1. 진행되고 있는 사이트를 테스트해본다.
2. 구조분석
   1. 설치방법(vite 처리방법선택)
   2. 설치된 라이브러리(package.json 버전)
   3. 환경설정(eslint.config.js) rules 우리가설정
3. 코드분석

3.1index.html ->리액트 대표 페이지

3.2main.jsx : index.html : <div id="root"></div> root 문석객체를 찿는다.

strictMode 해제시켜라.

3.3App.jsx : 라우터설정을 진행한다.

 <RouterProvider router={root} />

3.4.src/router/root.jsx (모든 경로설정)

주의]화면설계와 경로설계가 모두 이루어진 사항에서 진행한다.

const ModifyPage = lazy(() => import("../page/todo/ModifyPage"));

const root = createBrowserRouter([

{

    path: "/todo/modify/:tno",

    element: (

      <Suspense fallback={<Loading />}>

        <ModifyPage />

      </Suspense>

    ),

  },

]); 제작

3.5 Loading.jsx 로딩처리되는 컴포넌트 화면제작

import Loading from "../page/Loading";

3.6 공통컴포넌트를 제작(부트스트랩 : 공통컴포넌트)

Src/include/~, src/component/common/~ : 또다른 변수: 팀장미팅

Header, Footer, MenuAside, Button, Cart……….

우리가 가장신경써야할부분이 화면(리액트용 부트스트랩 제작판매용)

가격: 10만원~5만원 (4팀이 합쳐)

3.7 페이지제작(투드리스트: TodoAddPage, TodoListPage,

TodoModify, TodoRead, TodoCart, ~) src/page/todo/~

MainPage, AboutPage(사이트맵)

3.8 페이지제작 todo/ModifyPage.jsx

const ModifyPage = () => {

//URL 통해서 전달되는 페이지(1.param, 2.queryString)

const { tno } = useParams();

//이벤트처리는 공통으로관리 (/hooks/useCustomMove)

  const { moveToList, moveRead } = useCustomMove();

  return (

    <Container>

      <Header />

//3. 자식컴포넌트 (porps: pk, 이벤트처리)

      <ModifyComponent tno={tno} moveToList={moveToList} moveRead={moveRead} />

<Footer />

    </Container>

  );

};

3.9 내가공동처리할 작업 (전체관리)

- 이벤트처리 방식((/hooks/useCustomMove)

- 외부데이타 요청방식(Axios 구조 비동기처리방식)

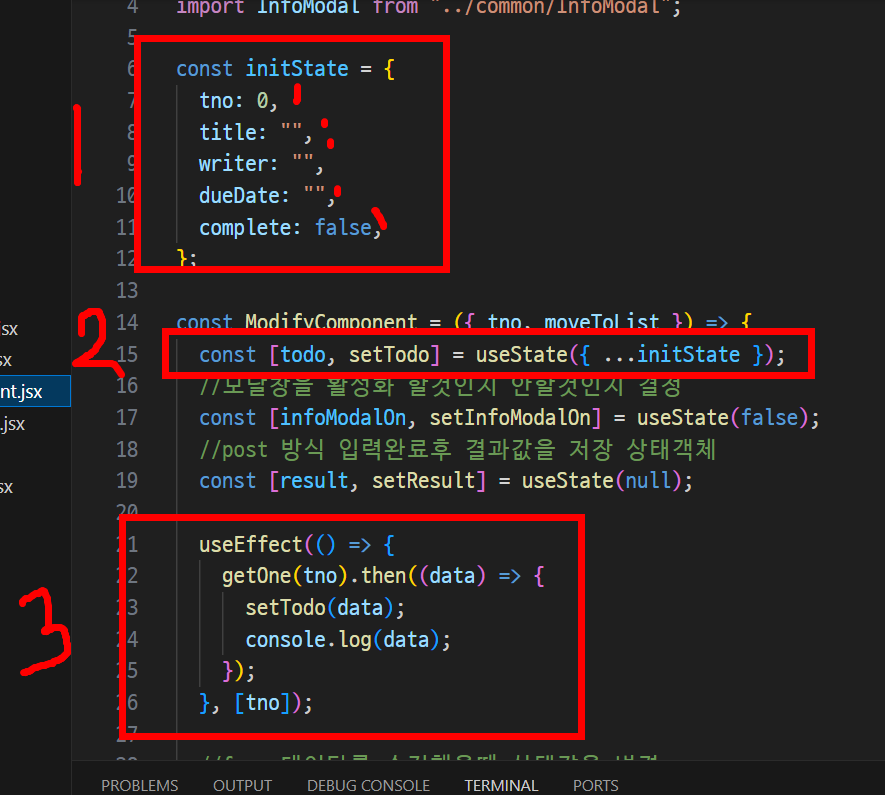
/api/TodoApi.jsx

-내영역에 필요한 공통컴포넌트 /component/todo/~

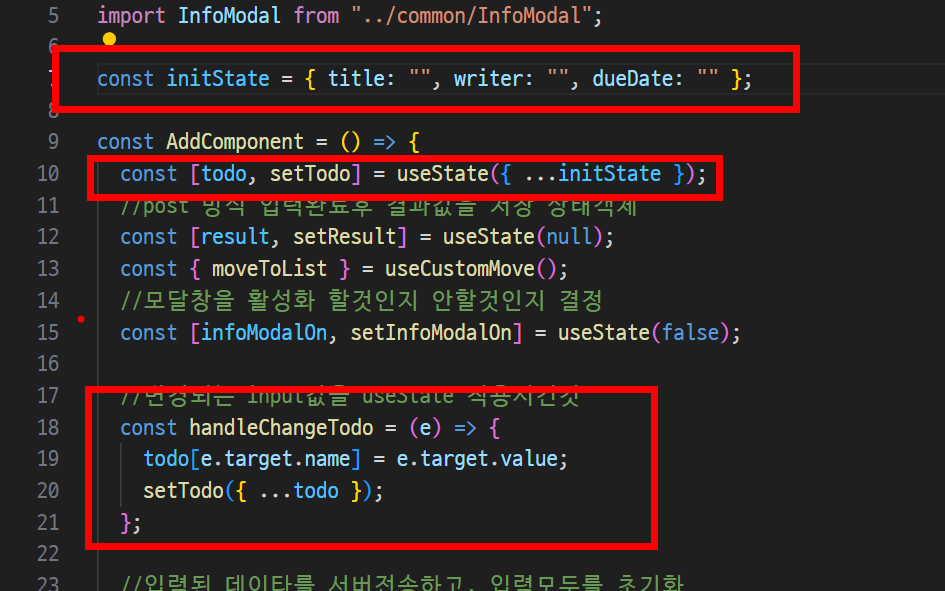
3.10 내가 해야될 개인화면(부트스트랩)

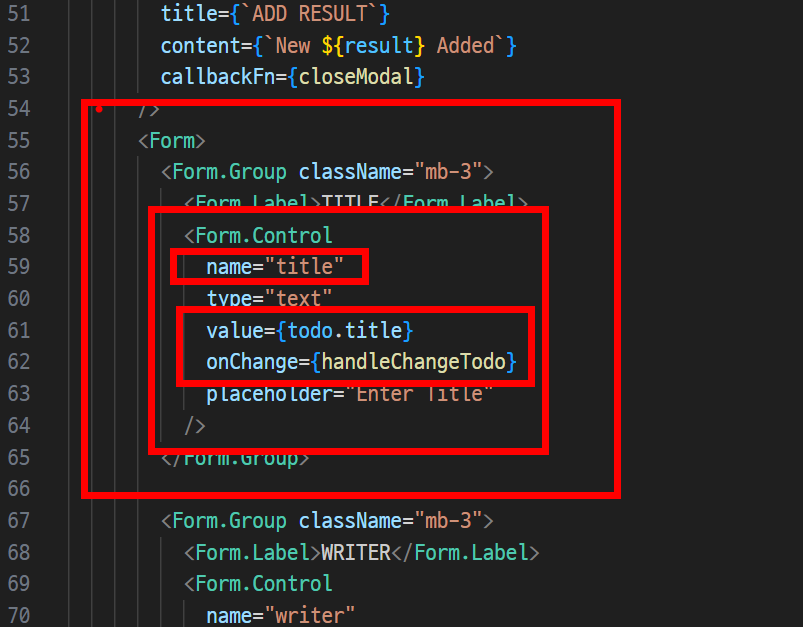
3.11 내가 컴포넌트에서 외부에서 자료 가져올 때 해야될방식

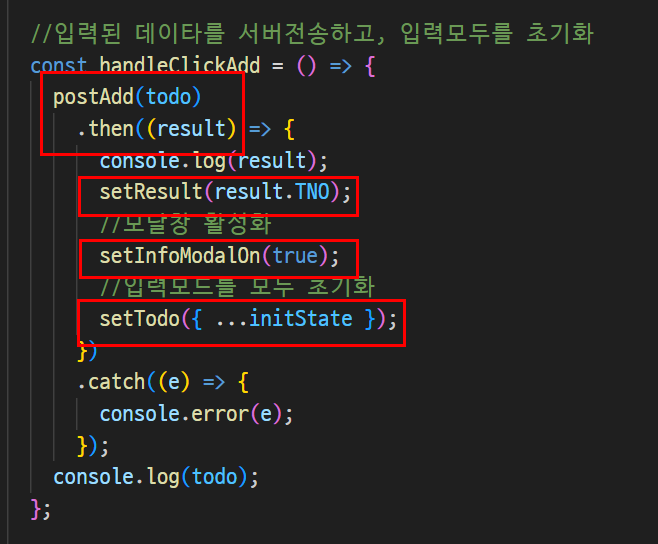
- useEffect 기능을 활용해라.



내가 컴포넌트에서 만든 내부에 외부로 보낼 때 해야될방식







3.12 외부에서 가져온 데이터를 어떻게 처리할까? 하나의 객체로 오면 화면에서 {todo.tno} ~~~~

객체배열로 받아다면, ( 객체배열.map((data,index) =>{}) ( 객체배열.filter((data,index) =>{}) ( 객체배열.foreach((data,index) =>{}), 삼항연산자 처리방식

3.13 나머지는 모두 화면처리 방식이다. (팀원들과 같이 화면설계를 들어간다.)

스프링부트

1. Sts3 선택 , jdk17, war, jpa, oracle, spring web, Lombok, web dev tool 설치진행
2. Pom.xml 의존성관련된 라이브러리 추가

**<!-- jasper의존성 설정 jsp -> servlet 변환 -->**

**<dependency>**

**<groupId>org.apache.tomcat.embed</groupId>**

**<artifactId>tomcat-embed-jasper</artifactId>**

**</dependency>**

**<!-- jstl 에 대한 두 개 라이브러리-->**

**<dependency>**

**<groupId>org.glassfish.web</groupId>**

**<artifactId>jakarta.servlet.jsp.jstl</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>jakarta.servlet.jsp.jstl</groupId>**

**<artifactId>jakarta.servlet.jsp.jstl-api</artifactId>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->**

**<dependency>**

**<groupId>org.projectlombok</groupId>**

**<artifactId>lombok</artifactId>**

**<version>1.18.30</version>**

**<scope>provided</scope>**

**</dependency>**

**//모델매퍼: jpa 를 사용하게되면 Todo.Class(디비에 테이블과 같은 매핑)**

**//클라이언트 요청을 받을 때 사용되는 매개변수 급 클래스로 취급하기에는 너무 중요하다고**

**//판단되서 예전처럼 자료를 받을수 있는 TodoVO.class 필요하고 그 역할을 해줄것을 찾는다.**

**//DTO(data transfer object) 그러면서 디비에 직접입력, 수정, 삭제, 리스트 (Todo.class)**

**// 역할을 분담시키고 (컨트롤러, 서비스, 서비스 임플 : DTO => repository Todo => 디비**

**결과값은 List<Todo>, Todo 상호변환이 필요하다. 이런부분을 자동으로 처리하는 프로그램을 만들고 싶다.**

**<!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->**

**<dependency>**

**<groupId>org.modelmapper</groupId>**

**<artifactId>modelmapper</artifactId>**

**<version>3.1.1</version>**

**</dependency>**

1. 환경설정
   1. application.properties
2. **pring.application.name=mallapi**
3. **## view resolver**
4. **spring.mvc.view.prefix=/WEB-INF/views/**
5. **spring.mvc.view.suffix=.jsp**
6. **## jdbc 설정**
7. **spring.datasource.driver-class-name=oracle.jdbc.OracleDriver**
8. **spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe**
9. **spring.datasource.username=mallapi**
10. **spring.datasource.password=mallapi**

**## jpa 설정**

1. **spring.jpa.database=oracle (jpa => 오라클 연결사용)**
2. **spring.jpa.database-platform=org.hibernate.dialect.OracleDialect**
3. **spring.jpa.properties.hibernate.id.new\_generator\_mappings=false**
4. **spring.jpa.hibernate.ddl-auto=update (테이블이 있으면 만들지 않고, 없으면 만든다)**
5. **## spring.jpa.hibernate.ddl-auto=create (테이블이 있으면 모두 제거해버리고 다시 만든다.)**
6. **spring.jpa.properties.hibernate.format\_sql=true (쿼리문이 만들어지는 것을 화면으로 정리해서 보여줘)**
7. **spring.jpa.generate-ddl=true**
8. **spring.jpa.show-sql=true**
9. **logging.level.org.hibernate=info (진행정보를 info 수준에서 보여줘.)**

3.2

-외부에서 오는 문자열에 날짜를 (문자열-날짜, 날짜 – 문자열)

-외부에서 cors 를 어떻게 처리할것인지 셋팅

**package com.kh.mallapi.config;**

**@Configuration**

**public class CustomServletConfig implements WebMvcConfigurer {**

**@Override**

**public void addFormatters(FormatterRegistry registry) {**

**registry.addFormatter(new LocalDateFormatter());**

**}**

**@Override**

**public void addCorsMappings(CorsRegistry registry) {**

**registry.addMapping("/\*\*")**

**.allowedOrigins("\*")**

**.allowedMethods("HEAD", "GET", "POST", "PUT", "DELETE", "OPTIONS")**

**.maxAge(300)**

**.allowedHeaders("Authorization", "Cache-Control", "Content-Type");**

**}**

**}**

매퍼전략(기준점이 무엇으로 매핑시킬것인데 알려줘)

**@Configuration**

**public class RootConfig {**

**@Bean**

**public ModelMapper getMapper() {**

**ModelMapper modelMapper = new ModelMapper();**

**//1. 두개클래스에서 필드이름이 같으면 매칭**

**//2. 접근제어자 private 매칭**

**//3. 접근전략 글자가 유사해도 매칭**

**modelMapper.getConfiguration().setFieldMatchingEnabled(true)**

**.setFieldAccessLevel(org.modelmapper.config.Configuration.AccessLevel.*PRIVATE*)**

**.setMatchingStrategy(MatchingStrategies.*LOOSE*);**

**return modelMapper;**

**}**

**}**

3.3 예외처리 추가

**@RestControllerAdvice**

**public class CustomControllerAdvice {**

**// 없는데이터를 요청했을때**

**@ExceptionHandler(NoSuchElementException.class)**

**protected ResponseEntity<?> notExist(NoSuchElementException e) {**

**String msg = e.getMessage();**

**return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).body(Map.*of*("msg", msg));**

**}**

**//폼양식에 맞지 않는 데이터를 전송했을때**

**@ExceptionHandler(MethodArgumentNotValidException.class)**

**protected ResponseEntity<?> handleIllegalArgumentException(MethodArgumentNotValidException e) {**

**String msg = e.getMessage();**

**return ResponseEntity.*status*(HttpStatus.*NOT\_ACCEPTABLE*).body(Map.*of*("msg", msg));**

**}**

**//예외를 처리하고 싶을때 추가하면됨.**

**}**

3.4 LocalFormatter

package com.kh.mallapi.controller.formatter;

import java.text.ParseException;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.util.Locale;

import org.springframework.format.Formatter;

public class LocalDateFormatter implements Formatter<LocalDate> {

@Override

public String print(LocalDate object, Locale locale) {

return DateTimeFormatter.ofPattern("yyyy-MM-dd").format(object);

}

@Override

public LocalDate parse(String text, Locale locale) throws ParseException {

return LocalDate.parse(text, DateTimeFormatter.ofPattern("yyyy-MM-dd"));

}

}

1. 각자컨틀롤러 프로그램 진행
   1. jpa 도메인설계 -> jpaDTO 설계 -> jpa Repository 설계 -> 서비스설계 -> 서비스impl 설계

* 컨트럴로 설계
  1. jpa 도메인설계

|  |
| --- |
| 1. package com.kh.mallapi.domain; 2. import java.time.LocalDate; 3. import jakarta.persistence.Entity; 4. import jakarta.persistence.GeneratedValue; 5. import jakarta.persistence.GenerationType; 6. import jakarta.persistence.Id; 7. import jakarta.persistence.SequenceGenerator; 8. import jakarta.persistence.Table; 9. import lombok.AllArgsConstructor; 10. import lombok.Builder; 11. import lombok.Getter; 12. import lombok.NoArgsConstructor; 13. import lombok.ToString; 14. @Entity 15. @Table(name = "tbl\_todo") 16. @SequenceGenerator(name = "TODO\_SEQ\_GEN", sequenceName = "TODO\_SEQ", initialValue = 1, allocationSize = 1) 17. @Getter 18. @ToString 19. @Builder 20. @AllArgsConstructor 21. @NoArgsConstructor 22. public class Todo { 23. @Id 24. @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "TODO\_SEQ\_GEN") 25. private Long tno; 26. private String title; 27. private String writer; 28. private boolean complete; 29. private LocalDate dueDate; 30. public void changeTitle(String title) { 31. this.title = title; 32. } 33. public void changeWriter(String writer) { 34. this.writer = writer; 35. } 36. public void changeComplete(boolean complete) { 37. this.complete = complete; 38. } 39. public void changeDueDate(LocalDate dueDate) { 40. this.dueDate = dueDate; 41. } 42. } |

4.3 TodoDTO 설계

|  |
| --- |
| package com.kh.mallapi.dto;  import java.time.LocalDate;  import com.fasterxml.jackson.annotation.JsonFormat;  import lombok.AllArgsConstructor;  import lombok.Builder;  import lombok.Data;  import lombok.NoArgsConstructor;  @Data  @Builder  @AllArgsConstructor  @NoArgsConstructor  public class TodoDTO {  private Long tno;  private String title;  private String writer;  private boolean complete;  @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd")  private LocalDate dueDate;  } |

4.5 TodoRepository.class

|  |
| --- |
| package com.kh.mallapi.repository;  import org.springframework.data.jpa.repository.JpaRepository;  import com.kh.mallapi.domain.Todo;  //c save, u findByid() => setter => save, r findByid(), d 설정 deleteByid  public interface TodoRepository extends JpaRepository<Todo, Long> {  } |

4.6 TodoService.class

|  |
| --- |
| package com.kh.mallapi.service;  import com.kh.mallapi.dto.PageRequestDTO;  import com.kh.mallapi.dto.PageResponseDTO;  import com.kh.mallapi.dto.TodoDTO;  public interface TodoService {  //insert  public Long register(TodoDTO todoDTO);  //update  void modify(TodoDTO todoDTO);  //select  public TodoDTO get(TodoDTO todoDTO);  //delete  public void remove(TodoDTO todoDTO);  //페이징처리 및 리스트요청  public PageResponseDTO<TodoDTO> list(PageRequestDTO pageRequestDTO);  } |

4.7 TodoServiceImpl.class

|  |
| --- |
| package com.kh.mallapi.service;  import java.util.List;  import java.util.Optional;  import java.util.stream.Collectors;  import org.modelmapper.ModelMapper;  import org.springframework.data.domain.Page;  import org.springframework.data.domain.PageRequest;  import org.springframework.data.domain.Pageable;  import org.springframework.data.domain.Sort;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  import com.kh.mallapi.domain.Todo;  import com.kh.mallapi.dto.PageRequestDTO;  import com.kh.mallapi.dto.PageResponseDTO;  import com.kh.mallapi.dto.TodoDTO;  import com.kh.mallapi.repository.TodoRepository;  import lombok.RequiredArgsConstructor;  import lombok.extern.log4j.Log4j2;  @Service  @Transactional  @Log4j2  @RequiredArgsConstructor  public class TodoServiceImpl implements TodoService {  // 생성자 의존성 주입  private final TodoRepository todoRepository;  private final ModelMapper modelMapper;    // todo insert  @Override  public Long register(TodoDTO todoDTO) {  Todo todo = modelMapper.map(todoDTO, Todo.class);  Todo saveTodo = todoRepository.save(todo);  return saveTodo.getTno();  }  //todo select  @Override  public TodoDTO get(TodoDTO todoDTO) {  Optional<Todo> result = todoRepository.findById(todoDTO.getTno());  Todo todo = result.orElseThrow();  TodoDTO \_todoDTO = modelMapper.map(todo, TodoDTO.class);  return \_todoDTO;  }    //todo Modify( 1.findById => 2.Todo => 3.setter 진행 => 4.save)  @Override  public void modify(TodoDTO todoDTO) {  //1번, 2번  Optional<Todo> result = todoRepository.findById(todoDTO.getTno());  Todo todo = result.orElseThrow();  //3번 수정하고 싶은것만 셋터진행  todo.changeTitle(todoDTO.getTitle());  todo.changeDueDate(todoDTO.getDueDate());  todo.changeComplete(todoDTO.isComplete());  //4번 save시키면 insert(테이블 tno 존재하지않을때)가 되지않고 update 진행이된다.  todoRepository.save(todo);  }  //Todo delete  @Override  public void remove(TodoDTO todoDTO) {  todoRepository.deleteById(todoDTO.getTno());  }    // paging 정보 pageable 클래스 통해서 요청(페이지, 한페이지 사이즈, 정렬기준) 등록하고  //findAll(pageable) => 페이징기법 끝난다. Page<Todo> 출력이 된다. 이속에서 페이징에 관련된 모든정보를 가지고 있다.  //PageRequestDTO (멤버변수: page, size) 외부에서 페이징요청처리받을 때 정보를 받는것이다.  @Override  //2. pageRequestDTO  public PageResponseDTO<TodoDTO> list(PageRequestDTO pageRequestDTO) {  //페이지정보 PageRequestDTO, 목록리스트, 엔티티 전체개수 PageResponseDTO<TodoDTO>  Pageable pageable = PageRequest.of(pageRequestDTO.getPage() - 1,  pageRequestDTO.getSize(), Sort.by("tno").descending());    Page<Todo> result = todoRepository.findAll(pageable);    //1. 1페이지 10개 (10개의 레코드가 들어있다.)  List<TodoDTO> dtoList = result.getContent().stream().map(todo ->  modelMapper.map(todo, TodoDTO.class)).collect(Collectors.toList());    //3. 전체갯수  long totalCount = result.getTotalElements();  //생성자(List<TodoDTO>, PageRequestDTO, totalCount)  PageResponseDTO<TodoDTO> responseDTO = PageResponseDTO.<TodoDTO>withAll()  .dtoList(dtoList).pageRequestDTO(pageRequestDTO).totalCount(totalCount).build();    return responseDTO;  }  } |

4.8 TodoController.class

|  |
| --- |
| package com.kh.mallapi.controller;  import java.util.Map;  import org.springframework.web.bind.annotation.CrossOrigin;  import org.springframework.web.bind.annotation.DeleteMapping;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.PutMapping;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import com.kh.mallapi.dto.PageRequestDTO;  import com.kh.mallapi.dto.PageResponseDTO;  import com.kh.mallapi.dto.TodoDTO;  import com.kh.mallapi.service.TodoService;  import lombok.RequiredArgsConstructor;  import lombok.extern.log4j.Log4j2;  @RestController  @RequiredArgsConstructor  @Log4j2  @RequestMapping("/api/todo")  public class TodoController {  private final TodoService service;  //http://localhost:8080/api/todo/200 get select => Todo  @GetMapping("/{tno}")  public TodoDTO get(@PathVariable(name = "tno") Long tno) {  TodoDTO todoDTO = TodoDTO.builder().tno(tno).build();  return service.get(todoDTO);  }    //http://localhost:8080/api/todo/list get 페이징요청/ 페이지, 사이즈  @GetMapping("/list")  public PageResponseDTO<TodoDTO> list(PageRequestDTO pageRequestDTO) {  log.info(pageRequestDTO);  return service.list(pageRequestDTO);  }  //http://localhost:8080/api/todo/ post 방식 insert  @PostMapping("/")  public Map<String, Long> register(@RequestBody TodoDTO todoDTO) {  log.info("TodoDTO: " + todoDTO);  Long tno = service.register(todoDTO);  return Map.of("TNO", tno);  }  //http://localhost:8080/api/todo/100 post 방식 update  @PutMapping("/{tno}")  public Map<String, String> modify(@PathVariable(name = "tno") Long tno, @RequestBody TodoDTO todoDTO) {  todoDTO.setTno(tno);  log.info("Modify: " + todoDTO);  service.modify(todoDTO);  return Map.of("RESULT", "SUCCESS");  }    //http://localhost:8080/api/todo/100 get 방식 delete  @DeleteMapping("/{tno}")  public Map<String, String> remove(@PathVariable(name = "tno") Long tno) {  TodoDTO todoDTO = TodoDTO.builder().tno(tno).build();  log.info("Remove: " + tno);  service.remove(todoDTO);  return Map.of("RESULT", "SUCCESS");  }  } |

5. 테스팅방법

* 1. Jpa Domain, DTO, Repository 만들고나서 브라우저 요청이 들어 왔을 때 컨트롤러 에서 진행하는 c, u, r, d 테스팅에 결과를 확인해야한다.
  2. 테스팅이 끝나면, 포스트맨에 가서 외부에 프로그램으로 테스팅했던 방식으로 점검한다.