**I) Development process:**

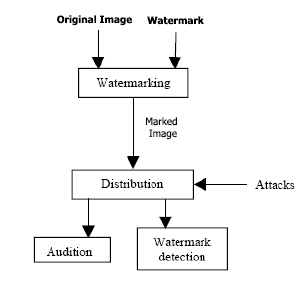
**I.1) Concept:**

A watermarking system is usually divided into three distinct steps, embedding, attack, and detection. In embedding, an algorithm accepts the host and the data to be embedded, and produces a watermarked image.

Then the watermarked digital image is transmitted or stored, usually transmitted to another person. If this person makes a modification, this is called an attack. While the modification may not be malicious, the term attack arises from copyright protection application, where third parties may attempt to remove the digital watermark through modification.

Detection (often called extraction) is an algorithm which is applied to the attacked image to attempt to extract the watermark from it. If the image was unmodified during transmission, then the watermark still is present and it may be extracted. In robust digital watermarking applications, the extraction algorithm should be able to produce the watermark correctly, even if the modifications were strong. In fragile digital watermarking, the extraction algorithm should fail if any change is made to the image.

Types of Watermarks: Visible Watermarks – These watermarks are visible. Invisible Watermarks – These watermarks are embedded in the media and use steganography technique. They are not visible by naked eyes.



Watermarking process

**I.2) Watermark an image:**

We develop our app using JavaScript programming language, and this language offers a great image processing and manipulation APIs, so in our case we’re using the “canvas fill text “API to embed certain text as a watermark on the input image.

**a) CanvasRenderingContext2D.fillText():**

The CanvasRenderingContext2D method fillText(), part of the Canvas 2D API, draws a text string at the specified coordinates, filling the string's characters with the current fillStyle. An optional parameter allows specifying a maximum width for the rendered text, which the user agent will achieve by condensing the text or by using a lower font size.

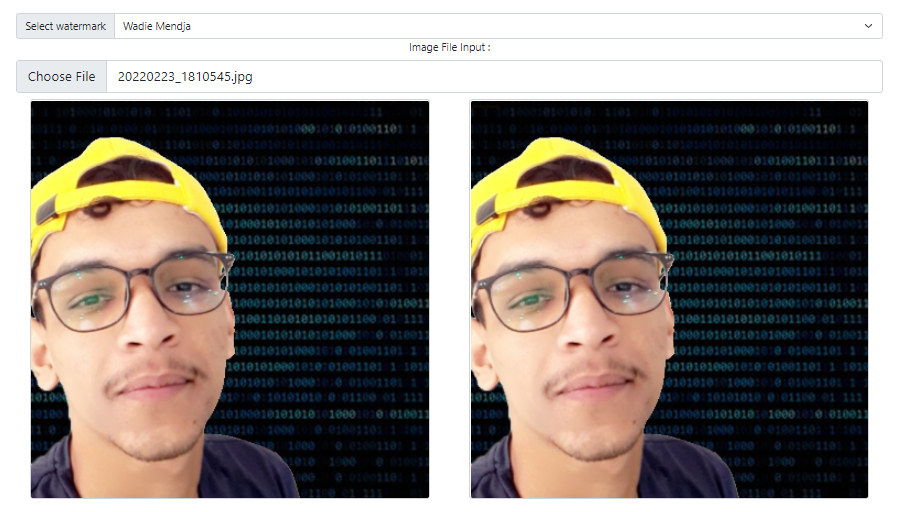
This method draws directly to the canvas without modifying the current path, so any subsequent fill() or stroke() calls will have no effect on it.

The text is rendered using the font and text layout configuration as defined by the font, textAlign, textBaseline, and direction properties.

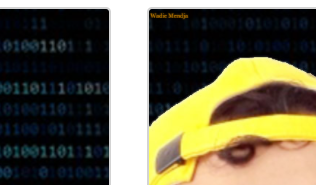
Syntax :

CanvasRenderingContext2D.fillText(text, x, y [, maxWidth]);

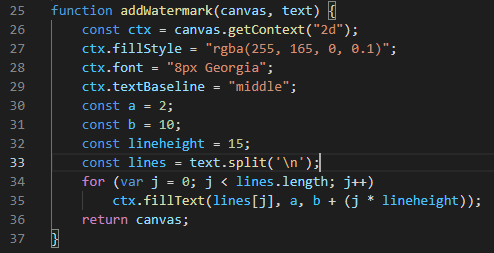
Here is an example of how it works, in the screenshot below we have on the left side the original image and the watermarked image on the right side, the “Select Watermark” text field contains the text that we should embed on the image as a watermark, we did that using the syntax that we talked about earlier



Maybe you didn’t notice any difference between the two images and that’s because we’re embedding an invisible watermark, but if we try to make the watermark visible it’s going to look like that:



And here is the function that does all this work:



In which the “fillStyle” property specified the color and the opacity (alpha component) of the text