YouTube
NAVER 카페
구멍가게 코딩단

코드로 배우는 리액트

10. 장바구니 API 만들기

10장. 장바구니 API 만들기

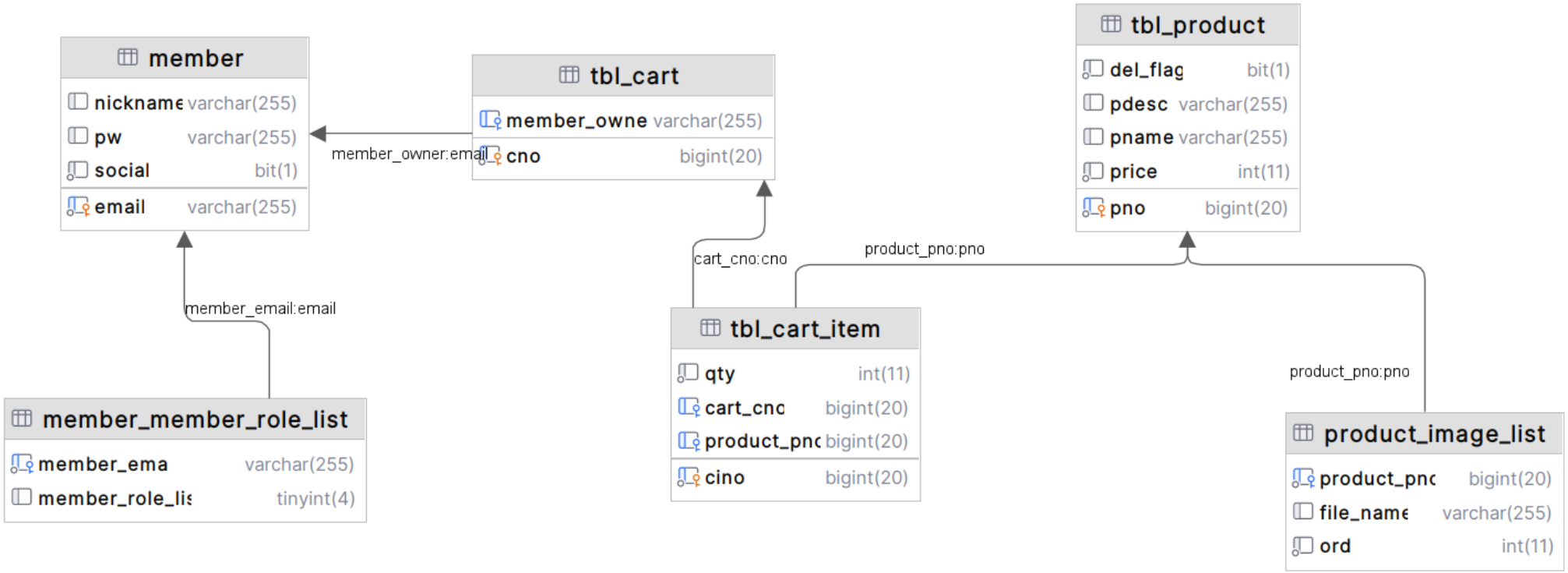
- 인증 처리가 완료된 후, 로그인한 사용자를 위한 장바구니 기능을 구현
- JPA의 연관관계를 활용하여 장바구니를 구성
- @Query를 사용하여 현재 사용자의 장바구니에 있는 상품 목록과 수량(장바구니 아이템)을 반환하는 기능을 구현

개발목표

1. JPA의 연관관계를 이용한 장바구니 설계
2. JPQL을 이용한 조인 처리와 Projections를 이용한 DTO 처리
3. 스프링 시큐리티의 로그인 정보를 활용한 사용자 인증 처리

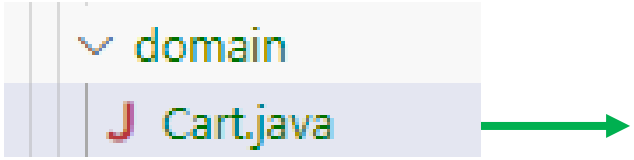
Index

장바구니 엔티티의 설계



장바구니관련 엔티티

→ domain 패키지에 Cart클래스를 추가

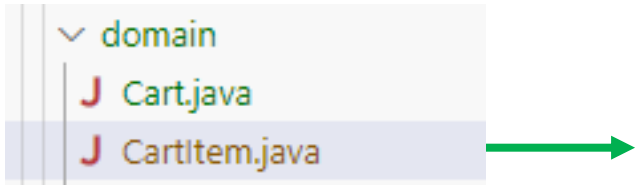



```java
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString(exclude = "owner")
@Table(
  name = "tbl_cart",
  indexes = { @Index(name="idx_cart_email", columnList = "member_owner") }
)
public class Cart {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long cno;

  @OneToOne
  @JoinColumn(name="member_owner")
  private Member owner;
}
```

장바구니관련 엔티티

→ 장바구니 아이템

CartItem.java



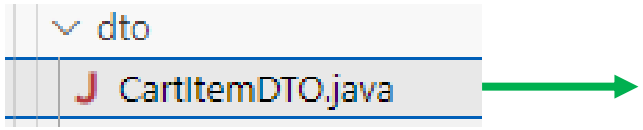

```java
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Builder
@ToString(exclude="cart")
@Table(name = "tbl_cart_item", indexes = {
    @Index(columnList = "cart_cno", name = "idx_cartitem_cart"),
    @Index(columnList = "product_pno, cart_cno", name="idx_cartitem_pno_cart")
})
public class CartItem {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long cino;

  @ManyToOne
  @JoinColumn(name = "product_pno")
  private Product product;

  @ManyToOne
  @JoinColumn(name = "cart_cno")
  private Cart cart;

  private int qty;

  public void changeQty(int qty){
    this.qty = qty;
  }
}
```

카카오 연동 설정

- 상품을 조회하는 화면에서 사용자가 자신의 장바구니에 상품을 추가하는 경우 - 전달되는 데이터는 사용자의 이메일, 추가하고 싶은 상품의 번호, 수량
- 장바구니 아이템 목록에서 상품 수량을 조정하는 경우 – 이미 만들어진 장바구니 아이템 번호(cino), 변경하고자 하는 수량

장바구니 DTO의 설정

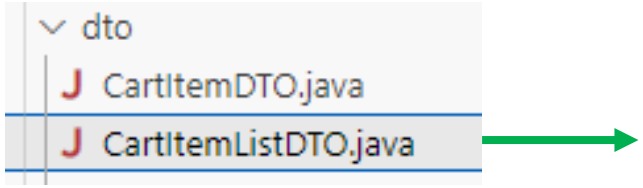
→ CartItemDTO.java




```java
package org.zerock.mallapi.dto;

import lombok.Data;

@Data
public class CartItemDTO {

  private String email;

  private Long pno;

  private int qty;

  private Long cino;

}
```

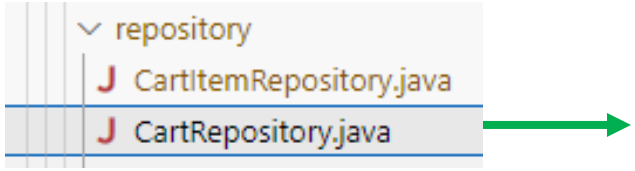
장바구니 DTO의 설정

→ CartItemListDTO.java




```java
package org.zerock.mallapi.dto;

import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
public class CartItemListDTO {

  private Long cino;
  private int qty;
  private Long pno;
 private String pname;
  private int price;
  private String imageFile;

  public CartItemListDTO(Long cino, int qty, Long pno, String pname, int
price, String imageFile){
    this.cino = cino;
    this.qty = qty;
    this.pno = pno;
    this.pname = pname;
    this.price = price;
    this.imageFile = imageFile;
  }
}
```

CartRepository




```java
package org.zerock.mallapi.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import org.zerock.mallapi.domain.Cart;

public interface CartRepository extends JpaRepository<Cart, Long>{

  @Query("select cart from Cart cart where cart.owner.email = :email")
  public Optional<Cart> getCartOfMember(@Param("email") String email);

}
```

CartItemRepository

→ 특정한 사용자의 이메일을 통해서 해당 사용자의 모든 장바구니 아이템들을 조회 기능

→ 로그인 했을 때 사용자가 담은 모든 장바구니 아이템 조회시에 사용

→ 사용자의 이메일과 상품 번호로 해당 장바구니 아이템을 알아내는 기능

→ 새로운 상품을 장바구니에 담고자 할 때 기존 장바구니 아이템인지 확인하기 위해서 필요

→ 장바구니 아이템이 속한 장바구니의 번호를 알아내는 기능

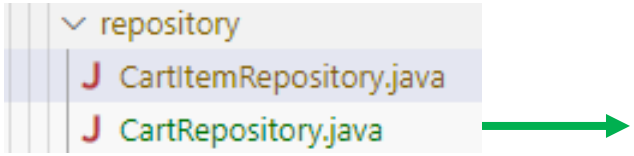
→ 해당 아이템을 삭제한 후 해당 아이템이 속해 있는 장바구니의 모든 아이템을 조회

→ 특정한 장바구니의 번호만으로 해당 장바구니의 모든 장바구니 아이템들을 조회하는 기능

→ 특정한 장바구니 아이템을 삭제한 후에 해당 장바구니 아이템이 속해 있는 장바구니의 모든 장바구니 아이템들을 조회할 때 필요

CartItemRepository

→ CartItemListDTO타입의 객체로 조회




```java
package org.zerock.mallapi.repository;

import java.util.List;
import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.zerock.mallapi.domain.CartItem;
import org.zerock.mallapi.dto.CartItemListDTO;

public interface CartItemRepository extends JpaRepository<CartItem, Long>{

}
```

CartItemRepository

→ CartItemListDTO타입의 객체로 조회

```
@Query("select " +
" new org.zerock.mallapi.dto.CartItemListDTO(ci.cino,  ci.qty,  p.pno, p.pname, p.price , pi.fileName )  " +
" from " +
"   CartItem ci inner join Cart mc on ci.cart = mc " +
"   left join Product p on ci.product = p " +
"   left join p.imageList pi" +
" where " +
"   mc.owner.email = :email and pi.ord = 0 " +
" order by ci desc ")
public List<CartItemListDTO> getItemsOfCartDTOByEmail(@Param("email") String email);
```

CartItemRepository

→ CartItemListDTO타입의 객체로 조회

```
@Query("select" +
" ci "+
" from " +
"   CartItem ci inner join Cart c on ci.cart = c " +
" where " +
"   c.owner.email = :email and ci.product.pno = :pno")
public CartItem getItemOfPno(@Param("email") String email, @Param("pno") Long pno );
```

CartItemRepository

→ CartItemListDTO타입의 객체로 조회

```
@Query("select " +
"  c.cno " +
"from " +
"  Cart c inner join CartItem ci on ci.cart = c " +
" where " +
"  ci.cino = :cino")
public Long getCartFromItem( @Param("cino") Long cino);
```

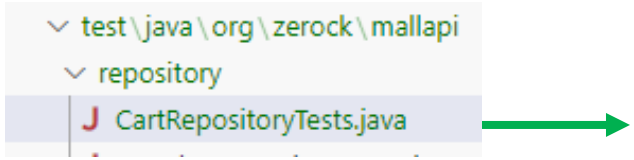
CartItemRepository

→ CartItemListDTO타입의 객체로 조회

```
@Query("select " +
" new org.zerock.mallapi.dto.CartItemListDTO(ci.cino,  ci.qty,  p.pno, p.pname, p.price , pi.fileName ) " +
" from " +
"   CartItem ci inner join Cart mc on ci.cart = mc " +
"   left join Product p on ci.product = p " +
"   left join p.imageList pi" +
" where " +
"  mc.cno = :cno and pi.ord = 0 " +
" order by ci desc ")
public List<CartItemListDTO> getItemsOfCartDTOByCart(@Param("cno") Long cno);
```

CartItemRepository

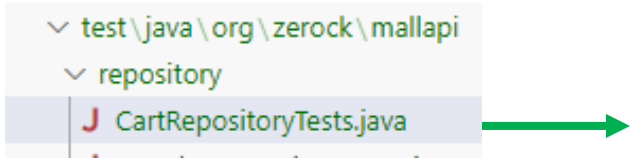
→ 장바구니 아이템 추가 테스트




```java
@SpringBootTest
@Log4j2
public class CartRepositoryTests {

  @Autowired
  private CartRepository cartRepository;

  @Autowired
  private CartItemRepository cartItemRepository;

  @Transactional
  @Commit
  @Test
  public void testInsertByProduct() {

  }
}
```

CartItemRepository

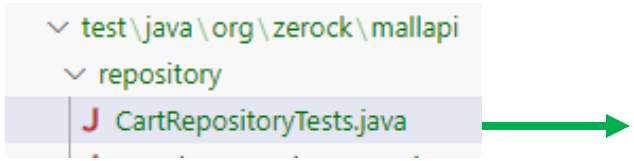
→ 장바구니 아이템 추가 테스트




```java
@Transactional
@Commit
@Test
public void testInsertByProduct() {
  log.info("test1----------------------");
  //사용자가 전송하는 정보
  String email = "user1@aaa.com";
  Long pno = 5L;
  int qty = 1;

  //만일 기존에 사용자의 장바구니 아이템이 있었다면

  CartItem cartItem = cartItemRepository.getItemOfPno(email, pno);

  if(cartItem != null) {
    cartItem.changeQty(qty);
    cartItemRepository.save(cartItem);
    return;
  }
```

CartItemRepository

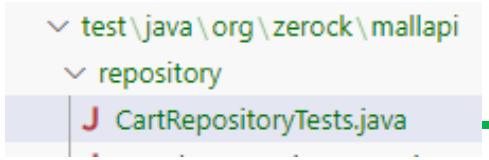
→ 장바구니 아이템 추가 테스트




```
//장바구니 아이템이 없었다면 장바구니부터 확인 필요
//사용자가 장바구니를 만든적이 있는지 확인
Optional<Cart> result = cartRepository.getCartOfMember(email);

Cart cart = null;

//사용자의 장바구니가 존재하지 않으면 장바구니 생성
if(result.isEmpty()) {
  log.info("MemberCart is not exist!!");
  Member member = Member.builder().email(email).build();
  Cart tempCart = Cart.builder().owner(member).build();
  cart = cartRepository.save(tempCart);
}else {
  cart = result.get();
}

log.info(cart);
```

CartItemRepository

→ 장바구니 아이템 추가 테스트




```
    //------------------------------------------------------

    if(cartItem == null){
      Product product = Product.builder().pno(pno).build();
      cartItem =
CartItem.builder().product(product).cart(cart).qty(qty).build();

    }
    //상품 아이템 저장
    cartItemRepository.save(cartItem);
  }


}
```

CartItemRepository

→ 장바구니 아이템 수정 테스트

```
@Test
@Commit
public void tesstUpdateByCino() {

  Long cino = 1L;

  int qty = 4;

  Optional<CartItem> result = cartItemRepository.findById(cino);

  CartItem cartItem = result.orElseThrow();

  cartItem.changeQty(qty);

  cartItemRepository.save(cartItem);

}
```

CartItemRepository

→ 현재 사용자의 장바구니 아이템 목록 테스트

```
@Test
public void testListOfMember() {

  String email = "user1@aaa.com";

  List<CartItemListDTO> cartItemList = cartItemRepository.getItemsOfCartDTOByEmail(email);

  for (CartItemListDTO dto : cartItemList) {
    log.info(dto);
  }

}
```

CartItemRepository

→ 장바구니 아이템 삭제와 목록 조회

```
@Test
public void testDeleteThenList() {

  Long cino = 1L;
  //장바구니 번호
  Long cno = cartItemRepository.getCartFromItem(cino);

  //삭제는 임시로 주석처리
  //cartItemRepository.deleteById(cino);

  //목록
  List<CartItemListDTO> cartItemList = cartItemRepository.getItemsOfCartDTOByCart(cno);

  for (CartItemListDTO dto : cartItemList) {
    log.info(dto);
  }
}
```

장바구니 서비스 계층의 설계/구현

→ service 패키지내에 CartService와 CartServiceImpl로 구현


```java
package org.zerock.mallapi.service;

import java.util.List;

import org.zerock.mallapi.dto.CartItemDTO;
import org.zerock.mallapi.dto.CartItemListDTO;

import jakarta.transaction.Transactional;

@Transactional
public interface CartService {
  //장바구니 아이템 추가 혹은 변경
  public List<CartItemListDTO> addOrModify(CartItemDTO cartItemDTO);
  //모든 장바구니 아이템 목록
  public List<CartItemListDTO> getCartItems(String email);
  //아이템 삭제
  public List<CartItemListDTO> remove(Long cino);
}
```

장바구니 서비스 계층의 설계/구현

→ service 패키지내에 CartService와 CartServiceImpl로 구현


```java
@RequiredArgsConstructor
@Service
@Log4j2
public class CartServiceImpl implements CartService {

  private final CartRepository cartRepository;
  private final CartItemRepository cartItemRepository;

  @Override
  public List<CartItemListDTO> addOrModify(CartItemDTO cartItemDTO) {…}
  //사용자의 장바구니가 없었다면 새로운 장바구니를 생성하고 반환
  private Cart getCart(String email ){ … }
  @Override
  public List<CartItemListDTO> getCartItems(String email) {…}
  @Override
  public List<CartItemListDTO> remove(Long cino) {…}
}
```

장바구니 서비스 계층의 설계/구현

→ service 패키지내에 CartService와 CartServiceImpl로 구현


```java
@Override
public List<CartItemListDTO> addOrModify(CartItemDTO cartItemDTO) {
  String email = cartItemDTO.getEmail();
  Long pno = cartItemDTO.getPno();
  int qty = cartItemDTO.getQty();
  Long cino = cartItemDTO.getCino();
  if(cino != null) { //장바구니 아이템 번호가 있어서 수량만 변경하는 경우
    Optional<CartItem> cartItemResult = cartItemRepository.findById(cino);
    CartItem cartItem = cartItemResult.orElseThrow();
    cartItem.changeQty(qty);
    cartItemRepository.save(cartItem);
    return getCartItems(email);
  }
  //장바구니 아이템 번호 cino가 없는 경우
  //사용자의 카트
  Cart cart = getCart(email);
  CartItem cartItem = null;
```

장바구니 서비스 계층의 설계/구현

→ service 패키지내에 CartService와 CartServiceImpl로 구현


```
    //이미 동일한 상품이 담긴적이 있을 수 있으므로
    cartItem = cartItemRepository.getItemOfPno(email, pno);

    if(cartItem == null){
      Product product = Product.builder().pno(pno).build();
      cartItem =
CartItem.builder().product(product).cart(cart).qty(qty).build();

    }else {
      cartItem.changeQty(qty);
    }

    //상품 아이템 저장
    cartItemRepository.save(cartItem);

    return getCartItems(email);
  }
```

장바구니 서비스 계층의 설계/구현

→ service 패키지내에 CartService와 CartServiceImpl로 구현


```java
//사용자의 장바구니가 없었다면 새로운 장바구니를 생성하고 반환
private Cart getCart(String email ){

  Cart cart = null;
  Optional<Cart> result = cartRepository.getCartOfMember(email);
  if(result.isEmpty()) {
    log.info("Cart of the member is not exist!!");
    Member member = Member.builder().email(email).build();
    Cart tempCart = Cart.builder().owner(member).build();
    cart = cartRepository.save(tempCart);
  }else {
    cart = result.get();
  }

  return cart;
}
```

장바구니 서비스 계층의 설계/구현

→ service 패키지내에 CartService와 CartServiceImpl로 구현


```java
@Override
public List<CartItemListDTO> getCartItems(String email) {

  return cartItemRepository.getItemsOfCartDTOByEmail(email);
}
```

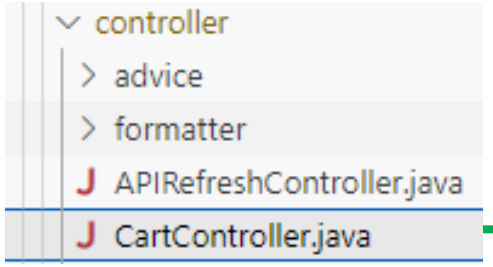
장바구니 서비스 계층의 설계/구현

→ service 패키지내에 CartService와 CartServiceImpl로 구현


```
  @Override
  public List<CartItemListDTO> remove(Long cino) {

    Long cno  = cartItemRepository.getCartFromItem(cino);

    log.info("cart no: " + cno);

    cartItemRepository.deleteById(cino);

    return cartItemRepository.getItemsOfCartDTOByCart(cno);
  }
}
```

컨트롤러 계층과 테스트

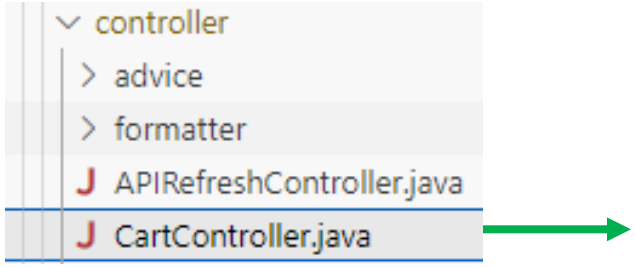
→ CartController는 스프링 시큐리티 관련 기능을 사용하도록 구현




```java
package org.zerock.mallapi.controller;

import org.springframework.web.bind.annotation.*;

import org.zerock.mallapi.service.CartService;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;

@RestController
@RequiredArgsConstructor
@Log4j2
@RequestMapping("/api/cart")
public class CartController {
  private final CartService cartService;
}
```

컨트롤러 계층과 테스트

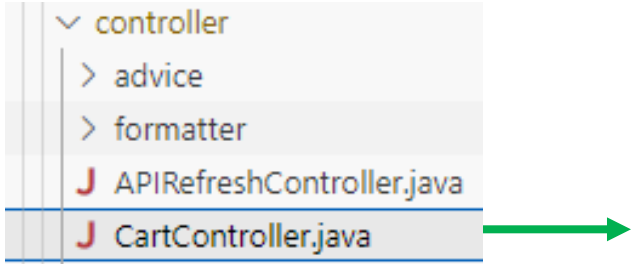
→ 장바구니 아이템의 추가/수정




```java
@RestController
@RequiredArgsConstructor
@Log4j2
@RequestMapping("/api/cart")
public class CartController {

  private final CartService cartService;

  @PreAuthorize("#itemDTO.email == authentication.name")
  @PostMapping("/change")
  public List<CartItemListDTO> changeCart(@RequestBody CartItemDTO itemDTO){

    log.info(itemDTO);
    if(itemDTO.getQty() <= 0) {
      return cartService.remove(itemDTO.getCino());
    }
    return cartService.addOrModify(itemDTO);
  }
}
```

사용자의 장바구니 목록

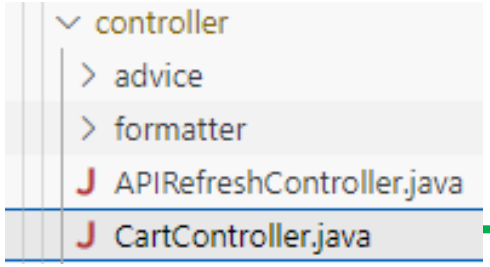
→ '/api/cart/items'로 조회




```java
@PreAuthorize("hasAnyRole('ROLE_USER')")
@GetMapping("/items")
public List<CartItemListDTO> getCartItems(Principal principal) {

  String email = principal.getName();
  log.info("-----------------------------------------");
  log.info("email: " + email );

  return cartService.getCartItems(email);

}
```

장바구니 아이템의 삭제

→ 장바구니 아이템 번호(cino)를 이용해서 DELETE 방식으로 호출




```java
@PreAuthorize("hasAnyRole('ROLE_USER')")
@DeleteMapping("/{cino}")
public List<CartItemListDTO> removeFromCart(
        @PathVariable("cino") Long cino
){

  log.info("cart item no: " + cino);

  return cartService.remove(cino);

}
```

감사합니다.