**Министерство образования Республики Беларусь**

**УО «Гродненский государственный университет им. Янки Купалы»**

**Институт повышения квалификации и переподготовки кадров**

**Кафедра современных технологий образования взрослых**

«Допустить к защите»

Заведующий кафедрой СТОВ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_ \_\_\_\_\_\_20\_\_\_ г.

УРБАНОВИЧ АЛЕКСАНДР ВЛАДИМИРОВИЧ

**WEB-ПРИЛОЖЕНИЕ ДЛЯ СОКРАЩЕНИЯ URL**

Дипломныйпроект слушателя

вечерней формы получения образования

Специальность 1-40 01 73 «Программное обеспечение информационных систем»

**Исполнитель:**

Слушатель группы № 16-06 П \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Урбанович А.В.

**Научный руководитель:**

старший преподаватель

кафедры СТОВ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Галицкая А.О.

Гродно, 2018

**РЕФЕРАТ**

**Дипломный проект** – web-приложение для сокращения url.

**Ключевые слова** – приложение, java, hibernate, spring, postgres, jsp.

**Объект исследования** – процесс создания web приложения.

**Предмет исследования** – приложение по сокращению URL ссылок.

**Цель дипломного проекта** – создать приложение по сокращению URL ссылок, для корпоративного использования.

**Методы исследования** – были использованы следующие методы исследования: анализ литературы по проблеме исследования, структурные методы анализа данных, инфологическое, даталогическое и физическое моделирование информационных систем.

**Авторская характеристика работы.** В результате работы над дипломным проектом, были изучены подобные приложения и учтены все требования ТЗ.

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ………………………………………………………………………4

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И СОЗДАНИЕ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ……………………………………………………………………..…..5

1.1. Описание предметной области………………………………………5

1.2. Обзор существующих решений……………….……………………..6

1.3. Концептуальная модель……………………….……………………..6

Выводы по главе 1………………………………………………………...7

2. ВЫБОР СРЕБСТВ СОЗДАНИЯ ПРИЛОЖЕНИЯ…………………………..8

2.1. Технологии для разработки web-приложений………………...……8

2.2. Проектирование web-приложения………………………………….17

2.3. Среда разработки………………………………………………

Выводы по главе 2……………………………………………………….

3. ПРОГРАМНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

3.1. Определение структуры приложения

Выводы по главе 3

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

**ВВЕДЕНИЕ**

В нашем мире всё больше растет потребность в совершенствовании технологий и программного обеспечения к ним, а также основой этого совершенствования является потребность общаться с людьми из далеких уголков земли, рассчитывать траекторию полета самолета, исследовать космос так же не возможно без цифровых технологий и т.д. Мир цифровых технологий позволит вам не только услышать собеседника, но и увидеть. Вы можете сидя дома посмотреть, изучить различного рода информацию при помощи Интернета.

Сейчас в нашем мире очень важна техническая грамотность и знание компьютера в целом.

Данный программный продукт будет представлять собой web-приложение для сокращения url, который позволит более эффективно распространять информацию на различных информационных ресурсах.

**Цель дипломного проекта -** создание web-приложения для сокращения url. Данное приложение должно помочь в продвижении различных товаров торговых компаний и любой другой информации.

**Задачи дипломного проекта:**

1. Исследование приложений-конкурентов.

2. Построение модели приложения.

3. Анализ методов решения поставленной цели.

4. Выбор средств и технологий разработки.

5. Программная реализация проекта.

**1.АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И СОЗДАНИЕ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ**

**1.1 Описание предметной области**

Данное приложение разрабатывается как альтернатива известным web-сервисам по сокращению url ссылок. Преимущества данного приложения заключается в том, что оно будет обладать разграниченным доступом, что позволит обезопасить данные. Так же в отличии от конкурентов, приложение будет обладать возможностью просмотра всех url и возможностью удаления не актуальных url.

Далее необходимо определить функции приложения.

**Назначение web-приложения.** Разрабатываемое web-приложение должно обеспечивать решение следующих задач:

* Добавление новых ссылок в приложение.
* Просмотр всех имеющихся ссылок в приложении.
* Удаление не актуальных ссылок.

**Требования к приложению.** При разработке web-приложения должны учитываться следующие параметры:

* Приложение должно иметь ограничения доступа для разных групп пользователей.
* Приложение должно быть интуитивно понятным.
* Приложение не должно быть перегружено графическими решениями.

Опираясь на эту информацию, можно приступить к разработке концепции и предполагаемого варианта проектируемого web-приложения. Список задач, в свою очередь дает возможность сформировать перечень путей их решения и привлекаемых для этого ресурсов.

**1.2 Обзор существующих решений**

Когда возникает необходимость создать web-приложение, неминуемо возникает проблема выбора языка, на котором будет написано приложение, а также инструментального средства разработки.

Из всего многообразия языков программирования был выбран язык Java, как язык занимающий 2-е место по популярности в мире. В качестве среды разработки я выбрал Intellij IDEA, а используемая БД PostgreSQL.

Все выше перечисленное было выбрано мною из-за удобства и простоты использования, что подтвердилось при написании уже ни одного проекта.

Так же было изучено приложение-конкурент goo.gl. Это приложение имеет приятный дизайн и интуитивно понятно, но у него отсутствует русский язык и нет разграничения по правам доступа, что делает его не пригодным для применения в качестве приложения для торговых магазинов и их сайтов в русскоговорящем регионе.

**1.3 Концептуальная модель**

Разработку любого приложения можно сравнить с моделированием из конструктора. Но происходит удивительная вещь – элементы конструктора одни и те же, а в результате их использования в одном случае получается полезное и интересное приложение, а в другом – вещь неизвестного назначения.

Концептуальная модель web-приложения – это предварительное определение целевой аудитории, анализ, сбор и документирование требований пользователей и выработка способов их реализации. При разработке важно, чтобы приложение было ориентировано на конкретных пользователей и отвечало их требованиям.

Целевой аудиторией данного приложения являются торговые площадки продвигающие свои товары, а также любой кому необходимо разместить ссылку на свой ресурс или конкретную страницу на которой описаны условия акции или какая-любо другая важная информация, но полный url адрес выглядит слишком длинным и не привлекательным с точки зрения маркетинга.

**Выводы по главе 1**

В соответствии с поставленной задачей проведен анализ предметной области. Выполнен обзор существующих решений. Кроме того, описана концептуальная модель разрабатываемого приложения.

**2. ВЫБОР СРЕДСТВ СОЗДАНИЯ ПРИЛОЖЕНИЯ**

**2.1 Технологии для разработки web-приложений**

Способы разработки web-приложений могут быть разделены на три большие категории: расширение web-сервера; подходы, основанные на использовании шаблонов web-страниц, включающих вставки кода скриптов и специальных серверных тэгов; объектные среды (каркасы, фреймворки).

Хотя между этими категориями и имеются пересечения, большинство широко известных подходов связаны с одной конкретной категорией.

Рассмотрим программные подходы. В данном походе web-приложение является внешняя программа, составленная на некотором универсальном языке программирования высокого уровня (например, таком как Java или C++) или скрипт, составленный с помощью скриптового языка, выполнение которого производится также с помощью внешней программы – интерпретатора скриптов. Основной проблемой с программным подходом к разработке web-приложений является их ориентация на написание кода. Разметка HTML и другие конструкции форматирования встраиваются в логику работы программы с помощью операторов вывода.

Это ограничивает возможности web-дизайнеров вносить свой вклад в оформление создаваемой приложением страницы. Web-дизайнер может разрабатывать макет страницы, а программист должен затем преобразовать его в код и связать со скриптом или программой. Для изменения практически любого элемента формируемой страницы требуется вмешательство программиста, касается ли это изменения логики работы программы, либо изменения оформления и расположения элементов страницы.

Простейший способ динамически формировать web-страницы в ответ на HTTP запрос заключается в том, чтобы передать работу по решению требуемой задачи и формированию HTML страницы внешней программе, которая должна получать переданные в HTTP запросе входные параметры с сформировать выходную страницу на языке HTML.

Первой широко используемой, независимой от типа web-сервера, программной технологией создания и выполнения web-приложений была технология Common Gateway Interface (CGI, общий шлюзовой интерфейс). Она определяет набор правил, которым должна следовать программа, чтобы она могла выполняться на разных HTTP серверах и операционных системах.

В соответствии с CGI технологией при поступлении на web-сервер HTTP запроса, который включает ссылку не на статическую страницу, а на CGI программу (например: program.exe), создается новый процесс, в котором запускается требуемая прикладная программа. Технология CGI задает способ передачи такой программе параметров, входящих в состав HTTP запроса. Передача входных данных может выполняться либо с помощью фиксированного набора переменных среды, которые могут создаваться одно программой и использоваться другими программами, либо через входные данные функции, с которой начинается работа программы, а результаты работы программы возвращаются с помощью стандартного потока вывода STDOUT.

Технология CGI позволяет использовать любой язык программирования, который может работать со стандартными устройствами ввода/вывода. Кроме того, CGI программы можно писать с использованием скриптовых языков, которые называются «CGI скриптами». Примерами скриптовых CGI языков являются, например, Perl, Python или Tcl. При использовании скрипта web-приложение вызывает на выполнение внешнюю программу – интерпретатор скриптов, который передает данные HTTP запроса и имя файла, в котором содержится запрашиваемые пользователем скрипт. А затем данная программа выполняет указанный скрипт и возвращает серверу сформированную HTML страницу.

Технология CGI является достаточно простым способом динамически формировать информацию в web-сети, но она имеет существенные недостатки, которые делают ее не практичной в большинстве случаев:

* основной проблемой является производительность: для каждого HTTP запроса к CGI программе web-сервер запускает новый процесс, который заканчивает работу только после завершения программы. Работа по созданию и завершению процессов является достаточно трудоемкой, что может очень быстро понизить производительность системы, кроме этого различные активные процессы начинают конкурировать за системные ресурсы, такие как оперативная память.
* для составления и отладки CGI программ разработчик должен обладать достаточно большим опытом программирования на одном из языков, на которых можно программировать CGI программы.
* в CGI программах программный код и код разметки полностью перемешаны. Дизайнер должен знать программирование, что бы менять структуру web-страниц.

Попытки объединить переносимость CGI приложений с эффективностью является технология FastCGI. Данная технология основывается на простой идее: вместо необходимости каждый раз запускать новый процесс для обработки CGI скрипта, FastCGI позволяет не закрывать процессы, связанные с CGI скриптами, после окончания обработки, а использовать их для обработки новых запросов к CGI программам. А это означает, что не требуется постоянно запускать и удалять новые процессы, так как один и тот же процесс может использоваться многократно для обработки запросов. Такие процессы могут инициализироваться только один раз при их создании.

Модули сервера, которые выполняют функциональность FastCGI, взаимодействуют с HTTP сервером с помощью своих собственных API. Эти API стараются скрыть детали реализации и конфигурирования от FastCGI приложений, но разработчики все равно должны знать особенности реализвции технологии FastCGI, так как модули различных типов серверов не совместимы между собой.

Недостатки технологии CGI можно также преодолеть путем расширения возможностей web-серверов с помощью специальных компонентов. Используя такие расширения, программы, формирующие HTTP ответы, могут выполняться более эффективно, без необходимости их завершения после обработки каждого запроса и за счет использования общих ресурсов несколькими приложениями. Такие технологии обычно предоставляют возможность хранить в основной памяти данные сеансов работы пользователей, которые взаимодействуют с приложением в течении большого числа HTTP запросов.

Другой широко используемой технологией расширения архитектуры web-сервера является прикладной интерфейс Java Servlet API, который связывает web-сервер с виртуальной машиной Java Virtual Machine (JVM). Виртуальная машина JVM поддерживает выполнение специальной Java программы (контейнеров сервлетов), которая отвечает за управление данными сеанса работы и выполнения Java-сервлетов.

Сервлеты это специальные классы на языке Java, которые имеют доступ к информации из HTTPзапросов. Они формируют HTTP ответы, которые возвращаются браузерам. Контейнер сервлетов отвечает за получение от web-сервера HTTP запросов на выполнение сервлетов; создание сеанса работы пользователя, если это требуется; вызов сервлета связанного с HTTP запросом; передачу сервлету параметров, которые в HTTP запросе, представленных в виде Java объектов. В отличии от ISAPI расширений, технология Servlet API является переносимой между разными web-серверами, операционными системами и компьютерными платформами. Сервлеты выполняются одинаково в любой среде, которая предоставляет совместимый с ним контейнер сервлетов. Технология Servlet API используется большим количество разработчиков и поддерживается многими известными web серверами.

Подходы, основанные на шаблонах (template approaches, шаблонные подходы) используют в качестве адресуемых объектов (имеющий URL-адрес) не программы или скрипта, а «шаблоны». По существу шаблоны являются HTML файлами с дополнительными «тэгами», которые задают методы включения динамически формируемого контента. Таким образом, файл шаблона содержит HTML код, который описывает общую структуру страницы, и дополнительные серверные тэги, размещенные таким образом, чтобы формируемое с их помощью содержание страницы имело требуемый вид.

В настоящее время к наиболее распространенным технологиям разработки web-приложений на основе шаблонов, относятся следующие: Server-Side Includes (SSI), Cold Fusion, PHP, Active Server Pages (ASP) и Java Server Pages (JSP).

Другой достаточно популярной технологией, основанной на шаблонах, является технология Cold Fusion, разработанная компанией Adobe. Преимущество данного подхода заключается в том, что данный шаблон может создавать и поддерживаться дизайнером страницы, который имеет базовые знания языка HTML и web-графики, но не имеет опыта программирования. Специальные тэги, которые являются «расширением» HTML.

Технология «PHP Hypertext Preprocessor» или просто PHP позволяет разработчикам встраивать программный код в шаблоны, с помощью языка, сходного с языком скриптов Perl.

Компания Microsoft разработала технологию ASP (Active Server Pages), которая объединила возможности создания шаблонов, включающих скрипты, с доступом к набору OLE и COM объектов, имеющихся в операционной системе Windows, в том числе и к ODBC источникам данных. Данная технология, объединенная с бесплатным web сервером Internet Information Server (IIS), быстро стала популярной среди программистов, использующих Visual Basic, которые оценили возможность использования в шаблонах языка VBScript. Как и PHP шаблоны, ASP страницы могут включать блоки скриптов, вперемешку с HTML форматированием. В отличии от таких объектно-ориентированных языков, как Java или C++, язык, используемый в ASP страницах, был плоским, линейным и строго процедурным.

В отличии от технологии PHP, ASP не связан с одним конкретным скриптовым языком. В ASP в качестве стандартного языка используется язык Visual Basic Scripting Edition (VBScript), но может использоваться и язык JavaScript.

В ASP шаблоны могут включаться блоки, выделенные с помощью тэгов <%...%>, которые содержат код скрипта, выполняемый интерпретатором ASP шаблонов, при формировании ответа. HTML разметка, находится вне таких блоков, рассматривается как исходный HTML код и просто переписывается в формируемую HTML страницу. Кроме этого в начало шаблона могут добавляться директивы страницы, такие, как например, <% @LANGUAGE = VBScript %>, которая информирует систему об используемом скриптовом языке.

Обычные скриптовые технологии на стороне сервера используют обычные объекты, но не позволяют разрабатывать и использовать собственные классы и создавать на их основе объекты. В связи с этим дальнейшее развитие web-технологий было связано с созданием специальных объектно-ориентированных технологий разработки web-приложений. Использование данных технологий позволяет сделать разработку web-приложений более сходной с разработкой объектно-ориентированного программного обеспечения.

Объектные среды представляют собой следующий уровень совершенствования разработки web-приложений. Вместо объединения разметки и логики в единый модуль, объектные среды поддерживают принцип отделения содержания от представления. Модули ответственные за создание контента отделяются от модулей, которые показывают это содержание в конкретном формате.

В настоящее время есть два подхода к созданию объектно-ориентированных web-приложений:

* подходы, основанные на наборе специальных web-страниц, связанных с описаниями классов, объекты которых будут создаваться и использоваться при вызове;
* подходы, основанные на использовании наборов классов, соответствующих шаблону Model-View-Controller (MVC).

В соответствии с архитектурным шаблоном MVC все классы, составляющие приложение делятся на три основные группы (компоненты): модель (model), представление (view) и контроллер (controller). Каждый из этих компонентов отвечает за свои задачи:

* модель (model) – это набор классов, реализующий всю бизнес-логику web-приложения. Эти классы отвечают за обработку данных, размещение их в БД, чтение из БД, а также за взаимодействие между объектами, составляющими такие данные;
* представление (view) – набор классов, отвечающих за интерфейс взаимодействия с пользователями (user interface, UI). С их помощью формируются HTML страницы, показывающие пользователям данные. Представление использует данные из модели и предоставляет пользователям возможность выполнять их редактирование;
* контроллер (controller) – это связующее звено между первыми двумя компонентами. Классы данного компонента получают данные о запросе к серверу и передают их в модель для обработки и сохранения. После обработки полученных данных контроллер выбирает, каким способом показать их клиенту с помощью некоторого класса из представления.

В результате такого разделения web-приложения на компоненты, разработчик получает полный контроль над формируемым HTML документом, упрощается структура приложения, облегчается задача выполнения тестирования приложения, достигается полное отделение логики работы приложения от представления данных.

Примерами технологий разработки на основе MVC являются:

* технология Strurs (основанная на языке Java);
* технология ASP.Net MVC, входящая в состав набора технологий ASP.Net платформы .Net Framework;
* технология Ruby on Rails (Rubi – язык программирования, а Rails – фреймворк, использующий данный язык) особенно популярный в последнее время.

Основной целью архитектора программного обеспечения при проектировании web-приложений является максимальное упрощение их структуры путем разделения задач на функциональные области, обеспечивая при этом безопасность и высокую производительность. Для эффективной работы web-приложений в обычных для них сценариях, необходимо:

* выполнить логическое разделение функциональности приложения, используя многослойную структуру для логического разделения приложения на слои представления, бизнес-слой и слой доступа к данным. Это помогает создать удобный в обслуживании код и позволит отслеживать и оптимизировать производительность каждого слоя в отдельности. Четкое логическое разделение также обеспечивает более широкие возможности масштабирования приложения;
* использовать абстракцию для реализации слабого связывания между слоями. Этот подход можно реализовать с помощью интерфейсных типов или абстрактных базовых классов, можно определить совместно используемую абстракцию, которая должна быть реализована интерфейсными компонентами;
* определиться, как будет реализовано взаимодействие компонентов друг с другом. Для этого необходимо понимать сценарии развертывания, которые должно поддерживать приложение;
* использовать кэширование для сокращения количества сетевых вызовов и обращений к базе данных;
* использовать протоколирование и инструментирование. Необходимо выполнять аудит и протоколирование действий в слоях и уровнях приложения. Журналы регистрации событий могут использоваться для выявления подозрительных действий, что часто обеспечивает ранее обнаружение атак на систему;
* продумать аспекты аутентификации пользователей на границах доверия. При проектировании приложения необходимо предусмотреть аутентификацию пользователей при пересечении границ доверия, например, при доступе к удаленному бизнес-слою из слоя представления;
* не передавать важные конфиденциальные данные по сети в виде открытого текста. Если требуется передавать по сети конфиденциальные данные, такие как пароль или куки аутентификации, использовать для этого шифрование и подписи данных либо шифрование с использованием протокола Secure Sockets Layer (SSL);
* Проектировать web-приложение для выполнения с менее привилегированной учетной записью. Процесс должен иметь ограниченный доступ к файловой системе и другим ресурсам системы. Это позволит максимально сократить возможные негативные последствия на случай, если злоумышленник попытается взять процесс под свой контроль.

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям проектирования: обработка запросов приложения, аутентификация, кэширование, управление исключениями, протоколирование и инструментирование, навигация, компоновка страницы, формирование визуального отображения страницы, управление сеансами, проверка введенных данных (валидация).

**2.2 Проектирование web-приложения**

Для создания приложения будут использоваться разные фреймворки и библиотеки.

Apache Maven - [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для [автоматизации сборки](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%81%D0%B1%D0%BE%D1%80%D0%BA%D0%B8) проектов на основе описания их структуры в файлах на языке [POM](https://en.wikipedia.org/wiki/Project_Object_Model), являющемся подмножеством [XML](https://ru.wikipedia.org/wiki/XML). Проект Maven издаётся сообществом Apache Software Foundation, где формально является частью [Jakarta Project](https://ru.wikipedia.org/wiki/Apache_Jakarta_Project" \o "Apache Jakarta Project).

Название программы, maven, — является словом из языка [идиш](https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B8%D1%88), смысл которого можно примерно выразить как «собиратель знания».

Maven обеспечивает [декларативную](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), а не [императивную](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) (в отличие от средства автоматизации сборки [Apache Ant](https://ru.wikipedia.org/wiki/Apache_Ant" \o "Apache Ant)) сборку проекта. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения. Все задачи по обработке файлов, описанные в спецификации, Maven выполняет посредством их обработки последовательностью встроенных и внешних плагинов.

Maven используется для построения и управления проектами, написанными на [Java](https://ru.wikipedia.org/wiki/Java" \o "Java), [C#](https://ru.wikipedia.org/wiki/C_Sharp), [Ruby](https://ru.wikipedia.org/wiki/Ruby" \o "Ruby), [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), и других языках.

Maven был создан канадцем Джейсоном ван Зилом (Jason van Zyl) и организованной им фирмой Sonatype. Он начался как подпроект Apache Turbine в 2002 г.. В 2003 году Maven был квалифицирован как Apache-проект верхнего уровня, тогда же появилась его первая версия — Maven 1.x. Она была опубликована 13 июля 2004 как версия 1.0. Это происходило, однако, так быстро, что некоторые частности оказались непродуманными. Например, слишком много конфигурации, проблемы с производительностью.

Поэтому концепция была доработана и с 2005 года началась параллельная разработка Maven 2.x, которая была сдана в версии 2.0 19 октября 2005 года.

Maven 1.x не разрабатывается дальше и ограничивается поддержкой пользователей и устранением ошибок.

Разработка Maven 3.0 началась в 2008 г.. После восьми альфа-релизов, первая бета-версия Maven 3.0 была опубликована в октябре 2010 г. Особенное внимание было уделено её обратной совместимости с Maven 2. Для большинства проектов переход от версии Maven 2 к версии Maven 3 не требует никаких изменений.

Информация для сборки проекта, поддерживаемого Apache Maven, содержится в [XML](https://ru.wikipedia.org/wiki/Extensible_Markup_Language)-файле с названием pom.xml. При запуске Maven проверяет, содержит ли конфигурационный файл все необходимые данные и все ли данные синтаксически правильно записаны.

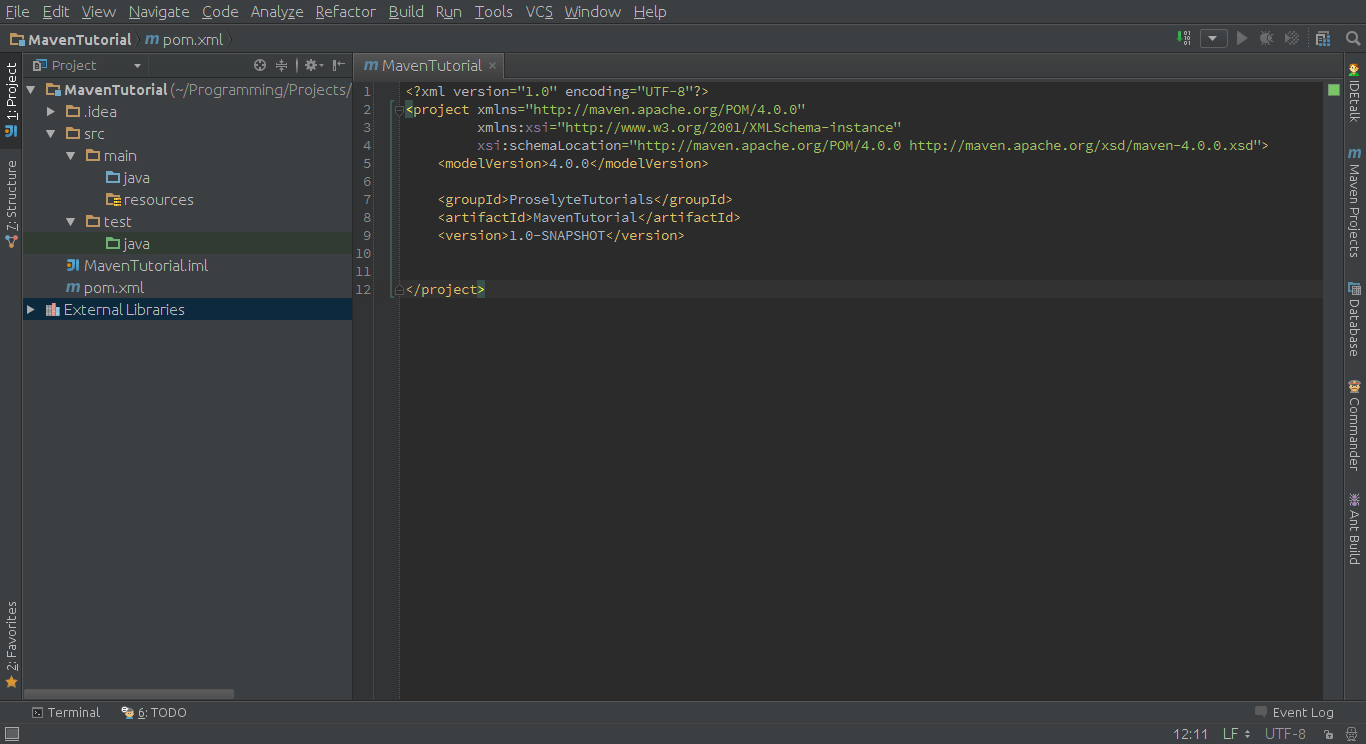


Рисунок 2.1 – Пример pom.xml

Минимальная конфигурация включает версию конфигурационного файла, имя проекта, его автора и версию. С помощью pom.xml конфигурируются зависимости от других проектов, индивидуальные фазы процесса построения проекта (build process), список [плагинов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD), реализующих порядок сборки.

Крупные проекты могут быть поделены на несколько модулей, или подпроектов, каждый со своим собственным POM. Операции над модулями могут выполняться через общий корневой POM единой командой.

POM файлы подпроектов могут наследовать конфигурацию от других файлов конфигурации. В то же время все файлы конфигурации обязательно наследуются от «Супер POM» файла по умолчанию. Супер POM обеспечивает конфигурацию по умолчанию, например, стандартная структура каталогов, используемые по умолчанию плагины, привязка к фазам жизненного цикла и прочее.

Жизненный цикл Maven проекта — это список поименованных фаз, определяющий порядок действий при его построении. Жизненный цикл Maven содержит три независимых порядка выполнения:[]](https://ru.wikipedia.org/wiki/Apache_Maven#cite_note-15)

* clean — жизненный цикл для очистки проекта. Содержит следующие фазы:
  1. pre-clean
  2. clean
  3. post-clean
* default — основной жизненный цикл, содержащий следующие фазы:
  1. validate - выполняется проверка, является ли структура проекта полной и правильной.
  2. generate-sources
  3. process-sources
  4. generate-resources
  5. process-resources
  6. compile - компилируются исходные тексты.
  7. process-test-sources
  8. process-test-resources
  9. test-compile
  10. test - собранный код тестируется заранее подготовленным набором тестов.
  11. package - упаковка откомпилированных классов и прочих ресурсов. Например, в JAR-файл.
  12. integration-test - программное обеспечение в целом или его крупные модули подвергаются интеграционному тестированию. Проверяется взаимодействие между составными частями программного продукта.
  13. install - установка программного обеспечения в локальный Maven-репозиторий, чтобы сделать его доступным для других проектов текущего пользователя.
  14. deploy - стабильная версия программного обеспечения распространяется на удаленный Maven-репозиторий, чтобы сделать его доступным для других пользователей.
* site — жизненный цикл генерации проектной документации. Состоит из фаз:
  1. pre-site
  2. site
  3. post-site
  4. site-deploy

Стандартные жизненные циклы могут быть дополнены функционалом с помощью Maven-плагинов. Плагины позволяют вставлять в стандартный цикл новые шаги (например, распределение на сервер приложений) или расширять существующие шаги.

Maven базируется на [plugin](https://ru.wikipedia.org/wiki/Plugin" \o "Plugin)-архитектуре, которая позволяет применять плагины для различных задач (compile, test, build, deploy, checkstyle, pmd, scp-transfer) для данного проекта, без необходимости их в явном виде инсталлировать. Это возможно за счет того, что информация поступает плагину через стандартный вход, а результаты пишутся в его стандартный выход. Теоретически, это позволяет кому угодно писать плагины для взаимодействия со средствами построения проекта (компиляторы, средства тестирования, и т. п.) для любого другого языка. В реальности, поддержка других языков кроме Java сейчас минимальна.

Количество доступных плагинов в настоящее время очень велико и включает, в том числе, плагины, позволяющие непосредственно из Maven запускать web-приложение для тестирования его в браузере; плагины, позволяющие тестировать или создавать банки данных; плагины, позволяющие генерировать Web Services. Задачей разработчика в такой ситуации является найти и применить наиболее подходящий набор плагинов.

В файле pom.xml задаются зависимости, которые имеет управляемый с помощью Maven проект. Менеджер зависимостей основан на нескольких основных принципах:

* Репозитории. Maven ищет необходимые файлы в локальных каталогах или в локальном Maven-репозитории. Если зависимость не может быть локально разрешена, Maven подключается к указанному Maven-репозиторию в сети и копирует в локальный репозиторий. По умолчанию Maven использует Maven Central Repository, но разработчик может конфигурировать и другие публичные Maven-репозитории, такие, как Apache, Ibiblio, Codehaus или Java.Net.
* Транзитивные зависимости. Необходимые библиотеки подгружаются в проект автоматически. При разрешении конфликта версий используется принцип «ближайшей» зависимости, то есть выбирается зависимость, путь к которой через список зависимых проектов является наиболее коротким.
* Исключение зависимостей. Файл описания проекта предусматривает возможность исключить зависимость в случае обнаружения цикличности или отсутствия необходимости в определённой библиотеке.
* Поиск зависимостей. Поиск зависимостей (open-source-библиотек и модулей) ведётся по их координатам (groupId, artifactId и version). Эти координаты могут быть определены с помощью специальных поисковых машин, например, Maven search engine. Например, по поисковому признаку «pop3» поисковая машина предоставляет результат с groupId="com.sun.mail" и artifactId="pop3".
* Менеджеры репозиториев. Репозитории реализуются с помощью менеджеров репозиториев Maven (Maven Repository Manager), таких как Apache Archiva, Nexus (ранее Proximity), Artifactory, Codehaus Maven Proxy или Dead Simple Maven Proxy.

Область распространения зависимости позволяет включать зависимости только на определённую стадию построения проекта. Существует 6 возможных областей:

1. compile. Область по умолчанию. Зависимость доступна во всех путях поиска классов в проекте. Распространяется на зависимые проекты.
2. provided. Область аналогична compile, за исключением того, что JDK или контейнер сам предоставит зависимость во время выполнения программы.
3. runtime. Зависимость не нужна для компиляции, но нужна для выполнения.
4. test. Зависимость не нужна для нормальной работы приложения, а нужна только для компиляции и запуска тестов.
5. system. Область аналогична provided за исключением того, что содержащий зависимость JAR указывается явно. Артефакт не ищется в репозитории.
6. import (начиная с версии Maven 2.0.9) используется только с зависимостью типа pom в секции <dependencyManagement>. Зависимости текущего POM заменяются на зависимости из указанного POM.

Библиотека Hibernate, используемая для разработки Java приложений, предназначена для решения задач объектно-реляционного отображения ORM (object-relational mapping). Hibernate представляет собой свободное программное обеспечение с открытым исходным кодом, распространяемое на условиях GNU Lesser General Public License.

Для использования Hibernate необходимо создать легкий в использовании каркас (фреймворк), отображающий объектно-ориентированную модель данных в традиционные реляционные базы данных. Подключить библиотеку к приложению можно как в процессе проектирования java классов и sql таблиц «с нуля», так и при работе с уже существующей базой данных.

Hibernate обеспечивает связь между Java классами и таблицами базы данных, соответствие типов данных Java с типами данных SQL. Также библиотека предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных. Таким образом, при использовании Hibernate можно значительно уменьшить время разработки, связанное с ручным написанием SQL-запросов JDBC-кода.

Одним из основных достоинств библиотеки Hibernate является автоматическая генерация SQL-запросов и обработка результирующего набора данных по преобразованию объектов, т.е. выполнение сериализации объектов. Таким образом максимально облегчается перенос (портирование) приложения на любые другие базы данных SQL. То есть, Hibernate обеспечивает прозрачную поддержку сохранности данных (persistence) для «POJO» (Plain Old Java Object). POJO класс содержит только поля, без дополнительной логики их обработки. Доступ ко всем полям такого класса осуществляется только через методы get/set. Пример простого класса POJO приведен на странице описания компонента [JavaBean](http://java-online.ru/jsp-javabean.xhtml" \l "pojo).

Для связи Java-классов с таблицами БД Hibernate использует Mapping (сопоставление, проецирование), которое осуществляется с помощью конфигурационных XML-файлов или Java-аннотаций, и обеспечивает возможности по организации отношения между классами «один-ко-многим» и «многие-ко-многим». В дополнение к управлению связями между объектами Hibernate может также управлять рефлексивными отношениями, где объект имеет связь «один-ко-многим» с другими экземплярами своего собственного типа данных.

Hibernate поддерживает использование обобщенных классов (generics), введеные в Java 5, и может быть настроен на «ленивые» (отложенные) загрузки [коллекций](http://java-online.ru/java-collection.xhtml), которые является вариантом, используемым по умолчанию.

Cвязанные объекты Hibernate позволяет настроить на каскадные операции. Например, родительский класс Album (музыкальный альбом) может быть настроен на каскадное сохранение и/или удаление своего потомка Track, что существенно сокращает время разработки и обеспечивает целостность данных. Функция проверки изменения данных (dirty checking) позволяет избежать ненужной записи действий в БД, выполняя SQL-обновление только при изменении полей персистентных (persist) объектов.

Описание Hibernate было бы неполным, если не сказать об использовании SQL-подобного языка Hibernate Query Language (HQL), который позволяет выполнять SQL-подобные запросы, определенные вместе с объектами данных Hibernate.

Hibernate можно использовать как в desktop'ных приложениях Java, так и в WEB разработках Java EE, выполняемых на сервере. Также он может быть использован как дополнительная возможность с другими языками программирования.

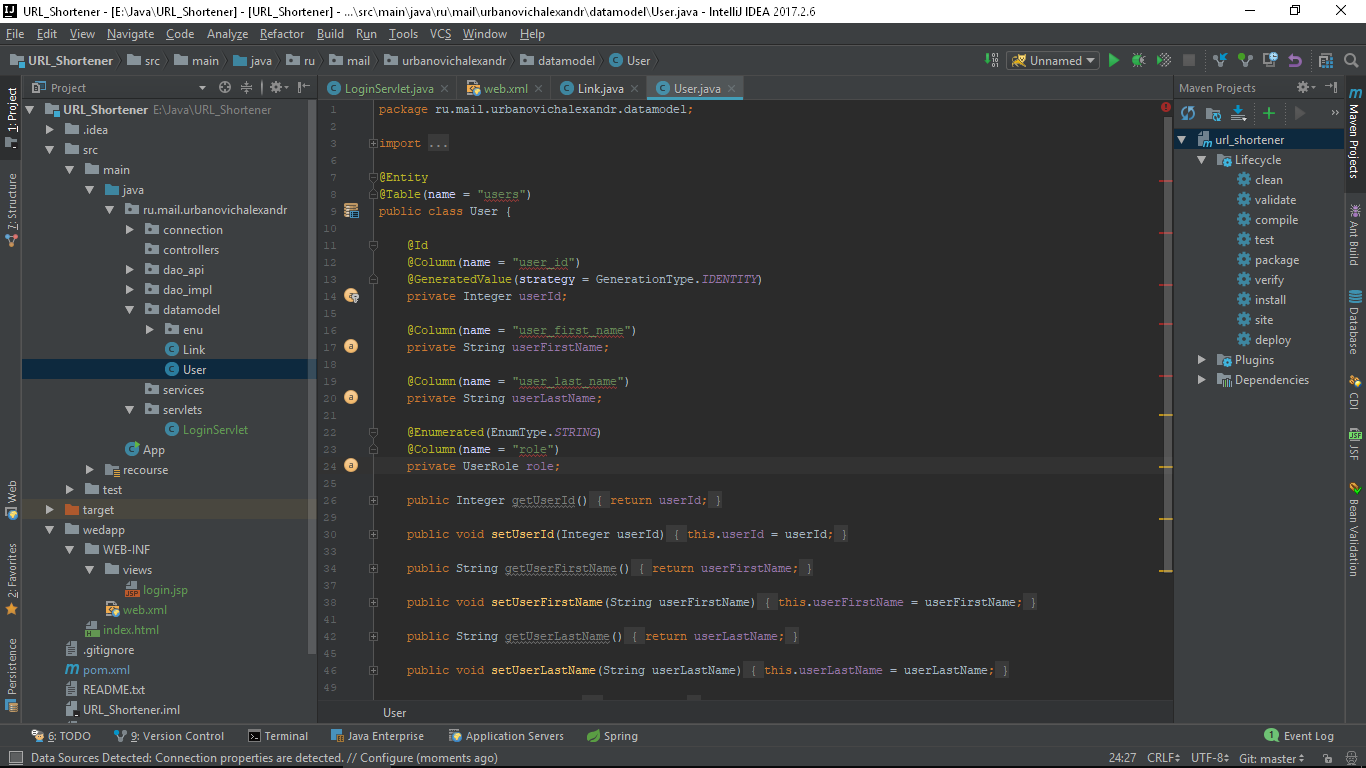


Рисунок 2.2 – Mapping по средствам Java-классов

Spring – универсальная платформа с открытым исходным кодом, которая предоставляет облегченное решение по созданию готовых корпоративных приложений с возможностью использования RMI и Web-служб, отправки сообщений по электронной почте, обработки данных в базе данных, декларативного управления транзакциями, предоставляет среду выполнения MVC, способы интеграции АОП и хорошо структурированную иерархию исключений. Spring был впервые выпущен в июне 2003 года и получил широкое распространение.

Плюсы Spring:

• Spring можно использовать для построение любого приложения на языке Java, что выгодно отличает его от многих других платформ (таких как Apache Struts);

• для использования ядра Spring нужно внести минимальные изменения в код приложения (принцип философии Spring – минимальное воздействие);

• Spring является модульной средой и позволяет использовать отдельные свои части без необходимости вводить остальные;

• Возможность работы с POJO (без контейнеров EJB);

• Существует большое количество расширений Spring для построения приложений на Java Enterprise платформе;

• Сообщество Spring – одно из лучших сообществ из всех проектов с открытым исходным кодом, списки рассылки и форумы всегда активны;

• У Spring отличная подробная документация;

• Spring активно развивается.

Возможности Spring:

• Использование внедрения зависимостей (DI);

• Поддержка аспектно-ориентированного программирования (в том числе интеграция с AspectJ);

• Язык выражений Spring (SpEL) – позволяет приложению манипулировать объектами Java во время выполнения;

• Встроенная поддержка Bean Validation API – позволяет один раз описать логику проверки достоверности данных и использовать ее как в пользовательском интерфейсе, так и на уровне работы с БД;

• Spring обеспечивает великолепную интеграцию с большинством инструментов доступа к данным (JDBC, Hibernate, MyBatis, JDO, JPA и т.п.);

• Поддержка Object to XML Mapping – преобразование компонентов JavaBean в XML и наоборот (как правило, используется для обмена данными с другими системами);

• Интеграция с JEE – внедрение бинов Spring в компоненты EJB;

• MVC на веб-уровне;

• Поддержка электронной почты;

• Поддержка планирования заданий;

• Поддержка динамических сценариев (Groovy, JRuby, BeanShell).

**Выводы по главе 2**

В главе произведен анализ и отбор технологий для создания web-приложения. Выбраны следующие средства: язык программирования Java, фреймворки: hiberhate, spring, maven.