, которые характерны только для производного объекта [10].

Кроме того, при разработке управления аварийно-профилактической группой была использована концепция Model-view-controller (MVC, «модель-представление-контроллер). Это схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

Основная цель применения этой концепции состоит в разделении бизнес-логики (модели) от её визуализации (представления, вида). За счет такого разделения повышается возможность повторного использования. К одной модели можно присоединить несколько видов, не затрагивая при этом реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы. Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных), для этого достаточно использовать другой контроллер.

Модель (англ. Model) предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

Представление, вид (англ. View) отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.

Контроллер (англ. Controller) обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели [12].

Опишем процесс разработки проекта. С самого начала необходимо было создать проект с использованием сборщика Gradle и указанием в файле build.gradle всех необходимых зависимостей для дальнейшей работы. Для разработки логики приложения необходимы были следующие зависимости:

- org.springframework.boot:spring-boot-starter-data-jpa – для работы с Hibernate с помощью Spring;

- org.springframework.boot:spring-boot-starter-web – для создания веб-контейнера;

- org.springframework.boot:spring-boot-starter-security – для создания безопасности веб-приложения, которая включает авторизацию и аутентификацию;

- org.springframework.boot:spring-boot-starter-actuator – для анализа работы служебных данных во время работы приложения, например, бинов и эндпоинтов;

- org.springframework.boot:spring-boot-starter-aop - для использования аспектов, например, для отображения сообщений о действиях пользователя;

- org.springframework.boot:spring-boot-starter-data-rest – для автоматической генерации рестфул сервисов;

- org.projectlombok:lombok - плагин, который позволяет существенно сократить написание кода, например, не нужно прописывать геттеры и сеттеры в коде, достаточно будет лишь прописать аннотации @Getter и @Setter.

Далее необходимо создать файл с настройками для Spring Boot приложения, где прописываем настройки для логирования, связи с базой данных и Hibernate. В нашем проекте таким файлом является файл application.yml, который обязательно должен находиться в папке ресурсы, откуда указанные настройки и конфигурации будут автоматически считываться при запуске проекта. Файл application.yml выглядит следующим образом:

spring:

datasource: # для связи с БД

#default-catalog: library

url: jdbc:mysql://localhost:3306/library

jndi-name: jdbc/Library # для связи с glassfish

username: root

password: 1234si

jpa: #настройки Hibernate

database-platform: org.hibernate.dialect.MySQL5InnoDBDialect

properties:

hibernate:

current\_session\_context\_class: org.springframework.orm.hibernate5.SpringSessionContext # пока спринг контейнер будет активный для нашего приложения, сессия Hibernate тоже будет держаться

dialect: org.hibernate.dialect.MySQL5Dialect

format\_sql: true

show-sql: true

data:

rest:

base-path: /repo

logging:

file: "c:\temp\app.log pattern: %d{HH:mm:ss} %-5p [%c] - %m%n"

level:

ROOT: error

org.springframework: warn

javax.faces: warn

org.omnifaces: warn

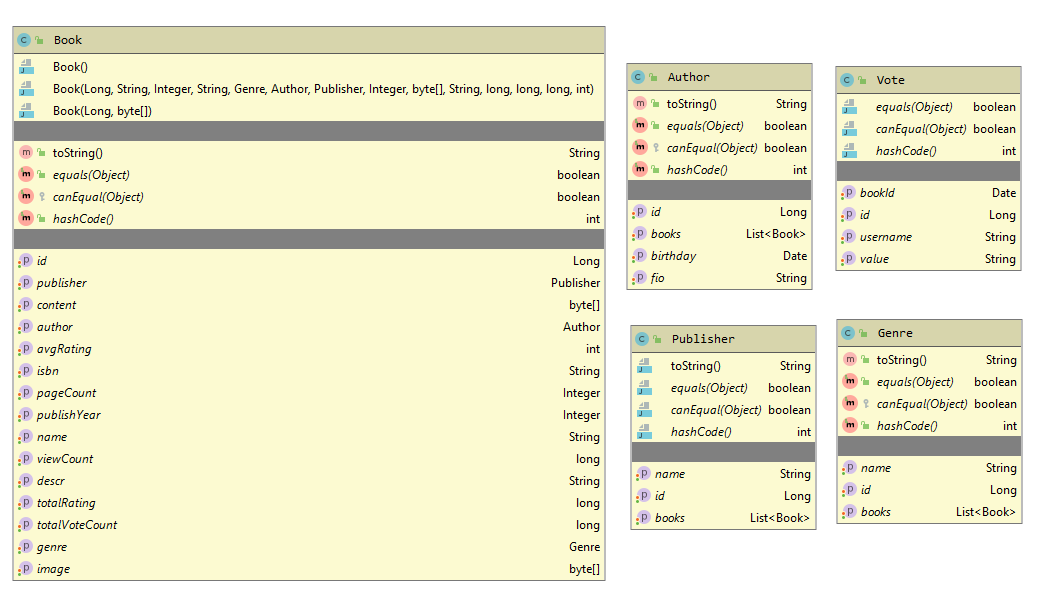
org.springframework.security: warn

org.hibernate: warn

org.hibernate.type.descriptor.sql: warn

Далее приступаем к разработке классов-сущностей, с помощью которых Hibernate сможет отправлять запросы в базу данных и получать ответы. На этом этапе создается абстрактный уровень описания объектов для Hibernate.

Классы – сущности представлены на рисунке 2.2.



**Рисунок 2.2 – Классы-сущности**

Класс «Author» содержит следующие поля:

­ private Long id – идентификационный номер автора, для которого указываются аннотации @Id - для обозначения первичного ключа класса и @GeneratedValue(strategy = GenerationType.IDENTITY) – для автоинкремента поля;

­ private String fio –имя и фамилия автора;

­ private Date birthday – информация о дате рождения автора.

Класс «Book» содержит следующие поля:

- private Long id – идентификационный номер книги, для которого указываются аннотации @Id - для обозначения первичного ключа класса и @GeneratedValue(strategy = GenerationType.IDENTITY) – для автоинкремента поля;

­ private String name – название книги;

­ private byte[] content – содержание книги;

- private Integer pageCount – информация количестве страниц;

- private String isbn – специальный номер книги;

- private Genre genre – жанр книги, также указываем аннотацию @JoinColumn для получения названия жанра, а не его идентификационного номера;

- private Author author – автор книги, также указываем аннотацию @JoinColumn для получения имени автора, а не его идентификационного номера;

- private Publisher publisher– издательство, аннотация @JoinColumn используется для получения названия издательства, а не его идентификационного номера;

- private Integer publishYear – год издания книги;

- private byte[] image – изображение обложки книги;

- private long viewCount – счетчик просмотров;

- private long totalRating – рейтинг книги;

- private long totalVoteCount – общее количество просмотров;

- private int avgRating – средний рейтинг.

Класс «Genre» содержит следующие поля:

- private Long id – идентификационный номер жанра, для которого ука-зываются аннотации @Id - для обозначения первичного ключа класса и @GeneratedValue(strategy = GenerationType.IDENTITY) – для автоинкремента поля;

­ private String name – название жанра.

Класс «Publisher» содержит следующие поля:

- private Long id – идентификационный номер издательства, для которого указываются аннотации @Id - для обозначения первичного ключа класса и @GeneratedValue(strategy = GenerationType.IDENTITY) – для автоинкремента поля;

­ private String name – название издательства.

Класс «Vote» содержит следующие поля:

- private Long id – идентификационный номер голоса, для которого ука-зываются аннотации @Id - для обозначения первичного ключа класса и @GeneratedValue(strategy = GenerationType.IDENTITY) – для автоинкремента поля;

­ private String value – значение голоса (в данном случае количесвтво звезд, выставленное пользователем);

- private long bookId – идентификационный номер книги, за которую проголосовали;

- private String username – имя пользователя;

Программный код используемых классов представлен в приложении В.

Необходимо отметить, что связывание джава – классов с таблицами в базе данных происходит при помощи мэппинга, когда название класса является названием таблицы, а поля класса- столбцами таблицы в базе данных.

Следующим шагом в создании информационной системы управления аварийно-профилактической группой является создание DAO (Data Access Object) – шаблон проектирования; верхний (абстрактный) слой доступа к данным, который задает основные бизнес-процессы для реализации. DAO представляют собой интерфейсы, в которых прописываются методы, описывающие поведение объектов. Центральным интерфейсом является параметризованный (обобщенный) интерфейс «GeneralDAO», который сможет работать с разными типами данных. Остальные интерфейсы такие как: «AuthorDAO», «BookDAO», «GenreDAO» и «PublisherDAO» являются его наследниками и переопределяют его методы.

В интерфейсе «GeneralDAO» прописаны следующие методы:

List<T> getAll() –получение всех записей (без постраничности);

T get(long id) - получение объекта по идентификационному номеру;

T save(T obj) - редактирование и добавление объекта;

void delete(T object) - удаление объекта;

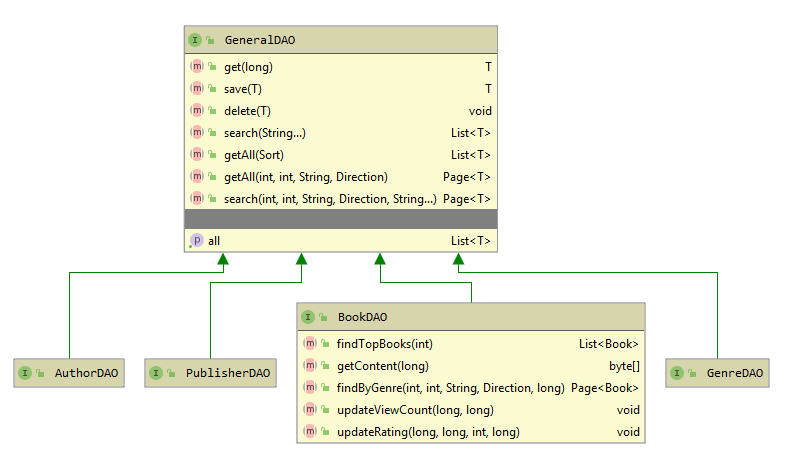
List<T> search(String... searchString) - поиск записей с любым количеством параметров;

List<T> getAll(Sort sort) - получение всех данных с сортировкой результата;

Page<T> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) - получение всех записей с постраничностью;

Page<T> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String...searchString) - поиск записей с постраничностью.

Интерфейсы DAO представлены рисунке 2.3.



**Рисунок 2.3 – DAO интерфейсы**

Для полноценной работы с базой данных нам необходимо создать репозиторий – специализированные интерфейсы с дефолтной реализацией доступа к данным. Каждый репозиторий работает со своим классом – сущностью. В проекте разработаны следующие репозитории: «AuthorRepository», «BookRepository», «GenreRepository», «PublisherRepository», каждый из которых наследуется от JpaRepository, что позволяет использовать готовые методы, например, CRUD (create, read, update, delete) операции и встроенные возможности для постраничного вывода.

Каждый репозиторий должен быть помечен специальным Spring бином @Repository, благодаря которому Spring различает эти интерфейсы как репозиторий и ведет себя с ними соответственно.

Для создания запросов будем использовать именованные запросы, которые поддерживает Hibernate, а также HQL- запросы для специфических запросов.

На примере созданного «BookRepository» можно увидеть реализацию разных видов запросов:

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

Page<Book> findByNameContainingIgnoreCaseOrAuthorFioContainingIgnoreCaseOrderByName(String name, String fio, Pageable pageable ) – именованный запрос для поиска книги по ее названию или имени автора;

@Query("select new ru.javabegin.training.library.springlibrary\_2.domain.Book (b.id, b.name, b.pageCount, b.isbn, b.genre, b.author, b.publisher, b.publishYear, b.image, b.descr, b.viewCount, b.totalRating, b.totalVoteCount, b.avgRating) from Book b") – HQL запрос;

Page<Book> findAllWithoutContent(Pageable pageable) – возвращает список книг с постраничностью с помощью HQL запроса;

Аналогичным способом разрабатываем другие запросы. Необходимо отметить, что в классах - сущностях должны быть прописаны соответствующие конструкторы, чтобы можно было оперировать полученными данными и если запрос изменяет данные, то нужно указать аннотацию @Modifying.

@Modifying(clearAutomatically = true)

@Query("update Book b set b.content=:content where b.id =:id")

void updateContent(@Param("content") byte[] content, @Param("id") long id) – обновление контента по идентификационному номеру книги;

@Query("select new ru.javabegin.training.library.springlibrary\_2.domain.Book(b.id, b.image) from Book b")

List<Book> findTopBooks(Pageable pageable) – отображение топовых книг;

@Query("select new ru.javabegin.training.library.springlibrary\_2.domain.Book (b.id, b.name, b.pageCount, b.isbn, b.genre, b.author, b.publisher, b.publishYear, b.image, b.descr, b.viewCount, b.totalRating, b.totalVoteCount, b.avgRating) from Book b where b.genre.id=:genreId")

Page<Book> findByGenre(@Param("genreId") long genreId, Pageable pageable) – поиск книг по жанру;

@Query("select b.content FROM Book b where b.id = :id")

byte[] getContent(@Param("id") long id) – получение контента по идентификационному номеру книги;

@Modifying

@Query("update Book b set b.viewCount=:viewCount where b.id =:id")

void updateViewCount(@Param("viewCount") long viewCount, @Param("id") long id) – обновление статистики просмотров для книги;

@Modifying

@Query("update Book b set b.totalVoteCount=:totalVoteCount, b.totalRating=:totalRating, b.avgRating=:avgRating where b.id =:id")

void updateRating(@Param("totalRating") long totalRating, @Param("totalVoteCount") long totalVoteCount, @Param("avgRating") int avgRating, @Param("id") long id) – обновление данных рейтинга.

При разработке «AuthorRepository» использовались именованные запросы такие как:

List<Author> findByFioContainingIgnoreCaseOrderByFio(String fio) – поиск автора по его имени или фамилии;

Page<Author> findByFioContainingIgnoreCaseOrderByFio(String fio, Pageable pageable) – поиск автора по его имени и фамилии с постраничностью, что позволяет выводить данные по частям и не перегружать базу данных.

Аналогичные запросы использовались при разработке «GenreRepository» и «PublisherRepository», которые позволяют осуществлять поиск по жанру и издательству с постраничностью и без соответственно.

При разработке информационной системы управления аварийно-профилактической группой было спроектировано разделение на репозиторий, который отвечает только за доступ к данным, и сервис (помечается специальным Spring бином @Service), который отвечает за реализацию бизнес – процессов и использует репозиторий для получения нужных данных с помощью аннотации @Autowired. Таким образом, везде из кода идет обращение к сервису, а не напрямую к репозиторию или классам- сущностям.

Сервисный уровень должен реализовывать DAO и предоставлять удобное API.

В данном проекте разработаны «AuthorService», «BookService», «GenreService» и «PublisherService» классы-сервисы.

Рассмотрим класс «AuthorService», который наследует интерфейс «AuthorDAO» и переопределяет его методы.

public class AuthorService implements AuthorDAO {

@Autowired – аннотация для внедрения «AuthorRepository» репозитория;

private AuthorRepository authorRepository;

@Override – аннотация для переопределения метода;

public List<Author> getAll() {

return authorRepository.findAll() – получение списка всех авторов;

}

public List<Author> getAll(Sort sort) {

return authorRepository.findAll(sort) – получение всех аворов с сортировкой;

}

@Override

public Page<Author> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

return authorRepository.findAll( PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField))) – постраничное выведение всех авторов;

}

@Override

public List<Author> search(String... searchString) {

return authorRepository.findByFioContainingIgnoreCaseOrderByFio(searchString[0]) – поиск авторов по имени;

}

@Override

public Page<Author> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String... searchString) {

return authorRepository.findByFioContainingIgnoreCaseOrderByFio(searchString[0], PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField))) – поиск авторов по имени с постраничным выведением;

}

@Override

public Author save(Author author) {

return authorRepository.save(author) – сохранение автора;

}

@Override

public void delete(Author author) {

authorRepository.delete(author) – удаление автора;

}

@Override

public Author get(long id) {

Optional<Author> bookmark = authorRepository.findById(id) - *Optional* - обертка, в которой может быть значение или пусто (используется для исключение ошибки NullPointerException;

if (bookmark.isPresent()) {

return bookmark.get();

} else {

return null;

Класс «BookService» также наследует интерфейс «BookDAO» и переопределяет его методы.

@Autowired – аннотация для внедрения репозитория «BookRepository»;

private BookRepository bookRepository;

@Override

public List<Book> getAll() {

return bookRepository.findAll() – получение всех книг;

}

@Override

public List<Book> getAll(Sort sort) {

return bookRepository.findAll(sort) – получение книг с сортировкой;

}

@Override

public Page<Book> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

return bookRepository.findAllWithoutContent(PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField))) – получение книг без содержания;

}

@Override

public List<Book> search(String... searchString) {

return null;

}

@Override

public Page<Book> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String... searchString) {

return bookRepository.findByNameContainingIgnoreCaseOrAuthorFioContainingIgnoreCaseOrderByName(searchString[0], searchString[0], PageRequest.of (pageNumber, pageSize, Sort.by (sortDirection, sortField))) – поиск книг по названию или имени автора с постраничным выведением;

}

@Override

public Book save(Book book) {

bookRepository.save(book) – отдельное сохранение книг;

if (book.getContent()!=null) {

bookRepository.updateContent(book.getContent(), book.getId()) – отдельное сохраниение содержания книги;

}

return book;

}

@Override

public void delete(Book book) {

bookRepository.delete(book) – удаление книги;

}

@Override

public Book get(long id) {

Optional<Book> bookmark = bookRepository.findById(id);

if (bookmark.isPresent()) {

return bookmark.get() – возвращение книги по идентификационному номеру;

} else {

return null;

}

}

@Override

public byte[] getContent(long id) {

return bookRepository.getContent(id) – получение содержания книги;

}

public List<Book> findTopBooks(int limit) {

return bookRepository.findTopBooks( PageRequest.of(0,limit, Sort.by(Sort.Direction.DESC, "viewCount"))) – получение топовых книг;

}

@Override

public Page<Book> findByGenre(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, long genreId) {

return bookRepository.findByGenre(genreId, PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField))) – получение книг соответствующего жанра;

}

@Override

public void updateViewCount(long viewCount, long id) {

bookRepository.updateViewCount(viewCount, id) – обновление количества просмотров книги;

}

@Override

public void updateRating(long totalRating, long totalVoteCount, int avgRating, long id) {

bookRepository.updateRating(totalRating, totalVoteCount, avgRating, id) – обновление рейтинга книги;

}

Аналогичные методы разработаны для класса «GenreService», который наследует интерфейс «GenreDAO» и переопределяет его методы, и класса «PublisherService», наследующий интерфейс «PublisherDAO» и переопределяющий его методы.

При запуске проекта мы попадаем в класс - контроллер «RedirectController», откуда потом происходит перенаправление на страницу «Index.xhtml».

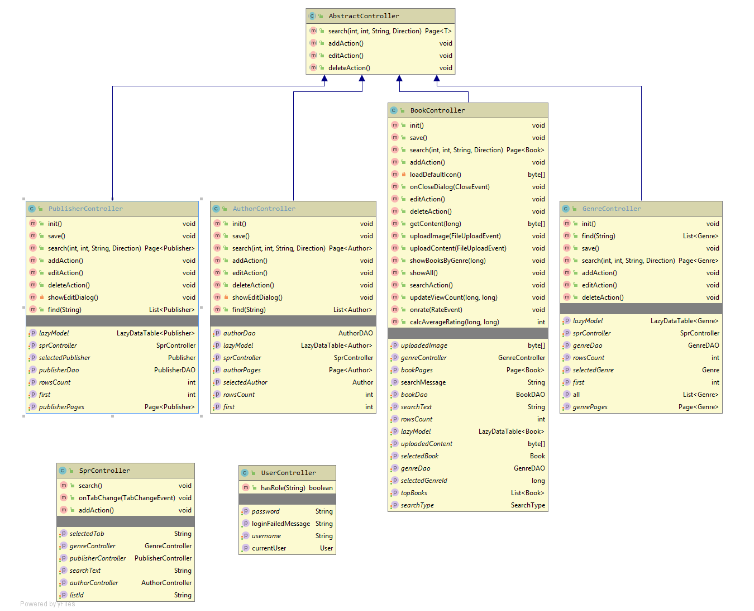
@RequestMapping(value = "", method = RequestMethod.GET)

public String baseUrlRedirect(HttpServletRequest request, HttpServletResponse httpServletResponse) {

return "redirect:" + request.getRequestURL().append("/index.xhtml").toString();

Необходимо отметить, что JSF и Spring имеют свои контейнеры, которые нужно совместить. Так, все запросы «.xhtml» будут обрабатываться JSF, а все остальные запросы – контроллерами Spring. Именно контроллеры JSF будут использоваться для обработки действий пользователя и отображения результата. Для постраничного вывода данных используется специальный объект LazyDataModel, специальный класс, который вызывается при каждом переходе по страницам, который в свою очередь вызывает API сервисного уровня для получения данных из базы данных.

Для обработки действий пользователя разработаны классы – контроллеры JSF, являющиеся связующим звеном JSF и API сервисного уровня. На рисунке 2.4 представлены разработанные классы контроллеры.



**Рисунок 2.4 – Классы - контроллеры JSF**

Рассмотрим подробнее некоторые из них. Центральным классом является абстрактный класс «AbstractController», который описывает необходимое поведение для всех JSF контроллеров, таких как: «AuthorController», «BookController», «GenreController», «PublisherController», которые наследуют «AbstractController», а также «SprController», являющийся общим контроллером для «Справочников», и «UserController», использующийся для авторизации и разграничения ролей пользователей.

В классе «AbstactController» прописаны следующие общие методы:

public abstract Page<T> search(int first, int count, String sortField, Sort.Direction sortDirection) – постраничное выведение книг;

public abstract void addAction() – добавление книг;

public abstract void editAction() – редактирование книг;

public abstract void deleteAction() – удаление книг.

Для того, чтобы со страницы JSF можно было получать методы и данные джава классов, классы-контроллеры необходимо обозначить специальными бинами «@ManageBean» (управляемый бин) и «@Component» (позволяет искать бины-сервисы автоматически).

Рассмотрим реализацию класса «BookController».

public class BookController extends AbstractController<Book> {

public static final int DEFAULT\_PAGE\_SIZE = 20 - указывает количество книг, отображаемое по-умолчанию;

public static final int TOP\_BOOKS\_LIMIT = 5 – указывает на количество популярных книг для отображения;

private int rowsCount = DEFAULT\_PAGE\_SIZE – количество записей на странице, которое будет сохраняться;

private SearchType searchType - запоминание последнего выбранного варианта поиска;

@Autowired

private BookDAO bookDao – автоматически будет подставлен «BookService», т.к. Spring контейнер ищет бин по типу

@Autowired

private GenreDAO genreDao- автоматически будет подставлен «GenreService»;

@Autowired

private GenreController genreController;

private Book selectedBook - ссылка на текущую книгу (которую редактируют, хотят удалять и пр.) - т.е. над какой книгой в данный момент производим действие;

private LazyDataTable<Book> lazyModel - класс-утилита, которая помогает выводить данные постранично (работает в паре с компонентами на странице JSF);

private byte[] uploadedImage - сюда будет сохраняться загруженная пользователем новая обложка (при редактировании или при добавлении книги);

private byte[] uploadedContent - сюда будет сохраняться загруженный пользователем PDF контент (при редактировании или при добавлении книги);

private Page<Book> bookPages - хранит список найденных книг;

private List<Book> topBooks - хранит полученные ТОП книги (может использоваться, например, для получения изображений книги);

private String searchText - введенный текст для поиска;

private long selectedGenreId - выбранный жанр для поиска;

@PostConstruct - после того, как создан «@ManagedBean», будет автоматически вызываться метод init

public void init() {

lazyModel = new LazyDataTable(this);

}

public void save() {

if (uploadedImage != null) { - если было выбрано новое изображение

selectedBook.setImage(uploadedImage);

}

if (uploadedContent != null) { - если был выбран новый PDF контент

selectedBook.setContent(uploadedContent);

}

bookDao.save(selectedBook);

RequestContext.getCurrentInstance().execute("PF('dialogEditBook').hide()") - вызов JS из java кода;

}

@Override - переопределенный метод автоматически вызывается из LazyDataTable;

public Page<Book> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

if (sortField == null) {- если не указано поле сортировки берем поле "name";

sortField = "name";

}

if (searchType == null){

bookPages = bookDao.getAll(pageNumber, pageSize, sortField, sortDirection) - отображение всех книг постранично, если не указан тип поиска;

}else {

switch (searchType) {

case SEARCH\_GENRE: - поиск по жанру;

bookPages = bookDao.findByGenre(pageNumber, pageSize, sortField, sortDirection, selectedGenreId);

break;

case SEARCH\_TEXT: - поиск по введенному тексту;

bookPages = bookDao.search(pageNumber, pageSize, sortField, sortDirection, searchText);

break;

case ALL: - отображение всех книг;

bookPages = bookDao.getAll(pageNumber, pageSize, sortField, sortDirection);

break;

}

}

return bookPages;

}

@Override

public void addAction() {

selectedBook = new Book();

uploadedImage = loadDefaultIcon();

uploadedContent = null;

RequestContext.getCurrentInstance().execute("PF('dialogEditBook').show()");

}

private byte[] loadDefaultIcon(){- загрузка картинки по-умолчанию для обложки

InputStream stream = FacesContext.getCurrentInstance().getExternalContext().getResourceAsStream("/resources/images/no-cover.jpg");

try {

return IOUtils.toByteArray(stream);

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

public void onCloseDialog(CloseEvent event) {- очищать загруженный контент из переменной при закрытии диалогового окна

uploadedContent = null;

}

@Override

public void editAction() { - редактирование книги;

uploadedImage = selectedBook.getImage();

RequestContext.getCurrentInstance().execute("PF('dialogEditBook').show()");

}

@Override

public void deleteAction() { - удаление книги;

bookDao.delete(selectedBook);

}

public String getSearchMessage(){- сообщение, сколько данных найдено и по какому критерию;

ResourceBundle bundle = ResourceBundle.getBundle("library", FacesContext.getCurrentInstance().getViewRoot().getLocale()) - доступ к файлам локализации;

String message=null;

if (searchType==null){

return null;

}

switch (searchType) {

case SEARCH\_GENRE:

message = bundle.getString("genre")+ ": '"+genreDao.get(selectedGenreId)+"'";

break;

case SEARCH\_TEXT:

if (searchText==null || searchText.trim().length()==0){

return null;

}

message = bundle.getString("search")+ ": '"+searchText+"'";

break;

}

return message;

}

public byte[] getContent(long id) {- получение PDF контент книги для чтения

byte[] content;

if (uploadedContent != null) {

content = uploadedContent;

} else {

content = bookDao.getContent(id);

}

return content;

}

public void uploadImage(FileUploadEvent event) {- при загрузке обложки - она будет сохраняться в переменную «uploadedImage»;

if (event.getFile() != null) {

uploadedImage = event.getFile().getContents();

}

}

public void uploadContent(FileUploadEvent event) {- при загрузке PDF контента - он будет сохраняться в переменную «uploadedContent»;

if (event.getFile() != null) {

uploadedContent = event.getFile().getContents();

}

}

public List<Book> getTopBooks() {

topBooks = bookDao.findTopBooks(TOP\_BOOKS\_LIMIT);

return topBooks;

}

public void showBooksByGenre(long selectedGenreId){ - поиск по определенному жанру;

searchType = SearchType.SEARCH\_GENRE;

this.selectedGenreId = selectedGenreId;

}

public void showAll(){

searchType = SearchType.ALL;

}

public void searchAction(){

searchType = SearchType.SEARCH\_TEXT;

}

public Page<Book> getBookPages(){

return bookPages;

}

public void updateViewCount(long viewCount, long id){ - обновление количества просмотров;

bookDao.updateViewCount(viewCount+1, id);

}

public void onrate(RateEvent rateEvent) {- вызывается при голосовании за книгу;

Map<String, String> params = FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap();

int bookIndex = Integer.parseInt(params.get("bookIndex")) - параметр индекса книги, который передается со страницы;

Book book = bookPages.getContent().get(bookIndex) - по индексу получаем книгу, для которой проголосовали;

long currentRating = Long.valueOf(rateEvent.getRating().toString()).longValue() - какой рейтинг поставил пользователь;

long newRating = book.getTotalRating() + currentRating - новый рейтинг (суммарный);

long newVoteCount = book.getTotalVoteCount()+1 - сколько проголосовало;

int newAvgRating = calcAverageRating(newRating, newVoteCount)- среднее значение, которое показывается на странице;

bookDao.updateRating(newRating, newVoteCount, newAvgRating, book.getId());

}

public int calcAverageRating(long totalRating, long totalVoteCount) {

if (totalRating == 0 || totalVoteCount == 0) {

return 0;

}

int avgRating = Long.valueOf(totalRating / totalVoteCount).intValue();

return avgRating;

}

Класс- контроллер для справочников «SprController» выглядит следующим образом:

public class SprController {

@Autowired

private AuthorController authorController;

@Autowired

private GenreController genreController;

@Autowired

private PublisherController publisherController;

private String listId = "tabView:authorForm:authorList" - по-умолчанию открытым табом является таб для редактирования авторов;

private String searchText - текст поиска;

private String selectedTab = "tabAuthors" - выбранный пользователем таб (по-умолчанию выбран первый таб с авторами);

public void search(){- вызывается при поиске справочных значений;

RequestContext context = RequestContext.getCurrentInstance();

context.update(listId);

}

public void onTabChange(TabChangeEvent event) {- возвращает нужный идентификационный номер для обновления области на странице;

searchText = null;

selectedTab = event.getTab().getId();

switch (selectedTab){

case "tabAuthors":

listId = "tabView:authorForm:authorList";

break;

case "tabGenres":

listId = "tabView:genreForm:genreList";

break;

case "tabPublishers":

listId = "tabView:publisherForm:publisherList";

break;

}

RequestContext.getCurrentInstance().update("searchForm:searchInput");

}

В зависимости от выбранного таба - будем показываться соответствующее диалоговое окно:

public void addAction(){

switch (selectedTab){

case "tabAuthors":

authorController.addAction();

RequestContext.getCurrentInstance().update("tabView:dialogAuthor");

break;

case "tabGenres":

genreController.addAction();

RequestContext.getCurrentInstance().update("tabView:dialogGenre");

break;

case "tabPublishers":

publisherController.addAction();

RequestContext.getCurrentInstance().update("tabView:dialogPublisher");

break;

}

Рассмотрим также контроллер для авторизации «UserController»:

public class UserController {

private String username – имя пользователя;

private String password – пароль пользователя;

public User getCurrentUser(){

return (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

}

Проверка есть ли роль у пользователя:

public boolean hasRole(String role){

return getCurrentUser().getAuthorities().stream().filter(x -> x.getAuthority().equals("ROLE\_"+role)).count()>0;

}

Локализованное сообщение о том, что неправильно ввели логин или пароль:

public String getLoginFailedMessage(){

Object obj = FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("loginFailed");

if (obj==null) return "";

FacesContext context = FacesContext.getCurrentInstance();

ResourceBundle bundle = context.getApplication().getResourceBundle(context, "msg");

String message = bundle.getString("login\_failed");

FacesContext.getCurrentInstance().getExternalContext().getSessionMap().remove("loginFailed");

if (Strings.isNullOrEmpty(message)){

return "";

}else{

return message;

}

Программный код используемых классов представлен в приложении В.

**2.3 Интерфейс информационной системы управления аварийно-профилактической группой**

В разрабатываемом проекте для связи бизнес – процессов с интерфейсом используется официальная стандартная технология для JAVA – JSF (Java Server Facelets), отвечающая за отображение UI по технологиям XHTML и Facelets.

Первым делом необходимо подключить нужные зависимости в файл build.gradle. Для создания интерфейса используются следующие зависимости:

- org.primefaces:primefaces – для создания внешних компонентов отображения данных;

- org.omnifaces:omnifaces – для улучшения функциональности инфраструктуры JSF;

- commons-io:commons-io и commons-fileupload:commons-fileupload - для загрузки файлов в приложении;

Для указания настроек JSF веб-проекта создается папка WEB-INF с файлами web.xml, glassfish-web.xml и faces-config.xml, где и указывается специальный класс «org.springframework.web.jsf.el.SpringBeanFacesELResolver», который позволяет совместить контейнеры Spring и JSF между собой, а также, например, название файла с ресурсами для изменения локали (используются русский и английский языки).

Перед созданием непосредственно интерфейса веб-страницы необходимо ее разметить, определить необходимые области, для которых в дальнейшем создаются шаблоны страниц, с помощью библиотеки BootsFaces, а именно системы сеток (Grid system). В данном проекте были разработаны следующие области:

- меню, которое занимает 12 столбцов;

- топовые книги занимают также 12 столбцов;

- список жанров занимает 2 столбца;

- список книг – 10 столбцов;

- футер занимает также 12 столбцов.

Загрузка проекта начинается с попадания в «RedirectController», в котором происходит перенаправление в файл «index.xhtml», в котором реализован интерфейс страницы авторизации. Алгоритм работы представлен в приложении Б. Рассмотрим основные элементы данного файла:

<h:head>

<title>${msg.online\_library}</title> - отображение названия проекта;

<h:outputStylesheet name="css/index.css" /> - подключение стилей;

</h:head>

<h:body>

<f:view locale="#{localeChanger.currentLocale}" > - отображение локали;

<div id="login">

<h:outputText id="loginFailed" value= "#{userController.loginFailedMessage}" styleClass = "error"/> - ошибка о неверноем логине или пароле;

<form id="login\_form" action="#{request.contextPath}/login" method="post"> - отображение формы авторизации;

<span class="fontawesome-user"></span>

<input

type="text" id="username" name="username" class="ui-corner-all" – поле ввода логина;

required="required" placeholder="username"/>

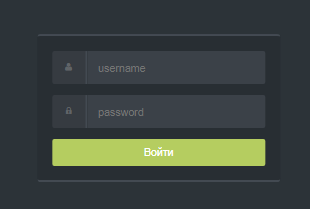
<span class="fontawesome-lock"></span>

<input type="password" id="password" name="password" – поле ввода пароля;

class="ui-corner-all" required="required" placeholder="password"/>

<input type="submit" value="#{msg.login}"/> - кнопка «Войти».

</form>



**Рисунок 2.5 – Страница авторизации**

После прохождения авторизации пользователь попадает на главную страницу управления аварийно-профилактической группой. С точки зрения реализации, данная страница, представленная на рисунке 2.6, состоит из нескольких подключаемых страниц, что дает возможность использовать уже написанный код несколько раз, а также облегчает и ускоряет разработку. Основной страницей является «Books-template.xhtml» страница, которая содержит следующие элементы:

<title>${msg.online\_library}</title> - отображение названия проекта;

</h:head>

<f:view locale="#{localeChanger.currentLocale}"> - указание текущего языка;

<ui:include src="books-header.xhtml"/> - подключение страницы меню;

<ui:include src="books-top.xhtml"/> - подключение страницы топовых книг;

<b:container>

<b:row>

<b:column span="2">

<ui:include src="books-leftmenu.xhtml"/> - подключение страницы списка жанров;

</b:column>

<b:column span="10">

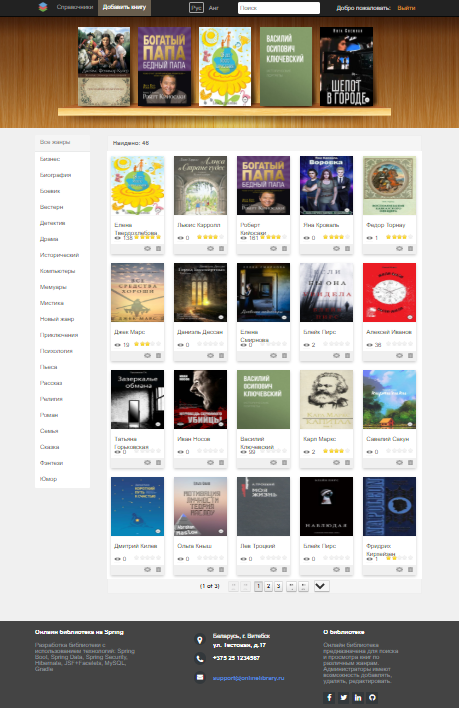
<ui:insert name="content"/> - подстановка контента (в данном случае со страницы «Books.xhtml»;

</b:column>

</b:row>

</b:container>

<ui:include src="../part/footer.xhtml"/> - подключение футера проекта.

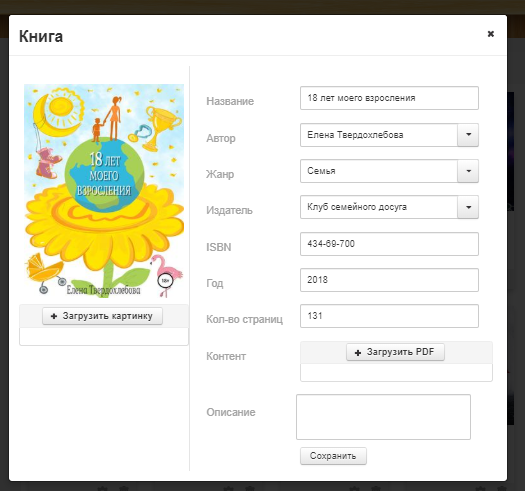


**Рисунок 2.6 – Главная страница управления аварийно-профилактической группой**

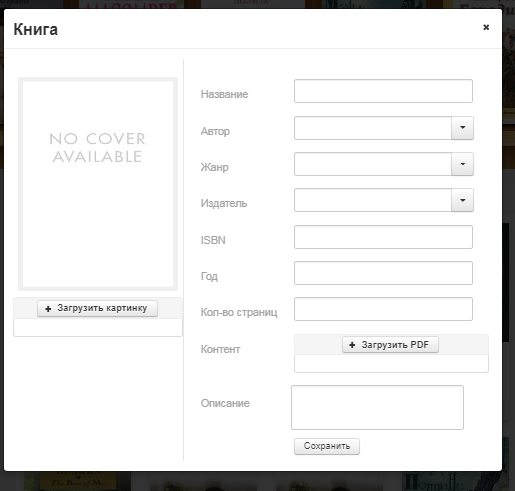
Слева располагается список жанров, и при нажатии пользователем на один из них, происходит поиск книг по определенному жанру. При нажатии на обложку книги, открывается pdf файл с содержанием выбранной книги в отдельном окне, где пользователь может его читать, скачивать и распечатывать. При открытии содержания книги, увеличивается количество ее просмотров, которое отображается после названия книги. Также у пользователя есть возможность проголосовать за книгу и переключить язык на английский.

Если пользователь является администратором, то у него появляется возможность добавлять, редактировать, удалять книги.

Внешний вид окон добавления и редактирования книги представлен на рисунках 2.6 и 2.7.



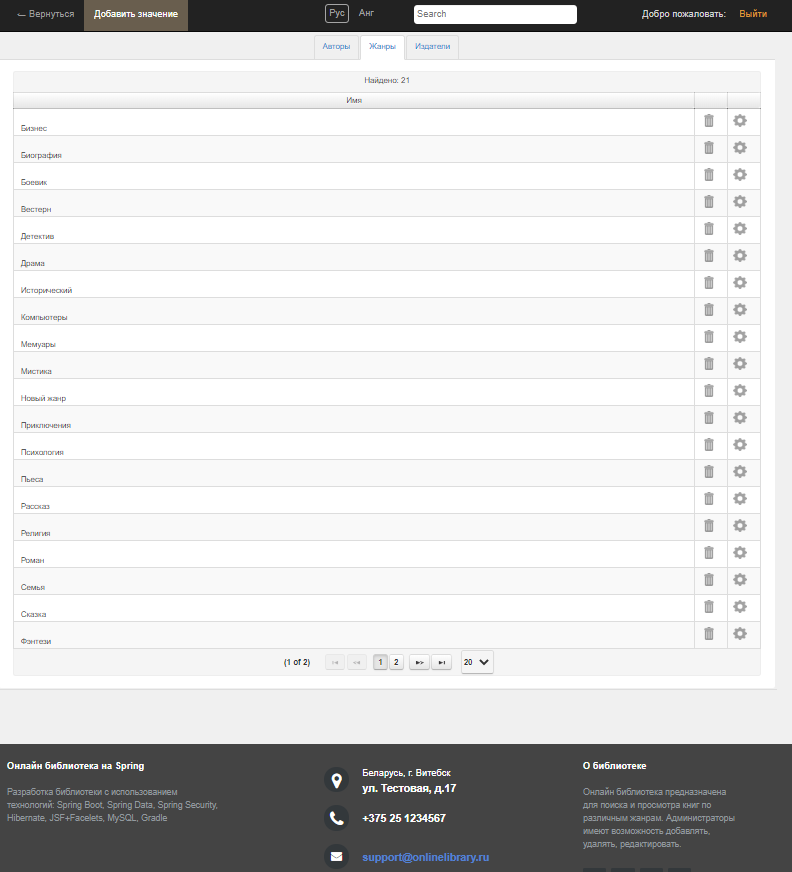
**Рисунок 2.6 – Диалоговое окно редактирования книги**



**Рисунок 2.7 – Диалоговое окно добавления книги**

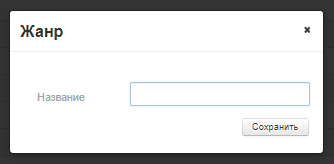
Администратор также имеет возможность добавлять авторов, жанры и издательства, которые в последствии отображаются в ниспадающих списках при добавлении и редактировании книг.

Для добавления авторов, жанров и издательств администратору необходимо перейти в раздел «Справочники», нажав на одноименную кнопку, расположенную в верхнем меню главной страницы. Внешний вид «Справочников» представлен на рисунке 2.8.



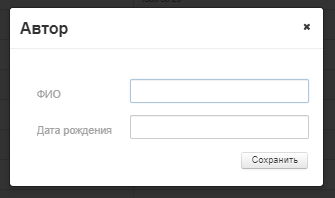
**Рисунок 2.8 – Страница «Справочники»**

Авторы, жанры и издательства расположены на отдельных вкладках. Для добавления жанра появляется соответствующее диалоговое окно, представленное на рисунке 2.9.



**Рисунок 2.9 – Диалоговое окно добавления жанра**

При добавлении издательства, которое осуществляется при нажатии на кнопку «Добавить значение» находясь на вкладке «Издатели», появляется аналогично окно. Диалоговое окно добавления автора представлено на рисунке 2.10.



**Рисунок 2.10 – Диалоговое окно добавления автора**

При удалении книг, авторов, жанров или издательств – появляется соответствующее окно подтверждения действия.

Таким образом, были описаны основные страницы и окна информационной системы управления аварийно-профилактической группой, доступные как обычным пользователям, так и администратору. Достоинствами проекта, является простота администрирования: добавления, удаления и редактирования книг, авторов, жанров и издательств; и приятный дизайн.

# 2.4 Тестирование информационной системы управления аварийно-профилактической группой

Тестирование программного обеспечения – процесс исследования, испытания программного продукта, имеющий две различные цели:

* продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
* выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации [12].

Комплексное решение задач обеспечения качества программных средств предполагает разработку и внедрение той или иной системы управления качеством. В мировой практике наибольшее распространение получила система, основанная на международных стандартах серии ISO 9000, включающей, в том числе стандарт, регламентирующий обеспечение качества ПО (ISO 9000/3). Эти стандарты должны служить руководством для ведущих специалистов компаний, разрабатывающих ПО на заказ.

Определения характеристик и субхарактеристик качества (ISO 9126-1) [13]:

**Функциональные** возможности – способность программного средства обеспечивать решение задач, удовлетворяющих сформулированные потребности заказчиков и пользователей при применении комплекса программ в заданных условиях.

Задачей, было создать информационную систему управления аварийно-профилактической группой соответствующий следующим техническим требованиям:

* наличие системы авторизации и разграничения прав доступа к содержимому проекта;
* гибкость и удобство использования;
* возможность добавлять, редактировать, удалять книги и другую информацию (для администратора) и искать, читать, скачивать, распечатывать (для пользователей) необходимые книги и справочные материалы.

Разработанная информационная система управления аварийно-профилактической группой соответствует всем этим требованиям: она представляет собой веб-проект, оснащенный системами авторизации, а также возможностью добавлять, редактировать, удалять книги и другую информацию (для администратора) и искать, читать, скачивать, распечатывать (для пользователей) необходимые книги и справочные материалы. Проект простой в использовании и интуитивно понятен как для пользователя, так и для администратора.

**Правильность** (корректность) – способность программного средства обеспечивать правильные или приемлемые для пользователя результаты и внешние эффекты.

Правильность функционирования информационной системы управления аварийно-профилактической группой проверялась как в процессе её разработки (промежуточное тестирование корректности функционирования), так и на заключительном этапе на локальной машине. По ходу тестирования в код вводились некоторые правки.

**Защищенность** – способность компонентов программного средства защищать программы и информацию от любых негативных воздействий.

Благодаря функционированию системы разграничения доступа, у пользователя нет возможности каким-либо образом менять содержание библиотеки, препятствуя правильному его функционированию.

**Практичность** (применимость) – свойства программного средства, обусловливающие сложность его понимания, изучения и использования, а также привлекательность для квалифицированных пользователей при применении в указанных условиях.

Разработанная информационная система управления аварийно-профилактической группой достаточно проста в освоении, интуитивно понятна как в пользовательской, так и в административной части.

**Сопровождаемость** – приспособленность программного средства к модификации и изменению конфигурации и функций.

Благодаря использованию MVC модели наш программный продукт является масштабируемым, можно легко добавить новые функции или модифицировать старые, не меняя логику работы приложения. Возможно также повторное использование кода.

**Выводы:**

Таким образом, были успешно пройдены все этапы разработки информационной системы управления аварийно-профилактической группой :

* спроектирована структура базы данных и реализована на практике;
* разработаны и реализованы основные классы в рамках MVC модели;
* описаны особенности интерфейса информационной системы управления аварийно-профилактической группой и представлены основные типы страниц и диалоговых окон;
* проведен анализ функциональности образовательно-информационного ресурса на основе его соответствия характеристикам и субхарактеристикам качества (ISO 9126-1).

# Заключение

В процессе разработки дипломного проекта были пройдены все основные этапы информационной системы управления аварийно-профилактической группой: описание предметной области, проектирование программного комплекса, программная реализация и тестирование программного обеспечения. Результатом стало создание управления аварийно-профилактической группой, оснащенной системой авторизации, а также возможностью добавлять, редактировать, удалять книги и другую информацию (для администратора) и искать, читать, скачивать, распечатывать (для пользователей) необходимые книги и справочные материалы. В данной дипломной работе решены следующие задачи:

* изучены основные характеристики электронных библиотек;
* обозначена предметная область разрабатываемой системы управления аварийно-профилактической группой, определены функции, которые она выполняет и информационная составляющая;
* выбраны программные средства для реализации.

В результате проведенных работ на базе выбранных технологий была разработана и реализована информационная система управления аварийно-профилактической группой. Проект имеет интуитивный интерфейс и простую систему навигации и администрирования.

Таким образом, были выполнены и успешно реализованы на практике все поставленные цели и задачи.

# Список использованных источников

1. Визуальное программирование [Электронный ресурс] . – Режим доступа : https://ru.wikipedia.org/wiki/%D0%92B5. – Дата доступа: 15.01.2020.

2. ISO/IEC/IEEE 24765:2010(en) Systems and software engineering – Vocabulary [Электронный ресурс]. – Режим доступа: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-1:v1:en. – Дата доступа: 15.01.2020.

3. ISO/IEC 2382-1:1993, Information technology — Vocabulary — Part 1: Fundamental terms.01.01.38. [Электронный ресурс]. – Режим доступа: https://www.iso.org/standard/7229.html. – Дата доступа: 15.01.2020.

4. Общий обзор спецификации Java Enterprise Edition и сервера GlassFish [Электронный ресурс]. – Режим доступа: http://foresight-pro.ru/index.php/informatsiya-i-publikatsii/71-glassfish. – Дата доступа: 15.01.2020.

5. Электронная библиотека [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0)Электронная\_библиотека. - Дата доступа: 02.07.2020.

6. Электронные библиотеки как факт современного информационного ландшафта [Электронный ресурс]. – Режим доступа: <http://www.aselibrary.ru/press_center/digital_resources6970/digital_resources69702373/>. - Дата доступа: 15.01.2020.

7. Википедия. Java [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Java>.– Дата доступа: 15.01.2020.

8. Википедия. UML [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/UML. – Дата доступа: 18.07.2019.

9. Википедия. CSS [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/CSS. – Дата доступа: 01.07.2019.

10. Википедия. XHTML [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/XHTML>. – Дата доступа: 01.07.2019.

11. Java – Обзор языка программирования: введение, история и преимущества [Электронный ресурс]. – Режим доступа <http://proglang.su/java/introduction-to-programming>.– Дата доступа: 15.01.2020.

12. IntelliJ IDEA [Электронный ресурс] – Режим доступа: <https://www.jetbrains.com/ru-ru/idea/features/#choose-your-edition>. Дата доступа: 15.01.2020.

13. Википедия. Model-view-controller [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Model-view-controller. – Дата доступа: 18.07.2019.

14. Википедия. Тестирование программного обеспечения [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/ Тестирова-ние\_программного\_обеспечения. – Дата доступа: 10.08.2019.13. Spring Framework [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Spring_Framework>. Дата доступа: 15.01.2020.

15. Руководство по Spring. Введение [Электронный ресурс] – Режим доступа: https://proselyte.net/tutorials/spring-tutorial-full-version/introduction. Дата доступа: 15.01.2020.

16. Учебно-методический комплекс дисциплины «Технологии баз данных и знаний». Этапы проектирования базы данных и их процедуры [Электронный ресурс]. – Режим доступа: http://www.bseu.by/it/tohod/lekcii4\_6.htm. – Дата доступа: 20.07.2019.

17. Липаев, В. Оценка качества программных средств / В. Липаев // Сетевой журнал [Электронный ресурс]. – Ре-жим доступа: http://www.setevoi.ru/cgi-bin/text.pl/magazines/2002/3/52. – Дата доступа: 18.08.2019.

18. Введение в JSF [Электронный ресурс] – Режим доступа:

http://www.websphererus.com/development/introducing-to-jsf. Дата доступа: 15.01.2020.

# ПРИЛОЖЕНИЕ А

**Листинг создания базы данных «library»**

Структура таблицы `author`

**CREATE** **TABLE** **IF** NOT **EXISTS** `author` (

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

`fio` varchar(300) NOT NULL,

`birthday` date NOT NULL,

**PRIMARY** **KEY** (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=127 ;

Структура таблицы `book`

**CREATE** **TABLE** **IF** NOT **EXISTS** `book` (

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

`name` varchar(45) NOT NULL,

`content` longblob,

`page\_count` int NOT NULL,

`isbn` varchar(100) NOT NULL,

`genre\_id` bigint NOT NULL,

`author\_id` bigint NOT NULL,

`publish\_year` int NOT NULL,

`publisher\_id` bigint NOT NULL,

`image` longblob,

`avg\_rating` int DEFAULT `0`,

`total\_vote\_count` bigint DEFAULT `0`,

`total\_rating` bigint DEFAULT `0`,

`view\_count` bigint DEFAULT `0`,

**PRIMARY** **KEY** (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=127 ;

Структура таблицы `genre`

**CREATE** **TABLE** **IF** NOT **EXISTS** `genre` (

`id` bigint NOT NULL AUTO\_INCREMENT,

`name` varchar(100) NOT NULL,

**PRIMARY** **KEY** (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=32 ;

Структура таблицы `publisher`

**CREATE** **TABLE** **IF** NOT **EXISTS** `publisher` (

`id` bigint NOT NULL AUTO\_INCREMENT,

`name` varchar(100) NOT NULL,

**PRIMARY** **KEY** (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=127 ;

Структура таблицы `vote`

**CREATE** **TABLE** **IF** NOT **EXISTS** `vote` (

`id` bigint NOT NULL AUTO\_INCREMENT,

`value` int DEFAULT `0`,

`book\_id` bigint NOT NULL,

`username` varchar (100) NOT NULL,

**PRIMARY** **KEY** (`id`),

UNIQUE KEY `id\_UNIQUE` (`id`),

KEY `fk\_book\_id\_idx` (`book\_id`),

KEY `fk\_user\_id\_idx` (`username`),

CONSTRAINT `fk\_book\_id` FOREIGN KEY (`book\_id`) REFERENCES `book` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=391 ;

Структура таблицы `user`

**CREATE** **TABLE** **IF** NOT **EXISTS** `user` (

`id` bigint NOT NULL AUTO\_INCREMENT,

`username` varchar (50) DEFAULT NULL,

`password` varchar (200) DEFAULT NULL,

`enabled` tinyint (1) DEFAULT NULL,

`password\_sha256` varchar (100) DEFAULT NULL,

**PRIMARY** **KEY** (`id`),

UNIQUE KEY `uniq\_user` (`username`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=3;

Структура таблицы `user\_roles`

**CREATE** **TABLE** **IF** NOT **EXISTS** `user\_roles` (

`id` bigint NOT NULL AUTO\_INCREMENT,

`username` varchar (50) DEFAULT NULL,

`role` varchar (50) DEFAULT NULL,

**PRIMARY** **KEY** (`id`),

UNIQUE KEY `uniq\_user` (`username`, `role`)

CONSTRAINT `fk\_username` FOREIGN KEY (`username`) REFERENCES `user` (`username`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO\_INCREMENT=5;

**ПРИЛОЖЕНИЕ Б**

**Алгоритм начала работы**

Начало

Подключение и инициализация конфигурационных настроек, поподание в Reditect контроллер

Соединение с БД MySQL

Пользователь является администратором

Инициализация структуры

Полный список элементов

Список элементов доступный пользователю

Подключается файл меню

Показывается страница авторизации

Действие (“action”) пользователя отражает контроллер

Подключается контроллер

2

1

1

2

3

4

5

**ДА**

6

**НЕТ**

**ДА**

**НЕТ**

7

8

9

10

11

Подключается файл отображения основной информации страницы

Подключается файл footer

1

Конец

Подключается файл левого меню

Подключается файл топовых книг

12

13

14

15

16

# ПРИЛОЖЕНИЕ В

**Листинг информационной системы управления аварийно-профилактической группой**

**Author.java**

package ru.javabegin.training.library.springlibrary\_2.domain;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.SelectBeforeUpdate;

import javax.persistence.\*;

import java.util.Date;

import java.util.List;

//JPA

@Entity //класс является сущностью; все поля класса будут автоматически связаны со столбцами таблицы

@Table (catalog = "library")

@EqualsAndHashCode(of = "id") //создает хэш-код по id и сравнивает по id тоже благодаря lombok библиотеки

@Getter @Setter //благодаря библиотеки lombok мы можем сократить код н: просто указать аннотации на геттер-сеттер, а не прописывать их

@DynamicUpdate //update и insert нужны для того, чтобы когда мы будем вставлять новые объекты в таблицу (или обновлять), онбудет использ только те поля, которые изменились

@DynamicInsert

@SelectBeforeUpdate //чтобы hibernate мог проверить перед update-ом, а нужно ли объект обновлять (если мы вызываем update)

public class Author {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) //автоинкремент поля

private Long id;

private String fio;

private Date birthday;

@JsonIgnore //для избежания рекурсии (зацикливания) (т.к.у одного автора может быть несколько книг) при получении json. В json , например, нам не нужны книги автора, только сам автор.

@Basic(fetch =FetchType.LAZY)

@OneToMany(mappedBy = "author")

private List<Book> books;

@Override

public String toString(){

return fio;

}

}

**Book.java**

package ru.javabegin.training.library.springlibrary\_2.domain;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.SelectBeforeUpdate;

import javax.persistence.\*;

@Entity

@Table(catalog = "library")

@DynamicUpdate

@DynamicInsert

@SelectBeforeUpdate

@Setter @Getter

@EqualsAndHashCode(of = "id")

public class Book {

public Book(){

}

public Book(Long id, String name, Integer pageCount, String isbn, Genre genre, Author author, Publisher publisher, Integer publishYear,

byte[] image, String descr, long viewCount, long totalRating, long totalVoteCount, int avgRating) {

this.id =id;

this.name = name;

this.pageCount = pageCount;

this.isbn = isbn;

this.genre = genre;

this.author = author;

this.publisher = publisher;

this.publishYear = publishYear;

this.image = image;

this.descr = descr;

this.viewCount = viewCount;

this.totalRating = totalRating;

this.totalVoteCount = totalVoteCount;

this.avgRating = avgRating;

}

public Book(Long id, byte[] image) {

this.id= id;

this.image = image;

}

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@JsonIgnore // не добавлять значение в Json

@Lob //указание на большие объекты

@Column(updatable = false)

private byte[] content; //byte - двоичное значение, которое потом буем конвертировать в нужный формат

@Column(name = "page\_count")

private Integer pageCount;

private String isbn;

@ManyToOne

@JoinColumn

private Genre genre;

@ManyToOne //двусторонняя связь книги-автор

@JoinColumn //будем получать готовый объект, а не их id (т.е. н: не id 20, а Борис Акунин

private Author author;

@ManyToOne

@JoinColumn

private Publisher publisher;

@Column(name = "publish\_year")

private Integer publishYear;

private byte[] image;

private String descr;

@Column(name = "view\_count")

private long viewCount;

@Column(name = "total\_rating")

private long totalRating;

@Column(name = "total\_vote\_count")

private long totalVoteCount;

@Column(name = "avg\_rating")

private int avgRating;

@Override

public String toString(){

return name;

}

**Genre.java**

package ru.javabegin.training.library.springlibrary\_2.domain;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.SelectBeforeUpdate;

import javax.persistence.\*;

import java.util.List;

@EqualsAndHashCode(of = "id")

@Table(catalog = "library")

@DynamicUpdate

@DynamicInsert

@SelectBeforeUpdate

@Entity

@Getter @Setter

public class Genre {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@JsonIgnore

@Basic(fetch = FetchType.LAZY)

@OneToMany(mappedBy = "genre")

private List<Book> books;

@Override

public String toString(){

return name;

}

}

**Publisher.java**

package ru.javabegin.training.library.springlibrary\_2.domain;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.SelectBeforeUpdate;

import javax.persistence.\*;

import java.util.List;

@EqualsAndHashCode(of = "id")

@Table(catalog = "library")

@DynamicUpdate

@DynamicInsert

@SelectBeforeUpdate

@Entity

@Getter @Setter

public class Publisher {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@JsonIgnore

@Basic(fetch = FetchType.LAZY)

@OneToMany(mappedBy = "publisher")

private List<Book> books;

@Override

public String toString(){

return name;

}

**Vote.java**

package ru.javabegin.training.library.springlibrary\_2.domain;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.DynamicInsert;

import org.hibernate.annotations.DynamicUpdate;

import org.hibernate.annotations.SelectBeforeUpdate;

import javax.persistence.\*;

@Entity

@EqualsAndHashCode(of = "id")

@Table(catalog = "library")

@Getter

@Setter

@DynamicUpdate

@DynamicInsert

@SelectBeforeUpdate

public class Vote {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String value;

@Column(name = "book\_id")

// private Date bookId;

private long bookId;

private String username;

}

**GeneralDAO**

package ru.javabegin.training.library.springlibrary\_2.dao;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import java.util.List;

public interface GeneralDAO<T> {

List<T> getAll(); //получение всех записей (без постраничности)

T get(long id); // получение объекта по id

//Optional<T> get(long id);

T save(T obj); // обновляет и добавляет объект (один метод на 2 действия)

void delete(T object); // удаление объекта

List<T> search(String... searchString); //поиск записей с любым количеством параметров

List<T> getAll(Sort sort); //получение всех данных с сортировкой результата

Page<T> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection); //получение всех записей с постраничностью

Page<T> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String...searchString); //поиск записей с постраничностью

}

**AuthorDAO.java**

package ru.javabegin.training.library.springlibrary\_2.dao;

import ru.javabegin.training.library.springlibrary\_2.domain.Author;

public interface AuthorDAO extends GeneralDAO<Author> {

}

**BookDAO.java**

package ru.javabegin.training.library.springlibrary\_2.dao;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import ru.javabegin.training.library.springlibrary\_2.domain.Book;

import java.util.List;

public interface BookDAO extends GeneralDAO<Book> {

List<Book> findTopBooks(int limit); //поиск топовых книг

byte[] getContent(long id); // получение контента по id

Page<Book> findByGenre(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, long genreId); //постраничный вывод книг определенного жанра

// обновление статистики просмотра книги

void updateViewCount(long viewCount, long id);

// обновить данные рейтинга

void updateRating(long totalRating, long totalVoteCount, int avgRating, long id);

}

**GenreDAO**

package ru.javabegin.training.library.springlibrary\_2.dao;

import ru.javabegin.training.library.springlibrary\_2.domain.Genre;

public interface GenreDAO extends GeneralDAO<Genre> {

}

**PublisherDAO**

package ru.javabegin.training.library.springlibrary\_2.dao;

import ru.javabegin.training.library.springlibrary\_2.domain.Publisher;

public interface PublisherDAO extends GeneralDAO<Publisher>{

}

**AuthorService.java**

package ru.javabegin.training.library.springlibrary\_2.dao.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import ru.javabegin.training.library.springlibrary\_2.dao.\*;

import ru.javabegin.training.library.springlibrary\_2.domain.Author;

import ru.javabegin.training.library.springlibrary\_2.spring.repository.AuthorRepository;

import java.util.List;

import java.util.Optional;

// сервисный уровень для работы с авторами

// API для реализованных бизнес процессов

// код должен обращаться только через Service (не к Repository напрямую)

@Service // сервисный Spring бин

@Transactional

// методы помечаются как транзакционные (для запросов применяются настройки транзакций по-умолчанию, уровень изоляции и пр.)

public class AuthorService implements AuthorDAO {

@Autowired

private AuthorRepository authorRepository;

@Override

public List<Author> getAll() {

return authorRepository.findAll();

}

public List<Author> getAll(Sort sort) {

return authorRepository.findAll(sort);

}

@Override

public Page<Author> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

return authorRepository.findAll( PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public List<Author> search(String... searchString) {

return authorRepository.findByFioContainingIgnoreCaseOrderByFio(searchString[0]);

}

@Override

public Page<Author> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String... searchString) {

return authorRepository.findByFioContainingIgnoreCaseOrderByFio(searchString[0], PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public Author save(Author author) {

return authorRepository.save(author);

}

@Override

public void delete(Author author) {

authorRepository.delete(author);

}

@Override

public Author get(long id) {

Optional<Author> bookmark = authorRepository.findById(id); // Optional - обертка, в котором может быть значение или пусто (используется для исключение ошибки NullPointerException

if (bookmark.isPresent()) { // если значение представлено - вернуть его

return bookmark.get();

} else {

return null;

}

}

**BookService.java**

package ru.javabegin.training.library.springlibrary\_2.dao.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import ru.javabegin.training.library.springlibrary\_2.dao.BookDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Book;

import ru.javabegin.training.library.springlibrary\_2.spring.repository.BookRepository;

import java.util.List;

import java.util.Optional;

@Service

@Transactional

public class BookService implements BookDAO {

@Autowired

private BookRepository bookRepository;

@Override

public List<Book> getAll() {

return bookRepository.findAll();

}

@Override

public List<Book> getAll(Sort sort) {

return bookRepository.findAll(sort);

}

@Override

public Page<Book> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

// return bookRepository.findAllWithoutContent(new PageRequest(pageNumber, pageSize, new Sort(sortDirection, sortField)));

return bookRepository.findAllWithoutContent(PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public List<Book> search(String... searchString) {

return null;

}

@Override

public Page<Book> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String... searchString) {

// чтобы название метода не было слишком длинным - можно использовать @Query c HQL (если больше 2-х переменных)

return bookRepository.findByNameContainingIgnoreCaseOrAuthorFioContainingIgnoreCaseOrderByName(searchString[0], searchString[0], PageRequest.of (pageNumber, pageSize, Sort.by (sortDirection, sortField)));

}

@Override

public Book save(Book book) {

// отдельно сохраняем данные книги

bookRepository.save(book);

if (book.getContent()!=null) {

// отдельно сохраняем контент

bookRepository.updateContent(book.getContent(), book.getId());

}

return book;

}

@Override

public void delete(Book book) {

bookRepository.delete(book);

}

@Override

public Book get(long id) {

Optional<Book> bookmark = bookRepository.findById(id); // Optional - обертка, в котором может быть значение или пусто (используется для исключение ошибки NullPointerException

if (bookmark.isPresent()) { // если значение представлено - вернуть его

return bookmark.get();

} else {

return null;

}

}

@Override

public byte[] getContent(long id) {

return bookRepository.getContent(id);

}

public List<Book> findTopBooks(int limit) {

// return bookRepository.findTopBooks(new PageRequest(0,limit, new Sort(Sort.Direction.DESC, "viewCount")));

return bookRepository.findTopBooks( PageRequest.of(0,limit, Sort.by(Sort.Direction.DESC, "viewCount")));

}

@Override

public Page<Book> findByGenre(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, long genreId) {

return bookRepository.findByGenre(genreId, PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public void updateViewCount(long viewCount, long id) {

bookRepository.updateViewCount(viewCount, id);

}

@Override

public void updateRating(long totalRating, long totalVoteCount, int avgRating, long id) {

bookRepository.updateRating(totalRating, totalVoteCount, avgRating, id);

}

**GenreService.java**

package ru.javabegin.training.library.springlibrary\_2.dao.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import ru.javabegin.training.library.springlibrary\_2.dao.GenreDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Genre;

import ru.javabegin.training.library.springlibrary\_2.spring.repository.GenreRepository;

import java.util.List;

import java.util.Optional;

@Service

@Transactional

public class GenreService implements GenreDAO {

@Autowired

private GenreRepository genreRepository;

@Override

public List<Genre> getAll() {

return genreRepository.findAll();

}

public List<Genre> getAll(Sort sort) {

return genreRepository.findAll(sort);

}

@Override

public Page<Genre> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

return genreRepository.findAll(PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public List<Genre> search(String... searchString) {

return genreRepository.findByNameContainingIgnoreCaseOrderByName(searchString[0]);

}

@Override

public Page<Genre> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String... searchString) {

return genreRepository.findByNameContainingIgnoreCaseOrderByName(searchString[0], PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public Genre save(Genre genre) {

return genreRepository.save(genre);

}

@Override

public void delete(Genre genre) {

genreRepository.delete(genre);

}

@Override

public Genre get(long id) {

Optional<Genre> bookmark = genreRepository.findById(id); // Optional - обертка, в котором может быть значение или пусто (используется для исключение ошибки NullPointerException

if (bookmark.isPresent()) { // если значение представлено - вернуть его

return bookmark.get();

} else {

return null;

}

}

**PublisherService.java**

package ru.javabegin.training.library.springlibrary\_2.dao.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import ru.javabegin.training.library.springlibrary\_2.dao.PublisherDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Publisher;

import ru.javabegin.training.library.springlibrary\_2.spring.repository.PublisherRepository;

import java.util.List;

import java.util.Optional;

@Service

@Transactional

public class PublisherService implements PublisherDAO {

@Autowired

private PublisherRepository publisherRepository;

@Override

public List<Publisher> getAll() {

return publisherRepository.findAll();

}

public List<Publisher> getAll(Sort sort) {

return publisherRepository.findAll(sort);

}

@Override

public Page<Publisher> getAll(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

return publisherRepository.findAll(PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public List<Publisher> search(String... searchString) {

return publisherRepository.findByNameContainingIgnoreCaseOrderByName(searchString[0]);

}

@Override

public Page<Publisher> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection, String... searchString) {

return publisherRepository.findByNameContainingIgnoreCaseOrderByName(searchString[0], PageRequest.of(pageNumber, pageSize, Sort.by(sortDirection, sortField)));

}

@Override

public Publisher save(Publisher publisher) {

return publisherRepository.save(publisher);

}

@Override

public void delete(Publisher publisher){

publisherRepository.delete(publisher);

}

@Override

public Publisher get(long id) {

Optional<Publisher> bookmark = publisherRepository.findById(id); // Optional - обертка, в котором может быть значение или пусто (используется для исключение ошибки NullPointerException

if (bookmark.isPresent()) { // если значение представлено - вернуть его

return bookmark.get();

} else {

return null;

}

}

**AbsractController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.controller;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import java.io.Serializable;

// описывает необходимое поведение для всех JSF контроллеров

public abstract class AbstractController<T> implements Serializable {

// постранично выводит книги

public abstract Page<T> search(int first, int count, String sortField, Sort.Direction sortDirection);

public abstract void addAction();

public abstract void editAction();

public abstract void deleteAction();

}

**AuthorController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.controller;

import com.google.common.base.Strings;

import lombok.Getter;

import lombok.Setter;

import org.primefaces.context.RequestContext;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Component;

import ru.javabegin.training.library.springlibrary\_2.dao.AuthorDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Author;

import ru.javabegin.training.library.springlibrary\_2.jsfui.model.LazyDataTable;

import javax.annotation.PostConstruct;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

import java.util.List;

@ManagedBean

@SessionScoped

@Component

@Getter

@Setter

public class AuthorController extends AbstractController<Author> {

// из JSF таблицы обязательно должна быть ссылки на переменные, иначе при использовании постраничности dataTable работает некорректно

// также - выбранное пользователем значение (кол-во записей на странице) будет сохраняться

private int rowsCount = 20;

private int first;

@Autowired

private AuthorDAO authorDao;

@Autowired

private SprController sprController;

private Author selectedAuthor;

private LazyDataTable<Author> lazyModel;

private Page<Author> authorPages;

@PostConstruct

public void init() {

lazyModel = new LazyDataTable(this);

}

public void save() {

authorDao.save(selectedAuthor);

RequestContext.getCurrentInstance().execute("PF('dialogAuthor').hide()");

}

@Override

public Page<Author> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

if (sortField == null) {

sortField = "fio";

}

// для удобной проверки строк - используем библиотеку Google Guava и метод isNullOrEmpty

if (Strings.isNullOrEmpty(sprController.getSearchText())) {

authorPages = authorDao.getAll(pageNumber, pageSize, sortField, sortDirection);

} else {

authorPages = authorDao.search(pageNumber, pageSize, sortField, sortDirection, sprController.getSearchText());

}

return authorPages;

}

@Override

public void addAction() {

selectedAuthor = new Author();

showEditDialog();

}

@Override

public void editAction() {

// выбранный author уже будет записан в переменную selectedAuthor (как только пользователь кликнет на редактирование)

// он отобразится в диалоговом окне

showEditDialog();

}

@Override

public void deleteAction() {

// выбранный author уже будет записан в переменную selectedAuthor (как только пользователь кликнет на удаление)

authorDao.delete(selectedAuthor);

}

private void showEditDialog() {

// показывает диалоговое окно со значениями selectedAuthor

RequestContext.getCurrentInstance().execute("PF('dialogAuthor').show()");

}

public List<Author> find(String fio) {

return authorDao.search(fio);

}

}

**BookController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.controller;

import lombok.Getter;

import lombok.Setter;

import lombok.extern.java.Log;

import org.apache.commons.io.IOUtils;

import org.primefaces.context.RequestContext;

import org.primefaces.event.CloseEvent;

import org.primefaces.event.FileUploadEvent;

import org.primefaces.event.RateEvent;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Component;

import ru.javabegin.training.library.springlibrary\_2.dao.BookDAO;

import ru.javabegin.training.library.springlibrary\_2.dao.GenreDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Book;

import ru.javabegin.training.library.springlibrary\_2.jsfui.enums.SearchType;

import ru.javabegin.training.library.springlibrary\_2.jsfui.model.LazyDataTable;

import javax.annotation.PostConstruct;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

import javax.faces.context.FacesContext;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;

import java.util.Map;

import java.util.ResourceBundle;

@ManagedBean

@SessionScoped

@Component

@Getter @Setter

@Log

public class BookController extends AbstractController<Book> {

public static final int DEFAULT\_PAGE\_SIZE = 20;// по-умолчанию сколько книг отображать на странице

public static final int TOP\_BOOKS\_LIMIT = 5;// сколько показывать популярных книг

// из JSF таблицы обязательно должна быть ссылки на переменные, иначе при использовании постраничности dataGrid работает некорректно (не отрабатывает bean)

// также - выбранное пользователем значение (кол-во записей на странице) будет сохраняться

private int rowsCount = DEFAULT\_PAGE\_SIZE;

private SearchType searchType; // запоминает последний выбранный вариант поиска

@Autowired

private BookDAO bookDao; // будет автоматически подставлен BookService, т.к. Spring контейнер ищет бин по типу

@Autowired

private GenreDAO genreDao;

@Autowired

private GenreController genreController;

private Book selectedBook; // ссылка на текущую книгу (которую редактируют, хотят удалять и пр.) - т.е. над какой книгой в данный момент производим действие

private LazyDataTable<Book> lazyModel; // класс-утилита, которая помогает выводить данные постранично (работает в паре с компонентами на странице JSF)

private byte[] uploadedImage; // сюда будет сохраняться загруженная пользователем новая обложка (при редактировании или при добавлении книги)

private byte[] uploadedContent; // сюда будет сохраняться загруженный пользователем PDF контент (при редактировании или при добавлении книги)

private Page<Book> bookPages; //хранит список найденных книг

private List<Book> topBooks;// хранит полученные ТОП книги (может использоваться наприемр для получения изображений книги)

private String searchText; // введенный текст для поиска

private long selectedGenreId; // выбранынй жанр для поиска

@PostConstruct //после того, как создан ManagedBean, будет автоматически вызываться метод init

public void init() {

lazyModel = new LazyDataTable(this);

} //this это ссылка на текущий контроллер

public void save() {

// если было выбрано новое изображение

if (uploadedImage != null) {

selectedBook.setImage(uploadedImage);

}

// если был выбран новый PDF контент

if (uploadedContent != null) {

selectedBook.setContent(uploadedContent);

}

bookDao.save(selectedBook);

RequestContext.getCurrentInstance().execute("PF('dialogEditBook').hide()");// вызов JS из java кода

}

// метод автоматически вызывается из LazyDataTable

@Override

public Page<Book> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

if (sortField == null) { //если не указано поле сортировки

sortField = "name"; // берем поле "name"

}

if (searchType == null){

bookPages = bookDao.getAll(pageNumber, pageSize, sortField, sortDirection); //отображаем все книги постранично, если не указан тип поиска

}else {

switch (searchType) {

case SEARCH\_GENRE:

bookPages = bookDao.findByGenre(pageNumber, pageSize, sortField, sortDirection, selectedGenreId);

break;

case SEARCH\_TEXT:

bookPages = bookDao.search(pageNumber, pageSize, sortField, sortDirection, searchText);

break;

case ALL:

bookPages = bookDao.getAll(pageNumber, pageSize, sortField, sortDirection);

break;

}

}

return bookPages;

}

@Override

public void addAction() {

selectedBook = new Book();

uploadedImage = loadDefaultIcon();

uploadedContent = null;

RequestContext.getCurrentInstance().execute("PF('dialogEditBook').show()");

}

// загрузить картинку по-умолчанию для обложки

private byte[] loadDefaultIcon(){

InputStream stream = FacesContext.getCurrentInstance().getExternalContext().getResourceAsStream("/resources/images/no-cover.jpg");

try {

return IOUtils.toByteArray(stream);

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

// при закрытии диалогового окна - очищать загруженный контент из переменной

public void onCloseDialog(CloseEvent event) {

uploadedContent = null;

}

@Override

public void editAction() {

uploadedImage = selectedBook.getImage();

// выбранный book уже будет записан в переменную selectedBook (как только пользователь кликнет на редактирование)

// книга отобразится в диалоговом окне

RequestContext.getCurrentInstance().execute("PF('dialogEditBook').show()");

}

@Override

public void deleteAction() {

bookDao.delete(selectedBook);

}

// сообщение, сколько данных найдено и по какому критеорию

public String getSearchMessage(){

// для доступа к файлам локализации

ResourceBundle bundle = ResourceBundle.getBundle("library", FacesContext.getCurrentInstance().getViewRoot().getLocale());

String message=null;

if (searchType==null){

return null;

}

switch (searchType) {

case SEARCH\_GENRE:

message = bundle.getString("genre")+ ": '"+genreDao.get(selectedGenreId)+"'";

break;

case SEARCH\_TEXT:

if (searchText==null || searchText.trim().length()==0){

return null;

}

message = bundle.getString("search")+ ": '"+searchText+"'";

break;

}

return message;

}

// получить PDF контент книги для чтения

public byte[] getContent(long id) {

byte[] content;

if (uploadedContent != null) {

content = uploadedContent;

} else {

content = bookDao.getContent(id);

}

return content;

}

// при загрузке обложки - она будет сохраняться в переменную uploadedImage

public void uploadImage(FileUploadEvent event) {

if (event.getFile() != null) {

uploadedImage = event.getFile().getContents();

}

}

// при загрузке PDF контента - он будет сохраняться в переменную uploadedContent

public void uploadContent(FileUploadEvent event) {

if (event.getFile() != null) {

uploadedContent = event.getFile().getContents();

}

}

public List<Book> getTopBooks() {

topBooks = bookDao.findTopBooks(TOP\_BOOKS\_LIMIT);

return topBooks;

}

//поиск по определенному жанру

public void showBooksByGenre(long selectedGenreId){

searchType = SearchType.SEARCH\_GENRE;

this.selectedGenreId = selectedGenreId;

}

public void showAll(){

searchType = SearchType.ALL;

}

public void searchAction(){

searchType = SearchType.SEARCH\_TEXT;

}

public Page<Book> getBookPages(){

return bookPages;

}

public void updateViewCount(long viewCount, long id){

bookDao.updateViewCount(viewCount+1, id);

}

// вызывается при голосовании за книгу

public void onrate(RateEvent rateEvent) {

Map<String, String> params = FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap();

int bookIndex = Integer.parseInt(params.get("bookIndex"));// параметр индекса книги, который передается со страницы

Book book = bookPages.getContent().get(bookIndex);// по индексу получаем книгу, для которой проголосовали

// какой рейтинг поставил пользователь

long currentRating = Long.valueOf(rateEvent.getRating().toString()).longValue();

// новый рейтинг (суммарный)

long newRating = book.getTotalRating() + currentRating;

// сколько проголосовало

long newVoteCount = book.getTotalVoteCount()+1;

// среднее значение, которое показывается на странице

int newAvgRating = calcAverageRating(newRating, newVoteCount);

bookDao.updateRating(newRating, newVoteCount, newAvgRating, book.getId());

}

public int calcAverageRating(long totalRating, long totalVoteCount) {

if (totalRating == 0 || totalVoteCount == 0) {

return 0;

}

int avgRating = Long.valueOf(totalRating / totalVoteCount).intValue();

return avgRating;

}

**GenreController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.controller;

import com.google.common.base.Strings;

import lombok.Getter;

import lombok.Setter;

import org.primefaces.context.RequestContext;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Component;

import ru.javabegin.training.library.springlibrary\_2.dao.GenreDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Genre;

import ru.javabegin.training.library.springlibrary\_2.jsfui.model.LazyDataTable;

import javax.annotation.PostConstruct;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

import java.util.List;

@ManagedBean

@SessionScoped

@Component

@Getter

@Setter

public class GenreController extends AbstractController<Genre> {

// из JSF таблицы обязательно должна быть ссылки на переменные, иначе при использовании постраничности dataTable работает некорректно

// также - выбранное пользователем значение (кол-во записей на странице) будет сохраняться

private int rowsCount = 20;

private int first;

@Autowired

private GenreDAO genreDao;

@Autowired

private SprController sprController;

private Genre selectedGenre; // над каким жанром в данный момент производим действие (удаление, редактирование)

private LazyDataTable<Genre> lazyModel; // справочные значения также выводятся постранично, как и книги (создается новый экземпляр lazyModel)

private Page<Genre> genrePages; // найденные жанры

@PostConstruct

public void init() {

lazyModel = new LazyDataTable(this);

}

public List<Genre> find(String name) {

return genreDao.search(name);

}

public void save() {

genreDao.save(selectedGenre);

RequestContext.getCurrentInstance().execute("PF('dialogGenre').hide()");

}

@Override

public Page<Genre> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

if (sortField == null) {

sortField = "name";

}

// для удобной проверки строк - используем библиотеку Google Guava и метод isNullOrEmpty

if (Strings.isNullOrEmpty(sprController.getSearchText())) {

genrePages = genreDao.getAll(pageNumber, pageSize, sortField, sortDirection);

} else {

genrePages = genreDao.search(pageNumber, pageSize, sortField, sortDirection, sprController.getSearchText());

}

return genrePages;

}

@Override

public void addAction() {

selectedGenre = new Genre();

showEditDialog();

}

@Override

public void editAction() {

// выбранный genre уже будет записан в переменную selectedGenre (как только пользователь кликнет на редактирование)

// он отобразится в диалоговом окне

showEditDialog();

}

@Override

public void deleteAction() {

genreDao.delete(selectedGenre);

}

private void showEditDialog() {

// показывает диалоговое окно со значениями selectedGenre

RequestContext.getCurrentInstance().execute("PF('dialogGenre').show()");

}

// вызывается для отображения всех жанров слева на странице

public List<Genre> getAll() {

return genreDao.getAll(Sort.by(Sort.Direction.ASC, "name"));

}

**PublisherController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.controller;

import com.google.common.base.Strings;

import lombok.Getter;

import lombok.Setter;

import org.primefaces.context.RequestContext;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Component;

import ru.javabegin.training.library.springlibrary\_2.dao.PublisherDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Publisher;

import ru.javabegin.training.library.springlibrary\_2.jsfui.model.LazyDataTable;

import javax.annotation.PostConstruct;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

import java.util.List;

@ManagedBean

@SessionScoped

@Component

@Getter

@Setter

public class PublisherController extends AbstractController<Publisher> {

// из JSF таблицы обязательно должна быть ссылки на переменные, иначе при использовании постраничности dataTable работает некорректно

// также - выбранное пользователем значение (кол-во записей на странице) будет сохраняться

private int rowsCount = 20;

private int first;

private Page<Publisher> publisherPages;

@Autowired

private PublisherDAO publisherDao;

@Autowired

private SprController sprController;

private Publisher selectedPublisher;

private LazyDataTable<Publisher> lazyModel;

@PostConstruct

public void init() {

lazyModel = new LazyDataTable(this);

}

public void save() {

publisherDao.save(selectedPublisher);

RequestContext.getCurrentInstance().execute("PF('dialogPublisher').hide()");

}

@Override

public Page<Publisher> search(int pageNumber, int pageSize, String sortField, Sort.Direction sortDirection) {

if (sortField == null) {

sortField = "name";

}

// для удобной проверки строк - используем библиотеку Google Guava и метод isNullOrEmpty

if (Strings.isNullOrEmpty(sprController.getSearchText())) {

publisherPages = publisherDao.getAll(pageNumber, pageSize, sortField, sortDirection);

} else {

publisherPages = publisherDao.search(pageNumber, pageSize, sortField, sortDirection, sprController.getSearchText());

}

return publisherPages;

}

@Override

public void addAction() {

selectedPublisher = new Publisher();

showEditDialog();

}

@Override

public void editAction() {

// выбранный publisher уже будет записан в переменную selectedPublisher (как только пользователь кликнет на редактирование)

// он отобразится в диалоговом окне

showEditDialog();

}

@Override

public void deleteAction() {

// выбранный publisher уже будет записан в переменную selectedPublisher (как только пользователь кликнет на удаление)

publisherDao.delete(selectedPublisher);

}

private void showEditDialog() {

// показывает диалоговое окно со значениями selectedPublisher

RequestContext.getCurrentInstance().execute("PF('dialogPublisher').show()");

}

public List<Publisher> find(String name) {

return publisherDao.search(name);

}

}

**SprController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.controller;

import lombok.Getter;

import lombok.Setter;

import org.primefaces.context.RequestContext;

import org.primefaces.event.TabChangeEvent;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import javax.annotation.ManagedBean;

import javax.faces.bean.RequestScoped;

// общий контроллер для всех справочников

@ManagedBean

@Component

@RequestScoped

@Getter

@Setter

public class SprController {

@Autowired

private AuthorController authorController;

@Autowired

private GenreController genreController;

@Autowired

private PublisherController publisherController;

private String listId = "tabView:authorForm:authorList";// по-умолчанию - открытым табом является таб для редактирования авторов

private String searchText; // текст поиска

private String selectedTab = "tabAuthors";// выбранный пользователем таб, по-умолчанию выбран первый таб с авторами

// вызывается при поиске справочных значений

public void search(){

RequestContext context = RequestContext.getCurrentInstance();

context.update(listId);

}

// возвращает нужный id для обновления области на странице

public void onTabChange(TabChangeEvent event) {

searchText = null;

selectedTab = event.getTab().getId();

switch (selectedTab){

case "tabAuthors":

listId = "tabView:authorForm:authorList";

break;

case "tabGenres":

listId = "tabView:genreForm:genreList";

break;

case "tabPublishers":

listId = "tabView:publisherForm:publisherList";

break;

}

RequestContext.getCurrentInstance().update("searchForm:searchInput");

}

// в зависимости от выбранного таба - будем показываться соответствующее диалоговое окно

public void addAction(){

switch (selectedTab){

case "tabAuthors":

authorController.addAction();

RequestContext.getCurrentInstance().update("tabView:dialogAuthor");

break;

case "tabGenres":

genreController.addAction();

RequestContext.getCurrentInstance().update("tabView:dialogGenre");

break;

case "tabPublishers":

publisherController.addAction();

RequestContext.getCurrentInstance().update("tabView:dialogPublisher");

break;

}

**UserController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.controller;

import com.google.common.base.Strings;

import lombok.Getter;

import lombok.Setter;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.User;

import org.springframework.stereotype.Component;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.ViewScoped;

import javax.faces.context.FacesContext;

import java.util.ResourceBundle;

@ManagedBean

@ViewScoped

@Component

@Getter @Setter

public class UserController {

private String username;

private String password;

public User getCurrentUser(){

return (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

}

// есть ли роль у пользователя

public boolean hasRole(String role){

return getCurrentUser().getAuthorities().stream().filter(x -> x.getAuthority().equals("ROLE\_"+role)).count()>0;

}

// локализованное сообщение о том, что неправильно ввели логин или пароль

public String getLoginFailedMessage(){

Object obj = FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("loginFailed");

if (obj==null) return "";

FacesContext context = FacesContext.getCurrentInstance();

ResourceBundle bundle = context.getApplication().getResourceBundle(context, "msg");

String message = bundle.getString("login\_failed");

FacesContext.getCurrentInstance().getExternalContext().getSessionMap().remove("loginFailed");

if (Strings.isNullOrEmpty(message)){

return "";

}else{

return message;

}

**AuthorConverter.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.converter;

import com.google.common.base.Strings;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import ru.javabegin.training.library.springlibrary\_2.dao.AuthorDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Author;

import javax.faces.component.UIComponent;

import javax.faces.context.FacesContext;

import javax.faces.convert.Converter;

import javax.faces.convert.FacesConverter;

// для преобразования выбранного значения из выпадающего списка в конкретный объект Author

@FacesConverter(forClass = Author.class)

@Component

public class AuthorConverter implements Converter {

@Autowired

private AuthorDAO authorDao;

@Override

public Object getAsObject(FacesContext context, UIComponent component, String value) {

if (Strings.isNullOrEmpty(value)) {

return null;

}

return authorDao.get(Integer.valueOf(value));

}

@Override

public String getAsString(FacesContext context, UIComponent component, Object value) {

if (value == null) {

return null;

}

return ((Author)value).getId().toString();

}

}

**GenreController.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.converter;

import com.google.common.base.Strings;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import ru.javabegin.training.library.springlibrary\_2.dao.GenreDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Genre;

import javax.faces.component.UIComponent;

import javax.faces.context.FacesContext;

import javax.faces.convert.Converter;

import javax.faces.convert.FacesConverter;

// для преобразования выбранного значения из выпадающего списка в конкретный объект Genre

@FacesConverter(forClass = Genre.class)

@Component

public class GenreConverter implements Converter {

@Autowired

private GenreDAO genreDao;

@Override

public Object getAsObject(FacesContext context, UIComponent component, String value) {

if (Strings.isNullOrEmpty(value)) {

return null;

}

return genreDao.get(Integer.valueOf(value));

}

@Override

public String getAsString(FacesContext context, UIComponent component, Object value) {

if (value == null) {

return null;

}

return ((Genre) value).getId().toString();

}

}

**PublisherConverter.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.converter;

import com.google.common.base.Strings;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import ru.javabegin.training.library.springlibrary\_2.dao.PublisherDAO;

import ru.javabegin.training.library.springlibrary\_2.domain.Publisher;

import javax.faces.component.UIComponent;

import javax.faces.context.FacesContext;

import javax.faces.convert.Converter;

import javax.faces.convert.FacesConverter;

// для преобразования выбранного значения из выпадающего списка в конкретный объект Publisher

@FacesConverter(forClass = Publisher.class)

@Component

public class PublisherConverter implements Converter {

@Autowired

private PublisherDAO publisherDao;

@Override

public Object getAsObject(FacesContext context, UIComponent component, String value) {

if (Strings.isNullOrEmpty(value)) {

return null;

}

return publisherDao.get(Integer.valueOf(value));

}

@Override

public String getAsString(FacesContext context, UIComponent component, Object value) {

if (value == null) {

return null;

}

return ((Publisher)value).getId().toString();

}

}

**LocaleChanger.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.locale;

import ru.javabegin.training.library.springlibrary\_2.jsfui.util.CookieHelper;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

import java.io.Serializable;

import java.util.Locale;

@ManagedBean (eager = true)

@SessionScoped

public class LocaleChanger implements Serializable {

// хранится текущая выбранная пользователем локаль на уровне его сессии

private Locale currentLocale = new Locale("ru");

public LocaleChanger() {

// есть ли сохраненная локаль

if (CookieHelper.getCookie(CookieHelper.COOKIE\_LANG)==null){

return;

}

String cookieLang = CookieHelper.getCookie(CookieHelper.COOKIE\_LANG).getValue();

if (cookieLang != null){

currentLocale = new Locale(cookieLang);

}

}

public void changeLocale(String localeCode) {

currentLocale = new Locale(localeCode);

// сохранить в куки браузера выбранный язык

CookieHelper.setCookie(CookieHelper.COOKIE\_LANG, currentLocale.getLanguage(), 3600);

}

public Locale getCurrentLocale() {

return currentLocale;

}

}

**LazyDataTable.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.model;

import lombok.Getter;

import lombok.Setter;

import org.primefaces.model.LazyDataModel;

import org.primefaces.model.SortOrder;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Sort;

import ru.javabegin.training.library.springlibrary\_2.jsfui.controller.AbstractController;

import java.util.List;

import java.util.Map;

// модель для постраничного вывода списка книг при любом поиске

// можно применять не только к книгам, но и к любым типам данных, т.к. используется Generics

@Getter

@Setter

public class LazyDataTable<T> extends LazyDataModel<T> {

private AbstractController<T> abstractController;

public LazyDataTable(AbstractController<T> abstractController) {

this.abstractController = abstractController;

} //abstractController хранит ссылку на переданный контроллер

@Override

public List<T> load(int first, int pageSize, String sortField, SortOrder sortOrder, Map<String, Object> filters) { //вызывается, когда делаем какие-то манипуляции на странице JSF (н: выбираем, сколько показать книг на странице)

int pageNumber = first / pageSize;// считываем номер страницы

Sort.Direction sortDirection = Sort.Direction.ASC;// по-умолчанию - сортировка по возрастанию

if (sortOrder!=null) {

// все текущие настройки DataTable (сортировка, поле сортировки) будут передаваться в SQL запрос

switch (sortOrder) {

case DESCENDING:

sortDirection = Sort.Direction.DESC;

break;

}

}

Page<T> searchResult = abstractController.search(pageNumber, pageSize, sortField, sortDirection); //вызывает сервисный уровень и возвращает результат в searchResult;

this.setRowCount((int) searchResult.getTotalElements()); //setRowCount -указываем, сколько строк было найдено, чтобы (он) правильно распределил все служебные данные; вызываем getTotalElements метод у searchResult;

return searchResult.getContent(); //возвращаем List типа Book

}

**PdfContent.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.servlets;

import org.springframework.context.ApplicationContext;

import org.springframework.web.context.support.WebApplicationContextUtils;

import ru.javabegin.training.library.springlibrary\_2.jsfui.controller.BookController;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.io.OutputStream;

@WebServlet(name = "PdfContent",

urlPatterns = {"/PdfContent"})

public class PdfContent extends HttpServlet {

private ApplicationContext context;

@Override

public void init() throws ServletException {

// получение доступа к Spring контексту (контейнеру)

context = WebApplicationContextUtils.getRequiredWebApplicationContext(this.getServletContext());

}

/\*\*

\* Processes requests for both HTTP

\* <code>GET</code> and

\* <code>POST</code> methods.

\*

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

OutputStream out = response.getOutputStream();

try {

// считываем параметры

long id = Long.valueOf(request.getParameter("id"));

long viewCount = Long.valueOf(request.getParameter("viewCount"));

// получаем Spring бин

BookController bookController = ((BookController)context.getBean("bookController"));

// получить контент по id

byte[] content = bookController.getContent(id);

if (content==null){

response.sendRedirect(request.getContextPath()+"/error/error-pdf.html");

}else {

response.setContentType("application/pdf");

// увеличить кол-во просмотров книги на 1

bookController.updateViewCount(viewCount, id);

response.setContentLength(content.length);

out.write(content);

}

} catch (Exception ex) {

ex.printStackTrace();

} finally {

out.close();

}

}

// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">

/\*\*

\* Handles the HTTP

\* <code>GET</code> method.

\*

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

/\*\*

\* Handles the HTTP

\* <code>POST</code> method.

\*

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

/\*\*

\* Returns a short description of the servlet.

\*

\* @return a String containing servlet description

\*/

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

**AspectManagerHandler.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.util;

import lombok.extern.java.Log;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.primefaces.context.RequestContext;

import org.springframework.stereotype.Component;

import org.springframework.transaction.UnexpectedRollbackException;

import javax.faces.application.FacesMessage;

import javax.faces.context.FacesContext;

import java.util.ResourceBundle;

import java.util.logging.Level;

@Log

@Aspect // спец класс, который отображает текст ошибок или успешной операции (всплывающее сообщение)

@Component

public class AspectMessageHandler {

// не нужно перехватывать все ошибки в коде, AOP все сделает за нас

@Around("execution(\* ru.javabegin.training.library.springlibrary\_2.jsfui.controller.\*.deleteAction(..))")

// перехватывать все ошибки во всех методах подпакетов и классов

public void deleteConstraint(ProceedingJoinPoint jp) {

FacesContext context = FacesContext.getCurrentInstance();

ResourceBundle bundle = context.getApplication().getResourceBundle(context, "msg");

try {

jp.proceed(); // выполнить метод

context.addMessage(null, new FacesMessage(null, bundle.getString("deleted")));

RequestContext.getCurrentInstance().update("info");

} catch (Throwable throwable) {

if (throwable instanceof UnexpectedRollbackException) {// если транзакция откатилась

log.log(Level.WARNING, throwable.getMessage());

throwable.printStackTrace();

// если это ошибка constraint при удалении

if (((UnexpectedRollbackException) throwable).getMostSpecificCause() instanceof java.sql.SQLIntegrityConstraintViolationException) {

// показываем сообщение пользователю на странице

context.addMessage(null, new FacesMessage(null, bundle.getString("constraint\_delete\_record")));

}

}

}

// чтобы компонент показал сообщение - его нужно обновить

RequestContext.getCurrentInstance().update("info");

}

@Around("execution(\* ru.javabegin.training.library.springlibrary\_2.dao.\*.save(..))")

public void addNewSprValue(ProceedingJoinPoint jp) {

try {

Object obj = jp.proceed();

FacesContext context = FacesContext.getCurrentInstance();

ResourceBundle bundle = context.getApplication().getResourceBundle(context, "msg");

// показываем сообщение пользователю на странице

context.addMessage(null, new FacesMessage(null, bundle.getString("added")+": \""+obj.toString()+"\""));

RequestContext.getCurrentInstance().update("info");

} catch (Throwable throwable) {

log.log(Level.WARNING, throwable.getMessage());

throwable.printStackTrace();

}

}

**CookieHelper.java**

package ru.javabegin.training.library.springlibrary\_2.jsfui.util;

import javax.faces.context.FacesContext;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class CookieHelper {

public static final String COOKIE\_LANG = "spring-library-lang";

public static void setCookie(String name, String value, int expiry) {

FacesContext facesContext = FacesContext.getCurrentInstance();

HttpServletRequest request = (HttpServletRequest) facesContext.getExternalContext().getRequest();

Cookie cookie = null;

Cookie[] userCookies = request.getCookies();

if (userCookies != null && userCookies.length > 0 ) {

for (int i = 0; i < userCookies.length; i++) {

if (userCookies[i].getName().equals(name)) {

cookie = userCookies[i];

break;

}

}

}

// чтобы перезаписать существующий cookiе - name и path должны совпадать (иначе может создасться cookie с одним именем, но разными path)

if (cookie != null) {

cookie.setValue(value);

} else {

cookie = new Cookie(name, value);

}

cookie.setPath(request.getContextPath());

cookie.setMaxAge(expiry);

HttpServletResponse response = (HttpServletResponse) facesContext.getExternalContext().getResponse();

response.addCookie(cookie);

}

public static Cookie getCookie(String name) {

FacesContext facesContext = FacesContext.getCurrentInstance();

HttpServletRequest request = (HttpServletRequest) facesContext.getExternalContext().getRequest();

Cookie cookie = null;

Cookie[] userCookies = request.getCookies();

if (userCookies != null && userCookies.length > 0 ) {

for (int i = 0; i < userCookies.length; i++) {

if (userCookies[i].getName().equals(name)) {

cookie = userCookies[i];

return cookie;

}

}

}

return null;

}

}

**AuthHandler.java**

package ru.javabegin.training.library.springlibrary\_2.spring.auth;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.DefaultRedirectStrategy;

import org.springframework.security.web.RedirectStrategy;

import org.springframework.security.web.authentication.AuthenticationFailureHandler;

import org.springframework.stereotype.Component;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

@Component

public class AuthHandler implements AuthenticationFailureHandler {

private RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();

@Override

public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, AuthenticationException exception) throws IOException, ServletException {

HttpSession session = request.getSession(false);

if (session != null) {

session.setAttribute("loginFailed", "login failed");

}

if (response.isCommitted()) {

return;

}

redirectStrategy.sendRedirect(request, response, "/index.xhtml");

}

**FacesConfig.java**

package ru.javabegin.training.library.springlibrary\_2.spring.config;

import org.primefaces.webapp.filter.FileUploadFilter;

import org.springframework.boot.web.servlet.FilterRegistrationBean;

import org.springframework.boot.web.servlet.ServletContextInitializer;

import org.springframework.boot.web.servlet.ServletRegistrationBean;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import ru.javabegin.training.library.springlibrary\_2.jsfui.servlets.PdfContent;

import javax.faces.webapp.FacesServlet;

import javax.servlet.MultipartConfigElement;

@Configuration

public class FacesConfig {

@Bean

public ServletContextInitializer servletContextInitializer() {

return servletContext -> {

servletContext.setInitParameter("primefaces.CLIENT\_SIDE\_VALIDATION", "true");

servletContext.setInitParameter("primefaces.THEME", "bootstrap");

servletContext.setInitParameter("primefaces.FONT\_AWESOME", "true");

servletContext.setInitParameter("primefaces.UPLOADER", "commons");

};

}

@Bean

public FilterRegistrationBean fileUploadFilterRegistrationBean() {

FilterRegistrationBean registrationBean = new FilterRegistrationBean();

registrationBean.addUrlPatterns("\*.xhtml", "/javax.faces.resource/\*");

registrationBean.setFilter(new FileUploadFilter());

return registrationBean;

}

//чтобы загружать двоичные данные на странице

@Bean

public ServletRegistrationBean servletRegistrationBean(MultipartConfigElement multipartConfigElement) {

FacesServlet servlet = new FacesServlet();

ServletRegistrationBean servletRegistrationBean = new ServletRegistrationBean(servlet, "\*.xhtml", "/javax.faces.resource/\*");

servletRegistrationBean.setMultipartConfig(multipartConfigElement);

return servletRegistrationBean;

}

// сервлет для чтения PDF контента

@Bean

public ServletRegistrationBean pdfServletRegistraiton() {

ServletRegistrationBean pdfServlet = new ServletRegistrationBean(new PdfContent(), new String[]{"/PdfContent"});

pdfServlet.setName("PdfContent");

return pdfServlet;

}

}

**SecurityConfig.java**

package ru.javabegin.training.library.springlibrary\_2.spring.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

import ru.javabegin.training.library.springlibrary\_2.spring.auth.AuthHandler;

import javax.sql.DataSource;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter { // класс для настройки безопасности приложения

@Autowired

private AuthHandler authHandler;

@Autowired

private DataSource dataSource;// этот бин создает автоматичеки spring boot, поэтому мы просто его используем

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.jdbcAuthentication().dataSource(dataSource)

.usersByUsernameQuery(

"select username,password,enabled from library.user where username = ?")

.authoritiesByUsernameQuery(

"select username, role from library.user\_roles where username = ?").passwordEncoder(passwordEncoder());

}

@Override

protected void configure(HttpSecurity http) throws Exception {

// настройка ограничений доступа к страницам

http.authorizeRequests()

.antMatchers("/\*\*").permitAll()

.antMatchers("/pages/spr.xhtml").hasRole("ADMIN")// здесь автоматически будет добавлен префикс ROLE\_, поэтому указываем название роли без него

.antMatchers("/pages/books.xhtml").hasAnyRole("ADMIN", "USER")// здесь автоматически будет добавлен префикс ROLE\_, поэтому указываем название роли без него

.and()

.exceptionHandling().accessDeniedPage("/index.xhtml")// при ошибке доступа - будет перенправляться на страницу с книгами

.and()

.csrf().disable()

// окно аутентификации

.formLogin()

.loginPage("/index.xhtml")

.failureHandler(authHandler)

.defaultSuccessUrl("/pages/books.xhtml")

.loginProcessingUrl("/login")

.passwordParameter("password")

.usernameParameter("username")

.and()

// настройка выхода пользователя из системы

.logout()

.logoutRequestMatcher(new AntPathRequestMatcher("/logout"))

.logoutSuccessUrl("/index.xhtml")

.deleteCookies("JSESSIONID", "SPRING\_SECURITY\_REMEMBER\_ME\_COOKIE")

.invalidateHttpSession(true);

/\* .and()

.requiresChannel().anyRequest().requiresSecure(). // все запросы будут перенаправляться в HTTPS протокол

and()

.portMapper()

.http(8080).mapsTo(8181)// порты для HTTP и HTTPS

;\*/

}

}

**RedirectController.java**

package ru.javabegin.training.library.springlibrary\_2.spring.controller;

import lombok.extern.java.Log;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@Controller

@Log

public class RedirectController {

//при запуске проекта - первый запрос попадает сюда

@RequestMapping(value = "", method = RequestMethod.GET)

public String baseUrlRedirect(HttpServletRequest request, HttpServletResponse httpServletResponse) {

// перенаправление на страницу индекс

return "redirect:" + request.getRequestURL().append("/index.xhtml").toString();

}

}

**AuthorRepository.java**

package ru.javabegin.training.library.springlibrary\_2.spring.repository;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import org.springframework.data.rest.core.annotation.RestResource;

import org.springframework.stereotype.Repository;

import ru.javabegin.training.library.springlibrary\_2.domain.Author;

import java.util.List;

@RepositoryRestResource

@Repository // специальный Spring bean, который помечает интерфейс как Repository

public interface AuthorRepository extends JpaRepository<Author, Long> {// JpaRepository - содержит CRUD функционал + постраничность

@RestResource(path = "findByFio")

// на основании имени метода будет построен Hibernate запрос

List<Author> findByFioContainingIgnoreCaseOrderByFio(String fio);

@RestResource(path = "findByFioPage")

// Page содержит результаты выполнения запроса и служебные данные для постраничности

Page<Author> findByFioContainingIgnoreCaseOrderByFio(String fio, Pageable pageable); // Pageable - параметры для постраничности

**BookRepository.java**

package ru.javabegin.training.library.springlibrary\_2.spring.repository;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Modifying;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import ru.javabegin.training.library.springlibrary\_2.domain.Book;

import java.util.List;

// примечания:

// при получении списка книг - контент для каждой книги - пустой. Только когда пользователь нажимает на чтение книги - делаем запрос в БД на получение контента (по требованию)

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

// поиск книг по названию или автору

// если имя метода получается слишком длинным - можно использовать @Query с HQL

Page<Book> findByNameContainingIgnoreCaseOrAuthorFioContainingIgnoreCaseOrderByName(String name, String fio, Pageable pageable );

@Query("select new ru.javabegin.training.library.springlibrary\_2.domain.Book(b.id, b.name, b.pageCount, b.isbn, b.genre, b.author, b.publisher, b.publishYear, b.image, b.descr, b.viewCount, b.totalRating, b.totalVoteCount, b.avgRating) from Book b")

Page<Book> findAllWithoutContent(Pageable pageable); //возвращает список книг с постраничностью

// если запрос изменяет данные - нужен @Modifying

@Modifying(clearAutomatically = true)

@Query("update Book b set b.content=:content where b.id =:id")

void updateContent(@Param("content") byte[] content, @Param("id") long id);

//Для топовых книг показываем только изображение (поэтому остальные поля не выбираем)

//В классе Book должен быть соответствующий конструктор

@Query("select new ru.javabegin.training.library.springlibrary\_2.domain.Book(b.id, b.image) from Book b")

List<Book> findTopBooks(Pageable pageable);

//поиск книг по жанру

@Query("select new ru.javabegin.training.library.springlibrary\_2.domain.Book(b.id, b.name, b.pageCount, b.isbn, b.genre, b.author, b.publisher, b.publishYear, b.image, b.descr, b.viewCount, b.totalRating, b.totalVoteCount, b.avgRating) from Book b where b.genre.id=:genreId")

Page<Book> findByGenre(@Param("genreId") long genreId, Pageable pageable);

// получение контента по id книги

@Query("select b.content FROM Book b where b.id = :id")

byte[] getContent(@Param("id") long id);

// обновить статистику просмотров для книги

@Modifying

@Query("update Book b set b.viewCount=:viewCount where b.id =:id")

void updateViewCount(@Param("viewCount") long viewCount, @Param("id") long id);

// обновить данные рейтинга

@Modifying

@Query("update Book b set b.totalVoteCount=:totalVoteCount, b.totalRating=:totalRating, b.avgRating=:avgRating where b.id =:id")

void updateRating(@Param("totalRating") long totalRating, @Param("totalVoteCount") long totalVoteCount, @Param("avgRating") int avgRating, @Param("id") long id);

}

**GenreRepository.java**

package ru.javabegin.training.library.springlibrary\_2.spring.repository;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.rest.core.annotation.RestResource;

import org.springframework.stereotype.Repository;

import ru.javabegin.training.library.springlibrary\_2.domain.Genre;

import java.util.List;

@Repository

public interface GenreRepository extends JpaRepository<Genre, Long> {

@RestResource(path = "findByName")

List<Genre> findByNameContainingIgnoreCaseOrderByName(String name);

@RestResource(path = "findByNamePage")

Page<Genre> findByNameContainingIgnoreCaseOrderByName(String name, Pageable pageable);

}

**PublisherRepository.java**

package ru.javabegin.training.library.springlibrary\_2.spring.repository;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.rest.core.annotation.RestResource;

import org.springframework.stereotype.Repository;

import ru.javabegin.training.library.springlibrary\_2.domain.Publisher;

import java.util.List;

@Repository

public interface PublisherRepository extends JpaRepository<Publisher, Long> {

@RestResource(path = "findByName")

List<Publisher> findByNameContainingIgnoreCaseOrderByName(String name);

@RestResource(path = "findByNamePage")

Page<Publisher> findByNameContainingIgnoreCaseOrderByName(String name, Pageable pageable);

}

**ServletInitializer.java**

package ru.javabegin.training.library.springlibrary\_2;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

import org.springframework.context.annotation.EnableAspectJAutoProxy;

@SpringBootApplication

@EnableAspectJAutoProxy // включаем использование AspectJ

//@EnableAutoConfiguration

//@ComponentScan(basePackages = {"ru.javabegin.training.library.springlibrary\_2"}) //уже не требуется указывать

public class ServletInitializer extends SpringBootServletInitializer {

/\* public static void main(String[] args) {

SpringApplication.run(ServletInitializer.class, args);

}\*/

}

**Books.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://java.sun.com/jsf/html"

xmlns:ui="http://java.sun.com/jsf/facelets"

xmlns:b="http://bootsfaces.net/ui"

xmlns:o="http://omnifaces.org/ui"

xmlns:p="http://primefaces.org/ui"

xmlns:f="http://xmlns.jcp.org/jsf/core"

>

<h:body>

<ui:composition template="/templates/books/books-template.xhtml"> <!-- указываем ui:composition , чтобы потом использовать template -->

<ui:define name="content">

<ui:include src="/templates/dialogs/edit-dialog.xhtml"/>

<ui:include src="/templates/dialogs/confirm-dialog.xhtml"/>

<!-- форма должны быть вне таблицы с данными, чтобы отрабатывать все кнопки и ссылки внутри формы -->

<h:form id="booksForm">

<p:dataGrid id="booksList" var="book" value="#{bookController.lazyModel}" columns="5"

paginatorPosition="bottom"

paginator="true"

rows="#{bookController.rowsCount}"

emptyMessage=""

rowIndexVar="rowIndexVar"

rowsPerPageTemplate="10,15,20,30"

paginatorTemplate="{CurrentPageReport} {FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"

lazy="true">

<!--результаты поиска-->

<f:facet name="header">

<p:outputPanel styleClass="table-header">

<p:outputLabel value="#{msg.founded}: #{bookController.bookPages.totalElements} "/>

<p:outputLabel value="&#160;(#{bookController.getSearchMessage()})"

rendered="#{bookController.getSearchMessage() != null}"/>

</p:outputPanel>

</f:facet>

<!--карточка с данными книги-->

<p:outputPanel styleClass="card">

<!--обложка книги-->

<div class="book-icon">

<h:outputLink value="#{request.contextPath}/PdfContent" target="\_blank"

styleClass="read-link">

<f:param name="id" value="#{book.id}"/>

<f:param name="viewCount" value="#{book.viewCount}"/>

<o:graphicImage cache="false" value="#{book.image}" dataURI="true" />

</h:outputLink>

</div>

<!--описание книги-->

<div class="book-info">

<div class="author">

<h:outputText value="#{book.author.fio}" styleClass="author"/>

</div>

<!-- нижняя часть под картинкой -->

<div class="add-info">

<div style="width:10%; float:left;">

<o:graphicImage name="images/see.png" styleClass="see-count"/>

</div>

<!-- кол-во просмотров -->

<div style="width:20%; float:left; margin-left: 10px; font-size: 14px;">

<h:outputText value="#{book.viewCount}" styleClass="view-count" id="viewCountLabel"/>

</div>

<!--голосование-->

<div style="width:60%; float:right;">

<p:rating id="rating" value="#{book.avgRating}" styleClass="rating"

cancel="false"

stars="5">

<p:ajax event="rate" listener="#{bookController.onrate}"

update="@([id$=booksList])"

onsuccess="alert('Спасибо')"/>

<param name="bookIndex" value="#{rowIndexVar}"/>

</p:rating>

</div>

</div>

</div>

<!--инструменты администратора (редактирование, удаление)-->

<p:outputPanel layout="block" styleClass="admin-region"

rendered="${userController.hasRole('ADMIN')}">

<p:commandLink title="#{msg.delete}"

action="#{bookController.deleteAction()}" update=":booksForm:booksList">

<p:confirm header="#{msg.confirm}" message="#{msg.prompt\_delete}"

icon="ui-icon-alert"/>

<f:setPropertyActionListener value="#{book}"

target="#{bookController.selectedBook}"/>

<o:graphicImage name="images/delete.png" styleClass="delete-icon" dataURI="true"/>

</p:commandLink>

<p:commandLink title="#{msg.edit}"

action="#{bookController.editAction()}"

update="@([id$=dialogEditBook])">

<f:setPropertyActionListener value="#{book}"

target="#{bookController.selectedBook}"/>

<o:graphicImage name="images/edit.png" styleClass="edit-icon" dataURI="true"/>

</p:commandLink>

</p:outputPanel>

</p:outputPanel>

</p:dataGrid>

</h:form>

</ui:define>

</ui:composition>

</h:body>

</html>

**Spr.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui"

xmlns:o="http://omnifaces.org/ui"

>

<h:body>

<ui:composition template="/templates/spr/spr-template.xhtml">

<ui:define name="content">

<!-- чтобы использовать диалоговое окно подтверждения при удалении -->

<ui:include src="/templates/dialogs/confirm-dialog.xhtml"/>

<!-- табы для редактирования справочников -->

<p:tabView dynamic="true" cache="false" id="tabView">

<p:ajax event="tabChange" listener="#{sprController.onTabChange}"/>

<!--таб для редактирования авторов-->

<p:tab title="#{msg.authors}" id="tabAuthors">

<h:form id="authorForm">

<p:dataTable id="authorList" var="author" value="#{authorController.lazyModel}"

paginatorPosition="bottom"

paginator="true"

rows="#{authorController.rowsCount}"

first="#{authorController.first}"

paginatorTemplate="{CurrentPageReport} {FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"

style="margin:5px" lazy="true"

rowsPerPageTemplate="20,50,100"

>

<!-- сколько элементов найдено для отображения -->

<f:facet name="header">

<p:outputLabel value="#{msg.founded}: #{authorController.authorPages.totalElements}" id="authorCount"/>

</f:facet>

<!-- столбец имя автора-->

<p:column headerText="#{msg.name}">

<p:commandLink title="#{msg.edit}" action="#{authorController.editAction()}" update="@([id$=dialogAuthor])" process="@form" immediate="true">

<h:outputText value="#{author.fio}"/>

<!-- при выполнении действия editAction - записывает в переменную selectedAuthor текущего автора, чтобы в диалоговом окне отобразить его данные для редактирования -->

<f:setPropertyActionListener value="#{author}" target="#{authorController.selectedAuthor}"/>

</p:commandLink>

</p:column>

<!-- столбец дата рождения автора-->

<p:column headerText="#{msg.birthday}">

<p:commandLink title="#{msg.edit}" action="#{authorController.editAction()}"

update="@([id$=dialogAuthor])" process="@form" immediate="true">

<p:outputLabel value="#{author.birthday}" style="width:100%">

<f:convertDateTime pattern="yyyy-MM-dd" type="date" />

</p:outputLabel>

<!-- при выполнении действия editAction - записывает в переменную selectedAuthor текущего автора, чтобы в диалоговом окне отобразить его данные для редактирования -->

<f:setPropertyActionListener value="#{author}" target="#{authorController.selectedAuthor}"/>

</p:commandLink>

</p:column>

<!-- столбец с иконкой удаления-->

<p:column style="width:50px">

<p:commandLink title="#{msg.delete}"

action="#{authorController.deleteAction()}"

update="@([id$=authorList])" process="@form" immediate="true"

/>

<o:graphicImage name="images/delete.png" styleClass="delete-icon" dataURI="true"/>

<p:confirm header="#{msg.confirm}" message="#{msg.prompt\_delete}?"

icon="ui-icon-alert" />

<!-- при выполнении действия deleteAction - записывает в переменную selectedAuthor текущего издателя, чтобы знать кого удалять -->

<f:setPropertyActionListener value="#{author}"

target="#{authorController.selectedAuthor}"/>

</p:commandLink>

</p:column>

<!-- столбец с иконкой редактирования-->

<p:column style="width:50px">

<p:commandLink title="#{msg.edit}" action="#{authorController.editAction()}"

update="@([id$=dialogAuthor])" process="@form" immediate="true">

<o:graphicImage name="images/edit.png" styleClass="edit-icon" dataURI="true"/>

<!-- при выполнении действия editAction - записывает в переменную selectedAuthor текущего автора, чтобы в диалоговом окне отобразить его данные для редактирования -->

<f:setPropertyActionListener value="#{author}"

target="#{authorController.selectedAuthor}"/>

</p:commandLink>

</p:column>

</p:dataTable>

</h:form>

<!-- диалоговое окно для редактирования author -->

<p:dialog id="dialogAuthor" header="#{msg.author}" widgetVar="dialogAuthor" resizable="false"

modal="true"

showEffect="fade"

closeOnEscape="true"

/>

<h:form id="dialogAuthorForm">

<div class="popup-info">

<div class="ava-info no-border">

<div class="block">

<p>#{msg.fio}</p>

<div class="ava-form">

<p:inputText id="fio" value="#{authorController.selectedAuthor.fio}" required="true" />

</div>

</div>

<div class="block">

<p>#{msg.birthday}</p>

<div class="ava-form">

<p:calendar value="#{authorController.selectedAuthor.birthday}"

style="width:100%" pattern="yyyy-MM-dd"

mask="true" required="true" />

</div>

</div>

<div class="block">

<div class="ava-form">

<p:commandButton value="#{msg.save}"

actionListener="#{authorController.save}"

update="@([id$=authorList])"

process="@([id$=dialogAuthorForm])" style="float:right"/>

</div>

</div>

</div>

</div>

</h:form>

</p:dialog>

</p:tab>

<!--таб для редактирования жанров-->

<p:tab title="#{msg.genres}" id="tabGenres">

<h:form id="genreForm">

<p:dataTable id="genreList" var="genre" value="#{genreController.lazyModel}"

paginatorPosition="bottom"

paginator="true"

rows="#{genreController.rowsCount}"

first="#{genreController.first}"

paginatorTemplate="{CurrentPageReport} {FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"

style="margin:5px" lazy="true"

rowsPerPageTemplate="20,50,100"/>

<!-- сколько элементов найдено для отображения -->

<f:facet name="header">

<p:outputLabel value="#{msg.founded}: #{genreController.genrePages.totalElements}"

id="genreCount"/>

</f:facet>

<p:column headerText="#{msg.name}">

<p:commandLink title="#{msg.edit}" action="#{genreController.editAction()}"

update="@([id$=dialogGenre])" process="@form" immediate="true">

<h:outputText value="#{genre.name}"/>

<!-- при выполнении действия editAction - записывает в переменную selectedGenre текущего жанра, чтобы в диалоговом окне отобразить его данные для редактирования -->

<f:setPropertyActionListener value="#{genre}"

target="#{genreController.selectedGenre}"/>

</p:commandLink>

</p:column>

<!-- столбец с иконкой удаления-->

<p:column style="width:50px">

<p:commandLink title="#{msg.delete}"

action="#{genreController.deleteAction()}"

update="@([id$=genreList])" process="@form"

immediate="true">

<o:graphicImage name="images/delete.png" styleClass="delete-icon" dataURI="true"/>

<p:confirm header="#{msg.confirm}" message="#{msg.prompt\_delete}?"

icon="ui-icon-alert"/>

<!-- при выполнении действия deleteAction - записывает в переменную selectedGenre текущего издателя, чтобы знать кого удалять -->

<f:setPropertyActionListener value="#{genre}"

target="#{genreController.selectedGenre}"/>

</p:commandLink>

</p:column>

<!-- столбец с иконкой редактирования-->

<p:column style="width:50px">

<p:commandLink title="#{msg.edit}" action="#{genreController.editAction()}"

update="@([id$=dialogGenre])" process="@form" immediate="true">

<o:graphicImage name="images/edit.png" styleClass="edit-icon" dataURI="true"/>

<!-- при выполнении действия editAction - записывает в переменную selectedGenre текущего жанра, чтобы в диалоговом окне отобразить его данные для редактирования -->

<f:setPropertyActionListener value="#{genre}"

target="#{genreController.selectedGenre}"/>

</p:commandLink>

</p:column>

</p:dataTable>

</h:form>

<!-- диалоговое окно для редактирования genre -->

<p:dialog id="dialogGenre" header="#{msg.genre}" widgetVar="dialogGenre" resizable="false"

modal="true"

showEffect="fade"

closeOnEscape="true"

/>

<h:form id="dialogGenreForm">

<div class="popup-info">

<div class="ava-info no-border">

<div class="block">

<p>#{msg.subj\_name}</p>

<div class="ava-form">

<p:inputText id="nameGenre" value="#{genreController.selectedGenre.name}"/>

</div>

</div>

<div class="block">

<div class="ava-form">

<p:commandButton value="#{msg.save}"

actionListener="#{genreController.save}"

update="@([id$=genreList])"

process="@([id$=dialogGenreForm])" style="float:right"/>

</div>

</div>

</div>

</h:form>

</p:dialog>

</p:tab>

<!--таб для редактирования издателей-->

<p:tab title="#{msg.publishers}" id="tabPublishers">

<h:form id="publisherForm">

<p:dataTable id="publisherList" var="publisher" value="#{publisherController.lazyModel}"

paginatorPosition="bottom"

paginator="true"

rows="#{publisherController.rowsCount}"

first="#{publisherController.first}"

paginatorTemplate="{CurrentPageReport} {FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"

style="margin:5px" lazy="true"

rowsPerPageTemplate="20,50,100"

/>

<!-- сколько элементов найдено для отображения -->

<f:facet name="header">

<p:outputLabel value="#{msg.founded}: #{publisherController.publisherPages.totalElements}" id="publisherCount"/>

</f:facet>

<p:column headerText="#{msg.name}">

<p:commandLink title="#{msg.edit}" action="#{publisherController.editAction()}"

update="@([id$=dialogPublisher])" process="@form" immediate="true">

<h:outputText value="#{publisher.name}"/>

<!-- при выполнении действия editAction - записывает в переменную selectedPublisher текущего издателя, чтобы в диалоговом окне отобразить его данные для редактирования -->

<f:setPropertyActionListener value="#{publisher}"

target="#{publisherController.selectedPublisher}"/>

</p:commandLink>

</p:column>

<!-- столбец с иконкой удаления-->

<p:column style="width:50px">

<p:commandLink title="#{msg.delete}"

action="#{publisherController.deleteAction()}"

update="@([id$=publisherList])" process="@form"

immediate="true"

id="deletePublisher"

>

<o:graphicImage name="images/delete.png" styleClass="delete-icon" dataURI="true"/>

<p:confirm header="#{msg.confirm}" message="#{msg.prompt\_delete}?"

icon="ui-icon-alert"/>

<!-- при выполнении действия deleteAction - записывает в переменную selectedPublisher текущего издателя, чтобы знать кого удалять -->

<f:setPropertyActionListener value="#{publisher}" target="#{publisherController.selectedPublisher}"/>

</p:commandLink>

</p:column>

<!-- столбец с иконкой редактирования-->

<p:column style="width:50px">

<p:commandLink title="#{msg.edit}" action="#{publisherController.editAction()}" update="@([id$=dialogPublisher])" process="@form" immediate="true">

<o:graphicImage name="images/edit.png" styleClass="edit-icon" dataURI="true"/>

<!-- при выполнении действия editAction - записывает в переменную selectedPublisher текущего издателя, чтобы в диалоговом окне отобразить его данные для редактирования -->

<f:setPropertyActionListener value="#{publisher}" target="#{publisherController.selectedPublisher}"/>

</p:commandLink>

</p:column>

</p:dataTable>

</h:form>

<!-- диалоговое окно для редактирования publisher -->

<p:dialog id="dialogPublisher" header="#{msg.publisher}" widgetVar="dialogPublisher"

resizable="false"

modal="true"

showEffect="fade"

closeOnEscape="true"

>

<h:form id="dialogPublisherForm">

<div class="popup-info">

<div class="ava-info no-border">

<div class="block">

<p>#{msg.subj\_name}</p>

<div class="ava-form">

<p:inputText id="namePublisher"

value="#{publisherController.selectedPublisher.name}" />

</div>

</div>

<div class="block">

<div class="ava-form">

<p:commandButton value="#{msg.save}"

actionListener="#{publisherController.save}"

update="@([id$=publisherList])"

process="@([id$=dialogPublisherForm])"

style="float:right"/>

</div>

</div>

</div>

</div>

</h:form>

</p:dialog>

</p:tab>

</p:tabView>

</ui:define>

<!-- подставляем компонент для поиска справочных значений -->

<ui:define name="search">

<p:inputText style="margin-right:10px" placeholder="#{msg.search}" value="#{sprController.searchText}"

id="searchInput"/>

<p:commandButton value="#{msg.search}" actionListener="#{sprController.search}" process="@form"/>

</ui:define>

</ui:composition>

</h:body>

</html>

**Books-header.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:o="http://omnifaces.org/ui"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:p="http://primefaces.org/ui">

<f:view>

<div class="header-div">

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<div style="width: 50%">

<ul class="nav navbar-nav">

<ui:fragment ><li><o:graphicImage name="images/books.png" style="height:50px; padding:10px;" dataURI="true"/></li></ui:fragment>

<ui:fragment rendered="${userController.hasRole('ADMIN')}"><li> <h:outputLink value="#{request.contextPath}/pages/spr.xhtml"

> #{msg.dict}</h:outputLink></li></ui:fragment>

<ui:fragment rendered="${userController.hasRole('ADMIN')}"><li class="colored"> <p:commandLink title="Add button" rendered="${userController.hasRole('ADMIN')}" actionListener="#{bookController.addAction()}" update="@([id$=dialogEditBook])" styleClass="new-book-item">

<p:outputLabel

value="#{msg.add\_book}" styleClass="new-book-label"/>

</p:commandLink>

</li>

</ui:fragment>

</ul>

</div>

<div style="width:50%; float: right">

<div style="width:50%; float: left; padding-right: 20px; padding-left: 20px; height: 50px;">

<h:form styleClass="search-form">

<p:inputText style="margin-right:10px" placeholder="#{msg.search}"

value="#{bookController.searchText}" styleClass="search"/>

<p:commandButton styleClass="search-button" value="search" process="@form"

update="@([id$=booksList])"

actionListener="#{bookController.searchAction}"

style="visibility: hidden"/>

</h:form>

</div>

<ui:include src="../part/logout.xhtml"/>

</div>

<ui:include src="../part/locales.xhtml"/>

</div>

</div>

</nav>

</div>

<p:ajaxStatus id="ajaxStatusPanel">

<f:facet name="start">

<div id="loading\_wrapper">

<p>#{msg.loading}...</p>

</div>

</f:facet>

<f:facet name="complete">

<h:outputText value="" />

</f:facet>

</p:ajaxStatus>

</f:view>

</html>

**Books-leftmenu.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:p="http://primefaces.org/ui"

xmlns:b="http://bootsfaces.net/ui"

xmlns:f="http://xmlns.jcp.org/jsf/core">

<f:view>

<div class="genres">

<!--ссылка на поиск всех жанров-->

<p:commandLink styleClass="list-group-item all-genres" actionListener="#{bookController.showAll()}"

update="@([id$=booksList])">

<p:outputLabel value="#{msg.all\_genres}" styleClass="genre-name all-genres"/>

</p:commandLink>

<!--меню с названиями жанров (для поиска книг по жанрам) -->

<p:repeat value="#{genreController.all}" var="genre">

<p:commandLink class="list-group-item" actionListener="#{bookController.showBooksByGenre(genre.id)}"

update="@([id$=booksList])">

<p:outputLabel value="#{genre.name}" styleClass="genre-name"/>

</p:commandLink>

</p:repeat>

</div>

</f:view>

**Books-template.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<!-- основной файл шаблона, с которого начинается загрузка страницы books -->

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui"

xmlns:b="http://bootsfaces.net/ui"

xmlns:ui="http://java.sun.com/jsf/facelets">

<h:head>

<h:outputStylesheet name="css/dialog.css" />

<h:outputStylesheet name="css/books.css" />

<h:outputStylesheet name="css/footer.css" />

<!-- порядок загрузки стилей тоже важен, т.к. одинаковые селекторы могут перезаписываться -->

<title>${msg.online\_library}</title>

</h:head>

<f:view locale="#{localeChanger.currentLocale}">

<ui:include src="books-header.xhtml"/>

<ui:include src="books-top.xhtml"/>

<b:container>

<b:row>

<b:column span="2">

<ui:include src="books-leftmenu.xhtml"/>

</b:column>

<b:column span="10">

<ui:insert name="content"/>

</b:column>

</b:row>

</b:container>

<ui:include src="../part/footer.xhtml"/>

</f:view>

</html>

**Books-top.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui"

xmlns:o="http://omnifaces.org/ui"

>

<f:view>

<div class="box">

<div class="main2">

<div class="module-specials">

<div class="boxIndent">

<ul class="box-product">

<li>

<!-- для огранизации цикла-->

<p:repeat value="#{bookController.topBooks}" var="book">

<!-- для каждой итерации вставлять этот блок -->

<div class="row-product">

<div class="featuredIndent">

<div class="product\_image\_container">

<div class="product\_image\_shadow">

<h:outputLink value="#{request.contextPath}/PdfContent" target="\_blank">

<o:graphicImage width="157" height="236" value="#{book.image}"

dataURI="true"/>

<f:param name="id" value="#{book.id}"/>

<f:param name="filename" value="#{book.name}"/>

<f:param name="viewCount" value="#{book.viewCount}"/>

</h:outputLink>

</div>

</div>

</div>

</div>

</p:repeat>

</li>

</ul>

</div>

</div>

</div>

</div>

</f:view>

</html>

**Confirm-dialog.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui"

>

<f:view>

<p:confirmDialog global="true" showEffect="fade" hideEffect="fade">

<p:commandButton value="#{msg.yes}" type="button" styleClass="ui-confirmdialog-yes" icon="ui-icon-check"/>

<p:commandButton value="#{msg.no}" type="button" styleClass="ui-confirmdialog-no" icon="ui-icon-close"/>

</p:confirmDialog>

</f:view>

</html>

**Edit-dialog.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui"

xmlns:o="http://omnifaces.org/ui"

>

<f:view>

<p:dialog id="dialogEditBook" header="#{msg.book}" widgetVar="dialogEditBook" resizable="false" modal="true"

showEffect="fade" width="750">

<!-- действие при закрытии диалогового окна -->

<p:ajax event="close" listener="#{bookController.onCloseDialog}" />

<h:form id="formEditBook">

<div class="popup-info">

<div class="avatar-box">

<!-- загрузка обложки книги -->

<o:graphicImage width="240" height="320" cache="false" value="#{bookController.uploadedImage}"

rendered="#{not empty bookController.uploadedImage}" id="bookImage"

dataURI="true"/>

<p:fileUpload id="imageUpload" fileUploadListener="#{bookController.uploadImage}"

mode="advanced"

auto="true"

dragDropSupport="true" label="#{msg.image\_load}"

update="image\_upload\_messages @([id$=imageUpload])" sizeLimit="1000000"

allowTypes="/(\.|\/)(gif|jpe?g|png)$/"/>

<p:growl id="image\_upload\_messages" showDetail="true"/>

<!-- показывает все сообщения об ошибках при редактировании -->

<p:messages autoUpdate="true" redisplay="false"/>

</div>

<div class="ava-info">

<div class="block">

<p>#{msg.subj\_name}</p>

<div class="ava-form">

<p:inputText id="inputBookName" value="#{bookController.selectedBook.name}" required="true" requiredMessage="#{msg.required\_book\_name}" />

</div>

</div>

<div class="block">

<p>#{msg.author}</p>

<div class="ava-form">

<p:autoComplete id="authorList" dropdown="true" converter="#{authorConverter}" var="author"

value="#{bookController.selectedBook.author}"

completeMethod="#{authorController.find}" itemLabel="#{author.fio}"

itemValue="#{author}" required="true" requiredMessage="#{msg.required\_author}"/>

</div>

</div>

<div class="block">

<p>#{msg.genre}</p>

<div class="ava-form">

<p:autoComplete id="genreList" dropdown="true" converter="#{genreConverter}" var="genre"

value="#{bookController.selectedBook.genre}"

completeMethod="#{genreController.find}" itemLabel="#{genre.name}"

itemValue="#{genre}" required="true" requiredMessage="#{msg.required\_genre}"/>

</div>

</div>

<div class="block">

<p>#{msg.publisher}</p>

<div class="ava-form">

<p:autoComplete id="publisherList" dropdown="true" converter="#{publisherConverter}"

var="publisher"

value="#{bookController.selectedBook.publisher}"

completeMethod="#{publisherController.find}" itemLabel="#{publisher.name}"

itemValue="#{publisher}" required="true" requiredMessage="#{msg.required\_publisher}"/>

</div>

</div>

<div class="block">

<p>ISBN</p>

<div class="ava-form">

<p:inputText id="isbn" value="#{bookController.selectedBook.isbn}" required="true" requiredMessage="#{msg.required\_isbn}" />

</div>

</div>

<div class="block">

<p>#{msg.year}</p>

<div class="ava-form">

<p:inputMask id="year" value="#{bookController.selectedBook.publishYear}" maxlength="4" placeHolder="\_" mask="9999" required="true" requiredMessage="#{msg.required\_year}"/>

</div>

</div>

<div class="block">

<p>#{msg.page\_count}</p>

<div class="ava-form">

<p:inputNumber id="page\_count" value="#{bookController.selectedBook.pageCount}"

decimalPlaces="0" thousandSeparator="" required="true" requiredMessage="#{msg.required\_page\_count}"/>

</div>

</div>

<div class="block">

<p>#{msg.content}</p>

<div class="ava-form">

<div class="loaded">

<!-- загрузка PDF контента -->

<p:fileUpload id="contentUpload" fileUploadListener="#{bookController.uploadContent}"

mode="advanced"

dragDropSupport="true" label="#{msg.content\_load}" auto="true"

styleClass="ui-fileupload"

update="content\_upload\_messages"

allowTypes="/(\.|\/)(pdf)$/"/>

<p:growl id="content\_upload\_messages" showDetail="true"/>

</div>

</div>

</div>

<div class="block">

<p>#{msg.descr} <span><h:outputText id="counter" /></span></p>

<p:inputTextarea cols="30" id="descr" value="#{bookController.selectedBook.descr}"

required="false"/>

<div class="ava-form">

<p:commandButton value="#{msg.save}" update="@([id$=booksList])" process="@form" actionListener="#{bookController.save}" />

</div>

</div>

</div>

</h:form>

</p:dialog></f:view>

</html>

**Footer.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:f="http://xmlns.jcp.org/jsf/core"

>

<f:view>

<footer class="footer-distributed">

<div class="container">

<div class="footer-left">

<p class="footer-company-about" style="width: 70%; ">

<span>Онлайн библиотека на Spring </span>

Разработка библиотеки с использованием технологий: Spring Boot, Spring Data, Spring Security, Hibernate, JSF+Facelets, MySQL, Gradle

</p>

</div>

<div class="footer-center">

<div>

<i class="fa fa-map-marker"></i>

<p><span>Беларусь, г. Витебск</span> ул. Тестовая, д.17</p>

</div>

<div>

<i class="fa fa-phone"></i>

<p>+375 25 1234567</p>

</div>

<div>

<i class="fa fa-envelope"></i>

<p><a href="mailto:support@onlibrary.ru">support@onlinelibrary.ru</a></p>

</div>

</div>

<div class="footer-right">

<p class="footer-company-about">

<span>О библиотеке</span>

Онлайн библиотека предназначена для поиска и просмотра книг по различным жанрам. Администраторы имеют возможность добавлять, удалять, редактировать.

</p>

<div class="footer-icons">

<a href="#"><i class="fa fa-facebook"></i></a>

<a href="#"><i class="fa fa-twitter"></i></a>

<a href="#"><i class="fa fa-linkedin"></i></a>

<a href="#"><i class="fa fa-github"></i></a>

</div>

</div>

</div>

</footer>

</f:view>

</html>

**Locales.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

>

<f:view>

<div class="lang">

<h:form>

<div class="lang">

<h:commandLink actionListener="#{localeChanger.changeLocale('ru')}" value="#{msg.ru}" styleClass="#{localeChanger.currentLocale == 'ru' ? 'active':''} lang-link"/>

<h:commandLink actionListener="#{localeChanger.changeLocale('en')}" value="#{msg.en}" styleClass="#{localeChanger.currentLocale == 'en' ? 'active':''} lang-link"/>

</div>

</h:form>

</div>

</f:view>

</html>

**Logout.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui"

>

<f:view>

<div style="float:right; width: 50%; text-align: right; ">

<p:outputLabel value="#{msg.welcome}: "

style=" color:lightgray; font-size: 14px; font-weight: normal; padding-top: 15px;"/>

<h:outputLink value="#{request.contextPath}/logout"

style="color:#dd993a; margin-left: 20px;">#{msg.logout}</h:outputLink>

</div>

</f:view>

</html>

**Spr-header.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:p="http://primefaces.org/ui"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:b="http://bootsfaces.net/ui"

xmlns:o="http://omnifaces.org/ui"

>

<!-- верхнее меню-->

<f:view>

<div class="header-div">

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<div style="width: 50%">

<ul class="nav navbar-nav">

<li>

<h:outputLink

value="#{request.contextPath}/pages/books.xhtml"> &#8592; #{msg.back}</h:outputLink>

</li>

<li class="colored">

<h:form>

<p:commandLink actionListener="#{sprController.addAction()}" title="#{msg.add\_value}"

styleClass="new-book-item" process="@form"><p:outputLabel

value="#{msg.add\_value}" styleClass="new-book-label"/></p:commandLink>

</h:form>

</li>

</ul>

</div>

div style="width:50%; float: right">

<div style="width:50%; float: left; padding-right: 20px; padding-left: 20px; height: 50px;">

<h:form styleClass="search-form">

<p:inputText style="margin-right:10px" placeholder="Search"

value="#{sprController.searchText}" styleClass="search" id="searchInput"/>

<p:commandButton styleClass="search-button" value="#{msg.search}" process="@form" actionListener="#{sprController.search}"

style="visibility: hidden"/>

</h:form>

</div>

<ui:include src="../part/logout.xhtml"/>

</div>

<ui:include src="../part/locales.xhtml"/>

</div>

</div>

</nav>

</div>

</f:view>

</html>

**Spr-template.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui"

xmlns:b="http://bootsfaces.net/ui"

xmlns:ui="http://java.sun.com/jsf/facelets">

<h:head>

<f:loadBundle var="msgs" basename="library"/>

<h:outputStylesheet name="css/books.css"/>

<h:outputStylesheet name="css/footer.css"/>

<h:outputStylesheet name="css/dialog.css" />

<title>${msg.online\_library}</title>

</h:head>

<!-- основной шаблон, с которого начинается сборка страницы справочников -->

<f:view transient="true" locale="#{localeChanger.currentLocale}">

<ui:include src="spr-header.xhtml"/>

<b:container>

<p:growl id="info" globalOnly="true" showSummary="false" showDetail="true" life="4000" />

<b:row>

<ui:insert name="content"/>

</b:row>

</b:container>

<ui:include src="../part/footer.xhtml"/>

</f:view>

</html>

**Index.xhtml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:f="http://xmlns.jcp.org/jsf/core">

<h:head>

<title>${msg.online\_library}</title>

<h:outputStylesheet name="css/index.css" />

</h:head>

<h:body>

<f:view locale="#{localeChanger.currentLocale}" >

<div id="login">

<h:outputText id="loginFailed" value="#{userController.loginFailedMessage}" styleClass="error"/>

<form id="login\_form" action="#{request.contextPath}/login" method="post">

<span class="fontawesome-user"></span>

<input

type="text" id="username" name="username" class="ui-corner-all"

required="required" placeholder="username"/>

<span class="fontawesome-lock"></span>

<input type="password" id="password" name="password"

class="ui-corner-all" required="required" placeholder="password"/>

<input type="submit" value="#{msg.login}"/>

</form>

</div>

</f:view>

</h:body>

</html>

**Faces-config.xml**

<?xml version='1.0' encoding='UTF-8'?>

<faces-config version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig\_2\_2.xsd">

<application>

<!-- упрощает работу с ресурсами для веб страницы (js, css и пр.) -->

<resource-handler>org.omnifaces.resourcehandler.UnmappedResourceHandler</resource-handler>

<locale-config>

<default-locale>ru</default-locale>

<supported-locale>ru</supported-locale>

<supported-locale>en</supported-locale>

</locale-config>

<resource-bundle>

<base-name>library</base-name>

<var>msg</var>

</resource-bundle>

<!-- подружить JSF и Spring контейнеры, чтобы видели бины друг друга -->

<el-resolver>

org.springframework.web.jsf.el.SpringBeanFacesELResolver

</el-resolver>

</application>

</faces-config>

**Glassfish-web.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Servlet 3.0//EN"

"http://glassfish.org/dtds/glassfish-web-app\_3\_0-1.dtd">

<glassfish-web-app>

<parameter-encoding default-charset="UTF-8"/>

</glassfish-web-app>

**Web.xml**

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0"

metadata-complete="true"

>

<!--DispatcherServlet здесь указывать не нужно - он настраивается автоматически с помощью Spring Boot-->

<servlet>

<servlet-name>facesServlet</servlet-name>

<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>facesServlet</servlet-name>

<url-pattern>/javax.faces.resource/\*</url-pattern>

<url-pattern>\*.xhtml</url-pattern>

</servlet-mapping>

<error-page>

<exception-type>javax.faces.application.ViewExpiredException</exception-type>

<location>/index.xhtml</location>

</error-page>

<!-- для правильного отображения дат в компонентах JSF -->

<context-param>

<param-name>javax.faces.DATETIMECONVERTER\_DEFAULT\_TIMEZONE\_IS\_SYSTEM\_TIMEZONE</param-name>

<param-value>true</param-value>

</context-param>

</web-app>