**Курсов проект**

**Дисциплина: Проектиране и интегриране на софтуерни системи**

**Реализация на системата Distributed Image Lab**

**Версия 1.x**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Фак. №** | **Име на студент** | **.NET Backend (Producer)** | **Python Worker (Consumer)** | **Frontend (UI)** | **Инфраструктура** |
| **2MI0600012** | **Таня Илиева** | File System & Controller | Pillow Effects (Standard) | HTML Structure & Razor | Architecture Docs |
| **9MI0600020** | **Клементина Картевска** | Redis Messaging | AI Logic & Event Loop | JavaScript & CSS | Docker & Scripts |

**Януари, 2025**

**Съдържание**

[1 Въведение 3](#_heading=h.xemzwfj9yv3q)

[1.1 Цел 3](#_heading=h.v67xirehh8p5)

[1.2 Резюме 3](#_heading=h.iy67an12yg3z)

[1.3 Дефиниции и акроними 3](#_heading=h.61y3k8qiqn8c)

[2 Използвани технологии 4](#_heading=h.hs0lw6fmxfjx)

[3 Реализация на базата от данни 5](#_heading=h.23kqt3k7bb35)

[4 Реализация на бизнес логиката 6](#_heading=h.em76finnhqyl)

[5 Реализация на потребителския интерфейс 7](#_heading=h.9kd0i4mxww31)

[6 Внедряване на системата 8](#_heading=h.wzz6knq1ocqf)

[7 Разпределение на дейностите по реализацията 9](#_heading=h.rj1o7d2kzmkt)

[8 Приложения 10](#_heading=h.euew9z1u8jbx)

# Въведение

## Цел

Целта на този документ е да опише техническата реализация на системата **"Distributed Image Lab"**. Документът детайлизира архитектурните решения, използваните технологии, структурата на данните и алгоритмите, залегнали в разработката на разпределеното приложение за асинхронна обработка на изображения. Той служи като ръководство за разбиране на вътрешното действие на системата и процеса на нейното внедряване.

## Резюме

Документът е структуриран в няколко основни части. Първо са представени използваните технологии и инструменти за разработка (Секция 2). След това се разглежда реализацията на комуникационния слой и данните (Секция 3). Основният акцент е поставен върху бизнес логиката, реализирана чрез модела Producer-Consumer (Секция 4), и потребителския интерфейс (Секция 5). Накрая са описани стъпките за внедряване и стартиране на системата (Секция 6).

## Дефиниции и акроними

* ***Producer-Consumer Pattern***: Архитектурен модел, при който една част от системата (Producer) генерира задачи, а друга (Consumer) ги обработва асинхронно.
* ***Redis (Remote Dictionary Server)***: In-memory структура от данни, използвана в проекта като брокер на съобщения (Message Broker) и опашка за задачи.
* ***ASP.NET Core MVC***: Уеб рамка за изграждане на уеб приложения, използваща архитектурата Model-View-Controller.
* ***Pillow***: Библиотека на Python за обработка на изображения (Python Imaging Library).
* ***Docker***: Платформа за контейнеризация, използвана за лесно стартиране на инфраструктурата (Redis).
* ***Polling***: Техника, при която клиентът (браузърът) периодично запитва сървъра за нови данни.

# Използвани технологии

* ***Backend (Web Server / Producer):***
  + ***Език***: C#
  + ***Framework***: .NET 8 (ASP.NET Core MVC)
  + ***Роля***: Приема потребителски заявки, валидира входните данни, записва файловете и изпраща задачи към опашката.
* ***Backend (Worker / Consumer):***
  + ***Език:***Python 3.x
  + ***Библиотеки:***
    - ***redis****:* За комуникация с Redis сървъра.
    - ***Pillow (PIL)****:* За манипулация на изображения (филтри, ротация, воден знак).
    - ***rembg и onnxruntime****:* За AI-базирано премахване на фон.
  + ***Роля:***Слуша опашката за нови задачи и извършва тежката изчислителна обработка.
* ***Инфраструктура:***
  + ***Redis( Message Broke):***Използва се за съхранение на опашката от задачи (List структура).
  + ***Docker & Docker Compose:***За контейнеризиране и управление на Redis инстанцията.
  + ***Batch Scripts (.bat):***За автоматизирано стартиране на цялата среда (StartALL.bat).
* ***Frontend:***
  + ***HTML5 / CSS3:***За структура и стил (Bootstrap framework).
  + ***JavaScript:***За асинхронно обновяване на галерията (AJAX/Fetch API).

# Реализация на данните

Тъй като системата обработва файлове, основното съхранение на изображенията се извършва върху файловата система (в директория wwwroot/uploads). Ролята на база данни се изпълнява от Redis, който съхранява метаданните за задачите. Структурата е следната:

* ***Тип****:* Key-Value Store (In-memory).
* ***Ключ на опашката****:* image\_queue
* ***Тип данни****:* Списък (List), реализиращ FIFO (First-In-First-Out) опашка.
* ***Формат на съобщенията****:* JSON стринг.

Примерна структура на JSON обект в Redis:

*JSON*

*{*

*"Path": "D:\\Project\\wwwroot\\uploads\\image123.jpg",*

*"Rotation": "left",*

*"Effect": "sepia",*

*"Watermark": "My Project 2025"*

*"Watermark": "My Project 2025"*

*}*

Този подход позволява на уеб приложението и работника да бъдат напълно независими (decoupled), споделяйки само достъп до Redis и файловата система.

# Реализация на бизнес логиката

Бизнес логиката е разделена на два основни компонента, работещи паралелно:

* ***Уеб компонент (Producer - C#)***

Реализиран е в HomeController.cs. Процесът включва:

* + **UploadImage (POST)**: Приема файл и параметри от формата.
  + **File IO**: Записва оригиналния файл на диска с уникално име (GUID).
  + **Serialization**: Създава обект със задачите и го сериализира до JSON.
  + **Enqueue**: Използва Redis клиента (StackExchange.Redis), за да добави съобщението в списъка image\_queue чрез команда ListLeftPushAsync.
* ***Работник (Consumer - Python)***

Реализиран е в worker.py. Процесът е безкраен цикъл (while True):

* + **Dequeue**: Използва brpop (Blocking Pop), за да чака задача от Redis. Това предотвратява натоварване на процесора, когато няма задачи.
  + **Deserialization**: Разчита JSON съобщението.
  + **Processing** (Chain of Responsibility):
    - Зарежда изображението с Pillow.
    - Прилага ротация (ако е избрана).
    - Прилага ефект (филтри или AI премахване на фон).
    - Добавя воден знак (ако е въведен текст).
  + **Output**: Записва обработения файл с префикс processed в същата директория.

# Реализация на потребителския интерфейс

Потребителският интерфейс е реализиран чрез Razor Views (Index.cshtml) и предоставя интуитивен начин за взаимодействие.

Основни елементи:

* ***Форма за качване****:* Позволява избор на файл, dropdown менюта за трансформации и текстово поле за воден знак. При submit бутонът се деактивира и показва "Spinner", за да индикира работа.
* ***Галерия с резултати****:* Секция, която визуализира обработените изображения.
* ***Динамично обновяване (Client-side Polling):***
  + JavaScript функция updateGallery() се извиква на всеки 3 секунди чрез setInterval.
  + Тя прави fetch заявка към endpoint /Home/GetProcessedImages.
  + Ако сървърът върне списък нови файлове, чрез DOM манипулация дървото се обновява автоматично, без да се презарежда страницата.
  + Добавени са CSS анимации (fade-in) за по-добро потребителско преживяване при поява на нова снимка.

# Внедряване на системата

За внедряване на системата е създаден автоматизиран процес, който минимизира ръчните настройки.

**Изисквания:**

* *Инсталиран Docker Desktop.*
* *Инсталиран .NET 8 SDK.*
* *Инсталиран Python 3.x.*

**Последователност на стартиране (реализирана в StartALL.bat):**

* ***Инфраструктура:***Стартиране на docker compose up, което вдига Redis сървър на порт 6379.
* ***Уеб Сървър****:* Навигиране до папка WebApp и изпълнение на dotnet run.
* ***Работник:***Навигиране до папка Worker, активиране на Python виртуална среда (venv), инсталиране на зависимостите (pip install -r requirements.txt) и стартиране на worker.py.

Този подход гарантира, че всички компоненти се стартират в правилния ред и с необходимите зависимости.

# Разпределение на дейностите по реализацията

Екипът възприе подхода на съвместна разработка (Co-Development), като трите ключови компонента на системата бяха разделени логически, за да може всеки студент да допринесе равностойно за всяка част от архитектурата.

1. Компонент 1: .NET Web Producer (Backend)

Този компонент отговаря за приемането на файлове и комуникацията с Redis.

* + Таня Илиева: Реализира логиката за управление на файловата система (създаване на директории, запис на файлове, генериране на уникални имена) и базовия контролер (HomeController.cs).
  + Клементина Картевска: Реализира логиката за сериализация на задачите в JSON формат и асинхронното им изпращане към Redis опашката (ListLeftPushAsync).

1. Компонент 2: Python Image Worker (Consumer)

Този компонент извършва същинската обработка на изображенията.

* + Таня Илиева: Разработи алгоритмите за стандартна обработка чрез библиотеката Pillow (филтри, ротация, воден знак).
  + Клементина Картевска: Разработи основния цикъл (Event Loop), който слуша Redis за нови съобщения, и интегрира AI модула (rembg) за премахване на фон.

1. Компонент 3: User Interface (Frontend)

Този компонент осигурява визуализацията и обратната връзка с потребителя.

* + Таня Илиева: Изгради HTML структурата и Razor Views (Index.cshtml), включително формата за качване и макета на галерията.
  + Клементина Картевска: Написа JavaScript логиката за динамично обновяване на галерията (Polling механизъм) и CSS стилизацията за адаптивен дизайн.

1. Допълнително: Клементина подготви инфраструктурните скриптове за Docker, а Таня оформи документацията на архитектурата.

# Приложения

* ***Приложение 1:*** *Изходен код на HomeController.cs*

using Microsoft.AspNetCore.Mvc;

using StackExchange.Redis;

using WebApp.Models;

using System.Text.Json;

*namespace* WebApp.Controllers;

public *class* HomeController : Controller

{

    private static Lazy<ConnectionMultiplexer> \_lazyRedis = new Lazy<ConnectionMultiplexer>(() => {

    return ConnectionMultiplexer.Connect("127.0.0.1");

});

public static ConnectionMultiplexer Redis => \_lazyRedis.Value;

    public async Task<IActionResult> Index()

    {

        try {

            using (*var* client = new HttpClient()) {

                client.DefaultRequestHeaders.UserAgent.ParseAdd("Mozilla/5.0");

                ViewBag.Weather = await client.GetStringAsync("https://wttr.in/Sofia?format=3");

            }

        } catch { ViewBag.Weather = "Weather service unavailable"; }

*var* uploadDir = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "uploads");

        if (Directory.Exists(uploadDir))

        {

*var* files = Directory.GetFiles(uploadDir, "processed\_\*.\*")

                                 .Select(f => Path.GetFileName(f))

                                 .OrderByDescending(f => f)

                                 .ToList();

            ViewBag.Images = files;

        }

        else

        {

            ViewBag.Images = new List<string>();

        }

        return View("Index");

    }

    public IActionResult Privacy()

    {

        return View();

    }

    [HttpPost]

    [HttpPost]

    public async Task<IActionResult> UploadImage(IFormFile file, string rotation, string effect, string watermarkText)

    {

        if (file != null && file.Length > 0)

        {

*var* uploadDir = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "uploads");

            if (!Directory.Exists(uploadDir)) Directory.CreateDirectory(uploadDir);

*var* fileName = Guid.NewGuid().ToString() + Path.GetExtension(file.FileName);

*var* fullPath = Path.Combine(uploadDir, fileName);

            using (*var* stream = new FileStream(fullPath, FileMode.Create))

            {

                await file.CopyToAsync(stream);

            }

*var* taskData = new

            {

                Path = fullPath,

                Rotation = rotation ?? "none",

                Effect = effect ?? "none",

                Watermark = watermarkText ?? ""

            };

            string jsonMessage = JsonSerializer.Serialize(taskData);

            try

            {

*var* db = Redis.GetDatabase();

                await db.ListLeftPushAsync("image\_queue", jsonMessage);

                ViewBag.Message = "Снимката е изпратена успешно!";

            }

            catch (Exception ex)

            {

                ViewBag.Message = $"ГРЕШКА: {ex.Message}";

            }

        }

        return await Index();

    }

    [HttpGet]

    public IActionResult GetProcessedImages()

    {

*var* uploadDir = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "uploads");

        if (!Directory.Exists(uploadDir)) return Json(new List<string>());

*var* files = Directory.GetFiles(uploadDir, "processed\_\*.\*")

                             .Select(f => Path.GetFileName(f))

                             .OrderByDescending(f => f)

                             .ToList();

        return Json(files);

    }

}

* ***Приложение 2:*** *Изходен код на worker.py*

import redis

import time

import os

import sys

import json

from rembg import remove

try:

    from PIL import Image, ImageFilter, ImageOps, ImageDraw

except *ImportError*:

**print**("Error: Missing libraries. Run: pip install redis pillow")

    sys.exit(1)

try:

    r = redis.Redis(*host*='localhost', *port*=6379, *db*=0)

except:

**print**("Error: Redis is not reachable.")

    sys.exit(1)

**print**(" [\*] Mega Worker v4.0 started. Waiting for tasks...")

while True:

    try:

        item = r.brpop('image\_queue', *timeout*=5)

    except:

        continue

    if item:

        try:

            json\_str = item[1].decode('utf-8')

            data = json.loads(json\_str)

            file\_path = data['Path']

            rotation = data.get('Rotation', 'none')

            effect = data.get('Effect', 'none')

            watermark\_text = data.get('Watermark', '')

**print**(*f*" [x] Task: Rotate={rotation}, Effect={effect}, Watermark={watermark\_text}")

            if os.path.exists(file\_path):

                with Image.open(file\_path) as img:

                    if img.mode != 'RGBA' and img.mode != 'RGB':

                        img = img.convert('RGBA')

                    processed = img

                    if rotation == 'left':

                        processed = processed.rotate(90, *expand*=True)

                    elif rotation == 'right':

                        processed = processed.rotate(-90, *expand*=True)

                    elif rotation == '180':

                        processed = processed.rotate(180, *expand*=True)

                    if effect == 'grayscale':

                        processed = processed.convert("L")

                    elif effect == 'sepia':

                        gray = processed.convert("L")

                        processed = ImageOps.colorize(gray, "#704214", "#C0C0C0")

                    elif effect == 'invert':

                        if processed.mode == 'RGBA':

                             processed = processed.convert('RGB')

                        processed = ImageOps.invert(processed)

                    elif effect == 'blur':

                        processed = processed.filter(ImageFilter.GaussianBlur(5))

                    elif effect == 'sharpen':

                        processed = processed.filter(ImageFilter.SHARPEN)

                    elif effect == 'emboss':

                        processed = processed.filter(ImageFilter.EMBOSS)

                    elif effect == 'contour':

                        processed = processed.filter(ImageFilter.CONTOUR)

                    elif effect == 'remove\_bg':

**print**(" [.] Running AI Background Removal...")

                        processed = remove(processed)

                    else:

                        processed = img.convert("L")

                    if watermark\_text:

                        if processed.mode != 'RGB' and processed.mode != 'RGBA':

                            processed = processed.convert('RGB')

                        draw = ImageDraw.Draw(processed)

                        draw.text((20, 20), watermark\_text, *fill*=(255, 0, 0))

**print**(*f*" [i] Watermark added.")

                    directory = os.path.dirname(file\_path)

                    filename = os.path.basename(file\_path)

                    filename\_no\_ext = os.path.splitext(filename)[0]

                    new\_path = os.path.join(directory, *f*"processed\_{rotation}\_{effect}\_{filename\_no\_ext}.png")

                    processed.save(new\_path)

**print**(*f*" [V] Done! Saved as {os.path.basename(new\_path)}")

            else:

**print**(" [!] File missing.")

        except *Exception* as e:

**print**(*f*" [!] Error: {e}")

* ***Приложение 3:*** *Конфигурация docker-compose.yml*

services:

  redis:

    image: redis:alpine

    ports:

      - "6379:6379"

  redis-commander:

    image: rediscommander/redis-commander:latest

    environment:

      - REDIS\_HOSTS=local:redis:6379

    ports:

      - "8081:8081"

    depends\_on:

      - redis