Frontend API Reference

Overview

The frontend communicates with the backend Driver Drowsiness Detection API via HTTP requests. This document details all API endpoints and their usage.

Base URL

```
http://localhost:8000
```

For production, update the base URL in:

- `src/App.jsx` (line ~14)
- `src/components/ImageUpload.jsx` (line ~43)

Endpoints

1. Health Check

Check if the backend API is running.

Endpoint: `GET /health`

Usage Location: Optional health check

Request:

```
fetch('http://localhost:8000/health')
```

Response:

```
{
  "status": "ok"
}
```

Status Code: 200

---

2. Image Analysis

Analyze a single image for drowsiness and yawning detection.

Endpoint: `POST /analyze`

Usage Location: `src/components/ImageUpload.jsx` (handleAnalyze function)

Request Headers:

```
Content-Type: multipart/form-data
```

Request Body:

```
FormData {
  file: File,              // Image file
  ear_threshold: float,    // Optional (default: 0.25)
  mar_threshold: float     // Optional (default: 0.70)
}
```

Usage Code:

```
const formData = new FormData()
formData.append('file', selectedFile)

const response = await fetch('http://localhost:8000/analyze', {
  method: 'POST',
  body: formData,
})

const data = await response.json()
```

Response:

```
{
  "ear": 0.35,
  "mar": 0.45,
  "drowsy": false,
  "yawning": false,
  "annotated_image": "..."
}
```

Response Fields:

- `ear` (float): Eye Aspect Ratio value (0.0 - 1.0)
- `mar` (float): Mouth Aspect Ratio value (0.0 - 1.0)
- `drowsy` (boolean): true if ears < ear_threshold
- `yawning` (boolean): true if MAR > mar_threshold
- `annotated_image` (string): Base64-encoded PNG image with annotations

Status Codes:

- 200: Success
- 400: Invalid image format
- 500: Server error

Thresholds:

- **EAR Threshold**: 0.25 (default)
  - Below 0.25 = Drowsy
- **MAR Threshold**: 0.70 (default)
  - Above 0.70 = Yawning

Image Requirements:

- Format: JPEG, PNG, BMP, GIF
- Size: < 10MB
- Contains at least one face
- Good lighting recommended

---

3. Live Status

Get current webcam analysis status in real-time.

Endpoint: `GET /status`

Usage Location: `src/App.jsx` (useEffect hook, polling)

Polling Interval: 500ms

Request:

```
const response = await fetch('http://localhost:8000/status')
const data = await response.json()
```

Response:

```
{
  "ear": 0.42,
  "mar": 0.38,
  "drowsy": false,
  "yawning": false
}
```

Response Fields:

- `ear` (float): Current Eye Aspect Ratio

- `mar` (float): Current Mouth Aspect Ratio
- `drowsy` (boolean): Current drowsiness status
- `yawning` (boolean): Current yawning status

Status Code: 200

Notes:

- Backend must be running webcam loop for accurate data
- Requires webcam access on server machine
- Updates continuously while webcam is active

Error Handling

Network Errors

```
try {
  const response = await fetch(url)
  if (!response.ok) {
    throw new Error(`HTTP ${response.status}`)
  }
  const data = await response.json()
} catch (error) {
  console.error('API Error:', error)
  // Display user-friendly error message
}
```

Common Error Codes

Code	Meaning	Solution
400	Bad Request	Check image format, ensure valid file
404	Not Found	Check endpoint URL, verify backend running
500	Server Error	Check backend logs, restart server
Connection Error	Backend not running	Start backend: `uvicorn main:app --reload`

CORS Errors

If you see CORS errors in browser console:

Error: `Access to XMLHttpRequest at 'http://localhost:8000/analyze' from origin 'http://localhost:5173' has been blocked by CORS policy`

Solution: Add CORS middleware to backend

```python
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5173", "http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Rate Limiting

Currently no rate limiting implemented. For production, consider:

```python
from slowapi import Limiter
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)

@app.post("/analyze")
@limiter.limit("30/minute")
async def analyze(request: Request, ...):
    ...
```

Authentication (Future)

If you add authentication:

```javascript
const response = await fetch('http://localhost:8000/analyze', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${token}`
  },
  body: formData
})
```

Performance Metrics

Typical Response Times

Operation	Time	Notes

Operation	Time	Notes
/health	10ms	Simple health check
/analyze	500-2000ms	Depends on image size & ML model
/status	50ms	Real-time data fetch

Optimization Tips

1. **Image Analysis**:
    - Compress images before upload
    - Use smaller image dimensions
    - Batch multiple analyses on backend
2. **Live Monitoring**:
    - Adjust polling interval (currently 500ms)
    - Skip frames if needed
    - Use caching headers

Data Structure Examples

Drowsy Detection Example

```
{
  "ear": 0.18,
  "mar": 0.45,
  "drowsy": true,
  "yawning": false,
  "annotated_image": "..."
}
```

Yawning Detection Example

```
{
  "ear": 0.38,
  "mar": 0.82,
  "drowsy": false,
  "yawning": true,
  "annotated_image": "..."
}
```

Alert State (Critical)

```
{
  "ear": 0.15,
  "mar": 0.85,
  "drowsy": true,
  "yawning": true,
  "annotated_image": "..."
}
```

Testing API Manually

Using cURL

```
# Health check
curl http://localhost:8000/health

# Image analysis
curl -X POST -F "file=@image.jpg" http://localhost:8000/analyze

# Status
curl http://localhost:8000/status
```

Using Postman

1. **Health Check**
   - Method: GET
   - URL: `http://localhost:8000/health`
2. **Image Analysis**
   - Method: POST
   - URL: `http://localhost:8000/analyze`
   - Body: form-data
     - Key: file, Value: [Select image file]
3. **Status**
   - Method: GET
   - URL: `http://localhost:8000/status`

Using Python

```
import requests

# Image analysis
with open('image.jpg', 'rb') as f:
```

```
    files = {'file': f}
    response = requests.post('http://localhost:8000/analyze', files=files)
    data = response.json()
    print(data)

# Status
response = requests.get('http://localhost:8000/status')
data = response.json()
print(data)
```

Frontend Implementation Details

ImageUpload.jsx Integration

```
const handleAnalyze = async () => {
  try {
    const formData = new FormData()
    formData.append('file', selectedFile)

    const response = await fetch('http://localhost:8000/analyze', {
      method: 'POST',
      body: formData,
    })

    if (!response.ok) {
      throw new Error(`Error: ${response.status}`)
    }

    const data = await response.json()
    setAnalysisResult(data)
  } catch (err) {
    setError(`Failed to analyze image: ${err.message}`)
  }
}
```

App.jsx Integration (Polling)

```
useEffect(() => {
  if (activeTab === 'webcam') {
    statusIntervalRef.current = setInterval(async () => {
      try {
        const response = await fetch('http://localhost:8000/status')
        if (response.ok) {
          const data = await response.json()
          setLiveStatus(data)
        }
      } catch (error) {
```

```
          console.error('Failed to fetch status:', error)
        }
      }, 500)  // Poll every 500ms
    }

    return () => {
      if (statusIntervalRef.current) {
        clearInterval(statusIntervalRef.current)
      }
    }
  }, [activeTab])
```

Security Considerations

Input Validation

- Image file validation (size, format)
- No sensitive data in API responses
- No authentication stored locally

HTTPS/TLS

For production:

```
const API_URL = 'https://api.yourdomain.com'
// HTTPS required for webcam in production
```

Best Practices

- Validate all API responses
- Handle errors gracefully
- Don't expose API URLs in client-side code
- Use environment variables
- Implement rate limiting
- Add authentication if needed

Webhooks (Future)

For real-time alerts:

```
@app.post("/webhook/drowsiness-alert")
async def drowsiness_webhook(alert: DrowsinessAlert):
    # Send notification
    await send_alert(alert)
```

Versioning

Current API Version: **1.0**

For future versions:

```
const API_URL = 'http://localhost:8000/api/v1'
const response = await fetch(`${API_URL}/analyze`)
```

Support

For API issues:

1. Check backend logs: `uvicorn main:app --reload`
2. Verify endpoints with curl/Postman
3. Check browser console for network errors
4. Ensure CORS is configured correctly

Last Updated: December 2024
API Version: 1.0
Status: Stable