Guía rápida:
Elaboración de templates.

Seleccionar imagen y borrar información

(a) (b)

Figura 1: Imágenes con sombras.

Áreas de interés

Mostrar en pantalla el template con un ajuste de tamaño a 700 pixeles de ancho.

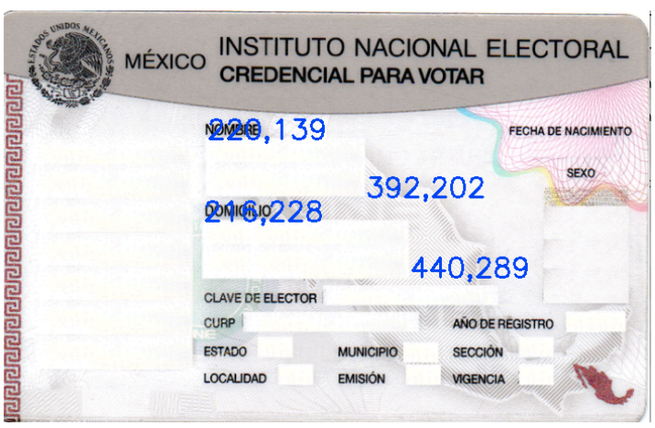

Figura 2: Valores en esquinas.

Validar el área cubierta por los dos puntos ingresados.

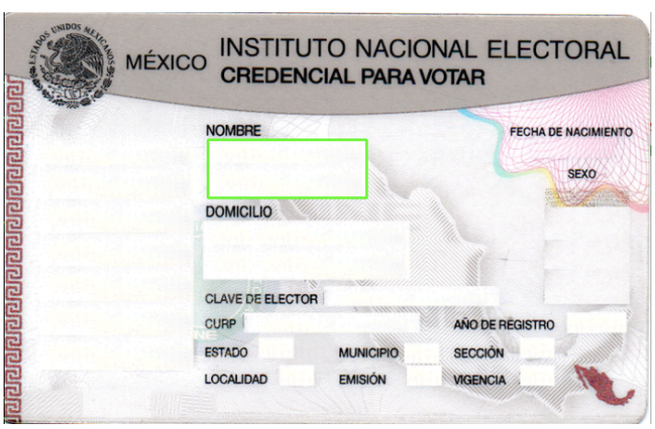

Figura 3: Validación de áreas.

template OCRLocations

- **IFE/INE**
  (`ReaderAPI/Functions/OCRLocations`)

  En el archivo *OCRLocations.py* se encuentran las ubicaciones de lectura del frente de los 4 tipos de formatos de IFE/INE. El tuple OCRLocation tiene tres componentes: id, bbox y filter_keywords. ID ayuda a reconocer el área que deseamos leer, bbox son los valores para el área a leer, los cuales son las coordenadas del punto superior izq, el ancho y el alto del recuadro y filter_keywords es para eliminar los renglones que contengan estas palabras.

```python
# Locations IFE C Type
OCRLocation = namedtuple("OCRLocation", ["id", "bbox","filter_keywords"])
OCR_LOCATIONS_IFEC = [
    OCRLocation("Nombre", (35, 125, 180, 85),["NOMBRE"]),
    OCRLocation("Domicilio", (35, 207, 220, 88),["DOMICILIO"]),
    OCRLocation("Clave de elector", (35, 313, 330, 25),["clave"]),
    OCRLocation("CURP", (35, 333, 270, 25),["CURP"]),
    OCRLocation("Año de registro", (220, 295, 230, 25),["año"]),
    OCRLocation("Edad", (375, 147, 80, 25),["EDAD"]),
    # NEW
    # OCRLocation("newID", (upperX, upperY, width, height), ["wordsToFilter"])
]
```

Figura 4: Ubicaciones de texto IFE/INE (OCRLocations.py)

- **CFE**
  (`ReaderAPI/Functions/cfeFunctions`)

  En el caso de los recibos de CFE y Telmex es algo distinto el procedimiento. Originalmente solo se contemplaba un área grande con el nombre y la dirección. A diferencia de las IFE/INE, estos recibos tienen varios templates con los que se trata de extraer la información, por lo cual, primero se recolectan las lecturas y al final se consolida la información.

  Dado que existen varios templates, hay información que se puede extraer de algunos más no de todos. Se recomienda que los IDs para el mismo tipo de lectura mantengan el mismo nombre. En la Figura 5 se puede ver que un template puede tener más LOCs que los restantes.

```python
def loadDefaults():
    templates = []
    templates.append(cv2.imread('Templates/templateCFE1.png'))
    templates.append(cv2.imread('Templates/templateCFE2.png'))
    templates.append(cv2.imread('Templates/templateCFE3.png'))
    templates.append(cv2.imread('Templates/templateCFE4.png'))
    templates.append(cv2.imread('Templates/templateCFE5.png'))
    templates.append(cv2.imread('Templates/templateCFE6.png'))
    templates.append(cv2.imread('Templates/templateCFE7.png'))
    templates.append(cv2.imread('Templates/templateCFE8.png'))

    OCR_Template1 = [
        OCRLocation("Name", (5, 120, 305, 130), ["Comision", "Federal", "De", "Electricidad"]),
        OCRLocation("Address", (100, 120, 305, 130), ["Comision", "Federal", "De", "Electricidad"])
        # OCRLocation("newID", (upperX, upperY, width, height), ["wordsToFilter"])
    ]

    OCR_Template2 = [
        OCRLocation("Name", (25, 92, 239, 78), ["Comision", "Federal", "De", "Electricidad"])
    ]

    OCR_Template3 = [
        OCRLocation("Name", (10, 110, 313, 97), ["Comision", "Federal", "De", "Electricidad"])
    ]
```

Figura 5: Ubicaciones de texto CFE (cfeFunctions.py).

También es necesario ajustar la lectura por cada tipo de ID, esto permite seleccionar un delimitador de caracteres de acuerdo al valor esperado. Por ejemplo, en el caso de conocer la estructura del número de contrato, se puede limitar solo a números y/o letras y descartar signos de puntuación.

```
            ### By every template
            if loc.id == "Name":
                finalText = ""
                for line in text.split("\n"):
                    count = sum([line.count(x) for x in loc.filter_keywords])
                    if count == 0:
                        line = dl.cleanup_AddressValues(line) ### Change by type of data
                        if len(line) > 2:
                            finalText = finalText + line + "\n"
                print(finalText)
                resultsName.append(finalText)
            elif loc.id == "Address":
                finalText = ""
                for line in text.split("\n"):
                    count = sum([line.count(x) for x in loc.filter_keywords])
                    if count == 0:
                        line = dl.cleanup_AddressValues(line) ### Change by type of data
                        if len(line) > 2:
                            finalText = finalText + line + "\n"
                print(finalText)
                resultsAddress.append(finalText)

    OCR_results.append(resultsName)
    OCR_results.append(resultsAddress)
```

Figura 6: Lectura por cada tipo de ID (cfeFunctions.py).

Además, es importante que en el archivo reading.py se ajuste el JSON de respuesta. En el caso del recibo CFE, se crea una lista por cada tipo de ID y después se consolida la información por cada tipo de ID. una vez que se tienen estas cadenas, se arma el JSON a enviar con los Keys deseados. La figura 8 muestra la salida de este ejemplo.

```
def cfeReading(img):
    templates, OCR_Locations = cfe.loadDefaults()
    alignedImages = cfe.alignToTemplates(img, templates)
    OCR_results = cfe.readLocations(alignedImages, OCR_Locations)
    bestStrings = []
    for result in OCR_results:
        bestStrings.append(cfe.finalResult(result))
    data = {'name':bestStrings[0], 'Sample01':bestStrings[1]} #According to templates

    return data
```

Figura 7: Preparación de respuesta CFE (reading.py).

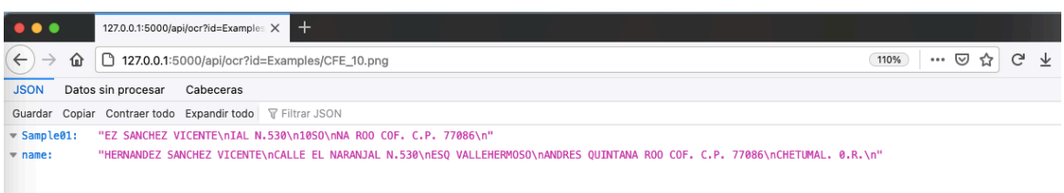

Figura 8: Visualización de respuesta.

- **Telmex**
  (`ReaderAPI/Functions/telmexFunctions`)

Como se mencionaba, el recibo telmex también utiliza varios templates para realizar la lectura de información. Un template puede tener más LOCs que los restantes.

```python
def loadDefaults():
    templates = []
    templates.append(cv2.imread('Templates/templateTelmex1.png'))
    templates.append(cv2.imread('Templates/templateTelmex2.png'))
    templates.append(cv2.imread('Templates/templateTelmex3.png'))
    templates.append(cv2.imread('Templates/templateTelmex4.png'))
    templates.append(cv2.imread('Templates/templateTelmex5.png'))
    templates.append(cv2.imread('Templates/templateTelmex6.png'))

    OCR_Template1 = [
        OCRLocation("Name", (2, 141, 360, 144), ["TELMEX", "TELEFONOS", "Parque", "Via", "06500", "RFC",
        OCRLocation("Sample", (100, 200, 360, 144), ["TELMEX", "TELEFONOS", "Parque", "Via", "06500", "R
        # OCRLocation("newID", (upperX, upperY, width, height), ["wordsToFilter"])
    ]

    OCR_Template2 = [
        OCRLocation("Name", (3, 298, 689, 311), ["TELMEX", "TELEFONOS", "Parque", "Via", "06500", "RFC",
    ]

    OCR_Template3 = [
        OCRLocation("Name", (8, 321, 682, 335), ["TELMEX", "TELEFONOS", "Parque", "Via", "06500", "RFC",
    ]
```

Figura 9: Ubicaciones de texto Telmex (telmexFunctions.py).

```python
            ### By every template
            if loc.id == "Name":
                finalText = ""
                for line in text.split("\n"):
                    count = sum([line.count(x) for x in loc.filter_keywords])
                    if count == 0:
                        line = dl.cleanup_AddressValues(line) ### Change by type of data
                        if len(line) > 2:
                            finalText = finalText + line + "\n"
                print(finalText)
                resultsName.append(finalText)
            elif loc.id == "Sample":
                finalText = ""
                for line in text.split("\n"):
                    count = sum([line.count(x) for x in loc.filter_keywords])
                    if count == 0:
                        line = dl.cleanup_AddressValues(line) ### Change by type of data
                        if len(line) > 2:
                            finalText = finalText + line + "\n"
                print(finalText)
                resultsSample.append(finalText)

    OCR_results.append(resultsName)
    OCR_results.append(resultsSample)
```

Figura 10: Lectura por cada tipo de ID (telmexFunctions.py).

En el caso de este documento, la consolidación de información es un poco distinta, ya que al momento, la lectura de templates se suma a la salida de lectura con referencia al código de barras. Finalmente, también se debe considerar cómo armar el JSON de respuesta una vez que se ha agregado la nueva zona de lectura.

```python
def telmexReading(img):
    orientation = telmex.detectOrientation(img)
    if orientation == False:
        img = cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)

    templates, OCR_Locations = telmex.loadDefaults()
    alignedImages = telmex.alignToTemplates(img, templates)
    OCR_results = telmex.readLocations(alignedImages, OCR_Locations)
    reads = telmex.readFromBarCode(alignedImages, OCR_Locations[0])
    data = telmex.finalResult(OCR_results, reads)

    bestStrings = []
    for i in range(len(OCR_results)):
        if i == 0:
            bestStrings.append(telmex.finalResult(OCR_results[i], reads))
        else:
            bestStrings.append(telmex.finalResult2(OCR_results[i]))
    data = {'name':bestStrings[0], 'SampleTelmex':bestStrings[1]} #According to templates
    print(data)

    return data
```

Figura 11: Preparación de respuesta Telmex (reading.py).

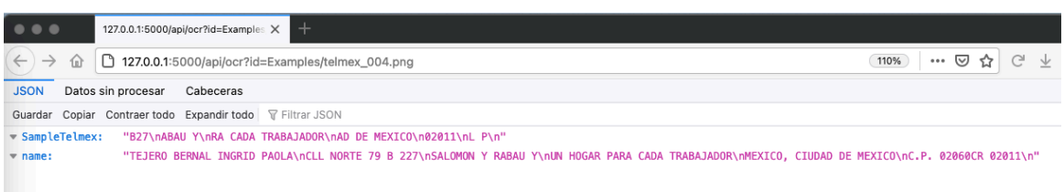

Figura 12: Visualización de respuesta.