

전자정부 표준프레임워크

# 종합실습 easycompany - solution

**eGovFrame**



## LAB 4-1 로그인 처리

Step 4-1-01. LoginController.java 에서 LoginService 를 이용하여 로그인 여부 체크 Account 객체를 리턴 받는다.

Step 4-1-02. LoginServiceImpl.java 에서 구현되어져 있는 LoginMapper의 authenticate 메소드를 이용하여 검증하여 결과(Account)객체를 리턴한다.

Step 4-1-03. LoginController.java 에서 가져온 account 객체를 session에 "UserAccount"라는 이름으로 저장한다.

Step 4-1-04. LoginController.java 에서 loginSuccess.do 로 호출된 처리를 위한 메소드를 만든다.(GET 방식만 지원)

# LAB 4-1-Solution

Step 4-1-01. LoginController.java 에서 LoginService 를 이용하여 로그인여부 체크 Account 객체를 리턴 받는다.

```
Account account = (Account) loginService.authenticate(id, password);
```

Step 4-1-02. LoginServiceImpl.java 에서 구현되어져 있는 LoginMapper의 authenticate 메소드를 이용하여 검증하여 결과(Account)객체를 리턴한다.

```
return loginMapper.authenticate(id, password);
```

Step 4-1-03. LoginController.java 에서 가져온 account 객체를 session에 "UserAccount"라는 이름으로 저장한다.

```
if (account != null) {  
    request.getSession().setAttribute("UserAccount", account);  
    return "redirect:/loginSuccess.do";  
} else {  
    return "login";  
}
```

Step 4-1-04. LoginController.java 에서 loginSuccess.do 로 호출된 처리를 위한 메소드를 만든다.(GET 방식만 지원)

```
@RequestMapping(value = "/loginSuccess.do", method = RequestMethod.GET)  
public void loginSuccess() {  
}
```

## LAB 4-2 사원목록 조회, ajax(autoComplete)

Step 4-2-01. `EmployeeController.java` 에서 `employeeList.do` 경로로 요청과 메소드를 매핑해준다. `@RequestMapping` 를 이용한다.

Step 4-2-02. `EmployeeController.java` 에서 화면에서 넘어오는 `pageNo` 파라미터 값을 반드시 넘어오지 않아도 됨을 명시적 표현해본다. `@RequestParam` 의 `required` 를 이용한다.

Step 4-2-03. `employeeList.jsp`에서 ajax사용을 위한 `jquery autocomplete` 코드를 생성한다.

# LAB 4-2-Solution

Step 4-2-01. EmployeeController.java 에서 employeeList.do 경로로 요청과 메소드를 매핑해준다. @RequestMapping 을 이용한다.

```
@RequestMapping(value = "/employeeList.do")
public String getEmpList(...
```

Step 4-2-02. EmployeeController.java 에서 화면에서 넘어오는 pageNo 파라미터 값을 반드시 넘어오지 않아도 됨을 명시적 표현해본다. @RequestParam 의 required 를 이용한다.

```
public String getEmpList(@RequestParam(value = "pageNo", required = false) String pageNo,
                        SearchCriteria searchCriteria, ModelMap model, @RequestParam Map<String, Object> commandMap) throws
Exception {
```

Step 4-2-03. employeeList.jsp에서 ajax사용을 위한 jquery autocomplete 코드를 생성한다.

```
$("#searchName").autocomplete({
    source: function(request, response){
        $.ajax({
            url: "<c:url value='/suggestName.do' />",
            contentType: "application/x-www-form-urlencoded; charset=UTF-8",
            data: {searchName : encodeURIComponent(request.term)},
            dataType: 'json',
            success: function(returnData, status){
                response(returnData.nameList);
            }
        });
    },
    minLength: 1,
    select: function(event, ui){
        $("#searchName").val(this.value);
    }
});
```

## LAB 4-3 사원정보 등록화면 이동

Step 4-3-01. `EmployeeController.java` 에서 요청되는 `insertEmployee.do` 와 메소드를 매핑 한다. 단 **GET** 방식에 대해서만 처리하도록 한다.

Step 4-3-02. `EmployeeController.java` 에서 `employee` 객체를 `ModelAttributes` 를 이용하여 세팅하여보자.

Step 4-3-03. `EmployeeController.java` 에서 `deptInfoOneDepthCategory` 객체를 `ModelAttributes` 를 이용하여 세팅하여보자.

# LAB 4-3-Solution

Step 4-3-01. EmployeeController.java 에서 요청되는 insertEmployee.do 와 메소드를 매핑 한다. 단 GET 방식에 대해서만 처리하도록 한다.

```
@RequestMapping(value = "/insertEmployee.do", method = RequestMethod.GET)
public String setupForm(Model model) {
```

Step 4-3-02. EmployeeController.java 에서 employee 객체를 ModelAttributes 를 이용하여 세팅하여보자.

```
@ModelAttribute("employee")
public Employee defaultEmployee() {
    return new Employee();
}
```

Step 4-3-03. EmployeeController.java 에서 deptInfoOneDepthCategory 객체를 ModelAttributes 를 이용하여 세팅하여보자.

```
@ModelAttribute("deptInfoOneDepthCategory")
private Map<String, String> referenceDataOneDepthDept() {
    return departmentService.getDepartmentIdNameList("1");
}
```

## LAB 4-4 사원정보 등록화면, 부서 멀티콤보 처리

Step 4-4-01. `EmployeeController.java` 에서 요청되는 `insertEmployee.do` 와 메소드를 매핑 한다. 단, **POST** 방식에 대해서만 처리하도록 한다.  
리턴타입은 **String** 이다.

Step 4-4-02. `EmployeeController.java` 에서 `DefaultBeanValidator` 를 이용하여 `employee` 객체 값을 체크한다.

Step 4-4-03. `EmployeeController.java` 에서 `DefaultBeanValidator` 를 실행한 결과 에러가 있을 경우 `addemployee.jsp` 원래 페이지를 다시 보여주고  
에러 메시지를 뿌려준다.

Step 4-4-04. `addemployee.jsp` 에서 `jquery` 이용하여 `ajax` 방식으로 통신하여 근무부서를 조회한다.



# LAB 4-4-Solution

Step 4-4-01. EmployeeController.java 에서 요청되는 insertEmployee.do 와 메소드를 매핑한다. 단, POST 방식에 대해서만 처리하도록 한다.

리턴타입은 String 이다.

```
@RequestMapping(value = "/insertEmployee.do", method = RequestMethod.POST)
public String insertEmployee(
```

Step 4-4-02. EmployeeController.java 에서 DefaultBeanValidator 를 이용하여 employee 객체 값을 체크한다.

```
beanValidator.validate(employee, bindingResult);
```

Step 4-4-03. EmployeeController.java 에서 DefaultBeanValidator 를 실행한 결과 에러가 있을 경우 addemployee.jsp 원래 페이지를 다시 보여주고

에러 메시지를 뿌려준다.

```
if (bindingResult.hasErrors()) {
    return "addemployee";
}
```

Step 4-4-04. addemployee.jsp 에서 jquery 이용하여 ajax 방식으로 통신하여 근무부서를 조회한다.

```
$('#superdeptid').change(function() {
    $.ajax({
        url: "<c:url value='/autoSelectDept.do' />",
        contentType: "application/x-www-form-urlencoded; charset=UTF-8",
        data: {depth:2, superdeptid:encodeURIComponent($('#superdeptid option:selected').val())},
        dataType: 'json',
        success: function(returnData, status){
            $('#departmentid').loadSelectDept(returnData,"근무부서를 선택하세요.");
        }
    });
});
```

## LAB 4-5 사원정보 등록, 상세조회

Step 4-5-01. EmployeeServiceImpl.java 에서 insertEmployee 메소드는 Employee 를 파라미터로 받아 empMapper 를 실행한다. (결과는 저장된 개수)

Step 4-5-02. EmpMapper.java 에서 insertEmployee 메소드는 EgovAbstractMapper에서 제공되는 insert()를 이용하여 com.easycompany.mapper.EmpMapper.insertEmployee ID 를 가진 sql문을 실행한다.

Step 4-5-03. Employee.xml 파일에 insertEmployee 에 해당하는 sql문을 만들어준다. (DBIO 를 이용하여 생성하도록 한다. )

Target 폴더 안의 Employee.xml 파일은 수정 할 필요가 없다.

Insert 문의 id : insertEmployee 이고 parameterClass 는 com.easycompany.service.Employee

이고 SQL 문은

insert into employee ( employeeid, name, age, departmentid, email, password )

values ( #{employeeid}, #{name}, #{age}, #{departmentid}, #{email}, #{password} )

# LAB 4-5-Solution

Step 4-5-01. EmployeeServiceImpl.java 에서 insertEmployee 메소드는 Employee 를 파라미터로 받아 empMapper 를 실행한다. (결과는 저장된 개수)

```
return empMapper.insertEmployee(emp);
```

Step 4-5-02. EmpMapper.java 에서 insertEmployee 메소드는 EgovAbstractMapper에서 제공되는 insert()를 이용하여 com.easycompany.mapper.EmpMapper.insertEmployee ID 를 가진 sql문을 실행한다.

```
insert("com.easycompany.mapper.EmpMapper.insertEmployee", emp);  
result = 1;
```

Step 4-5-03. Employee.xml 파일에 insertEmployee 에 해당하는 sql문을 만들어준다. (DBIO 를 이용하여 생성하도록 한다.)

Target 폴더 안의 Employee.xml 파일은 수정 할 필요가 없다.

Insert 문의 id : insertEmployee 이고 parameterClass 는 com.easycompany.service.Employee

이고 SQL 문은

insert into employee ( employeeid, name, age, departmentid, email, password )

values ( #{employeeid}, #{name}, #{age}, #{departmentid}, #{email}, #{password} )

```
<insert id="insertEmployee" parameterType="com.easycompany.service.Employee">  
  <![CDATA[  
    insert into employee ( employeeid, name, age, departmentid, email, password )  
    values ( #{employeeid}, #{name}, #{age}, #{departmentid}, #{email}, #{password} )  
  ]]>  
</insert>
```

## LAB 4-6 사원정보 수정

Step 4-6-01. `EmployeeController.java` 에서 `updateEmployee.do` 요청 시 `@RequestMapping` 에 `value` 와 `method (GET)`를 만들어준다.

Step 4-6-02. `EmployeeController.java` 에서 `getEmployeeInfo` 메소드에 `updateEmployee.do` 요청 시 `Employee` 객체를 가져오도록 `ModelAttribute` 를 설정한다. 파라미터로 “`employeeid`” key 가 넘어온다. 이것으로 `EmployeeService` 의 `getEmployeeInfoById` 메소드를 호출하여 `Employee`를 구해 리턴한다.  
(참고: `referenceDataOneDepthDept()` )

# LAB 4-6-Solution

Step 4-6-01. EmployeeController.java 에서 updateEmployee.do 요청 시 @RequestMapping 에 value 와 method (GET)를 만들어준다.

```
@RequestMapping(value = "/updateEmployee.do", method = RequestMethod.GET)
public String defaultUpdateEmployee(@RequestParam("employeeid") String employeeid, ModelMap model) {
```

Step 4-6-02. EmployeeController.java 에서 getEmployeeInfo 메소드에 updateEmployee.do 요청 시 Employee 객체를 가져오도록 ModelAttribute 를 설정한다. 파라미터로 “employeeid” key 가 넘어온다. 이것으로 EmployeeService 의 getEmployeeInfoById 메소드를 호출하여 Employee를 구해 리턴한다.

(참고: referenceDataOneDepthDept() )

```
@ModelAttribute("employee")
public Employee getEmployeeInfo(String employeeid) {
    return employeeService.getEmployeeInfoById(employeeid);
}
```

수고 하셨습니다. ^^\*

