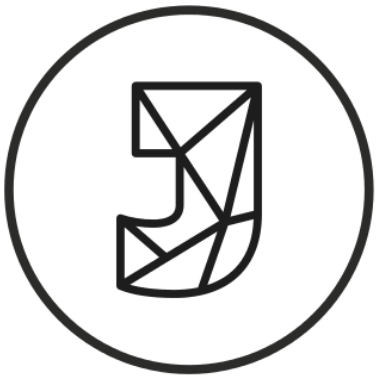
JEELIZ WIDGET



JEELIZ

AI IN YOUR BROWSER

INTRODUCTION

This document describes how to integrate Jeeliz Virtual Tryon solution into a third party website, as a widget.

ARCHITECTURE

This bundle should have the following files and directories:

- *css*: contains 1 CSS stylesheet, used to change the style of the viewer,
- *images*: contains the images referenced by the CSS stylesheet,
- *integration*: contains advanced integration examples, for inspiration,
- *snippets*: code snippets,
- *demo.html*: a basic integration demo, with some buttons to test the features.

INTEGRATION

For an integration example, look at the source code of *demo.html*.

HTTPS SERVER

For both production and test server, you should use HTTPS, otherwise you cannot access to your webcam stream for browser security reasons and you will got some CORS errors in the web console. Even if you host on your local computer, you should have an HTTPS server.

javascript

A single ES5 JavaScript script is included:

```
<script type='text/javascript'
        src='https://appstatic.jeeliz.com/jeewidget/JeelizNNCwidget.js'></script>
```

All Jeeliz Widget will be methods of the global object *JEEWIDGET*:

- *JEEWIDGET.start(<object options>)*: start the virtual tryon module. It can be launched on the page loading (with *body.onload* method like in *demo.html* example, or when the user clicks on a specific link. *options* attributes are:
  - <*string*> *sku*: load a specific glasses model as soon as the widget is loaded,
  - <*string*> *searchImageMask*: URL of an image which will be used to display a rotating loading pattern when the widget is looking for a face. If not provided, it uses the Jeeliz logo,
  - <*number*> *searchImageColor*: hexadecimal color of the rotating loading pattern when the widget is looking for a face,

  - *<function> onError*: function launched if an error happens, look at *demo.html* for error codes,
  - *<function> callbackReady*: function launched when the widget is ready. The first argument is a boolean matching whether the widget is in video mode or fallback image mode.

- *JEEWIDGET.load(<string> SKU, <function> callback)*: load the model which SKU = SKU, then launch a callback function,
- *JEEWIDGET.pause()*: pause the viewer (works only in non image mode). This is a good idea to pause the viewer when it is not visible or not used, because it uses quite a lot of resources,
- *JEEWIDGET.resume()*: resume the pause (works only in non image mode),
- *JEEWIDGET.capture_image(<int> nSteps, <function> callback, <boolean> isNotAllocate)*: capture the image of the current fitting after *nSteps* additionnal detection steps (15 is a recommended value). Launch the callback function with a javascript image instance as argument. If *isNotAllocate* is set to *false* (default), return a new HTML canvas element, otherwise reuse a generic canvas element,
- *JEEWIDGET.set_videoRotation(<int>angle)*: Set the rotation angle of the camera. The value is in degrees and it can only be 0, 90, -90 or 180. Warning: you may have to play with the canvas CSS rotation too and resize it. Indeed, this function controls the camera rotation angle, NOT the display rotation,
- *JEEWIDGET.resize(<int> width, <int> height)* : should be called to resize the canvas.

HTML

After including the javascript script, you have to copy and paste the HTML code of *snippets/snippet.html* where you want the viewer to appear. You can remove HTML comments, add CSS classes, but you should keep the *id* HTML attributes of the elements.

CSS

The base stylesheet of the viewer is *css/style.css*. We strongly encourage you to style the viewer to fit to your own website. Just make sure that you do not change the visibility of the elements (*display* CSS attribute).

SCALABILITY

The main javascript file and 3D assets (glasses 3D meshes and textures) are delivered through a content delivery network, so there should not be any scalability problem with the realtime video version. In this case, the video stream is captured and fully processed client side.

If there are many users who are not compatible with WebGL, or whose devices are too low end, the virtual try-on widget will only show the fallback

version, where the user uploads a picture which is processed server side. In this case, we can have scalability problems. However, our VTO servers rate of use is still very low, and we have available servers we can quickly add in this case.

TROUBLESHOOTINGS

In case of integration problems, you can contact xavier@jeeliz.com . Please detail:

- Your configuration (OS, graphic card, browser, browser version),
- The browser web console log,
- A screenshot of WebGL 1 config test and WebGL 2 config test (the screenshot must include the WebGL extensions list at the bottom of the page),
- A screenshot of chrome://gpu
- Your integration code,
- If the error occurs with the fallback version, please provide the picture you have used,
- All you have done to reproduce the bug.