Model Bank: A prototype for UK's first open-source unregulated Open Banking ASPSP API sandbox

Ioana Bacrau

4th Year Project Report Artificial Intelligence and Computer Science School of Informatics University of Edinburgh

2022

Abstract

The initiative of Open Banking was introduced in 2016 and realized in 2018 by OBIE (Open Banking Implementation Entity). Today, out of 1600 firms in the UK FinTech sector, only 245 are currently enrolled in the Open Banking directory. This raises the following questions: Why does Open Banking only hold 15% of the current UK FinTech market? And what can we do to help the remaining 85% join the Open Banking ecosystem? This paper proposes a prototype for an open-source ASPSP API Sandbox as a way to help FinTechs join the Open Banking ecosystem, improve the financial market and enable non-commercial research in this field.

Acknowledgements

I would like to acknowledge and thank my supervisor, Ross Anderson, for helping me with his expertise and continuous support.

Table of Contents

Chapter 1

Introduction

The term online banking became popular in the 1980s and was used to refer to the use of a terminal to access the banking system using a phone line. Today most banks around the world offer an online platform for users to access their financial data from home or anywhere else. Different marketing strategies and benefits across multiple banks have popularised the idea of multi-banking and today a quarter of the UK's consumers hold current accounts with more than one bank [16]. For the individual user, this has created a need for specialised platforms and services (i.e. account aggregation FinTechs) to help manage their finances across multiple providers (i.e. banks), which has started to get met by technology and innovation that aims to compete with traditional financial methods in the delivery of financial services. More than that the Competition and Markets Authority has recently acknowledged that newer banks and FinTechs find it difficult to access the market and grow. Therefore the term Open Banking was introduced in 2018 to encourage the use of Open APIs to enable third parties to build applications and services which integrate with the financial market. However, out of 1600 firms in the UK FinTech sector [9], only 245 are currently enrolled in the Open Banking directory [7]. This raises the following questions: Why does Open Banking only hold 15% of the current UK FinTech market? And what can we do to help the remaining 85% join the Open Banking ecosystem?

1.1 What is Open Banking

The need to exchange data between third-party financial applications and multiple banks was recognised in 2016 when The Competition and Markets Authority (CMA) published a report on the UK's retail banking market. The report found that older, larger banks don't have to compete hard enough to gain customers' business, while newer banks find it difficult to access the market and grow. One of the CMA's recommendations to tackle this problem was Open Banking. Since 2018, the Open Banking Ecosystem has enabled customers and Small to Medium-sized Enterprises (SMEs) to share their current account information securely with third-party providers, who use that data to tailor their apps and services to peoples' specific financial needs. The Second Payment Services Directive (PSD2) came into being in January 2018 in Europe

stated that Open Banking and the use of OpenAPIs would enable third-party developers to build apps, websites, and services around banks and financial institutions.

Today the Open Banking ecosystem is composed of specialized financial entities that serve their individual purpose and interact with each other to create a more robust financial network. When the CMA first announced the Open Banking initiative back in August 2016, nine major banks (known as CMA9) were identified in the UK (i.e. Barclays, RBS, Santander etc.) and required to create open APIs to expose their consumer data to third parties. These nine banks were chosen due to their large combined market share of over 90% of the UK's consumer and small business bank accounts. The CMA9s support the underlying infrastructure such as payment rails and implement checks on their customers such as Anti-Money Laundering and Combating the Financing of Terrorism measures. On top of those, the ecosystem contains over 200 FinTechs (who may be account aggregators, payment initiators, Forex consolidators etc.) who compete to provide UX and added-value services on top of this legacy infrastructure.

1.2 How does Open Banking work

1.2.1 OB entities

Open Banking defines the following entities: [11] (Figure 1.1)

Entity	Definition
PSU	A **Payment Services User** is a natural or legal person making use of a payment service as a payee, payer or both.
AISP	An **Account Information Service Provider** provides account information services as an online service to provide consolidated information on one or more payment accounts held by a payment service user with one or more payment service provider(s).
PISP	A **Payment Initiation Services Provider** provides an online service to initiate a payment order at the request of the payment service user with respect to a payment account held at another payment service provider.
ASPSP	**Account Servicing Payment Service Providers** provide and maintain a payment account for a payer as defined by the Payment Services Regulations and, in the context of the Open Banking Ecosystem are entities that publish Read/Write APIs to permit, with customer consent, payments initiated by third-party providers and/or make their customers' account transaction data available to third-party providers via their API end-points.
TPP	**Third-party Providers** are organisations or natural persons that use APIs developed to Standards to access customer's accounts, provide account information services and/or initiate payments. Third-party providers are either/both Payment Initiation Service Providers (PISPs) and/or Account Information Service Providers (AISPs).

Table 1.1: OB entities

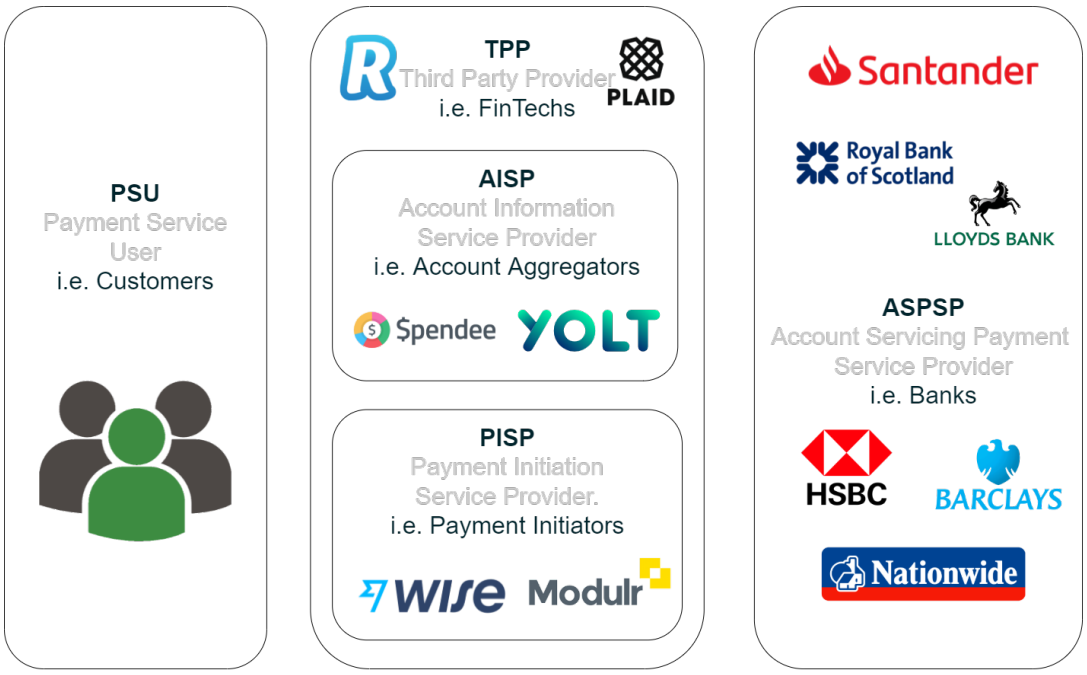

Figure 1.1: Open Banking entities with examples

1.2.2 OB workflow

1.2.2.1 How does Open Banking actually work?

At a high level, banks provide OpenAPIs which can be used by FinTechs to access consumer data on behalf of their users. Open Banking uses 2 key features as a means to authorize data transfer between a FinTech (TPP) and the user-selected bank (ASPSP):

- the OB directory (which ensures that the FinTech can be trusted)
- user consent (which ensures that the user agrees to share their bank data with selected FinTechs)

The workflow in Figure 1.2 describes the steps that need to be taken for a user to authorize data transfer between their FinTech (TPP) and a selected bank (ASPSP).

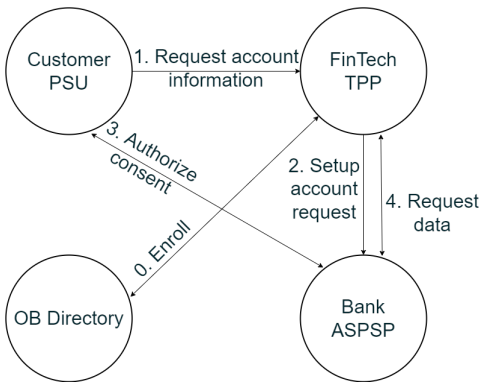

Figure 1.2: OB simplified workflow

As a prerequisite, the FinTech first needs to **enrol** in the Open Banking directory, which is a process that I will describe in the following section. The first step is the *user making a request* to access their data from the FinTech and specify a data source (select a bank, select an account etc.). In step two, the *FinTech makes a request to the bank on behalf of the user* and initiates the setup of a new **consent**. Then for step 3, the user needs to go to their bank website/platform/app and *consent* that they agree with the specified data share,

which effectively authorizes the **consent**. Finally, once the **consent** is live, the FinTech can *retrieve data* from the bank.

1.2.2.2 Enrolling into the OB directory

For FinTechs to access the Open Banking ecosystem and resources, they first need to enrol with Open Banking by following the steps below (snippet from openbanking.com)

1. **Get regulated**. Before providing open banking services, you will need *regulatory permission* from the Financial Conduct Authority (FCA) or your National Competent Authority (NCA). You'll need to demonstrate that you have a *PSD2-compliant business model* and appropriate *data privacy* and *security measures in place*.
2. **Enrol into the directory.** The OBIE Directory enables FinTechs to securely identify themselves to the banks and building societies that hold their customers' financial information.
3. **Test your service.** Our Directory sandbox is a test environment where you can run your service with dummy data or with other firms who are also working towards full enrolment.
   ! Once we've completed our identity and verification checks, our onboarding team will help you get set up.
4. **Launch.** There are just three final steps to going live:
   - Our Directory – we'll complete your enrolment once your regulatory permissions are confirmed.
   - Software statements and digital certificates – you can set up and manage these in the Directory, and start connecting with account providers to make sure your service works with them.
   - Developer Zone – use this to set up the technical configuration for your service.
5. **Support.** Register for our optional Dispute management system (DMS) – a communication framework for account providers and third-party providers (TPPs) to manage enquiries, complaints and disputes.

Figure 1.3: OB enrollment steps. Source: openbanking.org.uk

1.3 Problem description

The enrollment workflow (Section 1.2.2.2) describes a synchronous procedure which presents a considerable bottleneck between steps 2 and 3: developers can not begin developing the API integration until the company meets all the regulatory permissions because the OB software resources and developer portal only become available in the third step. The first step requires FinTechs to get regulated by the FCA (in the UK) which "takes up to 6 months ... but could take **up to 12 months**" [1]. The second step

requires OBIE to ”undertake some validation checks which can take **a few weeks to complete**” [6]. This means that for a start-up FinTech, it may take well over a year before their software developers can even begin to start writing code (and potentially a full quarter for established FinTechs). More than that, step 3 only presumes access to OB provided resources for developers which can not be used in production. For a FinTech to launch their product, they need to integrate real-life bank APIs (i.e. Barclay's ASPSP API) which may delay the first product launch even further.

1.4 Project motivation

This problem with the OB enrollment process and resource access first became apparent to me when I was working on the initial project I proposed for my honours, and it is also the cause why I concluded that my initial project idea was unfeasible in the current industry. For my honours project, I initially wanted to implement an Open Banking API Integration HUB.

Initial Idea: ”Create a developer-friendly aggregation service and payment platform which integrates access to multiple bank APIs and algorithmically interacts with their different authorisation profiles in line with Open Banking Standards. The API Integration HUB will algorithmically recognize the REST API's response format, integrate its authentication protocol and interact with it as well as preserve the lifespan of authorisation tokens, and furthermore manipulate the data extracted and posted to the API to conform to a single standardized format, based on user preferences (i.e. JSON, XML etc). This project aims to prove that the added Open Banking Standard on top of the specifications for FAPI[1], OpenAPI[2], OIDC[3], OAuth 2.0[4] and REST API are enough to automate API integration for TPPs. The HUB will make use of the above-mentioned standards and specifications but also the available individual API documentation to discover and identify ASPSPs' API endpoints, parameters and workflow to essentially get the data from the provider to the consumer while requiring developers to provide as little configuration as possible. This open-source service is intended to be used by TPPs to automatically integrate new ASPSP's APIs and by ASPSPs to automate testing of their deployed APIs. The business goal of this project is to reduce the cost of API implementation and integration, fast-track API upgradeability on both ends, reduce downtime caused by source service failures and overall improve problem resolution between API providers and consumers.”

While I still consider this an intriguing project idea, in these past months, I have learned that the current industry does not meet its prerequisites (or at least it can not be achieved in time for this project's deadlines). For my initial project to take place it requires: 1. one or more OB ASPSP API sandboxes to base the original code development on; 2. a few other commercial ASPSP API sandboxes to test my code on so I can successfully accomplish a comprehensive proof of concept and demonstrate that it can incorporate real-life scenarios. In order to accomplish this project, I have

[1] Financial-grade API https://openid.net/wg/fapi/
[2] OpenAPI https://www.openapis.org/
[3] OpenID Connect https://openid.net/connect/
[4] OAuth 2.0 https://oauth.net/2/

requested help from one of the University of Edinburgh's research partners GOFCoE (The Global Open Finance Centre of Excellence [2]) which advocates for Open Banking. I later learned that their technical infrastructure was not yet able to provide me with a way to access OB sandboxes. Looking for an alternative, I have found different European mock banks which could provide the easy-to-access sandboxes that the project needed. However, after discussing with other Europe-based developers, I have come to understand that the EU Open Banking space does not provide the kind of comprehensive infrastructure that my project needed. But more than that, I have realised that at the moment the UK Open Banking industry does not provide the comprehensive space for independent non-commercial research that the EU one does.

This has led me to believe that for the UK Open Banking to realise its potential, this industry requires opportunities for research and/or non-commercial external testing without being constrained by the regulations that these entities can not meet. Therefore, my current project proposes a prototype for the UK's first open-source unregulated Open Banking ASPSP API sandbox.

Chapter 2

Literature Review and Market Competitors

2.1 Literature Review

Open Banking was first introduced in 2017. Given the novelty of this concept, the available literature on this topic is quite sparse. Moreover, the only academic papers focused on this subject are usually coming from a finance or business point of view. This situation further advances the importance of a project like mine. This stands to prove that, quite literally, individual academics do not have the resources yet to investigate the topic of Open Banking from a technological point of view. More than that the number of industry articles on this topic is inversely proportional to the number of academic ones. Almost all the available technical information about open banking comes from peer articles from industry professionals or directly from the open banking specification.

2.2 Open Banking in the EU

Disambiguation of terms: The term Open Banking is used as an umbrella term across the world to refer to the use of OpenAPIs in the banking industry. Unfortunately, in the UK, the regulation implementing the initiative is also called *Open Banking*. In the EU, the regulation that implements this initiative is called *PSD2*, leaving the term Open Banking to only represent the concept.

2.2.1 PSD2 and XS2A

The revised Payment Services Directive (PSD2) is a European regulation for electronic payment services. It seeks to make payments more secure in Europe, boost innovation and help banking services adapt to new technologies. PSD2 is evidence of the increasing importance Application Program Interfaces (APIs) are acquiring in different financial sectors. PSD2 came into force in Europe in January 2018. At this point, all regulated entities (Payment Service Providers) had to ensure that they individually

comply with PSD2 and the Regulatory Technical Standards (RTS) set out by the European Banking Authority (EBA). The European Open Banking variant is defined at the PSD2 Article 66 [13] and its amended RTS with the complete name "Commission Delegated Regulation (EU) 2018/389 of 27 November 2017 supplementing Directive (EU) 2015/2366 of the European Parliament and of the Council with regard to regulatory technical standards for strong customer authentication and common and secure open standards of communication".

As referenced by Preta in their 2019 "Internet Security & eIDAS Certificates" guide [15]: Many experts believe that the financial industry is expected to organise itself to make sure that the implemented solutions for PSD2 are interoperable, which has exposed the electronic financial industry regulation to interpretation. The EU COM refused to define or standardize an API service for PISPs and ASPSPs. Each bank could define its own API as long as it meets the business requirements of the RTS Section 2, and the market was not happy about that. Everybody has expected something like a common SEPA rulebook [12] for all participants. The FinTech PISP were upset because they would have had to implement consumer APIs for a thousand different banking APIs. The ASPSPs were not happy, because each API has to be approved by their NCA (National Competent Authority, in Germany the BaFin). This could be a never-ending story if some PISPs would claim that the ASPSP's API is not compliant or not working.

For Third Party Providers (TPPs) to gain access to (Payment Service User) PSU data, they need to meet the Access to Account (XS2A) requirements of PSD2. Finally, the Berlin Group (despite its name the Berlin Group is a pan-European standards initiative from the payment sector) has developed a specification [14] which has been widely adopted, however, there are other implementation and national standards for XS2A.

2.2.2 EU vs UK Open Banking API Sandboxes

2.2.2.1 EU Open Banking API Sandboxes

Since PSD2 is a wider and less comprehensive standard than the UK Open Banking one, different banks have implemented vastly different APIs. In Europe, the Open Banking ASPSP API implementations are left at the discretion of each independent ASPSP with no strict standard or mandatorily required support (i.e. standard for documentation and provision of a comprehensive testing environment). Because of this, TPPs are expected to develop thousands of different API implementations, navigate vague and incomplete documentation and chase other developers (who are employed by the bank) for support and guidance.

This has created space in the industry for a new type of business venture: Bank API Sandboxes. This means that other businesses are now creating testing environments with APIs and dynamic test data that adhere to the PSD2 and XS2A guidelines in an effort to help TPP developers develop infrastructure to integrate real-life ASPSP APIs. There are 2 types of sandboxes which seem to be currently gaining popularity within the markets:

1. Test Banks: completely independent virtual ASPSPs with external APIs for

OpenBanking interaction purposes, and internal APIs for creation and maintenance of dynamic test data (i.e. MockBank [5])

2. Real-life bank APIs mirrors: API servers which imitate real-life ASPSPs API implementations with dynamic or static test data (i.e. Golden Dimension LLC [3])

2.2.2.2 UK Open Banking API Sandboxes

In the UK, the Open Banking Standard is stricter. It enforces banks to provide the required support in order for TPPs to join the OB ecosystem, which translates to better, more comprehensive sandboxes maintained by the ASPSPs. For a TPP to enrol and become regulated, before they access real-life PSU data, they need to test and demonstrate that their API integration is functional and meets the OB requirements, which implies a considerable amount of testing the integration before actually being able to access the bank API test environment. Because the UK standard is more comprehensive these API sandboxes are regulated and maintained by the ASPSPs themselves, it seems that the UK space doesn't need the above-mentioned European API mirrors. However, this paper argues that the need for "test banks" remains unsatisfied.

2.3 Open Banking in the UK

2.3.1 OB Developer Zone and Ozone

The Open Banking initiative in the UK proposes the use of Ozone Model Banks for API integration testing. This sandbox mimics the functionality that a typical ASPSP would provide and allows basic application API integration and testing to be performed. This is probably the best solution in the UK space for early interaction with an Open Banking API. Referring back to the OB enrollment workflow (Section 1.2.2.2), Ozone is the service you get access to at step 3, which implies that this interaction still requires steps 1 and 2, and the bottleneck that was identified in this paper is still problematic.

From Ozone's specification, the following prerequisites need to be met before onboarding onto Ozone:

1. The TPP has registered on the Directory Sandbox.
2. The TPP has at least one software statement created on the Directory Sandbox environment.
3. The TPP has at least one transport certificate created for each of its software statements.
4. The TPP has at least one redirect URI for each of its software statements.
5. The TPP has a copy of the OB root and issuing certificate attached.

Figure 2.1: Ozone integration prerequisites. Source: openbanking.atlassian.net [4]

2.3.2 API Integrators

Another solution to facilitate API integration is middle-ware business-to-business apps. The UK space (and not only) supports several middle-ware apps which act as a middleman between FinTechs and banks. They are already enrolled and integrated with several ASPSPs, so FinTechs only need to integrate with the middle-ware app once and they get instant access to all the partner ASPSPs. Some of the most noticeable apps are:

- TrueLayer - enables companies to capitalise on new Open Banking initiatives in the UK, and the broader, European-wide PSD2 rules by providing secure, clear and simple access to banking infrastructure.
- Tink - their open banking platform enables banks, FinTechs, and startups across Europe to develop data-driven financial services. Through one API, Tink allows customers to access aggregated financial data, initiate payments, enrich transactions and build personal finance management tools.
- Budget Insight - provides APIs to access accounts on more than 300 European banks and 200 invoice providers

We recognize middle-ware apps as a good solution for FinTechs to gain access to Open Banking data, without having to deal with the Open Banking enrollment, however, this is not a core problem solution and more than that it introduces a middle-man which can raise issues of its own.

Chapter 3

Project description

This project aims to implement a prototype of an open-source Open Banking ASPSP, but not actually regulated by Open Banking.

Project name breakdown

- Model Bank = the name was picked to suggest that this bank models the Open Banking standard
- a prototype = this project only focused on a limited scope of open banking, and stands as proof of concept
- for UK's first = in my research, I was unable to find any similar solution within the UK space
- open-source = this project serves as a demo and a test tool for developers, which is realized by it being openly available and accessible for free
- unregulated = this solution completely removes the need to interact with OB enrollment or complicated centralised authentication means
- Open Banking = this project aims to fully implement the data retrieval functionality, API functionality and software architecture as described in the Open Banking specification within the defined scope
- ASPSP API sandbox = this paper's main deliverable is an Account Servicing And Payments Service Provider as an API testing tool and environment

3.1 Scope

This project implementation follows the **Open Banking Read-Write API Profile - v3.1.9** Specification [8]. For the application of this prototype, I have chosen to only implement the necessary ASPSP functionality required to support a basic AISP (highlighted in bold below).

The Open Banking Standard defines the following API profiles:

- **Account and Transaction API Profile** - describes the flows and common functionality for the Accounts and Transaction API, which enables AISPs
- Payment Initiation API Profile - describes the flows and common functionality for the Payment Initiation API, which enables PISPs
- Confirmation of Funds API Profile - describes the flows and common functionality for the Confirmation of Funds API, which enables Card-Based Payment Instrument Issuers
- Variable Recurring Payments API Profile - describes the flows and common functionality for setting up VRP Consents and subsequently creating one or more payment orders that meet the limitations set by the VRP Consent
- Event Notification API Profile - describes the flows and common functionality to allow a TPP to receive event notifications. This is used by ASPSPs to notify TPPs when a data change has occurred (i.e. user changes their account nickname)

Within the Account and Transaction API Profile, the following resources are further described:

- **Account Access Consents** - used to create a new account-access-consent resource, retrieve the status of account-access-consent resource and delete the account-access-consent resource
- **Accounts**
- **Balances**
- **Transactions**
- Beneficiaries
- Direct Debits
- Standing Orders
- Products
- Offers
- Parties
- Scheduled Payments
- Statements

Given the limited amount of time assigned for this type of project, I have limited the scope of this project to only basic AISP functionality (Account and Transaction API Profile), containing only 4 types of objects: Consents, Accounts, Balances and Transactions (however, these objects contain other nested objects which I will describe later in this paper)

This project attempts to facilitate the Open Banking initiative by making it simple and readily available. In my research, I have concluded that a new FinTech may need to

wait up to a year before being to able to access Open Banking software resources, and up to a quarter for already regulated FinTechs. For this reason, this ASPSP fully implements the API specification for the defined scope, but also completely removes the Open Banking directory in an attempt to break the boundary of regulation imposed by the Open Banking ecosystem.

3.2 Components

This project delivers 2 providers: An Account Servicing Payment Service Provider (ASPSP) and an Account Information Service Provider (AISP) used to demonstrate end-to-end functionality but also used as a testing tool. These 2 providers implement 2 Web Applications with individual UIs, 2 SQL databases, a Web API with Swagger documentation and an object library.

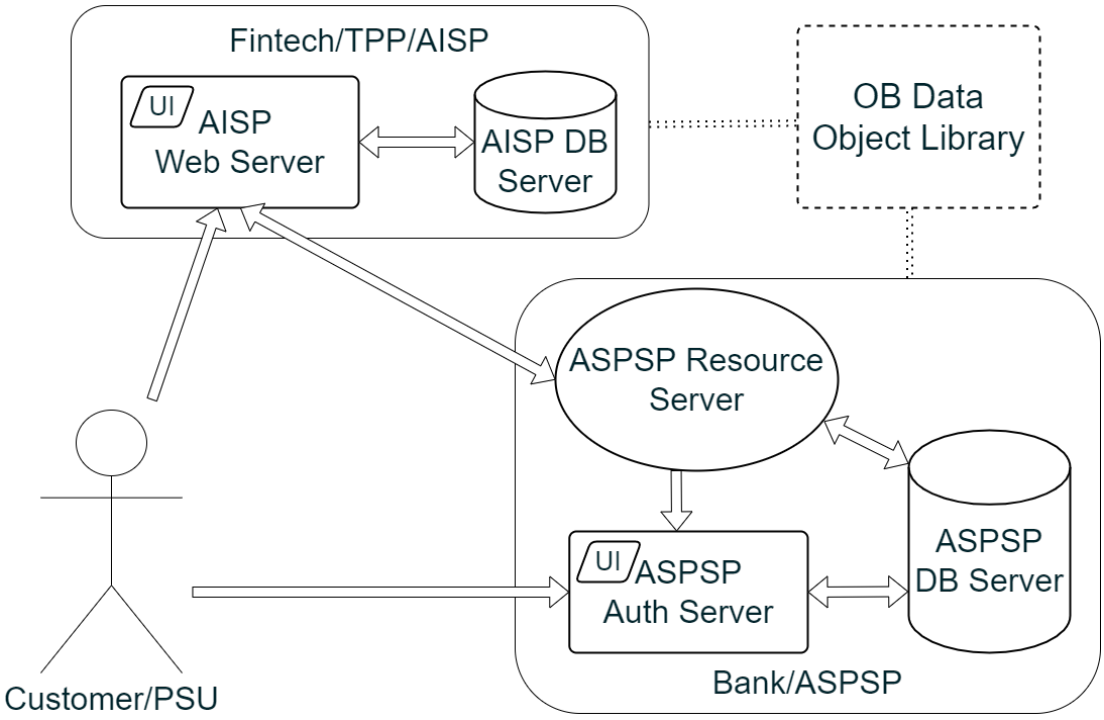

Figure 3.1: Components and interaction diagram

3.2.1 OB Data Object Library

3.2.1.1 Description

This library project contains all the necessary object class definitions defined by the Open Banking Specification. The scope described above references 4 types of resources (Account Access Consent, Account, Balance and Transaction) which will be transferred between the Web API provider and the client. Following the Specification, these 4 resources are composed of 19 enumerations, and 44 object classes.

UML diagrams for these resources and the nested classes within them are available in Appendix B

3.2.1.2 Structure

To maintain the high quality of the library, this project has sourced the OB-specific objects directly from the Specification together with their description. Some alterations have been made when the specification did not provide a feasible implementation.

This paper together with the Specification identifies 3 types of Enumerations:

a) Static Enumerations

b) Namespaced Enumerations

c) ISO Enumerations

And 2 types of Object Classes:

d) MaxText

e) OB Versioned

a) Static Enumerations
OBIE Specifications include various fields of Enumerated data types. Static Enumerations values are fixed to a OBIE-defined set of alternatives. This type presented no challenge to implement, however, out of 16 enumerations, 2 of them were missing definitions (**OBExternalPermissions1Code** and **OBExternalRequestStatus1Code**)

Example enumeration definition for **OBAccountStatus1Code**:

Name	Definition
Enabled	Account can be used for its intended purpose.
Disabled	Account cannot be used for its intended purpose, either temporarily or permanently.
Deleted	Account cannot be used any longer.
ProForma	Account is temporary and can be partially used for its intended purpose. The account will be fully available for use when the account servicer has received all relevant documents.
Pending	Account change is pending approval.

b) Namespaced Enumerations
Namespaced Enumerations values are flexible with an initial OBIE-defined set of alternatives, and ASPSPs can use/extend these alternatives. This type presented 2 significant challenges caused by incomplete specification and the format of the values (i.e. **UK.OBIE.SortCodeAccountNumber**). The object library project is implemented in **C# .Net Standard** which does not support dots (.) in the enum type values, therefore a direct implementation of the Specification was not possible. This implementation follows the Open Banking Read-Write API Profile - v3.1.9, however, this version does not provide a description of the Namespaced Enumerations, so the value definitions have been extracted from v3.1.

Example enumeration definition for **OBExternalAccountIdentification4Code**:

Name	Definition
UK.OBIE.BBAN	Basic Bank Account Number (BBAN) - an identifier used nationally by financial institutions, ie, in individual countries, generally as part of a National Account Numbering Scheme(s), to uniquely identify the account of a customer.
UK.OBIE.IBAN	An identifier used internationally by financial institutions to uniquely identify the account of a customer at a financial institution, as described in the latest edition of the international standard ISO 13616. ”Banking and related financial services - International Bank Account Number (IBAN)”.
UK.OBIE.PAN	Primary Account Number - identifier scheme used to identify a card account.
UK.OBIE.Paym	Paym Scheme to make payments via mobile.

c) ISO Enumerations
The CMA Order requires the CMA9 Banks to be aligned with the Regulatory and Technical Standards (RTS) under PSD2. A previous draft of the EBA RTS required that the interface ”shall use ISO 20022 elements, components or approved message definitions”. In keeping with that requirement, the API payloads are designed using the ISO 20022 message elements and components where available. The ISO object classes implementation presented various issues which this paper will discuss in Chapter 4 Subsection 4.1.1. For the purpose of this implementation, all the ISO Enumeration types have been replaced with native type string. These classes have been archived and removed from the functional code.

d) MaxText Object Classes
The literature specifies a number of different classes generically called **MaxNText** where **N** is a number between 35 and 256 (i.e. Max40Text), according to version 3.1.9 of the Specification. They act as a native string type but with a length limit. For this application, I initially tried to implement these at the code level, by overwriting the native type string, but this has caused significant issues with serialization, therefore these classes have been archived and removed from the functional code. I consider them very useful for the database layer implementation, therefore the task of enforcing them will be considered for a future non-prototype.

e) OB Versioned Object Classes
The Specification presents definitions for 27 Object Classes which include the ”OB” prefix and a version indicator suffix (i.e. OBCashAccount6). They are used for API response data modelling. For the 4 resources used by this application, they follow a similar nesting pattern: OBReadX, OBReadDatax, OBX. Example nesting for Accounts resource:

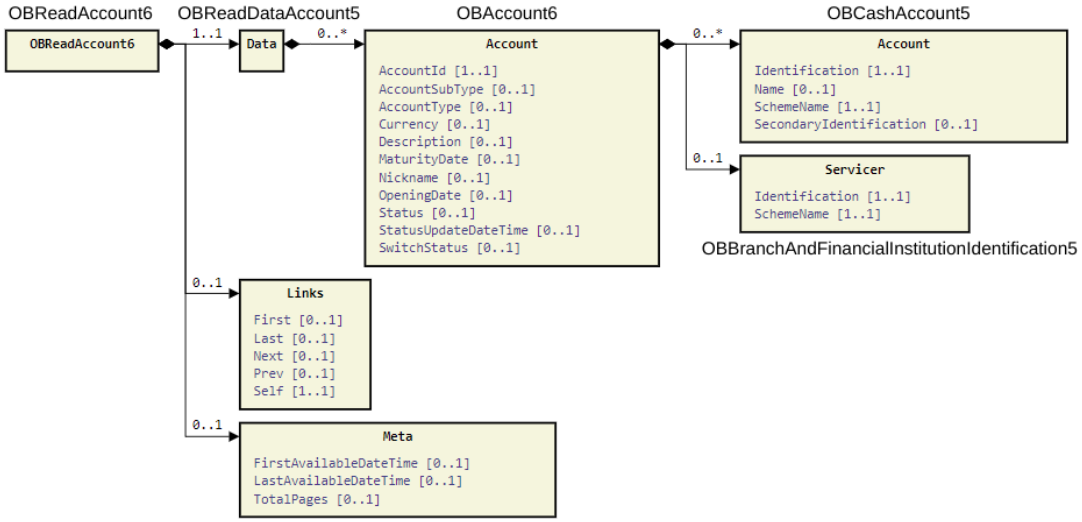

Figure 3.2: Account nesting example

3.2.1.3 Integration

This project contains the type definitions for all the classes defined by the Open Banking Specification. Since both the AISP and the ASPSP require these objects I have made an implementation decision to define them in an independent library project which holds no functionality. This way both providers can source the object definitions from this library. All objects have been acquired from the Open Banking v3.1.9 specification by recursively interrogating the 4 main resources included in this project's scope.

3.2.2 ASPSP database

The ASPSP database is implemented as a SQL Server Database. The database is hosted on an Azure SQL server, but can also be recreated/deployed locally and it contains tables for all of the objects described above. The data in the database is static, apart from consents which are dynamically created and updated by the ASPSP resource server according to the Open Banking Specification. All the static data is directly sourced from the examples in the Specification in order to validate its authenticity and serve as a means to test the contents of the ASPSP API responses. By providing static data, the system is capable of supporting multiple clients without requiring them to register or having to allocate individual space for each client. Moreover, the database it contains stored procedures to retrieve or save data, the stored procedures act as a middle layer between the code and actual exposed tables, so the code never accesses a table directly. All the table and stored procedures have respective SQL scripts to create which are available in this project's repo; this enables developers to recreate the database components on their own environment but also provides them with complete power to modify and adapt this solution according to the requirements of their business model.

3.2.3 ASPSP Resource Server

3.2.3.1 API Endpoints

The ASPSP Resource Server is a WebAPI with Swagger Documentation for both endpoints and schemas (Figures 3.3 and 3.4). It implements all the server-side required endpoints for AISP functionality and the resources within the scope as described by the Specification (Figure 3.5).

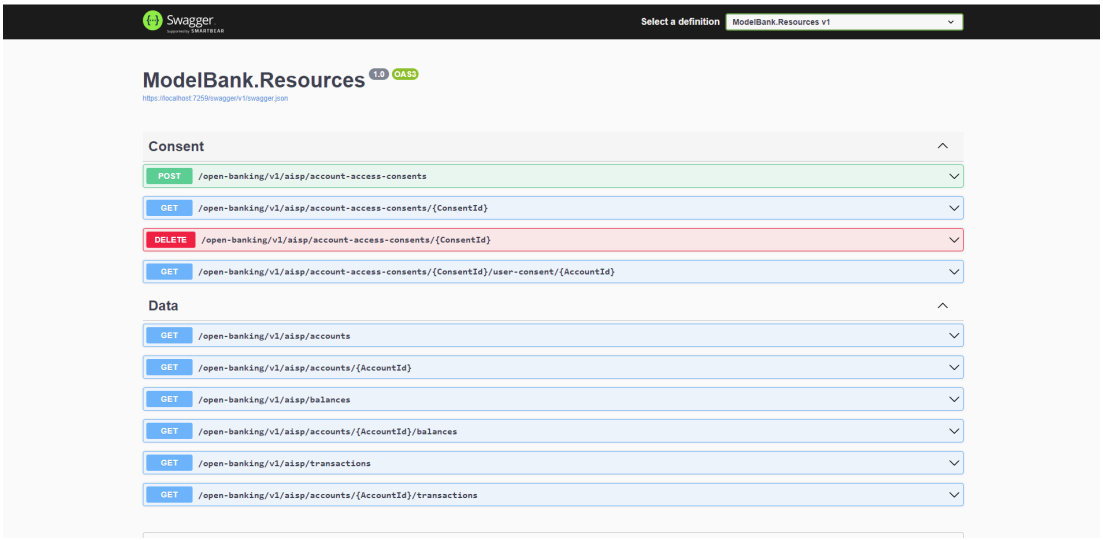

Figure 3.3: ASPSP Swagger Endpoints

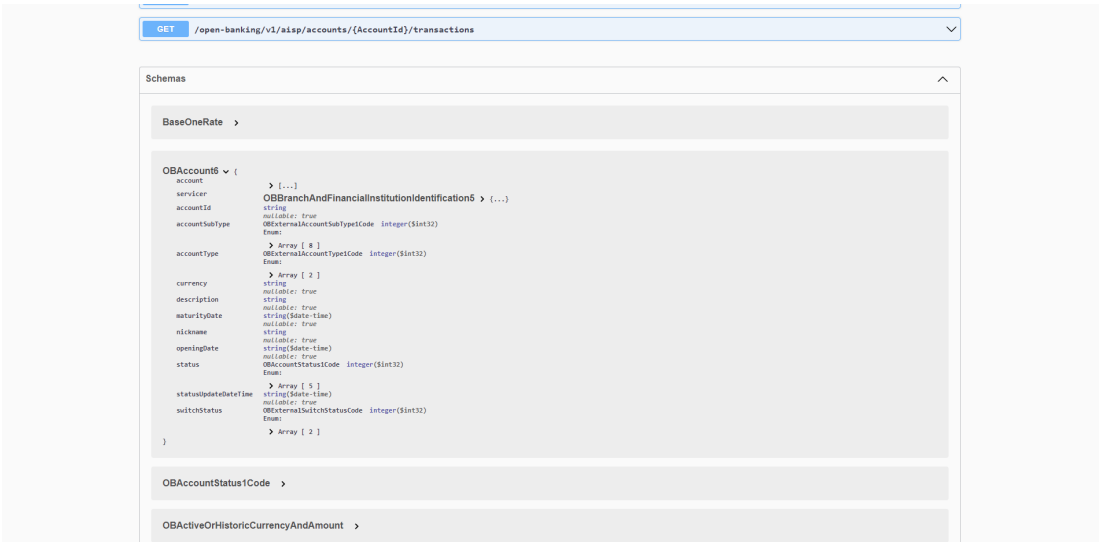

Figure 3.4: ASPSP Swagger Schemas

3.2.3.2 Structure and Integration

The code base implements SOLID principles hence it uses a modular architecture which effectively compartmentalises functionality into specialized classes(Figure 3.6). The controllers are responsible for receiving REST API requests which they process

Permissions	Endpoints	Data Cluster Description
ReadAccounts	/accounts /accounts/{AccountId}	Ability to read account information
ReadBalances	/balances /accounts/{AccountId}/balances	Ability to read balance information
ReadTransactions	/transactions /accounts/{AccountId}/transactions	Ability to read transaction information

Figure 3.5: OB Specification Endpoints

and then they call the back-end layer (Database static library class). The Database static library uses the SqlClient to query the database, reconstructs the Objects from the query results using deterministic type constructors and then passes the resulting objects back to the controller which will serialize and use them as request responses.

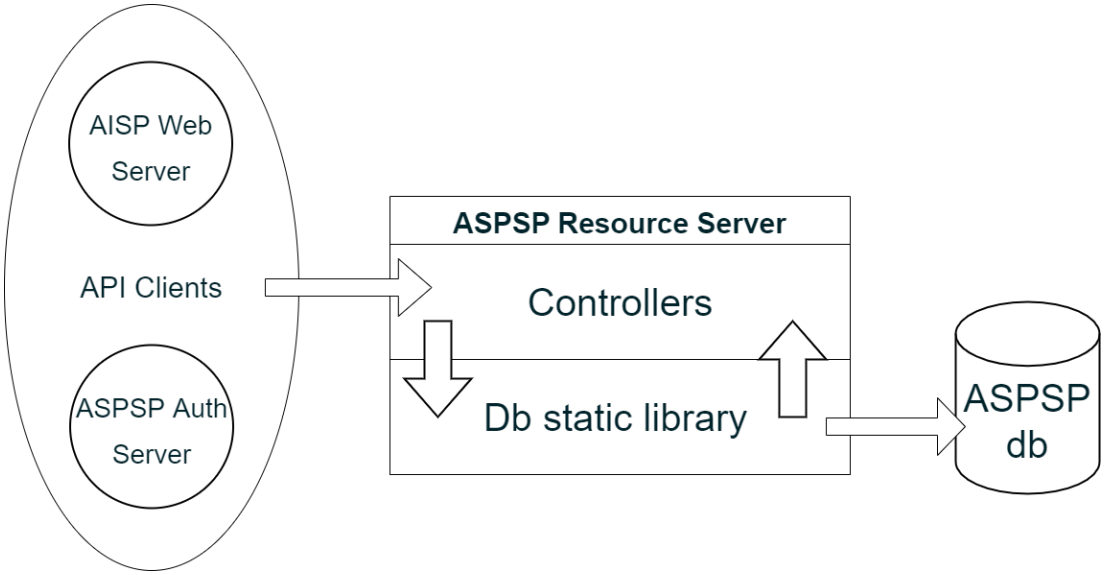

Figure 3.6: ASPSP Structure

3.2.3.3 Deterministic type constructors

The ASPSP back-end static library component implements functionality to effectively retrieve data from the database as well as deterministic type constructors from database query results to C# nested objects using the C native generic type **T** and Reflection (Figure 3.7).

3.2.4 ASPSP Authentication Server

The authentication server holds and manages basic PSU login functionality. It contains 2 UI pages. The first one which displays a standard home page at root level, which signifies that the service is on standby. The second one represents a login page (Figure 3.8) where the PSU gets redirected from the AISP to authenitcate with the ASPSP and authorize the consent. In the back-end, the auth server is responsible for sending a request to the ASPSP Resource Server to authorize the user consent for the authenti-

```
private static void GetObject<T>(DataRow row, ref T obj)
{
    Type myType = obj.GetType();
    IList<PropertyInfo> props = new List<PropertyInfo>(myType.GetProperties());

    foreach (PropertyInfo prop in props)
    {
        try
        {
            if (row.Table.Columns.Contains(prop.Name) && row[prop.Name] != null && row[prop.Name].GetType() != typeof(DBNull))
                prop.SetValue(obj, (string)row[prop.Name.ToString()]);

        }
        catch (ArgumentException)
        {
            if (Nullable.GetUnderlyingType(prop.PropertyType) == null)
            {
                var type = prop.PropertyType;
                // either a date
                if (type == typeof(DateTime)) prop.SetValue(obj, DateTime.Parse((string)row[prop.Name]));
                // or an ICollection
                else if (type == typeof(ICollection<string>))
                {
                    prop.SetValue(obj, JsonSerializer.Deserialize<List<String>>((string)row[prop.Name]));
                }
                // or an enum
                else prop.SetValue(obj, Enum.Parse(type, (string)row[prop.Name]));
            }
            [ ... ]

        }
    }
}
```

Figure 3.7: Code Snippet of Type Constructor

cated account. At the database level, the auth server interacts briefly with the ASPSP database to verify the PSU login credentials.

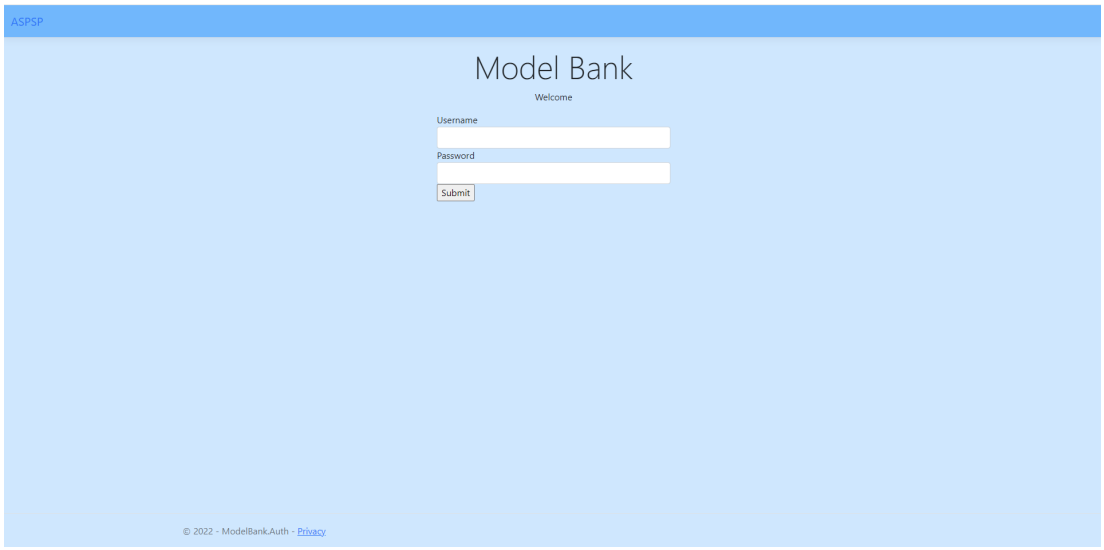

Figure 3.8: Authentication Server Login Page

3.2.5 AISP Web Application

3.2.5.1 Description

The OB Standard does not include an implementation specification for AISPs, therefore this whole Web Application has been specialized for the purpose of this project. It uses the C# Model-View-Controller architecture and it serves mainly as a demo re-

source to describe the workflow and the process. The UI is intended to both explain the Open Banking workflow as well as illustrate what an AISP UI would look like. The back-end is intended to provide a clear and concise API client for the ASPSP API following the Open Banking Specification.

3.2.5.2 UI

The UI's purpose is to explain the Open Banking workflow from both a theoretical and a practical point of view. The UI pages present 2 panels: (Figure 3.9)

1. theoretical list of steps on the left-hand side
2. practical functional UI on the right-hand side (Happy Fintech)

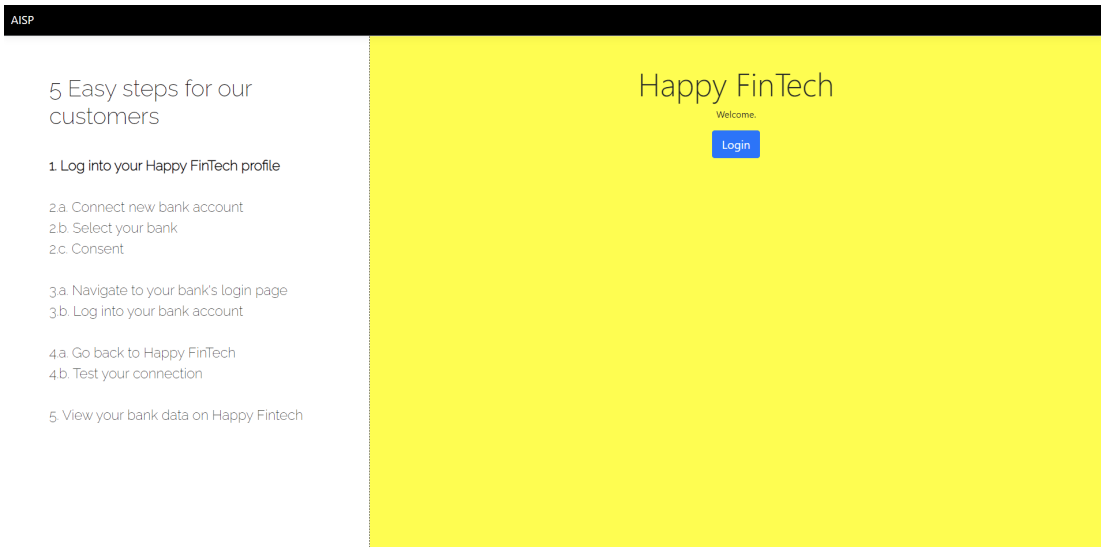

Figure 3.9: AISP Home Page

The UI uses Session data to keep track of user login and progress, different checkpoints in the workflow are used to highlight different steps in the left-hand-side list.

On the user profile page, the user is presented with a list of buttons: they can either connect a new bank account (Figure 3.10) or if they have a connected ASPSP account (Figure 3.11), they can view their data (accounts, balances or transactions).

3.2.5.3 Structure and Integration

The code base implements SOLID principles hence it uses a modular architecture which effectively compartmentalises functionality into specialized classes(Figure 3.12). The UI is realised by the View pages, the Controllers act as middle-ware and the back-end contains 2 specialized library classes which are responsible for implementing the SqlClient (for database integration) and the HttpClient (for sending web requests to the ASPSP API).

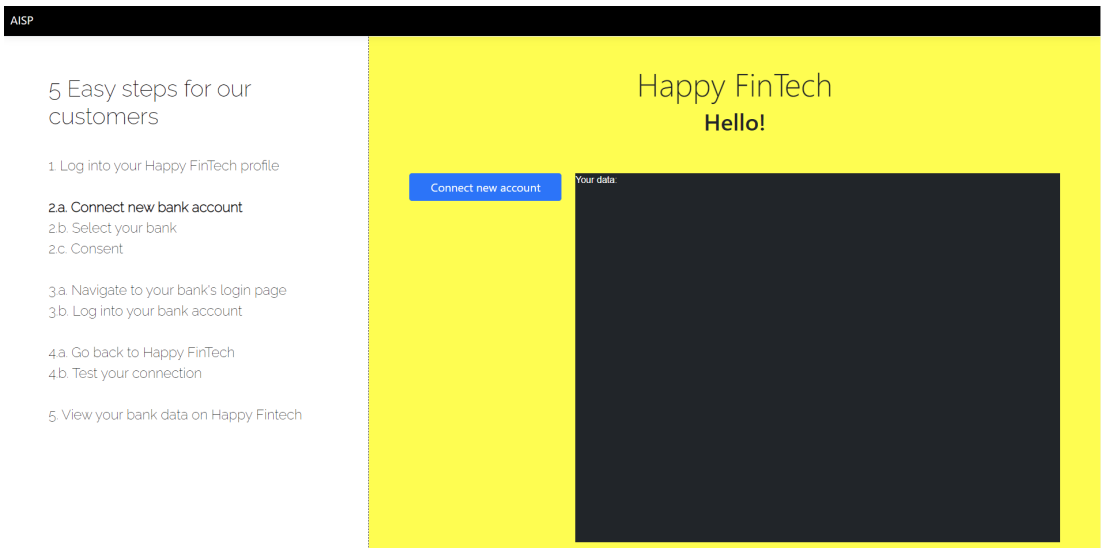

Figure 3.10: AISP Profile Page (no connected account)

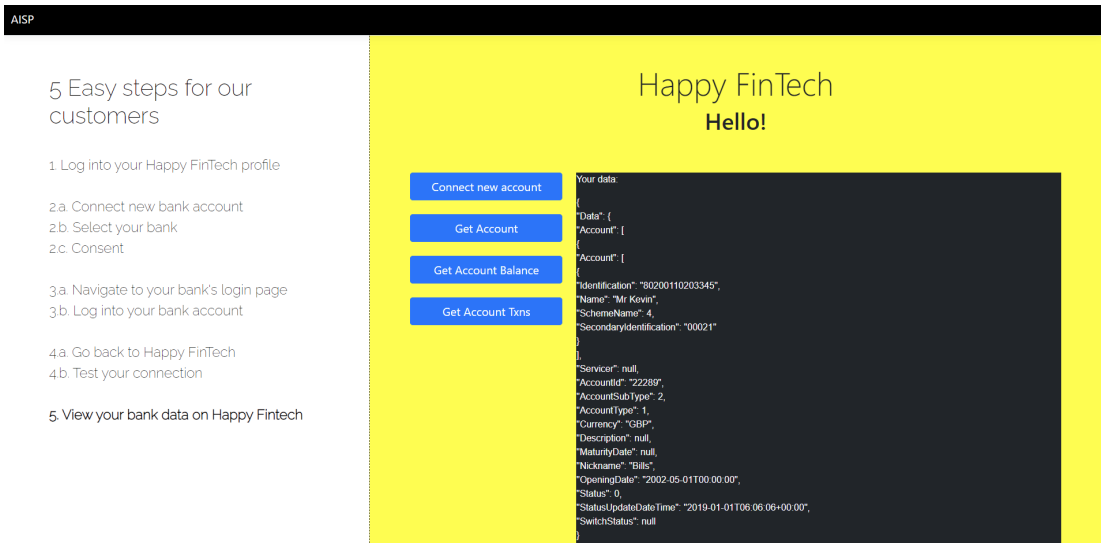

Figure 3.11: AISP Profile Page (one connected account)

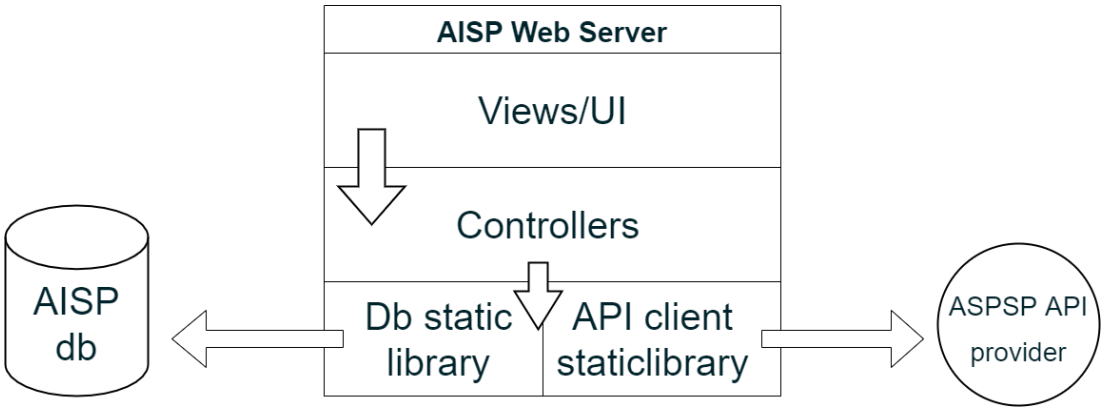

Figure 3.12: AISP Web Application structure

3.2.6 AISP database

The AISP database is an SQL Server database. For this application, it is hosted on the same Azure SQL server as the ASPSP database. It was designed to only support basic login functionality with dummy static login data, a login table and a stored procedure to authenticate the AISP user.

3.3 Technical Specification and System Requirements

It is a well-known fact that the banking and financial industries are commonly using C# or Java, and structured databases, most commonly SQL or some variation of it. Therefore this project uses C#.NET6 and .NetStandard for object libraries, with an SQL Server Database hosted on an Azure SQL Server. The project was deployed on an Azure VM, and I have used GitHub for version control.

3.3.1 Project source

- GitHub repository: https://github.com/biscuesuper/ModelBank
- Local system requirements: Windows 10, C#/.NET Core, .net6, SQL Server 2019

3.3.2 Hosted environments

- Application Server: http://model-bank.centralus.cloudapp.azure.com/
- Language: C#, .Net Core 6
- Database Server: model-bank-sql.database.windows.net
- SQL Server version: SQL 12.0.2000.8

3.4 Benefits and potential

The key 2 benefits of this project are: First of all, this project facilitates software development for open banking integration to begin ahead of time, therefore making the enrollment process asynchronous between programmers and admins(management, legal etc). Second of all: This project represents a complete end-to-end solution which provides the developers with enormous testing power over their integration. After discussing this project with some local and European FinTechs, they have concluded that, if implemented on a large scale, FinTech developers could easily integrate this with their solution as a robust and readily available code base and save time and cost of production.

Chapter 4

Discussion of the work undertaken

4.1 Challenges and lessons learned

4.1.1 ISO Enumeration Implementation

Open Banking implements ISO20022 and defines 5 Enumerations using ISO values. The problem with this is that for some of those enumerations, there could be thousands of values (i.e. for the Country Code ISO enumeration alone there are over 3000 values, according to Wikipedia [10]), therefore a hard-coded implementation of these values didn't seem like the best option. The Specification provides regex patterns to validate a given input string, however, the given patterns only check for the size and format of the string, not its actual value. After conducting some research on the standard ISO implementation in the industry, it seems the most often used solution is storing those values in a database table, and then comparing it against the received input to check whether a given string is a valid ISO code; I believe this solution is not accurately following the Specification and it furthermore introduces more risk into the system, however, it appears to be the industry standard.

4.2 Object mappings on relational databases

The Specification describes several nested objects which can not be straightforward represented on a relational database. Since the objects also do not provide ID properties, I have had to add them myself. The challenge came when I had to choose between child or parent referencing. The rules I used are:

- If the parent object contains a child object, then I create an ID for the child and assign responsibility to the parent object to know ”who their child it”.
- If the parent object contains an array of child objects, then I create an ID for the parent object and assign responsibility to the child object to know ”who they belong to”.

Unfortunately, given the nature of the objects described in the Specification, the objects are not consistently backwards or forwards referenced, so the result ended up with a

mix of those 2 rules.

4.3 Next steps

The next step for this prototype is to implement OAuth into the project, however, in my research, I have found that every tutorial provides a different implementation, and I believe that would affect the ”standard” quality of my project. More than that every implementation uses different libraries, which includes more dependencies and increases complexity. The OAuth standard is explicitly mentioned in the Specification, and it is essential for the AISP authentication and the ASPSP's consent management (the consent and access tokens are carried over the OAuth protocol request headers, more specifically the bearer). Therefore given the importance of this feature, more research has to be conducted to ensure a robust and secure OAuth implementation.

Chapter 5

Conclusions

This paper concludes that behind a centralized wall of FCA regulations and OBIE enrollment, Open Banking can not achieve its promise of being actually ”open”. I recognize that it is important to prevent unregulated entities from joining the live financial market, however, this paper argues that this does not justify the lack of access to this governmental initiative for non-commercial research. The CMA report found that in the market of 2016, it was difficult for SMEs to compete against larger financial institutions, so they have proposed Open Banking; however, Open Banking, in isn't current state in the UK, sits behind OBIE and the centralized ecosystem is still playing in favour of the larger and already well-established institutions while delaying SMEs' access to the market and potentially even denying non-commercial research altogether. More than that this paper notes that, in the current market, trying to integrate separate services of proprietary software presents a significant challenge for developers. The lack of end-to-end open software causes a deficit in testing which eventually leads to less robust financial software which in turn may present a global security and finance risk to all entities who participate in the international finance market. This project has also raised some concerns about the available specification, its significant issues and how those may affect individual implementations which have raised the question: are we sure our trusted financial institutions are implementing this ”right”? And how can we validate this since the other implementations are sitting behind the OBIE ecosystem wall? Finally, this paper argues the importance of this project and similar open-source ”unregulated” open banking APIs and API sandboxes.

Bibliography

[1] Financial conduct authority: Authorisation summary. Last accessed on 05/04/2022.

[2] The global open finance centre of excellence. Last accessed on 01/12/2021.

[3] Golden dimension. Last accessed on 01/12/2021.

[4] Integrating a tpp with the model bank provided by obie. Last accessed on 05/04/2022.

[5] Mockbank. Last accessed on 01/12/2021.

[6] Open banking account providers. Last accessed on 05/04/2022.

[7] Open banking fintechs. Last accessed on 01/12/2021.

[8] Open banking read-write api profile - v3.1.9. Last accessed on 05/04/2022.

[9] United kingdom - country commercial guide. Last accessed on 05/04/2022.

[10] Wikipedia, list of iso 3166 country codes. Last accessed on 05/04/2022.

[11] Open Banking. Open banking glossary. Last accessed on 01/12/2021.

[12] European Payments Council. Sepa credit transfer rulebook and implementation guidelines, 2021. Last accessed on 01/12/2021.

[13] EEA. Psd2 directive, 2015. Last accessed on 01/12/2021.

[14] Berlin Group. Berlin group implementation, 2021. Last accessed on 01/12/2021.

[15] Preta. Understanding internet security eidas certificates. 2017.

[16] Financial Times. One in four uk consumers has more than one current account.

Appendix A

Acronyms and Abbreviations

- AISP = Account Information Service Providers
- ASPSP = Account Servicing Payment Service Providers
- CMA = The Competition and Markets Authority
- FAPI = Financial grade API
- OB = Open Banking
- OBIE = Open Banking Implementation Entity
- OIDC = Open ID Connect
- PIISP = Payment Instrument Issuer Service Providers
- PISP = Payment Initiation Service Providers
- PSD2 = Second Payment Services Directive
- PSU = Payment Service User
- Screen Scraping = Screen scraping is the process of gathering data from one app by inputting user credentials (such as username and password) and displaying that data somewhere else
- SME = Small- to Medium-sized Enterprise
- TPP = Third-Party 'Payment Service' Providers

Appendix B

Open Banking resources UML diagrams

B.1 Account Access Consents

Source: openbanking.github.io

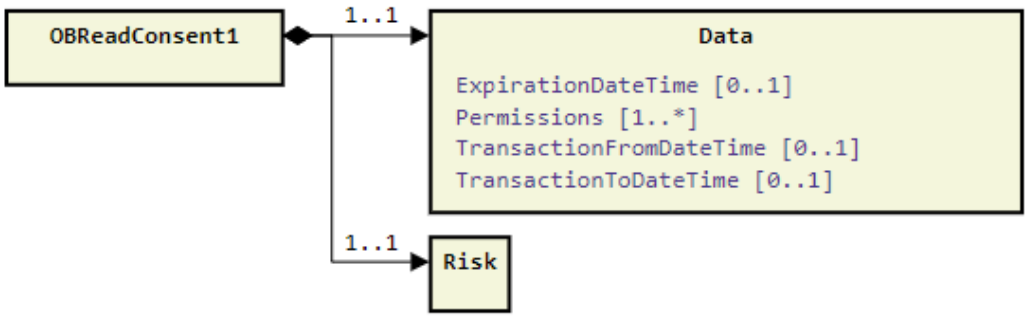

Figure B.1: Account Access Consents UML Diagram. Source: https://openbankinguk.github.io/

B.2 Accounts

Source: openbanking.github.io

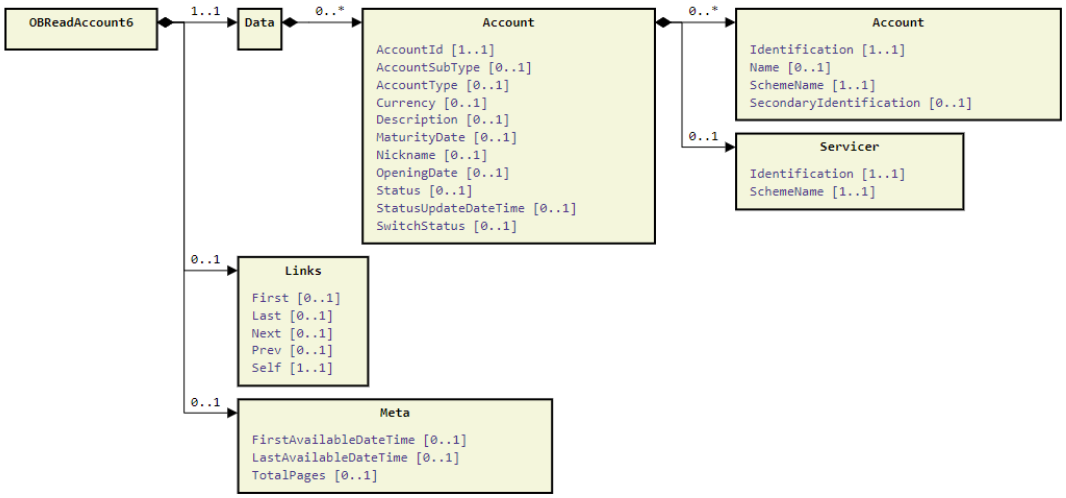

Figure B.2: Accounts UML Diagram. Source: https://openbankinguk.github.io/

B.3 Balances

Source: openbanking.github.io

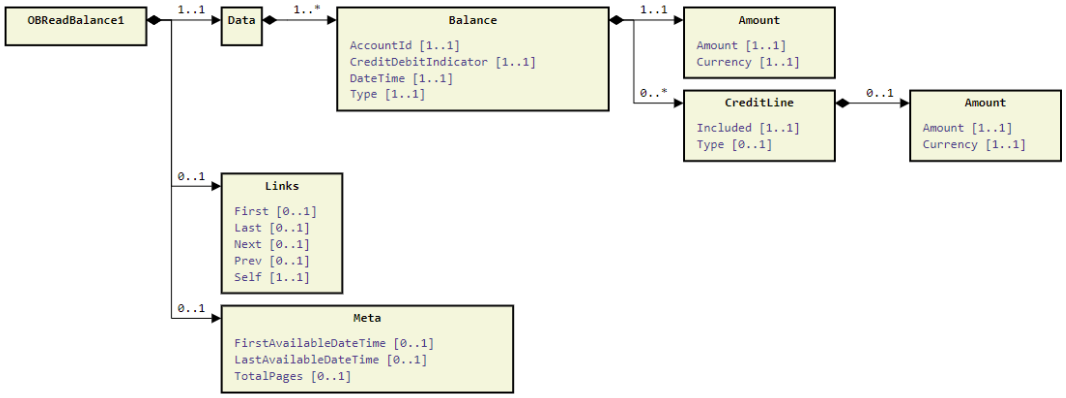

Figure B.3: Balances UML Diagram. Source: https://openbankinguk.github.io/

B.4 Transactions