KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

프레젠테이션(웹) 계층의 CRUD 구현

[KB] IT's Your Life

KB 국민은행

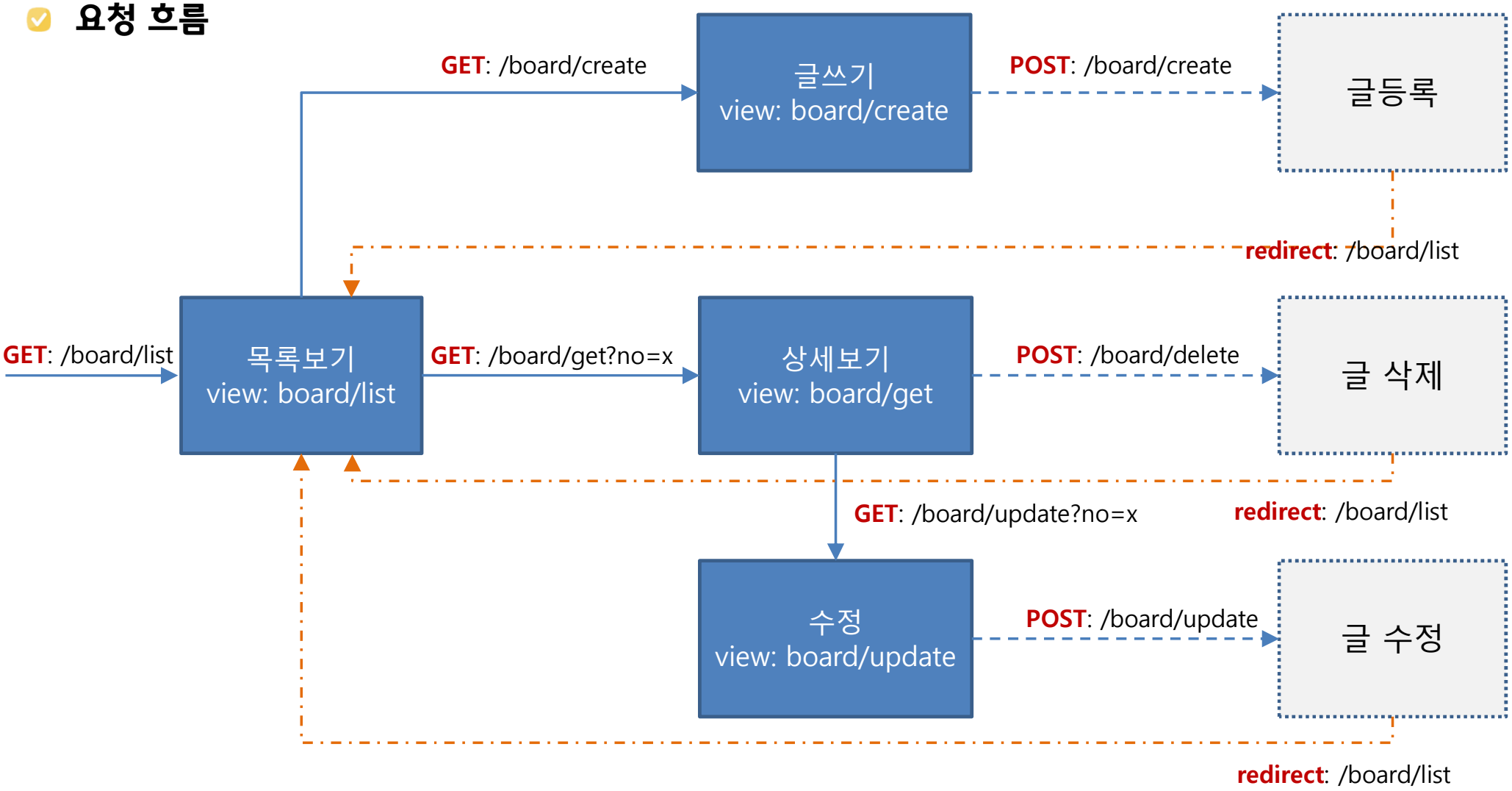
BoardController의 작성

BoardController의 분석

Task	URL	Method	Parameter	Form	URL 이동
전체 목록	/board/list	GET			
등록 처리	/board/create	POST	모든 항목	입력화면 필요	이동
조회	/board/get	GET	no=123		
수정 처리	/board/update	POST	no	입력화면 필요	이동
삭제 처리	/board/delete	POST	모든 항목	입력화면 필요	이동

화면 처리

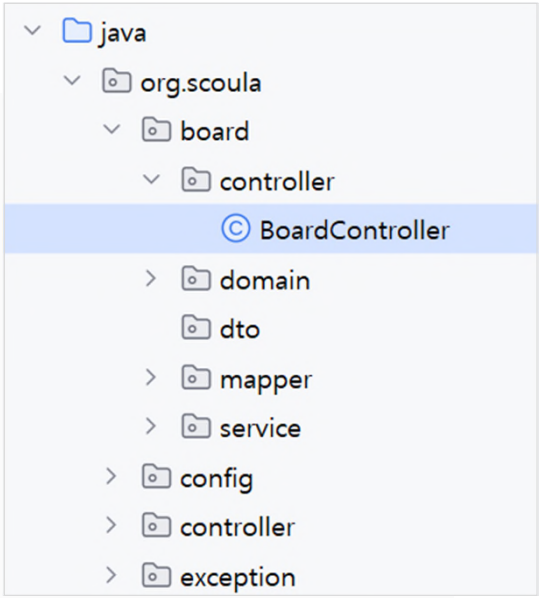
요청 흐름


GET: /board/create
글쓰기
view: board/create
POST: /board/create
글등록
redirect: /board/list
GET: /board/list
목록보기
view: board/list
GET: /board/get?no=x
상세보기
view: board/get
POST: /board/delete
글 삭제
redirect: /board/list
GET: /board/update?no=x
수정
view: board/update
POST: /board/update
글 수정
redirect: /board/list

BoardController의 작성

BoardController.java

```
package org.scoula.board.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import lombok.extern.log4j.Log4j2;

@Log4j2
@Controller
@RequestMapping("/board")
@RequiredArgsConstructor
public class BoardController {


        final private BoardService service;

}
```



BoardController의 작성

ServletConfig.java

```
...

@EnableWebMvc
@ComponentScan(basePackages = {
      "org.scoula.exception",
      "org.scoula.controller",
      "org.scoula.board.controller"
})
public class ServletConfig implements WebMvcConfigurer {
...
}
```

BoardController의 작성

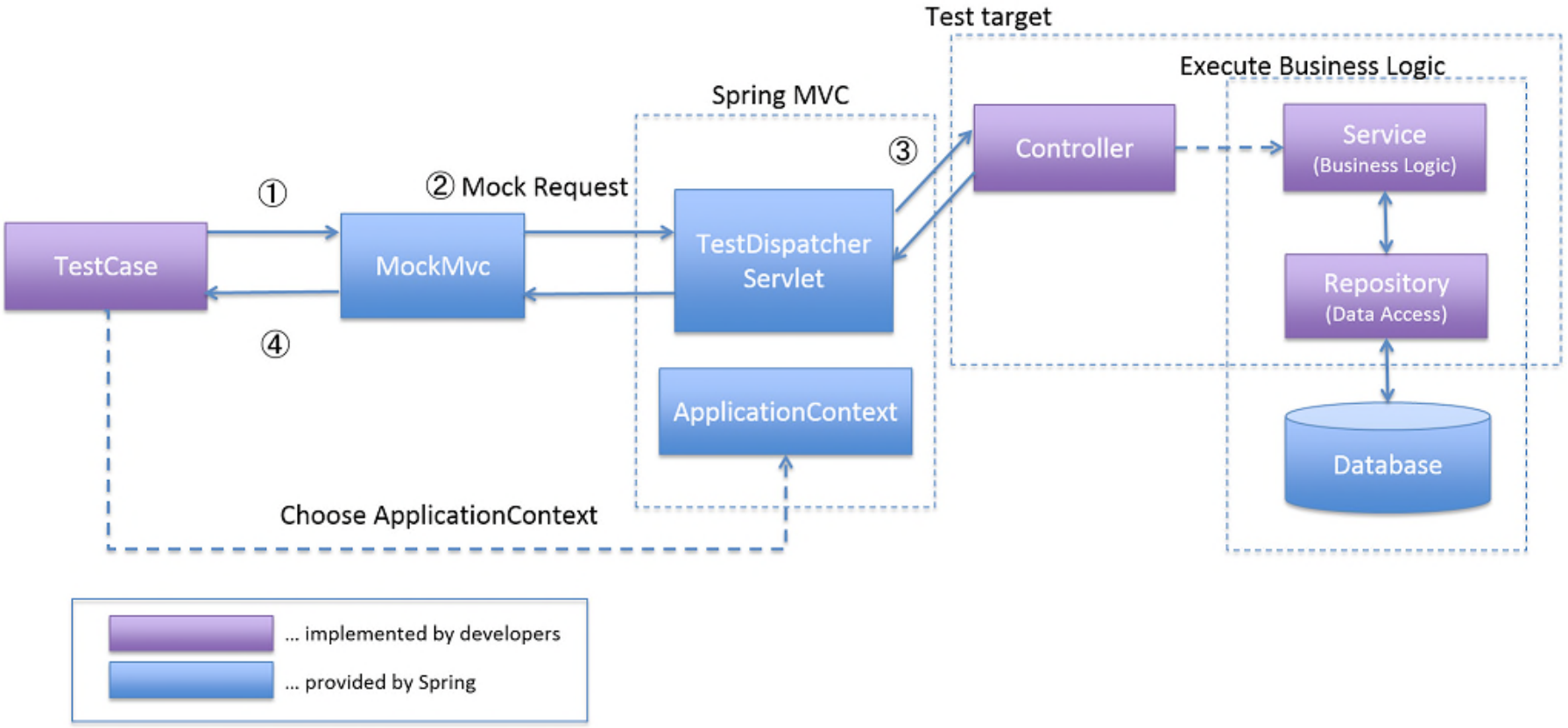
컨트롤러 테스트

- 웹 요청을 실제로 하는 것이 아님
- MockMvc
  - 웹 서버에 요청을 보낸 것과 같은 효과를 내는 메서드 제공
  - 메서드 호출로 웹 요청(GET/POST/UPDATE/DELETE 등)을 수행

  - 테스트 마다 새로운 MockMvc 객체 생성이 필요

  → @BeforeEach 어노테이션 메서드 설정

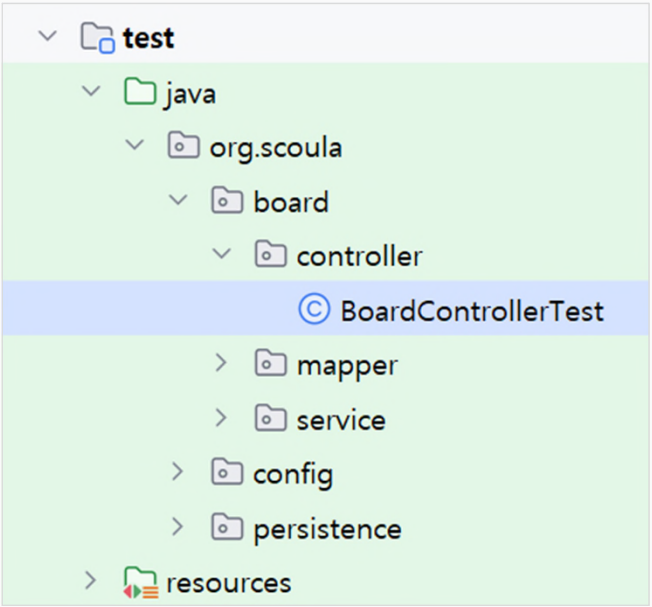
컨트롤러 테스트


Test target
Execute Business Logic
Spring MVC
Controller
Service
(Business Logic)
③
①
② Mock Request
TestCase
MockMvc
TestDispatcher
Servlet
Repository
(Data Access)
④
ApplicationContext
Database
Choose ApplicationContext
... implemented by developers
... provided by Spring

BoardController의 작성

test :: BoardControllerTest.java

```
…

@ExtendWith(SpringExtension.class)
@WebAppConfiguration
@ContextConfiguration(classes = {
    RootConfig.class,
    ServletConfig.class
})
@Log4j2
public class BoardControllerTest {
    @Autowired
    BoardService service;

    @Autowired
    private WebApplicationContext ctx;

    private MockMvc mockMvc;

    @BeforeEach
    public void setup() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
    }
}
```



BoardController의 작성 – 목록 처리와 테스트

목록 요청

- GET 요청
  - url: /board/list
  - Model 구성
    - 속성명 : list
    - 속성값: service의 getList()로 목록
  - View 이름: board/list

BoardController의 작성 – 목록 처리와 테스트

BoardController.java

```
...
@RequestMapping("/board")
public class BoardController {
        ...

        @GetMapping("/list")
        public void list(Model model) {

                log.info("list");
                model.addAttribute("list", service.getList());

        }
}
```

GET :: /board/list

BoardController의 작성 – 목록에 대한 처리와 테스트

Get 요청 만들기

- MockMvcRequestBuilders.get(url문자열)

MockMvc

- .perform(요청빌더)
  - 지정된 요청을 스프링 MVC가 처리
    → 지정된 URL을 처리하는 컨트롤러 메서드 호출
  - ResultActions 객체 리턴
  - ResultActions의 andReturn(): 컨트롤러의 처리를 결과를 리턴

MvcResult

- 컨트롤러에 의해 구성된 Model 및 View에 대한 정보 및 처리결과(상태 코드) 등을 가짐
- getModelAndView()　　: ModelAndView 객체 리턴

MockMvc

메서드	설명
standaloneSetup()	특정 컨트롤러를 MockMvc에 설정하여 테스트할 수 있는 환경을 구성
perform()	MockMvc를 사용하여 HTTP 요청을 실행(필수) ResultActions을 리턴
andExpect()	컨트롤러의 응답을 검증
andExpect(status().isOk())	응답 상태 코드가 200인지 확인
andExpect(content().string("expected"))	응답 본문의 내용이 "expected"인지 확인
andExpect(jsonPath("$.property").value("expected"))	JSON 응답에서 특정 속성의 값이 "expected"인지 확인.
andExpect(view().name("expectedView"))	응답에 대한 뷰의 이름이 "expectedView"인지 확인.
andExpect(model().attribute("attributeName", "expectedValue"))	모델 속성의 값이 "expectedValue"인지 확인.
andExpect(redirectedUrl("expectedUrl"))	리다이렉트된 URL이 "expectedUrl"인지 확인.

ResultActions

메서드	설명
andReturn()	**해결된 MvcResult 객체를 반환**
andReturn(MvcResult)	반환할 MvcResult를 설정
andDo(ResultHandler)	결과에 대해 추가 작업을 수행
andDo(ResultMatcher)	결과에 ResultMatcher를 추가
andExpect(ResultMatcher)	결과에 대한 기대치로 ResultMatcher를 추가
andForward()	요청을 다음 핸들러로 전달
andForward(String)	요청을 지정된 URL로 전달
andExpect(MockMvcResultMatchers)	MockMvcResultMatchers에서 ResultMatcher를 추가
andExpect(MockMvcResultHandlers)	MockMvcResultHandlers에서 ResultHandler를 추가
andReverse()	이전 전달을 뒤집습니다.
andForwardDefault()	요청을 기본 핸들러로 전달
andForwardDefault(String)	요청을 지정된 URL로 기본 핸들러로 전달

ResultActions

메서드	설명
andReturnDefault()	기본 핸들러에 대한 해결된 MvcResult 객체를 반환
andReturnDefault(MvcResult)	기본 핸들러에 반환할 MvcResult를 설정
andDoDefault()	기본 핸들러에 대해 추가 작업을 수행
andDoDefault(ResultHandler)	지정된 핸들러를 사용하여 기본 핸들러에 대해 추가 작업을 수행
andDoDefault(ResultMatcher)	기본 핸들러에 ResultMatcher를 추가
andExpectDefault(ResultMatcher)	기본 핸들러에 대한 결과 기대치로 ResultMatcher를 추가
andForwardNamed(String)	지정된 URL로 명명된 핸들러로 요청을 전달
andReturnNamed(String)	명명된 핸들러에 대한 해결된 MvcResult 객체를 반환
andDoNamed(String)	명명된 핸들러에서 추가 작업을 수행
andDoNamed(String, ResultHandler)	지정된 핸들러를 사용하여 명명된 핸들러에서 추가 작업을 수행
andDoNamed(String, ResultMatcher)	명명된 핸들러에 ResultMatcher를 추가
andExpectNamed(String, ResultMatcher)	명명된 핸들러에 대한 결과 기대치로 ResultMatcher를 추가

BoardController의 작성 - 목록 처리와 테스트

test :: BoardControllerTest.java

```
        @Test
        public void list() throws Exception {
    ModelMap model = mockMvc.perform(MockMvcRequestBuilders.get("/board/list"))   // ResultActions 리턴
            .andReturn()                                                          // MvcResult 리턴
            .getModelAndView()      // ModelAndView 리턴
            .getModelMap();                         // Model 리턴
    log.info(model);
}
```

INFO org.scoula.board.controller.BoardController(list:21) - list
INFO org.scoula.board.service.BoardServiceImpl(getList:23) - getList.....
INFO jdbc.sqlonly(sqlOccurred:228) - **select * from tbl_board order by no desc**

INFO org.scoula.board.controller.BoardControllerTest(list:48) - {list=[BoardDTO(no=8, title=새로 작성하는 글, content=새로 작성하는 내용, writer=user1, regDate=Wed Jan 15 13:23:02 KST 2025, updateDate=Wed Jan 15 13:23:02 KST 2025), BoardDTO(no=7, title=새로 작성하는 글, content=새로 작성하는 내용, writer=user0, regDate=Wed Jan 15 12:56:07 KST 2025, updateDate=Wed Jan 15 12:56:07 KST 2025), BoardDTO(no=6, title=새로 작성하는 글, content=새로 작성하는 내용, writer=user0, regDate=Wed Jan 15 12:52:46 KST 2025, updateDate=Wed Jan 15 12:52:46 KST 2025), BoardDTO(no=5, title=수정된 제목, content=수정된 내용, writer=user00, regDate=Wed Jan 15 11:53:10 KST 2025, updateDate=Wed Jan 15 12:57:42 KST 2025), BoardDTO(no=4, title=테스트 제목4, content=테스트 내용4, writer=user00, regDate=Wed Jan 15 11:53:10 KST 2025, updateDate=Wed Jan 15 11:53:10 KST 2025), BoardDTO(no=1, title=제목 수정합니다., content=테스트 내용1, writer=user00, regDate=Wed Jan 15 11:53:10 KST 2025, updateDate=Wed Jan 15 13:24:31 KST 2025)]}

BoardController의 작성 - 등록 처리와 테스트

새 글 등록

- GET 요청
  - url: /board/create
  - view 이름: board/create

- POST 요청
  - url: /board/create
  - 파라미터: BoardDTO
  - 처리: service.create()로 실제 등록
  - view 이름 : redirect:/board/list

BoardController의 작성 - 등록 처리와 테스트

BoardController.java

```
...
@GetMapping("/create")
public void create() {
        log.info("create");
}

@PostMapping("/create")
public String create(BoardDTO board) {

        log.info("create: " + board);

        service.create(board);

        return "redirect:/board/list";
}
```

GET :: /board/create

POST :: /board/create

MockMvc로 Post 요청 테스트하기

```
MockMvcRequestBuilders.post(url 문자열)
                      .param(키1, 값1)                    // form 요소
                      .param(키2, 값2)
```

BoardController의 작성 - 등록 처리와 테스트

test :: BoardControllerTest.java

```
@Test
void create() throws Exception {
   String viewName = mockMvc.perform(MockMvcRequestBuilders.get("/board/create"))          // ResultActions 리턴
            .andReturn()                                  // MvcResult 리턴
            .getModelAndView()      // ModelAndView 리턴
            .getViewName();
   log.info(viewName);
}
```

```
INFO  org.springframework.mock.web.MockServletContext(log:455) - Initializing Spring TestDispatcherServlet ''
INFO  org.springframework.test.web.servlet.TestDispatcherServlet(initServletBean:525) - Initializing Servlet ''
INFO  org.springframework.test.web.servlet.TestDispatcherServlet(initServletBean:547) - Completed initialization in 0 ms
INFO  org.scoula.board.controller.BoardController(create:31) - create
INFO  org.scoula.board.controller.BoardControllerTest(create:59) - board/create
```

test :: BoardControllerTest.java

```
        @Test
        public void postCreate() throws Exception {

                String resultPage = mockMvc.perform(
                                        MockMvcRequestBuilders.post("/board/create")
                                                .param("title", "테스트 새글 제목")
                                                .param("content", "테스트 새글 내용")
                                                .param("writer", "user1")
    )
                                        .andReturn()
                                        .getModelAndView()
                                        .getViewName();

                log.info(resultPage);
        }
```

```
INFO  org.scoula.board.controller.BoardController(create:37) - create: BoardDTO(no=null, title=테스트 새글 제목, content=테스트 새글 내용, writer=user1, regDate=null, updateDate=null)
 INFO  org.scoula.board.service.BoardServiceImpl(create:43) - create.....BoardDTO(no=null, title=테스트 새글 제목, content=테스트 새글 내용, writer=user1, regDate=null, updateDate=null)
 INFO  jdbc.sqlonly(sqlOccurred:228) - insert into tbl_board (title, content, writer) values ('테스트 새글 제목', '테스트 새글 내용', 'user1')

 INFO  jdbc.sqlonly(sqlOccurred:228) - SELECT LAST_INSERT_ID()

 INFO  org.scoula.board.controller.BoardControllerTest(postCreate:78) - redirect:/board/list
```

BoardController의 작성 – 조회 처리와 테스트

글 상세 보기(조회)

GET 요청

- url: /board/get
- 파라미터
  - no: 글 번호
- Model 구성
  - 속성명 : board
  - 속성값: service.get()으로 BoardDTO 획득
- View 이름: board/get

수정의 GET 요청 처리와 동일

- url: /board/update

BoardController의 작성 – 조회 처리와 테스트

BoardController.java

GET :: /board/get
GET :: /board/update

```
...
@GetMapping({ "/get", "/update" })
public void get(@RequestParam("no") Long no, Model model) {

        log.info("/get or update");
        model.addAttribute("board", service.get(no));
}
```

BoardController의 작성 - 조회 처리와 테스트

Get 요청의 쿼리 파라미터 테스트

```
MockMvcRequestBuilders.get(url 문자열)
                      .param(키1, 값1)                    // 쿼리 파리미터
                      .param(키2, 값2)
```

BoardController의 작성 - 조회 처리와 테스트

test :: BoardControllerTest.java

```
@Test
public void get() throws Exception {
   ModelMap model = mockMvc.perform(MockMvcRequestBuilders.get("/board/get").param("no", "1"))
         .andReturn()
         .getModelAndView()
         .getModelMap();
   log.info(model);
}
```

```
INFO  org.scoula.board.controller.BoardController(get:48) - /get or update
INFO  org.scoula.board.service.BoardServiceImpl(get:32) - get.....1
INFO  jdbc.sqlonly(sqlOccurred:228) - select * from tbl_board where no = 1

INFO  org.scoula.board.controller.BoardControllerTest(get:91) - {board=BoardDTO(no=1, title=제목 수정합니다., content=테스트 내용1, writer=user00, regDate=Wed Jan 15 11:53:10 KST 2025, updateDate=Wed Jan 15 13:24:31 KST 2025), org.springframework.validation.BindingResult.board=org.springframework.validation.BeanPropertyBindingResult: 0 errors}
```

BoardController의 작성 – 수정 처리와 테스트

글 수정 처리

GET 요청

- url: /board/update
- BoardController의 get()에서 처리

POST 요청

- url: /board/update
- 파라미터: BoardDTO
- 처리: service.update()로 실제 수정
- view 이름 : redirect:/board/list

BoardController의 작성 – 수정 처리와 테스트

BoardController.java

POST :: /board/update

```
@PostMapping("/update")
public String update(BoardDTO board) {
        log.info("update:" + board);

        service.update(board);

        return "redirect:/board/list";
}
```

BoardController의 작성 – 수정 처리와 테스트

test :: BoardControllerTest.java

```
@Test
public void update() throws Exception {

        String resultPage = mockMvc.perform(
MockMvcRequestBuilders.post("/board/update")
  .param("no", "1")
  .param("title", "수정된 테스트 새글 제목")
                                                  .param("content", "수정된 테스트 새글 내용")
  .param("writer", "user00"))
                          .andReturn()
                        .getModelAndView()
                        .getViewName();

        log.info(resultPage);

}
```

INFO org.scoula.board.controller.BoardController(update:54) - update:BoardDTO(no=1, title=수정된 테스트 새글 제목, content=수정된 테스트 새글 내용, writer=user00, regDate=null, updateDate=null)
INFO org.scoula.board.service.BoardServiceImpl(update:52) - update.....BoardDTO(no=1, title=수정된 테스트 새글 제목, content=수정된 테스트 새글 내용, writer=user00, regDate=null, updateDate=null)
INFO jdbc.sqlonly(sqlOccurred:228) - **update tbl_board set title = '수정된 테스트 새글 제목', content = '수정된 테스트 새글 내용', writer = 'user00', update_date = now() where no = 1**

INFO org.scoula.board.controller.BoardControllerTest(update:107) - **redirect:/board/list**

BoardController의 작성 – 삭제 처리와 테스트

삭제 처리

- GET 요청 없음
- POST 요청
  - url: /board/delete
  - 파라미터: no – 삭제할 글 번호
  - 처리: service.delete()로 실제 삭제
  - view 이름 : redirect:/board/list

BoardController의 작성 – 삭제 처리와 테스트

BoardController.java

POST :: /board/delete

```
...
@PostMapping("/delete")
public String delete(@RequestParam("no") Long no) {

        log.info("delete..." + no);
        service.delete(no);

        return "redirect:/board/list";
}
```

BoardController의 작성 – 삭제 처리와 테스트

test :: BoardControllerTest.java

```
        @Test
        public void delete() throws Exception {
                // 삭제전 데이터베이스에 게시물 번호 확인할 것
                String resultPage = mockMvc.perform(
    MockMvcRequestBuilders
      .post("/board/delete")
      .param("no", "25")
)
                                                .andReturn()
                                                .getModelAndView()
                                                .getViewName();

                log.info(resultPage);
        }
```

```
INFO  org.scoula.board.controller.BoardController(delete:64) - delete...25
INFO  org.scoula.board.service.BoardServiceImpl(delete:60) - delete....25
INFO  jdbc.sqlonly(sqlOccurred:228) - delete from tbl_board where no = 25

INFO  org.scoula.board.controller.BoardControllerTest(delete:123) - redirect:/board/list
```