7️⃣

Section7. 스프링 MVC - 웹 페이지 만들기

프로젝트 생성

Dependencies

- spring web
- thymeleaf
- lombok

요구사항 분석

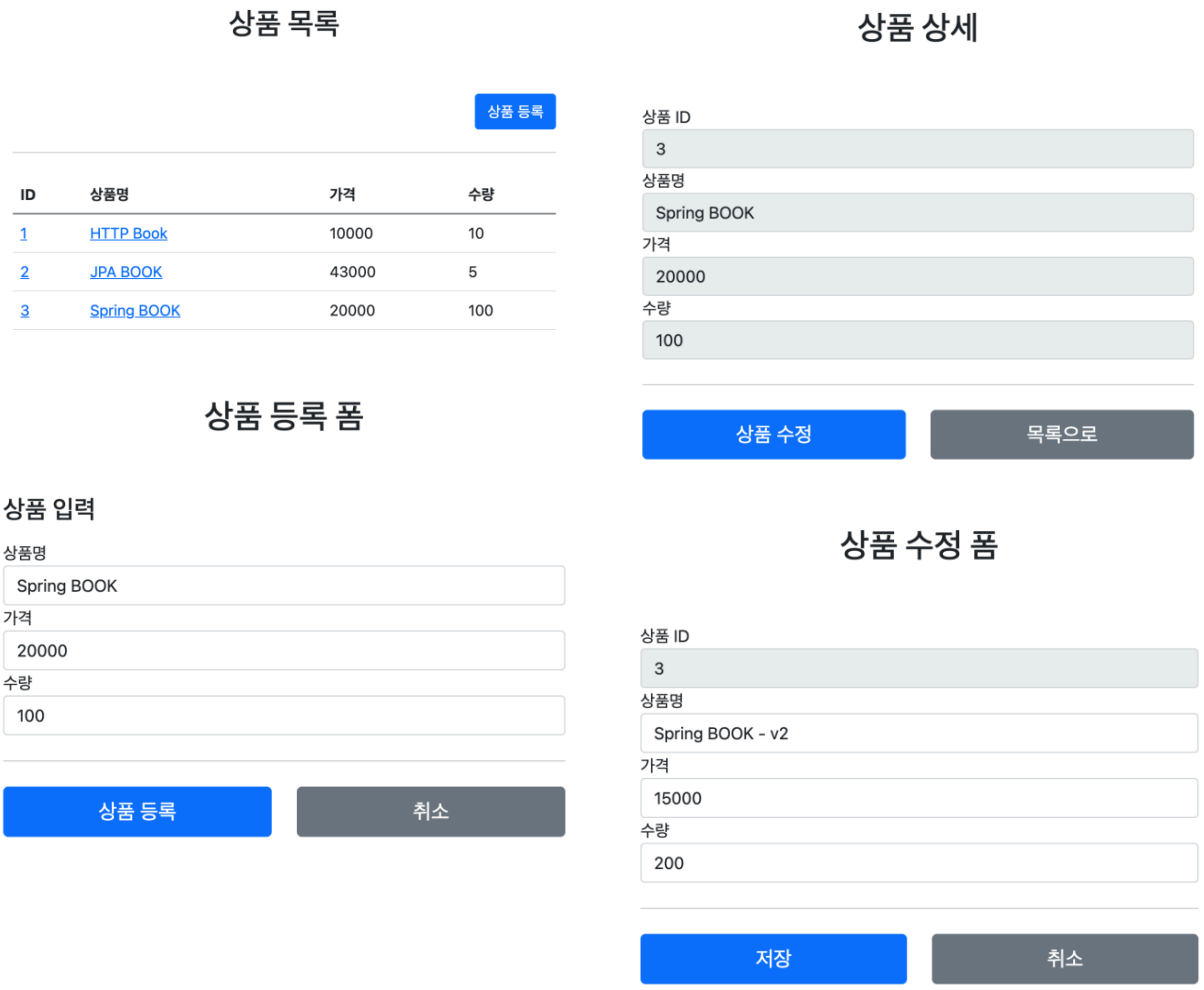
상품 도메인 모델

- 상품 ID
- 상품명
- 가격
- 수량

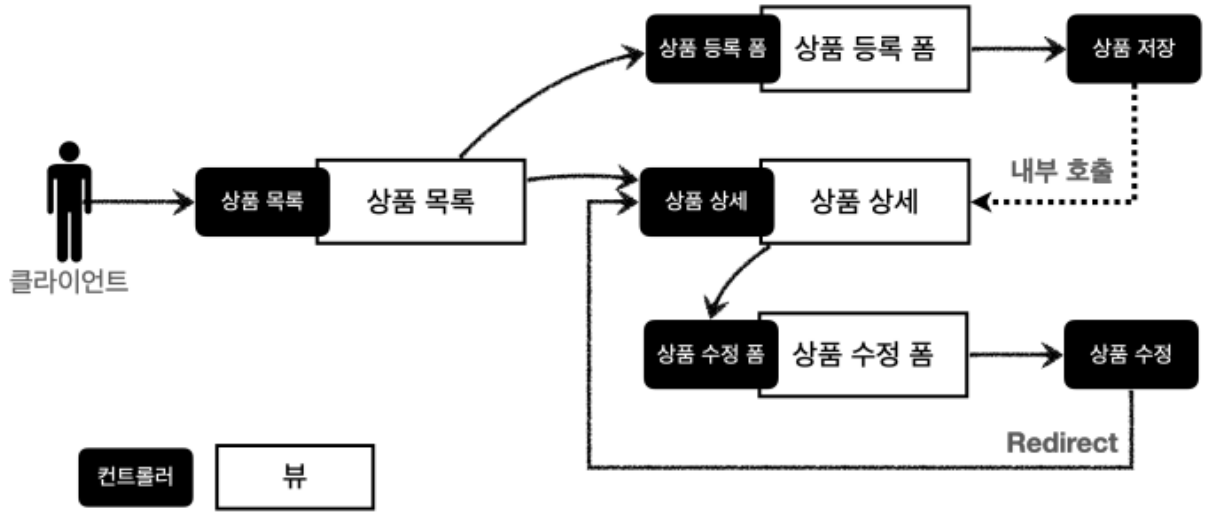
상품 관리 기능

- 상품 목록
- 상품 상세
- 상품 등록
- 상품 수정

서비스 화면


상품 목록
상품 등록
ID
상품명
가격
수량
1
HTTP Book
10000
10
2
JPA BOOK
43000
5
3
Spring BOOK
20000
100
상품 상세
상품 ID
3
상품명
Spring BOOK
가격
20000
수량
100
상품 수정
목록으로
상품 등록 폼
상품 입력
상품명
Spring BOOK
가격
20000
수량
100
상품 등록
취소
상품 수정 폼
상품 ID
3
상품명
Spring BOOK - v2
가격
15000
수량
200
저장
취소

서비스 제공 흐름


상품 등록 폼
상품 등록 폼
상품 저장
내부 호출
상품 목록
상품 목록
상품 상세
상품 상세
클라이언트
상품 수정 폼
상품 수정 폼
상품 수정
Redirect
컨트롤러
뷰

상품 도메인 개발

💡 실습 코드 작성
DTO: Item
Repository: save, findById, findAll, update

상품 서비스 HTML

💡 html 파일 복사 붙여넣기

- 정적 리소스가 공개되는 `/resources/static` 폴더에 HTML을 넣어두면, 실제 서비스에서도 공개된다.
- 서비스 운영 시 공개할 필요없는 HTML을 두는 것을 주의하자.

상품 목록 - 타임리프

컨트롤러 작성

```
@Controller
@RequestMapping("/basic/items")
@RequiredArgsConstructor
public class BasicItemController {
    private final ItemRepository itemRepository;

    @GetMapping
    public String items(Model model){
        List<Item> items = itemRepository.findAll();
        model.addAttribute("items", items);
        return "basic/items";
    }

    @PostConstruct
    public void init(){
        itemRepository.save(new Item("itemA", 10000, 10));
```

```
        itemRepository.save(new Item("itemB", 20000, 20));
    }
}
```

목록 조회 - 타임리프

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <link th:href="@{/css/bootstrap.min.css}"
          href="../../static/css/bootstrap.min.css" rel="st
</head>
<body>
<div class="container" style="max-width: 600px">
    ...
    <div class="row">
        <div class="col">
            <button class="btn btn-primary float-end"
                    onclick="location.href='addForm.html'" ty
                    th:onclick="|location.href=@{/basic/items
            >상품 등록
            </button>
        </div>
    </div>
    <div>
        <table class="table">
            ...
            <tbody>
            <tr th:each="item : ${items}">
                <td><a href="item.html" th:href="@{/basic/ite
                <td><a href="item.html" th:href="@{/basic/ite
                <td th:text="${item.price}">10000</td>
                <td th:text="${item.quantity}">10</td>
            </tr>
            </tbody>
        </table>
    </div>
```

```
</div> <!-- /container -->
</body>
</html>
```

타임리프 사용 선언

```
<html xmlns:th="http://www.thymeleaf.org">
```

속성 변경 - th:href

```
th:href="@{/css/bootstrap.min.css}"
```

- `href="value1"` 을 `th:href="value2"` 의 값으로 변경한다.
- 타임리프 뷰 템플릿을 거치게 되면 원래 값을 `th:xxx` 값으로 변경한다. 만약 값이 없다면 새로 생성한다.

타임리프 핵심

- `th:xxx` 가 붙은 부분은 서버사이드에서 렌더링 되고, 기존 것을 대체한다.
- HTML을 파일로 직접 열었을 때, `th:xxx` 가 있어도 웹 브라우저는 `th: 속성`을 알지 못하므로 무시한다.

URL 링크 표현식 - @{...}

- 타임리프는 URL 링크를 사용하는 경우 `@{...}` 를 사용한다. 이것을 **URL 링크 표현식**이라 한다.
- 서블릿 컨텍스트를 자동으로 포함한다.

속성 변경 - th:onclick

```
th:onclick="|location.href='@{/basic/items/add}'|"
```

💡 **※ 리터럴 대체 - |...| ※**

- 타임리프에서 문자와 표현식 등은 분리되어 있기 때문에 더해서 사용해야 한다.
  -
- 다음과 같이 리터럴 대체 문법을 사용하면, 더하기 없이 편리하게 사용할 수 있다.
  -

반복 출력 - th:each

```
<tr th:each="item : ${items}">
```

- 반복은 `th:each` 를 사용한다.
- 모델에 포함된 items 컬렉션 데이터가 item 변수에 하나씩 포함
  - 반복문 안에서 item 변수를 사용할 수 있다.

변수 표현식 - ${...}

```
<td th:text="${item.price}">10000</td>
```

- 프로퍼티 접근법을 사용한다. (`item.getPrice()`)

내용 변경 - th:text

```
<td th:text="${item.price}">10000</td>
```

- 내용의 값을 `th:text` 의 값으로 변경한다.
  - 10000을 `${item.price}` 의 값으로 변경한다.

URL 링크 표현식2 - @{...}

```
th:href="@{/basic/items/{itemId}(itemId=${item.id})}"

th:href="@{/basic/items/{itemId}(itemId=${item.id}, query='te
// 생성 링크: http://localhost:8080/basic/items/1?query=test
```

- 경로 변수(`{itemId}`) 뿐만 아니라 쿼리 파라미터도 생성한다

> 💡 **순수 HTML을 그대로 유지하면서 뷰 템플릿도 사용**할 수 있는 타임리프의 특징을 네츄럴 템플릿(natural templates)이라 한다.
>
> JSP를 생각해보면, JSP 파일은 웹 브라우저에서 그냥 열면 JSP 소스코드와 HTML이 뒤죽박죽 되어서 정상적인 확인이 불가능하다.
>
> 오직 서버를 통해서 JSP를 열어야 한다.

상품 상세

컨트롤러

```
@GetMapping("/{itemId}")
public String item(@PathVariable long itemId, Model model){
    Item item = itemRepository.findById(itemId);
    model.addAttribute("item", item);
    return "basic/item";
}
```

타임리프

```
<!DOCTYPE HTML>
<html>
...
<body>
```

```
<div class="container">
    <div class="py-5 text-center">
        <h2>상품 상세</h2>
    </div>
    <div>
        <label for="itemId">상품 ID</label>
        <input type="text" id="itemId" name="itemId" class="f
    </div>
    <div>
        <label for="itemName">상품명</label>
        <input type="text" id="itemName" name="itemName" clas
    </div>
    <div>
        <label for="price">가격</label>
        <input type="text" id="price" name="price" class="for
    </div>
    <div>
        <label for="quantity">수량</label>
        <input type="text" id="quantity" name="quantity" clas
    </div>
</div> <!-- /container -->
</body>
</html>
```

속성 변경 - th:value

```
th:value="${item.id}"
```

- 모델에 있는 item 정보를 획득하고 프로퍼티 접근법으로 출력한다.

상품 등록 폼

속성 변경 - th:action

```
<form action="item.html" th:action="|@{/basic/items/add}|" me
```

- HTML form 에서 action에 값이 없으면 현재 URL에 데이터를 전송한다.

상품 등록 처리 - @ModelAttribute

POST - HTML Form

- 메시지 바디에 쿼리 파리미터 형식으로 전달

addItemV1 - @RequestParam 사용

```
public String addItemV1( @RequestParam String itemName, @Requ
    @RequestParam Integer quantity, Model model) {
    Item item = new Item();
    item.setItemName(itemName);
    item.setPrice(price);
    item.setQuantity(quantity);

    itemRepository.save(item);

    model.addAttribute("item", item);
    return "/basic/item";
}
```

addItemV2 - @ModelAttribute 사용

```
    @PostMapping("/add")
    public String addItemV2(@ModelAttribute("item") Item item
        itemRepository.save(item);
        // model.addAttribute("item", item); // 자동 추가, 생략
        return "/basic/item";
    }
```

@ModelAttribute 역할

- 요청 파라미터 처리
  - Item 객체를 생성
  - 요청 파라미터의 값을 프로퍼티 접근법(setXxx)으로 입력
- Model 추가
  - 모델(Model)에 `@ModelAttribute` 로 지정한 객체를 자동으로 넣어준다.
  - 이름은 `@ModelAttribute` 에 지정한 name(value) 속성을 사용한다.

- name을 생략해도 된다.
- 어노테이션 자체도 생략해도 된다.

상품 수정

```
@PostMapping("/{itemId}/edit")
public String edit(@PathVariable("itemId") Long itemId, Item 
    itemRepository.update(itemId, item);
    return "redirect:/basic/items/{itemId}";
}
```

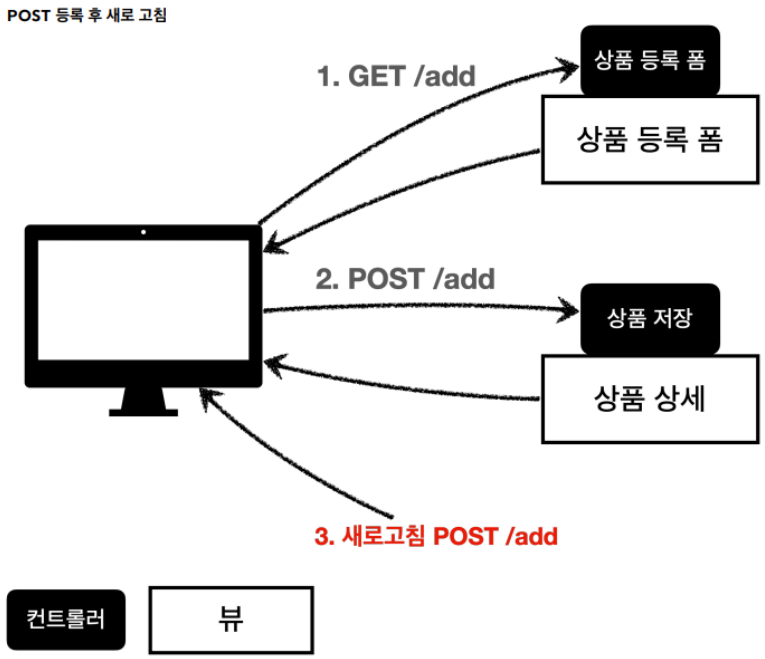
리다이렉트

- 상품 수정은 마지막에 뷰 템플릿을 호출 X
- 상품 상세 화면으로 이동하도록 리다이렉트를 호출한다.
- `redirect:/basic/items/{itemId}`
  - @PathVariable 의 값은 redirect 에도 사용 할 수 있다.

PRG Post/Redirect/Get

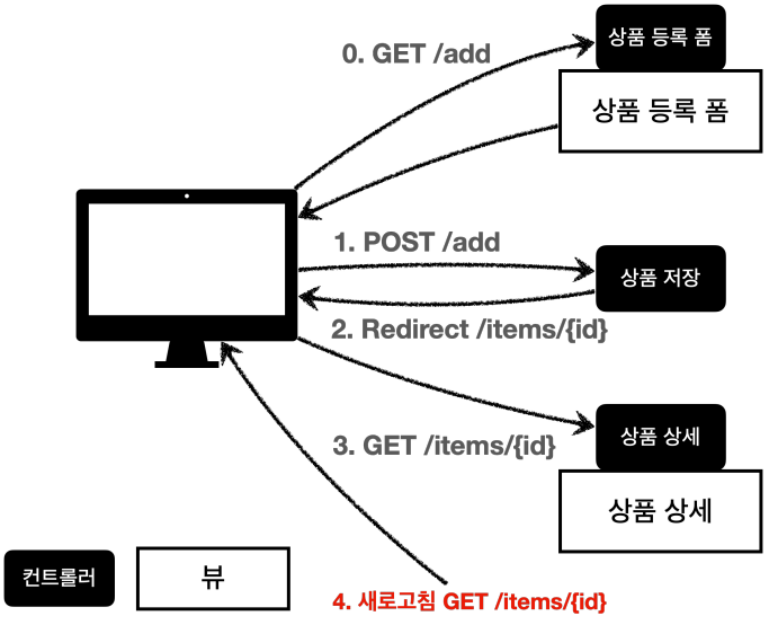
문제

상품 등록을 완료하고 웹 브라우저의 새로고침 버튼을 누르면 상품이 중복해서 등록되는 문제 발생



- 상품 등록을 완료한 후 현재 브라우저의 마지막 요청은 POST /add 인 상태이다.
  이때 마지막으로 서버에 전송한 데이터를 다시 전송해주는 새로고침 시 POST 요청이 계속된다.

해결 방법



1. 상품 저장 후 뷰 템플릿으로 이동하는 것이 아닌, 상품 상세 화면으로 리다이렉트를 호출해준다.

2. 웹 브라우저는 리다이렉트의 영향으로 상품 저장 후에 실제 상품 상세 화면으로 다시 이동한다.
3. 마지막 호출한 내용이 상품 상세 화면인 `GET /items/{id}` 가 된다.

코드

```
@PostMapping("/add")
public String addItemV3(@ModelAttribute("item") Item item) {
    itemRepository.save(item);
    // model.addAttribute("item", item); // 자동 추가, 생략 가능
    return "redirect:/basic/items/" + item.getId();
}
```

- **return "redirect:/basic/items/" + item.getId();**
  - 해당 부분에서 `+item.getId()` 처럼 URL에 변수를 더해서 사용 시 URL 인코딩이 안되기 때문에 위험하다.

    → **RedirectAttributes** 를 사용하자

RedirectAttributes

저장이 잘 되었으면 상품 상세 화면에 "저장되었습니다"라는 메시지를 보여달라는 요구사항

```
@PostMapping("/add")
public String addItemV6(Item item, RedirectAttributes redirec
    Item savedItem = itemRepository.save(item);
    redirectAttributes.addAttribute("itemId", savedItem.getId
    redirectAttributes.addAttribute("status", true);
    // model.addAttribute("item", item); // 자동 추가, 생략 가능
    return "redirect:/basic/items/{itemId}";
}
```

- 저장에 성공했을 때 redirect 시 `status=ture` 추가
  - 템플릿에서 이 값이 있으면, '저장완료' 라는 문구를 출력해보자.

RedirectAttributes

- URL 인코딩

- `redirect:/basic/items/{itemId}`

- PathVariable
  - pathVariable 바인딩: `{itemId}`
- 쿼리 파라미터
  - ?status=true

뷰 템플릿 메시지 추가

```
<!-- item.html -->
<h2 th:if="${param.status}" th:text="'저장 완료!'"></h2
```

- **th:if**
  - 해당 조건이 참이면 실행
- **${param.status}**
  - 타임리프에서 쿼리 파라미터를 조회하는 기능