+ Lớp src/main/java/controller/AuthenticationController.java

```
/*
Control Coupling
 Trong phương thức getMainUser(), nếu thời gian hết hạn (SessionInformation.expiredTime) đã qua thì nó sẽ gọi phương thức logout().
 Đây là control coupling vì nó kiểm soát cả quyết định logout trong trường hợp thời gian hết hạn.
 */
public User getMainUser() throws ExpiredSessionException {
        if (SessionInformation.mainUser == null || SessionInformation.expiredTime == null || SessionInformation.expiredTime.isBefore(LocalDateTime.now())) {
            logout();
            throw new ExpiredSessionException();
        } else return SessionInformation.mainUser.cloneInformation();
    }
```

+ Lớp src/main/java/controller/BaseController.java

```
/*
Control Coupling
Class này chứa các phương thức mà sử dụng trực tiếp đối tượng SessionInformation.cartInstance
 để thực hiện các tác vụ liên quan đến giỏ hàng (cart). Điều này tạo ra sự phụ thuộc giữa BaseController
 và đối tượng SessionInformation.cartInstance, và BaseController đang điều khiển hoạt động của SessionInformation.cartInstance.
 */
public class BaseController {
```

```
    /**
     * The method checks whether the Media in Cart, if it were in, we will return the CartMedia else return null
     * @param media
     * @return CartMedia or null
     */
    public CartItem checkMediaInCart(Media media){
        return SessionInformation.cartInstance.checkMediaInCart(media);
    }
    /**
     * This method gets the list of items in cart
     * @return List[CartMedia]
     */
    public List getListCartMedia(){
        return SessionInformation.cartInstance.getListMedia();
    }
}
```

+ Lớp src/main/java/controller/PaymentController.java

```
/*
Content Coupling
PaymentController phụ thuộc vào cấu trúc nội dung của các exception trong package common.exception
để xử lý các trường hợp ngoại lệ trong quá trình thanh toán.
 */
public class PaymentController extends BaseController {


        /**
         * Represent the card used for payment
```

```
     */
    private CreditCard card;
    /**
     * Represent the Interbank subsystem
     */
    private InterbankInterface interbank;
    /**
     * Validate the input date which should be in the format "mm/yy", and
then
     * return a {@link String String} representing the date in the
     * required format "mmyy" .
     *
     * @param date - the {@link String String} represents the input date
     * @return {@link String String} - date representation of the required
     *         format
     * @throws InvalidCardException - if the string does not represent a valid
date
     *                              in the expected format
     */
    private String getExpirationDate(String date) throws
InvalidCardException {
        String[] strs = date.split("/");
        if (strs.length != 2) {
            throw new InvalidCardException();
        }
        String expirationDate = null;
        int month = -1;
        int year = -1;
```

```
            try {
                month = Integer.parseInt(strs[0]);
                year = Integer.parseInt(strs[1]);
                if (month < 1 || month > 12 || year <
Calendar.getInstance().get(Calendar.YEAR) % 100 || year > 100) {
                    throw new InvalidCardException();
                }
                expirationDate = strs[0] + strs[1];
            } catch (Exception ex) {
                throw new InvalidCardException();
            }
            return expirationDate;
        }
        /**
         * Pay order, and then return the result with a message.
         *
         * @param amount         - the amount to pay
         * @param contents       - the transaction contents
         * @param cardNumber     - the card number
         * @param cardHolderName - the card holder name
         * @param expirationDate - the expiration date in the format "mm/yy"
         * @param securityCode   - the cvv/cvc code of the credit card
         * @return {@link Map Map} represent the payment result with a
         *         message.
         */
        public Map<String, String> payOrder(int amount, String contents, String
cardNumber, String cardHolderName,
                String expirationDate, String securityCode) {
```

```
                Map<String, String> result = new Hashtable<String, String>();
                result.put("RESULT", "PAYMENT FAILED!");
                try {
                        this.card = new CreditCard(
                                        cardNumber,
                                        cardHolderName,
                                        getExpirationDate(expirationDate),
                                        Integer.parseInt(securityCode));
                        this.interbank = new InterbankSubsystem();
                        PaymentTransaction transaction = interbank.payOrder(card,
amount, contents);
                        result.put("RESULT", "PAYMENT SUCCESSFUL!");
                        result.put("MESSAGE", "You have successfully paid the
order!");
                } catch (PaymentException | UnrecognizedException ex) {
                        result.put("MESSAGE", ex.getMessage());
                }
                return result;
        }
        public void emptyCart(){
      SessionInformation.cartInstance.emptyCart();
   }
}
```

+ Lớp src/main/java/controller/PlaceOrderController.java

```
/*
Content coupling
PlaceOrderController phụ thuộc vào các ngoại lệ như
InvalidDeliveryInfoException,
```

cũng như các lớp khác như DistanceCalculator, SessionInformation, và DeliveryInfo để thực hiện các chức năng của nó.

*/

```
public DeliveryInfo processDeliveryInfo(HashMap info) throws
InterruptedException, IOException, InvalidDeliveryInfoException {
        LOGGER.info("Process Delivery Info");
        LOGGER.info(info.toString());
        validateDeliveryInfo(info);
        DeliveryInfo deliveryInfo = new DeliveryInfo(
                String.valueOf(info.get("name")),
                String.valueOf(info.get("phone")),
                String.valueOf(info.get("province")),
                String.valueOf(info.get("address")),
                String.valueOf(info.get("instructions")),
                new DistanceCalculator());
        System.out.println(deliveryInfo.getProvince());
        return deliveryInfo;
    }


    /**
   * The method validates the info
   * @param info
   * @throws InterruptedException
   * @throws IOException
   */
    public void validateDeliveryInfo(HashMap<String, String> info) throws
InterruptedException, IOException, InvalidDeliveryInfoException {
        if (validatePhoneNumber(info.get("phone"))
```

```
    || validateName(info.get("name"))
    || validateAddress(info.get("address"))) return;
    else throw new InvalidDeliveryInfoException();
}

public boolean validatePhoneNumber(String phoneNumber) {
    if (phoneNumber.length() != 10) return false;
    if (!phoneNumber.startsWith("0")) return false;
    try {
        Integer.parseInt(phoneNumber);
    } catch (NumberFormatException e) {
        return false;
    }
    return true;
}

public boolean validateName(String name) {
    if (Objects.isNull(name)) return false;
    String patternString = "^[a-zA-Z\\s]*$";
    Pattern pattern = Pattern.compile(patternString);
    Matcher matcher = pattern.matcher(name);
    return matcher.matches();
}

public boolean validateAddress(String address) {
    if (Objects.isNull(address)) return false;
    String patternString = "^[a-zA-Z\\s]*$";
```

```
        Pattern pattern = Pattern.compile(patternString);
        Matcher matcher = pattern.matcher(address);
        return matcher.matches();
    }
}
```

+ Lớp src/main/java/controller/ViewCartController.java

```
/*
Common coupling
Class này phụ thuộc vào SessionInformation để truy cập vào giỏ hàng hiện tại (SessionInformation.cartInstance),
mà không có sự phụ thuộc vào các thành phần cụ thể khác ngoài SessionInformation.
 */
public class ViewCartController extends BaseController{

    /**
     * This method checks the available products in Cart
     * @throws SQLException
     */
    public void checkAvailabilityOfProduct() throws SQLException{
        SessionInformation.cartInstance.checkAvailabilityOfProduct();
    }
    /**
     * This method calculates the cart subtotal
     * @return subtotal
     */
    public int getCartSubtotal(){
        int subtotal = SessionInformation.cartInstance.calSubtotal();
```

```
        return subtotal;
    }
}
```

+ Lớp src/main/java/utils/ApplicationProgrammingInterface.java

```
/*
Common coupling
Class này đóng gói các phương thức để tương tác với các API thông qua HTTP.
Các phương thức này không phụ thuộc vào bất kỳ lớp hoặc module cụ thể nào trong hệ thống,
mà chỉ tương tác với API bên ngoài thông qua các yêu cầu HTTP.
nó không phụ thuộc vào các thành phần cụ thể trong hệ thống mà chỉ tương tác với các thành phần ngoại vi.
 */
public class ApplicationProgrammingInterface {


        public static DateFormat DATE_FORMATTER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        private static Logger LOGGER = Utils.getLogger(Utils.class.getName());
        public static String get(String url, String token) throws Exception {
                LOGGER.info("Request URL: " + url + "\n");
                HttpURLConnection conn = setupConnection(url);
                conn.setRequestMethod("GET");
                conn.setRequestProperty("Authorization", "Bearer " + token);
                BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
                String inputLine;
                StringBuilder respone = new StringBuilder(); // ising StringBuilder for the sake of memory and performance
                while ((inputLine = in.readLine()) != null)
```

```
                System.out.println(inputLine);
            respone.append(inputLine + "\n");
            in.close();
            LOGGER.info("Respone Info: " + respone.substring(0,
respone.length() - 1).toString());
            return respone.substring(0, respone.length() - 1).toString();
        }
        public static String post(String url, String data) throws IOException {
            allowMethods("PATCH");
            HttpURLConnection conn = setupConnection(url);
            conn.setRequestMethod("PATCH");
            String payload = data;
            LOGGER.info("Request Info:\nRequest URL: " + url + "\n" +
"Payload Data: " + payload + "\n");
            Writer writer = new BufferedWriter(new
OutputStreamWriter(conn.getOutputStream()));
            writer.write(payload);
            writer.close();
            BufferedReader in;
            String inputLine;
            if (conn.getResponseCode() / 100 == 2) {
                in = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
            } else {
                in = new BufferedReader(new
InputStreamReader(conn.getErrorStream()));
            }
            StringBuilder response = new StringBuilder();
            while ((inputLine = in.readLine()) != null)
```

```
                response.append(inputLine);
            in.close();
            LOGGER.info("Respone Info: " + response.toString());
            return response.toString();
        }
        private static HttpURLConnection setupConnection(String url) throws
IOException {
            HttpURLConnection conn = (HttpURLConnection) new
URL(url).openConnection();
            conn.setDoInput(true);
            conn.setDoOutput(true);
            conn.setRequestProperty("Content-Type", "application/json");
            return conn;
        }
        private static void allowMethods(String... methods) {
            try {
                Field methodsField =
HttpURLConnection.class.getDeclaredField("methods");
                methodsField.setAccessible(true);
                Field modifiersField =
Field.class.getDeclaredField("modifiers");
                modifiersField.setAccessible(true);
                modifiersField.setInt(methodsField,
methodsField.getModifiers() & ~Modifier.FINAL);
                String[] oldMethods = (String[]) methodsField.get(null);
                Set<String> methodsSet = new
LinkedHashSet<>(Arrays.asList(oldMethods));
                methodsSet.addAll(Arrays.asList(methods));
                String[] newMethods = methodsSet.toArray(new String[0]);
```

```
                methodsField.set(null/* static field */, newMethods);
            } catch (NoSuchFieldException | IllegalAccessException e) {
                throw new IllegalStateException(e);
            }
        }
    }
```