REST API 개발

EHR API를 제공하기 위한 방법을 설명합니다.

연계 시스템이나 외부에서 EHR 시스템에 데이터를 요청할 경우 해당 요청에 대한 응답 데이터를 제공할 때 사용합니다.

배포된 API는 내부 결함등의 수정은 가능하나 request/response에 대한 내용이 변경된 경우 버전을 변경하여 새롭게 배포해야 합니다.

REST API 방식으로 제공하며, request와 respone의 포맷은 applincaiton/json을 사용합니다.

GET 메소드인 경우 request 포맷은 설정하지 않습니다. URL패턴과 파라미터로만 구성합니다.

REST API는 정형화된 표준이 없어서 API 설계가 어려울 수 있습니다. 그래서 REST API에 대한 개념을 알고 설계를 해야 합니다.

대략적인 REST API 대한 내용을 간략하게 설명해 놓은 웹 페이지입니다. 참조하세요.

http://meetup.toast.com/posts/92

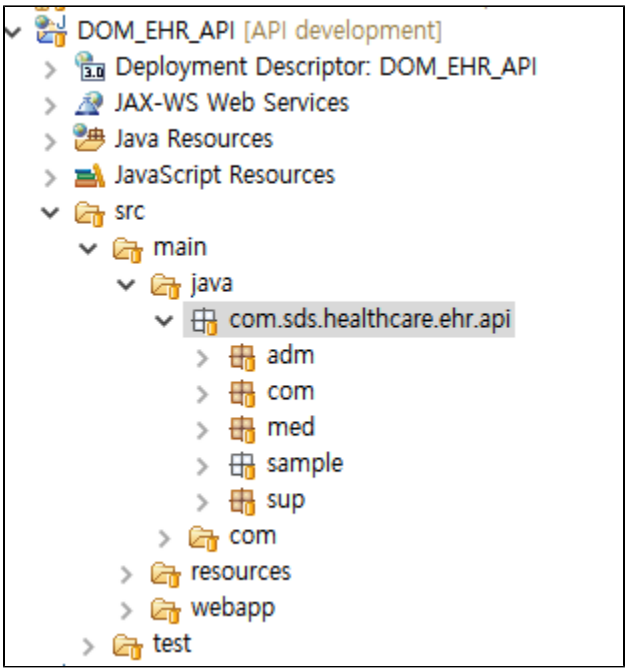
1. 개발 방법

API서비스를 위해 별도의 애플리케이션으로 분리되어 개발/운영 예정입니다.

API 개발은 http://xxx.xxx.xxx.xxx/nexmed/svr_ehr_api.git 형상에서 진행합니다.

0) 프로젝트 구성

하위 패키지는 온라인 서비스의 패키지 구조와 동일하고 RestController를 위해 ctl 패키지 구조를 추가합니다.



1) RestContorller 및 Svc 클래스 추가

최하위 패키지의 ctl패키지를 추가합니다. (svc, biz, dao, vo 와 같은 레벨)

클래스를 추가하고 어노테이션을 아래와 같이 설정합니다.

클래스 명 : 기존의 클래스 명과 동일한 규칙으로 작성하며 postfix를 API로 합니다. ex) CcmBoardAPI

RequestMapping : 요청시 url path를 설정하며 /api 로 설정합니다.

추가적인 url은 메소드에 value 어노테이션에 추가합니다.

tags : 최하위 패키지명(3자리)-업무약어명 (모두 소문자로 작성) ex) ccm-xxxx, mnr-xxxx

description : 설명 (한글로 작성 가능)

```
@RestController
@Api(tags = {"exm-board"}, description=" ///")
@RequestMapping("/api")
public class ExmBoardAPI {
...
}
```

- API를 제공하기 위한 로직을 작성합니다. 기존 로직개발과 동일하나 클래스 명에 API를 추가합니다.

항목	클래스 명
서비스 인터페이스	xxxAPISvc.java
서비스 클래스	xxxAPISvcImpl.java
SVO, BVO, DVO 클래스	xxxAPISVO.java
OutSVO 클래스 (리턴VO)	xxxOutSVO.java
OutDVO 클래스 (OutSVO에서 사용하는 DVO클래스)	xxxOutDVO.java
InSVO 클래스 (입력VO)	xxxInSVO.java
InDVO 클래스	xxxInDVO.java

- 비즈니스 로직은 업무에 맞는 온라인 업무 공통 프로젝트(ADMCO, COM, MEDCO,...)에 생성하며, API 서비스 클래스에서 해당 Biz클래스를 호출합니다.

** 기존의 작성된 Biz클래스를 재활용하기 위해 AP, AC, SL, SD 프로젝트는 예외적으로 빌드시 배포됩니다. 신규 작성되는 Biz클래스는 반드시 업무 공통 프로젝트에 작성합니다.

- Svc 인터페이스에서 Online 서비스와 다르게 @ServiceIdMapping 은 설정하지 않습니다.

- 리턴VO는 입력과 출력값이 포함되어 있으므로 출력값만 존재하는 별도의 OutSVO를 생성합니다. OutSVO는 EHRCommonVO를 상속받지 않습니다.

내부적으로 DVO를 사용한 경우도 마찬가지로 출력값에 해당하는 별도의 OutDVO를 생성해야 합니다. OutDVO는 EHRCommonVO를 상속받지 않습니다.

- OutSVO와 OutDVO의 경우 VO매니저를 사용하여 저장한 후에는 EHRCommonVO 상속 구문(extends EHRCommonVO)을 삭제해주시기 바랍니다.

- 필드에 대한 설명은 필드명에 APIModelProperty 어노테이션을 이용해서 작성합니다.

설정 방법 : 항목명 타입(길이) ex) 순번 BigDecimal(10)

필요시 항목명에 설명을 추가적으로 입력 가능합니다.

```
@ApiModelProperty(value=" BigDecimal(10)")
private BigDecimal seq;
```

- RestController 에서는 Svc를 Injection받아서 사용합니다. 모든 RestController는 Svc를 호출하도록 합니다. Controller에서 Biz, DAO 바로 호출할 수 없습니다.

```
...
public class ExmBoardAPI {
        @Resource
        ExamBoardAPISvc examBoardAPISvc;
    ...
}
```

2) API를 제공하기 위한 메소드 추가

@ApiOperation 의 value에 API기능명, notes에 설명을 입력합니다.

@RequestMapping에는 method 값에 메소드 타입(GET, POST, PUT, DELETE) 과 value 값에 url-path등 을 설정합니다.

value에는 /v1(버전)/최하위 패키지명(3자리)/리소스명/xxx(옵션) 구조로 설정합니다. 리소스명을 설정할 때는 단어로만 구성하고 길어지는 경우는 단계별로 URL을 구성합니다.

url에 해당하는 path는 무조건 소문자로 입력하고 문자가 긴 경우는 - 을 이용한다. PathVariable의 변수값은 CamelCase로 작성합니다. RequestParam 도 CamelCase로 작성합니다.

버전을 올릴 경우 추가적인 메소드를 구현하고 메소드명 뒤에 버전 숫자를 입력하여 추가합니다.

ex) 버전이 v1인 getCustomer() 메소드에서 v2로 버전을 올린 경우 getCustomer2 메소드를 추가로 생성하고 버전을 v2로 설정합니다.

produces에는 Response 데이터 포멧(MediaType.APPLICATION_JSON_VALUE) 을 설정합니다 .

* 주의

ApiParam과 PathVariable 혹은 ApiParam과 RequestParam 어노테이션의 required속성은 동일하게 설정합니다.

ApiParam은 문서화시 사용되는 정보이며, RequestParam, PathVariable 은 실제 동작시 사용되는 속성입니다.

```
@ApiOperation(value = " ", notes = "     .")
@RequestMapping(method = RequestMethod.GET
, value = "/v1/exm/boards/{blbrNo}"
, produces = MediaType.APPLICATION_JSON_VALUE)
)
public ResponseEntity<ExamBoardOutSVO> getBoard(
  @ApiParam(value = " ", required=true)
  @PathVariable(required=true) Integer blbrNo)
{
 ExamBoardAPISVO svo = new ExamBoardAPISVO();
 svo.setBlbrNo(new BigDecimal(blbrNo));
 svo = examBoardAPISvc.getBoardAPI(svo);
 ExamBoardOutSVO outSVO = new ExamBoardOutSVO();
 VoUtil.copy(svo, outSVO);
 return new ResponseEntity<ExamBoardOutSVO>(outSVO, HttpStatus.OK);
}
```

GET 메소드로 개발할 때 속성에 해당하는 값들은 @RequestParam을 이용해서 값을 입력 받습니다.

ex) /api/v1/smp/boards/1?replyseq=2

어노테이션에 입력한 내용은 문서화 때 활용되므로 최대한 자세히 입력합니다.

3) API 에러처리

의도되지 않는 에러가 발생하는 경우에는 다음과 같은 포맷으로 응답 메시지가 전달됩니다.

에러코드는 Default : E01, BizException : E02, APIException : E03(default) 입니다.

APIException 을 활용하는 경우는 코드를 API 별로 정의해서 활용할 수 있습니다.

```
{
        "fault": {
                "code" : "",
            "message" : ""
                "description" : ""
        }
}
```

- Default 에러 (HttpStatus : 500)

의도하지 않은 에러로 발생한 경우이며 Exception의 메시지가 전달됩니다. (ex) NullPointerExeption)

```
{
        "fault": {
                "code" : "E01",
            "message" : "Exception.getMessage() "
                "description" : ""
        }
}
```

- BizException 에러 (HttpStatus : 500)

온라인 프로그램에서 사용하는 EHRBizException을 사용하여 throw를 한 경우 메시지 코드에 해당하는 메시지가 전달됩니다.

```
{
        "fault": {
                "code" : "E02",
            "message" : "   "
                "description" : ""
        }
}
```

- APIException 에러 (HttpStatus : 500)

EHRAPIException을 사용하여 throw를 한 경우 해당 코드와 메시지가 전달됩니다.

ex) throw new EHRAPIException("에러메시지"); //default 에러 코드 E03 으로 설정됨.

throw new EHRAPIException("E99", "에러메시지");

```
{
        "fault": {
                "code" : " (default E03)",
            "message" : " "
                "description" : ""
        }
}
```

- 기본 설정 HttpStatus

아래의 에러코드 내용은 기본적으로 명세에 포함되므로 별도로 설정하지 않아도 됩니다.

HttpStatus	에러코드(error)	에러내용(message)
400	E10	Bad Request
401	E11	Unauthorized
403	E12	Forbidden
404	E13	Forbidden
405	E14	Method Not Allowed
500	E01	Internal Server Error

500	E02	BizException 예외
500	E03	APIException 예외

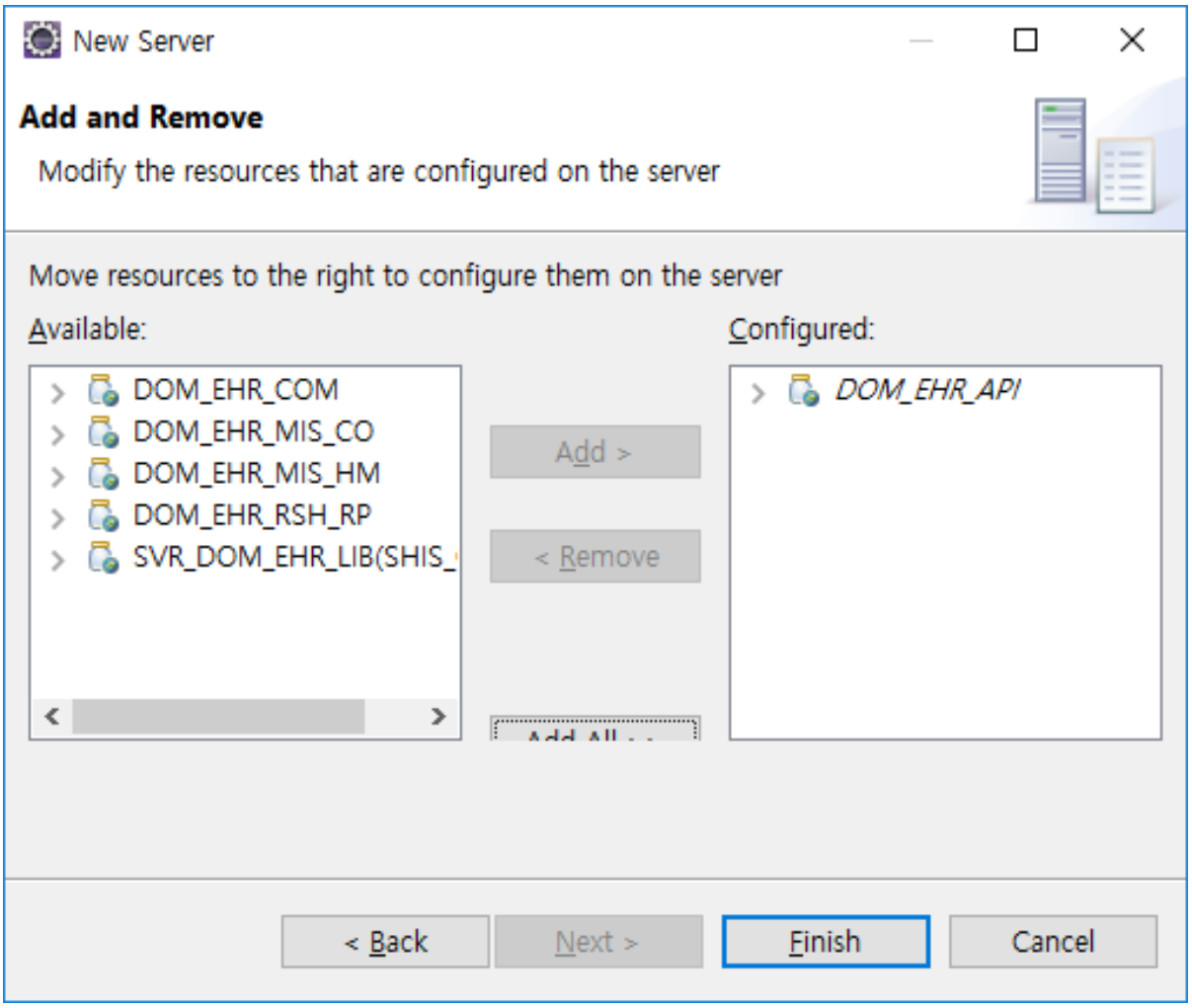
ex) 405 에러 발생시 응답 메시지

4) API 테스트 (Swagger 이용)

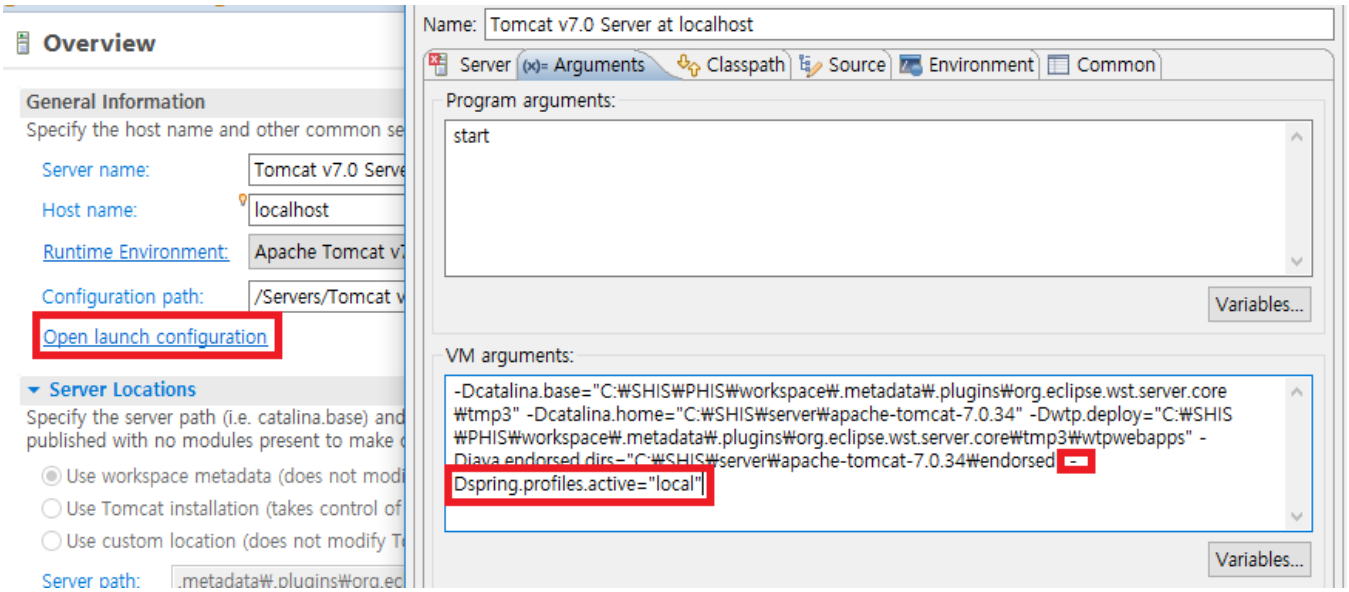
APIM에 등록하기 전에 간단히 로컬에서 테스트할 수 있는 방법입니다.

최종 테스트는 APIM을 이용해서 테스트를 해야 합니다.

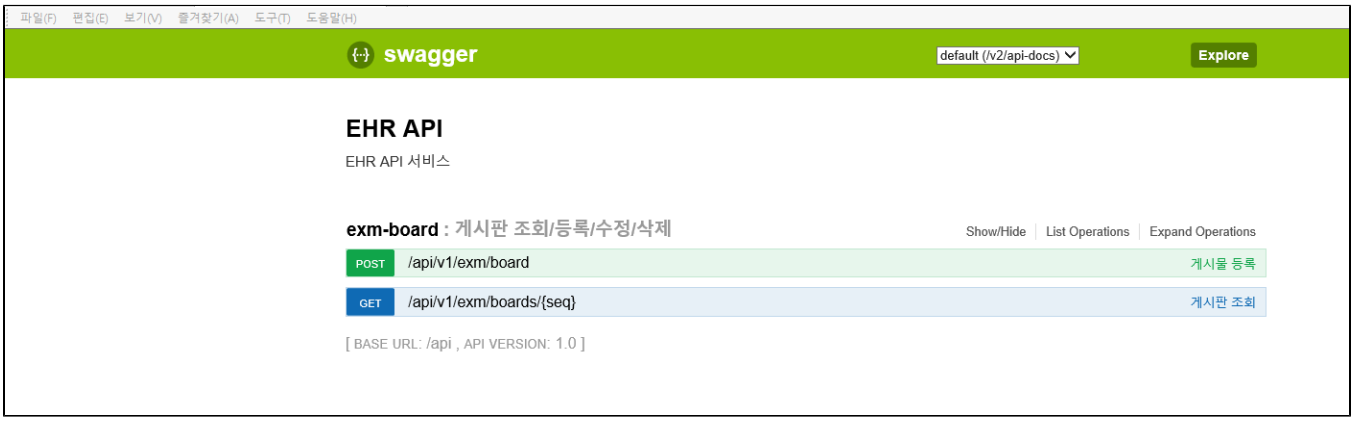
로컬 테스트를 위해서 Tomcat을 추가합니다.



각 업무 서버와 마찬가지로 Open launch configuration 에 -Dspring.profiles.active="local" 를 추가합니다. (개발자 환경에 따라 local_kumc 등 다를 수 있음)

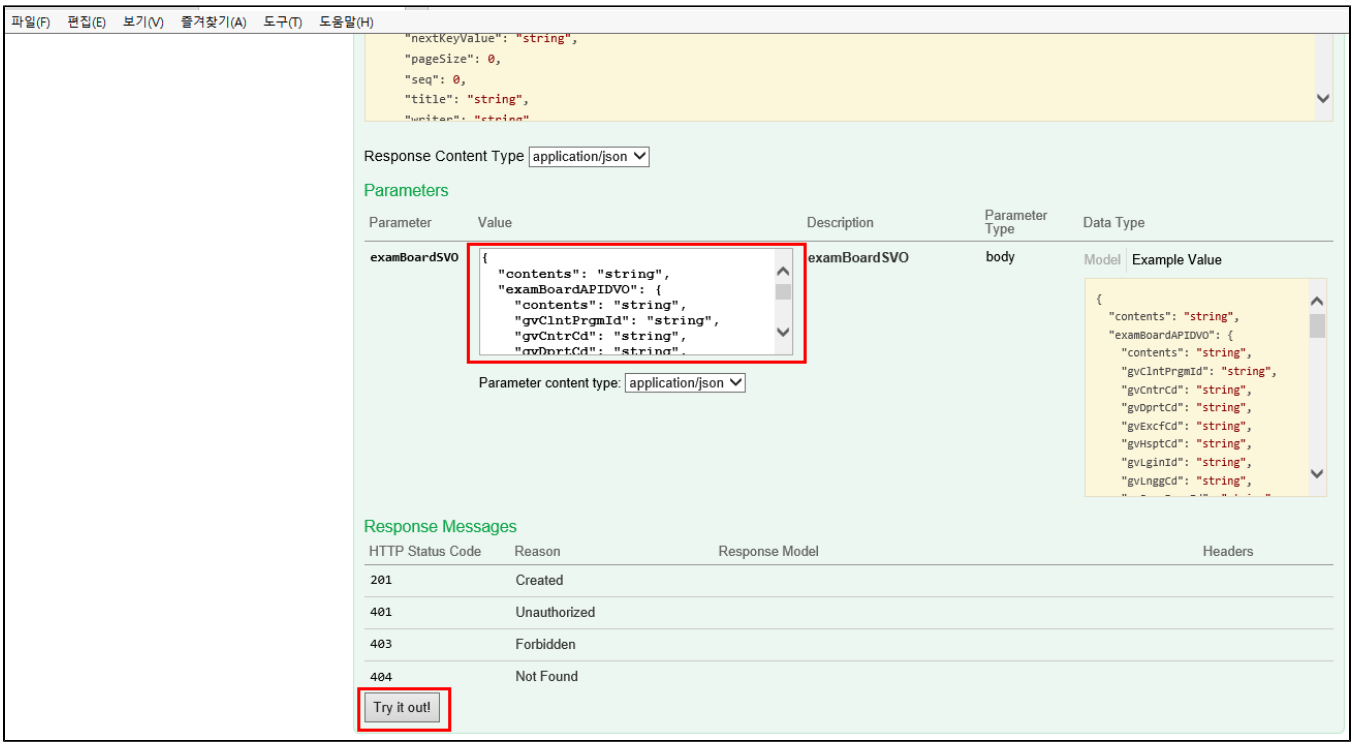


서버 Srart 후 http://localhost:8080/DOM_EHR_API/swagger-ui.html 로 접속합니다.



테스트 하고자 하는 API를 선택한 후 파라미터 값등을 입력합니다.

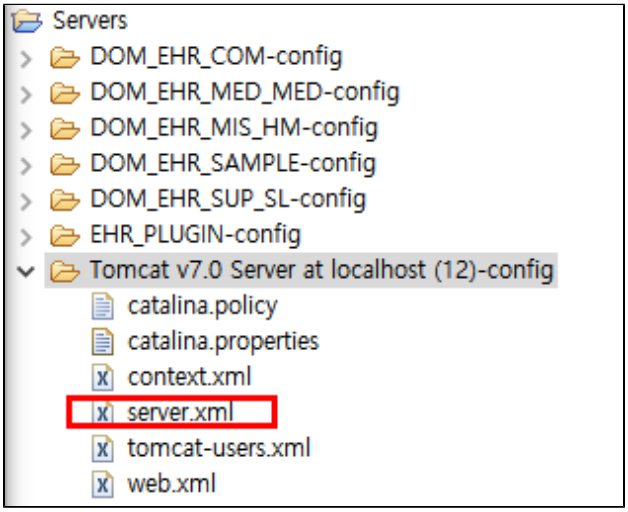
Try it out! 버튼을 클릭해서 응답 데이터를 확인 할 수 있습니다.



해당 샘플소스는 DOM_EHR_API 프로젝트의 sample 패키지내에 소스를 참조합니다.

■ **파라미터가 한글일 때 깨지는 현상이 발생하는 경우 조치방법**

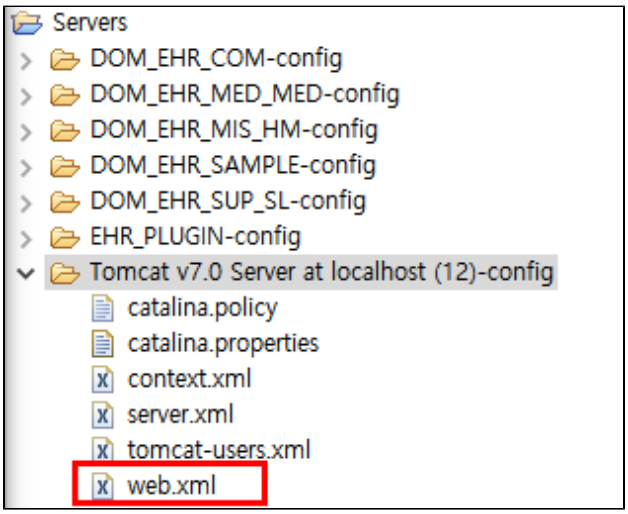
1) API 서버 설정파일인 server.xml을 수정합니다.



`URIEncoding="UTF-8"` .

```
<Connector connectionTimeout="20000" port="7090" protocol="HTTP/1.1" redirectPort="8443" URIEncoding="UTF-8"/>
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" URIEncoding="UTF-8"/>
```

2) API 서버 설정파일인 web.xml을 수정합니다.



.

```
<filter>
     <filter-name>setCharacterEncodingFilter</filter-name>
     <filter-class>org.apache.catalina.filters.SetCharacterEncodingFilter</filter-class>
     <init-param>
         <param-name>encoding</param-name>
         <param-value>UTF-8</param-value>
     </init-param>
     <async-supported>true</async-supported>
</filter>


<filter-mapping>
     <filter-name>setCharacterEncodingFilter</filter-name>
     <url-pattern>/*</url-pattern>
</filter-mapping>
```