ДОДАТОК А

Структури проектування користувацького інтерфейсу

Таблиця А.1 – Компонентно-функціональна структура для користувачів

Мета	Задачі	Процедури	Дії користувачів та системи
1	2	3	4
1			Дізнатись, чи є зображення дипфейком
	1.1		Натиснути кнопку «Choose image»
		1.1.1	Завантажити зображення з комп'ютеру
		1.1.2	Натиснути кнопку «Analyze image». При оновленні сторінки система повертається до початкового стану
		1.1.3	Отримати результат аналізу. При оновленні сторінки система повертається до початкового стану
		1.1.4	Отримати інформацію про систему
2			Отримати інформацію про систему
	2.1		Отримати інформацію про систему з веб-сторінки
		2.1.1	Натиснути кнопку «About» на навігаційній панелі

Таблиця А.2 – Функціонально-об’єктна структура для користувачів

Номер процедури	Ергатичний елемент	Засоби інтерфейсу	Знаряддя праці	Предмет праці	Продукти праці
1	2	3	4	5	6
1.1.1	Користувач	Веб-сторінка	Рука, миш, комп’ютер	Браузер, інтерфейс	Відкриття вікна вибору зображення
1.1.2				Файлова система	Обране зображення
1.1.3				Браузер, інтерфейс	Відправка зображення на сервер
1.1.4					Отриманий текстовий результат аналізу
1.2.1					Текстовий опис системи

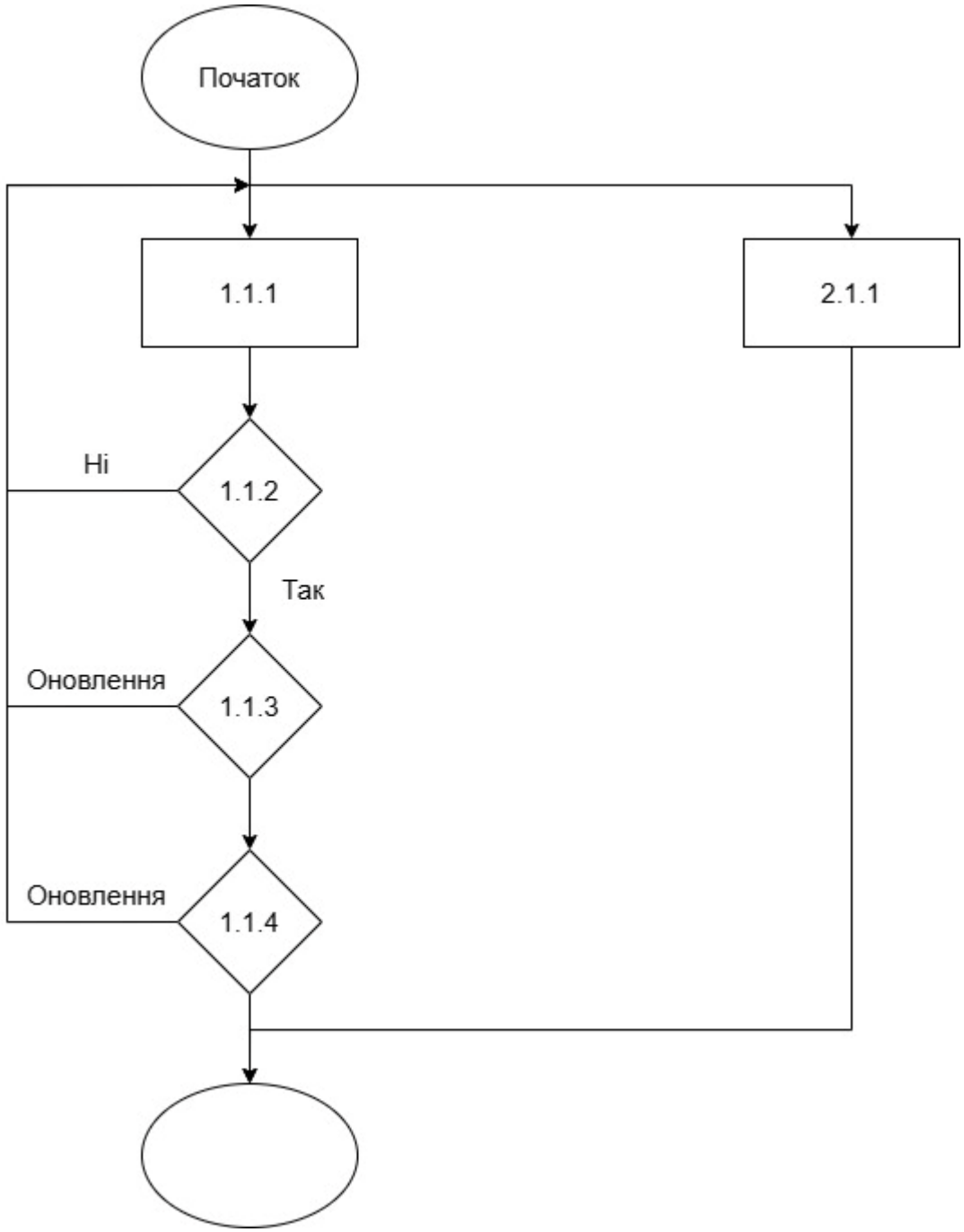

Рисунок А.1 – Функціонально-часова структура

ДОДАТОК Б

Файли з програмним кодом системи

Лістинг Б.1 – Реалізація серверної частини

```
from flask import Flask, request, jsonify
import tempfile
from PIL import Image
import cv2
import os
from classifier import predict
from segmentation import count_limbs, visualize_limbs
from light import detect_light
from Gemini import analyze_image
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route('/analyze', methods=['POST'])
def analyze():
    if 'image' not in request.files:
        return jsonify({'error': 'No image uploaded'}), 400

    file = request.files['image']

            with    tempfile.NamedTemporaryFile(delete=False,
suffix=".jpg") as temp_file:
        image_path = temp_file.name
        file.save(image_path)

    try:
        print("STEP 1: Opening original image")
        original = Image.open(image_path).convert("RGB")
        print("original:", type(original))
```

Продовження лістингу Б.1

```
            print("STEP 2: Running prediction")
            probabilities = predict(original)
            print("probabilities:", probabilities)

            print("STEP 3: Counting limbs")
            limbs_count = count_limbs(original)
            print("limbs_count:", limbs_count)

            print("STEP 4: Visualizing limbs")
            visualized_limbs = visualize_limbs(original)
            print("visualized_limbs:", type(visualized_limbs))

            print("STEP 5: Detecting shadows")
            shadow_image = detect_light(original)
            print("shadow_image:", type(shadow_image))

            print("STEP 6: Calling analyze_image()")
                    result_text   =   analyze_image(original,
visualized_limbs, shadow_image, probabilities, limbs_count)

            return jsonify({'analysis': result_text})

        except Exception as e:
            print("Exception:", e)
            return jsonify({'error': str(e)}), 500

        finally:
            os.remove(image_path)

    if __name__ == '__main__':
        app.run(debug=True)
```

Лістинг Б.2 – Тренування моделі ResNet-50

```
import torch
from torch import nn
from torchvision import transforms
from torch.utils.data import DataLoader
from torchvision.models import resnet50, ResNet50_Weights
from tqdm.notebook import tqdm
from PIL import Image
import random
from sklearn.metrics import classification_report
from datasets import load_dataset
from datasets import DatasetDict
from PIL import Image
import random

def train_model(model, train_loader, device, lr, epochs):
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    model.train()
    for epoch in range(epochs):
        total_loss = 0
          progress_bar = tqdm(train_loader, desc=f"Epoch
{epoch+1}/{epochs}")
        for images, labels in progress_bar:
                images, labels = images.to(device),
labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
                progress_bar.set_postfix({'loss':
f'{loss.item():.4f}'})
```

Продовження лістингу Б.2

```
                print(f"Epoch [{epoch+1}/{epochs}], Loss:
{total_loss / len(train_loader):.4f}")

    def evaluate_model(model, dataloader, criterion,
class_names, device):
        model.eval()
        total_loss = 0
        correct = 0
        total = 0
        all_preds = []
        all_labels = []

        with torch.no_grad():
                for images, labels in tqdm(dataloader,
desc="Evaluating"):
                    images, labels = images.to(device),
labels.to(device)
                outputs = model(images)

                loss = criterion(outputs, labels)
                total_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
                correct += (predicted == labels).sum().item()
                total += labels.size(0)

                all_preds.extend(predicted.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        avg_loss = total_loss / len(dataloader)
        print(f"Loss: {avg_loss:.4f}")

        report = classification_report(all_labels, all_preds,
target_names=class_names)
```

Продовження лістингу Б.2

```
        print(f"Classification Report:\n{report}")

    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")

    random.seed(42)

    from datasets import load_dataset
    from torchvision import transforms
    from torch.utils.data import DataLoader
    from datasets import DatasetDict
    from PIL import Image
    import random

    dataset  =  load_dataset("prithivMLmods/AI-vs-Deepfake-vs-
Real")

    dataset = dataset["train"].train_test_split(test_size=0.2,
seed=42)
    dataset = DatasetDict({
        "train": dataset["train"],
        "val": dataset["test"]
    })

    train_transforms = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
```

Продовження лістингу Б.2

```
    val_transforms = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    def transform_batch(batch, transform):
        images = []
        for img in batch["image"]:
                 image = Image.open(img).convert("RGB") if
isinstance(img, str) else img.convert("RGB")
            images.append(transform(image))
        batch["pixel_values"] = images
        return batch

    dataset["train"] = dataset["train"].with_transform(lambda
batch: transform_batch(batch, train_transforms))
    dataset["val"] = dataset["val"].with_transform(lambda
batch: transform_batch(batch, val_transforms))

    def collate_fn(batch):
         pixel_values = torch.stack([item["pixel_values"] for
item in batch])
        labels = torch.tensor([item["label"] for item in batch])
        return pixel_values, labels

    train_loader = DataLoader(dataset["train"], batch_size=32,
shuffle=True, collate_fn=collate_fn)
    val_loader = DataLoader(dataset["val"], batch_size=32,
collate_fn=collate_fn)

    criterion = nn.CrossEntropyLoss()
```

Продовження лістингу Б.2

```
    weights = ResNet50_Weights.IMAGENET1K_V2
    model = resnet50(weights=weights)

    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, 3)

    train_model(model, train_loader, device, lr=0.001,
epochs=5)
    evaluate_model(model, val_loader, criterion, ['AI',
'FAKE', 'REAL'], device)
    torch.save(model.state_dict(),
'/kaggle/working/model.pth')
```

Лістинг Б.3 – Інференс моделі ResNet-50

```
    import torch
    import torch.nn.functional as F
    from torchvision.models import resnet50, ResNet50_Weights
    from PIL import Image

    device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')

    class_names = ['Artificial', 'Deepfake', 'Real']
    weights = ResNet50_Weights.IMAGENET1K_V2
    model = resnet50(weights=weights).to(device)
    num_features = model.fc.in_features
    model.fc = torch.nn.Linear(num_features, 3).to(device)
    model.load_state_dict(torch.load('models/model.pth',
map_location=device))
    model.eval()
    preprocess = weights.transforms()

    def predict(image: Image.Image):
        image = image.convert("RGB")
```

Продовження лістингу Б.3

```
        input_tensor = preprocess(image).unsqueeze(0)
        input_tensor = input_tensor.to(device)
        with torch.no_grad():
            logits = model(input_tensor)
            probs = F.softmax(logits, dim=1)
             percentages = [round(p.item() * 100, 1) for p in
probs[0]]
          return {class_names[i]: percentages[i] for i in
range(len(class_names))}
```

Лістинг Б.4 – Тренування моделі Mask R-CNN

```
from pycocotools.coco import COCO
import os
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import functional as F
import torchvision.transforms as T
from PIL import Image
import numpy as np
from tqdm import tqdm
import torchvision
from torchvision.models.detection import MaskRCNN
from torchvision.models.detection.backbone_utils import
resnet_fpn_backbone
from torch.utils.data import Subset
import random

device = torch.device('cuda') if torch.cuda.is_available()
else torch.device('cpu')

class COCOLimbSegmentationDataset(Dataset):
    def __init__(self, root, annFile, transforms=None):
        self.root = root
        self.coco = COCO(annFile)
```

Продовження лістингу Б.4

```
            self.transforms = transforms
            self.img_ids = list(self.coco.imgs.keys())
                                    self.person_cat_id       =
self.coco.getCatIds(catNms=['person'])

        def __getitem__(self, idx):
            img_id = self.img_ids[idx]
                ann_ids  =  self.coco.getAnnIds(imgIds=img_id,
catIds=self.person_cat_id, iscrowd=None)
            anns = self.coco.loadAnns(ann_ids)
            path = self.coco.loadImgs(img_id)[0]['file_name']
                   img   =   Image.open(os.path.join(self.root,
path)).convert("RGB")
            masks = []
            boxes = []
            labels = []

            for ann in anns:
                            if   'segmentation'   in   ann   and
isinstance(ann['segmentation'], list):
                    mask = self.coco.annToMask(ann)
                    if mask.sum() > 1000:
                        masks.append(mask)
                        bbox = ann['bbox']
                       boxes.append([bbox[0], bbox[1], bbox[0]
+ bbox[2], bbox[1] + bbox[3]])
                        labels.append(1)

            if len(masks) == 0:
                masks = torch.zeros((0, img.height, img.width),
dtype=torch.uint8)
               boxes = torch.zeros((0, 4), dtype=torch.float32)
                labels = torch.tensor([], dtype=torch.int64)
            else:
```

Продовження лістингу Б.4

```
                    masks = torch.as_tensor(np.stack(masks),
dtype=torch.uint8)
                        boxes = torch.as_tensor(boxes,
dtype=torch.float32)
                       labels = torch.as_tensor(labels,
dtype=torch.int64)

            target = {
                "boxes": boxes,
                "labels": labels,
                "masks": masks,
                "image_id": torch.tensor([img_id])
            }

            if self.transforms:
                img = self.transforms(img)
            return img, target

        def __len__(self):
            return len(self.img_ids)

    def segmentation_model(num_classes):
              backbone = resnet_fpn_backbone('resnet50',
pretrained=True)
        model = MaskRCNN(backbone, num_classes=num_classes)
        return model

    def get_transform():
        return T.Compose([T.ToTensor()])

    random.seed(42)

    dataset = COCOLimbSegmentationDataset(
```

Продовження лістингу Б.4

```
                                root="/kaggle/input/coco-2017-
dataset/coco2017/train2017",
                               annFile="/kaggle/input/coco-2017-
dataset/coco2017/annotations/person_keypoints_train2017.json",
        transforms=get_transform())
    num_samples = int(0.1 * len(dataset))
    indices = random.sample(range(len(dataset)), num_samples)
    subset_dataset = Subset(dataset, indices)
    data_loader = DataLoader(
        subset_dataset,
        batch_size=4,
        shuffle=True,
        collate_fn=lambda x: tuple(zip(*x)))

    model = segmentation_model(2).to(device)

    params = [p for p in model.parameters() if p.requires_grad]
    optimizer     =     torch.optim.AdamW(params,     lr=1e-3,
weight_decay=0.01)
    epochs = 5

    for epoch in range(epochs):
        model.train()
        epoch_loss = 0.0
                with     tqdm(data_loader,     desc=f"Epoch
{epoch+1}/{epochs}") as pbar:
            for images, targets in pbar:
                images = list(img.to(device) for img in images)
                    targets = [{k: v.to(device) for k, v in
t.items()} for t in targets]

                loss_dict = model(images, targets)
               losses = sum(loss for loss in loss_dict.values())
```

Продовження лістингу Б.4

```
                optimizer.zero_grad()
                losses.backward()
                optimizer.step()

                loss_value = losses.item()
                epoch_loss += loss_value
                pbar.set_postfix(loss=loss_value)

        avg_loss = epoch_loss / len(data_loader)
              print(f"[Epoch    {epoch+1}]    Average    Loss:
{avg_loss:.4f}")
                                  torch.save(model.state_dict(),
f"/kaggle/working/model3_epoch_{epoch+1}.pth")
```

Лістинг Б.5 – Інференс моделі Mask R-CNN

```
import torch
from torchvision.models.detection import MaskRCNN
from  torchvision.models.detection.backbone_utils  import
resnet_fpn_backbone
from torchvision.transforms import functional as F
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from PIL import ImageDraw
import cv2

model_path = "models/model_epoch_10.pth"
confidence = 0.5

def segmentation_model(num_classes):
          backbone    =    resnet_fpn_backbone('resnet50',
pretrained=False)
        model = MaskRCNN(backbone, num_classes=num_classes)
        return model
```

Продовження лістингу Б.5

```
    model = segmentation_model(num_classes=2)
    model.load_state_dict(torch.load(model_path,
map_location='cpu'))
    model.eval()

    def count_limbs(image: Image.Image, threshold=confidence)
-> int:
        image_tensor = F.to_tensor(image).unsqueeze(0)
        with torch.no_grad():
            predictions = model(image_tensor)[0]
        limb_count = sum(
            1 for i in range(len(predictions["scores"]))
            if predictions["scores"][i].item() >= threshold and
predictions["labels"][i].item() == 1)
        return limb_count

    def visualize_limbs(image: Image.Image,
threshold=confidence) -> Image.Image:
        image_tensor = F.to_tensor(image).unsqueeze(0)
        with torch.no_grad():
            predictions = model(image_tensor)[0]

        valid_masks = [
            predictions["masks"][i] > 0.5
            for i in range(len(predictions["scores"]))
            if predictions["scores"][i].item() >= threshold and
predictions["labels"][i].item() == 1
        ]

        image_np = np.array(image).astype(np.uint8).copy()

        for mask in valid_masks:
                                            mask_np =
mask.squeeze().cpu().numpy().astype(np.uint8) * 255
```

Продовження лістингу Б.5

```
                contours, _ = cv2.findContours(mask_np,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            cv2.drawContours(image_np, contours, -1, (255, 0,
0), thickness=2)
        return Image.fromarray(image_np)
```

Лістинг Б.6 – Модуль аналізу освітлення

```
import cv2
import mediapipe as mp
import numpy as np
from typing import Union
from PIL import Image

mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils

def detect_light(image_input: Image.Image) -> Image.Image:
    image_np = np.array(image_input).astype(np.uint8)
    image = cv2.cvtColor(image_np, cv2.COLOR_RGB2BGR)

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (7, 7), 0)

    inverted = cv2.bitwise_not(blurred)
      _, shadow_mask = cv2.threshold(inverted, 50, 255,
cv2.THRESH_BINARY)

        shadow_boost = cv2.bitwise_and(image, image,
mask=shadow_mask)
    enhanced = cv2.addWeighted(image, 1.0, shadow_boost, -
0.5, 0)

          return Image.fromarray(cv2.cvtColor(enhanced,
cv2.COLOR_BGR2RGB))
```

Лістинг Б.7 – Модуль великої мовної моделі

```
import google.generativeai as genai
from PIL import Image
import re

API_KEY = "AIzaSyD5atbin6Gtl3ea0CE0kdN4UoFZgKblmyA"
genai.configure(api_key=API_KEY)

def build_prompt(probabilities, limbs_count) -> str:
    return f"""
There is probabilities of the image being a deepfake, real or artificial, in percentage:
{probabilities}

Analyze the image based on the following parameters:
1. Facial features: Look for inconsistencies in facial features such as eyes, mouth, and nose.
2. Body: Check for unnatural body poses or extra limbs that do not match the body's natural anatomy.
3. Lighting: Ensure that the lighting and shadows on the face matches the rest of the image.

You are provided with three images:
- Original image
- Image with visualized limbs. The limbs count is {limbs_count}
- Image with enhanced lighting and shadows

Think step by step. Provide a detailed and structured analysis using all images and the probabilities. After that, write a clear and concise conclusion in the following format:
Answer the question: Does this image look like a deepfake or not?
```

Продовження лістингу Б.7

```
    Final answer: [Yes, it looks like a deepfake/No, it doesn't
look like a deepfake], because [brief explaining].
    """

    def analyze_image(original: Image.Image, pose: Image.Image,
shadow: Image.Image, probabilities: dict, limbs_count) -> str:
        prompt = build_prompt(probabilities, limbs_count)
        model = genai.GenerativeModel('gemini-2.5-pro-preview-
06-05')

         response = model.generate_content([prompt, original,
pose, shadow], stream=False)
             match = re.search(r"Final answer:\s*(.*)",
response.text)
        if match:
            extracted_text = match.group(1).strip()
            cleaned_text = extracted_text.replace('**', '')
            return cleaned_text
        else:
            return response.text
```

Лістинг Б.8 – Користувацький інтерфейс головної сторінки

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Common Sense Image Analysis</title>
  <link rel="stylesheet" href="styles.css" />
</head>
<body>
  <nav class="navbar">
      <div class="nav-links">
                  <a   href="index.html"   class="nav-link
active">Home</a>
```

Продовження лістингу Б.8

```
            <a href="about.html" class="nav-link">About</a>
          </div>
        </nav>
        <div class="container">
          <div class="header">
            <h1>Common Sense Image Analysis</h1>
            <p class="subtitle">
                 Upload an image for deepfake detection and
authenticity analysis
            </p>
          </div>

          <div class="upload-area" id="uploadArea">
            <div class="upload-icon">📁</div>
            <p class="upload-text">Drag & drop your image</p>
            <p class="upload-hint">or click to browse files (JPG,
PNG)</p>
             <input type="file" id="imageInput" accept="image/*"
/>
          </div>

          <img id="preview" />

         <button class="btn btn-primary" onclick="uploadImage()"
id="analyzeBtn">
            Analyze Image
          </button>
          <div id="result">
            <div class="result-header">
              <div class="result-icon">🔍</div>
              <div class="result-title">Analysis Result</div>
            </div>
           <div class="result-content" id="resultContent"></div>
```

Продовження лістингу Б.8

```
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

Лістинг Б.9 – Користувацький інтерфейс інформаційної сторінки

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>About</title>
  <link rel="stylesheet" href="styles.css" />
</head>
<body>
  <nav class="navbar">
      <div class="nav-links">
        <a href="index.html" class="nav-link">Home</a>
                  <a   href="about.html"   class="nav-link
active">About</a>
      </div>
  </nav>
  <div class="container">
    <div class="header">
      <h1>About</h1>
      <p class="subtitle2">
         This system is an experimental tool designed to
assist users in detecting deepfake images through the
application of commonsense reasoning, based common sense
reasoning.
      </p>
       <ul class="subtitle2"> The model consists of several
modules:
```

Продовження лістингу Б.9

```
                <li>Keypoints analysis module</li>
                <li>Lighting analysis module</li>
                <li>Classifier module</li>
                <li>GPT module</li>
            </ul>
          <p class="subtitle2">
            It helps to explain the results of the analysis and provide insights into the authenticity of the image.
          </p>
        </div>
    </body>
    </html>
```

Лістинг Б.10 – Стилі користувацького інтерфейсу

```
    * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
                font-family:     'Inter',     -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen, Ubuntu, sans-serif;
    }

    body {
        background: linear-gradient(135deg, #f5f9ff 0%, #f0f7fa 100%);
        min-height: 100vh;
        display: flex;
        justify-content: center;
        align-items: center;
        padding: 80px;
    }
    .container {
```

Продовження лістингу Б.10

```
    background: rgba(255, 255, 255, 0.95);
    backdrop-filter: blur(10px);
    border-radius: 24px;
    box-shadow: 0 12px 40px rgba(0, 0, 0, 0.08);
    width: 100%;
    max-width: 520px;
    padding: 40px;
    text-align: center;
    border: 1px solid rgba(241, 245, 249, 0.6);
    transform: translateY(0);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.container:hover {
    box-shadow: 0 15px 50px rgba(0, 0, 0, 0.12);
    transform: translateY(-5px);
}

.header {
    margin-bottom: 32px;
}

    .navbar {
      position: fixed;
      top: 0;
      left: 0;
      width: 100%;
      display: flex;
      justify-content: center;
      padding: 20px 0;
      background: rgba(255, 255, 255, 0.95);
      backdrop-filter: blur(10px);
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);
      z-index: 1000;
```

Продовження лістингу Б.10

```
    }

    .nav-links {
      display: flex;
      gap: 20px;
      background: rgba(241, 245, 249, 0.6);
      padding: 10px 20px;
      border-radius: 50px;
      border: 1px solid rgba(226, 232, 240, 0.6);
    }

    .nav-link {
      padding: 10px 24px;
      border-radius: 50px;
      text-decoration: none;
      font-weight: 600;
      color: #64748b;
      transition: all 0.3s ease;
    }

    .nav-link:hover {
      color: #3b82f6;
      background: rgba(226, 232, 240, 0.4);
    }

    .nav-link.active {
      color: white;
         background:  linear-gradient(90deg,  #4361ee  0%,
#3a0ca3 100%);
      box-shadow: 0 2px 8px rgba(67, 97, 238, 0.3);
    }

  h1 {
    font-size: 32px;
```

Продовження лістингу Б.10

```
        font-weight: 700;
        background: linear-gradient(90deg, #4361ee 0%, #3a0ca3
100%);
        -webkit-background-clip: text;
        background-clip: text;
        color: transparent;
        margin-bottom: 12px;
        letter-spacing: -0.5px;
    }

    .subtitle {
        color: #64748b;
        font-size: 16px;
        font-weight: 500;
        line-height: 1.6;
        max-width: 80%;
        margin: 0 auto;
    }

    .subtitle2 {
        color: #64748b;
        font-size: 16px;
        font-weight: 500;
        line-height: 1.6;
        max-width: 80%;
        margin: 10px 0 0 24px;
        padding-left: 0;
        text-align: left;
    }

    .subtitle2 ul {
        margin: 10px 0 20px 24px;
        padding-left: 24px;
        list-style-type: disc;
```

Продовження лістингу Б.10

```
}

.upload-area {
    border: 2px dashed #dbeafe;
    border-radius: 16px;
    padding: 40px 30px;
    margin-bottom: 30px;
    background: #f8fafc;
    transition: all 0.3s ease;
    cursor: pointer;
    position: relative;
}

.upload-area:hover {
    border-color: #93c5fd;
    background: #f1f8ff;
}

.upload-area.active {
    border-color: #60a5fa;
    background: #dbeafe;
}

.upload-icon {
    font-size: 48px;
    margin-bottom: 16px;
    color: #3b82f6;
}

.upload-text {
    font-size: 18px;
    font-weight: 600;
    color: #1e293b;
    margin-bottom: 8px;
```

Продовження лістингу Б.10

```
}

.upload-hint {
    color: #94a3b8;
    font-size: 14px;
    margin-top: 6px;
}

input[type="file"] {
    display: none;
}

#preview {
    width: 100%;
    max-width: 100%;
    max-height: 400px;
    border-radius: 12px;
    margin: 25px 0;
    display: none;
    object-fit: contain;
    background-color: #fff;
    padding: 10px;
    box-shadow: 0 6px 15px rgba(0, 0, 0, 0.08);
}

.btn {
    display: inline-flex;
    align-items: center;
    justify-content: center;
    padding: 16px 32px;
    font-size: 16px;
    font-weight: 600;
    border-radius: 12px;
    border: none;
```

Продовження лістингу Б.10

```
        cursor: pointer;
        transition: all 0.25s cubic-bezier(0.4, 0, 0.2, 1);
        box-shadow: 0 4px 6px rgba(50, 92, 241, 0.15);
        margin: 10px 0;
    }

    .btn-primary {
        background: linear-gradient(90deg, #4361ee 0%, #3a0ca3
100%);
        color: white;
        padding: 16px 40px;
    }

    .btn-primary:hover {
        transform: translateY(-2px);
        box-shadow: 0 7px 14px rgba(50, 92, 241, 0.25);
    }

    .btn-primary:active {
        transform: translateY(0);
    }

    .btn:disabled {
        opacity: 0.7;
        cursor: not-allowed;
        transform: none;
        box-shadow: none;
    }

    #result {
        background: #f8fafc;
        border-radius: 16px;
        padding: 24px;
        margin-top: 30px;
```

Продовження лістингу Б.10

```
    text-align: left;
    display: none;
    border-left: 4px solid #3b82f6;
    animation: fadeIn 0.4s ease-out;
}

.result-header {
    display: flex;
    align-items: center;
    margin-bottom: 16px;
}

.result-icon {
    font-size: 24px;
    margin-right: 12px;
}

.result-title {
    font-size: 18px;
    font-weight: 700;
    color: #1e293b;
}

.result-content {
    color: #334155;
    line-height: 1.7;
    font-size: 16px;
    white-space: pre-wrap;
}

.loading {
    display: inline-flex;
    align-items: center;
    gap: 12px;
```

Продовження лістингу Б.10

```
    font-weight: 600;
    color: #3b82f6;
}

.spinner {
    width: 24px;
    height: 24px;
    border: 3px solid rgba(59, 130, 246, 0.2);
    border-top: 3px solid #3b82f6;
    border-radius: 50%;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}

@media (max-width: 600px) {
    .container {
        padding: 30px 20px;
    }
    h1 {
        font-size: 26px;
    }
    .subtitle {
        font-size: 14px;
    }
}
```

Лістинг Б.11 – Скрипти користувацького інтерфейсу

```
const uploadArea = document.getElementById('uploadArea');
const imageInput = document.getElementById('imageInput');
const preview = document.getElementById('preview');
const analyzeBtn = document.getElementById('analyzeBtn');
const resultDiv = document.getElementById('result');
const resultContent = document.getElementById('resultContent');

uploadArea.addEventListener('dragover', (e) => {
  e.preventDefault();
  uploadArea.classList.add('active');
});

uploadArea.addEventListener('dragleave', () => {
  uploadArea.classList.remove('active');
});

uploadArea.addEventListener('drop', (e) => {
  e.preventDefault();
  uploadArea.classList.remove('active');

  if (e.dataTransfer.files.length) {
    imageInput.files = e.dataTransfer.files;
    previewImage();
  }
});

uploadArea.addEventListener('click', () => {
  imageInput.click();
});

imageInput.addEventListener('change', previewImage);

function previewImage() {
```

Продовження лістингу Б.11

```
    if (imageInput.files && imageInput.files[0]) {
      const reader = new FileReader();
      reader.onload = function (e) {
        preview.src = e.target.result;
        preview.style.display = 'block';
        resultDiv.style.display = 'none';
      };
      reader.readAsDataURL(imageInput.files[0]);
    }
  }

  async function uploadImage() {
    resultContent.textContent = '';
    resultDiv.style.display = 'none';

    if (!imageInput.files.length) {
      alert('Please select an image first');
      return;
    }

    analyzeBtn.disabled = true;
    analyzeBtn.innerHTML =
        '<div  class="loading"><div  class="spinner"></div>
Analyzing...</div>';

    const formData = new FormData();
    formData.append('image', imageInput.files[0]);

    try {
                const       response       =       await
fetch('http://localhost:5000/analyze', {
        method: 'POST',
        body: formData,
      });
```

Продовження лістингу Б.11

```
        const data = await response.json();

        resultDiv.style.display = 'block';

        if (data.analysis) {
          resultContent.textContent = data.analysis;
        } else if (data.error) {
          resultContent.textContent = 'Error: ' + data.error;
        } else {
          resultContent.textContent = 'Unexpected response from
server';
        }
      } catch (error) {
        resultDiv.style.display = 'block';
          resultContent.textContent = 'Request failed: ' +
error.message;
      } finally {
        analyzeBtn.disabled = false;
        analyzeBtn.textContent = 'Analyze Image';
      }
    }
```

ДОДАТОК В

Скріншоти результатів роботи системи

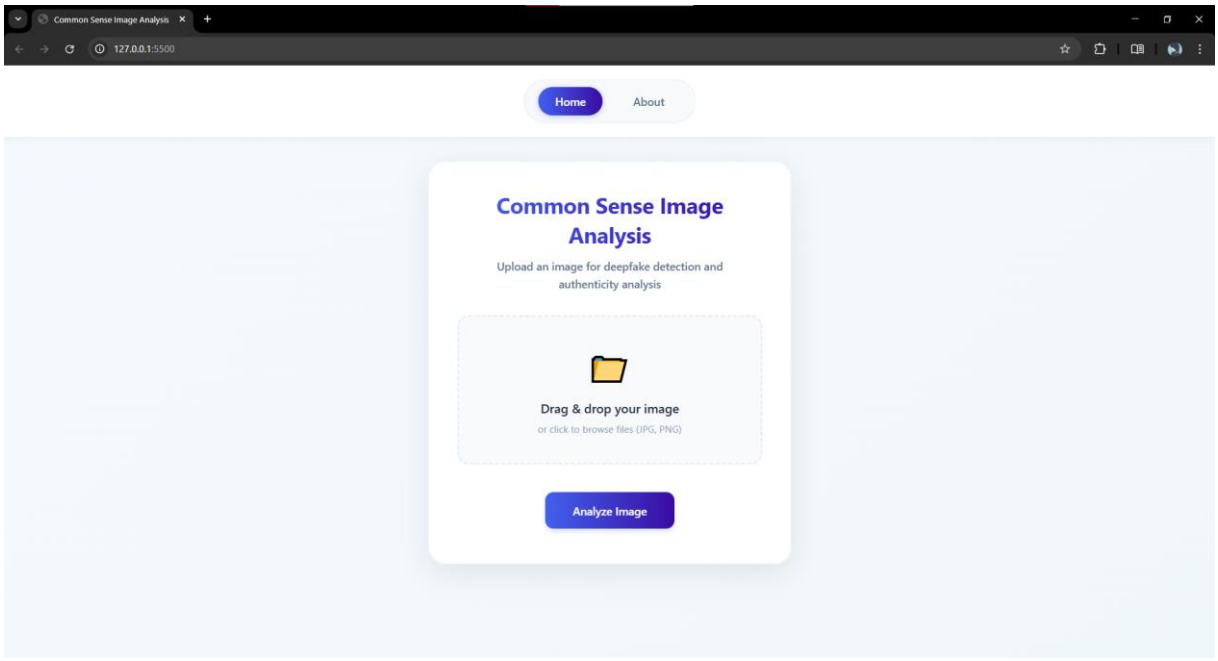

Рисунок В.1 – Скріншот головної сторінки

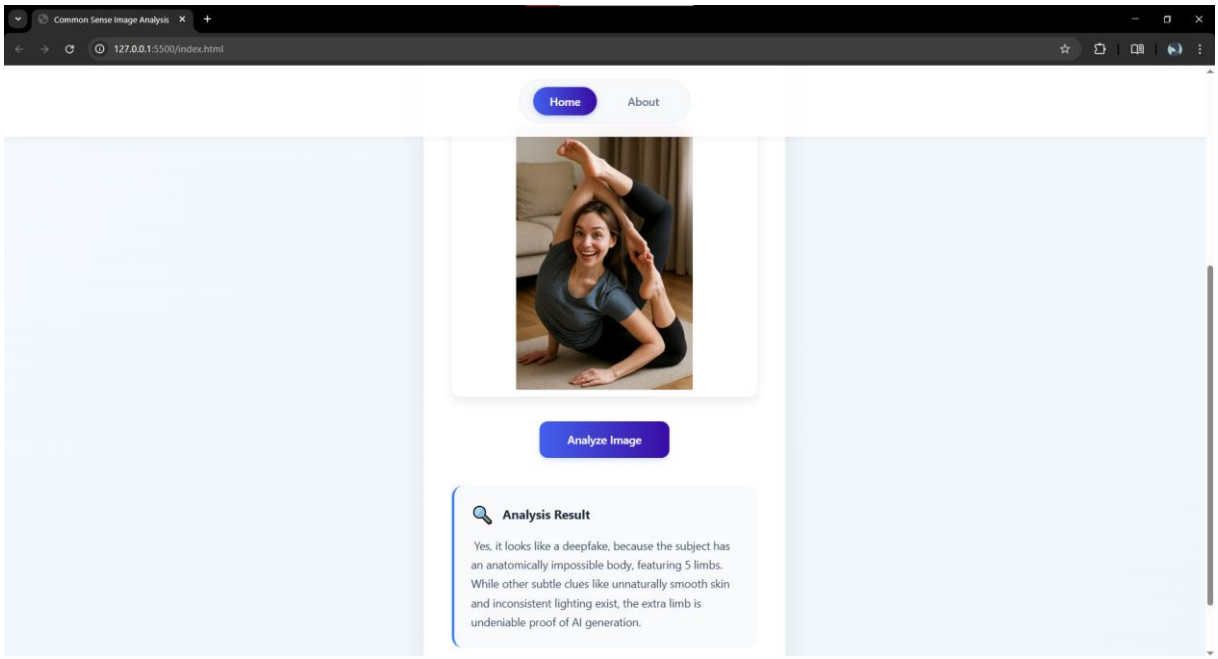

Рисунок В.2 – Скріншот результату обробки першого зображення

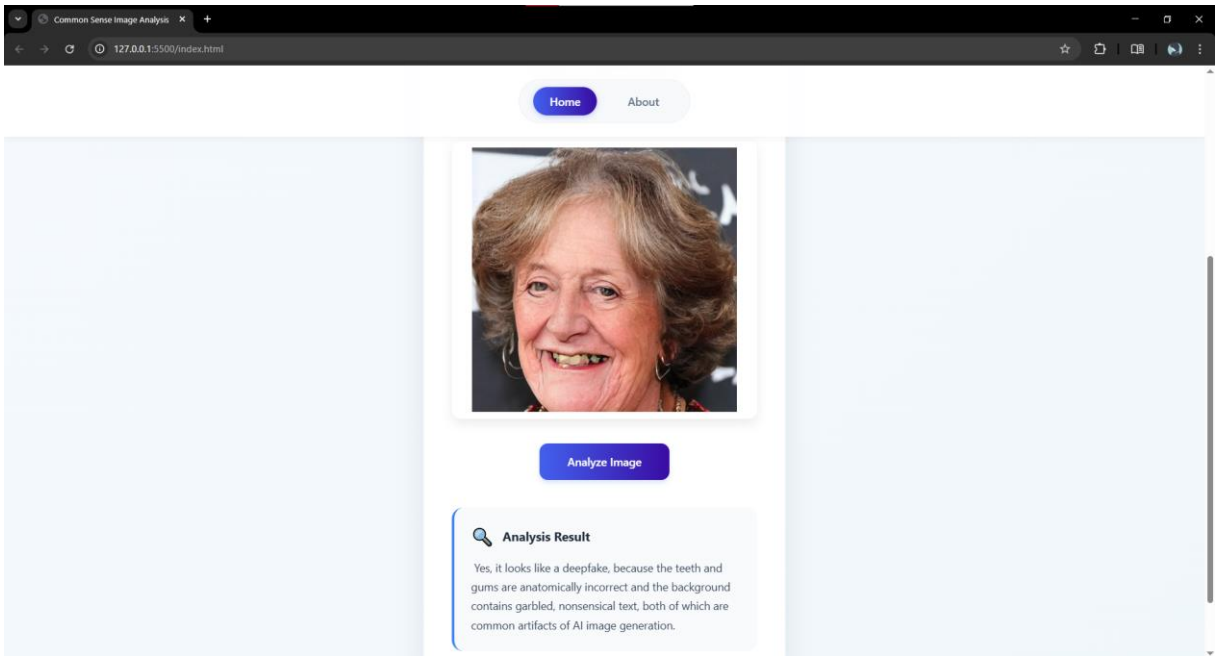

Рисунок В.3 – Скріншот результату обробки другого зображення

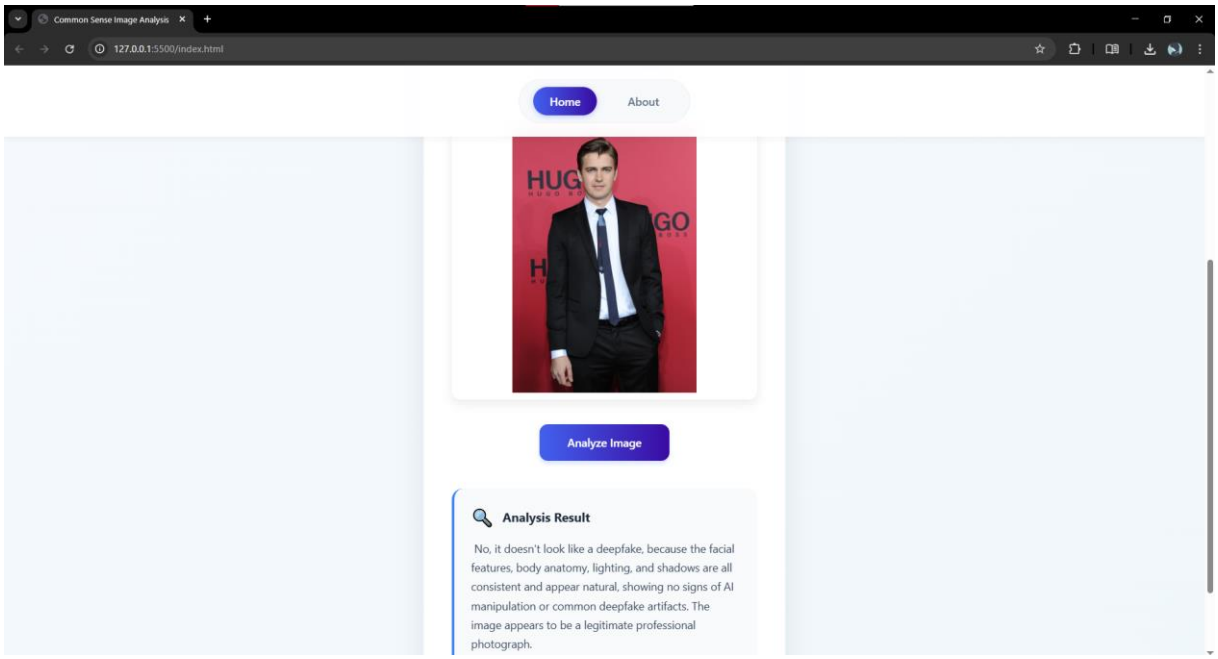

Рисунок В.4 – Скріншот результату обробки третього зображення

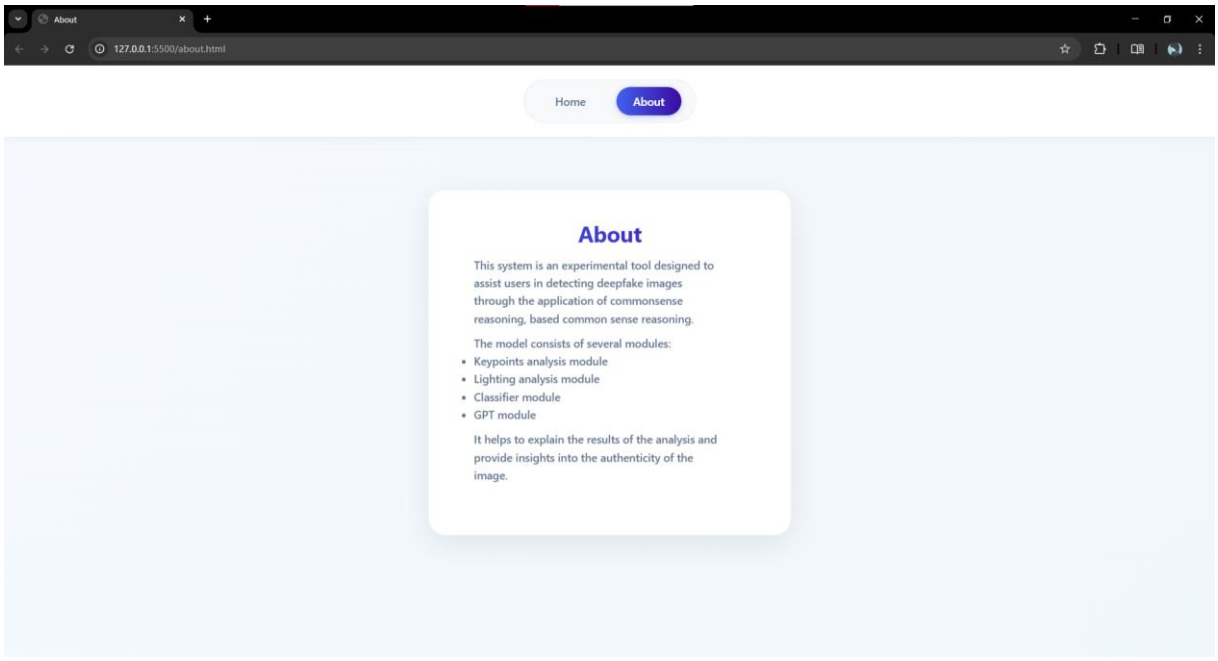

Рисунок В.5 – Скріншот інформаційної сторінки

ДОДАТОК Г

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	118 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Почерніна О.М.			Технології міркування здорового глузду для обробки зображень та виявлення дипфейків	Шифр групи	Код напр./спец.
Перевірив		Головянко М.В.				ІТШІ-21-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					