ShotStack Integration Guide for Basketball Analysis

Quick Start Guide for Developers

🚀 Quick Start

1. Install Dependencies

```
pip install requests python-dotenv
```

2. Load Environment Variables

```
from dotenv import load_dotenv
load_dotenv('.env.shotstack')
```

3. Basic Usage

```
from shotstack_helpers import ShotStackClient, BasketballVideoEditor

# Initialize
client = ShotStackClient(environment='sandbox')
editor = BasketballVideoEditor(client)

# Create analysis video
response = editor.create_shooting_form_analysis(
    video_url="https://example.com/shot.mp4",
    annotations=[{'text': 'Good form!', 'start': 0, 'length': 2}],
    duration=5.0
)

# Get render ID
render_id = response['response']['id']
print(f"Rendering: {render_id}")

# Wait for completion
status = client.wait_for_render(render_id)
video_url = status['response']['url']
print(f"Done: {video_url}")
```

🏀 Basketball Analysis Workflow

Complete Pipeline

```python
from shotstack_helpers import create_basketball_analysis_video
import json

# 1. Get video from user upload
video_path = "https://storage.example.com/user_shot.mp4"

# 2. Run pose estimation (RoboFlow)
# This would come from your RoboFlow integration
skeleton_data = {
    'keypoints': [...],  # From pose estimation
    'confidence': 0.95
}

# 3. Calculate angles and generate feedback
analysis_results = {
    'feedback': [
        'Excellent elbow alignment at 90°',
        'Good knee bend for power',
        'Follow through could be smoother'
    ],
    'angles': {
        'elbow': {
            'value': 90.5,
            'position': {'x': 0.2, 'y': 0.0},
            'status': 'good'
        },
        'knee': {
            'value': 135.0,
            'position': {'x': 0.1, 'y': 0.3},
            'status': 'good'
        },
        'release': {
            'value': 45.0,
            'position': {'x': 0.3, 'y': -0.2},
            'status': 'needs_work'
        }
    }
}

# 4. Create analysis video
output_url = create_basketball_analysis_video(
    video_path=video_path,
    skeleton_data=skeleton_data,
    analysis_results=analysis_results,
    output_path="analysis.mp4",
    environment='sandbox'  # Use 'production' when ready
)

print(f"Analysis complete: {output_url}")
```

📋 Common Use Cases

Use Case 1: Add Coaching Annotations

```python
from shotstack_helpers import BasketballVideoEditor, ShotStackClient

client = ShotStackClient(environment='sandbox')
editor = BasketballVideoEditor(client)

annotations = [
    {
        'text': 'Setup Phase - Good stance',
        'start': 0,
        'length': 1.5,
        'position': 'bottom',
        'color': '#00ff00',
        'background': '#000000'
    },
    {
        'text': 'Release - Watch elbow alignment',
        'start': 1.5,
        'length': 1.5,
        'position': 'bottom',
        'color': '#ffff00',
        'background': '#000000'
    },
    {
        'text': 'Follow Through - Excellent!',
        'start': 3.0,
        'length': 2.0,
        'position': 'bottom',
        'color': '#00ff00',
        'background': '#000000'
    }
]

response = editor.create_shooting_form_analysis(
    video_url="https://example.com/shot.mp4",
    annotations=annotations,
    duration=5.0
)
```

Use Case 2: Show Angle Measurements

```
angles = [
    {
        'name': 'Elbow Angle',
        'value': 90.5,
        'position': {'x': 0.2, 'y': 0.0},
        'start': 0,
        'length': 5
    },
    {
        'name': 'Knee Bend',
        'value': 135.0,
        'position': {'x': 0.1, 'y': 0.3},
        'start': 0,
        'length': 5
    },
    {
        'name': 'Release Angle',
        'value': 45.0,
        'position': {'x': 0.3, 'y': -0.2},
        'start': 2,
        'length': 3
    }
]

response = editor.create_shooting_form_analysis(
    video_url="https://example.com/shot.mp4",
    angles=angles,
    duration=5.0
)
```

Use Case 3: Before/After Comparison

```
response = editor.create_split_screen_comparison(
    video1_url="https://example.com/before_coaching.mp4",
    video2_url="https://example.com/after_coaching.mp4",
    title1="Before Coaching",
    title2="After 2 Weeks",
    duration=5.0
)

render_id = response['response']['id']
status = client.wait_for_render(render_id)
comparison_url = status['response']['url']
```

Use Case 4: Add Skeleton Overlay

```
# First, generate skeleton overlay image from pose estimation
skeleton_overlay_url = "https://i.ytimg.com/vi/6IW6oImq3RM/maxresdefault.jpg?sqp=-oay-mwEmCIAKENAF8quKqQMa8AEB-AH-CYAC0AWKAgwIABABGH8gEy-gUMA8=&rs=AOn4CLCSnzQYTmpvjjnXUTD99kqG9AmxOg"

response = editor.create_shooting_form_analysis(
    video_url="https://example.com/shot.mp4",
    skeleton_overlay_url=skeleton_overlay_url,
    annotations=[
        {'text': 'Pose estimation overlay', 'start': 0, 'length': 2}
    ],
    duration=5.0
)
```

🎨 Customization Options

Position Values

Positions are relative to the video frame:
- `"top"` - Top of frame
- `"center"` - Center of frame
- `"bottom"` - Bottom of frame
- `"left"` - Left side
- `"right"` - Right side

Offset Values

Fine-tune positioning with offset:

```
offset = {
    'x': 0.2,   # 20% right from center (-1 to 1)
    'y': -0.1   # 10% up from center (-1 to 1)
}
```

Color Codes

Use hex color codes:
- `#00ff00` - Green (good)
- `#ffff00` - Yellow (warning)
- `#ff0000` - Red (needs work)
- `#ffffff` - White
- `#000000` - Black

Font Families

Available fonts:
- Arial
- Helvetica
- Times New Roman
- Courier
- Verdana

- Georgia
- Comic Sans MS
- Impact
- Permanent Marker (bold, stylized)

🔄 Integration with RoboFlow

Step 1: Get Pose Estimation Data

```python
from roboflow import Roboflow

# Initialize RoboFlow
rf = Roboflow(api_key="your_api_key")
project = rf.workspace().project("basketball-pose")
model = project.version(1).model

# Run inference
result = model.predict("shot.mp4")

# Extract keypoints
keypoints = result.json()['predictions'][0]['keypoints']
```

Step 2: Calculate Angles

```python
import math

def calculate_angle(p1, p2, p3):
    """Calculate angle between three points"""
    a = math.sqrt((p2['x'] - p1['x'])**2 + (p2['y'] - p1['y'])**2)
    b = math.sqrt((p3['x'] - p2['x'])**2 + (p3['y'] - p2['y'])**2)
    c = math.sqrt((p3['x'] - p1['x'])**2 + (p3['y'] - p1['y'])**2)

    angle = math.acos((a**2 + b**2 - c**2) / (2 * a * b))
    return math.degrees(angle)

# Calculate elbow angle
elbow_angle = calculate_angle(
    keypoints['shoulder'],
    keypoints['elbow'],
    keypoints['wrist']
)
```

Step 3: Generate Feedback

```python
def generate_feedback(angles):
    """Generate coaching feedback based on angles"""
    feedback = []

    if 85 <= angles['elbow'] <= 95:
        feedback.append('✓ Perfect elbow angle (90°)')
    else:
        feedback.append(f'⚠ Elbow angle is {angles["elbow"]:.1f}° (target: 90°)')

    if 120 <= angles['knee'] <= 140:
        feedback.append('✓ Good knee bend for power')
    else:
        feedback.append('⚠ Adjust knee bend for better power')

    if 40 <= angles['release'] <= 50:
        feedback.append('✓ Optimal release angle')
    else:
        feedback.append('⚠ Release angle needs adjustment')

    return feedback

feedback = generate_feedback(angles)
```

Step 4: Create Analysis Video

```python
from shotstack_helpers import create_basketball_analysis_video

analysis_results = {
    'feedback': feedback,
    'angles': {
        'elbow': {
            'value': elbow_angle,
            'position': {'x': 0.2, 'y': 0.0}
        },
        # ... more angles
    }
}

output_url = create_basketball_analysis_video(
    video_path=video_url,
    skeleton_data=keypoints,
    analysis_results=analysis_results,
    output_path="analysis.mp4",
    environment='sandbox'
)
```

🎯 Advanced Features

Custom JSON Templates

Create your own templates for specific analysis types:

```python
def create_custom_template(video_url, custom_data):
    """Create a custom analysis template"""

    template = {
        "timeline": {
            "background": "#000000",
            "tracks": [
                # Base video
                {
                    "clips": [{
                        "asset": {"type": "video", "src": video_url},
                        "start": 0,
                        "length": 5
                    }]
                },
                # Custom overlay
                {
                    "clips": [{
                        "asset": {
                            "type": "image",
                            "src": custom_data['overlay_url']
                        },
                        "start": 0,
                        "length": 5,
                        "opacity": 0.6
                    }]
                },
                # Dynamic text
                {
                    "clips": [{
                        "asset": {
                            "type": "text",
                            "text": custom_data['title'],
                            "font": {"size": 48, "color": "#ffffff"}
                        },
                        "start": 0,
                        "length": 2,
                        "position": "top"
                    }]
                }
            ]
        },
        "output": {
            "format": "mp4",
            "fps": 30,
            "size": {"width": 1920, "height": 1080}
        }
    }

    return template
```

Batch Processing

Process multiple videos:

```python
def batch_process_videos(video_urls, analysis_data):
    """Process multiple videos in batch"""

    client = ShotStackClient(environment='sandbox')
    editor = BasketballVideoEditor(client)

    render_ids = []

    for video_url, data in zip(video_urls, analysis_data):
        response = editor.create_shooting_form_analysis(
            video_url=video_url,
            annotations=data['annotations'],
            angles=data['angles'],
            duration=5.0
        )
        render_ids.append(response['response']['id'])

    # Wait for all renders
    results = []
    for render_id in render_ids:
        status = client.wait_for_render(render_id)
        results.append(status['response']['url'])

    return results
```

Progress Tracking

Track render progress:

```python
import time

def track_render_progress(client, render_id):
    """Track and display render progress"""

    while True:
        status = client.get_render_status(render_id)
        state = status['response']['status']
        progress = status['response'].get('progress', 0)

        print(f"Status: {state} - Progress: {progress}%")

        if state in ['done', 'failed']:
            break

        time.sleep(5)

    return status
```

🧪 Testing

Test Script

```python
#!/usr/bin/env python3
"""Test ShotStack integration"""

from shotstack_helpers import ShotStackClient, BasketballVideoEditor

def test_basic_render():
    """Test basic video rendering"""
    client = ShotStackClient(environment='sandbox')
    editor = BasketballVideoEditor(client)

    # Test data
    test_video = "https://shotstack-assets.s3.amazonaws.com/footage/beach-over-
head.mp4"

    response = editor.create_shooting_form_analysis(
        video_url=test_video,
        annotations=[
            {'text': 'Test annotation', 'start': 0, 'length': 2}
        ],
        duration=3.0
    )

    render_id = response['response']['id']
    print(f"✓ Render started: {render_id}")

    status = client.wait_for_render(render_id, timeout=120)

    if status['response']['status'] == 'done':
        print(f"✓ Render complete: {status['response']['url']}")
        return True
    else:
        print(f"✗ Render failed: {status['response'].get('error')}")
        return False

if __name__ == "__main__":
    print("Testing ShotStack integration...")
    success = test_basic_render()
    print(f"\nTest {'PASSED' if success else 'FAILED'}")
```

📊 Performance Tips

1. Optimize Video Length

- Keep analysis videos short (5-10 seconds)
- Trim unnecessary footage before processing
- Use lower FPS for slower motion analysis

2. Cache Results

```python
import hashlib
import json

def get_cache_key(video_url, config):
    """Generate cache key for rendered video"""
    data = f"{video_url}:{json.dumps(config, sort_keys=True)}"
    return hashlib.md5(data.encode()).hexdigest()

# Check cache before rendering
cache_key = get_cache_key(video_url, analysis_config)
if cache_key in video_cache:
    return video_cache[cache_key]
```

3. Use Appropriate Resolution

```python
# For mobile viewing
output_mobile = {
    "format": "mp4",
    "fps": 30,
    "size": {"width": 720, "height": 1280}  # 720p vertical
}

# For desktop viewing
output_desktop = {
    "format": "mp4",
    "fps": 30,
    "size": {"width": 1920, "height": 1080}  # 1080p horizontal
}
```

4. Parallel Processing

```python
from concurrent.futures import ThreadPoolExecutor

def process_videos_parallel(videos, max_workers=5):
    """Process multiple videos in parallel"""

    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        futures = [
            executor.submit(process_single_video, video)
            for video in videos
        ]

        results = [future.result() for future in futures]

    return results
```

🐛 Debugging

Enable Debug Logging

```
import logging

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger('shotstack')

# Add to your code
logger.debug(f"Rendering video: {video_url}")
logger.debug(f"Config: {json.dumps(edit_config, indent=2)}")
```

Common Errors

Error: “Invalid JSON”

```
# Validate JSON before sending
import json

try:
    json.dumps(edit_config)
except json.JSONDecodeError as e:
    print(f"Invalid JSON: {e}")
```

Error: “Asset not found”

```
# Verify URL is accessible
import requests

response = requests.head(video_url)
if response.status_code != 200:
    print(f"Video URL not accessible: {video_url}")
```

Error: “Render timeout”

```
# Increase timeout for longer videos
status = client.wait_for_render(render_id, timeout=600)  # 10 minutes
```

📚 Additional Resources

- **ShotStack Docs:** https://shotstack.io/docs/
- **API Reference:** https://shotstack.io/docs/api/reference/
- **Examples:** https://shotstack.io/docs/api/examples/
- **Support:** https://shotstack.io/support/

✅ Checklist

Before deploying to production:

- [ ] Test in sandbox environment
- [ ] Verify all video URLs are accessible
- [ ] Validate JSON configurations
- [ ] Test error handling
- [ ] Implement caching
- [ ] Set up monitoring
- [ ] Configure production API keys
- [ ] Test with real basketball videos
- [ ] Optimize for performance
- [ ] Document custom templates

Last Updated: December 13, 2025