TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Aleksandr Babõkin 174960IDDR

Design and Implementation of the Personal Data Processing Component for the AfterPay Cross-organisational Processes

Diploma Thesis

Supervisor: Aleksandr Kormiltsõn
MsC

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleksandr Babõkin 174960IDDR

Isikuandmete töötlemise komponendi disain ja rakendamine AfterPay organisatsioonidevahelistele protsessidele

Diplomitöö

Juhendaja: Aleksandr Kormiltsõn
MsC

Tallinn 2021

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aleksandr Babõkin

17.05.2021

Abstract

In cross-organisational business processes, partners have to respect each other's internal rules, such as data protection. Some organizations must supply a solution that is compliant with the regulations of other partners in order for the collaboration to be effective.

The main goal of the thesis is to design and implement a Personal Data Processing Component that eliminates the online merchant's obligations and risks related to sensitive data processing during the AfterPay API checkout process.

AfterPay as a system and the AfterPay Checkout process are introduced together, and the value of the Personal Data Processing Component for merchants is discussed. After that, an analysis of the requirements for the Personal Data Processing Component is presented.

Finally, the development process starts with specification gathering, by writing the behavior driven development scenarios. Then the author considers the appropriate architecture and design patterns which match the application's scope and requirements. The paper covers bothe the back-end and front-end implementation processes. The result is a working application.

The thesis is in English and contains 32 pages of text, 8 chapters, 7 figures.

Annotatsioon

Isikuandmete töötlemise komponendi disain ja rakendamine AfterPay organisatsioonidevahelistele protsessidele

AfterPay on teenus, mis võimaldab maksta tellimuse eest pärast kohaletoimetamist. Selleks, et leevendada riske teeb AfterPay ostjale maksevõime eelkontrolli, kasutades kliendipoolset isiku tuvastamise informatsiooni. Selleks, et veebipood sai enda klientidele AfterPay maksmisteenust pakkuda peaks kaupleja klienti isikuandmed käsitlema.

AfterPay puutus kokku probleemiga, et kauplejad ehk veebipoed keelavad AfterPay-d enda ostu vormistamise protsessi integreerida.

Lõputöö põhiline eesmärk on lahendada probleemi disainides ja implementeerides Isikuandmete töötlemise komponent, mis võimaldaks kauplejatel isikutuvastamise informatsiooni käsitlemisest mööda minema ja pakkudes samal ajal AfterPay teenust klientidele.

Esiteks autor seletab, mis on AfterPay ja kirjeldab selle protsessid. On näidatud, miks isiku tuvastamise informatsiooni on AfterPay-le vajalik ja selgitakse, kuidas AfterPay on teiste äridega seotud. Teiseks, muude maksuplatvormide kaasaegsed lähendused on analüüsitud ja selgitatud kuidas nad töötavad. Kolmandaks, alustab autor analüüsi nõuete kogumisest. Nende nõuete baasil kirjeldab autor tulevikusüsteemi, mis hakkab ostjate isikliku informatsiooni koguma. Neljandaks, kirjeldab autor tehnoloogia valikut. Lõpuks autor seletab lahti rakendamise protsessi, alustades BDD stsenaariumite kirjutamisega koos AfterPay esindajatega. Edasi autor kirjeldab rakenduse arhitektuurilised lahendused, implementatsiooni detailid ja süsteemi hinnangu kriteeriumid.

Lõputöö tulemuseks on valmislahendus, mis annab kauplejatele võimalus kasutada AfterPay makseteenuse ilma isiku tuvastamise informatsiooni puutumatu.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 32 leheküljel, 8 peatükki, 7 joonist.

List of abbreviations and terms

Customer/Shopper	A customer is an individual or business who purchases products or services from Merchant [1]
REST	Representational State Transfer
API	Application Programming Interface
Merchant	A merchant is a business that engages in e-commerce and has implemented or plans to use the AfterPay service [1]
GDPR	General Data Protection Regulation
CI/CD	Continuous integration/Continuous delivery or deployment
SSN	Social Security Number
IBAN	International Bank Account Number
SDK	Software Development Kit
HTML	HyperText Markup Language
Nonce	Number that can only be used once
UI	User Interface
CSS	Cascading Style Sheets
JWT	JSON Web Token
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
SQL	Structured Query Language
XML	Extensible Markup Language
YAML	Yet Another Markup Language
JSON	JavaScript Object Notation
BSON	Binary JavaScript Object Notation
DOM	Document Object Model
XHR	XMLHttpRequest
BDD	Behavior Driven Development

Table of Contents

List of Figures

1. Introduction

AfterPay[1] is a payment-after-delivery solution that allows customers to pay for their online purchases after receiving them [2]. The REST-based Payment API lets merchants integrate AfterPay into their shopping experience and at the same time forces them to process the customers' sensitive data, so the payment for a specific shopper and basket can be approved, including full fraud and credit scoring [3].

The fraud and risk evaluation can involve cross-organizational processes as the check will be executed outside AfterPay. This makes sensitive data handling crucial for AfterPay as a service provider.

While business processes that involve personal data are mostly cross-organizational legal requirements for personal data processing are becoming even stricter, for example with the introduction of the EU's General Data Protection Regulation[2]. The current AfterPay solution forces merchants to handle personally identifiable information within their systems.

The need for the solution described in this paper came from the fact that some merchants do not want to, or cannot, handle customers' personally identifiable information in a sufficiently secure and compliant way. As a result, merchants refuse to integrate AfterPay into their checkout experience.

The main goal of the thesis was to solve the problem by designing and implementing the Personal Data Processing Component that eliminates the obligations and risks related to sensitive data processing by online merchants.

The paper methodology is based on the internal analysis and research performed in AfterPay, including consultations with merchants and legal specialists. The solution is aligned with GDPR and the requirements of all the stakeholders.

[1] `https://developer.afterpay.io/`
[2] `https://gdpr.eu/`

1.1 Scope

The scope of this paper is limited to AfterPay's internal processes. The author considered the use of architecture and design patterns for achieving a highly decoupled application. The paper focuses on a specific application, and any other topics, such as cloud infrastructure, CI/CD pipelines and integration within the AfterPay API, are out of scope.

2. Background

This section introduces AfterPay as a system, the AfterPay Checkout process and the importance of the Personal Data Processing Component for merchants.

AfterPay is a pay-after-delivery solution developed by Arvato Financial Solutions. It allows omni-channel businesses to isolate checkout and payment, and covers the risk of consumers not paying, thus eliminating credit and fraud risk for merchants. An increasing number of customers make purchases on mobile devices, so it is important to make the checkout process in web shops as quick and easy as possible for on-the-go consumers, frequent shoppers or those who want to experience the goods and make a payment later [3].

In Fig. 1, [4] the AfterPay simplified process is introduced. The customer chooses 'post payment' on the merchant checkout page, and receives their purchase and invoice. After that, it is time to make a payment.

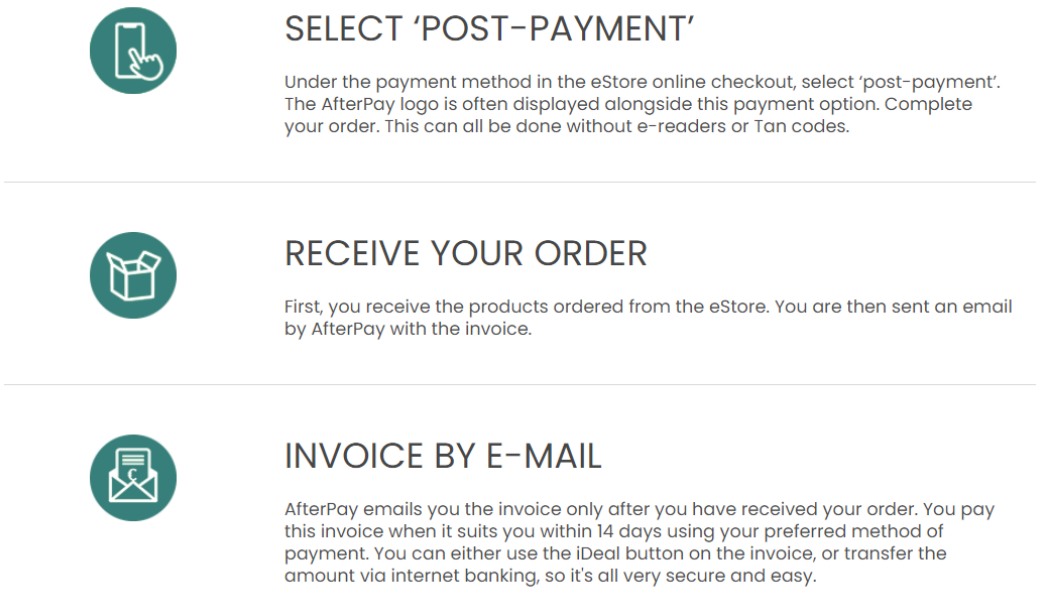

Figure 1. *Simplified overview of the pay-after-delivery process*

AfterPay offers an invoice that should be paid within 14 days as a default option. There is also a possibility to split payments or pause the payments for a certain period, or consolidate smaller purchases into one payment [3].

AfterPay is an international product operating in nine countries - Sweden, Norway, Finland, Germany, Switzerland, Austria, Denmark, the Netherlands and Belgium [3].

2.1 Overview of the AfterPay checkout process

After the shopping basket is filled in the web shop, the checkout process to complete the purchase can be initialized. The checkout process can be designed in various ways. Typically the following information blocks are relevant for payment:

1. Personal information about the shopper including the invoice/delivery address
2. Shipping options
3. Payment method selection
4. Order confirmation

Checkout begins with the identification step, where shoppers are required to enter their personal information, as well as the invoice and delivery address. AfterPay is trying to provide a quick and easy checkout experience, so it is important to minimise the amount of data given by the shopper. Therefore, several look-up services (as well as an bank account and address validation services) are provided by AfterPay. Having entered the required information, shoppers can choose the shipping provider before continuing to the payment method selection. AfterPay offers several different payment options that can be displayed at the payment selection step. After selecting a payment method, the shopper receives either a payment confirmation, or a message that the payment has been declined. A "declined" message redirects the shopper back to the payment method selection page. Once payment to the merchant is confirmed, it is important to give the consumer a confirmation including an overview of the shipping process as well as the payment method [3].

2.2 Customers' sensitive information and AfterPay

Shoppers' personal information is a crucial part of AfterPay's checkout flow. Only by knowing the shopper's identity can AfterPay eliminate the risk of fraud or non-payment. The identification process can happen several ways: the shopper can provide an SSN (Social Security NUmber) and authenticate in via some electronic identity application that will guarantee that the person who entered the SSN has a connection with it. Sometimes the shopper wants to pay with a debit card, so an IBAN check has to be executed at the beginning of the checkout. With the current setup, all personal identity checks have to happen on the merchant web page but at the same time, merchants do not need such checks for their internal business processes. As a result, some merchants cannot or will

not integrate AfterPay payment methods into their own checkout. One of the reason why merchants do not want to handle personal data is GDPR where merchants are referring to GDPR, whose Article 4 states: "'Personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person." [5].

Merchant may, therefore, refuse to integrate AfterPay payment methods into their own checkout instead requesting that AfterPay to handles all or some of the shopper's input in a separate page or element, hosted by AfterPay itself.

This solution could be either:

1. A separate page, and thereby an extra step in the checkout process, or
2. A page or area embedded inside the merchant's page.

2.3 Cross-organisational processes

Businesses are constantly looking for methods to cut costs, lower overheads, and grow their resources further. Collaboration with other businesses on a single project, or perhaps building a more sustainable long-term collaboration, is one way to make budgets and businesses go further [6].

For example, the e-commerce business process involves cooperation with payment gateways and shipping providers. Customers may buy things quickly and easily with the help of online payment channels [6]. Shipping is the physical transfer of goods from one location to another, such as the moving of merchandise from the warehouse to the customer [7]. Without such cooperation, an e-commerce business will not achieve success.

Every organisation has a different set of rules. At its most basic level, a business rule is a precise instruction that restricts or defines a business activity. These rules may be used to in almost every part of an organisation, including supply chain standards, data management, and customer relations. Business rules are meant to assist an organization in achieving its objectives and can be applied to computing systems [8].

AfterPay is a payment provider that works in partnership with e-commerce businesses and external providers of customer credit scores. For a successful collaboration, all parties

have to respect each other's internal rules, or provide a new solution that will respect the rules of the partner.

3. State of the Art

In this section, the state of the art is investigated. The thesis author investigate what solutions other payment providers have for merchants, who do not want to touch personally identifiable information by themselves. Also, the ways how those solutions are working is explained.

3.0.1 Visa

Visa Inc. is one of the most recognized global financial services brand, which provides all sort of APIs that meet customer specific needs [9, 10].

Visa Checkout is a digital payment service designed to simplify the checkout experience using a secure, single sign-on across channels and devices, as the customer's preferred payment method [11].

According to documentation, the Visa Checkout Widget loads over a web page when the customer selects Visa Checkout as a payment method on a checkout page. The customer is then prompted to sign in to their Visa Checkout account or create a new account before proceeding. After that, the customer reviews a payment information within the Visa Checkout Widget, make adjustments, if needed, and clicks the Continue or Pay button to send a confirmation back to the merchant. Then the customer is returned to the merchant's website so the order can be submitted. Finally, an Order Confirmation page is shown to the customer [11].

In order to integrate Visa Checkout, a JavaScript SDK has to be included at the end of a page body element. In the body of the page a Visa Checkout button has to be placed. The button itself is an HTML image tag with a specific class and source attribute to the image itself. After that, the Visa Checkout widget can be initialised by calling a JavaScript function with the granted API key, encryption key and paymentRequest which is filled with required properties. Also, event handlers should be added for possible payment events, such as success, cancel, and error. An HTML page example can be found in Appendix 2 [12].

As a result, by clicking the Visa Checkout button, a modal window is displayed on top of

the page where the button is located [13]. The modal window itself is an iframe that is initialized by Visa Checkout JavaScript SDK.

3.0.2 Braintree

Braintree is a payments platform that makes it easy to accept payments in mobile or web applications. From single-touch payments to mobile SDKs, Braintree provide everything needed to start accepting payments [14].

Braintree has client and server SDKs and merchant has to use both of them. Client SDK is needed to securely collect payment information from customers and the server SDK provides method to act on the collected payment information [15].

In Fig. 3, [15] diagram shows how merchant front-end client application, merchant back-end application and Braintree interact with each other. Front-end requests a client token from merchants back-end and initializes the client SDK. Back-end application generates and sends a client token back to front-end client using the server SDK. The customer submits payment information, the front-end SDK communicates that information to Braintree and gets a payment method nonce, which is an identifier. Front-end sends the nonce to merchant back-end server. Finally, the merchant back-end code receives the payment method nonce and then uses the server SDK to create a transaction [15].

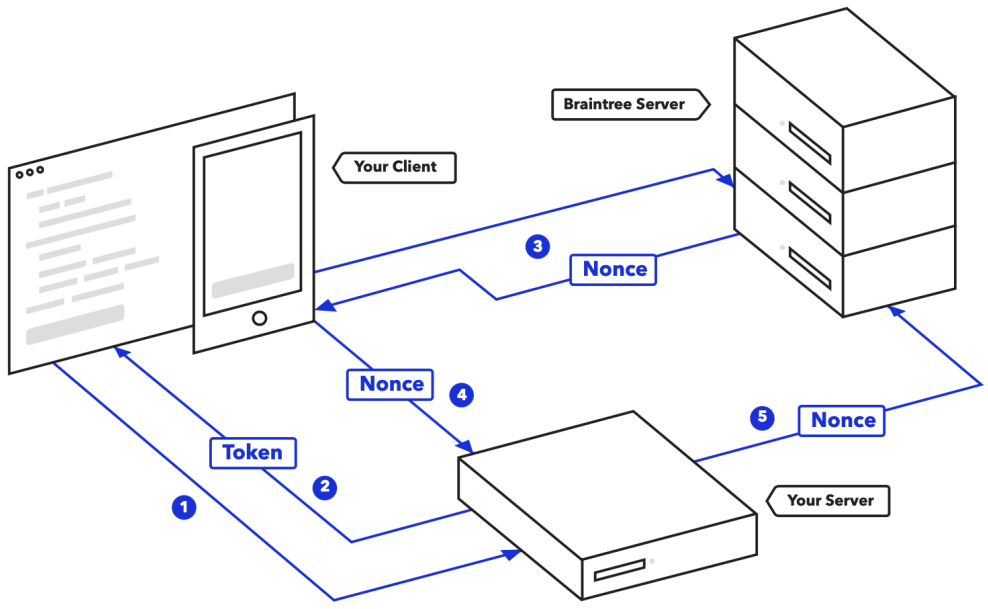

Figure 2. *Braintree interaction with merchant back and front-end applications.*

Braintree provides two UI components which will handle customer information with right

level of customization. The Drop-in UI is the Braintree ready-made payment solution which accept credit card payments securely. It has a pre-formatted payment form with no customization available. The Drop-in UI integration requires the latest Braintree JavaScript SDK and some manipulations in merchant checkout page code, basically a container for inputs, usually a HTML div tag, a submit button and a JavaScript which will initialize the inputs and a send the nonce to merchant server [16]. A code example can be found in Appendix 3 [17]. The Hosted Fields allows to collect personal information the same way as Drop-in UI, but provides control of the field style, so merchant existing styles and layout can be used and the experience of your checkout can be customised according to merchant needs. Hosted Fields basic integration requires the latest Braintree JavaScript SDK and a container for inputs. The custom CSS should be defined in initialization method. Also, the Hosted Field events can be defined, this allows update the UI based on state of the fields. [18, 19, 20, 21].

The correctly working integration will render credit card input fields. As Drop-in UI is ready made it might not match with merchants existing page style. Hosted Fields provide such possibility, so end will not see any difference between inputs which were rendered by Hosted Fields or by merchant. Both Drop-in UI and Hosted Fields are initialised as an iframe.

3.0.3 State of the art summary

Proposed solutions have different functionality for the end user, but actually have a common design with JavaScript SDK, some back-end for processing the data and some authorization service. All solutions have client SDKs which merchant has to integrate into the checkout page. The client SDKs render a page inside the iframe, the end user will see it as a page above the checkout page or it can be a part of the checkout page with different or even similar look. The rendering will happen only if correct authorization token is provided to initialization method in client SDK. Merchant has to add button which will trigger the sending of the personal data to a third party service which will return some identifier back to the merchant.

4. Analysis

After state of the art investigation, the analysis requirements for the Personal Data Processing Component were collected from merchants, their representatives, AfterPay analysts, AfterPay product owners, and the AfterPay software architect. After that, the solution design of the application is placed determined, and the appropriate technology is selected.

4.1 List of functional requirements

A functional requirement describes what a system or system component must do to provide its users with the required functionality [22].

- The system should be able to work in the merchant's checkout page.
- The system should provide a page or text-box area for personal data input.
- The system should have possibility to style the page or text-box area according to the merchant's needs.
- The system should save the customer's personal data and return an identifier back to the merchant.

4.2 List of non-functional requirements

A non-functional requirement expresses how well the software system must work when performing one or some functions [22].

- The merchant must not be able to access or edit the customers' personal data.
- Each identifier, which is associated with the customer's personal data, must be used only once.
- The system must work in all common browsers.
- The system must provide the customer's personal information only to a trusted resource.
- The system must be designed and developed according to the cloud-native techniques and technologies.

4.3 Solution design

After gathering of the requirements and state-of-art analyse the sequence diagram for Personal Data Processing Component is created. Fig. 3 is a simplified Personal Data Processing Component sequence diagram.

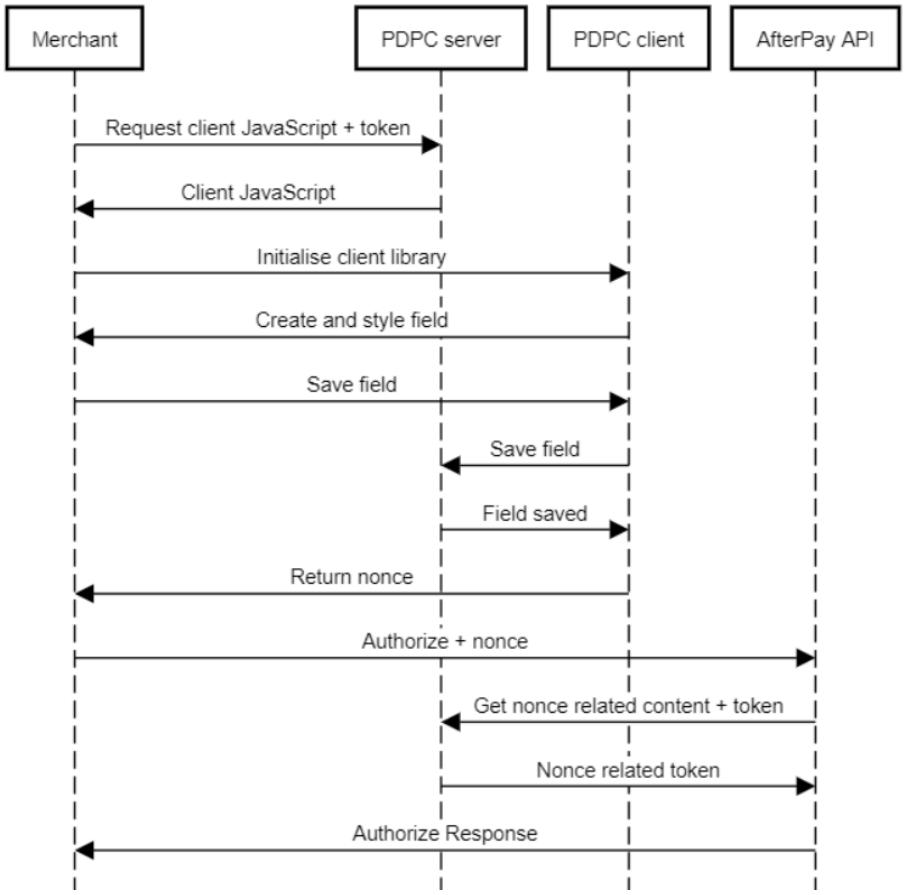

Figure 3. *Simplified Personal Data Processing Component sequence diagram.*

For authorization merchant will use already existing in-house build service which generates the JWT tokens. JSON Web Token is a JSON object which is used for securely transmitting information between parties. JWT token consist of three parts, the header, the payload and the signature, those are separated by dots. The header usually consists of two parts: the type of the token and the signing algorithm being used. The payload contains the claims, which are statements about an entity (typically, the issuer). These is a list of predefined claims which are highly recommended, but not mandatory. The custom claims can be used instead. To generate the signature, the encoded header and payload are taken, along with a secret, and signed with the algorithm specified in the header. The signature is used to

ensure that the token was not changed along the way [23]. The front-end SDK is needed for creating the input fields on a merchant page, it is done by injecting the iframe HTML tag into the merchant page. The iframe source has to be a page which is hosted on AfterPay side and all operation with customer sensitive data will be done inside that iframe. One of the operation is to save personal information in AfterPay database, so the data can be used in later stages. The front-end SDK is going to send a request with JWT token and personal data to the back-end application, which will save the personal data if the token is valid. As a result the nonce will be send back to front-end SDK and populated to some hidden container in a checkout page. The nonce is a unique identifier for customers personal data, that can only be used once. Merchant will use the nonce to complete the AfterPay checkout flow later on.

According to a non-functional requirement the system has to be designed and developed according to cloud-native techniques and technologies. Organizations can use cloud native technology to develop and run scalable applications in new, complex environments like public, private, and hybrid clouds. This approach is exemplified by containers, service meshes, microservices, immutable infrastructure, and declarative APIs. These techniques allow to develop resilient, manageable, and measurable loosely coupled systems. Software engineers can make high-impact improvements regularly and predictably with minimal effort [24].

The microservice architecture style is a method of building a single program as a collection of small services, each of which runs in its own process and communicates with lightweight mechanisms, most commonly an HTTP resource API [25]. Each microservice should have its own database which is suited for specific microservice needs [26, 27]. Personal Data Processing component can be defined as a microservice which is dealing with a concrete domain, in that case the domain is customers personal data.

5. Technology Stack

The technology stack is based on the requirements and solution design which were covered in previous sections. Technologies which are used in the AfterPay development organisation are also considered.

5.1 Back-end

The author of the thesis is working in a team of C#/.NET software engineers, who specialize on a REST APIs. Author decided to continue with .NET platform and develop a REST API as a back-end service. .NET is an open source developer platform, created by Microsoft, for building wide-range of applications. For building a REST Web API .NET provides a framework called ASP.NET Core. ASP.NET Core is a cross-platform framework which allows application to be executed inside Docker[1] container. At the time the application was developed (10.03.21) it was common to use .NET 5 version [28, 29, 30]. The application is written using C#, which is a modern, object-oriented, and type-safe programming language [31]. The IDE which are commonly used for developing .NET applications are JetBrains Rider, Visual Studio ja Visual Studio Code [32]. The paper author is working with Visual Studio Professional, which is provided by the authors employer. A system for developers to produce, exchange, and consume valuable code is a essential tool for any modern development environment. Such code is frequently bundled into "packages," which contain compiled code as well as other content required by the projects that consume these packages. NuGet is the Microsoft-supported system for exchanging code for .NET platform. It defines how .NET packages are generated, hosted, and consumed, as well as providing tools for each of those functions [33].

REST API is an application programming interface that respects the limits of REST architectural style and provide a standardized way for two applications to send data back and forth. The data will be communicated via the HTTP protocol [34, 35].

[1] `https://www.docker.com/`

5.2 Database

In this section the analyses of databases is based on two database categories: relational and non-relational. Non-relational databases subcategories are also considered.

5.2.1 Relational databases

Introduced by E.F. Codd in 1970, a relational database is a digital database organized based on the relational model of data. This model divides data into tables with rows and columns, with its own unique key for each row. In general, each database entity type has its own table, with rows representing instances of that entity type and columns representing values assigned to those instances. Since each row in a table has its own unique key, rows in one table may be connected to rows in other tables by storing the unique key of the row to which it should be linked, such unique key is also known as a “foreign key”. Microsoft SQL Server, SQL Server, MySQL, or PostgreSQL are some of the relational database examples. Common thing for those examples is that they have abbreviation SQL in their names. SQL (Structured Query Language) is a standard language for accessing and manipulating relational databases, to be precise SQL allows to insert, delete, and update the data itself or create, delete, or alter table in the database. SQL allows to join multiple tables and order or group the data according to the needs [36]. Relational databases face a significant obstacle in terms of scalability and elasticity. Relational databases were created at a time when data could be kept small, neat, and organized. To protect the consistency of the table mappings and escape the challenges of distributed computing, relational databases are built to operate on a single server. With this design, if a system needs to scale then bigger, more complex, and more expensive proprietary hardware with more processing power has to be acquired [37].

5.2.2 Non-relational databases

Non-relational database is a design that offers schema-less data storage and retrieval beyond relational databases’ conventional table structures. Non-relational databases can be called NoSQL, which stand for "Not only SQL" to emphasize that they do not support SQL-like query languages, although some NoSQL databases do support SQL-compatible queries. Despite the fact that NoSQL databases have been around for a long time, they have only recently gained popularity in the age of cloud, big data, and high-volume applications. They are chosen today for their characteristics of scale, performance and ease of use. NoSQL databases are designed for large scale on distributed platforms, rather than being limited by the limitations of single-server architectures. They will scale out “horizontally,” which means they can run on several servers, each sharing part of the load. A NoSQL

database can run on hundreds of servers and process tens of thousands of transactions per second using this approach. It can also do all of this on low-cost commodity hardware in any environment. Another advantage is that if one node fails, the workload can be picked up by the others, removing a single point of failure [37].

The most common types of NoSQL databases are [36, 38]:

- Key-value databases
- Document-oriented databases
- Columnar-oriented databases
- Graph databases

As the name implies, a key-value database is non-relational database that has a key and a value for each document. The key can be used to identify the value, much as in a dictionary. Developers choose key-value databases when the data they are dealing with is not complex and speed is a requirement. For example, key-value databases are ideal for storing configuration information [39].

A document-oriented data store maintains a collection of named string fields and object data values in an entity known as a "document", which is usually stored as JSON documents and can be encoded in a number of ways, including XML, YAML, JSON, BSON, or plain text. Document fields are exposed, allowing applications to query and sort data based on field values [40].

Data in graph databases is stored in nodes and edges. Nodes typically store information about entities while edges store information about the relationships between the nodes. This database style is especially useful for visualizing, analyzing, and helping in the discovery of relations between various pieces of data [41].

In contrast to relational databases, columnar databases store data in columns rather than rows. Columnar databases are built to read data faster and respond to queries more quickly. The ability to compress data is one of the key advantages of a columnar database. Columnar operations such as MIN, MAX, SUM, COUNT, and AVG may be done faster because to the compression. Another advantage is that, because column-based databases are self-indexing, they take up less disk space than a relational database management system containing the same data [42]. The disadvantage of columnar databases is that reading individual entries can be time consuming [43].

5.2.3 Personal Data Processing Component database

For the Personal Data Processing Component author decided to use a document-oriented database. Fig. 4 represents data structure which will be saved in the database. The structure itself is quite simple and do not require any additional tables or relations, that is why the relational and graph databases are not a good choice for the system. The columnar databases are bad in reading individual entries. The key-value non-relational database can be a good option, but it does not support complex queries which might be used for retrieving the data [44].

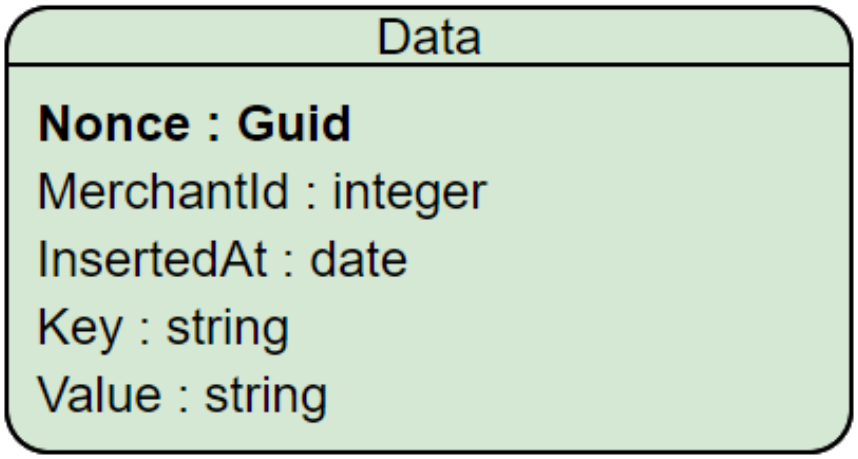

Figure 4. *Personal Data Processing Component data structure.*

Author decided to use MongoDB as a database, at the time the thesis was written (12.05.21) MongoDB is the most popular document-oriented database [45]. In addition, MongoDB provides official .NET driver, which allows asynchronous interacting with the database [46].

5.3 Front-end

HTML, CSS, and JavaScript work together to create a website's front-end design by applying information that changes the site's content, style, and interaction. A HyperText Markup Language (HTML) describes and defines the website's content and fundamental structure. This is accomplished through the use of specific tags that instruct the browser on what to do. HTML is the foundation of any website. A Cascading Style Sheet (CSS) is in charge of defining the colors, font, and placement of content on a website. It gives some style and shape to the content. To use the CSS capabilities, it must be linked within the HTML content so that the style may be applied to the website. CSS dictate the browser on how to render the current HTML. JavaScript is a lightweight interpreted programming

language that is used to change web pages and generate interactive functionality. A website will still work without JavaScript, but only to a limited extent [47]. Some of the JavaScript features are DOM and XHR API. DOM API allows to add, remove, and change HTML as well as apply new styles to web page. XHR API is used to communicate with the servers [48, 49]. One of the requirements for Personal Data Processing Component is that the system must work in popular browsers. According to caniuse.com 97.78% of browsers are supporting JavaScript with ECMAScript 5 specification [50]. JavaScript is an example of a loosely typed language. Loosely typed languages are ones in which type confusion can occur, resulting in errors that are difficult to find and detect, as opposed to strongly typed languages, in which such mistakes are identified either at compilation or during runtime. On the contrary, a loosely typed languages are usually way faster and offer a lot more flexibility [51].

An alternative language for coding front-end applications is Typescript. TypeScript is an open-source language that adds static type definitions to JavaScript. TypeScript can not be interpret by the browsers, so it has to be compiled to a JavaScript, so the code may be run on any browser or server equipped with JavaScript engine. The main advantage of the TypeScript is that compilation helps to catch bugs faster and earlier, especially when front-end application code base become larger [52].

For Personal Data Processing Component JavaScript was chosen as the front-end SDK programming language. Author of the paper had some experience with JavaScript previously and no time for learning TypeScript specific structures. TypeScript is a better choice for projects with large code base and when several developers are working on the project at the same time, but that is not the case for the current project [53, 54].

6. Development Process

Based on the problem, collected requirements, and analysis the author decided to start the development from describing the application features and behaviour driven development scenarios.

6.1 Behaviour driven development

BDD is an agile development approach which focuses on defining a specifications of the behaviour of the system, in a way that it can be automated. BDD encourages collaboration between all project participants, especially developers and business stakeholders, by using a pre-defined simple ubiquitous language, which is domain independent and used for structure user stories and scenarios [55].

The user story should have a story title which describes an activity that is done by a user in a given role and answer the questions [55]:

- What is the role of the user?
- What feature does the user want?
- What value can the user gain if the system provides the feature?

The user story template:
[StoryTitle]
As a **[Role]**
I want a **[Feature]**
So that I can get **[Benefit]**.

For one user story, there may be different versions in different contexts. The specific instances of a user story are called scenarios. Scenarios should describe specific contexts and outcomes of the user story, which should be provided by customers. Scenarios in BDD are used as acceptance criteria [55]. In AfterPay development organisation the BDD scenarios are written using the Gherkin syntax and have the unified format across all AfterPay development teams.

Each scenario has many layers of information that must be transparent, both individually and collectively. The scenario's title is the first layer. The title should make clear what to do with scenario right away. As a consequence, the title must contain an action that describes the original purpose or instruction of a scenario. Second level of details is a bunch of given-when-then statements. The given-when-then statements should give an overview of what has to accomplished. The final level of detail is a set of different scenarios that are linked by the same label that have to be implemented as one logical unit. Scenarios must be as short as possible, as additional words make understanding more difficult. The scenario should not be written for any one group (technical or business), but for all parties involved. If scenario is not clear to someone, then there is something wrong with it [56].

The scenario format:
@ (Label - semantic tag (marked as "@") to group scenarios into one logical unit)
Scenario: title which describes scenario purpose
Given some precondition
When an action occurs
And some other action occurs
Then some result happens

6.1.1 Handling customer data

This feature describes the process of how Personal Data Processing Component handles the customer identifiable information.

As a merchant
I want to integrate with AfterPay without touching user input
So that all the customer data is handled by AfterPay

Scenario: Personal information entering into a hosted field
Given Hosted field is visible on merchant page
When customer submits personal information
Then customer input is saved in a Hosted fields DB with nonce

@Authentication
Scenario: Authentication failed, when customer token is invalid
Given Hosted field is visible on merchant page
When customer submits personal information
And customer token is invalid
Then Hosted fields server returns status code 401 with a proper error message

@Authentication
Scenario: Authentication failed, when customer token is expired
Given Hosted field is visible on merchant page
When customer submits personal information
And customer token is expired
Then Hosted fields server returns status code 401 with a proper error message

6.1.2 Entering personal information

This feature describes initialization and styling of hosted fields.

As a customer
I want to see merchant page with hosted fields
So that I can enter personal details

@Initialization
Scenario: Hosted field initialization fails without authentication token
Given authentication token does not exist
When Hosted fields client initializes client library
Then error message is provided
And Hosted field is not visible
And hidden nonce field is not added to the form

@Initialization
Scenario: Hosted field initialization fails without a proper DOM element
Given authentication token exists
And Hosted field container does not exist on the page
When Hosted fields client initializes client library
Then error message is provided
And Hosted field is not visible
And hidden nonce field is not added to the form

@Initialization
Scenario: Client-side library with one field initialization
Given authentication token exists
And Hosted field container exists on the page
When Hosted fields client initializes client library
Then Hosted field is visible
And hidden nonce field is added to the form

@Initialization
Scenario: Iframe creation
Given authentication token exists
When Hosted fields client initializes client library
Then Hosted field container has an iframe

As a merchant
I want AfterPay to style and validate hosted fields
So hosted fields are styled and validated when "onchange" event is triggered

@Hosted-field-profile
Given merchant defined profile in config
And merchant defined "onChange" event in config
And page with hosted field rendered
When customer fulfills hosted field
Then status object returned to merchant

@Hosted-fields-profile
Scenario: Placeholder is visible in a hosted field, when merchant defined profile in config
Given merchant defined "IBAN-de-DE" profile in config
When page with hosted field renders
Then placeholder with German "IBAN" example visible in hosted field

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with German "IBAN" example visible in hosted field
When customer enters non German "IBAN"
Then "onChange" event triggered with status code "400.000"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with German "IBAN" example visible in hosted field
When customer enters incomplete German "IBAN"
Then "onChange" event triggered with status code "400.001"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with German "IBAN" example visible in hosted field
When customer enters German "IBAN" with typo

Then "onChange" event triggered with status code "400.002"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with German "IBAN" example visible in hosted field
When customer enters invalid German "IBAN"
Then "onChange" event triggered with status code "400.005"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with German "IBAN" example visible in hosted field
When customer enters valid German "IBAN"
Then "onChange" event triggered with status code "200"

Appendix 4 contains further scenarios.

6.2 Back-end architecture and design

In this section, the architecture and design of a system is considered. For software application, an Architecture Pattern represents a fundamental structural organization. It defines the roles of predefined subsystems and includes rules and guidelines for organizing the relationships between them. A Design Pattern is a method for optimizing a software system's subsystems or components, as well as the relationships between them. It's a term that refers to a recurrent arrangement of interacting components that solves a general design problem in a specific situation [57].

The author decided to use the Clean Architecture approach together with mediator pattern for designing the system.

6.2.1 The clean architecture

One way to handle the scope of an application as it grows in complexity is to split it up into layers based on its responsibilities or concerns. This methodology adheres to the division of concerns principles which can help in the organization of an expanding code base so that developers can quickly locate where specific software is applied [58].

The back-end architecture pattern is based on the Clean Architecture principles. The clean architecture was proposed by Robert C. Martin in 2011 [59] .

The Clean Architecture allows to build software with very low coupling and independence from technical implementation like databases and frameworks. As a result, the program becomes simple to manage and modify. The decoupled application becomes intrinsically testable as well [60, 61].

In Fig. 5 [62] diagram demonstrates back-end architecture for the Personal Data Processing Component. Enterprise logic and types are located in the Domain layer, while business logic and types are found in the Application layer. Enterprise logic, on the other hand, can be shared across several systems, whereas business logic is commonly used only within the system. The Domain and Application layers are at the heart of Clean Architecture's nature, it is also common to name those as Core. Core should not be reliant on data access or other infrastructure concerns such as external providers, because those dependencies are inverted. Presentation layer is responsible for showing data to the end-user, it can be a REST API, Blazor WebAssembly application or a desktop application. All dependencies flow inwards, and Core has no external dependencies. Infrastructure and Presentation are both dependent on Core, but not on one another [62].

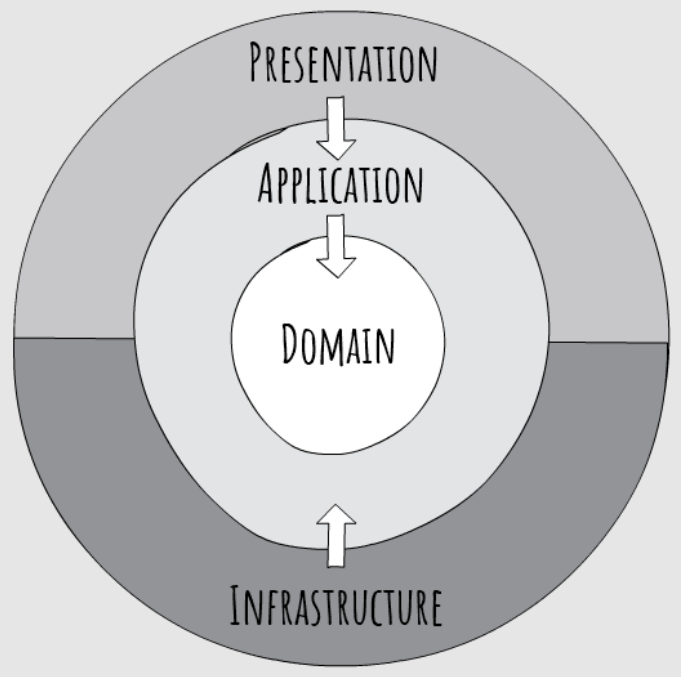

Figure 5. *Clean Architecture Diagram.*

Usually, it is common to add business logic to a Presentation level, but that will break a Clean Architecture principle, by adding a redundant dependency. It is important to ensure that business logic is independent of the Presentation. A good approach to solve this problem is to use the mediator pattern.

6.2.2 The mediator pattern

The distribution of actions among objects is encouraged in object-oriented design. As a consequence of this distribution, an entity system with several relations between objects will emerge; in the worst-case scenario, each object becomes aware of the others. These issues can be avoided by encapsulating collective behavior in a separate mediator object. Controlling and organizing the activities of a group of objects is the job of a mediator [63].

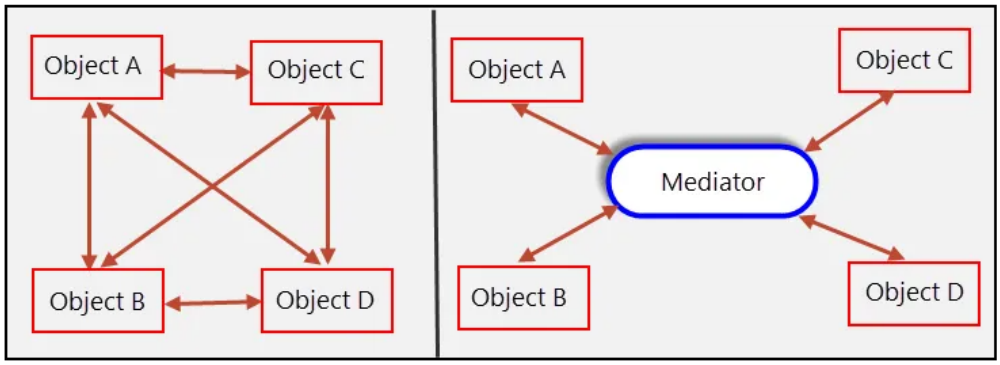

Figure 6. *A set of objects which interact directly and a set of objects which interact through mediator.*

Fig. 6 [64] shows that mediator acts as a intermediary, preventing objects in the group from referring to each other explicitly. Since the objects just recognize the mediator, the number of interconnections is reduced [63].

For mediator pattern author decided to use an already existing, open sourced MediatR[1] library which is a a simple and unambitious mediator implementation on .NET platform. The library can be added to a project as a NuGet package.

6.3 Back-end development

The back-end development started from creating a solution project in Visual Studio. The solution has three layers. First of all, author added the ASP.NET Core Web API project to the solution, it is responsible for Presentation layer. Secondly, a Class library for Infrastructure layer has been added. Finally, another Class library has been added, this time for Core layer which contains business logic.

The entity has to be created that describes the object which is going to be stored in the database, it can be placed in Core layer inside Domain folder. The entity itself has dictionary as a public property, both dictionary key and value are strings. The dictionary

[1] `https://github.com/jbogard/MediatR`

key will represent the value, for example, the key can be IBAN or SSN.

The author added an interface for the repository inside Core layer Interface folder. Currently, the interface having only one method for saving the entity in the database. The interface will be implemented inside Infrastructure layer, so the Core layer will not be dependent on Infrastructure.

Both Core and Presentation layer use MediatR for process the requests. Fig. 7 shows that MediatR is smart enough to redirect to the right request handler depending on the request type.

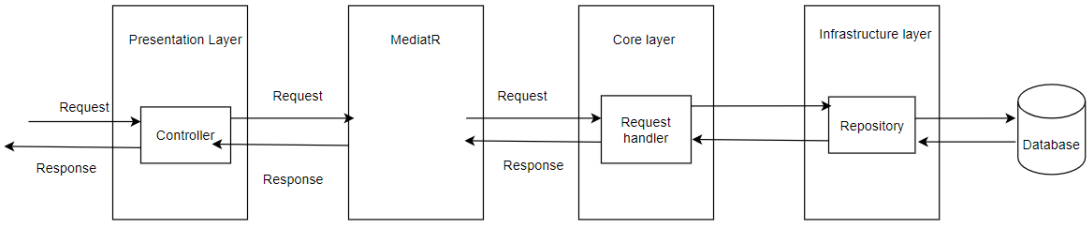

Figure 7. *Request processing using MediatR package.*

6.3.1 Infrastructure layer

The Infrastructure layer which enables a software system to interact with external systems by receiving, storing and providing data when requested. Previously, author has chosen the MongoDb as a database for Personal Data Processing Component. In order to connect database to the application MongoDb provides a C#/.NET driver. The driver itself is supported by MongoDb and can be downloaded as a NuGet package [65, 66].

The Infrastructure layer is using the repository pattern which is a strategy for abstracting data accessing. To be precise, data access is made up of the code in an application that deals with storing and retrieving data [67]. The solution will have only one entity to manipulate, so one interface was added to the Application layer and implemented in the Infrastructure layer.

For data manipulation application has to connect to a database itself. It is done using he MongoDb driver. The connection begins with a MongoClient object which will be the root object. It is thread-safe and is all that is needed to handle connection to server. The database connection string should be provided to a MongoClient as a parameter. A database in a MongoDB server is represented by an IMongoDatabase interface. The GetDatabase

method is used to retrieve databases from an IMongoClient instance. The GetDatabase method should get the name of the database as a paramether. If the database does not already exist on the server, it will be generated upon first use. Last step is to access the collection itself. The GetCollection<TDocument> method is used to extract collections from an IMongoDatabase. The collection name has to be to a method as a parameter. The type of document that is stored in collection can be a generic TDocument or a custom class [68].

6.3.2 Core layer

The author decided to merge Domain and Application layers into one Core layer, because lack of enterprise logic. The Core layer was implemented using the mediator design patters which was described previously. The Core Layer contains all business logic which is describe in BDD scenarios.

As was mentioned previously, for implementing the mediator patter author uses the MediatR library which can be downloaded as a NuGet package. The MediatR forces developer to have a request and a handler per feature. First of all, the request has to be defined. The the handler, which contain business logic to solve the request, has to be created. Request is a class that implements the MediatR.IRequest<T> interface, where T is a return type. Handler is a class that implements the MediatR.IRequestHandler<T,U> interface, where T is a request type and U is a response type. The handler dependents on a repository interface, so it has to be initialized from the constructor.

6.3.3 Presentation layer

The Presentation layer is dependent on MediatR package, because it has to send requests to mediator. Another required package is needed for dependency injection. It is called MediatR.Extensions.Microsoft.DependencyInjection and it is useful for applications using the default Microsoft Dependency Injection libraries.

The presentation layer takes care of incoming HTTP requests and responds to the caller. In ASP.NET framework it can be done using the controller. The controller is going to communicate with the business logic via mediator. The mediator has to be initialized from the constructor. The controllers method will usually have only couple lines of code, basically it will only send a request to a mediator using IMediator.Send() method, which takes request as a parameter.

For presentation layer the controller has to be created, which takes care of incoming HTTP

requests and responds to the caller. The controller is dependent only on a mediator,

6.4 Front-end SDK development

In this section the Front-End SDK development is considered. SDK is a collection of software resources and programs offered by software vendors that enable third-party developers to create applications for particular platforms. The front-end SKD is implemented using the JavaScript.

The SDK has following structure:

- main.js - is a hearth of the SDK, containing functions which end client suppose to use.
- field.js - have specific functionality for the field itself, such as field value validation.
- field.html - the HTML file which has HTML input and script tag. Script tag has a field.js as a source.
- manager.js - has functionality for sending the data to Personal Data Processing Component back-end service.
- manager.html - the HTML file which has script HTML tag with source to manager.js file.

Both field and manager HTML pages have to be injected to the merchant checkout page as an iframe sources. The communication between iframes is done by the MessageEvent interface, using the window.postMessage() method. Every JavaScript file has an event listener with a 'message' type. Messages will be consumed only if the event origin is whitelisted [69].

The SDK consists of four functions:

1. initialize function
2. onError function
3. onChange function
4. save function

6.4.1 Initialize function

The initialize function takes configuration object as a parameter. The configuration object is a set of required and optional properties. The mandatory properties are authorization token and fields array. Array can contain multiple field objects, each field should have

required properties, like a field name and a container id. Field name represents what the field suppose to contain, for example SSN or IBAN. Container id is a CSS selector which marks HTML container where input field will be created. HTML container is usually a div tag section.

The main purpose of initialization function is to add an event listener with type 'message' and a function that will be called when the required event is transmitted to the destination, check the configuration required properties and inject two iframes within the merchants page. The iframe creation is done by using the HTML DOM createElement() method with paramether 'iframe'. Then the iframe source is defined, it should be a HTML page which is hosted on the AfterPay side. First iframe will have a field.html page as a source, second iframe source will be manager.html page. Second iframe is unseen to end users. appendChild() method is used to inject iframes to merchant page. The field iframe will be appended in the end of container which is marked with CSS id selector. The manager iframe will be appeded in the end of the document body.

6.4.2 OnError function

OnError is a callback function, which exposes the errors. It is important to set up error handling before fields initialization, as it can cause some issues.

Possible errors:

- ERROR - generic error, the concrete cause has to be investigated in browsers console.
- NO_FIELDS - There are no fields configured, the fields property in the configuration is missing.
- NO_TOKEN - In the configuration, no authorization token was passed.
- SAVE_ERROR - The specific cause of an error storing the field must be explored in the browser console.
- NO_CONTAINER - The container element where the field must be generated could not be found.

6.4.3 OnChange function

OnChange is a callback function which returns field validation status. The validation will happen only if each field profile is defined in configuration object. The profile itself is a string, for example 'IBAN-de-DE' or 'SSN-sv-FI'. Each profile has a default field placeholder, which is added to the field automatically. The profile is devided on three parts. First part is a field key, second part is a language of the customer, third part is

the country. The Personal Data Processing Component supports Finnish, Swedish and Norwegian SSNs and all European IBANs. All supported SSNs have different formats and, as a result, different validation logic. IBAN is an international system, which is used to identify bank accounts, so the validation is unified.

In field.js file the event listener with type 'input' is added to a field in field.html. That will trigger the validation every time the value in the field is changing. The onChange method returns validation status object with the exact field name which validation is failed, validation overall status and a message with detailed explanation what went wrong.

Error messages:

- Input invalid - default error message
- Too short - the value of the input is shorter than it should be
- Too long - the value of the input is longer than it should be
- Too young - customer age is less than 18 years
- Checksum incorrect - field validation failed

6.4.4 Save function

Save method is a callback function which sends data from input fields to back-end application and returns the nonce. The saving process starts from getting the field value from input field and saving it in the browser local storage. Saving data in browser local storage will prevent personally identifiable information moving from one iframe to another. localStorage.setItem(key, value) method is used for saving the data in the browser local storage. The data is saved as a key-value pair, where key is a field name and value is a field value. When browsers local storage is filled, then sending data to the back-end application can be initiated in manager.js file. First of all, the data itself has to be retrieved from local storage by the key. Then, the request can be sent via XMLHttpRequest.send() method. The request is initialized with XMLHttpRequest.open() method, which first parameter is the HTTP request method, 'POST' to be exact and second parameter is url where to send the request. After that, some request headers are added using the XMLHttpRequest.setRequestHeader() method, which takes the name of the header whose value is to be set and the value to set as the body of the header as a parameters. Then, a request is sent via the XMLHttpRequest.send() method, which takes request as a parameter. When successful response is received, the nonce is returned to a merchant page.

6.5 Development evaluation

The evaluation of the development is based on multiple criteria, such as:

1. BDD scenarios
2. Unit tests
3. Acceptance testing
4. Feedback from the merchants

The first criteria is quality of written BDD scenarios, which basically means, how understandable those scenarios are for the domain experts, such as AfterPay analytics and product owners together with business stakeholders and software engineers. The feedback about BDD scanarios was positive. The second criteria is unit tests which are a sort of software testing that examines individual software units or components [70]. The third criteria is system acceptance testing. Acceptance criteria outline intended behavior and are used to assess if a feature of feature component has been developed successfully [71]. The acceptance testing is done using the UI tests, which are based on BDD scenarios. UI testiing is a mechanism for testing the parts of any software that a user will interact with. This generally means putting the visual aspects to the test to see if they work as expected [72]. If the tests are written correctly, then it ensures that the system behaves as it was originally intended. Sometimes, the BDD scenario itself can be wrong, then it has to be corrected. Overall, the system passed acceptance criteria and was presented to merchants who did not want to touch their customers' personally identifiable information. Early on, several merchants began integrating the system into their checkouts, and as a consequence, they were able to identify and report several problems.

7. Results

The result of the thesis is the Personal Data Processing Component, which allows merchants to bypass the handling of personally identifiable information handling on their side and still provide AfterPay payment provider for their customers. Initial requirements for the system were achieved, but some future development might be required.

During the development process the thesis author learned how to write BDD scenarios, which are understandable to the whole team of developers and business stakeholders. In addition, the author improved knowledge about the clean architecture approach and what patterns can be used to achieve it. Among other things, the author gained confidence in developing front-end applications using JavaScript.

7.1 Future developments

The solution was introduced to AfterPay clients and first feedback about potential new requirements and improvements was gathered. The Personal Data Processing Component is lacking field style customization. Clients wanted to have possibility to customize the fields so their checkout page style is consistent. Another comment from a client was that the solution does not work in Chrome[1] Incognito mode, the browser's local storage, which is used in front-end SDK, can be a potential cause of that issue. The front-end codebase is bigger than the author initially thought, and sometimes it is hard to understand the code, porting JavaScript code to TypeScript can improve its overall readability. Covering the system with automation tests using the BDD scenarios is another future improvement. The automation tests suite will guarantee that the system works as it was described in the scenarios.

[1] `https://www.google.com/intl/et/chrome/`

8. Summary

The thesis' major purpose was to reduce the online merchant's obligations and risks associated with sensitive data processing throughout the AfterPay API checkout process. As result of this thesis, the Personal Data Processing Component was designed and developed. The system consists of a REST API and front-end SDK. The front-end SDK handles the personally identifiable information on the merchant's checkout page and communicates it to the REST API, which saves the data to a database. The Personal Data Processing Component is already in use by the merchants in production.

The thesis is divided into five sections: background, state of the art, analysis, technology stack and implementation process. An overview of AfterPay and the importance of personally identifiable information to its operation are covered in the thesis' background section. In addition, cross-organisational processes are described.

In the analysis section, system requirements are presented together with the state-of-art investigation. Based on this analysis, a solution design is proposed.

In the implementation section the development process of the Personal Data Processing Component is covered. First of all, the explanation why BDD scenarios are valuable is presented. Then the architecture and design patterns are chosen. Lastly, the implementation of the component is described.

Bibliography

[1] AfterPay. *Terminology*. [Online]. [Accessed: 15-03-2021]. URL: https://developer.afterpay.io/terminology.

[2] AfterPay. *AfterPay Basics*. [Online]. [Accessed: 07-03-2021]. URL: https://developer.afterpay.io/basics.

[3] AfterPay. *AfterPay Technical White Papep Version 2.6*. [Online]. [Accessed: 07-03-2021]. URL: https://documents.afterpay.io/guidelines/Technical_White_Paper.pdf.

[4] AfterPay. *AfterPay How does it work*. [Online]. [Accessed: 26-03-2021]. URL: https://www.afterpay.nl/en/customers/how-does-it-work.

[5] GDPR. *What is considered personal data under the EU GDPR?* [Online]. [Accessed: 26-03-2021]. URL: https://gdpr.eu/eu-gdpr-personal-data/#:~:text='Personal%20data'%20means%20any%20information,location%20data%2C%20an%20online%20identifier.

[6] Patrick Hosch. *Is Cross-Organizational Collaboration Key to your Org's Future?* [Online]. [Accessed: 15-05-2021]. URL: https://www.nintex.com/blog/cross-organizational-collaboration-key-businesss-future/.

[7] Ecommerce Platforms. *What is Shipping? What does shipping mean?* [Online]. [Accessed: 15-05-2021]. URL: https://ecommerce-platforms.com/glossary/shipping.

[8] Techopedia. *Business Rule*. [Online]. [Accessed: 15-05-2021]. URL: https://www.techopedia.com/definition/28018/business-rule.

[9] Visa Inc. *About Visa*. [Online]. [Accessed: 18-03-2021]. URL: https://ht.visa.com/about-visa.html.

[10] Visa Inc. *Merchant APIs*. [Online]. [Accessed: 18-03-2021]. URL: https://developer.visa.com/apibrowser/#segment=Merchants.

[11] Visa Inc. *Visa Checkout*. [Online]. [Accessed: 18-03-2021]. URL: https://developer.visa.com/capabilities/visa_checkout.

[12] Visa Inc. *How to Use Visa Checkout*. [Online]. [Accessed: 19-03-2021]. URL: https://developer.visa.com/capabilities/visa_checkout/docs-how-to.

[13] w3schools. *How TO - CSS/JS Modal*. [Online]. [Accessed: 26-03-2021]. URL: https://www.w3schools.com/howto/howto_css_modals.asp.

[14] Braintree. *Braintree FAQ*. [Online]. [Accessed: 21-03-2021]. URL: https://www.braintreepayments.com/ee/faq.

[15] Braintree. *Braintree Get Started Overview*. [Online]. [Accessed: 21-03-2021]. URL: https://developers.braintreepayments.com/start/overview.

[16] Braintree. *Braintree Drop-in Setup*. [Online]. [Accessed: 25-03-2021]. URL: https://developers.braintreepayments.com/guides/drop-in/setup-and-integration/javascript/v3.

[17] Braintree. *Braintree Drop-in Tutorial*. [Online]. [Accessed: 21-03-2021]. URL: https://developers.braintreepayments.com/start/tutorial-drop-in-node.

[18] Braintree. *Braintree Hosted Fields Overview*. [Online]. [Accessed: 25-03-2021]. URL: https://developers.braintreepayments.com/guides/hosted-fields/overview/javascript/v3.

[19] Braintree. *Braintree Hosted Fields Setup and Integration*. [Online]. [Accessed: 25-03-2021]. URL: https://developers.braintreepayments.com/guides/hosted-fields/setup-and-integration/javascript/v3.

[20] Braintree. *Braintree Hosted Fields Styling*. [Online]. [Accessed: 25-03-2021]. URL: https://developers.braintreepayments.com/guides/hosted-fields/styling/javascript/v3.

[21] Braintree. *Braintree Hosted Fields Events*. [Online]. [Accessed: 25-03-2021]. URL: https://developers.braintreepayments.com/guides/hosted-fields/events/javascript/v3.

[22] Mohammad Dabbagh, Reza M. Parizi, and Sai Peck Lee. *Functional and non-functional requirements prioritization: empirical evaluation of IPA, AHP-based, and HAM-based approaches*. [Online]. [Accessed: 11-03-2021]. 2015. URL: https://www.researchgate.net/publication/280089647_Functional_and_non-functional_requirements_prioritization_empirical_evaluation_of_IPA_AHP-based_and_HAM-based_approaches.

[23] jwt.io. *Introduction to JSON Web Tokens*. [Online]. [Accessed: 08-05-2021]. URL: https://jwt.io/introduction/.

[24] GitHub(CNCF). *CNCF Cloud Native Definition v1.0*. [Online]. [Accessed: 11-05-2021]. URL: https://github.com/cncf/toc/blob/main/DEFINITION.md.

[25] Martin Fowler. *Microservices*. [Online]. [Accessed: 11-05-2021]. URL: https://martinfowler.com/articles/microservices.html.

[26] Samir Behara. *Breaking the Monolithic Database in Your Microservices Architecture*. [Online]. [Accessed: 11-05-2021]. URL: https://dzone.com/articles/breaking-the-monolithic-database-in-your-microserv.

[27] Chris Richardson. *Pattern: Database per service*. [Online]. [Accessed: 11-05-2021]. URL: https://microservices.io/patterns/data/database-per-service.html.

[28] Microsoft Corporation. *When to choose .NET for Docker containers*. [Online]. [Accessed: 25-04-2021]. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/net-core-net-framework-containers/net-core-container-scenarios.

[29] Microsoft Corporation. *What is .NET?* [Online]. [Accessed: 10-05-2021]. URL: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet.

[30] Microsoft Corporation. *What is ASP.NET?* [Online]. [Accessed: 10-05-2021]. URL: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet.

[31] Microsoft Corporation. *A tour of the C# language*. [Online]. [Accessed: 10-05-2021]. URL: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

[32] slant.co. *What are the best C# IDEs?* [Online]. [Accessed: 15-05-2021]. URL: https://www.slant.co/topics/4118/~c-ides.

[33] Microsoft Corporation. *An introduction to NuGet*. [Online]. [Accessed: 15-05-2021]. URL: https://docs.microsoft.com/en-us/nuget/what-is-nuget.

[34] Inc. Red Hat. *What is a REST API?* [Online]. [Accessed: 15-05-2021]. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[35] Jamie Juviler. *REST APIs: How They Work and What You Need to Know*. [Online]. [Accessed: 15-05-2021]. URL: https://blog.hubspot.com/website/what-is-rest-api.

[36] James Serra. *Relational databases vs Non-relational databases*. [Online]. [Accessed: 11-05-2021]. URL: https://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/.

[37] Matt Allen. *Relational Databases Are Not Designed For Scale*. [Online]. [Accessed: 15-05-2021]. URL: https://www.marklogic.com/blog/relational-databases-scale/.

[38] IBM Corp. *NoSQL Databases*. [Online]. [Accessed: 25-04-2021]. URL: https://www.ibm.com/cloud/learn/nosql-databases.

[39] Laura M. *Different Types of Databases: What Should You Know?* [Online]. [Accessed: 11-05-2021]. URL: https://www.bitdegree.org/tutorials/types-of-databases/.

[40] Tamara Pattinson. *RELATIONAL VS NON-RELATIONAL DATABASES*. [Online]. [Accessed: 11-05-2021]. URL: https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases.

[41] Lauren Schaefer. *What is NoSQL?* [Online]. [Accessed: 11-05-2021]. URL: https://www.mongodb.com/nosql-explained.

[42] TechTarget. *Columnar database*. [Online]. [Accessed: 15-05-2021]. URL: https://searchdatamanagement.techtarget.com/definition/columnar-database.

[43] Mark Drake. *A Comparison of NoSQL Database Management Systems and Models*. [Online]. [Accessed: 15-05-2021]. URL: https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models.

[44] Informit. *NoSQL Key-Value Database Simplicity vs. Document Database Flexibility*. [Online]. [Accessed: 15-05-2021]. URL: https://www.informit.com/articles/article.aspx?p=2429466#:~:text=Document%5C%20databases%5C%20organize%5C%20documents%5C%20into,analogous%5C%20to%5C%20a%5C%20relational%5C%20schema.

[45] db-engines.com. *DB-Engines Ranking*. [Online]. [Accessed: 12-05-2021]. URL: https://db-engines.com/en/ranking.

[46] MongoDB Inc. *MongoDB C#/.NET Driver*. [Online]. [Accessed: 15-05-2021]. URL: https://docs.mongodb.com/drivers/csharp/.

[47] Juan van Niekerk. *How Do HTML, CSS and JavaScript Work Together?* [Online]. [Accessed: 15-05-2021]. URL: https://www.itonlinelearning.com/blog/how-do-html-css-and-javascript-work-together/.

[48] MDN Web Docs. *What is JavaScript?* [Online]. [Accessed: 15-05-2021]. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.

[49] MDN Web Docs. *XMLHttpRequest*. [Online]. [Accessed: 15-05-2021]. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest.

[50] caniuse.com. *Can I use Search EcmaScript 5?* [Online]. [Accessed: 15-05-2021]. URL: https://caniuse.com/?search=ecmascript%5C%205.

[51] Gary Cordero Rosa. *Loosely Typed and Strongly Type Languages*. [Online]. [Accessed: 15-05-2021]. URL: https://garycordero1690.medium.com/loosely-typed-and-strongly-type-languages-550ce60b2739.

[52] typescriptlang.org. [Online]. [Accessed: 15-05-2021]. URL: https://www.typescriptlang.org/.

[53] altexsoft.com. *The Good and the Bad of TypeScript*. [Online]. [Accessed: 15-05-2021]. URL: https://www.altexsoft.com/blog/typescript-pros-and-cons/.

[54] Sasha Andrieiev. *What Are The Differences, Advantages, and Disadvantages, between TypeScript and JavaScript?* [Online]. [Accessed: 15-05-2021]. URL: https://javascript.plainenglish.io/what-are-the-differences-advantages-and-disadvantages-between-typescript-and-javascript-492fb3764870.

[55] Carlos Solis and Xiaofeng Wang. *A Study of the Characteristics of Behaviour Driven Development*. [Online]. [Accessed: 12-03-2021]. 2011. URL: https://www.researchgate.net/publication/224265781_A_Study_of_the_Characteristics_of_Behaviour_Driven_Development.

[56] D. North. *Introducing BDD*. [Online]. [Accessed: 12-03-2021]. 2006. URL: https://dannorth.net/introducing-bdd/.

[57] The Open Group. *Architecture Patterns*. [Online]. [Accessed: 12-04-2021]. URL: https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap28.html.

[58] Microsoft Corporation. *Common web application architectures*. [Online]. [Accessed: 12-04-2021]. URL: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures.

[59] Robert C. Martin. *Clean Architecture*. [Online]. [Accessed: 5-04-2021]. URL: http://blog.cleancoder.com/uncle-bob/2011/11/22/Clean-Architecture.html.

[60] Carlos Schults. *An Introduction to Clean Architecture*. [Online]. [Accessed: 5-04-2021]. URL: https://blog.ndepend.com/introduction-clean-architecture/.

[61] Robert C. Martin. *Clean Architecture*. [Online]. [Accessed: 12-04-2021]. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

[62] Jason Taylor. *Clean Architecture with .NET Core: Getting Started*. [Online]. [Accessed: 5-04-2021]. URL: https://jasontaylor.dev/clean-architecture-getting-started/.

[63] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [Online]. [Accessed: 20-04-2021]. URL: https://archive.org/details/designpatternsel00gamm/.

[64] John Thompson. *The Mediator Pattern: Deep Dive*. [Online]. [Accessed: 11-05-2021]. URL: https://dzone.com/articles/mediator-pattern-1.

[65] Janalta Interactive. *Infrastructure Layer*. [Online]. [Accessed: 20-04-2021]. URL: https://www.techopedia.com/definition/32090/infrastructure-layer#:~:text=The%20infrastructure%20layer%20enables%20a,to%20connect%20with%20other%20systems.

[66] MongoDB Inc. *Start Developing with MongoDB*. [Online]. [Accessed: 22-04-2021]. URL: https://docs.mongodb.com/drivers/.

[67] Joe Petrakovich. *The WHY Series: Why should you use the repository pattern?* [Online]. [Accessed: 22-04-2021]. URL: https://makingloops.com/why-should-you-use-the-repository-pattern/#:~:text=The%20repository%20pattern%20is%20a,list%20items%20in%20a%20table..

[68] MongoDB Inc. *Connection String*. [Online]. [Accessed: 11-05-2021]. URL: https://mongodb.github.io/mongo-csharp-driver/2.12/reference/driver/connecting/.

[69] MDN Web Docs. *MessageEvent*. [Online]. [Accessed: 15-05-2021]. URL: https://developer.mozilla.org/en-US/docs/Web/API/MessageEvent.

[70] LLC Innolution. *Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE*. [Online]. [Accessed: 15-05-2021]. URL: https://www.guru99.com/unit-testing-guide.html.

[71] LLC Innolution. *Acceptance criteria*. [Online]. [Accessed: 15-05-2021]. URL: https://innolution.com/resources/glossary/acceptance-criteria.

[72] Shreya Bose. *UI Testing: A Detailed Guide*. [Online]. [Accessed: 15-05-2021]. URL: https://www.browserstack.com/guide/ui-testing-guide.

Appendix 1 - Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Aleksandr Babõkin

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Design and Implementation of the Personal DataProcessing Component for the AfterPayCross-Organisational Processes" , supervised by Aleksandr Kormiltsõn
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

17.05.2021

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Visa Checkout code example

```
<html>
<head>
 <script type="text/javascript">
  function onVisaCheckoutReady(){
  V.init( {
  apikey: "...",
  encryptionKey: "...",
  paymentRequest:{
  currencyCode: "USD",
    subtotal: "11.00"
  }
  });
V.on("payment.success", function(payment)
  {alert(JSON.stringify(payment)); });
V.on("payment.cancel", function(payment)
  {alert(JSON.stringify(payment)); });
V.on("payment.error", function(payment, error)
  {alert(JSON.stringify(error)); });
}
</script>
</head>

<body>
<img alt="Visa Checkout" class="v-button" role="button"
src="https://sandbox.secure.checkout.visa.com/wallet-services-web/
 ↪  xo/button.png"/>
<script type="text/javascript"
src="https://sandbox-assets.secure.checkout.visa.com/
checkout-widget/resources/js/integration/v1/sdk.js">
</script>
</body>
</html>
```

Appendix 3 - Braintree Drop-in UI code example

```
<div id="dropin-wrapper">
  <div id="checkout-message"></div>
  <div id="dropin-container"></div>
  <button id="submit-button">Submit payment</button>
</div>
<script>
  var button = document.querySelector('#submit-button');

  braintree.dropin.create({
    // Insert your tokenization key here
    authorization: '<use_your_tokenization_key>',
    container: '#dropin-container'
  }, function (createErr, instance) {
    button.addEventListener('click', function () {
      instance.requestPaymentMethod(function
       ↪  (requestPaymentMethodErr, payload) {
        // When the user clicks on the 'Submit payment'
         ↪  button this code will send the
        // encrypted payment information in a variable
         ↪  called a payment method nonce
        $.ajax({
          type: 'POST',
          url: '/checkout',
          data: {'paymentMethodNonce': payload.nonce}
        }).done(function(result) {
          // Tear down the Drop-in UI
          instance.teardown(function (teardownErr) {
            if (teardownErr) {
              console.error('Could not tear down Drop-in
               ↪  UI!');
            } else {
              console.info('Drop-in UI has been torn
               ↪  down!');
```

```
29                // Remove the 'Submit payment' button
30                $('#submit-button').remove();
31              }
32            });
33
34            if (result.success) {
35
                ↪ $('#checkout-message').html('<h1>Success</h1><p>Your
                ↪ Drop-in UI is working! Check your <a
                ↪ href="https://sandbox.braintreegateway.com/login">
                ↪ sandbox Control Panel</a> for your test
                ↪ transactions.</p><p>Refresh to try another
                ↪ transaction.</p>');
36            } else {
37              console.log(result);
38
                ↪ $('#checkout-message').html('<h1>Error</h1><p>Check
                ↪ your console.</p>');
39            }
40          });
41        });
42      });
43    });
44  </script>
```

Appendix 4 - BDD Scenarios

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Norwegian "SSN" example visible in hosted field
When customer enters incomplete Norwegian "SSN"
Then "onChange" event triggered with status code "400.001"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Norwegian "SSN" example visible in hosted field
When customer enters Norwegian "SSN" with typo
Then "onChange" event triggered with status code "400.002"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Norwegian "SSN" example visible in hosted field
When 17 years old customer enters Norwegian "SSN"
Then "onChange" event triggered with status code "400.003"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Norwegian "SSN" example visible in hosted field
When 126 years old customer enters Norwegian "SSN"
Then "onChange" event triggered with status code "400.004"

@Hosted-fields-profile
Scenario: Placeholder is visible in a hosted field, when merchant defined profile in config
Given merchant defined "IBAN-de-AT" profile in config
When page with hosted field renders
Then placeholder with Austrian "IBAN" example visible in hosted field

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Austrian "IBAN" example visible in hosted field

When customer enters non Austrian IBAN
Then "onChange" event triggered with status code "400.000"
And status object includes next fields fulfilled: ’fieldName’, ’isValid’, ’statusCode’ and ’message’

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Austrian "IBAN" example visible in hosted field
When customer enters incomplete Austrian IBAN (<22 symbols)
Then "onChange" event triggered with status code "400.001"
And status object includes next fields fulfilled: ’fieldName’, ’isValid’, ’statusCode’ and ’message’

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Austrian "IBAN" example visible in hosted field
When customer enters Austrian IBAN with typo (>22 symbols)
Then "onChange" event triggered with status code "400.002"
And status object includes next fields fulfilled: ’fieldName’, ’isValid’, ’statusCode’ and ’message’

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Austrian "IBAN" example visible in hosted field
When customer enters invalid Austrian IBAN
Then "onChange" event triggered with status code "400.005"
And status object includes next fields fulfilled: ’fieldName’, ’isValid’, ’statusCode’ and ’message’

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Austrian "IBAN" example visible in hosted field
When customer enters valid Austrian IBAN
Then "onChange" event triggered with status code "200"
And status object includes next fields fulfilled: ’fieldName’, ’isValid’, ’statusCode’, ’message’, ’bic’ and ’bankName’

@Hosted-fields-profile
Scenario: Placeholder is visible in a hosted field, when merchant defined profile in config

Given merchant defined "SSN-sv-SE" profile in config
When page with hosted field renders
Then placeholder with Swedish "SSN" example visible in hosted field

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters Swedish "SSN" with wrong separator
Given placeholder with Swedish "SSN" example visible in hosted field
When customer enters Swedish "SSN" with wrong separator
Then "onChange" event triggered with status code "400.000"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters incomplete Swedish "SSN"
Given placeholder with Swedish "SSN" example visible in hosted field
When customer enters incomplete Swedish "SSN"
Then "onChange" event triggered with status code "400.001"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters Swedish "SSN" with typo
Given placeholder with Swedish "SSN" example visible in hosted field
When customer enters Swedish "SSN" with typo
Then "onChange" event triggered with status code "400.002"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer is too young to use our service
Given placeholder with Swedish "SSN" example visible in hosted field
When 17 years old customer enters Swedish "SSN"
Then "onChange" event triggered with status code "400.003"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer is too old to use our service
Given placeholder with Swedish "SSN" example visible in hosted field
When 126 years old customer enters Swedish "SSN"
Then "onChange" event triggered with status code "400.004"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters valid SSN into hosted field input
Given placeholder with Swedish "SSN" example visible in hosted field

When customer enters valid Swedish "SSN"
Then "onChange" event triggered with status code "200"

@Hosted-fields-profile
Scenario: Placeholder is visible in a hosted field, when merchant defined profile in config
Given merchant defined "SSN-nb-NO" profile in config
When page with hosted field renders
Then placeholder with Norwegian "SSN" example visible in hosted field

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Norwegian "SSN" example visible in hosted field
When customer enters valid Norwegian "SSN"
Then "onChange" event triggered with status code "200"

@Hosted-fields-profile
Scenario: Placeholder is visible in a hosted field, when merchant defined profile in config
Given merchant defined "SSN-fi-FI" profile in config
When page with hosted field renders
Then placeholder with Finnish "SSN" example visible in hosted field

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Finnish "SSN" example visible in hosted field
When customer enters Finnish "SSN" with wrong separator
Then "onChange" event triggered with status code "400.000"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Finnish "SSN" example visible in hosted field
When customer enters incomplete Finnish "SSN"
Then "onChange" event triggered with status code "400.001"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Finnish "SSN" example visible in hosted field
When customer enters Finnish "SSN" with typo
Then "onChange" event triggered with status code "400.002"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Finnish "SSN" example visible in hosted field
When 17 years old customer enters Finnish "SSN"
Then "onChange" event triggered with status code "400.003"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Finnish "SSN" example visible in hosted field
When 126 years old customer enters Finnish "SSN"
Then "onChange" event triggered with status code "400.004"

@Hosted-field-profile
Scenario: "onChange" event triggered when customer enters value into hosted field input
Given placeholder with Finnish "SSN" example visible in hosted field
When customer enters valid Finnish "SSN"
Then "onChange" event triggered with status code "200"