FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Unified SDK for integration with Erste group PSD2 API
Student:	Artem Kravchenko
Supervisor:	Ing. Nikolay Barbariyskiy
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

Introduce the Revised Payment Services Directive (PSD2). Introduce the different technical documents available for PSD2 such as the Berlin Group NextGenPSD2 Specifications.

The Erste SDK project aims to provide a unified interface for developers meaning to use the PSD2 APIs of different banks of the Erste Group. Introduce the Erste SDK project, its goals and strategies. Identify and analyse discrepancies between different business flows, domain models, and error handling. Propose a unified user-centric interface mitigating these differences. Introduce and explain the concept of user-centric interfaces.

The practical part of the thesis will focus mainly on the Authorisation flow and Account Information Service (AIS) part of the Erste SDK project. Analyse, model and implement the authorisation and account sharing consent flows, AIS API, and error handling. The code should adhere to open source standards and conventions and be thoroughly tested and testable.

Electronically approved by Ing. Michal Valenta, Ph.D. on 22 April 2021 in Prague.

Bachelor's thesis

UNIFIED SDK FOR INTEGRATION WITH ERSTE GROUP PSD2 API

Artem Kravchenko

Faculty of Information Technology, CTU in Prague
Department of Software Engineering
Supervisor: Ing. Nikolay Barbariyskiy
June 27, 2021

Contents

List of Figures

List of Tables

Seznam výpisů kódu

I would like to thank my thesis supervisor Ing. Nikolay Barbariyskiy for proposing this topic and always providing helpful advice no matter if we were discussing the thesis or programming. I would also like to thank BcA. Pavel Michalík for sharing his expertise with financial technology and especially PSD2. Finally my heartfelt thank you to Andrea Švancarová for supporting and motivating me throughout difficult times.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

Done in Prague, 27 June 2021 .....

Abstrakt

Hlavním cílem práce je představit projekt Erste SDK na integraci PSD2 API Erste Group a uvést legislativu PSD2 a PSD2 API v rozsahu potřebném pro pochopení projektu. Kromě jádra projektu je ještě podrobněji vysvětlena autorizace a AIS API. Práce představí ke každému API vybranou architekturu pro sjednocení, popíše řešení nastalých problémů a popíše výsledné unifikované modely a procesy. Následně se zhodnotí přínos projektu oproti manuální integraci bankovních API bez použití SDK.

Klíčová slova PSD2, PSD, FinTech, Unifikace API, SDK, Bankovní aplikace, Erste Group, Erste, Analýza API, OAuth

Abstract

The main goal of this work is to introduce the Erste SDK project for unifying the PSD2 API of Erste Group, and provide an overview of PSD2 and PSD2 API to better understand the SDK project. Along with project core, authorization and AIS API will be explained in detail. A solution architecture will be introduced for every API along with describing encountered unification discrepancies and an explanation of the unified models and flows. Finally the benefits of the project will be examined and compared to manual integration of bank APIs without using the SDK.

Keywords PSD2, PSD, FinTech, API unification, SDK, Banking application, Erste Group, Erste, API Analysis, OAuth

Acronyms

PSD2	Revised Payment Services Directive
RTS	Regulatory Technical Standards
AIS	Account Information Service
PIS	Payment Initiation Service
CIS	Card Issuing Service
AISP	Account Information Service Provider
PISP	Payment Initiation Service Provider
CISP	Card Issuing Service Provider
ASPSP	Account Servicing Payment Service Providers
TPP	Third Party Provider
TPPSP	Third Party Payment Service Providers
IDP	Identity Provider
PSU	Payment Service User
SCA	Strong Customer Authorization
CSC	Common and Secure Communication
QWAC	Qualified website authentication certificate
CSAS	Česká spořitelna a.s., Czechia
SLSP	Slovenská sporiteĺňa, a.s., Slovakia
BCR	Banka Comerciala Romana, Romania
EBH	Erste Bank Hungary Zrt., Hungary
EBC	Erste&Steiermärkische Bank d.d., Croatia
EBOE	Erste Bank and Sparkassen, Austria
EGB	Erste Group

Chapter 1
Introduction

This thesis will introduce the Erste SDK project for integration of Erste Group PSD2 API. The Erste SDK project aims to provide an open source unified interface for third party developers meaning to use the PSD2 APIs of the different banks of the Erste Group. Even though the banks belong to the same banking group, the standards and needs specific to their local regions mean their PSD2 implementation differs in small and big details alike. The differences will be examined in detail in chapters focusing on specific parts of the API.

This project is composed of two parts. The Java SDK and the functionally identical Node.js SDK. This thesis will focus exclusively on the Java project. The project is still in active development and not officially released yet. It will be available on Maven Central and on Github later this year. Given that the Java project needs to be as close as possible to the Node.js project, the full documentation is yet to be done as it should be unified across both parts. The introduction in this thesis, existing documentation in the project and on the Erste developer portal should provide enough context and information for the SDK to be usable now. However using it in real projects should be postponed until official release.

1.1 Thesis Structure

The thesis will cover the basics of PSD2 required for the purposes of the project and give a quick overview of the Erste Group. Choice of programming language, dependencies and specific goals of the project will be explained before delving deeper into architectural choices. Basic overview of the project and its core parts is then followed by chapters focusing on the solution architecture of the specific parts. The challenges encountered during unification of the APIs of different Erste banks, such as business model and flow discrepancies, will be explained in their respective chapters.

- Authorization - covering the basics of OAuth, consents and the unified flows.
- AIS - focusing on the issues such as differing models and mocking non existing endpoints.

The thesis is then concluded by showing examples of the SDK in use, explaining possible benefits and summarizing personal experience with the project.

Chapter 2

Revised Payment Services Directive (PSD2)

> This legislation is a step towards a digital single market; it will benefit consumers and businesses, and help the economy grow.
>
> *Jonathan Hill, former European Commissioner*

This chapter will cover the basics of PSD2 needed to better understand the context of what the Erste SDK project is based on. It will introduce the history of PSD2, its predecessors and alternatives. It will explain what the different technical documents surrounding the regulation are.

2.1 Introduction

The main goal of PSD2 is to simplify and secure online payments by requiring strong customer authentication (SCA), which will be explained in a later section; and to facilitate innovation [4] and competition by making it easier for third party providers (TPPs) to enter the online financial market in Europe. In this effect it overwrites the existing Payment Services Directive effective from 2007.

PSD2 was proposed in the European Parliament and passed by the Council of the European Union in 2015 and came into effect in 2019, but the requirements for SCA were extended until 31 December 2020 [3].

2.2 History

PSD2 builds on the existing Payment Services Directive and the Single Euro Payments Area initiatives of the EU.

2.2.1 Payment Service Directive (PSD)

The original PSD paved way for a new category of financial services called payment service providers [24]. This required a regulatory framework allowing new organizations to provide payment services that previously only banks, central banks and governments could. Those companies, commonly referred to as *fintech* (financial technology), could apply for a so-called passport certificate in one EU country to provide payment services all over Europe. The regulation also helped the adoption of SEPA, brought more transparency and information to consumers, set more strict limits on maximum payment processing time and fees, and strengthened refund rights among other things [5].

2.2.2 Single Euro Payments Area (SEPA)

Since 2008 the aim of SEPA was to speed up and simplify bank transfers in euro currency. It unified the different payment rules and standards, which allowed easy payments all over the eurozone just like national payments, therefore bypass costly and timely international payment fees [2, 21]. It introduced guaranteed one day and instant transfers with SEPA Instant Credit Transfer [7]. This was expected to reduce the total cost of banking and processing payments by up to €100bn a year [34]. As SEPA only covered euro transactions, European transactions between countries not in the eurozone were not affected and remained as complicated as before.

2.3 Technical Documents

2.3.1 Regulatory Technical Standards (RTS)

In order to codify the requirements of PSD2 RTS was adopted based on a draft submitted by the European Banking Authority [4]. RTS introduces two key security measures that TPPs and banks must observe when providing their services. The first of those is Strong Customer Authentication regulating user authentication and the second one is Common Secure Communication establishing different PSD2 actors and methods of their interactions [10].

2.3.1.1 Strong customer authentication (SCA)

To prove their identity users will need to provide at least two separate elements out of these three [4]:

- Something they know - a password, a PIN.
- Something they own - a card, a mobile phone.
- Something they are - biometrics (a fingerprint).

RTS codifies that SCA must be used when initiating payments.

2.3.1.2 Common Secure Communication

Establishes terminology, hierarchy and interactions between clients, banks and applications [15]. Introduces the following terms:

- Third Party Payment Service Providers (TPPSP) - Third parties (TPP) which do not own the client's accounts, but can provide services.
  - Account Information Service Provider (AISP) - TPP providing an overview of client's bank accounts.

  - Payment Initiation Service Provider (PISP) - TPP able to initiate payments on client's behalf. The payment still needs to be confirmed by the client using SCA before it is accepted by the bank.
- Account Servicing Payment Service Provider (ASPSP) - the providers of the payment accounts for customers, including banks and other payment institutions.
- Customer Consent - clients must give ASPSPs explicit consent for TPPs to use the API. It's not possible for AISP to access data or PISP to initiate payment from accounts where consent to do so was not given.
- Common communication - ASPSPs must allow TPPs to access their PSD2 API without difficulty to prevent insecure screen scraping. Screen scraping is a method where the client gives TPPs their account login information and TPPs automatically extract data or initiate payments from client's mobile banking application. With PSD2 the European Commission effectively prohibited screen scraping and enforced PSD2 APIs for such communication [38].
- Secure communication - ASPSPs must communicate with TPPs using secure messaging when accessing client's data at all times. This secure channel will also enable the parties to identify each other using certificates [6]. The certificates will be explained in the next section covering eIDAS.

2.3.2 Electronic Identification, Authentication & Trust Services Regulation (eIDAS)

While not a part of PSD2, eIDAS is an important regulation for the concept of Secure communication of the PSD2 RTS. It was announced in July 2014 as a regulation on electronic identifications, signatures and safe electronic transactions in the internal EU market [33, 9].

For the purposes of this thesis the most important concept is the Qualified website authentication certificate (QWAC). This type of certificate is meant for use in payment systems in accordance with the PSD2 guidelines to prove the identity of the holder. Unlike commonly used SSL/TLS certificates QWAC is a qualified certificate [28]. A qualified certificate is issued by a qualified trust service provider, which is a legal entity creating and validating electronic certificates and signatures. It is recognized as a trusted identification by public administrations across the EU. Therefore a qualified electronic signature and certificate hold the samme weight as a paper signature or identification [8].

2.4 PSD2 Implementations

Even though the RTS helped answer some uncertainties about PSD2 implementations, the regulation is still a long way away from being a guide for banks to know exactly which endpoints accepting which parameters to implement for their PSD2 API since the regulation only specifies which data should be made available by the API. The end result is that there could be potentially limitless different APIs which are all PSD2 compliant [27].

This is especially pronounced as PSD2 allows some leeway for different banks to not provide or accept information that is not already present in their business flows. This will become very apparent when examining different Erste banks' PSD2 implementations further in this work.

In order to mitigate this issue different banking groups have proposed how PSD2 implementations might look in practice. The following sections will introduce three of those proposals.

2.4.1 Berlin Group Standard - NextGenPSD2

Berlin Group first formed in 2004 surrounding the creation of SEPA. Berlin Group believes that the national banking scheme differences are not just a nuisance to be dealt with by unification, but that those differences allow for the highest level of efficiency in the banking space for those countries. Therefore any unification effort should be made with this in mind [17].

NextGenPSD2 is a detailed technical implementation guideline that allows implementation of a PSD2 compliant banking interface. The majority of Erste PSD2 implementations are based on this proposal.

2.4.2 Open Banking (United Kingdom)

The UK specific PSD2 standard was adopted by over 90% of their market even though the UK is no longer a member of the EU or the European Economic Area and therefore exempt from the regulation [11].

2.4.3 Czech Open Banking Standard - ČOBS

The Czech national standard proposed by the Czech banking association, currently on version 4.1 from 25 May 2020. A 449 page document proposing different endpoints, entity models and possible error codes in great detail [12]. Erste CSAS PSD2 implementation is based on this document.

2.5 PSD2 API

PSD2 defines three distinct APIs each serving a different need [29]. The following section will briefly examine every API defined.

2.5.1 Account Information Service (AIS)

AIS enables TPPs to access account information such as the IBAN, transaction history, and balance of a client's account.

2.5.2 Payment Initiation Service (PIS)

The PIS API enables TPPs to initiate a payment from the client's accounts. Each payment must still be confirmed by the client using SCA to avoid misuse.

2.5.3 Confirmation of Funds - Card Issuing Service (CIS)

The functionally simplest API defined by PSD2, CIS is used only for confirmation of funds on a payment card. TPP sends a request asking if the client has sufficient funds for a transaction of specified amount and the ASPSP either confirms or denies.

Chapter 3

Erste Group

Erste Group is one of the largest financial providers in Central and Eastern Europe with around 16.2 million clients in 7 countries.[18, 20] It was originally founded in 1819 in Austria as a savings bank. In 1997 the Erste Group went public and started expansion beyond Austria to other countries. To list a few countries of interest:

- Hungary (EBH) - Erste Bank Hungary Zrt.
- Slovakia (SLSP) - Slovenská sporiteĺňa, a.s.
- Romania (BCR) - Banca Comercială Română.
- Croatia (EBC) - Erste&Steiermärkische Bank d.d.
- Czech Republic (CSAS) - Česká spořitelna a.s. is a bank with the largest amount of clients in the Czech Republic with around 4.5 million customers. Among other services it also provides the biggest ATM network in the Czech Republic.[13]
- Serbia - as a non EU member state it is exempt from the PSD2 regulation, therefore not a part of the project.

The Group focuses on both retail and corporate banking with offerings such as mortgages, loans, advisory services and access to capital markets.

Figure 3.1 Map of Erste Group activities in Europe[19]

Chapter 4

Erste SDK

4.1 Goals

The goal of the project is to mitigate the differences between banks so that developers wanting to communicate with Erste banks would ideally only need to learn to use the SDK without studying the developer portal documentation for every bank they interact with. The Erste SDK should be very light on dependencies as to minimize the adoption friction as much as possible, and be created with the idea of user-centered interfaces in mind. The project should adhere to open source standards, be thoroughly tested and testable.

Analysis in further chapters proved that the goal of completely erasing the differences between the banks is impossible at the very least because of the different authorization flows among many other reasons. In short all banks support authorization via the OAuth Authorization Code Grant, but only some banks support the OAuth Implicit Grant as well. Aside from that three of the seven banks also deal with Consents for interaction with specific accounts as a part of this flow, while the rest have the consent mechanic integrated in the login flow, as the bank shows a message that logging in gives the TPP consent to access the account information of their accounts.

The majority of the added value of this project is in unifying the endpoints, methods, models and error handling across the domain as much as possible. Developers still need to reference the Erste developer portal or the project documentation to confirm which specific arguments are required for requests, or conversely which attributes are always included in the response model and which are optional. However all of the busywork of configuring endpoints, enforcing arguments, converting responses and handling different error mappings across banks is done for them.

4.2 Differences between SDK and Library

The main difference between a library and an SDK is that a library aims to provide functionality in one area or for one task. On the other hand an SDK contains all necessary tools required for development in a specific area such as a hardware platform, operating system or programming language [23, 36, 35]. Given that there isn't a concrete definition that draws a clear line between the two it could be just as successfully argued that this project is a library similar to Google API Client Library that allows users to interact with Google services such as listing calendar events or uploading files to Google Drive [16]. Same as it could be said that this is an SDK similar

to Facebook SDK that provides functionality such as wrapping API calls or enabling to use the Like Button on a website [14].

While this definition looks good in theory in practice it is not concrete enough. Therefore where exactly the Erste SDK project is a question without a concrete answer. Arguments could be made for both sides and the project self labels as an SDK containing everything needed to develop an Erste PSD2 application.

4.3 User-centric interface

> User-Centered Design (UCD) is a user interface design process that focuses on usability goals, user characteristics, environment, tasks, and workflow in the design of an interface. UCD follows a series of well-defined methods and techniques for analysis, design, and evaluation of mainstream hardware, software, and web interfaces. The UCD process is an iterative process, where design and evaluation steps are built in from the first stage of projects, through implementation.
>
> *Shawn Lawton Henry and Mary Martinson, Accessibility in User-Centered Design*

User-centered interface is an important value to the Erste SDK project. It is based on the term user-centered design and aims to extract the benefits of this approach in the programming context.

User-centered design is based on the principle of adapting the product to what the user wants, expects, and is used to. The product should be designed with frequent feedback from the user from start to finish [37].

The main takeaway for the Erste SDK Project is abstraction of implementational details of all the different banks and unification of business models. The ideal outcome is to enable the end user to focus on delivering business value as much as possible instead of working around API differences. In this sense it should aim to complement the current Erste developer portal documentation for each bank. The end user of the SDK should only spend time integrating and learning the SDK for a bank once, and then have access to all the other banks with minimal effort.

4.4 Implementation

4.4.1 Technology

The project was contracted in Java, the specific version was left up to our discretion. The programming language used for this project is Java 8. To this day Java is one of the leading enterprise programming languages [25]. Java 8 released in March 2014 [30] is still the most widely used Java version. 75% of Java developers reported using Java 8 regularly and 44% used Java

EE 8 regularly in an enterprise context [26]. In both metrics Java 8 was the clear winner and was therefore chosen as the version for the Java. Seeing as Java is backwards compatible there was no question of the cost versus benefit ratio of having new language features against the missed target audience that would not be able to use the SDK written on a newer version.

The question of a package manager came down only to personal preferences, as that has minimal, if any, effect on the end user. In the end Maven was chosen as it was proven on countless projects, and we had ample experience with it.

4.4.2 Dependencies

One of the other goals of the project was to also keep the dependency list as light as possible as to minimize the risk of version conflicts and forcing users to use libraries they do not want or need. This was sometimes done at a cost to ease of development, for example the Apache HTTP client is somewhat minimalistic and lacks certain usability features compared to the Spring HTTP solution. This will be covered in more detail in the chapter pertaining to the HttpWrapper class.

Dependencies thus far only include Jackson databind for conversion to and from JSON, and Apache HTTP Client. Java 8 is lacking easy to use native support in these two areas, which was addressed in newer versions of Java. For testing TestNG, Mockito and PowerMock were used, however only in Maven testing scope, therefore they are not included in the runtime classpath but only in compilation and testing.

Chapter 5

Overview

This chapter will go into more detail about the different components of Erste SDK. It will explain the rationale between the architecture of the project and provide a basis for all parts of the project.

Once concept that the architecture of Erste SDK project is based upon is that all the banks aim to provide the same business service. The differences between them being mainly implementational. While one bank's OAuth flow requires PKCE and anoter one's does not, the OAuth logic does not change between them. The same logic applies to the rest of the provided APIs or even the entity models of the banks. An account returned by CSAS is the same business entity returned by SLSP, even if both of them are represented with different attributes.

The main solution chosen for the majority of the unification challenges was therefore composed of three parts:

- Business interface - an interface representing the business meaning of the API or model. This is the only part of the implementation relevant to the end user of the SDK project. To have a concrete example: When the end user wants to obtain an access token from a new bank, it's not relevant for them that their bank uses a different representation of an access token response, or that it has a different endpoint. They only need to instantiate a new SDK for their bank.
- Base implementation - an abstract class representing the main implementational logic of the problem. It implements the business interface. This is the *main* class for the given challenge domain. It contains the driving logic that is common amongs all (or most) banks. Most often this is the class that will initiate communication with Erste API. The interchangeable specific attributes such as the concrete endpoint used for an API call is usually passed to the base class via constructors using configuration classes or plain values.

  To summarize, base implementation contains the core implementation for the interface. It parses data passed by the user. Fills in headers, parameters and body of the request and sends it via `HttpWrapper` to the correct API endpoint. Then it will parse and map the responses to then end user models and returns them.
- Bank implementation - Concrete bank implementation dealing with specific challenges of a given bank It extends the base implementation class. The specific challenges are usually solved in the form of overriding methods from the base implementation class. An example for that is BCR using HTTP POST method instead of GET like other banks when exchanging the authorization code.

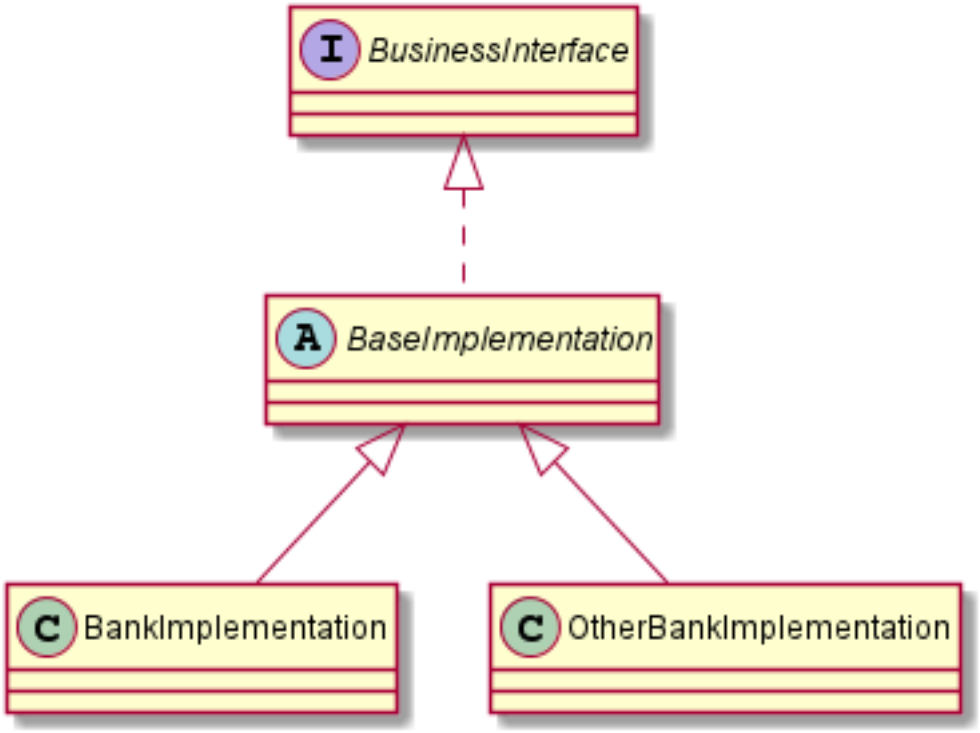

Figure 5.1 Base architecture overview

5.1 Erste SDK

`ErsteSDK` The main interface that is used by the end user is the ErsteSDK interface. It is a wrapper object containing specific domain focused interfaces, such as the `AuthorizationCodeGrant` for OAuth Authorization Code Grant flow, or `AISProvider` for interacting with AIS API. It should not be instantiated with a `new` keyword, instead `ErsteSDKBuilder` using builder pattern was created for instantiating and configuring the SDK.

It also contains `ErsteSDKConfiguration` object representing the configuration of the SDK. It enables the user to quickly check or change the configuration. It will be explained in more detail in `ErsteSDKBuilder` section.

5.1.1 Authorization Grant

Authorization flows is explained in detail in the Authorization unification chapter. This section will provide a simple overview.

`AuthorizationGrant` is the base business interface for authorization. It contains the `generateAuthorizationUri` method responsible for creating the authorziation URL the bank client is redirected to in OAuth flow. This method is used in both the Implicit Grant and the Code Grant flows, therefore both implementations implement this method. The logic is contained in the `BaseAuthorizationGrant` abstract class. Each implementation calls this method with different parameters necessary for the respective flows.

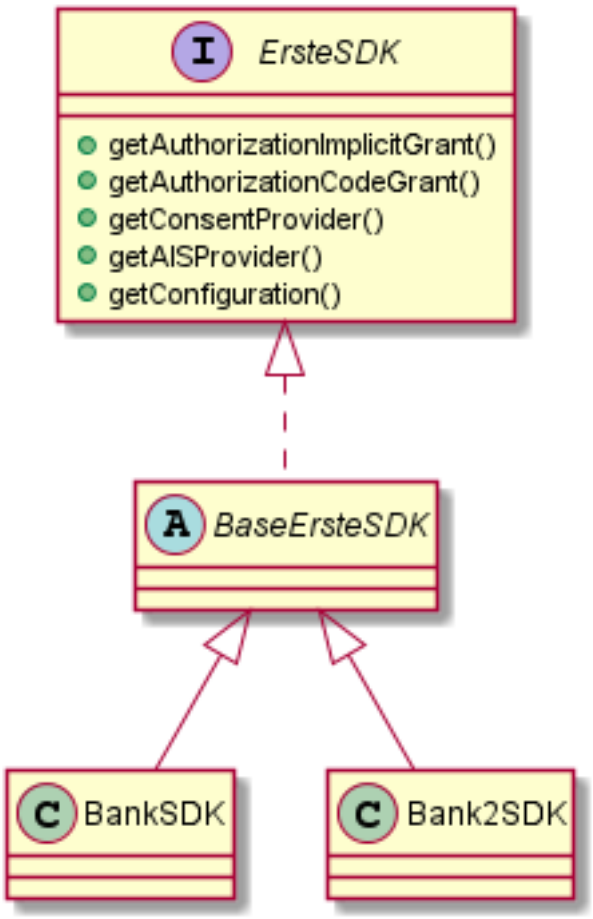

Figure 5.2 Erste SDK Architecture

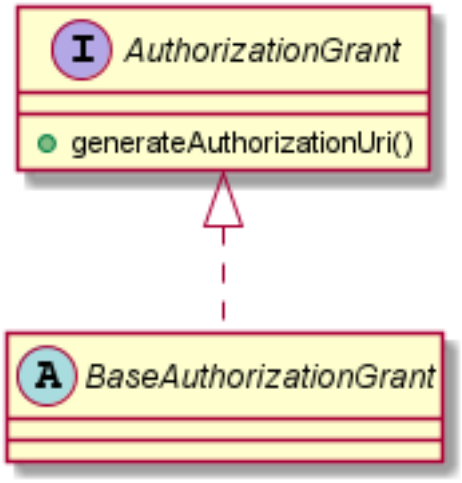

Figure 5.3 Authorization Grant overview

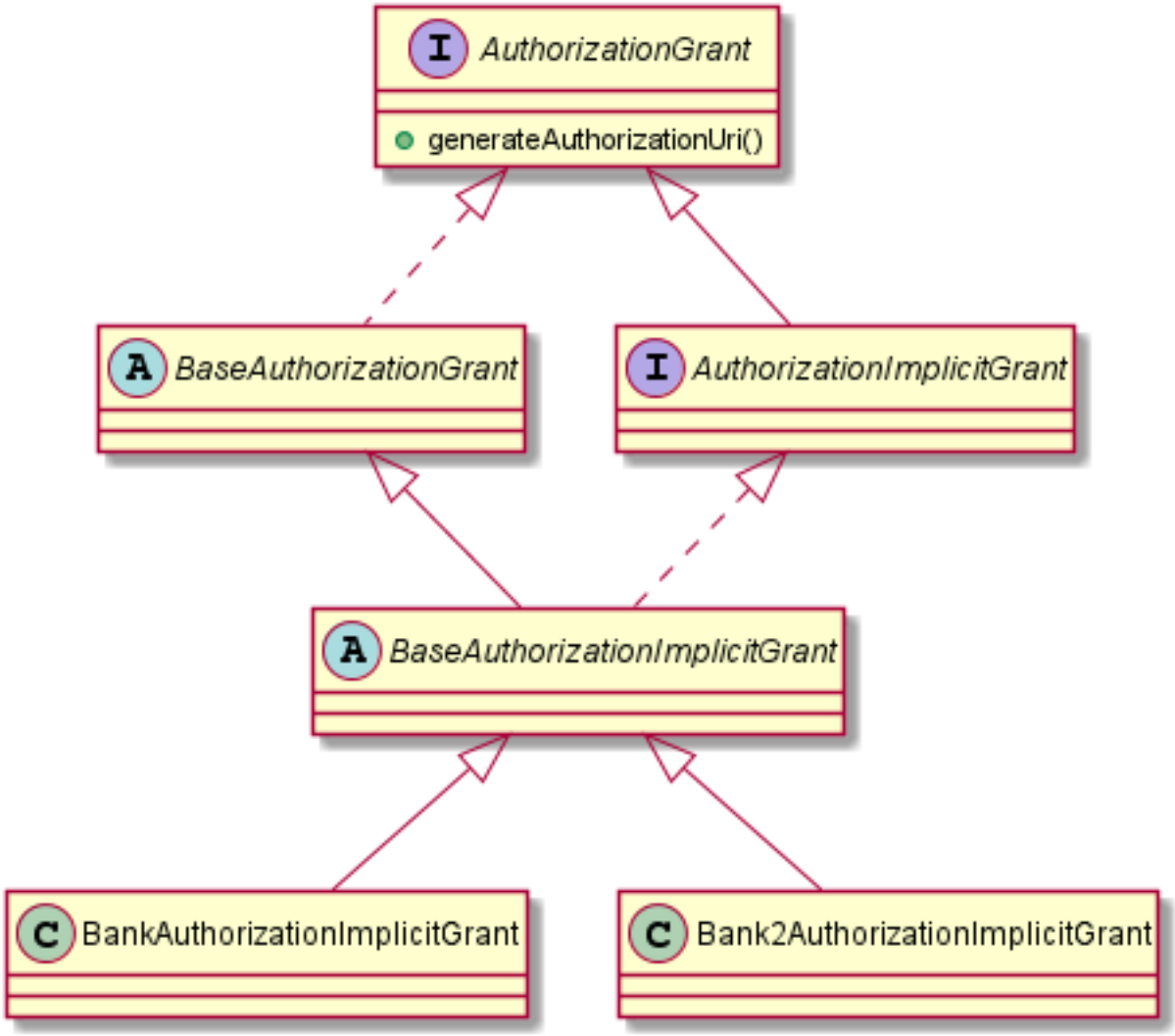

Figure 5.4 Authorization Implicit Grant overview

5.1.2 Authorization Implicit Grant

`AuthorizationImplicitGrant` is only used for clear naming. It does not provide any new functions from the base `AuthorizationGrant` interface. The main purpose behind moving the logic one level up is to prevent confusion that would arise from having `AuthorizationCodeGrant` inherit from `AuthorizationImplicitGrant` directly. Or having Implicit grant chosen as default, if only the abstraction one level up existed.

5.1.3 Authorization Code Grant

`AuthorizationCodeGrant` contains methods specific for the Authorization Code Grant flow - namely `authorization` used to exchange the authorization code for the access token and refresh token. And `refresh` to exchange the refresh token for a new access token.

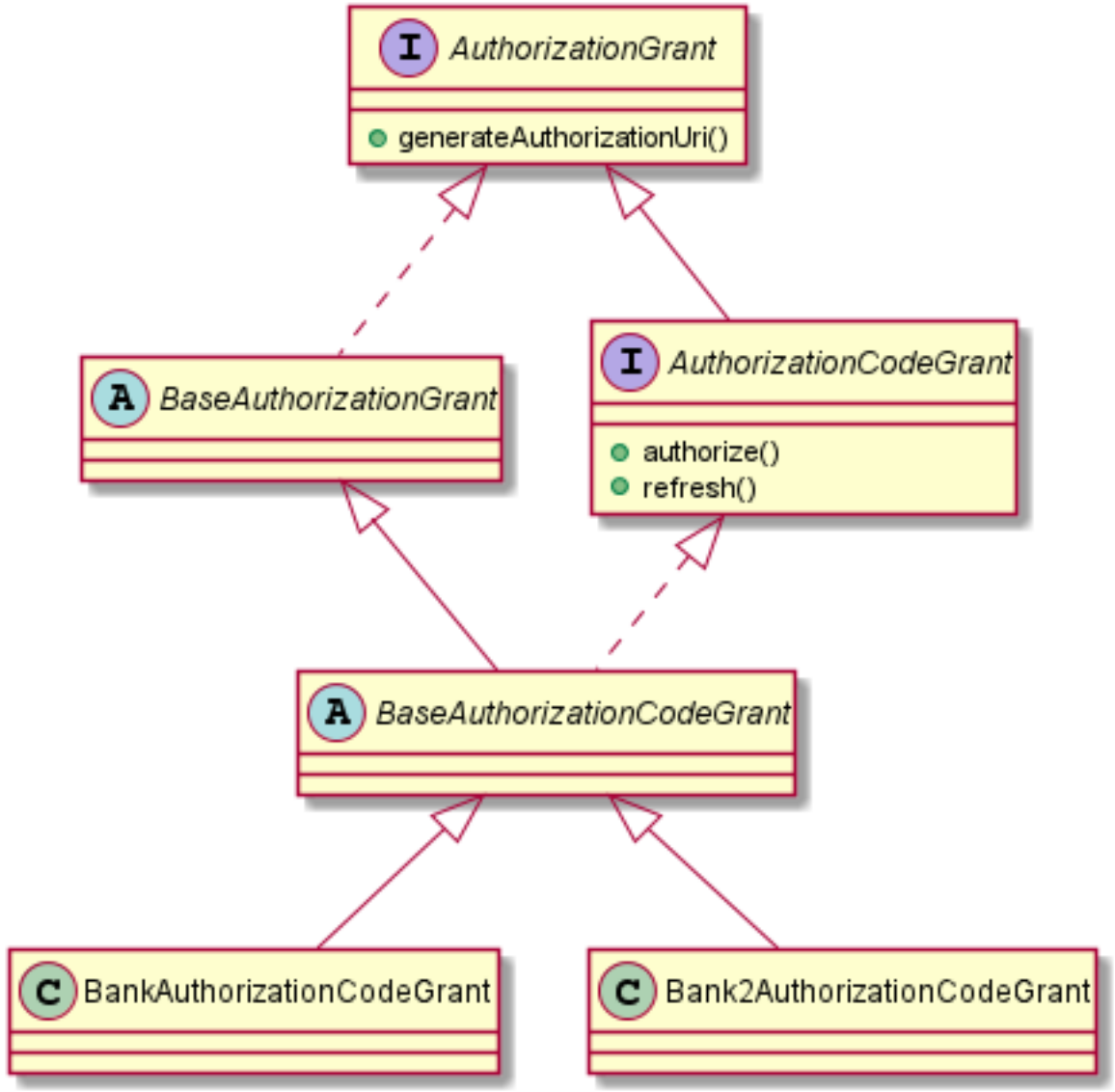

Figure 5.5 Authorization Code Grant overview

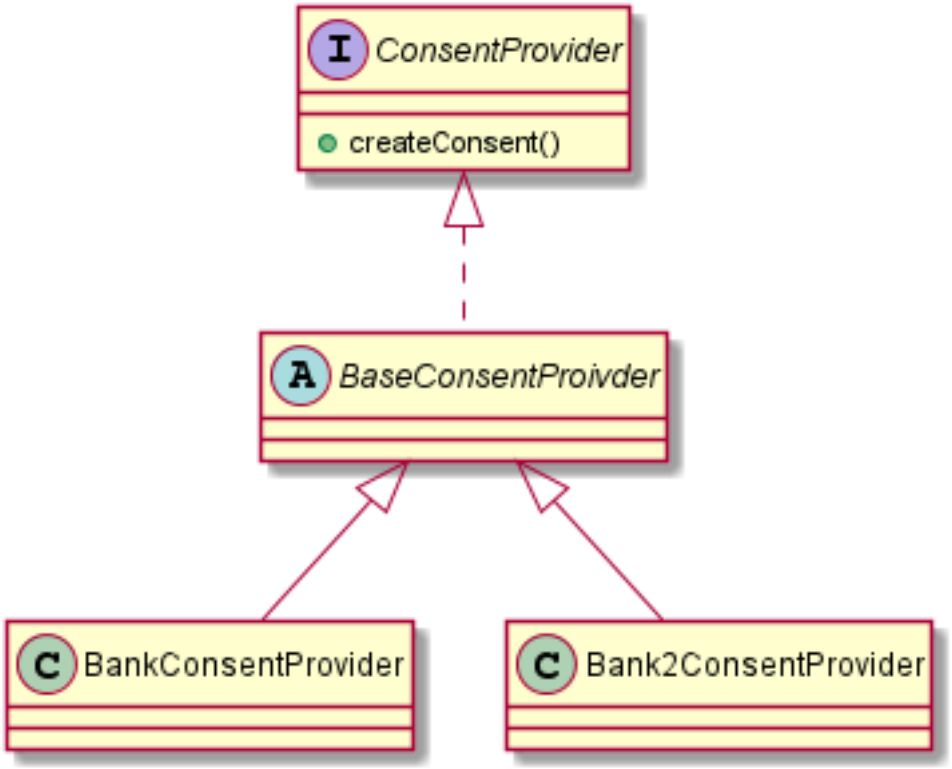

Figure 5.6 Consent Provider overview

5.1.4 Consent Provider

`ConsentProvider` is used for obtaining consent from the bank client. Obtaining consent is a part of the authorization flow for several banks the rest have implicit consent displayed by the ASPSP during normal login. For some banks it is obtained before authorization, for others after. Consent is used as another layer of confirmation that the bank client authorizes actions of the TPP.

5.1.5 AIS Provider

AIS API is explained in detail in the AIS unification chapter. This section will provide a simple overview.

`AISProvider` is used to obtain the following information from the bank.

- Basic account information, such as the iban, owner name or the currency.
- Current balances of the accounts.
- Transaction history of the accounts.

This is where most of the challenges with unification were present, as the bank differences were most pronounced in this API. For example, several banks include balances in their accounts response. So in order for the end result to be as unified as possible it was decided that for banks

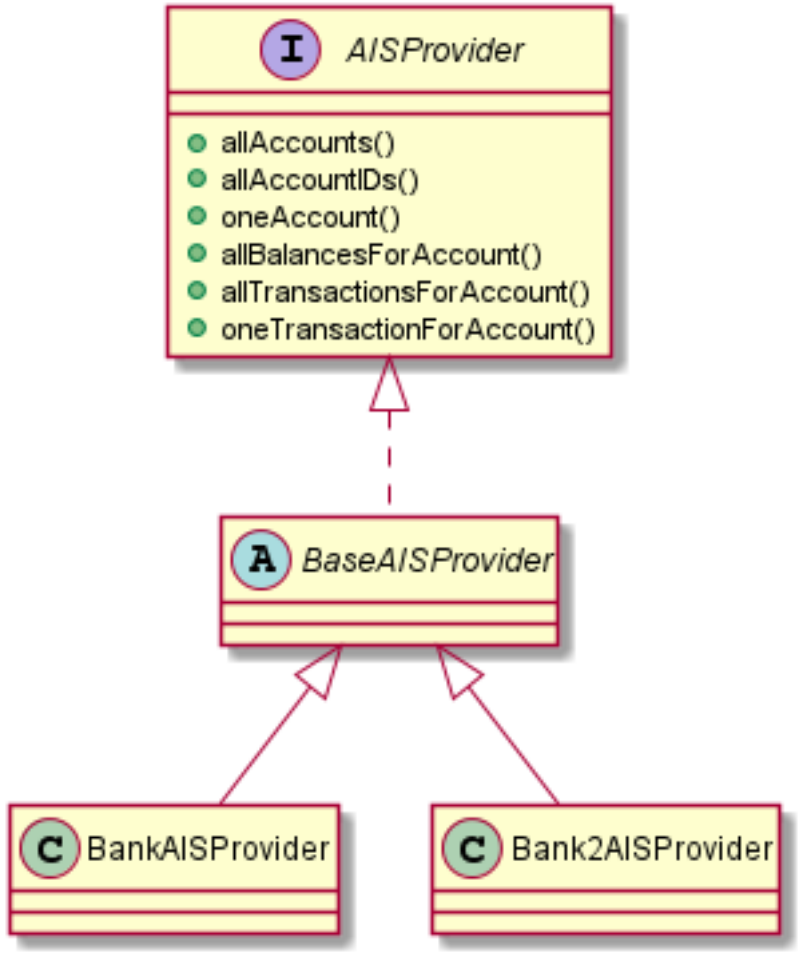

Figure 5.7 AIS Provider overview

without this a balance request will be made and added to the returned Account models. Other example is that not all banks support `oneAccount` api call. Therefore to mock this functionality an `allAccounts` request is sent and filtered for the searched for account.

From experience we know that the end users sometimes need to only know the list of account IDs, and do not need the full object. The banks however do not provide this endpoint, so an `allAccounts` request is made, and the end result is mapped into an array of IDs.

5.1.6 PIS Provider

PIS implementation is out of scope of this thesis. For now, it is assumed it will use the same basic architecture as `AISProvider`.

5.1.7 CIS Provider

CIS implementation is out of scope of this thesis. For now, it is assumed it will use the same basic architecture as `AISProvider`.

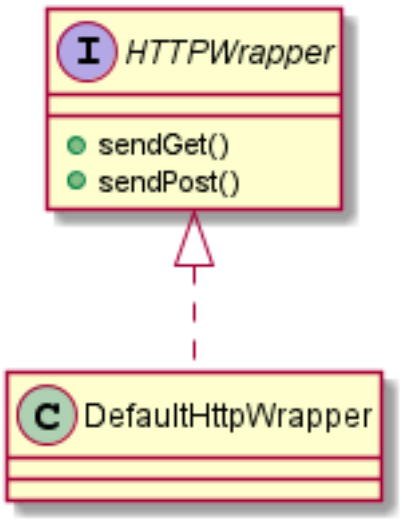

Figure 5.8 HTTP Wrapper Overview

5.2 HTTP Wrapper

`HTTPWrapper` is a custom HTTP solution for the Erste SDK project. A default implementation in the form of `DefaultHTTPWrapper` is provided using Apache HTTP implementation. However if the end user is not satisfied with the default implementation. Be it because their infrastrucutre requires special routing, headers, etc. for outgoing requests or because they do not wish to use the Apache library for any reason. It is possible to implement this interface with a custom version and provide it while building the SDK in `ErsteSDKBuilder`

The main purpose is to accept a configured `ErsteSDKHttpRequest`. Send a HTTP GET or POST request to the specified URL with provided headers, parameters and body. And then map the recieved response to a response class provided by the request either by using the default JSON Jackson `ObjectMapper`, or an optional mapper function contained in the request. This achieved via generics. All response classes are expected to implement the `ErsteSDKHttpResponse` interface so that original API response can be included for every request.

5.2.1 Erste SDK HTTP Request

`ErsteSDKHttpRequest` is a wrapper class for a HTTP request. It contains, the URL, headers, parameters and body. The headers implicity contain a Content type header set to value of `application/json` representing that the body will be sent in the JSON format. It also contains a generic `Class` that the recieved response should be mapped into. If the mapping requires special care and cannot be done by the Jackson `ObjectMapper` a custom mapping can be provided.

5.2.2 Erste SDK HTTP Response

`ErsteSDKHttpResponse` is an interface that all request response classes should implement. It is used to set the original response message for every request, so that the end user can use it if needed. For example transactions are probably the most differing object across the banks, and if the end user needs to access an original field that was transformed during mapping they have the ability to do so.

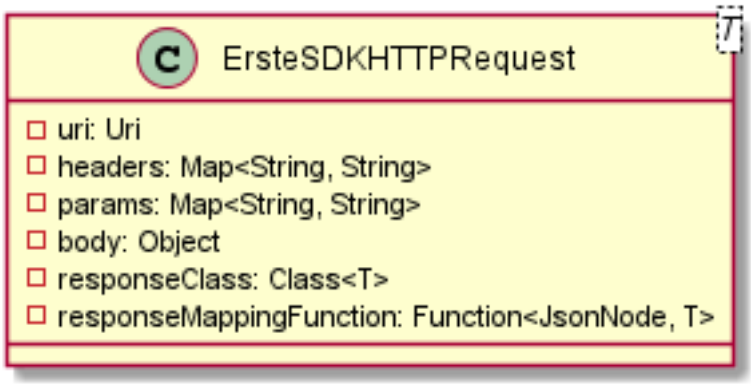

Figure 5.9 Erste SDK HTTP Request overview

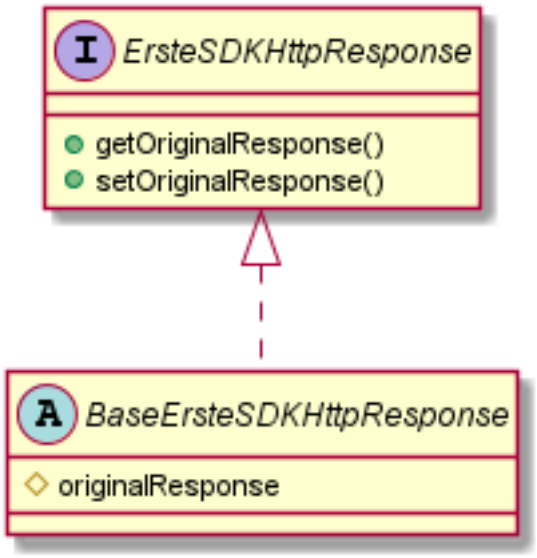

Figure 5.10 Erste SDK HTTP Response overview

An interface and a base class is provided. The response object can either extend the base class, or if it is not able to (because it already inehrits from another class) it can simply implement the interface.

5.3 Erste SDK Builder

`ErsteSDKBuilder` is responsible for instantiating the correct `ErsteSDK` from the passed parameters, which are then stored in a `ErsteSDKConfiguraiton` and accessible form the SDK. The following parameters are mandatory:

- `bankType` - enum representing the bank type of the SDK.
- `webApiKey` - application API key. Obtained from the developer portal.
- `clientId` - application client ID for the given bank. Obtained from the developer portal.

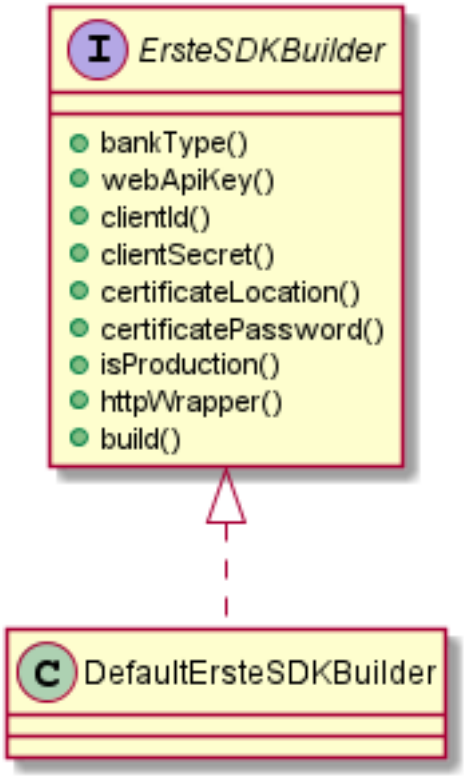

Figure 5.11 Erste SDK Builder overview

- `clientSecret` - application client secret for the given bank. Obtained from the developer portal.

  The following parameters are optional:

- `certificateLocation` and `certificatePassword` - QWAC certificate location certificate location and password for access. The developer portal provides certificates for the sandbox environment, but not all banks require sandbox signing. The certificate is expected to have the public and private key combined into a file in the `.jks` format. If one parameter is given the other is expected as well or an exception is thrown. If parameters are provided and a custom `HttpWrapper` is not provided a `DefaultHttpWrapper` is created with Apache `HttpClient` using certificate signing.
- `isProduction` - `true` if production, `false` if sandbox. Defaults to `true`.
- `httpWrapper` - a custom `HttpWrapper` to be used. Defaults to `DefaultHttpWrapper` with a default Apache `HttpClient` otherwise.

After passing the parameters the `build()` method is called by the end user. It validates that all required parameters are present and if certificate location was provided the password was provided as well. It then creates an `ErsteSDKConfiguration` from the parameters and

5.3.1 Erste SDK Configuration

`ErsteSDKConfiguration` is the main class containing SDK settings. It is created when building the SDK with the `ErsteSDKBuilder` and then used as an attribute of the built `ErsteSDK`. The

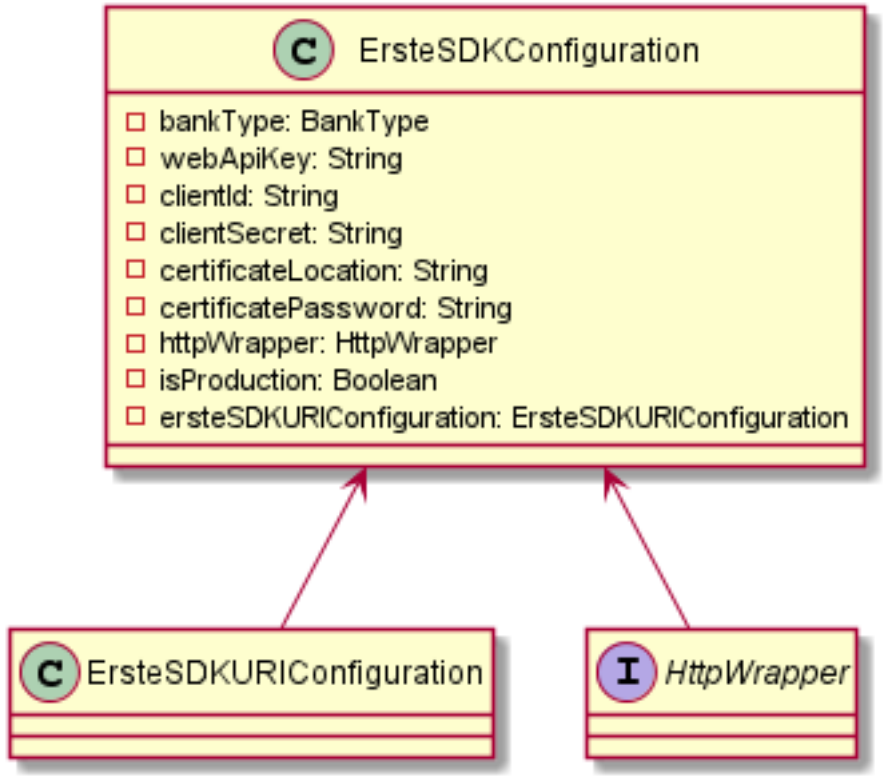

Figure 5.12 Erste SDK Configuration overview

configuration is then accessed by the API providers when creating the requests. Most configuration attributes are modifiable with the change being proliferated to the API providers. Certain attributes however do not affect the existing SDK.

- `bankType` - to change the bank of the SDK it has to be rebuilt as the bank specific logic is contained in the bank specific classes.
- `certificateLocation` and `certificatePassword` - since the HTTP client of the `httpWrapper` is already using the supplied ceritificate it is possible to only change the wrapper itself to one using the new certificate.
- `ersteSDKURIConfiguration` - base bank endpoints do not dynamically change between requests. If a bank modifies its endpoint it is likely to come with an API change that would need to be reflected in the SDK as well. This issue would then need to be resolved in an SDK update anyway without users needing to manually modify endpoints.

5.3.1.1 Erste SDK URI Configuration

`ErsteSDKURIConfiguration` is the class responsible for holding URI information for Erste endpoints. For each endpoint a BCR example value will be presented.

- `schema` represents the type of HTTP protocol used. For most banks it has the value of `https`.
- `baseURL` - the base URI for all API: `bcr.ro`.

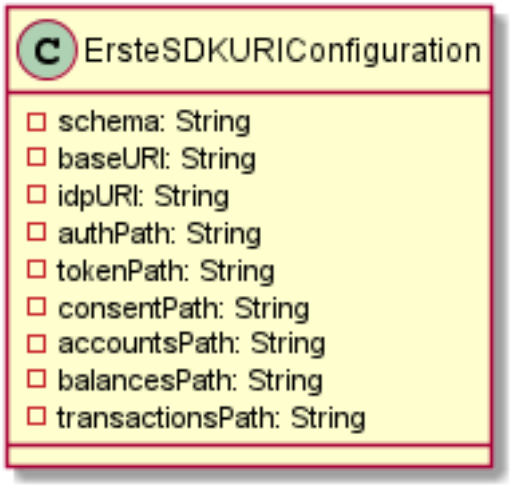

Figure 5.13 Erste SDK URI Configuration overview

- `idpURL` - some banks use a different base URI for their identity provider (endpoint used for OAuth). `login.bcr.ro`
- `authPath` - endpoint for the login page to redirect the client to.
- `tokenPath` - endpoint for OAuth code to access token exchange. Also used for OAuth refresh.
- `consentPath` - endpoint for consent creation and access.
- `accountsPath` - endpoint for retrieving account information. It is the base endpoint for all AIS API endpoints.
- `balancesPath` - endpoint for retrieving account balances.
- `transactionsPath` - endpoint for retrieving account transactions.

AIS endpoints are accessed in the format of `${accountsPath}/${accountId}${balancesPath}`.

5.4 Exceptions

The base exception class of the SDK is `ErsteSDKException`. It is a runtime exception, therefore it is not necessary to surround every SDK call with a `try...catch` block. Every other exception extends from it, so it is possible to only catch the main exception and simplify exception handling. It contains an `errorMessage` attribute containing the custom error message provided by the SDK. There are three other main specific exceptions:

- `ErsteSDKOperationNotSupportedException` - thrown when the end user tries to call a method that the SDK for that specific bank does not support. For example calling the athorization method in a bank that requires additional consent and PKCE without providing them.
- `ErsteSDKBuilderException` - thrown on a failed build of the SDK by the `ErsteSDKBuilder`. Examples include not providing mandatory attributes or passing a corrupted certificate.

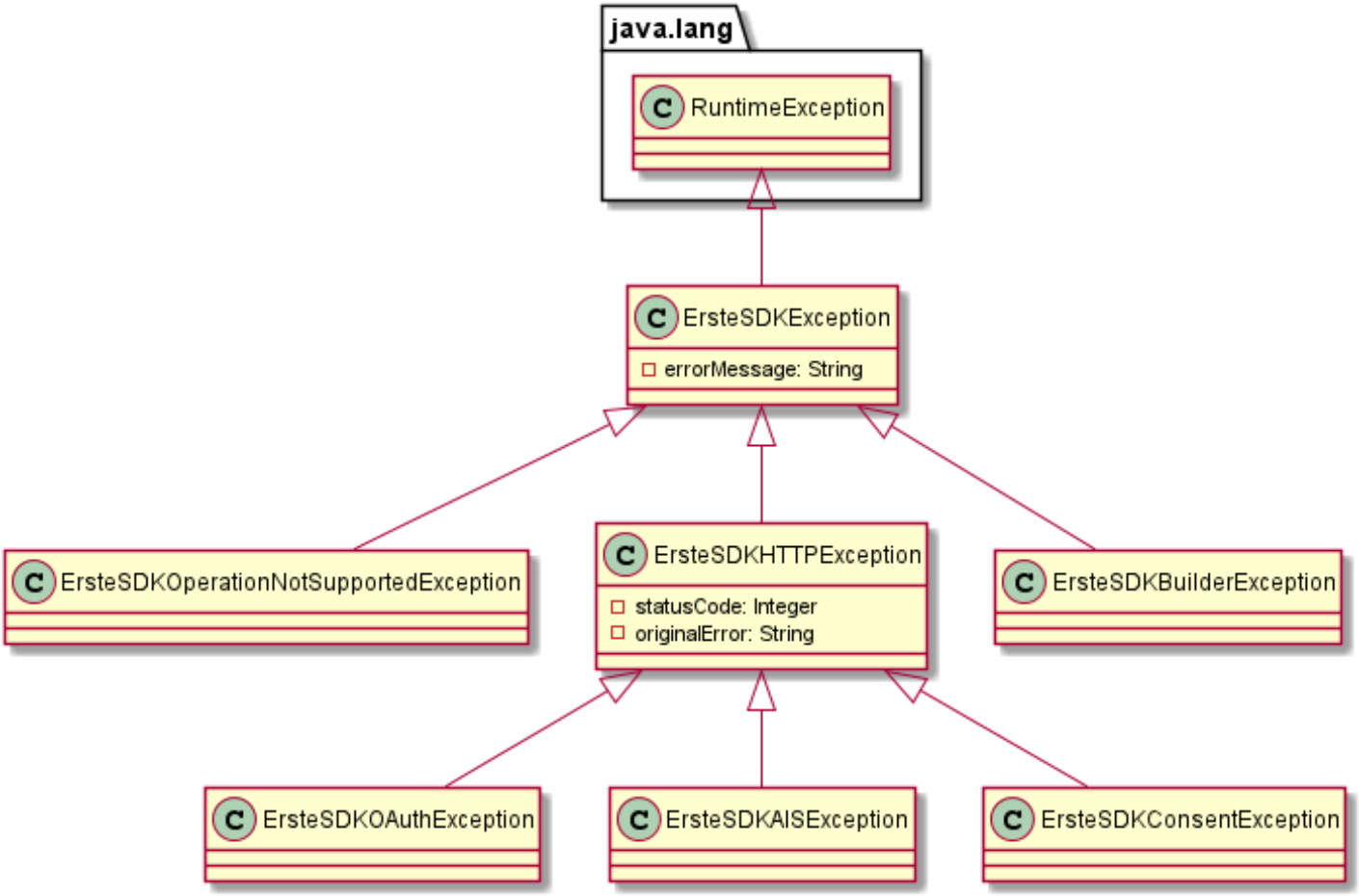

Figure 5.14 Excpetion architecture overview

- `ErsteSDKHTTPException` - a wrapper class for any failed HTTP call. It Contains two attributes. `statusCode` integer representing the status code of the failed HTTP request. `originalError` is the unmodified error response from the API. The end user should never receive a raw HTTP Exception. Instead it should be mapped to a specific API exception as shown below.
  - `ErsteSDKOAuthException` - thrown if an OAuth flow API call fails.
  - `ErsteSDKAISException` - thrown on failed AIS call.
  - `ErsteSDKConsentException` - thrown on failed Consent API call.

5.5 Base API Provider

The `BaseAPIProvider` class is another layer of abstraction and simplification for the `BaseImplementation` classes providing API logic. It is a generic class which expects a type extending the `ErsteSDKHttpException` and serves as the specific API exception for the `BaseImplementation`. It also serves as a convenient holder for the configuration and the URI Configuration used by every API. Its main purpose however is exception mapping. It contains the `exceptionMessagesByStatusCode` map containing custom error messages for every returned error status code. The `fillInExceptionMessagesMap`

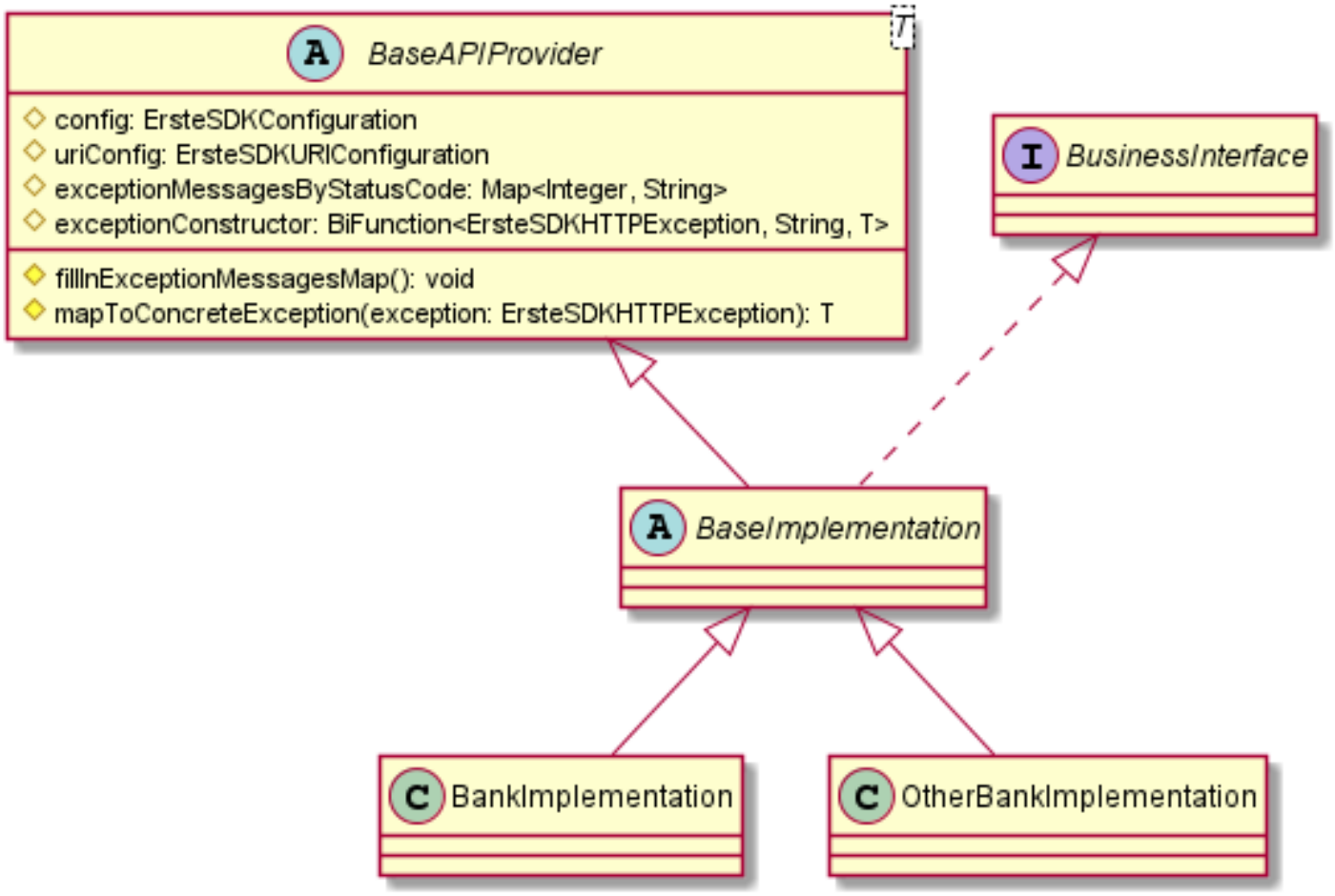

Figure 5.15 Base API Provider

method maps the most common errors to their messages, but it is possible to override this method for banks returning unique or different error codes.

When an HTTP Exception is caught within an API Provider after making an unsucessful API call it should call the `mapToConcreteException` method. This method gets the appropriate custom error message from the map and instantiates the correct exception using the `exceptionConstructor` bifunction passed in the mandatory constructor of the `BaseAPIProvider` class. The return value is an API exception (e.g., `ErsteSDKAISException`) with the returned status code and original error of the failed call as well as a custom error message provided by the SDK.

Chapter 6

Authorization Flow unification

This chapter will focus on authorization flows of different banks. In order to access the PSD2 API an OAuth access token must be obtained. Additionally some banks also require the client to specifically grant consent to the application to access specific data with specific frequency. For example a client may only choose to grant consent to only check the balances of their account in euros once per day, but keep their domestic Czech crown account private.

This chapter will use some OAuth specific terminology and will focus on unification of OAuth flows. Therefore a section devoted to OAuth basics follows next, but can be safely skipped if one is familiar with OAuth.

6.1 OAuth 2.0

OAuth standard as published in RFC 6749 defines four roles:[32]

- Resource owner (Erste Bank Client)
- Resource server (the API - Erste ASPSP)
- Authorization server (can be the same server as the API - Erste IDP)
- Client (the third-party app - TPP)

Notice the different meaning of the word client in the OAuth flow and in a bank business sense.

The OAuth 2.0 framework enables clients to access resource owner's resources (both data and API endpoints) on the resource server without resource owner's direct involvement. Instead the owner authorizes the client only once using the authorization server and the client receives a time limited access token which enables them to access those resources.[22]

The OAuth framework defines two ways of obtaining the access token - the Implicit grant and the Code grant.

6.1.1 Authorization Code Grant

The client redirects the resource owner to an authorization server where the server authenticates the owner and obtains authorization. The authorization server redirects the owner back to the

client with the authorization code, which the client then exchanges for the access and optionally refresh tokens.

While the refresh token is still valid the client may present it to the authorization server to obtain a new access token, validity of which is usually much shorter - In case of CSAS the access token has a validity of 5 minutes, whereas the refresh token can be valid for up to 90 days.

All Erste banks support Authorization Code Grant, however EBH does not support refresh tokens.

6.1.1.1 The Proof Key for Code Exchange (PKCE)

The Proof Key for Code Exchange (PKCE, pronounced *pixie*) is an additional security measure for OAuth Code Grant flow. The client creates a secret and encrypts it using a method known to the server (e.g., SHA-256). The plain secret is called a *code verifier* and the encrypted secret is called *code challenge*. [31]

The application sends the code challenge along with the first login request. The server returns the authorization code as usual. However, when the application exchanges the code for an access token it also needs to send the code verifier. The server then encrypts the code verifier with the same agreed upon encryption method and checks whether it is equal to the code challenge the application sent in the first request. Access (and refresh) tokens are only returned if the codes match. [1]

6.1.2 Authorization Implicit Grant

The Authorization Implicit Grant is the simpler, but arguably less secure of the two. Instead of issuing the client an authorization code, the client is issued with an access token directly. This method is optimized for clients implemented in a browser, since it reduces the number of round trips required to obtain an access token.

6.2 Cross bank Authorization flow analysis

Unlike other APIs the Authorization flow is fairly unified from the beginning. The main difference between the banks is the presence or lack thereof of a Consent ID creation step either as an authorization pre-step or a post-step. Other differences are the requirement of PKCE and certificate request signing, or both.

6.2.1 Consent Id

The Consent mechanic was explained in a chapter focusing on PSD2. In short all API actions must have backing of a customer consent. Some banks implicitly create consent when logging in by showing a warning that the user consents by logging in, while other banks require manual consent creation. For authorization it is only relevant for certain banks as a sort of a secondary access token. Aside from authorizing the TPP to access their Erste bank accounts as a whole the Client has a choice of only authorizing specific bank accounts. The issued Access Token is then only valid when presented with the Consent ID it was issued with.

The pre-step and post-step differ only in whether a Consent ID is needed to obtain the Access Token in case of pre-step, or the Access Token is needed to obtain the Consent ID in case of post-step.

Only three banks support Consents. EBC as pre-step, EBOE and SLSP as post-step.

6.2.2 PKCE

PKCE was explained in detail in section pertaining to OAuth. PKCE is only used by EBC and EBH Retail banks. In addition the application can choose the encryption method via a `code_challenge_method` parameter.

6.2.3 Request Signing

The Certificate technology used is Qualified website authentication certificate (QWAC). All banks support certificate request signing, but only EBC requires it. The certificate is obtained from the developer portal.

6.3 Cross bank Authorization flow solution architecture

The following diagrams illustrate the complete authorization flow from the end user making a login request in a TPP application to the backend server obtaining the access token in case of the Authorization Code Grant, or the client application in case of Authorization Implicit Grant. In both flows an optional Consent routine is present illustrating when it should occur as a pre-step or post-step. The flow of the consent routine is then illustrated in detail in the third diagram.

6.3.1 Authorization Code Grant

The user tries to log in on the TPP application, which sends the login request to the backend server. The server initializes the SDK and if the user is from the EBC bank the server initiates the consent routine. The SDK then generates the redirect URL for the client which is then redirected to the bank's IDP server and logs in. The returned authorization code is exchanged for the access token and if necessary the server initiates the consent routine.

6.3.2 Authorization Implicit Grant

The user tries to log in on the TPP application. The application initializes the SDK and goes through the consent routine if necessary. The SDK then generates the redirect URL and the client is redirected to the bank's IDP server and logs in. The application then receives the access token and initiates the consent routine if necessary.

6.3.3 Consent Routine

The initiator (TPP application or server) gets account identification and consent settings (e.g., number requests per day) from the user and passes it to the SDK. The SDK creates a request according to the settings and sends it to the ASPSP (Erste bank), and then parses and returns the response.

6.4 Cross bank Authorization solution architecture

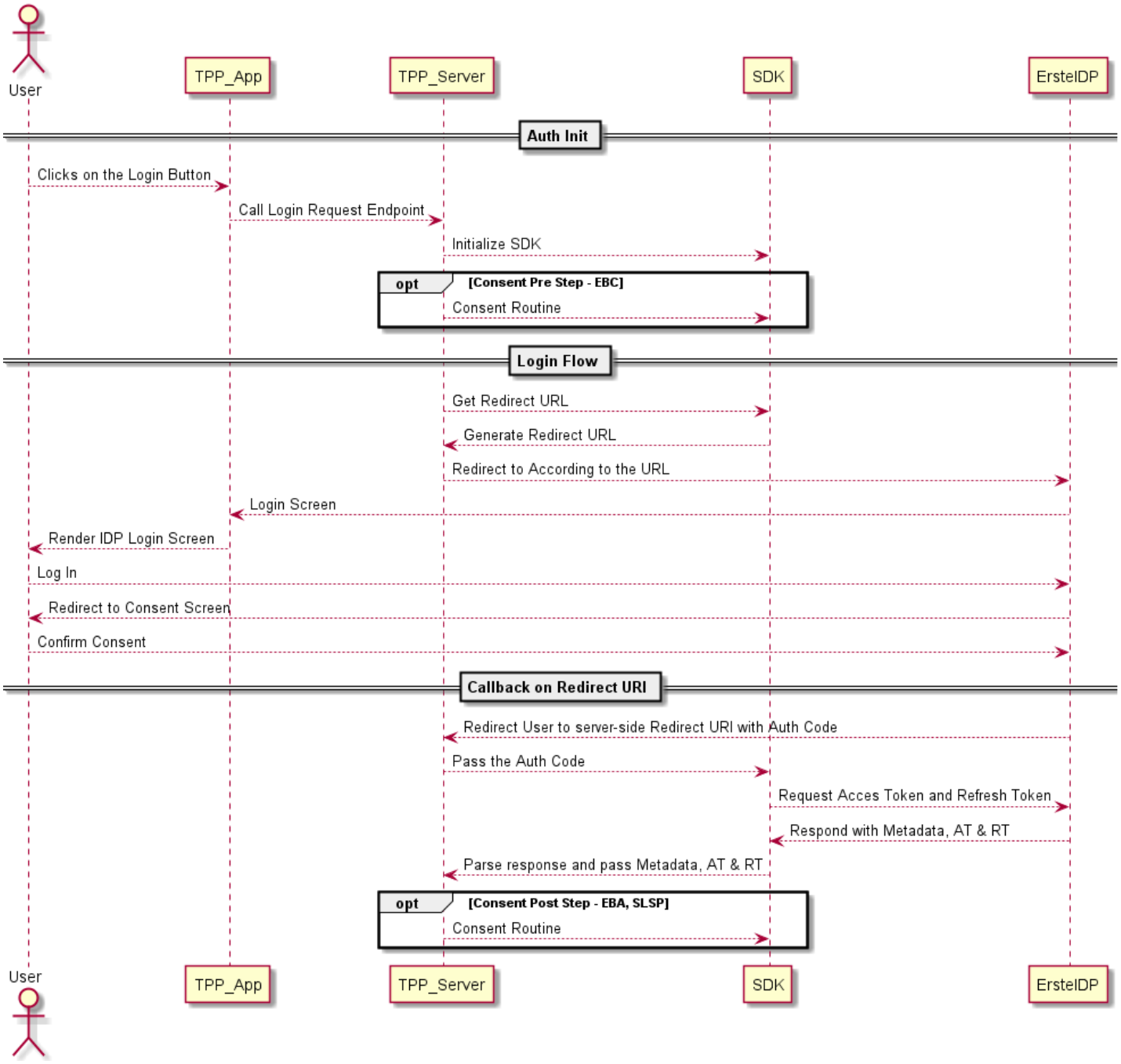

Figure 6.1 OAuth Authorization Code Grant Flow

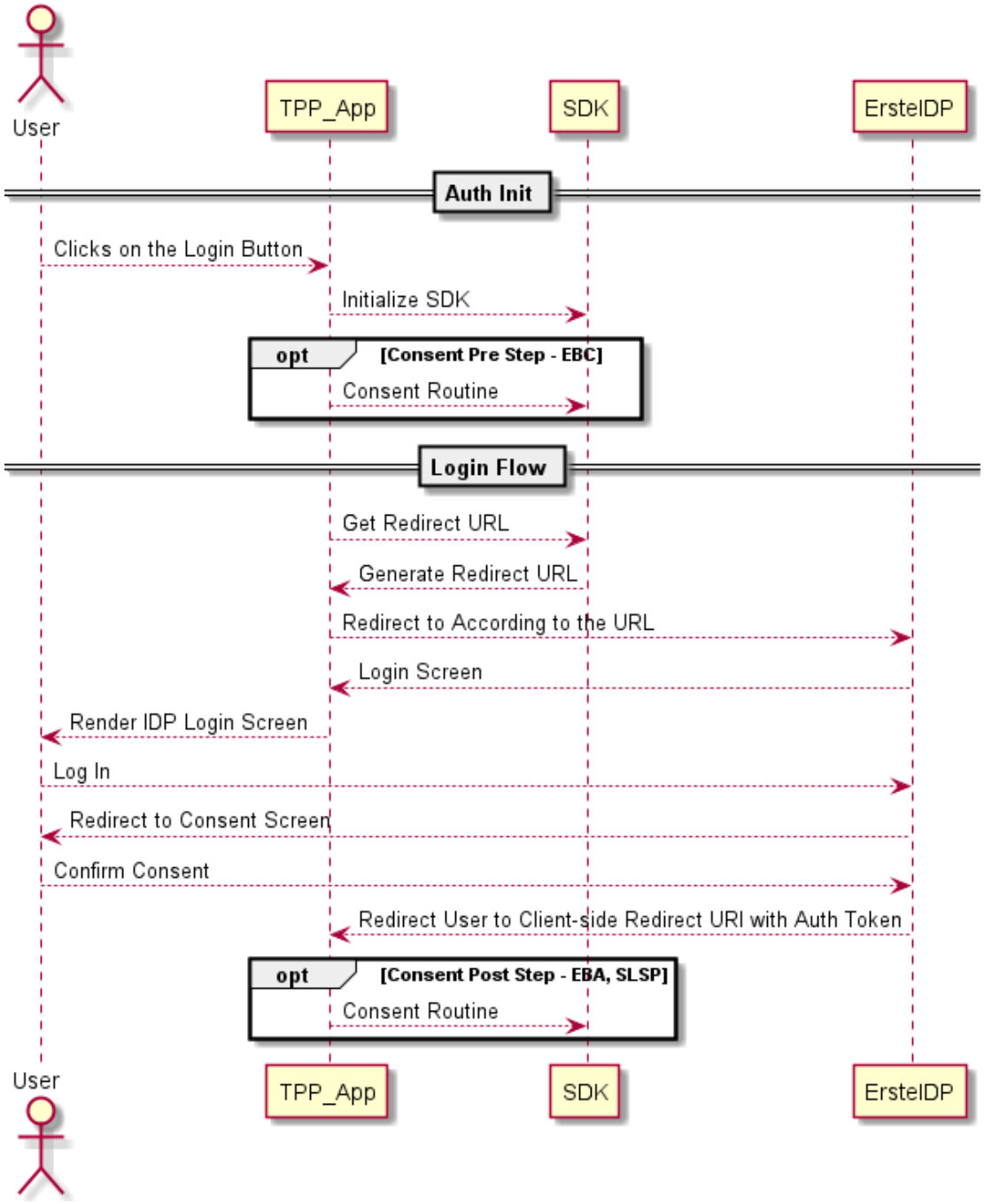

Figure 6.2 OAuth Authorization Implicit Grant Flow

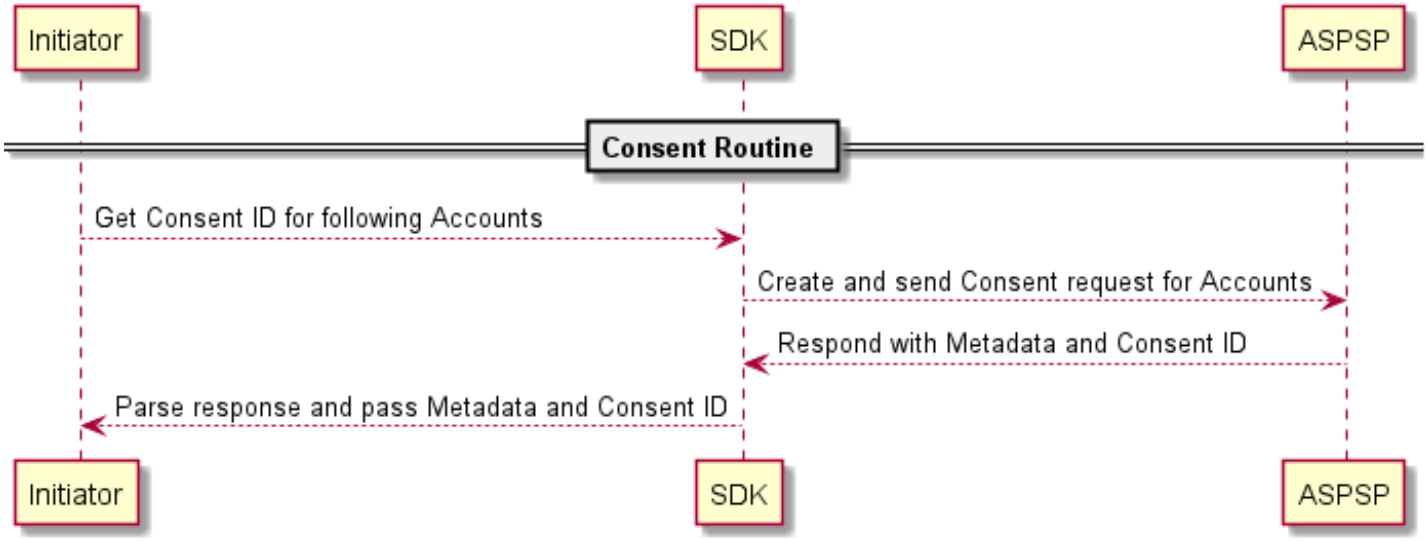

Figure 6.3 Consent Routine

6.4.1

The final implementation of interfaces was designed around a main `AuthorizationGrant` interface, which the interfaces for Authorization Code Grant and Implicit Grant extend from. The many overloaded methods correspond to different requirements of banks. For Example EBC requires both PKCE and Consent arguments, whereas EBH only requires PKCE arguments.

6.4.2 Tokens

The main Authorization object returned by the Authorization flow is the `OAuthToken` representing an access or refresh token depending on the `type` attribute. The `token` string is the actual token string to be used in requests, the `expiresIn` integer represents the number of seconds the token is valid for, and finally the `scope` is the API scope the token is valid for (AIS, PIS or CIS).

Authorization Code Grant flow returns an `OAuthTokens` wrapper containing access and refresh tokens from the code exchange.

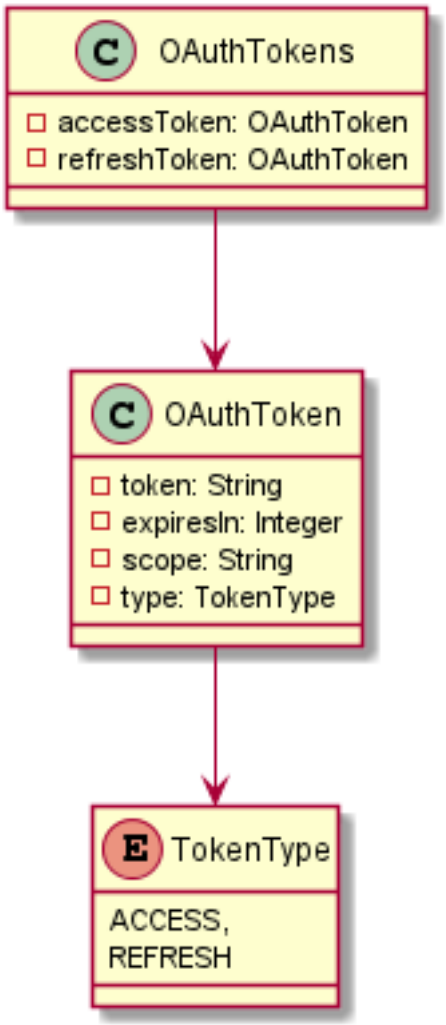

Figure 6.4 Tokens

Chapter 7

Account Information Service unification

This chapter will cover the unification efforts for the PSD2 AIS API. It will first cover the available Erste AIS API endpoints and how they were unified. Then it will present a description of unified and bank specific models and then present how the models were unified.

7.1 Erste AIS APIs

There are 5 main methods for accessing the AIS API provided by Erste:

- All accounts - returns a basic overview of all available bank accounts. Available to all banks.
- One account - selects a specific account Not available to all banks, the functionality is mocked by calling all accounts and filtering.
- All balances - returns a balance list for the specified account. Available to all banks.
- All transactions - returns a list of pending and booked transactions for the account. Available to all banks.
- One transaction - select a specific transaction for account. Not available to all banks, the functionality is mocked by calling all transactions and filtering.

A new method was created for the SDK based on experience with fintech apps: `allAccountIDs`, which makes an all account request and transforms the response into a list of strings of account IDs.

7.2 Unified and Specific Model Architecture

The general principle behind the unification architecture is that all banks represent the same business object, albeit in different ways. The concept will be explained on the account model, but it applies to balances and transactions in the same way. This abstract business object is represented by the `BaseAccount` interface which has a `toAccount` method, which converts the concrete bank specific representation to an end user object `Account` (called `EndUserAccountModel` in the UML for clarity). This end user object also implements the `ErsteSDKHttpResponse` interface explained in previous chapters, which allows the end user to access the original bank response in the form of a string if needed.

The end user object was created as a sort of an intersection of the bank specific models. To have a couple of concrete examples:

- All models of an account have an `id`, some banks call this attribute `resourceId`, others simply `id` but they represent the identification of an account, and therefore the end user object has an `id` attribute.
- All balances have a represenation of an Amount. Half of the banks represent the amount (e.g., 3.14) as a string ("3.14"), others as two integers - one for full value (314), the other for scale (2 - amount of numbers after the decimal point). For the end user model the second approach was chosen and the banks using the string representation converted their amount to two integers in the `toBalance` method of the `BaseBalance` interface.
- CSAS transaction model has a vast amount of extra information compared to other banks. If all the bank specific attributes from all the banks were included in the end user object it would massively grow in size. Those attributes would also always be `null` unless the endpoint was called with the SDK of that bank. Therefore if users require a bank specific attribute they need to call the `getOriginalResponse` and extract it from the original JSON string.

This is the basic principle under which the end user objects were unified. Of course there are exceptions to this rule - if there is an attribute that only a couple of banks do not have, but the rest do, it will be included in the end user object anyway even if it will be `null` for those banks.

The abstract model class (`BaseAccountModel` in the example) was used for unifying all models of BCR, EBC, EBOE and SLSP. Those banks were the most unified out of the box, being inspired by the NextGenPSD2. CSAS (inspired by ČOBS) and EBH use their own models for most models.

7.2.1 Account

Even as the most extensive of the unified models, most of the attributes shold be self explanatory. This model was fairly unified across the banks.

7.2.2 Balance

There were two main challenges when unifying balances. Represenation of amount and timezones.

For amount the two options were repsenting the amount as a string or two integers - one for the full value as an integer without the decimal point, and the other representing amount of numbers after the decimal point. In Java there was also a possibility to use the `BigDecimal` class, which internally stores the number as two integers, but provides methods that allow treating it as one number, while avoiding the dangers of arithmetic errors that come with `float` or `double`. However Node.js does not provide such a class natively, and to keep the dependency list light and two SDKs unified the two integer apporach was chosen. This approach also has the benefit of no ambiguousness of what character is used for the decimal point as in the string. While at the same time still allowing for relatively easy calculations unlike the string.

The timezone issue is related to the `lastChangeDateTime` and `referenceDate` attributes. For some banks, the date time is in the local timezone, while others use UTC. It was decided that all date times will be converted to UTC, but the dates will remain in the local time zone, even if it will sometime cause conflicts. For example local datetime is 1 hour after midnight in the UTC + 2 timezone. When converted to utc, the datetime is 11 PM of the previous day,

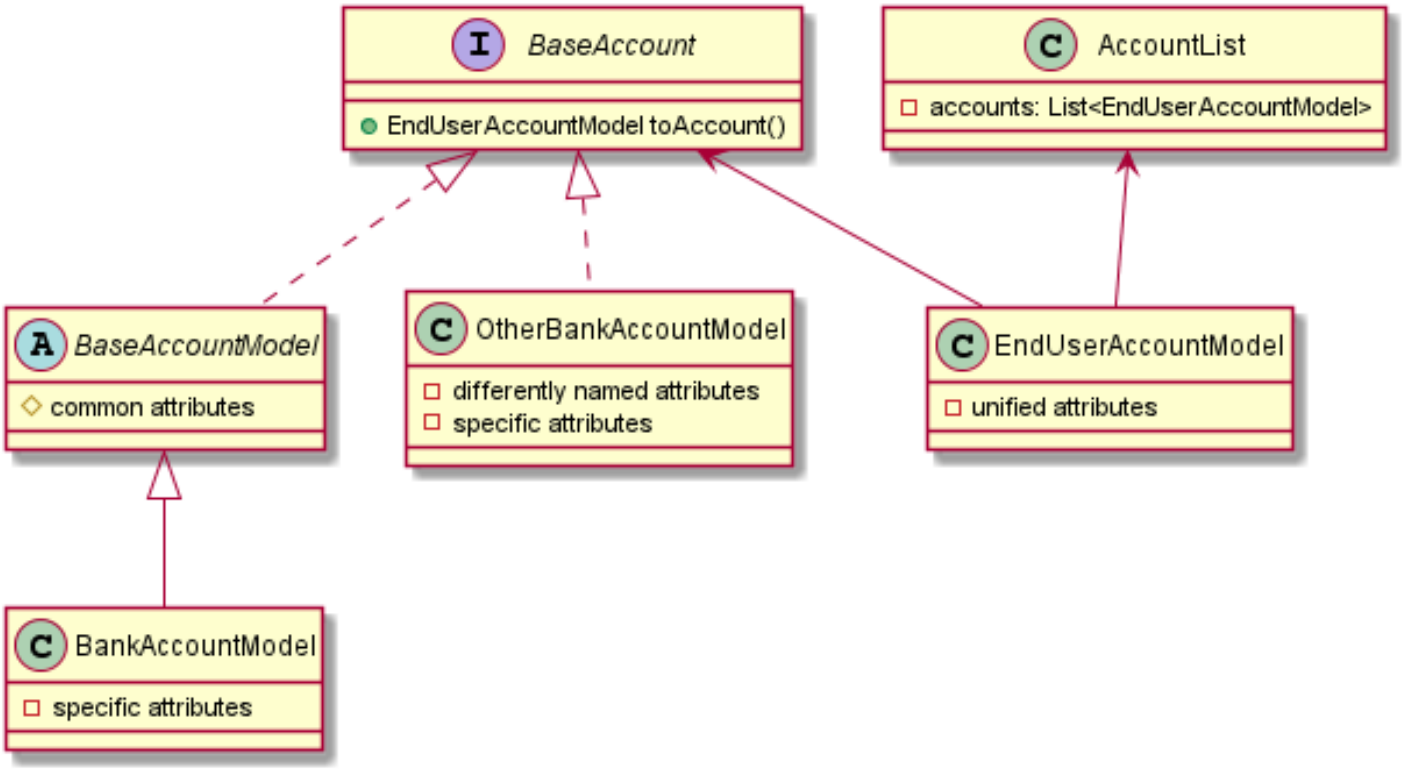

Figure 7.1 Unified and Specific Model Architecture

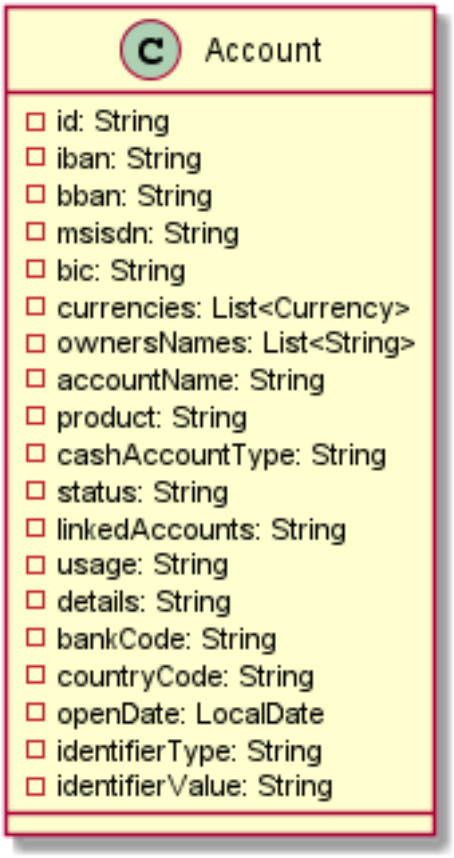

Figure 7.2 Unified Account Model

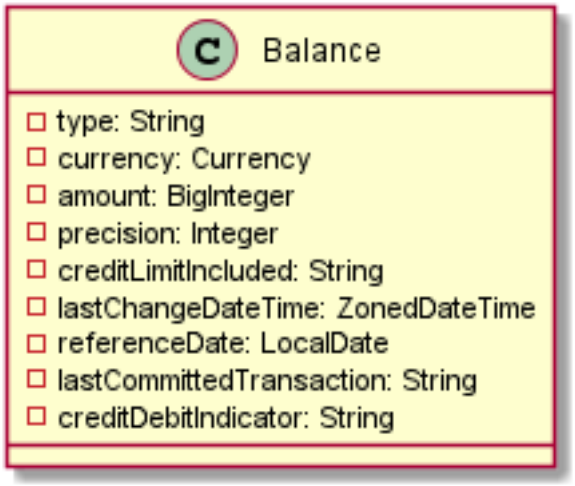

Figure 7.3 Unified Balance Model

but the date attribute is still the original day. This date issue was left unadressed, because in the Transaction API there is a possibility to filter the transactions according to dates, and a situation could arise where the desired transaction would not be included in the filter because of the date change. Timezone however were still converted to UTC for unification and allowing easier operations without checking for timezone everytime.

7.2.3 Transaction

Transaction was the hardest model to unify as most banks (even among the fairly unified NextGenPSD2 group) had differing models. In the end a different approach was chosen inspired by the Erste George online banking application. In the transaction overview of the application only the most crucial information is displayed, and the model is fairly small. It is believed that for most use cases this model is enough, and for special use cases the original response is always available with the `getOriginalResponse` method.

Unlike other models the `TransactionList` is a part of the end user model inspired by the NextGenPSD2 transactions response which also returns an object containing arrays of pending and booked transactions.

7.3 AIS Mapper

The main class responsible for mapping the raw JSON response to the end user objects with the help of generics. It does so by first mapping the raw JSON to a bank specific model (passed as a generic type in the constructor), which can be trivially accomplished using the Jackson `ObjectMapper`. The bank specific models are expected to extend the base model interface (e.g., `BaseAccount`) therefore they have the method, which returns an end user object with the data from this bank specific object (`toAccount`) with filled in `originalResponse` attribute containing only data relevant to that specific object. This is accomplished in the protected `mapSingle...` methods.

The public methods from the mapper interface operate with the raw json response. Their responsibility is to extract parts of JSON representing the business object and pass them to the

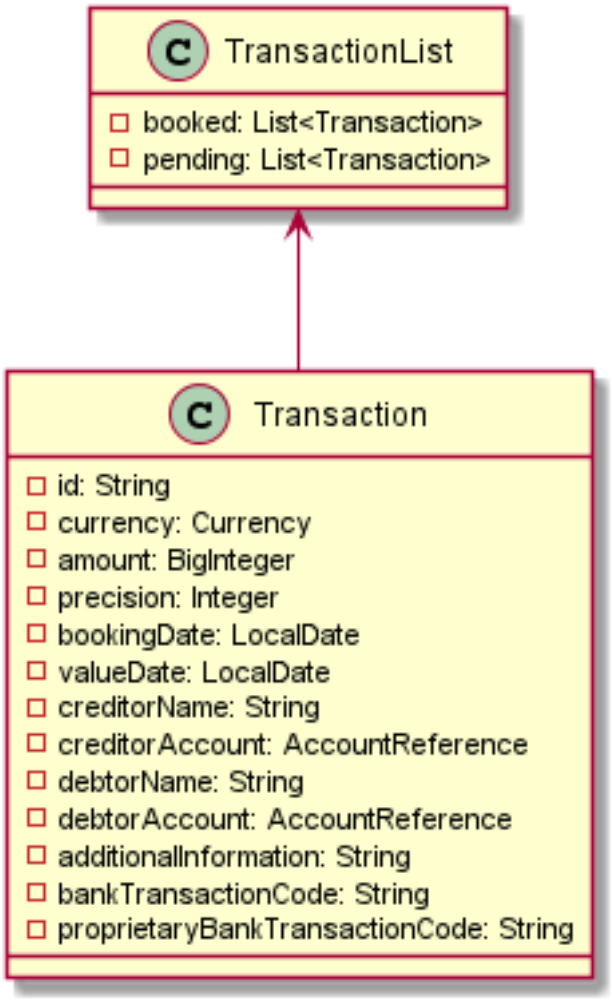

Figure 7.4 Unified Transaction Model

`mapSingle...` methods which return the desired end user object.

The Mapper returns a custom List wrapper object (`AccountList`). While transactions are returned by the AIS Provider in that same wrapper object, for accounts and balances the provider returns the actual list of objects (`List<Account>`). This wrapper object is used to store the original response of the whole request in case it will be needed in the wrapper. If not the provider simply extracts the list and returns it.

7.4 Sandbox limitations

Unfortunately some unification efforts were hindered by the issues in the sandbox environment on the Erste developer portal. To list the two most important ones:

- The sandbox APIs ignored all query parameters for the `/transactions` endpoint, which meant that pagination of the results or filtering of results were not implemented properly. Additionally EBOE transactions could not be tested as they had a required query parameter.
- EBC not returning any accounts in the `/accounts` endpoint which means that the whole API is untestable, as other endpoints depend on accessing a single account.

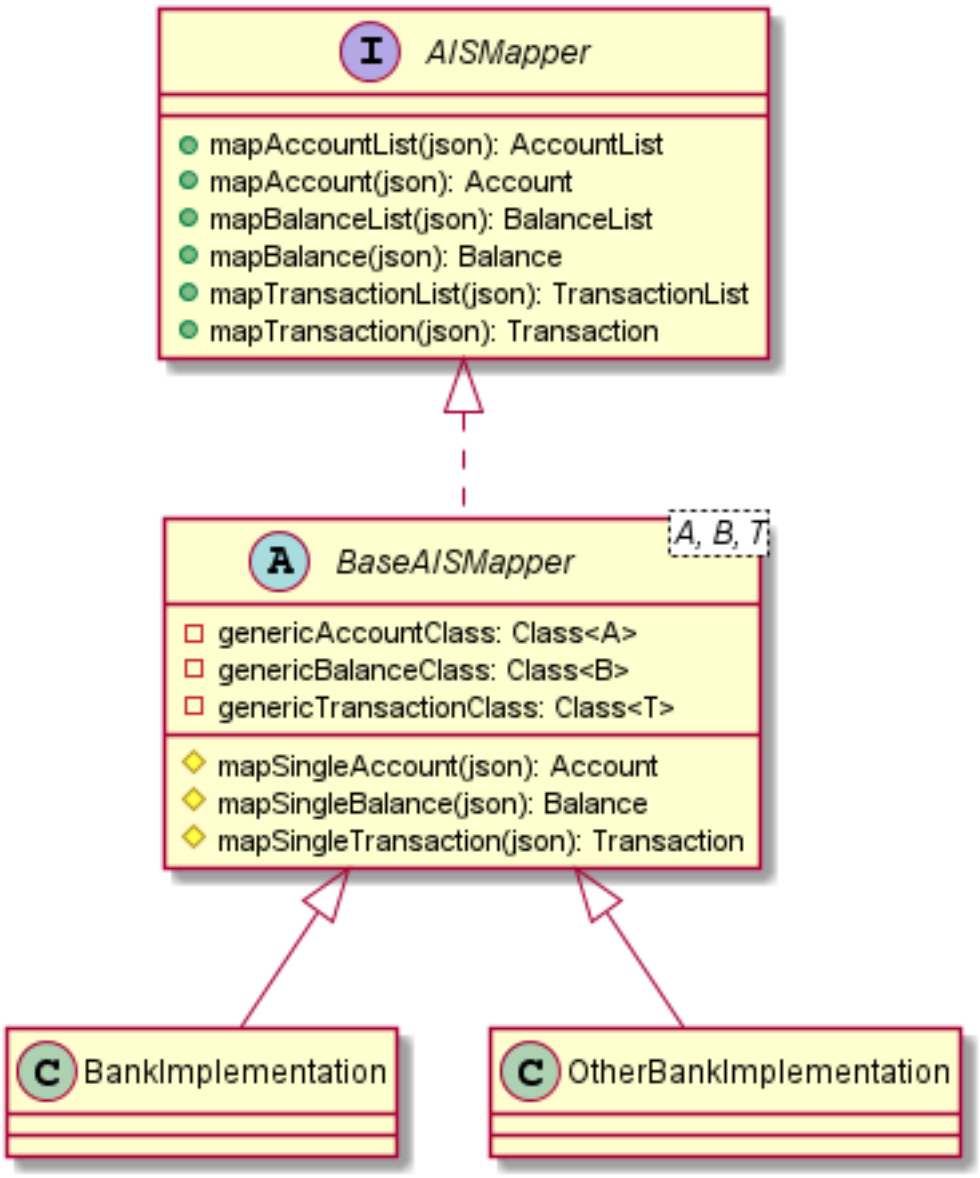

Figure 7.5 Mapper Architecture

Chapter 8
Examples

This chapter will first show and descripe a couple of SDK examples from instantiating the SDK to calling AIS API with a received access token. The following section will then explain how to setup the included example project for testing.

8.1 Example Usage

8.1.1 Instantiating the SDK

The following code sample provides a configured instance of a CSAS SDK.

```
ErsteSDK sdk = new DefaultErsteSDKBuilder()
        .bankType(BankType.CSAS)
        .webApiKey("someWebApiKey")
        .clientId("someClientId")
        .clientSecret("someClientSecret")
        .certificateLocation("c:\example\certificate.jks")
        .certificatePassword("123456")
        .isProduction(false)
        .build();
```

8.1.2 Obtaining an Access Token using Authorization Code Grant

This sample generates a redirect URI using PKCE with `S256` encryption method. The user will be asked to consent to use of AIS API access as per the passed `AuthorizationScopeType`

```
AuthorizationCodeGrant grant = sdk.getAuthorizationCodeGrant();
URI redirectUri = grant.generateAuthorizationUri("callbackURL",
        "someState",
        Collections.singleton(AuthorizationScopeType.AIS),
        "S256",
        "codeChallenge");
// redirect user to the URI
```

After the user logs in, ErsteIDP will call the passed `callbackURL` with the authorization code and the passed state for identifying the user. The code can then be exchanged for a token using the SDK.

```
AuthorizationCodeGrant grant = sdk.getAuthorizationCodeGrant();
OAuthTokens tokens = grant.authorize("returnedCode",
        Constants.REDIRECT_URL,
        "codeVerifier");
```

8.1.3 Accessing the AIS API

This part will show three AIS calls.

The first is an all accounts call.

```
AISProvider aisp = sdk.getAISProvider();
List<Account> accounts = aisp.allAccounts("accessToken",
        "someRequestId");
```

An all balances call.

```
AISProvider aisp = sdk.getAISProvider();
List<Balance> balances = aisp.allBalancesForAccount("accessToken",
        "someRequestId",
        "accountId");
```

Finally a one transaction call.

```
AISProvider aisp = sdk.getAISProvider();
Transaction transaction = aisp.oneTransactionForAccount("accessTok",
        "someRequestId",
        "accountId",
        "transactionId");
```

8.2 Example Project

A simple example project in the form of a Spring application is included as an attachment to this thesis. However since the project is not yet officially released on maven, the only way to currently get it working is to install the SDK locally from the sources in the attachment using `mvn clean install`. This installs it in the local `.m2` maven repository and the build of the example project will not fail then.

After that an account must be created on the Erste Developer Portal on `https://developers.erstegroup.com/` . After creating an account with a personal organization, you can create an application and add banks to it. It is recommended to have only one application with all the banks for testing, as it allows you to use a single certificate for all of them. When connecting banks make sure to connect the PSD2 AIS API, with EBH take special care to connect the retail API, not the corporate one. Also remember to configure the OAuth callback URI with the one configured in the example application `http://localhost:3000/api/v1/auth/callback` .

After that the `credentials.properties` file must be filled in with the credentials obtained in the developer portal. Then the `.key` and `.pem` certificate files must be combined into a Java KeyStore `.jks` file. There are guides how to accomplish that online, but the general process is to convert the `.key` file into a `.pem` file, which you can do simply by renaming the file. And then finally running these commands in the terminal

```
openssl pkcs12 -export -in [path to certificate] \
-inkey [path to private key] -certfile [path to certificate ] \
-out testkeystore.p12

keytool -importkeystore -srckeystore testkeystore.p12 \
-srcstoretype pkcs12 -destkeystore wso2carbon.jks \
-deststoretype JKS
```

Chapter 9
Conclusion

9.1 Impact of the project

To summarize the main benefit of the project is abstraction of the implementational differences between the PSD2 API of the Erste Group. While all being under the same banking group, the differences between the banks were sometimes immense, and they give some context to the benefits of PSD2 unification efforts such as the Berlin Group's NextGenPSD2. What previously required hours of studying the developer portal, and building models and flows for every new bank, is now one SDK integration away.

The solution provided by the project should save TPP's time and money and hopefully enable easier access to leveraging the many benefits of open banking that PSD2 promised to bring to the fintech space. The development on the project will continue and PIS will be the next unified API. While PIS is arguably the most important API of PSD2, the project in its current state already brings tangible value in providing a solution for authorization, consent creation and accessing account information necessary for future payment creation.

9.2 Impact of the thesis

The main goals of this thesis were to introduce the PSD2 regulation for the purposes of the Erste SDK project, and the project itself with focus on Authorization and Account Information Service unification architecture. Those topics were covered in their own respective chapters. Additionally, a chapter covering Erste to establish more context, a chapter dedicated to general overview and internal technical workings of the project, and finally a chapter covering example usage.

The thesis should provide enough context for both the business and technical perspectives. Ideally, after reading it a software engineer should be able to not only use the SDK project in their own financial applications (after consulting the project documentation as well), but also have a better understanding of PSD2 and its requirements and possibilities.

9.3 Final words

My personal experiences with the project itself were very positive. This was my first time working on a project of this scale from the beginning to the end, and I am looking forward to continuing

the work on polishing up AIS and moving on to the most complex part of the PSD2 API - PIS. From the first analysis and proof of concept, to the final testing and improvements this was a challenge for me and a space to use and refine what I learned in school in practice.

On a less cheery note, due to personal reasons this thesis is not as polished as it deserved to be, as a year and a half in pandemic lockdown and burnout has taken a big toll on my mental well being. As well as picking up the thesis in the second half of the final semester, as the thesis I was working on the previous semester and a half slowly fell apart due to issues in scope and technical difficulties. That's why I would like to once again express my sincere gratitude to Ing. Nikolay Barbariyskiy for suggesting this topic and helping immensly given the short time schedule and my limited capabilities. But most importantly to Andrea Švancarová for always being there and helping me carry on when I was so often and so close to quitting. Thank you both.

Bibliography

[1] auth0. Authorization code flow with proof key for code exchange (pkce). Accessed: 2021-05-18. URL: https://auth0.com/docs/flows/authorization-code-flow-with-proof-key-for-code-exchange-pkce.

[2] European Central Bank. Single euro payments area (sepa), 10 2020. Accessed: 2021-06-25. URL: https://www.ecb.europa.eu/paym/integration/retail/sepa/html/index.en.html.

[3] Pierre E. BERGER and Isabelle Van BIESEN. Strong customer authentication - new deadline for completing sca implementation - 31 december 2020, 10 2019. Accessed: 2021-06-01. URL: https://www.dlapiper.com/en/france/insights/publications/2019/10/strong-customer-authentication-new-deadline-for-completing-sca-implementation-31-december-2020/

[4] European Comission. Payment services directive (psd2): Regulatory technical standards (rts) enabling consumers to benefit from safer and more innovative electronic payments, 11 2017. Accessed: 2021-06-01. URL: https://ec.europa.eu/commission/presscorner/detail/en/MEMO_17_4961.

[5] European Comission. Payment services directive: frequently asked questions, 01 2018. Accessed: 2021-06-25. URL: https://ec.europa.eu/commission/presscorner/detail/fr/MEMO_15_5793.

[6] European Comission. Frequently asked questions: Making electronic payments and online banking safer and easier for consumers, 09 2019. Accessed: 2021-06-23. URL: https://ec.europa.eu/commission/presscorner/detail/en/qanda_19_5555.

[7] European Payments Council. Sepa instant credit transfer. Accessed: 2021-06-25. URL: https://www.europeanpaymentscouncil.eu/what-we-do/sepa-instant-credit-transfer.

[8] Cryptomathic. What is a qualified certificate? Accessed: 2021-05-18. URL: https://www.cryptomathic.com/products/authentication-signing/digital-signatures-faqs/what-is-a-qualified-certificate.

[9] Turner M. DAWN. Understanding eidas, 01 2016. Accessed: 2021-06-23. URL: https://www.cryptomathic.com/news-events/blog/understanding-eidas.

[10] Deloitte. Psd2 finalised standard on sca and csc: the wait is over, but questions remain. Accessed: 2021-06-01. URL: https://www2.deloitte.com/cy/en/pages/financial-services/articles/psd2-finalised-standard-on-sca-and-csc.html.

[11] Open Banking Implementation Entity. Welcome to the open banking standard. Accessed: 2021-06-01. URL: https://standards.openbanking.org.uk/.

[12] Česká bankovní asociace. Český standard pro open banking, 09 2018. Accessed: 2021-06-01. URL: https://cbaonline.cz/cesky-standard-pro-open-banking.

[13] Česká spořitelna. Kdo jsme. Accessed: 2021-06-24. URL: https://www.csas.cz/cs/o-nas/kdo-jsme.

[14] Facebook. Quickstart: Facebook sdk for javascript. Accessed: 2021-06-25. URL: https://developers.facebook.com/docs/javascript/quickstart.

[15] SEPA for Corporates. Explained: Rts, sca and csc wrt psd2 [epc infographic], 02 2018. Accessed: 2021-06-01. URL: https://www.sepaforcorporates.com/payments-news-2/wtf-is-rts-sca-and-csc-wrt-psd2-by-the-epc-infographic/.

[16] Google. Easily access google apis from java. Accessed: 2021-06-25. URL: https://developers.google.com/api-client-library/java.

[17] Berlin Group. About. Accessed: 2021-06-01. URL: https://www.berlin-group.org/.

[18] Erste Group. About us. Accessed: 2021-06-24. URL: https://www.erstegroup.com/en/about-us.

[19] Erste Group. Erste group annual report 2020. Accessed: 2021-06-23. URL: https://cdn0.erstegroup.com/content/dam/at/eh/www_erstegroup_com/en/Investor_Relations/2020/Reports/AR2020_FINAL_en.pdf?forceDownload=1.

[20] Erste Group. Erste group at a glance. Accessed: 2021-06-24. URL: https://www.erstegroup.com/en/news-media/erstegroup-at-a-glance.

[21] Paysend Group. Sepa transfers: Streamlining euro payments, 07 2019. Accessed: 2021-06-25. URL: https://paysend.com/cs-eg/blog/article-sepa-transfers-streamlining-euro-payments.

[22] Ed HARDT. The oauth 2.0 authorization framework, 10 2012. Accessed: 2021-05-18. URL: https://datatracker.ietf.org/doc/html/rfc6749.

[23] Red Hat. What is an sdk? Accessed: 2021-06-25. URL: https://www.redhat.com/en/topics/cloud-native-apps/what-is-SDK.

[24] iBanFirst. What do psd1 and psd2 mean and why are they important?, 10 2018. Accessed: 2021-06-25. URL: https://blog.ibanfirst.com/en/what-are-psd1-and-psd2-and-why-are-they-important.

[25] IEEE. https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020. Accessed: 2021-05-10. URL: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020.

[26] JetBrains. The state of developer ecosystem 2020 - java. Accessed: 2021-05-10. URL: https://www.jetbrains.com/lp/devecosystem-2020/java/.

[27] Angelica MARI. Psd2 and the api challenge for open banking, 03 2018. Accessed: 2021-06-27. URL: https://diginomica.com/psd2-and-the-api-challenge-for-open-banking.

[28] SSL Market. Qualified website authentication certificate (qwac) pro psd2. Accessed: 2021-05-18. URL: https://www.sslmarket.cz/ssl/quovadis-qualified-website-authentication-certificate-qwac-pro-psd2/.

[29] mBank. Psd2 - payment service directive 2 co pro klienty mbank znamená evropská směrnice o platebních službách? Accessed: 2021-06-01. URL: https://www.mbank.cz/informace-k-produktum/info/jine/psd2.html.

[30] Oracle. Java releases. Accessed: 2021-05-10. URL: https://java.com/en/download/help/release_dates.html.

[31] Aaron PARECKI. Protecting mobile apps with pkce. Accessed: 2021-05-18. URL: https://www.oauth.com/oauth2-servers/pkce/.

[32] Aaron PARECKI. Terminology reference. Accessed: 2021-05-18. URL: https://www.oauth.com/oauth2-servers/definitions/.

[33] European Parliament. Regulation (eu) no 910/2014 of the european parliament and of the council of 23 july 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing directive 1999/93/ec, 01 2016. Accessed: 2021-06-23. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2014.257.01.0073.01.ENG.

[34] RTÉ. Agreement reached on cross-border banking, 03 2007. Accessed: 2021-06-25. URL: https://www.rte.ie/news/2007/0327/87216-banking/.

[35] Kristopher SANDOVAL. What is the difference between an api and an sdk?, 06 2016. Accessed: 2021-06-25. URL: https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/.

[36] Shashvat SHUKLA. Framework vs library vs platform vs api vs sdk vs toolkits vs ide, 06 2017. Accessed: 2021-05-10. URL: https://shashvatshukla.medium.com/framework-vs-library-vs-platform-vs-api-vs-sdk-vs-toolkits-vs-ide-50a9473999db.

[37] W3C. Notes on user centered design process (ucd). Accessed: 2021-05-10. URL: https://www.w3.org/WAI/redesign/ucd.

[38] Yapily. Psd2: What you need to know about screen scraping and apis, 02 2018. Accessed: 2021-06-23. URL: https://www.yapily.com/blog/psd2-screenscraping-apis/.

Contents of enclosed SD card

- `readme.txt` ..... brief overview of SD card contents
- `src`
  - `example` ..... source code of the example project using the SDK
  - `sdk` ..... source code of the SDK
  - `thesis` ..... source code of the thesis in LaTeX
- `text`
  - `thesis.pdf` ..... thesis in PDF