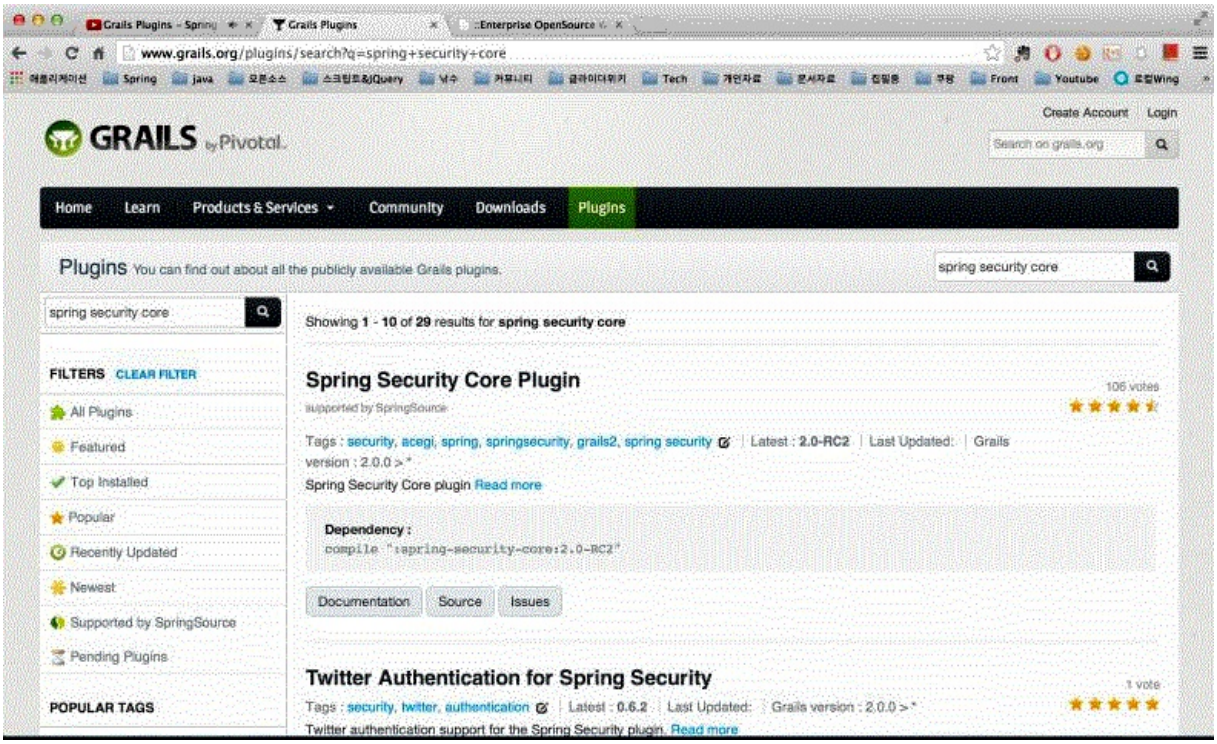
004.Grails와 Spring Framework (part1.Spring Security Core Plugin 적용)

Spring Security Core plugin

Table of Contents

http://www.grails.org/plugins/search?q=spring+security+core 검색



Grails 공식 사이트에서 Plugins 메뉴에서 Spring security core 로 검색해보면 해당 플러그인을 검색할 수 있습니다.

제목을 클릭하고 들어가면 아래와 같은 내용을 확인 할 수 있습니다.

여기서 실제 우리가 어플리케이션에서 필요한 사항은 아래의 두가지 입니다.

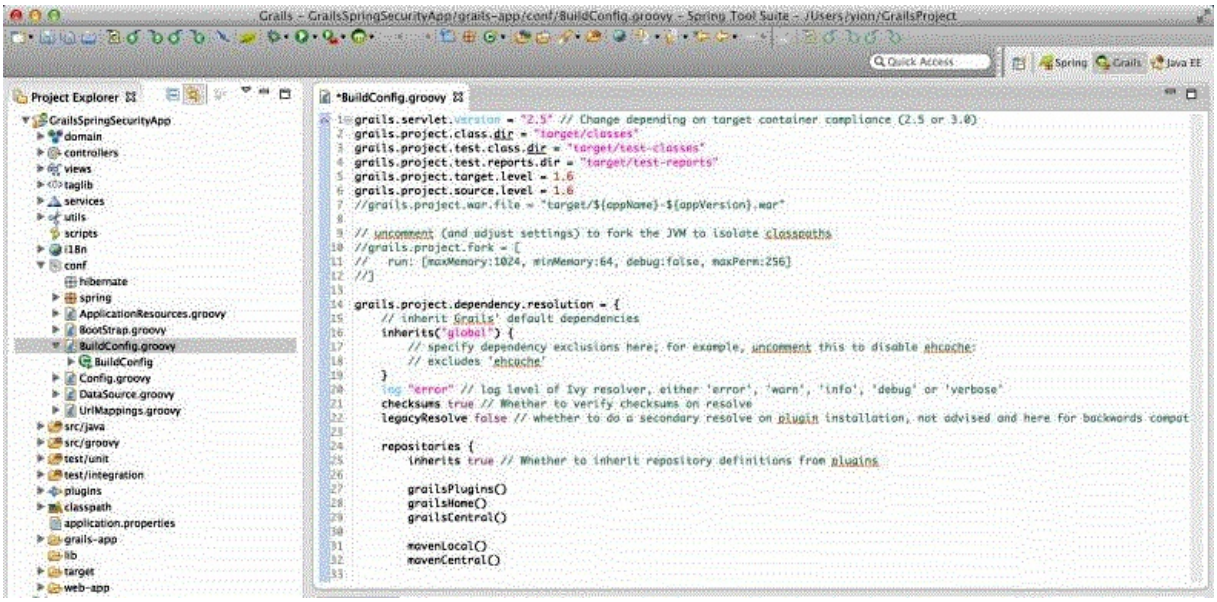
Dependency :
compile ":spring-security-core:2.0-RC2"

Custom repositories :
mavenRepo "http://repo.spring.io/milestone/"

어플리케이션에 관련 jar를 추가하기 위해 Dependency 와 Repository를 추가 해 주어야 합니다.
이 부분은 conf/BuildConfig.groovy 파일에 수정함으로써 적용할 수 있습니다.

일단 새 Grails Project 를 생성합니다. 여기서는 GrailsSpringSecurityApp 으로 생성 했습니다.

BuildConfig.groovy 파일을 열어서 내용을 편집합니다.

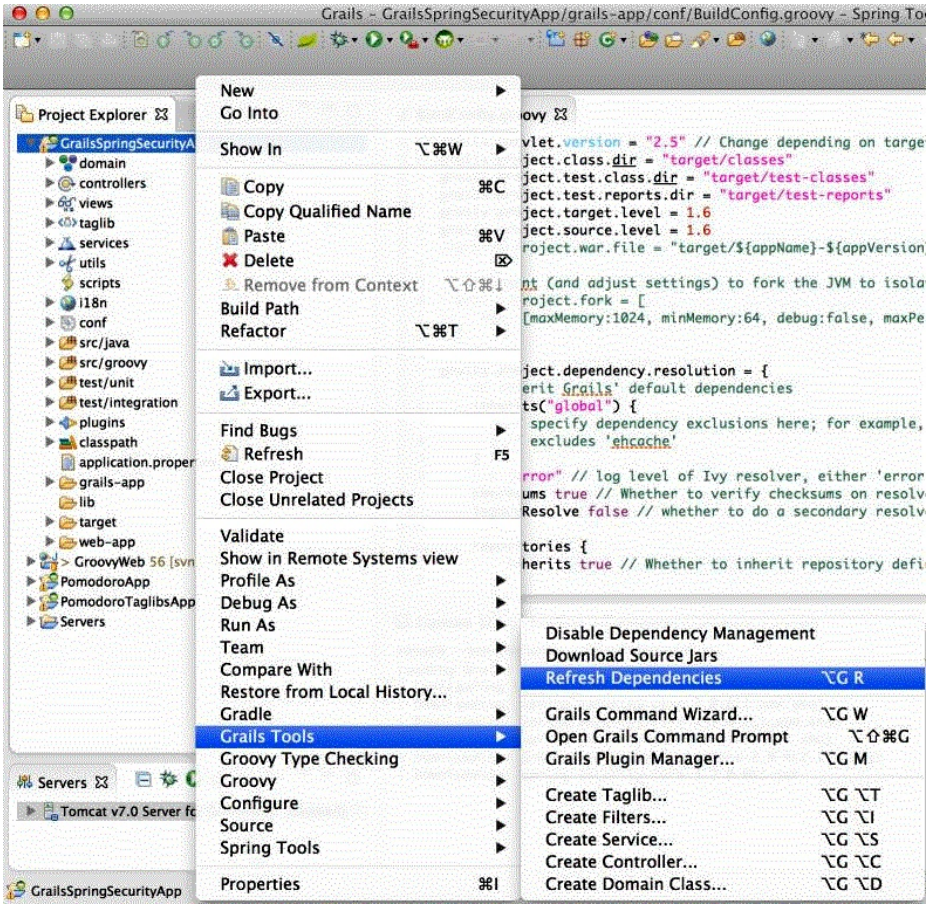


plugins 부분에 (아래 소스에서는 52라인) Dependency에 나와있는 compile ":spring-security-core:2.0-RC2" 를 추가해주었습니다.
또한 repositories 부분(아래 소스에서는 39라인) Custom repositories 에 명시되어 있는 mavenRepo "http://repo.spring.io/milestone/" 부분을 추가해 주었습니다.

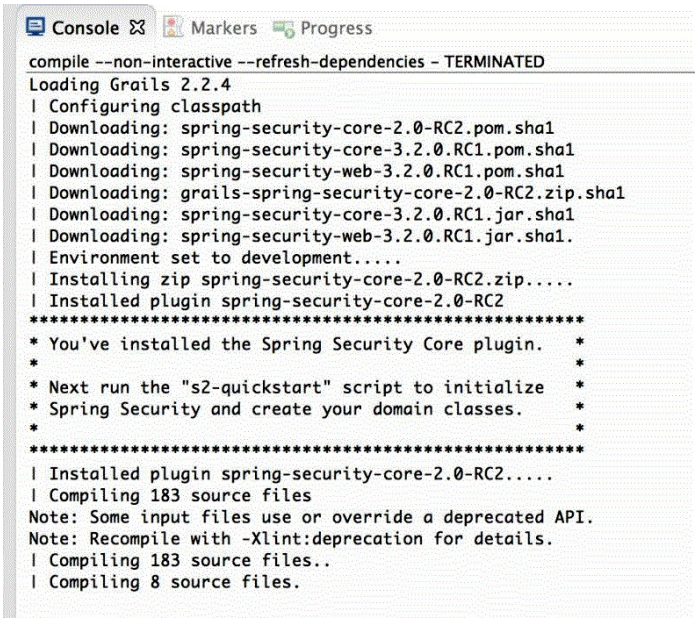
```
1  grails.servlet.version = "2.5" // Change depending on target container compliance (2.5 or 3.0)
2  grails.project.class.dir = "target/classes"
3  grails.project.test.class.dir = "target/test-classes"
4  grails.project.test.reports.dir = "target/test-reports"
5  grails.project.target.level = 1.6
6  grails.project.source.level = 1.6
7  //grails.project.war.file = "target/${appName}-${appVersion}.war"
8
9  // uncomment (and adjust settings) to fork the JVM to isolate classpaths
10 //grails.project.fork = [
11 //   run: [maxMemory:1024, minMemory:64, debug:false, maxPerm:256]
12 //]
13
14 grails.project.dependency.resolution = {
15     // inherit Grails' default dependencies
16     inherits("global") {
17         // specify dependency exclusions here; for example, uncomment this to disable ehcache:
18         // excludes 'ehcache'
19     }
20     log "error" // log level of Ivy resolver, either 'error', 'warn', 'info', 'debug' or 'verbose'
21     checksums true // Whether to verify checksums on resolve
22     legacyResolve false // whether to do a secondary resolve on plugin installation, not advised and here for backwards compatibility
23
24     repositories {
25         inherits true // Whether to inherit repository definitions from plugins
26
27         grailsPlugins()
28         grailsHome()
29         grailsCentral()
30
31         mavenLocal()
32         mavenCentral()
33
34         // uncomment these (or add new ones) to enable remote dependency resolution from public Maven repositories
35         //mavenRepo "http://snapshots.repository.codehaus.org"
36         //mavenRepo "http://repository.codehaus.org"
37         //mavenRepo "http://download.java.net/maven/2/"
38         //mavenRepo "http://repository.jboss.com/maven2/"
39         mavenRepo "http://repo.spring.io/milestone/"
40     }
```

```
    }

    dependencies {
        // specify dependencies here under either 'build', 'compile', 'runtime', 'test' or 'provided' scopes e.g.

        // runtime 'mysql:mysql-connector-java:5.1.22'
    }

    plugins {
        runtime ":hibernate:$grailsVersion"
        runtime ":jquery:1.8.3"
        runtime ":resources:1.2"
    compile ":spring-security-core:2.0-RC2"

        // Uncomment these (or add new ones) to enable additional resources capabilities
        //runtime ":zipped-resources:1.0"
        //runtime ":cached-resources:1.0"
        //runtime ":yui-minify-resources:0.1.5"

        build ":tomcat:$grailsVersion"

        runtime ":database-migration:1.3.2"

        compile ':cache:1.0.1'
    }
}
```

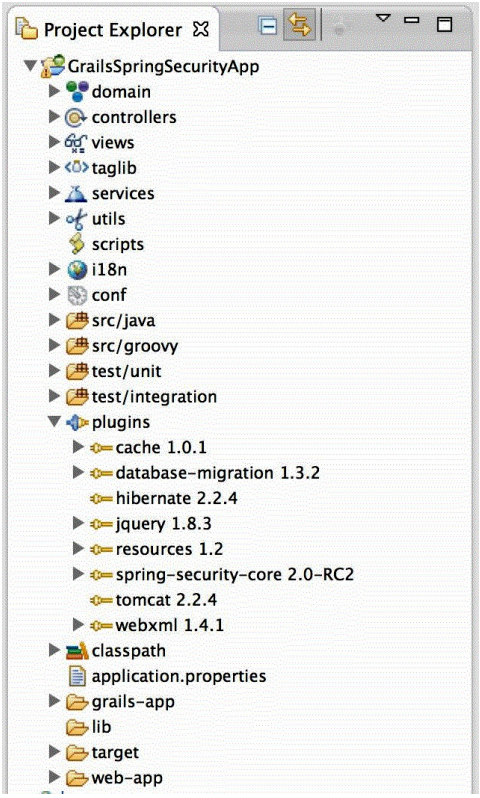
수정이 완료 되었으면, 저장을 한 후 우리가 생성한 GrailsSpringSecurityApp 프로젝트의 메뉴에서 Grails Tool을 선택하여 Refresh dependencies 를 실행하면 새로 변경된 플러그인이 적용이 됩니다.



Console에 필요한 파일 목록들이 다운로드 되는 것을 확인 할 수 있습니다.



실제 완료가 되었다면 아래처럼 plugins 디렉토리 밑에 spring-security-core:2.0-RC2 가 생성된 것을 확인 할 수 있습니다.



Console에 출력된 메시지를 살펴보면 아래와 같이 나옵니다.

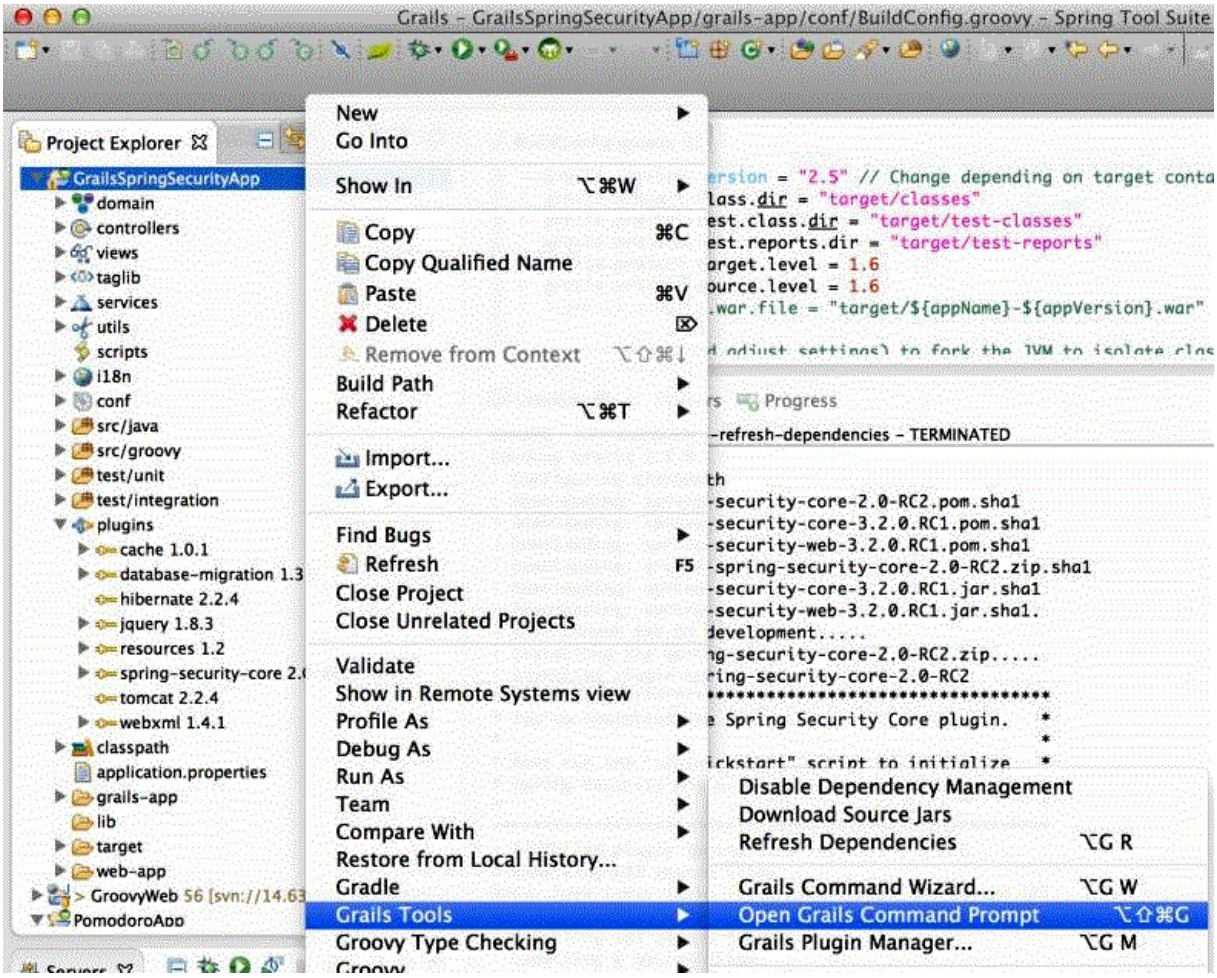
```
**********
* You've installed the Spring Security Core plugin. *
*
* Next run the "s2-quickstart" script to initialize *
* Spring Security and create your domain classes. *
*
**********
```

이제부터는 Spring Security Plugin 에서 제공하는 s2-quickstart 스크립트를 활용하여 GrailsSpringSecurityApp 프로젝트에 도메인 클래스를 생성하게 될 예정입니다.

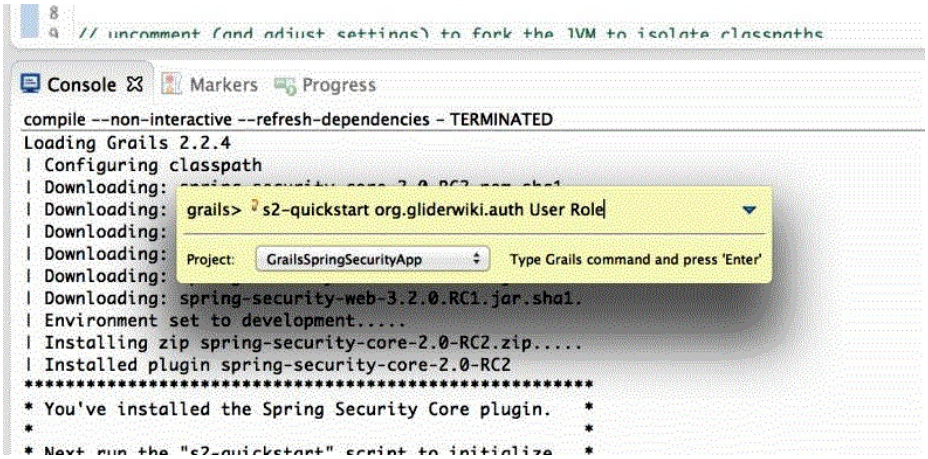
먼저 프로젝트 >> Grails Tools 에 Open Grails Command Prompt 를 선택하여 아래와 같이 입력합니다.

Command 명령어 입력

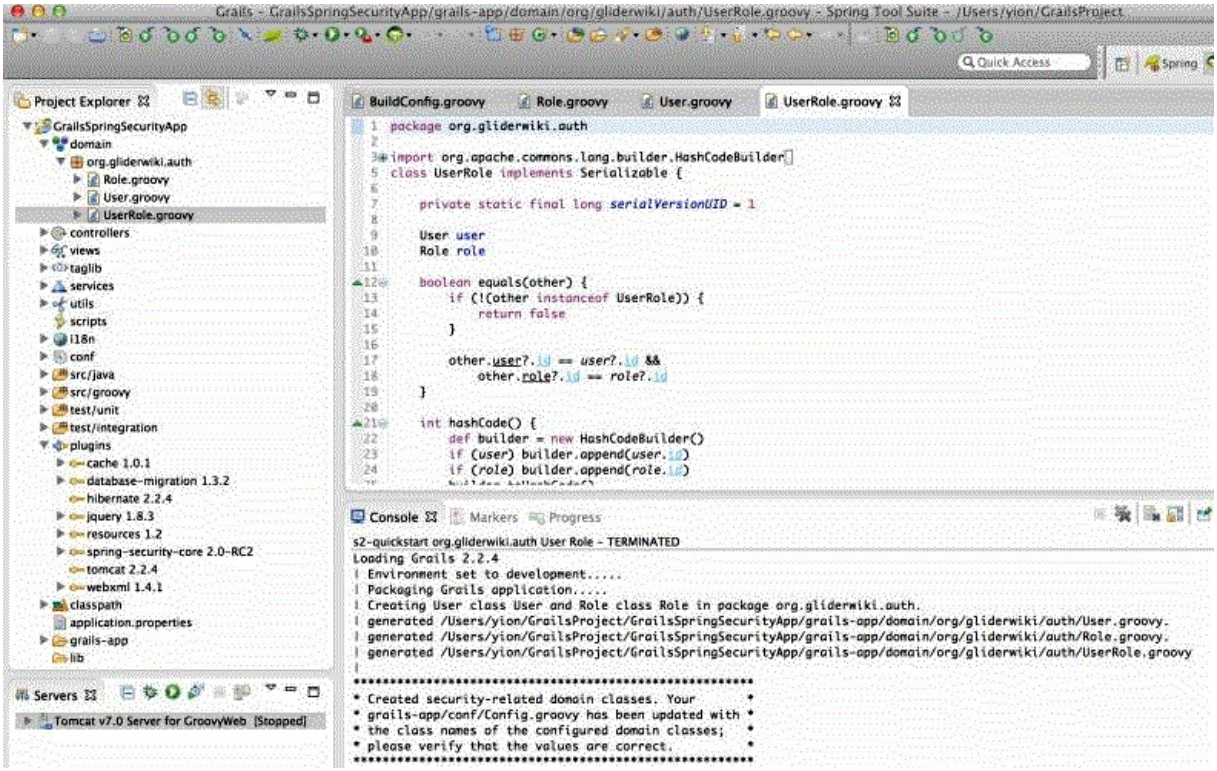
```
s2-quickstart org.gliderwiki.auth User Role
```



위에 언급 한데로 s2-quickstart 스크립트를 이용하여 패키지 경로, 생성해야 할 파일을 지정해주었습니다.



생성이 올바르게 되었다면 관련 파일들이 열리고 도메인 파일이 패키지 명 하위에 생성되게 됩니다.



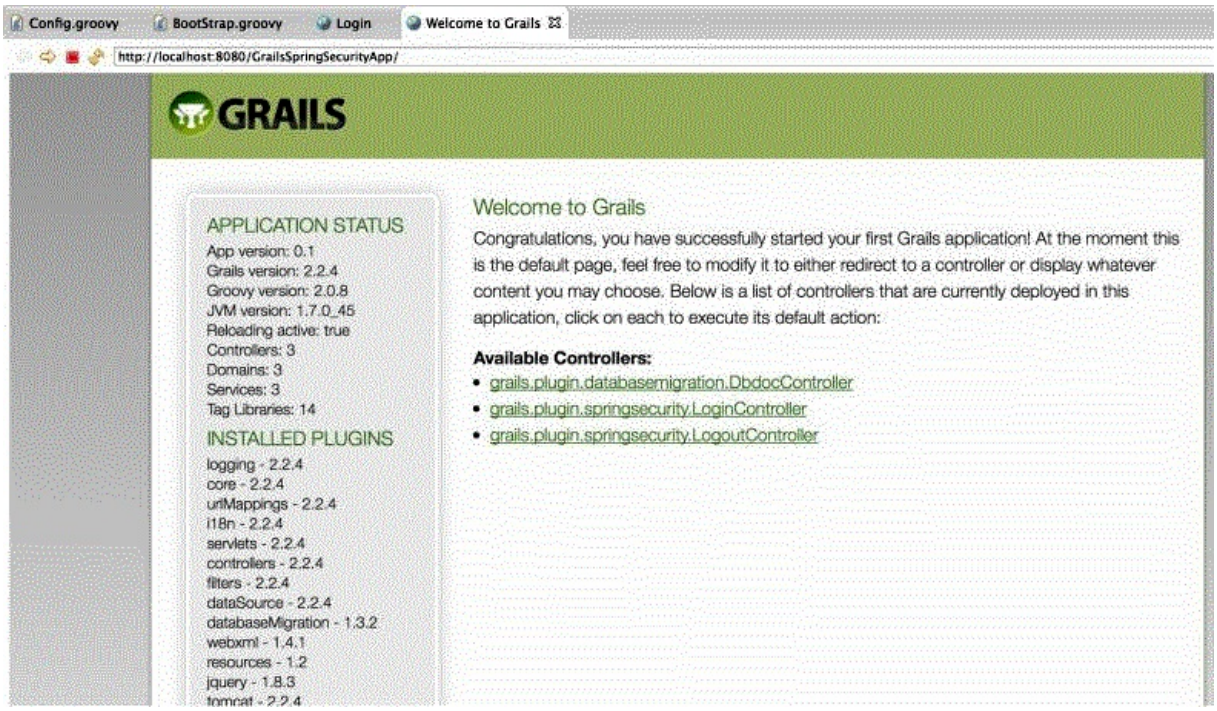
또한 기본 소스들도 완성된 형태로 제공됨을 확인 할 수 있습니다.

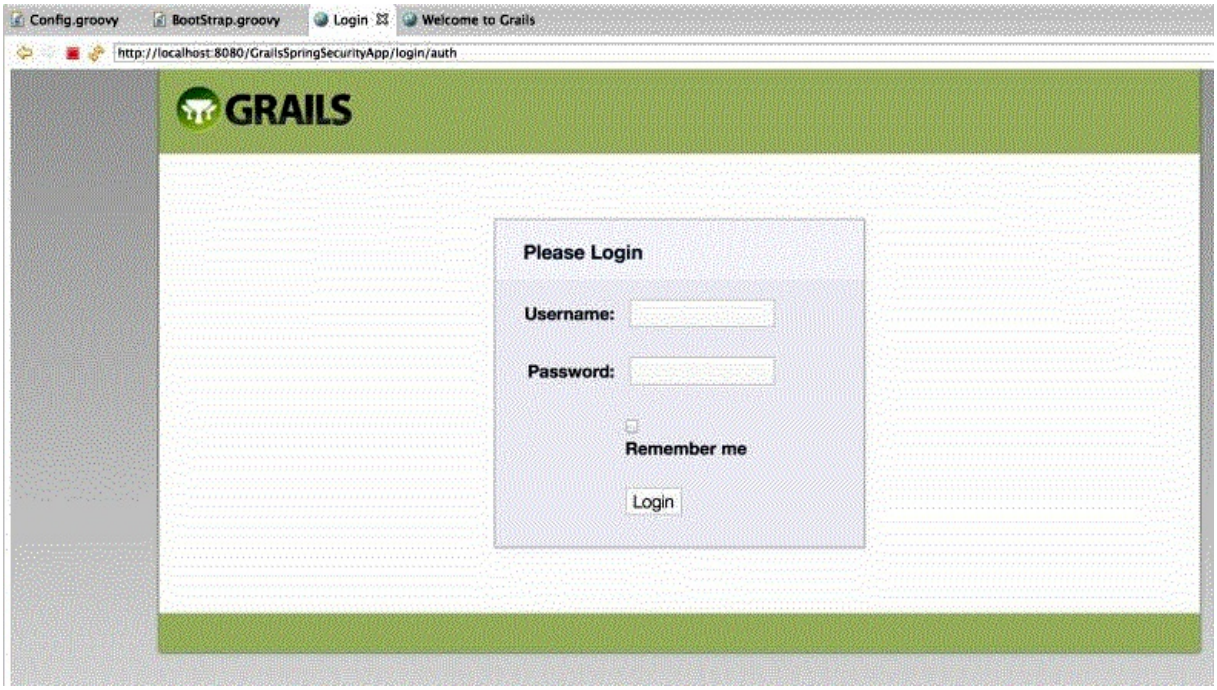
Spring Security 는 Authentication(인증)과 Authorization(권한부여) 를 통해 보안 처리를 쉽게 구현해주는 것으로 흔히 하는 방식인 크리덴셜(Credential) 기반의 비밀번호 / 아이디 를 받아 일치 여부를 판단하는 것 이외에 접근 권한 (Role) 을 통해 어플리케이션의 일부 (혹은 전체) 에 접근 가능한지를 검사, 승인 해주는 것들을 제공합니다.
웹 어플리케이션으로 본다면, 로그인부터 특정 페이지에 대한 접근 권한, 수정 권한, 뷰 권한등을 제어하고 저장하며 정보 요청자의 인증 후 권한 여부를 판별하여 특정한 정보를 제공할 수 있습니다.

당연히 비밀번호에 대한 암호화, 복호화나 DB접근, 메뉴접근 정보등을 일관성있게 어플리케이션 레벨에서 제공하고 있기 때문에 DB에 종속적인 암호화 알고리즘이나 기타 다른 서드파티 보안 모듈로 인해 확장에 저해되는 어플리케이션 설계로 부터 자유롭다고 볼 수 있습니다.

이제 생성된 어플리케이션에 접근해 보겠습니다.

GrailsSpringSecurityApp 을 실행한 후 접속해보면 아래와 같이 자동으로 생성된 화면을 확인 할 수 있습니다.





Spring Security Core 플러그인을 이용하여 s2-quickstart를 입력한 후 org.gliderwiki.auth 패키지 하위에 Role과 User를 생성하였습니다. 또한 자동으로 UserRole.groovy라는 파일이 생성되었습니다. 소스를 살펴보면

```groovy
package org.gliderwiki.auth

class Role {

	String authority

	static mapping = {
		cache true
	}

	static constraints = {
		authority blank: false, unique: true
	}
}
```

```groovy
package org.gliderwiki.auth

class User {

	transient springSecurityService

	String username
	String password
	boolean enabled = true
	boolean accountExpired
	boolean accountLocked
	boolean passwordExpired

	static transients = ['springSecurityService']

	static constraints = {
		username blank: false, unique: true
		password blank: false
	}

	static mapping = {
		password column: '`password`'
	}

	Set<Role> getAuthorities() {
		UserRole.findAllByUser(this).collect { it.role } as Set
	}

	def beforeInsert() {
		encodePassword()
	}

	def beforeUpdate() {
		if (isDirty('password')) {
			encodePassword()
		}
	}

	protected void encodePassword() {
		password = springSecurityService.encodePassword(password)
	}
}
```

```groovy
package org.gliderwiki.auth

import org.apache.commons.lang.builder.HashCodeBuilder

class UserRole implements Serializable {

    private static final long serialVersionUID = 1

    User user
    Role role

    boolean equals(other) {
        if (!(other instanceof UserRole)) {
            return false
        }

        other.user?.id == user?.id &&
            other.role?.id == role?.id
    }

    int hashCode() {
        def builder = new HashCodeBuilder()
        if (user) builder.append(user.id)
        if (role) builder.append(role.id)
        builder.toHashCode()
    }

    static UserRole get(long userId, long roleId) {
        UserRole.where {
            user == User.load(userId) &&
            role == Role.load(roleId)
        }.get()
    }

    static UserRole create(User user, Role role, boolean flush = false) {
        new UserRole(user: user, role: role).save(flush: flush, insert: true)
    }

    static boolean remove(User u, Role r, boolean flush = false) {

        int rowCount = UserRole.where {
            user == User.load(u.id) &&
            role == Role.load(r.id)
        }.deleteAll()

        rowCount > 0
    }

    static void removeAll(User u) {
        UserRole.where {
            user == User.load(u.id)
        }.deleteAll()
    }

    static void removeAll(Role r) {
        UserRole.where {
            role == Role.load(r.id)
        }.deleteAll()
    }

    static mapping = {
        id composite: ['role', 'user']
        version false
    }
}
```

이전 강좌 05. 5주차. Grails 웹 어플리케이션 생성과 개발 (MVC)에서
자동으로 설치된 내장 DB에 접근하는것에 권한이 없었는데, 아무나 어플리케이션의 DB 접속이 가능하면 안되므로, 이 부분에 접근 제어를 걸어보도록 하겠습니다.
dbconsole 같이 외부에서 접근이 불가능할 경우 로그인 화면 자체를 보여줄 필요가 없습니다.
일반적으로 많은 어플리케이션에서는 최소한 로그인 화면을 보여주고 비밀번호 입력이 맞는지 여부로 보안을 처리하는데, 이는 다소 위험한 문제를 야기 할 수 있으므로 화면 자체를 보여주지 않거나, 권한이 없으면 URL자체를 거부 하거나 하는 방식으로 인증과 권한 처리까지 해주는 것이 보안에 좀 더 적합하다고 할 수 있습니다.

conf/Config.groovy 파일에 아래와 같이 추가 합니다.

```groovy
// Added by the Spring Security Core plugin:
grails.plugin.springsecurity.userLookup.userDomainClassName = 'org.gliderwiki.auth.User'
grails.plugin.springsecurity.userLookup.authorityJoinClassName = 'org.gliderwiki.auth.UserRole'
grails.plugin.springsecurity.authority.className = 'org.gliderwiki.auth.Role'
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
    '/':                ['permitAll'],
    '/index':           ['permitAll'],
    '/index.gsp':       ['permitAll'],
    '/**/js/**':        ['permitAll'],
    '/**/css/**':       ['permitAll'],
    '/**/images/**':    ['permitAll'],
    '/**/favicon.ico':  ['permitAll'],
    '/dbconsole/**':    ['ROLE_ADMIN']
]
```

staticRules 에 dbconsole 로 들어오는 요청을 ROLE_ADMIN 로 지정하였습니다.
서버를 시작한 후 URL로 접근해보면(http://localhost:8080/GrailsSpringSecurityApp/dbconsole) 요청 자체를 튕겨버리는 것을 확인할 수 있습니다.
아에 페이지에 접근할 수 없게 어플리케이션의 로그인 화면(http://localhost:8080/GrailsSpringSecurityApp/login/auth)으로 튕겨버리는 것을 확인할 수 있습니다.

만약 ROLE_ADMIN 이 아니라 접근 허가 (permitAll) 권한을 주고 다시 접근해보면 dbconsole에 접근이 가능한 것을 확인할 수 있습니다.

그러나 애초 설계대로, 로그인 사용자중 접근 권한이 있는 사용자만 접근하도록 하는 것이 보안 원칙에는 맞을 것입니다.

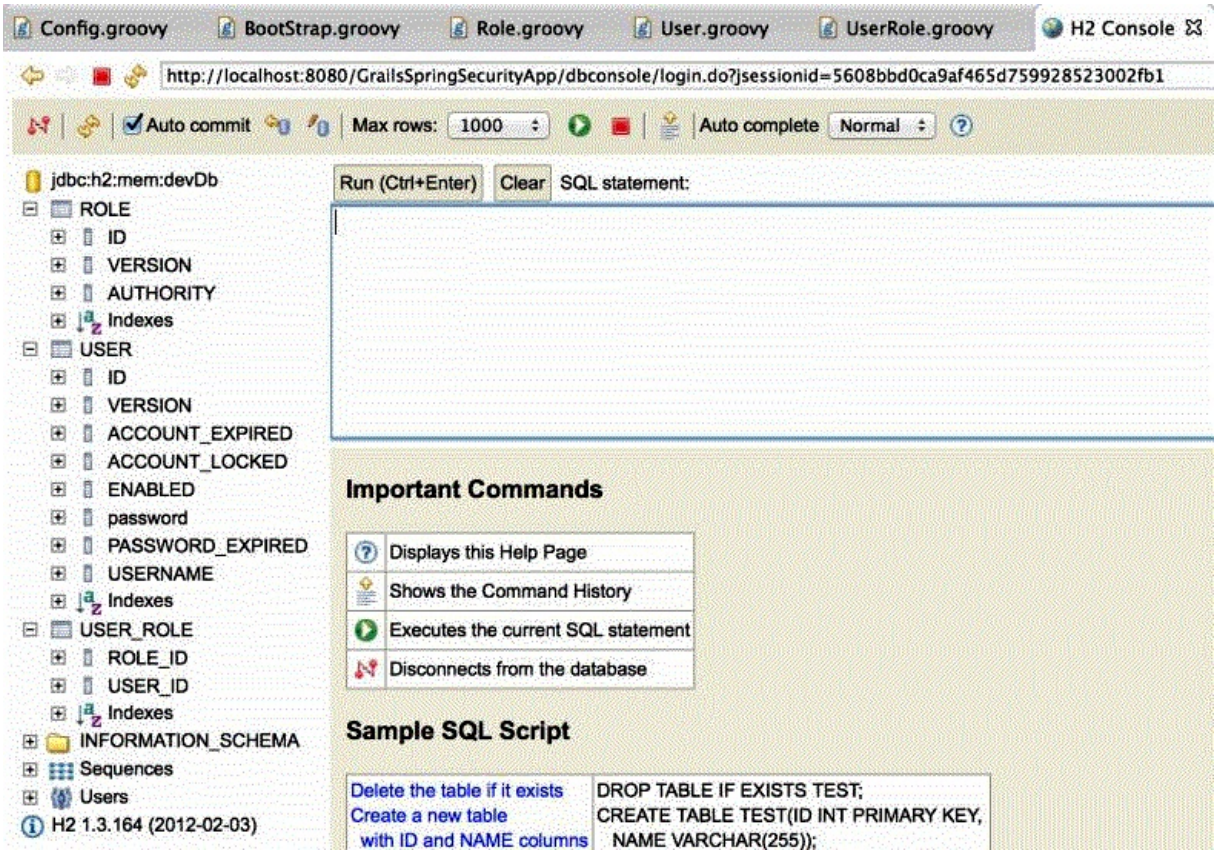
이제, 접근 가능한 권한을 가진(ROLE_ADMIN 권한) 사용자가 접근할 경우 해당 어플리케이션에서 이동이 되는지 한번 확인해보도록 하겠습니다.

conf/BootStrap.groovy 에 아래와 같이 코드를 추가합니다.

```
1  import org.gliderwiki.auth.*
2
3  class BootStrap {
4
5      def init = { servletContext ->
6          def adminRole = Role.findOrSaveWhere(authority:"ROLE_ADMIN")
7          def user = User.findOrSaveWhere(username:"cafeciel@hanmail.net", password:"password")
8
9          if(!user.authorities.contains(adminRole)) {
10             UserRole.create(user, adminRole, true)
11
12         }
13     }
14     def destroy = {
15     }
16 }
```

여기서는 cafeciel@hanmail.net 계정을 통해 ROLE_ADMIN 을 부여 하였습니다. 로그인이 되면 dbconsole에 접근하여 ROLE_ADMIN 이 부여되었는지 확인할 수 있습니다.
물론 Config.groovy 에 dbconsole 도 ROLE_ADMIN으로 되어있어야 합니다.

어플리케이션을 기동한 후 http://localhost:8080/GrailsSpringSecurityApp/dbconsole 접근해 봅니다.
로그인이 되지 않은 상태(ROLE_ADMIN) 이고 dbconsole 또한 permitAll이 아니라면 튕겨내는것을 확인할 수 있습니다.
이제 cafeciel@ 계정으로 로그인 후 다시 dbconsole에 접근하면 로그인 계정이 ROLE_ADMIN의 권한이 있으므로 제대로 콘솔 접근화면이 나오는 것을 확인할 수 있습니다.



dbconsole에서 각 테이블에 데이터가 어떻게 저장되어있는지 확인해보길 바랍니다.

링크 목록

- 05. 5주차. Grails 웹 어플리케이션 생성과 개발 (MVC) - http://www.gliderwiki.org/wiki/195

관련 키워드

Grails
