Fall 9-2020

DESIGN AND IMPLEMENTATION OF A SECURE ACCESS LAYER FOR A PSD2 COMPLIANT CONSENT MANAGEMENT ENGINE

Mentor Elshani

Programi për Shkenca Kompjuterike dhe Inxhinieri

DESIGN AND IMPLEMENTATION OF A SECURE ACCESS LAYER FOR A PSD2 COMPLIANT CONSENT MANAGEMENT ENGINE

Shkalla Bachelor

Mentor Elshani

Shtator / 2020
Prishtinë

Programi për Shkenca Kompjuterike dhe Inxhinierisë

Punim Diplome
Viti akademik 2016 – 2017

Mentor Elshani

DESIGN AND IMPLEMENTATION OF A SECURE ACCESS LAYER FOR A PSD2 COMPLIANT CONSENT MANAGEMENT ENGINE

Mentori: Ramiz Hoxha, Lecturer at UBT Prishtine
Mbikqyrës: Ludwig Adam, CTO at petaFuel GmbH
Mbikqyrës: Max Gröning, Project Manager at petaFuel GmbH

Shtator / 2020

Abstract

Online banking is growing fast, and customers are becoming increasing more comfortable with it than with the traditional bank services. Banks have to remain up to date with the latest trends in technology to satisfy customers' demands for their everyday usage of banking services. Various banks often offer mobile applications with many features so that the user does not have to talk over the phone or chat via the internet with customer support, or even to physically go to the bank branch. When a bank releases its own online banking app for smartphones, the customer is likely to download it and start using it, as it is typically the only available choice. This certainly represents a secure way of accessing the bank account information, since there is no third party between the bank and the customer, regardless of the limitations that the online banking system can have.

According to a new European directive, called PSD2, all EU banks must set up online banking interfaces, which should be opened for access by third parties. Third parties could be social media platforms, video games or even virtual banking apps. The entities referred to as third party providers in PSD2 can perform every online banking service, such as downloading revenue and transaction statementsor making transfers, the same services which before only the bank could offer. This is expected to certainly encourage third party providers to compete for bringing attractive solutions for customers.

Of course, a third party provider will be able to mediate such sensitive information, only after the customer gives permission to the provider for the service in question. For a secure way of giving permission, PSD2 presented the so-called Strong Customer Authentication. Strong Customer Authentication, shortly known as SCA, is a new European regulatory requirement that aims to reduce fraud and make online banking services more secure. For security reasons, SCA is done without the mediation of the third party provider, so it is a complete process between the customer and the bank. This way of authentication requires the usage of at least two out of three private elements of the customer, which could be: something that the customer knows (for example a password or a PIN), something that the customer has (for example a phone or a hardware token) or something that the customer is (for example his/her fingerprint or face recognition).

In this thesis the role of a third party in the process of SCA is explained. Regarding PSD2, one of the ways to perform SCA is by doing an Authorization Code Flow of the OAuth protocol. In this case the third party provider is the client application and the bank – or as it is called in PSD2, the Account Service Payment Service Provider (ASPSP) –is the authorization server. The main focus of this thesis is the design and the implementation of the SCA using the OAuth Approach.

In addition, this thesis also elaborates the design of a PSD2-compliant XS2A Client, which is able to communicate with multiple complex interfaces of different banks. The XS2A Client offers a unified and protected REST API which wraps the PSD2-required services.

This thesis is developed in cooperation with petaFuel GmbH and the code for its implementation waswritten in Java as an open source project.

Abstrakti

Përdorimi i shërbimeve bankare në internet po rritet me shpejtësi dhe klientët po bëhen gjithnjë e më të kënaqur me të sesa me shërbimet tradicionale bankare. Bankat duhet të qëndrojnë të azhurnuara me trendet më të fundit në teknologji për të kënaqur kërkesat e klientëve për përdorimin e tyre të përditshëm të shërbimeve bankare. Bankat e ndryshme shpesh ofrojnë aplikacione mobile me shumë funksione në mënyrë që përdoruesi të mos ketë nevojë të flasë në telefon ose të komunikojë përmes internetit me mbështetjen e klientit, apo edhe të shkojë fizikisht në degën e bankës. Kur një bankë lëshon aplikacionin e saj bankar në internet për smartfonë, klienti detyrohet ta shkarkojë atë dhe të fillojë ta përdorë atë, pasi është zakonisht zgjedhja e vetme në dispozicion. Kjo sigurisht përfaqëson një mënyrë të sigurt për të hyrë në informacionin e llogarisë bankare, pasi që nuk ka asnjë palë të tretë midis bankës dhe klientit, pavarësisht nga kufizimet që mund të ketë sistemi bankar në internet.

Sipas një direktive të re Evropiane, të quajtur PSD2, të gjitha bankat e BE-së duhet të krijojnë ndërfaqe bankare në internet, të cilat duhet të hapen për qasje nga palët e treta. Palët e treta mund të jenë platforma të mediave sociale, lojëra video apo edhe aplikacione bankare virtuale. Subjektet e referuara si ofrues të palëve të treta në PSD2 mund të kryejnë çdo shërbim bankar në internet, të tilla si shkarkimi i deklaratave të të ardhurave dhe transaksioneve ose kryerja e transfereve të parave, të njëjtat shërbime të cilat vetëm banka mund t'i ofronte. Kjo pritet që sigurisht të inkurajojë ofruesit e palëve të treta të konkurrojnë për sjelljen e zgjidhjeve tërheqëse për klientët.

Sigurisht, një ofrues i palës së tretë do të jetë në gjendje të ndërmjetësojë në një informacion kaq sensitiv, vetëm pasi klienti t'i japë leje ofruesit për shërbimin në fjalë. Për një mënyrë të sigurt të dhënies së lejes, PSD2 paraqiti të ashtuquajturën Strong Customer Authentication. Strong Customer Authentication, i njohur shkurtimisht si SCA, është një kërkesë e re rregullatore Evropiane që synon të zvogëlojë mashtrimin dhe t'i bëjë shërbimet bankare në internet më të sigurta. Për arsye sigurie, SCA bëhet pa ndërmjetësimin e ofruesit të palës së tretë, kështu që është një proces i plotë midis klientit dhe bankës. Kjo mënyrë e vërtetimit kërkon përdorimin e të paktën dy në tre elementeve private të klientit, të cilat mund të jenë: diçka që klienti e di (për shembull një fjalëkalim ose një PIN), diçka që klienti ka (për shembull një telefon ose një pajisje harduerike) ose diçka që klienti është (për shembull shenjat e gishtave ose fytyra e tij / saj).

Në këtë tezë shpjegohet roli i një pale të tretë në procesin e SCA. Lidhur me PSD2, një nga mënyrat për të kryer SCA është duke bërë një Authorization Flow Code të protokollit OAuth. Në këtë rast, ofruesi i palës së tretë është aplikacioni i klientit dhe banka - ose siç quhet në PSD2, Account Service Payment Service Provide (ASPSP) - është serveri i autorizimit. Fokusi kryesor i kësaj teze është hartimi dhe implementimi i SCA-së duke përdorur Qasjen OAuth.

Për më tepër, kjo tezë shtjellon gjithashtu modelin e një XS2A Client të përputhshëm me PSD2, i cili është në gjendje të komunikojë me ndërfaqe të shumta komplekse të bankave të ndryshme. Ky XS2A Client ofron një API të unifikuar dhe të mbrojtur REST i cili mbështjell shërbimet e kërkuara nga PSD2.

Kjo tezë është zhvilluar në bashkëpunim me petaFuel GmbH dhe kodi për zbatimin e tij është shkruar në Java si një projekt Open Source.

Acknowledgements

I am grateful to the team of petaFuel, a group of experts who participated in this project and supported my thesis. I would especially like to thank Ludwig Adam, owner of petaFuel GmbH and Max Gröning, project manager at petaFuel GmbH, both advisors of this thesis, who gave me the opportunity to do my thesis in such an interesting project, after my one-year internship in the company.

I’d also like to thank the lecturers of my faculty at UBT College - Pristina. The literature and exercises from the courses I took during my studies were very useful in conducting the necessary research and implementing this project. In fact, this bachelor project is a combination of most of the subjects I took during my studies, including: Computer Science 1 & 2 courses, where we learned Object Oriented Programming in Java; Software Engineering courses, where we learned project development methodologies; Database Systems; Algorithms and Data Structures; Project Management; and Computer Networks.

Table of Content

List of Figures

List of Tables

List of Abbreviations

API – Application Programming Interface

ASPSP – Account Service Payment Service Provider

PSD – Payment Services Directive

PSU – Payment Service User

RTS – Regulatory Technical Standards

SCA – Strong Customer Authentication

TPP – Third Party Provider

XS2A – Access to Accounts (Bank Accounts)

HTTP(S) – Hypertext Transfer Protocol (Secure)

1. Introduction

This chapter provides an introduction to the related topics, such as, online banking and its security issues, PSD2 and its benefits, Styx - the Open Source Project which this thesis is part of, and also briefly covers the solved problems of this research.

1.1 PSD2 and Online Banking

Online banking has had a huge impact in the banking industry, since customers can now access their bank account information in secure apps and websites, which serve as “virtual banks”. While customers find this useful, they are also concerned that their data can be compromised on the Internet. Therefore, banks aim to provide security to their customers by implementing the latest technological trends.

According to EU Directive, PSD2, all EU banks (ASPSPs) are obligated to set up interfaces that comply with their security regulations (based on the latest release on 14 September 2019). This allows third party providers, such as social media platforms and messaging apps, to access customer account information or make transfers by using the ASPSPs interfaces. PSD2 is expected to revolutionize the payment industry, because banks will have to create competitive and attractive solutions, otherwise they will lose most of their customer interactions. This agreement also impacts the monopoly of the banks over customer accounts and pushes TPPs towards quicker innovation in the fintech space.

More than 60 banks throughout Europe, which together represent the “Berlin Group”, collaborated and worked on a detailed specification, the NextGenPSD2 XS2A Framework, which is currently the biggest PSD2 specification. Throughout this framework, XS2A communications will be way faster, cheaper, and safer, less complex, easier to maintain, and have lower fragmentation risk.

1.2 Styx - A PSD2 XS2A Client

This thesis is a part of Styx, a product of petaFuel GmbH. Styx is a PSD2 XS2A Client that implements the Berlin Group NextGenPSD2 specification. The project development is Open-Source and published under https://github.com/petafuel/styx. The project is written in Java and uses a Postgres DB.

The interface aims to communicate securely with ASPSPs via TLS (Transport Layer Security) using a certificate issued by a qualified trust service provider according to the eIDAS regulation. The partner company of petaFuel GmbH, PayCenter GmbH, is an E-Money institution which is allowed to have such a valid eIDAS-certificate from the national certification authority, which enables the implementation of this system to work. [2]

Fetching account information is only possible after the customer, respectively the PSU, grants a validated consent to Styx. A consent is actually a “key”, which the ASPSP creates for a user, with the permission which the PSU sets. The Berlin Group Specification introduced different SCA (Strong Customer Authentication) methods, which the ASPSPs can implement for validating consents or initiating payments.

This thesis will explain how PSU can perform the SCA with the OAuth2 protocol, for validating a consent or for initiating payments. It will also explain how client applications can send authenticated HTTP requests to the Styx REST API.

2. State of the Art

This chapter contains the basic theoretical explanations of the relevant topics, such as standards, agreements, protocols and development tools, used for the implementation of the solution.

Chapter 2.1 covers the theoretical background, while Chapter 2.2 explains the development tools that are used in solving the problems that arise in implementation.

2.1 Theoretical Background

2.1.1 NextGenPSD2 XS2A Framework

The Berlin Group NextGenPSD2 XS2A Framework is the biggest standard for an XS2A Interface. This framework supports TPPs in offering innovative solutions to customers, using flexible APIs, which offer secure access to bank accounts. According to PSD2 regulation, PSUs do not have to share the card details with the TPP. Therefore, this specification presents the sophisticated SCA approaches for authorizing Styx, as a TPP, to validate consents or confirm initialized payments. SCA is also a requirement of PSD2, which is concerned with multi-factor authentication, in order to increase the security of online payments. [9]

Until April 2019, more than 1700 ASPSPs (banks) have opened their XS2A interfaces, implemented according to the Berlin Group specification. [1]

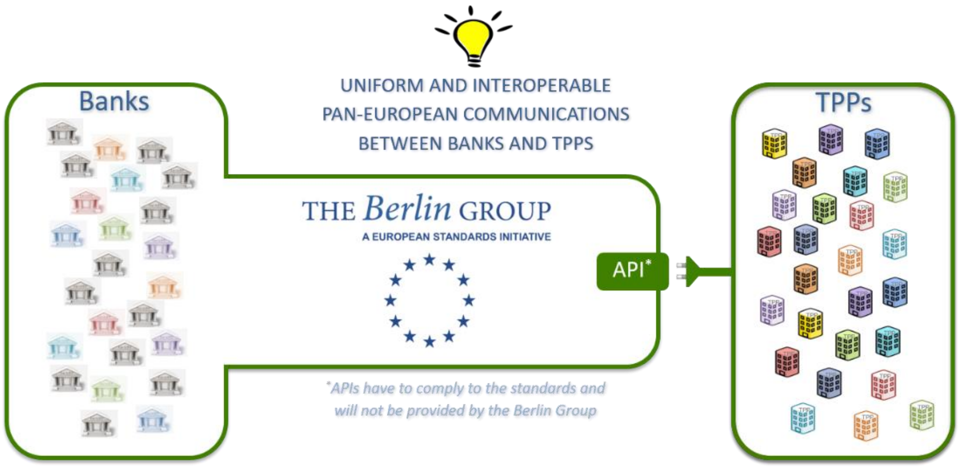

Banks
UNIFORM AND INTEROPERABLE
PAN-EUROPEAN COMMUNICATIONS
BETWEEN BANKS AND TPPS
THE Berlin GROUP
A EUROPEAN STANDARDS INITIATIVE
API*
TPPs
*APIs have to comply to the standards and
will not be provided by the Berlin Group

Figure 1 - Berlin Group[1]

[1] PSD2 Access to Bank Accounts – Berlin Group (https://www.berlin-group.org/psd2-access-to-bank-accounts)

The Berlin Group standard covers the core services defined by PSD2, AIS (Account Information Service), PIS (Payment Initiation Service) and PIIS (Payment Instrument Issuing Service). Through these services, TPPs are allowed to interact with PSUs data.

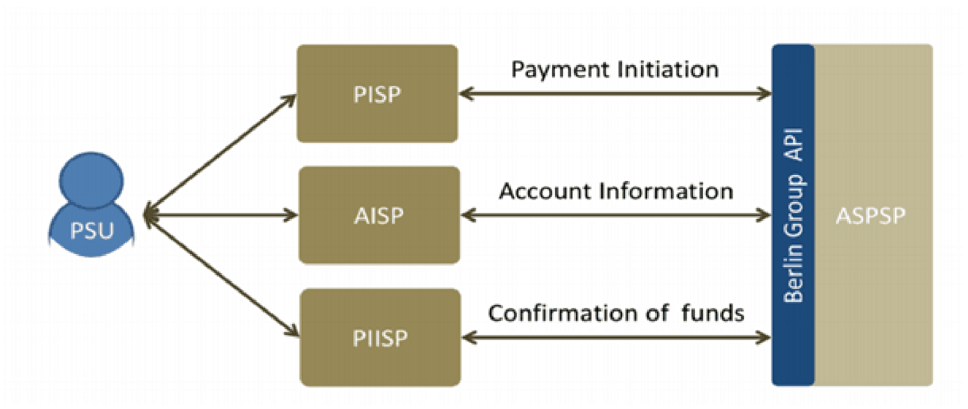


Figure 2 - Scope of Berlin Group API[2]

2.1.2 PSD2 Replacing FinTS (Financial Transaction Services)

Before PSD2, a lot of third-party providers in Germany were using FinTS, formerly known as HBCI (Homebanking Computer Interface). FinTS is a bank-independent protocol for online banking, designed and used by German banks. This protocol was meant for home banking and allows any software developer to create a HBCI client-side, which can access all HBCI-supporting banks, to manage their own account. In the HBCI protocol there is no such thing as a Third-Party Provider, since all the functionality is directly between the customer and the bank.

Several banks have announced they are switching off their FinTS/HBCI services soon, because of the PSD2 upgrade.

2.1.3 Strong Customer Authentication

A very sensitive requirement of the PSD2 Directive for the ASPSPs is the so-called Strong Customer Authentication, or SCA. According to the regulatory technical standards (RTS), SCA regulates the access of account information and online payment initiation, in order to make them more secure.

SCA requires authentication which uses at least two of the following forms:

[2] NextGenPSD2 Downloads – Berlin Group (https://www.berlin-group.org/nextgenpsd2-downloads)

1. Something the customer **knows** (for example: password or PIN)
2. Something the customer **has** (for example: smartphone or hardware token)
3. Something the customer **is** (for example: fingerprint or face recognition)

and these should be independent from one another. [7]

PSD2 specifies that a TPP can rely on the ASPSP to authenticate the customer using Strong Customer Authentication (SCA). Three types of approaches have been defined:

- Embedded: the TPP captures the personal security credentials (e.g. username/password/ 2nd factor) of the customer and submits them to the ASPSP.
- Redirection: the PSU is redirected to the ASPSP's authorization webpage for authenticating the customer.
- Decoupled: SCA is done through a dedicated device and/or software.

2.1.4 Single Euro Payments Area

Single Euro Payments Area, known as SEPA, is a payment-integration initiative of the EU, which aims to simplify transfers between the EU countries + Iceland, Liechtenstein, Norway, Switzerland and the United Kingdom. It aims to improve the efficiency of cross-border payments, by allowing customers to make euro transfers in the located area without any fee. [4]

This initiative is also included in the PSD agreement, which supports the following instruments of SEPA:

- SEPA Credit Transfer
- SEPA Instant Credit Transfer

Both of the payment products can be present as a pain.001.001.03 XML element, which is a ISO 20022 Standard. This is the most used version of a Credit Transfer message. It is compact to be used for every type of payment, such as: simple single transfers/payments, future dated payments, multiple payments, or periodic payments (e.g. salaries).

An example of a single payment interpreted as a pain.001.001.03 message:

```xml
<Document xmlns="urn:iso:std:iso:20022:tech:xsd:pain.001.001.03">
<CstmrCdtTrfInitn>
<GrpHdr>
<MsgId>PSD294387538754378</MsgId>
<CreDtTm>2018-12-10T09:40:47.314+01:00</CreDtTm>
<NbOfTxs>1</NbOfTxs>
<CtrlSum>100</CtrlSum>
<InitgPty>
<Nm>Name InitgPty</Nm>
</InitgPty>
</GrpHdr>
<PmtInf>
<PmtInfId>NOTPROVIDED</PmtInfId>
<PmtMtd>TRF</PmtMtd>
<NbOfTxs>1</NbOfTxs>
<CtrlSum>100</CtrlSum>
<ReqdExctnDt>1999-01-01</ReqdExctnDt>
<Dbtr>
<Nm>Debtor Name</Nm>
</Dbtr>
<DbtrAcct>
<Id>
<IBAN>DE86999999990000001000</IBAN>
</Id>
</DbtrAcct>
<DbtrAgt>
<FinInstnId>
<BIC>TESTDETT421</BIC>
</FinInstnId>
</DbtrAgt>
<ChrgBr>SLEV</ChrgBr>
<CdtTrfTxInf>
<PmtId>
<EndToEndId>EndToEndId</EndToEndId>
</PmtId>
<Amt>
<InstdAmt Ccy="EUR">100</InstdAmt>
</Amt>
<Cdtr>
<Nm>Hans Handbuch</Nm>
</Cdtr>
<CdtrAcct>
<Id>
<IBAN>DE98999999990000009999</IBAN>
</Id>
</CdtrAcct>
<RmtInf>
```

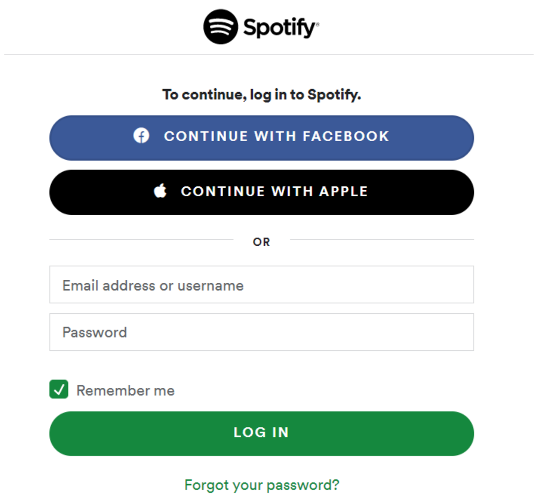
```
<Ustrd>POSTPaymentEmbPain_OK</Ustrd>
</RmtInf>
</CdtTrfTxInf>
</PmtInf>
</CstmrCdtTrfInitn>
</Document>
```

2.2 Frameworks, Libraries and Patterns

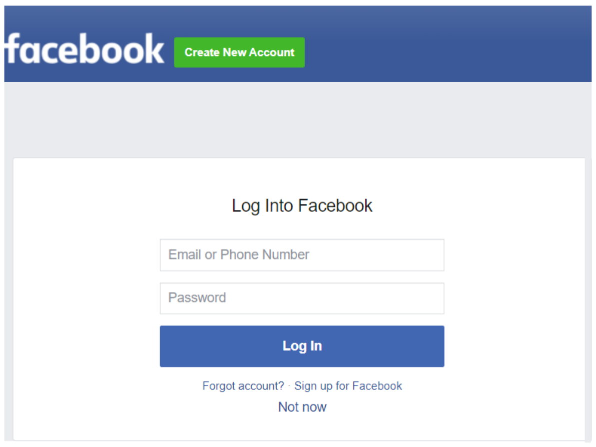
2.2.1 OAuth2

OAuth (Open Authentication) is a secure authorization protocol that regulates the authorization of third-party applications to retrieve the customer information. The principle is that the third-party application will use authorization tokens to prove the identity between customers and the service providers. [5]

The involved parties are the Resource Owner (user), the Client (application) and the Resource Provider. The Provider (such as Google, Twitter, Facebook) provides the authentication token to the Client applications that want access to the user data.



(a)



(b)

Figure 3 - (a) OAuth Client - Spotify; (b) OAuth Provider - Facebook

For example, you can tell Facebook that it is ok for Spotify to access your profile information, such as preferences, or to post updates to your timeline, without having to give Spotify your Facebook password. This minimizes risk in a major way: in case that Spotify faces an attack, your Facebook password will remain safe.

2.2.1.1 OAuth 2.0 Authorization Code Grant

The **OAuth 2.0** is an authorization framework that defines a number of grants/methods for client applications to collect an access token, which can be used to authenticate a request to the API endpoint of the provider. The access token represents the user's permission for the client app to access their data. [6] The five grants for getting an access token are:

- Authorization Code Grant
- Implicit Grant
- Resource Owner Credentials Grant
- Client Credentials Grant
- Refresh Token Grant

The Authorization Code Grant is used for both getting access and refreshing tokens. It is a redirect-based flow, so the client app must provide interfaces to interact with the resource owner and to be able to receive incoming requests from the authorization server (via redirection).

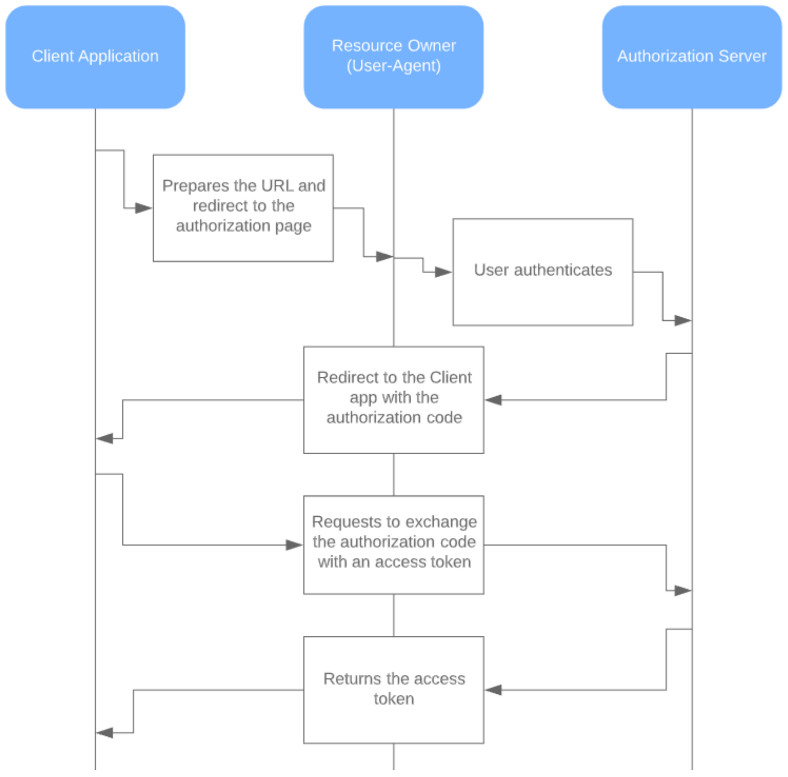

Client Application
Resource Owner (User-Agent)
Authorization Server
Prepares the URL and redirect to the authorization page
User authenticates
Redirect to the Client app with the authorization code
Requests to exchange the authorization code with an access token
Returns the access token

Figure 4 - OAuth 2.0 - Authorization Code Flow

The diagram in Figure 4 explains the following steps:

1. The client app builds the link to the authorization page of the resource server, where the variables in the table below must be added as query parameters.

Parameter	**Description**
response_type	Value must be set to “code”.
client_id	A unique string identifier, which the authorization server generates and returns, during the registration of the client application.
redirect_uri	The redirect endpoint URI of the client application, where the authorization server redirects the resource owner’s user-agent.
Scope	One or more scope values which refer to the part of the user’s account that you wish to access. The list of possible values should be defined by the authorization server.
State	The random unique string value generated by the client application to identify the request and the callback. The authorization server includes this parameter when redirecting the user-agent back to the client application.
code_challenge	A URL-safe base64-encoded SHA256 hash of the “code_verifier” (a random 43-128 character String).
code_challenge_method	The hashing method used for the code_challenge. It is usually set to “S256”.

Table 1 - Query Parameters of the Authorization Endpoint

An example of the URL where the user-agent will be first redirected:

https://my.authorizationserver.net/auth?response_type=code&client_id=6779ef20e75817b79602&redirect_uri=https%3A%2F%2Fmy.clientapp.net%2F%0A&scope=photos&state=8b241d9a-f0ce-4f81-beb1-176241bd06fb&code_challenge=sXyxOllWNm8HmBFTDxEP41vi-Aan62kErHus-5HGLGc=&code_challenge_method=S256

2. The user will be redirected to the authorization page, where he agrees that the client application can retrieve his data from the resource server. The client will have to give his credentials to finish this step.
3. The authorization validates the user's request, and if the validation is successful, it redirects the request to the given *redirect_uri*,where they append the following query parameters:

Parameter	**Description**
code	The authorization code, which will be later exchanged with the access token.
state	The same parameter which was generated in Step 1. See Table 1.

Table 2 - Query Parameters of the Callback Endpoint

4. Now the client app needs to exchange the received authorization code with an access token. Therefore, the client application should send an HTTP POST Request to the token endpoint of the authorization server with the following parameters:

Parameter	**Description**
grant_type	Value must be set to "authorization_code".
code	The code which was received in Step 3.
client_id	See Table 1.
code_verifier	The cryptographically random key that was used to generate the code_challenge in Step 1.
redirect_uri	Should be identical to the redirect_uri sent in Step 1.

Table 3 - Body Parameters of the Token Endpoint

The authorization server validates the request, hashes the code_verifier with the given hashing method in Step 1, compares it with the code_challenge given is Step 1, and if the request is valid and authorized, it returns the following parameters in a JSON object:

Attribute	Description
access_token	The access token issued by the authorization server.
token_type	The type of token. In most of the cases it is “Bearer”.
refresh_token (optional)	The refresh token, which can be used to obtain new access tokens using the same authorization grant.
expires_in (optional)	Number of seconds before this token will be set to invalid.

Table 4 - Response Attributes of the Token Endpoint

At the end of this procedure, the Authenticated Requests using the provided access token can be executed.

```
curl --location --request GET 'https://my.resource-server.net/me' \
--header 'Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia'
```

2.2.2 Entity-Control-Boundary Pattern

The Entity-Control-Boundary is an object-oriented architectural pattern used for structuring the back-end of a software. A service package should group the three types of classes, in order to be used as units.

- Boundary classes represent the REST API, which expose the functionality of the component. They communicate with external Actors and control classes only.
- Control classes represent the process or activity handling, and are made for reusable service behind a boundary. They communicate with boundaries and entities, and if needed other control classes.
- Entities represent object-oriented or procedural objects. They could relate to other entities, but only communicate with Control classes.

This pattern is often compared to MVC (Model View Controller), even though ECB is not appropriate for user interfaces. The control here is obviously mirroring the controller, the entity is similar to the model, and the boundary defines the REST API, since there are no views in ECB.

2.2.3 Unit testing - JUnit Framework

Software testing is important as it allows systems to verify that a part of the software works properly as expected.

Unit testing can be done in a manual or automated way. Running tests automatically helps find the possible incorrectness that could happen when changing the source code.

A unit test is a piece of code that aims to execute a function of the system, usually a short functionality as a method or a class.

For a proper execution of the functionality, when creating unit tests, external dependencies should be replaced with test implementations known as ‘mocks’.

JUnit is a unit testing framework for the Java programming language. It is an open source framework which is used for writing & running tests. JUnit provides Annotation to identify the test methods and utility methods to assert expected results.[8]

JUnit tests can be organized into test suites which contain test cases and other test suites.

This framework can be easily integrated with either of the following automation tools:

- Eclipse
- Ant
- Maven

Early coverage of unit testing allows developers to maintain the project test cases properly.

3. Problem Definition

This chapter describes the issues that are solved in the thesis.

3.1 Performing SCA for Consents and Payments Using the OAuth2 Protocol

The Berlin Group NextGenPSD2 XS2A allowed ASPSPs to integrate the OAuth2 framework to support the authorization of the PSU towards the TPP for initiating payments or for creating consent to retrieve account information. In this case, the PSU is the resource owner, the ASPSP is the Resource server, which has to set up the authorization server, and the TPP will be the client application. Customers will follow the OAuth "Authorization Code" Grant Type to perform Strong Customer Authentication, shortly known as SCA, after each payment; as well as every time they create a consent, by which they give access to TPP to fetch account information.

3.2 Implementing the Client Authentication for the REST Services of Styx

Styx, as a PSD2 XS2A client, offers REST services for accessing account information and for initiating payments. Client applications which integrate Styx as a XS2A provider, should send only authenticated requests in order to access the offered services.

3.3 Supporting Multiple PSD2 XS2A Standards

Styx aims to communicate and be compatible with the XS2A interfaces of most EU Banks. The Styx API server should accept requests from the client and while processing them, in order to communicate with the corresponding ASPSP, it should build the request depending on which PSD2 standard the ASPSP follows. Each ASPSP is supposed to follow a PSD2 XS2A standard and each standard has its own requirements and definition. This means that Styx should save each ASPSP and its requirements in a database for handling the requested service.

4. Design & Implementation

This chapter elaborates the implemented solution for each of the problems described in Chapter 3. It provides the design procedures and the final implementation.

4.1 ASPSP Directory - Database

As mentioned above, the system is able to handle requests and prepare them for executing it on the REST API of the associated bank. The standard, which the bank follows, and its requirements are stored in a database, where Styx is guided oriented to process the requested service.

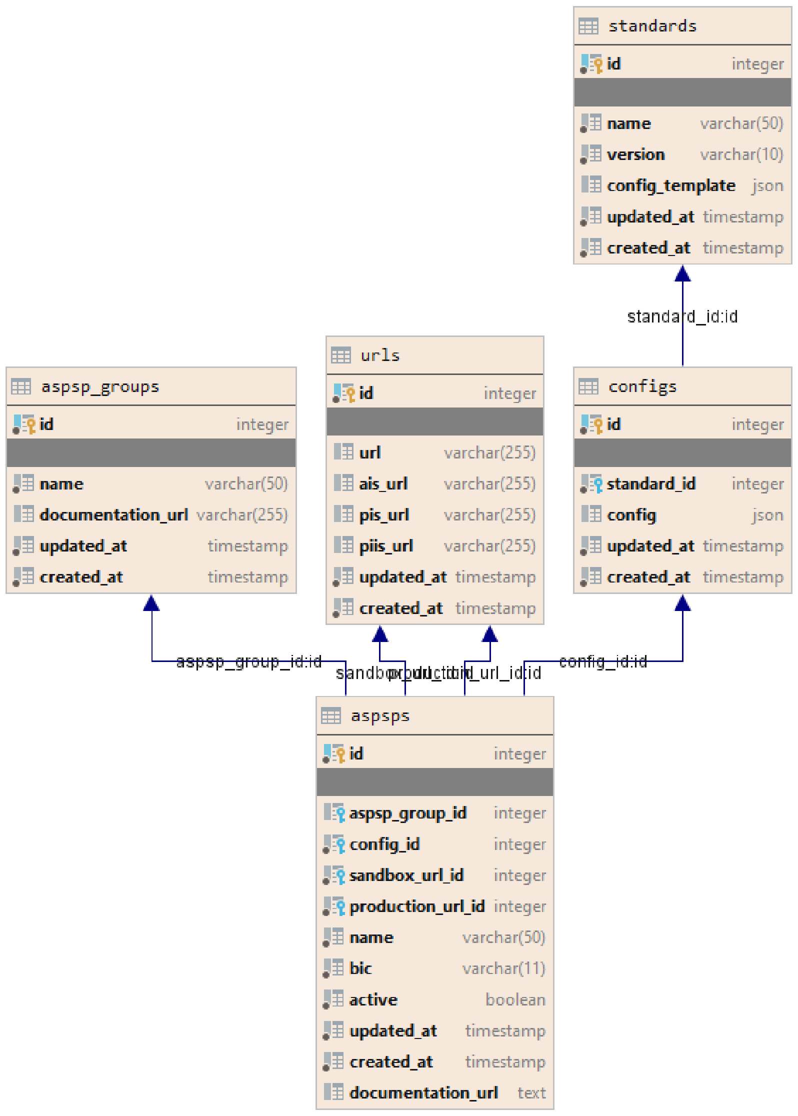


Figure 5 - ER Diagram of the ASPSP Directory

The client applications should always provide the BIC (Bank Identifier Code), which is a unique column of the *aspsps* table, as a parameter in the REST API calls.

Table	Description
aspsps	This table contains the basic information of the ASPSP, such as the name and the BIC, and the foreign keys for aspsp_group, the urls (for sandbox and production environments) and config.
aspsp_groups	ASPSPs can belong to a bank group, which have the same documentation, requirements (configs and standard), but each bank has its own BIC.

	However, an ASPSPs does not necessarily have to belong to a group and may have its foreign key not set.
Urls	This table contains the base urls of the ASPSP REST API. The base urls may be different for each of the services below: • account information service (AIS) • payment issuer instrument service (PIIS) • payment initiation service (PIS)
Standards	A standard has a name, a version and a template of configs stored in a JSON object, later called implementer options. The implementer options contain a set of specifications, requirements and features that the XS2A standard defines. The ASPSPs which belong to the XS2A standard are free to choose how they want to build their API by setting the implementer options in their preferred way.
Configs	A config can belong to one or multiple banks (ASPSPs) and contains the requirements of the bank, according to its standard. An example for a requirement in the config would be: Bank A expects the request body for payment initiation as a pain.001.001.03 XML element (see 2.1.4), but Bank B expects the request body as a JSON object. Styx has to check and build the request as the bank expects.

Table 5 - Description of the SAD Database Tables

Styx should be able to find the URL of the REST API and prepare the request for execution using the given BIC.

4.2 Development

4.2.1 Supporting Multiple PSD2 XS2A Standard and Performing OAuth SCA Approach

The source code classes structure is similar to the ER Diagram of the database.

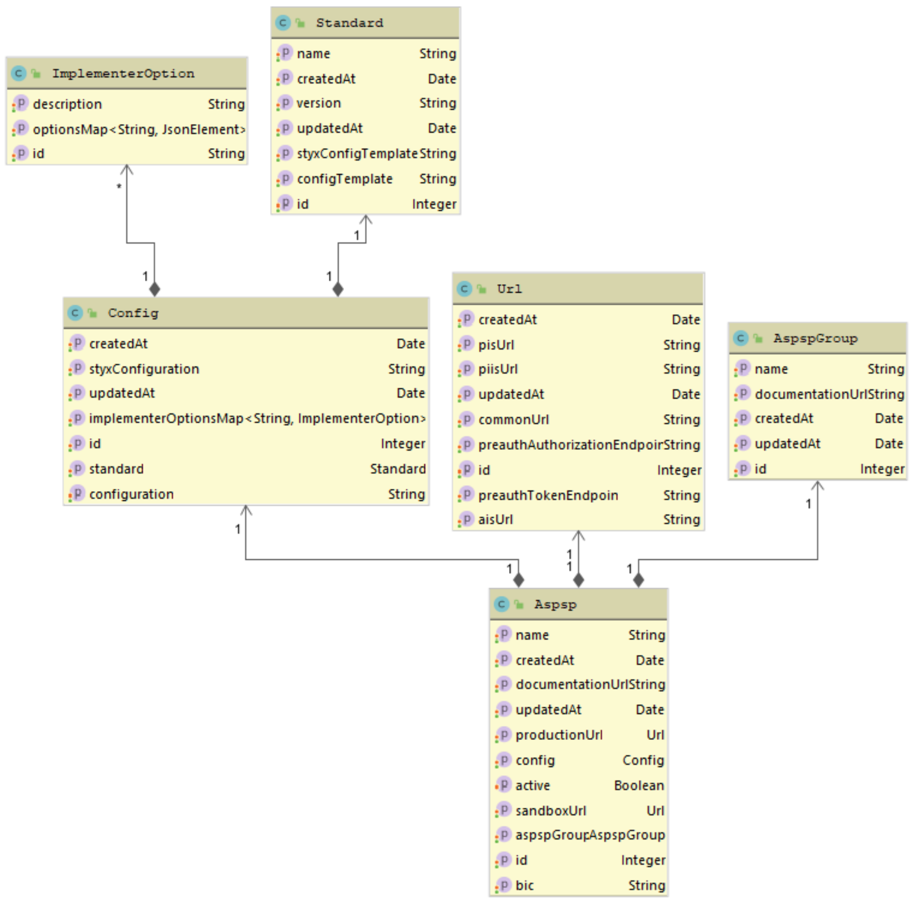


Figure 6 - UML Diagram of the ASPSP Directory

The classes are initialized, after executing a database query, where the system gets a joined record of all the entities above, based on the given BIC. BIC is the unique identifier for an ASPSP.

According to the PSD2 agreement, ASPSPs should offer open interfaces for the following services, regardless of which standard they belong to:

- account information service (AIS)
- payment issuer instrument service (PIIS)

- payment initiation service (PIS)

The initialized ASPSP support building an XS2AStandard instance, which is designed as in Figure 7:

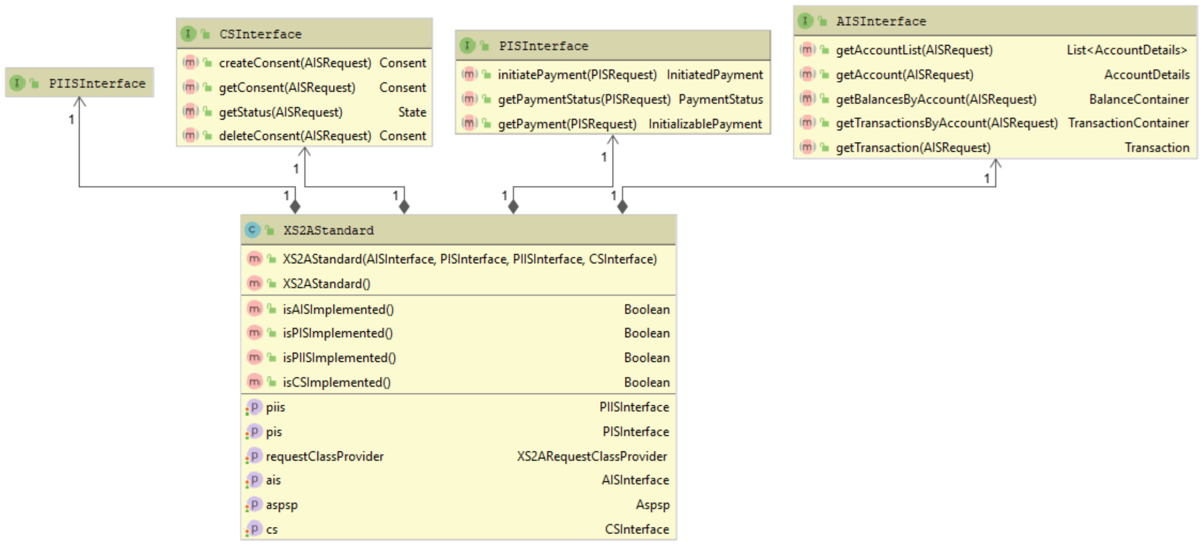

PIISInterface
CSInterface
createConsent(AISRequest) Consent
getConsent(AISRequest) Consent
getStatus(AISRequest) State
deleteConsent(AISRequest) Consent
PISInterface
initiatePayment(PISRequest) InitiatedPayment
getPaymentStatus(PISRequest) PaymentStatus
getPayment(PISRequest) InitializablePayment
AISInterface
getAccountList(AISRequest) List<AccountDetails>
getAccount(AISRequest) AccountDetails
getBalancesByAccount(AISRequest) BalanceContainer
getTransactionsByAccount(AISRequest) TransactionContainer
getTransaction(AISRequest) Transaction
1
XS2AStandard
XS2AStandard(AISInterface, PISInterface, PIISInterface, CSInterface)
XS2AStandard()
isAISImplemented() Boolean
isPISImplemented() Boolean
isPIISImplemented() Boolean
isCSImplemented() Boolean
piis PIISInterface
pis PISInterface
requestClassProvider XS2ARequestClassProvider
ais AISInterface
aspsp Aspsp
cs CSInterface

Figure 7 - XS2AStandard UML Diagram

The information fetched from the database tables will support the initialization of the XS2AStandard instance. This solves the third problem from Chapter 3, ‘Supporting Multiple PSD2 XS2A Standards’. Each standard and its version have their associated classes which implement the interfaces from the diagram above and offer the related functionality. These classes have their own serializers for converting Java objects to request body, and deserializers for converting the response body to Java objects, since the format, the structure and media type are dependent on the standard which the ASPSP follows.

After initializing the XS2AStandard instance, Styx is able to use the implemented method of the services. An example for using the PIS service, for initiating a payment, would look like below:

```
InitiatedPayment initiatedPayment =
getXS2AStandard().getPis().initiatePayment(paymentInitiationRequest);
```

where *paymentInitiationRequest* is an instance of a PISRequest (see diagram on Figure 7).

After the execution of the request above, the ASPSP has received the payment, but the status is still not confirmed until the customer performs Strong Customer Authentication, known as SCA, as explained in 2.1.3.

According to the PSD2 agreement, one of the four ways the user can use to perform the SCA is by following the OAuth Approach. In this case, the customer and the bank combine to create an Authorization Code Grant.

The parties which are involved here are:

- Resource Owner - the customer who makes the transaction
- Client Application - Styx, as a third-party provider
- Resource Provider - the bank itself, which authorizes the TPP to execute payment initiations

The flow of execution of performing SCA using the OAuth approach for a payment initiation is explained in the diagram in Figure 8.

In order to handle the callbacks, Styx also needs to save each session separately. That is why there is a database table, called *oauth_sessions*, with columns as below:

Name	**Data type**	**Not null**	**Unique**
id	bigserial	X	X
authorization_endpoint	text	X	
token_endpoint	text	X	
code_verifier	text	X	X
state	text	X	X
scope	text	X	
access_token	text		X
token_type	text		

refresh_token	text		X
expires_at	timestamp		
created_at	timestamp	X	
authorized_at	timestamp		

Table 6 - Columns of the oauth_sessions Table

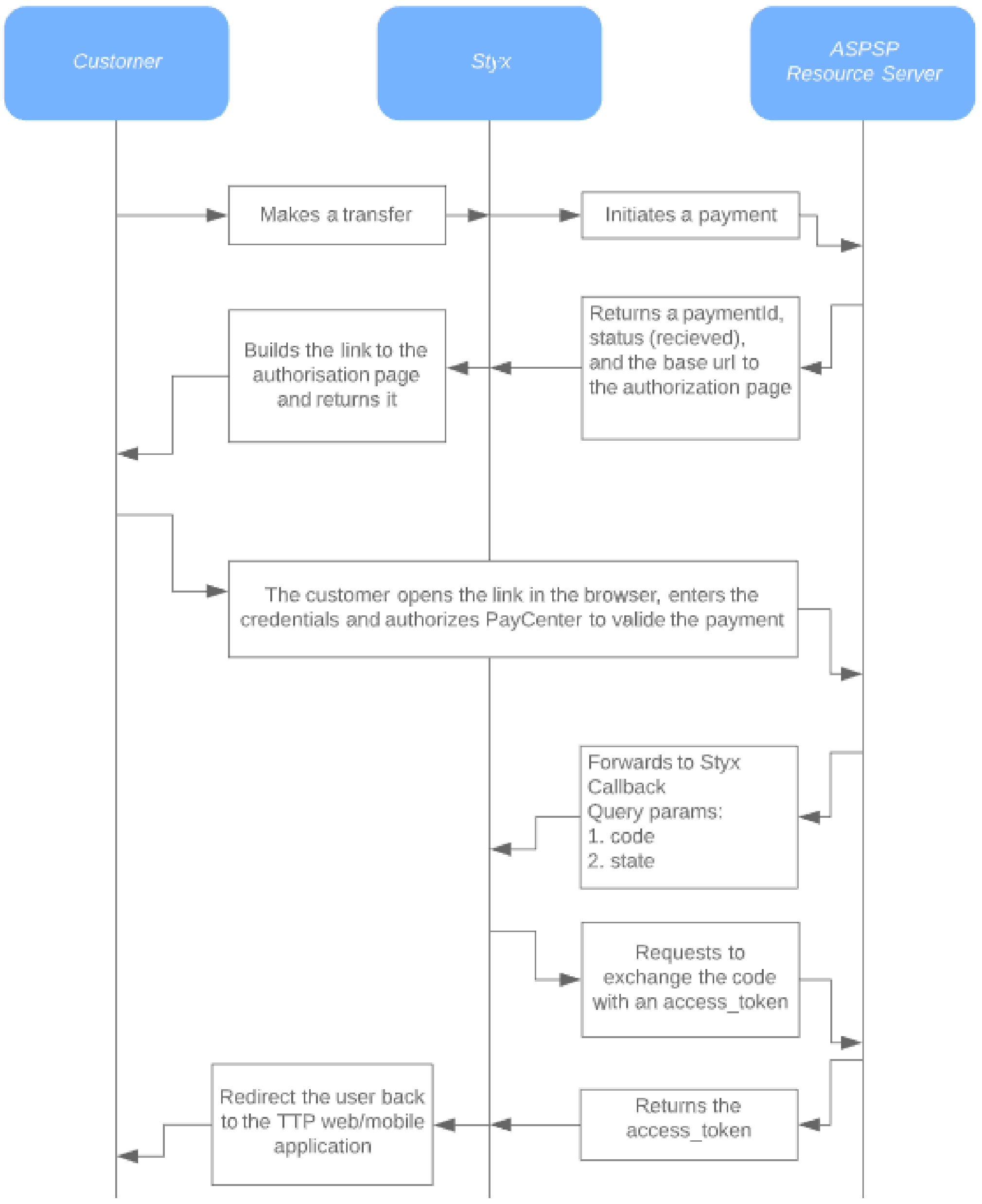

Customer
Styx
ASPSP Resource Server
Makes a transfer
Initiates a payment
Builds the link to the authorisation page and returns it
Returns a paymentId, status (recieved), and the base url to the authorization page
The customer opens the link in the browser, enters the credentials and authorizes PayCenter to valide the payment
Forwards to Styx Callback Query params: 1. code 2. state
Requests to exchange the code with an access_token
Redirect the user back to the TTP web/mobile application
Returns the access_token

Figure 8 - Flow Diagram - OAuth SCA Approach for a Payment Initiation

After the user initializes a payment, the ASPSP returns the paymentId, the status of the payment and the base urls of the authorization and token endpoints.

Styx builds a link, which should be opened by the user in the browser. The link starts with the base url of the authorization endpoint of the ASPSP, followed by 5 parameters in the query string, according to the OAuth standard as explained in 2.2.1 OAuth2. A detailed description for the query parameters, while performing the OAuth SCA Approach, is provided in Table 7.

Name	**Description**
response_type	The value should be "code" since we are performing the Authorization Code Grant.
state	A random UUID generated in Styx, which is later passed in the callback request to identify the oauth session.
client_id	Styx uses the valid eIDAS-certificate from the national certification authority. Styx also received a client_id, which represents Styx during the authorization.
scope	The scope must contain the used service and the resource id. In the case of a payment initiation if would be: "PIS:{paymentId}", where {paymentId} is a UUID, generated by the corresponding ASPSPs, which we received after initiating the payment.
redirect_uri	The URI where we expect to receive the callback request. In our case in would be: "https://styx.paycenter.de/v1/callbacks/oauth"
code_challenge	First we must generate a random String called *code_verifier*. The code_challenge is a URL-safe base64-encoded SHA256 hash of the code_verifier.
code_challenge_method	The hashing method used for the code_verifier. In our case this is *S256*.

Table 7 - Query Parameters o of the SCA Authorization Endpoint

An example of the builded link would look as below:

```
https://authorization-server.com/auth?response_type=code&client_id=PSDDE-
XYZT-
12345&redirect_uri=https://styx.paycenter.de/v1/callbacks/oauth&scope=PIS:%
2093f938ae-cf7a-416e-a08b-3ded2a676ed8&state=0fa70acc-b4a4-4682-8b31-
6ba677ca63ef&code_challenge=ZmlbcEA1WDG2gpVAfQb2RqRTX_yeYrFILqGgdoQdZFA&cod
e_challenge_method=S256;
```

Before returning the response with this link to the client, the oauth-relevant information should be stored in the oauth_sessions table.

```
INSERT INTO oauth_sessions
(
 authrization_endpoint,
 token_endpoint,
 code_verifier,
 state
)
VALUES (
 ‘https://authorization-server.com/auth’,
 ‘https://authorization-server.com/token’,
 ‘MIyl0fy2wPPkTuieErk638BIErocBBOXc06mSE5W-3s’,
 ‘0fa70acc-b4a4-4682-8b31-6ba677ca63ef’
);
```

At this point, the response to the client is returned. The response contains the link to the authorization page, the payment id, and the payment status. The customer will open the authorization page and will confirm that she/he authorizes PayCenter (holder of the eIDAS certificate which Styx uses), to proceed with the transaction in this case. The customer has to enter the bank account credentials, and depending on the bank, maybe also a SMS-Tan for a two-factor authentication.

After a successful authentication, the authorization server of the ASPSP forwards the request to the given *redirect_uri*, where it also adds two query parameters: *code* and *state*. Code represents a String generated by the ASPSP authorization server, which we exchange with the *access_token* in the next step, while state contains the same UUID which we added on the authorization endpoint link.

```
https://styx.paycenter.de/v1/callbacks/oauth?code=ef519d38-d332-38c1-913f-
d3e0887af120-000-7009-7678083&state=0fa70acc-b4a4-4682-8b31-6ba677ca63ef
```

The callback function of Styx is responsible for exchanging the given query parameter *code* with the OAuth Access Token. The function first accepts the query parameter state and searches in the *oauth_sessions* table for the corresponding record.

```
SELECT token_endpoint, code_verifier FROM oauth_sessions WHERE state =
‘0fa70acc-b4a4-4682-8b31-6ba677ca63ef’;
```

Now the POST request on the *token_endpoint* can be executed in order to exchange the code with the access_token. The request body parameters are explained in Table 8.

Parameter name	**Description**
grant_type	“authorization_code”
code	The code which we received as a query parameter.
client_id	Explained in the previous table.
code_verifier	The code_verifier which is stored in the database while building the link to the authorization page.
redirect_uri	The exact redirect_uri which is added as a query parameter while building the link to the authorization page.

Table 8 - Body Parameters for the SCA Token Endpoint

The response of the token endpoint contains the access_token, token_type, refresh_token, expires_in and the scope. After accepting the response, data is stored in the database.

```
UPDATE oauth_sessions SET
 access_token = ‘eyJ0eXAiOi...JsonWebToken’,
 token_type = ‘Bearer’,
 refresh_token = ‘4a39aa45-05fe-4d3a-9b31-98c5feb72621’,
 expires_at = ‘2020-08-04 11:28:32.596605’,
 authorized_at = now()
WHERE state = ‘0fa70acc-b4a4-4682-8b31-6ba677ca63ef’;
```

After the aforementioned procedure is completed, it is considered that the SCA is finished successfully and the payment initiation is accepted. Therefore, Styx can now redirect the request to the client application.

As mentioned in the Chapter 3, SCA should be performed for payment initiation and for consent creation. The payment initiation authorization is what we explained above. The identical flow also applies for authorizing a consent. The scope in that case would be ‘AIS: {consentId}’. This is a solution to Problem ‘3.1 Performing SCA for Consents and Payments Using the OAuth2 Protocol’.

4.2.2 REST API and Client Authentication

As seen on Figure 7, each XS2AStandard has to offer the AIS (AISInterface and CSInterface) and PIS services with their respective methods. Styx offers a REST API with an endpoint for each of the methods in the interfaces. The access layer is developed such that the API server expects a valid access token is present so that the endpoint can be reached.

An access token is a unique key, generated by Styx and it is bound to corresponding service (‘ais’, ‘pis’ or the combined service ‘aispis’). This kind of token will authorize the client to use the REST API endpoints.

There is an authentication endpoint, *POST /v1/auth*, where the clients can get access tokens.

The call expects the following parameters as HTTP headers (see Table 9):

Parameter name	**Description**	**Required**
token	The master token of the client application. Each client application which uses Styx, should be given a master_token during the registration.	X
service	The corresponding service. The available values are: • ais • pis • aispis	X
expiresIn	Defines the time (in seconds) left before an unused Styx Access Token expires. As soon as a token will be used for an actual service (e.g. PIS or AIS) this expiration time is ignored. It affects only tokens that were requested but not used for a Styx service within the given time.This parameter is optional, therefore if not given, the token will be valid for 300 seconds (5 minutes) by default.	

Table 9 - Header Parameters for the Authentication Endpoint

The master token is stored and is the primary key of the database table called *client_apps*, and it is a foreign key in the *tokens* table, where the access tokens are saved.

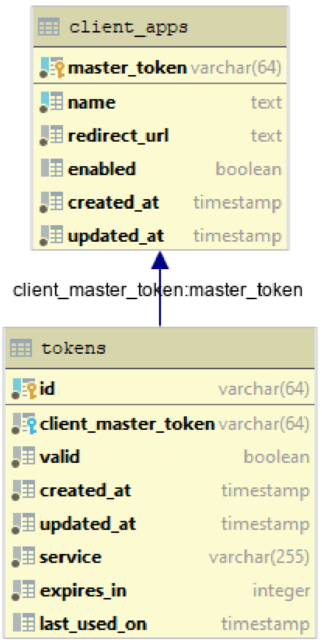


Figure 9 - ER Diagram of the Client Apps and Tokens

During the authentication process Styx generates the access token which is a JWT (JSON Web Token). Styx hashes the generated token using the *SHA256* algorithm, saves the hashed value in the database and returns the plain value as a HTTP response in a JSON body. After the creation, the token is used as a HTTP header in each of the following interactions between client applications and Styx API Server.

A filter class is developed to validate the access token during each request. The class is called as a Java annotation, in every Boundary Class, which requires the authentication. The logic is built to check the validity in this order:

1. Checks if an access token is present during the request
2. Checks if the SHA256-hashed value of the given token is stored in the database
3. Checks if the stored database record has not expired and belongs to an enabled client app
4. Check if the *service* column of the database record matches with the requested endpoint

If the validation passes the 4 aforementioned checks, the *last_used_on* attribute will be updated with the current time and the request will proceed to the corresponding boundary method.

The creation and each usage of the token will be logged for troubleshooting, infrastructure performing and security purposes.

5. Results

5.1 REST API

The REST API of Styx offers endpoints for each method of the PSD2 Services. Each endpoint requires client authentication.

Service	Endpoint Name	Endpoint path	HTTP Method	Requires SCA
AIS	Create a consent	/v1/consents	POST	X
AIS	Get consent details	/v1/consents/{consentId}	GET	
AIS	Get consent status	/v1/consents/{consentId}/status	GET	
AIS	Delete a consent	/v1/consents/{consentId}	DELETE	
AIS	Get list of accounts	/v1/accounts	GET	
AIS	Get account details	/v1/accounts/{accountId}	GET	
AIS	Get list of transactions	/v1/accounts/{accountId}/transactions	GET	
AIS	Get list of balances	/v1/accounts/{accountId}/balances	GET	
PIS	Initiate a single payment	/v1/payments/{payment-product}	POST	X
PIS	Initiate a periodic payment	/v1/periodic-payments/{payment-product}	POST	X

PIS	Initiate a bulk payment	/v1/bulk-payments/{payment-product}	POST	X
PIS	Get payment details	/v1/{payment-service}/{payment-product}/{paymentId}	GET	
PIS	Get payment status	/v1/{payment-service}/{payment-product}/{paymentId}/status	GET	

Table 10 - The List of Styx REST API Endpoints

A detailed explanation of the REST API, the parameters of the requests and the response bodies for the unmentioned endpoints can be found here https://petafuel.github.io/styx/api/.

5.1.1 Technical Documentation of the Authentication Endpoint

This endpoint is used to authenticate the client, and only works for client applications, which are saved in the database. The response contains only the access token, which will be used in further calls.

Path: /v1/auth

HTTP Method: POST

Name	**Parameter type**	**Description**	**Example**	**Required**
token	Header	The master token of the client application.	“ee884d888633653bf305919ee8b6e9b018c1ff3484e2f997e3fdd05603444513”	X
expiresIn	Header	Defines the time (in seconds) left before an unused Styx Access Token expires. As soon as a token will be used for an actual service (e.g. PIS or AIS) this expiration time is ignored. It affects only tokens that were		

		requested but not used for a Styx service within the given time. This parameter is optional, therefore if not given, the token will be valid for 300 seconds (5 minutes) by default.		
service	Header	The corresponding service.	“ais” “pis” “aispis”	X

Table 11 - Header Parameters for the Authentication Endpoint

Request example:

```
POST https://{{styx}}/v1/auth
Accept: */*
Cache-Control: no-cache
Content-Type: application/json
token: 735da99de1bade885fbf1b327e749ea4c75f2d913c81638480ae887c43821bf9
service: pis
expiresIn: 3600

{
    // empty body
}
```

Response example:

Status: 200 OK

```
{
  "token": "74323f356c142.."
}
```

In case if one of the validation rules from 4.2.2 fails, the API Server will return the corresponding status code with the explanation error message.

5.1.2 Technical Documentation of the Payment Initiation Endpoint for a Bank that Supports the OAuth SCA Approach

This endpoint is used to initiate single payments. It is the first step when using the PIS service. The request contains the information of the creditor, the debtor and the transaction itself, such as the currency, the amount, the execution date and so on.

In case of a payment that has to be authorized using the SCA OAuth Approach, the response will contain a link to the authorization page, besides the paymentId and the payment status.

Path: /v1/payments/{payment-product}

HTTP Method: POST

Name	Parameter type	Description	Example	Required
token	Header	Styx access token, generated during the /v1/auth call.	“ee884d888633653bf305919ee8b6e9b018c1ff3484e2f997e3fdd05603444513”	X
redirectPreferred	Header	Boolean value which sets the preferred SCA method for the payment. The value “true” lets the server know that the client prefers to perform a Redirect/OAuth SCA. However, this parameter might be ignored if the ASPSP does not support the preferred SCA approach.	true false	
PSU-Geo-Location	Header	Geo location.	“42.209538,20.740631”	
PSU-Device-ID	Header	UUID identifying the device.	“89uia3ab-c63d-4681-a156-	

			e24bd15d458e”	
PSU-ID	Header	Id of the customer, recognized by the ASPSP.	“123456789”	X
PSU-BIC	Header	BIC (Bank Identifier Code/ISO 9362) of the ASPSP where the user's account belongs. This parameter is important for recognizing the XS2A standard.	“BYLADEM101”	X
payment-service	Path parameter	Payment servicer identifier. The possible values are set by the corresponding standard.	"sepa-credit-transfers" "instant-sepa-credit-transfers"	X
Requested execution date	Body Parameter	The requested execution date for a future payment. The payment will be interpreted as an instant payment if this parameter is not present.	"requestedExecutionDate": “2020-01-01”	

Debtor account	Body Parameter	A JSON object which contains the information about the debtor account.	"debtorAccount": {
			"iban": "DE97120300001033475285"
			}
				X

Creditor account	Body Parameter	A JSON object which contains the information about the creditor account.	"creditorAccount": {
			"iban": "DE97120300001033475286"
			}
				X

Creditor name	Body Parameter	The full name of the creditor.	“creditorName”: “Firstname Lastname”	X

Instructed amount	Body Parameter	Information about the requested transfer amount and the currency.	"instructedAmount" : { "currency": "EUR", "amount": "100" }	X
Remittance Information	Body Parameter	Purpose of the transfer.	"remittanceInformationUnstructured": "Test transfer"	

Table 12 - Parameters for the Single Payment Initiation Endpoint

Request example:

```
POST https://{{styx}}//v1/payments/sepa-credit-transfers
Accept: */*
Cache-Control: no-cache
Content-Type: application/json
token: 0AA39D2EE634CD49FD4958D5F63EC690D5EFC9B0D5F04A0FC894E130E4CD7F27
redirectPreferred: true
PSU-ID: 123456789
PSU-BIC: BYLADEM101

{
   "requestedExecutionDate": "2020-12-31",
   "debtorAccount": "DEDE97120300001033475285",
   "payments":[
     {
      "debtorAccount": {
         "iban": "DE97120300001033475285"
      },
      "instructedAmount": {
        "currency": "EUR",
       "amount": "100"
      },
      "creditorAccount": {
       "iban": "DE97120300001033475286"
      },
      "creditorName": "Test Name",
      "remittanceInformationUnstructured": "Test transfer"
     }
   ]
}
```

Response Example:

Status: 201 Created

```
{
    "transactionStatus": "RCVD",
    "paymentId": "2dbf5051-26ab-4f18-a1b7-3fe8bc13148d",
    "links": {
        "scaOAuth": {
            "href": "https://authorization-
server.com/auth?response_type=code&client_id=PSDDE-XYZT-
12345&redirect_uri=https://styx.paycenter.de/v1/callbacks/oauth&scope=PIS:%
202dbf5051-26ab-4f18-a1b7-3fe8bc13148d&state=0fa70acc-b4a4-4682-8b31-
6ba677ca63ef&code_challenge=ZmlbcEA1WDG2gpVAfQb2RqRTX_yeYrFILqGgdoQdZFA&cod
e_challenge_method=S256;"
            },
        "self": {
            "href": "https://styx.paycenter.de/v1/payments/sepa-credit-
transfers/2dbf5051-26ab-4f18-a1b7-3fe8bc13148d"
        },
        "status": {
            "href": "https://styx.paycenter.de/v1/payments/sepa-credit-
transfers/2dbf5051-26ab-4f18-a1b7-3fe8bc13148d/status"
        }
    }
}
```

In caseswhen the validation fails or something goes wrong during the request, the API Server will return the corresponding status code with the explanation error message. The response will also contain an enum which tells if the error was caused by the client, the server (Styx) or by the corresponding ASPSP.

5.2 Unit testing

The software itself is based on the communication with the interfaces of other software. Unit tests are written to cover the functionality of the serializers, deserializers, and http requests.

Below is a code snippet of a unit test which uses the JUnit Framework to test a payment initialization via the Styx API.

```
@Test
@Category(IntegrationTest.class)
public void initiateSinglePayment_Consors() {
    Invocation.Builder invocationBuilder = target("/v1/payments/sepa-
credit-transfers").request();
    invocationBuilder.header("token", pisAccessToken);
    invocationBuilder.header("PSU-ID", "PSU-Successful");
    invocationBuilder.header("PSU-BIC", "CSDBDE71");
    invocationBuilder.header("PSU-IP-Address", "192.168.8.78");
    invocationBuilder.header("redirectPreferred", true);

    Jsonb jsonb = JsonbBuilder.create();
    SinglePaymentInitiation singlePaymentInitiation =
jsonb.fromJson("{\"payments\":[{\"debtorAccount\":{\"currency\":\"EUR\",\"i
ban\":\"DE60760300800500123456\"},\"instructedAmount\":{\"currency\":\"EUR\
",\"amount\":\"520.00\"},\"creditorAccount\":{\"currency\":\"EUR\",\"iban\"
:\"DE15500105172295759744\"},\"creditorName\":\"WBG\",\"remittanceInformati
onUnstructured\":\"Ref.NumberWBG-1222\",\"requestedExecutionDate\":\"" +
currentDate + "\"}]}", SinglePaymentInitiation.class);
```

```
    Invocation invocation =
invocationBuilder.buildPost(Entity.entity(singlePaymentInitiation,
MediaType.APPLICATION_JSON));
    Response response = invocation.invoke(Response.class);
    Assert.assertEquals(201, response.getStatus());
}
```

A simple test method like this in the software proves that:

- Database connection is correct
- Client Authentication is correct
- Database is filled with data
- Serializers and deserializers are working as expected
- The ASPSP Server is running

6. Discussion & Conclusion

6.1 Advantages and Disadvantages of PSD2

PSD2, as an EU directive for the regulation of payment services and payment service providers, has a lot of advantages.

Advantages

By implementing PSD2:

- Customers can access all their accounts in one place, regardless of which bank their account belongs to.
- Customers will have more alternatives for daily banking without banks. There will be choices of a lot of convenient web or mobile interfaces for online banking.
- The competition between established players will increase. Business models will be more convenient.

Disadvantages

Despite the overall positive impact of the PSD2 directive, there are also some associated difficulties with implementing this directive, including:

- Banks will have to build new systems, which increase costs.
- If banks do not continue to cross-sell their products, because of the reduced time in front of the customer, they will lose profits.

6.2 Styx - Open Source

This project is a product of petaFuel GmbH, and it is at the same time the first open source project of the company. The source code and the installation guidelines can be found in GitHub https://github.com/petafuel/styx. The project is written in Java and uses a PostgreSQL Database Management System. Styx implements the Berlin Group NextGenPSD2 specification, which is led by nearly 40 institutions, banks, banking associations, card issuers, and payment processors. [10]

The next step of the development is to also implement other national and cross-border standardization initiatives such as the PolishAPI, PayPal, UK Open Banking, and so on.

Aim of the system is to integrate the Styx interface as a payment provider into the banking app VIMpay, which is also a product of petaFuelGmbH. Regardless, Styx is also available to be used for private or commercial purposes and for modifications.

6.3 Other SCA Approaches

When banks implement SCA, there are a few popular approaches that different banks can follow. This thesis elaborated the OAuth approach, which is a type of a redirect approach. Banks can also follow the simple Redirect, which is very similar to the OAuth method. The difference between them is that the link to the authorization page is built by the ASPSP and there is no need for a token request during the callback function.

Redirect is one of the most widely used and easiest to implement SCA approaches. When the payment service user (PSU) starts interacting with the TPP, he is redirected to a web interface of an ASPSP for authentication. This means that in a payment scenario, when a user purchases online and wants to make a payment, they will be redirected to their banking website to be authenticated by entering their credentials. A major advantage of a redirect approach is that no additional detailed information about the user is shared with the TPP. To complete the authorization process, a pre-completed transfer screen will be displayed for user confirmation.

The decoupled SCA approach is very similar to the redirect approach. The main difference is that the customer does not redirect to the ASPSP's authentication website. This authentication must be made via an independent application or device from the ASPSP end (e.g. a mobile application). This means that the customer should be authenticated by a method that should be decoupled from the main authentication flow. Similar to the redirect approach, it is easy to implement and an effective approach because the interaction is between the ASPSP and PSU.

An embedded SCA approach means that the process is fully automated, and the payment is initiated by the TPP on behalf of the customer. The user shares the credentials with the TPP, who authenticates and initiates the payment in the background and embeds the interaction with the ASPSP.A service provider should be registered and licensed to perform this action as the entire approach is taken by the TPP, which is not allowed to store shared credentials from the power supply. To add to the security factor to this approach, the costumer may need to fill in an SCA, like a one-time password to reduce fraud. [3]

7. References

[1] W. Machielse, “NextGenPSD2: More than 1700 banks have met the first regulatory PSD2 deadline on time,” 02-Apr-2019. [Online]. Available: https://nisp.online/nextgenpsd2-more-than-1700-banks-have-met-the-first-regulatory-psd2-deadline-on-time/. [Accessed: 14-Sep-2020].

[2] Open Banking Europe, “Understanding Internet Security & eIDAS Certificates,” 2019. [Online]. Available: https://www.openbankingeurope.eu/media/1177/preta-obe-mg-001-004-psd2-xs2a-understanding-internet-security-eidas-certificates-guide.pdf. [Accessed: 14-Sep-2020].

[3] The European Commission, “Commission Delegated Regulation (Eu) 2018/389,” *Official Journal of the European Union*, vol. L, no. 69, pp. 23–43, Mar. 2018. [Online]. Available:https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32018R0389&from=EN

[4] European Payment Council, EPC, Brussels, tech., 2018. Available: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2018-11/EPC121-16%20SCT%20Inst%20C2B%202019%20V1.0.pdf

[5] M. Anicas, “An Introduction to OAuth 2,” DigitalOcean, 21-Jul-2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. [Accessed: 14-Sep-2020].

[6] D. Hart, Ed., “The OAuth 2.0 Authorization Framework,” IETF Tools, Oct-2012. [Online]. Available: https://tools.ietf.org/html/rfc6749. [Accessed: 14-Sep-2020].

[7] D. Premanantha, “Strong Customer Authentication and Dynamic Linking for PSD2,” WSO2, 19-Jun-2019. [Online]. Available: https://wso2.com/library/articles/2019/06/strong-customer-authentication-and-dynamic-linking-for-psd2/. [Accessed: 14-Sep-2020].

[8] L. Vogel, “Unit Testing with JUnit - Tutorial,” vogella.com, 21-Jun-2016. [Online]. Available: https://www.vogella.com/tutorials/JUnit/article.html. [Accessed: 14-Sep-2020].

[9] A. Constantinovici, W. Machielse, and O. Scheja, “What does the future hold for TPPs and banks in a post-PSD2 era and the art of standardising APIs,” The Paypers, 29-Nov-2019. Available: https://thepaypers.com/interviews/what-does-the-future-hold-for-tpps-and-banks-in-a-post-psd2-era-and-the-art-of-standardising-apis-interview-with-the-berlin-group. [Accessed: 14-Sep-2020].

[10] “The list of PSD-2 banking API standards,” 12-Aug-2019. [Online]. Available: https://banqup.com/index.php/blog/technology/11-business-6. [Accessed: 15-Sep-2020].