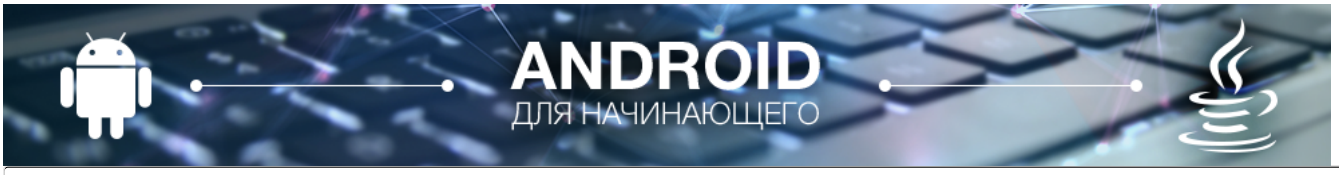
DevColibri

Главная



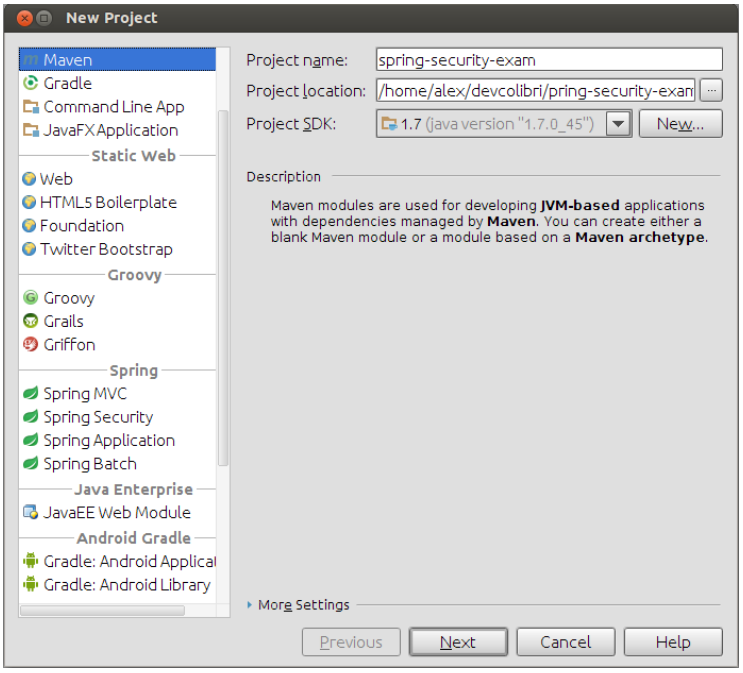
Быстрый старт в Spring Security

fB этом уроке будет рассмотрен пример реализации аутентификации пользователя используя Spring Security.

Скачать

Шаг 1. Создание проекта и подключение зависимостей

Начнем с традиционного создания проекта:



Создали Maven проект, теперь добавим зависимости в **pom.xml** их будет довольно таки много, поэтому прикладываю полный pom файл:

```
<?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>spring-security-exam</groupId>
    <artifactId>spring-security-exam</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <spring.mvc>4.0.0.RELEASE</spring.mvc>
        <javax.servlet>3.0.1</javax.servlet>
        <jstl.version>1.2</jstl.version>
        <spring.securiry>3.2.0.RELEASE</spring.securiry>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.mvc}</version>
        </dependency>

        <dependency>
            groupId>org.springframework</groupId>
            artifactId>spring-web</artifactId>
            version>${spring.mvc}</version>
```

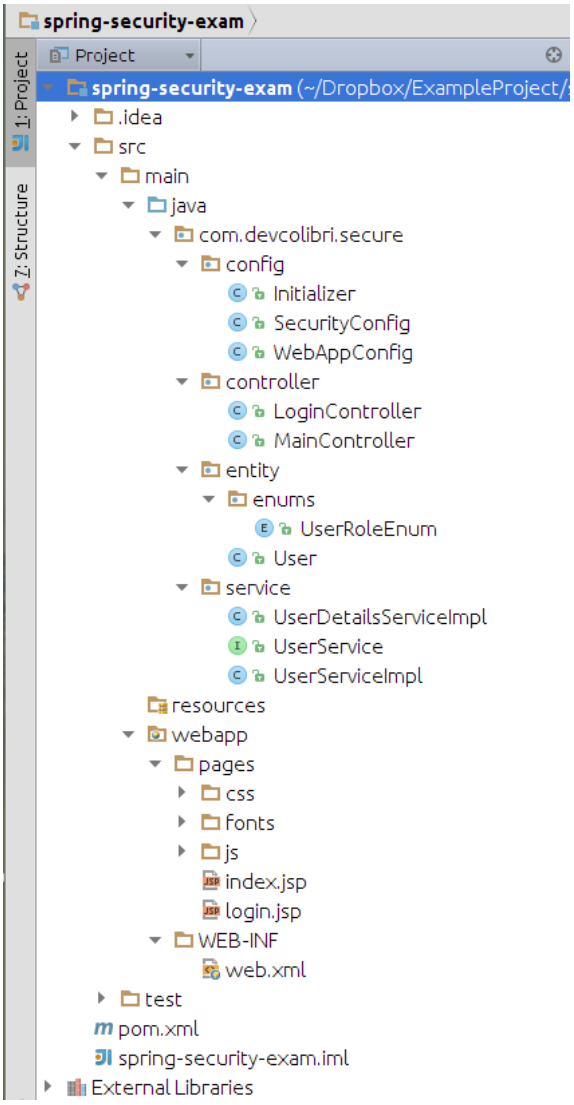

```
endency>

ndency>
groupId>javax.servlet</groupId>
artifactId>javax.servlet-api</artifactId>
version>${javax.servlet}</version>
scope>provided</scope>
endency>

ndency>
groupId>jstl</groupId>
artifactId>jstl</artifactId>
version>${jstl.version}</version>
endency>

ndency>
groupId>org.springframework.security</groupId>
artifactId>spring-security-web</artifactId>
version>${spring.securiry}</version>
endency>

ndency>
groupId>org.springframework.security</groupId>
artifactId>spring-security-core</artifactId>
version>${spring.securiry}</version>
endency>

ndency>
groupId>org.springframework.security</groupId>
artifactId>spring-security-config</artifactId>
version>${spring.securiry}</version>
endency>
```

```
62
63          <dependency>
64              <groupId>org.springframework.security</groupId>
65              <artifactId>spring-security-taglibs</artifactId>
66              <version>${spring.securiry}</version>
67          </dependency>
68
69      </dependencies>
70
71      <build>
72          <finalName>secure-exam</finalName>
73          <plugins>
74              <plugin>
75                  <artifactId>maven-compiler-plugin</artifactId>
76                  <configuration>
77                      <source>1.7</source>
78                      <target>1.7</target>
79                  </configuration>
80              </plugin>
81          </plugins>
82      </build>
83
84  </project>
```

Наверх

му из зависимостей не буду останавливаться, но включил самые обязательные зависимости для реализации Spring Security.

покажу вам какая должна быть структура проекта, чтобы вы не терялись в создании классов и файлов.



Шаг 2. Начинаем с конца, делаем внешний вид

Для большей привликательности, как к внешнему виду так и мотивации изучить этот урок я решил использовать Bootstrap 3.

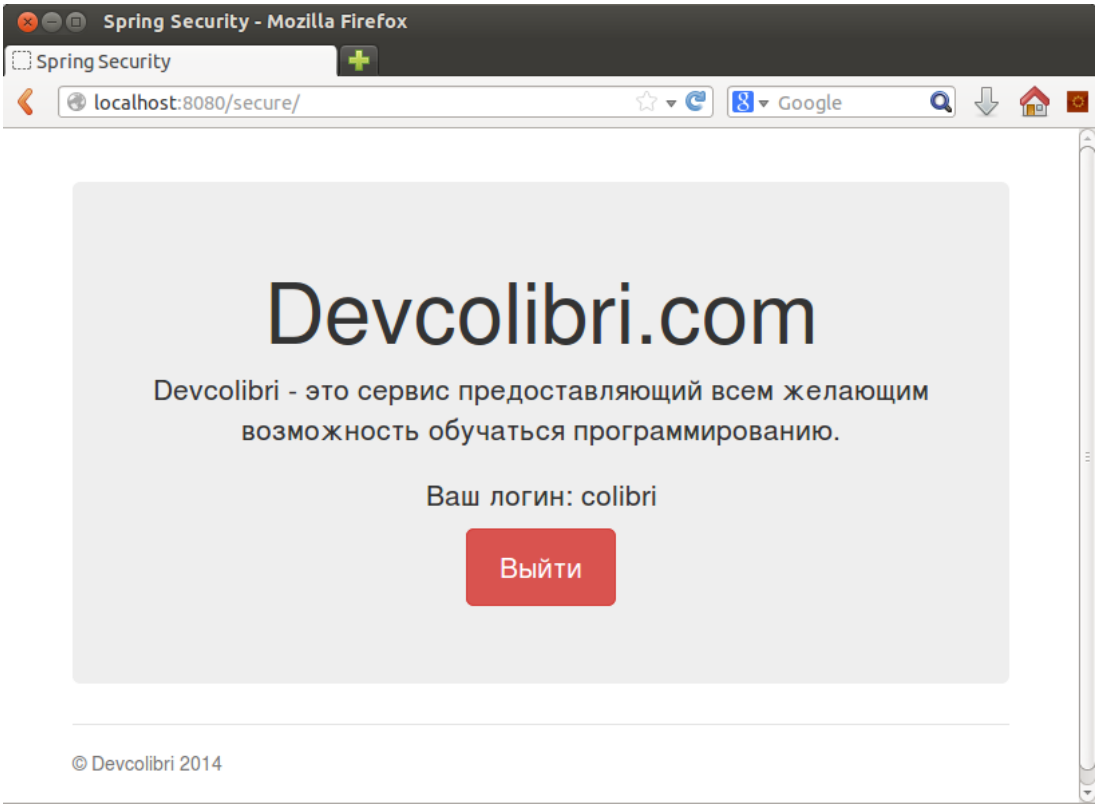
Скачиваем bootstrap и в проекте создаем папку **webapp/pages/** и копируем туда файлы bootstrap-а. (*смотрите на структуре проекта*)

Теперь создаем webapp/pages/**index.jsp:**

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="">

    <title>Spring Security</title>

    <!-- Bootstrap core CSS -->
    <link href="<c:url value="/pages/css/bootstrap.css" />" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="<c:url value="/pages/css/jumbotron-narrow.css" />" rel="stylesheet">

    <!-- HTML5 shim and Respond.js IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
    <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
    <script src="https://oss.maxcdn.com/libs/respond.js/1.3.0/respond.min.js"></script>
    <![endif]-->
</head>

<body>

<div class="container">

    <div class="jumbotron" style="margin-top: 20px;">
        <h1>Devcolibri.com</h1>
        <p class="lead">
            Devcolibri - это сервис предоставляющий всем желающим возможность обучаться программированию.
        </p>
```

```
        <sec:authorize access="!isAuthenticated()">
            <p><a class="btn btn-lg btn-success" href="<c:url value="/login" />" role="button">Войти</a></p>
        </sec:authorize>
        <sec:authorize access="isAuthenticated()">
            <p>Ваш логин: <sec:authentication property="principal.username" /></p>
            <p><a class="btn btn-lg btn-danger" href="<c:url value="/logout" />" role="button">Выйти</a></p>

        </sec:authorize>
    </div>

    <div class="footer">
        <p>© Devcolibri 2014</p>
    </div>

</div>
</body>
</html>
```

Выглядеть эта страница будет так:



Создаем следующую страницу на которой собственно и будет происходить вход на сайт webapp/pages/**login.jsp**:

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="">

    <title>Spring Security</title>

    <!-- Bootstrap core CSS -->
    <link href="<c:url value="/pages/css/bootstrap.css" />" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="<c:url value="/pages/css/signin.css" />" rel="stylesheet">

    <!-- HTML5 shim and Respond.js IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
    <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
    <script src="https://oss.maxcdn.com/libs/respond.js/1.3.0/respond.min.js"></script>
    <![endif]-->
</head>

<body>

<div class="container" style="width: 300px;">
    <c:url value="/j_spring_security_check" var="loginUrl" />
    <form action="${loginUrl}" method="post">
        <h2 class="form-signin-heading">Please sign in</h2>
        <input type="text" class="form-control" name="j_username" placeholder="Email address" required autofocus value="colibri"
        <input type="password" class="form-control" name="j_password" placeholder="Password" required value="1234">
        <button class="btn btn-lg btn-primary btn-block" type="submit">Войти</button>
    </form>
</div>

</body>
</html>
```

Тут обратите особое внимание на обя тега input, а именно на их name:

name=»**j_username**»

name=»**j_password**»

В данном случае эти поля должны быть именно с такими значениями.

А также вы возможно уже увидели эту строку:

```
<c:url value="/j_spring_security_check" var="loginUrl" />
```

так мы создали переменную, которая будет хранить ссылку на security check, он выполняет аутентификация, где значение **value** обязательн таким.

Выглядеть она будет так:

Думаю получилось не плохо :)

Шаг 3. Создаем контроллеры

О создании MVC проекта на Spring можно почитать урок Spring MVC Hello World

Создаем пакет **controller** и класс **MainController**:

```
package com.devcolibri.secure.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/")
public class MainController {

    @RequestMapping(method = RequestMethod.GET)
    public String start(Model model){
        return "index";
    }

}
```

Этот контроллер будет просто направлять пользователя на страницу **index.jsp.**

А теперь создадим второй аналогичный контроллер, который будет показывать пользователю страничку **login.jsp.**

Создаем класс и называем его **LoginController:**

```
package com.devcolibri.secure.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/login")
public class LoginController {

    @RequestMapping(method = RequestMethod.GET)
    public String loginPage(Model model){
        return "login";
    }

}
```

Теперь мы сможем ходить по страничкам.

Шаг 4. Конфигурирование Spring MVC

Теперь нам нужно сконфигурировать Spring MVC чтобы он имел возможность разворачивать свой контекс и мы могли получать доступ к наш

Создаем в пакете config класс **WebAppConfig** и наследуем его от WebMvcConfigurerAdapter:

```
package com.devcolibri.secure.config;

import com.devcolibri.secure.service.UserDetailsServiceImpl;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;
import org.springframework.web.servlet.view.UrlBasedViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan("com.devcolibri")
public class WebAppConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/pages/**").addResourceLocations("/pages/");
    }

    @Bean
    public UrlBasedViewResolver setupViewResolver() {
        UrlBasedViewResolver resolver = new UrlBasedViewResolver();
        resolver.setPrefix("/pages/");
        resolver.setSuffix(".jsp");
        resolver.setViewClass(JstlView.class);

        return resolver;
    }

}
```

Подробную информацию о **Java Config** для Spring вы можите посмотреть в уроке Spring 3. JavaConfig на примере Spring MVC.

Шаг 5. Создание Entity

Так как цель урока показать пример использования Spring Security, то я не стал реализовывать работу с БД, а всего лишь создал все необх этого, чтобы в будущем реализовать Spring DATA.

Создаем entity по сути это простой класс, так как мы не используем не ORM фреймворков, назовем его User:

```
package com.devcolibri.secure.entity;

public class User {

    private String login;
    private String password;

    public User(String login, String password) {
        this.login = login;
        this.password = password;
    }

    public User() {

    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

В будущем вы можите этот объект пополнять своими свойствами, но для примера нам достаточно знать **Login** (*логин*) и **Password** (*пароль*).

И еще в этом же пакете создадим пакет enums и в нем создаем enum **UserRoleEnum,** который будет определять роли пользователя**:**

```
package com.devcolibri.secure.entity.enums;

public enum UserRoleEnum {

    ADMIN,
    USER,
    ANONYMOUS;

    UserRoleEnum() {
    }

}
```

Теперь у нас есть 3 роли, которые мы немного позже будем использовать.

Шаг 6. Создаем слой Services

Обычно проекты имеют несколько уровней слоёв, о которых я еще напишу статью, но один из этих слоёв мы реалезуем прям сейчас.

Нам нужно реальзовать Service tier (*сервис слой либо уровень обслуживания*). Создаем пакет service и в нем создаем интерфейс **UserServic**

```
package com.devcolibri.secure.service;

import com.devcolibri.secure.entity.User;

public interface UserService {

    User getUser(String login);

}
```

Этот сервисный интерфейс говорит о том, что у нас будет сервис который позволит получать User не важно как и откуда, но он позволит на

А теперь давайте напишем первую реализацию данного интерфейса, для этого создаем на том же уровне класс **UserServiceImpl**:

```
package com.devcolibri.secure.service;

import com.devcolibri.secure.entity.User;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl implements UserService {

    @Override
    public User getUser(String login) {
        User user = new User();
        user.setLogin(login);
        user.setPassword("7110eda4d09e062aa5e4a390b0a572ac0d2c0220");

        return user;
    }

}
```

Как видите реализация довольно простая, тут мы просто заполняем объект **User** используя **setters**, но мы также можем в этом севис методе из DAO, который бы достал нам этого юзера с БД например либо получил бы его с Web Service.

Обратите внимание, что я установил в поле Password специфичный пароль а именно зашифрованый в **SHA1** формате.

То есть я взял пароль «**1234**» и зашифровал его в **SHA1** формат с помощью online сервиса SHA1 online generator и получил уже зашифрован «**7110eda4d09e062aa5e4a390b0a572ac0d2c0220**«.

Шаг 7. Создаем реализацию UserDetailsService

Для того, чтобы связать наш сервис UserService со Spring Security, нам нужно реализовать специальный интерфейс фреймворка Spring Secu позволит выполнять аутентификацию пользователя на основании данных полученых с UserService написанного высше.

Создаем класс **UserDetailsServiceImpl** и реализуем его от UserDetailsService:

```
package com.devcolibri.secure.service;

import com.devcolibri.secure.entity.User;
import com.devcolibri.secure.entity.enums.UserRoleEnum;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.HashSet;
import java.util.Set;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserService userService;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        // с помощью нашего сервиса UserService получаем User
        User user = userService.getUser("colibri");
        // указываем роли для этого пользователя
        Set<GrantedAuthority> roles = new HashSet();
        roles.add(new SimpleGrantedAuthority(UserRoleEnum.USER.name()));

        // на основании полученныйх даных формируем объект UserDetails
        // который позволит проверить введеный пользователем логин и пароль
        // и уже потом аутентифицировать пользователя
        UserDetails userDetails =
                new org.springframework.security.core.userdetails.User(user.getLogin(),
                                                                       user.getPassword(),
                                                                       roles);

        return userDetails;
    }

}
```

Этот сервис и является основной логикой аутентификаци с использованием Spring Security.

Шаг 8. Добавляем инициализацию бина UserDetailsServiceImpl в конфигурацию WebA

Для того чтобы наша реализация интерфейса UserDetailsService смогла инициализыроваться контейнером Spring нам нужно добавить его в

```
@Bean
public UserDetailsService getUserDetailsService(){
    return new UserDetailsServiceImpl();
}
```

После этого Spring контейнер будет знать какую реализацию интерфейса UserDetailsService нужно брать.

Шаг 9. Конфигурирование Spring Security

Теперь самое главное, заставить все это заработать :)

Создаем в пакете config класс **SecurityConfig** и наследуем его от WebSecurityConfigurerAdapter:

```
package com.devcolibri.secure.config;

import com.devcolibri.secure.service.UserDetailsServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.encoding.ShaPasswordEncoder;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    // регистрируем нашу реализацию UserDetailsService
    // а также PasswordEncoder для приведения пароля в формат SHA1
    @Autowired
    public void registerGlobalAuthentication(AuthenticationManagerBuilder auth) throws Exception {
        auth
                .userDetailsService(userDetailsService)
                .passwordEncoder(getShaPasswordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // включаем защиту от CSRF атак
        http.csrf()
                .disable()
                // указываем правила запросов
                // по которым будет определятся доступ к ресурсам и остальным данным
                .authorizeRequests()
                .antMatchers("/resources/**", "/**").permitAll()
                .anyRequest().permitAll()
                .and();

        http.formLogin()
                // указываем страницу с формой логина
                .loginPage("/login")
                // указываем action с формы логина
                .loginProcessingUrl("/j_spring_security_check")
                // указываем URL при неудачном логине
                .failureUrl("/login?error")
                // Указываем параметры логина и пароля с формы логина
                .usernameParameter("j_username")
                .passwordParameter("j_password")
                // даем доступ к форме логина всем
                .permitAll();

        http.logout()
                // разрешаем делать логаут всем
                .permitAll()
                // указываем URL логаута
                .logoutUrl("/logout")
                // указываем URL при удачном логауте
                .logoutSuccessUrl("/login?logout")
                // делаем не валидной текущую сессию
                .invalidateHttpSession(true);

    }

    // Указываем Spring контейнеру, что надо инициализировать <b></b>ShaPasswordEncoder
    // Это можно вынести в WebAppConfig, но для понимаемости оставил тут
    @Bean
    public ShaPasswordEncoder getShaPasswordEncoder(){
        return new ShaPasswordEncoder();
    }

}
```

Это базовая конфигурация, которая нужна для наших нужд, она может расширятся.

Шаг 10. Регистрация конфигураций в контексте Spring

Чтобы Spring видел наши конфигурации, а именно WebAppConfig и SecurityConfig их нужно зарегистрировать в контексте Spring.

Создаем в пакете config класс **Initializer** и реализуем WebApplicationInitializer:

```
package com.devcolibri.secure.config;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration.Dynamic;

public class Initializer implements WebApplicationInitializer {

    private static final String DISPATCHER_SERVLET_NAME = "dispatcher";

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();
        // регистрация конфигураций в Spring контексте
        ctx.register(WebAppConfig.class);
        ctx.register(SecurityConfig.class);
        servletContext.addListener(new ContextLoaderListener(ctx));

        ctx.setServletContext(servletContext);

        Dynamic servlet = servletContext.addServlet(DISPATCHER_SERVLET_NAME, new DispatcherServlet(ctx));
        servlet.addMapping("/");
        servlet.setLoadOnStartup(1);
    }
}
```

На этом этап конфигурации проекта можно щитать законченым.

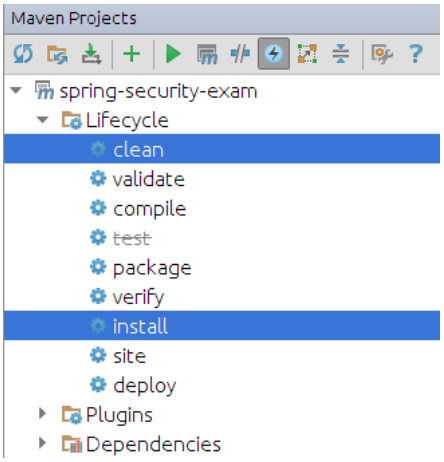
В **web.xml** нужно добавить фильтр, который будет определять наши реквесты и проверять валидность сессии:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

Шаг 11. Развертывание и тестирование

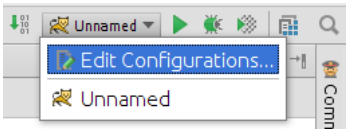
Пора все собрать и развернуть на сервере приложений.

Собираем проект с помощью Maven:

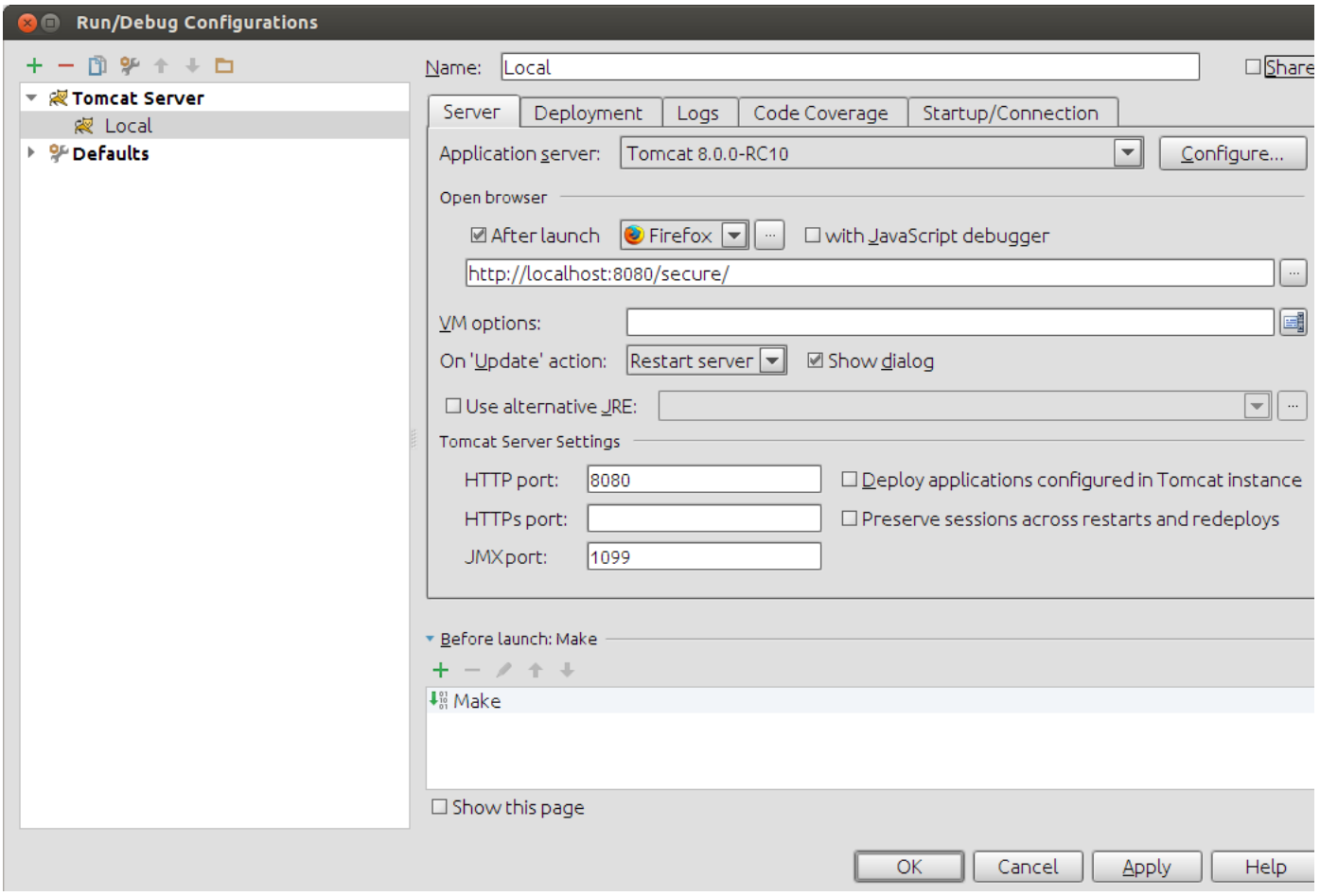


Теперь развертываем собранный собраный war на сервере приложений, я выбрал Tomcat 8.

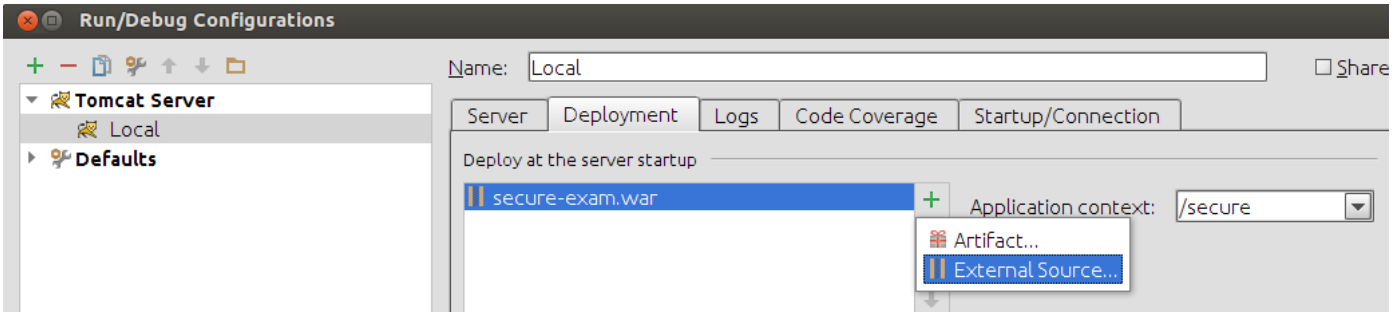
Конфигурируем Intellij IDEA под Tomcat. Заходим в **Edit Configuration...**



Нажимаем по плюсику (**Add New Configuration...**) ищем там **Tomcat Server -> Local:**

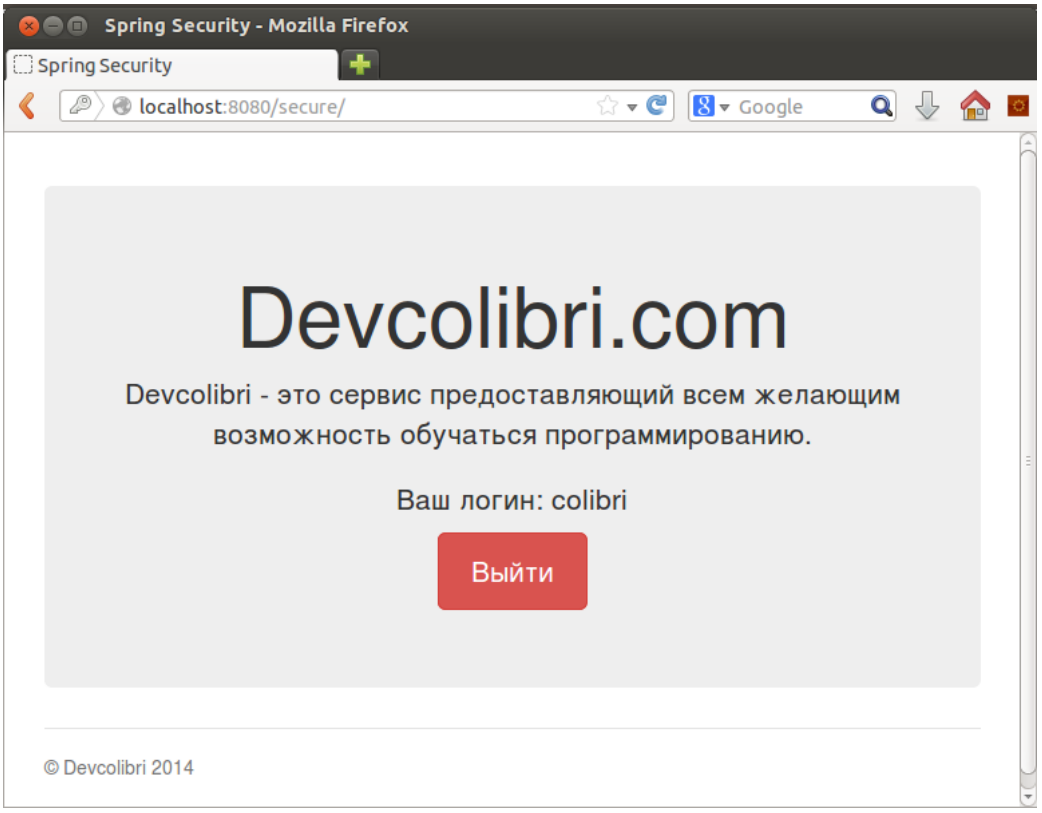


Конфигурируем также как показанно на скриншоте, потом переходим в раздел **Deployment**:



Жмем **Add -> External Source...** в корне вашего проекта будет папка **target** в ней будет **secure-exam.war** выбираем его и в поле **Application co /secure.**

После этого запускаем. Попадпем на главное окно жмем Войти переходим на форму с логином и нажимаем Войти, после этого вас перенапр страницу где вы увидите свой логин.



На этом все. Надеюсь данный урок вам поможет.

Урок создан: 28 Январь 2014 | Просмотров: 33133 | Автор: Александр Барчук | Правила перепечатки

Добавить комментарий

Комментарии:

Дмитрий 19 I

Спасибо за статью! еще бы ссылочку для скачивания
Ответить

Александр Барчук 19 I

НА начале урока есть кнопка СКАЧАТЬ :)
Ответить

Дмитрий 19 I

:) меня можно было сразу посылать к окулисту после такого комментария. А куда жертвовать деньги за такие статьи?
Ответить

Александр Барчук 20 I

:) Как я понял вам понравился урок :)
Ответить

Askat 02

Как в eclipse открыть?
Ответить

Armen 22 I

вот это пост! первый раз вижу настоящий спринг, и руку программиста.
Ответить

Александр Барчук 22 I

Спасибо, приятно слышать :)
Ответить

Геннадий 28 I

Замечательная статья, кратко и все по делу.
Ответить

Геннадий 28 I

Для того чтобы наша реализация интерфейса UserDetailsService согла — опечатка 8 пункт
Ответить

Александр Барчук 28 I

Спасибо, поправил.
Ответить

Геннадий 01 Ап

Шаг 8. Добавляем инициализацию бина UserDetailsServiceUmpl в конфигурацию WebAppConfig — тоже опечатка — UserDetailsService
Ответить

Александр Барчук 01 Ап

Геннадий, спасибо)) Исправил.
Ответить

Михаил 31 I

Доброго времени! Столкнулся с одной неприятностью. На шаге 7 @Autowired private UserService userService; возвращает null, а точнее любой класс в имплементации UserDetailsService возвращает null.
Ответить

Михаил 31 I

А нет, это я не внимательный! всё заработало

И шаг 10 у меня реализован иначе. Не уверен, есть ли принципиальная разница, но всё же:

```
public class Init extends AbstractAnnotationConfigDispatcherServletInitializer {
@Override
public void onStartup(ServletContext servletContext) throws ServletException {
super.onStartup(servletContext);
// Setting container parameters
servletContext.setInitParameter(«defaultHtmlEscape», «true»);

// Register character encoding filter
FilterRegistration charEncodingFilterReg = servletContext.addFilter(«CharacterEncodingFilter», CharacterEncodingFilter.class);
charEncodingFilterReg.setInitParameter(«encoding», «UTF-8”);
charEncodingFilterReg.setInitParameter(«forceEncoding», «true»);
charEncodingFilterReg.addMappingForUrlPatterns(null, false, «/*»);

}

@Override
protected Class[] getRootConfigClasses() {
return new Class[] {SecurityConfig.class, WebMvcConfig.class};
}

@Override
protected Class[] getServletConfigClasses() {
return new Class[] {AppConfig.class};
}

@Override
protected String[] getServletMappings() {
return new String[] { «/» };
}
}
```

Ответить

Вася 01 Ап

Описать радость которую принесла даная статья просто невозможно!
Ответить

Alex 07 Ап

Отличная статья. Да и ресурс хороший)
очепятка — «Создаем класс UserDetailsServiceImpl и реализуем его от UserDetailsServiceImpl:»
Ответить

Александр Барчук 08 Ап

Спасибо за исправление)
Ответить

Дмитрий 10 А

Практически вся конфигурация приложения без использования xml, но в web.xml присутствует пару строчек, без которых авторизация не ра
пожалуйста для чего они и возможно добавьте в туториал, без них у меня не работало, пока не скачал проект и не сравнил, а Вам лично спа
Ответить

Александр Барчук 10 А

Спасибо за правку, добавил.
Ответить

Богдан 16 Сент

Или если работать без web.xml, а с JavaConfig нужно добавить такой класс
public class SpringSecurityInitializer extends AbstractSecurityWebApplicationInitializer {
}
Ответить

Егор 22 А

Шаг 8 делать не нужно, т.к. в 7-м шаге проставлена аннотация @Service. Верно?
Ответить

Александр Барчук 25 А

Я могу ошибаться, но думаю нужно, так как интерфейс спринговый а реализация наша, и думаю что спрингу не будет видно эту реализа
Ответить

Егор 25 А

Вроде работает, а объяснение,как мне кажется, следующее: мы пометили класс аннотацией @Service, поэтому спринг его считает би
автоматически сам подымет с контекстом. А т.к. в нашем приложении всего одна реализация интерфейса UserDetailsService, то в 9-м
спокойно делаем @Autowired.
Ответить

Антон 29 А

У меня на шаге 10 возникли проблемы
servletContext.addListener(new ContextLoaderListener(ctx)); addListener подсвечен красным
servletContext.addServlet(DISPATCHER_SERVLET_NAME, new DispatcherServlet(ctx)); addServlet подсвечен красным
и следовательно не запускается
как можно исправить или какой web.xml должен быть для него?
Ответить

Генри 07 Ян

А можете рассказать поподробнее, как именно в этом примере реализовать регистрацию пользователя и запись данных в БД используя hiber
данным котоыре хранятся в БД. Ибо я так понял, тут при логине просто сверняются данные которые мы прописали в нашем классе userservic
Ответить

mega koder 11 Ян

${spring.securiry} — secutiry*
Ответить

leming 27 Ян

Очепятка ${spring.securiry}
Ответить

Илья 04 I

Столкнулся с проблемой, во время запуска выскакивает ошибка 04-Mar-2015 12:48:30.449 WARNING [http-apr-8080-exec-10]
org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver.handleHttpRequestMethodNotSupported Request method 'HEAD' no

не могу решить проблему!

Ответить

Евгений 05 I

получилось разобраться довольно бытро) Спасибо
Ответить

Константин 05

Спасибо, всё хорошо и понятно описано.
Ждем-с статью про регистрацию новых пользователей :)
Ответить

Нияз 01 I

Спасибо огромное и низкий поклон автору сайта за такой содержательный и полезный материал! Я все сделал, как написано, но у меня поче
страницах не распознаются теги: taglib uri, prefix пишет : taglib uri is not allowed here, форумы тоже особо ничего не дали. С остальными опи
ситуация тоже такая же, как с uri. Еще не видит пути. Не могли бы подсказать как правильно сделать, что бы все заработало. Был бы очень
Ответить

Антон 24 I

В зависимости добавь jstl core
Ответить

Andrey 26 I

Зачем в проекте томкат из idea? Ведь те кто изучают язык явно используют community-edition. Проще добавить к зависимостям плагин, хоть
За урок спасибо.

org.apache.tomcat.maven
tomcat7-maven-plugin
2.2

/
Ответить

Рустам 29 I

Здравствуйте. Есть несколько вопросов, которые просто уже некому задать. Я в полном замешательстве. Не могли бы вы написать мне на e-r
связаться с вами???
Ответить

Евгений 04 Сент

Добрый день, прочитал вашу статью, очень интересно, пробую на примере, все получилось. НО, как только попробовал настроить Матчеры r
тут началась беда. Вопрос, если брать на данном примере как настроить доступ к ресурсам в зависимости от роли? Как я сделал:
http.csrf()
.disable()
// указываем правила запросов
// по которым будет определятся доступ к ресурсам и остальным данным
.authorizeRequests()
.antMatchers(«/login», «/registration»).permitAll()
.antMatchers(«/»).hasAnyRole(Roles.USER.name(), Roles.ADMIN.name(), Roles.ANONYMOUS.name())
.and();
При успешной авторизации меня перебрасывает на корневую страничку приложения (http://localhost:9999/myapp/) и выпадает ошибка 403.
происходит? Я ведь указал, что пользователь с любой из перечисленных ролей может посетить данную страницу. Что не так, подскажите по
Ответить

Евгений 04 Сент

Пытаюсь проверить в контроллере корректно ли инициализируется юзер:
UserDetails userDetails = (UserDetails) SecurityContextHolder.getContext().
getAuthentication().getPrincipal();
System.out.println(userDetails.getPassword());
System.out.println(userDetails.getUsername());
System.out.println(userDetails.getAuthorities().toString());
получаю такой вывод:
null
colibri
USER
Ответить

Максим 29 Окт

У меня входит даже если я вместо логина пишу любую цифру, в чем может быть проблема? as — wildfly, скачал архив
Ответить

Vadim 02 Но

Максим. Это нормально. Там нет проверки на логин. В SecurityConfig передаётся userDetailsService, который в свою очередь, берёт логин UserServiceImpl. Пароль там задаётся зашифрованным, а логин будет любым

Ответить

Максим 02 Дек

Благодарю за ответ) уже привязал с хибернейтом) все ок)

Ответить

Константин 04 Но

Просто великолепная статья, спасибо огромное)

Ответить


Андрей 23 Но

У меня ошибка «Could not autowire. No beans of 'UserDetailsServiceImpl' type found. (at line 23) «

Ответить


Андрей 23 Но

И еще у меня «UserServiceImpl» подсвечивается как неиспользуемый

Ответить

Renat 23 Но

В браузере название вкладки — Быстрый страт) Исправь)

Ответить

Илья 27 Но

Немного не понял вот этот момент:
// включаем защиту от CSRF атак
http.csrf()
.disable()
Насколько я знаю, по умолчанию защита включена, а эта строчка ее как раз выключает. Так ведь?

Ответить

Дмитрий 29 Но

Спасибо за отличную статью! У меня вопрос. Обязательна ли эта переменная , ведь url можно было сразу в форму прописать?

Ответить

Дмитрий 29 Но

c:url value=»/j_spring_security_check» var=»loginUrl»

Ответить

КАТЕГОРИИ

Я Девелопер (1)

Devcolibri

Devcolibri
Подписаны 4 819 человек

Подписаться