JPA Querydsl

JPA

❖ Querydsl - http://www.querydsl.com/

✓ @Query annotation을 사용했을 때 의 단점
- @Query annotation 안에 JPQL 문법으로 문자열을 입력하기 때문에 잘못 입력하면 컴파일 시점에 에러를 발견할 수 없음
- 동적인 쿼리를 생성하는 것이 어려움

✓ Querydsl
- JPQL을 코드로 작성할 수 있도록 도와주는 빌더 API
- Querydsl은 SQL 구문을 문자열이 아닌 코드로 작성하기 때문에 컴파일러의 도움을 받을 수 있어서 소스 작성 시 오타가 발생하면 개발자에게 오타가 있음을 바로 알려줌
- JPQL은 문자를 계속 더해야 하기 때문에 동적 쿼리 작성이 어려음

✓ Querydsl 장점
- 조건에 맞게 동적으로 쿼리를 생성할 수 있음
- 비슷한 쿼리를 재사용할 수 있으며 제약 조건 조립 및 가독성을 향상시킬 수 있음
- 문자열이 아닌 자바 소스 코드로 작성하기 때문에 컴파일 시점에 오류를 발견할 수 있음
- IDE의 도움을 받아서 자동 완성 기능을 이용할 수 있기 때문에 생산성을 향상시킬 수 있음

JPA

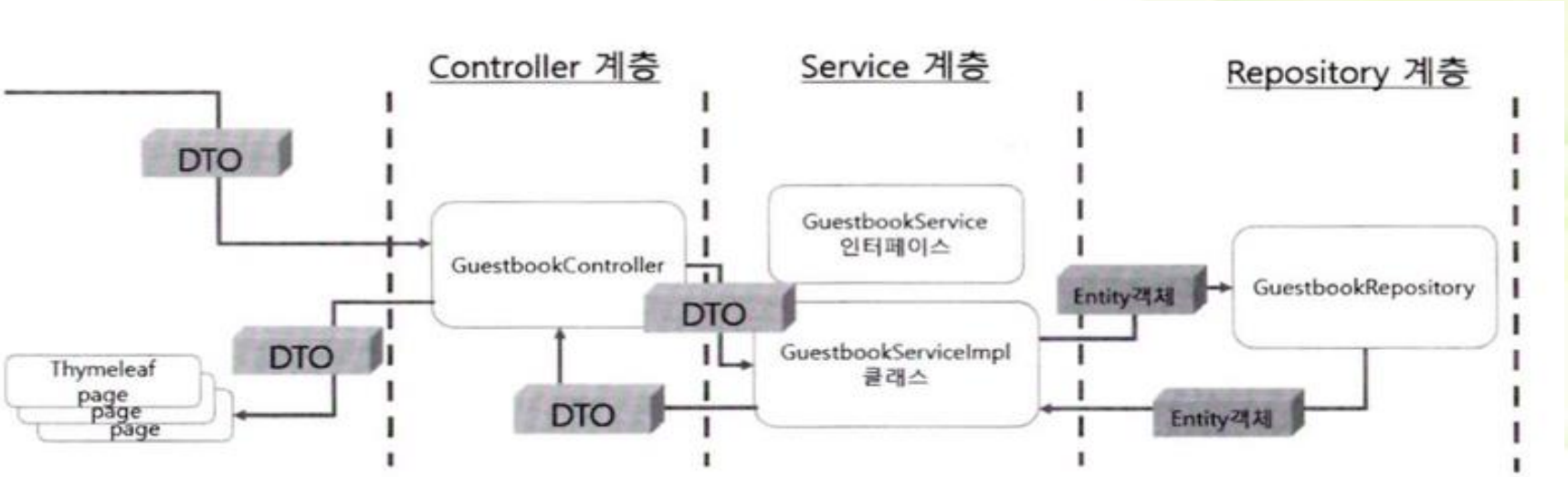
- MappedSuperclass
  - ✓ 데이터의 등록 시간 과 수정 시간은 자동으로 추가되어야 하고 변경되어야 하는 컬럼
  - ✓ 여러 테이블에서 자동으로 추가되어야 하고 변경되어야 하는 컬럼을 매번 처리하는 것은 번거롭기 때문에 Annotation을 이용해서 자동으로 처리할 수 있도록 해주면 편리
  - ✓ @MappedSuperclass: 테이블로 생성하지 않는 Entity 클래스 생성

JPA

- JPA Auditing
  - ✓ Entity 객체에는 어떤 변화가 일어나는 것을 감지하는 리스너(listener)가 있음
  - ✓ @EntityListeners(value = { AuditingEntityListener.class }): Entity 객체가 생성되고 변경되는 것을 감지하는 리스너 역할을 수행하는 클래스를 설정하는데 이 설정을 사용하기 위해서는 SpringBootApplication 클래스에 @EnableJpaAuditing 이 추가되어야 함

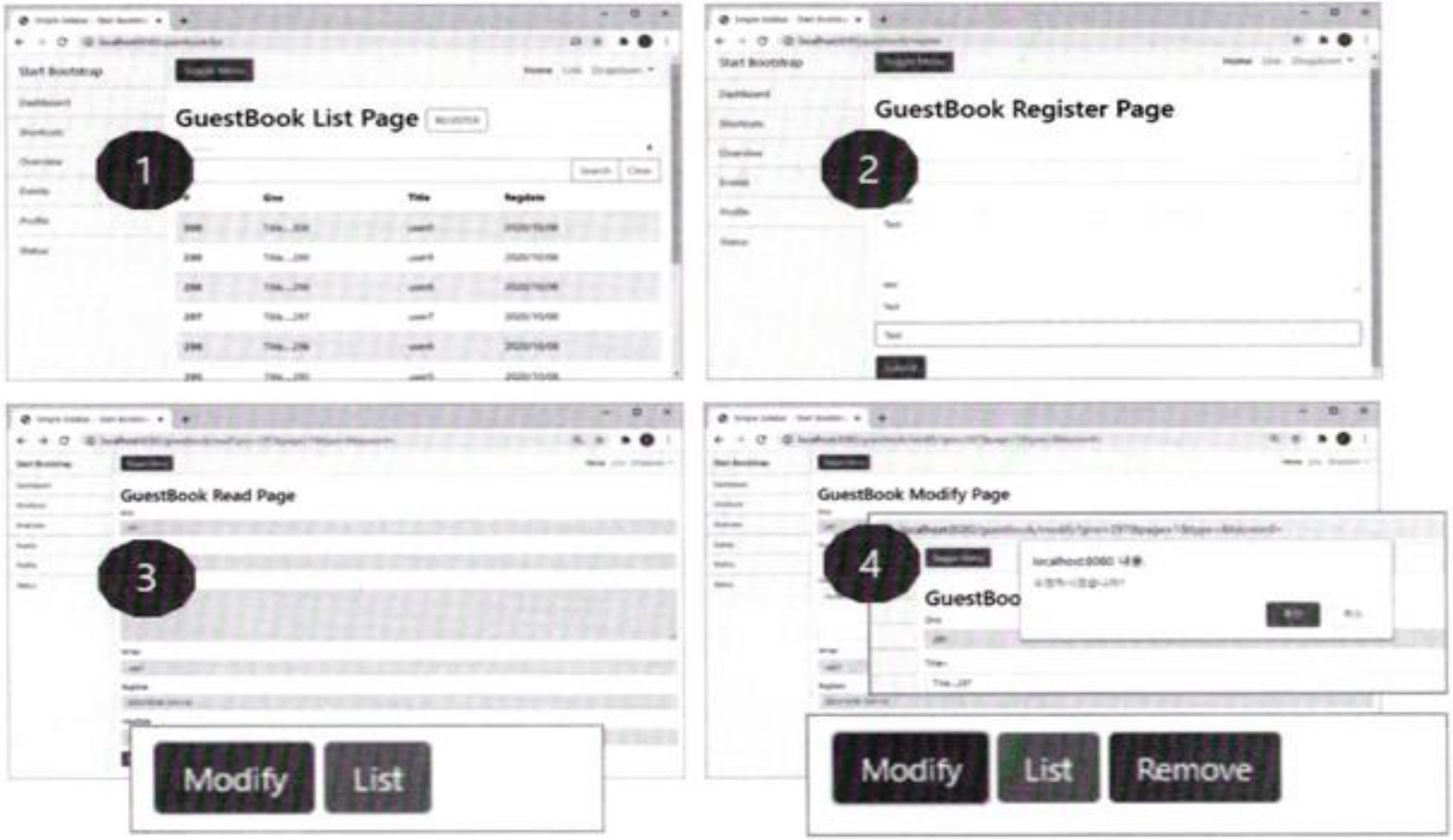
JPA

- DTO 와 Entity
  - ✓ 브라우저에서 전달되는 Request는 Controller에서 DTO의 형태로 처리하는데 데이터를 주고받을 때는 Entity 클래스 자체를 사용하면 안되고 데이터 전달용 객체를 생성해서 사용해야 하는데 데이터베이스의 설계를 외부에 노출할 필요도 없으며 요청과 응답 객체가 항상 Entity 와 일치하지 않기 때문
  - ✓ Repository는 Entity 타입을 이용하므로 중간에 Service 계층에서는 DTO와 Entity의 변환을 수행해야 함
  - ✓ JPA를 이용하는 경우 Entity 객체는 항상 JPA가 관리하는 콘텍스트(context)에 속해 있기 때문에 가능하면 JPA 영역을 벗어나지 않도록 작성하는 방식을 권장



WireFrame

❖ 화면 구성



WireFrame

- 화면 구성
  - ✓ 목록 화면(번호 1) - 전체 목록을 페이징 처리해서 조회할 수 있고 제목/내용/작성자 항목으로 검색 과 페이징 처리 가능
  - ✓ 등록 화면(번호 2) - 새로운 글을 등록할 수 있고 등록 처리 후 다시 목록 화면으로 이동 가능
  - ✓ 조회 화면(번호 3) - 목록 화면에서 특정한 글을 선택하면 자동으로 상세 보기로 이동하고 상세 보기 화면에서는 수정/삭제가 가능한 화면(번호 4)으로 버튼을 클릭해서 이동 가능
  - ✓ 수정/삭제 화면(번호 4) - 수정 화면에서 삭제가 가능하고 삭제 후에는 목록 페이지로 이동가능하고 글을 수정하는 경우에는 다시 상세 보기 화면(번호 3)으로 이동해서 수정된 내용을 확인할 수 있음

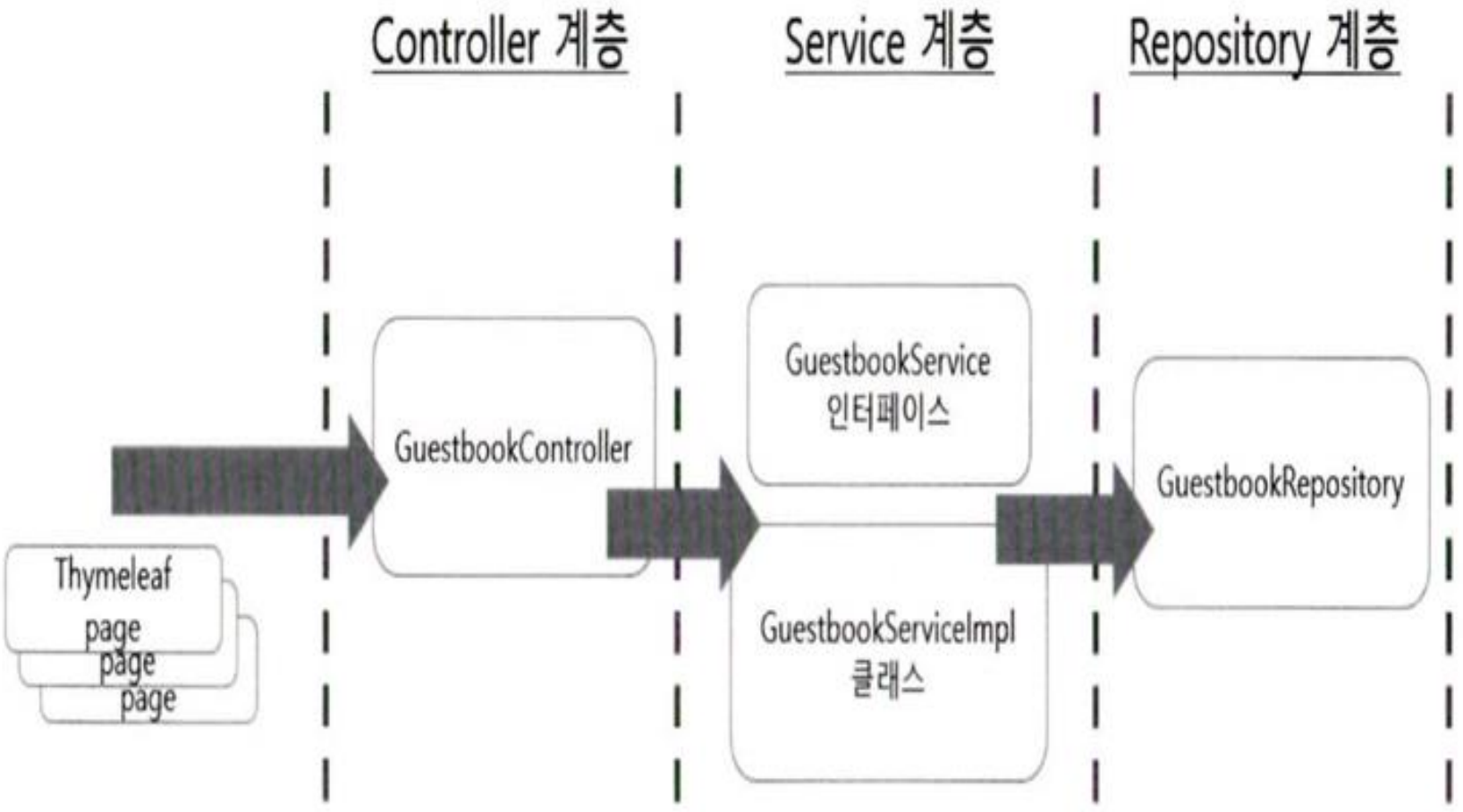
WireFrame

❖ 화면 구성

기능	URL	GET/POST	기능	Redirect URL
목록	/guestbook/list	GET	목록/페이징/검색	
등록	/guestbook/register	GET	입력 화면	
	/guestbook/register	POST	등록 처리	/guestbook/list
조회	/guestbook/read	GET	조회 화면	
수정	/guestbook/modify	GET	수정/삭제 가능 화면	
	/guestbook/modify	POST	수정 처리	/guestbook/read
삭제	/guestbook/remove	POST	삭제 처리	/guestbook/list

WireFrame

❖ 프로젝트 기본 구조



WireFrame

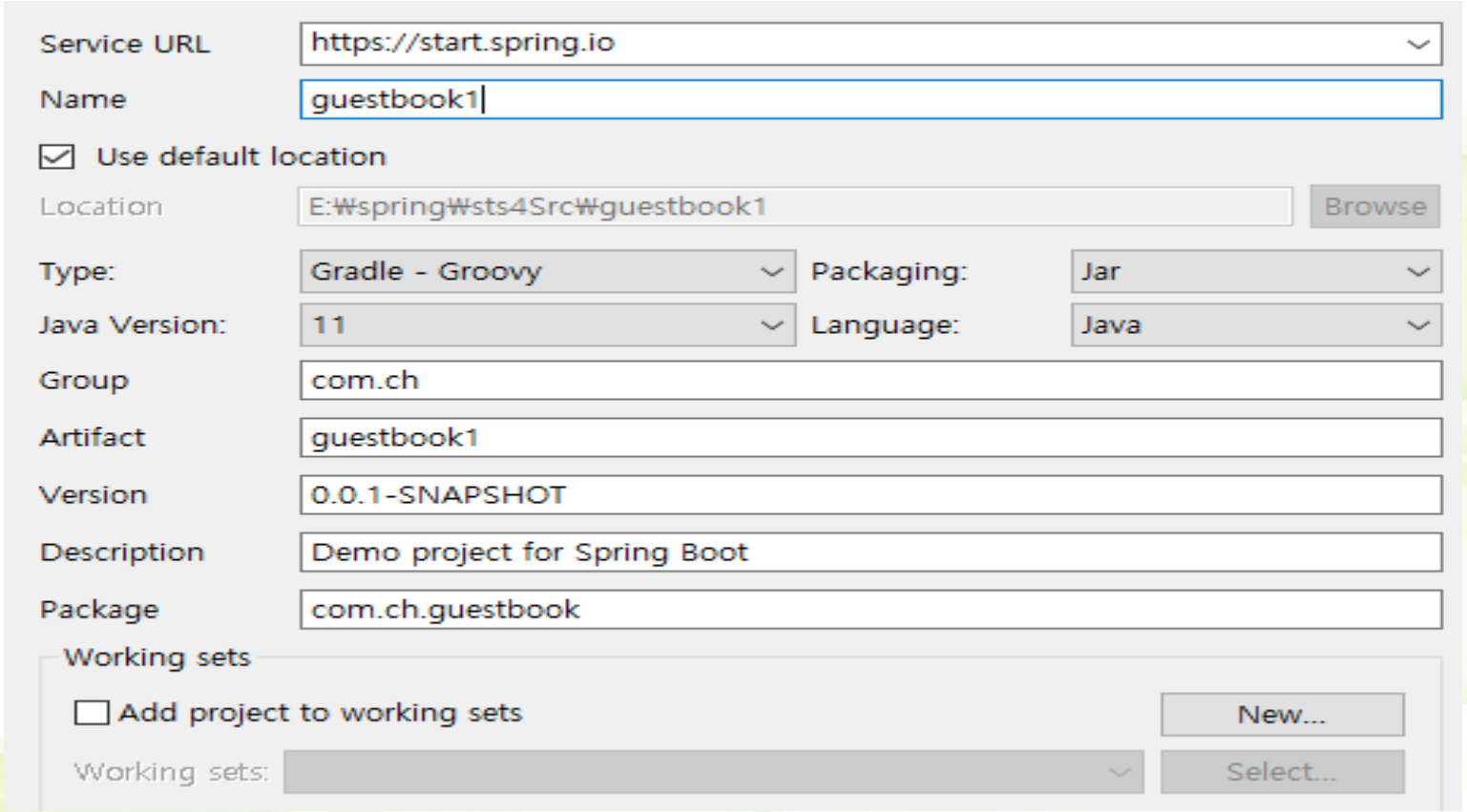
- 프로젝트 기본 구조
  - ✓ 브라우저에서 들어오는 Request는 GuestbookController라는 객체로 처리
  - ✓ GuestbookController는 GuestbookService 타입을 주입받는 구조로 만들고 이를 이용해서 원하는 작업을 처리
  - ✓ GuestbookRepository는 Spring Data JRA를 이용해서 구성하고 GuestbookServiceImpl 클래스에 주입해서 사용
  - ✓ 결과는 Thymeleaf를 이용해서 레이아웃 템플릿을 활용해서 처리

프로젝트 기본 설정

❖ 프로젝트 생성

✓ 프로젝트 옵션

- 프로젝트 이름: guestbook
- 빌드 도구: Gradle
- Packaging: Jar

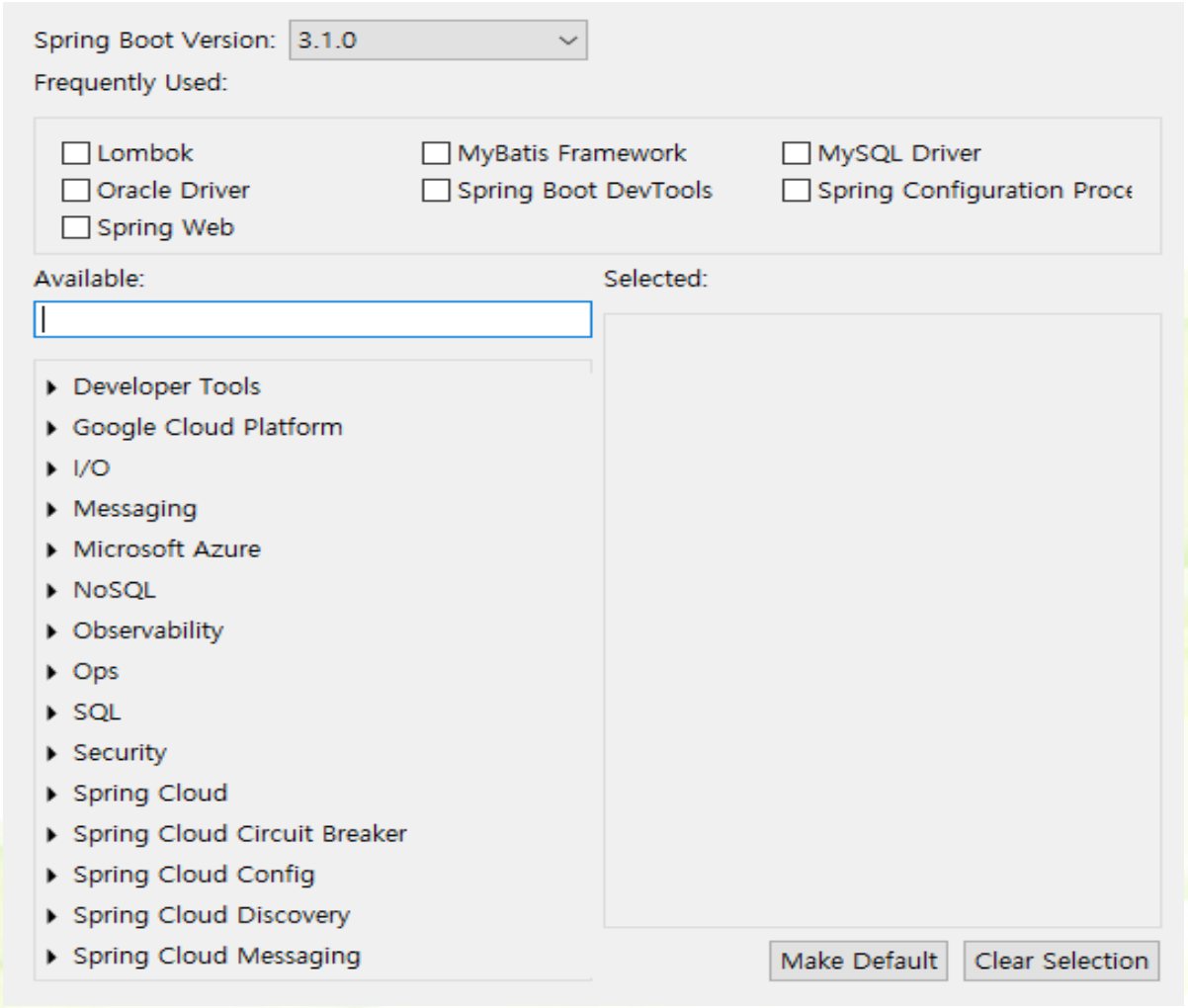


프로젝트 기본 설정

❖ 프로젝트 생성

✓ 의존성

- Spring Boot Dev Tools
- Lombok
- Spring Web
- Thymeleaf
- Spring Data JPA
- MySQL Driver



프로젝트 기본 설정

❖ 프로젝트 기본 설정

✓ buidl.gradle 에 java8 날짜 관련 라이브러리 추가

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'mysql:mysql-connector-java'
    annotationProcessor 'org.projectlombok:lombok'
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    implementation group: 'org.thymeleaf.extras', name: 'thymeleaf-extras-java8time'
}
```

프로젝트 기본 설정

- 프로젝트 기본 설정
  - ✓ application.properties 파일에 설정을 추가

```
#MySQL
server.port=80
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test?useUnicode=yes&c
    haracterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=mysql

spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
logging.level.org.hibernate.type.descriptor.sql=trace
#Live Reload
spring.devtools.livereload.enabled=true
spring.thymeleaf.cache=false

# spring.jpa.properties.hibernate.format_sql=true # sql모양 보기 좋게
```

프로젝트 기본 설정

❖ Controller 생성

✓ 기본 패키지.controller 패키지에 GuestBookController 클래스를 생성하고 작성

```
@Log4j2
@Controller
@RequiredArgsConstructor // 자동 주입을 위한 Annotation
public class GuestBookController {
    private final GuestBookService guestBookService;
    @GetMapping("/")
    public String index() {
            return "redirect:/guestbook/list";
    }
    @GetMapping("/guestbook/list")
    public void list(PageRequestDTO pageRequestDTO, Model model) {
            model.addAttribute("result", guestBookService.getList(pageRequestDTO));
    }
```

templates/guestbook 디렉토리에 header.html 파일을 생성하고 작성

```
<header>
<link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
<script type="text/javascript" th:src="@{/js/jquery.js}"></script>
<script type="text/javascript" th:src="@{/js/bootstrap.min.js}"></script>
<style type="text/css">
.err { color: red; font-weight: bold; }
</style></header>
```

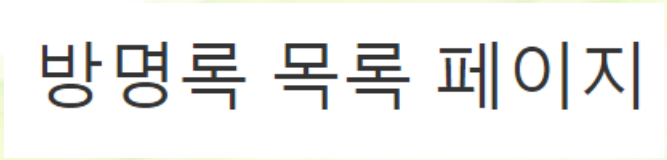
프로젝트 기본 설정

❖ 기본 화면 생성

✓ templates/guestbook 디렉토리에 list.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
        <th:block th:insert="/guestbook/header.html::header"></th:block>
<head><meta charset="UTF-8">
        <title>Insert title here</title></head>
<body></body>
    <div class="container" align="center">
        <h1 class="mt-4">방명록 목록 페이지</h1>
    </div>
</body>
</html>
```

✓프로젝트 실행



프로젝트 기본 설정

- 자동으로 처리되는 날짜/시간을 위한 Entity
  - ✓ SpringBootApplication 클래스에 Annotation 추가

```
@EnableJpaAuditing
@SpringBootApplication
public class GuestbookApplication {

    public static void main(String[] args) {
        SpringApplication.run(GuestbookApplication.class, args);
    }

}
```

프로젝트 기본 설정

- 자동으로 처리되는 날짜/시간을 위한 Entity
  - ✓ 기본 패키지에 entity 패키지를 생성하고 공통 속성을 정의

```
@MappedSuperclass
@EntityListeners(value = { AuditingEntityListener.class })
@Getter
abstract class BaseEntity {

    @CreatedDate
    @Column(name = "regdate", updatable = false)
    private LocalDateTime regDate;

    @LastModifiedDate
    @Column(name ="moddate" )
    private LocalDateTime modDate;

}
```

DAO

❖ 방명록을 위한 Entity 생성 – BaseEntity를 상속받은 GuestBook

```
import lombok.*;

import javax.persistence.*;

@Entity
@Getter
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class GuestBook extends BaseEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer  gno;
```

DAO

❖ 방명록을 위한 Entity 생성 – BaseEntity를 상속받은 GuestBook

```
    @Column(length = 100, nullable = false)
    private String title;

    @Column(length = 1500, nullable = false)
    private String content;

    @Column(length = 50, nullable = false)
    private String writer;

    public void changeTitle(String title){
        this.title = title;
    }

    public void changeContent(String content){
        this.content = content;
    }
}
```

❖ 프로젝트를 실행해서 테이블이 만들어지는지 확인

```
Use test;
Select * from guest_book;
```

DAO

- GuestBook 테이블 처리를 위한 Repository 클래스를 생성 – JpaRepository 클래스를 상속받는 클래스를 persistence 패키지를 생성해서 그 안에 GuestBookRepository 로 생성

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface GuestBookRepository extends JpaRepository<GuestBook,
    Integer> {
}
```

DAO

❖ Querydsl

✓ Sprint Boot 2.6.x 프로젝트의 경우 build.gradle 수정

```
buildscript {
    ext {
        queryDslVersion = "5.0.0"
    }
}

plugins {
    id 'org.springframework.boot' version '2.6.3'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
    id "com.ewerk.gradle.plugins.querydsl" version "1.0.10"
}

group = 'com.adamsoft'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'
```

DAO

- Querydsl
  - ✓ Sprint Boot 2.6.x 프로젝트의 경우 build.gradle 수정

```
configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}
```

DAO

- Querydsl
  - ✓ Sprint Boot 2.6.x 프로젝트의 경우 build.gradle 수정

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    # runtimeOnly 'com.oracle.database.jdbc:ojdbc8'
    runtimeOnly 'mysql:mysql-connector-java'
    annotationProcessor 'org.projectlombok:lombok'
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'

    implementation group: 'org.thymeleaf.extras', name: 'thymeleaf-extras-
    java8time '

    // QueryDSL
    implementation "com.querydsl:querydsl-jpa:${queryDslVersion}"
    implementation "com.querydsl:querydsl-apt:${queryDslVersion}"
}
```

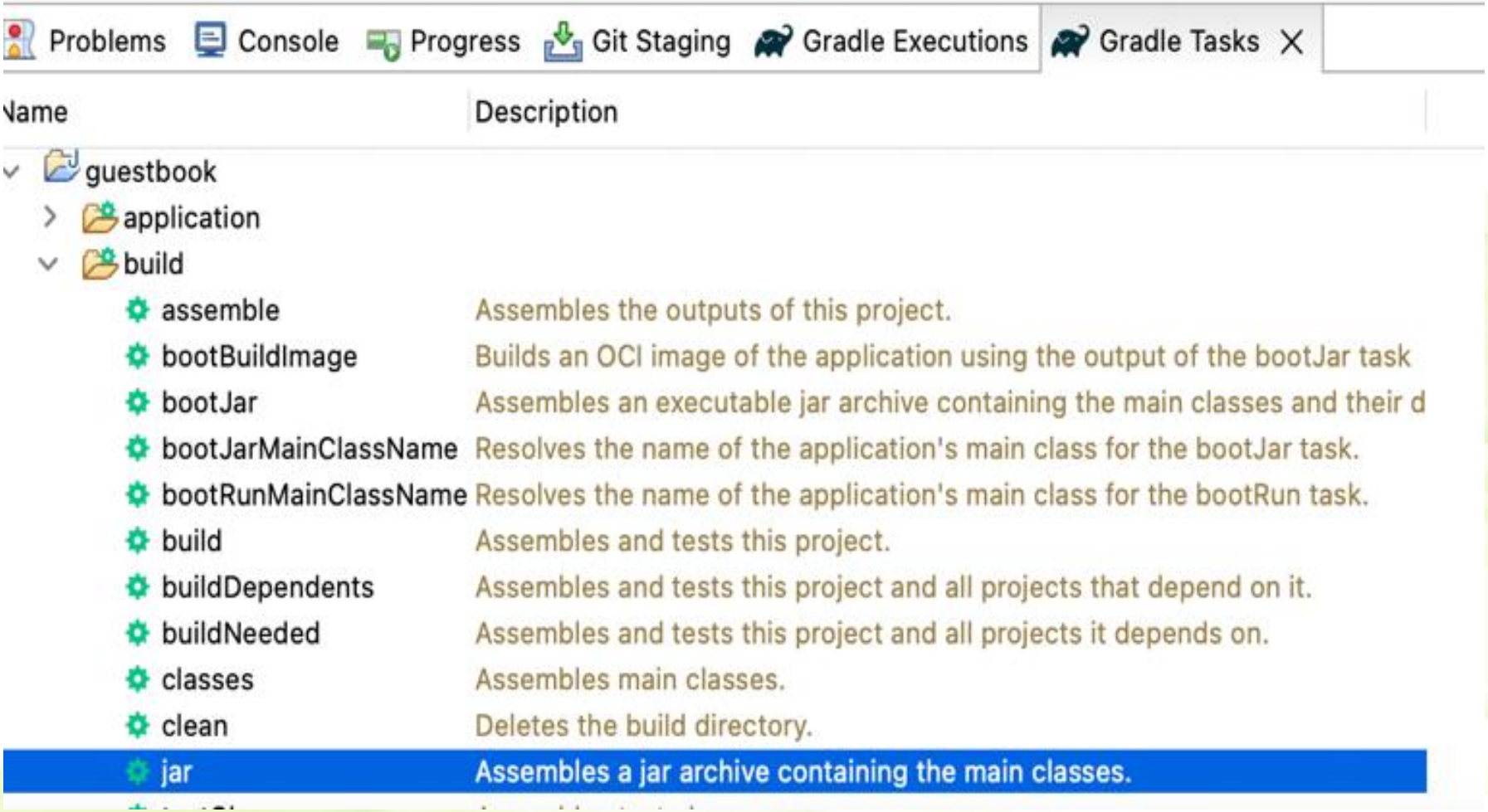
DAO

- Querydsl
  - ✓ Sprint Boot 2.6.x 프로젝트의 경우 build.gradle 수정

```
def querydslDir = "$buildDir/generated/querydsl"
querydsl {
    jpa = true
    querydslSourcesDir = querydslDir
}
sourceSets {
    main.java.srcDir querydslDir
}
configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
    querydsl.extendsFrom compileClasspath
}
compileQuerydsl {
    options.annotationProcessorPath = configurations.querydsl
}
tasks.named('test') {
    useJUnitPlatform()
}
```

DAO

- Querydsl
  - ✓ 프로젝트를 선택하고 마우스 오른쪽을 클릭하고 [Gradle] – [Refresh Gradle Project]
  - ✓ Gradle Tasks 탭에서 [build]에서 build 와 jar를 더블 클릭



DAO

- Querydsl
  - ✓ Repository 인터페이스의 선언 부분 수정 – QuerydslPredicateExecute 인터페이스를 상속받도록 수정

```
public interface GuestBookRepository extends JpaRepository<GuestBook,
    Integer>, QuerydslPredicateExecutor<GuestBook> {

}
```

DAO

- Entity 테스트
  - ✓ 데이터 삽입 테스트
    - Test 디렉토리의 기본 패키지에 테스트를 위한 클래스를 추가 – GuestBookRepositoryTests

DAO

- Entity 테스트
  - ✓ 데이터 삽입 테스트
    - GuestBookRepositoryTests 클래스에 데이터를 300개 삽입하는 메서드를 작성하고 실행

```
@SpringBootTest
public class GuestBookRepositoryTests {
    @Autowired
    private GuestBookRepository guestBookRepository;

    @Test
    public void insertDummies(){
        IntStream.rangeClosed(1,300).forEach(i -> {
            GuestBook guestbook = GuestBook.builder()
                    .title("Title...." + i)
                    .content("Content__ " +i)
                    .writer("user" + (i % 10))
                    .build();
            System.out.println(guestBookRepository.save(guestbook));
        });
    }
}
```

DAO

- Entity 테스트
  - ✓ 데이터 삽입 테스트
    - 데이터베이스에서 확인

```
select *
from guest_book;
```

	gno	moddate	regdate	content	title	writer
1	1	42:57.40540200	2022-01-11 18:42:57.405402000	Content__ 1	Title....1	us은r1
2	2	42:57.46985700	2022-01-11 18:42:57.469857000	Content__ 2	Title....2	us은r2
3	3	:42:57.47551300	2022-01-11 18:42:57.475513000	Content__ 3	Title....3	us은r3
4	4	:42:57.48115500	2022-01-11 18:42:57.481155000	Content__ 4	Title....4	us은r4
5	5	:42:57.48591000	2022-01-11 18:42:57.485910000	Content__ 5	Title....5	us은r5
6	6	:42:57.49031000	2022-01-11 18:42:57.490310000	Content__ 6	Title....6	us은r6
7	7	42:57.49489500	2022-01-11 18:42:57.494895000	Content__ 7	Title....7	us은r7
8	8	:42:57.49877100	2022-01-11 18:42:57.498771000	Content__ 8	Title....8	us은r8
9	9	42:57.50300500	2022-01-11 18:42:57.503005000	Content__ 9	Title....9	us은r9
10	10	42:57.50660900	2022-01-11 18:42:57.506609000	Content__ 10	Title....10	us은r0
11	11	:42:57.51096600	2022-01-11 18:42:57.510966000	Content__ 11	Title....11	us은r1
12	12	:42:57.51480200	2022-01-11 18:42:57.514802000	Content__ 12	Title....12	us은r2
13	13	:42:57.51867200	2022-01-11 18:42:57.518672000	Content__ 13	Title....13	us은r3
14	14	42:57.52224800	2022-01-11 18:42:57.522248000	Content__ 14	Title....14	us은r4
15	15	:42:57.52579500	2022-01-11 18:42:57.525795000	Content__ 15	Title....15	us은r5
16	16	42:57.53056800	2022-01-11 18:42:57.530568000	Content__ 16	Title....16	us은r6
17	17	:42:57.53407100	2022-01-11 18:42:57.534071000	Content__ 17	Title....17	us은r7

DAO

- Entity 테스트
  - ✓ 데이터 수정 테스트
    - GuestBookRepositoryTests 클래스에 데이터를 수정하는 메서드를 작성하고 실행

```
@Test
public void updateTest() {
        Optional<GuestBook> result =
    guestBookRepository.findById(300);
        //존재하는 번호로 테스트
        if(result.isPresent()){
            GuestBook guestBook = result.get();
            guestBook.changeTitle("Changed Title....");
            guestBook.changeContent("Changed Content....");
            guestBookRepository.save(guestBook);
        }
}
```

DAO

- Entity 테스트
  - ✓ 데이터 삽입 테스트
    - 데이터베이스에서 확인

```
select *
from guest_book
where gno = 300;
```

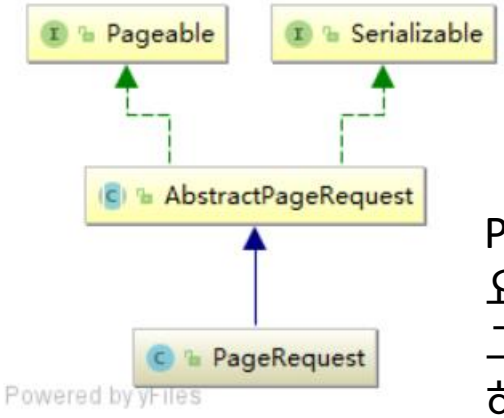
	gno	moddate	regdate	content	title
1	300	2022-01-11 18:57:53.017328000	2022-01-11 18:47:55.483285000	Changed Content....	Changed Title....

DAO

❖ Querydsl 테스트

✓ 사용법

- BooleanBuilder를 생성
- 조건에 맞는 구문은 Querydsl에서 사용하는 Predicate 타입의 함수를 생성
- BooleanBuilder에 작성된 Predicate를 추가하고 실행



PageRequest의 생성에는 페이지와 한 페이지의 크기가 중요합니다.
그리고 나중에 페이징을 하는 경우에, Sort를 생성자로 추가해서 PageRequest를 생성할 수 있다.

```
PageRequest pageRequest =
PageRequest.of(page, size, Sort.by("createdAt").descending());
```

DAO

- Querydsl 테스트
  - ✓ title에 1이라는 글자가 들어있는 Entity 조회 – 테스트 메서드를 생성한 후 확인

```
@Test
public void testQuery1() {
        Pageable pageable = PageRequest.of(0, 10,
    Sort.by("gno").descending());
        QGuestBook qGuestBook = QGuestBook.guestBook; //I
        String keyword = "1";
        BooleanBuilder builder = new BooleanBuilder(); //2
        BooleanExpression expression = qGuestBook.title.contains(keyword);
    //3
        builder.and(expression); //4
        Page<GuestBook> result = guestBookRepository.findAll(builder,
    pageable); //5
        result.stream().forEach(guestbook -> {
            System.out.println(guestbook);
        });
}
```

DAO

- Querydsl 테스트
  - ✓ title에 1이라는 글자가 들어있는 Entity 조회 – 테스트 메서드를 생성한 후 확인
    1. 동적으로 처리하기 위해서 QDomain 클래스를 얻어오는데 QDomain 클래스를 이용하면 Entity 클래스에 선언된 title, content 같은 필드들을 변수로 활용할 수 있음
    2. BooleanBuilder는 where문에 들어가는 조건들을 넣어주는 컨테이너
    3. 원하는 조건은 필드 값과 같이 결합해서 생성하는데 BooleanBuilder 안에 들어가는 값은 com.querydsl.core.types.Predicate 타입 (Java에 있는 Predicate 타입이 아니므로 주의)
    4. 만들어진 조건은 where문에 and나 or같은 키워드와 결합
    5. BooleanBuilder는 GuestbookRepository에 추가된 QuerydslPredicateExcutor 인터페이스의 findAll()을 사용해서 데이터 가져오기

DAO

- Querydsl 테스트
  - ✓ 다중 항목 검색 – title 또는 content 에 특정한 키워드가 있고 gno 가 0보다 큰 데이터 검색

```
@Test
public void testQuery2() {
        Pageable pageable = PageRequest.of(0, 10,
    Sort.by("gno").descending());
        QGuestBook qGuestBook = QGuestBook.guestBook;
        String keyword = "1";
        BooleanBuilder builder = new BooleanBuilder();
        BooleanExpression exTitle = qGuestBook.title.contains(keyword);
        BooleanExpression exContent =
    qGuestBook.content.contains(keyword);
        BooleanExpression exAll = exTitle.or(exContent); // 1---------------
        builder.and(exAll); //2-------
        builder.and(qGuestBook.gno.gt(0)); // 3----------------
        Page<GuestBook> result = guestBookRepository.findAll(builder,
    pageable);
        result.stream().forEach(guestbook -> { System.out.println(guestbook);
        });
}
```

DAO

- Querydsl 테스트
  - ✓ 다중 항목 검색 – title 또는 content 에 특정한 키워드가 있고 gno 가 0보다 큰 데이터 검색
    - 코드의 작성 과정은 이전과 유사한데 중간에 exTitle과 exContent라는 Boolean Expression을 결합하는 부분(1)과 이를 BooleanBuilder에 추가하고(2) 이후에 gno가 0 보다 크다 라는 조건을 추가한(3) 부분이 차이

Service Layer & DTO

- DTO(Data Transfer Object – Variable Object)
  - ✓ Entity 객체를 영속 계층 바깥쪽에서 사용하는 방식 보다는 DTO(Data Transfer Object)를 이용하는 방식을 권장
  - ✓ DTO는 Entity 객체와 달리 각 계층끼리 주고받는 우편물이나 상자의 개념으로 순수하게 데이터를 담고 있다는 점에서는 Entity 객체와 유사하지만 목적 자체가 데이터의 전달이므로 읽고, 쓰는 것이 모두 허용되는 점이 가능하고 일회성으로 사용되는 성격이 강함
  - ✓ JPA를 이용하게 되면 Entity 객체는 단순히 데이터를 담는 객체가 아니라 데이터베이스와 관련이 있고 내부적으로 Entity 매니저(entity manager)가 관리하는 객체
  - ✓ DTO가 일회성으로 데이터를 주고받는 용도로 사용되는 것과 달리 생명 주기(life cycle)도 전혀 다르기 때문에 분리해서 처리하는 것을 권장
  - ✓ 웹 애플리케이션을 제작할 때 HttpServletRequest나 HttpServletResponse를 서비스 계층으로 전달하지 않는 것을 원칙으로 하는데 유사하게 Entity 객체가 JPA에서 사용하는 객체이므로 JPA 외에서 사용하지 않는 것을 권장
  - ✓ 서비스 계층에서는 DTO로 Parameter 와 리턴 타입을 처리하도록 구성하는데 DTO를 사용하면 Entity 객체의 범위를 한정 지을 수 있기 때문에 좀 더 안전한 코드를 작성할 수 있고 화면과 데이터를 분리하려는 취지에도 좀 더 부합함
  - ✓ DTO를 사용하는 경우 가장 큰 단점은 Entity와 유사한 코드를 중복으로 개발한다는 것과 Entity 객체를 DTO로 변환하거나 반대로 DTO 객체를 Entity 객체로 변환하는 과정이 필요하다는 것

Service Layer & DTO

- DTO(Data Transfer Object – Variable Object)
  - ✓ 기본패키지에 domain 패키지를 생성하고 GuestBookDTO 클래스를 생성

```
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Builder
@NoArgsConstructor
@AllArgsConstructor
@Data
public class GuestBookDTO {

    private Integer gno;
    private String title;
    private String content;
    private String writer;
    private LocalDateTime regDate, modDate;
}
```

Service Layer & DTO

- Service Layer
  - ✓ 기본패키지에 service 패키지를 생성
  - ✓ service 패키지에 GuestBookService 인터페이스 생성

```
public interface GuestBookService {
}
```

  - ✓ GuestBookService 인터페이스에 DTO 클래스의 객체 와 Entity 객체 사이의 변환을 위한 메서드를 구현
    - 서비스 계층에서는 파라미터를 DTO 타입으로 받기 때문에 이를 JPA로 처리하기 위해서는 엔티티 타입의 객체로 변환해야하는 작업이 반드시 필요
    - 이 기능을 DTO 클래스에 직접 작성하거나 ModelMapper 라이브러리(http://modelmapper.org/)나 MapStruct(https://mapstruct.org/) 등을 이용하기도 함

Service Layer & DTO

- Service Layer
  - ✓ GuestBookService 인터페이스에 DTO 클래스의 객체 와 Entity 객체 사이의 변환을 위한 메서드를 구현

```
default GuestBook dtoToEntity(GuestBookDTO dto) {
        GuestBook entity = GuestBook.builder()
                .gno(dto.getGno())
                .title(dto.getTitle())
                .content(dto.getContent())
                .writer(dto.getWriter())
                .build();
        return entity;
}
default GuestBookDTO entityToDto(GuestBook entity){
        GuestBookDTO dto  = GuestBookDTO.builder()
                .gno(entity.getGno())
                .title(entity.getTitle())
                .content(entity.getContent())
                .writer(entity.getWriter())
                .regDate(entity.getRegDate())
                .modDate(entity.getModDate())
                .build();
        return dto;
}
```

Service Layer & DTO

- Service Layer
  - ✓ service 패키지에 GuestBookService 인터페이스를 implements 한 GuestBookServiceImpl 클래스를 생성

```
import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Service;
import lombok.RequiredArgsConstructor;

@Service
@Log4j2
@RequiredArgsConstructor //의존성 자동 주입
public class GuestBookServiceImpl implements GuestBookService{
    private final GuestBookRepository repository;
}
```

Service Layer & DTO

❖ 데이터 삽입

✓ GusetBookService 인터페이스에 새로운 GuestBook을 추가하는 메서드를 선언

```
public Integer register(GuestBookDTO dto);
```

✓ GusetBookServiceImpl 클래스에 새로운 GuestBook을 추가하는 메서드를 구현

```
@Override
public Integer register(GuestBookDTO dto) {
        log.info("DTO--------------------------------------");
        log.info(dto);
        GuestBook entity = dtoToEntity(dto);
        log.info(entity);
        repository.save(entity);
        return entity.getGno();
}
```

Service Layer & DTO

- 데이터 삽입
  - ✓ 테스트 클래스에 새로운 GuestBook을 추가하는 메서드를 테스트

```
@Autowired
private GuestBookService gusetBookService;

@Test
public void testRegister() {
        GuestBookDTO guestbookDTO = GuestBookDTO.builder()
                .title("Sample Title...")
                .content("Sample Content...")
                .writer("userO")
                .build();
        System.out.println(gusetBookService.register(guestbookDTO));
}
```

Service Layer & DTO

❖ 데이터 목록 보기

✓ 고려 사항

- 화면에서 필요한 목록 데이터에 대한 DTO 생성
- DTO를 Pageable 타입으로 전환
- Page<Entity>를 화면에서 사용하기 쉬운 DTO의 리스트 등으로 변환
- 화면에 필요한 페이지 번호 처리

Service Layer & DTO

❖ 데이터 목록 보기

✓ 출력 결과

Service Layer & DTO

- 데이터 목록 보기
  - ✓ 목록 처리를 위한 DTO
    - 목록을 처리하는 작업은 거의 모든 게시판 관련 기능에서 사용될 가능성이 높기 때문에 재사용이 가능한 구조를 생성하는 것이 효율적
    - 모든 목록을 처리하는 기능에는 페이지 번호나 한 페이지당 몇 개나 출력될 것인가 와 같은 공통적인 부분이 많기 때문에 이를 클래스로 만들어두면 여러 곳에서 사용 가능
    - 용도 별로 다르게 생성하는 것이 재사용 측면에서는 유리

Service Layer & DTO

- 데이터 목록 보기
  - ✓ 목록 처리를 위한 DTO
    - ● 페이지 요청 처리 DTO – PageRequestDTO

```
@Builder
@AllArgsConstructor
@Data
public class PageRequestDTO {
    private int page;
    private int size;

    public PageRequestDTO(){
        this.page = 1;
        this.size = 10;
    }

    public Pageable getPageable(Sort sort){
        return PageRequest.of(page -1, size, sort);
    }
}
```

Service Layer & DTO

- 데이터 목록 보기
  - ✓ 목록 처리를 위한 DTO
    - ● 페이지 요청 처리 DTO – PageRequestDTO
      - ◆ 목록 화면에서는 페이지 처리를 하는 경우가 많기 때문에 페이지 번호나 페이지 내 목록의 개수, 검색 조건 들이 많이 사용됨
      - ◆ PageRequestDTO는 이러한 Parameter를 DTO로 선언하고 나중에 재사용하는 용도로 생성
      - ◆ 페이지 번호 등은 기본값을 가지는 것이 좋기 때문에 1과 10이라는 값을 생성자에서 기본값으로 설정
      - ◆ JPA를 이용하는 경우에는 페이지 번호가 0 부터 시작한다는 점을 감안해서 1 페이지의 경우 0이 될 수 있도록 page-1을 하는 형태로 작성하고 정렬은 나중에 다양한 상황에서 쓰기 위해서 별도의 Parameter로 받도록 설계

Service Layer & DTO

- 데이터 목록 보기
  - ✓ 목록 처리를 위한 DTO
    - 페이지 결과 처리 DTO – PageReponseDTO

```
import lombok.Data;
import org.springframework.data.domain.Page;
import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;

@Data
public class PageResponseDTO <DTO, EN>{
    //DTO리스트
    private List<DTO> dtoList;

    //Page를 fn 함수를 적용해서 List로 변환해주는 메서드
    public PageResponseDTO(Page<EN> result, Function<EN,DTO>
    fn ){
        dtoList = result.stream().map(fn).collect(Collectors.toList());
    }
}
```

Service Layer & DTO

- ❖ 데이터 목록 보기
  - ✓ 목록 처리를 위한 DTO
    - ● 페이지 결과 처리 DTO – PageReponseDTO
      - ◆ PageResponseDTO 클래스는 다양한 곳에서 사용할 수 있도록 제네릭 타입을 이용해서 DTO와 EN이라는 타입을 지정
      - ◆ PageResponseDTO는 Page(Entity) 타입을 이용해서 생성할 수 있도록 생성자로 작성하는데 특별한 Function<EN , DTO>는 Entity 객체들을 DTO로 변환해 주는 기능
      - ◆ 위와 같은 구조를 이용하면 나중에 어떤 종류의 Page<E>타입이 생성되더라도, PageResponseDTO를 이용해서 처리할 수 있다는 장점이 있습니다. 처음에는 조금 복잡해 보일 수 있지만, 프로젝트를 진행하는 과정에서는 다양한 종류의 엔티티를 다루기 때문에 위와 같은 제네릭 방식으로 적용해 두면 나중에 추가적인 클래스를 작성하지 않고도 목록 데이터를 처리할 수 있음

Service Layer & DTO

- 데이터 목록 보기
  - ✓ GuestBookService 인터페이스에 목록 보기를 위한 메서드를 선언

```
//데이터 목록을 가져오는 메서드
PageResponseDTO<GuestBookDTO, GuestBook>
    getList(PageRequestDTO requestDTO);
```

  - ✓ GuestBookServiceImpl 클래스에 목록 보기를 위한 메서드를 구현

```
//데이터 목록을 가져오는 메서드
@Override
public PageResponseDTO<GuestBookDTO, GuestBook>
    getList(PageRequestDTO requestDTO) {
        Pageable pageable =
    requestDTO.getPageable(Sort.by("gno").descending());
        Page<GuestBook> result = repository.findAll(pageable);
        Function<GuestBook, GuestBookDTO> fn = (entity ->
    entityToDto(entity));
        return new PageResponseDTO<>(result, fn );
}
```

Service Layer & DTO

❖ 데이터 목록 보기

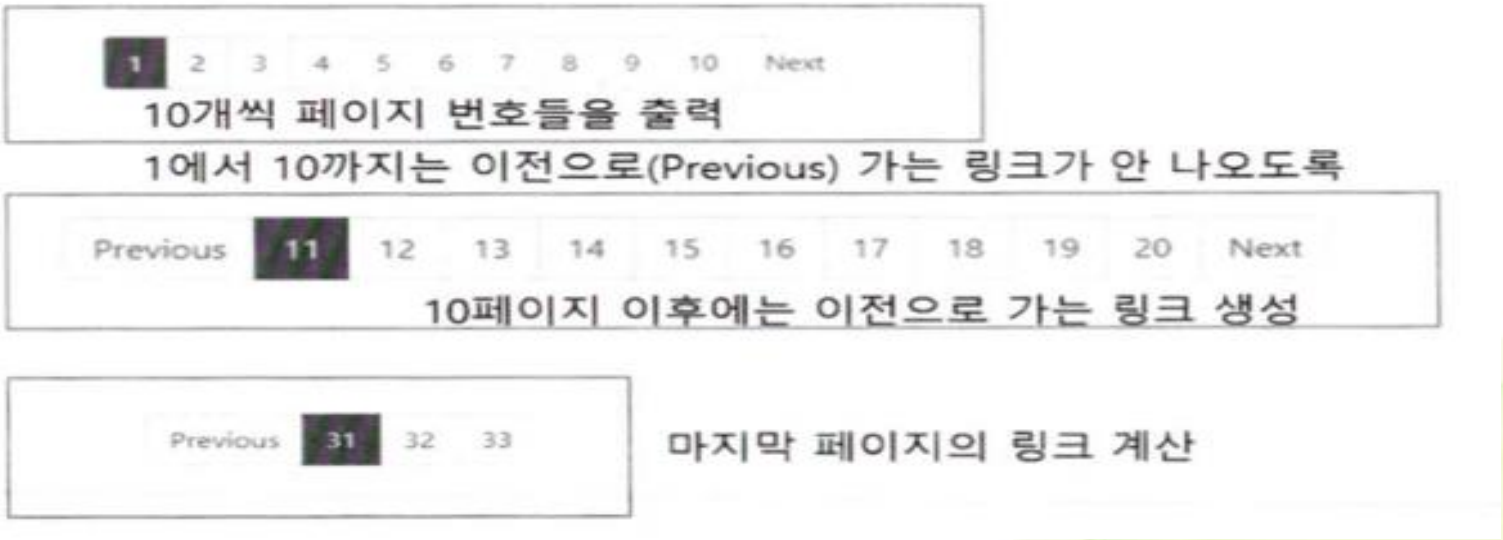
✓ 테스트 메서드를 작성하고 테스트

```
@Test
public void testList(){
        PageRequestDTO pageRequestDTO =
    PageRequestDTO.builder().page(1).size(10).build();
        PageResponseDTO<GuestBookDTO, GuestBook> resultDTO =
    gusetBookService. getList(pageRequestDTO);
        for (GuestBookDTO guestbookDTO : resultDTO.getDtoList()) {
            System.out.println(guestbookDTO);
        }
}
```

Service Layer & DTO

❖ 데이터 목록 보기

✓ 목록 데이터 페이지 처리



- 페이징 처리 구성 요소
  - ◆ 화면에서 시작 페이지 번호(start)
  - ◆ 화면에서 끝 페이지 번호(end)
  - ◆ 이전/다음 이동 링크 여부(prev, next)
  - ◆ 현재 페이지 번호(page)

Service Layer & DTO

- 데이터 목록 보기
  - ✓ 목록 데이터 페이지 처리
    - 끝 페이지 번호 중간 계산
      - 마지막페이지번호 = (int)(Math.ceil(현재페이지번호 / (double)페이지번호개수)) * 페이지번호개수;
    - 시작 페이지 번호 계산
      - 시작페이지번호 = 마지막페이지번호 – (페이지번호개수 – 1);
    - 끝 페이지 번호 계산
      - 전체 페이지 개수를 구한 후 전체 페이지 번호 개수보다 끝 페이지 번호가 크다면 전체 페이지 개수로 변경하고 그렇지 않으면 중간 계산된 페이지 번호를 사용
    - 이전 링크는 시작 페이지 번호가 1보다 크다면 생성
    - 다음 링크는 전체 페이지 개수가 끝 페이지 번호보다 크다면 생성

Service Layer & DTO

❖ 데이터 목록 보기

✓ PageResponseDTO 클래스 수정

```
@Data
public class PageResponseDTO <DTO, EN>{
    //DTO리스트
    private List<DTO> dtoList;

    //총 페이지 번호
    private int totalPage;
    //현재 페이지 번호
    private int page;
    //목록 사이즈
    private int size;

    //시작 페이지 번호, 끝 페이지 번호
    private int start, end;

    //이전, 다음
    private boolean prev, next;

    //페이지 번호  목록
    private List<Integer> pageList;
```

Service Layer & DTO

❖ 데이터 목록 보기

✓ PageResponseDTO 클래스 수정

```
private void makePageList(Pageable pageable){

    this.page = pageable.getPageNumber() + 1; // 0부터 시작하므로
1을 추가
    this.size = pageable.getPageSize();

    //temp end page
    int tempEnd = (int)(Math.ceil(page/10.0)) * 10;
    start = tempEnd - 9;
    prev = start > 1;
    end = totalPage > tempEnd ? tempEnd: totalPage;
    next = totalPage > tempEnd;
    pageList = IntStream.rangeClosed(start,
end).boxed().collect(Collectors.toList());

}
```

Service Layer & DTO

❖ 데이터 목록 보기

✓ PageResponseDTO 클래스 수정

```
        //Page를 fn 함수를 적용해서 List로 변환해주는 메서드
        public PageResponseDTO(Page<EN> result, Function<EN,DTO> fn ){
            dtoList = result.stream().map(fn).collect(Collectors.toList());
            totalPage = result.getTotalPages();
            makePageList(result.getPageable());
        }

    }
```

Service Layer & DTO

- 데이터 목록 보기
  - ✓ 테스트 메서드를 생성해서 테스트

```
@Test
public void testListInformation(){
        PageRequestDTO pageRequestDTO =
    PageRequestDTO.builder().page(1).size(10).build();
        PageResponseDTO<GuestBookDTO, GuestBook> resultDTO =
    gusetBookService. getList(pageRequestDTO);

        System.out.println("PREV: "+resultDTO.isPrev());
        System.out.println("NEXT: "+resultDTO.isNext());
        System.out.println("TOTAL: " + resultDTO.getTotalPage());

        System. out. println ("-----------------------------------------------------");
        for (GuestBookDTO guestBookDTO : resultDTO.getDtoList()) {
            System.out.println(guestBookDTO);
        }

    System.out.println("==================================");

        resultDTO.getPageList().forEach(i -> System.out.println(i));
}
```

Controller & View

❖ 데이터 목록 보기

방명록 목록 페이지

#	Title	Writer	Regdate
301	Sample Title...	userO	2022/01/12
300	Changed Title....	user0	2022/01/11
299	Title....299	user9	2022/01/11
298	Title....298	user8	2022/01/11
297	Title....297	user7	2022/01/11
296	Title....296	user6	2022/01/11
295	Title....295	user5	2022/01/11
294	Title....294	user4	2022/01/11
293	Title....293	user3	2022/01/11
292	Title....292	user2	2022/01/11

Controller & View

❖ 데이터 목록 보기

✓ GuestBookController 클래스 수정

```
@Log4j2
@Controller
@RequiredArgsConstructor //자동 주입을 위한 Annotation
public class GuestBookController {
    private final GuestBookService guestBookService;

    @GetMapping("/")
    public String index() {
        return "redirect:/guestbook/list";
    }

    @GetMapping("/guestbook/list")
    public void list(PageRequestDTO pageRequestDTO, Model model){
        log.info("list.....");
        model.addAttribute("result", guestBookService.getList(pageRequestDTO));
    }

}
```

Controller & View

- 데이터 목록 보기
  - ✓ list.html 파일 수정

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:insert="/guestbook/header.html::header"></th:block>
<head><meta charset="UTF-8">
<title>Insert title here</title></head>
<body>
<div class="container" align="center">
   <h1 class="text-primary">방명록 목록 페이지</h1>
   <table class="table table-striped">
         <thead>
               <tr>     <th scope="col">#</th>
                        <th scope="col">Title</th>
                        <th scope="col">Writer</th>
                        <th scope="col">Regdate</th>
               </tr>
         </thead>
```

Controller & View

- 데이터 목록 보기
  - ✓ list.html 파일 수정

```
<tbody>
    <tr th:each="dto : ${result.dtoList}">
            <th scope="row">[[${dto.gno}]]</th>
            <td>
            <!-- [[${dto.title}]] -->
            <!-- <a th:href="@{/guestbook/read(gno =
    ${dto.gno},page=${result.page})}">[[${dto.title}]]</a> -->
            <a th:href="@{/guestbook/read(gno = ${dto.gno},
    page=${result.page},type=${pageRequestDTO.type},keyword=${pageRe
    questDTO.keyword})}">
                    [[${dto.title}]]</a>
            </td>
            <td>[[${dto.writer}]]</td>
            <td>[[${#temporals.format(dto.regDate, 'yyyy/MM/dd')}]]</td>
    </tr>
    </tbody>
</table>
```

Controller & View

❖ 페이지 번호 처리

방명록 목록 페이지

#	Title	Writer	Regdate
211	Title....211	user1	2022/01/11
210	Title....210	user0	2022/01/11
209	Title....209	user9	2022/01/11
208	Title....208	user8	2022/01/11
207	Title....207	user7	2022/01/11
206	Title....206	user6	2022/01/11
205	Title....205	user5	2022/01/11
204	Title....204	user4	2022/01/11
203	Title....203	user3	2022/01/11
202	Title....202	user2	2022/01/11

1 2 3 4 5 6 7 8 9 10 다음

Controller & View

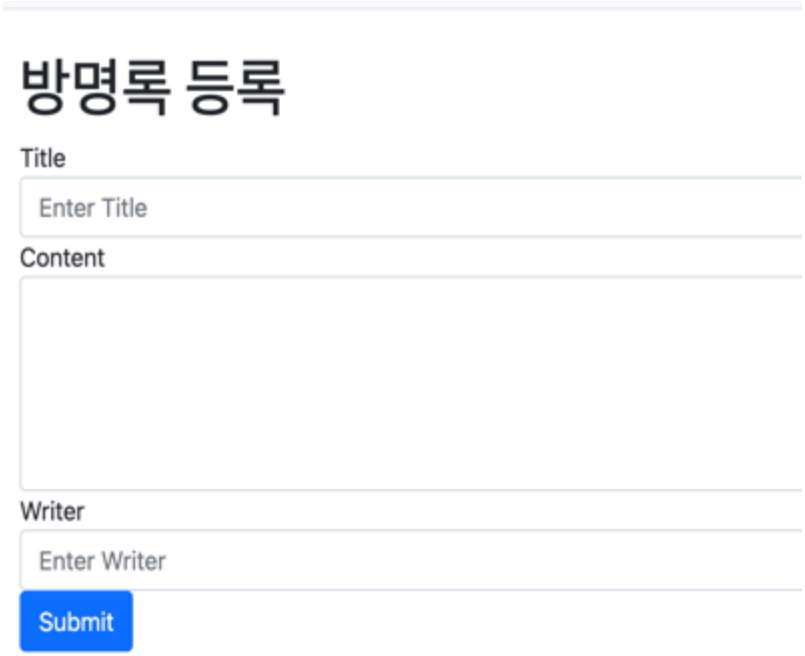
❖ 페이지 번호 처리

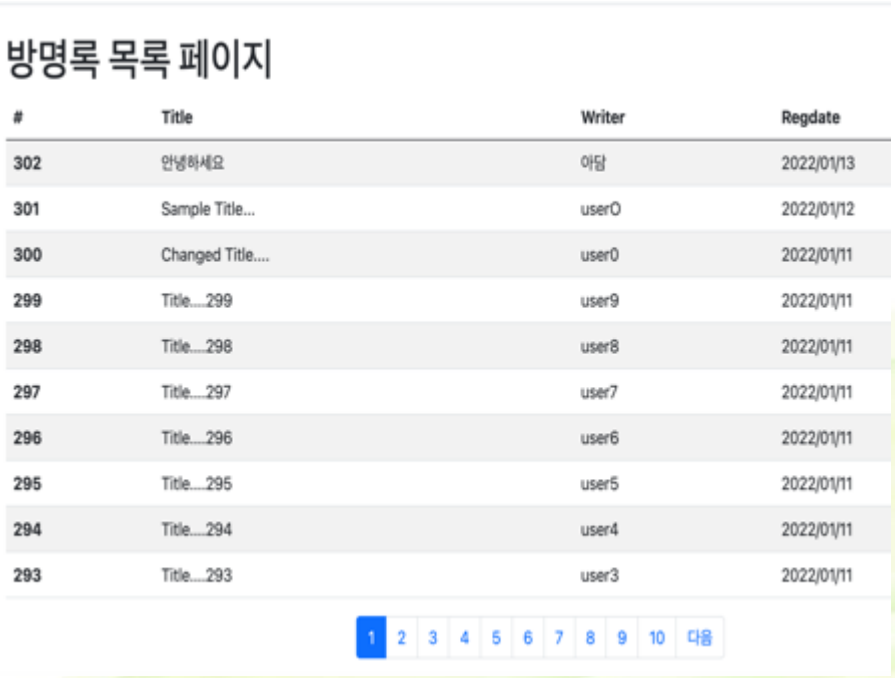
✓ list.html 파일의 위에 추가

```
<ul class="pagination">
        <!-- th:href="@{/guestbook/list(page= ${result.start -1})}" -->
        <li th:if="${result.prev}">
                <a th:href="@{/guestbook/list(page= ${result.start -1},
type=${pageRequestDTO.type}, keyword=${pageRequestDTO.keyword})}"
                tabindex="-1">이전</a>
        </li>
        <!-- th:href="@{/guestbook/list(page = ${page})}" -->
        <li th:class="${result.page == page?'active':''} "
        th:each="page: ${result.pageList}">
        <a th:href="@{/guestbook/list(page = ${page}, type=${pageRequestDTO.type},
keyword=${pageRequestDTO.keyword})}">
                [[${page}]] </a></li>
        <!-- th:href="@{/guestbook/list(page= ${result.end + 1})} -->
        <li th:if="${result.next}"><a th:href="@{/guestbook/list(page= ${result.end + 1},
type=${pageRequestDTO.type}, keyword=${pageRequestDTO.keyword})}">다음</a></li>
</ul>
```

Controller & View

❖GuestBook 등록





Controller & View

❖GuestBook 등록

✓ GuestBookController 클래스에 등록 처리를 위한 메서드 추가

```
//등록을 위한 메서드
@GetMapping("/guestbook/register") public void register() {
        log.info("regiser get... ");
}

@PostMapping( "/guestbook/register")
public String registerPost(GuestBookDTO dto, RedirectAttributes
redirectAttributes){
            log.info("dto..." + dto);
                    //새로 추가된 Entity의 번호
                    Integer gno = guestBookService.register(dto);
                    redirectAttributes.addFlashAttribute("msg", gno + " 작성");
                    return "redirect:/guestbook/list";
}
```

Controller & View

❖GuestBook 등록

✓ guestbook 디렉토리에 register.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:insert="/guestbook/header.html::header"></th:block>
<body><div class="container" align="center">
    <h1 class="text-priamry">방명록 등록</h1>
            <form th:action="@{/guestbook/register}" th:method="post">
               <div class="form-group">
                        <label>Title</label>  <input type="text" class="form-control"
                                                name="title" placeholder="Enter Title">
               </div>
               <div class="form-group">
                        <label>Content</label>
                        <textarea class="form-control" rows="5" name="content"></textarea>
                </div>
                <div class="form-group">
                        <label>Writer</label>  <input type="text" class="form-control"
                                    name="writer" placeholder="Enter Writer">
                 </div>
                        <button type="submit" class="btn btn-primary">Submit</button>
                        </form>
</div></body></html>
```

Controller & View

❖GuestBook 등록

✓ list.html 파일에 방명록 등록 요청을 생성

```
<h1 class="text-primary">방명록 목록 페이지
        <span>
            <a th:href="@{/guestbook/register}">
                <button type="button"
                  class="btn btn-outline-primary">방명록 작성
                </button>
            </a>
        </span>
</h1>
```

❖GuestBook 등록

✓ list.html 파일에 방명록 등록을 한 후 게시물을 보여 줄 때 메시지를 출력하는 코드를 추가

```
<div th:if = "${msg != null}" th:text="${msg}"> </div>
```

상세보기

방명록 상세보기

번호

798

제목

Title....296

콘텐츠

Content__ 296

작가

user6

등록 날짜

2022/11/20 11:12:38

수정 날짜

2022/11/20 11:12:38

수정 목록

상세보기

❖list.html 파일의 글 제목 출력 부분을 수정해서 상세보기 링크를 생성

```
<tr th:each="dto : ${result.dtoList}">
    <th scope="row">[[${dto.gno}]]</th>
    <td>
        <a th:href="@{/guestbook/read(gno = ${dto.gno},
page=${result.page})}">[[${dto.title}]] </a>
    </td>
    <td>[[${dto.writer}]]</td>
    <td>[[${#temporals.format(dto.regDate, 'yyyy/MM/dd')}]]</td>
</tr>
```

상세보기

❖GuestBookService 인터페이스에 상세보기를 위한 메서드 선언

```
GuestBookDTO read(Integer gno);
```

❖GuestBookServiceImpl 클래스에 상세보기를 위한 메서드 구현

```
@Override
public GuestBookDTO read(Integer gno) {
        Optional<GuestBook> guestBook = repository.findById(gno);
        return guestBook.isPresent()? entityToDto(guestBook.get()): null;
}
```

상세보기

❖GuestBookController 클래스에 상세보기 요청을 처리하기 위한 메서드 구현

```
@GetMapping("/guestbook/read")
public void read(long gno, @ModelAttribute("requestDTO") PageRequestDTO
requestDTO, Model model ){
        log.info("gno: " + gno);
        GuestBookDTO dto = guestBookService.read(gno);
        model.addAttribute("dto", dto);
}
```

상세보기

❖guestbook 디렉토리에 read.html 파일을 생성해서 dto를 출력하도록 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:insert="/guestbook/header.html::header"></th:block><body>
<div class="container" align="center">
    <th:block th:fragment="content">
            <h1 class="text-pri,ary">방명록 상세보기</h1>
            <div class="form-group">
                 <label>번호</label>  <input type="text" class="form-control" name="gno"
                        th:value="${dto.gno}" readonly="readonly"></div>
             <div class="form-group">
                <label>Title</label>  <input type="text" class="form-control" name="title"
                        th:value="${dto.title}" readonly></div>
            <div class="form-group"><label>Content</label>
                <textarea class="form-control" rows="5" name="content" readonly>
                        [[${dto.content}]]</textarea></div>
            <div class="form-group"><label>Writer</label>  <input type="text"
                 class="form-control" name="writer" th:value="${dto.writer}" readonly></div>
            <div class="form-group">
                <label>RegDate</label>  <input type="text" class="form-control"
                  name="regDate" th:value="${#temporals.format(dto.regDate,
                        'yyyy/MM/dd HH:mm:ss')}"     readonly></div>
```

상세보기

❖guestbook 디렉토리에 read.html 파일을 생성해서 dto를 출력하도록 작성

```
<div class="form-group">
    <label>ModDate</label>  <input type="text" class="form-control"
      name="modDate"  th:value="${#temporals.format(dto.modDate,
                'yyyy/MM/dd HH:mm:ss')}" readonly>    </div>
<!--  <a th:href="@{/guestbook/list(page=${requestDTO.page})}">
       <button  type="button" class="btn btn-info">목록</button></a>
      <a th:href="@{/guestbook/modify(gno = ${dto.gno}, page=${requestDTO.page})}">
      <button type="button" class="btn btn-primary">수정</button></a>  -->
      <!-- 검색 추가 -->
      <a th:href="@{/guestbook/modify(gno=${dto.gno},page=${requestDTO.page},
                type=${requestDTO.type}, keyword =${requestDTO.keyword})}">
      <button type="button" class="btn btn-primary">수정</button></a>
      <a th:href="@{/guestbook/list(page=${requestDTO.page},type=${requestDTO.type},
     keyword=${requestDTO.keyword})}">
      <button type="button" class="btn btn-info">목록</button></a>
      </th:block>
</div>
</html>
```

수정/삭제

❖수정/삭제

- ✓ Guestbook의 수정(1)은 POST 방식으로 처리하고 다시 수정된 결과를 확인할 수 있는 조회 화면으 로 이동
- ✓ 삭제(2)는 POST 방식으로 처리하고 목록 화면으로 이동
- ✓ 목록을 이동하는 작업은 GET 방식으로 처리하는데 기존에 사용하던 페이지 번호 등을 유지해서 이동

수정

❖read.html 파일에 수정을 위한 링크를 추가

```
<a th:href="@{/guestbook/modify(gno = ${dto.gno},
page=${requestDTO.page})}">
<button type="button" class="btn btn-primary">수정</button></a>
```

❖GuestBookController의 read 요청을 처리하는 메서드를 수정

```
@GetMapping({"/guestbook/read", "/guestbook/modify"})
public void read(long gno, @ModelAttribute("requestDTO") PageRequestDTO
requestDTO, Model model ){
        log.info("gno: " + gno);
        GuestBookDTO dto = service.read(gno);
        model.addAttribute("dto", dto);
}
```

수정

❖수정을 위한 modify.html 파일을 guestbook 디렉토리에 생성하고 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:insert="/guestbook/header.html::header"></th:block>
<head><meta charset="UTF-8">
<title>Insert title here</title></head><body>
<div class="container" align="center">
      <h1 class="text-primary">방명록 수정</h1>
      <form action="/guestbook/modify" method="post">
      <!-- 검색할 때 type과 keyword추가 -->
      <input type="hidden" name="type" th:value="${requestDTO.type}">
   <input type="hidden" name="keyword" th:value="${requestDTO.keyword}">
      <!--페이지 번호 -->
      <input type="hidden" name="page" th:value="${requestDTO.page}">
      <div class="form-group"><label>Gno</label> <input type="text"
      class="form-control" name="gno" th:value="${dto.gno}" readonly></div>
      <div class="form-group"><label>Title</label><input type="text"
      class="form-control" name="title" th:value="${dto.title}"></div>
      <div class="form-group"><label>Content</label><textarea
        class="form-control" rows="5" name="content">[[${dto.content}]]
      </textarea></div>
```

수정

❖수정을 위한 modify.html 파일을 guestbook 디렉토리에 생성하고 작성

```
<div class="form-group"><label>Writer</label> <input type="text" class="form-
control" name="writer" th:value="${dto.writer}" readonly="readonly"></div>
<div class="form-group"><label>RegDate</label> <input type="text" class="form-
control" th:value="${#temporals.format(dto.regDate, 'yyyy/MM/dd HH:mm:ss')}"
        readonly="readonly"></div>
<div class="form-group"><label>ModDate</label> <input type="text"
class="form-control" th:value="${#temporals.format(dto.modDate,
 'yyyy/MM/dd HH:mm:ss')}"          readonly="readonly"></div>
        </form>
<button type="button" class="btn btn-primary modifyBtn">수정</button>
<button type="button" class="btn btn-info listBtn">목록</button>
<button type="button" class="btn btn-danger removeBtn">삭제</button>
<script th:inline="javascript">
        var actionForm = $("form"); //form 태그 객체
        $(".removeBtn").click(function() {
                    actionForm.attr("action", "/guestbook/remove").attr("method", "post");
                    actionForm.submit();              });
        $(".modifyBtn").click(function() {
        if (!confirm("수정하시겠습니까?")) {       return;     }
                actionForm.attr("action", "/guestbook/modify").attr("method", "post").submit();
                                        });
```

수정

❖수정을 위한 modify.html 파일을 guestbook 디렉토리에 생성하고 작성

```
/* 검색 없을 때
$(".listBtn").click(function() {
	//var pageInfo = $("input[name='page']");
	var page = $("input[name='page']");
	actionForm.empty(); //form 태그의 모든 내용을 지우고
	actionForm.append(page);
	actionForm.attr("action", "/guestbook/list").attr("method", "get");
	actionForm.submit();
}); */
```

수정

❖수정을 위한 modify.html 파일을 guestbook 디렉토리에 생성하고 작성

```
/* 검색 추가 했을 때 */
$(".listBtn").click(function() {
        //var pageInfo = $("input[name='page']");
        var page = $("input[name='page']");
        var type = $("input[name='type']");
        var keyword = $("input[name='keyword']");
        actionForm.empty(); //form 태그의 모든 내용을 지우고
        actionForm.append(page);
        actionForm.append(type);
        actionForm.append(keyword);
        actionForm.attr("action", "/guestbook/list").attr("method", "get");
        actionForm.submit();
                                                });
        </script>
        </div></body>
</html>
```

수정

❖GuestBookService 인터페이스에 수정을 위한 메서드를 선언

```
void modify(GuestBookDTO dto);
```

❖GuestBookServiceImpl 클래스에 수정을 위한 메서드를 구현

```
@Override
public void modify(GuestBookDTO dto) {
        //업데이트 하는 항목은 '제목', '내용'
        Optional<GuestBook> result = repository.findById(dto.getGno());
        if(result.isPresent()){
            GuestBook entity = result.get();
            entity.changeTitle(dto.getTitle());
            entity.changeContent(dto.getContent());
            repository.save(entity);
        }
}
```

수정

❖GuestBookController 클래스에 수정 요청을 처리하기 위한 메서드를 구현

```
@PostMapping("/guestbook/modify")
public String modify(GuestBookDTO dto,
                     @ModelAttribute("requestDTO") PageRequestDTO
requestDTO,
                     RedirectAttributes redirectAttributes){

        log.info("post modify.....................................");
        log.info("dto: " + dto);

        guestBookService.modify(dto);

        redirectAttributes.addAttribute("page",requestDTO.getPage());
        redirectAttributes.addAttribute("gno",dto.getGno());

        return "redirect:/guestbook/read";
}
```

삭제

❖GuestBookService 인터페이스에 삭제를 위한 메서드를 선언

```
void remove(Integer gno);
```

❖GuestBookServiceImpl 클래스에 삭제를 위한 메서드를 구현

```
@Override
public void remove(Long gno) {
        repository.deleteById(gno);
}
```

❖GuestBookController 클래스에 삭제 요청을 처리하기 위한 메서드를 구현

```
@PostMapping("/guestbook/remove")
public String remove(long gno, RedirectAttributes redirectAttributes){
        log.info("gno: " + gno);
        guestBookService.remove(gno);
        redirectAttributes.addFlashAttribute("msg", gno + " 삭제");
        return "redirect:/guestbook/list";
}
```

검색

방명록 목록 페이지 방명록 작성

작성자

아담

Search Clear

#	Title	Writer	Regdate
304	jquery 링크 문제	아담	2022/01/13
303	모달 창 출력	아담	2022/01/13
302	안녕하세요	아담	2022/01/13

1

검색

❖PageRequestDTO 에 검색 타입(type) 과 키워드(keyword) 추가

```
private String type;
private String keyword;
```

❖GuestBookServiceImpl 클래스에 검색 조건을 만들어주는 메서드 생성

```
private BooleanBuilder getSearch(PageRequestDTO requestDTO){
        String type = requestDTO.getType();
        BooleanBuilder booleanBuilder = new BooleanBuilder();
        QGuestBook qGuestBook = QGuestBook.guestBook;
        String keyword = requestDTO.getKeyword();

        BooleanExpression expression = qGuestBook.gno.gt(0L); // gno > 0
조건만 생성
        booleanBuilder.and(expression);

        if(type == null || type.trim().length() == 0){ //검색 조건이 없는 경우
            return booleanBuilder;
        }
```

검색

❖GuestBookServiceImpl 클래스에 검색 조건을 만들어주는 메서드 생성

```
        //검색 조건을 작성하기
        BooleanBuilder conditionBuilder = new BooleanBuilder();
        if(type.contains("t")){
            conditionBuilder.or(qGuestBook.title.contains(keyword));
        }
        if(type.contains("c")){
            conditionBuilder.or(qGuestBook.content.contains(keyword));
        }
        if(type.contains("w")){
            conditionBuilder.or(qGuestBook.writer.contains(keyword));
        }

        //모든 조건 통합
        booleanBuilder.and(conditionBuilder);

        return booleanBuilder;
    }
```

검색

❖GuestBookServiceImpl 클래스의 목록을 가져오는 메서드를 수정

```
//데이터 목록을 가져오는 메서드
@Override
public PageResponseDTO<GuestBookDTO, GuestBook>
getList(PageRequestDTO requestDTO) {
    Pageable pageable = requestDTO.getPageable(Sort.by("gno").descending());
    //Page<GuestBook> result = repository.findAll(pageable);
    BooleanBuilder booleanBuilder = getSearch(requestDTO);
    Page<GuestBook> result = repository.findAll(booleanBuilder, pageable);

    Function<GuestBook, GuestBookDTO> fn = (entity -> entityToDto(entity));
    return new PageResponseDTO<>(result, fn );
}
```

검색

❖Test 클래스에 테스트 메서드를 작성해서 확인

```
@Test
public void testListSearch(){
        PageRequestDTO pageRequestDTO =
PageRequestDTO.builder().page(1).size(10).type("tc").keyword("모달").build();
        PageResponseDTO<GuestBookDTO, GuestBook> resultDTO =
gusetBookService. getList(pageRequestDTO);

        System.out.println("PREV: "+resultDTO.isPrev());
        System.out.println("NEXT: "+resultDTO.isNext());
        System.out.println("TOTAL: " + resultDTO.getTotalPage());
        System. out. println ("-----------------------------------------------");
        for (GuestBookDTO guestBookDTO : resultDTO.getDtoList()) {
            System.out.println(guestBookDTO);
        }
System.out.println("==================================");

        resultDTO.getPageList().forEach(i -> System.out.println(i));
}
```

검색

❖list.html 파일에 검색 폼 추가

```
<form action="/guestbook/list" method="get" id="searchForm">
        <div class="input-group">
            <input type="hidden" name="page" value = "1">
            <div class="input-group-prepend">
 <select class="custom-select" name="type">
  <option th:selected="${pageRequestDTO.type == null}">-------</option>
 <option value="t" th:selected="${pageRequestDTO.type =='t'}" >제목</option>
 <option value="c" th:selected="${pageRequestDTO.type =='c'}"  >내용</option>
 <option value="w"  th:selected="${pageRequestDTO.type =='w'}" >작성자</option>
 <option value="tc"  th:selected="${pageRequestDTO.type =='tc'}" >제목 +
내용</option>
  <option value="tcw"  th:selected="${pageRequestDTO.type =='tcw'}" >제목 + 내용 +
작성자</option>
            </select>
  </div>
```

검색

❖list.html 파일에 검색 폼 추가

```
<input class="form-control" name="keyword"
th:value="${pageRequestDTO.keyword}">
<div class="input-group-append" id="button-addon4">
<button class="btn btn-outline-secondary btn-search"
type="button">Search</button>
        <button class="btn btn-outline-secondary btn-clear"
type="button">Clear</button>
                </div>
            </div>
 </form>
```

❖list.html 파일에 스크립트 코드 추가

```
var searchForm = $("#searchForm");
$('.btn-search').click(function(e){
                searchForm.submit();
});
$('.btn-clear').click(function(e){
                searchForm.empty().submit();
});
```

검색

❖list.html 파일의 페이지 링크 수정

```
<ul class="pagination">
<!-- th:href="@{/guestbook/list(page= ${result.start -1})}" -->
      <li th:if="${result.prev}"><a   th:href="@{/guestbook/list(page= ${result.start -1},
                  type=${pageRequestDTO.type},keyword=${pageRequestDTO.keyword})}"
                  tabindex="-1">이전</a></li>
<!-- th:href="@{/guestbook/list(page = ${page})}" -->
      <li th:class="${result.page == page?'active':''} "
         th:each="page: ${result.pageList}">
         <a th:href="@{/guestbook/list(page = ${page},type=${pageRequestDTO.type},
                  keyword=${pageRequestDTO.keyword})}">[[${page}]] </a></li>
<!-- th:href="@{/guestbook/list(page= ${result.end + 1})} -->
      <li th:if="${result.next}"><a th:href="@{/guestbook/list(page= ${result.end + 1},
        type=${pageRequestDTO.type},
keyword=${pageRequestDTO.keyword})}">다음</a></li>
</ul>
```

❖list.html 파일의 상세 보기 링크 수정

```
<a th:href="@{/guestbook/read(gno = ${dto.gno}, page=${result.page} ,
type=${pageRequestDTO.type}, keyword=${pageRequestDTO.keyword})}">
                        [[${dto.title}]]
</a>
```

검색

❖read.html 파일의 수정 과 목록 링크 수정

```
<!--        <a th:href="@{/guestbook/modify(gno = ${dto.gno},
page=${requestDTO.page})}"><button type="button" class="btn btn-
primary">Modify</button></a>-->

<!--        <a
th:href="@{/guestbook/list(page=${requestDTO.page})}"><button
type="button" class="btn btn-info">List</button></a>-->

    <a th:href="@{/guestbook/modify(gno = ${dto.gno},
page=${requestDTO.page}, type=${requestDTO.type}, keyword
=${requestDTO.keyword})}">
      <button type="button" class="btn btn-primary">수정</button>
    </a>

    <a th:href="@{/guestbook/list(page=${requestDTO.page} ,
type=${requestDTO.type}, keyword =${requestDTO.keyword})}">
      <button type="button" class="btn btn-info">목록</button>
    </a>
```

검색

❖modify.html 파일의 form 에 추가

```
<input type="hidden" name="type" th:value="${requestDTO.type}" >
<input type="hidden" name="keyword" th:value="${requestDTO.keyword}" >
```

❖modify.html 파일의 목록 버튼을 누를 때 수행되는 스크립트 코드 수정

```
$(".listBtn").click(function() {
        //var pageInfo = $("input[name='page']");
        var page = $("input[name='page']");
        var type = $("input[name='type']");
        var keyword = $("input[name='keyword']");
        actionForm.empty(); //form 태그의 모든 내용을 지우고

        actionForm.append(page);
        actionForm.append(type);
        actionForm.append(keyword);
        actionForm
            .attr("action", "/guestbook/list")
            .attr("method","get");

        actionForm.submit();
    })
```

검색

❖GuestBookController 클래스의 데이터 수정 처리 메서드 수정

```
@PostMapping("/guestbook/modify")
public String modify(GuestBookDTO dto,
                     @ModelAttribute("requestDTO") PageRequestDTO
requestDTO,
                     RedirectAttributes redirectAttributes){

        log.info("post modify.....................................");
        log.info("dto: " + dto);
        guestBookService.modify(dto);

        redirectAttributes.addAttribute("page",requestDTO.getPage());
        redirectAttributes.addAttribute("type",requestDTO.getType());
        redirectAttributes.addAttribute("keyword",requestDTO.getKeyword());

        redirectAttributes.addAttribute("gno",dto.getGno());

        return "redirect:/guestbook/read";

}
```