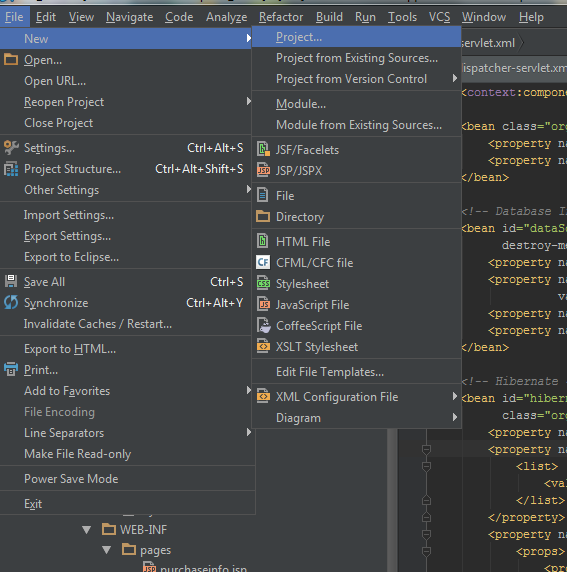
Лабораторная №1

Spring Framework.

Разработка WEB приложений на основе фреймворка **Spring MVC**.

Для начала необходимо создать проект в среде разработки (IntelliJ (желательно), NetBeans, Eclipse).



Для этого выбираем File->New->Project

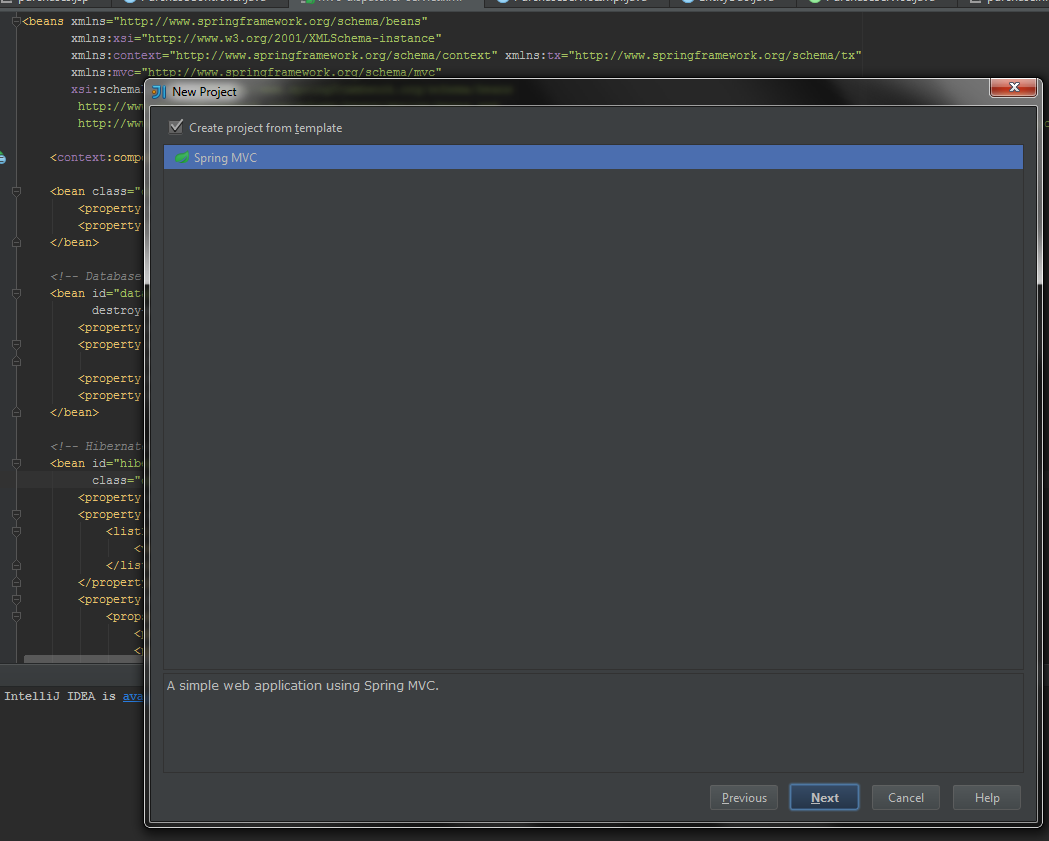
Выбираем Spring MVC.

# 

Далее среда разработки предложит создать проект на основе готового шаблона. Подтверждаем чек бокс, соглашаемся.

Если возникает проблема, что при создании через шаблон папка src пустая, то необходимо выполнить следующее:

File ---> Settings ---> Plugins ---> поставить галочку на против плагина Spring. В дальнейшем при создании проекта будет возможность выбрать Spring шаблон. Данная часть будет работать только если установить архетип с github. Иначе создавайте обычный Spring проект.



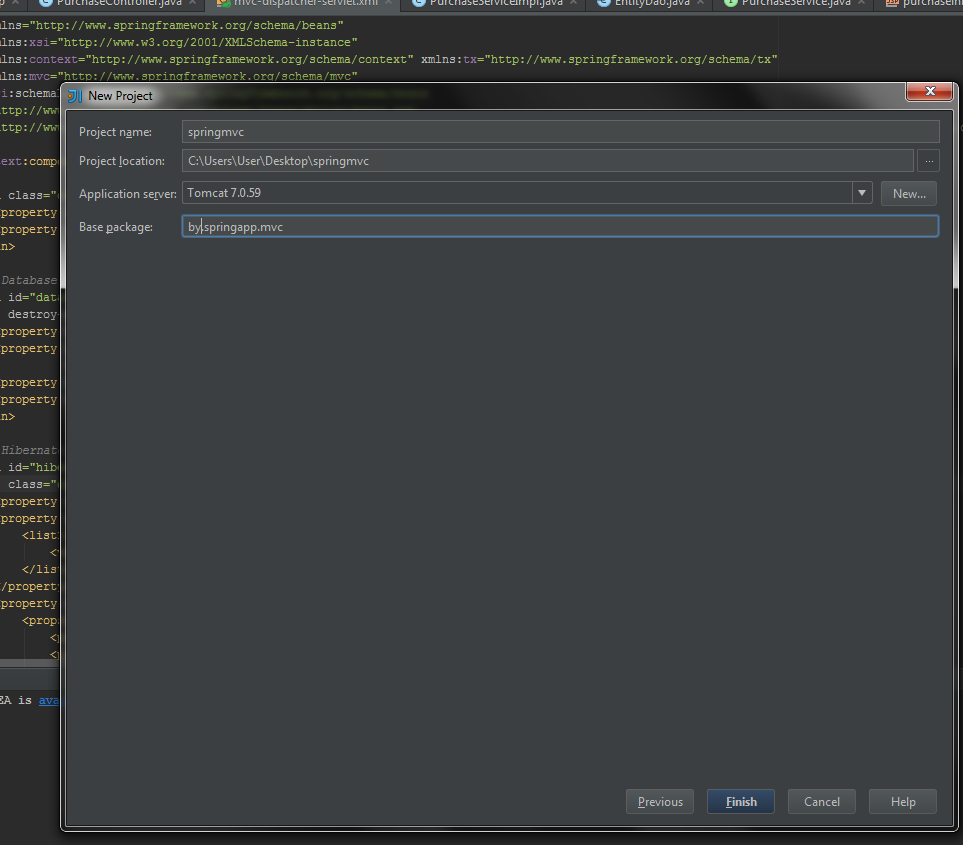
Переходим к следующему этапу.

Задаем: Project name – имя проекта.

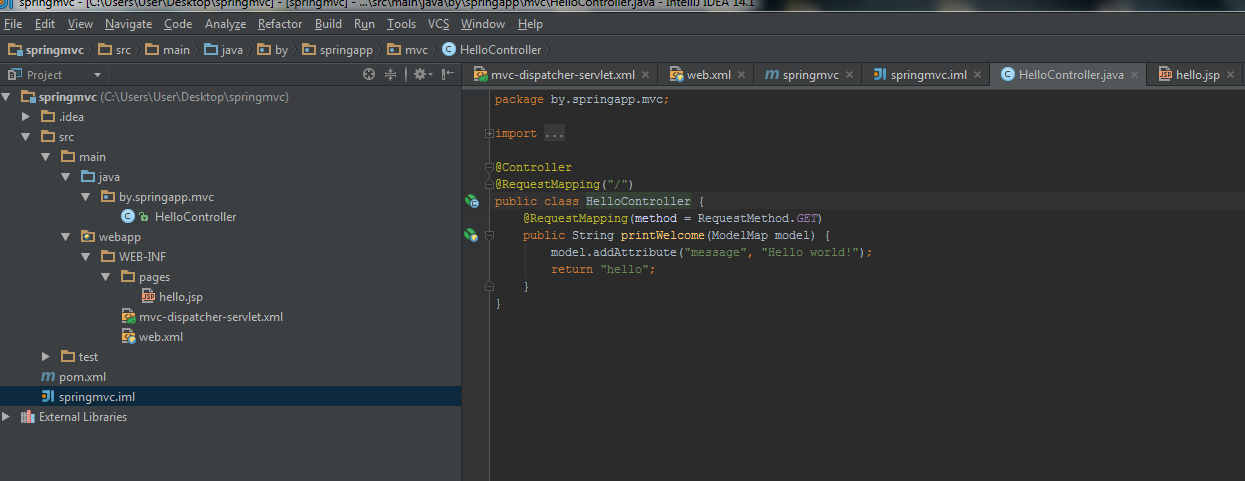
Корневой пакет.

Выбираем сервер, на котором впоследствии будет развернуто наше web приложение.

Подтверждаем -> Finish.



Далее должен появится следующий шаблон:



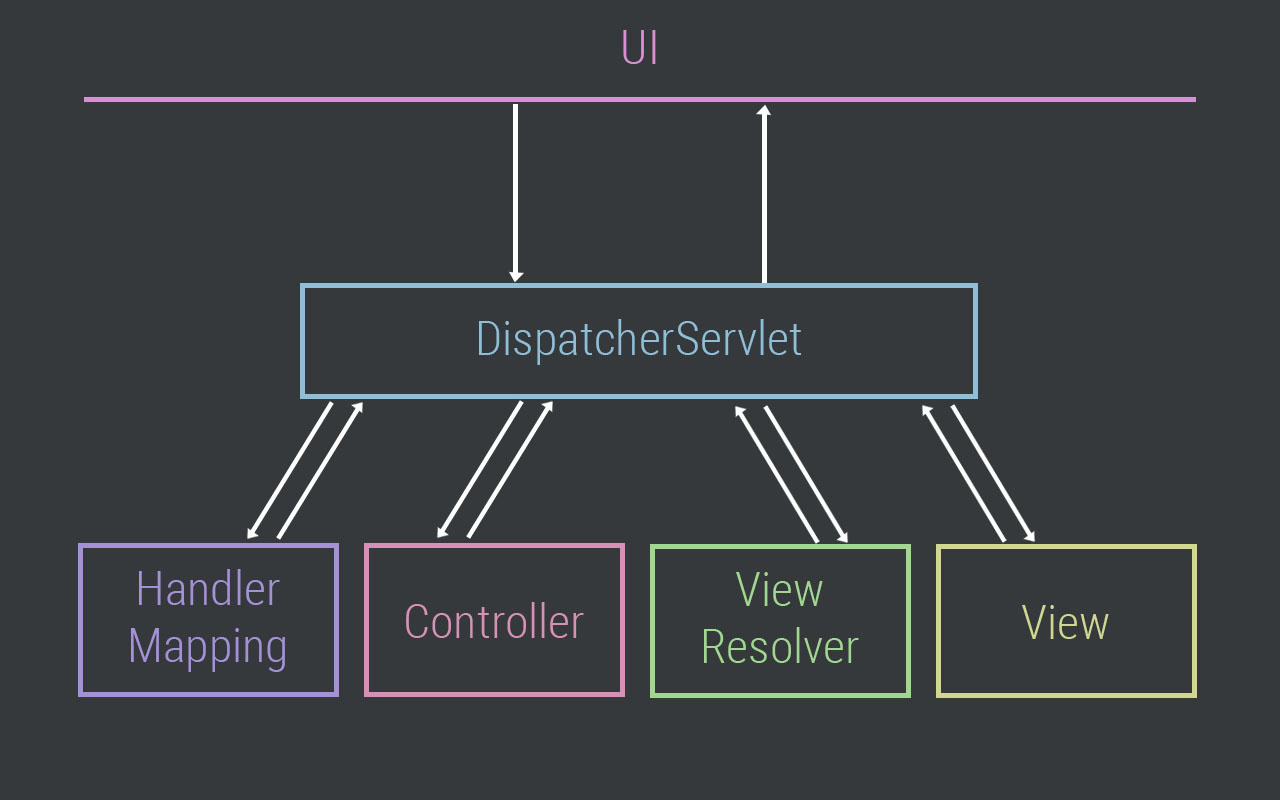
Пример приложения будет рассмотрен в теоретической части.

Теоритическая часть

# Spring MVC – основные принципы

Spring имеет собственную [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller" \o "Model-View-Controller) – платформу веб-приложений, которая не была первоначально запланирована. Разработчики Spring решили написать её как реакцию на то, что они восприняли как неудачность конструкции (тогда) популярного [Apache Struts](https://ru.wikipedia.org/wiki/Apache_Struts), а также других доступных веб-фреймворков. В частности, по их мнению, было недостаточным разделение между слоями представления и обработки запросов, а также между слоем обработки запросов и моделью.

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.



Класс DispatcherServlet является основным контроллером фрэймворка и отвечает за делегирование управления различным интерфейсам, на всех этапах выполнения HTTP-запроса. Об этих интерфейсах следует сказать более подробно.

Как и Struts, Spring MVC является фреймворком, ориентированным на запросы. В нем определены стратегические интерфейсы для всех функций современной запросно-ориентированной системы. Цель каждого интерфейса — быть простым и ясным, чтобы пользователям было легко его заново имплементировать, если они того пожелают. MVC прокладывает путь к более чистому front-end-коду. Все интерфейсы тесно связаны с Servlet API. Эта связь рассматривается некоторыми как неспособность разработчиков Spring предложить для веб-приложений абстракцию более высокого уровня. Однако эта связь оставляет особенности Servlet API доступными для разработчиков, облегчая все же работу с ним. Наиболее важные интерфейсы, определенные Spring MVC, перечислены ниже:

1. HandlerMapping: выбор класса и его метода, которые должны обработать данный входящий запрос на основе любого внутреннего или внешнего для этого запроса атрибута или состояния.
2. HandlerAdapter: вызов и выполнение выбранного метода обработки входящего запроса.
3. Controller: включен между Моделью (Model) и Представлением (View). Управляет процессом преобразования входящих запросов в адекватные ответы. Действует как ворота, направляющие всю поступающую информацию. Переключает поток информации из модели в представление и обратно.
4. View: ответственно за возвращение ответа клиенту в виде текстов и изображений. Некоторые запросы могут идти прямо во View, не заходя в Model; другие проходят через все три слоя.
5. ViewResolver: выбор, какое именно View должно быть показано клиенту.
6. HandlerInterceptor: перехват входящих запросов. Сопоставим, но не эквивалентен сервлет-фильтрам (использование не является обязательным и не контролируется DispatcherServlet-ом).
7. LocaleResolver: получение и, возможно, сохранение локальных настроек (язык, страна, часовой пояс) пользователя.
8. MultipartResolver: обеспечивает Upload — загрузку на сервер локальных файлов клиента.

Наглядный пример представлен на рисунке ниже.



Spring MVC предоставляет разработчику следующие возможности:

1. Ясное и прозрачное разделение между слоями в MVC и запросах.
2. Стратегия интерфейсов — каждый интерфейс делает только свою часть работы.
3. Интерфейс всегда может быть заменен альтернативной реализацией.
4. Интерфейсы тесно связаны с Servlet API.
5. Высокий уровень абстракции для веб-приложений.

В веб-приложениях можно использовать различные части Spring, а не только Spring MVC.

# Доступ к данным

Spring предоставляет свой слой доступа к базам данных и поддерживает все популярные СУБД.

JDBC, iBatis / MyBatis, Hibernate, JDO, JPA, Oracle TopLink, Apache OJB, Apache Cayenne и т. д.

Для всех этих фреймворков Spring предоставляет такие особенности:

1. Управление ресурсами — автоматическое получение и освобождение ресурсов базы данных
2. Обработка исключений — перевод исключений при доступе к данным в исключения Spring-а
3. Транзакционность — прозрачные транзакции в операциях с данными
4. Распаковка ресурсов — получение объектов базы данных из пула соединений
5. Абстракция для обработки BLOB и CLOB

# Управление транзакциями

Фреймворк управления транзакциями в Spring привносит механизм абстракций для платформы Java. Основные возможности этих абстракций:

1. работа с локальными и глобальными транзакциями
2. работа с вложенными транзакциями
3. работа с точками сохранения в транзакциях

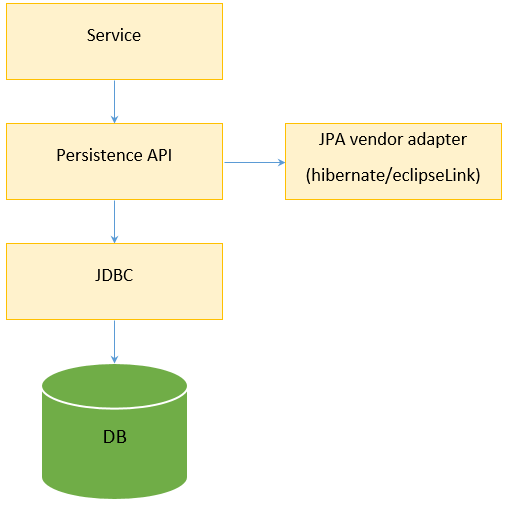
# Интеграция

Spring Integration — фреймворк для JEE, который предоставляет функции, необходимые для отправки сообщений или для построения событийно-ориентированной архитектуры: роутеры, трансформеры, адаптеры для интеграции с другими технологиями и системами (HTTP, AMQP, JMS, XMPP, SMTP, IMAP, FTP (FTPS/SFTP), файловые системы и т. д.), фильтры, активаторы сервисов, аудит и управление.

### **Что такое JPA и зачем он нужен**

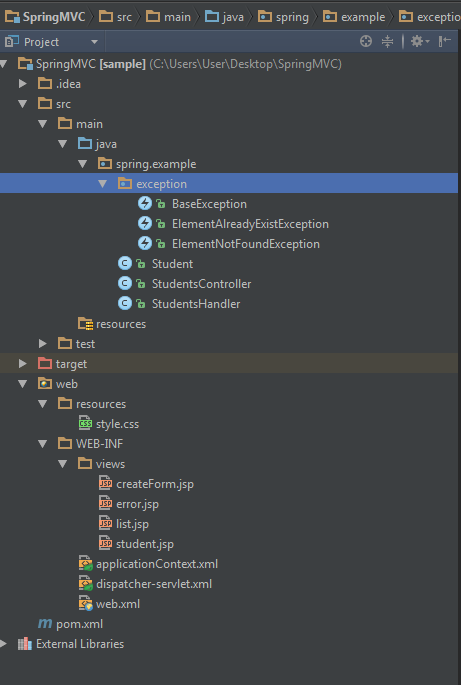
**JPA** это спецификация описанная в документе [JSR-317](http://download.oracle.com/otndocs/jcp/persistence-2.0-fr-oth-JSpec/) в котором определен общий набор интерфейсов, аннотаций и прочих служб, которые поставщики постоянства **JPA** (такие как **hibernate**) должны реализовать. Говоря человеческим языком есть интерфейсы, перечисленные в пакете **javax.persistence**, удовлетворяющие спецификации **JPA**, которые должны быть реализованы конкретным поставщиком постоянства, таким как **hibernate**.  
 **JPA это просто стандартный API-интерфейс доступа к данным**. В ранних версиях джавы (даже не джавы, а спецификации JEE) когда общего стандарта доступа к данным как такового не было, приходилось обращаться к базе данных через **JDBC**, но **JDBC** все таки не стандартный метод доступа к данным с точки зрения **API**, так как у каждой базы данных свой драйвер, синтаксис и пр. и тогда придумали **ORM-фреймворк**, который уже предоставлял свой интерфейс доступа к данным абстрагируясь от вендора базы данных. Но и тут были проблемы, например если в проекте по каким-либо причинам надо было заменить ORM-фреймворк на другой у которого другой API интерфейс, то приходилось менять всю DAO-часть на **API** другого фрэимворка. То есть все упиралось в **API**, на этот раз не самой базы данных, а ORM-фреимворка, но *пагромистам* от этого легче не становилось, так как приходилось переписывать код поддерживающий **API** другого ORM-фреймворка. Вот тогда было решено сделать общий интерфейс для всех ORM-фреимворков, то есть абстрагироваться пришлось еще на один лэвэл выше, уже не от поставщика базы данных, а от поставщика ORM-фреимворка, а сам ORM-фреимворк выступает в роли **лежащего в основе поставщика службы постоянства JPA**, который для *пограмиста* становился прозрачным. В итоге этот интерфейс получил называние **Java Persistence API – JPA** который является частью стэка технологий **JEE**.

Возьмем конкретный пример. Вот представим такую ситуацию, допустим у нас есть проект где используется **hibernate** для доступа к данным. Потом по какой-нибудь причине было решено заменить в проекте **hibernate** на **eclipseLink**. Это будет сделать не так и просто, так как придется переделать почти весь проект, выкинуть классы **Session**, добавить **EntityManager** и в результате приложение будет делать все то же самое, вытаскивать те же данные, только делать это будет через другой фреймворк. **JPA** позволяет избежать этих больших изменений в проекте и абстрагироваться от используемого фреймворка. С **JPA** нам нужно только прописать логику доступа к данным не заботясь какой будет поставщик постоянства, то есть мы работаем с базой данных всегда через **EntityManager**, а поставщик постоянства может быть как **hibernate** так и **eclipseLink** так и **OpenJPA** или какой-нибудь другой, так как стандарт **JPA** позволяет переключаться между разными поставщиками постоянства **JPA**.



# Пример 1

Структура проекта представлена на рисунке ниже:



Рассмотрим структуру проекта подробнее:

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>sample</groupId>  
 <artifactId>sample</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>war</packaging>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-war-plugin</artifactId>  
 <configuration>  
 <warSourceDirectory>web</warSourceDirectory>  
 <webXml>web/WEB-INF/web.xml</webXml>  
 <goal>package</goal>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>4.3.9.RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>4.3.9.RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>4.3.9.RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>jstl</groupId>  
 <artifactId>jstl</artifactId>  
 <version>1.2</version>  
 </dependency>  
 </dependencies>  
</project>

Добавим в корневой элемент тег **<packaging />** со значением **war**, таким образом определив, что данный проект должен собираться в **war**архив.

Подключим необходимые зависимости: **spring-context**, **spring-core** необходимы для работы со Spring; **spring-webmvc** необходим для создания web приложения. Также необходимо подключить библиотеку **jstl** для вывода результатов на страницах jsp.

Также подключим плагины. **Maven-compiler-plugin** необходим для компиляции проекта. Здесь в параметрах задается версия java, которая используется в проекте. В данном случае 1.8. Также подключим **maven-war-plugin** для сборки проекта в warархив. В конфигурации данного плагина используя тег **<warSourceDirectory />** укажем базовую папку с web ресурсами (в данном случае папка называется web) и в теге **<webXml />** укажем путь к дескриптору развертывания (web/WEB-INF/web.xml). Заметим, что данные пути к папке и файлу должны быть указаны относительно корневой папки проекта. Также укажем, что создание warархива должно происходить на этапе сборки проекта. Для этого в тег **<goal />** запишем значение **packaging**.

1. Сконфигурируем web-приложение с помощью файла 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\_3\_1.xsd"  
 version="3.1">  
 <context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>/WEB-INF/applicationContext.xml</param-value>  
 </context-param>  
 <listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
 </listener>  
 <servlet>  
 <servlet-name>dispatcher</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <load-on-startup>1</load-on-startup>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>dispatcher</servlet-name>  
 <url-pattern>/</url-pattern>  
 </servlet-mapping>  
</web-app>

Для работы со Spring необходимо задать в теге **<context-param />** путь к файлу с конфигурацией контекста. Путь задается относительно папки, указанной как **webLocationDirectory**. Также в теге **<listener />**укажем класс обработчика события загрузки контекста. Всё это необходимо для корректной загрузки контекста Spring.

Затем опишем конфигурацию сервлета. Для этого используем тег **<servlet />**. В нём зададим уникальный идентификатор сервлета в теге **<servlrt-name />** и класс сервлета (как правило, используется стандартный класс org.springframework.web.servlet.DispatcherServlet) в теге **<servlet-class />**. Будем загружать сервлет при старте приложения, для этого добавим тег **<load-on-startup />** с неотрицательным значением (как правило, используется значение 1). Если же задать отрицательное значение, то загрузка сервлета будет происходить при первом обращении к нему.

Зададим базовый адрес, относительно которого будет происходить формирование адресов запросов, обрабатываемых данным сервлетом. Для этого в теге **<servlet-mappling/>** укажем идентификатор конфигурируемого сервлета используя вложенный тег **<servlet-name />** и путь (тег **<url-pattern />)**.Сервлет с данным идентификатором должен быть объявлен ранее.

1. Конфигурация контекста Spring описывается в файле applicationContext.xml. Так как используется конфигурация контекста с помощью аннотаций, то бины в данном файле регистрировать не нужно.

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
</beans>

1. Конфигурация сервлета приведена в файле dispatcher-servlet.xml (название данного файла формируется как «идентификатор сервлета» + «servlet.xml»).

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">  
  
 <mvc:annotation-driven />  
  
 <context:component-scan base-package="spring.example" />  
  
 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/views/" />  
 <property name="suffix" value=".jsp" />  
 </bean>  
  
 <mvc:resources mapping="/resources/\*\*" location="/resources/" />  
</beans>

Так как конфигурация бинов создаваемого приложения производиться с использованием аннотаций, то в конфигурационный файл необходимо добавить тег **<mvc:annotation-driven/>**, предварительно подключив при этом пространство имён **mvc**. Также в атрибуте **base-package** тега **<context:component-scan />** из пространства имён **context**, укажем директорию, в которой контексту необходимо производить поиск бинов.

В данном файле также объявляется конфигурация для создания бина **ViewResolver**, который определяет, какое представление (то есть какой именно файл **jsp**) соответствует каждому строковому идентификатору. Данный бин имеет 2 свойства: префикс и постфикс. Они необходимы для определения пути к файлу представления. Путь формируется как «префикс» + «идентификатор» + «постфикс». Например, если идентификатор равен «result», то ему будет соответствовать файл, расположенный по адресу «/WEB-INF/views/result.jsp».

Зададим месторасположение статических ресурсов (файлы CSS, JS скрипты, картинки, иконки и т.д.).Для этого используется тег **<resources/>,** находящийся в пространстве имён **mvc**. При этом ресурсами будут считаться файлы находящиеся в папке, заданной с помощью атрибута **location**, для обращения к ним браузера будут сформированы GET запросы с путём, заданным в атрибуте **mapping**. Например, если в атрибуте locationуказано значение «/res/files/»и в папке filesнаходиться файл style.css, а в атрибуте mappingзадано значение«/resouces/\*\*», то получить данный файл будет можно по адресу«/resources/style.css» Отметим, что путь к файлам ресурсов задается относительно папки, указанной как **webLocationDirictory**.

1. Рассмотрим классы данного приложения.
   1. Была создана следующая иерархия исключений.

Базовый класс BaseException:

package spring.example.exception;  
  
public class BaseException extends Exception {  
 public BaseException(String message) {  
 super(message);  
 }  
}

Класс ElementAlreadyExistException, соответствующий исключительной ситуации, когда при добавлении нового студента, студент с таким id уже зарегистрирован в системе.

package spring.example.exception;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.web.bind.annotation.ResponseStatus;  
  
@ResponseStatus(HttpStatus.*CONFLICT*)  
public class ElementAlreadyExistException extends BaseException {  
 private static String *template* = "Element with id %s already exist.";  
  
 public ElementAlreadyExistException(Integer elementId) {  
 super(String.*format*(*template*, elementId));  
 }  
}

Класс ElementNotFoundException соответствует ситуации, когда студент с запрашиваемымidне существует в системе.

package spring.example.exception;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.web.bind.annotation.ResponseStatus;  
  
@ResponseStatus(HttpStatus.*NOT\_FOUND*)  
public class ElementNotFoundException extends BaseException {  
 private static String *template* = "Element with id %s was not found.";  
  
 public ElementNotFoundException(Integer elementId) {  
 super(String.*format*(*template*, elementId));  
 }  
}

Аннотация **@ResponseStatus** используется для того, чтобы указать статус-код ответа сервера при возникновении данной исключительной ситуации.

* 1. Рассмотрим класс Student.

package spring.example;  
  
public class Student {  
 private Integer id;  
 private String name;  
 private Integer age;  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public Integer getId() {  
 return id;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
 public String getName() {  
 return name;  
 }  
  
 public void setAge(Integer age) {  
 this.age = age;  
 }  
 public Integer getAge() {  
 return age;  
 }  
}

* 1. Класс StudentsHandler используется для хранения в оперативной памяти списка добавленных в систему студентов и является бином.

package spring.example;  
import org.springframework.stereotype.Service;  
import spring.example.exception.ElementAlreadyExistException;  
import spring.example.exception.ElementNotFoundException;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class StudentsHandler {  
 private List<Student> students = new LinkedList<>();  
  
 public void addStudent(Student student) throws ElementAlreadyExistException {  
 Optional<Student> existStudent = students.stream().filter(e -> e.getId() == student.getId()).findFirst();  
 if(existStudent.isPresent()) {  
 throw new ElementAlreadyExistException(student.getId());  
 }  
 students.add(student);  
 }  
  
 public List<Student> getStudents() {  
 return students;  
 }  
  
 public Student getStudentById(Integer id) throws ElementNotFoundException {  
 Optional<Student> student = students.stream().filter(e -> e.getId() == id).findFirst();  
 if(!student.isPresent()) {  
 throw new ElementNotFoundException(id);  
 }  
 return student.get();  
 }  
}

* 1. Перейдем к рассмотрению контроллера. В данном приложении контроллером является экземпляр класса StudentsController.

package spring.example;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.validation.BindException;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.servlet.ModelAndView;  
import spring.example.exception.BaseException;  
import spring.example.exception.ElementAlreadyExistException;  
import spring.example.exception.ElementNotFoundException;  
  
@Controller  
public class StudentsController {  
  
 @Autowired  
 private StudentsHandler studentsHandler;  
  
 @RequestMapping(value = "/student/new", method = RequestMethod.*GET*)  
 public String createForm() {  
 return "createForm";  
 }  
  
 @RequestMapping(value = "/add", method = RequestMethod.*POST*)  
 public ModelAndView addStudent(@ModelAttribute Student student) throws ElementAlreadyExistException {  
 studentsHandler.addStudent(student);  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("student");  
 modelAndView.addObject("student", student);  
  
 return modelAndView;  
 }  
  
 @RequestMapping(value = "/student/{id}", method = RequestMethod.*GET*)  
 public ModelAndView getStudentById(@PathVariable("id") int id) throws ElementNotFoundException {  
 Student student = studentsHandler.getStudentById(id);  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("student");  
 modelAndView.addObject("student", student);  
  
 return modelAndView;  
 }  
  
 @RequestMapping(value = "/students", method = RequestMethod.*GET*)  
 public ModelAndView getStudents() {  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("list");  
 modelAndView.addObject("students", studentsHandler.getStudents());  
  
 return modelAndView;  
 }  
  
 @ExceptionHandler(BaseException.class)  
 public ModelAndView handleCustomExceptions(BaseException ex) {  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("error");  
 modelAndView.addObject("message", ex.getMessage());  
  
 return modelAndView;  
 }  
  
 @ExceptionHandler(BindException.class)  
 public ModelAndView numberFormatError() {  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("error");  
 modelAndView.addObject("message", "Error with parsing number attribute.");  
 return modelAndView;  
 }  
}

В данном контроллере объявляются методы, обрабатывающие различные REST-запросы.

На каждый запрос создаётся отдельный обработчик. К каждому такому методу-обработчику добавляется аннотация **@RequestMapping**, в которой указывается, по какому URL-адресу будет происходить обработка запроса и какой метод (GET, POST, DELETE, PUTи т.д.) необходимо использовать для обработки запроса.

Метод-обработчик может возвращать объект модель-представление (**ModelAndView**), строку с идентификатором представления или не возвращать ничего (в таком случае клиент не увидит каких-либо изменений после обработки запроса).

В объекте модель-представление указывается, на каком представлении необходимо отобразить содержащиеся в модели данные. После возвращения объекта класса **ModelAndView** (как и при возвращении методом строки с идентификатором представления) **ViewResolver** по идентификатору определяет, какую страницу необходимо вернуть, а сервлет добавляет объекты модели в **HttpServletResponse** объект.

Параметры в методы-обработчики могут быть переданы одним из следующих способов (автоматически подставятся в соответствующий параметр метода):

1. Как параметр запроса (например, при запросе «/student?name=Ivanov»). Для параметра используется аннотация **@RequestParam(value = “name”).**
2. Как переменная в запросе (например, идентификаторсо значением 1 из запроса «/student/1»). Необходимо добавить аннотацию**@PathVariable**.
3. Как тело запроса. Если данные передаются из формы, как в рассмотренном примере, необходимо использовать аннотацию **@ModelAttribute**. Если же передаётся JSON объект, то используется аннотация **@RequestBody**.

Также в контроллер можно добавлять обработчики исключений. Для этого используется аннотация **@ExceptionHandler**, куда в качестве атрибута передаётся класс, соответствующий обрабатываемому исключению. При этом будут обработаны исключения как данного класса, так и наследуемых от него. Также заметим, что если в методе, отмеченном данной аннотацией объявить параметр, соответствующий экземпляру исключения, то перехваченное исключение автоматически подставиться в него.

1. В данном приложении использованы следующие файлы-представления:
   1. list.jsp для вывода информации обо всех студентах(используется библиотека **JSTL** для вывода по циклу)

<%@**taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%@**page** contentType="text/html; charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>All Students</title>  
 <link href="<**c:url** value="/resources/style.css" />" rel="stylesheet">  
</head>  
<body>  
  
<table class="result">  
 <caption>All students</caption>  
 <tr>  
 <th>ID</th>  
 <th>Name</th>  
 <th>Age</th>  
 </tr>  
 <**c:forEach** items="**${**students**}**" var="student">  
 <tr>  
 <td>**${**student.id**}**</td>  
 <td>**${**student.name**}**</td>  
 <td>**${**student.age**}**</td>  
 </tr>  
 </**c:forEach**>  
</table>  
<a class="button" href="/student/new">Add new student</a>  
</body>  
</html>

* 1. student.jsp для вывода информации об одном студенте.

<%@**taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Student</title>  
 <link href="<**c:url** value="/resources/style.css" />" rel="stylesheet">  
</head>  
<body>  
  
<table class="result">  
 <caption>Submitted Student Information</caption>  
 <tr>  
 <td>ID</td>  
 <td>**${**student.id**}**</td>  
 </tr>  
 <tr>  
 <td>Name</td>  
 <td>**${**student.name**}**</td>  
 </tr>  
 <tr>  
 <td>Age</td>  
 <td>**${**student.age**}**</td>  
 </tr>  
 <a class="button" href="/students">To list of students</a>  
</table>  
</body>  
</html>

* 1. createForm.jsp содержит форму для добавления нового студента.

<%@**taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Add Student</title>  
 <link href="<**c:url** value="/resources/style.css" />" rel="stylesheet">  
</head>  
<body>  
<form method="POST" action="/add">  
 <table>  
 <caption>Student Information</caption>  
 <tr>  
 <td>ID</td>  
 <td><input name="id" /></td>  
 </tr>  
 <tr>  
 <td>Name</td>  
 <td><input name="name" /></td>  
 </tr>  
 <tr>  
 <td>Age</td>  
 <td><input name="age" /></td>  
 </tr>  
 </table>  
 <input class="button" type="submit" value="Submit"/>  
</form>  
</body>  
</html>

* 1. error.jsp для вывода информации об ошибках.

<%@**taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Error</title>  
 <link href="<**c:url** value="/resources/style.css" />" rel="stylesheet">  
</head>  
<body>  
 <div class="error">Error: **${**message**}**</div>  
 <a class="button" href="/students">To list of students</a>  
</body>  
</html>

* 1. Файл со стилями style.css

body {  
 font-family: "Open Sans", sans-serif;  
 line-height: 1.25;  
 text-align: center;  
}  
form {  
 display: inline-block;  
}  
table.result {  
 border: 1px solid #ccc;  
 border-collapse: collapse;  
 margin: 0 10%;  
 padding: 0;  
 width: 80%;  
 table-layout: fixed;  
}  
table.result caption {  
 font-size: 1.5em;  
 margin: .5em 0 .75em;  
}  
table.result tr {  
 background: #f8f8f8;  
 border: 1px solid #ddd;  
 padding: .35em;  
}  
table.result th,  
table.result td {  
 padding: .625em;  
}  
table.result th {  
 font-size: .85em;  
 letter-spacing: .1em;  
 text-transform: uppercase;  
}  
.button, .button:hover {  
 border: 1px solid #999999;  
 display: inline-block;  
 padding: 5px 15px;  
 margin: 10px;  
 text-decoration: none;  
 color: #a00;  
 box-shadow: 0 0 5px rgba(0,0,0,0.3);  
 background: linear-gradient(to bottom, #fcfff4, #e9e9ce);  
}  
div.error {  
 background: #ffc4ba;  
 border: 1px solid #a80000;  
 border-radius: 5px;  
 color: #a80000;  
}

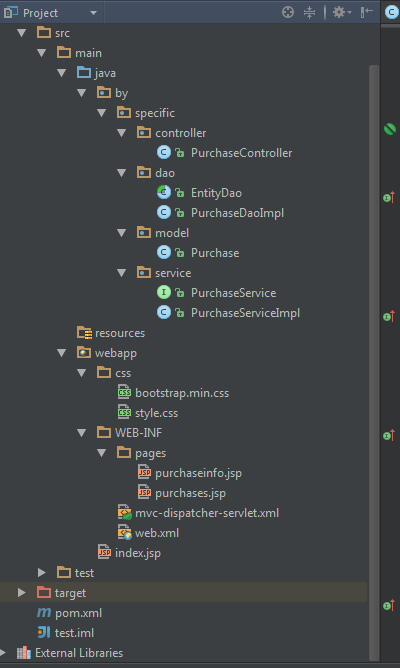
1. Для запуска приложения необходимо выполнить сборку проекта. Для этого в консоли Windows перейдем в корневую папку проекта, используя команду «**cd**».

Затем выполним команду «**mvncleanpackage**».

После этого в папке target должен появиться war-архив, который необходимо разместить на любом сервере веб-приложений, например, Tomcat 8.5.20.

# Пример 2

Рассмотрим еще один пример, реализующий магазин по продаже электротоваров – Структура проекта представлена ниже:



Так же рассмотрим структуру данного проекта более подробно:

Pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  
<modelVersion>4.0.0</modelVersion>  
 <groupId>com.springapp</groupId>  
 <artifactId>testSpecific</artifactId>  
 <packaging>war</packaging>  
 <version>1.0-SNAPSHOT</version>  
 <name>testSpecific</name>  
  
 <properties>  
 <java-version>1.8</java-version>  
 <org.springframework-version>4.0.3.RELEASE</org.springframework-version>  
 <org.aspectj-version>1.7.4</org.aspectj-version>  
 <org.slf4j-version>1.7.5</org.slf4j-version>  
 <hibernate.version>4.3.5.Final</hibernate.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${org.springframework-version}</version>  
 <exclusions>  
 *<!-- Exclude Commons Logging in favor of SLF4j -->* <exclusion>  
 <groupId>commons-logging</groupId>  
 <artifactId>commons-logging</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${org.springframework-version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-tx</artifactId>  
 <version>${org.springframework-version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>4.2.1.RELEASE</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-orm</artifactId>  
 <version>${org.springframework-version}</version>  
 </dependency>  
  
 *<!-- Hibernate -->* <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-core</artifactId>  
 <version>${hibernate.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-entitymanager</artifactId>  
 <version>${hibernate.version}</version>  
 </dependency>  
  
 *<!-- Apache Commons DBCP -->* <dependency>  
 <groupId>commons-dbcp</groupId>  
 <artifactId>commons-dbcp</artifactId>  
 <version>1.4</version>  
 </dependency>  
  
 *<!-- Logging -->* <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>${org.slf4j-version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>jcl-over-slf4j</artifactId>  
 <version>${org.slf4j-version}</version>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-log4j12</artifactId>  
 <version>${org.slf4j-version}</version>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>1.2.15</version>  
 <exclusions>  
 <exclusion>  
 <groupId>javax.mail</groupId>  
 <artifactId>mail</artifactId>  
 </exclusion>  
 <exclusion>  
 <groupId>javax.jms</groupId>  
 <artifactId>jms</artifactId>  
 </exclusion>  
 <exclusion>  
 <groupId>com.sun.jdmk</groupId>  
 <artifactId>jmxtools</artifactId>  
 </exclusion>  
 <exclusion>  
 <groupId>com.sun.jmx</groupId>  
 <artifactId>jmxri</artifactId>  
 </exclusion>  
 </exclusions>  
 <scope>runtime</scope>  
 </dependency>  
  
 *<!-- Servlet -->* <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>servlet-api</artifactId>  
 <version>2.5</version>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <groupId>javax.servlet.jsp</groupId>  
 <artifactId>jsp-api</artifactId>  
 <version>2.1</version>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>jstl</artifactId>  
 <version>1.2</version>  
 </dependency>  
  
 *<!-- Test -->* <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.7</version>  
 <scope>test</scope>  
 </dependency>  
  
 *<!--MysQL-->* <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.36</version>  
 </dependency>  
  
 <dependency>  
 <groupId>commons-codec</groupId>  
 <artifactId>commons-codec</artifactId>  
 <version>1.4</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <finalName>testSpecific</finalName>  
 <plugins>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.6</source>  
 <target>1.6</target>  
 </configuration>  
 </plugin>  
 <plugin>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <configuration>  
 <includes>  
 <include>\*\*/\*Tests.java</include>  
 </includes>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

Mvc-dispatcher-server для конфигурации сервлета

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">  
  
 <context:component-scan base-package="by.specific"/>  
  
 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/pages/"/>  
 <property name="suffix" value=".jsp"/>  
 </bean>  
  
 *<!-- Database Information -->* <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"  
 destroy-method="close">  
 <property name="driverClassName" value="com.mysql.jdbc.Driver"/>  
 <property name="url"  
 value="jdbc:mysql://localhost:3306/ris6"/>  
 <property name="username" value="root"/>  
 <property name="password" value="333"/>  
 </bean>  
  
 *<!-- Hibernate 4 SessionFactory Bean definition -->* <bean id="hibernate4AnnotatedSessionFactory"  
 class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">  
 <property name="dataSource" ref="dataSource"/>  
 <property name="annotatedClasses">  
 <list>  
 <value>by.specific.model.Purchase</value>  
 </list>  
 </property>  
 <property name="hibernateProperties">  
 <props>  
 <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>  
 <prop key="hibernate.connection.autocommit">true</prop>  
 <prop key="hibernate.show\_sql">true</prop>  
 </props>  
 </property>  
 </bean>  
  
 *<!--BookDao and BookService beans-->* <bean id="purchaseDao" class="by.specific.dao.PurchaseDaoImpl">  
 <property name="sessionFactory" ref="hibernate4AnnotatedSessionFactory"/>  
 </bean>  
 <bean id="purchaseService" class="by.specific.service.PurchaseServiceImpl">  
 <property name="purchaseDao" ref="purchaseDao"/>  
 </bean>  
  
 <context:component-scan base-package="by.specific"/>  
  
 <tx:annotation-driven transaction-manager="transactionManager"/>  
  
 <bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">  
 <property name="sessionFactory" ref="hibernate4AnnotatedSessionFactory"/>  
 </bean>  
  
 <mvc:default-servlet-handler/>  
 <mvc:annotation-driven/>  
 <mvc:resources mapping="/css/\*\*" location="/css/"/>  
</beans>

web.xml

<web-app version="2.4"  
 xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee  
 http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">  
  
 <display-name>Book Manager</display-name>  
  
 <servlet>  
 <servlet-name>mvc-dispatcher</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <load-on-startup>1</load-on-startup>  
 </servlet>  
  
 <servlet-mapping>  
 <servlet-name>mvc-dispatcher</servlet-name>  
 <url-pattern>/</url-pattern>  
 </servlet-mapping>  
 <filter>  
 <filter-name>encodingFilter</filter-name>  
 <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  
 <init-param>  
 <param-name>encoding</param-name>  
 <param-value>UTF-8</param-value>  
 </init-param>  
 </filter>  
 <filter-mapping>  
 <filter-name>encodingFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
 </filter-mapping>  
 <welcome-file-list>  
 <welcome-file>index.jsp</welcome-file>  
 </welcome-file-list>  
</web-app>

В пакете контроллеров находится контроллер PurchaseController

package by.specific.controller;  
import by.specific.model.Purchase;  
import by.specific.service.PurchaseService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;  
import java.util.List;  
  
  
@Controller  
public class PurchaseController {  
 private PurchaseService purchaseService;  
 private static boolean *flag*=true;  
 private static List<Purchase> *purchaseList* = new ArrayList<Purchase>();  
 @Autowired(required = true)  
 @Qualifier(value = "purchaseService")  
 public void setPurchaseService(PurchaseService purchaseService) {  
 this.purchaseService = purchaseService;  
 }  
  
 @RequestMapping(value = "purchases", method = RequestMethod.*GET*)  
 public String listPurchases(Model model){  
 if(*flag*) {  
 *purchaseList* = this.purchaseService.listPurchases();  
 }  
 *flag* = true;  
 model.addAttribute("purchase", new Purchase());  
 model.addAttribute("listPurchases", *purchaseList* );  
 return "purchases";  
 }  
  
 @RequestMapping(value = "/purchases/find")  
 public String authorizationAccount(@ModelAttribute("purchase") Purchase purchase){  
 *purchaseList* = this.purchaseService.findPurchases(purchase.getName());  
 *flag* = false;  
 return "redirect:/purchases";  
 }  
  
 @RequestMapping(value = "/purchases/add", method = RequestMethod.*POST*)  
 public String addPurchase(@ModelAttribute("purchase") Purchase purchase){  
 if(purchase.getId() == 0){  
 this.purchaseService.addPurchase(purchase);  
 }else {  
 this.purchaseService.updatePurchase(purchase);  
 }  
  
 return "redirect:/purchases";  
 }  
  
 @RequestMapping("/remove/{id}")  
 public String removePurchase(@PathVariable("id") int id){  
 this.purchaseService.removePurchase(id);  
  
 return "redirect:/purchases";  
 }  
  
 @RequestMapping("edit/{id}")  
 public String editPurchase(@PathVariable("id") int id, Model model){  
 model.addAttribute("purchase", this.purchaseService.getPurchaseById(id));  
 model.addAttribute("listPurchases", this.purchaseService.listPurchases());  
  
 return "purchases";  
 }  
  
 @RequestMapping("purchaseinfo/{id}")  
 public String purchaseData(@PathVariable("id") int id, Model model){  
 model.addAttribute("purchase", this.purchaseService.getPurchaseById(id));  
  
 return "purchaseinfo";  
 }  
}

Модель товара представлена в классе model

package by.specific.model;  
  
import javax.persistence.Entity;  
import javax.persistence.Table;  
  
import javax.persistence.\*;  
  
  
@Entity  
@Table(name = "purchases")  
public class Purchase {  
  
 @Id  
 @Column(name = "id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
  
 @Column(name = "name")  
 private String name;  
  
 @Column(name = "model")  
 private String model;  
  
 @Column(name = "amount")  
 private int amount;  
  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getModel() {  
 return model;  
 }  
  
 public void setModel(String model) {  
 this.model = model;  
 }  
  
 public int getAmount() {  
 return amount;  
 }  
  
 public void setAmount(int amount) {  
 this.amount = amount;  
 }  
  
  
 public Purchase(){}  
  
 public Purchase(String name, String model, int amount) {  
 this.name = name;  
 this.model = model;  
 this.amount = amount;  
 }  
}

В пакете service представлен интерфейс PurchaseService и конкретный класс реализации его методов PurchaseServiceImpl.

PurchaseService

package by.specific.service;  
  
import by.specific.model.Purchase;  
  
import java.util.List;  
  
public interface PurchaseService {  
 public void addPurchase(Purchase purchase);  
  
 public void updatePurchase(Purchase purchase);  
  
 public void removePurchase(int id);  
  
 public Purchase getPurchaseById(int id);  
  
 public List<Purchase> findPurchases(String name);  
  
 public List<Purchase> listPurchases();  
}

PurchaseServiceImpl

package by.specific.service;  
  
import by.specific.model.Purchase;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import by.specific.dao.PurchaseDaoImpl;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Service  
public class PurchaseServiceImpl implements PurchaseService {  
  
 private PurchaseDaoImpl purchaseDao;  
  
 public void setPurchaseDao(PurchaseDaoImpl purchaseDao) {  
 this.purchaseDao = purchaseDao;  
 }  
  
 @Override  
 @Transactional  
 public void addPurchase(Purchase purchase) {  
 this.purchaseDao.add(purchase);  
 }  
  
 @Override  
 @Transactional  
 public void updatePurchase(Purchase purchase) {  
 this.purchaseDao.update(purchase);  
 }  
  
 @Override  
 @Transactional  
 public void removePurchase(int id) {  
 this.purchaseDao.remove(id);  
 }  
  
 @Override  
 @Transactional  
 public Purchase getPurchaseById(int id) {  
 return this.purchaseDao.getById(id);  
 }  
  
 @Override  
 @Transactional  
 public List<Purchase> findPurchases(String name) {  
 List list = new ArrayList();  
 for (Purchase purch:this.purchaseDao.getList()) {  
 if(purch.getName().equals(name))  
 list.add(purch);  
 }  
 return list;  
 }  
  
 @Override  
 @Transactional  
 public List<Purchase> listPurchases() {  
 return this.purchaseDao.getList();  
 }  
}

Далее рассмотрим пакет dao. Тут представлены два класса: EntityDao – абстрактный класс и конкретный класс EntityDaoImpl. Данные классы непосредственно необходимы для работы с БД.

EntityDao

package by.specific.dao;  
  
import org.hibernate.SessionFactory;  
  
import java.util.List;  
  
public abstract class EntityDao<T> {  
  
 protected SessionFactory sessionFactory;  
 public void setSessionFactory(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
 }  
  
 public abstract void add(T entity);  
 public abstract void update(T entity);  
 public abstract void remove(int id);  
 public abstract T getById(int id);  
 public abstract List<T> getList();  
}

EntityDaoImpl

package by.specific.dao;  
  
import by.specific.model.Purchase;  
import org.hibernate.Session;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
*/\*\*  
 \* Created by purchase1 on 6/23/2017.  
 \*/*@Repository  
public class PurchaseDaoImpl extends EntityDao<Purchase> {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(PurchaseDaoImpl.class);  
 @Override  
 public void add(Purchase entity) {  
 Session session = this.sessionFactory.getCurrentSession();  
 session.persist(entity);  
 *logger*.info("Purchase successfully saved. Purchase details: " + entity);  
 }  
  
 @Override  
 public void update(Purchase entity) {  
 Session session = this.sessionFactory.getCurrentSession();  
 session.update(entity);  
 *logger*.info("Purchase successfully updated. Purchase details: " + entity);  
 }  
  
 @Override  
 public void remove(int id) {  
 Session session = this.sessionFactory.getCurrentSession();  
 Purchase purchase = (Purchase) session.load(Purchase.class, new Integer(id));  
 if (purchase != null) {  
 session.delete(purchase);  
 }  
 *logger*.info("Purchase successfully removed. Purchase details: " + purchase);  
 }  
  
 @Override  
 public Purchase getById(int id) {  
 Session session = this.sessionFactory.getCurrentSession();  
 Purchase purchase = (Purchase) session.load(Purchase.class, new Integer(id));  
 *logger*.info("Purchase successfully loaded. Purchase details: " + purchase);  
 return purchase;  
 }  
  
 @Override  
 public List<Purchase> getList() {  
 Session session = this.sessionFactory.getCurrentSession();  
 List<Purchase> purchaseList = session.createQuery("from Purchase").list();  
 for (Purchase purchase: purchaseList) {  
 *logger*.info("Purchase list: " + purchase);  
 }  
 return purchaseList;  
 }  
}

А так же представлены каскадные таблицы стилей и jsp файлы.

style.css

@media screen and (max-width: 768px) {  
 tr {  
 display: flex;  
 flex-flow: row wrap;  
 justify-content: space-around;  
 }  
  
 td, th {  
 display: block;  
 width: 33%;  
 }  
  
 th:first-child,  
 td:first-child {  
 text-align: center;  
 background: #efefef;  
 width: 100%;  
 }  
  
 th:first-child {  
 display: none;  
 }  
}

purchaceinfo.jsp

<%--  
 Created by IntelliJ IDEA.  
 User: Дима  
 Date: 16.11.2017  
 Time: 22:36  
 To change this template use File | Settings | File Templates.  
--%>  
<%@ **taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%@ **taglib** uri="http://www.springframework.org/tags" prefix="spring" %>  
<%@ **taglib** uri="http://www.springframework.org/tags/form" prefix="form" %>  
<%@ **taglib** prefix="from" uri="http://www.springframework.org/tags/form" %>  
<%@ **page** session="false" %>  
<%@ **page** contentType="text/html; charset=UTF-8" %>  
<meta http-equiv="refresh" content="30">  
<html>  
<head>  
 <meta name="viewport" content="width=device-width, initial-scale=1.0"><link rel='stylesheet' type="text/css" href="<**c:url** value="/css/bootstrap.min.css"/>"/>  
 <link rel="stylesheet" type="text/css" href="<**c:url** value="/css/bootstrap.min.css"/> "/>  
</head>  
<body>  
<br/>  
<div>  
 <div class="container">  
 <div class="row">  
 <a class="btn btn-primary col-xs-4 col-xs-offset-1 col-sm-2 col-md-1 col-lg-1" align="center" type="submit" href="/purchases">Выход</a>  
 </div>  
 <div class="row">  
 <br/>  
 <br/>  
 <div class="col-xs-12 col-sm-10 col-sm-offset-1 col-md-8 col-md-offset-2 col-lg-8 col-lg-offset-2 ">  
 <table class="table">  
 <tr class="bg-info">  
 <td >№</td>  
 <td >**${**purchase.id**}**</td>  
 </tr>  
 <tr>  
 <td >Марка</td>  
 <td >**${**purchase.name**}**</td>  
 </tr>  
 <tr>  
 <td >Марка</td>  
 <td >**${**purchase.model**}**</td>  
 </tr>  
 <tr>  
 <td >Количество</td>  
 <td >**${**purchase.amount**}**</td>  
 </tr>  
 </table>  
  
 </div>  
 </div>  
 </div>  
</div>  
</body>  
</html>

purchases.jsp

<%--  
 Created by IntelliJ IDEA.  
 User: Дима  
 Date: 16.11.2017  
 Time: 22:21  
 To change this template use File | Settings | File Templates.  
--%>  
<%@ **taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%@ **taglib** uri="http://www.springframework.org/tags" prefix="spring" %>  
<%@ **taglib** uri="http://www.springframework.org/tags/form" prefix="form" %>  
<%@ **taglib** prefix="from" uri="http://www.springframework.org/tags/form" %>  
<%@ **page** session="false" %>  
<%@ **page** contentType="text/html; charset=UTF-8" %>  
<meta http-equiv="refresh" content="30">  
<html>  
<head>  
 <meta name="viewport" content="width=device-width, initial-scale=1.0"><link rel='stylesheet' type="text/css" href="<**c:url** value="/css/bootstrap.min.css"/>"/>  
 <link rel="stylesheet" type="text/css" href="<**c:url** value="/css/bootstrap.min.css"/> "/>  
</head>  
<body>  
  
<br/>  
<br/>  
<div>  
 <**c:if** test="**${**!**empty** listPurchases**}**">  
 <div class="container">  
 <div class="row">  
 <div class="col-xs-12 col-sm-10 col-sm-offset-1 col-md-8 col-md-offset-2 col-lg-8 col-lg-offset-2 ">  
 <div class="panel-body">  
 <**c:url** var="inputAction" value="/purchases/find"/>  
 <**form:form** action="**${**inputAction**}**" commandName="purchase">  
 <fieldset>  
 <div class="row">  
 <div class="form-group">  
 <table>  
 <tr>  
 <td class="col-xs-5 col-sm-5 col-md-5 col-lg-5 " >  
 <**form:input** class="form-control" path="name" placeholder="Марка" name="login" type="text"  
 autofocus=""/>  
 </td>  
 <td class="col-xs-1 col-sm-1 col-md-1 col-lg-1 " >  
 <input class="form-control btn-primary" type="submit"  
 value="<**spring:message** text="Поиск"/>"/>  
 </td>  
 </tr>  
 </table>  
 </div>  
 </div>  
 </fieldset>  
 </**form:form**>  
 </div>  
 <table class="table table-bordered">  
 <tr class="bg-info row">  
 <th class="col-xs-1 col-sm-1 col-md-1 col-lg-1 ">№</th>  
 <th class="col-xs-3 col-sm-3 col-md-3 col-lg-3 ">Марка</th>  
 <th class="col-xs-3 col-sm-3 col-md-3 col-lg-3 ">Модель</th>  
 <th class="col-xs-5 col-sm-5 col-md-5 col-lg-5 ">Количество</th>  
 <th></th>  
 <th></th>  
 </tr>  
 <**c:forEach** items="**${**listPurchases**}**" var="purchase">  
 <tr class="row">  
 <td class="col-xs-1 col-sm-1 col-md-1 col-lg-1 ">**${**purchase.id**}**</td>  
 <td class="col-xs-3 col-sm-3 col-md-3 col-lg-3 "><a href="/purchaseinfo/**${**purchase.id**}**">**${**purchase.name**}**</a></td>  
 <td class="col-xs-5 col-sm-5 col-md-5 col-lg-5 " >**${**purchase.model**}**</td>  
 <td class="col-xs-5 col-sm-5 col-md-5 col-lg-5 " >**${**purchase.amount**}**</td>  
 <td><a href="<**c:url** value='/edit/**${**purchase.id**}**'/>">Изменить</a></td>  
 <td><a href="<**c:url** value='/remove/**${**purchase.id**}**'/>">Удалить</a></td>  
 </tr>  
 </**c:forEach**>  
 </table>  
 </div>  
 </div>  
 </div>  
 </**c:if**>  
</div>  
  
  
<**c:url** var="addAction" value="/purchases/add"/>  
<div class="container">  
 <div class="row">  
 <div class="col-xs-12 col-sm-10 col-sm-offset-1 col-md-8 col-md-offset-2 col-lg-8 col-lg-offset-2 ">  
<**form:form** action="**${**addAction**}**" commandName="purchase">  
 <table class="table">  
 <**c:if** test="**${**!**empty** purchase.name**}**">  
 <tr class="bg-info">  
 <td>  
 <**form:label** path="id">  
 <**spring:message** text="ID"/>  
 </**form:label**>  
 </td>  
 <td class="form-group">  
 <**form:input** path="id" readonly="true" size="8" disabled="true"/>  
 <**form:hidden** path="id"/>  
 </td>  
 </tr>  
 </**c:if**>  
 <tr>  
 <td>  
 <**form:label** path="name">  
 <**spring:message** text="Марка"/>  
 </**form:label**>  
 </td>  
 <td class="form-group">  
 <**form:input** path="name"/>  
 </td>  
 </tr>  
 <tr>  
 <td>  
 <**form:label** path="model">  
 <**spring:message** text="Модель"/>  
 </**form:label**>  
 </td>  
 <td class="form-group">  
 <**form:input** path="model"/>  
 </td>  
 </tr>  
 <tr>  
 <td>  
 <**form:label** path="amount">  
 <**spring:message** text="Количество"/>  
 </**form:label**>  
 </td>  
 <td class="form-group">  
 <**form:input** path="amount"/>  
 </td>  
 </tr>  
 <tr>  
 <td colspan="2">  
 <**c:if** test="**${**!**empty** purchase.name**}**">  
 <input class="btn btn-primary" type="submit"  
 value="<**spring:message** text="Изменить заказ"/>"/>  
 </**c:if**>  
 <**c:if** test="**${empty** purchase.name**}**">  
 <input class="btn btn-primary" type="submit"  
 value="<**spring:message** text="Добавить заказ"/>"/>  
 </**c:if**>  
 </td>  
 </tr>  
 </table>  
</**form:form**>  
 </div>  
 </div>  
</div>  
</body>  
</html>

Скрипт БД:

CREATE DATABASE IF NOT EXISTS `ris6` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;

USE `ris6`;

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `purchases`

--

DROP TABLE IF EXISTS `purchases`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `purchases` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`model` varchar(45) NOT NULL,

`amount` int(11) NOT NULL,

PRIMARY KEY (`id`),

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

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `purchases`

--

LOCK TABLES `purchases` WRITE;

/\*!40000 ALTER TABLE `purchases` DISABLE KEYS \*/;

INSERT INTO `purchases` VALUES (1,'Asus','K100',15),(2,'Lenovo','I200',10),(5,'sdfsdf','sdfsdf',23);

/\*!40000 ALTER TABLE `purchases` ENABLE KEYS \*/;

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

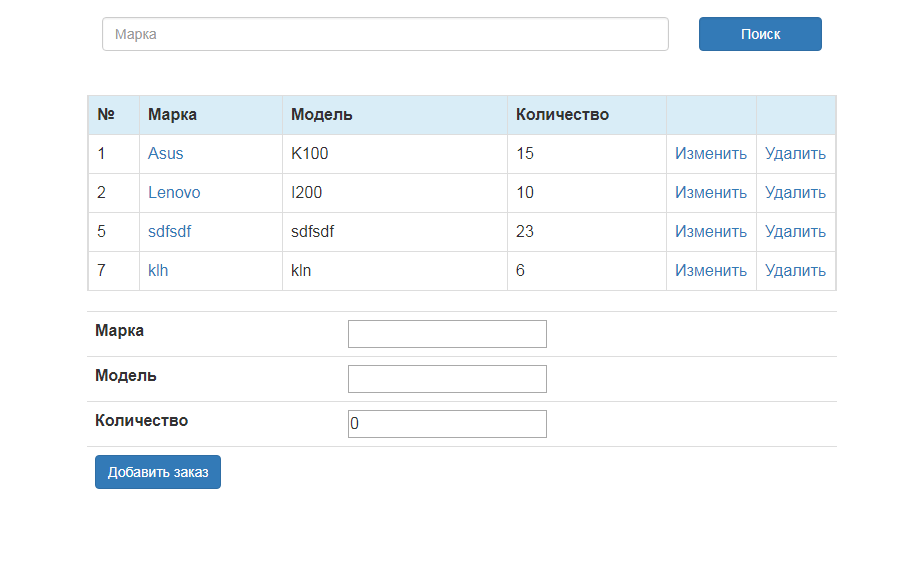
/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

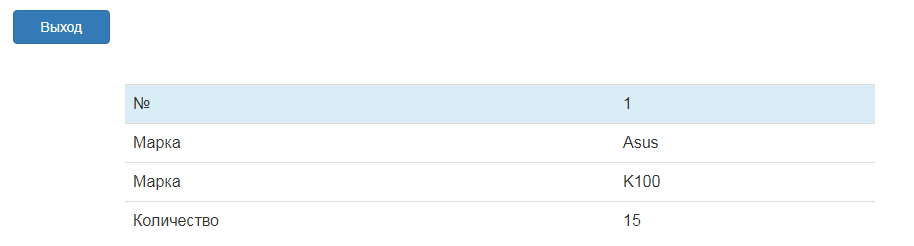
/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

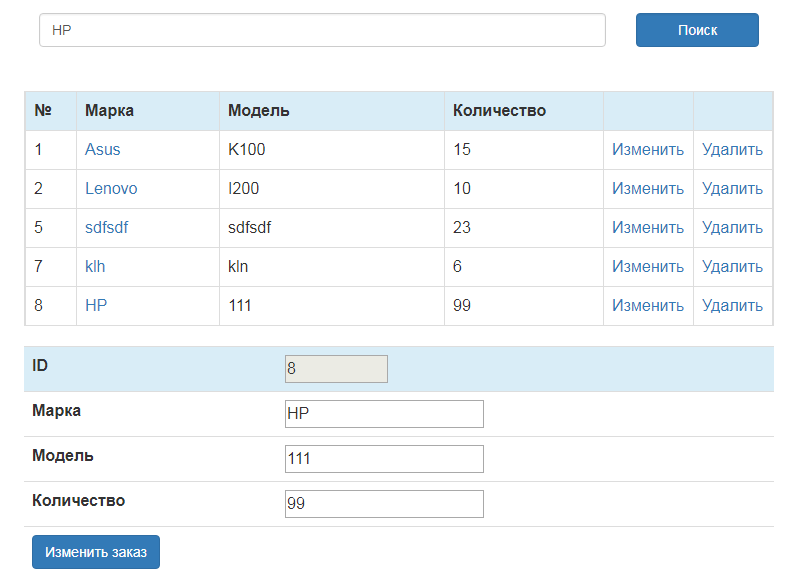
/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

Результат работы







Практическая часть

Программа представляет из себя web-приложение, реализованное на архитектуре Spring MVC. Элементы, которые **обязательно** должны присутствовать в программной реализации:

- базовый класс, минимум 4 метода;

На 8 баллов:

- Хранение данных в коллекции объектов;

На 9-10 баллов

- хранение данных в БД (dao. Применение Hibirnate).

1. Программа для размещения депозита в банке. Изучить предметную область, её основные сущности. Система, дает возможность клиенту выбрать депозит, данные о которых (не менее 5) хранятся в БД. После выбора клиенту становятся доступными дополнительные функции: просмотр остатка на определенное число, расчет процентых начислений за выбранный период, снятие процентов, пополнение счета (можно дополнить). Предусмотреть сохрание истории операций и формирование выписки (отчета) о проделанных операциях за выбранный промежуток времени.
2. Программа продажи билетов на автобус с регистрацией пассажиров. Данные об автобусах и количестве свободных мест хранятся в БД. Клиенту предоставляется возможность выбрать маршрут, а потом доступную дату, на которую есть билеты. Клиент вводит свои данные (ФИО, номер телефона, серия и номер паспорта – предусмотреть валидацию). Данные о пассажире сохраняются. Предусмотреть возможность отменить регистрацию.
3. Система учета проката велосипедов. Данные о велосипедах хранятся в БД. Клиент может просматривать доступные велосипеды в разное время (например, да пару дней). Предусмотреть несколько вариантов проката (1 час, 3 часа, сутки – например, как минимум). При заказе оформляется договор с данными клиента, клиент может получить свою копию договора. Вести учет сданных в прокат велосипедов. Предусмотреть комиссию за просроченную сдачу велосипеда обратно, рассчитать сумму. Для владельца прокатом разработать форму для расчета периода окупаемости велосипеда при различных факторах.
4. Программа для сдачи в аренду квартир. Данные о квартирах хранятся БД. Собственник может добавить объявление о квартире, предусмотреть разделение по регионам. Клиенты выбирают регион и квартиру из списка для просмотра, предусмотреть подсчет просмотров для каждой квартиры. Клиент может сохранить понравившуюся квартиру в текстовый файл, название файла вводится клиентом через консоль вручную. Предусмотреть возможные ошибки. Добавить поиск и несколько фильтров для поиска квартир.
5. Система ведения бонусной программы магазина. Регистрировать новые бонусные карточки клиентов, данные хранить в xml-формате. При совершении покупки вводить номер бонусной клиента (предусмотреть валидацию) и зачислять процент от всей стоимости покупки. Снимать баллы в счет оплаты. Для клиента предусмотреть просмотр баланса бонусов. Делать отчет по карточке (зачисление, снятие) в текстовом формате.

Контрольные вопросы:

1. Расскажите о Spring Framework.
2. Каковы некоторые из важных особенностей и преимуществ Spring Framework?
3. Приведите названия некоторых важных Spring модулей.
4. Что такое Spring бин?
5. Какое значение имеет конфигурационный файл Spring Bean?
6. Что такое связывание в Spring и расскажите об аннотации @Autowired?
7. Что такое контроллер в Spring MVC?
8. Какая разница между аннотациями @Component, @Repository и @Service в Spring?
9. Расскажите, что вы знаете о DispatcherServlet и ContextLoaderListener.
10. Как обрабатывать исключения в Spring MVC Framework?
11. Можем ли мы иметь несколько файлов конфигурации Spring?
12. Что такое Hibernate Framework?
13. Какие важные преимущества дает использование Hibernate Framework?
14. Каковы преимущества Hibernate над JDBC?
15. Назовите некоторые важные интерфейсы Hibernate.
16. Что такое конфигурационный файл Hibernate?
17. Что такое Hibernate mapping file?
18. Назовите некоторые важные аннотации, используемые для отображения в Hibernate.
19. Что вы знаете о Hibernate SessionFactory и как его сконфигурировать?
20. Является ли Hibernate SessionFactory потокобезоспансым?