Installation : <https://reactnative.dev/docs/environment-setup>

Youtube: <https://www.youtube.com/watch?v=mEUSNId1Hfc>

**Steps front react native**

1. **npm install -g expo-cli**

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

1. expo init name\_project

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* Choose blank: a minimal app as clean as an empty canvas

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

* Finished
* Now, we have a projet AppCrossword :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* cd AppCrossword/
* npm start

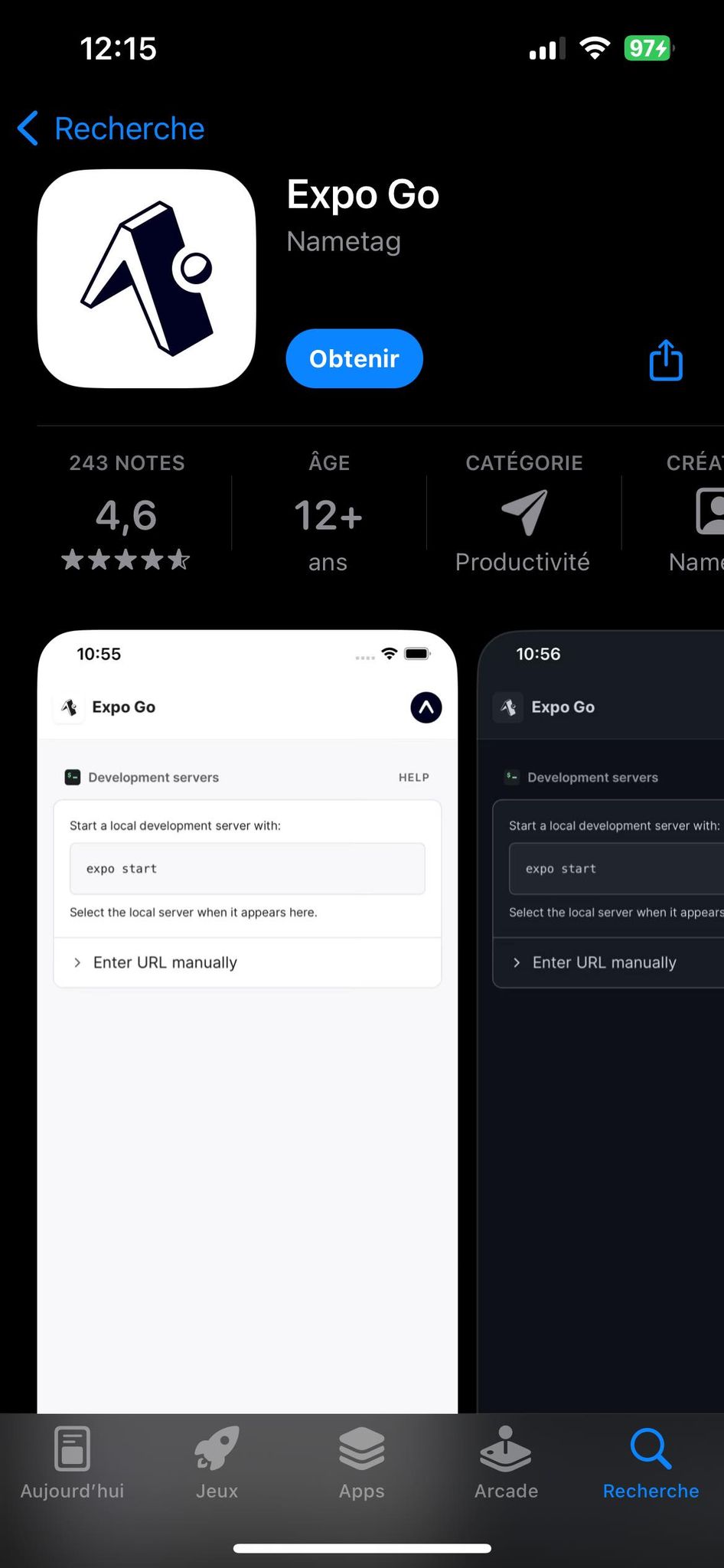
Une image contenant texte, capture d’écran, mots croisés, motif

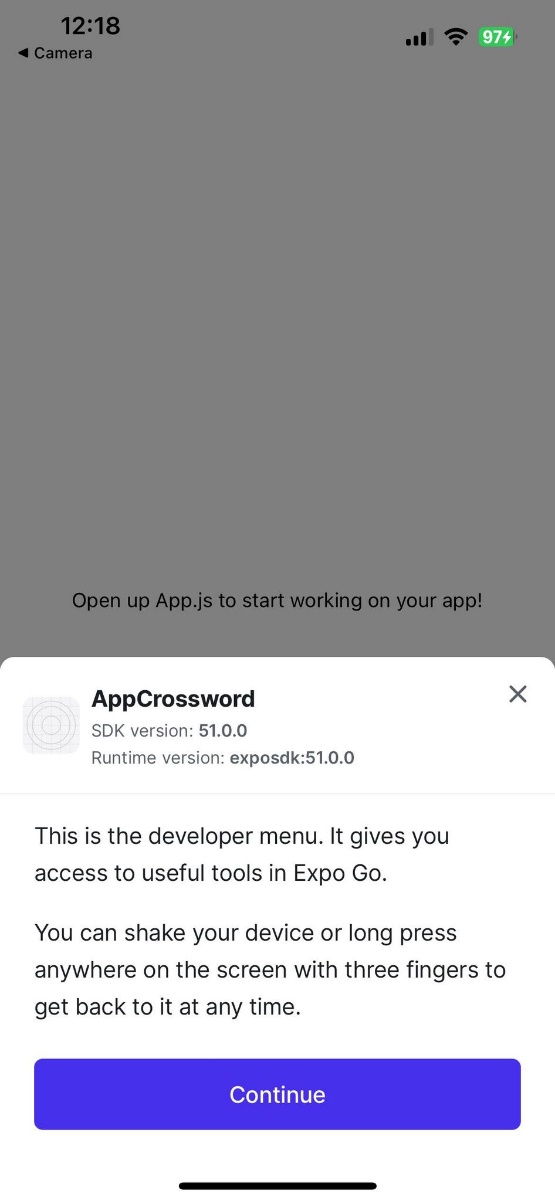
Description générée automatiquement

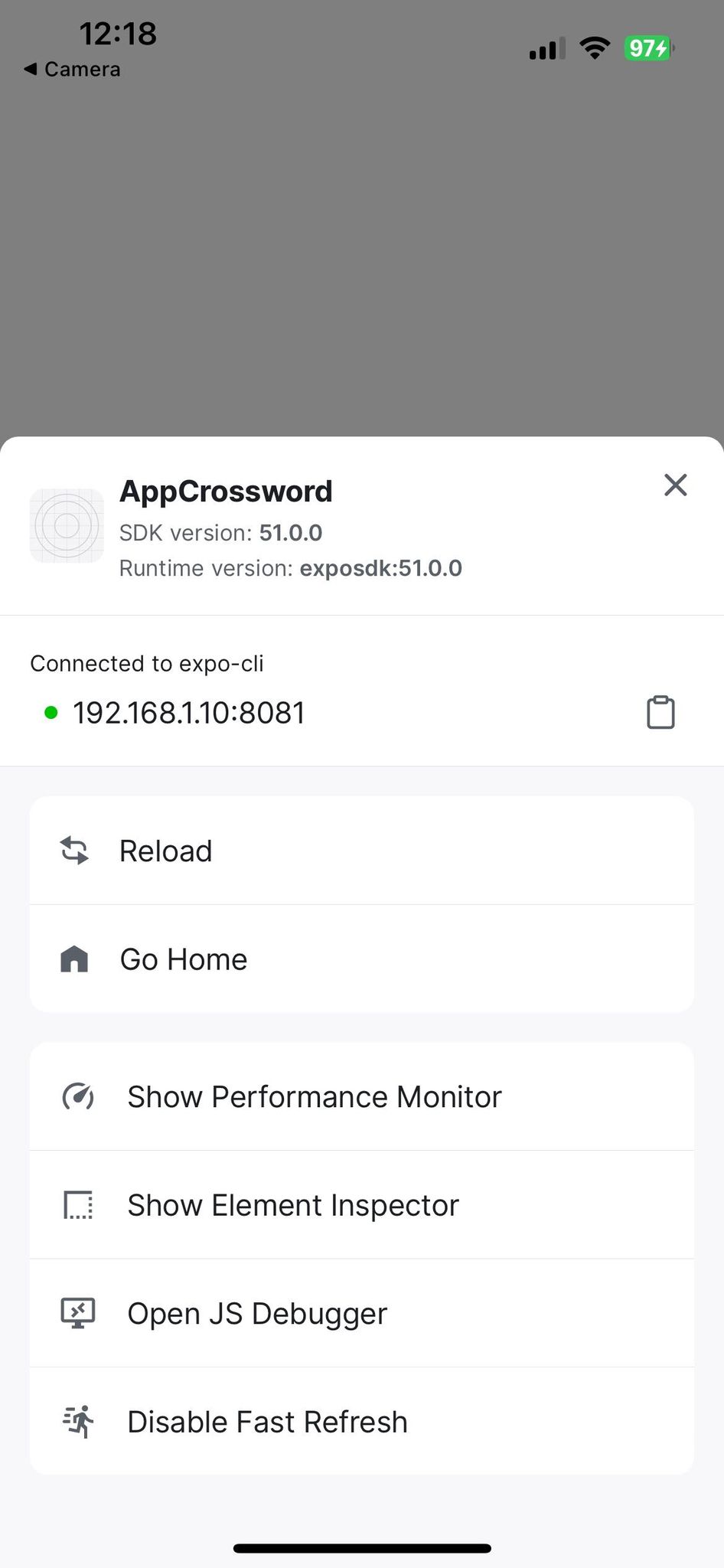
Une image contenant texte, capture d’écran, Police

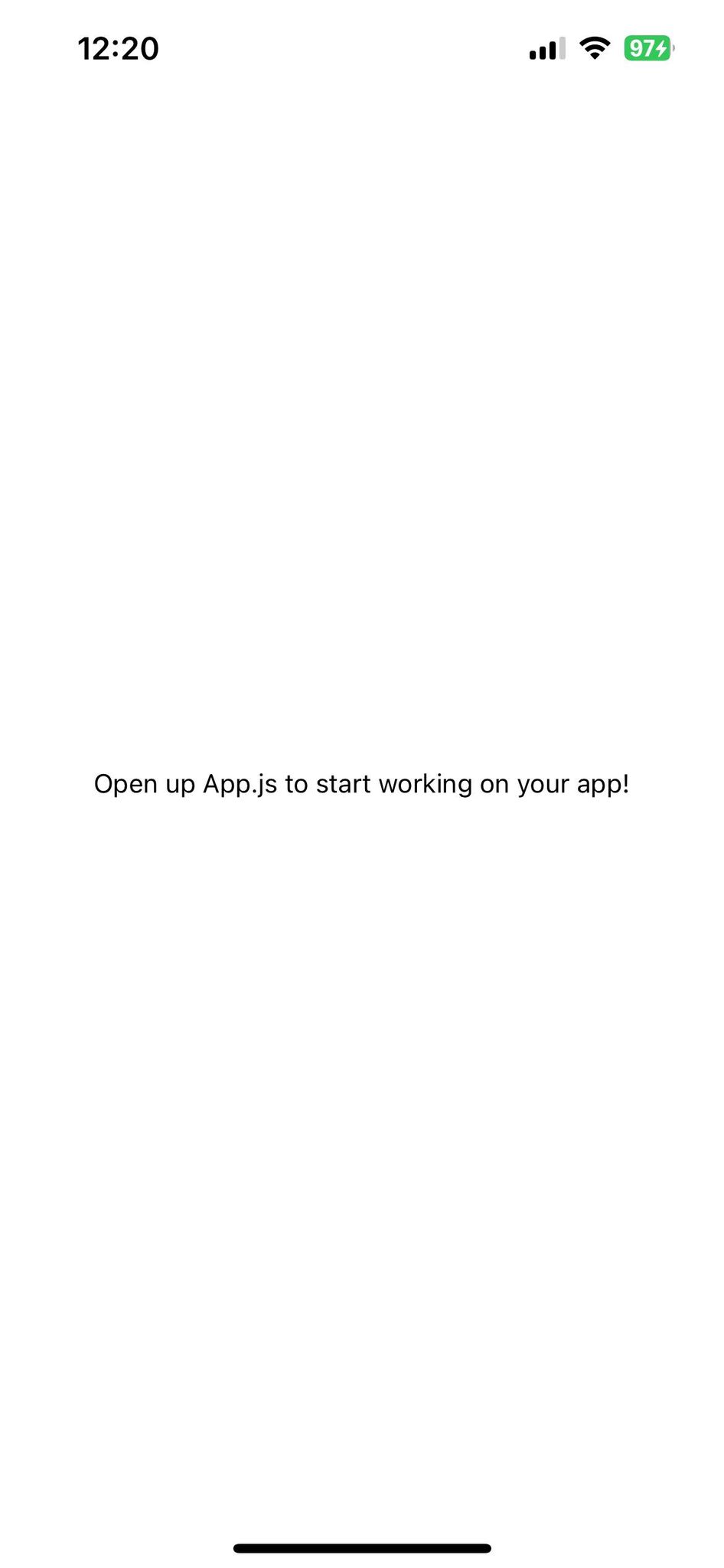
Description générée automatiquement

* install application “Expo go” in appStore, CHplay
* open camera to scan qr code by phone



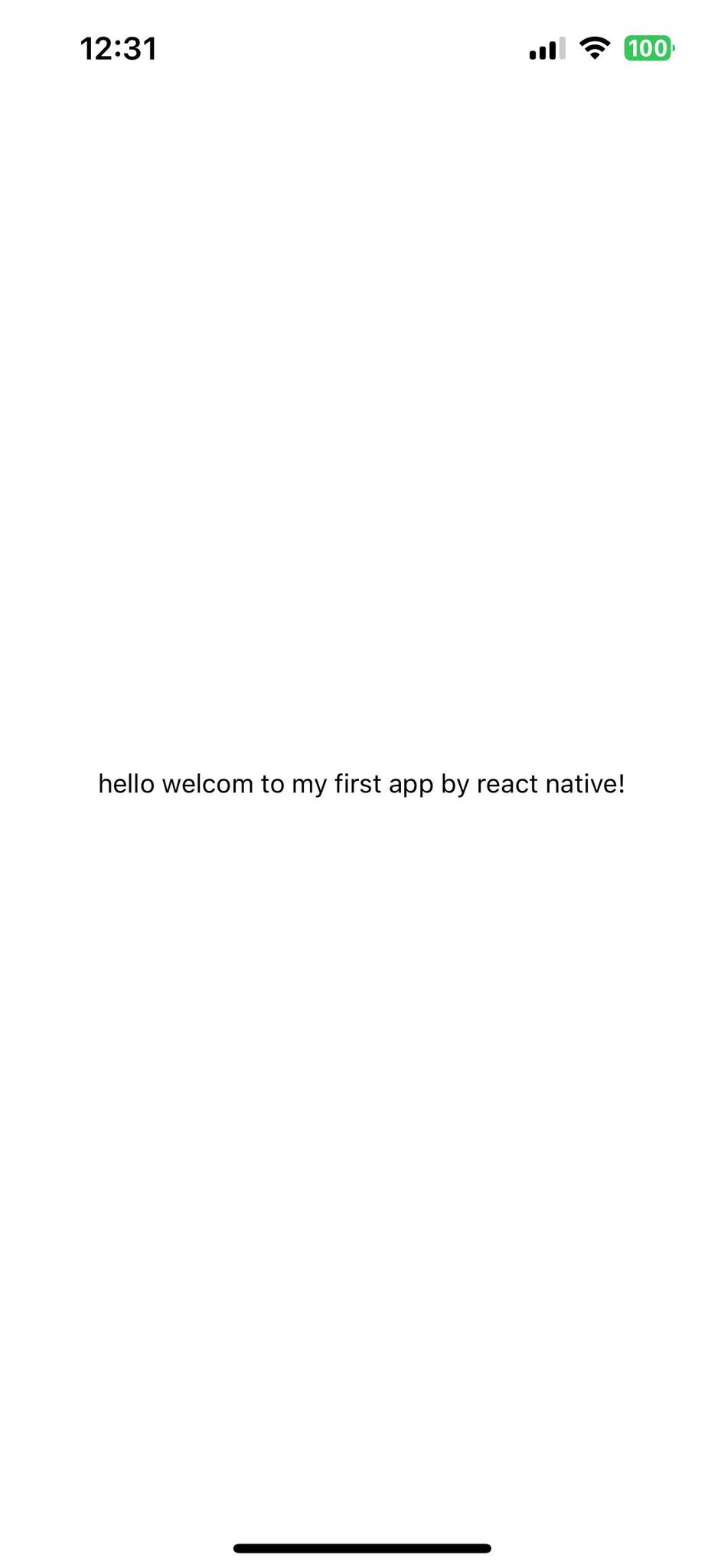






* and now we will just change the text in app in App.js

      <Text>hello welcom to my first app by react native!</Text>



1. start to make camera in app

* install libraries :

**npx expo install expo-camera expo-image-picker**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ctrl +c

**npx expo start -c** (Xóa bộ nhớ cache và khởi động lại dự án)

npx expo start

npx expo install expo-camera

**Back end python flask**

1. **install : pip install flask flask-cors**
2. **cd backend**=> run to flask server: python app.py
3. **npm install axios**
4. **il faut avoir même wifi pour pc et iphone**

**export FLASK\_APP=app.py**

**export FLASK\_RUN\_HOST=0.0.0.0**

**flask run**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Lưu lại process 1

import React, { useState, useEffect } from 'react';

import { StyleSheet, Text, View, Button, Image, ActivityIndicator } from 'react-native';

import \* as ImagePicker from 'expo-image-picker';

import axios from 'axios';

export default function App() {

  const [gridImage, setGridImage] = useState(null);

  const [definitionImage, setDefinitionImage] = useState(null);

  const [matrix, setMatrix] = useState(null);

  const [loading, setLoading] = useState(false);

  const [page, setPage] = useState('welcome');

  const pickImage = async (setter) => {

    let result = await ImagePicker.launchImageLibraryAsync({

      mediaTypes: ImagePicker.MediaTypeOptions.Images,

      allowsEditing: true,

      aspect: [4, 3],

      quality: 1,

    });

    if (!result.canceled) {

      const uri = result.assets[0].uri;

      setter(uri);

      uploadImage(uri, setter === setGridImage ? 'grid.jpg' : 'definition.jpg');

    }

  };

  const uploadImage = async (uri, filename) => {

    const formData = new FormData();

    formData.append('file', {

      uri,

      name: filename,

      type: 'image/jpeg',

    });

    try {

      const response = await axios.post('http://192.168.1.10:5000/upload', formData, {

        headers: {

          'Content-Type': 'multipart/form-data',

        },

      });

      console.log('Upload success:', response.data);

    } catch (error) {

      console.error('Upload error:', error);

    }

  };

  const processGrid = async () => {

    setLoading(true);

    try {

      const response = await axios.post('http://192.168.1.10:5000/process-grid');

      console.log('Processing success:', response.data);

      setMatrix(response.data.matrix\_binaire);

      setPage('showGrid');

    } catch (error) {

      console.error('Processing error:', error);

    }

    setLoading(false);

  };

  const AnimalAnimation = () => {

    const [position, setPosition] = useState(0);

    useEffect(() => {

      const interval = setInterval(() => {

        setPosition((prev) => (prev < 100 ? prev + 1 : 0));

      }, 50);

      return () => clearInterval(interval);

    }, []);

    return (

      <View style={[styles.animal, { left: `${position}%` }]}>

        <Text>🐾</Text>

      </View>

    );

  };

  if (page === 'welcome') {

    return (

      <View style={styles.container}>

        <Text>Welcome to the application to solve crossword puzzles</Text>

        <Button title="Start" onPress={() => setPage('addImages')} />

      </View>

    );

  }

  if (page === 'addImages') {

    return (

      <View style={styles.container}>

        <Text>Upload your images</Text>

        <View style={styles.buttonContainer}>

          <Button title="Pick Grid Image" onPress={() => pickImage(setGridImage)} />

          {gridImage && <Image source={{ uri: gridImage }} style={styles.image} />}

        </View>

        <View style={styles.buttonContainer}>

          <Button title="Pick Definition Image" onPress={() => pickImage(setDefinitionImage)} />

          {definitionImage && <Image source={{ uri: definitionImage }} style={styles.image} />}

        </View>

        {gridImage && definitionImage ? (

          <Button title="Confirm Images" onPress={() => processGrid()} />

        ) : (

          <Button title="Back to Welcome" onPress={() => setPage('welcome')} />

        )}

        {loading && <ActivityIndicator size="large" color="#00ff00" />}

      </View>

    );

  }

  if (page === 'showGrid') {

    return (

      <View style={styles.container}>

        <Text>Here is your crossword grid:</Text>

        <View style={styles.matrixContainer}>

          {matrix && matrix.map((row, rowIndex) => (

            <View key={rowIndex} style={styles.matrixRow}>

              {row.map((cell, cellIndex) => (

                <View

                  key={cellIndex}

                  style={[

                    styles.matrixCell,

                    { backgroundColor: cell === 1 ? 'white' : 'orange' },

                  ]}

                />

              ))}

            </View>

          ))}

        </View>

        <Button title="Next Step" onPress={() => setPage('animalAnimation')} />

        <Button title="Back to Home" onPress={() => setPage('welcome')} />

      </View>

    );

  }

  if (page === 'animalAnimation') {

    return (

      <View style={styles.container}>

        <Text>Solving...</Text>

        <AnimalAnimation />

        <Button title="Back to Home" onPress={() => setPage('welcome')} />

      </View>

    );

  }

  return null;

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

  buttonContainer: {

    margin: 10,

  },

  image: {

    width: 200,

    height: 200,

    margin: 10,

  },

  matrixContainer: {

    marginTop: 20,

  },

  matrixRow: {

    flexDirection: 'row',

  },

  matrixCell: {

    width: 20,

    height: 20,

    borderWidth: 1,

    borderColor: 'black',

  },

  animal: {

    position: 'absolute',

    top: '50%',

    marginTop: -10,

  },

});

from flask import Flask, request, jsonify

import os

import process\_grid\_image

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = 'uploads'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

@app.route('/upload', methods=['POST'])

def upload\_file():

    if 'file' not in request.files:

        return jsonify({"error": "No file part"}), 400

    file = request.files['file']

    if file.filename == '':

        return jsonify({"error": "No selected file"}), 400

    if file:

        filename = file.filename

        file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        file.save(file\_path)

        return jsonify({"message": "File uploaded successfully", "filename": filename}), 200

@app.route('/process-grid', methods=['POST'])

def process\_grid():

    grid\_image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], 'grid.jpg')

    try:

        cropped\_crossword = process\_grid\_image.parse\_grid(grid\_image\_path)

        binary\_matrix = process\_grid\_image.convert\_to\_binary\_matrix(cropped\_crossword, crossword\_size=10)

        return jsonify({"matrix\_binaire": binary\_matrix.tolist()}), 200

    except Exception as e:

        return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5000)

Chỉ cofnbij mơi hori

import cv2

import numpy as np

import requests

import io

import json

import re

import os

def filter\_and\_store\_lines(ocr\_text):

    lines = ocr\_text.split('\n')

    valid\_lines = []

    for line in lines:

        line = line.strip()

        if not line:

            continue

        # Lọc các dòng chỉ có key hoặc chữ cái với các ký hiệu

        if re.match(r'^\d+[-.]?$|^[A-Za-z][-.,]?$', line):

            continue

        valid\_lines.append(line)

    return valid\_lines

def remove\_keys\_from\_lines(valid\_lines):

    filtered\_lines = []

    for line in valid\_lines:

        # Loại bỏ key có sẵn nếu có

        line\_without\_key = re.sub(r'^\d+[-.]?\s\*', '', line)

        filtered\_lines.append(line\_without\_key)

        print(f"Filtered Line without Key: {line\_without\_key}")

    return filtered\_lines

def process\_filtered\_lines(filtered\_lines):

    horiz\_definitions = {}

    vert\_definitions = {}

    current\_section = None

    current\_lines = []

    vert\_key\_index = 0

    vert\_keys = 'ABCDEFGHIJ'

    horiz\_key\_index = 1

    for line in filtered\_lines:

        if "HORIZONTALEMENT" in line:

            current\_section = "horizontal"

            current\_lines = []

            continue

        elif "VERTICALEMENT" in line:

            current\_section = "vertical"

            current\_lines = []

            continue

        if current\_section == "horizontal":

            horiz\_definitions[str(horiz\_key\_index)] = line.strip()

            horiz\_key\_index += 1

        elif current\_section == "vertical":

            if vert\_key\_index < len(vert\_keys):

                vert\_definitions[vert\_keys[vert\_key\_index]] = line.strip()

                vert\_key\_index += 1

    return {'HORIZONTALEMENT': horiz\_definitions, 'VERTICALEMENT': vert\_definitions}

def process\_definition(image\_path, api\_key):

    img = cv2.imread(image\_path)

    height, width, \_ = img.shape

    roi = img

    roi\_resized = cv2.resize(roi, (width // 2, height // 2))

    \_, compressedimage = cv2.imencode(".jpg", roi\_resized, [cv2.IMWRITE\_JPEG\_QUALITY, 80])

    file\_bytes = io.BytesIO(compressedimage)

    if file\_bytes.getbuffer().nbytes <= 1024 \* 1024:

        url\_api = "https://api.ocr.space/parse/image"

        files = {"file": ("definition.jpg", file\_bytes, "image/jpeg")}

        result = requests.post(url\_api,

                        files=files,

                        data={"apikey": api\_key,

                              "language": "fre"})

        result = result.content.decode()

        result = json.loads(result)

        if 'ParsedResults' in result:

            parsed\_results = result['ParsedResults'][0]

            ocr\_text = parsed\_results['ParsedText']

            print("OCR Text:", ocr\_text)

            valid\_lines = filter\_and\_store\_lines(ocr\_text)

            for line in valid\_lines:

                print(f"Filtered Line: {line}")

            filtered\_lines = remove\_keys\_from\_lines(valid\_lines)

            json\_result = process\_filtered\_lines(filtered\_lines)

            print(json\_result)

            with open('json/definition.json', 'w', encoding='utf-8') as f:

                json.dump(json\_result, f, ensure\_ascii=False, indent=4)

            print(f"JSON result saved to json/definition.json")

        else:

            print("No results returned or an error occurred.")

            print("Full response from OCR API:", result)

    else:

        raise ValueError("Image size exceeds limit of 1MB.")

if \_\_name\_\_ == "\_\_main\_\_":

    if not os.path.exists('json'):

        os.makedirs('json')

    image\_path = "uploads/definition.jpg"

    api\_key = "K89603096888957"

    process\_definition(image\_path, api\_key)

lưu lại process 2

python image definition

import cv2

import numpy as np

import requests

import io

import json

import re

import os

def filter\_and\_store\_lines(ocr\_text):

    lines = ocr\_text.split('\n')

    valid\_lines = []

    for line in lines:

        line = line.strip()

        if not line:

            continue

        # Lọc các dòng chỉ có key hoặc chữ cái với các ký hiệu

        if re.match(r'^\d+[-.]?$|^[A-Za-z][-.,]?$', line):

            continue

        valid\_lines.append(line)

    return valid\_lines

def remove\_keys\_from\_lines(valid\_lines):

    filtered\_lines = []

    for line in valid\_lines:

        # Loại bỏ key có sẵn nếu có

        line\_without\_key = re.sub(r'^\d+[-.]?\s\*', '', line)

        filtered\_lines.append(line\_without\_key)

        print(f"Filtered Line without Key: {line\_without\_key}")

    return filtered\_lines

def process\_filtered\_lines(filtered\_lines):

    horiz\_definitions = {}

    vert\_definitions = {}

    current\_section = None

    current\_lines = []

    vert\_key\_index = 0

    vert\_keys = 'ABCDEFGHIJ'

    horiz\_key\_index = 1

    for line in filtered\_lines:

        if "HORIZONTALEMENT" in line:

            current\_section = "horizontal"

            current\_lines = []

            continue

        elif "VERTICALEMENT" in line:

            current\_section = "vertical"

            current\_lines = []

            continue

        if current\_section == "horizontal":

            horiz\_definitions[str(horiz\_key\_index)] = line.strip()

            horiz\_key\_index += 1

        elif current\_section == "vertical":

            if vert\_key\_index < len(vert\_keys):

                vert\_definitions[vert\_keys[vert\_key\_index]] = line.strip()

                vert\_key\_index += 1

    return {'HORIZONTALEMENT': horiz\_definitions, 'VERTICALEMENT': vert\_definitions}

def process\_definition(image\_path, api\_key):

    img = cv2.imread(image\_path)

    height, width, \_ = img.shape

    roi = img

    roi\_resized = cv2.resize(roi, (width // 2, height // 2))

    \_, compressedimage = cv2.imencode(".jpg", roi\_resized, [cv2.IMWRITE\_JPEG\_QUALITY, 80])

    file\_bytes = io.BytesIO(compressedimage)

    if file\_bytes.getbuffer().nbytes <= 1024 \* 1024:

        url\_api = "https://api.ocr.space/parse/image"

        files = {"file": ("definition.jpg", file\_bytes, "image/jpeg")}

        result = requests.post(url\_api,

                        files=files,

                        data={"apikey": api\_key,

                              "language": "fre"})

        result = result.content.decode()

        result = json.loads(result)

        if 'ParsedResults' in result:

            parsed\_results = result['ParsedResults'][0]

            ocr\_text = parsed\_results['ParsedText']

            print("OCR Text:", ocr\_text)

            valid\_lines = filter\_and\_store\_lines(ocr\_text)

            for line in valid\_lines:

                print(f"Filtered Line: {line}")

            filtered\_lines = remove\_keys\_from\_lines(valid\_lines)

            json\_result = process\_filtered\_lines(filtered\_lines)

            print(json\_result)

            return json\_result

        else:

            raise Exception("No results returned or an error occurred.")

    else:

        raise ValueError("Image size exceeds limit of 1MB.")

if \_\_name\_\_ == "\_\_main\_\_":

    if not os.path.exists('json'):

        os.makedirs('json')

    image\_path = "uploads/definition.jpg"

    api\_key = "K89603096888957"

    definitions = process\_definition(image\_path, api\_key)

    print(definitions)

    # Lưu kết quả vào tệp json/definition.json

    with open('json/definition.json', 'w', encoding='utf-8') as f:

        json.dump(definitions, f, ensure\_ascii=False, indent=4)

    print(f"JSON result saved to json/definition.json")

App.py

from flask import Flask, request, jsonify

import os

import process\_definition\_image

import process\_grid\_image

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = 'uploads'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

@app.route('/upload', methods=['POST'])

def upload\_file():

    if 'file' not in request.files:

        return jsonify({"error": "No file part"}), 400

    file = request.files['file']

    if file.filename == '':

        return jsonify({"error": "No selected file"}), 400

    if file:

        filename = file.filename

        file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        file.save(file\_path)

        return jsonify({"message": "File uploaded successfully", "filename": filename}), 200

@app.route('/process-grid', methods=['POST'])

def process\_grid():

    grid\_image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], 'grid.jpg')

    try:

        cropped\_crossword = process\_grid\_image.parse\_grid(grid\_image\_path)

        binary\_matrix = process\_grid\_image.convert\_to\_binary\_matrix(cropped\_crossword, crossword\_size=10)

        return jsonify({"matrix\_binaire": binary\_matrix.tolist()}), 200

    except Exception as e:

        print(f"Error: {str(e)}")

        return jsonify({"error": str(e)}), 500

@app.route('/process-definition', methods=['POST'])

def process\_definition():

    definition\_image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], 'definition.jpg')

    api\_key = "K89603096888957"  # Replace with your actual API key

    try:

        print(f"Processing definition for image: {definition\_image\_path}")

        definitions = process\_definition\_image.process\_definition(definition\_image\_path, api\_key)

        print(f"Definitions: {definitions}")

        return jsonify(definitions), 200

    except Exception as e:

        print(f"Error: {str(e)}")

        return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5000)

app.js

import React, { useState } from 'react';

import { StyleSheet, Text, View, Button, Image, ActivityIndicator, ScrollView } from 'react-native';

import \* as ImagePicker from 'expo-image-picker';

import axios from 'axios';

export default function App() {

  const [gridImage, setGridImage] = useState(null);

  const [definitionImage, setDefinitionImage] = useState(null);

  const [matrix, setMatrix] = useState(null);

  const [definitions, setDefinitions] = useState(null);

  const [loading, setLoading] = useState(false);

  const [page, setPage] = useState('welcome');

  const pickImage = async (setter) => {

    let result = await ImagePicker.launchImageLibraryAsync({

      mediaTypes: ImagePicker.MediaTypeOptions.Images,

      allowsEditing: true,

      aspect: [4, 3],

      quality: 1,

    });

    if (!result.canceled) {

      const uri = result.assets[0].uri;

      setter(uri);

      uploadImage(uri, setter === setGridImage ? 'grid.jpg' : 'definition.jpg');

    }

  };

  const uploadImage = async (uri, filename) => {

    const formData = new FormData();

    formData.append('file', {

      uri,

      name: filename,

      type: 'image/jpeg',

    });

    try {

      const response = await axios.post('http://192.168.1.10:5000/upload', formData, {

        headers: {

          'Content-Type': 'multipart/form-data',

        },

      });

      console.log('Upload success:', response.data);

    } catch (error) {

      console.error('Upload error:', error);

    }

  };

  const processGrid = async () => {

    setLoading(true);

    try {

      const response = await axios.post('http://192.168.1.10:5000/process-grid');

      console.log('Processing success:', response.data);

      setMatrix(response.data.matrix\_binaire);

      setPage('showGrid');

    } catch (error) {

      console.error('Processing error:', error);

    }

    setLoading(false);

  };

  const processDefinition = async () => {

    setLoading(true);

    try {

      const response = await axios.post('http://192.168.1.10:5000/process-definition');

      console.log('API Response:', response);

      if (response.data) {

        console.log('Processing success:', response.data);

        setDefinitions(response.data);

        setPage('showDefinitions');

      } else {

        console.error('Processing failed: No data received');

      }

    } catch (error) {

      console.error('Processing error:', error);

    }

    setLoading(false);

  };

  const AnimalAnimation = () => {

    const [position, setPosition] = useState(0);

    React.useEffect(() => {

      const interval = setInterval(() => {

        setPosition((prev) => (prev < 100 ? prev + 1 : 0));

      }, 50);

      return () => clearInterval(interval);

    }, []);

    return (

      <View style={[styles.animal, { left: `${position}%` }]}>

        <Text>🐾</Text>

      </View>

    );

  };

  if (page === 'welcome') {

    return (

      <View style={styles.container}>

        <Text>Welcome to the application to solve crossword puzzles</Text>

        <Button title="Start" onPress={() => setPage('addImages')} />

      </View>

    );

  }

  if (page === 'addImages') {

    return (

      <View style={styles.container}>

        <Text>Upload your images</Text>

        <View style={styles.buttonContainer}>

          <Button title="Pick Grid Image" onPress={() => pickImage(setGridImage)} />

          {gridImage && <Image source={{ uri: gridImage }} style={styles.image} />}

        </View>

        <View style={styles.buttonContainer}>

          <Button title="Pick Definition Image" onPress={() => pickImage(setDefinitionImage)} />

          {definitionImage && <Image source={{ uri: definitionImage }} style={styles.image} />}

        </View>

        {gridImage && definitionImage ? (

          <View>

            <Button title="Confirm Grid" onPress={() => processGrid()} />

            <Button title="Confirm Definitions" onPress={() => processDefinition()} />

          </View>

        ) : (

          <Button title="Back to Welcome" onPress={() => setPage('welcome')} />

        )}

        {loading && <ActivityIndicator size="large" color="#00ff00" />}

      </View>

    );

  }

  if (page === 'showGrid') {

    return (

      <View style={styles.container}>

        <Text>Here is your crossword grid:</Text>

        <View style={styles.matrixContainer}>

          {matrix && matrix.map((row, rowIndex) => (

            <View key={rowIndex} style={styles.matrixRow}>

              {row.map((cell, cellIndex) => (

                <View

                  key={cellIndex}

                  style={[

                    styles.matrixCell,

                    { backgroundColor: cell === 1 ? 'white' : 'orange' },

                  ]}

                />

              ))}

            </View>

          ))}

        </View>

        <Button title="Next Step" onPress={() => setPage('animalAnimation')} />

        <Button title="Back to Home" onPress={() => setPage('welcome')} />

      </View>

    );

  }

  if (page === 'showDefinitions') {

    return (

      <ScrollView contentContainerStyle={styles.container}>

        <Text>Definitions:</Text>

        <View>

          {definitions && Object.keys(definitions).map((direction) => (

            <View key={direction}>

              <Text style={styles.definitionHeader}>{direction}:</Text>

              {Object.keys(definitions[direction]).map((key) => (

                <View key={key} style={styles.definitionContainer}>

                  <Text style={styles.definitionKey}>{key}</Text>

                  <Text style={styles.definitionText}>{definitions[direction][key]}</Text>

                </View>

              ))}

            </View>

          ))}

        </View>

        <Button title="Back to Home" onPress={() => setPage('welcome')} />

      </ScrollView>

    );

  }

  if (page === 'animalAnimation') {

    return (

      <View style={styles.container}>

        <Text>Solving...</Text>

        <AnimalAnimation />

        <Button title="Back to Home" onPress={() => setPage('welcome')} />

      </View>

    );

  }

  return null;

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

    padding: 20,

  },

  buttonContainer: {

    margin: 10,

  },

  image: {

    width: 200,

    height: 200,

    margin: 10,

  },

  matrixContainer: {

    marginTop: 20,

  },

  matrixRow: {

    flexDirection: 'row',

  },

  matrixCell: {

    width: 20,

    height: 20,

    borderWidth: 1,

    borderColor: 'black',

  },

  animal: {

    position: 'absolute',

    top: '50%',

    marginTop: -10, // Adjust this value to vertically center the animal

  },

  definitionHeader: {

    fontWeight: 'bold',

    fontSize: 18,

    marginTop: 10,

  },

  definitionContainer: {

    flexDirection: 'row',

    alignItems: 'flex-start',

    marginTop: 5,

  },

  definitionKey: {

    fontWeight: 'bold',

    marginRight: 5,

  },

  definitionText: {

    flex: 1,

    flexWrap: 'wrap',

  },

});

RESULTS 1 : IMPORT IMAGES and ANALYZE IMAGES

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Code :

App.py :

import json

from flask import Flask, request, jsonify

import os

import process\_definition\_image

import process\_grid\_image

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = 'uploads'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

@app.route('/upload', methods=['POST'])

def upload\_file():

    if 'file' not in request.files:

        return jsonify({"error": "No file part"}), 400

    file = request.files['file']

    if file.filename == '':

        return jsonify({"error": "No selected file"}), 400

    if file:

        filename = file.filename

        file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        file.save(file\_path)

        return jsonify({"message": "File uploaded successfully", "filename": filename}), 200

@app.route('/process-grid', methods=['POST'])

def process\_grid():

    grid\_image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], 'grid.jpg')

    try:

        cropped\_crossword = process\_grid\_image.parse\_grid(grid\_image\_path)

        binary\_matrix = process\_grid\_image.convert\_to\_binary\_matrix(cropped\_crossword, crossword\_size=10)

        return jsonify({"matrix\_binaire": binary\_matrix.tolist()}), 200

    except Exception as e:

        print(f"Error: {str(e)}")

        return jsonify({"error": str(e)}), 500

@app.route('/process-definition', methods=['POST'])

def process\_definition():

    definition\_image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], 'definition.jpg')

    api\_key = "K89603096888957"

    try:

        print(f"Processing definition for image: {definition\_image\_path}")

        definitions = process\_definition\_image.process\_definition(definition\_image\_path, api\_key)

        if not os.path.exists('json'):

            os.makedirs('json')

        with open('json/definition.json', 'w', encoding='utf-8') as f:

            json.dump(definitions, f, ensure\_ascii=False, indent=4)

        print(f"JSON result saved to json/definition.json")

        return jsonify(definitions), 200

    except Exception as e:

        print(f"Error: {str(e)}")

        return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5000)

App.js

import React, { useState } from 'react';

import { StyleSheet, Text, View, Button, Image, ActivityIndicator, ScrollView, Dimensions } from 'react-native';

import \* as ImagePicker from 'expo-image-picker';

import axios from 'axios';

const { width, height } = Dimensions.get('window');

export default function App() {

  const [gridImage, setGridImage] = useState(null);

  const [definitionImage, setDefinitionImage] = useState(null);

  const [matrix, setMatrix] = useState(null);

  const [definitions, setDefinitions] = useState(null);

  const [loading, setLoading] = useState(false);

  const [page, setPage] = useState('welcome');

  const pickImage = async (setter) => {

    let result = await ImagePicker.launchImageLibraryAsync({

      mediaTypes: ImagePicker.MediaTypeOptions.Images,

      allowsEditing: true,

      aspect: [4, 3],

      quality: 1,

    });

    if (!result.canceled) {

      const uri = result.assets[0].uri;

      setter(uri);

      uploadImage(uri, setter === setGridImage ? 'grid.jpg' : 'definition.jpg');

    }

  };

  const uploadImage = async (uri, filename) => {

    const formData = new FormData();

    formData.append('file', {

      uri,

      name: filename,

      type: 'image/jpeg',

    });

    try {

      const response = await axios.post('http://192.168.1.10:5000/upload', formData, {

        headers: {

          'Content-Type': 'multipart/form-data',

        },

      });

      console.log('Upload success:', response.data);

    } catch (error) {

      console.error('Upload error:', error);

    }

  };

  const processImages = async () => {

    setLoading(true);

    try {

      const gridResponse = await axios.post('http://192.168.1.10:5000/process-grid');

      const definitionResponse = await axios.post('http://192.168.1.10:5000/process-definition');

      console.log('Processing success:', gridResponse.data, definitionResponse.data);

      setMatrix(gridResponse.data.matrix\_binaire);

      setDefinitions(definitionResponse.data);

      setPage('showGridAndDefinitions');

    } catch (error) {

      console.error('Processing error:', error);

    }

    setLoading(false);

  };

  const AnimalAnimation = () => {

    const [position, setPosition] = useState(0);

    React.useEffect(() => {

      const interval = setInterval(() => {

        setPosition((prev) => (prev < 100 ? prev + 1 : 0));

      }, 50);

      return () => clearInterval(interval);

    }, []);

    return (

      <View style={[styles.animal, { left: `${position}%` }]}>

        <Text>🐾</Text>

      </View>

    );

  };

  if (page === 'welcome') {

    return (

      <View style={styles.centeredContainer}>

        <Text>Welcome to the Crossword Solver App</Text>

        <Button title="Start" onPress={() => setPage('addImages')} />

      </View>

    );

  }

  if (page === 'addImages') {

    return (

      <View style={styles.centeredContainer}>

        <Text>Upload your images</Text>

        <View style={styles.buttonContainer}>

          <Button title="Pick Grid Image" onPress={() => pickImage(setGridImage)} />

          {gridImage && <Image source={{ uri: gridImage }} style={styles.image} />}

        </View>

        <View style={styles.buttonContainer}>

          <Button title="Pick Definition Image" onPress={() => pickImage(setDefinitionImage)} />

          {definitionImage && <Image source={{ uri: definitionImage }} style={styles.image} />}

        </View>

        {gridImage && definitionImage ? (

          <View>

            <Button title="Confirm" onPress={() => processImages()} />

            <Button title="Back to Welcome" onPress={() => setPage('welcome')} />

          </View>

        ) : (

          <Button title="Back to Welcome" onPress={() => setPage('welcome')} />

        )}

        {loading && <ActivityIndicator size="large" color="#00ff00" />}

      </View>

    );

  }

  if (page === 'showGridAndDefinitions') {

    return (

      <View style={styles.container}>

        <Text style={styles.title}>Here is your crossword grid and definitions:</Text>

        <View style={styles.gridAndDefinitionsContainer}>

          <View style={styles.gridContainer}>

            <Text style={styles.sectionTitle}>Crossword Grid</Text>

            <ScrollView horizontal>

              <View style={styles.matrixContainer}>

                {matrix && matrix.map((row, rowIndex) => (

                  <View key={rowIndex} style={styles.matrixRow}>

                    {row.map((cell, cellIndex) => (

                      <View

                        key={cellIndex}

                        style={[

                          styles.matrixCell,

                          { backgroundColor: cell === 1 ? 'white' : 'orange' },

                        ]}

                      />

                    ))}

                  </View>

                ))}

              </View>

            </ScrollView>

          </View>

          <ScrollView style={styles.definitionsScrollContainer}>

            <View style={styles.definitionsContainer}>

              <View style={styles.column}>

                <Text style={styles.definitionHeader}>HORIZONTALEMENT:</Text>

                {definitions && Object.keys(definitions.HORIZONTALEMENT).map((key) => (

                  <View key={key} style={styles.definitionContainer}>

                    <Text style={styles.definitionKey}>{key}</Text>

                    <Text style={styles.definitionText}>{definitions.HORIZONTALEMENT[key]}</Text>

                  </View>

                ))}

              </View>

              <View style={styles.column}>

                <Text style={styles.definitionHeader}>VERTICALEMENT:</Text>

                {definitions && Object.keys(definitions.VERTICALEMENT).map((key) => (

                  <View key={key} style={styles.definitionContainer}>

                    <Text style={styles.definitionKey}>{key}</Text>

                    <Text style={styles.definitionText}>{definitions.VERTICALEMENT[key]}</Text>

                  </View>

                ))}

              </View>

            </View>

          </ScrollView>

        </View>

        <Button title="Next" onPress={() => setPage('animalAnimation')} />

        <Button title="Back to Home" onPress={() => setPage('welcome')} />

      </View>

    );

  }

  if (page === 'animalAnimation') {

    return (

      <View style={styles.centeredContainer}>

        <Text>Solving...</Text>

        <AnimalAnimation />

        <Button title="Back to Home" onPress={() => setPage('welcome')} />

      </View>

    );

  }

  return null;

}

const styles = StyleSheet.create({

  centeredContainer: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

    padding: 20,

  },

  container: {

    flex: 1,

    backgroundColor: '#fff',

    padding: 20,

  },

  buttonContainer: {

    margin: 10,

  },

  image: {

    width: 200,

    height: 200,

    margin: 10,

  },

  gridAndDefinitionsContainer: {

    flex: 1,

    marginTop: 20,

  },

  gridContainer: {

    height: height \* 0.3,  // 1/3 của màn hình

    alignItems: 'center',

    marginBottom: 20,

  },

  definitionsScrollContainer: {

    height: height \* 0.6,  // 2/3 của màn hình

  },

  definitionsContainer: {

    flexDirection: 'row',

  },

  matrixContainer: {

    marginTop: 20,

  },

  matrixRow: {

    flexDirection: 'row',

  },

  matrixCell: {

    width: 15,

    height: 15,

    borderWidth: 1,

    borderColor: 'black',

  },

  column: {

    flex: 1,

    paddingHorizontal: 10,

  },

  definitionHeader: {

    fontWeight: 'bold',

    fontSize: 14,

    marginBottom: 10,

  },

  definitionContainer: {

    flexDirection: 'row',

    alignItems: 'flex-start',

    marginTop: 5,

  },

  definitionKey: {

    fontWeight: 'bold',

    marginRight: 5,

  },

  definitionText: {

    flex: 1,

    flexWrap: 'wrap',

  },

  title: {

    fontSize: 20,

    fontWeight: 'bold',

    marginVertical: 20,

    textAlign: 'center',

  },

  sectionTitle: {

    fontSize: 18,

    fontWeight: 'bold',

    marginBottom: 10,

    textAlign: 'center',

  },

  animal: {

    position: 'absolute',

    top: '50%',

    marginTop: -10,

  },

});

Definiton

import cv2

import numpy as np

import requests

import io

import json

import re

import os

def filter\_and\_store\_lines(ocr\_text):

    lines = ocr\_text.split('\n')

    valid\_lines = []

    for line in lines:

        line = line.strip()

        if not line:

            continue

        # Lọc các dòng chỉ có key hoặc chữ cái với các ký hiệu

        if re.match(r'^\d+[-.]?$|^[A-Za-z][-.,]?$', line):

            continue

        valid\_lines.append(line)

    return valid\_lines

def remove\_keys\_from\_lines(valid\_lines):

    filtered\_lines = []

    for line in valid\_lines:

        # Loại bỏ key có sẵn nếu có

        line\_without\_key = re.sub(r'^\d+[-.]?\s\*', '', line)

        filtered\_lines.append(line\_without\_key)

        print(f"Filtered Line without Key: {line\_without\_key}")

    return filtered\_lines

def process\_filtered\_lines(filtered\_lines):

    horiz\_definitions = {}

    vert\_definitions = {}

    current\_section = None

    current\_lines = []

    vert\_key\_index = 0

    vert\_keys = 'ABCDEFGHIJ'

    horiz\_key\_index = 1

    for line in filtered\_lines:

        if "HORIZONTALEMENT" in line:

            current\_section = "horizontal"

            current\_lines = []

            continue

        elif "VERTICALEMENT" in line:

            current\_section = "vertical"

            current\_lines = []

            continue

        if current\_section == "horizontal":

            horiz\_definitions[str(horiz\_key\_index)] = line.strip()

            horiz\_key\_index += 1

        elif current\_section == "vertical":

            if vert\_key\_index < len(vert\_keys):

                vert\_definitions[vert\_keys[vert\_key\_index]] = line.strip()

                vert\_key\_index += 1

    return {'HORIZONTALEMENT': horiz\_definitions, 'VERTICALEMENT': vert\_definitions}

def process\_definition(image\_path, api\_key):

    img = cv2.imread(image\_path)

    height, width, \_ = img.shape

    roi = img

    roi\_resized = cv2.resize(roi, (width // 2, height // 2))

    \_, compressedimage = cv2.imencode(".jpg", roi\_resized, [cv2.IMWRITE\_JPEG\_QUALITY, 80])

    file\_bytes = io.BytesIO(compressedimage)

    if file\_bytes.getbuffer().nbytes <= 1024 \* 1024:

        url\_api = "https://api.ocr.space/parse/image"

        files = {"file": ("definition.jpg", file\_bytes, "image/jpeg")}

        result = requests.post(url\_api,

                        files=files,

                        data={"apikey": api\_key,

                              "language": "fre"})

        result = result.content.decode()

        result = json.loads(result)

        if 'ParsedResults' in result:

            parsed\_results = result['ParsedResults'][0]

            ocr\_text = parsed\_results['ParsedText']

            print("OCR Text:", ocr\_text)

            valid\_lines = filter\_and\_store\_lines(ocr\_text)

            for line in valid\_lines:

                print(f"Filtered Line: {line}")

            filtered\_lines = remove\_keys\_from\_lines(valid\_lines)

            json\_result = process\_filtered\_lines(filtered\_lines)

            print(json\_result)

            return json\_result

        else:

            raise Exception("No results returned or an error occurred.")

    else:

        raise ValueError("Image size exceeds limit of 1MB.")

if \_\_name\_\_ == "\_\_main\_\_":

    if not os.path.exists('json'):

        os.makedirs('json')

    image\_path = "uploads/definition.jpg"

    api\_key = "K89603096888957"

    definitions = process\_definition(image\_path, api\_key)

    print(definitions)

    # Lưu kết quả vào tệp json/definition.json

    with open('json/definition.json', 'w', encoding='utf-8') as f:

        json.dump(definitions, f, ensure\_ascii=False, indent=4)

    print(f"JSON result saved to json/definition.json")

Grid

import json

import cv2

import numpy as np

import operator

import os

def distance\_between(p1, p2):

    """Returns the scalar distance between two points"""

    a = p2[0] - p1[0]

    b = p2[1] - p1[1]

    return np.sqrt((a \*\* 2) + (b \*\* 2))

def pre\_process\_image(img, skip\_dilate=False, flag=0):

    """Uses blurring, adaptive thresholding, and dilation to expose the main features of an image."""

    proc = cv2.GaussianBlur(img.copy(), (9, 9), 0)

    proc = cv2.adaptiveThreshold(proc, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)

    proc = cv2.bitwise\_not(proc, proc)

    if not skip\_dilate:

        kernel = np.array([[0., 1., 0.], [1., 1., 1.], [0., 1., 0.]], np.uint8)

        proc = cv2.dilate(proc, kernel)

    return proc

def find\_corners\_of\_largest\_polygon(img):

    """Finds the 4 extreme corners of the largest contour in the image."""

    contours, h = cv2.findContours(img.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    contours = sorted(contours, key=cv2.contourArea, reverse=True)

    polygon = contours[0]

    bottom\_right, \_ = max(enumerate([pt[0][0] + pt[0][1] for pt in polygon]), key=operator.itemgetter(1))

    top\_left, \_ = min(enumerate([pt[0][0] + pt[0][1] for pt in polygon]), key=operator.itemgetter(1))

    bottom\_left, \_ = min(enumerate([pt[0][0] - pt[0][1] for pt in polygon]), key=operator.itemgetter(1))

    top\_right, \_ = max(enumerate([pt[0][0] - pt[0][1] for pt in polygon]), key=operator.itemgetter(1))

    return [polygon[top\_left][0], polygon[top\_right][0], polygon[bottom\_right][0], polygon[bottom\_left][0]]

def crop\_and\_warp(img, crop\_rect, flag=0):

    """Crops and warps a rectangular section from an image into a square of similar size."""

    top\_left, top\_right, bottom\_right, bottom\_left = crop\_rect[0], crop\_rect[1], crop\_rect[2], crop\_rect[3]

    src = np.array([top\_left, top\_right, bottom\_right, bottom\_left], dtype='float32')

    side = max([

        distance\_between(bottom\_right, top\_right),

        distance\_between(top\_left, bottom\_left),

        distance\_between(bottom\_right, bottom\_left),

        distance\_between(top\_left, top\_right)

    ])

    dst = np.array([[0, 0], [side - 1, 0], [side - 1, side - 1], [0, side - 1]], dtype='float32')

    m = cv2.getPerspectiveTransform(src, dst)

    warp = cv2.warpPerspective(img, m, (int(side), int(side)))

    return warp

def parse\_grid(path, flag=0):

    original = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

    if original is None:  # Check if the image was loaded successfully

        raise FileNotFoundError(f"Unable to load image at path: {path}")

    processed = pre\_process\_image(original, flag=flag)

    corners = find\_corners\_of\_largest\_polygon(processed)

    if not corners:  # Check if corners were found

        raise ValueError("No corners were found in the image.")

    cropped = crop\_and\_warp(original, corners, flag)

    return cropped

def convert\_to\_binary\_matrix(img, crossword\_size=10):

    img\_resized = cv2.resize(img, (crossword\_size \* 10, crossword\_size \* 10))

    binary\_matrix = np.zeros((crossword\_size, crossword\_size), dtype=np.uint8)

    for i in range(crossword\_size):

        for j in range(crossword\_size):

            cell = img\_resized[i\*10:(i+1)\*10, j\*10:(j+1)\*10]

            if np.mean(cell) > 127:

                binary\_matrix[i, j] = 1

            else:

                binary\_matrix[i, j] = 0

    return binary\_matrix

def export\_to\_json(matrix, filename):

    matrix\_list = matrix.tolist()

    cross\_dict = {"matrix\_binaire": matrix\_list}

    with open(filename, 'w') as json\_file:

        json.dump(cross\_dict, json\_file, indent=4)

    print(f"Matrix exported to {filename}")