DivideBuy
The smarter way to pay

02 September 2021

Retailer Integration Guide v1.10

Contents

1.0 Introduction

DivideBuy provides Interest-Free Credit services to retail customers across multiple ecommerce sectors. The integration of our technology capability with our own credit facility makes us unique in the marketplace. Our technology enables us to provide customers with accurate information specific to their purchase and to take fast and effective credit lending decisions enabling customers to purchase at the point of interest.

This requires communication between a Retailer and the DivideBuy platform to provide sufficient information exchange to support credit searches and to complete credit lending decisions. DivideBuy integrates with both the Retailers front end website and its Order Management System in order to achieve this. We support the customer journey from initial purchase decisions through to order completion, including after sales support. DivideBuy is therefore more than simply a payment gateway and consequently requires additional integration to assure a frictionless customer experience.

This document summarises the scope of activities required to integrate with DivideBuy, including an overview of the integration, detail the DivideBuy features specific to the Retailer and summaries of the key message exchanges. In addition, it provides the context to support bespoke Retailer integrations by outlining the functionality required. A contextual level of information is provided to guide this process. Due to commercial sensitivities, code level detail is not provided but can be obtained subject to approval by DivideBuy and acceptance of appropriate non-disclosure agreements.

DivideBuy supports integration with the major platforms via the provision of an extension plug-in installed on the Retailer site. Detailed implementation guides for major retail platforms are available upon request. For bespoke integrations additional development will be required to ensure frictionless integration. Message exchange is implemented via a set of bespoke APIs with this guide detailing the sequencing, triggers, and request/response content for each API.

1.1 Document Overview

DivideBuy offers Interest-Free Credit to our retail partners as a customer facility to enable them to spread the cost of their purchase over a period of 12 months or less. In order for this to be an easy transition for the Retailer, we have developed an Extension Pack which must be installed onto a Retailers CMS system so that the Interest-Free Credit option appears on their website. The Extension Pack will update the Retailers Sales Grid, Admin Panel and add additional content to some Retailer pages, including the checkout page. Our Technical Team will upload the Extension Pack via FTP/SFTP or via Version Control Software (GIT) and install the software directly onto the Retailers CMS. In addition to this, an API connection between the Retailer and DivideBuy is also integrated with the Retailer's website, enabling order details to be exchanged. The API connection will send information in relation to Order Initiation, Order Completion, Order Updates, Order Refunds/Returns etc.

Due to server security and the need for our API communication flow, we occasionally need our server IP addresses to be whitelisted as a required of the overall module installation/functionality and testing process, without which the functionality of the module will be impaired. If this is required, our Technical Team will liaise with the Retailers Technical Team to resolve.

Our Technical Team will support installation of the Extension Pack on the Retailer's site which will include extensive testing prior to live implementation. If our Technical Team locate any issues during the installation with the Retailer's site, they will liaise directly with the Retailers Technical Team for any changes required. Our Technical Teams have supported over 200 implementations and therefore have significant experience to assure the quality of service provided.

2.0 Extensions

The term 'Extension' is used collectively to describe the e-Commerce retail platforms which DivideBuy supports. This section will outline the key features for each extension type and steps to implement. We currently support Magento (1 and 2); WooCommerce and Shopify, with development in progress for Open Cart and Craft Commerce.

Bespoke integrations are supported which require custom development to implement. Agreement with a Retailer on approach, timescales and cost is a pre-requisite prior to DivideBuy commencing work in these cases. To aid the understanding of work required, this document will outline the Retailer side implementations and development needed to integrate with DivideBuy APIs.

2.1 Magento (1 and 2); WooCommerce and Shopify

When the DivideBuy Extension Pack is installed onto an e-Commerce Platform, a configuration area will be added into the platforms Administration console (aka Admin Panel). This will enable a Retailer to manage the relevant 'DivideBuy settings' for their site, including:

- Two attributes will be created for each product on the platform:
  - DivideBuy Enable/Disable
  - DivideBuy Product Tax Class – to be set at the VAT level appropriate to the product.

  If the product is selected within the Admin Panel, the Retailer will be able to define the values for these attributes as per their requirements.
- A column is added into the Retailers Product Grid to show which products are DivideBuy enabled.
- A settings area will be added to allow the Retailer to add Token and Authentication details. Once entered, the details will be sent to DivideBuy via their Core API and authenticated. Once validated, the Retailers information will be added into the DivideBuy system.
- The Configuration section allows the Retailer to upload the Product Page Banner; Cart page Banner and other relevant configuration details.
- A new payment method will be added to the Retailers Admin Panel. The DivideBuy payment method must be enables before it will be visible on the checkout page for the user.

Once all configurations are saved the customer will be able to see:

  - DivideBuy Banner for products that are DivideBuy enabled.
  - Choose the option to Checkout with DivideBuy.

- If a customer chooses the option to checkout with DivideBuy – APIs have been created and installed via the Extension Pack which will communicate with DivideBuy's checkout page. The customer will be redirected to the Retailers DivideBuy site and DivideBuy's Core API will communicate with the Retailers API, returning all order details to show in the checkout.
- When an order is completed/cancelled – DivideBuy's Core API will communicate with the Retailer to confirm the action and provide supporting information.
- Functionality is added to ensure that if an order is refunded an API is called from Retailer to Core API to provide the relevant data for the action to be processed.
- Once an order has been placed, the Retailer will have the functionality to enter Tracking Information where the details are sent via DivideBuy's Core API to store the tracking details on DivideBuy's system.

2.2 Bespoke E-Commerce Platforms

Unfortunately, the Extension Pack that we have developed is not compatible with Bespoke E-Commerce Platforms, however, by developing the functionality listed below, most e-Commerce Platforms are able to integrate with DivideBuy:

2.2.1 Resource Provided by DivideBuy

- Product Banners – for Desktop and Mobiles
- DivideBuy Branded Cart Buttons
- DivideBuy Checkout Logo
- Pop-Up Header Logo
- Bespoke DivideBuy Branded Banner (if required)
- Credit Page
- Token and Authentication API Keys
- API Message Definitions

2.2.2 System Integration Features Required

General Setup

- **API Credentials** – Credentials for the API must be provided for the exchange of information to be carried out securely and seamlessly.

Retailer Configurations

- **Add to the Retailer settings/configurations, another tab or area called: 'DivideBuy Settings'** – this is integral to the Extension Pack as it will store any DivideBuy setting and include all configurations required for the retailer website. Typically, this would cover General Settings, Product Management and Cart Management.
- **Adding the DivideBuy Payment Method** – the DivideBuy payment method must be added to the Admin Portal of the e-Commerce Platform with all the required configurations.

Product Management

- **Create attributes to enable/disable DivideBuy from Products** – this functionality allows the Retailer to enable/disable the DivideBuy option on different products within their product list. Providing the Retailer with full control of their products they wish to offer the DivideBuy service.

- **Add a column into the Product Grid displaying if the product is DivideBuy enabled** – this will provide the Retailer with an efficient view of their products lists and which ones are enabled to show the DivideBuy features.
- **Adding DivideBuy filter to the Product Grid** – by adding a DivideBuy filter, the Retailer will be able to view their DivideBuy enabled products within the Product Grid and filter with ease.
- **Add DivideBuy Banner below the Add to Cart button** – for enabled products the DivideBuy banner must be displayed below the 'Add to Cart' button on each product page. The DivideBuy banner must include the DivideBuy Tooltip, which displays the monthly instalments (3, 6, 9, 12 months or 3, 4, 5, 6, months, depending on preference) on hover. The DivideBuy Tooltip will use the product price as a basis for the figures displayed.

Cart Management

- **Add DivideBuy Banner within the Cart** – for enabled products the DivideBuy Banner must be displayed within the cart/basket page which displays the monthly instalments (3, 6, 9, 12 months or 3, 4, 5, 6, months, depending on preference), via the DivideBuy Tooltip, on hover. The Tooltip will use the total cart price as a basis for figures displayed.
- **DivideBuy Payment Method on Checkout Page** – a DivideBuy Radio Button/Logo must be displayed as a selection method within the Retailer checkout page, which will redirect the customer to the Retailers DivideBuy site.
- **DivideBuy Pop-Up functionality from the Cart or Checkou**t – this pop-up will include all scenarios for the customer to log in:
    - Guest Account button and Existing Customer section with Username and Password will be displayed within the pop-up. Both features can be utilised, or 1 or the other, depending on the Retailer's usual functionality.
    - After Log In the pop-up displays a postcode field. This is used to display the Retailers delivery/shipping types and values.
    - After the postcode section, Retailers shipping/delivery values are displayed. If there is only a single value displayed e.g. Free Shipping, then the value is automatically selected; if more than one the customer will need to select this.
    - After the shipping selection, the order is redirected to DivideBuy for the Credit Application process.

- **Management Non-DivideBuy Products Within the Cart** – ensuring the checkout processes DivideBuy enabled products only and redirecting order cancel and completion appropriately when considering additional non-DivideBuy products a customer may want to also order.
- **Adding Notice on Cancelled Orders** – if a customer has created a DivideBuy account and an order has been cancelled for any reason, a notice must be sent to DivideBuy to ensure our records are updated accordingly.

Order Management System

- **Placing Order when user Checkout from Cart Page** – calling DivideBuy API to process orders and ensuring the successful order notifications are processed to complete orders in Retailer's CMS.
- **Adding Inventory Adjustments** – this functionality will hide pending or incomplete orders with the stock allocated to the order form the Order Grid. This will be complete via the Admin settings by managing a flag at database level or by customer order status. If an order is cancelled the stock will return to the product list.
- **Order Refund Process** – a refund process must be initiated where DivideBuy's Core API is called when a full or partial refund is initiated from the Retailer.
- **Tracking Information** – tracking information must be sent via DivideBuy's Core API when orders are shipped to the customer.
- **Validations** – certain validations need to be put in place when placing an order with DivideBuy, including:
    - Checking the Minimum Order Value
    - Checking for Non-DivideBuy Products in Cart
    - Checking Availability of Stock

API Integration

- **Get Order Details API** – this provides DivideBuy with the order details that have been placed.
- **Success Order API** – this API ensure that DivideBuy are notified when an order has been successfully completed.
- **Cancel Order API** – this API ensures that DivideBuy are notified when an order has been cancelled.
- **Verify Postcode API** – this verifies the postcode entered within the pop-up screen.
- **Retailer Configurations API** – this provides DivideBuy with the Retailer configurations entered into their e-Commerce Platform.

2.3 **Create Guarantor Order API** – this API is used to create a clone of a cancelled order and verify the stock availability and order value in DivideBuy guarantor flow**DivideBuy General Communication Flow**

We believe that our communication flow is simple with different response and request triggers within the API, based upon different actions that have been taken. The below diagram outlines the functionality of DivideBuy when an order is placed via a Retailer's E-Commerce Platform.

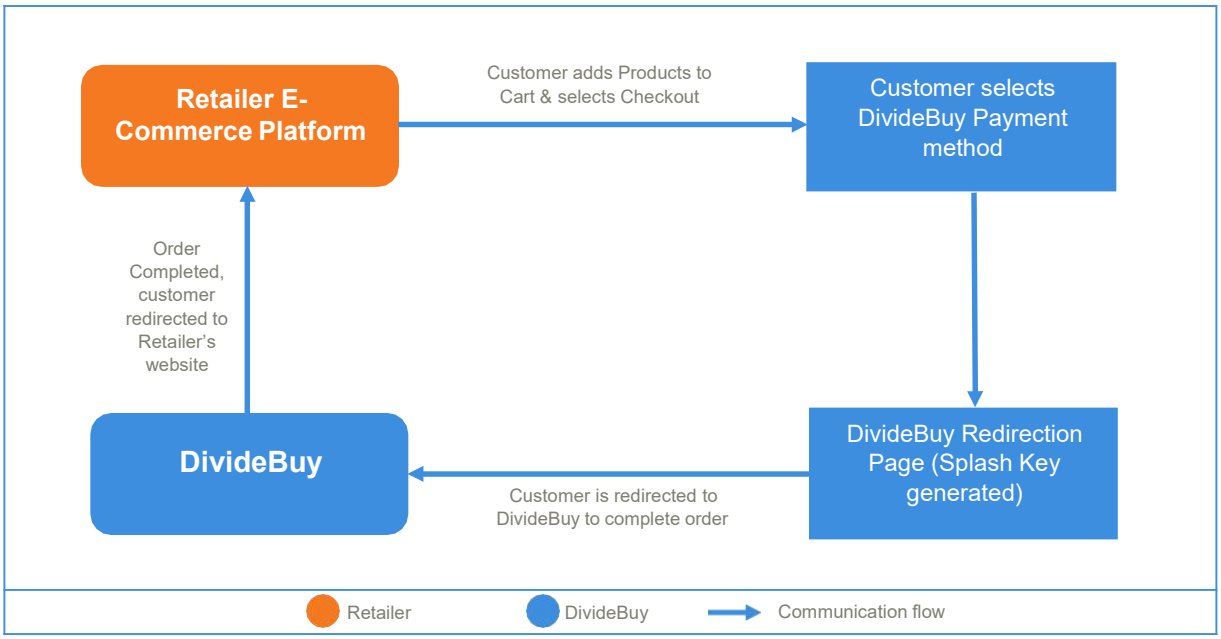

Figure 1: General Communication Flow

2.4 DivideBuy Pre-Requisites

In order to ensure that you have all you need before getting started, please find below a list of the pre-requisites we believe need to be in place:

- DivideBuy should always be treated as a Payment Method within the Retailers E-Commerce Configurations:
  - The DivideBuy Payment Method for the Checkout page: **Compulsory**
  - The DivideBuy Payment Method for the Cart page: **Optional**

3.0 DivideBuy Installation

This section will outline the installation setup for the integration with our API and fully define the methods, responses and requests that are used

3.1 Retailer Configuration Pre-Requisites

In order to ensure that you have all you need before getting started, please find below a list of the pre-requisites we believe need to be in place:

- DivideBuy exclude specific postal codes, and therefore Shipping Information should not be provided for these.
- DivideBuy have a 'Supported Couriers List' which will be stored on the Retailers E-Commerce Platform, once the integration has been configured. If Courier details are added or updated the 'Fetch Courier' method is broadcasted to update these details for all Retailers.

3.2 Retailer Setup and Configuration Process

This diagram and section outlines the process of the setup and configuration of a Retailer via DivideBuy, which also includes the process of confirming a Retailer.

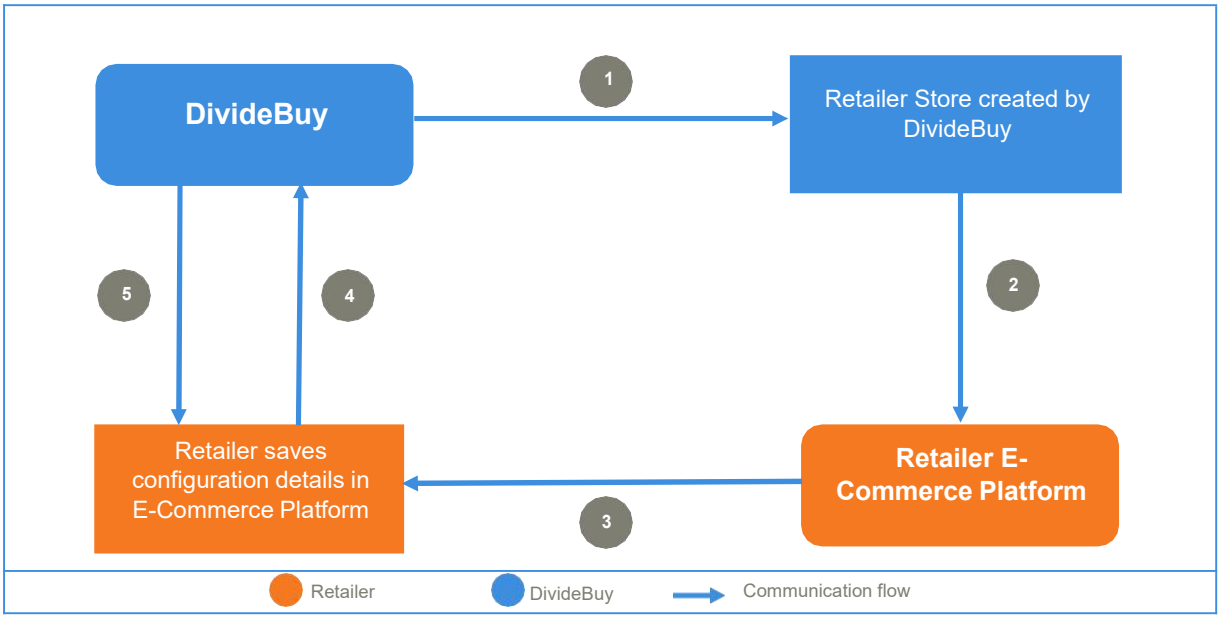

Figure 2: Retailer Setup and Configuration Process

Step 1: DivideBuy create a Retailer Store within the DivideBuy Portal. DivideBuy:

- Generate a Retailer Store Token and Store Authentication
- Setup Retailer Configuration details

Step 2: The Store Token and Store Authentication are sent offline to the Retailer.

Step 3: The Retailer saves the configurations details within their E-Commerce Platform.

Step 4: A Request is made from the Retailer to confirm their details via **PUT /api/confirmretailer**.
Step 5: A Response is returned to the Retailer from DivideBuy advising if the retail is confirmed.

3.3.1 Confirm Retailer

This section will provide further details regarding the process of confirming a Retailer and the methods that should be used.

- When the Retailer saves the configuration details (Step 3) a *PUT /api/confirmretailer* request call is initiated from DivideBuy to the Retailer, which will retrieve the Retailers Configuration for DivideBuy and store them within the Retailers Database.
- The following entities of the Retailer configuration are generated by DivideBuy and saved within the Retailers E-Commerce Platform:
  API: *retailerurl/dividebuy/api/response*;
  Method: *retailerConfiguration*
  - excludePostCodes
  - taxClass
  - inStoreCollection
  - flagToDeleteOrders
  - logoUrl
  - instalments
- To store the Couriers list within the Retailers E-Commerce Platform:
  API: retailerurl/dividebuy/api/response;
  Method: updateCouriers
  If an update has been made within the Courier List. POST /api/fetchcouriers is broadcasted to update the details in all Retailers.
- A flag labelled ‘call_retailer’ with a value of 0 or 1 is sent as a request parameter of api/confirmretailer.
  - If there is an update to the Retailer Store Token; Store Authentication and/or environment details at DivideBuy, the ‘call_retailer’ value should be set to 1 in the API request.
    - If ‘call_retailer’ = 1, an API call (retailerurl/dividebuy/api/response with method: retailerConfiguration) is initiated by DivideBuy to the Retailer to update the field values within the Retailers E-Commerce Platform.
    - When details are updated a Response will be sent from the Retailer to DivideBuy with the status ‘ok’.
    - ‘call_retailer’ is not sent in the request parameter of the API if there is no update to the above fields.

3.3 Retailer Configuration Details

This section will outline the specific configuration details that the Retailer must set.

3.3.1 DivideBuy VAT Hierarchy

Within the configuration details, DivideBuy offer 3 types of VAT, which are outlined below:

(i) **DivideBuy Product Level VAT** – This is VAT that is defined by the Retailer within their own E-Commerce Platform.

If this is set, consider this VAT

(ii) **Retailer Product VAT** – This is VAT that is defined by the respective Retailers for each product.

If (1) is not set, consider this VAT

(iii) **DivideBuy Default VAT** – This VAT is defined by default at the time the Retailer store is created by DivideBuy.

If (2) is not present, consider this VAT, with a default value of 20% VAT

3.3.2 Activate/Deactivate Retailer

Within the configuration details, a Retailer can be activated or deactivated in two different scenarios, which are outlined below:

(i) **Retailer is Activated/Deactivated by DivideBuy**

- When a Retailer is activated/deactivated by DivideBuy an API is called when the Retailer Configuration is saved – under Method: retailerConfiguration, an attribute labelled 'is_deactivate' is added as a parameter to the API request.
    - A field must be created within the Retailers E-Commerce Platform to store the current value of the 'is_deactivate' parameter.
        - 'is_deactivate' = 0 (activated) – the Retailer must update the 'retailer_id' field (which is provided as a parameter within the API call).
        - 'is_deactivate' = 1 (deactivated) – the Retailer will no longer be able to configure any aspects regarding DivideBuy.

(ii) **Retailer is Deactivated via the Retailers E-Commerce Platform**

- If a Retailer activates/deactivates themselves, the DivideBuy API (api/updateretailerstatus) will be called from the Retailer to DivideBuy with the following request parameters:

Request – Where the ***Retailer Status*** value can be **0** (activate) or **1** (deactivate):

```
{
  storeToken
  storeAuthentication
  retailerStatus
}
```

4.0 DivideBuy Order Placement

This section will outline the process flow of when an order is placed and highlights the way in which DivideBuy handles different outcomes that can occur when placing an order.

4.1 Order Placement Pre-Requisites

In order to ensure that you have all you need before getting started, please find below a list of the pre-requisites we believe need to be in place:

- **A Minimum Order Value** must be set during the Retailer Configuration & Setup stage. This amount should always be greater than or equal to the '£XX.XX' order value set.
    - A Minimum Order Value is considered after any discounts have been applied and excludes VAT added to shipping & handling costs.
- Customers shipping information must be the same as their billing address, which is retrieved as a response parameter of *POST dividebuy/api/getorderdetails.*
- When an order is created by the customer a hidden order must be created by the Retailer to ensure that the stock selected by the customer is frozen until the order is completed. This will allow the customer to complete the order without any issues in relation to stock being unavailable.

4.2 Order Placement General Flow

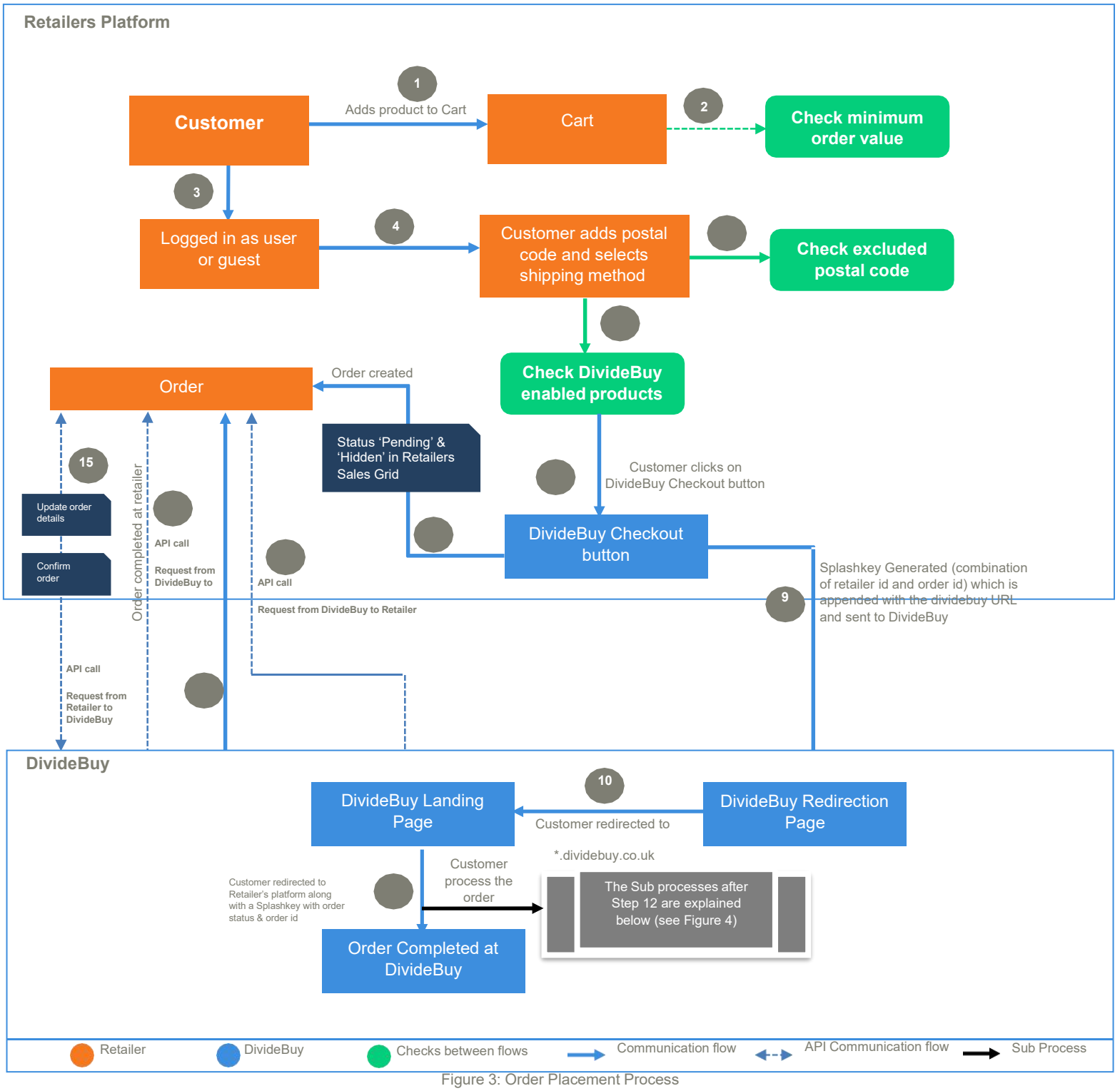

Figure 3: Order Placement Process

Step 1: A customer enters the Retailers Website and adds products to their cart.

Step 2: A check is made on the Minimum Order Value entered within the Retailer Configuration – if the amount is equal to or above the Minimum Order Value the DivideBuy buttons will be displayed.

Step 3: The customer logs in to their User Account for the Retailer or checks out as a guest user.

Step 4: Customer enters their postal code and selects their shipping method.

Step 5: The postal code is checked to ensure it is not one of the excluded postal codes – if the postal code is excluded and error will be displayed.

Step 6: A check is made on the cart Products within the Retailer Configuration- To check whether the selected products are DivideBuy Enabled or not, if all products are enabled “DivideBuy checkout button” will be displayed.

Step 7: The customer clicks on the DivideBuy Checkout button.

Step 8: Order is created at Retailers’ end with a status of ‘Pending’ and as ‘Hidden’ in the retailer’s sales grid

Step 9: Splashkey will be generated at Retailer’s end which includes the Order ID and Retailer ID, Which is appended with the DivideBuy URL and redirection page is displayed.

Step 10: The customer is redirected to the Retailers DivideBuy Checkout Landing Page.

Step 11: A number of requests (via POSTdividebuy/api/getorderdetails) and responses are carried out based upon the outcome of the order. (Refer section 6.4.3)

Step 12: Customer process the order and the order is completed at DivideBuy’s end

Note: There are different sub processes carried out after the order is processed till it is completed at DivideBuy’s end (Refer 4.2.1)

Step 13: User is redirected to retailer along with a Splashkey which includes order status and order id.

Step 14: In parallel with Step 13, request (via POST/api/response method: order Success) and response is carried out to complete order to make sure if the user is not redirected then also the order is completed at retailer side (Refer section 6.4.1)

Step 15 (a): A request (via POST/api/getuserorder) and response is carried out from Retailer to DivideBuy to update the order details at Retailer’s end (Refer section 6.3.4)

Step 15 (b): Also, a request (via POST/api/syncretorder) and response is carried out from Retailer to DivideBuy to confirm the order details are successfully synced at retailer’s end (Refer section 6.3.5)

4.2.1 Order Placement Sub Processes

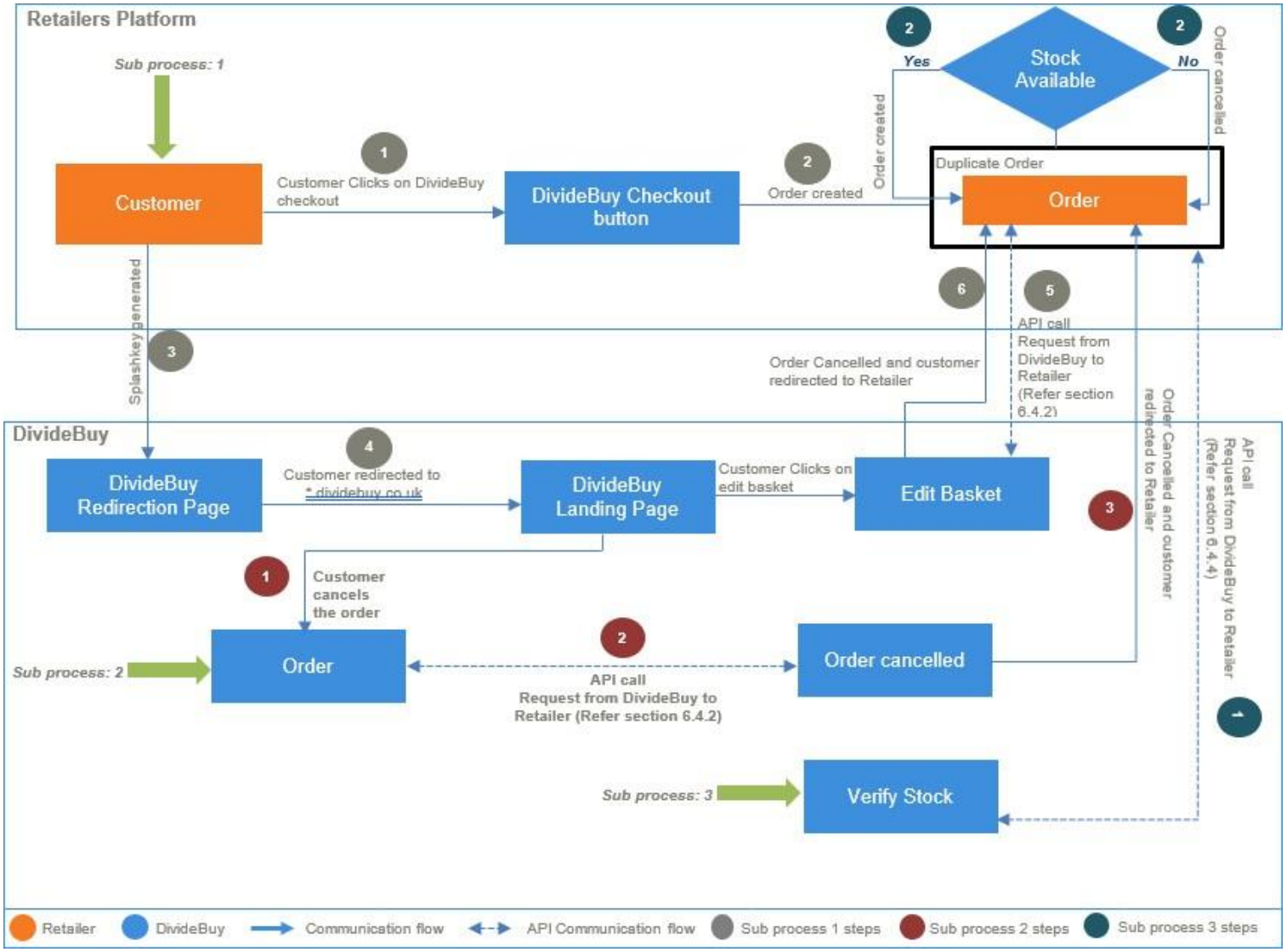

Figure 4: Order Placement Sub processes

Sub process: 1(Create Order and Edit Basket)

Step 1: As explained in the above 4.2 after Step 6, customer clicks on DivideBuy checkout button.

Step 2: Order is created at Retailers' end

Step 3: Splashkey generated. Redirection page displayed

Step 4: The customer is redirected to the Retailers DivideBuy Checkout Landing Page.

Step 5: On the checkout landing page, if the customer clicks on "Edit Basked "button, a request (via POST/api/response –method: orderCancel) & response is carried out to update the order status as "cancel" at both DivideBuy and Retailer side.

Step 6: Order is cancelled and customer is redirected to Retailer's end.

Sub process: 2(Cancel Order)

Step 1: Customer manually cancels the order (via “cancel order” button).

Step 2: A request (via POST/api/response –method: orderCancel) & response is carried out to update the order status as “cancel” at both DivideBuy and Retailer side.

Step 3: Order is cancelled and customer is redirected to Retailer’s end.

Sub process: 3 (Verify Stock)

In the DivideBuy guarantor flow we need to release the stock and after a certain period of time we need to verify the stock is available or if there is any change in the order value.

The flow we follow is cancel the old order that was present in the retailer side to release the stock and when we need to verify , if stock is available to complete the order, DivideBuy calls “createguarantororder” API which follows the below steps:

Step 1: To process the cancelled order, a request (via POST /api/createguarantororder) and response is carried out to create clone order (Refer section **Error! Reference source not found.**)

Step 2: If Stock is available, duplicate order will be created at retailer side and send the updated store order id, if the old order is not cancelled at any point then send back the same order id with status = “ok” , else send the response code as mentioned in section **Error! Reference source not found.**.

4.2.2 Order Created (Retailer’s Sales Grid)

When a customer confirms their postal code and selects the ‘DivideBuy Checkout’ button, the customer is redirected to the DivideBuy Landing Page and the order is created within the Retailer’s Extension with a **Status of Pending** and as a **Hidden order** in the Retailer’s Sales Grid

Note: The order is hidden in Retailer’s sales grid based on a flag maintained at retailer side, the flag value is updated once the order is successfully completed at retailer’s end.

4.2.3 Successful Order

When an order has been successfully completed within the DivideBuy landing page, DivideBuy will initiate a request call:

API: POST/api/response;

Method: orderSuccess

The order details received must be stored by the Retailer. Once received the Retailer a response must be sent to DivideBuy, as displayed in the example below:

```
{
  “Status”:”ok”
  “order_id”:”6667”
  “message”:”Order placed successfully”
}
```

4.2.4 Cancel Order Process

DivideBuy require the Retailer to create the following methods within their Platform to ensure that the order status is updated and to deliver a response if the order has been cancelled via DivideBuy:

(i) **Update Order Status**

If an order has been cancelled via DivideBuy the Retailer will receive an 'order_status' of 'cancel' - this order will need to be cancelled on the Retailers side.

API: *retailerurl/dividebuy/api/response*

Method: *orderCancel*

(ii) **Redirect User to a Landing Page**

When the order has been cancelled and the above API is called the following URL must be called in parallel when an order has been cancelled by DivideBuy: ***retailerurl/dividebuy/payment/splashKey=Y2FuY2VsOjQ1MTQ=***

The Retailer must direct the customer to a specific page displaying a message stating that the order has been cancelled.

4.2.5 Refund Process

In relation to refunds, DivideBuy offer two categories for Refunds:

(i) Full Refund

(ii) Partial Refund

If a refund is initiated when the Retailer is offline, the Retailer should initiate a request via:

API: POST/api/refund

Method: refundType (the type must be specified i.e. Full or Partial Refund)

4.2.6 Verify Postal Code Process

DivideBuy currently have postal codes that are excluded, therefore the postcode entered by the customer must be verified to ensure that the postcode is not excluded:

- When a Registered DivideBuy user logs into their DivideBuy account to place the order, the postcode is verified against the customer's order details (dividebuy/api/getorderdetails).
- If the value of the Postcode of the Registered DivideBuy user is different between the Retailers Platform and DivideBuy's details, the below API call is initiated:
    - A call is made via api/verifypostcode which initiates a request from DivideBuy to the Retailer to check available shipping methods and costs for the postcode entered:
        - **No change in values:** The Retailer must send an 'ok' in response to the API.
        - If we are unable to cater for this request, an error will appear, the order will be cancelled, and the customer will be redirected to the Retailers Website.

4.2.7 Tracking Process

DivideBuy have the ability to store Tracking Numbers for orders that have been placed. This functionality allows the Retailer to enter the Tracking Number via an API, which is then updated on DivideBuy's system.

- When a Tracking Number has been defined within the Retailers E-Commerce Platform, a request for POST/api/tracking is initiated from the Retailer to DivideBuy:
    - The API includes Retailer information and Tracking information in the request parameters.
    - If there is no error in the information received, DivideBuy will return a status: 'ok' as a response to the API request.

4.2.8 Minimum Order Value

The Minimum Order Value defines the lowest value at which the Retailer offers DivideBuy's interest free services. This will be defined within the 'Instalments' section of the initial Retailer Configuration part of the process.

Retailer ID	Type	Key	Instalment Value (£)
17	Instalment	3	50
	Instalment	6	250
	Instalment	9	300
	Instalment	12	500

A 'Minimum Order Value' is the lowest value added amongst all instalment amounts defined by the Retailer e.g.:

Within this table the lowest Instalment Value = £50. Therefore, as this is the lowest value over all four Instalment Values, the minimum_order value for any order must ≥£50.

5.0 DivideBuy Environments

The Retailer can enable DivideBuy configurations, within the following environments.

Mode	URL Redirection
Live Environment	**Checkout:** dividebuy.co.uk/#/login **API:** api.dividebuy.co.uk
Test Environment	**Checkout:** dividebuysandbox.co.uk/#/login **API:** api.dividebuysandbox.co.uk

6.0 API Communication

The below section describes the communication flow and message exchange between DivideBuy and the Retailer.

The API provides testing environment, which enables retailer to test the API at their end before getting it live.

DIVIDEBUY_API_URL_STAGING = https://api.dividebuysandbox.co.uk

DIVIDEBUY_ORDER_URL_STAGING = https://{{retailername}}.dividebuysandbox.co.uk/#/login

DIVIDEBUY_API_URL_PRODUCTION = https://api.dividebuy.co.uk/

DIVIDEBUY_ORDER_URL_PRODUCTION = https://{{retailername}}.dividebuy.co.uk/#/login

All API calls start with: `https://api.dividebuy.co.uk`

Path: For this documentation, we will assume that every request begins with the above path.

Format: All calls are returned in **JSON**

6.1 Status Codes

100	Continue	**410**	Gone
101	Switching Protocols	**411**	Length Required
102	Processing	**412**	Precondition Failed
200	Ok	**413**	Request Entity Too Large
201	Created	**414**	Request URL Too Long
202	Accepted	**415**	Unsupported Media Type
203	Non Authoritative Information	**416**	Request Range Not Satisfiable
204	No Content	**417**	Expectation Failed
205	Reset Content	**418**	I Am A Teapot
206	Partial Content	**421**	Misdirected Request
207	Multi Status	**422**	Unprocessable Entity
208	Already Reported	**423**	Locked
226	Im Used	**424**	Failed Dependency
300	Multiple Choices	**425**	Reserved for Webdav Advanced Collections Expired Proposal
301	Moved Permanently	**426**	Upgrade Required
302	Found	**428**	Precondition Required
303	See Other	**429**	Too Many Requests
304	Not Modified	**431**	Request Header Fields Too Large
305	Use Proxy	**451**	Unavailable For Legal Reasons
306	Reserved	**500**	Internal Server Error
307	Temporary Redirect	**501**	Not Implemented
308	Permanently Redirect	**502**	Bad Gateway
400	Bad Request	**503**	Service Unavailable
401	Unauthorized	**504**	Gateway Timeout

402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
505	Version Not Supported
506	Variant Also Negotiates Experimental
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required

6.2 API Communication Flows

This section will provide diagrams depicting the flow of information within different scenarios, outlining the communication flow between the customer, Retailer and DivideBuy.

6.2.1 Retailer Setup & Configuration API Flow

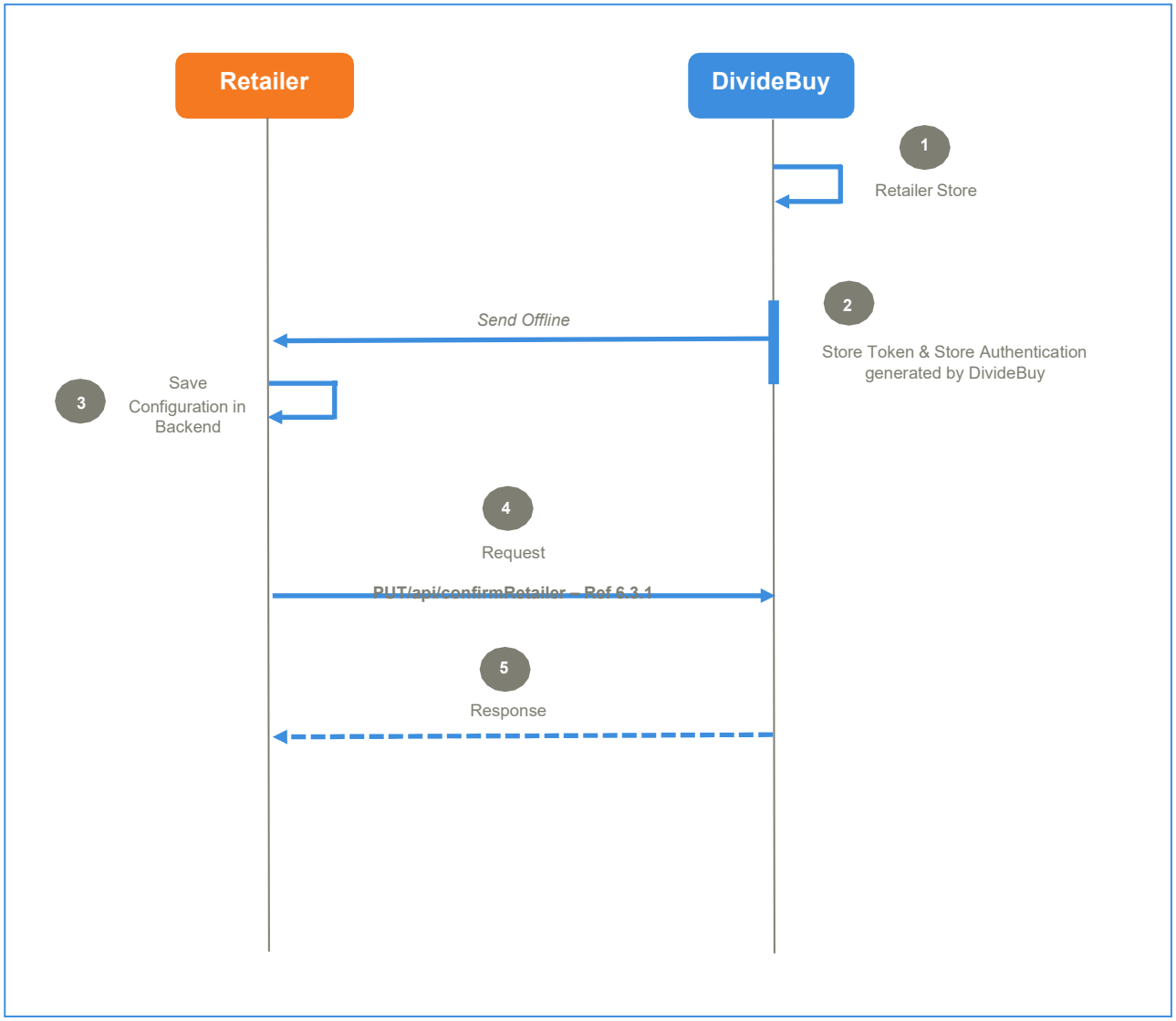

Figure 4: Retailer Setup and Configuration Flow

Step 1: DivideBuy create a Retailer Store within their Platform.

Step 2: A Store Token and Store Authentication are generated by DivideBuy and sent to the Retailer offline.

Step 3: The Retailer updates the Retailer Configuration section and Saves this to their Platform.

Step 4: An API call is made sending the information to DivideBuy (PUT/api/confirmRetailer).

Step 5: DivideBuy respond to the API call confirming the Retailer

6.2.2 Confirm Retailer

PUT/api/confirmretailer

From: Retailer E-Commerce Platform to DivideBuy Request

Request

Variable Name	Type	Length	Valid Value	Description
storeToken	String	255		
storeAuthentication	String	255		
retailerStoreCode	String	250		
callRetailer	Integer	10	0/1	

Example Request:

```
{
   "storeToken": "5LIH1TaW8ewd",
   "storeAuthentication": "3aa7Sgt76sz7",
   "retailerStoreCode": "default",
   "callRetailer": "1",
}
```

Response

Variable Name	Type	Length	Valid Value	Description
status	String		ok	
retailerId	Integer	10		

Example Response:

```
{
   "status": "ok",
   "retailerId": 150
}
```

6.2.3 Retailer Configuration

POST /api/response

From: DivideBuy to Retailer E-Commerce Platform

Request

Variable Name	Type	Length	Valid Value	Description
method	String		retailerConfigurations	
retailer_id	Integer	10		
retailer_store_code	String	255		
store_token	String	255		
store_authentication	String	255		
retailerConfigurationDetails				
- type	Enum	255	themes | global | instalments | shopify	
- key	String	255		
- value	String	255		
- retailer_id	Integer	10		

Example Request:


```
{
  "method": "retailerConfigurations",
  "retailer_id": 150,
  "retailer_store_code":",
  "store_token": "5LIH1TaW8ewd",
  "store_authentication": "3aa7Sgt76sz7",
  "retailerConfigurationDetails": [
    {
      "type": "global",
      "key": "taxClass",
      "value": "10",
      "retailer_id": "150"
    },
    {
      "type": "global",
      "key": "excludePostCodes",
      "value": "DE4 3ED",
      "retailer_id": "150"
    },
    {
      "type": "themes",
      "key": "basketColour",
      "value": "#ec0000",
      "retailer_id": "150"
    },
    {
      "type": "themes",
      "key": "contactUrl",
      "value": "contact.com",
      "retailer_id": "150"
   },
   {
     "type": "themes",
     "key": "logoUrl",
     "value": "testlogo",
     "retailer_id": "150"
   },
   {
```

```
      "type": "themes",
      "key": "themeColour",
      "value": "#116d96",
      "value": "#116d96",
      "retailer_id": "150"
    },
    {
      "type": "instalments",
      "key": "3",
      "value": "75",
      "retailer_id": "150"
    },
    {
      "type": "instalments",
      "key": "4",
      "value": "160",
      "retailer_id": "150"
    },
    {
      "type": "instalments",
      "key": "5",
      "value": "200",
      "retailer_id": "150"
    },
    {
      "type": "instalments",
      "key": "9",
      "value": "699",
      "retailer_id": "150"
    },
    {
      "type": "instalments",
      "key": "12",
      "value": "999",
      "retailer_id": "150"
    }
  ]
}
```

Response

Variable Name	Type	Length	Valid Value	Description
error	Boolean		1 | 0	1 = Error; 0 = No Error
success	Boolean		1 | 0	1 = Request Successful; 0 = Request Not Successful
status	String		ok	

Example Response:

```
{
   "error": 0,
   "success": 1,
   "status": "ok"
}
```

6.2.4 DivideBuy Order Placement API

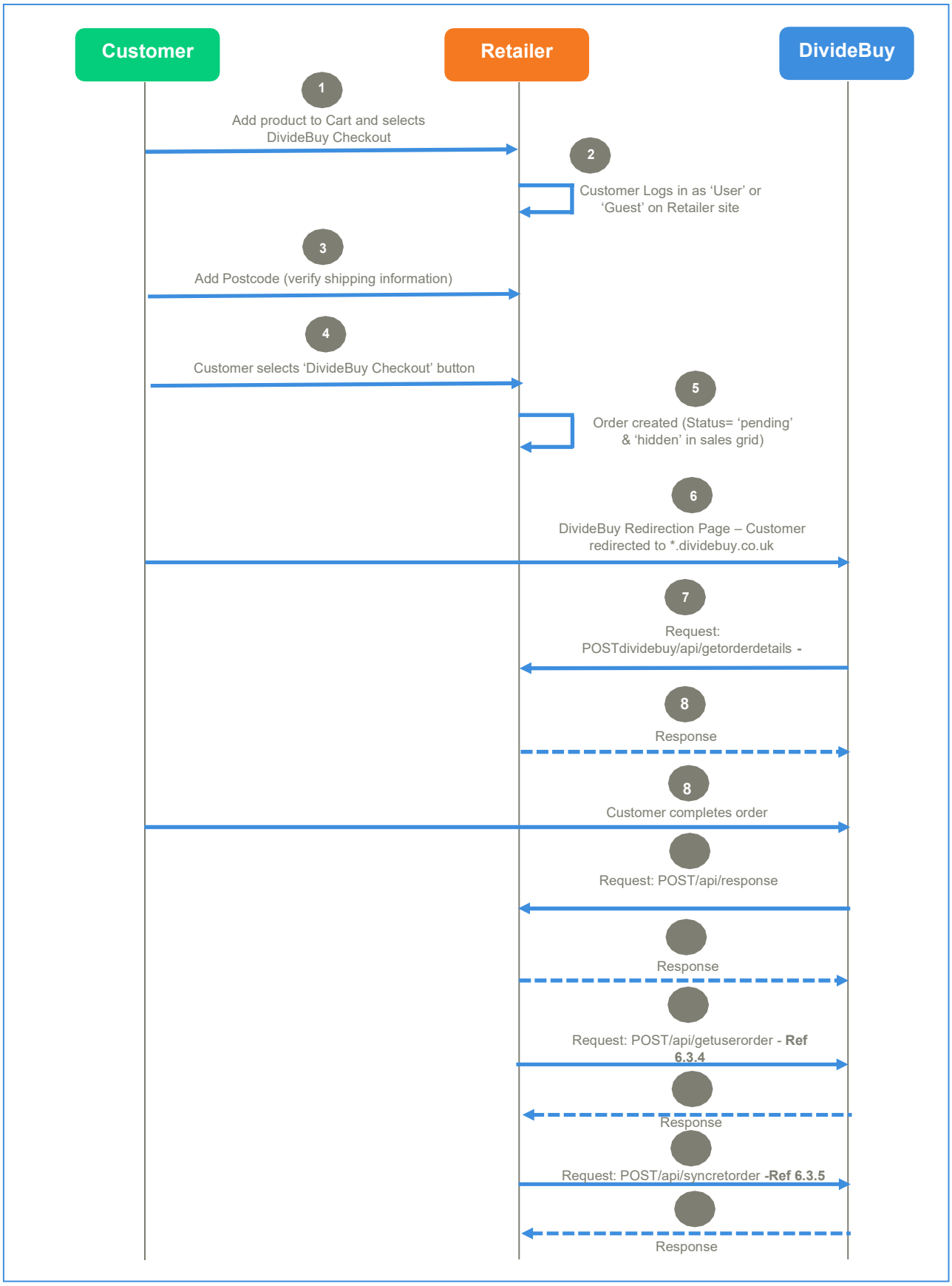

Figure 5: DivideBuy Order Placement

Step 1: Customer adds Retailer Products into their Cart and selects DivideBuy Checkout.

Step 2: Customer Logs into the Retailer site either with their Account details or as a Guest.

Step 3: Customer enters their Postcode, this is verified against DivideBuy's excluded postal codes and if it is accepted the Retailers Shipping Information is displayed.

Step 4: A order is created within the Retailers end, with a status of 'Pending' and Hidden' in the Retailers Sales Grid

Step 5: Customer selects the DivideBuy Checkout Button.

Step 6: Customer lands on a redirection page which redirects them to the Retailers DivideBuy Checkout.

Step 7: An API call is made to the Retailer to get the order details (POSTdividebuy/api/getorderdetails).

Step 8: A Response is returned from the Retailer to confirm that they have received the information.

Step 9: Customer process the order and the order is completed at DivideBuy's end, user will be redirected to Retailer

Step 10: An API call is made to the Retailer once the order is completed at DivideBuy's end to make sure if the user is not redirected then also the order is completed at retailer side *(*POST/api/response method: orderSuccess)

Step 11: A Response is returned from the Retailer to confirm that the order is completed at retailer's end.

Step 12: An API call is made from Retailer to DivideBuy to store the order details at retailer side and Sync the order details in retailer's Database (POST/api/getuserorder)

Step 13: A Response is returned from the DivideBuy to confirm that the order details have been successfully stored at Retailer's end

Step 14: An API call is made from Retailer to DivideBuy - to confirm the order details are synced at Retailer's end (POST/api/syncretorder)

Step 15: A Response is returned from the DivideBuy with Status "ok".

6.2.5 DivideBuy Process APIs

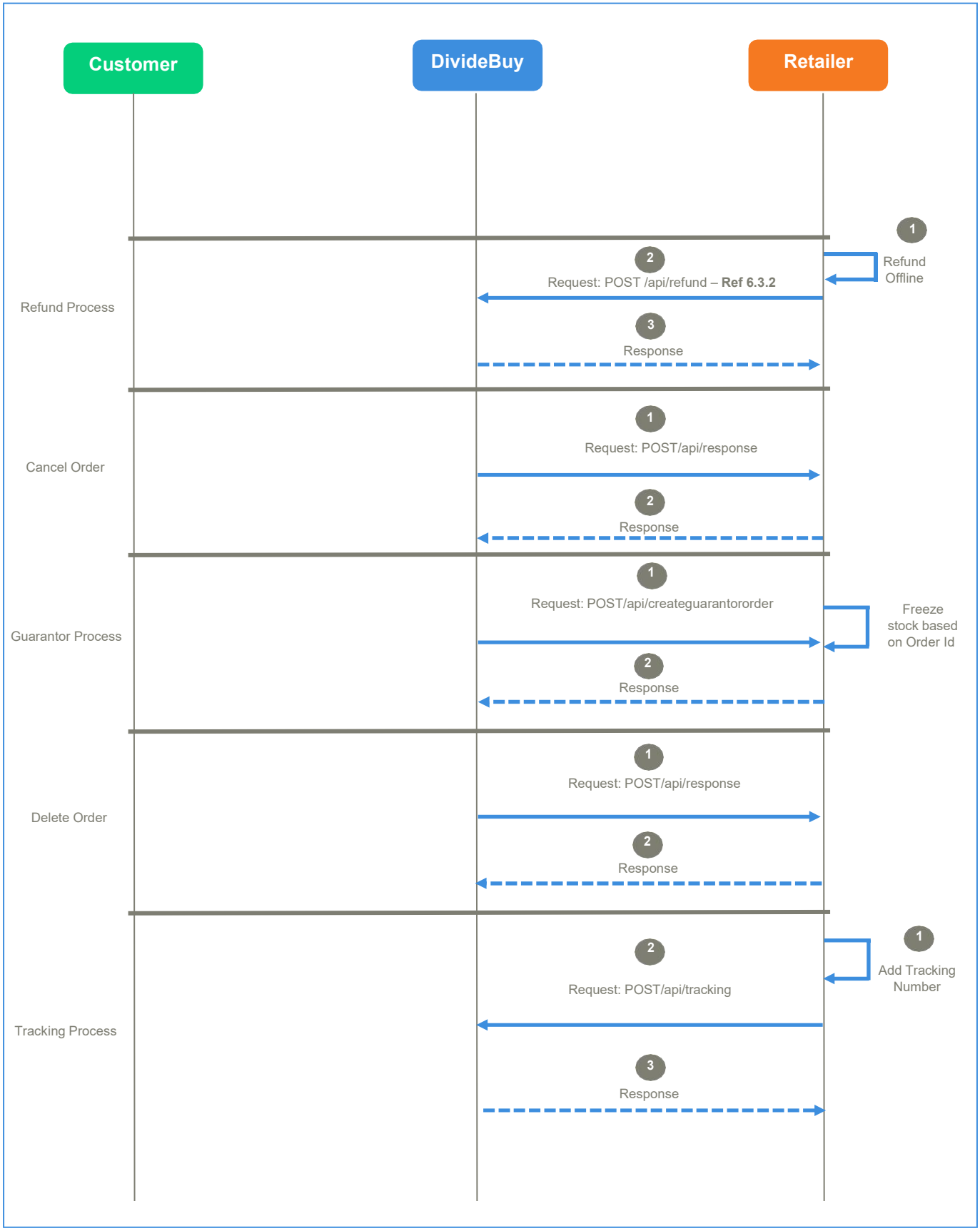

Figure 6: Process API Flows

6.3 DivideBuy Checkout

This API is used by Retailer E-Commerce Platforms to communicate with DivideBuy.

6.3.1 Get Order Details

POST dividebuy/api/getorderdetails

From: DivideBuy to Retailer E-Commerce Platform

Request

Variable Name	Type	Length	Valid Value	Description
orderId	Integer	20		
retailerStoreCode	String	255		
storeAuthentication	String	255		
storeToken	String	255		

Example Request:

```
{
   "orderId":"6667",
   "retailerStoreCode":"default",
   "storeAuthentication":"5LIH1TaW8ewd",
   "storeToken":"3aa7Sgt76sz7"
}
```

Response

Variable Name	Type	Length	Valid Value	Description
order_detail				
- store_order_id	Integer	10		
- store_order_increment_id	Varchar	10		
- store_token	String	255		
- store_authentication	String	255		
- logo_url	String	255		
- grand_total	Decimal	14,3		
- subtotal	Decimal	14,3		
- subtotalInclVat	Decimal	14,3		
- discount	Decimal	14,3		
- discountApplied	String	255	beforeVat | afterVat	
- shipping	Decimal	14,3		
- shippingInclVat	Decimal	14,3		
- shipping_label	Text			
- shipping_method	Text			
- is_default_shipping	Boolean		1 | 0	1 = default shipping; 0 = not default shipping
- is_default_billing	Boolean		1 | 0	1 = default billing; 0 = not default billing
- vat	Decimal	14,2		
product_details				
- name	String	255		
- sku	String	255		
- qty	Decimal	14,3		
- price	Decimal	14,3		
- priceInclVat	Decimal	14,3		
- rowTotal	Decimal	14,3		

- rowTotalInclVat	Decimal	14,3		
- discount	Decimal	14,3		
- short_description	Text			
- product_type	String	255		
- product_weight	String	255		
- product_visibility				
- divVat	Decimal	14,3		
- image_url	String	255		
- **product_options**				
○ colour	String	255		
○ size	String	255		
shipping_address				
- first_name	String	255		
- last_name	String	255		
- email	String	255		
- street	String	255		
- postcode	String	255		
- region	String	255		
- city	String	255		
billing_address				
- first_name	String	255		
- last_name	String	255		
- email	String	255		
- street	String	255		
- postcode	String	255		
- region	String	255		
- city	String	255		

Example Response:


```
{
   "order_detail":
      { "store_order_id":
      “6667",
      "store_order_increment_id": “145006485",
      "store_token": “5LIH1TaW8ewd",
      "store_authentication": “3aa7Sgt76sz7",
      "logo_url": “https://moduleinstalledmagento1.dbuytest.info/media/dividebuy/",
      "grand_total": 318.4,
      "subtotal": 265.33,
      "subtotalInclVat": 318.4,
      "discount": 0,
      "discountApplied": "beoforeVat",
      "shipping": 0,
      "shippingInclVat": 0,
      "shipping_label": "Free Shipping - Free",
      "shipping_method": "freeshipping_freeshipping",
      "is_default_shipping": 0,
      "is_default_billing": 0,
      "vat": 53.07
   },
   "product_details": [
      {
         "name": “Some product name",
         "sku": “SKU",
         "qty": "1.0000",
         "price": "249.1700",
         "priceInclVat": "299.0000",
         "rowTotal": "249.1700",
         "rowTotalInclVat": "299.0000",
         "discount": "0.0000",
         "short_description": “Some Product",
         "product_type": "simple",
         "product_weight": "35.5000",
         "product_visibility": "4",
         "DivVat": "20",
         "image_url": “some url.jpg"
      }
   ],
   "shipping_address":
      { "first_name":
      “name","last_name":
      “name",
      "email": “retailer@dividebuy.co.uk",
      "street": [
         “Address 1",
         “Adress 2"
      ],
      "postcode": “DE4 3ED",
      "region": “County",
      "city": “town"
   },
   "billing_address":
      { "first_name":
      “name","last_name":
      “name",
      "email": “retailer@dividebuy.co.uk",
      "street": [
         “Address 1",
         “Adress 2"
      ],
      "postcode": “ST15 8YR",
      "region": “County",
      "city": “town"
   }
}
```

6.3.2 Cancel Order

POST /api/response

From: DivideBuy to Retailer E-Commerce Platform

Request

Variable Name	Type	Length	Valid Value	Description
method	String		orderCancel	
store_order_id	Integer	20		
retailer_store_code	String	255		
store_token	String	255		
store_authentication	String	255		
delete_user_order	Boolean		1 | 0	1 = delete order; 0 = do not delete order after cancellation

Example Request:

```
{
  "method": "orderCancel",
  "store_order_id": 6667,
  "retailer_store_code": "default",
  "store_token": "5LIH1TaW8ewd",
  "store_authentication": "3aa7Sgt76sz7",
  "delete_user_order": 1
}
```

Response

Variable Name	Type	Length	Valid Value	Description
error	Boolean		1 | 0	1 = Error; 0 = No Error
success	Boolean		1 | 0	1 = request successful; 0 = request not successful
status	String		ok	
order_id	Integer	10		

Example Response:

```
{
  "error": 0,
  "success": 1,
  "status": "ok",
  "order_id": "6667"
}
```

6.3.3 Verify Postcode

POST /api/verifypostcode

From: DivideBuy to Retailer E-Commerce Platform

Request

Variable Name	Type	Length	Valid Value	Description
orderId	Integer	20		
userPostcode	String	255		
retailerStoreCode	String	255		
storeToken	String	255		
storeAuthentication	String	255		

Example Request:

```
{
    "orderId": "6667",
    "userPostCode": "AA1 1AA",
    "storeToken": "5LIH1TaW8ewd",
    "storeAuthentication": "3aa7Sgt76sz7",
    "retailerStoreCode": "default"
}
```

Response

Variable Name	Type	Length	Valid Value	Description
error	Boolean		1 | 0	1 = Error; 0 = No Error
success	Boolean		1 | 0	1 = Request Successful; 0 = Request Not Successful
message	ok	255	ok	
status	Integer	255	200	

Example Response:

```
{
   "error": 0,
   "success": 1,
    "message": "ok",
   "status": "200"
}
```

6.3.4 Success Order

POST /api/response

From: DivideBuy to Retailer E-Commerce Platform

Request

Variable Name	Type	Length	Valid Value	Description
method	String		orderSuccess	
store_order_id	Integer	20		
store_token	String	255		
store_authentication	String	255		
retailer_store_code	String	255		
customer_email	String	255		
address				
- first_name	String	255		
- last_name	String	255		
- street	String	255		
- postcode	String	255		
- region	String	255		
- city	String	255		
- contact_number	String	255		
- house_number	String	255		
- house_name	String	255		

Example Request:


```
{
  "method": "orderSuccess"
  "store_order_id":"6667",
  "store_token":"5LIH1TaW8ewd",
  "store_authentication":"3aa7Sgt76sz7",
    "retailer_store_code":"default",
    "customer_email":"retailer@dividebuy.co.uk",
    "order_ref_id":"10000000067",
    "orderTime":"27-11-2019 12:00:00",
    "address":{
        "prefix":"Mr",
        "first_name":"John",
        "last_name":"Smith",
        "street":"Brunswick Street",
        "address2":"high street",
        "postcode":"ST5 1HH",
        "region":"Region Name",
        "city":"City Name",
        "contact_number": "08000850885",
        "house_number": "1",
        "house_name": "Brunswick Court",
    }
}
```

Response

Variable Name	Type	Length	Valid Value	Description
status	String		ok	
order_id	Integer	10		OrderId of the generated order
message	Text			

Example Response:

```
{
   "status":"ok",
   "order_id":"6667",
   "message":"Order placed successfully"
}
```

6.3.5 Get User Order

POST /api/getuserorder

From: Retailer E-Commerce Platform to DivideBuy

Request

Variable Name	Type	Length	Valid Value	Description
storeOrderId	Integer	20		
storeToken	String	255		
storeAuthentication	String	255		

Example Request:

```
{
   storeOrderId: "6667",
   storeToken: "5LIH1TaW8ewd",
   storeAuthentication: "3aa7Sgt76sz7"
}
```

NB: Response on next page

Response

Variable Name	Type	Length	Valid Value	Description
status	String		ok	
error	Boolean		1 | 0	1 = Error; 0 = No Error
data				
• method	String		orderSuccess	
• store_order_id	Integer	20		
• store_token	String	255		
• store_authentication	String	255		
• order_status	String	255		
• orderTime	DateTime			
• retailer_store_code	String	255		
• customer_email	String	255		
• **address**				
○ prefix	String	255		
○ first_name	String	255		
○ last_name	String	255		
○ contact_number	String	255		
○ house_name	String	255		
○ house_number	String	255		
○ street	String	255		
○ address2	String	255		
○ postcode	String	255		
○ region	String	255		
○ city	String	255		

NB: Example Response on next page

Example Response:


```
{
"status":"ok","data":
{
  "method":"orderSuccess",
  "store_order_id":"6667",
  "store_token":"5LIH1TaW8ewd",
  "store_authentication":"3aa7Sgt76sz7",
  "order_status":"success",
  "orderTime":"2017-08-23 08:31:36",
  "retailer_store_code":"",
  "customer_email":"retailer@dividebuy.co.uk",
  "order_ref_id":"10000000067",
  "address": {
      "prefix": "Mr",
      "first_name": "John",
      "last_name": "Smith",
      "contact_number": "08000850885",
      "house_name": "Brunswick Court",
      "house_number": "1",
      "street": "Brunswick Street",
      "address2": "Newcastle-under-Lyme",
      "postcode": "ST5 1HH",
      "region": "Postal County",
      "city": "Big City"
    }
  }
}
```

6.3.6 Sync Retailer Order

POST /api/syncretorder

From: Retailer E-Commerce Platform to DivideBuy

Request

Variable Name	Type	Length	Valid Value	Description
storeOrderId	Integer	10		
storeToken	String	255		
storeAuthentication	String	255		

Example Request:

```
{
  storeOrderId: "6667",
  storeToken: "5LIH1TaW8ewd",
  storeAuthentication: "3aa7Sgt76sz7"
}
```

Response

Variable Name	Type	Length	Valid Value	Description
status	String		ok	
error	Boolean		1 | 0	1 = Error; 0 = No Error

Example Response:

```
{
  "status": "ok",
  "error”: 0
}
```

6.4 Order Lifecycle

This API is used by DivideBuy to communicate with the Retailers E-Commerce Platform.

6.4.1 Refund

POST /api/refund

From: Retailer E-Commerce Platform to DivideBuy

Request

Variable Name	Type	Length	Valid Value	Description
product				
• sku	String	255		
• productName	String	255		
• qty	Integer	11		
• rowTotal	Decimal	14,3		
• rowInclTotal	Decimal	14,3		
totalRefund	Decimal	14,3		
orderId	Integer	20		
reason	String	255		
shippingCostAmount	Decimal	14,3		
shippingTaxAmount	Decimal	14,3		
adjustmentRefund	Decimal	14,3		
adjustmentFee	Decimal	14,3		
taxAmount	Decimal	14,3		
refundType	Enum		FULL | PARTIAL	
retailer				
• retailerId	Integer	10		
• storeAuthentication	String	255		
• storeToken	String	255		

Example Request:

```
{
   product: [
     {
       sku: "abl003",
       productName: "Broad St. Flapover Briefcase",
       qty: 20,
       rowTotal: 11400,
       rowInclTotal: 13680
     }
   ],
   totalRefund: 13685.99,
   orderId: "6667",
   reason: "",
   shippingCostAmount: 5.99,
   shippingTaxAmount: "5.99",
   adjustmentRefund: 0,
   adjustmentFee: 0,
   taxAmount: 2280,
   refundType: "FULL",
   retailer: {
     retailerId: "150",
     storeAuthentication: "3aa7Sgt76sz7",
     storeToken: "5LIH1TaW8ewd"
   }
}
```

Response

Variable Name	Type	Length	Valid Value	Description
status	String		ok	

Example Response:

```
{
   "status": "ok"
}
```

6.4.2 Tracking

POST /api/tracking

From: Retailer E-Commerce Platform to DivideBuy

Request

Variable Name	Type	Length	Valid Value	Description
retailerId	Integer	10		
storeOrderId	Integer	10		
storeToken	String	255		
storeAuthentication	String	255		
deleteTracking	Boolean		1 | 0	1 = Delete; 0 = do not delete
trackingInfo				
• trackNumber	String	255		
• description	Text			
• title	String	255		
• carrierCode	String	255		
productDetails				
• sku	String	255		
• qty	Integer	11		

Example Request:

```
{
   retailerId: "150",
   storeOrderId: "6667",
   storeToken: "5LIH1TaW8ewd",
   storeAuthentication: "3aa7Sgt76sz7",
   deleteTracking: 0,
   trackingInfo: [
     {
       trackNumber: "123456789",
       description: null,
       title: "DHLeCommerce",
       carrierCode: "dividebuy_custom"
     }
   ],
   productDetails: [
     {
       sku: "abl003",
       qty: 1
     }
   ]
}
```

Response

Variable Name	Type	Length	Valid Value	Description
status	String		ok	
error	Boolean		1 | 0	1 = Error; 0 = No Error

Example Response:

```
{
  "status": "ok",
  "error”: 0
}
```

6.4.3 Delete Order

POST /api/response

From: DivideBuy to Retailer E-Commerce Platform

Request

Variable Name	Type	Length	Valid Value	Description
method	String		orderDelete	
store_order_id	Integer	20		
retailer_store_code	String	255		
store_token	String	255		
store_authentication	String	255		
delete_user_order	Boolean		1 | 0	

Example Request:

```
{
   "method": "orderDelete",
   "store_order_id": 6667,
   "retailer_store_code": "default",
   "store_token": "5LIH1TaW8ewd",
   "store_authentication": "3aa7Sgt76sz7",
   "delete_user_order": 1
}
```

Response

Variable Name	Type	Length	Valid Value	Description
error	Boolean		1 | 0	1 = Error; 0 = No Error
success	Boolean		1 | 0	1 = Request Successful; 0 = Request Not Successful
status	String		ok	

Example Response:

```
{
   "error": 0,
   "success": 1,
   "status": "ok"
}
```

6.5 Algorithm to Generate the Splash Key

When a user is redirected to a new page after placing an order, a Splash Key is sent along with the URL e.g. https://api.dividebuy.co.uk/redirectUrl?splashKey=[ValueOfSplashKey]

The Splash Key is generated using a combination of the **tokenNumber, orderId, authenticationNumber** and **retailerId (base64 encode(tokenNumber:orderId):base64 encode(authenticationKey:retailerId))**, and then passing this to be **base64 encoded**.

Example:

orderId: **6667**

retailerId: **150**

tokenNumber :**5LIH1TaW8ewd**

authenticationKey :**3aa7Sgt76sz7**

Value to be encoded:

base64_encode(base64_encode(5LIH1TaW8ewd:6667):base64_encode(3aa7Sgt76sz7:150))

Resultant Splashkey (base64 encoded value):

TlV4SINERlVZVmM0WIhka09qWTJPVFE9Ok0yRmhOMU5uZERjMmMzbzNPakUxTUE9PQ==

Please note:

- The ***retailerId*** can be obtained from 'Confirm Retailer' API response.
- The ***orderId*** will be generated by the retailer store and passed as a ***store_order_id*** in a successful, cancelled or deleted order APIs.

7.0 Contact Us

We trust that this document will provide you with all the information you require during the process. However, if you do require any further information, please contact your Business Development Executive, or for technical assistance, please email us at onboarding@dividebuy.co.uk.

8.0 Summary

This document provides a full overview of DivideBuy's flows with the E-Commerce Platform and provides a detailed view of the API communication flows. This document will allow the Retailer to integrate directly with DivideBuy's systems so that they are able to offer their customers Interest-Free Credit via our services.