EN - Web API v7 Implementation Documentation Overview

# Web API v7 Implementation Documentation

## Table of Contents



---

## 1. Introduction

### Software Overview

Web API version 7 represents a newly designed online application that enables automated retrieval and submission of product specifications. This version introduces important improvements and new functionalities:

- Support for both XML and JSON formats
- Enhanced performance and scalability
- Extensive validation capabilities
- Optimized data exchange

### Target Audience

This documentation is specifically compiled for:

- Producers and wholesalers who want to implement the API
- Customers who need automated access to product information
- Technical teams responsible for integration
- Developers who will implement the API

---

## 2. Installation Guide

### System Requirements

The following minimum requirements apply for using the Web API:

- A reliable and stable internet connection
- Knowledge of endpoints and authentication handling
- Access to the specific URL or network address
- Valid authentication credentials

#### Basic Settings

1. Obtain access to the API environment via PS
2. Configure authentication credentials
3. Test basic endpoints
4. Implement error handling

---

### 3. Getting Started - Your First API Call

This section guides you step-by-step through the process of your first API call, from authentication to retrieving a product. Ideal for developers who want to get started immediately.

#### What Do You Need?

Before you begin, make sure you have the following:

- **API Base URL**: `<https://webapi.psinfoodservice.com>`
- **Username**: Received from PS in Foodservice
- **Password**: Received from PS in Foodservice
- **Product ID**: A valid product sheet ID (psId) to which you have access
- **HTTP client**: cURL, Postman, or programming code in your favorite language

#### Step 1: Login and Obtain Token

To use the API, you first need a JWT token. You use this token for all subsequent API calls.

**Endpoint**

```
POST https://webapi.psinfoodservice.com/v7/json/account/login
```

**Request Headers**

```
Content-Type: application/json
```

**Request Body**

```
{
  "username": "your_username",
  "password": "your_password"
}
```

**Response Success - 200 OK**

```
{
  "accesstoken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshtoken": "a8d9f7e6c5b4a3d2e1f0...",
  "expiresin": 3600
}
```

**Important:**

- The `accesstoken` is valid for the number of seconds indicated in `expiresin` (default 3600 seconds = 1 hour)
- Store the `refreshtoken` securely - you can use it to request a new accesstoken without logging in again
- NEVER store tokens in source code or public repositories

#### Step 2: Retrieve Product with Token

Now that you have a token, you can retrieve product information.

**Endpoint**

```
GET https://webapi.psinfoodservice.com/v7/json/productsheet/{psId}
```

or with specific language:

```
GET https://webapi.psinfoodservice.com/v7/json/productsheet/{language}/{psId}
```

**Parameters:**

- `{psId}` : Product ID (for example: 1234567)
- `{language}` : (optional) Language code: `nl` , `en` , `de` , or `fr`

**Request Headers**

```
Authorization: Bearer {your_accesstoken}
Content-Type: application/json
```

**Response Success - 200 OK**

```
{
  "logistic": {
    "psId": 1234567,
    "gtince": "8712345678901",
    "gtinhe": "18712345678908",
    "productName": "Example Product"
  },
  "product": {
    "articleNumber": "ART12345",
    "brand": "Example Brand"
  },
  "specification": {
  }
}
```

Complete Examples in Different Languages

cURL (Command Line)

**Step 1: Login**

```
curl -X POST https://webapi.psinfoodservice.com/v7/json/account/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "your_username",
    "password": "your_password"
  }'
```

**Step 2: Retrieve product**

```
curl -X GET https://webapi.psinfoodservice.com/v7/json/productsheet/1234567 \
  -H "Authorization: Bearer {TOKEN}" \
  -H "Content-Type: application/json"
```

C# (.NET 6+)

```
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Threading.Tasks;

public class Program
{
    private static readonly HttpClient client = new HttpClient();
    private const string BaseUrl = "https://webapi.psinfoodservice.com/v7/json";

    public static async Task Main(string[] args)
    {
        try
        {
            var loginData = new
            {
                username = "your_username",
                password = "your_password"
            };

            var loginResponse = await client.PostAsJsonAsync(
                $"{BaseUrl}/account/login",
                loginData
```

```
            );

            loginResponse.EnsureSuccessStatusCode();

            var tokenResponse = await
loginResponse.Content.ReadFromJsonAsync<TokenResponse>();
            Console.WriteLine($"Token obtained, valid for
{tokenResponse.ExpiresIn} seconds");

            client.DefaultRequestHeaders.Authorization =
                new AuthenticationHeaderValue("Bearer",
tokenResponse.AccessToken);

            int productId = 1234567;
            var productResponse = await client.GetAsync(
                $"{BaseUrl}/productsheet/{productId}"
            );

            productResponse.EnsureSuccessStatusCode();

            var productSheet = await
productResponse.Content.ReadAsStringAsync();
            Console.WriteLine("Product retrieved:");
            Console.WriteLine(productSheet);
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine($"Request error: {e.Message}");
        }
    }
}

public class TokenResponse
{
    public string AccessToken { get; set; }
    public string RefreshToken { get; set; }
    public int ExpiresIn { get; set; }
}
```

JavaScript/Node.js (with fetch)

```
const baseUrl = 'https://webapi.psinfoodservice.com/v7/json';

async function getProduct() {
  try {
    const loginResponse = await fetch(`${baseUrl}/account/login`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        username: 'your_username',
        password: 'your_password'
      })
    });

    if (!loginResponse.ok) {
      throw new Error(`Login failed: ${loginResponse.status}`);
    }

    const tokenData = await loginResponse.json();
    console.log(`Token obtained, valid for ${tokenData.expiresin}
seconds`);

    const productId = 1234567;
    const productResponse = await fetch(
      `${baseUrl}/productsheet/${productId}`,
      {
        method: 'GET',
        headers: {
          'Authorization': `Bearer ${tokenData.accesstoken}`,
          'Content-Type': 'application/json'
        }
      }
    );

    if (!productResponse.ok) {
      throw new Error(`Product retrieval failed:
${productResponse.status}`);
    }

    const productData = await productResponse.json();
    console.log('Product retrieved:', productData);

    return productData;

  } catch (error) {
    console.error('Error:', error.message);
```

```
    throw error;
  }
}

getProduct();
```

Python (with requests library)

```python
import requests
import json

BASE_URL = "https://webapi.psinfoodservice.com/v7/json"

def get_product():
    try:
        login_data = {
            "username": "your_username",
            "password": "your_password"
        }

        login_response = requests.post(
            f"{BASE_URL}/account/login",
            json=login_data,
            headers={"Content-Type": "application/json"}
        )

        login_response.raise_for_status()
        token_data = login_response.json()

        access_token = token_data["accesstoken"]
        print(f"Token obtained, valid for {token_data['expiresin']}
seconds")

        product_id = 1234567
        headers = {
            "Authorization": f"Bearer {access_token}",
            "Content-Type": "application/json"
        }

        product_response = requests.get(
            f"{BASE_URL}/productsheet/{product_id}",
            headers=headers
        )

        product_response.raise_for_status()
        product_data = product_response.json()

        print("Product retrieved:")
        print(json.dumps(product_data, indent=2))

        return product_data

    except requests.exceptions.RequestException as e:
        print(f"Error: {e}")
        raise

if __name__ == "__main__":
    get_product()
```

PHP

```php
<?php

$baseUrl = "https://webapi.psinfoodservice.com/v7/json";

function getProduct() {
    global $baseUrl;

    try {
        $loginData = [
            'username' => 'your_username',
            'password' => 'your_password'
        ];

        $loginOptions = [
            'http' => [
                'method' => 'POST',
                'header' => 'Content-Type: application/json',
                'content' => json_encode($loginData)
            ]
        ];

        $loginContext = stream_context_create($loginOptions);
        $loginResponse = file_get_contents(
            "$baseUrl/account/login",
            false,
            $loginContext
        );
```

```
28
29          if ($loginResponse === false) {
30              throw new Exception("Login failed");
31          }
32
33          $tokenData = json_decode($loginResponse, true);
34          $accessToken = $tokenData['accesstoken'];
35
36          echo "Token obtained, valid for {$tokenData['expiresin']}
    seconds\n";
37
38          $productId = 1234567;
39
40          $productOptions = [
41              'http' => [
42                  'method' => 'GET',
43                  'header' => "Authorization: Bearer $accessToken\r\n" .
44                              "Content-Type: application/json"
45              ]
46          ];
47
48          $productContext = stream_context_create($productOptions);
49          $productResponse = file_get_contents(
50              "$baseUrl/productsheet/$productId",
51              false,
52              $productContext
53          );
54
55          if ($productResponse === false) {
56              throw new Exception("Product retrieval failed");
57          }
58
59          $productData = json_decode($productResponse, true);
60
61          echo "Product retrieved:\n";
62          echo json_encode($productData, JSON_PRETTY_PRINT) . "\n";
63
64          return $productData;
65
66      } catch (Exception $e) {
67          echo "Error: " . $e->getMessage() . "\n";
68          throw $e;
69      }
70  }
71
72  getProduct();
73
74  ?>
```

#### Using XML Format

If you want to use XML format instead of JSON, replace `/json/` with `/xml/` in the URLs:

**Login (XML):**

```
1  POST https://webapi.psinfoodservice.com/v7/xml/account/login
2  Content-Type: application/json
```

**Retrieve product (XML):**

```
1  GET https://webapi.psinfoodservice.com/v7/xml/productsheet/1234567
2  Authorization: Bearer {token}
```

**Note:** The request body for login remains JSON, but the response will be in XML format.

#### Token Refresh (Refresh Token)

When your access token is about to expire, you can request a new token without logging in again:

**Endpoint**

```
1  POST https://webapi.psinfoodservice.com/v7/json/account/refreshtoken
```

**Request Body**

```
1  {
2    "accessToken": "your_current_accesstoken",
3    "refreshToken": "your_refreshtoken"
4  }
```

### Response

```
{
  "accesstoken": "new_accesstoken",
  "refreshtoken": "new_refreshtoken",
  "expiresin": 3600
}
```

#### Troubleshooting

##### Problem: 401 Unauthorized during login

### Possible causes:

- Incorrect username or password
- IP address is blocked after multiple failed login attempts (24 hour block)
- Account is not active

### Solution:

- Check your credentials
- Wait 24 hours if your IP is blocked
- Contact support: info@psinfoodservice.com

##### Problem: 401 Unauthorized when retrieving product

### Possible causes:

- Token has expired
- Token is not correctly included in the Authorization header
- Format of the Authorization header is incorrect

### Solution:

```
# Correct format:
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

# NOT:
Authorization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

##### Problem: 403 Forbidden when retrieving product

### Possible causes:

- You don't have access to this specific product
- Product ID doesn't exist
- Your account doesn't have the correct permissions

### Solution:

- Check if the product ID is correct
- Check if your account has access to this product
- Contact support for access questions

##### Problem: 404 Not Found

### Possible causes:

- Product doesn't exist in the system
- Wrong endpoint used
- Typo in URL

### Solution:

- Check URL spelling
- Verify that the product ID exists

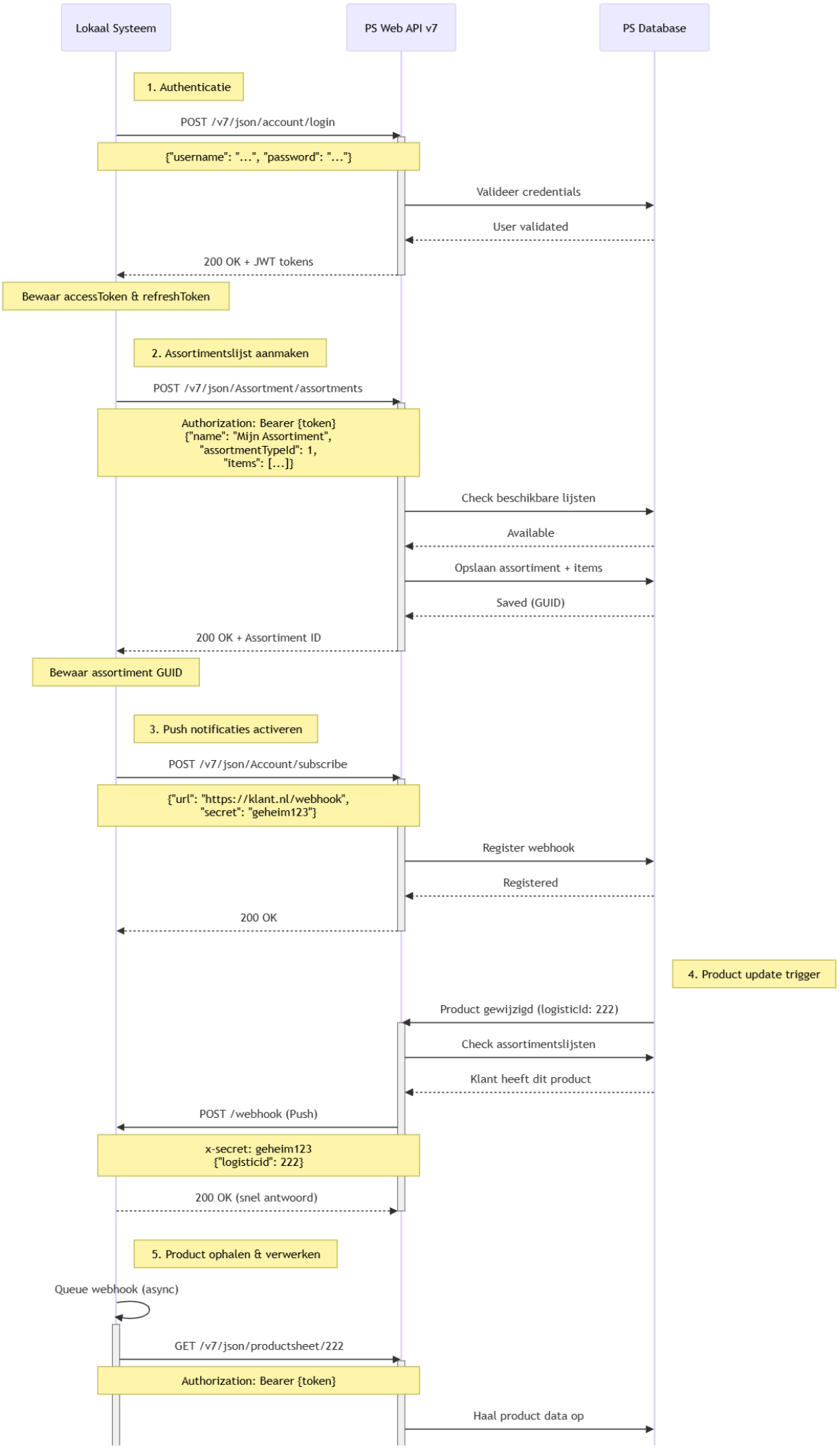
- Check the API documentation for correct endpoints

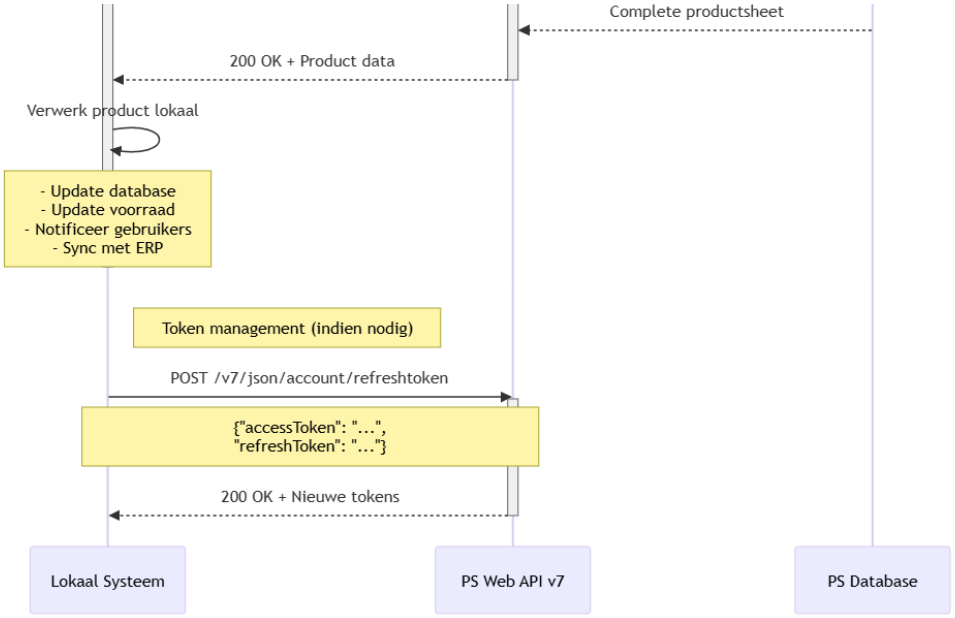
#### Problem: 400 Bad Request

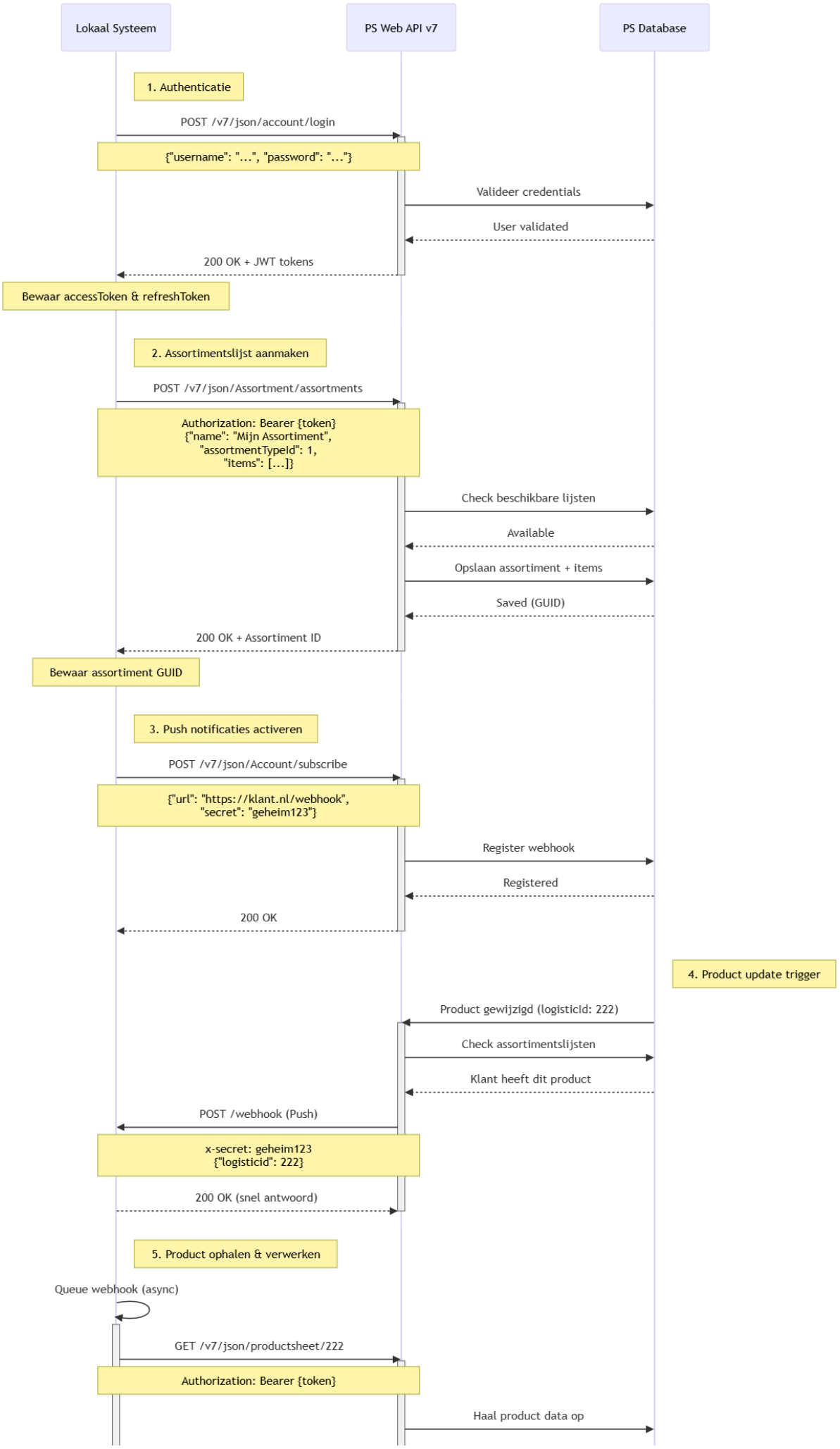
### Possible causes:

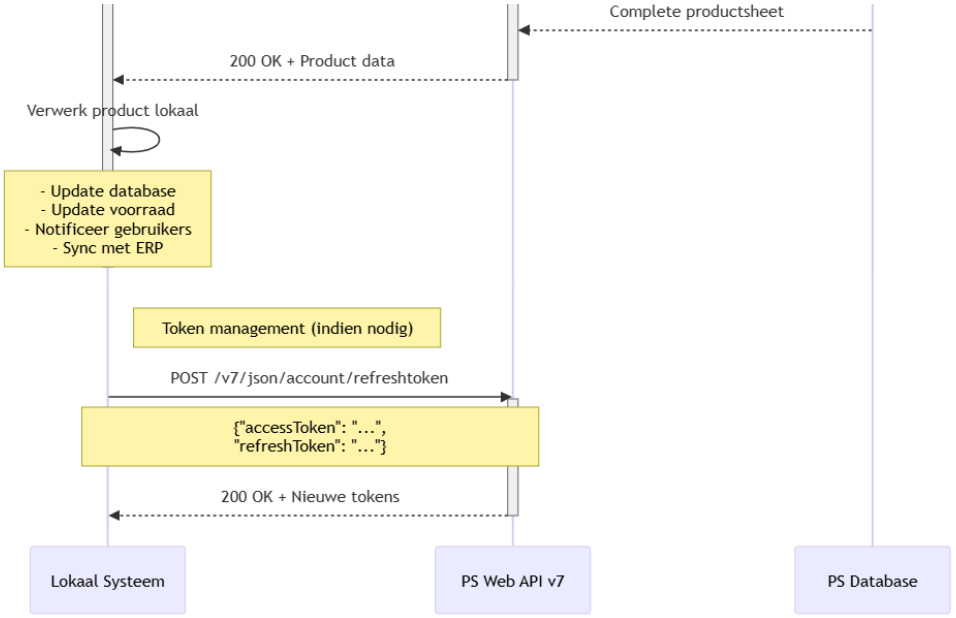
- Invalid JSON in request body
- Missing required fields
- Wrong data types
- Invalid language code (must be: nl, en, de, or fr)

#### Best Practices for Getting Started


Lokaal Systeem
PS Web API v7
PS Database
1. Authenticatie
POST /v7/json/account/login
{"username": "...", "password": "..."}
Valideer credentials
User validated
200 OK + JWT tokens
Bewaar accessToken & refreshToken
2. Assortimentslijst aanmaken
POST /v7/json/Assortment/assortments
Authorization: Bearer {token}
{"name": "Mijn Assortiment",
"assortmentTypeId": 1,
"items": [...]}
Check beschikbare lijsten
Available
Opslaan assortiment + items
Saved (GUID)
200 OK + Assortiment ID
Bewaar assortiment GUID
3. Push notificaties activeren
POST /v7/json/Account/subscribe
{"url": "https://klant.nl/webhook",
"secret": "geheim123"}
Register webhook
Registered
200 OK
4. Product update trigger
Product gewijzigd (logisticId: 222)
Check assortimentslijsten
Klant heeft dit product
POST /webhook (Push)
x-secret: geheim123
{"logisticid": 222}
200 OK (snel antwoord)
5. Product ophalen & verwerken
Queue webhook (async)
GET /v7/json/productsheet/222
Authorization: Bearer {token}
Haal product data op


Complete productsheet
200 OK + Product data
Verwerk product lokaal
- Update database
- Update voorraad
- Notificeer gebruikers
- Sync met ERP
Token management (indien nodig)
POST /v7/json/account/refreshtoken
{"accessToken": "...",
"refreshToken": "..."}
200 OK + Nieuwe tokens
Lokaal Systeem
PS Web API v7
PS Database


Lokaal Systeem
PS Web API v7
PS Database
1. Authenticatie
POST /v7/json/account/login
{"username": "...", "password": "..."}
Valideer credentials
User validated
200 OK + JWT tokens
Bewaar accessToken & refreshToken
2. Assortimentslijst aanmaken
POST /v7/json/Assortment/assortments
Authorization: Bearer {token}
{"name": "Mijn Assortiment",
"assortmentTypeId": 1,
"items": [...]}
Check beschikbare lijsten
Available
Opslaan assortiment + items
Saved (GUID)
200 OK + Assortiment ID
Bewaar assortiment GUID
3. Push notificaties activeren
POST /v7/json/Account/subscribe
{"url": "https://klant.nl/webhook",
"secret": "geheim123"}
Register webhook
Registered
200 OK
4. Product update trigger
Product gewijzigd (logisticId: 222)
Check assortimentslijsten
Klant heeft dit product
POST /webhook (Push)
x-secret: geheim123
{"logisticid": 222}
200 OK (snel antwoord)
5. Product ophalen & verwerken
Queue webhook (async)
GET /v7/json/productsheet/222
Authorization: Bearer {token}
Haal product data op




1. **Token Management**
   - Store tokens securely (e.g., environment variables, secure storage)
   - Implement token refresh logic for expired tokens
   - NEVER log tokens in console or log files
2. **Error Handling**
   - Implement try-catch blocks for all API calls
   - Handle different HTTP status codes separately
   - Log errors for debugging, but without sensitive information
3. **Testing**
   - Test first with Postman or cURL before writing code
   - Use the Swagger documentation for endpoint details
   - Test both success and error scenarios
4. **Rate Limiting**
   - The API has rate limiting per endpoint
   - Implement exponential backoff for rate limit errors
   - Cache results where possible
5. **Production Deployment**
   - Always use HTTPS (never HTTP)
   - Store credentials in environment variables
   - Implement logging and monitoring
   - Test thoroughly in staging environment

#### Next Steps

Now that you've successfully made your first API call, you can continue with:

- **Section 5**: Learn more about all available endpoints
- **Section 6**: Discover product sheet functionality in detail
- **Section 9**: Implement assortment lists
- **Section 10**: Configure webhooks for real-time notifications
- **Section 12**: Best practices for production use

For more detailed information about specific endpoints, see the Swagger documentation:
Swagger UI

## 4. Authentication & Security

#### Authentication Flow

The authentication process proceeds through the following steps:

1. Login with username/password
2. Receipt of JWT token
3. Use token for subsequent calls

#### Authorization Token

Tokens contain the following information:

- User identity
- Permissions
- Validity duration

**Important:** Tokens have a limited validity duration and must be renewed via the authentication process.

---

## 5. API Endpoints & Functionality

#### Available Endpoints

### Base URL structure:

```
https://webapi.psinfoodservice.com/v7/{format}/{endpoint}/{parameters}
```

Where:

- `{format}` = 'json' or 'xml'
- `{endpoint}` = specific functionality
- `{parameters}` = additional parameters

#### Important Endpoints

### Product Sheet

```
GET /ProductSheet/{id}
GET /ProductSheet/{language}/{id}
```

### Master Data

```
GET /master/all
GET /master/{type}
```

### Updates

```
POST /Update/EAN
```

### My Products

```
GET /myproducts
```

#### Data Formats

### XML Format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <logistic>
    <!-- Logistic data -->
  </logistic>
  <product>
    <!-- General product data -->
  </product>
```

```
  <specification>
    <!-- Product specifications (optional) -->
  </specification>
</root>
```

### JSON Format

```
{
  "logistic": {
  },
  "product": {
  },
  "specification": {
  }
}
```

#### Request & Response Examples

### Check Updates

Request:

```
{
  "searchCriteria": [
    "87000000000",
    "87000000001",
    "87000000002"
  ],
  "lastUpdatedAfter": "2023-12-22",
  "isPrivateLabel": true,
  "isPubliclyVisible": true,
  "targetMarket": "All"
}
```

Response:

```
[
  {
    "LogisticId": 1690881,
    "TargetMarketId": 1,
    "LastChanged": "2023-08-24T12:53:24.823"
  }
]
```

---

## 6. Product Sheet Functionality

#### Structure

The product sheet consists of three main sections:

### 1. Logistic Section (required)

- Packaging data
- Logistic information

### 2. Product Section (required)

- General product data
- Product identification

### 3. Specification Section (optional)

- Food/Non-food specific data
- Additional product information

#### Language Support

- Default: all available languages
- Specific language via URL parameter
- Translated fields via master lists

#### Output Options

- **all**: complete product information
- **summary**: brief summary
- **productcontent**: product content only
- **logistics**: logistic data only

---

### 7. Updates & Synchronization

#### Checking Updates

The update process includes:

1. Check for changes since date
2. Comparison with existing data
3. Processing of differences
4. Feedback of results

#### Submitting Data

When submitting data, the following is checked:

- Validity of the data
- Referential integrity
- Required fields
- Data type correctness

#### Validation Process

1. Syntactic validation
2. Semantic validation
3. Business rules validation
4. Master data reference control

---

### 8. Master Data Management

#### Available Master Lists

- Units
- Allergens
- Countries
- Translations
- Packaging materials
- Additional product details

#### Usage and Implementation

1. Retrieving master lists
2. Implementation of updates
3. Processing in own system
4. Synchronization strategy

#### Caching Strategy

- Implement local cache
- Regular updates

- Version control
- Cache invalidation

---

### 9. Assortment Lists

#### Purpose and Functionality

Assortment lists make it possible to:

- Link products to customers
- Manage assortments
- Create relationships between suppliers and customers
- Manage articles structurally in categories

Customers can create and manage multiple assortment lists, add and remove articles, and request and view assortments.

#### API Endpoints

All assortment list endpoints require authentication with JWT token and the Read role. The endpoints automatically filter based on the RelationId from the token.

##### 9.1 POST /v7/{format}/Assortment/assortments

**Create a new assortment list**

Creates a new assortment list with optional items. First checks if there are still available lists for the relation.

**Parameters:**

- `{format}`: Response format (json/xml)

**Request Body:**

```
{
  "id": "guid",
  "name": "string (required)",
  "assortmentTypeId": "int (required, > 0)",
  "items": [
    {
      "id": 0,
      "articleNumber": "string",
      "articleName": "string",
      "articleBrand": "string",
      "gtince": "string",
      "gtinhe": "string",
      "relationGln": "string",
      "relationName": "string",
      "relationArticleNumber": "string"
    }
  ]
}
```

**Response Codes:**

- **200 OK**: Assortment list successfully created
- **400 Bad Request**: Invalid input data
- **409 Conflict**: "No available lists" - no more available lists

##### 9.2 GET /v7/{format}/Assortment/assortments

**Retrieve all assortment lists**

Retrieves all assortment lists belonging to the logged-in relation, including all items per list.

**Parameters:**

- `{format}` : Response format (json/xml)

**Response Codes:**

- **200 OK**: List of assortments
- **204 No Content**: No assortments found

#### 9.3 PUT /v7/{format}/Assortment/assortments/items

**Add items to an assortment list**

Adds new items to an existing assortment list. Existing items are preserved.

**Parameters:**

- `{format}` : Response format (json/xml)

**Request Body:**

```
{
  "id": "guid (required)",
  "items": [
    {
      "id": 0,
      "articleNumber": "string",
      "articleName": "string",
      "articleBrand": "string",
      "gtince": "string",
      "gtinhe": "string",
      "relationGln": "string",
      "relationName": "string",
      "relationArticleNumber": "string"
    }
  ]
}
```

**Response Codes:**

- **200 OK**: Items successfully added
- **400 Bad Request**: Invalid input data
- **409 Conflict**: "No assortment list with Id {id} found."

#### 9.4 DELETE /v7/{format}/Assortment/assortments/{id}/items

**Remove items from an assortment list**

Removes specific items from an existing assortment list. The list itself remains.

**Parameters:**

- `{format}` : Response format (json/xml)
- `{id}` : GUID of the assortment list (route parameter, required)

**Response Codes:**

- **200 OK**: Items successfully removed
- **400 Bad Request**: Invalid input data
- **409 Conflict**: "No assortment list with name {name} found."

#### 9.5 GET /v7/{format}/Assortment/assortments/{id}/items

**Retrieve a specific assortment list**

Retrieves a specific assortment list including all items and metadata.

**Parameters:**

- `{format}` : Response format (json/xml)

- `{id}` : GUID of the assortment list (required)

**Response Codes:**

- **200 OK**: Assortment list found
- **204 No Content**: No assortment list found with this ID

#### 9.6 DELETE /v7/{format}/Assortment/assortment/{id}

**Delete a complete assortment list**

Permanently deletes a complete assortment list including all items.

**Parameters:**

- `{format}` : Response format (json/xml)
- `{id}` : GUID of the assortment list (required)

**Response Codes:**

- **200 OK**: Assortment list successfully deleted
- **400 Bad Request**: Invalid ID
- **401 Unauthorized**: Not authorized
- **403 Forbidden**: No access to this list

#### Data Models

##### AssortmentDto (Output)

Represents a complete assortment list with all metadata and items.

```
public class AssortmentDto {
    public Guid Id { get; set; }
    public string Name { get; set; }
    public AssortmentTypeDto AssortmentType { get; set; }
    public List<AssortmentItemDto> Items { get; set; }
}
```

##### AssortmentInsertDto (Input)

For creating a new assortment list.

```
public class AssortmentInsertDto {
    public Guid Id { get; set; }
    public string Name { get; set; }
    public int AssortmentTypeId { get; set; }
    public List<AssortmentItemUpdateDto> Items { get; set; }
}
```

##### AssortmentItemDto

Represents an article within an assortment list.

```
public class AssortmentItemDto {
    public int Id { get; set; }
    public string ArticleNumber { get; set; }
    public string ArticleName { get; set; }
    public string ArticleBrand { get; set; }
    public string GTINCE { get; set; }
    public string GTINHE { get; set; }
    public string RelationGln { get; set; }
    public string RelationName { get; set; }
    public string RelationArticleNumber { get; set; }
}
```

##### AssortmentTypeDto

Represents a type/category of assortment with multilingual names.

```
public class AssortmentTypeDto {
    public int Id { get; set; }
    public List<TranslationDto> Name { get; set; }
}
```

#### Implementation

**Steps for implementation:**

1. **Compile List** - Determine the assortment type, collect article data, validate required fields
2. **Create List** - POST to /v7/json/Assortment/assortments, check availability, store the generated ID
3. **Manage Items** - Add items via PUT, remove via DELETE
4. **Request List** - GET for all lists or specific list
5. **Delete List** - DELETE for complete list

#### Authentication & Authorization

**Requirements for all endpoints:**

- JWT Token in Authorization header
- Role: Read role required
- Token contains: RelationId and UserId for filtering

**Token structure:**

```
Authorization: Bearer <jwt-token>
```

#### Rate Limiting & Size Limits

**POST and PUT endpoints:**

- Request size limit: **100 MB**
- Enables large bulk uploads
- Suitable for complete assortment lists

---

### 10. Webhooks

#### Overview

Webhooks provide the ability to receive real-time notifications when products in your assortment list are changed. Instead of periodically polling the API for updates, you automatically receive a push message as soon as a change occurs.

#### How Does It Work?

Whenever a product from your assortment list is updated, the Web API automatically sends a POST request to your registered webhook URL.

#### Webhook Endpoints

##### Subscribe - Register for push notifications

**Endpoint:**

```
POST /v7/{format}/Account/subscribe
```

**Description:**

Register a webhook URL to receive notifications when products change.

**Parameters:**

- `{format}` : Response format (json/xml)

**Request Body:**

```
{
  "url": "https://your-domain.com/webhook/productupdate",
  "secret": "your-secret-key"  // optional
```

```
}
```

**Response Codes:**

- **200 OK**: Webhook successfully registered
- **400 Bad Request**: Invalid URL or request
- **401 Unauthorized**: Not authorized

#### Unsubscribe - Unregister from push notifications

**Endpoint:**

```
POST /v7/{format}/Account/unsubscribe
```

**Description:**

Remove the registered webhook URL to stop receiving notifications.

**Parameters:**

- `{format}`: Response format (json/xml)

**Response Codes:**

- **200 OK**: Webhook successfully removed
- **400 Bad Request**: No webhook registered
- **401 Unauthorized**: Not authorized

#### Webhook Payload

When a product in your assortment list is changed, you will receive a POST request to your registered webhook URL with the following payload:

**Request Method:** POST

**Content-Type:** application/json

**Headers:**

```
{
  "x-secret": "your-secret-key" // if secret is specified at subscribe
}
```

**Body:**

```
{
  "logisticid": 222
}
```

#### Using Secret

When you specify a secret when subscribing, this value will be included in the x-secret header of each push message. Use this to verify the authenticity of incoming webhook messages.

#### Processing Webhook Messages

After receiving a webhook message, you can retrieve the full product data using the received `logisticid`:

**Step 1:** Receive webhook notification

```
{
  "logisticid": 222
}
```

**Step 2:** Retrieve product data via the API

```
GET
https://webapi.psinfoodservice.com/v7/json/productsheet/{logisticid}
```

```
Authorization: Bearer {your_accesstoken}
```

### Implementation Example

**Webhook Endpoint (C# / ASP.NET Core, an open-source web development framework | .NET Core)**

```
[ApiController]
[Route("webhook")]
public class WebhookController : ControllerBase
{
    private readonly IProductService _productService;
    private readonly ILogger<WebhookController> _logger;

    public WebhookController(IProductService productService, ILogger<WebhookController> logger)
    {
        _productService = productService;
        _logger = logger;
    }

    [HttpPost("productupdate")]
    public async Task<IActionResult> HandleProductUpdate([FromBody] WebhookPayload payload)
    {
        // Validate webhook secret
        if (!ValidateWebhookSecret())
        {
            _logger.LogWarning("Received webhook with invalid secret");
            return Unauthorized();
        }
        _logger.LogInformation($"Webhook received for logisticid: {payload.LogisticId}");

        try
        {
            var product = await _productService.GetProductByIdAsync(payload.LogisticId);
            await _productService.ProcessProductUpdateAsync(product);
            return Ok();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, $"Error processing webhook for logisticid: {payload.LogisticId}");
            return StatusCode(500);
        }
    }
    private bool ValidateWebhookSecret()
    {
        var expectedSecret = _configuration["Webhook:Secret"];
        if (string.IsNullOrEmpty(expectedSecret))
        {
            _logger.LogWarning("Geen webhook secret geconfigureerd");
            return true;
        }
        if (!Request.Headers.TryGetValue("x-secret", out var receivedSecret))
        {
            return false;
        }
        return CryptographicOperations.FixedTimeEquals(
            Encoding.UTF8.GetBytes(expectedSecret),
            Encoding.UTF8.GetBytes(receivedSecret.ToString())
        );
    }
}

public class WebhookPayload
{
    public int LogisticId { get; set; }
}
```

**Webhook Endpoint (Node.js / Express)**

```
const express = require('express');
const crypto = require('crypto');
const app = express();

app.use(express.json());

// Configuration
const WEBHOOK_SECRET = process.env.WEBHOOK_SECRET || 'your-secret-key';
const ACCESS_TOKEN = process.env.ACCESS_TOKEN;

// Middleware voor webhook secret verificatie
```

```
function validateWebhookSecret(req, res, next) {
    const receivedSecret = req.headers['x-secret'];
    if (!WEBHOOK_SECRET) {
        console.warn('No webhook secret configured');
        return next();
    }
    if (!receivedSecret) {
        console.warn('Webhook ontvangen zonder secret header');
        return res.status(401).send('Unauthorized');
    }
    const expectedBuffer = Buffer.from(WEBHOOK_SECRET);
    const receivedBuffer = Buffer.from(receivedSecret);
    if (expectedBuffer.length !== receivedBuffer.length ||
        !crypto.timingSafeEqual(expectedBuffer, receivedBuffer)) {
        console.warn('Received webhook with invalid secret');
        return res.status(401).send('Unauthorized');
    }
    next();
}

app.post('/webhook/productupdate', async (req, res) => {
    const { logisticid } = req.body;

    console.log(`Webhook received for logisticid: ${logisticid}`);

    try {
        const productResponse = await fetch(
`https://webapi.psinfoodservice.com/v7/json/productsheet/${logisticid}`,
            {
                headers: {
                    'Authorization': `Bearer ${accessToken}`,
                    'Content-Type': 'application/json'
                }
            }
        );
        if (!productResponse.ok) {
            throw new Error(`API error: ${productResponse.status}`);
        }

        const productData = await productResponse.json();
        await processProductUpdate(productData);

        res.status(200).send('OK');
    } catch (error) {
        console.error(`Error processing webhook: ${error.message}`);
        res.status(500).send('Error');
    }
});

app.listen(3000, () => {
    console.log('Webhook server running on port 3000');
});

async function processProductUpdate(productData) {
    // Implementeer hier uw eigen logica
    console.log('Product update verwerken:', productData);
}

**Environment variables (.env bestand):**
```
WEBHOOK_SECRET=uw-geheime-sleutel
ACCESS_TOKEN=uw-access-token
```

Best Practices for Webhooks

1. **Security**
   - Use HTTPS for your webhook endpoint
   - Validate incoming requests
   - Implement rate limiting on your endpoint
2. **Reliability**
   - Return a 200 OK response quickly
   - Process the actual update asynchronously
   - Implement a queue for incoming webhooks
3. **Error Handling**
   - Log all incoming webhook requests
   - Implement retry logic for failed API calls
```

- Monitor your webhook endpoint for errors

4. **Idempotency**
   - Design your processing to be idempotent (processing the same update multiple times gives the same result)
   - Track which updates have already been processed
5. **Timeout Handling**
   - Respond within 30 seconds to webhook requests
   - Use background processing for long-running operations

---

### 11. Error Handling

#### HTTP Status Codes

| Code | Meaning | When |
|---|---|---|
| 200 | OK | Successful operation |
| 204 | No Content | No data found |
| 400 | Bad Request | Invalid request or parameters |
| 401 | Unauthorized | Not authorized, token missing/invalid |
| 403 | Forbidden | Access denied, insufficient rights |
| 404 | Not Found | Resource not found |
| 409 | Conflict | Conflict with existing data |
| 500 | Internal Server Error | Server error |

#### Error Messages

Error messages contain:

- Unique error code
- Descriptive message in clear language
- Additional details about the error
- Logging reference for support

**Example error response:**

```
{
  "error": "ValidationError",
  "message": "No assortment list with Id {guid} found.",
  "statusCode": 409,
  "timestamp": "2024-01-15T10:30:00Z"
}
```

#### Troubleshooting

**Common problems:**

**1. Connection problems**

- Check network connectivity
- Verify SSL/TLS configuration

- Check firewall settings
- Test with ping/traceroute

**2. Authentication problems**

- Check API key/token validity
- Verify expired credentials
- Check IP whitelist settings
- Validate user rights

**3. Validation errors**

- Check required fields
- Validate data types
- Check data format (JSON/XML)
- Verify character encoding

**4. Rate limiting**

- Monitor number of API calls
- Implement backoff strategy
- Use caching where possible
- Optimize batch sizes

---

## 12. Best Practices

#### Implementation Guidelines

1. **Use the right endpoints**
   - Choose the right HTTP method (GET/POST/PUT/DELETE)
   - Use correct URL structure
   - Specify desired format (json/xml)
2. **Implement error handling**
   - Try-catch blocks
   - Retry logic for network errors
   - Logging of errors
   - User-friendly error messages
3. **Validate input data**
   - Client-side validation
   - Check required fields
   - Validate data types
   - Sanitize user input
4. **Monitor performance**
   - Track response times
   - Log API call volume
   - Monitor error rates
   - Set up alerts
5. **Document changes**
   - Code comments
   - API documentation

- Change logs
- Version control

#### Performance Optimization

1. **Implement caching**
   - Cache master data locally
   - Use HTTP caching headers
   - Implement client-side cache
   - Set appropriate TTL
2. **Limit number of calls**
   - Batch requests where possible
   - Use pagination
   - Filter data server-side
   - Minimize payload size
3. **Optimize data traffic**
   - Use compression (gzip)
   - Minimize response data
   - Select only needed fields
   - Use efficient data formats
4. **Monitor response times**
   - Set performance baselines
   - Track trends over time
   - Identify bottlenecks
   - Optimize slow endpoints

#### Caching Strategies

**Master data caching:**

- Cache duration: 24 hours
- Update trigger: version control
- Invalidation: on new version

**Product data caching:**

- Cache duration: 1-4 hours
- Update trigger: timestamp control
- Invalidation: on change detection

**Token management:**

- Refresh before expiration
- Secure storage
- Auto-renewal
- Error recovery

**Cache invalidation:**

- Time-based (TTL)
- Event-based (updates)
- Manual (force refresh)
- Version-based (API updates)

## 13. Support

#### Contact Information

**Email technical questions:** ict@psinfoodservice.com

**Email commercial questions:** info@psinfoodservice.com

**Phone:** +31 085 044 18 96

**Support hours:**

- Monday to Friday: 09:00 - 17:00 CET
- Emergency support: via email

#### FAQ

Frequently asked questions and answers are available on the support website.

**Common topics:**

- Authentication problems
- Rate limiting
- Data format questions
- Implementation tips
- Error handling

#### Additional Resources

**Swagger documentation:**
Swagger UI

**Available resources:**

- Postman collection for testing
- Code examples (C#, PHP, Python)
- Implementation guides

---

## Appendix

#### Glossary

| Term | Definition |
| --- | --- |
| JWT | JSON Web Token - authentication token format |
| GTIN | Global Trade Item Number - product identification |
| GLN | Global Location Number - location identification |
| EAN | European Article Number - barcode standard |
| TTL | Time To Live - cache validity duration |

| | |
|---|---|
| DTO | Data Transfer Object - data transport model |

---

*Last update: Jan, 2026*

*Documentation version: 1.1.0*

*API version: 7.0.0.1*