CelestialEye API Documentation

Overview

This documentation describes the REST APIs and WebSocket endpoints provided by the CelestialEye backend service. The API offers various computer vision capabilities including barcode detection, QR code scanning, OCR, face recognition, and general object detection.

Base URL

All API URLs referenced in this documentation have the base URL: `/api`

Authentication

Cross-Origin Resource Sharing (CORS) is enabled for all origins (*).

Common Response Format

Most endpoints return responses in the following format:

```
{
  "success": true|false,
  "data": {...},  // Response data when success is true
  "error": "..."   // Error message when success is false
}
```

API Endpoints

Barcode Detection

Process Document

- **URL**: `/api/barcode/process`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **Parameters**:
  - `file` (required): PDF, TIFF, or image file containing barcodes
- **Response**:

```
{
  "success": true,
  "totalPages": 1,
  "detectedPages": 1,
  "totalBarcodes": 2,
  "results": [
```

```
    {
      "pageNumber": 1,
      "barcodes": [
        {
          "content": "123456789",
          "format": "CODE_128",
          "confidence": 0.95,
          "boundingBox": {
            "x1": 100,
            "y1": 100,
            "x2": 200,
            "y2": 150
          }
        }
      ]
    }
  ]
}
```


QR Code Detection

Detect QR Codes

- **URL**: `/api/qrcode/detect`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **Parameters**:
  - `image` (required): Image file containing QR codes
- **Response**:


```
{
  "success": true,
  "data": {
    "imageInfo": "image.jpg (800x600)",
    "totalQrCodes": 1,
    "qrCodes": [
      {
        "content": "https://example.com",
        "type": "URL",
        "boundingBox": {
          "x1": 100,
          "y1": 100,
          "x2": 200,
          "y2": 200
        }
      }
    ],
    "processingTimeMs": 150
```

```
  }
}
```

Get QR Detection Status

- **URL**: `/api/qrcode/status`
- **Method**: `GET`
- **Response**:

```
{
  "success": true,
  "status": "QR Code detection service is running",
  "library": "BoofCV",
  "capabilities": [
    "QR Code detection",
    "Multi-QR support",
    "Bounding box coordinates",
    "Content extraction"
  ]
}
```

OCR (Optical Character Recognition)

Perform OCR Detection

- **URL**: `/api/ocr/detect/{modelName}`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **Parameters**:
  - `modelName` (path parameter): Name of the OCR model to use
  - `image` (required): Image file to perform OCR on
- **Response**:

```
{
  "success": true,
  "data": {
    "detections": [
      {
        "text": "Sample text",
        "confidence": 0.95,
        "boundingBox": {
          "x1": 100,
          "y1": 100,
          "x2": 200,
          "y2": 150
        }
```

```
        }
      ]
    }
  }
```

Face Recognition

Register Face

- **URL**: `/api/face/register`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **Parameters**:
  - `image` (required): Image file containing the face
  - `name` (required): Name of the person
- **Response**:

```
{
  "personName": "John Doe",
  "timestamp": "2023-12-20T10:30:00Z"
}
```

Authenticate Face

- **URL**: `/api/face/authenticate`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **Parameters**:
  - `image` (required): Image file containing the face to authenticate
- **Response**:

```
{
  "authenticated": true,
  "personName": "John Doe",
  "timestamp": "2023-12-20T10:30:00Z"
}
```

List Registered Faces

- **URL**: `/api/face/list`
- **Method**: `GET`
- **Response**: Array of registered face data

Object Detection

Detect Objects

- **URL**: `/api/detection/detect/{modelName}`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **Parameters**:
  - `modelName` (path parameter): Name of the detection model to use
  - `image` (required): Image file to perform detection on
  - `classNames` (optional): Comma-separated list of class names
  - `confThreshold` (optional): Confidence threshold value
- **Response**:

```
{
  "success": true,
  "modelName": "yolo_v8",
  "imageName": "test.jpg",
  "imageWidth": 800,
  "imageHeight": 600,
  "processingTime": 150,
  "detections": [
    {
      "class": "person",
      "confidence": 0.95,
      "box": {
        "x1": 100,
        "y1": 100,
        "x2": 300,
        "y2": 400
      }
    }
  ]
}
```

List Available Models

- **URL**: `/api/detection/models`
- **Method**: `GET`
- **Response**:

```
{
  "success": true,
  "models": [
    {
      "name": "yolo_v8",
      "description": "Object detection model"
    }
```

```
  ]
}
```

Clear Model Cache

- **URL**: `/api/detection/cache/{modelName}`
- **Method**: `DELETE`
- **Parameters**:
  - `modelName` (path parameter): Name of the model to clear cache for
- **Response**:

```
{
  "message": "Cache cleared for model: yolo_v8"
}
```

Clear All Cache

- **URL**: `/api/detection/cache`
- **Method**: `DELETE`
- **Response**:

```
{
  "message": "All cache cleared"
}
```

Health Check

- **URL**: `/api/detection/health`
- **Method**: `GET`
- **Response**: Health status of the detection service

Model Management

Upload Model

- **URL**: `/api/models/upload`
- **Method**: `POST`
- **Content-Type**: `multipart/form-data`
- **Parameters**:
  - `file` (required): ONNX model file
  - `name` (required): Model name
  - `description` (optional): Model description
- **Example Request**:

```
curl -X POST -F "file=@model.onnx" -F "name=my_yolo_model" -F
"description=YOLOv8 model" /api/models/upload
```

List Models

- **URL**: `/api/models/list`
- **Method**: `GET`
- **Response**: List of uploaded models

Get Model Information

- **URL**: `/api/models/{modelName}`
- **Method**: `GET`
- **Parameters**:
  - `modelName` (path parameter): Name of the model
- **Response**: Detailed model information

Delete Model

- **URL**: `/api/models/{modelName}`
- **Method**: `DELETE`
- **Parameters**:
  - `modelName` (path parameter): Name of the model to delete
- **Response**: Deletion confirmation

WebSocket Endpoints

QR Code & CCCD Detection Stream

- **URL:** `/ws/detect-stream`
- **Protocol:** WebSocket
- **Description:** Stream video frames for real-time QR code and CCCD detection

Example Usage:

```
const ws = new WebSocket('ws://your-server/ws/detect-stream');

// Send video frame
ws.send(imageBlob); // Send binary image data

// Receive detection results
ws.onmessage = (event) => {
    const response = JSON.parse(event.data);
    // Example response:
    // {
    //     "success": true,
```

```
    //     "data": {
    //         "totalQrCodes": 1,
    //         "results": [{
    //             "content": "123456789",
    //             "type": "CCCD",
    //             "cccdInfo": {
    //                 "id": "123456789",
    //                 "name": "Nguyen Van A",
    //                 "birth": "01/01/1990"
    //             }
    //         }]
    //     }
    // }
};
```

Face Recognition Stream

- **URL:** `/ws/face-stream`
- **Protocol:** WebSocket
- **Description:** Stream video frames for real-time face registration and authentication

Commands:

1. Set Registration Mode:

```
ws.send(JSON.stringify({
    action: "setMode",
    mode: "register",
    personName: "John Doe"
}));
```

2. Set Authentication Mode:

```
ws.send(JSON.stringify({
    action: "setMode",
    mode: "authenticate"
}));
```

Example Usage:

```
const ws = new WebSocket('ws://your-server/ws/face-stream');

// Set mode first
ws.send(JSON.stringify({
    action: "setMode",
    mode: "register",
```

```
    personName: "John Doe"
}));

// Send video frame
ws.send(imageBlob); // Send binary image data

// Receive results
ws.onmessage = (event) => {
    const response = JSON.parse(event.data);
    // Example registration response:
    // {
    //     "success": true,
    //     "action": "register",
    //     "data": {
    //         "id": "123e4567-e89b-12d3-a456-426614174000",
    //         "personName": "John Doe",
    //         "timestamp": 1624233600000
    //     }
    // }

    // Example authentication response:
    // {
    //     "success": true,
    //     "action": "authenticate",
    //     "authenticated": true,
    //     "personName": "John Doe",
    //     "timestamp": 1624233600000
    // }
};
```

Barcode Detection Stream

- **URL:** `/ws/barcode-stream`
- **Protocol:** WebSocket
- **Description:** Stream video frames for real-time barcode detection

Example Usage:

```
const ws = new WebSocket('ws://your-server/ws/barcode-stream');

// Send video frame
ws.send(imageBlob); // Send binary image data

// Receive detection results
ws.onmessage = (event) => {
    const response = JSON.parse(event.data);
    // Example response:
    // {
    //     "success": true,
    //     "totalPages": 1,
```

```
    //     "detectedPages": 1,
    //     "totalBarcodes": 2,
    //     "results": [{
    //         "pageNumber": 1,
    //         "barcodes": [{
    //             "content": "123456789",
    //             "format": "CODE_128",
    //             "confidence": 0.95
    //         }]
    //     }]
    // }
};
```

Common WebSocket Considerations

1. **Error Handling:**
    - Always implement error handling for WebSocket connections
    - Handle connection drops and implement reconnection logic
    - Parse JSON responses in try-catch blocks
2. **Performance Optimization:**
    - Compress image data when possible
    - Consider reducing frame rate if bandwidth is limited
    - Implement rate limiting on the client side
3. **Security:**
    - Use wss:// (WebSocket Secure) in production
    - Implement authentication if required
    - Validate all incoming and outgoing data
4. **Best Practices:**
    - Close WebSocket connections when they're no longer needed
    - Implement heartbeat mechanism for long-running connections
    - Handle cleanup in window.onbeforeunload

Error Handling

All endpoints return appropriate HTTP status codes:

- 200: Success
- 400: Bad Request (invalid parameters)
- 404: Not Found
- 500: Internal Server Error

Error responses include a descriptive message in the `error` field.

Rate Limiting

Currently, there are no rate limits implemented on the API endpoints.

Support

For technical support or questions about the API, please refer to the repository documentation or open an issue.