**Face Liveness Verification SDK Integration Guide**

**Executive Summary**

This document provides comprehensive instructions for integrating the Face Liveness Verification SDK into your web application. The SDK enables secure biometric verification through face liveness checks and property photo capture, with flexible integration options including popup windows and embedded iframes.

**Table of Contents**

1. [SDK Overview](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#1-sdk-overview)
2. [Integration Methods](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#2-integration-methods)
3. [Implementation Guide](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#3-implementation-guide)
4. [Complete Flow Explanation](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#4-complete-flow-explanation)
5. [API Reference](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#5-api-reference)
6. [Result Object Structure](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#6-result-object-structure)
7. [Server Integration](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#7-server-integration)
8. [Troubleshooting](https://chatgpt.com/c/6842c299-9930-8005-a780-4cfe3ef6ddeb#8-troubleshooting)

**1. SDK Overview**

The Face Liveness Verification SDK provides a secure, user-friendly way to verify user identity through biometric face verification and capture property photos. Key features include:

* **Face Liveness Detection**: Ensures the person is physically present and not a photo/video
* **Property Photo Capture**: Allows users to take photos of properties with geolocation
* **Flexible Integration**: Choose between popup windows or embedded iframes
* **Real-time Processing**: Optional WebSocket support for faster verification
* **Vertical Tracking**: Analytics support for different industry verticals

**2. Integration Methods**

**Popup Mode**

* Opens verification in a new browser window
* Minimal impact on your existing UI
* Requires handling window communication

**Iframe Mode**

* Embeds verification directly in your page
* Seamless user experience within your application
* Requires container element in your DOM

**3. Implementation Guide**

**Step 1: Include the SDK**

const SERVER\_URL = 'http://localhost:5103';

const FRONTEND\_URL = 'http://localhost:5173/fastapi-liveness';

const VerificationTools = {

openPopup: function(options) {

if (!options || !options.callbackUrl) {

console.error('VerificationTools: callbackUrl is required');

return null;

}

const origin = window.location.origin || document.location.href;

const url = new URL(`${FRONTEND\_URL}/fastapi-liveness`);

url.searchParams.append('origin', origin);

url.searchParams.append('callbackUrl', options.callbackUrl);

if (options.requestId) url.searchParams.append('requestId', options.requestId);

if (options.mode) url.searchParams.append('mode', options.mode);

if (options.useWebSocket) url.searchParams.append('ws', 'true');

if (options.vertical) url.searchParams.append('vertical', options.vertical);

const popup = window.open(

url.toString(),

'VerificationPopup',

'width=800,height=600,resizable=yes,scrollbars=yes,status=yes'

);

return popup;

},

embedIframe: function(containerId, options) {

const container = document.getElementById(containerId);

if (!container) {

console.error(`VerificationTools: Container element with ID "${containerId}" not found`);

return null;

}

if (!options || !options.callbackUrl) {

console.error('VerificationTools: callbackUrl is required');

return null;

}

const origin = window.location.origin || document.location.href;

const url = new URL(`${FRONTEND\_URL}/standalone-liveness`);

url.searchParams.append('origin', origin);

url.searchParams.append('callbackUrl', options.callbackUrl);

if (options.requestId) url.searchParams.append('requestId', options.requestId);

if (options.mode) url.searchParams.append('mode', options.mode);

if (options.useWebSocket) url.searchParams.append('ws', 'true');

if (options.vertical) url.searchParams.append('vertical', options.vertical);

const iframe = document.createElement('iframe');

iframe.src = url.toString();

iframe.width = '100%';

iframe.height = '600px';

iframe.style.border = 'none';

iframe.allow = 'camera; microphone; geolocation';

container.innerHTML = '';

container.appendChild(iframe);

return iframe;

}

};

**Step 2: Set Up Result Listeners**

window.addEventListener('message', function(event) {

if (event.data && event.data.type === 'LIVENESS\_RESULT') {

console.log('Received verification result:', event.data.data);

handleVerificationResult(event.data.data);

}

}, false);

function handleVerificationResult(results) {

console.log("Verification results:", results);

if (results.liveness\_result && results.liveness\_result.passed) {

alert("Verification successful!");

}

if (results.dms\_id) {

const imageType = results.type === 'property\_photo' ? 'property' : 'liveness';

const imageUrl = `${SERVER\_URL}/get\_${imageType === 'property' ? 'property\_photo' : 'captured\_image'}/${results.dms\_id}`;

console.log("Image URL:", imageUrl);

}

}

function handleUrlCallback() {

const urlParams = new URLSearchParams(window.location.search);

const callbackData = urlParams.get('callbackData');

if (callbackData) {

try {

const decodedData = JSON.parse(decodeURIComponent(callbackData));

console.log("Received callback data in URL:", decodedData);

handleVerificationResult(decodedData);

const cleanUrl = window.location.pathname;

window.history.replaceState({}, document.title, cleanUrl);

} catch (e) {

console.error("Error parsing callback data:", e);

}

}

}

handleUrlCallback();

**Step 3: Launch Face Liveness Check**

**Popup Mode**

const requestId = 'req\_' + Math.random().toString(36).substring(2, 15);

VerificationTools.openPopup({

callbackUrl: window.location.href,

requestId: requestId,

mode: 'liveness',

useWebSocket: true,

vertical: 'insurance'

});

**Iframe Mode**

<div id="livenessContainer" style="width: 100%; height: 600px;"></div>

const requestId = 'req\_' + Math.random().toString(36).substring(2, 15);

VerificationTools.embedIframe('livenessContainer', {

callbackUrl: window.location.href,

requestId: requestId,

mode: 'liveness',

useWebSocket: true,

vertical: 'insurance'

});

**Step 4: Launch Property Photo Capture**

**Popup Mode**

const requestId = 'prop\_' + Math.random().toString(36).substring(2, 15);

VerificationTools.openPopup({

callbackUrl: window.location.href,

requestId: requestId,

mode: 'property',

vertical: 'insurance'

});

**Iframe Mode**

<div id="propertyContainer" style="width: 100%; height: 600px;"></div>

const requestId = 'prop\_' + Math.random().toString(36).substring(2, 15);

VerificationTools.embedIframe('propertyContainer', {

callbackUrl: window.location.href,

requestId: requestId,

mode: 'property',

vertical: 'insurance'

});

**4. Complete Flow Explanation**

**1. Initialization**

* Your application calls openPopup() or embedIframe()
* The SDK constructs a URL with all necessary parameters
* The React verification application loads in a popup or iframe

**2. User Verification Process**

* The app initializes the camera and UI components
* The user completes verification:
  + For liveness: Face detection, movement challenges
  + For property: Framing and capturing the property

**3. Server Processing**

* If WebSocket is enabled:
  + Real-time processing with feedback
  + Frames are streamed to server
* If WebSocket is disabled:
  + Final image is sent via HTTP
  + Processed in one step

**4. Result Communication**

* React app gets results from server
* Sends result to application via:
  + window.postMessage() (preferred)
  + URL redirection with encoded data

**5. Result Handling**

* Application receives results
* Displays outcome
* Retrieves captured image using DMS ID

**5. API Reference**

**VerificationTools.openPopup(options)**

Opens the verification in a popup window.

**Parameters:**

* callbackUrl (required): URL to receive results
* requestId (optional): Unique ID for request
* mode (optional): 'liveness' or 'property'
* useWebSocket (optional): Enable real-time
* vertical (optional): Industry vertical

**Returns:** Popup window object

**VerificationTools.embedIframe(containerId, options)**

Embeds verification in an iframe.

**Parameters:**

* containerId (required): HTML element ID
* options (same as above)

**Returns:** Iframe element

**6. Result Object Structure**

{

"type": "liveness\_check", // or "property\_photo"

"requestId": "req\_abc123",

"timestamp": "2023-06-15T14:30:45Z",

"dms\_id": "12345",

"liveness\_result": {

"passed": true,

"confidence": 0.98,

"details": {}

},

"geoLocation": {

"latitude": 37.7749,

"longitude": -122.4194,

"accuracy": 10.5

},

"vertical": "insurance"

}

**7. Server Integration**

* Liveness Image: ${SERVER\_URL}/get\_captured\_image/${dms\_id}
* Property Photo: ${SERVER\_URL}/get\_property\_photo/${dms\_id}

**8. Troubleshooting**

**Common Issues**

1. **Camera Access Issues**
   * Use HTTPS
   * Check browser permissions
   * Ensure iframe has correct allow attributes
2. **Popup Blocked**
   * Instruct users to allow popups
   * Call openPopup() on user interaction
   * Use iframe if needed
3. **Communication Errors**
   * Check for CORS issues
   * Verify origin and callbackUrl
4. **Verification Failures**
   * Ensure lighting
   * Clear visibility of face
   * Proper server setup
5. **Integration Issues**
   * Confirm URLs
   * Validate parameters
   * Use dev tools to inspect network errors