005.Grails와 Spring Framework (part2 Spring Security Core Plugin 적용)

페이지별 접근 제어

Table of Contents

비로그인 사용자와 로그인 사용자의 접근 권한에 따라 view의 접근제어를 하는 예제를 살펴보도록 합니다.

1. User 도메인 수정

추후 사용자 정보를 조회하기 위해 User 도메인에 아래와 같이 변수를 추가합니다.
String firstName
String lastName
String emailAddress

```
package org.gliderwiki.auth

class User {

    transient springSecurityService

    String username
    String password
    boolean enabled = true
    boolean accountExpired
    boolean accountLocked
    boolean passwordExpired
    String firstName
    String lastName
    String emailAddress


    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)
    }
}
```

2. BootStrap.groovy 수정
현재는 유저 한명의 정보만 담아 놓고 테스트 하기 때문에 User도메인에서 추가한 프러퍼티를 BootStrap에도 추가해주도록 합니다.

```
import org.gliderwiki.auth.*

class BootStrap {

    def init = { servletContext ->
        def adminRole = Role.findOrSaveWhere(authority:"ROLE_ADMIN")
        def user = User.findOrSaveWhere(username:"cafeciel@hanmail.net", password:"password", firstName:"first", lastName:"last", emai

        if(!user.authorities.contains(adminRole)) {
            UserRole.create(user, adminRole, true)
        }
    }
    def destroy = {
    }
```

```
}
```

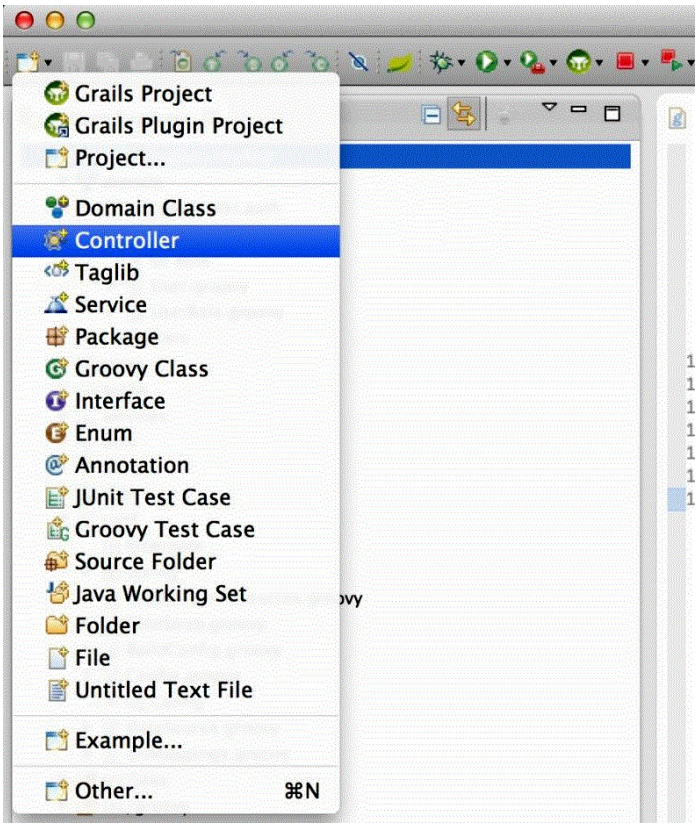
findOrSaveWhere 구문은 GORM 에서 지원하는 구문으로, Grails 2.0 버전부터 지원되기 시작했습니다.

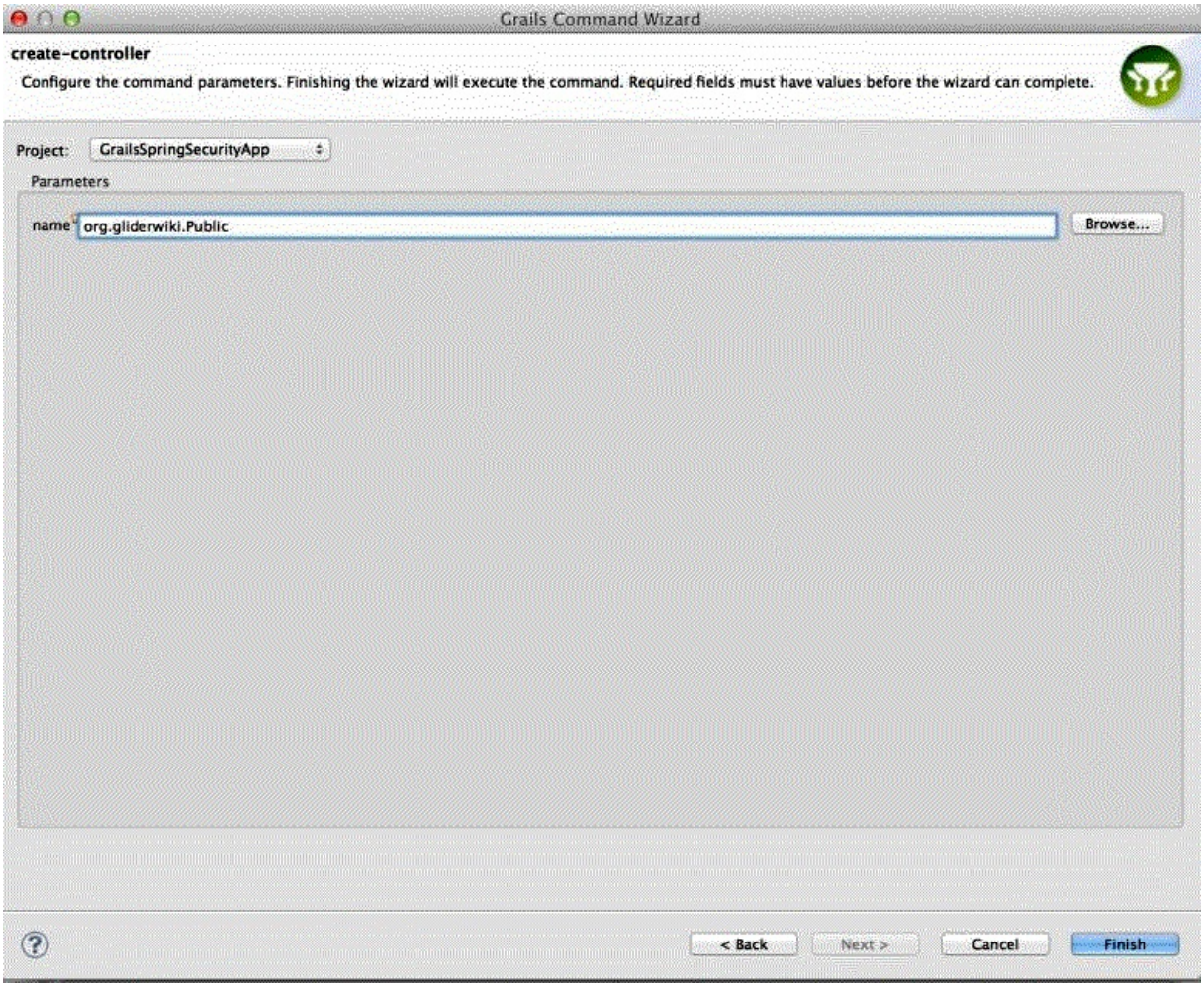
find결과가 있을 경우 결과를 리턴하고, 결과값이 없다면 save 구문을 실행하는 것으로 예를 들면 아래와 같은 구문으로 이해할 수 있습니다.

```
// 원래 구문
def adminRole = Role.findOrSaveWhere(authority:"ROLE_ADMIN")

// findOrSaveWhere 구문을 풀어서 쓴 코드
def adminRole = Role.findByAuthority('ROLE_ADMIN')
if(!adminRole) {
    adminRole = new AdminRole(authority:"ROLE_ADMIN")
    adminRole.save(failOnError:true)
}
```

3. Controller 추가

View 페이지별 접근 제어를 하기 위하 Controller 를 추가해 줍니다.
File > New > Controller 를 추가해준후 패키지 경로는 org.gliderwiki 로 지정하고 각각의 컨트롤러인 Public 과 Private 을 생성해주도록 합니다.





실제 PublicContoller의 역할은 비 로그인 사용자도 볼 수 있는 View를 연결하고, PrivateController의 경우 로그인 사용자별 권한에 따라 View의 제공여부를 결정하는 역할을 합니다.

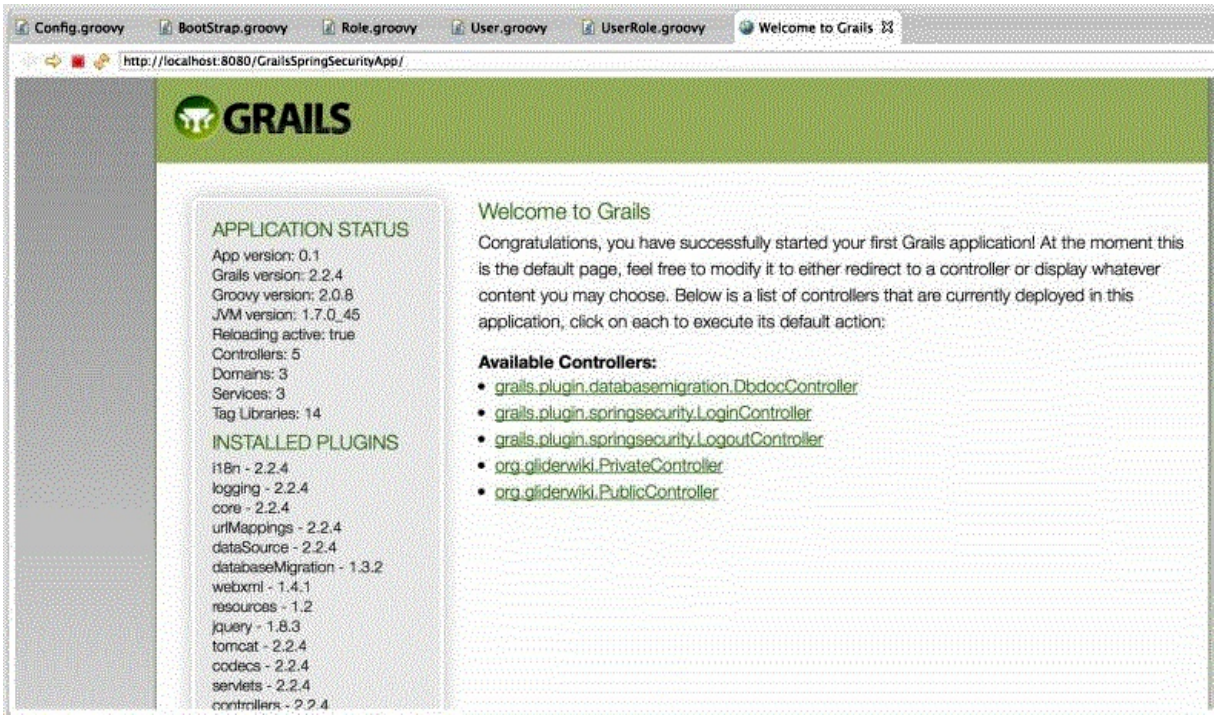
이제, 실제 Controller별로 접근 제어를 위한 어노테이션을 추가해주도록 합니다.

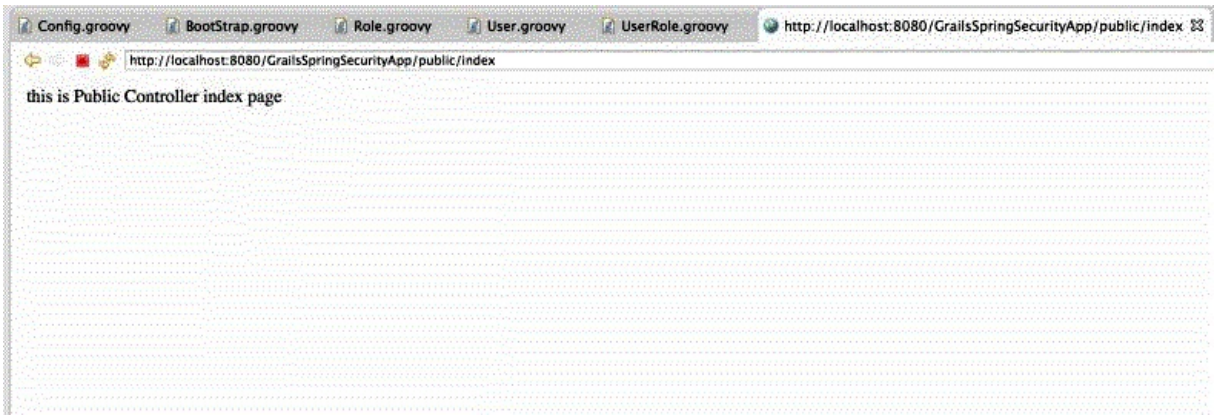
4. PublicController 에 @Secured 어노테이션 추가 (permitAll)

PublicController 에 아래와 같이 Secured 어노테이션을 추가해봅니다.

```
1  package org.gliderwiki
2
3  import grails.plugin.springsecurity.annotation.Secured
4
5  @Secured(['permitAll'])
6  class PublicController {
7
8      def index() {
9          render "this is Public Controller index page"
10     }
11 }
```

Secured 어노테이션을 이용하여 접근 제어의 레벨을 permitAll 로 지정해주었습니다.
서버를 띄워서 접근해보도록 하겠습니다.



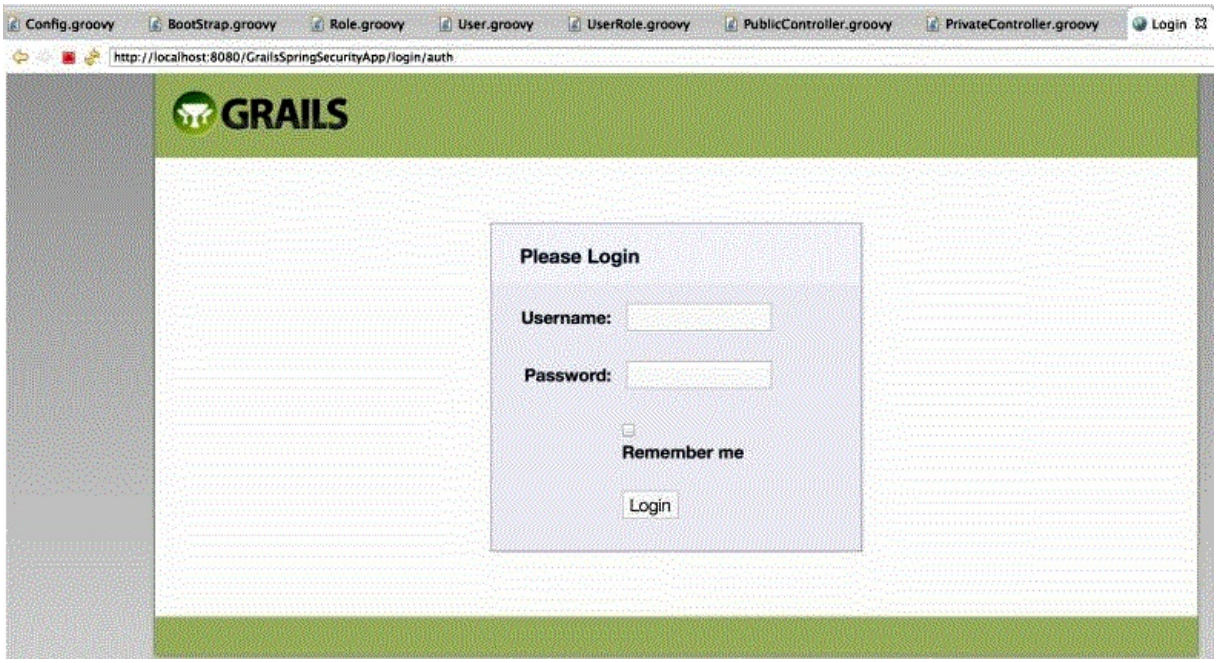


permitAll 에 해당하는 제어를 통해 로그인 유저가 아니라도 페이지에 접근함을 확인 할 수 있습니다.

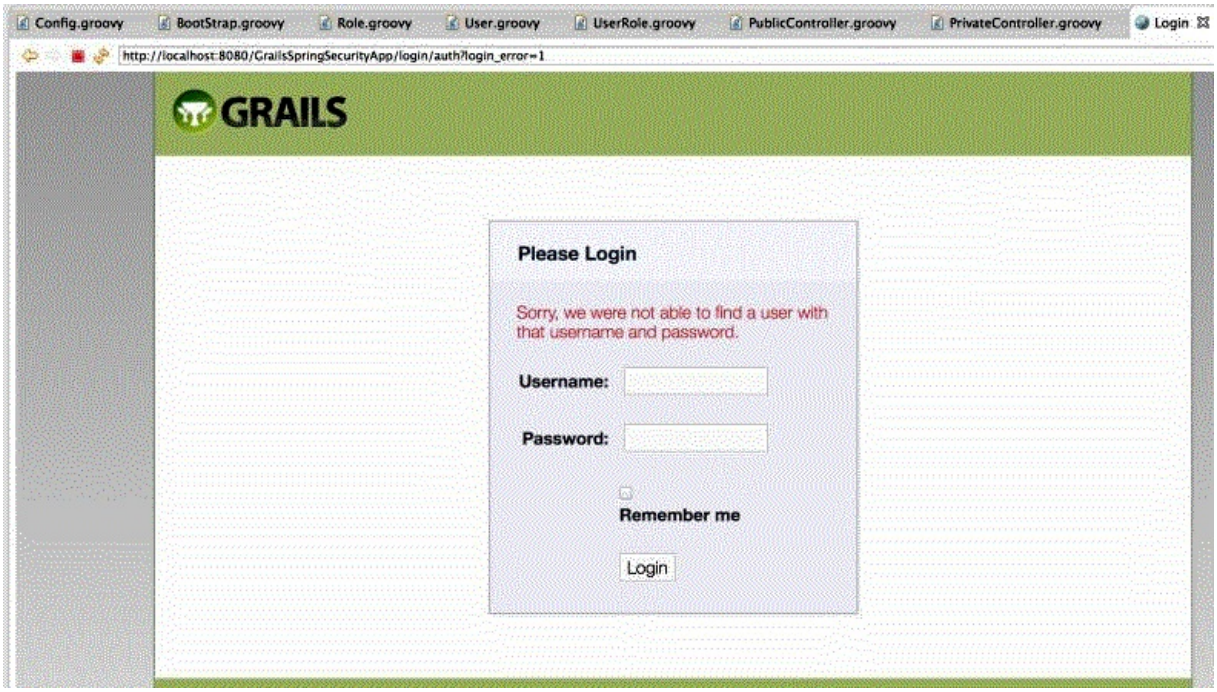
이제 Private에는 로그인 사용자 중 특정 권한을 가진 사용자가 접근할 수 있도록 코드를 추가해보도록 하겠습니다.

```
1  package org.gliderwiki
2  import grails.plugin.springsecurity.annotation.Secured
3
4  @Secured(['ROLE_ADMIN'])
5  class PrivateController {
6
7      def index() {
8          render "this is Private Controller index page"
9      }
10 }
```

서버에 접근해보면 PrivateController 에 권한은 ROLE_ADMIN이기 때문에 Login 인증 화면으로 이동함을 확인할 수 있습니다.
즉 Private 페이지 접근시에는 권한이 없으면 login 페이지, admin으로 로그인시 index를 호출함을 알수있습니다.



아이디와 비밀번호가 틀리면 아래와 같은 에러 메시지를 확인할 수 있습니다. 기존 강좌에서 message.properties를 수정한 에러 메시지 변경 처리를 해준바 있습니다.



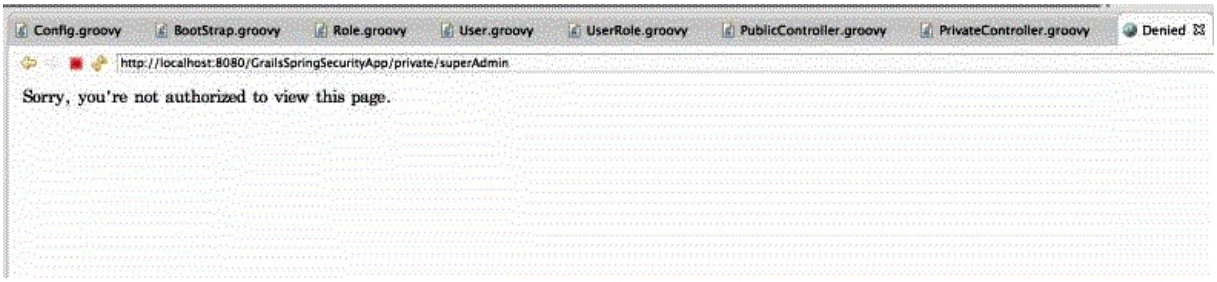
여기서는 BootStrap에 하드 코딩해준 로그인 아이디와 비밀번호(password 라고 지정) 를 입력하고 로그인을 하면 ROLE_ADMIN으로 접근이 허용된 Private page에 접근하는 것을 확인할 수 있습니다.

이제 권한을 좀 더 다른 레벨로 조정해주고 페이지별로 (Controller의 메서드별) 인증 레벨을 지정해주도록 하겠습니다.

6. PrivateController 에 superAdmin 권한 추가 - @Secured 어노테이션 추가 (ROLE_SUPERADMIN)
PrivateController 에 아래와 같이 ROLE_SUPERADMIN 레벨을 추가해서 테스트 해보도록 합니다.

```
package org.gliderwiki
import grails.plugin.springsecurity.annotation.Secured

class PrivateController {

    @Secured(['ROLE_ADMIN'])
    def index() {
        render "this is Private Controller index page"
    }

    @Secured(['ROLE_SUPERADMIN'])
    def superAdmin() {
        render "SUPER Admin Page!!!"
    }

}
```

이제 주소(http://localhost:8080/GrailsSpringSecurityApp/private/superAdmin) 로 접근해보면 현재 로그인 사용자의 권한이 없기 때문에 아래와 같은 메시지가 나오는 것을 확인할 수 있습니다.



마지막으로 Bootstrap에 ROLE_SUPERADMIN 권한이 있는 유저를 생성하여 로그인을 하면, 해당하는 권한의 페이지가 열리는지 확인해보도록 하겠습니다.

BootStrap.groovy의 소스는 아래와 같습니다.

```
import org.gliderwiki.auth.*

class BootStrap {

    def init = { servletContext ->
        def adminRole = Role.findOrSaveWhere(authority:"ROLE_ADMIN")
        def user = User.findOrSaveWhere(username:"cafeciel@hanmail.net", password:"password", firstName:"first", lastName:"last", emai

        def superRole = Role.findOrSaveWhere(authority:"ROLE_SUPERADMIN")
        def userAdmin = User.findOrSaveWhere(username:"admin@admin.com", password:"password", firstName:"super", lastName:"adn

        if(!user.authorities.contains(adminRole)) {
            UserRole.create(user, adminRole, true)
            UserRole.create(userAdmin, superRole, true)
        }
    }
    def destroy = {
    }
}
```

이제 위의 URL에 접근하면, SUPER User에 해당하는 아이디로 로그인 했을 때 페이지가 열리는 것을 확인할 수 있습니다.

관련 키워드

Grails

첨부파일목록


ZIP