НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАНОЇ ТЕХНІКИ
КАФЕДРА ОБЧИСЛЮВАНОЇ ТЕХНІКИ

КУРСОВА РОБОТА

з дисципліни «Інженерія програмного забезпечення»
на тему: «Растровий графічний редактор»

Студентки II курсу групи ІО-64
напряму підготовки: 123 – Комп'ютерна інженерія
Бровченко Анастасії Вікторівни НЗК 6403

Керівник:старший викладач, к.т.н., с.н.с.
Антонюк Андрій Іванович

Національна оцінка: _____
Кількість балів: _____

Науковий керівник _____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Члени комісії _____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2018 рік

ЗМІСТ

ЗМІСТ 2
ВСТУП 3
1. ОГЛЯД MVC 4
1.1. Загальні положення MVC 4
1.2. Структура MVC 4
1.3. Шаблони програмування, що використовуються в MVC 5
1.4. Завдання MVC 5
1.5. Python 5
1.6. Використання концепції MVC в даній роботі 5
2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ 6
2.1. Прецеденти 6
2.2. Діаграма граничних класів 8
3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ 9
3.1. Діаграма класів 9
4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 11
ВИСНОВКИ 13
ЛІТЕРАТУРА 14
ДОДАТКИ 15
ДОДАТОК 1: ПРОГРАМНИЙ КОД ПРОЕКТУ 15

ВСТУП

Об’єктом розробки даної курсової роботи є растровий редактор зображень з можливістю збереження відредагованого зображення в форматі png.

Метою курсової роботи є закріплення теоретичних знань і практичних навичок з проектування, моделювання , розробки та тестування програмного забезпечення з графічним інтерфейсом.

Результат курсової роботи – готовий програмний додаток з графічним інтерфейсом, написаний на мові програмування Python.

Графічний інтерфейс користувача – інтерфейс між комп’ютером і його користувачем, що використовує вікна, меню, вказівний засіб для виконання команд, реалізований за допомогою бібліотеки tkinter.

1. ОГЛЯД MVC

1.1. Загальні положення MVC

Model-view-controller, (MVC) (Модель–вигляд–контролер) – архітектурний шаблон, що використовується у проектуванні та розробці програмного забезпечення.

Даний шаблон передбачає розділення системи на три частини: модель даних, інтерфейс користувача та модуль керування, так щоб модифікація одного з компонентів мінімально впливала на інші.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

1.2. Структура MVC

Програма поділяється на три окремі і взаємопов’язані частини з розподілом функцій між компонентами:

Модель (Model) відповідає за зберігання даних і забезпечення інтерфейсу до них.

- Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.

Вигляд (View) відповідальний за представлення цих даних користувачеві.

- Вигляд може являти собою будь-яке представлення інформації, одержуване на виході. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації.

Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача і передає дані у модель.

- Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Модель не залежить від процесу вводу чи виводу даних, цей компонент інкапсулює ядро даних і основний функціонал їхньої обробки. Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані. У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас.

1.3. Шаблони програмування, що використовуються в MVC

Для реалізації схеми «Model-View-Controller» використовується досить велика кількість шаблонів проектування, основні з яких - «спостерігач», «стратегія», «компонувальник».

Найбільш типова реалізація - в якій уявлення відокремлено від моделі шляхом встановлення між ними протоколу взаємодії, що використовує «апарат подій» (підписка/сповіщення) : при кожній особливій зміні внутрішніх даних в моделі (позначеному як «подія»), вона сповіщає про нього всі залежні представлення і вони оновлюються. Так використовується шаблон «спостерігач».

При обробці реакції користувача представлення вибирає, в залежності від реакції, потрібний контролер, який забезпечує той чи інший зв'язок з моделлю. Для цього використовується шаблон «стратегія», або модифікація з використанням шаблону «команда».

Для можливості однотипного поводження з підоб'єктами складно складеного ієрархічного виду - використовується шаблон «компонувальник».

1.4. Завдання MVC

Основною метою застосування концепції MVC є відділення бізнес-логіки (Model) від її візуалізації (View). Через такий поділ виникає можливість повторно використовувати код програми. Найбільш корисним застосуванням концепції є випадок, коли користувач повинен одночасно бачити в різних контекстах ті ж самі дані. Виконуються наступні завдання:

- До однієї моделі можна приєднати кілька видів, при цьому не зачіпаючи реалізацію моделі;
- Не торкаючись реалізації видів, можна змінити реакції на дії користувача, для цього досить використовувати інший контролер;
- Ряд розробників розробляють графічний інтерфейс, або розробляють бізнес-логіку. Можливо досягти того, що програмісти, які займаються розробкою бізнес-логіки, не будуть обізнані про те, який вигляд буде використовуватися.

1.5. Python

Дану курсову роботу буде виконано, використовуючи мову програмування Python, з використанням tkiner - багато-платформної графічної бібліотеки інтерфейсів на основі засобів Tk.

1.6. Використання концепції MVC в даній роботі

В даній роботі буде використовуватися модифікована версія шаблону MVC, яка об’єднує представлення і контролер в один логічний об’єкт. MVC розділяє систему на дві частини: модель даних (база даних), вигляд даних (графічний інтерфейс користувача) об’єднаний з

керуванням (логікою). Це відокремить модель даних від інтерфейсу користувача, щоб зміни в будь-якій з цих частин системи мінімально впливали на іншу частину системи.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

2.1. Прецеденти

Можливості користувача додатку можна побачити на Рис. 2.1.

Користувач може вибрати колір кисті, змінити її розмір, редагувати зображення, зберегти відредаговане зображення в обрану папку комп'ютера.

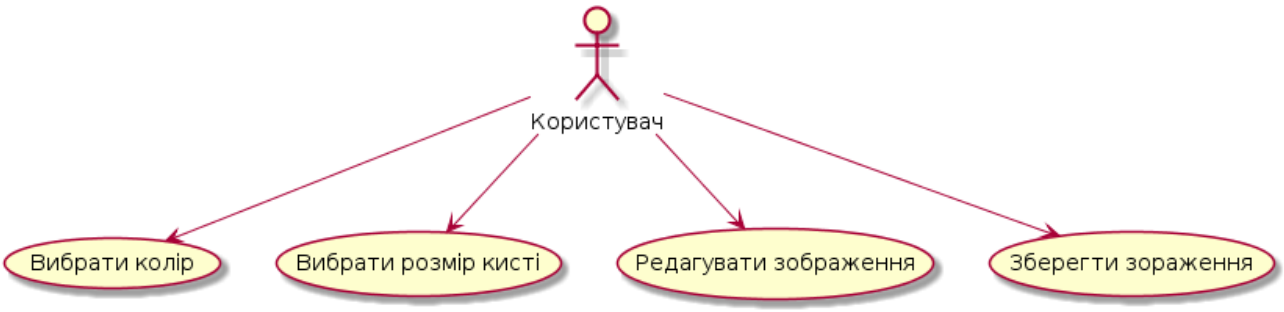

Рис. 2.1. Можливості користувача

Користувач може змінити колір кисті, обравши колір з палітри (Рис. 2.2).

Рис. 2.2. Операція вибору кольору кисті

Користувач може змінити розмір кисті у відповідному полі (Рис. 2.3).

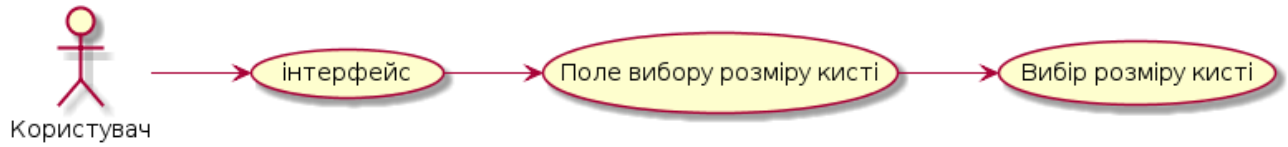

Рис. 2.2. Операція зміни розміру кисті

Користувач може очистити повністю поле, натиснувши кнопку “Clear all” (Рис. 2.4).

Рис. 2.4. Операція очищення поля

Користувач може зберегти відредаговане зображення, натиснувши кнопку “Save image”, обравши папку для збереження файлу та ввівши назву файлу. (Рис. 2.5)

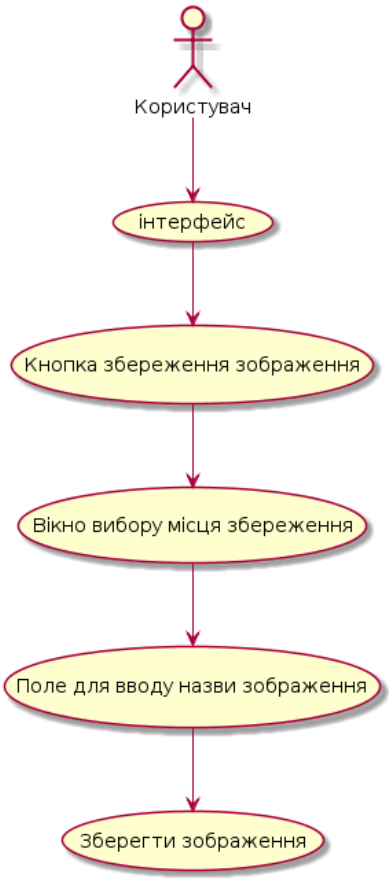

Рис. 2.5. Операція збереження відредагованого зображення

2.2. Діаграма граничних класів

На Рис.2.17 зображено діаграму граничних класів для бази даних, тобто функціоналу адміністратора

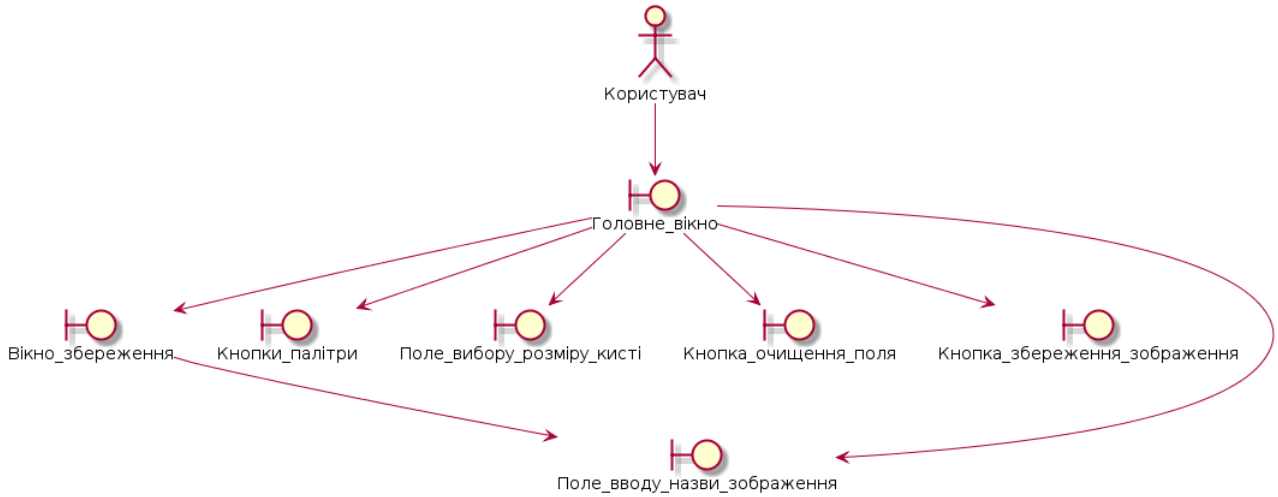

Рисунок 2.17. – Діаграма граничних класів

1.1 Таблиця відповідності елементів бібліотеці Tkinter

Елемент інтерфейсу	Клас, реалізуючий елемент
Головне вікно	tkinter.Tk
Кнопки палітри	tkinter.ttk.Button
Кнопка очищення поля	tkinter.ttk.Button
Кнопка збереження зображення	tkinter.ttk.Button
Кнопки вибору курсу	tkinter.ttk.Button
Поля «Color», «Brush size»	tkinter.Label
Поле вибору розміру	tkinter.Spinbox
Вікно збереження зображення	tkinter.filedialog.asksaveasfilename

3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1. Діаграма класів

На рисунку 3.1 зображено діаграму класів програми. На ній видно класи, що реалізують моделі (models) об’єктів бази даних, та їхні поля, які забезпечують її коректну роботу.

Опис класів

Класи містить такі поля:

- brush_size – розмір кисті;
- color – колір;
- canv – поле редагування.

Клас PaintModel – клас, що описує модель графічного редактора.

Містить такі методи:

- set_color – функція для зміни кольору кисті;
- set_brush_size – функція для зміни розміру кисті;
- draw_point – функція для малювання при нерухомому курсорі;
- paint – функція для малювання при рухомому курсорі;
- reset – функція для очищення робочого поля;
- save_image – функція для збереження відредагованого зображення.

Клас PaintViewController – клас, що описує вигляд і роботу графічного редактора.

Містить такі методи:

- create_color_btn – створює кнопку для вибору кольору кисті;
- setUI – виклиг головного вікна програми.

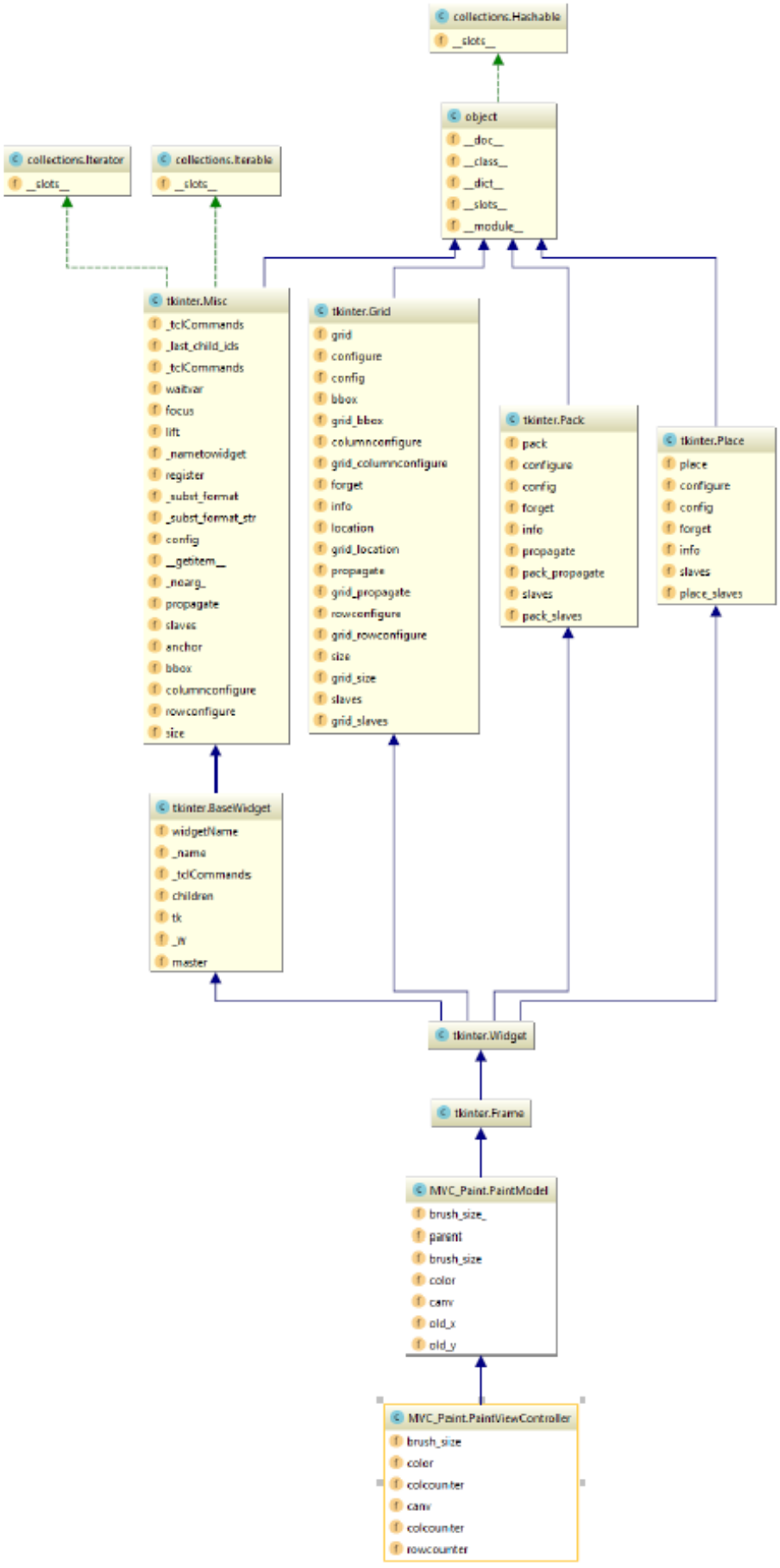

Рисунок 3.1. – Діаграма класів

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Запустивши програму, перед користувачем відкривається головне вікно програми (Рис. 4.1).

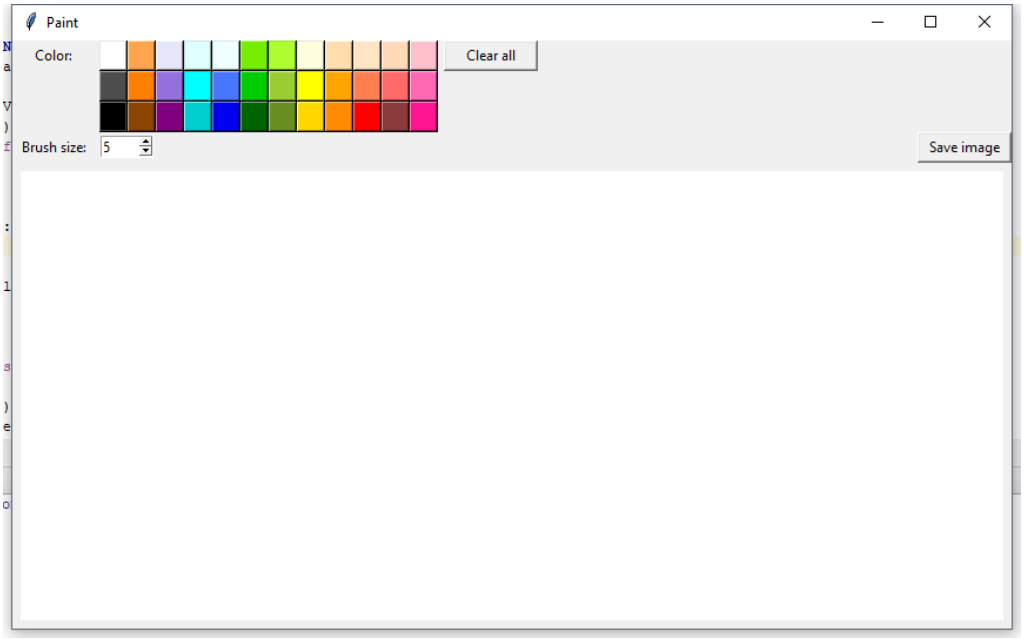

Рис. 4.1. Головне вікно програми

При зміні розміру вікна об’єкти у вікні залишаються на своїх місцях (Рис. 4.2).

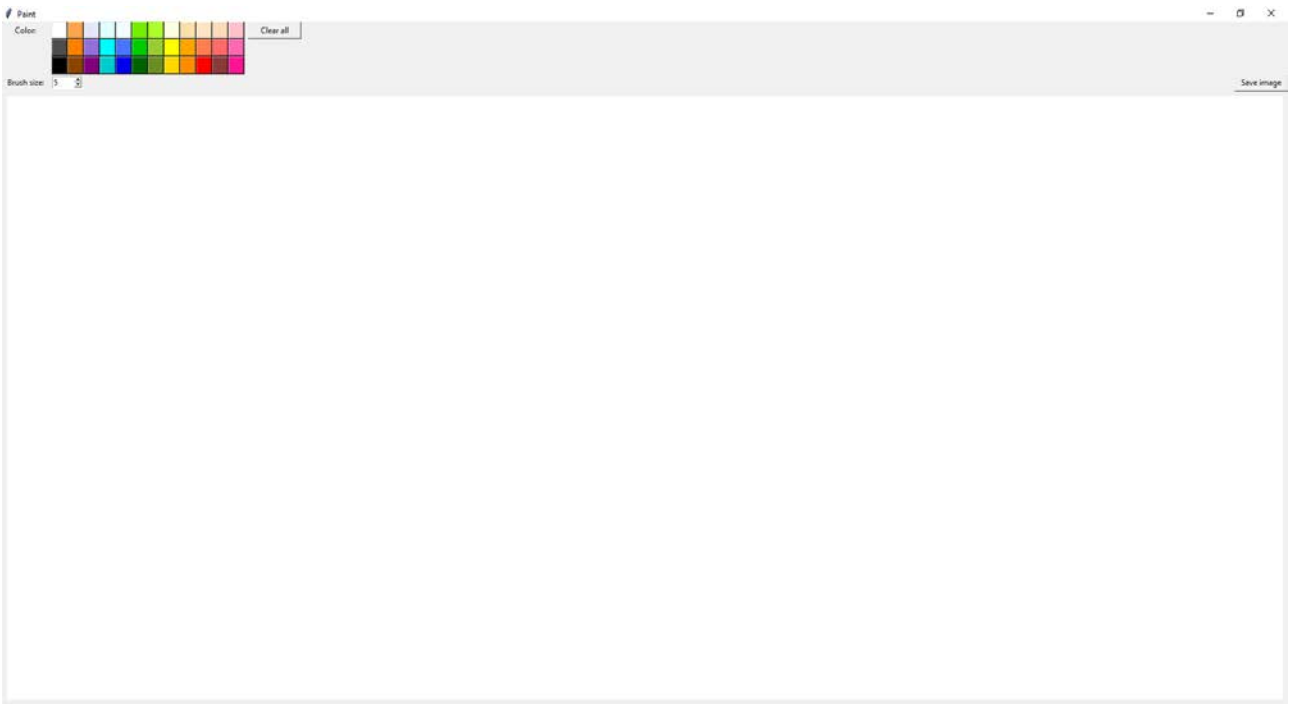

Рис. 4.2. Зміна розміру головного вікна програми

Користувач може обрати потрібний колір з палітри та розмір кисті для малювання (Рис. 4.3).

Рис. 4.3. Вибір кольору, розміру кисті, редагування зображення

За необхідності, користувач може зберегти зображення в будь-якій папці (Рис. 4.4).

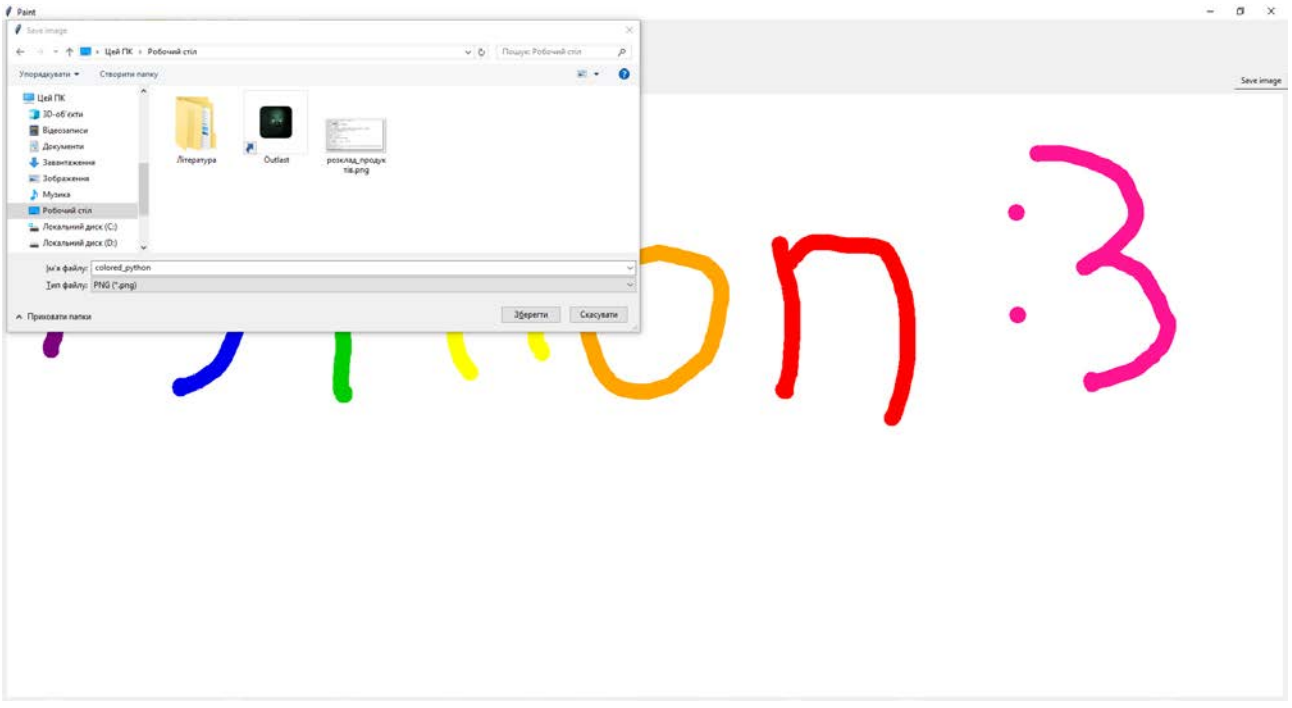

Рис. 4.4. Збереження зображення

Зображення зберігається у форматі PNG (Рис. 4.5).

Рис. 4.5 Збережене зображення

ВИСНОВКИ

У ході виконання даної курсової роботи:

Були закріплені теоретичні знання з проектування, моделювання , розробки та тестування програмного забезпечення з графічним інтерфейсом, було вивчено шаблон програмування MVC, покращено практичні навички роботи з мовою програмування Python та Django – Python-фреймворку для розробки веб-систем.

Розроблено механізм створення бази даних, збереження та обробки об’єктів бази даних для сайту інтернет-магазину ноутбуків.

Розроблено графічний інтерфейс, який задовольняє усім вимогам, поставленим під час проектування.

Метою було розробити додаток який дозволяє адміністратору інтернет-магазину вносити зміни до каталогу товарів тобто додавати, видаляти товари та редагувати інформацію про них, а користувачу вільно переміщатися по сайту та інтуїтивно просто робити замовлення.

https://github.com/W-h-o-v-i-a-n/Paint_project - посилання на проект.

ЛІТЕРАТУРА

1. Марк Лутц Программирование на Python. Том 1, 4-е издание – «Символ-Плюс», 2011 – 992 с. — ISBN 978-1-449-35573-9.

2. Бандура, В. В. Архітектура та проектування програмного забезпечення конспект лекцій / В. В. Бандура, Р. І. Храбатин. - Івано-Франківськ: ІФНТУНГ, 2012. - 240 с.

3. http://uk.wikipedia.org/wiki/Шаблони проектування програмного забезпечення.

ДОДАТКИ

ДОДАТОК 1: ПРОГРАМНИЙ КОД ПРОЕКТУ

```
from tkinter import *
from tkinter import filedialog
from PIL import ImageGrab

default_color = "black"
default_brush_size = 2

colors = ['white', 'tan1', 'lavender', 'lightcyan', 'azure', 'chartreuse2', 'greenyellow', 'lightyellow', 'navajowhite',
'bisque', 'peachpuff', 'pink',
        'gray30', 'darkorange1', 'mediumpurple', 'cyan', 'royalblue1', 'green3', 'yellowgreen', 'yellow', 'orange', 'coral',
'indianred1', 'hotpink',
        'black', 'darkorange4', 'purple', 'cyan3', 'blue2', 'darkgreen', 'olivedrab', 'gold', 'darkorange', 'red', 'indianred4',
'deeppink']


class PaintModel(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.parent = parent
        self._brush_size = IntVar()
        self._brush_size.set(5)
        self.canv = Canvas(self, bg="white")
        self.old_x: int = None
        self.old_y: int = None

    def setCursor(self, place):
        pass

    def set_color(self, new_color):
        self.color = new_color

    def set_brush_size(self):
        self.brush_size = int(self._brush_size.get())

    def draw_point(self, event):
        self.canv.create_oval(event.x - self.brush_size / 2,
                              event.y - self.brush_size / 2,
                              event.x + self.brush_size / 2,
                              event.y + self.brush_size / 2,
                              fill=self.color, outline=self.color)

    def paint(self, event):
        if self.old_x and self.old_y:
            self.canv.create_line(self.old_x, self.old_y, event.x, event.y,
                                  width=self.brush_size, fill=self.color,
                                  capstyle=ROUND, smooth=TRUE, splinesteps=36)
        self.old_x = event.x
        self.old_y = event.y

    def reset(self, event):
        self.old_x, self.old_y = None, None
```

```
    def save_image(self):
        x = self.winfo_rootx() + self.canv.winfo_x()
        y = self.winfo_rooty() + self.canv.winfo_y()
        x1 = x + self.canv.winfo_width()
        y1 = y + self.canv.winfo_height()

        try:
            self.filename = filedialog.asksaveasfilename(initialdir="/", title="Save image",
                                        filetypes=(("PNG", "*.png"), ("all files", "*.*")))
            ImageGrab.grab().crop((x, y, x1, y1)).save(rf"{self.filename}.png")

        except: pass


class PaintViewController(PaintModel):
    def __init__(self, parent=None):
        super().__init__(parent)
        Frame.__init__(self, parent)
        self.canv = Canvas(self, bg="white")
        self.color = default_color
        self.brush_size = default_brush_size
        self.setUI()

    colcounter = 0
    rowcounter = 0

    def create_color_btn(self, color):
        _c = 1  # column
        _r = 0  # row
        Button(self, width=2, bg=color, command=lambda: self.set_color(color)). \
            grid(column=_c + self.colcounter, row=_r + self.rowcounter)
        self.colcounter += 1
        if self.colcounter == 12:
            self.colcounter = 0
            self.rowcounter += 1

    def setUI(self):
        self.parent.title("Paint")
        self.pack(fill=BOTH, expand=1)  # Размещаем активные элементы на родительском окне

        self.columnconfigure(14, weight=1)  # возможность растягиваться
        self.rowconfigure(4, weight=1)

        # Создаем поле для рисования, устанавливаем белый фон
        self.canv.grid(row=4, column=0, columnspan=15, padx=5, pady=5, sticky=E + W + S + N)
        self.canv.bind("<Button-1>", self.draw_point)
        self.canv.bind("<B1-Motion>", self.paint)
        self.canv.bind('<ButtonRelease-1>', self.reset)

        # Color buttons
        Label(self, text="Color: ").grid(row=0, column=0, padx=6)
        for i in colors:
            self.create_color_btn(i)

        Button(self, text="Clear all", width=10, command=lambda: self.canv.delete("all")) \
            .grid(row=0, column=13, sticky=W, padx=5)

        # Brush size
        Label(self, text="Brush size: ").grid(row=3, column=0, padx=5)
```

```
        spinbox = Spinbox(self, from_=1, to=228, width=5, textvariable=self._brush_size, command=self.set_brush_size)
        spinbox.grid(row=3, column=1, columnspan=2)

        Button(self, text='Save image', width=10, command=self.save_image).grid(row=3, column=14, sticky=E)


if __name__ == '__main__':
    root = Tk()
    root.geometry("850x500+300+300")
    PaintViewController(parent=root)
    root.mainloop()
```