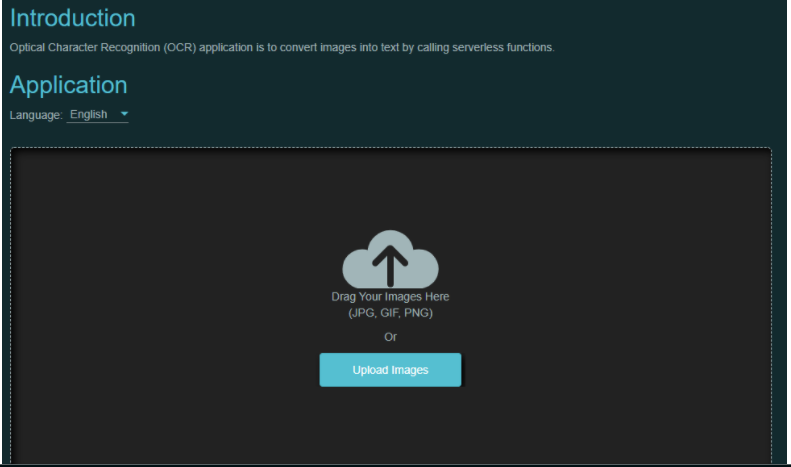
## The OCR UI (frontend)

In this example, the OCR frontend is built with React which we store in the web folder. But, you could use any Framework such as Vue, Angular, jQuery, ASP.NET, Ruby on Rails, etc.

When you first load your application, you have three options to interact with the UI. You can 1) drag and drop an image that's in the JPG, GIF, or PNG format, 2) click the Upload Image button to pull up a folder on your computer, or 3) you can select the language you'd like the application to read out loud in.

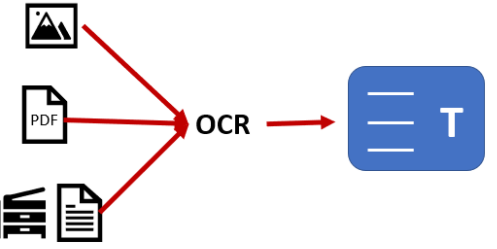


**Techniques**

The technique we will use in the image to text conversion project is the OCR techniques. And the other technique we will use for our project is the asp.net framework for our project.

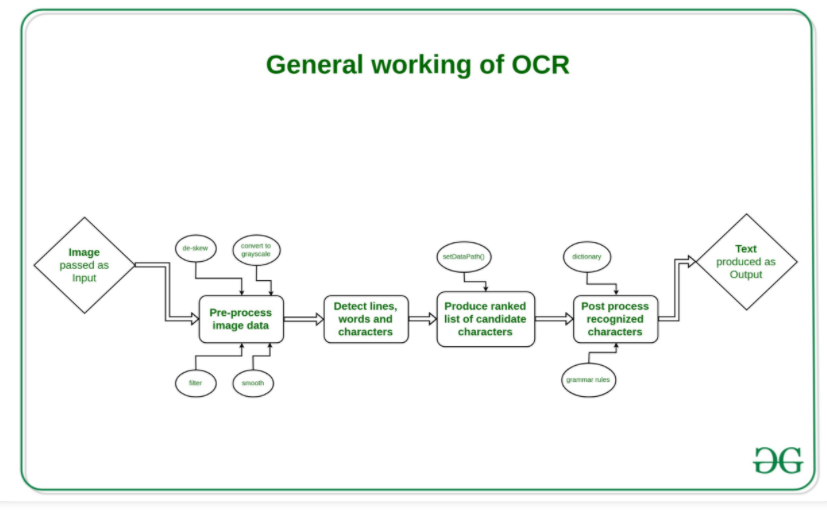
What is OCR

The technique that we have searched for our project is the OCR we will use this OCR techniques in our project.  OCR stands for Optical Character Recognition. It is a widespread technology to recognise text inside images, such as scanned documents and photos. OCR technology is used to convert virtually any kind of images containing written text (typed, handwritten or printed) into machine-readable text data.As you read these words on your computer screen, your eyes and brain are carrying out optical character recognition without you even noticing! Your eyes are recognizing the patterns of light and dark that make up the characters (letters, numbers, and things like punctuation marks) printed on the screen and your brain is using those to figure out what I'm trying to say (sometimes by reading individual characters but mostly by scanning entire words and whole groups of words at once).



### How does the OCR actually work?

An ordinary scanner or photocopy machine creates what is known as a raster image, or a collection of black and white or colored dots. To take and repurpose data from camera images or image-only PDFs, you need OCR software that will take the letters from those images to create words, then sentences, to access and edit the original content on the page. This is done through looking at each line of the image, with the OCR scanner figuring out if the black and white dots represent a certain letter or number.



**Algorithms**

using IronOcr;

using System.Drawing; //for image export

// We can delve deep into OCR results as an object model of

// Pages, Barcodes, Paragraphs, Lines, Words and Characters

// This allows us to explore, export and draw OCR content using other APIs/

var Ocr = new IronTesseract();

Ocr.Configuration.EngineMode = TesseractEngineMode.TesseractAndLstm;

Ocr.Configuration.ReadBarCodes = true;

using (var Input = new OcrInput(@"img\Potter.tiff"))

{

OcrResult Result = Ocr.Read(Input);

foreach (var Page in Result.Pages)

{

// Page object

int PageNumber = Page.PageNumber;

string PageText = Page.Text;

int PageWordCount = Page.WordCount;

// null if we don't set Ocr.Configuration.ReadBarCodes = true;

OcrResult.Barcode[] Barcodes = Page.Barcodes;

System.Drawing.Bitmap PageImage = Page.ToBitmap(Input);

int PageWidth = Page.Width;

int PageHeight = Page.Height;

foreach (var Paragraph in Page.Paragraphs)

{

// Pages -> Paragraphs

int ParagraphNumber = Paragraph.ParagraphNumber;

String ParagraphText = Paragraph.Text;

System.Drawing.Bitmap ParagraphImage = Paragraph.ToBitmap(Input);

int ParagraphX\_location = Paragraph.X;

int ParagraphY\_location = Paragraph.Y;

int ParagraphWidth = Paragraph.Width;

int ParagraphHeight = Paragraph.Height;

double ParagraphOcrAccuracy = Paragraph.Confidence;

OcrResult.TextFlow paragrapthText\_direction = Paragraph.TextDirection;

foreach (var Line in Paragraph.Lines)

{

// Pages -> Paragraphs -> Lines

int LineNumber = Line.LineNumber;

String LineText = Line.Text;

System.Drawing.Bitmap LineImage = Line.ToBitmap(Input); ;

int LineX\_location = Line.X;

int LineY\_location = Line.Y;

int LineWidth = Line.Width;

int LineHeight = Line.Height;

double LineOcrAccuracy = Line.Confidence;

double LineSkew = Line.BaselineAngle;

double LineOffset = Line.BaselineOffset;

foreach (var Word in Line.Words)

{

// Pages -> Paragraphs -> Lines -> Words

int WordNumber = Word.WordNumber;

String WordText = Word.Text;

System.Drawing.Image WordImage = Word.ToBitmap(Input);

int WordX\_location = Word.X;

int WordY\_location = Word.Y;

int WordWidth = Word.Width;

int WordHeight = Word.Height;

double WordOcrAccuracy = Word.Confidence;

if (Word.Font != null)

{

// Word.Font is only set when using Tesseract Engine Modes rather than LTSM

String FontName = Word.Font.FontName;

double FontSize = Word.Font.FontSize;

bool IsBold = Word.Font.IsBold;

bool IsFixedWidth = Word.Font.IsFixedWidth;

bool IsItalic = Word.Font.IsItalic;

bool IsSerif = Word.Font.IsSerif;

bool IsUnderLined = Word.Font.IsUnderlined;

bool IsFancy = Word.Font.IsCaligraphic;

}

foreach (var Character in Word.Characters)

{

// Pages -> Paragraphs -> Lines -> Words -> Characters

int CharacterNumber = Character.CharacterNumber;

String CharacterText = Character.Text;

System.Drawing.Bitmap CharacterImage = Character.ToBitmap(Input);

int CharacterX\_location = Character.X;

int CharacterY\_location = Character.Y;

int CharacterWidth = Character.Width;

int CharacterHeight = Character.Height;

double CharacterOcrAccuracy = Character.Confidence;

// Output alternative symbols choices and their probability.

// Very useful for spell checking

OcrResult.Choice[] Choices = Character.Choices;

}

}

}

}

}

}