ShelfXplore API Documentation

1. Introduction

Overview: ShelfXplore is a backend API service designed to leverage computer vision for retail environments. The application helps detect, classify, and localize products on store shelves by analyzing images of the shelves provided by the client.

Target Audience: This documentation is intended for developers integrating ShelfXplore into retail applications, as well as those maintaining the system.

Purpose of the Documentation: The purpose of this documentation is to provide an overview of how to use the ShelfXplore API, detailing the structure of API calls, expected inputs and outputs, and how to interpret the responses.

---

2. Getting Started

2.1. Prerequisites

- A valid **API key** to authenticate requests.
- Basic understanding of HTTP requests (GET, POST).
- Knowledge of JSON format.

2.2. Authentication

All requests to the ShelfXplore API require an **API Key** for authentication. The API key must be included in the `Authorization` header as a Bearer token.

---

3. API Endpoints

3.1. POST /startSession

This endpoint allows you to start a session with our server.

Request

Headers:

Authorization: Bearer YOUR_API_KEY
Content-Type: application/json

Body: The request body must contain the following JSON data:

```
{
  ‘sessionId’: ‘sessionid_12345’
  ‘categoryCodes’: [‘1234’, ‘5678’, ‘1234’, ‘5645’]
}
```

- `sessionId`: Unique session ID
- `categoryCodes`: An array of product categories to use for classification. Each product category corresponds to the specific classification model (similar products are classified by the same ML model).
  Category codes should be passed as strings.

Response

If the request is successful, the API will return a 200 code and you may proceed with the next API call.

3.2. POST /addImageToSession

This endpoint allows you to submit an image of a retail shelf for product detection, classification, and localization.

Request

Headers:

Authorization: Bearer YOUR_API_KEY
Content-Type: application/json

Body: The request body must contain the following JSON data:

```
{
  ‘sessionId’: ‘sessionid_12345’
  ‘imageId’ : ‘imageid_12345’
  ‘image’ : b64_image
```

‘roi’ : [x, y, w, h]
}

- `sessionId`: Unique session ID, same as in the previous StartSession API call
- `imageID`: Unique image ID
- `image`: The image to be analyzed, encoded in base64 format.
- `roi`: Rectangular area of the photo to be analyzed defined by top left starting coordinates, width and height.
    - `x`, `y`: Coordinates of the top-left corner of the region of interest.
    - `width`, `height`: The dimensions of the region of interest.

Response

If the request is successful, the API will return a 200 code and you may proceed with the next API call.

3.3. GET /getSessionImageResult

This endpoint allows you to probe our server in loop, for the result of the specific session, if the last two calls (startSession, addImageToSession) were successful.

Request

Headers:

Authorization: Bearer YOUR_API_KEY
Content-Type: application/json

Body: The request body must contain the following JSON data:

{
‘sessionId’: ‘sessionid_12345’
‘imageId’ : ‘imageid_12345’
}

- `sessionId`: Unique session ID, same as in the previous StartSession and AddImageToSession API calls
- `imageID`: Unique image ID, same as in the previous AddImageToSession API call

Response

If the request is successful, the API will return a JSON object containing the results of the analysis:


```
{
  "status": "success",
  "data": {
    "products": [
      {
        "uuid": "product_12345",
        "bounding_box": {
          "x": 100,
          "y": 150,
          "width": 50,
          "height": 100
        },
        "shelf_index": 0,
        "classification_accuracy": 0.92
      },
      {
        "uuid": "product_67890",
        "bounding_box": {
          "x": 200,
          "y": 250,
          "width": 75,
          "height": 150
        },
        "shelf_index": 1,
        "classification_accuracy": 0.85
      }
    ]
  }
}
```


- `uuid`: A unique identifier for each detected product.
- `bounding_box`: The location of the detected product on the shelf (coordinates in pixels).
    - `x`, `y`: Coordinates of the top-left corner of the bounding box.
    - `width`, `height`: The dimensions of the bounding box.
- `shelf_index`: The index of the shelf where the product was detected.
- `classification_accuracy`: The confidence score of the classification (range 0-1).

Error Response

If the request is unsuccessful, you will receive an error message:

```
{
  "status": "error",
  "message": "Invalid image format"
}
```

Error codes and descriptions:

- `400`: Bad Request (e.g., missing parameters, incorrect format).
- `401`: Unauthorized (invalid or missing API key).
- `500`: Internal Server Error (e.g., server-side issues).

---

4. Request and Response Details

4.1. Image Format

The image should be provided in base64-encoded format. The image must meet the following criteria:

- Supported formats: PNG, JPEG, or JPG.
- Maximum image size: 5 MB.

4.3. Response Time

- The average response time for an image analysis is approximately **10 seconds**.

---

5. Error Codes and Troubleshooting

- **400 Bad Request**: The request is malformed or missing required parameters.
    - **Example**: Missing `image` or `categories` field.
    - **Solution**: Ensure the image is base64 encoded and the categories are specified correctly.
- **401 Unauthorized**: Invalid API key or missing authentication.
    - **Solution**: Ensure the `Authorization` header includes the correct API key.

- **500 Internal Server Error**: An error occurred on the server side.
    - **Solution**: Retry the request or contact support if the issue persists.

---

6. Database and model training

6.1. Database

Clients can map their existing product database with our own, or they can use our database from scratch. The only restriction is that each of the product categories must correspond to its respective classification model.

6.2. Adding and training new products

- **Product Addition & Basic Classification**:
    - New products are added to the database.
    - 'Thumbnail' images of the products are used to perform initial basic classification.
- **Data Collection for Model Improvement**:
    - During the training period, on-shelf product images are gathered and added to the datasets.
    - These real-world images are crucial for improving the model's accuracy and handling real-world variability.
    - The initial training process generally takes **2-3 weeks** to achieve a classification accuracy above **95%**
- **Manual Quality Control with Admin/Annotator Interface**:
    - We use an **Admin/Annotator interface** to manually review and, if necessary, reclassify on-shelf product images before adding them to the datasets.
    - This step ensures high-quality datasets by addressing any misclassifications and refining the model's understanding of product categories.

6.3. Retraining classification models

- **Retraining the Classification Model**:
    - The manually verified and updated images are used to retrain the classification model, incorporating new product classes.
- **Model Retraining Frequency**:
    - Models are retrained periodically to ensure optimal performance.
    - Alternatively, models can be retrained **upon client request**, allowing for timely updates based on new product data or changes in the client's requirements.

This approach ensures that the system is continuously improving, staying up to date with the client's evolving product catalog, and maintaining high accuracy in product classification.

---

7. Versioning and Changelog

7.1. Current Version

- **Version**: 1.0
- **Release Date**: January 2025

7.2. Changelog

- **Version 1.0** (January 2025):
    - Initial release.
