**컨트롤러에서 결과를 받아 출력시킬 view들 만들기.  
🡺 03->6.1 화면처리**

**컨트롤러의 동작에 대하여**

@Controller  
@RequestMapping(“/ex”)  
public class ???Controller{  
 @GetMapping(“/home”)  
 public String home( ){  
 return “index”;  
 }  
   
 @PostMapping(“/create”)  
 public String create(BoardDTO board, RedirectAttributes ra){  
 service.create(board);  
 ra.addFlashAttribute(“result”, board.getNo( ));  
 return “redirect:/board”;  
 }  
  
 @GetMapping({“/get”, “/update”})  
 public void get(@RequestParam(“no”)Long no, Model model){  
 model.addAttribute(“board”, service.get(no));  
 }  
}

**뷰 페이지의 데이터 활용에 대하여 EL과 JSTL**

${ 변수 }

${ 객체변수.필드명 }

<c:forEach var=”board” items=”${list}”> …${ board.no } … ${ board.name } … </c:forEach>

…

**뷰 꾸미기**

“Bootstrap & fontawesome cdn” Header에 넣기  
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css"/>  
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">  
<link rel="stylesheet" href="/resources/css/main.css" />  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>

공통파트 작성 후 임포트하기  
<%@ include file=”공통파트헤더.jsp경로“ %>  
 본문 내용.  
<%@ include file=”공통파트푸터.jsp경로“ %>

전역 스타일시트 작성  
@import url('https://fonts.googleapis.com/css2?family=Noto+Sans+KR:wght@100;200;300;400;500;600;700;800;900&display=swap');  
\* { font-family: 'Noto Sans KR' }

각 페이지 Jstl 설정  
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

**파일 업로드  
A파일시스템에 업로드된 파일을 저장해 놓고 파일에 대한 경로를 디비에 저장/관리.  
B디비 테이블에 파일 자체를 blob타입으로 저장/관리.  
A방법으로 진행. binary데이터 다룰때는 stream작업이 필요함.**

파일 업로드 동작 과정 개요 :

A 인풋에 등록된 파일(바이너리 데이터)가 요청 바디에 들어가 서버로 전송된다.

B 서버에 도착해서, 필터들을 통과하고 해당 url을 받는 컨트롤러가 요청을 받는다.

C 해당 url을 담당하는 메소드에서 인자로써

파일 업로드 동작 상세과정 :

파일 업로드 동작 구현 :

**환경 및 준비 :**   
**파일 업로드를 위한 설정과 과정 – 바이너리 데이터는 멀티 파트란 인코딩을 사용한다**먼저 servletconfig에 MultipartResolver를 생성하며 Bean객체로 컨텍스트에 등록하는 메서드가 필요  
@Bean public MultipartResolver multipartResolver( ){  
 StandardServletMulitpartResolver resolver=new StandardServiceMultipartResolver( );  
 return resolver;  
}

그리고 webconfig에 파일업로드위치, 한파일최대사이즈, 한요청마다최대사이즈, 메모리쓸지 임시파일에 기록할지에 대한 기준 용량.에 대한 지정을하고 이를 등록하는 코드가 필요하다.  
@Override protected void customizeRegistration(ServletRegistration.Dynamic registration){  
 MultipartConfigElement multipartConfig=new MultipartConfigElement(  
 LOCATION, MAX\_FILE\_SIZE, MAX\_REQUEST\_SIZE, FILE\_SIZE\_THRESHHOLD  
 );  
 registration.setMultipartConfig(multipartConfig);  
}

**데이터 준비 :**

활용 :

파일 업로드 페이지 src > main > webapp > web-inf > views > sample > uploadform.jsp  
<form action=”/sample/uploadformpost” method=”post” enctype=”multipart/form-data”>  
 <input type=”file” name=”files”/>  
 <input type=”file” name=”files”/>  
 <input type=”submit”/>  
</form>  
업로드할 파일들을 받는 컨트롤러 메서드로 받을 수 있다.  
@PostMapping(“/uploadformpost”) public void uploadpost(ArrayList<MultipartFile> files){ … }

파일 다운로드  
A어디다 저장할지 저장 대화 상자가 표시되는 경우  
B그냥 바로보기/바로다운로드로 넘어갈 때