**Spring MVC Form example.**

В этом проекте показан базовый пример использования фреймворка Spring MVC 3+, а именно обработка форм.

**Запуск примера:**

Для запуска примера вам нужно иметь:

- Исходники проекта. Их можно скачать с Github репозитория: <https://github.com/Rudeg/Spring-MVC-Example>

**-** Одну из современных IDE для работы с Java. (IntelliJ IDEA, Eclipse, Netbeans).

- Tomcat версии 7.\*

Вам нужно скачать исходники проекта с репозитория. После этого, выполнить импорт проекта в IDE. Далее настроить автоматический деплой на аппликейшн сервер(Tomcat). После этих действий проект должен заработать.

Видео запуска проекта в разных IDE:

* Idea: <http://screencast.com/t/x1ISPAaEx>
* Eclipse: <http://screencast.com/t/huO1WyaLCD>
* Netbeans: <http://screencast.com/t/LYXUpCwADX>

*Запуск проекта без IDE:*

Для запуска проекта без IDE вам потребуется:

- Apache Maven версии 3.\*

- Tomcat версии 7.\*

Для начала нужно собрать исходники проекта в war файл. Это выполняется с помощью Mavena(команда mvn package). Чтобы это выполнить, нужно перейти в папку с исходниками проектами (в ней находится .pom файл) и выполнить команду “mvn package”.

После этого должна появиться папка Target, в которой будет war файл SpringExample.war.

Теперь этот файл нужно скопировать в папку Tomcata - webapp. Это делается для того, чтоб при запуске томката он автоматически развернул war файл и запустил его. Tomcat запускается скриптом startup из папки bin.

После всех этих действий, проект будет доступен по адрессу <http://localhost:port/warname> , где

port – порт на котором запускается Tomcat; warname – имя War файла.

Видео запуска без IDE - <http://screencast.com/t/tEDioL6Y>

**Validation:**

В Spring присутствует возможность валидации бинов (Bean Validation) JSR-303. Для валидации мы будем использовать Hibernate Validation. Рассмотрим валидацию на примере класса User, который содержит сущностью аутентифицированного пользователя.

Для использования валидации необходимо создать POJO класс. **POJO** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) ***Plain Old Java Object***) — «простой Java-объект в старом стиле», простой [Java](http://ru.wikipedia.org/wiki/Java)-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели.

Итак, мы создали POJO класс для пользователя. У нас он также будет использоваться как доменная модель, которая будет сохраняться в базе данных и оттуда извлекаться. Т.к. этот класс реализует «бизнес-модель», то нам надо снабдить ее необходимыми и не избыточными свойствами. То есть добавить приватные поля, который будут отражать сущность модели, а также публичные геттеры и сеттеры для них. В данном случае нам в первую очередь понадобятся поля “username” (логин пользователя), “lastname” (фамилия), “firstname” (имя), “passwordHash” (хеш от пароля, т.к. хранить пароли в базе в открытом виде или даже зашифрованном – не секурно и они могут быть скомпроментированы. По правде говоря в таком виде их тоже лучше не хранить.), “sex” (пол пользователя мужчина/женщины/иное), “progLang” (языки программирования), “receiveNewsLetter” (хочет ли пользователь получать почтовую рассылку), “country” (страна проживания пользователя).

Итак, мы выбрали достаточную модель пользователя – теперь надо аннотировать необходимые поля ограничениями. Для этого нам понадобится пространство имен org.hibernate.validator.constraints.\*. Здесь присутствуют наиболее часто используемые валидаторы, среди них: ConstraintComposition, CreditCardNumber, Email, Length, ModCheck, NotBlank, NotEmpty, Range, SafeHtml, ScriptAssert, URL. Нам, к счастью, понадобятся пользо NotNull, NotBlank. NotNull явно указывает что в данном поле должно что-то присутствовать, а NotBlank указывает что оно не должно быть null, и после того как его тримнули (убрали пробелы в начале и в конце строки) его длина должна быть не нулевой. Пример использования вы можете посмотреть в папке src/main/java/model/entity/User.

В результате всех этих действий можно указывать в \*.jsp поля ошибок и они сами будут подставляться если поле не было провалидировано. Это выглядит так:

**…**

**<form:errors path="\*" cssClass="errorblock" element="div"/>**

**…**

**<form:errors path="firstname" cssClass="error"/>**

**….**

Полный код файла вы можете найти в репозитории. В данном случае это src/main/webapp/WEB-INF/pages/registration.jsp.

Также будет полезно знать что можно задавить локализацию и настраиваемое отображение ошибок в файле src/main/resources/messages.properties. Выглядит это так:  
**NotBlank.user.username = Username may not be empty!**

**NotBlank.user.firstname = First name may not be empty!**

**NotBlank.user.lastname = Last name may not be empty!**

Также можно создавать свои собственные валидаторы, но мы это рассматривать пока не будем.