학습 목표

- 1. 주문, 주문이력 조회, 주문 취소 기능 구현을 통해서 주문 프로세스를 학습한다.

- 2. Spring Data JPA를 이용하여 주문 데이터 조회 시 조회를 최적화하는 방법을 학습한다.

7.1 주문 기능 구현

1. 주문하면 현재 상품의 재고에서 주문 수량만큼 재고 감소

2. 상품의 주문 수량보다 재고의 수가 적을 때 발생시킬 exception을 정의

[함께 해봐요 7-1] 주문 기능 구현하기

com.shop.exception.OutOfStockException.java

```
01  package com.shop.exception;
02
03  public class OutOfStockException extends RuntimeException{
04
05      public OutOfStockException(String message) {
06          super(message);
07      }
08
09  }
```

7.1 주문 기능 구현

```
                                                         com.shop.entity.Item.java
01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  import com.shop.exception.OutOfStockException;
06
07  @Entity
08  @Table(name="item")
09  @Getter @Setter
10  @ToString
11  public class Item extends BaseEntity{
12
13      .....코드 생략.....
14
15      public void removeStock(int stockNumber){
16          int restStock = this.stockNumber - stockNumber;  ─────────────── ❶
17          if(restStock<0){
18              throw new OutOfStockException("상품의 재고가 부족 합니다.
    (현재 재고 수량: " + this.stockNumber + ")");  ─────────────── ❷
19          }
20          this.stockNumber = restStock;  ─────────────── ❸
21      }
22
23  }
```

7.1 주문 기능 구현

com.shop.entity.Item.java

```java
01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  import com.shop.exception.OutOfStockException;
06
07  @Entity
08  @Table(name="item")
09  @Getter @Setter
10  @ToString
11  public class Item extends BaseEntity{
12
13      .....코드 생략.....
14
15      public void removeStock(int stockNumber){
16          int restStock = this.stockNumber - stockNumber;           ➊
17          if(restStock<0){
18              throw new OutOfStockException("상품의 재고가 부족 합니다.
    (현재 재고 수량: " + this.stockNumber + ")");                        ➋
19          }
20          this.stockNumber = restStock;                             ➌
21      }
22
23  }
```

7.1 주문 기능 구현

- 주문할 상품과 주문 수량을 통해 OrderItem 객체를 만드는 메소드를 작성

```java
                                              com.shop.entity.OrderItem.java
01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  @Entity
06  @Getter @Setter
07  public class OrderItem extends BaseEntity{
08
09      .....코드 생략.....
10
11      public static OrderItem createOrderItem(Item item, int count){
12          OrderItem orderItem = new OrderItem();
13          orderItem.setItem(item);                          ❶
14          orderItem.setCount(count);                        ❷
15      orderItem.setOrderPrice(item.getPrice());             ❸
16
17          item.removeStock(count);                          ❹
18          return orderItem;
19      }
20
21      public int getTotalPrice(){                           ❺
22          return orderPrice*count;
23      }
24  }
```

7.1 주문 기능 구현

- 주문할 상품과 주문 수량을 통해 OrderItem 객체를 만드는 메소드를 작성

```
01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  @Entity
06  @Getter @Setter
07  public class OrderItem extends BaseEntity{
08
09      .....코드 생략.....
10
11      public static OrderItem createOrderItem(Item item, int count){
12          OrderItem orderItem = new OrderItem();
13          orderItem.setItem(item);                          ①
14          orderItem.setCount(count);                        ②
15      orderItem.setOrderPrice(item.getPrice());             ③
16
17          item.removeStock(count);                          ④
18          return orderItem;
19      }
20
21      public int getTotalPrice(){                           ⑤
22          return orderPrice*count;
23      }
24  }
```

com.shop.entity.OrderItem.java

7.1 주문 기능 구현

```
                                              com.shop.entity.Order.java
01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  @Entity
06  @Table(name = "orders")
07  @Getter @Setter
08  public class Order extends BaseEntity{
09
10      .....코드 생략.....
11
12      public void addOrderItem(OrderItem orderItem){        ❶
13          orderItems.add(orderItem);
14          orderItem.setOrder(this);                         ❷
15      }
16
17      public static Order createOrder(Member member, List<OrderItem> orderItemList){
18          Order order = new Order();
19          order.setMember(member);                          ❸
20          for(OrderItem orderItem : orderItemList){         ❹
21              order.addOrderItem(orderItem);
22          }
23          order.setOrderStatus(OrderStatus.ORDER);          ❺
24          order.setOrderDate(LocalDateTime.now());          ❻
25          return order;
26      }
27
28      public int getTotalPrice(){                           ❼
29          int totalPrice = 0;
30          for(OrderItem orderItem : orderItems){
31              totalPrice += orderItem.getTotalPrice();
32          }
33          return totalPrice;
34      }
35  }
```

- 주문 상품 객체를 이용하여 주문 객체를 만드는 메소드 작성

7.1 주문 기능 구현

```java
package com.shop.dto;

import lombok.Getter;
import lombok.Setter;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

@Getter @Setter
public class OrderDto {

    @NotNull(message = "상품 아이디는 필수 입력 값입니다.")
    private Long itemId;

    @Min(value = 1, message = "최소 주문 수량은 1개 입니다.")
    @Max(value = 999, message = "최대 주문 수량은 999개 입니다.")
    private int count;

}
```

com.shop.dto.OrderDto.java

7.1 주문 기능 구현

```
01  package com.shop.service;
02
03  import com.shop.dto.OrderDto;
04  import com.shop.entity.*;
05  import com.shop.repository.ItemRepository;
06  import com.shop.repository.MemberRepository;
07  import com.shop.repository.OrderRepository;
08  import lombok.RequiredArgsConstructor;
09  import org.springframework.stereotype.Service;
10  import org.springframework.transaction.annotation.Transactional;
11
12  import javax.persistence.EntityNotFoundException;
13  import java.util.ArrayList;
14  import java.util.List;
```

7.1 주문 기능 구현

```java
16  @Service
17  @Transactional
18  @RequiredArgsConstructor
19  public class OrderService {
20
21      private final ItemRepository itemRepository;
22      private final MemberRepository memberRepository;
23      private final OrderRepository orderRepository;
24
25      public Long order(OrderDto orderDto, String email){
26          Item item = itemRepository.findById(orderDto.getItemId())      ❶
27                  .orElseThrow(EntityNotFoundException::new);
28          Member member = memberRepository.findByEmail(email);           ❷
29
30          List<OrderItem> orderItemList = new ArrayList<>();
31          OrderItem orderItem =
    OrderItem.createOrderItem(item, orderDto.getCount());               ❸
32          orderItemList.add(orderItem);
33
34          Order order = Order.createOrder(member, orderItemList);        ❹
35          orderRepository.save(order);                                   ❺
36
37          return order.getId();
38      }
39
40  }
```

7.1 주문 기능 구현

com.shop.controller.OrderController.java

```java
01  package com.shop.controller;
02
03  import com.shop.dto.OrderDto;
04  import com.shop.service.OrderService;
05  import lombok.RequiredArgsConstructor;
06  import org.springframework.http.HttpStatus;
07  import org.springframework.http.ResponseEntity;
08  import org.springframework.stereotype.Controller;
09  import org.springframework.validation.BindingResult;
10  import org.springframework.validation.FieldError;
11  import org.springframework.web.bind.annotation.PostMapping;
12  import org.springframework.web.bind.annotation.RequestBody;
13  import org.springframework.web.bind.annotation.ResponseBody;
14
15  import javax.validation.Valid;
16  import java.security.Principal;
17  import java.util.List;
18
19  @Controller
20  @RequiredArgsConstructor
```

7.1 주문 기능 구현

```
21  public class OrderController {
22
23      private final OrderService orderService;
24
25      @PostMapping(value = "/order")
26      public @ResponseBody ResponseEntity order (@RequestBody @Valid OrderDto orderDto,
              BindingResult bindingResult, Principal principal){ ───────────────❶
27
28          if(bindingResult.hasErrors()){ ───────────────❷
29              StringBuilder sb = new StringBuilder();
30              List<FieldError> fieldErrors = bindingResult.getFieldErrors();
31              for (FieldError fieldError : fieldErrors) {
32                  sb.append(fieldError.getDefaultMessage());
33              }
34              return new ResponseEntity<String>(sb.toString(),
                  HttpStatus.BAD_REQUEST); ───────────────❸
35          }
36
37          String email = principal.getName(); ───────────────❹
38          Long orderId;
39
40          try {
41              orderId = orderService.order(orderDto, email); ───────────────❺
42          } catch(Exception e){
43              return new ResponseEntity<String>(e.getMessage(),
                  HttpStatus.BAD_REQUEST);
44          }
45
46          return new ResponseEntity<Long>(orderId, HttpStatus.OK); ───────────────❻
47      }
48
49  }
```

7.1 주문 기능 구현

[함께 해봐요 7-2] 주문 기능 테스트하기

com.shop.service.OrderServiceTest.java

```
01  package com.shop.service;
02
03  import com.shop.constant.ItemSellStatus;
04  import com.shop.dto.OrderDto;
05  import com.shop.entity.Item;
06  import com.shop.entity.Member;
07  import com.shop.entity.Order;
08  import com.shop.entity.OrderItem;
09  import com.shop.repository.ItemRepository;
10  import com.shop.repository.MemberRepository;
11  import com.shop.repository.OrderRepository;
12  import org.junit.jupiter.api.DisplayName;
13  import org.junit.jupiter.api.Test;
14  import org.springframework.beans.factory.annotation.Autowired;
15  import org.springframework.boot.test.context.SpringBootTest;
16  import org.springframework.test.context.TestPropertySource;
17  import org.springframework.transaction.annotation.Transactional;
18
19  import javax.persistence.EntityNotFoundException;
20
21  import java.util.List;
22
23  import static org.junit.jupiter.api.Assertions.assertEquals;
24
```

```
25  @SpringBootTest
26  @Transactional
27  @TestPropertySource(locations="classpath:application-test.properties")
28  class OrderServiceTest {
29
30      @Autowired
31      private OrderService orderService;
32
33      @Autowired
34      private OrderRepository orderRepository;
35
36      @Autowired
37      ItemRepository itemRepository;
38
39      @Autowired
40      MemberRepository memberRepository;
41
42      public Item saveItem(){  ·····················································  ❶
43          Item item = new Item();
44          item.setItemNm("테스트 상품");
45          item.setPrice(10000);
46          item.setItemDetail("테스트 상품 상세 설명");
47          item.setItemSellStatus(ItemSellStatus.SELL);
48          item.setStockNumber(100);
49          return itemRepository.save(item);
50      }
```

7.1 주문 기능 구현

```
52    public Member saveMember(){                                                    ➋
53        Member member = new Member();
54        member.setEmail("test@test.com");
55        return memberRepository.save(member);
56    }
57
58    @Test
59    @DisplayName("주문 테스트")
60    public void order(){
61        Item item = saveItem();
62        Member member = saveMember();
63
64        OrderDto orderDto = new OrderDto();
65        orderDto.setCount(10);                                                      ➌
66        orderDto.setItemId(item.getId());                                           ➍
67
68        Long orderId = orderService.order(orderDto, member.getEmail());   ➎
69
70        Order order = orderRepository.findById(orderId)                            ➏
71                .orElseThrow(EntityNotFoundException::new);
72
73        List<OrderItem> orderItems = order.getOrderItems();
74
75        int totalPrice = orderDto.getCount()*item.getPrice();                      ➐
76
77        assertEquals(totalPrice, order.getTotalPrice());                           ➑
78    }
79
80 }
```

7.1 주문 기능 구현

[그림 7-1] 주문 테스트 코드 실행 결과

7.1 주문 기능 구현

- 상품 상세 페이지에서 구현한 주문 로직을 호출하는 코드 작성


```
[함께 해봐요 7-3] 주문 호출 구현하기
                                              resources/templates/item/itemDtl.html

01  <script th:inline="javascript">
02
03      .....코드 생략.....
04
05      function order(){
06          var token = $("meta[name='_csrf']").attr("content");      ❶
07          var header = $("meta[name='_csrf_header']").attr("content");   ❷
08
09          var url = "/order";
10          var paramData = {                                         ❸
11              itemId : $("#itemId").val(),
12              count : $("#count").val()
13          };
14
15          var param = JSON.stringify(paramData);                    ❹
16
17          $.ajax({
18              url     : url,
19              type    : "POST",
20              contentType : "application/json",                     ❺
21              data    : param,
22              beforeSend : function(xhr){
```

7.1 주문 기능 구현

```
23                  /* 데이터를 전송하기 전에 헤더에 csrf 값을 설정 */
24                  xhr.setRequestHeader(header, token);
25              },
26              dataType : "json",                                      ⑥
27              cache    : false,
28              success  : function(result, status){                    ⑦
29                  alert("주문이 완료 되었습니다.");
30                  location.href='/';
31              },
32              error : function(jqXHR, status, error){
33
34                  if(jqXHR.status == '401'){                          ⑧
35                      alert('로그인 후 이용해주세요');
36                      location.href='/members/login';
37                  } else{
38                      alert(jqXHR.responseText);                      ⑨
39                  }
40
41              }
42          });
43      }
44  </script>
```

7.1 주문 기능 구현

- 주문하기 버튼 클릭 시 order() 함수 호출 로직 추가


```
                                                    resources/templates/item/itemDtl.html
01  <button type="button" class="btn btn-primary btn-lg" onclick="order()">주문하기
    </button>
```

7.1 주문 기능 구현

[그림 7-2] 상품 주문 성공

7.2 주문 이력 조회

```java
01  package com.shop.dto;
02
03  import com.shop.entity.OrderItem;
04  import lombok.Getter;
05  import lombok.Setter;
06
07  @Getter @Setter
08  public class OrderItemDto {
09
10      public OrderItemDto(OrderItem orderItem, String imgUrl){        ──────●
11          this.itemNm = orderItem.getItem().getItemNm();
12          this.count = orderItem.getCount();
13          this.orderPrice = orderItem.getOrderPrice();
14          this.imgUrl = imgUrl;
15      }
16
17      private String itemNm; //상품명
18
19      private int count; //주문 수량
20
21      private int orderPrice; //주문 금액
22
23      private String imgUrl;   //상품 이미지 경로
24
25  }
```

7.2 주문 이력 조회

```java
                                                    com.shop.dto.OrderHistDto.java
01  package com.shop.dto;
02
03  import com.shop.constant.OrderStatus;
04  import com.shop.entity.Order;
05  import lombok.Getter;
06  import lombok.Setter;
07
08  import java.time.format.DateTimeFormatter;
09  import java.util.ArrayList;
10  import java.util.List;
11
12  @Getter @Setter
13  public class OrderHistDto {
14
15      public OrderHistDto(Order order){ ──────────────────────────①
16          this.orderId = order.getId();
17          this.orderDate =
    order.getOrderDate().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm"));
18          this.orderStatus = order.getOrderStatus();
19      }
20
21      private Long orderId; //주문아이디
22
23      private String orderDate; //주문날짜
24
25      private OrderStatus orderStatus; //주문 상태
26
27      //주문 상품 리스트
28      private List<OrderItemDto> orderItemDtoList = new ArrayList<>();
29
30      public void addOrderItemDto(OrderItemDto orderItemDto){ ──────────②
31          orderItemDtoList.add(orderItemDto);
32      }
33
34  }
```

7.2 주문 이력 조회

- 주문이력 조회 쿼리 작성

```
com.shop.repository.OrderRepository.java
01  package com.shop.repository;
02
03  .....기존 임포트 생략.....
04
05  import org.springframework.data.domain.Pageable;
06  import org.springframework.data.jpa.repository.Query;
07  import org.springframework.data.repository.query.Param;
08
09  import java.util.List;
10
11  public interface OrderRepository extends JpaRepository<Order, Long> {
12
13      @Query("select o from Order o " +
14              "where o.member.email = :email " +
15              "order by o.orderDate desc"
16      )
17      List<Order> findOrders(@Param("email") String email, Pageable pageable); ❶
18
19      @Query("select count(o) from Order o " +
20              "where o.member.email = :email"
21      )
22      Long countOrder(@Param("email") String email); ────────❷
23
24  }
```

7.2 주문 이력 조회

- 주문 상품의 대표 이미지를 보여주기 위한 쿼리 작성

```
                                          com.shop.repository.ItemImgRepository.java
01   package com.shop.repository;
02
03   .....기존 임포트 생략.....
04
05   public interface ItemImgRepository extends JpaRepository<ItemImg, Long> {
06
07       List<ItemImg> findByItemIdOrderByIdAsc(Long itemId);
08
09       ItemImg findByItemIdAndRepimgYn(Long itemId, String repimgYn);
10   }
```

7.2 주문 이력 조회

```java
01  package com.shop.service;
02
03  .....기존 임포트 생략.....
04
05  import com.shop.dto.OrderHistDto;
06  import com.shop.dto.OrderItemDto;
07  import com.shop.repository.ItemImgRepository;
08  import org.springframework.data.domain.Page;
09  import org.springframework.data.domain.PageImpl;
10  import org.springframework.data.domain.Pageable;
11
12  @Service
13  @Transactional
14  @RequiredArgsConstructor
15  public class OrderService {
16
17      private final ItemRepository itemRepository;
18      private final MemberRepository memberRepository;
19      private final OrderRepository orderRepository;
20      private final ItemImgRepository itemImgRepository;
21
22      .....코드 생략.....
23
```

7.2 주문 이력 조회

```
24      @Transactional(readOnly = true)
25      public Page<OrderHistDto> getOrderList(String email, Pageable pageable) {
26
27          List<Order> orders  = orderRepository.findOrders(email, pageable); ❶
28          Long totalCount = orderRepository.countOrder(email); ──────────❷
29
30          List<OrderHistDto> orderHistDtos = new ArrayList<>();
31
32          for (Order order : orders) { ───────────────────────────❸
33              OrderHistDto orderHistDto = new OrderHistDto(order);
34              List<OrderItem> orderItems = order.getOrderItems();
35              for (OrderItem orderItem : orderItems) {
36                  ItemImg itemImg = itemImgRepository.findByItemIdAndRepimgYn
    (orderItem.getItem().getId(), "Y"); ──────────────────────────❹
37                  OrderItemDto orderItemDto =
    new OrderItemDto(orderItem, itemImg.getImgUrl());
38                  orderHistDto.addOrderItemDto(orderItemDto);
39              }
40
41              orderHistDtos.add(orderHistDto);
42          }
43
44          return new PageImpl<OrderHistDto>(orderHistDtos, pageable, totalCount); ❺
45      }
46
47 }
```

7.2 주문 이력 조회

```
                                        com.shop.controller.OrderController.java
01  package com.shop.controller;
02
03  .....기존 임포트 생략.....
04
05  import org.springframework.web.bind.annotation.GetMapping;
06  import org.springframework.web.bind.annotation.PathVariable;
07  import com.shop.dto.OrderHistDto;
08  import org.springframework.data.domain.Page;
09  import org.springframework.data.domain.PageRequest;
10  import org.springframework.data.domain.Pageable;
11  import org.springframework.ui.Model;
12  import java.util.Optional;
13
14  @Controller
15  @RequiredArgsConstructor
16  public class OrderController {
17
18      .....코드 생략.....
19
20      @GetMapping(value = {"/orders", "/orders/{page}"})
21      public String orderHist(@PathVariable("page") Optional<Integer> page,
    Principal principal, Model model){
22          Pageable pageable = PageRequest.of(page.isPresent() ? page.get() : 0, 4);  ❶
23
24          Page<OrderHistDto> ordersHistDtoList =
25  orderService.getOrderList(principal.getName(), pageable);  ─────────────────  ❷
25
26          model.addAttribute("orders", ordersHistDtoList);
27          model.addAttribute("page", pageable.getPageNumber());
28          model.addAttribute("maxPage", 5);
29          return "order/orderHist";
30      }
31
32  }
```

7.2 주문 이력 조회

- 주문 이력 페이지 orderHist.html 작성

resources/templates/order/orderHist.html

```
01  <!DOCTYPE html>
02  <html xmlns:th="http://www.thymeleaf.org"
03       xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
04       layout:decorate="~{layouts/layout1}">
05
06  <head>
07      <meta name="_csrf" th:content="${_csrf.token}"/>
08      <meta name="_csrf_header" th:content="${_csrf.headerName}"/>
09  </head>
10
11  <!-- 사용자 CSS 추가 -->
12  <th:block layout:fragment="css">
13      <style>
14          .content-mg{
15              margin-left:30%;
16              margin-right:30%;
17              margin-top:2%;
18              margin-bottom:100px;
19          }
20          .repImgDiv{
21              margin-right:15px;
22              margin-left:15px;
23              height:auto;
24          }
```

```
25          .repImg{
26              height:100px;
27              width:100px;
28          }
29          .card{
30              width:750px;
31              height:100%;
32              padding:30px;
33              margin-bottom:20px;
34          }
35          .fs18{
36              font-size:18px;
37          }
38          .fs24{
39              font-size:24px
40          }
41      </style>
42  </th:block>
43
```

7.2 주문 이력 조회

```html
44  <div layout:fragment="content" class="content-mg">
45
46      <h2 class="mb-4">
47          구매 이력
48      </h2>
49
50      <div th:each="order : ${orders.getContent()}">
51
52          <div class="d-flex mb-3 align-self-center">
53              <h4 th:text="${order.orderDate} + ' 주문'"></h4>
54              <div class="ml-3">
55                  <th:block th:if="${order.orderStatus ==
    T(com.shop.constant.OrderStatus).ORDER}">
56                      <button type="button" class="btn btn-outline-secondary"
    th:value="${order.orderId}">주문취소</button>
57                  </th:block>
58                  <th:block th:unless="${order.orderStatus ==
    T(com.shop.constant.OrderStatus).ORDER}">
59                      <h4>(취소 완료)</h4>
60                  </th:block>
61              </div>
62          </div>
63          <div class="card d-flex">
```

```
64                    <div th:each="orderItem : ${order.orderItemDtoList}"
   class="d-flex mb-3">
65                        <div class="repImgDiv">
66                            <img th:src="${orderItem.imgUrl}"
   class = "rounded repImg" th:alt="${orderItem.itemNm}">
67                        </div>
68                        <div class="align-self-center w-75">
69                            <span th:text="${orderItem.itemNm}"
   class="fs24 font-weight-bold"></span>
70                            <div class="fs18 font-weight-light">
71                                <span th:text="${orderItem.orderPrice} +'원'">
                                 </span>
72                                <span th:text="${orderItem.count} +'개'"></span>
73                            </div>
74                        </div>
75                    </div>
76                </div>
77
78        </div>
79
```

7.2 주문 이력 조회

```
80      <div th:with="start=${(orders.number/maxPage)*maxPage + 1},
    end=(${(orders.totalPages == 0) ? 1 : (start + (maxPage - 1)
    < orders.totalPages ? start + (maxPage - 1) : orders.totalPages)})" >
81          <ul class="pagination justify-content-center">
82
83          <li class="page-item"
    th:classappend="${orders.number eq 0}?'disabled':''">
84              <a th:href="@{'/orders/' + ${orders.number-1}}"
    aria-label='Previous' class="page-link">
85                  <span aria-hidden='true'>Previous</span>
86              </a>
87          </li>
88
89          <li class="page-item"
    th:each="page: ${#numbers.sequence(start, end)}"
    th:classappend="${orders.number eq page-1}?'active':''">
90              <a th:href="@{'/orders/' + ${page-1}}" th:inline="text"
    class="page-link">[[${page}]]</a>
91          </li>
92
93          <li class="page-item"
    th:classappend="${orders.number+1 ge orders.totalPages}?'disabled':''">
```

7.2 주문 이력 조회

```
94                    <a th:href="@{'/orders/' + ${orders.number+1}}"
      aria-label='Next' class="page-link">
95                        <span aria-hidden='true'>Next</span>
96                    </a>
97               </li>
98
99          </ul>
100     </div>
101
102 </div>
103
104 </html>
```

7.2 주문 이력 조회
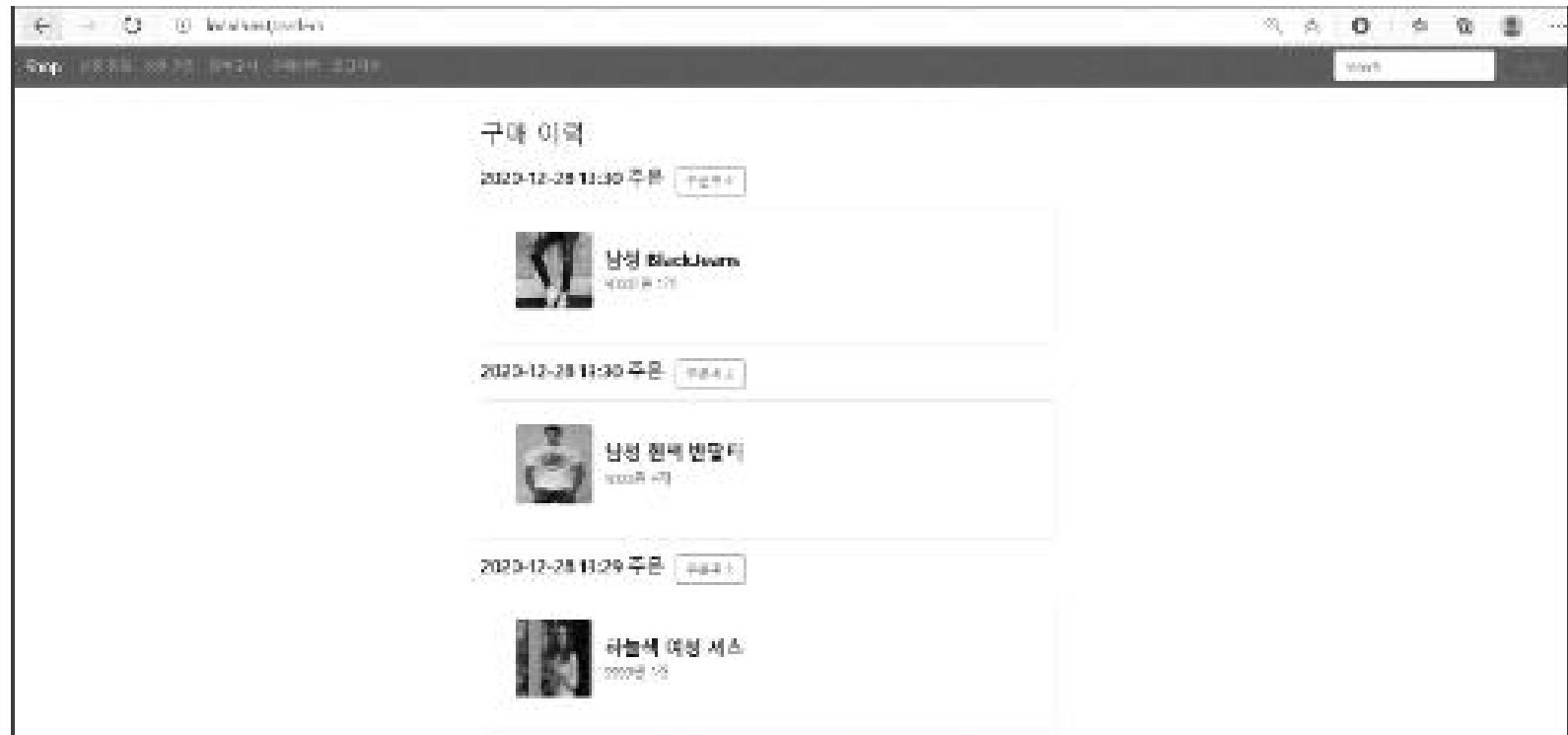
[그림 7-3] 주문 이력 페이지

7.2 주문 이력 조회

- OrderService 클래스에 구현한getOrderList() 메소드에서 for문을 순회하면서 order.getOrderItems()를 호출할 때마다 조회 쿼리문이 추가로 실행

```java
@Transactional(readOnly = true)
public Page<OrderHistDto> getOrderList(String email, Pageable pageable) {

    List<Order> orders = orderRepository.findOrders(email, pageable);
    Long totalCount = orderRepository.countOrder(email);

    List<OrderHistDto> orderHistDtos = new ArrayList<>();

    for (Order order : orders) {
        OrderHistDto orderHistDto = new OrderHistDto(order);
        List<OrderItem> orderItems = order.getOrderItems();
        for (OrderItem orderItem : orderItems) {
            ItemImg itemImg = itemImgRepository.findByItemIdAndRepimgYn(orderItem.getItem().getId(), repimgYn "Y");
            OrderItemDto orderItemDto = new OrderItemDto(orderItem, itemImg.getImgUrl());
            orderHistDto.addOrderItemDto(orderItemDto);
        }

        orderHistDtos.add(orderHistDto);
    }

    return new PageImpl<OrderHistDto>(orderHistDtos, pageable, totalCount);
}
```

[그림 7-4] 주문 리스트 조회 로직

7.2 주문 이력 조회

- OrderService 클래스에 구현한getOrderList() 메소드에서 for문을 순회하면서 order.getOrderItems()를 호출할 때마다 조회 쿼리문이 추가로 실행

[그림 7-5] 주문 상품 리스트 조회 쿼리

7.2 주문 이력 조회

- orders의 주문 아이디를 "where order_id in (209, 210, 211, 212)" 이런 식으로 in 쿼리로 한번에 조회할 수 있다면 100개가 실행될 쿼리를 하나의 쿼리로 조회가능

- "default_batch_fetch_size"라는 옵션을 사용하여 조회 쿼리 한 번으로 지정한 사이즈 만큼 한 번에 조회 가능

application.properties 설정 추가하기

```
#기본 batch size 설정
spring.jpa.properties.hibernate.default_batch_fetch_size=1000
```

7.2 주문 이력 조회

- 옵션 추가 후 조건절에 in 쿼리문이 실행되는 것을 볼 수 있음

- JPA 사용 시 N+1 문제를 많이 만나게 되는데 성능상 이슈가 생길 수 있기 때
  문에 조심해서 사용 필요

[그림 7-6] default_batch_fetch_size 적용 후 주문 상품 리스트 조회 쿼리

7.3 주문 취소

- 주문할 때 상품의 재고를 감소시켰던 만큼 다시 더해주는 addStock 메소드 구현


```
[함께 해봐요 7-5] 주문 취소 기능 구현하기                    com.shop.entity.Item.java

01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  @Entity
06  @Table(name="item")
07  @Getter @Setter
08  @ToString
09  public class Item extends BaseEntity{
10
11      .....코드 생략.....
12
13      public void addStock(int stockNumber){ -------------------------------- ❶
14          this.stockNumber += stockNumber;
15      }
16
17  }
```

7.3 주문 취소

com.shop.entity.OrderItem.java

```
01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  @Entity
06  @Getter @Setter
07  public class OrderItem extends BaseEntity{
08
09      .....코드 생략.....
10
11      public void cancel() {                                          ❶
12          this.getItem().addStock(count);
13      }
14
15  }
```

7.3 주문 취소

```
01  package com.shop.entity;
02
03  .....기존 임포트 생략.....
04
05  @Entity
06  @Table(name = "orders")
07  @Getter @Setter
08  public class Order extends BaseEntity{
09
10      .....코드 생략.....
11
12      public void cancelOrder(){
13          this.orderStatus = OrderStatus.CANCEL;
14
15          for(OrderItem orderItem : orderItems){
16              orderItem.cancel();
17          }
18      }
19
20  }
```

7.3 주문 취소

```
01  package com.shop.service;
02
03  .....기존 임포트 생략.....
04
05  import org.thymeleaf.util.StringUtils;
06
07  @Service
08  @Transactional
09  @RequiredArgsConstructor
10  public class OrderService {
11
12      .....코드 생략.....
13
14      @Transactional(readOnly = true)
15      public boolean validateOrder(Long orderId, String email){  ————————❶
16          Member curMember = memberRepository.findByEmail(email);
17          Order order = orderRepository.findById(orderId)
18                  .orElseThrow(EntityNotFoundException::new);
19          Member savedMember = order.getMember();
20
21          if(!StringUtils.equals(curMember.getEmail(), savedMember.getEmail())){
22              return false;
23          }
24
25          return true;
26      }
27
28      public void cancelOrder(Long orderId){
29          Order order = orderRepository.findById(orderId)
30                  .orElseThrow(EntityNotFoundException::new);
31          order.cancelOrder();  ————————————————————————❷
32      }
33  }
```

7.3 주문 취소

com.shop.controller.OrderController.java

```
01  package com.shop.controller;
02
03  .....기존 임포트 생략.....
04
05  @Controller
06  @RequiredArgsConstructor
07  public class OrderController {
08
09      .....코드 생략.....
10
11      @PostMapping("/order/{orderId}/cancel")
12      public @ResponseBody ResponseEntity cancelOrder
    (@PathVariable("orderId") Long orderId , Principal principal){
13
14          if(!orderService.validateOrder(orderId, principal.getName())){    ──❶
15              return new ResponseEntity<String>("주문 취소 권한이 없습니다.",
          HttpStatus.FORBIDDEN);
16          }
17
18          orderService.cancelOrder(orderId);    ────────────────────────❷
19          return new ResponseEntity<Long>(orderId, HttpStatus.OK);
20      }
21
22  }
```

7.3 주문 취소

```java
01  package com.shop.service;
02
03  .....기존 임포트 생략.....
04
05  import com.shop.constant.OrderStatus;
06
07  @SpringBootTest
08  @Transactional
09  @TestPropertySource(locations="classpath:application-test.properties")
10  class OrderServiceTest {
11
12      .....코드 생략.....
13
14      @Test
15      @DisplayName("주문 취소 테스트")
16      public void cancelOrder(){
17          Item item = saveItem();                                      ❶
18          Member member = saveMember();                                ❷
19
20          OrderDto orderDto = new OrderDto();
21          orderDto.setCount(10);
22          orderDto.setItemId(item.getId());
23          Long orderId = orderService.order(orderDto, member.getEmail());  ❸
24
25          Order order = orderRepository.findById(orderId)
26                  .orElseThrow(EntityNotFoundException::new);          ❹
27          orderService.cancelOrder(orderId);                           ❺
28
29          assertEquals(OrderStatus.CANCEL, order.getOrderStatus());    ❻
30          assertEquals(100, item.getStockNumber());                    ❼
31      }
32
33  }
```

7.3 주문 취소

[그림 7-7] 주문 취소 테스트 실행 결과

7.3 주문 취소

- 주문 취소 기능을 호출하는 자바스크립트 함수 구현

[함께 해봐요 7-7] 주문 취소 호출 구현하기

resources/templates/order/orderHist.html

```html
01  <!-- 사용자 스크립트 추가 -->
02  <th:block layout:fragment="script">
03
04      <script th:inline="javascript">
05          function cancelOrder(orderId) {
06              var token = $("meta[name='_csrf']").attr("content");
07              var header = $("meta[name='_csrf_header']").attr("content");
08
09              var url = "/order/" + orderId + "/cancel";
10              var paramData = {                                          ①
11                  orderId : orderId,
12              };
13
14              var param = JSON.stringify(paramData);
15
```

```
16          $.ajax({
17              url      : url,
18              type     : "POST",
19              contentType : "application/json",
20              data     : param,
21              beforeSend : function(xhr){
22                  /* 데이터를 전송하기 전에 헤더에 csrf 값을 설정 */
23                  xhr.setRequestHeader(header, token);
24              },
25              dataType : "json",
26              cache    : false,
27              success  : function(result, status){ ················· ❷
28                  alert("주문이 취소 되었습니다.");
29                  location.href='/orders/' + [[${page}]];
30              },
31              error : function(jqXHR, status, error){
32                  if(jqXHR.status == '401'){
33                      alert('로그인 후 이용해주세요');
34                      location.href='/members/login';
35                  } else{
36                      alert(jqXHR.responseText);
37                  }
38              }
39          });
40      }
41  </script>
42
43 </th:block>
```

7.3 주문 취소

- 주문 취소 버튼 클릭 시 cancelOrder함수 호출


```
                                                        resources/templates/order/orderHist.html
01   <button type="button" class="btn btn-outline-secondary"
        th:value="${order.orderId}" onclick="cancelOrder(this.value)">주문취소</button>
```

7.3 주문 취소

[그림 7-8] 주문 취소 완료

Thank you for your attention