DigitalGate SARL
Bangui, Central African Republic
E-mail: info@digitalgateca.com
October 2025

DigitalGate

DATA ARCHITECTURE

National system for labeling and tracking goods.

CAR.

Data collection and processing. Technical description

Version 2.0.1

Table of contents

1. Terms and Abbreviations

Terms and Abbreviations	**Definitions**
UAA	User Authentication and Authorization
AMS	Access Management System. DigitalGate SARL subsystem for managing user access rights
API	Application Programming Interface
ATTA	Authorized Territorial Tax Authority
B2B	Business-to-Business
B2C	Business-to-Consumer
C2C	Consumer-to-Consumer
DB	Database
DDA	DigitalGate SARL Digital Desk Audit
DEA	DigitalGate SARL Digital Track and Trace for Governments
OECR	Online Electronic Cash Register
PDF	Portable Document Format
TAW	Taxpayer Workplace
TIN	Taxpayer Identification Number
TOW	Tax Officer Workplace
VAT	Value-Added Tax
Bit register	Fixed-length sequence of bits, numbered from the right; the length of the register is measured in bytes
FCA	Fiscal Confirmation Attribute
FD	Fiscal Document
FDA	Fiscal Document Attribute
FDF	Fiscal Document/Data Format

FDO	DigitalGate SARL Fiscal Data Operator. A subsystem or module responsible for receiving, storing, and processing fiscal data, including receipts
FDT	Fiscal Data
FLC	Format Logic Control
FM	Fiscal Module
FMA	Fiscal Message Attribute for Fiscal Data Operator or Tax Authority
FMVC	Fiscal Module Validation Code

2. Summary

DigitalGate SARL Digital Track and Trace for Governments is part of the DigitalGate SARL ecosystem and is a platform designed to provide full traceability and control of marked excise goods, from production to retail sale. The platform supports the issuance, application, aggregation, movement, and write-off of unique product marking codes, ensuring regulatory compliance and market transparency.

The DigitalGate SARL Digital Track and Trace for Governments architecture includes a variety of services, including labeling, acceptance, document management, issuance, and user authentication (User Authentication Authority, UAA), in a scalable environment based on Kubernetes. The architecture supports web and mobile applications for real-time management and verification of marking codes.

Key components of the data infrastructure include:

- PostgreSQL for structured data,
- ClickHouse for high-speed analysis of large volumes of data,
- ScyllaDB for large-scale archiving,
- Redis for in-memory caching,
- Neo4j for tracking graph relationships.

These components provide high availability, efficient performance, and advanced data analytics capabilities.

Through secure APIs, DigitalGate SARL Digital Track and Trace for Governments enables interaction between manufacturers, retailers, and tax authorities, supporting automated reporting, electronic document management, and regulatory compliance analytics.

Modular architecture and real-time data flows enable improved tax compliance, fraud detection, and transparency throughout the supply chain of marked excise goods.

3. Marking code management and processing

The platform provides a unified infrastructure for generating, applying, aggregating, verifying, and writing off product marking codes at all stages of the supply chain. All marking codes are stored in a centralized repository along with metadata about product types, statuses, and traceability data.

The platform includes an integrated code issuance module and supports the generation of code formats based on product categories and taxpayer roles. Invalid or duplicate codes are isolated in a separate queue, logged for further analysis, and displayed in detailed reports and platform dashboards.

Any interactions with labeling codes, whether manual user operations or platform-initiated processes, are recorded in a single event log. Integration with external log storage systems is supported, ensuring traceability and transparency.

All participants in the tagging system are authenticated through a single access layer. Market participants use access control based on Keycloak identification.

All communications and data exchanges in the labeling system are protected using secure HTTPS channels, which guarantee the confidentiality and integrity of the transmitted information.

5. DigitalGate SARL Digital Track and Trace for Governments data transfer architecture

5.1. General information

The DigitalGate SARL Digital Track and Trace for Governments platform is a unified environment for generating, issuing, applying, aggregating, tracking the movement, and writing off marking codes. All events related to marking codes are collected via secure API endpoints and stored in a centralized repository for traceability and reporting to market participants.

The platform guarantees cryptographic protection of marking codes and provides real-time or asynchronous verification at all key stages of the marked product's life cycle: issuance, application, aggregation, entry into circulation, and withdrawal from circulation.

5.1.1. DigitalGate SARL API Gateway Service Digital Track and Trace for Governments

The API gateway service provides secure access points for external systems (manufacturers, distributors, retailers), mobile applications, and provides authentication and authorization of API requests.

Authentication is based on token exchange protocols (OAuth 2.0). Initial authentication is performed when registering a token, and the access token received is used for subsequent interactions.

Tokens and their attributes are stored in the system and checked for each incoming request.

5.1.2. Marking code data reception service

The marking code data reception service processes incoming data related to the following events:

- Applying marking codes;
- Aggregation events;
- Product movement events (movement between participants);
- Sales-related events;
- Write-off events (due to loss, damage, or other reasons).

The data received must comply with predefined JSON schemas corresponding to different types of events.

The acceptance service checks the structural integrity of the received data, including verification of mandatory attributes and basic validation rules.

5.1.3. Marking code data processing queues

After acceptance and initial validation, the data is sent to processing queues:

- **Clean events queue**: includes valid and structurally correct events about marking codes, ready for use in traceability and analytics modules.
- **Rejected events queue**: contains data that has not passed the initial structure check, has missing mandatory attributes, or has incorrect references to marking codes.

Participants receive immediate feedback if an event fails initial checks.

Regulatory authorities can track rejected events through reports.

5.1.4. Statistics collection for analytics and reporting

The DigitalGate SARL Digital Track and Trace for Governments platform includes a statistics collection service that collects key metrics from all events related to marking codes. The statistics collected cover:

- Number of marking codes issued;

- Volume of aggregation events;
- Number of movements and sales;
- Number and reasons for code write-offs.

Statistics from both queues — clean and rejected — are stored in a central database and are available for analytical reports.

5.1.5. Calculation and storage of aggregates for analytical tasks

Verified events about marking codes are stored in an analytical database, where aggregate values are calculated to support regulated reporting and operational analysis. Analytical data includes product movement flows, inventory balances, and tracking of marking code statuses.

5.1.6. API functions of DigitalGate SARL Digital Track and Trace for Governments

The REST API of the DigitalGate SARL Digital Track and Trace for Governments platform provides the following key functions:

- Authentication and authorization;
- Token renewal;
- Marking code issuance management;
- Code application reporting;
- Aggregation reporting;
- Reporting on product transfer;
- Reporting on sales based on expenditure documents;
- Reporting on write-off events.

All API methods ensure communication security and data integrity, allowing seamless integration with participants' internal systems and fiscal systems.

6. API methods DigitalGate SARL Digital Track and Trace for Governments

6.1. Login

Method: **POST /api/v1/uaa/default/login**

Purpose: Authenticates user using login and password.

Request body (x-www-form-urlencoded):

```
{
    "login": "user@example.com",
    "password": "password123"
```

```
}
```

Successful response: Returns accessToken, refreshToken, accessExpiresAt, refreshExpiresAt, and profile.

Responses:

- 200 OK – Login successful
- 400 Bad Request – Invalid credentials
- 401 Unauthorized – Authentication failed
- 500 Internal Server Error – Login service error

6.2. Refresh Access Token

Method: **POST /api/v1/uaa/default/refresh-token**

Purpose: Issues a new access token for user using a valid refresh token.

Request body (x-www-form-urlencoded):

```
{
    "refreshToken": "eyJhbGciOiJIUzI1..."
}
```

Successful response: Returns accessToken, refreshToken, accessExpiresAt, refreshExpiresAt.

Responses:

- 200 OK – Token refreshed
- 400 Bad Request – Invalid or expired refresh token
- 500 Internal Server Error – Token refresh error

6.3. Logout

Method: **POST /api/v1/uaa/default/logout**

Purpose: Logs out the user by invalidating the refresh token.

Request body (x-www-form-urlencoded):

```
{
    "refreshToken": "eyJhbGciOiJIUzI1..."
}
```

Responses:

- 200 OK – Logout successful
- 400 Bad Request – Invalid token

- 500 Internal Server Error – Logout service error

6.4. Get Authenticated User Profile

Method: **GET /api/v1/uaa/default/profile**

Purpose: Returns the authenticated user's detailed profile.

Successful response:

Returns fields: username, email, emailVerified, phone, phoneVerified, firstName, middleName, lastName, fullName, authorities.

Responses:

- 200 OK – Profile data returned
- 401 Unauthorized – Invalid or missing token
- 500 Internal Server Error – Profile retrieval error

6.5. Add New Receipt

Method: **POST /api/v1/receipt**

Purpose: Submits a new fiscal receipt for processing.

Request body (application/json):


```
{
  "receipt": {
    "kktRegId": "123456789",
    "userInn": "9876543210",
    "dateTime": "2025-04-29T12:00:00Z",
    "fiscalDriveNumber": "9287000100000001",
    "fiscalDocumentNumber": 12345,
    "shiftNumber": 1,
    "requestNumber": 1,
    "taxationType": 1,
    "operationType": 1,
    "operator": "Operator Name",
    "user": "Retail User",
    "retailAddress": "123 Retail St",
    "totalSum": 1000.00,
    "cashTotalSum": 500.00,
    "ecashTotalSum": 500.00,
```

```
    "prepaidSum": 0.00,
    "creditSum": 0.00,
    "provisionSum": 0.00,
    "ndsNo": 0.00,
    "nds0": 0.00,
    "nds7": 0.00,
    "nds20": 0.00,
    "items": [
     {
      "productCode": "01234567891234",
      "name": "Product 1",
      "paymentType": 1,
      "price": 500.00,
      "quantity": 1,
      "nds": 20,
      "sum": 500.00
     }
    ]
   }
}
```


Successful response: Returns receiptId (UUID).

Responses:

- 200 OK – Receipt accepted
- 400 Bad Request – Invalid data
- 500 Internal Server Error – Receipt processing error

6.6. Get Receipt Information

Method: **GET /api/v1/receipt/info**

Purpose: Retrieves detailed information about a specific receipt using its receiptId.

Query parameters:

- receiptId (required, string)

Successful response:

Returns receiptId, status (CREATED, DECLINED, APPROVED, PROCESSED), receipt details, and any errors.

Responses:

- 200 OK – Receipt information returned
- 400 Bad Request – Missing or invalid receiptId
- 404 Not Found – Receipt not found
- 500 Internal Server Error – Data retrieval error

6.7. Add Utilization Report

Method: **POST /api/v1/marking/utilization**

Purpose: Creation of a document for applying marking codes (affixing)

Request body (application/json):

```
{
  "codes": ["code1", "code2"],
  "factoryId": "factory-uuid",
  "productionLineId": "line-uuid"
}
```

Successful response:

Returns reportId (UUID).

Responses:

- 200 OK – Report accepted
- 400 Bad Request – Invalid data
- 500 Internal Server Error – Report processing error

6.8. Add UPD Report

Method: **POST /api/v1/marking/upd**

Purpose: Submits a UPD (Universal Transfer Document) report as a raw JSON string.

Request body (application/json):

Raw JSON string (format defined externally).

Successful response:

Returns reportId (UUID).

Responses:

- 200 OK – Report accepted
- 400 Bad Request – Invalid UPD format
- 500 Internal Server Error – Report processing error

6.9. Add Dropout Report

Method: **POST /api/v1/marking/dropout**

Purpose: Submits a report for dropout (loss or spoilage) of marking codes.

Request body (application/json):

```
{
 "codes": ["code1", "code2"],
 "reason": "Lost during transport",
 "withChild": true
}
```

Successful response: Returns reportId (UUID).

Responses:

- 200 OK – Report accepted
- 400 Bad Request – Invalid data
- 500 Internal Server Error – Report processing error

6.10. Add Commissioning Report

Method: **POST /api/v1/marking/commissioning**

Purpose: Submits a report for commissioning (introduction into circulation) of marking codes.

Request body (application/json):

```
{
 "productionDate": "2025-04-29T00:00:00Z",
 "codes": ["code1", "code2"]
}
```

Successful response: Returns reportId (UUID).

Responses:

- 200 OK – Report accepted
- 400 Bad Request – Invalid data
- 500 Internal Server Error – Report processing error

6.11. Add Aggregation Report

Method: **POST /api/v1/marking/aggregation**

Purpose: Submits a report for aggregation of marking codes into larger logistical units.

Request body (application/json):

```
{
 "aggregate": [
  {
```

"aggregationType": "PALLET",
"aggregationNumber": "PALLET-123",
"codes": ["code1", "code2"]
}
]
}

Successful response: Returns reportId (UUID).

Responses:

- 200 OK – Report accepted
- 400 Bad Request – Invalid data
- 500 Internal Server Error – Report processing error

6.12. Get Report Information

Method: **GET /api/v1/marking/info**

Purpose: Retrieves information about a specific marking report based on reportId.

Query parameters:

- reportId (optional, string, UUID format) – Unique identifier of the report.

Successful response:

Returns an object containing:

- reportId (string, UUID) – Report identifier.
- status (string) – Status of the report. Possible values:
  - CREATED
  - DECLINED
  - APPROVED
  - PROCESSED
- reportType (string) – Type of the report. Possible values:
  - UTILISATION (utilisation report)
  - DROPOUT (dropout report)
  - AGGREGATE (aggregation report)
  - INTRODUCE (commissioning report)
  - UPD (UPD document report)
- reportInfo (array) – Detailed data related to the report, format depends on the report type.

- errors (optional, array) – List of errors associated with the report, each containing:
    - code (string) – Error code.
    - message (string) – Error description.

Example of a successful response:


```
{
 "result": [
  {
   "reportId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
   "status": "APPROVED",
   "reportType": "UTILISATION",
   "reportInfo": [
    {
     // Report-specific data
    }
   ],
   "errors": [
    {
     "code": "ERR001",
     "message": "Invalid code format"
    }
   ]
  }
 ]
}
```


Error Responses:

- 400 Bad Request – Invalid or missing query parameters.
- 404 Not Found – No report found with the specified reportId.
- 500 Internal Server Error – Unexpected server error while retrieving report information.

6.13. Create Emission Order (External Clients)

Method: **POST /api/v1/emission/orders**

Purpose: Creates an emission order for marking codes by an external client, specifying products, factory, and production line.

Request body (application/json):

{
"country": "RU",
"factoryId": "factory-uuid",
"productionLineId": "line-uuid",
"products": [
{
"type": "CONSUMER_UNIT",
"gtin": "04601234567890",
"quantity": 1000,
"templateId": "template-uuid"
}
]
}

Successful response: Returns the created orderId in UUID format.

Example response:

{
"orderId": "f27a34b9-5f64-40da-81d6-5db46dbab43e"
}

Response codes:

- 200 OK – Order successfully created
- 400 Bad Request – Invalid data or missing fields
- 500 Internal Server Error – Emission system error

6.14. Get Emission Order Status (External Clients)

Method: **GET /api/v1/emission/orders**

Purpose: Retrieves the status and detailed info about an emission order created by an external client.

Query parameters (optional):

- orderId (string, UUID format) – Identifier of the emission order.

Successful response: Returns a list of statuses for matching orders.

Example response:

{
"result": [
{

```
  "orderId": "f27a34b9-5f64-40da-81d6-5db46dbab43e",
    "status": "APPROVED",
    "orderInfos": [
     {
      "gtin": "04601234567890",
      "totalCodes": 1000,
      "availableCodes": 950
     }
    ],
    "errors": []
   }
  ]
}
```


Status values:

- CREATED
- DECLINED
- VERIFIED
- APPROVED
- READY
- CLOSED

Response codes:

- 200 OK – Order info returned
- 400 Bad Request – Invalid or missing orderId
- 404 Not Found – Order not found
- 500 Internal Server Error – Retrieval error

6.15. Get Marking Codes

Method: **GET /api/v1/emission/codes**

Purpose: Retrieves marking codes generated under a specific emission order.

Query parameters (optional):

- orderId (string, UUID) – Identifier of the emission order
- gtin (string) – GTIN of the product
- quantity (integer) – Number of codes to retrieve

- lastBlockId (string, UUID) – For pagination or continuation

Successful response: Returns one or more blocks of codes.

Example response:


```
{
 "result": [
  {
   "blockId": "15b6d2d4-92b8-472e-9391-91f92b750b47",
   "codes": [
    "010460123456789021abcd1234",
    "010460123456789021abcd1235"
   ],
   "errors": []
  }
 ]
}
```


Response codes:

- 200 OK – Codes returned
- 400 Bad Request – Invalid parameters
- 404 Not Found – Codes not found
- 500 Internal Server Error – Retrieval error

6.16. Get Marking Code Blocks

Method: **GET /api/v1/emission/codes/blocks**

Purpose: Retrieves blocks of marking codes grouped by GTIN and order ID. Useful for analyzing previously issued code batches.

Query parameters (optional):

- orderId (string, UUID) – Identifier of the emission order
- gtin (string) – GTIN of the product

Successful response: Returns a list of blocks for each matching GTIN.

Example response:


```
{
 "result": [
  {
   "orderId": "2e7f885e-c91e-4a41-a06e-f94d33edee15",
```

```
    "gtin": "04601234567890",
    "blocks": [
     {
      "blockId": "6d7c88ef-59c9-4229-8bfe-7d928e434b20",
      "blockDateTime": 1714406400000,
      "quantity": 500
     }
    ],
    "errors": []
   }
  ]
}
```


Response codes:

- 200 OK – Blocks returned
- 400 Bad Request – Invalid query
- 404 Not Found – No blocks found
- 500 Internal Server Error – Internal error

6.17. Get Factory with Lines Details

Method: **GET /api/v1/service/profile/factories/{factoryId}**

Purpose: Returns detailed information about a specific factory, including all related factory lines.

Path parameter:

- factoryId (integer, required) – ID of the factory

Success response: Returns complete information about the factory and a list of its production lines.

Example response:


```
{
 "id": 101,
 "address": "Industrial Zone 5, Block 12",
 "manufacturerId": 150,
 "name": "Factory A",
 "code": "FAC-A1",
 "country": "Germany",
 "countryCode": "DE",
 "factoryLines": [
```

{
"id": 501,
"factoryId": 101,
"code": "LINE-01",
"name": "Line 1",
"productName": "Product X",
"lastActivityTime": "2025-04-20T15:32:00Z",
"productsCount": 5000,
"packsCount": 1000,
"defectPercentage": 0.5,
"active": true
}
]
}

Response codes:

- 200 OK – Data retrieved successfully
- 400 Bad Request – Invalid factory ID

6.18. Get List of Factories

Method: **GET /api/v1/service/profile/factories**

Purpose: Retrieves a list of factories for a MANUFACTURER client.

Success response: Returns a list of factories.

Example response:

[
{
"id": 1,
"address": "Industrial Street 1",
"manufacturerId": 100,
"name": "Factory One",
"code": "FAC-ONE",
"factoryLinesTotalCount": 3,
"country": "USA",
"countryCode": "US"
},

{
"id": 2,
"address": "Industrial Street 2",
"manufacturerId": 100,
"name": "Factory Two",
"code": "FAC-TWO",
"factoryLinesTotalCount": 5,
"country": "USA",
"countryCode": "US"
}
]

Response codes:

- 200 OK – Factories retrieved successfully
- 400 Bad Request – Invalid request

6.19. Get List of Countries

Method: **GET /api/v1/service/profile/countries**

Purpose: Retrieves a list of all available countries.

Success response: Returns an array of countries.

Example response:

[
{
"code": "US",
"country": "United States"
},
{
"code": "DE",
"country": "Germany"
}
]

Response codes:

- 200 OK – Countries retrieved successfully
- 400 Bad Request – Invalid request

6.20. Get All Product Groups

Method: **GET /api/v1/service/product-groups**

Purpose: Returns a list of all product groups in a short format, suitable for emission logic.

Success response: Returns an array of product groups.

Example response:


```
[
 {
  "code": "TOBACCO",
  "sampleDescription": "Tobacco Products"
 },
 {
  "code": "ALCOHOL",
  "sampleDescription": "Alcoholic Beverages"
 }
]
```


Response codes:

- 200 OK – Product groups retrieved successfully
- 400 Bad Request – Invalid request

6.21. Validate Marking Code

Method: **POST /api/v1/validation/codes**

Purpose: Validates a single marking code.

Request body: (multipart/form-data)

markingCode (string, required) — The marking code to validate.

Success response: Returns the validation result for the marking code, including confirmation status and additional product information.

Example response:


```
{
  "markingCodeConfirmed": true,
  "errors": [],
  "info": {
   "markingCode": "0101234567890123abcd",
   "producedAt": "2025-04-25T09:10:00Z",
   "productName": "Product A",
```

"manufacturerName": "Manufacturer X",
"factoryName": "Factory Y",
"factoryLineName": "Line 1",
"factoryLineCode": "L001",
"storeName": "Store ABC",
"storeAddress": "Main Street 123",
"productGroupCode": "TOBACCO",
"gtin": "01234567890123"
}
}

Response codes:

- 200 OK – Marking code validated successfully
- 400 Bad Request – Invalid input data

6.22. Get List of Receipts

Method: **POST /api/v1/service/receipts**

Purpose: Retrieves a list of receipts available for the current client with pagination.

Request body:

{
"paging": {
"pageSize": 10,
"pageRef": []
}
}

Success response: Returns a paginated list of receipts with basic information.

Example response:

{
"nextPageRef": ["c29tZS1uZXh0LXBhZ2UtcmVm"],
"data": [
{
"receiptId": "a3bb88a5-7f4e-4b98-a05b-2c3c7d92f5ec",
"createdAt": "2025-04-26T13:00:00Z",
"operationType": 1,
"status": "APPROVED",

"manufacturerName": "Manufacturer A"
},
{
"receiptId": "d5fa2cbe-6b10-49c3-b688-5f8145b82476",
"createdAt": "2025-04-26T15:30:00Z",
"operationType": 2,
"status": "PROCESSED",
"manufacturerName": "Manufacturer B"
}
]
}

Response codes:

- 200 OK – Receipts list retrieved successfully
- 400 Bad Request – Invalid input data

6.23. Get Receipt Details

Method: **POST /api/v1/service/receipts/details**

Purpose: Retrieves detailed information about a specific receipt.

Request body:

{
"receiptId": "a3bb88a5-7f4e-4b98-a05b-2c3c7d92f5ec"
}

Success response: Returns detailed information about the receipt, including user information, fiscal data, and status.

Example response:

{
"createdAt": "2025-04-26T13:00:00Z",
"receiptId": "a3bb88a5-7f4e-4b98-a05b-2c3c7d92f5ec",
"user": "John Doe",
"retailAddress": "123 Market Street",
"userInn": "1234567890",
"taxationType": 1,
"operator": "Operator X",
"shiftNumber": 12,

"dateTime": "2025-04-26T13:15:00Z",
"fiscalDocumentNumber": 987654,
"fiscalDriveNumber": "DRIVE123456",
"productsTotalCount": 5,
"status": "APPROVED",
"errors": []
}

Response codes:

- 200 OK – Receipt details retrieved successfully
- 400 Bad Request – Invalid input data

6.24. Get Product Details by GTIN

Method: **POST /api/v1/service/products/details**

Purpose: Retrieves full information about a product by its GTIN or marking code. GTIN can be null if searching by marking code only.

Request body:

{
"gtin": "04612345678901",
"markingCode": "0104612345678901212vA7dE39rL0G1Z"
}

Success response: Returns detailed information about the product and its status.

Example response:

{
"markingCode": "0104612345678901212vA7dE39rL0G1Z",
"status": "INTRODUCED",
"packagingType": "PACK",
"packagingLevel": 1,
"product": {
"gtin": "04612345678901",
"manufacturerName": "Tobacco Corp",
"productName": "Marlboro Red",
"productTrademark": "Marlboro",
"productVendorCode": "MR123",
"manufacturerId": 1001,

```
  "productGroupCode": "TOBACCO"
 },
 "statuses": {
  "INTRODUCED": {
   "status": "INTRODUCED",
   "operationTime": "2025-04-28T10:00:00Z"
  }
 },
 "operations": [
  {
   "operationAction": "INTRODUCED",
   "operationTime": "2025-04-28T10:00:00Z",
   "clientName": "Retailer Ltd"
  }
 ],
 "childrenTotalCount": 0,
 "owner": "Retailer Ltd"
}
```


Response codes:

- 200 OK – Product details retrieved successfully
- 400 Bad Request – Invalid input data

6.25. Get Aggregate Products

Method: **POST /api/v1/service/products/details/children**

Purpose: Retrieves a list of aggregate products by GTIN and/or aggregation number. GTIN can be null if aggregation number is provided.

Request body:


```
{
 "gtin": "04612345678901",
 "aggregationNumber": "AGGREGATE-001",
 "paging": {
  "pageNumber": 0,
  "pageSize": 10
 }
```

```
}
```

Success response:

Returns a list of products inside the specified aggregate.

Example response:

```
[
 {
  "gtin": "04612345678901",
  "markingCode": "0104612345678901212vA7dE39rL0G1Z",
  "status": "AGGREGATED",
  "productName": "Marlboro Red",
  "manufacturerName": "Tobacco Corp",
  "packagingType": "CARTON",
  "modifiedAt": "2025-04-28T10:30:00Z",
  "packagingLevel": 2,
  "productGroupCode": "TOBACCO",
  "brand": "Marlboro",
  "owner": "Retailer Ltd"
 }
]
```

Response codes:

- 200 OK – Aggregate products retrieved successfully
- 400 Bad Request – Invalid input data

6.26. Get List of Product Cards

Method: **POST /api/v1/service/product-cards**

Purpose: Returns a paginated list of product cards available for the current client. Supports filtering and sorting.

Request body:

```
{
 "paging": {
  "pageNumber": 0,
  "pageSize": 10
 },
 "filters": [
```

```
    {
      "id": "productName",
      "value": {
        "value": ["Marlboro Red"]
      }
    }
  ],
  "sortings": [
    {
      "id": "productVendorCode",
      "desc": false
    }
  ]
}
```


Success response: Returns a list of product card objects with pagination.

Example response:


```
[
  {
    "gtin": "04612345678901",
    "manufacturerName": "Tobacco Inc.",
    "productName": "Marlboro Red",
    "productVendorCode": "MR123",
    "productTrademark": "Marlboro",
    "manufacturerId": 101,
    "productGroupCode": "TOBACCO",
    "packageType": "PACK"
  }
]
```


Response codes:

- 200 OK – List of product cards returned successfully
- 400 Bad Request – Invalid filters, sorting, or pagination input

6.27. Get Emission Orders

Method: **POST /api/v1/service/orders**

Purpose: Retrieves a paginated list of emission orders available to the current client. This operation returns N+1 elements for page size N if the next page exists.

Request body:


```
{
 "paging": {
  "pageNumber": 0,
  "pageSize": 10
 },
 "filters": [
  {
   "id": "status",
   "value": {
    "value": "APPROVED"
   }
  }
 ],
 "sortings": [
  {
   "id": "createdAt",
   "desc": true
  }
 ]
}
```


Request parameters:

- paging (object, optional) – Contains pageNumber and pageSize
- filters (array, optional) – Filtering criteria for emission orders (e.g., by status, date, etc.)
- sortings (array, optional) – Sorting rules for results (e.g., by creation date)

Success response: Returns a list of emission orders matching the provided filters and pagination.

Example response:


```
[
 {
  "orderId": "4e14cf3e-21ef-4b2a-9fd2-8421c2f0e1e4",
  "createdAt": "2025-04-25T08:10:00Z",
  "closedAt": "2025-04-26T15:00:00Z",
```

"status": "APPROVED",
"totalMarkingCodes": 100000,
"participant": "Example Manufacturer"
}
]

Response codes:

- 200 OK – Emission orders retrieved successfully
- 400 Bad Request – Invalid request structure or filtering parameters

6.28. Get Emission Order Details

Method: **POST /api/v1/service/orders/details**

Purpose: Retrieves detailed information about a specific emission order.

Request body:

{
"orderId": "4e14cf3e-21ef-4b2a-9fd2-8421c2f0e1e4"
}

Request parameters:

- orderId (string, required) – Unique identifier (UUID) of the emission order

Success response: Returns detailed information about the emission order, including status, factory ID, product group, and available codes.

Example response:

{
"result": {
"orderId": "4e14cf3e-21ef-4b2a-9fd2-8421c2f0e1e4",
"createdAt": "2025-04-25T08:10:00Z",
"closedAt": "2025-04-26T15:00:00Z",
"status": "APPROVED",
"totalMarkingCodes": 100000,
"factoryId": "12345",
"country": "Germany",
"productionLineId": "67890",
"productsTotalCount": 2,
"productGroupCode": "TOBACCO",
"availableCodes": 98765,

"errors": []
}
}

Response codes:

- 200 OK – Emission order details retrieved successfully
- 400 Bad Request – Invalid or missing order ID

6.29. Get List of Documents

Method: **POST /api/v1/service/documents**

Purpose: Returns a paginated list of documents (reports) available for the current client, filtered by type, creation date, or owner.

Request body:

{
"paging": {
"pageSize": 10,
"pageRef": []
},
"type": "UTILISATION",
"createdAt": "2024-12-01T00:00:00Z",
"docOrOwner": "1234567890"
}

Request parameters:

paging (object, optional) – Pagination settings:

pageSize (integer) – Number of results per page

pageRef (array of base64 strings) – Reference for the next page

type (string, optional) – Report type (UTILISATION, DROPOUT, AGGREGATE, INTRODUCE, UPD)

createdAt (string, optional) – Filter by creation date (ISO 8601)

docOrOwner (string, optional) – Document ID or owner identifier

Success response: Returns a paginated list of report summaries.

Example response:

{
"nextPageRef": [],
"data": [
{

"reportId": "d38a3ed2-2ef2-4d2b-a1e9-b4fd03aa98a3",
"status": "CREATED",
"type": "UTILISATION",
"createdAt": "2024-12-01T10:30:00Z",
"owner": "1234567890",
"seller": "Company A",
"buyer": "Company B"
}
]
}

Response codes:

- 200 OK – List retrieved successfully
- 400 Bad Request – Invalid input data

6.30. Get List of Invoices

Method: **GET /api/v1/service/invoice/list**

Purpose: Returns a paginated list of invoices available for the current client, filtered by creation date.

Request body:

{
"gtin": "string",
"paging": {
"pageNumber": 0,
"pageSize": 0
},
"filters": [
{
"id": "string",
"value": {}
}
],
"sortings": [
{
"id": "string",
"desc": true

```
            }
        ]
        }
```


Example response:


```
        {
        "totalElements": 0,
        "totalPages": 0,
        "data": [
            {
                "id": 0,
                "createdAt": "2025-12-25T06:41:39.537Z",
                "updatedAt": "2025-12-25T06:41:39.537Z",
                "status": "string",
                "url": "string",
                "senderTin": "string",
                "receiverTin": "string"
            }
        ]
        }
```


Response codes:

- 200 OK – List retrieved successfully
- 400 Bad Request – Invalid input data

6.31. Get Invoice Details

Method: **GET /api/v1/service/invoice/info**

Purpose: Returns detailed invoice information available for the current customer.

Request parameter:

- id (integer) – To get info about an invoice, you need to give the invoice ID from the list of available invoices for the current customer.

Example response:


```
        {
          "id": 0,
          "createdAt": "2025-12-25T06:44:13.281Z",
          "updatedAt": "2025-12-25T06:44:13.281Z",
```

"status": "string",
"url": "string",
"senderTin": "string",
"receiverTin": "string"
}

Response codes:

- 200 OK – List retrieved successfully
- 400 Bad Request – Invalid input data