PROJECT CODE:

MAIN.PY

```
import os
import shutil
import numpy as np
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt
import argparse
from fastapi import FastAPI, UploadFile, File
from fastapi.responses import JSONResponse
import uvicorn
from io import BytesIO
from PIL import Image
import uuid
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
from mainBackUp import run_local_segmentation

app = FastAPI()

# Mount static file directory
app.mount("/output", StaticFiles(directory="output"), name="output")

# Enable CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```
UPLOAD_DIR = "uploads"
os.makedirs(UPLOAD_DIR, exist_ok=True)

# Load trained model
MODEL_PATH = "models/unet_model.h5"
model = tf.keras.models.load_model(MODEL_PATH)

# Image size
IMG_HEIGHT = 256
IMG_WIDTH = 256

COLOR_MAP = {
    0: [155, 155, 155],      # Background (black)
    1: [254, 221, 58],    # Vegetation (green)
    2: [226, 169, 41],    # Water (blue)
    3: [60,16,152],    # Buildings (red)
    4: [110, 193, 228],  # Roads (yellow)
}

def preprocess_image(image):
    """Preprocess input image for model prediction."""
    img = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
    img = img / 255.0  # Normalize
    return img

def predict_mask(image):
    """Predict segmentation mask using U-Net model."""
    img = preprocess_image(image)
    img_input = np.expand_dims(img, axis=0)
    predicted_mask = model.predict(img_input)[0]
    predicted_mask = (predicted_mask > 0.5).astype(np.uint8)
    return img, predicted_mask

def apply_colormap(mask):
```

```
    """Apply custom colormap to the segmentation mask."""
    mask = mask.squeeze()
    colored_mask = np.zeros((*mask.shape, 3), dtype=np.uint8)

    for key, color in COLOR_MAP.items():
        colored_mask[mask == key] = color

    return colored_mask

def overlay_mask(original, mask):
    """Overlay the color segmentation mask on the original image."""
        mask_resized = cv2.resize(mask, (original.shape[1], original.shape[0]),
interpolation=cv2.INTER_NEAREST)
    overlay = cv2.addWeighted(original, 0.6, mask_resized, 0.4, 0)  # Blend images
    return overlay

@app.post("/upload/")
async def upload_file(file: UploadFile = File(...)):
    """Handle file upload and return segmentation output URL."""
    file_path = os.path.join(UPLOAD_DIR, file.filename)

    with open(file_path, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)

    return JSONResponse(content={
        "filename": file.filename,
        "message": "File uploaded successfully",
        "file_path": f"/{file_path}",
        "output_url": f"/segment/{file.filename}"
    })

@app.post("/segment/")
async def segment_image(file: UploadFile = File(...)):
    """Process uploaded image, generate a segmentation mask, and return URLs."""
    try:
```

```
        image = Image.open(BytesIO(await file.read()))
        original, mask = predict_mask(image)

        original_rgb = np.array(image)
        colored_mask = apply_colormap(mask)
        overlayed_image = overlay_mask(original_rgb, colored_mask)

        unique_id = str(uuid.uuid4())[:8]
        mask_filename = f"mask_{unique_id}.png"
        overlay_filename = f"overlay_{unique_id}.png"

        mask_path = os.path.join("output", mask_filename)
        overlay_path = os.path.join("output", overlay_filename)

        os.makedirs("output", exist_ok=True)
                                    cv2.imwrite(mask_path,     cv2.cvtColor(colored_mask,
cv2.COLOR_RGB2BGR))
                                cv2.imwrite(overlay_path,    cv2.cvtColor(overlayed_image,
cv2.COLOR_RGB2BGR))

        return JSONResponse(content={
            "message": "Segmentation successful",
            "mask_url": f"http://localhost:8000/output/{mask_filename}",
            "overlay_url": f"http://localhost:8000/output/{overlay_filename}"
        })
    except Exception as e:
        return JSONResponse(content={"error": str(e)}, status_code=500)

if _name_ == "_main_":
    parser = argparse.ArgumentParser(description="Run U-Net Segmentation")
    parser.add_argument("--image", type=str, help="Path to test image")
            parser.add_argument("--server",   action="store_true",   help="Run   FastAPI
server")

    args = parser.parse_args()
```

```
    if args.server:
        uvicorn.run(app, host="127.0.0.1", port=8000)
    elif args.image:
        run_local_segmentation(args.image)
    else:
        print("❌ Please provide --image <path> or --server to run API.")
```

APP.JSX

```
// export default App;
import { useState } from "react";
import axios from "axios";

function App() {
  const [image, setImage] = useState(null);
  const [maskImage, setMaskImage] = useState(null);
  const [overlayImage, setOverlayImage] = useState(null);
  const [loading, setLoading] = useState(false);

  const handleFileChange = (event) => {
    setImage(event.target.files[0]);
    setMaskImage(null);
    setOverlayImage(null);
  };

  const handleUpload = async () => {
    if (!image) {
      alert("Please select an image first!");
      return;
    }

    const formData = new FormData();
    formData.append("file", image);
```

```
  setLoading(true);
  setMaskImage(null);
  setOverlayImage(null);

  try {
    const response = await axios.post("http://localhost:8000/segment/", formData);

    console.log("API Response:", response.data);

    if (response.data.mask_url && response.data.overlay_url) {
      setMaskImage(response.data.mask_url);
      setOverlayImage(response.data.overlay_url);
    } else {
      alert("Failed to get processed images.");
    }
  } catch (error) {
    console.error("Error uploading image:", error);
    alert("Error processing image. Please try again.");
  } finally {
    setLoading(false);
  }
};

return (
  <div style={styles.container}>
    <h1 style={styles.heading}>U-Net Image Segmentation</h1>

    <div style={styles.inputContainer}>
      <input type="file" onChange={handleFileChange} style={styles.fileInput} />
      <button
        onClick={handleUpload}
        disabled={loading}
        style={{ ...styles.button, backgroundColor: loading ? "#ccc" : "#28a745" }}
      >
        {loading ? "Processing..." : "Upload & Segment"}
```

```
      </button>
    </div>

    {image && (
      <div style={styles.imageContainer}>
        <h3>Original Image:</h3>
                        <img   src={URL.createObjectURL(image)}   alt="Uploaded"
style={styles.image} />
      </div>
    )}

    {maskImage && overlayImage && (
      <div style={styles.sideBySideContainer}>
        <div style={styles.imageContainer}>
          <h3>Predicted Mask:</h3>
          <img src={maskImage} alt="Predicted Mask" style={styles.image} />
        </div>
        <div style={styles.imageContainer}>
          <h3>Segmented Overlay:</h3>
            <img src={overlayImage} alt="Segmented Overlay" style={styles.image}
/>
        </div>
      </div>
    )}
  </div>
 );
}

const styles = {
 container: {
   textAlign: "center",
   padding: "40px",
   fontFamily: "'Poppins', sans-serif",
   background: "linear-gradient(135deg, #74ebd5, #acb6e5)",
   minHeight: "100vh",
```

```
  display: "flex",
  flexDirection: "column",
  alignItems: "center",
  justifyContent: "center",
},
heading: {
  marginBottom: "30px",
  color: "#222",
  fontSize: "28px",
  fontWeight: "600",
},
inputContainer: {
  marginBottom: "25px",
},
fileInput: {
  padding: "10px",
  border: "2px solid #ccc",
  borderRadius: "5px",
  fontSize: "16px",
  cursor: "pointer",
},
button: {
  padding: "12px 20px",
  fontSize: "16px",
  cursor: "pointer",
  color: "#fff",
  border: "none",
  borderRadius: "6px",
  marginLeft: "10px",
  transition: "background 0.3s ease",
},
sideBySideContainer: {
  display: "flex",
  justifyContent: "center",
  gap: "30px",
```

```
    marginTop: "25px",
  },
  imageContainer: {
    textAlign: "center",
    backgroundColor: "#fff",
    padding: "15px",
    borderRadius: "8px",
    boxShadow: "0 4px 8px rgba(0, 0, 0, 0.1)",
  },
  image: {
    width: "320px",
    marginTop: "10px",
    borderRadius: "8px",
    boxShadow: "0px 4px 6px rgba(0, 0, 0, 0.15)",
  },
};

export default App;
```