Report: AU SAPRK Payment

Subject: CS4402 Object Oriented Concept

Submitted to

Dr. Kwankamol Nongpong

Submitted by

1. Pichayuth Kittisriswai 5713091

2.Thanabodee Bunnuch 5713356

This report is a part of the course Object Oriented Concept in the semester 1/2016

Overview

AU SPARK Payment is a new feature of AU SPARK system. This new feature will allow students to make their payment via AU SPARK application. The Students can pay their tuition fee and other fees such as temporary id card, visa fee and etc. Now, we have 2 payment gateway providers. First, Krungsri Bank and second, mPay. In the future, we may have new payment gateway providers. So, we have to make our system be able to add new payment gateway provider and easy to maintain.

Payments steps

We can separate payment step into 3 main steps.

1. Request step
    - AU SPARK system will get the request from the client(iOS, android and web). The system backend will prepare the data and send to the chosen gateway provider.
2. Payment gateway
    - In this step AU SPARK backend send the request to payment gateway provider and it will redirect to payment gateway provider website and the user will have to follow step by step to complete their payment.
3. Response step
    - After finish step 2, AU SPARK backend will receive payment response from payment gateway provider and our system will have to update payment status and create the receipt for each successful payment.

Current system

We will describe our current system design to make you understand the current system flow.

➢ Request Step

1. For the current system, we must have two functions for each payment gateway to handle tuition fee and other fees. So, now we have two payment gateway providers we have total of 4 functions.

Ex. Tuition fee function of mPay and Krungsri

```
public async Task<ActionResult> mPayTuition(string channel, string studentCode, string amount, string year, string semester)
```

```
0 references
public async Task<ActionResult> krungsriTuition(string channel, string studentCode, string amount, string year, string semester)
```

*Note: there will be another 2 functions for general fees.

2. For the function that we use to create order and make record into database.

```
2 references
public async Task<wsResult> Stamp(string gatewayName, string studentCode, int amount, string year, string semester, string channel)
```

We didn't use any common object handle it.

➢ Response step

For response step we must have separate response function for each payment gateway providers.

Ex. Krungsri

```
0 references
public ActionResult returnProcess(long MERCHANTNUMBER, long ORDERNUMBER, string PAYMENTTYPE, int AMOUNT, int CURRENCY, int AMOUNTEXP10, string LANGUAGE,
    string REF1, string REF2, string REF3, string REF4, string REF5, string RESPONSE, string AUTHCODE, string STATUS)
{
```

Ex. mPay

```
public ActionResult mPayReturn(string status, string respCode, string respDesc, string tranId, string saleId, string orderId, string currency, string exchangeRate
   , string purchaseAmt, string amount, string incCustomerFee, string excCustomerFee, string paymentStatus, string paymentCode, string orderExpireDate,
   string custEmail, string shipName, string shipAddress, string shipProvince, string shipZip, string shipCountry, string remark1, string remark2,
   string integrity, string customerId)
{
```

These two function have separate implementation to handle the response, but it turn out that there are somethings that are common and we didn't handle that part correctly. After receiving the response, we have to update the order status and create a payment receipt.

Ex. Update and create receipt

```
public async Task<wsResult> GetBackground(long MERCHANTNUMBER, long ORDERNUMBER, string PAYMENTTYPE, int AMOUNT, int CURRENCY, int AMOUNTEXP10,
    string LANGUAGE, string REF1, string REF2, string REF3, string REF4, string REF5, int RESPONSE, string AUTHCODE, string STATUS)
{
    wsResult reResult = new wsResult();
    using (AUSPARK2017_PaymentDataContext audb = new AUSPARK2017_PaymentDataContext())
    {
        audb.Connection.Open();
        audb.Transaction = audb.Connection.BeginTransaction();
```

For the update and create receipt function, we get the input parameters as individual parameter, but not all parameters are being used.

New System

➢ Request step

We see that it is a disaster if we will have new payment gateway providers.

We redesigned it and now we have only one function call for tuition and general fees and we also have a common object to handle request payment.

Ex. CallPayment

```
public async Task<ActionResult> CallPayment(string gatewayName, string amount, string channel, string paymenttype,
    string semester, string year, string studentCode,string otherfee)...
```

Ex. Request payment object

```
public class ReceivedPayment
{
    5 references
    public GatewayAttributes.GatewayName GATEWAYNAME { get; set; }
    3 references
    public GatewayAttributes.PaymentType PAYMENTYPE { get; set; }
    9 references
    public string CHANNEL { get; set; }
    5 references
    public string YEAR { get; set; }
    5 references
    public string SEMESTER { get; set; }
    2 references
    public string OTHERFEE { get; set; }
    6 references
    public int AMOUNT { get; set; }

    5 references
    public string STUDENTCODE { get; set; }
```

We use Factory design pattern to handle payment gateway providers. We have abstract class for all payment gateway to inherit and there is only one function to call which is SubmitPayment.

Ex. Abstract PaymentGateway

```
internal abstract class PaymentGateway
{
    3 references
    public abstract Task<VwResult> SubmitPayment(ReceivedPayment receivedPayment);
    4 references
    protected abstract Task<VwResult> CallGatewayApi(SendPayment sendPayment);
}
```

Ex. Factory pattern

```
private async Task<ActionResult> Payment(ReceivedPayment receivedPayment)
{
    var factory = new PaymentGatewayFactory();
    var gateway = factory.CreatePaymentGateway(receivedPayment.GATEWAYNAME);
    var result = await gateway.SubmitPayment(receivedPayment);
```

➢ Response step

For the response step, we still have separate function to handle the response, because it is a configuration between AU SAPRK and payment gateway providers.

Ex. Response functions

```
public ActionResult krungsriConfirm(long MERCHANTNUMBER, long ORDERNUMBER, string PAYMENTTYPE, int AMOUNT,
    int CURRENCY, int AMOUNTEXP10, string LANGUAGE, string REF1, string REF2, string REF3, string REF4,
    string REF5, string RESPONSE, string AUTHCODE, string STATUS)...

[AcceptVerbs(WebRequestMethods.Http.Get, WebRequestMethods.Http.Post)]

0 references
public ActionResult mpayConfirm(string status, string respCode, string respDesc, string tranId, string saleId,
    string orderId, string currency, string exchangeRate
    , string purchaseAmt, string amount, string incCustomerFee, string excCustomerFee, string paymentStatus,
    string paymentCode, string orderExpireDate,
    string custEmail, string shipName, string shipAddress, string shipProvince, string shipZip,
    string shipCountry, string remark1, string remark2,
  string integrity, string customerId)...
```

Now, we have a common object to extract needed information and send it to update and create receipt function, but because there is different response format from different payment gateway providers. We must have some way to handle it, so we have applied Template design pattern to make a response step work in the same way across all payment gateway providers. We have abstract class for each payment gateway provider to inherit and have their own response class.

Ex. Abstract class

```
public abstract class PaymentResponse
{
    protected string standardCompleteResponseCode = "00";
    protected string standardInCompleteResponseCode = "99";
    2 references
    public async Task<wsResult> StartPaymentResponseProcess(BackgroundPayment bg)
    {
        var reBg = MakeStandardResponseFormat(bg);
        var reResult = await CreateAndUpdatePaymentResponse(reBg);
        return reResult;
    }

    3 references
    protected abstract BackgroundPayment MakeStandardResponseFormat(BackgroundPayment bg);

    1 reference
    private async Task<wsResult> CreateAndUpdatePaymentResponse(BackgroundPayment bg)
    {
        var returnObject = await PaymentProvider.CreateDatabaseReceiptRecord(bg);
        return returnObject;
    }
}
```

Ex. Krungsri response

```
public class KrungsriResponse : PaymentResponse
{
    private string completeKrungsriResponseCode = "00";
    private string inCompleteKrungsriResponseCode = "99";
    private string completeKrungsriPaymentStatus = "COMPLETE";
    private string inCompleteKrungsriPaymentStatus = "INCOMPLETE";
    3 references
    protected override BackgroundPayment MakeStandardResponseFormat(BackgroundPayment bg)...
}
```

Ex. mPay response

```
public class mPayResponse : PaymentResponse
{
    private string completeMPayStatus = "S";
    private string completeMPayCode = "0000";
    private string completeMPayPaymentStatus = "SUCCESS";
    private string IncompleteMPayPaymentStatus = "FAIL";
    3 references
    protected override BackgroundPayment MakeStandardResponseFormat(BackgroundPayment bg)...
}
```

Because Krungsri and mPay have different response format and status and key words, so we have to have MakeStandardResponseFormat function to make a standard object(BackgroundPayment) to send to update and create receipt function.

Ex. Common object for payment response

```
public class BackgroundPayment
{
    5 references
    public long ORDERNUMBER { get; set; }
    8 references
    public string RESPONSECODE { get; set; }
    2 references
    public string RESPONSESTATUS { get; set; }
    15 references
    public string PAYMENTSTATUS { get; set; }
    3 references
    public GatewayAttributes.GatewayName PAYMENTGATEWAY { get; set; }
}
```

Ex. Update and create receipt function

```
2 references
public static async Task<wsResult> CreateDatabaseReceiptRecord(BackgroundPayment paymentObj)...
```

Adding new payment gateway providers (current design)

For current design, if we want to add new payment gateway provider, we will have to follow these steps

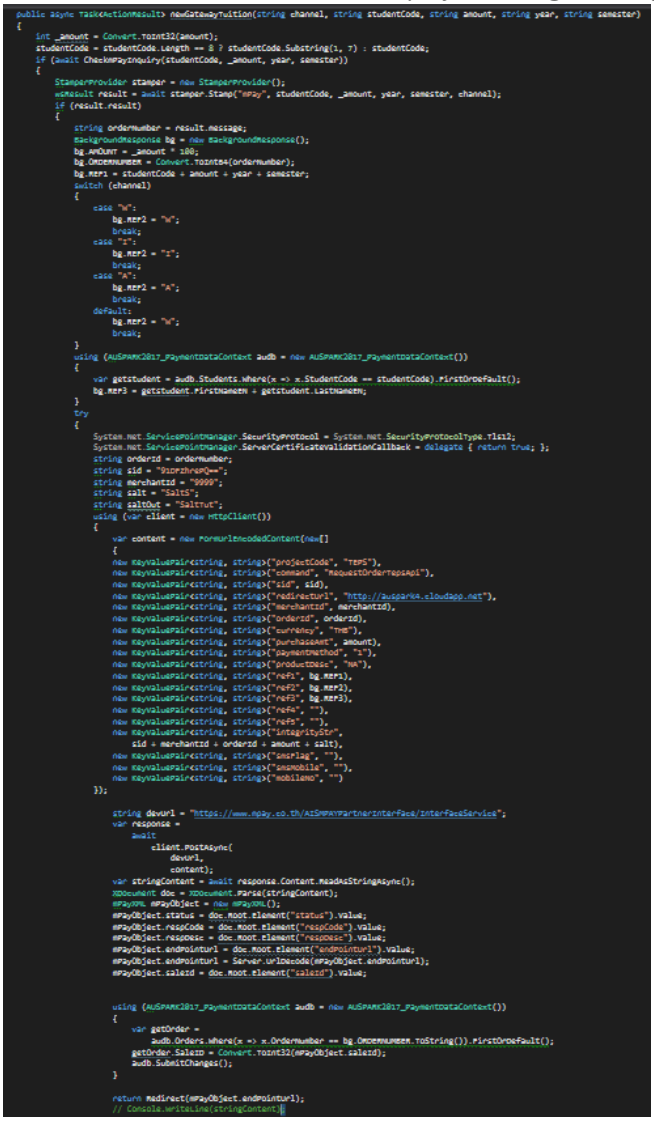
Step: 1 (Make system know there is a new payment gateway provider)

- Adding 2 new functions to PaymentController one for tuition fee and one for general fees.

```
public async Task<ActionResult> newGatewayTuition(string channel, string studentCode, string amount, string year, string semester)...

0 references
public async Task<ActionResult> newGatewayGeneral(string channel, string studentCode, string amount, string year, string semester, string otherFee)...
```

- Implementing the logic in those functions and assign each received parameter to the form that we will send to payment gateway provider.



Step: 2 (Create response function for new payment gateway provider)

- Adding response function for that gateway provider.

```
public ActionResult returnNewGateway(long MERCHANTNUMBER, long ORDERNUMBER, string PAYMENTTYPE, int AMOUNT,
    int CURRENCY, int AMOUNTEXP10, string LANGUAGE, string REF1, string REF2, string REF3, string REF4,
    string REF5, string RESPONSE, string AUTHCODE, string STATUS)
```

- Selecting some response parameters to send to update and create receipt function.

```
public ActionResult returnNewGateway(long MERCHANTNUMBER, long ORDERNUMBER, string PAYMENTTYPE, int AMOUNT,
    int CURRENCY, int AMOUNTEXP10, string LANGUAGE, string REF1, string REF2, string REF3, string REF4,
    string REF5, string RESPONSE, string AUTHCODE, string STATUS)
{
    try
    {
        string ip = Request.UserHostAddress;
        paymenttestController np = new paymenttestController();
        var returnObject = np.GetBackground(ORDERNUMBER, Convert.ToInt32(RESPONSE), STATUS);

        wsResult reResult = new wsResult();;
        reResult.result = false;
```

- Convert all response status to standard status manually with it this function.

```
if (status == "S")
{
    if (respCode == "0000")
    {
        getOrder.UpdatedDateTime = DateTime.Now;
        getOrder.Status = paymentStatus;
        audb.SubmitChanges();
        var temp = new wsResult();
        if (paymentStatus == "SUCCESS")
        {
            temp = CreatePaymentReceipt(getOrder);
        }
        if (getOrder.Status == "PENDING" || getOrder.Status == "AU PENDING")
        {
            if (temp.result)
                reResult.message = getOrder.Status;
            else
                reResult.message = temp.message;
        }
    }
    else
    {
        getOrder.UpdatedDateTime = DateTime.Now;
        reResult.message = getOrder.Status;
```

Adding new payment gateway providers (new design)

For the new design, if we want to add new payment gateway providers, we just have to follow the layout(design) that we have already designed.

Step: 1 (Make system know there is a new payment gateway provider)

- Add new class that inherit PaymentGateway abstract class.

- Implement CallGatewayApi function to make a request to the payment gateway provider.

```
internal class NewGateway : PaymentGateway
{
    private static readonly string newGatewayURL =
        "https://www.newgateway.net/PaymentInput";

    protected override Task<VwResult> CallGatewayApi(SendPayment sendPayment)
    {
        // implementation
        throw new NotImplementedException();
    }
}
```

- Add new payment gateway enum type in class GatewayAttributes.

```
public class GatewayAttributes
{
    15 references
    public enum GatewayName
    {
        Krungsri = 1,
        mPay = 2,
        NewGateway
    }
```

- Add new payment gateway provider class to PaymentGatewayFactory class to handle new payment gateway.

```
internal class PaymentGatewayFactory
{
    1 reference
    public PaymentGateway CreatePaymentGateway(GatewayAttributes.GatewayName gatewayName)
    {
        switch (gatewayName)
        {
            case GatewayAttributes.GatewayName.Krungsri:
                return new Krungsri();
            case GatewayAttributes.GatewayName.mPay:
                return new mPay();
            case GatewayAttributes.GatewayName.NewGateway:
                return new NewGateway();
            default:
                throw new NotSupportedException("This payment gateway is not support.");
        }
    }
```

Step: 2 (Create response function for new payment gateway provider)

- Add new function to PaymentController to handle the response from payment gateway provider.

```
public async Task<ActionResult> NewGateway(long MERCHANTNUMBER, long ORDERNUMBER, string PAYMENTTYPE, int AMOUNT,
  int CURRENCY, int AMOUNTEXP10, string LANGUAGE, string REF1, string REF2, string REF3, string REF4,
  string REF5, string RESPONSE, string AUTHCODE, string STATUS)
```

- Add new class that inherit PaymentResponse abstract class.
- Implement MakeStandardResponseFormat function to convert all received status, response code and payment status to standard format that is used in our system.

```
public class NewGatewayResponse : PaymentResponse
{
    4 references
    protected override BackgroundPayment MakeStandardResponseFormat(BackgroundPayment bg)
    {
        // implementation
        throw new NotImplementedException();
    }
}
```

Comparing current design with new design

As we can see there is only two big step, first we introduce the new payment gateway to the system and second we create a response function.

Step1:

- In this step, we can see that for the current design we didn't use any object and pattern. We just add new function with the whole implementation in it and most of the time the implementation for different payment gateway will have different implementation. But, for the new design there is a pattern and steps to follow which will make developers know where and what they have to add for new payment gateway provider. If we use the current messy design that is no pattern at all the developers will have to read all existing payment gateway methods and try to figure out what they have to do and this take time and most of the time very confused.

Step2:

- In this step, we have to add a response function which will be the same for current design and new design, but for the implementation will be different.
- For the current design, we will first extract some parameters out and convert it to standard format, but what is the standard format? Developers will have to read all existing response function and again try to figure it out what is the standard format and it cost a lot of time. But, for the new design the developers will know that they have to inherit ResponsePayment class and abstract function in the class will automatically tell the developers what they have to do next without figure anything by themselves, because the pattern and steps tell it all.

We can see that with the new design we can understand all the step that we have to do in order to add, delete or update payment gateway, because the pattern say it all unlike the current design that developers will have to figure almost everything by themselves and there is a high chance to get lost in the code and if we have to edit or config something we have to read all the implementation and find that part, because we don't know where that part is.