About the SPACE800X business model

SPACE800X Smart B2B Market (SMARKT) is an intermediary platform that offers companies within the EU a centralised platform for the combined purchase and sale of products, services, training and events via its own custom-built multi-vendor marketplace.

Target group

- Small to large companies that want to offer and procure their services and products efficiently.
- Service providers such as consultants or trainers who want to offer their services via a transparent platform.
- Processing only within the internal market of the European Union

USP (unique selling proposition)

1. Holistic approach: combination of technology, training and services in a single marketplace.
2. Multi-vendor functionality: Buyers can purchase products from different suppliers at the same time.
3. Automated invoicing: Correct invoicing under tax law including reverse charge procedure.
4. Cost structure: Suppliers bear all third-party fees alone, while buyers purchase free of charge and Space800x remains unencumbered.
5. Integrated processes: Automatic invoicing for buyers, suppliers and SPACE800X.
6. Fee structure: providers bear all fees; buyers trade free of charge.

Main features of the platform

- User profiles:
    - One user (one email) can manage multiple companies with separate PayPal accounts.
    - A user can have several companies. The main company (first company) is assigned to the user's e-mail. All other companies must have a different e-mail address.
    - Each company has an individual profile and can act as both a buyer and a supplier.

- Security mechanisms:
    - Suppliers must validate their companies once with Space800x and PayPal before products are available online.

    - Buyers do not need a separate validation, but have a PayPal Connect account that should be created for them automatically.

- Transparency:
    - Buyers receive legally compliant invoices directly from the suppliers.
    - Providers receive detailed invoices for PayPal fees from PayPal (if possible or included in the SPACE800X commission invoice).
    - Providers receive a commission invoice

Technical Guide - Stripe Tax API Integration for SPACE800x

Technical background information:

Bilingualism of the system

- PO-Editor is a suitable choice for translation management. It is important to use the Django internationalisation functionality (django.utils.translation) consistently.

Technological basis

- Programming language: Python 3.13
- Framework: Django 5.2
- Frontend: Bootstrap 4
- APIs: PayPal Connect API, Stripe Tax

General requirements for implementation:

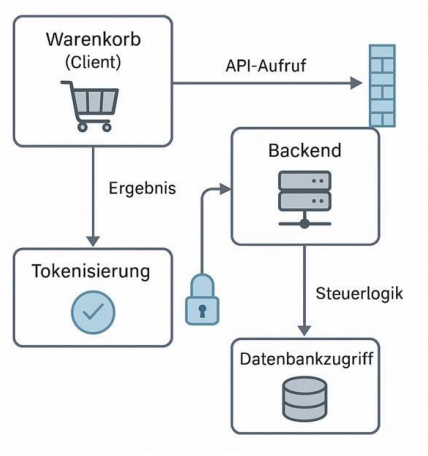
- We need a quick and cost-effective solution that can be implemented in practice. In addition, we would like a recommendation for a variant/action.
- The solution should be pragmatic, lean and free of unnecessary complexity or high costs.
- Priority is given to a clean implementation of the points mentioned with a structured but not overly complex architecture.
- Only successful implementation is remunerated - not the attempt at implementation.

1. aim of this document

This guide describes the technical architecture and procedure for the secure integration of Stripe Tax into the SPACE800x multivendor B2B platform. The aim is to provide external developers with precise instructions on which tasks are to be implemented - in strict compliance with our security architecture and the responsibilities defined here.

2. architectural overview

VAT is calculated exclusively on the server side. The shopping basket (client) communicates with secure API endpoints. The internal control logic, (de-)tokenisation and sensitive data remain completely encapsulated in the SPACE800x backend.



3. roles & responsibilities

- **SPACE800x team:** Responsible for the overall architecture, API specifications for internal services, tokenisation logic, security rules, database management, implementation of core control logic, VIES validation, e-invoicing (XRechnung), cronjobs for tax codes and cost monitoring, as well as all aspects of PayPal integration (by "Ronny").
- **External developer (Ivan):** Only implements the API endpoints clearly assigned in section 5 according to the interfaces defined here. Does not interfere with the control logic, database structure (except for defined read access) or core business logic.
- **Main developer SPACE800x:** Review, acceptance and merge of the components provided by the external developer into the production system.

4. technical specifications & architectural principles

- Tax calculation and logic (including calls to the Stripe Tax API tax.calculate and tax.transaction.create) take place exclusively in the SPACE800x backend and are **not** the responsibility of the external developer.
- Communication of the API endpoints to be created by the external developer takes place via secure mechanisms with authentication and IP filter (details are to be provided by the SPACE800x team and implemented by the external developer).
- **Tokenisation:** Control codes, buyer IDs and vendor IDs are tokenised internally by SPACE800x before they are transferred to the APIs to be implemented by the external developer. The (de-)tokenisation logic is internal to SPACE800x.
- **Database access by the external developer:** Read-only access to predefined tables (e.g. product categories, tax tokens, if required for the API implementation and explicitly released by SPACE800x). No direct write access to the database. No access to core database tables or business logic.
- All API keys (Stripe, etc.) are managed internally by SPACE800x on the server side and are not accessible to external developers.

5. implementation steps for the external developer

The external developer is responsible for implementing the following API endpoints and process steps:

5.1 Implement API endpoint /api/calculate-tax

- **Purpose**: Receives shopping basket data and initiates the server-side tax calculation by calling an internal SPACE800x service.
- **Input (request body, e.g. JSON):** The exact data structures must be specified by SPACE800x.
  - shopping basket_id: Unique ID of the current shopping basket.
  - kaeufer_id_token: Tokenised buyer ID.
  - items: Array of shopping basket items, each with:
    - product_id_token: Tokenised product ID.
    - quantity: Quantity.
    - vendor_id_token: Tokenised vendor ID for this position.
  - *(SPACE800x team must clarify whether further data such as delivery address tokens or similar are required, or whether these are resolved on the server side via kaeufer_id_token).*
- **Processing:**
  1. Validation of the input data.
  2. Calling an **internal** service/function provided by SPACE800x (e.g. internal_control_calculation_request(data)). The exact interface (endpoint, parameters, authentication) to this internal service must be defined by SPACE800x.
  3. The external developer **does not** call the Stripe Tax API directly.
- **Output (response body, e.g. JSON):** The exact data structures must be specified by SPACE800x.
  - total_total_incl_tax_per_vendor: Object or array that shows the calculated taxes and the total amount for each (tokenised) vendor ID (this data comes from the internal SPACE800x service).
  - platform fees: Platform fees have to be calculated, these are 10% of the net turnover plus 10 Euro per transaction.
- **Security:** Implementation of the authentication and IP filter mechanisms specified by SPACE800x for this endpoint.

5.2 Prepare data transfer for PayPal checkout

- **Purpose: To** provide the necessary amount information for the PayPal checkout process for which Ronny (SPACE800x team) is responsible.
- **Processing:** The tax amounts, total amounts per provider and platform fees received from the /api/calculate-tax endpoint (or the internal SPACE800x service) must be transferred to the PayPal process in a format defined by SPACE800x (Ronny) and via a defined interface.

- The exact type of this transfer (e.g. direct function call, internal event, temporary storage with read access for Ronny's process) must be specified by SPACE800x.

5.3 Implement API endpoint /api/finalise-transaction (or similarly named)

- **Purpose**: Is called after a successful PayPal payment by a PayPal webhook configured by SPACE800x (Ronny) to register the transaction with Stripe Tax (via internal service).
- **Input (request body, e.g. JSON, from the PayPal webhook):** The exact data structures are to be specified by SPACE800x (Ronny).
  - paypal_transaction_id: The transaction ID from PayPal.
  - shopping_basket_id or space800x_order_id: A reference to the original order/shopping basket in SPACE800x.
- **Processing:**
  1. Validation of the input data and the webhook signature (if required by SPACE800x and mechanism provided).
  2. Call of an **internal** service/function provided by SPACE800x (e.g. interne_stripe_transaktion_melden(daten)). This internal function is responsible for the actual tax.transaction.create call at Stripe. The exact interface must be defined by SPACE800x.
  3. The external developer **does not** call the Stripe Tax API directly.
- **Output (response to the PayPal webhook):**
  - HTTP 200 OK if the call to the internal service was successful.
  - A corresponding HTTP error code if the request could not be successfully forwarded to the internal service.
  - (The detailed error handling of the tax.transaction.create call itself, including retries and DLQ, is the task of the internal SPACE800x logic).
- **Security:** Implementation of the security mechanisms specified by SPACE800x for this webhook endpoint.

5.4 Compliance with all API security rules

- Strict compliance with the safety guidelines specified by SPACE800x.
- In particular, **no logging** of sensitive data (payloads, tokens, personal information) by the components created by the external developer.

6. notes on collaboration & approval process

All results of the external developer are checked and approved by the internal lead developer (Rinny) of SPACE800x. The external developer works exclusively on the clearly documented API endpoints and interfaces assigned in Section 5. Changes to the data model, the authentication for internal systems or the core business logic are not permitted and are not part of the contract.

7. time frame & effort estimate (for the external developer)

The implementation of the tasks described in section 5 or in the "Guide" section for the external developer (API endpoints /api/calculate-tax, /api/finalise-transaction, preparation of data transfer to PayPal) is scheduled for a maximum of 2 working days (approx. 12-16 hours) plus 1 day for the two XInvoices plus 0.8 days buffer and 0.2 days for setting up the Stripe tax function for SPACE800X and providers (maximum 32 hours in total).

The basis for this is that:

- All database tables that may need to be read-accessed (product categories, tax tokens) already exist and are defined by SPACE800x.
- The complex internal control logic services that are called by the external API endpoints are provided by SPACE800x and their interfaces are clearly specified.
- The external developer does not design its own architecture or make tax decisions, but simply implements the API requests and responses correctly according to specification.

---

*The following sections are taken from the original document "Leitfaden_STRIPE_TAX_Implementierung_Steuern_e-Invoices_Updated.docx" and serve as background information. The processes described therein are to be understood as the **internal responsibility of SPACE800x**, unless explicitly mentioned in section 5 or section "Guidelines" as the task of the external developer, and are implemented in accordance with the above principles.*

Task 1: Ivan alone

Onboarding for Stripe Tax application (internal process SPACE800x)

Stripe Tax can also be used without Stripe payment processing. However, this requires a "short onboarding" of all our users to perform the tax configuration. We can perform tax calculations via the API and must manually inform Stripe of completed transactions.

Automated onboarding for Stripe Tax (internal process SPACE800x)

In order to automatically integrate the providers into Stripe Tax, onboarding takes place during the second registration process on our platform and when the user adds (an) additional company(ies) to their profile with an additional email address. The process is as follows:

1. **Create company (registration)**
   - When creating a new company, the user enters all relevant information: Name, address, email per company, tax ID and product categories.
   - This data is stored in our database.
2. **Automatic creation in Stripe Tax (internal call via backend logic)**
   - After successful registration, our system (backend logic) sends the tax data to Stripe Tax via the API (tax.settings.update).
   - If necessary, the provider can later update its control configuration via a separate interface.
3. **Additional companies via the company creation profile (SPACE800x internal process)**
   - If a user creates another company, the same tax information is requested.
   - After completing the entry, the company is automatically registered in Stripe Tax (via backend logic).
4. **Validation of the control information (internal process SPACE800x, if necessary with VIES)**
   - If a VAT ID is stored, Stripe can perform a validation. VIES validation is also carried out by SPACE800x.
   - If no VAT ID number is available, the tax is calculated according to standard guidelines.
5. **Assignment of product control codes (internal process SPACE800x)**
   - Suitable tax classes are assigned based on the selected product categories and the internal mappings to Stripe Tax Codes.
6. **Notification and activation (internal process SPACE800x)**
   - After successful setup, the user receives a confirmation.
   - They can adjust tax settings in their dashboard if necessary.d

Background information: Ivan and Ronny

Integration process

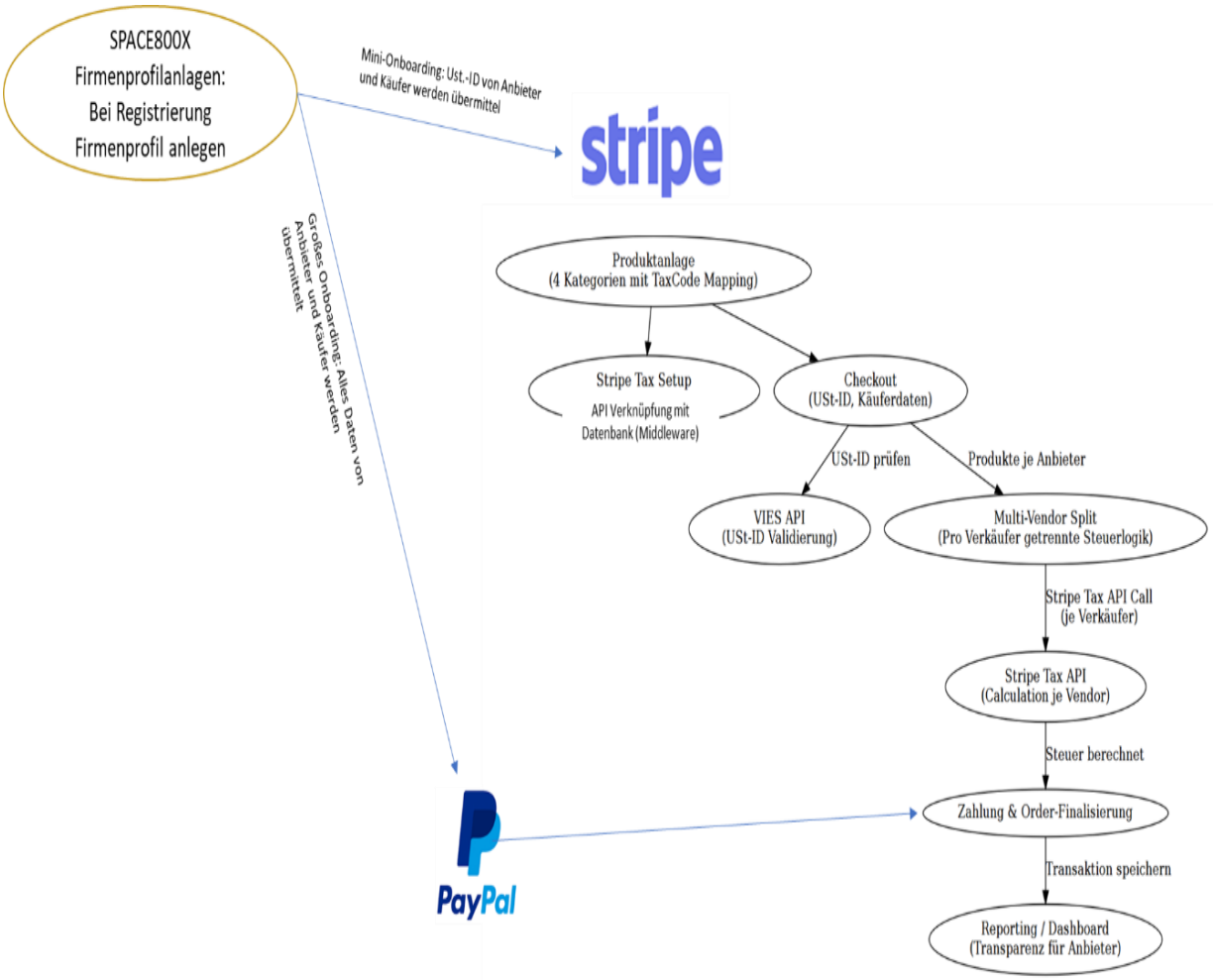
1. **Tax calculation in the shopping basket (external API call to /api/calculate-tax, internal logic executes Stripe call)**
   - When a customer adds products to the shopping basket, the frontend calls the /api/calculate-tax endpoint created by the external developer. This forwards the request to the internal SPACE800x tax logic, which calculates the taxes using Stripe's tax.calculate API.
   - This will not incur any costs as long as we stay under the 10 included API calls per transaction. Ivan please check this again!!!!!
2. **Payment via PayPal (Internal process SPACE800x - Ronny)**
   - The customer selects PayPal as the payment method and completes the payment there.
   - PayPal processes the payment.
3. **Manual message to Stripe after successful payment (PayPal webhook to external API call /api/finalise-transaction, internal logic executes Stripe call)**
   - As soon as the payment is successful, PayPal sends a webhook to the endpoint /api/finalise-transaction created by the external developer.
   - This endpoint forwards the request to the internal SPACE800x control logic, which reports a completed transaction to Stripe via the tax.transaction.create API.
   - Only at this point are costs incurred for Stripe (e.g. €0.45 per transaction, as of May 2025).
4. **Billing by Stripe (internal process SPACE800x)**
   - Stripe issues a separate invoice to SPACE800x for the use of Stripe Tax.
   - The fees are only charged for reported transactions and are VAT-exempt.
   - In order to pass on the costs to the providers, SPACE800x implements rebilling in its own system.

Costs (information for SPACE800x):

- Per completed transaction with tax calculation via tax.transaction.create: e.g. €0.45 (as of May 2025, check price from Stripe).
- Additional API calls for tax.calculate: Each transaction contains a certain number of included API calls (e.g. 10), after which further calls may cost (e.g. €0.04 per API call, as of May 2025, check price from Stripe).
- The Stripe costs must be checked regularly by SPACE800x and the internal calculations adjusted if necessary. The internal cron job for counting the API calls is used for cost control and recalculation.

Important findings (For SPACE800x)

- No automatic connection to PayPal: Stripe does not automatically learn of a successful payment and relies on the message from SPACE800x.
- Honest reporting required: SPACE800x is contractually obliged to inform Stripe about completed transactions.
- No fees for non-completed purchases: If a customer does not complete the purchase but a tax calculation (tax.calculate) has been made, no costs are incurred as long as the exemption limit for API calls is not exceeded.
- Cost transfer to provider: Must be implemented internally by SPACE800x.



Implementation of the project (task)

Guide to the implementation of tax calculations, e-invoices and PayPal integration (background and internal decisions SPACE800x)

Your main tasks:

1. Create API endpoint /api/calculate-tax.
2. Prepare data for PayPal checkout.
3. Create API endpoint /api/finalise-transaction.
4. Observe the safety rules.

Overview of your overall project:

1. Implement internal onboarding for Stripe Tax (via tax.settings.update).
2. Define interfaces for external API endpoints (/api/calculate-tax, /api/finalise-transaction) and internal control logic services.
3. Have external API endpoints implemented by external developers.
4. Implement internal tax logic services (calls to tax.calculate, tax.transaction.create, error handling).
5. PayPal integration (webhook configuration) by Ronny.
6. Implement VIES validation.
7. Implement XInvoice creation and dispatch.
8. Develop a mechanism for passing on costs to providers.
9. Implement cron jobs for tax code management and cost monitoring.
10. Testing of the entire workflow and verification of billing by Stripe.

Tools overview

1. stripe tax

Stripe Tax is an API service that enables tax calculations based on the VAT IDs of sellers and buyers. It offers support for the EU reverse charge procedure and can include product categorisations in the tax logic.

Compatible functions

- Tax calculation based on VAT IDs: The API enables the precise calculation of taxes according to buyer and seller data.
- Product categorisation: Different tax rates can be applied based on product categories (e.g. digital products vs. physical goods).
- Reverse charge procedure: Tax logic is automatically adjusted in accordance with EU B2B rules.
- Flexibility for multi-vendor scenarios: The API enables the calculation of tax rates for multiple vendors within a shopping basket.

www.stripe.com

2. e-Invoice solution (internal implementation SPACE800x)

Requirement: XRechnung is to be used for e-invoicing. **Implementation:** SPACE800x selects a suitable Python-based open source library (e.g. extensions for PyUBL, or others that can generate XRechnung-compliant XML files) and implements the logic for creating the legally compliant XRechnung invoices. The invoices are stored in the SPACE800x database and sent to users in the dashboard and by e-mail.

API link

- ubl2 Python Library
- xmlschema for XML processes
- **GitHub:** https://github.com/h3/django-simple-invoice
- **GitHub:** https://github.com/lincolnloop/django-invoice

If another open source or free e- or X-Invoice solution is preferred, or if there is positive experience with an existing solution, we are open to this. We are not committed to the proposed solution.
The only important thing is that there are no ongoing costs in order to keep the costs for our customers as low as possible and that all requirements are met (logo, invoice number, address, VAT ID no., etc.).

3. use of the VIES API (internal implementation SPACE800x)

The VIES API is used to validate VAT IDs and is essential for the reverse charge procedure. The API checks the validity of the buyer's VAT ID. **Implementation:** SPACE800x implements the VIES API query (a SOAP service, e.g. using the zeep Python library) in the shopping basket process before the final tax calculation. The results flow into the tax logic. If the buyer's VAT ID number is invalid, the purchase process is stopped and the buyer is asked to correct it. In the event of problems with seller VAT ID numbers (e.g. during onboarding), the seller is informed.

API link

VIES API

https://ec.europa.eu/taxation_customs/vies/

4. integration into the PayPal checkout (internal implementation SPACE800x - Ronny)

The tax calculation and validation of the VAT IDs must be seamlessly integrated into the PayPal checkout. **Implementation:** Ronny (SPACE800x team) is responsible for the PayPal integration. This includes the configuration of the PayPal webhook, which calls the endpoint /api/finalise-transaction created by the external developer after successful payment.

5. prices (information for SPACE800x)

- **Stripe Tax:** Costs per transaction (e.g. €0.45 for tax.transaction.create) and possibly for additional tax.calculate calls. Prices must be verified with Stripe.
- **VIES API:** Free of charge.
- **XInvoices:** Open source, no licence costs for the library itself.

Step-by-step guide for development work

Overview of the steps for integration

1. Step: Data transfer to the API:

- Submit the following information to the Stripe API:
  - Buyer details (incl. VAT ID).
  - Seller details (incl. VAT ID).
  - Product details (category, price, quantity).
  - Shipping and billing addresses.

2. Step: Validation of the VAT IDs:

- Use the VIES API to ensure the validity of the VAT IDs.
- Transfer the validated IDs to the Stripe API for tax calculation.

3. Step: API call for tax calculation:

- Call the Stripe API with the data provided to determine the tax rate or reverse charge procedure.

4. Step: Integration into the shopping basket:

- Update the shopping basket display with the calculated taxes for each vendor.
- Save the calculated taxes for later invoicing.

5. Step: Checkout integration

- The tax calculation and validation of VAT IDs must be seamlessly integrated into the PayPal checkout:

1. implement a webhook to receive transaction details from PayPal.
2. pass the buyer and seller data and product details to the Stripe API
3. integrate the calculated taxes into the checkout flow and the final payment display
4. generate the e-Invoice based on the calculated tax data.

Task 2: Ivan and Ronny

Stripe tax calculation, multi-vendor shopping cart and checkout transfer to PayPal in Python Django

1. requirements and objectives

- Automation of tax calculation for various product types (seminars, downloads, services, technologies).
- Support for multi-vendor shopping baskets with individually calculated tax rates per vendor and product.
- Integration of Stripe for:
  - VAT ID verification for suppliers and buyers.
  - Automated tax calculation based on product categories and product types, countries and B2B rules.
  - Updating of tax codes in the event of changes by Stripe.
  - Integration of PayPal for the checkout.
  - Creation and management of invoices for:
    - customers (incl. VAT).
    - provider (with deduction of platform fees and VAT).

2. expansion of the database

A. Customisation of the database tables

The database must be expanded so that the Stripe tax codes can be integrated directly via the Stripe API and linked to the existing product types. Middleware is not necessary as the data is imported **once** during the initial call or targeted refresh.

In concrete terms, this means

- The existing four database lists of product types (per product group) are to be expanded to include a field for storing the assigned **stripe tax code**.
- The complete list of available **Stripe tax codes** is retrieved via the API (https://docs.stripe.com/api/tax_codes/list) and stored in a separate database table.
- When creating a product, the link is made via a search and selection function in which the user selects the appropriate control code from the Stripe list.
- The information on the description and category of the tax codes (e.g. "Digital Document Download") is also stored, **but not the technical Stripe ID**, as this is not relevant for the end user.

- The link to the API documentation (https://docs.stripe.com/tax/tax-codes?locale=de-DE&tax_code=Paperwork) serves as a reference for the developers for orientation during implementation.

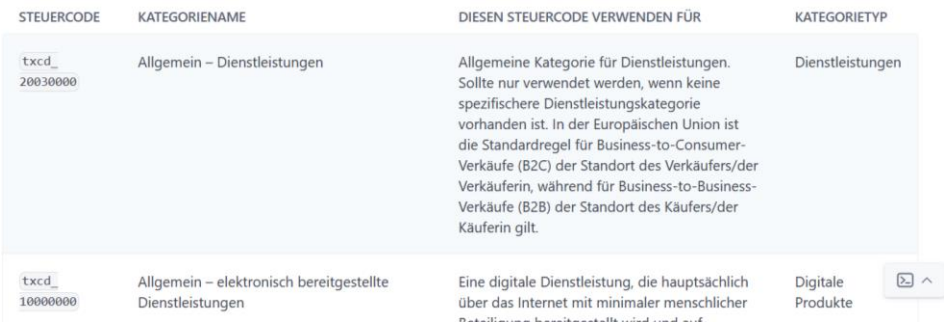
The aim is to carry out the tax calculation in the shopping basket in a clean and legally compliant manner based on the tax codes selected by the user - without manual maintenance and without static Excel assignments.

Our product categories are:

- Seminars (e.g. of the product type online/presence)
- Documents (e.g. of the product type patents, reports)
- Technology products (e.g. the product type agricultural robots, AI)
- services

Structure of the data structure:

- Create a table and adopt the structure of Stripe



STEUERCODE	KATEGORIENAME	DIESEN STEUERCODE VERWENDEN FÜR	KATEGORIETYP
txcd_20030000	Allgemein – Dienstleistungen	Allgemeine Kategorie für Dienstleistungen. Sollte nur verwendet werden, wenn keine spezifischere Dienstleistungskategorie vorhanden ist. In der Europäischen Union ist die Standardregel für Business-to-Consumer-Verkäufe (B2C) der Standort des Verkäufers/der Verkäuferin, während für Business-to-Business-Verkäufe (B2B) der Standort des Käufers/der Käuferin gilt.	Dienstleistungen
txcd_10000000	Allgemein – elektronisch bereitgestellte Dienstleistungen	Eine digitale Dienstleistung, die hauptsächlich über das Internet mit minimaler menschlicher	Digitale Produkte

- Create a table containing the following columns:

Product category (Smarkt)	**Stripe control code**	**Category name**	**Use this control code for**	**Category type (stripe)**

- Insert a column before the Tax code column. This first column must be based on the lists of product types per product category.

Data integration:

- Implement an interface to the Stripe API to retrieve the current control codes and import them into the table.
- Attention: Create a table once for the German side and once for the English side. Create Pos accordingly and make it available to me. Translations will be done by Space800x.

Add filters to the product form:

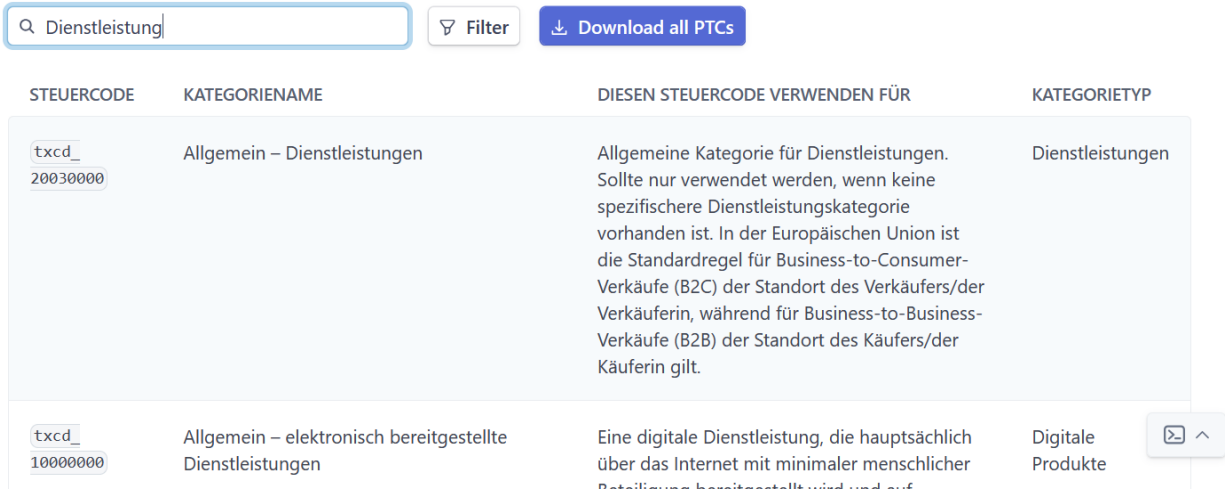
- Each product form is internally assigned a fixed Stripe category:
  - ➔ Events -> Training
  - ➔ Service ->Services (Products called in our System!!!!)
  - ➔ Tec product -> Digital products, physical goods
  - ➔ Resources -> Digital products
- This category is automatically set as a filter value for each creation form as soon as the form is loaded.
- Each product form has a specific control code preset as standard, which must be confirmed by the customer or can be changed. Here you will find the relevant ones for all four product groups:

Service	Product	txcd_20030000	General category for services. Should only be used if no more specific service category exists. In the European Union, the default rule for business-to-consumer (B2C) sales is the location of the seller, while for business-to-business (B2B) sales it is the location of the buyer.
Technologies (no software)	Physical goods	txcd_99999999	"physical", "A physical good that can be moved or touched. Also referred to as tangible personal property.", "General - Tangible goods"
Training courses	Event	txcd_20060044	A payment for training in which the buyer receives instructions. This includes training courses or workshops, but excludes physical exercises and workouts.

Resources	Digital download	txcd_10503000	Digital or other messages or documents (download, without subscription) with permanent transfer of rights

Search function:

- Develop a search function that allows customers to search for specific tax codes per product category. The search results should then automatically display the appropriate tax code in the table.



Important: As with Stripe, the description and the official name of the code should always be displayed in addition to the selection (e.g. 'Digital Document Download'). The actual Stripe tax code ID (e.g. txcd_10503000), on the other hand, should not be visible as it is not relevant for the user and could only be confusing.

- Manual assignment:
  - If the default setting does not match, the customer can manually select the appropriate tax code for their product type from the drop-down list.
  - It must be possible to restore the default settings for each product creation form if nothing suitable is found in the list.

B. Extension of the product creation form (Ivan)

- Each product form has a specific tax code as a default setting, which must be confirmed by the customer or can be changed.
- Search function with which customers can search for specific tax codes per product category

- The product categories for each product creation form can be selected from a drop-down list (data from the ProductCategory table).
- The four product forms are extended so that the corresponding Stripe tax codes can be assigned according to the product types using a drop-down function.
- The tax code is automatically linked and saved based on the category.
- The selected tax codes per product type are saved in the database to enable the tax calculation in the shopping basket.
- A short text should be displayed below the selection to help you. Please create a Po text field for this .Name the Po: Selection_Control codes per product type:

  A quick search function is available to you. The available product control codes are described in the list displayed. If no entry applies exactly to your product, please select one whose description most closely matches your product and contains the term "General". Optionally, you can also select the default setting if this is suitable for your product. FYi: You are solely responsible for making the correct tax allocation. SPACE800X accepts no liability for correctness.

C. Technical realisation

The lists for product types already exist as bilingual (German/English) dropdown functions. However, the descriptions and the link to the Stripe tax codes and Stripe product categories are still missing.

Tasks:

- Link the Stripe tax code list via APIs
- Make sure that the lists are synchronised with the complete Stripe Tax Code list.

API integration:

The Stripe API is used to regularly update the tax codes.

E. Stripe API integration

1. Stripe API Documentation
   - Entry point for all Stripe APIs.
2. List Tax Codes API
   - Retrieve the complete list of current tax codes.
3. Stripe Tax Code Search Tool
   - Online tool for direct search for tax codes.

Notes on API usage:

- Sandbox environment: https://api.stripe.com (for tests)
- Production environment: https://api.stripe.com (for live applications)

Resources:

- Stripe Tax API: Stripe Tax API Documentation

API reference: Stripe API reference

Task 3: Ivan

Multi-vendor shopping basket

1st settlement

- **Separate billing for each provider**: Each supplier in the shopping basket is billed separately.

---

2. tax calculation with Stripe Tax

2.1 Principles of tax determination

- **Automatic tax calculation by Stripe** based on:
  - Product category / product type (see Task 2)
  - Location of supplier and buyer
  - Existence and verification of the VAT ID (B2B)
  - Reverse charge procedure (if applicable)

2.2 Tax calculation in the shopping basket (tax.calculate)

- **Time**: Tax calculation takes place before the start of the payment process.
- **Data sources**:
  - Buyer data: Delivery address, VAT-ID
  - Seller data: Registered office, VAT-ID
  - Product data: Quantity, price, stripe tax code
- **Processing**:
  - The API response from tax.calculate is displayed in the shopping basket.

3. VAT-ID verification

3.1 When registering / creating a company

- Buyers and suppliers enter their VAT ID when creating the main company or additional companies.
- **Stripe automatically checks the validity via API**.
- **Invalid or expired VAT-IDs**:
  - **Buyer**: Error message in the shopping basket with a note for correction in the company profile (incl. direct link).
  - **Provider**: E-mail notification with request for correction. Products are temporarily switched offline until a valid VAT ID has been entered and validated by Stripe. (Must still be implemented, separate ticket required.)

3.2 Time of verification

- The VIES validation of the buyer VAT ID takes place **in the shopping basket**, **before the final tax calculation with Stripe Tax**.

Integration with Stripe

- **API calls:**
  - **Tax Code Listing API:** To retrieve the latest tax codes and categories.
  - **VAT-ID Validation API:** Verifies the VAT ID entered.
  - **Transaction Tax API:** Calculates the tax based on the shopping basket data.

- **Data structure for the API call:**
  - type: "SalesOrder"
  - companyCode: "YourCompanyCode"
  - lines: List of products (SKU, quantity, price, tax category)
  - customerCode: Customer number
  - addresses: Billing address information
  - e-mail: E-mail address per company of the supplier/buyer

Attention: Delivery address is currently not queried. We do not want this query either, suppliers only deliver to where the billing address is in order to minimise fraud. The buyer and supplier VAT ID, as well as the billing address, are already queried by the system. As far as I know, we do not issue a customer number in

the traditional sense, but it should be numbered consecutively, which can be used as a customer code.

- **Stripe API:** The CreateTransaction API is used to calculate the sales tax.

Parameter for the API request:

- Buyer information: VAT ID of the buyer
- Supplier information: VAT ID of the provider
- Product details:
  - Tax group (tax code)
  - Price
  - Quantity

Task 3: Ronny and Ivan (togther, whir Ivan takes vare for everything with stripe and Ronny with Paypal)

Checkout with PayPal

- After Stripe has calculated the taxes, the total amount is transferred to PayPal. Use the PayPal API.
- Use of the PayPals REST SDK for Python.

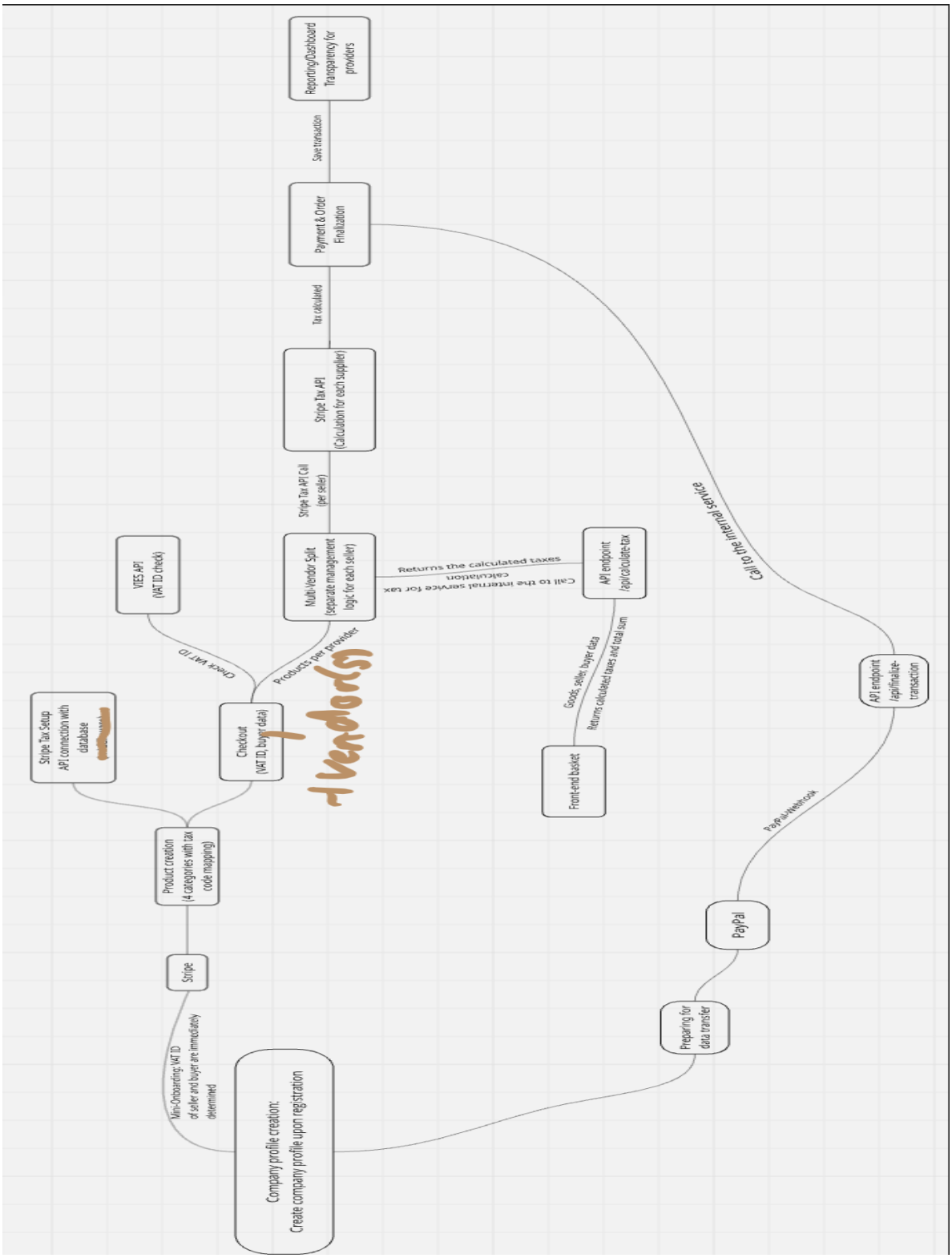
- Transfer of the calculated taxes and platform fees to PayPal.
- After successful payment: **Manual message to Stripe tax.transaction.create)**:

  - **Trigger:** A PayPal webhook signals a successfully completed payment. This webhook is configured by "Ronny" and calls an endpoint of the module created by the external developer.
  - **Data for the report:** The PayPal transaction ID (supplied by PayPal Webhook) and other necessary reference data for the transaction (from your database, e.g. the ID of the original tax calculation or shopping basket ID).
  - **Error handling (proposal)**:
    - **Logging:** Every attempt and every result of the tax.transaction.create call must be logged in detail.
    - **Retry mechanism:** In the event of failures (e.g. network problems, temporary Stripe API inaccessibility), an automatic retry mechanism with exponential backoff should be implemented (e.g. after 1 min, 5 min, 15 min, 1 hour).
    - **Dead letter queue (DLQ) / manual intervention:** After a defined number of unsuccessful retries (e.g. 3-5 attempts), the failed message should be moved to a separate queue or table (a type of DLQ) and a notification

sent to an administrator to enable a manual check and, if necessary, correction and re-reporting.

- **Idempotency:** Make sure that tax.transaction.create calls can be handled idempotently if Stripe supports this (by providing a unique idempotency ID per transaction message) to avoid duplicate messages on retries.
  - **Charging costs to providers:** Our internal system is responsible for this. The module created by the external developer (Ivan) must store the necessary data (which transaction, which provider, which Stripe costs) in the database in such a way that your system can retrieve it for the recalculation.
  - **Updating the Stripe costs (cron job)**:

    - **Responsibility:** The external developer (DevOps) is responsible for creating, deploying and operating the cron job.
    - **Purpose of the cron job:** The prices for Stripe Tax ($0.50 per tax.transaction.create API call, $0.05 per tax.calculate API call, or 0.5% of the transaction volume if Stripe is used for payment processing and tax collection in registered countries - which does not seem to be the case here, as PayPal is used) are currently documented on the Stripe website. There is no known Stripe API to dynamically query these *base prices*. Stripe usually communicates price changes directly.
    - The cron job should therefore primarily be used to count the *number* of tax.calculate and tax.transaction.create calls per provider and save them in the database. The costs per provider can then be calculated based on these counts and the known (and stored in your system) Stripe fee rates.
    - A manual check and, if necessary, adjustment of the Stripe fee rates stored in the system is necessary if Stripe changes its prices.



Please take care fpr the note of Ronny in Slack due to this chart!

Task 4: Ivan

Invoicing

- Generation of an invoice for the provider (with deduction of the platform fee and taxes). Including taxes or reverse-charge for Space800x. Paypal and Stripe are financial service providers and therefore exempt from VAT.

- **Decision:** XRechnung is required. A suitable open source library (e.g. taking into account PyUBL or alternatives that can generate XRechnung) must be selected. Mustangproject (Java) would only be an option via an API integration, which could mean additional complexity. Check Python-based solutions for XRechnung.
- **Data for invoicing:** All necessary data is obtained from your database.
- **Storage and delivery**:

    - Generated XInvoices are saved in a defined format (e.g. XML) in a table in your database.
    - They are made available to users (buyers/sellers) for download in the dashboard.
    - In addition, the invoice will be sent by e-mail.

-

Contents of invoices:

➔ Supplier invoice to buyer

➔ Space800x invoice to provider

- All invoices must contain the following relevant information:
    - Logo Space800x / Logo provider
    - Name and address of the provider/platform
    - VAT ID number of the provider or platform
    - Quantity and product designation,
    - Amount of the net fee and the calculated VAT and gross amount
    - Transaction data
    - Consecutive invoice number per provider or platform
    - If the reverse charge procedure is used, the tax liability is transferred to the recipient of the service and the following sentence appears on the invoice: *"Tax liability of the recipient of the service in accordance with § 13b UStG".*
    - The following must be stated on all invoices: *"The invoice amount will be collected automatically."*
    - All members will receive an invoice in English unless the language is easily customisable, in which case we will gladly take the more elegant option.

- Invoice to provider:

- A invoice with VAT is created for each platform fee. If the third-party providers do not issue their own invoice, Space800x must pass this on. It should be noted that PayPal is not subject to VAT. Stripe, on the other hand, is.

- Sales amount less:
- Net platform fees (plus VAT or reverse-charge note shown separately)

- Fees for Stripe and PayPal. without VAT.
  ATTENTION: PayPal and Stripe are financial service providers and therefore exempt from VAT. The following note must be included on the invoice: *Services to financial service providers exempt from VAT according to §4 No. 8 UStG.*

  Please ensure that this rate is shown correctly on the invoice

The creation of e-invoices can be done with an open source library PyUBL. PyUBL enables the generation of invoices in the standardised UBL format. Alternatively, libraries such as xmltodict can be used.

Integration of the e-Invoice solution

1. Tax calculations and product categorisations are carried out via Stripe.
2. VAT IDs are validated via the VIES API.
3. Invoices are generated with PyUBL in the UBL standard and can be forwarded digitally.
4. This solution fulfils legal requirements for e-invoices in the EU.

Summary of requirements for the developer

- ➔ Generate an XML-based XRechnung
- ➔ Use the correct UBL or UN/CEFACT standard
- ➔ Comply with mandatory information such as invoice number, date, tax, customer and supplier data
- ➔ Integrate financial service provider notice for PayPal & Stripe

Perform validation of the created XML invoice

Maintenance of the control codes

- Stripe APIs enable regular updates of the control codes.
- A cron job can be implemented to regularly load the latest data into the Django database.

Task 4: Ivan

Tax declaration of the platform:

Step 1: Process platform fees correctly

- VAT on platform fees:
  - Stripe calculates the VAT on our platform fees based on the location of the provider (reverse charge procedure for cross-border B2B transactions within the EU).
  - Enter this data in Stripe and transmit it automatically to the tax office.

- Notification and payment:
  - Use Stripe to correctly report the VAT of platform fees in the OSS (One-Stop-Shop) or in national tax returns.

Step 2: Reporting supplier turnover

The providers' turnover must also be reported to the tax authorities as follows:

- Record transaction data:
  - Set up mechanisms on your platform to capture the following data for each transaction:
    - Seller (supplier) and its VAT ID no.
    - Buyer and its VAT ID number (available).
    - Country of purchase (country of destination).
    - Sales amount and VAT (applicable).
  - Transfer this data to Stripe.

- Report preparation:
  - Use Stripe to generate reports on vendor sales that can be provided to the tax authorities. These reports should include:
    - Total sales per supplier.
    - Total amount of sales tax (will be charged).

- Reporting of sales:
  - Space800x must submit a platform sales report. Stripe can usually generate and submit these reports automatically. Please arrange this.

  ➔ In other words: both the stored data including the VAT IDs of the buyers as well as the tax information of the providers and that of Space800x can now be transmitted directly to Stripe and stored there: https://docs.stripe.com/tax/registrations-api

Left: Components Tax settings and tax registrations

Step 3: Ensure provider compliance

- Expand Stripe functionalities for providers:
  - Make sure that providers can process their tax declarations directly via Stripe (see previous steps for provider connection).
  - This relieves you of the responsibility of calculating or paying the supplier's VAT.

Step 4: Roles and responsibilities

- Space800X commitments:
  - Report and pay sales tax on our platform fees.
  - Ensure that all vendor sales processed via the platform are reported to the tax authorities.

- Provider obligations:
  - However, suppliers remain responsible for paying their own VAT.
  - Nevertheless, we support them through Stripe without bearing or paying their tax burden ourselves.

Technical implementation with Stripe

Stripe can be used for both requirements:

- Platform fees (Space800X revenue):
  - PayPal collects our platform fees incl. VAT for us
  - All stripe fees must also be collected by Paypal on our behalf. The Paypal fees incurred for this must also be paid by the supplier.
  - Space800X also issues the fee invoices to the providers, see below.

- Provider sales:
  - Enter all transaction data from our providers and transfer it to Paypal and into the designated tables in our system.
  - Paypal generates reports that can be used for tax purposes.

Fees

All fees from Stripe or PayPal are borne by the provider and are deducted directly from the turnover by PayPal and paid to Stripe directly by Space800x.

Activate Stripe Filing Service:

Use Stripe to automatically file tax returns for your platform fees and vendor reports, if applicable.

Links and documentation

Here you will find a consolidated list of all relevant links and documentation required for implementation:

1. **Stripe Tax:**

   Stripe Tax API

   List Tax Codes API

   https://docs.stripe.com/tax/tax-codes?locale=de-DE&tax_code=Paperwork

   https://docs.stripe.com/api/tax_codes/list

   https://docs.stripe.com/tax/tax-for-platforms

   https://docs.stripe.com/connect/supported-embedded-components/tax-settings

   https://docs.stripe.com/connect/supported-embedded-components/tax-registrations

   https://docs.stripe.com/tax/registrations-api

   OSS and tax registration with Stripe Tax

2. **VIES API:** VIES API
3. **e-Invoice solutions: see below**
4. **PayPal Developer:** PayPal Developer and
   ](https://developer.paypal.com/docs/api/overview/)