Jonni Larjomaa

Masters Thesis

Software Development Kit for Internet Payment Gateway Service

Metropolia Ammattikorkeakoulu

Master of Engineering

Information Technology

Master's Thesis

6.5.2016

Metropolia

Author(s) Title Number of Pages Date	Jonni Larjomaa Software Development Kit for Internet Payment Gateway Service 50 pages + 3 appendices 6 May 2016
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Mobile Programming
Instructor(s)	Ville Jääskeläinen, Principal Lecturer Arno Hietanen, Service Manager

In this thesis a Software Development Kit (SDK) was developed which simplifies the integration work of an Internet payment gateway service called the Payment Highway to a PHP based electronic commerce solution.

The Payment Highway is an Internet payment gateway solution developed by a software company called Solinor Oy. The payment gateway acts as middleware for authorizations, credit and debit transactions securely storing and handling the consumers' credit card information.

PHP is a widely adopted web-development programming language. One of the biggest electronic commerce solution, Magento, is written with PHP,. In order to ease the integration process the decision was made to develop a SDK for PHP language.

The SDK developed in this thesis has helped the existing and new customers to build integration faster and efficiently. The quality of the integrations has improved as the improved security features and updates are received through the SDK.

Keywords	PCI-DSS, Payment, SDK, Internet, Gateway, electronic commerce

Metropolia

Table of Contents

1 Introduction 1

2 Payment systems 3

2.1 Key Roles 3
2.1.1 Cardholder 3
2.1.2 Merchant 4
2.1.3 Acquirer 4
2.1.4 Issuer 4
2.1.5 Card Brands 4
2.2 Payment Processor 5
2.3 Payment Gateway 6
2.4 Payment Processing 6
2.5 Security 8

3 Tools and Concepts 12

3.1 HTTP 12
3.1.1 URI-Scheme 12
3.1.2 Methods 13
3.1.3 Headers 14
3.1.4 Status Codes 15
3.1.5 HTTPS 16
3.2 REST 17
3.3 JSON 17
3.4 PHP 18
3.4.1 PSR 18
3.4.2 Composer 19

4 Payment highway 20

4.1 API 21
4.2 Authentication 21
4.3 Form API 23
4.3.1 Add Card 25
4.3.2 Pay with Card 26
4.3.3 Add and Pay with Card 27
4.3.4 Pay with Token and CVC 27
4.4 Payment API 27
4.4.1 Tokenization 29

Metropolia

4.4.2 Commit Form Payment 30
4.4.3 Charge Card 30
4.4.4 Revert Payment 32
4.4.5 Transaction Status 32
4.4.6 Batch Reports 35
4.5 Challenges 36

5 Payment Highway SDK 37

5.1 Architecture 37
5.2 FormAPIService 41
5.3 PaymentAPIService 42
5.4 Installation 45

6 Results and Conclusions 47

References 49

Appendices
Appendix 1. SPH add card with form api flow diagram.
Appendix 2. SPH pay with card form api flow diagram.
Appendix 3. SPH add and pay with card form api flow diagram.

1 Introduction

This chapter first describes the company behind the Payment Highway product. The background of the thesis and why the thesis was carried out are then described. This chapter also includes the research question, the structure of the thesis and the research method.

Solinor is a software company founded in 2002. The name of the company comes from a phrase “Solutions from Innovators”. Solinor develops software and digital network services to all industries and brands utilizing the most modern technologies.

Solinor has a particular experience in the finance and payment industries. Solinor is the first software house in Finland certified to Payment Card Industry Data Security Standard (PCI DSS).

Both in 2013 and 2014 Solinor was the fastest growing tailored software provider in Finland according to Deloitte FAST50 listing. Solinor’s revenue was in order of 3.5 M€ and it had 40 employees at the end of 2014. The company personnel own Solinor fully at the time of writing this thesis.

This thesis project was based on Solinor’s requirement to make the Payment Highway-product deployment and integration much simpler. Payment Highway is a payment gateway application which makes the credit or debit card payments simpler for the electronic commerce products.

In its current state every deployment and integration requires a lot of customer-specific work. Many times the same solutions are built over and over again when integrating the Payment Highway product. Developing and designing an easily approachable and an intuitive Software Development Kit makes the development process of the Payment Highway in a customer-specific project more efficient.

The research problem that this thesis tries to solve can be stated as follow:

”How to develop a Software Development Kit which is feasible, intuitive to use and can be implemented to any project using the Payment Highway.”

This research project was conducted in the following way. The current state of payment systems was studied in order to understand what is the role of the Payment Highway in the payment systems scheme. After the current state analysis the Payment Highway product was researched and analyzed to get a clear view of the requirements for the Software Development Kit. When the requirements were gathered and understood the designing and architecting of the Software Development Kit was conducted. In the last phase of this research the development and publishing of the Software Development Kit was done. Finally, the conclusions were drawn and discussion on how well the thesis answered the research question at hand is also included.

This thesis is constructed in the following manner. The second chapter describes the basic elements and their roles in electronic payment systems and gives background information for the thesis. The third chapter presents the different tools and concepts used in the Payment Highway product and the Payment Highway Software Development Kit creating a deeper technical knowledge of the working domain. The fourth chapter describes the Payment Highway product, why it is used, what kind of features it has and most of all how it operates. In the fifth chapter the development of the Software Development Kit is reported, its architecture is presented and the functionalities are explained. The sixth and the final chapter includes the results, discussions and conclusions of this thesis.

2 Payment systems

Payment systems are a complex collection of multiple instances. This chapter describes the many roles involved in the payment processing of a single transaction when a customer or a cardholder wants to buy goods from a merchant that accepts a payment card as a payment method.

2.1 Key Roles

There are five key roles in the payment card processing industry: a cardholder, a merchant, an issuer, an acquirer and card brands (Visa, MasterCard etc.). In practice there are more participants in the payment transaction processing. In addition to what was listed above, there are also payment processors, payment gateways, software and hardware vendors who facilitate the payment transaction processing. [1]

2.1.1 Cardholder

A cardholder refers to the user of the payment card as they go shopping, buying and swiping the payment card. The cardholders are associated with a PAN (Personal account number). The PAN identifies the cardholder’s account which will be charged and the financial institution that keeps this account. The cardholder is mainly responsible for securing their PIN (Personal identification number) when interacting with the POI (Point of interaction) which is the actual card-reader. The cardholder who has lost his credit card can easily invalidate and renew the credit card by calling to the issuing bank of the payment card. The data stored in the payment card is called the Cardholder data. [1] [2]

2.1.2 Merchant

Shops, markets and e-commerce stores are called merchants. Merchants provide the goods to a cardholder which then uses the payment card to pay the goods. A merchant makes the decision of which payment card brands they accept and which payment methods. Merchants are also responsible for securing the cardholder data. Merchants also format the payment transaction messages and forwards them to an acquiring bank. [1]

2.1.3 Acquirer

An acquirer or acquiring bank authorizes and transmits the transaction settlement to an issuer. The payment processors route transactions to the corresponding acquirer based on the transaction and card type for the authorization and settlement. The acquirer controls the regulated fees and discounts given to a merchant. [1]

2.1.4 Issuer

An issuer or issuing bank manufactures and distributes the payment cards to the cardholders. The issuer also administrates the cardholder data. The issuer authorizes the transactions and deducts the amount of payment needed from the cardholder's account to the acquirer, so the acquirer can pay for the merchant. The issuer is also responsible for the physical security features of the payment cards. [1] [2]

2.1.5 Card Brands

A card brand or card association plays the most central role in the payment industry. They facilitate the whole payment processing and networking between the issuers and the acquirers. Some of the most recognizable card brands are: Visa, MasterCard, American Express, Diners Club. Visa and MasterCard does not build their own networking but rather licenses the use of their brand to the third party operators. American Express on the other hand issues and processes their payment cards. [1] [2]

Card brands have founded the PCI SSC (Payment card industry – security standards council) to maintain the security standards for the merchants to protect the sensitive cardholder data in a secure manner. This security standard is called PCI DSS (Payment card industry – data security standard). [1]

2.2 Payment Processor

A payment processor processes the authorization or the settlement received from the POS (Point of sales) device to the certain acquirer. Routing to the appropriate acquirer is based on the card type and brand. The processor can give further improvements to the payment card data security as a point-to-point encryption and tokenization but does not guarantee it as the processor is usually not situated at the merchant premises. The processor also creates financials reports for the merchant. The processor usually supports multiple types of cards in addition to credit and debit cards. [1] Payment processors for processing different payment methods are shown in Figure 1.

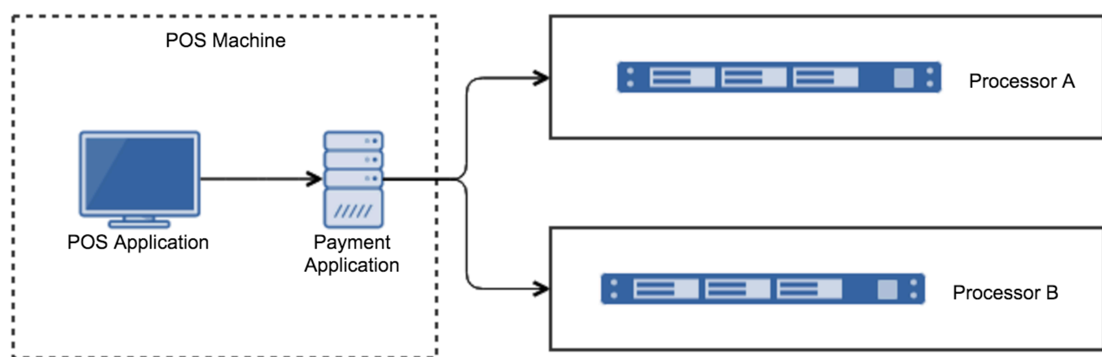


Figure 1. Merchant POS device connected to the payment system

The processor A could handle all the credit card transactions and the processor B handles all the gift card transactions

2.3 Payment Gateway

A payment gateway acts as a middleware solution between the payment application and the payment processors. The payment gateway's primary focus is to unify the connections between the merchant and the payment processor. This helps the merchant to easily transfer from one processor to another to e.g. achieve cost savings. [1] A payment gateway between payment application and payment processor illustrated in Figure 2.

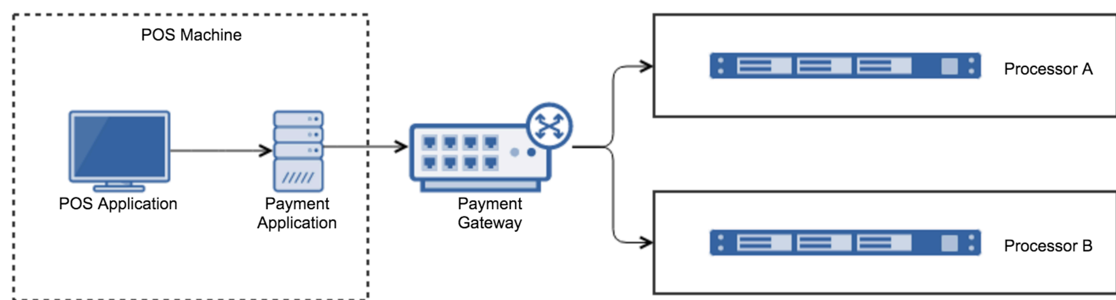


Figure 2. Payment gateway

The main difference between a payment processor and a gateway is that the processors, in addition to the switch functions provided by gateways, also maintain merchant accounts and facilitate settlement process. The payment gateway also offers other services such as point-to-point encryption, centralizing financial reports, tokenization and more. These services helps to secure the transactions but does not effect on the merchant responsibility to secure the Cardholder sensitive information. [1]

2.4 Payment Processing

Payment processing is the workflow starting from swiping a credit card at the POS (Point-of-sales) terminal and going through the whole payment network. Payment processing can be broken down to two main stages: the authorization and the settlement.

The authorization is first the stage where the payment card transaction is either allowed or denied based on whether the Cardholder bank account has enough funds or not. The authorization can fail for other reasons also, such as if the payment card is reported stolen, lost or the payment card has expired. The authorization flow is shown in Figure 3.

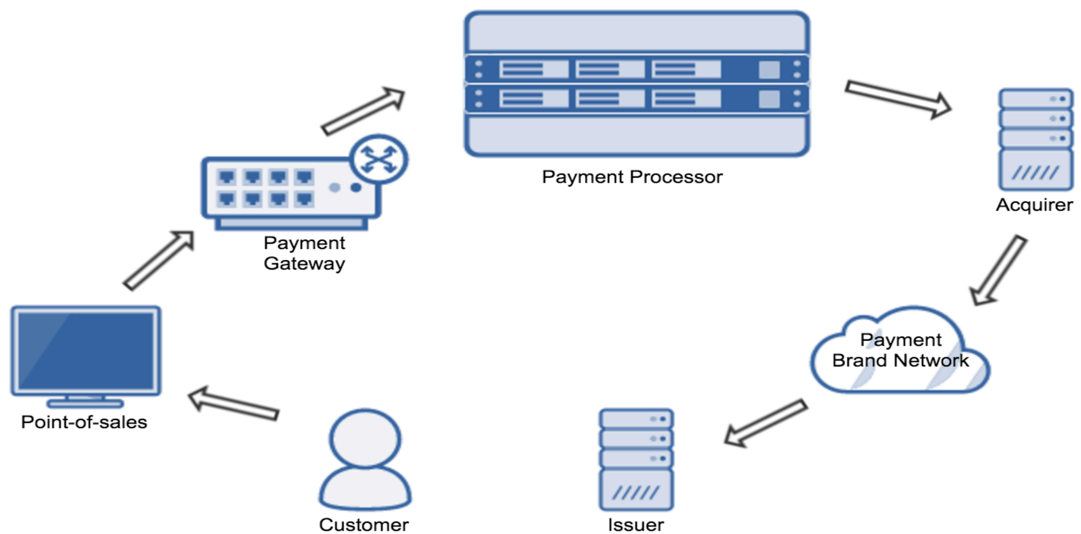


Figure 3. Authorization flow

The authorization request starts when the customer enters the payment card in the card reader. If the card has multiple payment methods, the card reader usually asks from the customer which method to use e.g. credit or debit. When the method has been selected and PIN is entered to the payment application the application sends the authorization request to the payment gateway or directly to the payment processor depending on the configuration. The payment processor then sends the request to the appropriate acquirer which then communicates with the proper issuer for the approval. [2]

If the method selected is debit, then the issuer checks whether the customer has the requested amount of money in his bank account and reserves the amount from that account. If the method selected is credit the issuer checks whether the customer has enough credit to pay the full amount requested. In case of insufficient funds, the authorization request is denied. If the customer has enough credit or money in the bank account, the authorization request is accepted. [1]

The authorization stage is the most vulnerable from the security point of view as all the sensitive Cardholder information needs to be sent over the network. This is the stage where most of the security violations happen. [1]

After a successful authorization the transaction has to be settled between the merchant, the acquirer and the issuer. The settlement flow is shown in Figure 4.

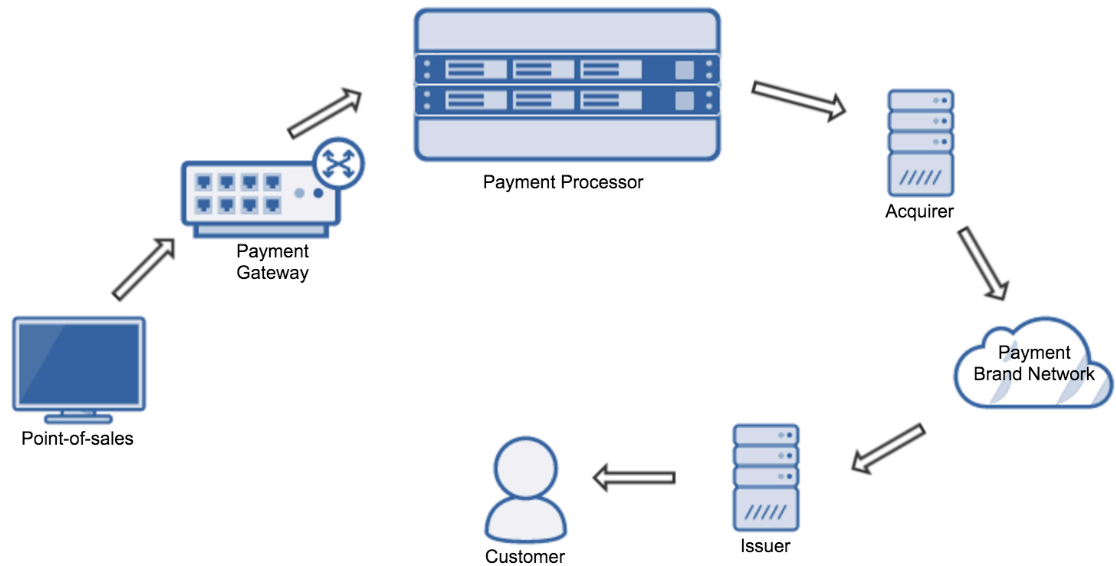


Figure 4. Settlement flow

As shown in Figure 4, the settlement is formed in the merchant's POS payment application and send to the acquirer. The acquirer then settles it with the issuer who then bills the customer and transfers funds to the acquirer who will finally pay to the merchant. The settlement stage is considered less critical from the security point of view as it only deals with a partial Cardholder data.

2.5 Security

Security is the most important part of the payment systems as many parts of the payment systems deal with the cardholder's most vulnerable data. Encryption at all levels is necessary and the use of strong algorithms is a must. To mitigate these problems and to establish well secured systems for handling the cardholder data, card brands have established the Payment Card Industry - Security Standards Council (PCI-SSC). The council is responsible for managing the security standards. The PCI standard compliance is enforced by the founding parties such as MasterCard, Visa, JBC and American Express. [3] The different levels of PCI compliance are shown in Figure 5.

Figure 5. PCI compliance standards [3]

Figure 5 shows the different PCI environment compliances. The PCI PTS (PIN Transaction Devices) includes requirements for the hardware manufacturers of the payment terminals and the cashing systems that deals with the Cardholder data. The PCI PA-DSS (Payment application digital security standard) includes requirements for the terminal and the point-of-sale systems software developers. The PCI DSS (Digital Security Standards) includes requirements for everyone involved in the payment systems: the merchants, the payment gateways, the payment processors, the acquirers and the issuers. All these standard are bound with the P2PE (Point-to-point-encryption) requirement to provide an encryption of sensitive data at all levels. [4] This chapter concentrates on the PCI DSS as the payment highway is under the PCI DSS scope.

The PCI DSS consists of twelve steps that mirror the best practices in the information security as shown in Figure 6.

Goals	PCI DSS Requirements
Build and Maintain a Secure Network and Systems	1. Install and maintain a firewall configuration to protect cardholder data 2. Do not use vendor-supplied defaults for system passwords and other security parameters
Protect Cardholder Data	3. Protect stored cardholder data 4. Encrypt transmission of cardholder data across open, public networks
Maintain a Vulnerability Management Program	5. Protect all systems against malware and regularly update anti-virus software or programs 6. Develop and maintain secure systems and applications
Implement Strong Access Control Measures	7. Restrict access to cardholder data by business need to know 8. Identify and authenticate access to system components 9. Restrict physical access to cardholder data
Regularly Monitor and Test Networks	10. Track and monitor all access to network resources and cardholder data 11. Regularly test security systems and processes
Maintain an Information Security Policy	12. Maintain a policy that addresses information security for all personnel

Figure 6. Twelve steps of PCI compliance [3]

The PCI DSS compliance is achieved by the following steps. The execution of an annual onsite security assessment is done by the QSA (Qualified Security Assessor) and it also reports the compliance. All the twelve steps listed above must be passed before the QSA will grant the compliance. The PCI scoped application network must be quarterly scanned for the vulnerabilities. The publicly facing application interfaces must be penetration tested annually. The scanning and the penetration testing is made by an ASV (Approved Scanning Vendor). [3]

The PCI DSS has been drawn up so that by following the requirements the threats that the merchants and the consumers face in the payment scheme are notably lower. The PCI DSS compliance also affects to the secure data leak aftermath. If a system that has been broken into is certified by PCI DSS, the card brands have insurances for covering the losses of financial damage caused to the merchant or consumer. [4]

The PCI DSS does not deny storing of the cardholder data as described in the requirement number 3. There are limitations though which the data can be stored and which is prohibited and cannot be stored at all. Figure 7 shows which Cardholder data can be stored and how.

Guidelines for Cardholder Data Elements

	Data Element	Storage Permitted	Render Stored Data Unreadable per Requirement 3.4
Cardholder Data	Primary Account Number (PAN)	Yes	Yes
	Cardholder Name	Yes	No
	Service Code	Yes	No
	Expiration Date	Yes	No
Sensitive Authentication Data[1]	Full Track Data[2]	No	Cannot store per Requirement 3.2
	CAV2/CVC2/CVV2/CID[3]	No	Cannot store per Requirement 3.2
	PIN/PIN Block[4]	No	Cannot store per Requirement 3.2

1 Sensitive authentication data must not be stored after authorization (even if encrypted)
2 Full track data from the magnetic stripe, equivalent data on the chip, or elsewhere.
3 The three- or four-digit value printed on the front or back of a payment card
4 Personal Identification Number entered by cardholder during a transaction, and/or encrypted PIN block present within the transaction message

Figure 7. Table of Cardholder data that is allowed or prohibited from storing [3]

The most important thing to notice in Table 7 is that while the cardholder data is permitted to be stored, the primary account number must be strongly encrypted when persisted to the database. The sensitive authentication data cannot be persisted in any circumstances. This causes that even in the Payment Highway product, the payments cannot be automated if the issuer requires the CVC code to be used with every authorization transaction.

3 Tools and Concepts

This chapter describes the tools and concepts used in the Payment Highway product and the Payment Highway SDK. The Payment Highway product is built on top of the HTTP (Hypertext Transfer Protocol) with the REST (Representational State Transfer) architecture using JSON (Javascript Object Notation) as the messaging format. The Payment Highway SDK is built with the PHP (PHP: Hypertext Preprocessor) programming language and uses PSR (PHP Standard Recommendations) as the best practices. The SDK is distributed and managed with a tool called Composer.

3.1 HTTP

The Payment Highway operates with the HTTP (Hypertext Transfer Protocol) or precisely with the HTTPS (HTTP Secure). The HTTP is a stateless application transfer protocol with a simple request / response model. This allows the hiding of the actual implementation of the software with well defined data structures on the transfer layer. The separation of the client and server is only matter of definition. The server can receive a request from the client and then act as a client itself. This kind of behaviour is usually called “proxying” but shows that the role is just a matter of definition at the specific time. [5]

3.1.1 URI-Scheme

The URI-scheme (Uniform Resource Identifier) is used to identify the requested resources. The URI-scheme is also used to indicate the redirects and the resource relationships. [5] An example of an URI is as follows:

http://www.host.com/path/to/resource?queryparam

The above example demonstrates the common parts of a HTTP-URI. Starting with the scheme definition http followedby two slashes which separate the scheme from the authority part. The host part identifies the origin of the URI. The path part identifies the resource requested and following with the query parameters. The query parameters are separated from the path by a question mark. [5]

3.1.2 Methods

The HTTP protocol has verbal request methods to differentiate between the actions. These methods are expressions of what is needed to be done in order to achieve a successful request for example: “GET”, “POST”, “PUT” and “DELETE”. This helps to unify the usage of HTTP requests for separate purposes. [6] Table 1 shows all the HTTP request methods.

Table 1. HTTP request methods

HTTP Method	**Description**
GET	Fetch current target resource.
HEAD	Fetch the status and headers of target resource
POST	Process the request payload identified by target resource
PUT	Replace or update object in the target resource.
DELETE	Remove all existing resource identified by target.
CONNECT	Establish a tunnel to the identified target resource
OPTIONS	Describe connection options of the target resource.
TRACE	Perform a loop-back test to target resource.

All general-purpose servers must support the methods GET and HEAD. All other methods are optional. If the server does not implement a method listed above the server should response with a status code 501 which stands for not implemented. In the Payment Highway product the most utilized methods are POST and GET. [6]

3.1.3 Headers

The HTTP protocol defines the request and the response header fields. These fields can be used in conjunction with the HTTP method to add more accurate information about the request or the response at hand. The header fields are defined to be key-value pairs separated with the colon ":". [6] Figure 8 display a set of request headers used in a HTTP request.

```
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

Figure 8 Examples of request header key value pairs.

The HTTP headers are usually used as a meta-data. Some services may use header fields to transfer information one example is Payment Highway which uses the header fields to transfer information about the payment transactions.

3.1.4 Status Codes

The HTTP protocol uses status codes that are used to indicate that the request was understand and satisfied. The status code is represented by a three-digit integer from 1XX to 5XX. There are five different categories of these status codes represented by the first digit. [6] Table 2 displays the different HTTP status code classes used in the responses.

Table 2. HTTP Status codes and their meanings

Status code class	**Description**
1XX	Informational statuses, request was received continuing with processing.
2XX	Success statuses, request was received and fulfilled successfully
3XX	Redirections, request received and redirection needs to take action.
4XX	Client side errors, received request included malformed information.
5XX	Server side errors, request could not be received something is wrong with the server.

The status codes in the 2XX class are always used to indicate that the request is successfully handled. The request returning a 3XX class response should always be forwarded to the defined resource to complete the request cycle. The error class response of 4XX means that the client did send data that cannot be processed or understood by the server. The client should not repeat the request that received a 4XX response. The 5XX class response implicates problems on the server. The server application may not have access to the resources that are needed to be able to handle the request successfully. The request can be repeated but after a short timeout period.

3.1.5 HTTPS

HTTPS is a HTTP connection that is operated over a TLS (transport layer security) or a SSL (Secure Sockets Layer) connection. The HTTP connection normally uses a standard port 80 for connections but with the HTTPS enabled that standard port is 443. The Payment Highway uses TLS v1.2 to secure all of its traffic. The trust between two end points is then established using certificates. The Payment Highway uses certificates implementing the EV (extended validation) signature. The EV certificates identify the domain used and the organisation behind the domain. This improves the security and allows users to verify that the site belongs to a validated organisation. [7] Figure 9 displays a valid EV certificate used in the Payment Highway HTTPS service.

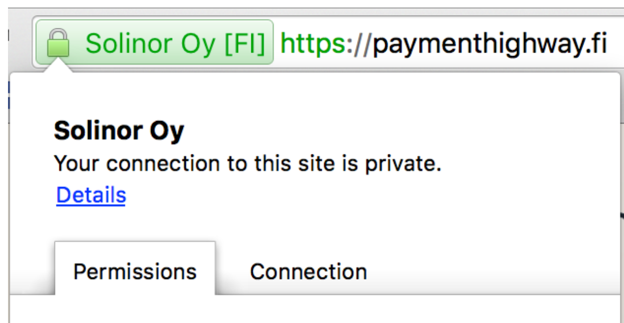


Figure. 9 EV certificate used in paymenthighway service. URI scheme starts with https:// to ensure secure connection is established.

The secure protocols supported by Payment Highway services are TLS 1.2 and SSLv3. By the summer 2018 the only supported protocol will be TLS 1.2 or higher. This requirement comes from the PCI DSS version 3.1.

3.2 REST

The REST (Representational State Transfer) is an architectural design style used with web applications. The key principles in REST are as follows:

1. Provide resources with an unique identifier, URI
2. Link resources with each other, by establishing relationships among resources
3. Use standard methods (HTTP, JSON)
4. The Application state can be represented through resources
5. The communication should be stateless using the HTTP

The REST uses client / server approach to separate the user interaction from the data storage. The REST request are stateless and use a uniform interface. The HTTP protocol verbs are used to present the characteristics of an action. The CRUD-modelled REST services which stands for create, read, update, delete use the HTTP methods as follows (Table 3):

Table 3. CRUD-model method mapping of RESTful services

REST action	HTTP method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

This is the most common approach when designing RESTful services. Many of the web services today implement the characteristics of the REST architecture. The Payment Highway is one the services using these REST architecture implementations. [8]

3.3 JSON

JSON (Javascript Object Notation) is a textual format representing serialized data. JSON is able to present a limited amount of primitive types such as strings, numbers, boolean and null. JSON also has two structured data types: an object and an array. [9] Figure 10 shows a valid JSON object with different data types and primitive type variables.

```
{
  “number”:  10,
  “String”: “Hello world”,
  “Boolean”:  true,
  “Object”: {},
  “ArrayWithInts": [116, 943, 234, 38793],
}
```

Figure 10 A JSON object presenting different JSON types and structures.

The JSON design principles has been to be minimal, portable, textual and a subset of Javascript as defined in the ECMAScript standard. JSON must be UNICODE formatted and by default it is in the UTF-8 encoding. [9]

When JSON is used with the HTTP the Content-type header value must be set to “application/json” so that the requests are understood as a JSON requests. [9]

3.4 PHP

The PHP (PHP: Hypertext Preprocessor) is a free, open source, and general purpose programming language. The PHP is excellent for developing and designing new applications because it is supported by multiple operating systems and web servers. The PHP syntax is following the principles of the other programming languages such as C, PERL and JAVA. The PHP is a dynamic language which gives the flexibility and lowers the learning curve. The PHP is a programming language that is well suited for the web application development. [10]

3.4.1 PSR

PSR (PHP Standards Recommendation) is a set of recommendation how the PHP language commonalities should be used. The PSR is organized by the PHP-FIG (PHP Framework Interop Group) which consists of well established PHP project members. The PSR was first found in 2009 and has evolved ever since. The first PSR was PSR-0 which was a autoloading standard. Now the PSR has around 13 standards where 6 of them has been already accepted and the rest of them are on a draft stage. The standard include recommendations for autoloading, interfaces and coding styles. [11]

3.4.2 Composer

The Composer is a distribution management tool developed for the PHP. Many of the projects consists of several libraries or packages. Normally these libraries are downloaded and managed manually which can become a very overwhelming task if there are many libraries. [12]

The Composer solves this problem by introducing a simple JSON schema by defining a package and its dependencies. The Composer manages these packages per project basis as default, but it can also manage the packages globally. [12]

There has been other package management tools such as PEAR and PECL. The community has chosen the Composer because of its flexibility and ability to do more than just download and install the packages. [13]

This concludes the technology overview used in Payment Highway product and the SDK. All the technologies discussed above are heavily utilized by the Payment Highway product. In the next chapter the Payment Highway product is discussed in more details. The Features of the Payment Highway are presented and how the Payment Highway helps customers and merchants. The technological details and working principles are explained.

4 Payment highway

This chapter describes the Payment Highway product. The Features of Payment Highway are described in detail. The question why Payment Highway is a competitive solution for electronic commerce is answered. The challenges of the product are also discussed as new customers adapt the service.

The Payment Highway product acts as a payment gateway. Payment Highway provides tokenization and secure storing of the CHD (Cardholder data). It also provides a secure way to show a web form where the user can input their Cardholder data. This way it improves the security of the electronic commerce shop and removes the merchant's responsibility to go through the complex task of the PCI DSS assessment. The merchant is still responsible for using this service in a secure manner. Payment Highway supports many acquirers and payment processors enabling the merchant to easily choose the cheapest solution.

The CHD input form service was designed to be highly customable HTML (Hypertext markup language) web form. The form was made with a responsive design and the customer can have their branding implemented on top of the form. Because of this the Payment Highway can be easily integrated to existing electronic commerce services. [8]

The tokenization service is a key component of the Payment Highway product. In the Payment Highway this means that the Cardholder data is securely stored and then substituted with a random hash representation identifying the card in the database. [1] The hash is then sent back to the merchant for storing and identifying the customer for later purchases. Weather the merchants service becomes compromised all the tokens belonging to a merchant can be revoked and regenerated. This manoeuvre makes the old tokens unusable and keeps the CHD in better safe.

4.1 API

The HTTP API (Application programming interface) was split into a two separate APIs, the Form API and the Payment API. The Form API allows the merchants to tokenize and make payments through the html rendered form. The Payment API operates only on the REST requests and responses with a JSON formatting.

The APIs are not completely independent from each other as actions made from the form API might need the payment API interaction to complete the payment or to retrieve the card token. [14]

4.2 Authentication

In order to talk with the HTTP API every request needs an authentication signature. This signature is used to verify the integrity and the validity of the request.

The authentication hash value is calculated from the authentication string using the chosen merchant secret key. The authentication string is formed from the request method, URI and the request parameters beginning with “sph-”-prefix. Then the values are trimmed and the key-value pairs are concatenated in alphabetical order by the key name. The parameter keys must be in lowercase. Each key and value is separated with a colon (“:”) and the different parameters are separated with a new line (“\n”) at the end of each value. [14] An example of a form API request data included in the calculation is shown in Figure 11.

```
POST
/form/view/pay_with_card
sph-account:test
sph-amount:990
sph-cancel-url:https://merchant.example.com/payment/cancel
sph-currency:EUR
sph-failure-url:https://merchant.example.com/payment/failure
sph-merchant:test_merchantId
sph-order:1000123A
sph-request-id:f47ac10b-58cc-4372-a567-0e02b2c3d479
sph-success-url:https://merchant.example.com/payment/success
sph-timestamp:2014-09-18T10:32:59Z
```

Figure 11. Request data included in signature calculation [14]

The calculation is made with the HMAC-SHA256 algorithm. The signature is composed with SPH1 prefix, key id and hash result of the calculation. The values are separated with a space. The signature when formed as a request header is shown in Figure 12.

```
POST
/form/view/pay_with_card
sph-account=test
sph-merchant=test_merchantId
sph-order=1000123A
sph-request-id=f47ac10b-58cc-4372-a567-0e02b2c3d479
sph-amount=990
sph-currency=EUR
sph-timestamp=2014-09-18T10:32:59Z
sph-success-url=https://merchant.example.com/payment/success
sph-failure-url=https://merchant.example.com/payment/failure
sph-cancel-url=https://merchant.example.com/payment/cancel
language=fi
description=Example payment of 10 balloons á 0,99EUR
signature=SPH1 testKey 960aeec47d172637325b15513b3a526e95c93ba74b5067da766f2825734
64d58
```

Figure 12. Full request data with a calculated signature [14]

After the signature is successfully attached to the request headers the request is ready to be sent over to the Payment Highway service. The Payment Highway service then uses the signature value to validate and authenticate the request. Based on the authentication result the request is either denied or handled successfully.

4.3 Form API

The form API has four different methods that can be used for adding the card to the Payment Highway, paying with the card one time, paying and adding the card to the Payment Highway or paying with the card stored in the Payment Highway. All of the actions display a html rendered form for the customer to enter their Cardholder data. All of the request parameters are sent and received as HTTP header parameters. Table 4 displays the HTTP headers that are needed to make a successful Form API call.

Table 4. Custom HTTP request headers used when communicating with Form API

Header	Actions	Description
sph-account	all	string typed account identifier (account can have multiple merchants)
sph-merchant	all	string typed merchant identifier
sph-amount	payment actions	amount in the lowest currency unit. E.g “1000”, meaning 10€ (if euros used).
sph-currency	payment actions	currency code e.g “EUR”
sph-order	payment actions	merchant defined order identifier. Should be unique for every transaction.
sph-success-url	all	Success URI, user is redirected to this URI after successful event
sph-failure-url	all	Failure URI, redirection to this URI after failure.
sph-cancel-url	all	Cancel URI, redirection to this URI after cancellation of operation.
sph-request-id	all	UUIDv4 formatted string to identify the request.

sph-timestamp	all	Request timestamp in ISO8601 combined date and time in UTC. “2025-09-18T10:32:59Z”
sph-token	token payment action	The card token which to be charged.
sph-accept-cvc-request	add card action	Allow adding a card even if it requires CVC for payments. Defaults to false.
language	all except token payment action	Two letter language code in ISO 639-1. E.g. “FI”, “EN”, “RU”.
description	payment actions	description of the payment, E.g. “shoppingcart97324”
signature	all	Authetication signature.

The Form API requests must be made with the HTTP POST method and the character set must be set to the UTF-8. The responses are redirected to a defined failure, cancel and success scenario URLs set in the request headers.

The Form API response headers include information about the success or failure of the API request. The response headers include action specific response values. Below are listed the common header values used in all of the responses (Table 5). [14]

Table 5. Common response headers from form actions

Header	Description
sph-account	Account identifier
sph-merchant	Merchan identifier
sph-request-id	UUIDv4 formatted request id, same when request is send to Form API
sph-timestamp	ISO-8601 formatted timestamp
sph-success	When request is successful this header is added with static text “OK”

sph-failure	When request fails, this header is added, with following possible values as message: • UNAUTHORIZED • INVALID • FAILURE • NO_ROUTE No route means that the merchant is not able to receive payments of certain card brand e.g. American Express.
sph-cancel	When request has been cancelled, e.g. user presses cancel button in form. Has static "CANCEL" text as value.
signature	Authentication signature.

Depending of the success of the request not all above listed headers are received. The sph-success, sph-failure and sph-cancel headers are used with corresponding status. The sph-success header is included in the response when the request is successful, the sph-failure is added when the request has failed and the sph-cancel header is included when the user has chosen to cancel the payment process when the form is displayed on the web-site.

4.3.1 Add Card

The add card action is made with a request to the */form/view/add_card* URI. The inserted Cardholder data is stored to the Payment Highway system and the response is sent back to the merchant. In the response the token id is also sent back to the merchant in order for the merchant to retrieve the token for later usage. No payment is made with this action. See Appendix 1. Table 6 introduces the custom header used in addition to the common headers used when receiving the response from the Payment Highway Form API.

Table 6. Custom header added in successful add card action addition to common response headers

Header	Description
sph-tokenization-id	The token id used in payment API to retrieve the actual token.

The sph-tokenization-id header includes the tokenization identifier used to retrieve the actual token with the Payment API. The tokenization identifier is always a random value but when used to retrieve the actual token from the Payment API it always returns the same token in the response.

4.3.2 Pay with Card

The pay with card action is made with a request to the */form/view/pay_with_card* URI. This action allows the payment of a single transaction. No cardholder data is stored to the system. This gives the ability for the customer to only pay for what is in the shopping cart and not storing the cardholder data. The response includes the transaction id of the payment which is then used by the Payment API to commit the transaction. See Appendix 2. Table 7. introduces the custom headers in addition to the common headers used when receiving the response from the Payment Highway Form API.

Table 7. Custom header added in successful pay with card action addition to common response headers

Header	Description
sph-amount	amount as the lowest current currency.
sph-currency	currency code e.g. “EUR”
sph-transaction-id	Transaction id used to commit transaction with payment API.
sph-order	merchant defined order identifier.

The transaction id received in the response can be used later to receive the transaction status from the Payment API. The transaction identifier must be used when reverting payment with the Payment API.

4.3.3 Add and Pay with Card

The add and pay with the card action is made with a request to the /form/view/add_and_pay_with_card URI. This action then enables the paying and storing of the card information with the Payment Highway. The response includes the transaction id to commit the payment. When the commit is done the payment API returns the token id for retrieval of the token. The response headers are exactly like the headers in the pay with card action. See Appendix 3.

4.3.4 Pay with Token and CVC

The pay with token and CVC (Card Verification Code) action is made with a request to the /form/view/pay_with_token_and_cvc URI. This action is called when the payment card is already stored to Payment Highway and the card is tokenized. The method exits because the CVC field is not allowed to be stored according to the PCI DSS. The feature is developed to handle situations where the issuer requires every authorization to include the CVC field. Providing the token and displaying the form with only the CVC field lowers the amount of the input data needed and speeds up the future payments in the merchant store.

4.4 Payment API

The Payment API has actions that complement the requests made from the Form API. These actions are used to actually commit the payment, and retrieve the card token with token id. The Payment API includes actions that allow purchasing, reverting of the transaction, fetching the status of a single transaction and fetching of the daily batch report. The Payment API actions are made with GET or POST HTTP methods depending wheatear the request has payload or not. The character encoding of all the requests must be set to the UTF-8.

All the responses from the Payment API include a result JSON object. In this object there are two fields, the response code and the response message. The JSON listed below shows an example of a result object.

```
{
  "result":
  {
    "code":100,
    "message":"OK"
  }
}
```

The result object indicates the many states that a response can have. Figure 13 displays all the response codes and relative messages with description in a table format.

RCODE	RMSG	Description, actions
100	Description of the succesful request	Request successful.
200	Reason for the failure	Authorization failed, unable to create a Debit transaction or Revert failed. For Debit transactions, please initialize a new transaction the next day (in case there was insufficient funds) and/or contact the cardholder. For transaction reverts, please see the status of the transaction with `GET /transaction/<id>`
209	Description of the timeout	Charge/Revert failed due to acquirer or issuer timeout.
210		Transaction already fully reverted: A full amount revert was performed on an already fully reverted transaction.
211		Insufficient balance for the revert: A partial revert with an amount greater than the amount left was performed on the transaction.
250	"Suspected fraud"	The transaction was rejected due to suspected fraud.
300		Transaction in progress. Given when there is already a debit transaction being processed with the ID.
900	Description of the error	Could not process the transaction, please try again.
901	Description of the validation error	Invalid input. Detailed information is in the `message` field.
902	Description and ID of the transcation	Transaction not found. Error is raised if the transaction iD is not found or when the transaction ID is trying to be used for a wrong type of operation.
910	Description of the operation type error	Invalid operation type. Either a credit transaction is performed on a debit ID or vice versa.
920	Description of the erroneous request parameters.	Unmatched request parameters. Request parameters do not match the previous parameters with the current transaction ID.
940	Route not found error	The transaction did not match any of the merchant's acquirer routing rules and thus is not allowed.
950	"The desired token already exists"	The card is already tokenized with the existing token.
990	Description of the error	Permanent failure. Cannot repeat the transaction. Please initialize a new transaction.

Figure 13. Response codes and messages of result object

After receiving the result from the Payment API the result object should always be checked for result code 100. If the received result object includes any other code than 100 the response should be handled as failed and closer analysis on the reasons should take action.

4.4.1 Tokenization

When the card is added through the Form APIs add card action the token is not received immediately. Instead the identifier of the token is sent back to the merchants application. The Payment API must be used to retrieve the generated token with the received identifier. This is made by issuing an action to the */tokenization/<id>* URI where the <id> is substituted with the correct identifier. The response is then sent back to the merchant with the correct card token information. See Appendix 1. JSON Response of tokenization fetch action is listed below.


```
{
  "card_token":"71435029-fbb6-4506-aa86-8529efb640b0",
  "card":
  {
    "type":"Visa",
    "partial_pan":"0024",
    "expire_year":"2017",
    "expire_month":"11"
  },
  "result":
  {
    "code":100,
    "message":"OK"
  }
}
```


The response object Includes masked card information such as card type, partial PAN, expiration year and expiration month. The response includes also the card token which is a UUIDv4 hash string and result object.

4.4.2 Commit Form Payment

When the payment is done through the Form API with one of the following actions, */form/view/pay_with_card,* */form/view/add_and_pay_with_card* or */form/view/pay_with_token_and_cvc*, the transaction is not sent forward until it is committed by issuing a commit payment action from the Payment API. This brings two benefits; first one can be sure that the payment info was received by merchant and second the payment was only made once. The action is made to the */transaction/<id>/commit* URI where the <id> is substituted with the correct transaction identifier. See Appendices 2 and 3.

4.4.3 Charge Card

The Payment API can also be used to directly charge a card. With this approach the merchant can implement so called one-click payments. This means that no form to input the Cardholder data is sent to the customer. The payment process is done solely on the backend side.

In order to be able to charge a card with the Payment API, a transaction identifier must be acquired first. The transaction identifier is the same as the one used when committing the transaction. The handle is received when calling the */transaction* action with the HTTP POST method on the Payment API. In the response the transaction identifier is received. The Transaction identifier must be included in the payment action. The received transaction identifier is a UUID v4 (Universally Unique Identifier) formatted string. The JSON listed below shows the response object received from a successful transaction handle request. [14]


```
{
  "id":"ebf19bf4-2ea7-4a29-8a90-f1abec66c57d",
  "result":
  {
    "code":100,
    "message":"OK"
  }
}
```

The Actual payment is done with the HTTP POST method to the */transaction/<id>/debit* URI where the <id> is substituted with the transaction identifier received from the handler call [14]. Figure 14 displays the transaction request sequence when a debit action is issued.

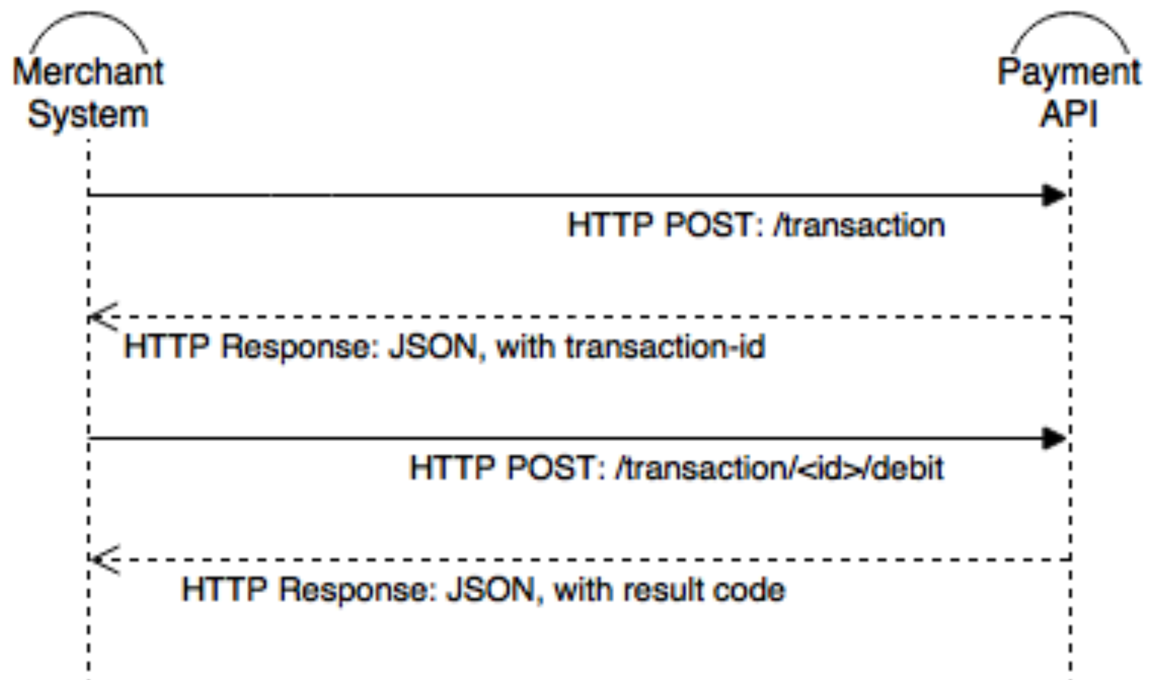


Figure 14. Paying action sequence with payment API

The request has a JSON formatted payload with the amount, currency and the card token information. This limits the usage of the action to be used only with the tokenized cards. Below is listed a valid request JSON to be used as a payload in pay with a token action. Required values are the amount, the currency and the card token.

```
{
  “amount” : 100,
  “currency” : EUR,
  “token” : {
    “id” : “71435029-fbb6-4506-aa86-8529efb640b0”
  }
}
```

After a successful debit action, the Payment API responses with a single result object. This result object includes a result code and message field.

4.4.4 Revert Payment

The payment API allows the system to revert an existing transaction. This is needed when the customer wants to cancel an order made in the merchant store or there has been some other issue with the transaction information e.g. wrong amount. [14] Figure 15 displays the request cycle of the revert action.

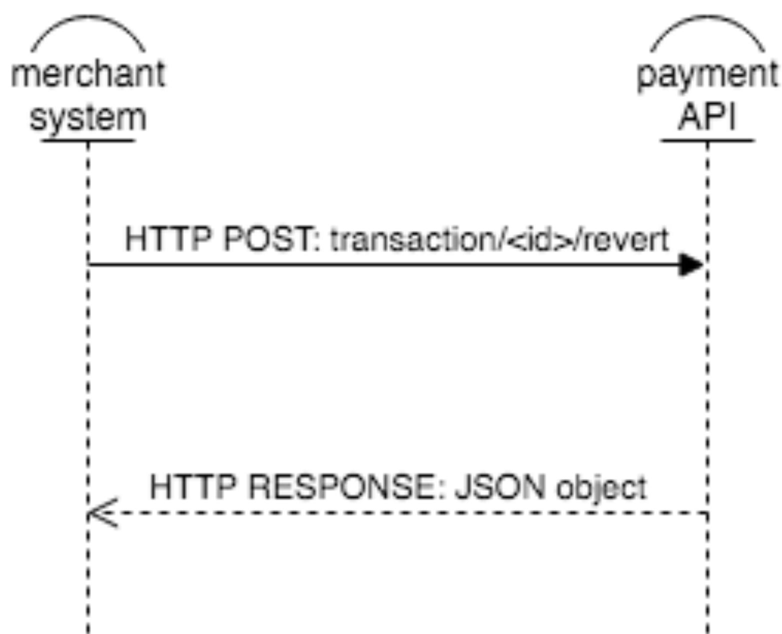


Figure 15 request lifecycle of revert action

Reverting the transaction is made with the following HTTP POST method to the */transaction/<id>/revert* URI where the <id> is substituted with the transaction identifier which is the target of the revert. In the successful event the response of the revert action includes just the result JSON object.

4.4.5 Transaction Status

If the merchant needs the status information of a single transaction it can be retrieved from the Payment API. To be able to retrieve the status information of the transaction an HTTP GET action must be issued to the */transaction/<id>* URI where the <id> is substituted with the transaction identifier which needs to be received. Figure 16 describes the request flow when fetching transaction status from the Payment API. [14]

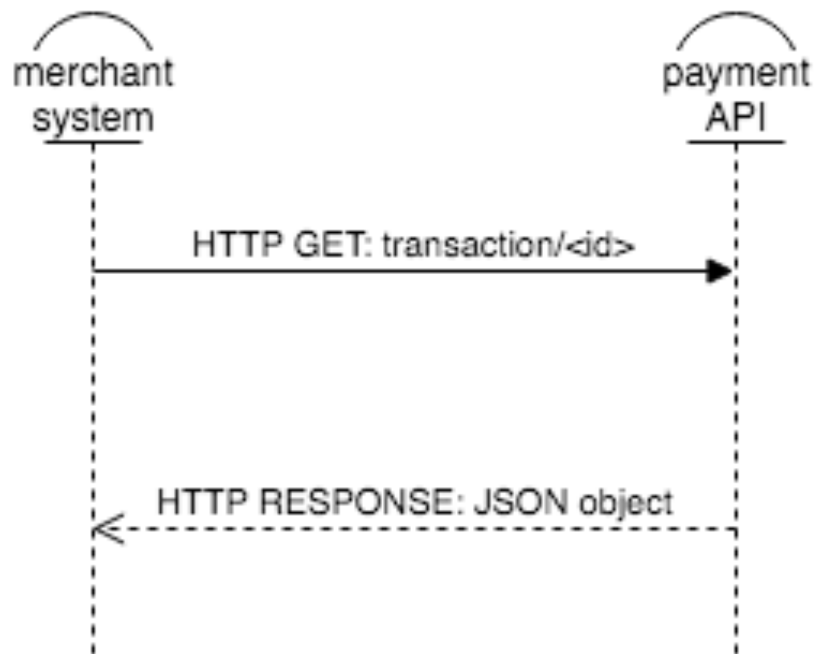


Figure 16. Request cycle in transaction status action

The received JSON object includes full information of the transaction at hand. The status response includes the card information used, the transaction identifier, the transaction status, the acquirer handling the transaction and other relevant meta data. The JSON received from a successful transaction status call is listed below.

```
{
  "transaction":
  {
    "id":"5a457896-1b74-48e2-a012-0f1016c64900",
    "acquirer":
    {
      "id":"nets",
      "name":"Nets"
    },
    "type":"debit",
    "amount":9999,
    "current_amount":9999,
    "currency":"EUR",
    "timestamp":"2015-04-28T12:11:12Z",
    "modified":"2015-04-28T12:11:12Z",
    "filing_code":"150428011232",
    "authorization_code":"639283",
    "status":
    {
      "state":"ok",
      "code":4000
    },
    "card":
    {
      "type":"Visa",
      "partial_pan":"0024",
      "expire_year":"2017",
      "expire_month":"11"
    }
  }
}
```


The transaction object includes the detailed information of the whole transaction. The transaction object includes the following key information: the card which was used in the transaction, the status that the transaction has at the moment and the acquirer used for this transaction.

4.4.6 Batch Reports

If the merchant needs to receive daily reports, a batch report action must be used to retrieve the settlements and transactions from the payment API for the whole day. This is achieved by issuing a HTTP GET request to the */report/batch/<yyyyMMDD>* URI where the *<yyyyMMDD>* part is substituted with the date formatted as required e.g. */report/batch/20160101*. Below is the listing of the response JSON received when a batch report is successfully received.


```
{
  "settlements":[
  {
    "id":"4b961b80-e808-487b-bc89-7cf83ffda4d7",
    "batch":"000017",
    "timestamp":"2015-03-23T22:00:09Z",
    "merchant":
    {
      "id":"test_merchantId",
      "name":"Test user"
    },
    "transaction_count":1,
    "net_amount":620,
    "currency":"EUR",
    "acquirer":
    {
      "id":"nets",
      "name":"Nets"
    },
    "transactions":[],
    "status":
    {
      "state":"ok",
      "code":4000
    },
    "reference":"11503231000000174"
  }],
  "result":
  {
    "code":100,
    "message":"OK"
  }
}
```

The settlement report is generally formed on a daily basis. The settlement report describes the daily transactions made through the Payment Highway. This example has an empty transactions array. The format of a single transaction object in the array is the same as when fetching a single transaction status.

4.5 Challenges

The Payment Highway business model is built as a transactional billing model. The amount of monthly billing is directly proportional to the transactions made in the Payment Highway product. This causes the problem that the longer the integration work takes time the possible income decreases. To mitigate this problem a well structured tool is needed to speed up the integration work.

As seen in this chapter, the Payment Highway has many features and functionalities that needs to be fulfilled before the Payment Highway HTTP API can work seamlessly with the electronic commerce product.

When new customers are introduced to the Payment Highway the amount of custom integration work that needs to be done is very intensive and many routine type of tasks needs to be repeated. There is no easy solution to use APIs as programmatically and Solinor OY required to build a Software Development Kit to lower the amount of work that needs to be done.

5 Payment Highway SDK

In this chapter the architecture of the Software Development Kit (SDK) is explained in detail and the common usage patterns of the SDK are described. The final part of this chapter explains how to deploy the SDK in a new or in an ongoing project.

Before this study the integration of an existing electronic commerce system to the Payment Highway was an exhaustive project and many times the same development work was repeated. The SDK project for the Payment Highway was started to ease the integration work to be done.

A well structured SDK shortens the time to take new customers on board. It also helps to lower the integration work costs and to increase the revenue of the Payment Highway product.

5.1 Architecture

The SDK intuitively mirrors the services and actions in the Payment Highway HTTP API. The SDK validates the input data and handles the complex tasks such as the authentication token calculation, the header information ordering and the HTTP request processing.

This thesis concentrates on the SDK written with the PHP programming language. The minimum PHP version required to run the SDK is PHP 5.4.

The SDK architecture was designed in an implementation independent way, meaning it can be used as a self contained library together with any framework or software written with the PHP language. The architecture was designed to be portable to the other programming languages as well.

The *FormAPIService* has simplest form of the architecture. The *FormAPIservice* has a single responsibility which is to create a form model that includes the correctly formed URI, request headers including a secure signature and the HTTP request method. Figure 17 displays the outlined architecture of the *FormAPIService*. Details of implementation are described later in this chapter.

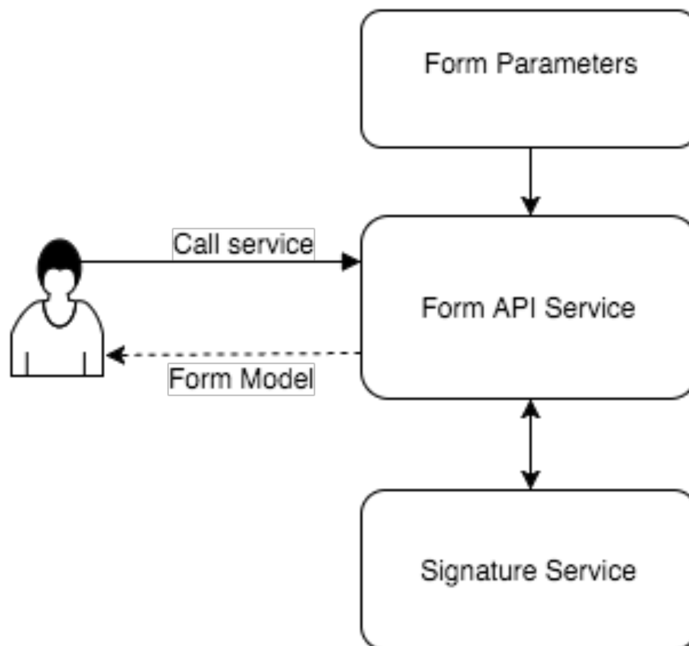


Figure 17. *FormAPIService* architecture

The *PaymentAPIService* has a more complex design compared to the *FormAPIService*. This is due to its wider responsibility as the *PaymentAPIService* handles the HTTP requests to the Payment Highway HTTP API and validates the response from the Payment Highway HTTP API. Figure 18 illustrates the behaviour and design of the *PaymentAPIService*.

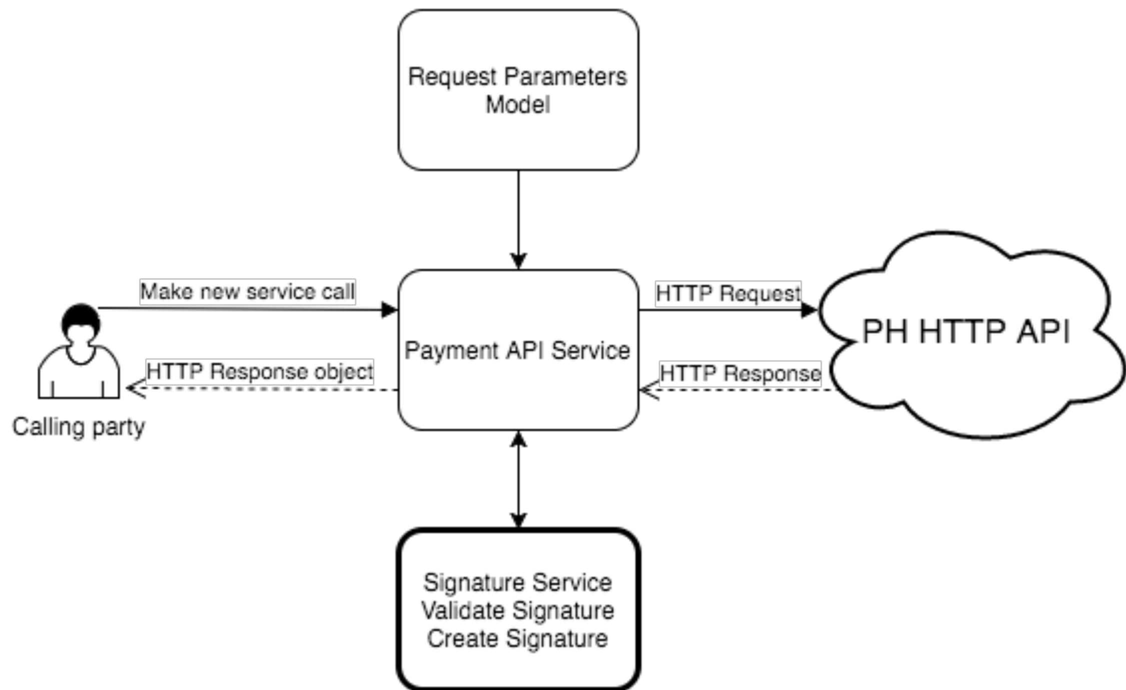


Figure 18. *PaymentAPIService* architecture outline

The project structure is explained in Table 8 below. Table 8 shows the namespaces and descriptions what these namespaces include. The namespaces follow the conventions mentioned in PSR-4. PSR-4 is the new autoloading standard for the PHP which replaces the old PSR-0 standard.

Table 8. Project structure by namespaces

Namespace	Description
\Solinor\PaymentHighway	Service classes for payment highway SDK. Mirroring the HTTP API Form and Payment API sides.
\Solinor\PaymentHihgway\Model	Models used to inject to PaymentAPIService calls and FormAPISerivce calls.
\Solinor\PaymentHihgway\Security	Security handling services, such as authentication signature calculator and signature validator.

The root namespace *\Solinor\PaymentHighway* includes the two main service classes. The *\Solinor\PaymentHihgway\Model* namespace includes the model classes used when creating the request objects which are given to the different methods used by the *FormAPIService* class or the *PaymentAPIService* class. The *\Solinor\PaymentHighway\Security* namespace includes classes that are used when security services are required. The security services are accessed internally by either *PaymentAPIService* class or *FormAPIService* class

5.2 FormAPIService

The *FormAPIService* class helps building the HTTP form parameters. When creating a new *FormAPIService* the common parameters for all methods are given in the constructor. It creates an instance of the service, then it uses the generated methods to receive a list of the parameters for each of the Form API call. Calling the new *FormAPIService* creates a new instance of the service as shown in Figure 19.

```
use \Solinor\PaymentHighway\FormAPIService

$method = "POST";
$signatureKeyId = "testKey";
$signatureSecret = "testSecret";
$account = "test";
$merchant = "test_merchantId";
$baseUrl = "https://v1-hub-staging.sph-test-solinor.com";
$successUrl = "https://example.com/success";
$failureUrl = "https://example.com/failure";
$cancelUrl = "https://example.com/cancel";
$language = "EN";

$formService = new FormAPIService($method, $signatureKeyId, $signatureSecret, $account,
                              $merchant, $baseUrl, $successUrl, $failureUrl,
                              $cancelUrl, $language);
```

Figure 19. Calling new *FormAPIService*.

The methods mirroring the Form API HTTP calls can now be invoked. These methods have the ability to inject the alternative parameters needed for each of the different method calls. Figure 20 shows the methods available for generating different parameters for each API action call.

```
$formService->generateAddCardParameters( $accept_cvc_required = false );
$formService->generatePaymentParameters($amount, $currency, $orderId, $description);
$formService->generateAddCardAndPaymentParameters($amount, $currency, $orderId, $description);
$formService->generatePayWithTokenAndCvcParameters( $tokenId, $amount, $currency, $orderId, $description);
```

Figure 20. Methods to generate different parameters for different Form API action calls

All the parameters are given as a string value. The only exception is the *$accept_cvc_required* variable which is given as boolen *true* or *false* value. By default, the *$accept_cvc_required* variable value is set to *false*.

When one of the methods is called it returns a *Form* object. This object includes the method, URI and the HTTP headers to be used when redirecting to the Form API. The HTTP headers include the signature header with the correct format and the calculated authentication signature.

```
// $form variable is the returned Form object.
$form = new Solinor\PaymentHighway\Model\Form();

// returns value as string.
$httpMethod = $form->getMethod();
$actionUrl = $form->getAction();

// Header parameters are stored as key => value paired array
$parameters = $form->getParameters();
```

Figure 21. Form model returned from service method invoke

The *FormAPIService* differentiates from the *PaymentAPIService* in such a way that the service is not creating any HTTP requests. The redirection to the *FormAPIService* is left to the developer. When leaving the redirection implementation to the developer the Software Development Kit does not cause side-effects to the program flow. Also the Software Development Kit does not take any assumptions over the actual implementation.

5.3 PaymentAPIService

The *PaymentAPIservice* class helps with the interactions made with the Payment API. The *PaymentAPIservice* works like the *FormAPIservice* with added HTTP functionality to do the actual request to the Payment Highway HTTP API. It creates an instance of the service, then it uses the request methods to make payment API calls. When creating a new *PaymentAPIService* the common parameters for all of the methods are injected to the constructor. Figure 22 shows the creation of the new *PaymentAPIService* with the parameters given to the constructor.

```php
use Solinor\PaymentHighway\PaymentAPIService

$serviceUrl = "https://v1-hub-staging.sph-test-solinor.com";
$signatureKeyId = "testKey";
$signatureSecret = "testSecret";
$account = "test";
$merchant = "test_merchantId";

$paymentApi = new PaymentAPIService($serviceUrl, $signatureKeyId, $signatureSecret, $account, $merchant);
```

Figure 22. Creating new *PaymentAPIService* object

When the *PaymentAPIService* instance is created the methods invoking the HTTP API become available. Every method takes in the alternative parameters needed to make a successful HTTP API call as shown in Figure 23.

```php
$paymentApi->initTransaction();
$paymentApi->commitFormTransaction( $transactionId, Transaction $transaction );
$paymentApi->debitTransaction( $transactionId, Transaction $transaction );
$paymentApi->revertTransaction($transactionId, $amount);
$paymentApi->statusTransaction( $transactionId );
$paymentApi->tokenize( $tokenizeId );
$paymentApi->getReport( $date );
```

Figure 23. All publicly callable methods of the *PaymentAPIService* object

The transaction model injected into the commit and debit transaction methods is actually a payload model. This model mirrors the JSON object that is send with the HTTP request to the Payment Highway HTTP API.

In the example shown in Figure 24 the *Transaction* model takes a *Token* model in the constructor. The *Token* model can be omitted when using the *Transaction* model with the *commitFormTransaction* method. This is because the commit action only needs the transaction identifier, amount, and currency. The *Token* model takes the token id as a string. The token id is a UUIDv4 formatted string.

```php
use Solinor\PaymentHighway\Model\Request\Transaction;
use Solinor\PaymentHighway\Model\Token;

$token = new Token( $id );
$transaction = new Transaction( $token, $amount, $currency );

$json = json_encode( $transaction );
```

Figure 24. Transaction model forming

The *Transaction* model accepts a *Token* object, amount as a numeric string, and a currency. The currency is given as a string which is a ISO-4217 formatted e.g "EUR". The Payment highway does not yet accept anything else but euros.

When the *Transaction* model is formed it can be transformed to a JSON string. This is done by implementing a *JsonSerialize* interface within the *Transaction* model. The *JsonSerialize* interface is a standard PHP library interface and it has a method that is called when the *json_encode* function is executed in the *Transaction* model.

When the *Transaction* model is transformed to a JSON string it is attached to the HTTP request body. The *PaymentAPISerivce* executes the HTTP request and returns a JSON in the response body to the caller. The accessing of the values of a result object in the returned JSON, decoded as PHP object is shown in Figure 25.

```php
// evaluates to "100"
$response->result->code

// evaluates to "OK"
$response->result->msg
```

Figure 25. The response object which is then returned to a caller

The response object returned to the caller is a standard PHP object. The access to any values returned in the JSON can be accessed in a same manner as displayed in the figure above.

5.4 Installation

The Software Development Kit is distributed as a Composer package. By default, the Composer uses a service called Packagist to index and list the available packages. The packages used by the Composer do not need to be listed on the Packagist but it enables the package discovery and adoption more efficiently.

Packages are usually hosted on a service such as Github or Bitbucket. These services are Software-as-a-service platforms for the version control systems such as GIT. The Payment Highway SDK is hosted on the Github.

To be able to list a package in Packagist the system needs a composer.json file to be located in the root of the project folder. Figure 26 displays a minimum list of settings found in the composer.json file.

```
{
  "name": "solinor/paymenthighwayio",
  "description": "Paymenthighway SDK",
  "type": "library",
  "homepage" : "https://paymenthighway.fi/dev/",
  "licence" : "MIT",
  "authors" : [
    {
      "name" : "Jonni Larjomaa",
      "email" : "jonni.larjomaa@solinor.com"
    }
  ],
  "require": {
    "php": ">=5.4.0",
    "ramsey/uuid" : "^2.8",
    "nategood/httpful": "*",
    "respect/validation": "^1.0"
  },
```

Figure 26. Example of composer.json file

This file includes meta-data used by the Packagist to index and display the package contents. This file has listed also the dependent packages for this project in the require section.

To be able to use this library with composer one has to add to their projects composer.json the requirement for this library. Figure 27 shows how the SDK package is added as a dependency in the composer.json files require section.

```
"require" : {
  "solinor/paymenthighwayio" : "1.0.0-RC1"
}
```

Figure 27. Adding requrement to composer.json for PH SDK

After the package has been added to the composer.json file the Composer must be executed to install or update the new packages. This can be achieved by executing the following (Figure 28) command on command line.

```
jonni@loki:~/codebase/test-ph-php-lib$ composer install
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Installing respect/validation (1.0.5)
    Downloading: 100%

  - Installing nategood/httpful (0.2.20)
    Downloading: 100%

  - Installing paragonie/random_compat (v2.0.2)
    Downloading: 100%

  - Installing ramsey/uuid (2.9.0)
    Downloading: 100%

  - Installing solinor/paymenthighwayio (1.0.0-RC1)
    Downloading: 100%
```

Figure 28. Composer installing PH SDK and all its dependencies.

This example expects that the Composer is installed and the executable is found from the system environment variable PATH. After the composer has done the installation the SDK is ready to be used with the project at hand.

6 Results and Conclusions

This chapter describes the key findings of the thesis. The outcome is discussed in general and the conclusions of the success of the thesis are summarised.

Payment systems consists of many assets and roles. The Payment industry has grown into a large ecosystem and multiple players have joined the field of payment technologies. The card brands control and regulate how the systems involved in the payment scheme work together by collaborating through the PCI-SSC. The Council is responsible for updating security rules used in the payment systems.

This thesis introduced a payment gateway solution called the Payment Highway. The Payment Highway has been developed by a software company called Solinor Oy. These payment gateways act as a middleware for authorizations, credit and debit transactions and they securely store and handle consumers credit card information.

The goal of this thesis was to write a Software Development Kit to ease the integration work to be done when using the Payment Highway as an electronic commerce payment solution.

The Payment Highway is written on top of two HTTP APIs. These APIs are used to handle the payments, authorizations, tokenization and reporting. The form API use a HTML rendered form to collect the Cardholder data and process it. The payment API is used as a backend function for committing the transactions made in the HTML form, fetching card token, creating tokenized payments and fetching the statuses and reports of the transactions and settlements.

When new customers were introduced to the Payment Highway the amount of custom integration work needed to be done was very intensive and many similar actions were repeated. There was no easy solution to use APIs as programmatically and Solinor required to build a Software Development Kit to lower the amount of work that needs to be done.

The developed Software Development Kit has a simple architecture and it was designed with a convention over configuration in mind. The SDK consists of two service classes, one for each API and a set of models and helper utilities. All these features together make the SDK very intuitive to use and easy to install. Installation is done by using a composer distribution management software that is developed for PHP based projects.

The main challenge during the SDK development was that the requirements and specifications were changing all the time. This caused the problem of “not getting it done”- dilemma. Finally in the end the decision to release the SDK was made and so the first version is now available through Packagist. The distribution of the SDK through packagist helps developers to easily update to the latest versions. This SDK has improved the quality of integrations and security of the deployments as new security features can be implemented faster.

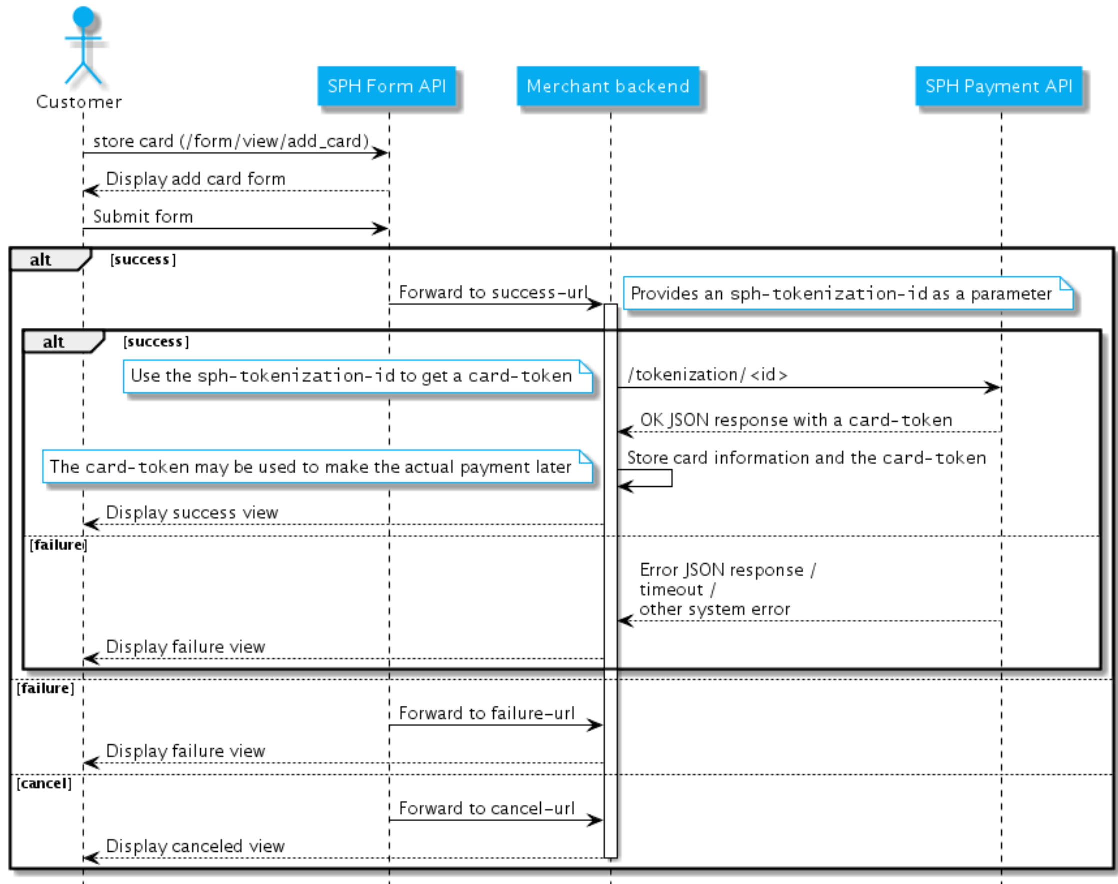
Further development with the SDK is needed for bug fixes and newest functionalities to be implemented. The following development steps could include solution specific libraries or plugins on top of this SDK. The SDK has improved integration work and accompanying new customers is now faster, which was the goal of this SDK.

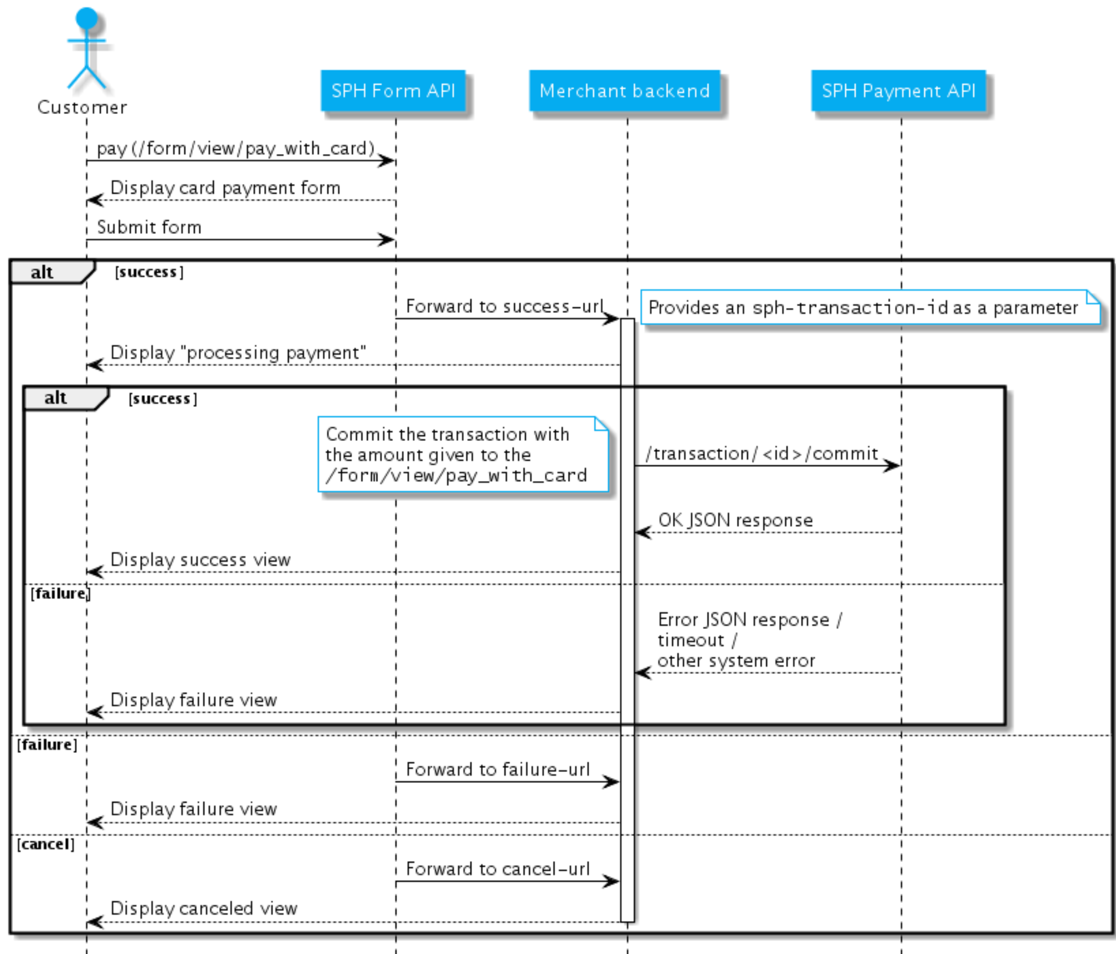
References

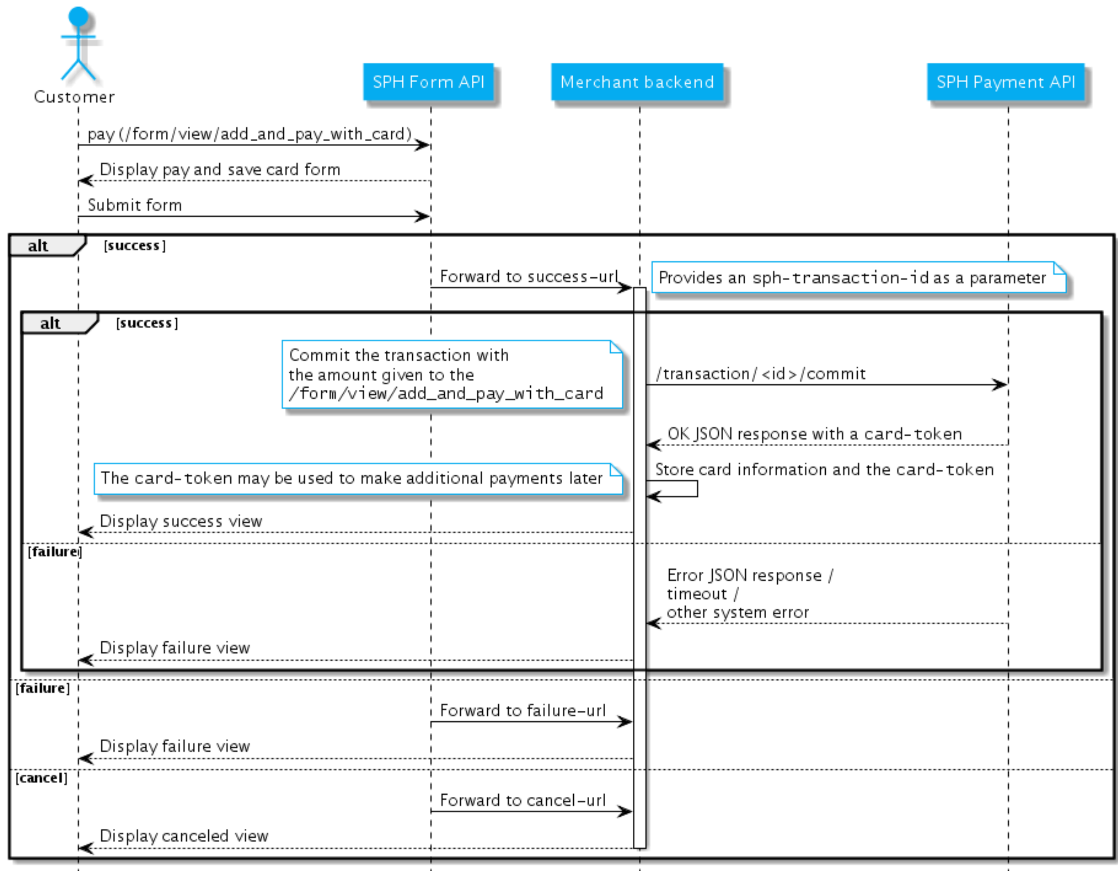
1. Gomzin S. Hacking point of sale Long C, editor. Indianapolis: Wiley; 2014.
2. Radu C. Implementing Electronic Card Payment Systems Norwood: Artech House; 2002.
3. PCI Security Standards Council. PCI DSS Quick Reference Guide: Understanding the Payment Card Industry Data Security Standard version 3.1 [PDF].; 2015.
4. Wright S. PCI DSS: A Practical Guide to Implementing and Maintaining Compliance. 3rd ed. Cambridge: GBR: IT Governance Publishing; 2011.
5. Internet Engineering Task Force (IETF). HTTP/1.1 Message Syntax and Routing. [Online].; 2014 [cited 2016 4 17. Available from: https://tools.ietf.org/html/rfc7230.
6. Internet Engineering Task Force (IETF). HTTP/1.1 Semantics and Content. [Online].; 2014 [cited 2016 4 18. Available from: https://tools.ietf.org/html/rfc7231.
7. Collin Jackson DRS,DST,AB. An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks [PDF].; 2007 [cited 2016 3 1. Available from: https://en.wikipedia.org/wiki/Extended_Validation_Certificate.
8. Abeysinghe S. RESTful PHP Web Services: Packt Publishing Ltd; 2008.
9. Internet Engineering Task Force (IETF). The JavaScript Object Notation (JSON) Data Interchange Format. [Online].; 2014 [cited 2016 4 18. Available from: https://tools.ietf.org/html/rfc7159.
10. The PHP Group. PHP Homepage. [Online].; 2016 [cited 2016 4 19. Available from: http://php.net.
11. PHP-FIG PHP - Framework Interop Group. PSR - PHP Standards Recommendations. [Online].; 2016 [cited 2016 4 20. Available from: http://www.php-fig.org/.
12. Nils Aldermann JB. Composer Homepage. [Online].; 2016 [cited 2016 4 19. Available from: http://getcomposer.org.
13. Potencier F. The rise of Composer and the fall of PEAR. [Online].; 2014 [cited 2016 4 19. Available from: http://fabien.potencier.org/the-rise-of-composer-and-the-fall-of-pear.html.
14. Solinor. Paymenthighway. [Online].; 2016 [cited 2016 3 1. Available from: https://paymenthighway.fi/dev.

15. Evans DS, Schmalensee R. Paying with Plastic : The Digital Revolution in Buying and Borrowing. 2nd ed.: MIT Press; 2005.

Appendices




Customer
SPH Form API
Merchant backend
SPH Payment API
pay (/form/view/pay_with_card)
Display card payment form
Submit form
alt
[success]
Forward to success-url
Provides an sph-transaction-id as a parameter
Display "processing payment"
alt
[success]
Commit the transaction with the amount given to the /form/view/pay_with_card
/transaction/<id>/commit
OK JSON response
Display success view
[failure]
Error JSON response / timeout / other system error
Display failure view
[failure]
Forward to failure-url
Display failure view
[cancel]
Forward to cancel-url
Display canceled view


Customer
SPH Form API
Merchant backend
SPH Payment API
pay (/form/view/add_and_pay_with_card)
Display pay and save card form
Submit form
alt
[success]
Forward to success-url
Provides an sph-transaction-id as a parameter
alt
[success]
Commit the transaction with the amount given to the /form/view/add_and_pay_with_card
/transaction/<id>/commit
OK JSON response with a card-token
Store card information and the card-token
The card-token may be used to make additional payments later
Display success view
[failure]
Error JSON response / timeout / other system error
Display failure view
[failure]
Forward to failure-url
Display failure view
[cancel]
Forward to cancel-url
Display canceled view