бюджетное профессиональное образовательное учреждение Вологодской области

«Череповецкий лесомеханический техникум им. В.П. Чкалова»

Специальность **09.02.07** «Информационные системы и программирование»

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**ПП по ПМ.03 РЕВЬЮИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ**

Выполнил студент 3 курса группы ИС-31

Шевель Милена Александровна

подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

место практики: ООО «Малленом Системс»

Период прохождения:

с «08» декабря 2024 года

по «21» декабря 2024 года

Руководитель практики от

предприятия, должность:

Южакова Н.В., специалист по кадрам

подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

МПРуководитель практики от

техникума: Материкова А.А.

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 года

Оглавление

[1. Выполнение заданий 3](#_Toc184822848)

[1.1. Задания первой недели 3](#_Toc184822849)

[1.1.1. Создание GIT репозитория «Practice\_PP03\_IS31» для производственной практики. 3](#_Toc184822850)

[1.1.2. Разработка модуля обработки и работы с изображениями и модуля взаимодействия с пользователем, формирования и хранения данных: Изменение размера изображения. 3](#_Toc184822851)

[1.1.3. Разработка модуля обработки и работы с изображениями и модуля взаимодействия с пользователем, формирования и хранения данных: Склеивание изображений в одно изображение. 8](#_Toc184822852)

[1.2. Задания второй недели 15](#_Toc184822853)

[1.2.1. Проведение обратного проектирования, используя графический язык UML . Создание и описание диаграммы Компонентов, Сценариев использования, Последовательностей, Деятельности. Изменение размера изображения. 15](#_Toc184822854)

# Выполнение заданий

## Задания первой недели

### Создание GIT репозитория «Practice\_PP03\_IS31» для производственной практики.

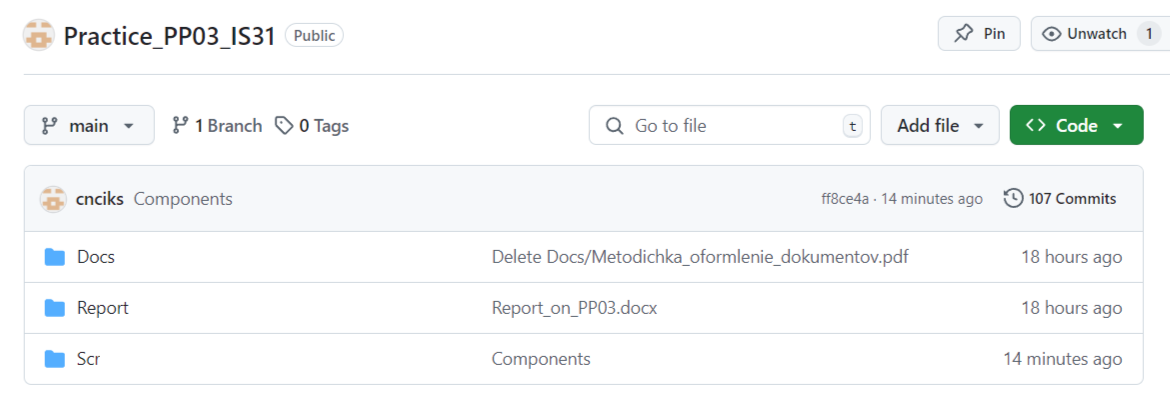


Рисунок 1 – репозиторий «Practice\_PP03\_IS31»

### Разработка модуля обработки и работы с изображениями и модуля взаимодействия с пользователем, формирования и хранения данных: Изменение размера изображения.

1. Код

import tkinter as tk

from tkinter import filedialog, messagebox, simpledialog

from PIL import Image, ImageTk

class ImageUtilityApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Утилита работы с изображениями, практикант: Шевель Милена Александровна")

# Переменные для хранения изображения

self.image = None

self.image\_path = None

# Кнопки

self.load\_button = tk.Button(root, text="Загрузить изображение", command=self.load\_image)

self.load\_button.grid(row=0, column=0, padx=10, pady=10)

self.resize\_button = tk.Button(root, text="Изменить размер", command=self.resize\_image)

self.resize\_button.grid(row=1, column=0, padx=10, pady=10)

# Блок информации

self.info\_label = tk.Label(root, text="Информация об изображении:")

self.info\_label.grid(row=2, column=0, padx=10, pady=10)

self.info\_text = tk.Text(root, width=40, height=10)

self.info\_text.grid(row=3, column=0, padx=20, pady=20)

# Область для отображения изображения

self.image\_label = tk.Label(root)

self.image\_label.grid(row=0, column=1, rowspan=4)

def load\_image(self):

self.image\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg;\*.jpeg;\*.png;\*.gif")])

if not self.image\_path:

return

self.image = Image.open(self.image\_path)

self.display\_image(self.image)

self.update\_info()

def resize\_image(self):

if self.image is None:

messagebox.showwarning("Warning", "Сначала загрузите изображение")

return

# Выбор новых размеров

new\_width = simpledialog.askinteger("Изменить размер", "Новая ширина:")

new\_height = simpledialog.askinteger("Изменить размер", "Новая высота:")

if new\_width is not None and new\_height is not None:

resized\_image = self.image.resize((new\_width, new\_height), Image.LANCZOS)

self.display\_image(resized\_image)

def display\_image(self, img):

img.thumbnail((400, 400)) # Уменьшается изображение, если оно слишком большое

self.tk\_image = ImageTk.PhotoImage(img)

self.image\_label.config(image=self.tk\_image)

self.image\_label.image = self.tk\_image

def update\_info(self):

self.info\_text.delete(1.0, tk.END)

if self.image:

width, height = self.image.size

self.info\_text.insert(tk.END, f"Файл: {self.image\_path}\\n. ")

self.info\_text.insert(tk.END, f"Размер: {width}x{height}")

else:

self.info\_text.insert(tk.END, "Изображение не загружено")

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

root = tk.Tk()

app = ImageUtilityApp(root)

root.mainloop()

1. Модули:

* AppModul.py - модуль содержит классы приложения и связывает все компоненты.

import tkinter as tk

from tkinter import filedialog, messagebox, simpledialog

from ui import UI

from image\_processor import ImageProcessor

class ImageUtilityApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Утилита работы с изображениями, практикант: Шевель Милена Александровна")

# Инициализация пользовательского интерфейса

self.ui = UI(root)

self.image\_processor = ImageProcessor()

# Связывание кнопок с действиями

self.ui.load\_button.config(command=self.load\_image)

self.ui.resize\_button.config(command=self.resize\_image)

def load\_image(self):

self.image\_processor.load\_image(filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg;\*.jpeg;\*.png;\*.gif")]))

if not self.image\_processor.image\_path:

return

self.ui.display\_image(self.image\_processor.image)

self.ui.update\_info(self.image\_processor.image\_path, self.image\_processor.image.size)

def resize\_image(self):

if self.image\_processor.image is None:

messagebox.showwarning("Warning", "Сначала загрузите изображение")

return

new\_width = simpledialog.askinteger("Изменить размер", "Новая ширина:")

new\_height = simpledialog.askinteger("Изменить размер", "Новая высота:")

if new\_width is not None and new\_height is not None:

resized\_image = self.image\_processor.resize\_image(new\_width, new\_height)

self.ui.display\_image(resized\_image)

* ImageModul.py - модуль содержит логику обработки изображений.

from PIL import Image

class ImageProcessor:

def \_\_init\_\_(self):

self.image = None

self.image\_path = None

def load\_image(self, path):

if path:

self.image\_path = path

self.image = Image.open(self.image\_path)

def resize\_image(self, new\_width, new\_height):

if self.image is not None:

resized\_image = self.image.resize((new\_width, new\_height), Image.LANCZOS)

return resized\_image

return None

* MainModul.py - основной файл, который запускает приложение.

from tkinter import Tk

from app import ImageUtilityApp

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

root = Tk()

app = ImageUtilityApp(root)

root.mainloop()

* UiModul.py - в этом модуле описан пользовательский интерфейс.

import tkinter as tk

from PIL import ImageTk

class UI:

def \_\_init\_\_(self, root):

# Кнопки

self.load\_button = tk.Button(root, text="Загрузить изображение")

self.load\_button.grid(row=0, column=0, padx=10, pady=10)

self.resize\_button = tk.Button(root, text="Изменить размер")

self.resize\_button.grid(row=1, column=0, padx=10, pady=10)

# Блок информации

self.info\_label = tk.Label(root, text="Информация об изображении:")

self.info\_label.grid(row=2, column=0, padx=10, pady=10)

self.info\_text = tk.Text(root, width=40, height=10)

self.info\_text.grid(row=3, column=0, padx=20, pady=20)

# Область для отображения изображения

self.image\_label = tk.Label(root)

self.image\_label.grid(row=0, column=1, rowspan=4)

def display\_image(self, img):

img.thumbnail((400, 400))

self.tk\_image = ImageTk.PhotoImage(img)

self.image\_label.config(image=self.tk\_image)

self.image\_label.image = self.tk\_image

def update\_info(self, image\_path, size):

self.info\_text.delete(1.0, tk.END)

if image\_path:

width, height = size

self.info\_text.insert(tk.END, f"Файл: {image\_path}\\n")

self.info\_text.insert(tk.END, f"Размер: {width}x{height}")

else:

self.info\_text.insert(tk.END, "Изображение не загружено")

### Разработка модуля обработки и работы с изображениями и модуля взаимодействия с пользователем, формирования и хранения данных: Склеивание изображений в одно изображение.

1. Код

import tkinter as tk

from tkinter import filedialog, messagebox

from PIL import Image, ImageTk

def load\_image1():

path = filedialog.askopenfilename(title="Выберите первое изображение", filetypes=[("Image files", "\*.jpg \*.jpeg \*.png \*.gif")])

if path:

entry\_image1.delete(0, tk.END)

entry\_image1.insert(0, path)

update\_image\_info(1, path)

def load\_image2():

path = filedialog.askopenfilename(title="Выберите второе изображение", filetypes=[("Image files", "\*.jpg \*.jpeg \*.png \*.gif")])

if path:

entry\_image2.delete(0, tk.END)

entry\_image2.insert(0, path)

update\_image\_info(2, path)

def update\_image\_info(image\_number, path):

try:

img = Image.open(path)

info = f"Изображение {image\_number}: {path}\\nРазмер: {img.size[0]}x{img.size[1]}"

if image\_number == 1:

label\_image\_info1.config(text=info)

elif image\_number == 2:

label\_image\_info2.config(text=info)

except Exception as e:

messagebox.showerror("Ошибка", str(e))

def resize\_and\_concatenate():

try:

path1 = entry\_image1.get()

path2 = entry\_image2.get()

width = int(entry\_width.get())

height = int(entry\_height.get())

# Открываем изображения и изменяем их размеры

img1 = Image.open(path1).resize((width, height))

img2 = Image.open(path2).resize((width, height))

# Склеиваем изображения

result\_img = Image.new('RGB', (width \* 2, height))

result\_img.paste(img1, (0, 0))

result\_img.paste(img2, (width, 0))

# Сохраняем результат

output\_path = path1.rsplit('/', 1)[0] + '/concatenated\_image.jpg'

result\_img.save(output\_path)

messagebox.showinfo("Успех", f"Изображения успешно склеены и сохранены по адресу: {output\_path}")

# Отображение склеенного изображения

display\_concatenated\_image(result\_img)

except Exception as e:

messagebox.showerror("Ошибка", str(e))

def display\_concatenated\_image(result\_img):

# Отображение обработанного изображения

img\_display = ImageTk.PhotoImage(result\_img)

label\_concatenated\_image.config(image=img\_display)

label\_concatenated\_image.image = img\_display

# Создание главного окна

root = tk.Tk()

root.title("Склеивание изображений")

# Создание фреймов для организации макета

frame\_left = tk.Frame(root)

frame\_left.pack(side=tk.LEFT, padx=10, pady=10)

frame\_right = tk.Frame(root)

frame\_right.pack(side=tk.RIGHT, padx=10, pady=10)

# Поля ввода для путей к изображениям

tk.Label(frame\_left, text="Первое изображение:").pack(anchor='w')

entry\_image1 = tk.Entry(frame\_left, width=50)

entry\_image1.pack()

tk.Button(frame\_left, text="Загрузить", command=load\_image1).pack()

tk.Label(frame\_left, text="Второе изображение:").pack(anchor='w')

entry\_image2 = tk.Entry(frame\_left, width=50)

entry\_image2.pack()

tk.Button(frame\_left, text="Загрузить", command=load\_image2).pack()

# Кнопка для выполнения склеивания

tk.Button(frame\_left, text="Склеить изображения", command=resize\_and\_concatenate).pack(pady=10)

# Поля ввода для размеров изображений

tk.Label(frame\_left, text="Ширина:").pack(anchor='w')

entry\_width = tk.Entry(frame\_left)

entry\_width.pack()

entry\_width.insert(0, "300") # Значение по умолчанию

tk.Label(frame\_left, text="Высота:").pack(anchor='w')

entry\_height = tk.Entry(frame\_left)

entry\_height.pack()

entry\_height.insert(0, "300") # Значение по умолчанию

# Метки для отображения информации об изображениях

label\_image\_info1 = tk.Label(frame\_left, text="")

label\_image\_info1.pack(anchor='w')

label\_image\_info2 = tk.Label(frame\_left, text="")

label\_image\_info2.pack(anchor='w')

# Метка для отображения склеенного изображения

label\_concatenated\_image = tk.Label(frame\_right)

label\_concatenated\_image.pack()

# Запуск главного цикла

root.mainloop()

1. Модули:

* ImageModul.py – модуль для обработки изображений (загрузка, изменение размера, склеивание).

from tkinter import filedialog, messagebox

from PIL import Image

class ImageHandler:

def \_\_init\_\_(self):

self.path1 = ""

self.path2 = ""

def load\_image(self, image\_number):

path = filedialog.askopenfilename(

title="Выберите изображение",

filetypes=[("Image files", "\*.jpg \*.jpeg \*.png \*.gif")]

)

if path:

if image\_number == 1:

self.path1 = path

elif image\_number == 2:

self.path2 = path

return path

return None

def resize\_and\_concatenate(self, width, height):

try:

img1 = Image.open(self.path1).resize((width, height))

img2 = Image.open(self.path2).resize((width, height))

result\_img = Image.new('RGB', (width \* 2, height))

result\_img.paste(img1, (0, 0))

result\_img.paste(img2, (width, 0))

output\_path = self.path1.rsplit('/', 1)[0] + '/concatenated\_image.jpg'

result\_img.save(output\_path)

return output\_path, result\_img

except Exception as e:

messagebox.showerror("Ошибка", str(e))

return None, None

* MainModul.py – основной файл, который инициализирует приложение.

import tkinter as tk

from image\_handler import ImageHandler

from ui import create\_ui

def main():

root = tk.Tk()

root.title("Склеивание изображений")

image\_handler = ImageHandler()

create\_ui(root, image\_handler)

root.mainloop()

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

main()

* UiModul.py – модуль для создания пользовательского интерфейса.

import tkinter as tk

from tkinter import messagebox

def create\_ui(root, image\_handler):

frame\_left = tk.Frame(root)

frame\_left.pack(side=tk.LEFT, padx=10, pady=10)

frame\_right = tk.Frame(root)

frame\_right.pack(side=tk.RIGHT, padx=10, pady=10)

tk.Label(frame\_left, text="Первое изображение:").pack(anchor='w')

entry\_image1 = tk.Entry(frame\_left, width=50)

entry\_image1.pack()

tk.Button(frame\_left, text="Загрузить",

command=lambda: load\_image(1, entry\_image1, image\_handler)).pack()

tk.Label(frame\_left, text="Второе изображение:").pack(anchor='w')

entry\_image2 = tk.Entry(frame\_left, width=50)

entry\_image2.pack()

tk.Button(frame\_left, text="Загрузить",

command=lambda: load\_image(2, entry\_image2, image\_handler)).pack()

tk.Button(frame\_left, text="Склеить изображения",

command=lambda: resize\_and\_concatenate(image\_handler)).pack(pady=10)

tk.Label(frame\_left, text="Ширина:").pack(anchor='w')

entry\_width = tk.Entry(frame\_left)

entry\_width.pack()

entry\_width.insert(0, "300") # Значение по умолчанию

tk.Label(frame\_left, text="Высота:").pack(anchor='w')

entry\_height = tk.Entry(frame\_left)

entry\_height.pack()

entry\_height.insert(0, "300") # Значение по умолчанию

label\_concatenated\_image = tk.Label(frame\_right)

label\_concatenated\_image.pack()

def load\_image(image\_number, entry\_widget, image\_handler):

path = image\_handler.load\_image(image\_number)

if path:

entry\_widget.delete(0, tk.END)

entry\_widget.insert(0, path)

def resize\_and\_concatenate(image\_handler):

width = int(entry\_width.get())

height = int(entry\_height.get())

output\_path, result\_img = image\_handler.resize\_and\_concatenate(width, height)

if output\_path:

messagebox.showinfo("Успех", f"Изображения успешно склеены и сохранены по адресу: {output\_path}")

display\_concatenated\_image(result\_img, label\_concatenated\_image)

def display\_concatenated\_image(result\_img, label\_widget):

img\_display = ImageTk.PhotoImage(result\_img)

label\_widget.config(image=img\_display)

label\_widget.image = img\_display

## Задания второй недели

### Проведение обратного проектирования, используя графический язык UML . Создание и описание диаграммы Компонентов, Сценариев использования, Последовательностей, Деятельности. Изменение размера изображения.



Рисунок 2 – диаграмма компонентов

* Пользовательский интерфейс взаимодействует с обработчиком изображений, чтобы отправлять выбранные пользователем параметры.
* Компонент загрузки изображений передает загруженное изображение в обработчик изображений.

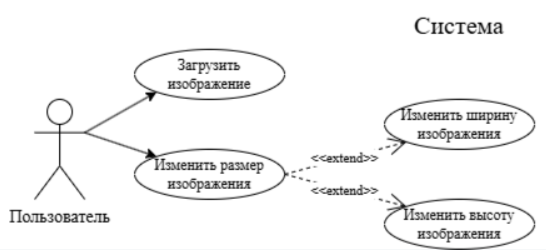


Рисунок 3 – диаграмма сценариев использования

* Загрузить изображение - Пользователь загружает изображение в систему.
* Выбрать размер - Пользователь выбирает новый размер для изображения (ширину и высоту).
* Пользователь взаимодействует со всеми вышеуказанными сценариями использования.

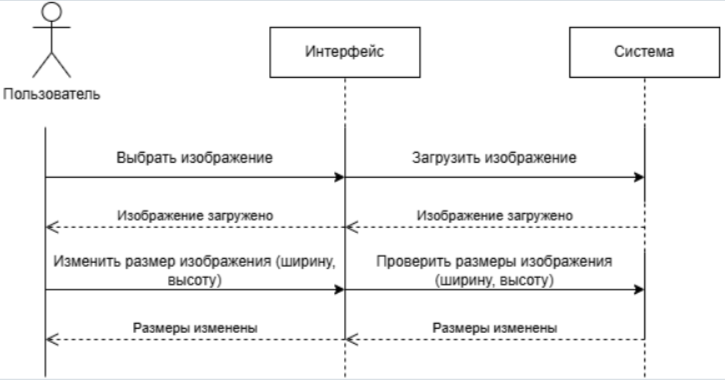


Рисунок 4 – диаграмма последовательностей

* Пользователь выбирает изображение.
* Система загружает выбранное изображение.
* Пользователь вводит новые размеры (ширина и высота).
* Система проверяет допустимость новых размеров.
* Система применяет изменения к изображени.
* Система уведомляет пользователя об успешном изменении размера.

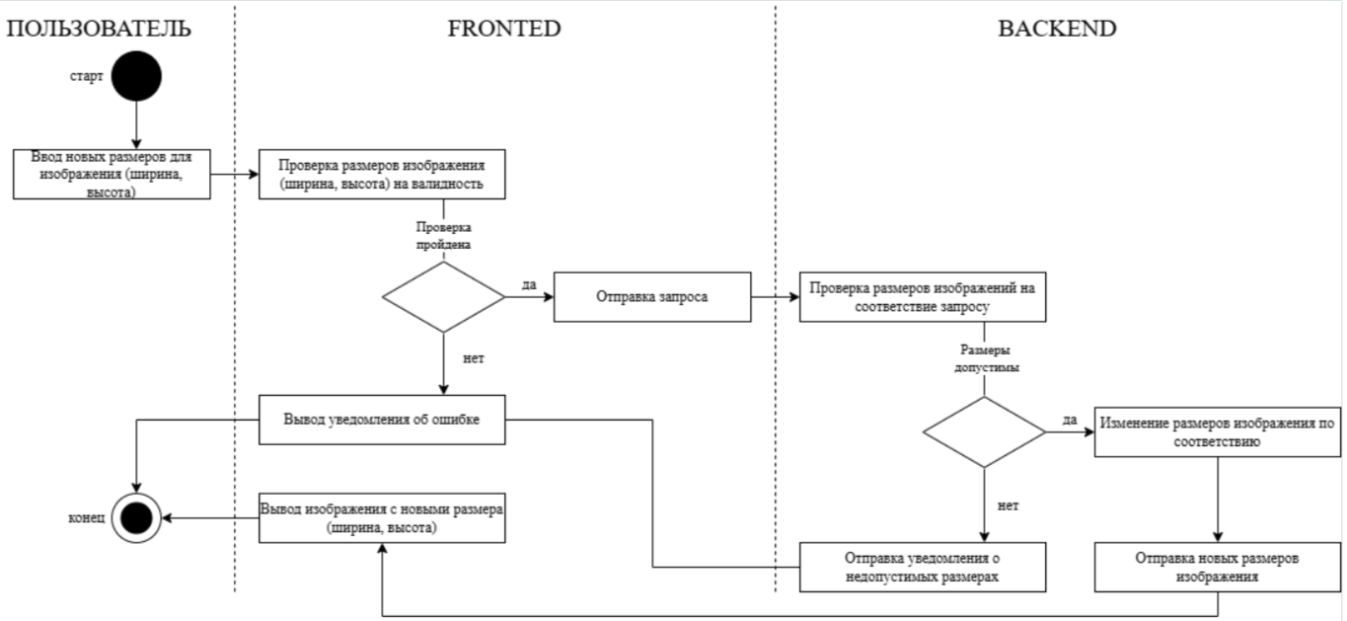


Рисунок 5 – диаграмма деятельности

* Пользователь вводит новые размеры для изображения (ширину и высоту).
* Fronted проверяет размеры изображения (ширину и высоту) на валидность.
* Если проверка не удалась:
* Fronted выводит уведомление об ошибке.
* Если проверка удалась:
* Fronted отправляет запрос Backend.
* Backend проверяет размеры изображения на соответствие запросу.
* Если размеры не допустимы:
* Backend отправляет уведомление о недопустимых размерах.
* Fronted выводит уведомление об ошибке.
* Если размеры допустимы:
* Backend изменяет размеры изображения по соответствию.
* Backend отправлет новые размеры изображения.
* Fronted выводит изображение с новыми размерами (шириной и высотой).