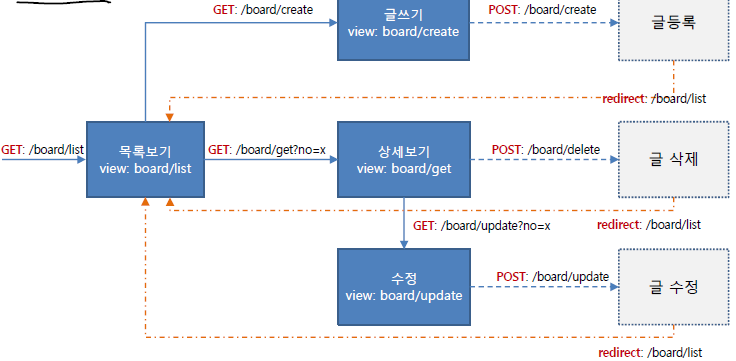
각 단계마다 테스트 코드를 짜서 테스트하는 것이 좋음. 해당 내용엔 테스트 관련 내용은 뺐음

1. 프로젝트 만들기
   1. 만든 spring 템플릿 사용.
   2. Settings.gradle의 rootProject.name=’현재프로젝트이름’ 수정
   3. 톰캣 없다면, edit configuration에서 톰캣 서버 추가 및 배포 아티팩트 추가 및 애플리케이션 컨텍스트 수정.
2. 영속 계층
   1. Data확보하기.
      1. 데이터베이스 확보하기 없으면 만들어
      2. 데이터베이스 접속을 위한 계정 및 데이터베이스 정보 application.properties 지정
   2. 데이터베이스와 연동하여 쿼리의 결과를 받을 도메인 VO타입 만들기// DAO
      1. 롬복 어노테이션을 활용. Setter와 생성자가 있어야 한다.
      2. VO타입의 필드명과 연동할 테이블의 컬럼명과 일치하면 좋지만 규칙적으로 변환이 필요할때가 있다. 해당 규칙을 mybatis-config.xml설정 파일에 등록해야 한다.  
         또한 커스텀 도메인 VO타입인 경우 후에 xml에 타입 지정시 경로를 포함한 풀네임을 줘야하는데, 이를 간편하게 해줄 type aliases 설정도 여기서 가능하다.
   3. SQL MAPPER, MYBATIS 활용을 위한, sql쿼리 결과를 VO객체에 넣기 위한  
      MAPPER 인터페이스와 그와 연동할 MAPPER XML 작성
      1. Mapper 인터페이스에 서비스(BL) 파일에서 사용할 mapper의 메소드를 선언한다.
      2. Mapper 인터페이스에서 선언한 메소드와 연동할 sql설정 및 내용을   
         해당 mapper 인터페이스를 타겟으로 하는 mapper xml을 작성. 각 메소드에 맞는 sql설정 가능.
      3. service파일에서 mybatis가 Mapper 인터페이스를 내부적으로 구현한 객체를 직접 사용하여 활용할 예정. 예) Mapper.getList( );
3. 비즈니스(service) 계층
   1. DB에서 데이터를 갖고 오고 받을 때를 제외한 모든 영역에서 사용되는 객체, DTO 타입 정의
      1. DTO는 VO와 비슷하지만 테이블 정보 말고도 BL에 필요한 추가 정보를 포함할 수 있다.
      2. DTO도 원할한 사용을 위해서, DI를 위해서, setter와 생성자가 있어야 한다.  
         롬복 어노테이션을 이용하여 정의하자
   2. VO와 DTO간의 변환이 빈번할 테니 서로 간의 변환 기능을 구현해야 한다.
      1. Vo->dto : vo를 가지고 빌더로 dto를 ‘생성’
      2. Dto->vo : dto를 가지고 빌더로 vo를 ‘생성’
      3. Vo와 dto를 분리하는 것이 좋다.
   3. Dto와 Mapper를 활용하여 실질적인 business로직을 수행하는 서비스 타입 인터페이스들을 정의하자.
      1. 이렇게 선언된 메서드들은 맞는 controller에서 호출되어 BL를 수행할 것이다.
      2. 이렇게 선언된 메서드들은 controller에서 데이터(dto포함)를 인자로 받고 데이터(dto포함)를 반환할 것이다.
   4. Dto와 Mapper를 활용하여 실질적인 business로직을 수행하는 서비스 타입 인터페이스들을 구현한 서비스 타입 클래스를 정의
      1. 활용할 mapper객체를 DI요청하기 위해 final 필드 멤버로 선언하고 final 필드 멤버를 초기화해주는 생성자를 만들어주는 롬복어노테이션 클래스에 적용.
      2. @Component의 자식이며 서비스라는 것을 명시해주는 @Service 클래스에 적용. Rootconfig에 등록하면, 스캔하여 컨텍스트에 빈 등록.
      3. 구현해야 할 메소드를 얻은 mapper객체를 활용하며 하나하나 구현.
4. 프레젠테이션(웹)(controllers) 계층
   1. 먼저 특정 url ( 예)/board )밑으로 오는 요청들이 뭐가 올지 예상, 분석.  
      /board/list, /board/create, /board/get, /board/update …
   2. 먼저 특정 url ( 예)/board )밑으로 오는 요청 흐름을 파악, 설계.  
      
   3. 설계한 것을 바탕으로 먼저 특정 url ( 예)/board )밑으로 오는 모든 요청들을 받고 처리할 controller 정의
      1. 컨트롤러들을 관리하는 servletconfig(front controller)(dispatch servlet)에 스캔 받아 컨텍스트에 빈 등록될 수 있게, @Controller을 컨트롤러 클래스에 적용.
      2. Handler mapping되게 클래스 레벨에 @RequsestMapping(“/board”) 적용.
      3. Rootconfig(root context)에 등록된 애들은 servletconfig에 등록된 애들이 사용해도 된다했죠?  
         rootconfig에 등록된 service를 사용하기 위해 필드멤버로 선언 후 DI받게끔 @RequiredArgConstructor 클래스에 적용
      4. Handler mapping되게 설계에서 본 세부 요청들을 메서드가 맡게 하기 위한   
         @GetMapping, @PostMapping … 적용
      5. 메소드 작성시 데이터 입력을 받을 매개변수를 지정. 혹은 포워딩이나 리다이렉션을 위한 데이터를 저장하기 위한 매개변수를 지정. Handler adapter기능이 동작함.  
         DTO객체, @RequestParam, @ModelAttribute, Model객체, RedirectAttirbutes객체
      6. 메소드 작성시 결과의 종류를 결정하는 반환형 지정.   
         String, void, DTO or VO, List< >, Model, ModelAndView, ResponseEntity, HttpHeader등이 있음.
      7. 메소드 본문에서는 매개변수와 클래스 필드인 service를 활용하여 원하는 BL 수행을 명령받고  
         결과를 받아서 처리.
   4. 컨트롤러에서 결과를 받아 출력시킬 view들 만들기.  
      🡺 03->6.1 화면처리