ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«Розробка інтерфейсу для програмного забезпечення підготовки зображення для верстату по виготовленню трикотажних виробів»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Білуха Данило Віталійович

_____ «___» _____ 202_ р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Чілікіна Тетяна Василівна

_____ «___» _____ 202_ р.
(підпис)

Рецензент _____

ПОЛТАВА 2025

ЗАТВЕРДЖУЮ
Завідувач кафедри _____ Олена ОЛЬХОВСЬКА
«____» вересня 202__ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка інтерфейсу для програмного забезпечення підготовки зображення для верстату по виготовленню трикотажних виробів»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра
Прізвище, ім'я, по батькові Білуха Данило Віталійович
Затверджена наказом ректора № ___-Н від «__» _____ 202__ р.
Термін подання студентом роботи «__» _____ 202__ р.
Вихідні дані до кваліфікаційної роботи: задоволення роботи з інтерфейсом користувача.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1 Проблематика обробки зображень у трикотажному виробництві
2.2 Аналіз інтерфейсів у спеціалізованому ПЗ
2.3 Основні принципи ефективного інтерфейсу
2.4 Висновки за результатами огляду
2.5 Висновки по розділу
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1 Аналіз вимог до інтерфейсу професійного програмного забезпечення
3.2 Принципи побудови інтерфейсу: UX/UI-орієнтований підхід
3.3 Вибір технологій для реалізації інтерфейсу
3.4 Порівняльний аналіз створення інтерфейсу в JavaFX та PyQt
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1 Приклад реалізації коду
4.2 Загальна структура програми
4.3 Завантаження та попередній перегляд зображення
4.4 Базове редагування зображення
4.5 Адаптація зображення до вимог верстата
4.6 Збереження результату та експорт
ВИСНОВКИ
СПИСОК ЛІТЕРАТУРИ
ДОДАТОК А
ДОДАТОК Б
Перелік графічного матеріалу: 10 рисунків.

Консультанти розділів кваліфікаційної роботи

Розділ	ПІБ, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Чілікіна Т.В.		
Інформаційний огляд	Чілікіна Т.В.		
Теоретична частина	Чілікіна Т.В.		
Практична реалізація	Чілікіна Т.В.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка завдання		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доопрацювання (за необхідності), рецензування		

Дата видачі завдання «____»_____ 202__ р.
Здобувач вищої освіти Білуха Данило Віталійович
Науковий керівник к. ф.-м. н., Чілікіна Тетяна Василівна

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____»_____ 202__ р.

Секретар ЕК _____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к. ф.-м. н. Олена ОЛЬХОВСЬКА
«____» _____ 202_ р

Погоджено
Науковий керівник _____
к. ф.-м. н _ Чілікіна Т.В._
«____» _____ 2024_ р

План

кваліфікаційної роботи на тему
«Проектування сайту «Розробка інтерфейсу для програмного забезпечення підготовки зображення для верстату по виготовленню трикотажних виробів»
зі спеціальності 122 Комп’ ютерні науки
освітня програма 122 «Комп’ютерні науки»
ступеня бакалавр
Прізвище, ім’я, по батькові Білуха Данило Віталійович

Здобувач вищої освіти _____ Данило БІЛУХА «____» _____ 202_ р.

РЕФЕРАТ

Записка: 56 с., 10 рис., 10 джерел.

ІНТЕРФЕЙС, ОБРОБКА ЗОБРАЖЕНЬ, СПРОЩЕННЯ ПАЛІТРИ, UX/UI-ДИЗАЙН.

Об’єкт дослідження — програмне забезпечення для підготовки зображень до друку на автоматизованих в'язальних верстатах.

Мета роботи — розробка адаптивного, функціонального та інтуїтивно зрозумілого інтерфейсу для професійного програмного забезпечення, що дозволяє дизайнерам трикотажної продукції швидко та ефективно редагувати і адаптувати зображення під технічні вимоги виробничого обладнання.

Методи дослідження — аналіз UX/UI-принципів побудови інтерфейсів, технології Java, JavaFX, FXML; методи редагування зображень (зміна яскравості, контрасту, кольорової палітри), адаптація графіки до технічних обмежень верстатів.

У роботі проведено аналіз проблематики підготовки зображень у сфері трикотажного виробництва, виявлено недоліки існуючих рішень та сформульовано вимоги до інтерфейсу, орієнтованого на дизайнерів. Запропоновано концепцію інтерфейсу, який поєднує функціональність професійного інструмента з простотою візуального редактора.

Розроблено інтерфейс із підтримкою завантаження зображень, їх редагування, автоматичного спрощення палітри, адаптації до шаблонів нитководів, попереднього перегляду результату та збереження файлів. Реалізовано підтримку адаптивного дизайну, світлої і темної тем, контекстної допомоги та можливості збереження параметрів.

У результаті створено прототип програмного забезпечення, який дозволяє значно скоротити час підготовки зображень, зменшити кількість помилок та підвищити ефективність дизайнерів. Проєкт має практичну цінність для підприємств, що працюють у галузі текстильного виробництва, та потенціал для подальшої інтеграції із ШІ і розширення функціоналу.

ЗМІСТ

Вступ

Актуальність. У сучасній промисловості, зокрема у виробництві трикотажних виробів, спостерігається стрімке зростання вимог до якості, швидкості та індивідуалізації дизайну продукції. Одним із ключових факторів підвищення ефективності цього процесу є розробка та впровадження спеціалізованого ПЗ, що полегшує роботу дизайнерів і забезпечує точну підготовку зображень для подальшої обробки на виробничих верстатах.

Значна частина помилок та затримок у виробництві виникає саме на етапі підготовки зображень, які мають відповідати технічним обмеженням верстатів — формат зображення, відповідність нитководам, наявність точної схеми шиття для верстата, тощо. Більшість існуючих рішень або складні у використанні, або не враховують специфіку роботи дизайнерів, які мають справу з великою кількістю шаблонів, стилістичних рішень та вимог до точності візуалізації.

Метою роботи є розробка зручного, інтуїтивно зрозумілого та функціонально насиченого інтерфейсу професійного програмного забезпечення, що дозволяє дизайнерам трикотажних виробів ефективно готувати, редагувати та адаптувати зображення під вимоги виробничого обладнання. Особливу увагу приділено гнучкості інтерфейсу, адаптивності до різних розмірів екранів, а також підтримці базових і розширених операцій із графічними матеріалами — таких як обрізання, зміна кольорової палітри, корекція яскравості, контрасту та зменшення кількості кольорів до заданого значення.

Об’єктом дослідження є програмне забезпечення для підготовки зображень до друку на автоматизованих в'язальних верстатах.

Предметом дослідження є створення користувацького інтерфейса та його програмна реалізація для професійного застосування у сфері дизайну трикотажної продукції.

У межах проекту було застосовано технології Java, JavaFX та FXML для побудови адаптивного інтерфейсу, враховано принципи UX/UI-дизайну, а також реалізовано функції, які значно скорочують час підготовки зображень до виробництва. У результаті розроблено інтерактивний графічний інтерфейс, здатний підтримувати роботу з великими обсягами графічних даних, адаптуватися до індивідуальних потреб користувача та забезпечувати високий рівень точності обробки.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

У процесі розробки трикотажної продукції дизайнери зіштовхуються з низкою проблем, пов’язаних із підготовкою графічних матеріалів для подальшого використання на виробничих верстатах. Зокрема, це — складність узгодження кольорової палітри з технічними обмеженнями верстату, необхідність ручного редагування файлів у кількох різних програмах, а також високі вимоги до точності передачі деталей зображення.

На більшості підприємств етап підготовки зображень до друку здійснюється вручну, часто у застарілому або неінтегрованому програмному середовищі, що значно уповільнює виробничий процес, підвищує ризик помилок та знижує ефективність праці дизайнерів. У зв’язку з цим виникає потреба у створенні професійного інтерфейсу, який би об’єднував усі ключові функції обробки в єдиному середовищі, був адаптивним, простим у використанні та гнучким до зміни вимог.

Основні завдання

1.Розробити програмний інтерфейс, який забезпечить зручну, швидку та точну підготовку зображень до обробки верстатом для виготовлення трикотажних виробів.

2.Інтерфейс має підтримувати базові та розширені операції редагування, відображати актуальні обмеження верстату (кількість кольорів, розміри, нитководи), забезпечувати інтуїтивну логіку взаємодії та бути адаптивним до різних екранів і пристроїв.

3. Створенний інтепфейс повинен бути зручним, зрозумілим та доступним у використанні.

Основні функціональні завдання:

1. Завантаження зображень у популярних форматах (BMP, PNG, JPG);
2. Попередній перегляд та обрізка зображення, налаштування розміру відповідно до параметрів шкарпеткових верстатів;
3. Регулювання кольорової палітри (зменшення до заданої кількості кольорів, автоматична кластеризація палітри);
4. Налаштування параметрів зображення — контраст, яскравість, насиченість, температурна корекція;
5. Вибір шаблону нитководів із врахуванням обмежень подачі кольору (наприклад, C1–C5, F1) та автоматичне попередження при перевищенні ліміту;
6. Експорт обробленого зображення у формат, придатний для подальшої роботи на верстаті або для передачі в інше програмне забезпечення;
7. Адаптивний дизайн інтерфейсу з урахуванням UX/UI-принципів — зручна навігація, зрозумілі підказки, повідомлення про помилки тощо;
8. Можливість локалізації інтерфейсу та збереження персональних налаштувань користувача.

Вихідні дані:

- Вхідне зображення у графічному форматі;
- Технічна документація до верстата: максимальна кількість кольорів, допустимі розміри, обмеження на розподіл нитководів;
- Набір дизайнерських вимог щодо стилю, кольору, структури зображень.

Очікувані результати:

- Інтерактивний інтерфейс із набором функцій, що задовольняють потреби дизайнерів трикотажних виробів;

- Можливість швидкої підготовки якісного зображення без перемикання між програмами;
- Зменшення кількості помилок у підготовці матеріалів до друку;
- Покращення продуктивності та ефективності дизайнерів на етапі створення виробів.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Проблематика обробки зображень у трикотажному виробництві

У сфері виробництва трикотажних виробів, зокрема шкарпеток, особливе значення має точна підготовка графічного матеріалу для друку на в’язальних верстатах. Дизайнери, які займаються розробкою візерунків і орнаментів, мають враховувати специфіку текстильного обладнання: обмежену кількість кольорів, сувору прив’язку до нитководів, фіксовані розміри області друку тощо.

Наразі на багатьох підприємствах цей процес залишається переважно ручним. Дизайнери використовують загальні графічні редактори — від Paint до Photoshop — для первинної обробки зображень, після чого переходять до конвертації у формати, сумісні з текстильним ПЗ. Такий багатоступеневий підхід призводить до втрати часу, збільшення ймовірності помилок, необхідності дублювати дії та постійно перемикатися між інструментами.

Особливої актуальності набуває створення єдиного середовища для виконання всіх операцій — від завантаження фото до підготовки фінального графічного файлу, придатного для верстата. У цьому контексті інтерфейс користувача стає ключовим елементом: саме він визначає зручність, швидкість та точність роботи дизайнера.

2.2 Огляд інтерфейсів у спеціалізованому ПЗ

У рамках дослідження було проведено аналіз існуючих інтерфейсів спеціалізованих програм для обробки зображень у промисловості. У сучасному світі існує безліч варіантів дизайну інтерфейсів, які відрізняються за стилем і функціональністю. Обидва ці аспекти є ключовими: функціонал визначає можливості програми, тоді як стиль впливає на зручність та естетичну привабливість

Таблиця 2.1 Порівняльний аналіз спеціалізованого ПЗ

Переваги спеціалізованого ПЗ:	**Недоліки спеціалізованого ПЗ:**
Широкий функціонал для конкретних завдань: дозволяє виконувати вузькопрофільні задачі ефективніше, ніж універсальні програми.	Висока складність для новачків: через велике різноманіття функцій користувачам потрібно витратити значний час на навчання.
Підвищена ефективність роботи: оптимізовані процеси дозволяють досягати високих результатів за короткий час.	Висока вартість: професійні програми часто мають високу ціну або вимагають регулярної оплати за підписку.
Гнучкість і налаштування: можливість адаптації інтерфейсу та функціоналу під індивідуальні потреби користувача.	Високі вимоги до апаратного забезпечення: стабільна робота потребує потужного обладнання.
Підтримка професійного рівня якості: забезпечує створення контенту, який відповідає індустріальним стандартам.	Тривалий час навчання: для повного опанування можливостей програми потрібні додаткові ресурси та практика.

Інтеграція з іншими інструментами: покращує робочий процес і сприяє більш злагодженій роботі в команді.	Можливість перенасичення функціями: багато функцій можуть залишатися невикористаними, якщо користувач працює з простими задачами.
	Складність підтримки та оновлення: часті оновлення можуть викликати труднощі в адаптації до нових змін.

Одним із прикладів, який демонструє ці аспекти, є Adobe Photoshop. Його інтерфейс поєднує стильний дизайн і багатий набір інструментів, але вимагає часу для освоєння. Темна кольорова схема та можливість персоналізації роблять його комфортним для тривалої роботи, однак для новачків програма може здаватися складною.

Основні висновки:

Складність освоєння: більшість програм орієнтовані на досвідчених технічних користувачів, тоді як дизайнери потребують більш інтуїтивного середовища.

Відсутність адаптивності: чимало програм мають фіксовану структуру, яка погано масштабується на різні типи пристроїв.

Функціональне перенасичення: надлишок вкладок, параметрів і технічних термінів ускладнює роботу з простими завданнями.

Мінімальний зворотний зв’язок: часто відсутні підказки, попередження про помилки або автоматичне коригування візуальних налаштувань [2].

Прикладом такого підходу може бути вбудоване ПЗ деяких китайських чи німецьких верстатів, де користувач повинен вручну адаптувати кольори, лінії, розміри, причому кожен параметр — у різному вікні або файлі. Це особливо неефективно для дизайнерів, які орієнтуються на візуальне сприйняття, а не на машинний код.

2.3 Основні принципи ефективного інтерфейсу

На основі аналізу UX/UI-дизайну сформульовано основні вимоги до майбутнього інтерфейсу:

Інтуїтивність: інтерфейс має бути зрозумілим без читання довідки. Назви кнопок, розташування елементів та логіка навігації повинні бути максимально природними.

Модульність: кожна функція має бути ізольованою, з можливістю комбінування в межах робочого простору.

Адаптивність: підтримка роботи на різних роздільних здатностях і пристроях.

Зворотний зв’язок: підтвердження дій, попередження про обмеження верстату (наприклад, перевищення кількості кольорів), індикатори стану.

Візуалізація в реальному часі: зміни в параметрах мають відображатися одразу на екрані без потреби повторної обробки.

Швидкий доступ до часто використовуваних функцій: обрізка, зміна палітри, кольорове квантування.

Базові принципи проектування, орієнтованого на користувача :

1) розуміння користувачів та їх задач, залучення користувачів на різних етапах життєвого циклу ПЗ;

2) визначення цілей, які можна виміряти; встановлення критеріїв успіху з точки зору користувачів та замовників;

3) проект повинен передбачати нову компетентність користувача, яка по відношенню до продукту включає пакетування, маркетинг, навчання, надруковану інформацію, налагодження параметрів, інсталяцію, екранні форми (екранна форма - графічно відображена форма (локація) в програмному забезпеченні; цифрове зображення, одержане програмним забезпеченням за командою користувача), графіку, довідки, іншу експлуатційну підтримку, оновлення та деінсталяцію;

4) оцінювання та тестування ПЗ за участю фактичних користувачів для визначення, чи досягнуті цілі та які є проблеми;

5) ітеративний підхід – якщо цілі не досягнуті або існують проблеми, слід внести виправлення та провести повторну перевірку.

Розглянемо різні підходи до проектування ПЗ .

1.Проектування "ззовні-всередину" (outside-in)

Спрямований на інтерфейс та доступні користувачу властивості програмного продукту

2.Проектування "зсередини-назовні" (inside-out)

Починається з розгляду внутрішніх властивостей системи

3 Однократне проектування (без ітерацій)

Проектування без встановлення планованого обсягу робіт з конструювання продукту та ІК

4 Багатократне (ітераційне) проектування

Концентрується на ітераційній побудові ІК та його основних факторах практичності

5 Проектування з використанням теорії "великого вибуху" (big bang theory)

Спроба розробити "все або нічого", тобто ПЗ проектується і реалізується паралельно

6 Еволюційне проектування

Зосереджене на побудові продукту з покроковим нарощуванням і уточненням можливостей продукту

Будь-який процес розроблення ІК повинен бути ітераційним, оскільки вдалий інтерфейс неможливо одержати без періодичного повернення до попередніх етапів. Критерієм для завершення ітераційної розробки є той факт, що усі вимоги користувачів задоволені, а сам продукт відповідає запланованим цілям. Може здаватись, що ітераційний процес займає багато часу через багаторазові проходження етапів розробки. Однак початкові проходження етапів допомагають створити варіанти розробок та прототипів, які в наступних ітераціях зекономлять час на впровадження та тестування [2].

2.4 Порівняльний аналіз створення інтерфейсу в JavaFX та PyQt

У процесі створення програмних інтерфейсів важливо обрати технологію, яка дозволить реалізувати інтуїтивно зрозумілий, гнучкий та функціональний користувацький інтерфейс. Одними з багатьох інструментів для цього є JavaFX (для мови Java) та PyQt (для мови Python). Обидва фреймворки мають схожі підходи до створення інтерфейсів, що ґрунтуються на компонуванні візуальних елементів та подієво-орієнтованому програмуванні.

JavaFX дозволяє будувати інтерфейс як безпосередньо в коді на Java, так і за допомогою декларативного опису у форматі FXML, що може редагуватись у візуальному редакторі SceneBuilder. В основі JavaFX лежить ієрархічна модель, де всі елементи додаються до сцени, яка розміщується у вікні (Stage). Приклад створення кнопки в JavaFX виглядає наступним чином:

А.1.1

Подібна логіка реалізується і в PyQt, де інтерфейс можна створити у коді або за допомогою Qt Designer. В PyQt також використовується ієрархічна модель, де елементи додаються до контейнера з використанням layout-компонувальників. Приклад кнопки:

Код А.1.2

Однією з основних переваг обох фреймворків є підтримка декларативного опису інтерфейсу — у JavaFX через FXML, у PyQt через UI-

файли. Це дозволяє розділити логіку програми та опис інтерфейсу, що є важливим у великих проєктах та при роботі в команді.

Для стилізації JavaFX підтримує власний діалект CSS, що дозволяє змінювати вигляд елементів за допомогою окремих стилів. У PyQt також можливо стилізувати інтерфейс через метод setStyleSheet(), який використовує синтаксис, близький до CSS.

Що стосується обробки подій, то JavaFX використовує лямбда-вирази або методи-обробники, які викликаються при взаємодії з елементами. У PyQt застосовується система сигналів і слотів, яка дозволяє підключати функції до подій, що робить обробку подій дуже гнучкою.

З точки зору продуктивності, JavaFX демонструє високу швидкість виконання завдяки запуску на Java Virtual Machine. PyQt, своєю чергою, забезпечує прийнятну продуктивність для більшості завдань, особливо з урахуванням того, що він використовується в поєднанні з Python, відомим своєю простотою та швидкістю розробки.

Обидва фреймворки є кросплатформенними й підтримують операційні системи Windows, Linux та macOS. JavaFX зазвичай застосовується в складніших десктопних рішеннях, наприклад, у редакторах даних, графічних програмах або засобах автоматизації. PyQt часто використовується для створення утиліт, наукових застосунків, GUI для скриптів та інструментів.

Таким чином, JavaFX і PyQt мають подібну архітектуру, але орієнтовані на різні екосистеми. JavaFX краще підходить для програм, що вимагають високої продуктивності та глибокої інтеграції з Java-середовищем. Натомість PyQt є чудовим вибором для швидкої розробки, прототипування та створення зручних інтерфейсів на Python.

2.5 Висновки за результатами огляду

Огляд існуючих рішень показав, що жодне з них повною мірою не відповідає сучасним потребам дизайнерів трикотажних виробів, особливо якщо йдеться про швидку, зручну та наочну підготовку зображень. Саме тому є потреба у створенні спеціалізованого програмного інтерфейсу, який поєднує функціональність професійного інструмента з простотою і гнучкістю візуального редактора.

Такий інтерфейс має стати основою для ефективного робочого середовища, де дизайнер може зосередитися на творчих завданнях, а технічні обмеження будуть автоматично враховуватись і візуалізуватись.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Аналіз вимог до інтерфейсу професійного програмного забезпечення

Розробка користувацького інтерфейсу для професійного програмного забезпечення вимагає чіткого розуміння задач, які стоять перед кінцевим користувачем. У нашому випадку користувачами є дизайнери трикотажних виробів, які працюють із графікою, готуючи її до друку на автоматизованому обладнанні.

Вимоги до програми зосереджувалися переважно на її функціональності. Основне завдання полягало в тому, щоб забезпечити точне і надійне перетворення фотографій для подальшого використання на верстаті. Це означало, що ключовими аспектами були ефективність обробки зображень, коректність алгоритмів і стабільність роботи програми.

Що стосується інтерфейсу, до нього не висувалися суворі естетичні вимоги. Основною метою було надати користувачеві простий і зрозумілий спосіб взаємодії з програмою. Тим не менш, повністю нехтувати зручністю використання було б неправильно. Навіть при відсутності високих стандартів щодо дизайну, інтерфейс мав бути достатньо організованим, щоб користувач міг легко орієнтуватися і швидко знаходити необхідні функції.

Програма не повинна виглядати так, ніби її розробляли поспіхом або без продумування зручності розташування елементів. Безсистемне розміщення кнопок і елементів управління могло б призвести до плутанини і ускладнити роботу з додатком, навіть якщо його функціональність виконувалася на високому рівні. Тому, хоча загальні

вимоги до інтерфейсу були відносно невисокими, логічна структура і розташування елементів залишалися важливими.

Користувачі повинні мати змогу інтуїтивно зрозуміти, як використовувати програму. Це включає чітко підписані кнопки, логічно розташовані елементи управління та мінімум зайвих дій для досягнення основної мети. Зрештою, добре продуманий інтерфейс, навіть при мінімалістичному підході, може значно підвищити загальний досвід роботи з програмою.

Основними вимогами до такого інтерфейсу є:

- Інтуїтивність: користувач повинен легко орієнтуватися у програмі без потреби проходити додаткове навчання;
- Фокус на графіку: максимальна площа інтерфейсу повинна бути відведена для роботи з зображенням, а панелі керування — зведені до мінімально необхідного набору;
- Інтеграція функцій редагування: програма повинна дозволяти змінювати розмір зображення, обрізати його, зменшувати кількість кольорів, застосовувати фільтри яскравості й контрасту — все це в одному вікні;
- Автоматична адаптація до обмежень верстату: при виборі шаблону нитководів інтерфейс має автоматично підказувати або блокувати недопустимі комбінації кольорів;
- Можливість швидкого попереднього перегляду результату: користувач повинен бачити, як зображення виглядатиме після обробки — до моменту збереження файлу.

3.2 Принципи побудови інтерфейсу: UX/UI-орієнтований підхід

Інтерфейс програми спроектовано на основі сучасних принципів UX/UI-дизайну, що дозволяє зробити взаємодію з додатком максимально ефективною та зручною:

- Єдина точка доступу до функцій: усі основні функції (завантаження, редагування, збереження) згруповані у верхньому меню або бічній панелі;
- Зв’язок між елементами керування та візуальним виводом: наприклад, зміна кількості кольорів одразу змінює відображення зображення в реальному часі;
- Контекстна допомога та підказки: наведення курсору на елемент відображає короткий опис його функції, що полегшує освоєння програми новими користувачами;
- Темна та світла тема інтерфейсу: реалізована можливість перемикання режимів для зменшення навантаження на зір під час тривалої роботи;
- Адаптивність макету: інтерфейс коректно масштабується під різні розміри вікна, що дозволяє працювати як на великих моніторах, так і на ноутбуках.

Побудова ефективного та приємного у використанні інтерфейсу є ключовим фактором успіху будь-якого цифрового продукту. Сучасний підхід до цього процесу базується на принципах UX (User Experience - досвід користувача) та UI (User Interface - користувацький інтерфейс). Ці два аспекти тісно пов’язані і взаємодоповнюють один одного, створюючи цілісний та позитивний досвід для користувача.

UX (User Experience) зосереджується на загальному враженні користувача від взаємодії з продуктом. Це не лише про те, як виглядає інтерфейс, а й про те, наскільки він зручний, інтуїтивно зрозумілий, ефективний та приємний у використанні.

Основні принципи UX включають:

1. Користувачоцентричність: Все починається з розуміння потреб, цілей та поведінки цільової аудиторії. Дизайн має бути орієнтований на користувача, вирішувати його проблеми та задовольняти його потреби.
2. Зручність використання (Usability): Інтерфейс має бути легким для освоєння та використання. Користувач повинен швидко знаходити потрібну інформацію та виконувати необхідні дії без зайвих зусиль.
3. Доступність (Accessibility): Продукт має бути доступним для людей з різними можливостями, включаючи людей з обмеженими можливостями. Це означає використання відповідних шрифтів, кольорів, альтернативних текстів для зображень тощо.
4. Корисність (Utility): Продукт має виконувати свою функцію та бути корисним для користувача. Він повинен вирішувати конкретну проблему або задовольняти певну потребу.
5. Бажаність (Desirability): Продукт має викликати позитивні емоції та бажання використовувати його. Це досягається за рахунок привабливого дизайну, приємної взаємодії та позитивного досвіду.
6. Надійність (Reliability): Продукт має працювати стабільно та без помилок. Користувач повинен бути впевнений, що продукт виконає свої функції належним чином.
7. Цінність (Value): Продукт має надавати цінність користувачеві, вирішуючи його проблеми або задовольняючи його потреби ефективніше, ніж альтернативні рішення.

UI (User Interface) стосується візуального оформлення та інтерактивних елементів інтерфейсу. Це те, що користувач бачить і з чим безпосередньо взаємодіє.

Основні принципи UI включають:

1.Візуальна ієрархія: Елементи інтерфейсу мають бути організовані таким чином, щоб користувач міг легко зрозуміти їхню важливість та взаємозв’язок. Використання розміру, кольору, шрифтів та відступів допомагає створити чітку візуальну ієрархію.

2.Послідовність (Consistency): Елементи інтерфейсу, їхнє розташування та поведінка мають бути послідовними на всіх екранах та в усіх частинах продукту. Це допомагає користувачеві швидко освоїти інтерфейс та передбачати його поведінку.

3.Зворотний зв’язок (Feedback): Інтерфейс має надавати користувачеві чіткий зворотний зв’язок про його дії. Наприклад, при натисканні кнопки вона може змінити колір або з’явитися повідомлення про успішне виконання дії.

4.Простота (Simplicity): Інтерфейс має бути максимально простим та зрозумілим. Уникайте зайвих елементів та складної навігації.

5.Естетика (Aesthetics): Візуальне оформлення інтерфейсу має бути привабливим та приємним для ока. Використання гармонійних кольорів, шрифтів та зображень створює позитивне враження.

6.Чіткість (Clarity): Текст та елементи інтерфейсу мають бути чіткими та легко читаними. Використання відповідних розмірів шрифтів, контрастності та відступів є важливим для забезпечення чіткості.

7.Ефективність (Efficiency): Інтерфейс має дозволяти користувачеві швидко та ефективно виконувати свої завдання. Це досягається за рахунок

оптимізації робочих процесів та мінімізації кількості кроків, необхідних для виконання дії.

UX/UI-орієнтований підхід означає, що дизайн інтерфейсу розглядається як цілісний процес, де UX та UI працюють разом для досягнення найкращого результату. Це не просто про створення красивого інтерфейсу, а про створення інтерфейсу, який вирішує проблеми користувача, є зручним у використанні та викликає позитивні емоції.

Етапи UX/UI-орієнтованого підходу:

-Дослідження та аналіз: Розуміння цільової аудиторії, її потреб, цілей та поведінки. Аналіз конкурентів та ринку.
-Проектування UX: Розробка структури продукту, навігації, сценаріїв використання. Створення прототипів та тестування з користувачами.
-Проектування UI: Розробка візуального стилю, вибір кольорів, шрифтів, іконок. Створення макетів інтерфейсу.
-Розробка та впровадження: Перетворення макетів на робочий продукт.
-Тестування та ітерації: Постійне тестування продукту з користувачами та внесення змін на основі зворотного зв’язку.

Переваги UX/UI-орієнтованого підходу:

Підвищення задоволеності користувачів: Продукт стає більш зручним та приємним у використанні.
Збільшення конверсії: Зручний та інтуїтивно зрозумілий інтерфейс сприяє досягненню цілей користувача (наприклад, здійснення покупки).
Зменшення витрат на підтримку: Менше запитань від користувачів, оскільки інтерфейс зрозумілий.

Підвищення лояльності користувачів: Позитивний досвід використання продукту сприяє поверненню користувачів.

Зміцнення бренду: Якісний та зручний продукт формує позитивне сприйняття бренду.

В цілому, UX/UI-орієнтований підхід є необхідним для створення успішних цифрових продуктів у сучасному світі. Він дозволяє зосередитися на потребах користувача та створити інтерфейс, який не лише виглядає добре, але й ефективно виконує свої функції та приносить задоволення від використання.

Розглянемо тестування як невід'ємну частину UX/UI-орієнтованого підходу, оскільки воно дозволяє перевірити, наскільки добре реалізовані принципи дизайну та чи відповідає продукт потребам користувачів.

Методи тестування інтерфейсу

Тестування інтерфейсу може проводитись на різних етапах розробки продукту, від ранніх прототипів до готового продукту. Воно допомагає виявити проблеми з юзабіліті, зрозуміти, як користувачі взаємодіють з інтерфейсом, та отримати цінний зворотний зв'язок для покращення.

1. Юзабіліті-тестування (Usability Testing):

Це один з найважливіших методів, який передбачає спостереження за тим, як реальні користувачі взаємодіють з інтерфейсом для виконання певних завдань.

- Модераційне юзабіліті-тестування: Тестування проводиться під наглядом модератора, який ставить завдання користувачеві, спостерігає за його діями, задає запитання та збирає зворотний зв'язок. Це дозволяє отримати глибоке розуміння проблем, з якими стикається користувач.

-Немодераційне юзабіліті-тестування: Користувачі виконують завдання самостійно, використовуючи спеціальні інструменти, які записують їхні дії, рух миші, кліки та голосові коментарі. Цей метод дозволяє охопити більшу кількість користувачів, але може бути менш глибоким у розумінні причин проблем.

- Віддалене юзабіліті-тестування: Тестування проводиться віддалено, коли користувач та модератор знаходяться в різних місцях. Це зручно для тестування з користувачами з різних регіонів.

- Тестування в лабораторії: Тестування проводиться в спеціально обладнаній лабораторії, де можна контролювати умови та використовувати спеціальне обладнання для запису поведінки користувачів (наприклад, відстеження руху очей).

2. А/Б-тестування (A/B Testing):

Цей метод передбачає порівняння двох або більше варіантів дизайну (наприклад, різних версій кнопки, заголовка або макету сторінки) для визначення, який з них є більш ефективним з точки зору досягнення певних цілей (наприклад, конверсії, кліків). Користувачі випадковим чином розподіляються між різними варіантами, і збираються дані про їхню поведінку.

3. Карткове сортування (Card Sorting):

Цей метод використовується для розуміння того, як користувачі організовують інформацію та як вони очікують знайти певні елементи на сайті чи в додатку. Користувачам надається набір карток з назвами контенту або функцій, і вони групують їх за категоріями, які, на їхню думку, є логічними. Це допомагає у проектуванні навігації та структури інформації.

4. Деревоподібне тестування (Tree Testing):

Цей метод є доповненням до карткового сортування і використовується для оцінки ефективності розробленої структури навігації. Користувачам надається завдання знайти певну інформацію або виконати дію, використовуючи лише текстове дерево навігації (без візуального інтерфейсу). Це допомагає виявити проблеми з логікою навігації.

5. Тестування першого кліка (First Click Testing):

Цей метод вимірює, наскільки легко користувачі можуть знайти потрібний елемент на сторінці. Користувачам надається завдання, і записується, куди вони клікають першим. Якщо перший клік веде до правильного місця, це свідчить про хорошу зрозумілість інтерфейсу.

6 Опитування та анкети (Surveys and Questionnaires):

Ці методи використовуються для збору кількісних та якісних даних про враження користувачів від інтерфейсу. Опитування можуть включати запитання про задоволеність, зручність використання, проблеми, з якими стикалися користувачі, тощо.

7. Аналіз веб-аналітики (Web Analytics Analysis):

Аналіз даних з інструментів веб-аналітики (наприклад, Google Analytics) може надати цінну інформацію про поведінку користувачів на сайті чи в додатку. Можна відстежувати такі показники, як час перебування на сторінці, показник відмов, шляхи користувачів, конверсії тощо. Це допомагає виявити проблемні місця в інтерфейсі.

8. Тестування доступності (Accessibility Testing):

Цей тип тестування спрямований на перевірку того, наскільки продукт доступний для людей з різними обмеженими можливостями (наприклад, проблеми із зором, слухом, моторикою). Використовуються спеціальні інструменти та методи для перевірки відповідності стандартам доступності (наприклад, WCAG).

9. Експертна оцінка (Expert Review):

Досвідчені UX/UI-дизайнери або експерти з юзабіліті можуть провести оцінку інтерфейсу на основі своїх знань та досвіду. Вони можуть виявити потенційні проблеми з юзабіліті, які можуть бути неочевидними для розробників.

10. Тестування сумісності (Compatibility Testing):

Цей тип тестування перевіряє, як інтерфейс відображається та функціонує на різних пристроях, операційних системах та браузерах. Це важливо для забезпечення послідовного досвіду користувача незалежно від платформи.

Вибір методу тестування залежить від:

Етапу розробки продукту

Цілей тестування

Доступних ресурсів (час, бюджет, кількість користувачів)

Типу продукту та його цільової аудиторії

Важливо пам’ятати:

Тестування має проводитись регулярно протягом усього процесу розробки. Необхідно тестувати з представниками цільової аудиторії.

Результати тестування мають бути проаналізовані, а виявлені проблеми виправлені. Ітераційний підхід до дизайну та тестування є ключовим для створення успішного продукту. Поєднання різних методів тестування дозволяє отримати більш повну картину про те, як користувачі взаємодіють з інтерфейсом, та виявити широкий спектр проблем [2-3].

3.3 Обгрунтування вибіру технологій для реалізації інтерфейсу

У процесі створення програмних інтерфейсів важливо обрати технологію, яка дозволить реалізувати інтуїтивно зрозумілий, гнучкий та функціональний користувацький інтерфейс. Одними з багатьох інструментів для цього є JavaFX (для мови Java) та PyQt (для мови Python). Обидва фреймворки мають схожі підходи до створення інтерфейсів, що ґрунтуються на компонуванні візуальних елементів та подієво-орієнтованому програмуванні.

Вибір платформи Java та бібліотеку JavaFX, був зумовлений тим,що вони забезпечують високий рівень гнучкості, продуктивність та кросплатформеність. Середовище розробки — IntelliJ IDEA, що підтримує інтеграцію з системою збірки Maven і дозволяє зручно керувати залежностями.

Основні переваги вибору JavaFX:

- FXML-структура — дозволяє розділити логіку (Java-код) від представлення (візуальна структура), що спрощує підтримку коду;
- Гнучкий механізм стилізації через CSS — дає змогу кастомізувати зовнішній вигляд інтерфейсу без зміни логіки;
- Підтримка двостороннього зв’язку з даними — що дає змогу інтерактивно зв’язувати змінні програми з елементами інтерфейсу (наприклад, слайдер із яскравістю);
- Можливість використання анімацій та ефектів — для покращення візуального сприйняття інтерфейсу.

JavaFX дозволяє будувати інтерфейс як безпосередньо в коді на Java, так і за допомогою декларативного опису у форматі FXML, що може

редагуватись у візуальному редакторі SceneBuilder. В основі JavaFX лежить ієрархічна модель, де всі елементи додаються до сцени, яка розміщується у вікні (Stage). Приклад створення кнопки в JavaFX виглядає наступним чином:

Код А.2.1

Подібна логіка реалізується і в PyQt, де інтерфейс можна створити у коді або за допомогою Qt Designer. В PyQt також використовується ієрархічна модель, де елементи додаються до контейнера з використанням layout-компонувальників. Приклад кнопки:

Код А.2.2

Однією з основних переваг обох фреймворків є підтримка декларативного опису інтерфейсу — у JavaFX через FXML, у PyQt через UI-файли. Це дозволяє розділити логіку програми та опис інтерфейсу, що є важливим у великих проєктах та при роботі в команді.

Для стилізації JavaFX підтримує власний діалект CSS, що дозволяє змінювати вигляд елементів за допомогою окремих стилів. У PyQt також можливо стилізувати інтерфейс через метод setStyleSheet(), який використовує синтаксис, близький до CSS.

Що стосується обробки подій, то JavaFX використовує лямбда-вирази або методи-обробники, які викликаються при взаємодії з елементами. У PyQt застосовується система сигналів і слотів, яка дозволяє підключати функції до подій, що робить обробку подій дуже гнучкою.

З точки зору продуктивності, JavaFX демонструє високу швидкість виконання завдяки запуску на Java Virtual Machine. PyQt, своєю чергою, забезпечує прийнятну продуктивність для більшості завдань, особливо з урахуванням того, що він використовується в поєднанні з Python, відомим своєю простотою та швидкістю розробки.

Обидва фреймворки є кросплатформенними й підтримують операційні системи Windows, Linux та macOS. JavaFX зазвичай застосовується в складніших десктопних рішеннях, наприклад, у редакторах даних, графічних програмах або засобах автоматизації. PyQt часто використовується для створення утиліт, наукових застосунків, GUI для скриптів та інструментів.

Таким чином, JavaFX і PyQt мають подібну архітектуру, але орієнтовані на різні екосистеми. JavaFX краще підходить для програм, що вимагають високої продуктивності та глибокої інтеграції з Java-середовищем. Натомість PyQt є чудовим вибором для швидкої розробки, прототипування та створення зручних інтерфейсів на Python.

3.4 Алгоритм роботи з програмою для автообробки зображень

Програма призначена для підготовки зображень до подальшої обробки на вишивальному або трикотажному верстаті. Вона поєднує автоматизовані алгоритми з можливістю ручного налаштування, що забезпечує гнучкість і точність у роботі з графічним матеріалом.

1. Запуск програми

Користувач відкриває програму, після чого ініціалізується графічний інтерфейс. Завантажуються бібліотеки, необхідні для обробки зображень,

а також створюється середовище для збереження поточного зображення та історії змін. Інтерфейс відображає панель керування, область попереднього перегляду зображення та набір інструментів редагування.

2. Завантаження робочого фото

На цьому етапі користувач натискає кнопку «Завантажити зображення», що відкриває стандартне вікно вибору файлу. Після вибору файл імпортується до програми та виводиться у вікні попереднього перегляду. В оперативній пам’яті створюється копія зображення для можливості відновлення його початкового вигляду в разі потреби.

3. Корекція яскравості та контрасту

Після завантаження зображення користувач має змогу скоригувати його яскравість і контраст. Для цього надаються повзунки, які дозволяють змінювати значення параметрів. Зміни відображаються в реальному часі у вікні перегляду. Після того як користувач задоволений результатом, він переходе до наступного кроку.

4. Синхронізація у пам’яті програми

Застосовані зміни зберігаються як поточний стан зображення. У пам’яті програми оновлюється буфер, у який додається копія нового зображення, що дозволяє надалі повертатися до проміжних етапів обробки. Це забезпечує зручність при багатоетапній роботі з матеріалом.

5. Запуск алгоритму обробки

Далі користувач переходить до основного етапу — обробки зображення за допомогою вбудованого алгоритму. Після натискання кнопки «Почати підбір» запускається алгоритм обробки. Також є приховані параметри, які

можна відкрити і змінити як алгоитм бед обробляти фото. Після налаштування параметрів та запуску алгоритму, у інтерфейсі з’являться 3 нових фото, з якими дизайнер буде далі працювати. Цими фото є оброблена фото для подальшої ручної роботи дизайнерів, схема для верстату та зображення симуляції малюнку на готовому продукті. Ще є 3 режими обробки фото, які потрібні дизайнерам через специфічність шиття деяких видів шкарпеток.

6. Зміна кольорової палітри

Після обробки користувач може змінити кольорову палітру зображення, що є важливим кроком перед передачею зображення на верстат. Справа від готових фото є палітра кольорів на кожний нитковод зі своїм автоматично підібраним кольором. Користувач натиснувши на один із цих кольорів відкриє вікно, де він може змінити потрібний для роботи.

7. Повторна обробка (опціонально)

Якщо результат обробки не задовольняє користувача або виникає потреба протестувати інші налаштування, передбачена можливість повторного запуску алгоритму. Після натискання кнопки «Почати підбір», програма повторно проходить через алгоритм і створює нові результати для дизайнерів.

8. Збереження результату

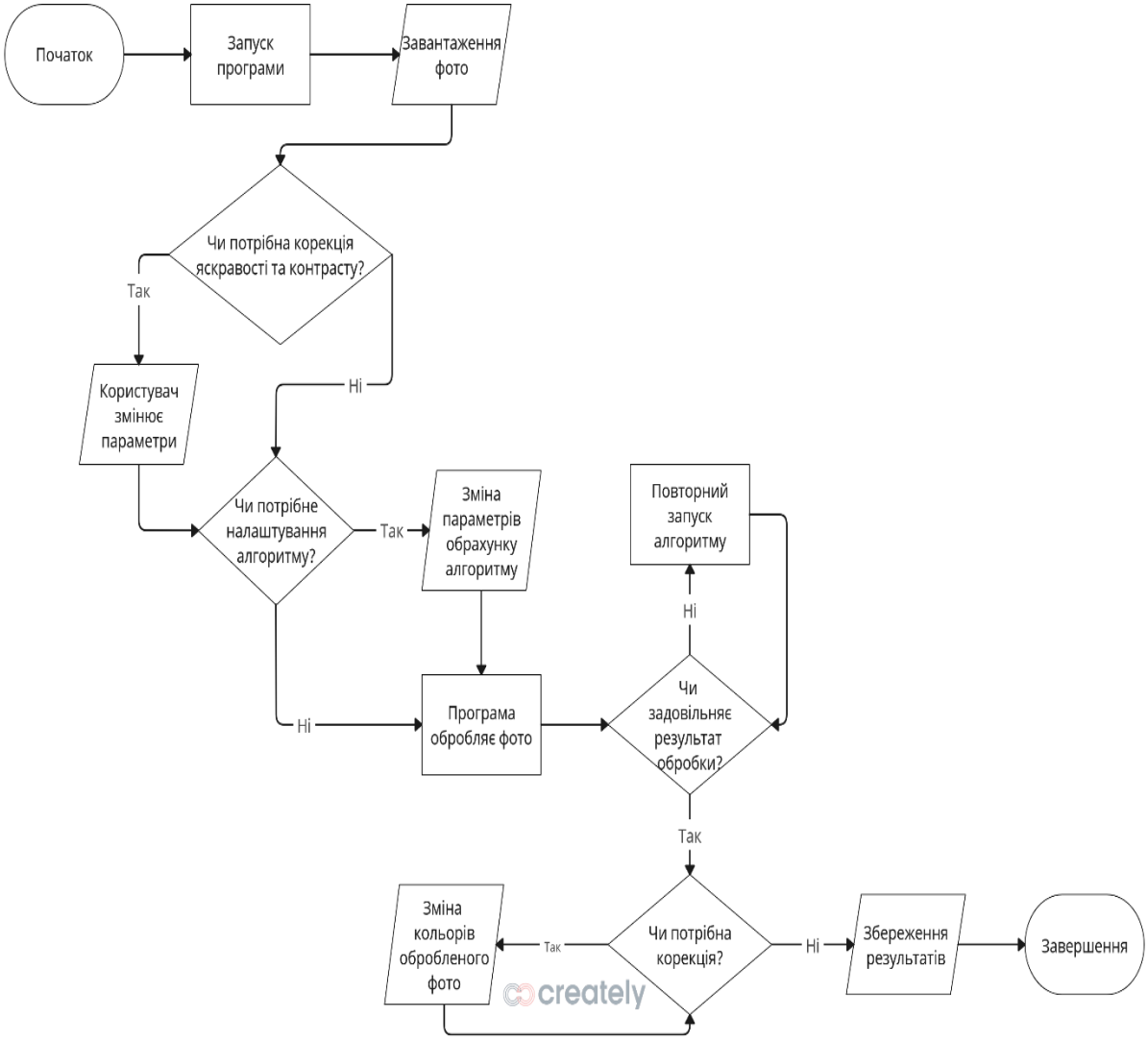
Після завершення усіх етапів обробки користувач обирає один або декілька фото, які хоче завантажити собі на ПК, та натискає кнопку «Зберегти». Після чого відкривається вікно проводника у якому користувач обирає шлях, де саме зберегти результати. Обравши потрібну папку, будуть

з’являтися нові вікна проводника, взалежності скільки результатів хоче завантажити, де користувач може змінити назву файлу.

9. Завершення роботи

Після збереження користувач може або очистити поточну сесію для початку роботи з новим зображенням натиснувши «Скинути все», або вийти з програми. Усі проміжні дані, які зберігалися в оперативній пам’яті, видаляються для забезпечення оптимального використання ресурсів системи.

3.5 Блок-схема принципу роботи користувача з програмою



Малюнок 3.5 Блок-схема принципу роботи користувача з програмою

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1 Програмна реалізація основних елементів інтерфейсу

У мові програмування Java за допомогою бібліотеки JavaFX створення графічного інтерфейсу користувача відбувається з використанням як програмного підходу (написання коду вручну), так і декларативного підходу, заснованого на файлах розмітки FXML. Останній варіант особливо зручний для розділення логіки програми та її графічного представлення. Це дозволяє дизайнерам інтерфейсу та розробникам програмної логіки працювати незалежно одне від одного. FXML є мовою розмітки на основі XML, яка дозволяє описувати структуру інтерфейсу користувача у зрозумілій та зручній формі, подібній до HTML.

Розробка інтерфейсу з використанням FXML зазвичай починається з створення файлу з розширенням .fxml, у якому описуються всі графічні елементи, які будуть використані у вікні програми. Це можуть бути кнопки, мітки, поля введення, зображення, панелі тощо. Наприклад, для створення простої панелі керування може бути використаний контейнер типу VBox, який міститиме кілька кнопок, що забезпечує зручне розташування компонентів. У FXML це виглядатиме як структурований набір XML-тегів, що відповідають JavaFX-компонентам.

```
<VBox id="main" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.example.socks.HelloController" spacing="5"
      stylesheets="@Style.css">
    <HBox id="Buttons" spacing="5">
        <Button fx:id="Load" text="Оберіть фото" onAction="#handleButtonClick"
                style="-fx-translate-x: 10; -fx-translate-y: 10;"/>
        <Button fx:id="Synk" text="Синхронізувати фото" onAction="#onSynk"
                style="-fx-translate-x: 10; -fx-translate-y: 10;"/>
        <Button fx:id="RemeuveAll" text="Скинути все" onAction="#onRemeuveAll"
                style="-fx-translate-x: 10; -fx-translate-y: 10;"/>
        <Button fx:id="RemeuveVizual" text="Скинути зміни кольорів" onAction="#onRemeuveVizual"
                style="-fx-translate-x: 10; -fx-translate-y: 10;"/>
        <Button fx:id="Start" text="Почати підбір" onAction="#onStartClick"
                style="-fx-translate-x: 10; -fx-translate-y: 10;"/>
        <Button fx:id="Save" text="Зберегти" onAction="#onSaveClick" style="-fx-translate-x: 10; -fx-translate-y: 10;"/>
        <Button fx:id="setting" text="Показати/Приховати параметри" onAction="#onSettingClick"
                style="-fx-translate-x: 10; -fx-translate-y: 10;"/>
        <VBox fx:id="Settings" visible="false" style="-fx-translate-y: 15;">
            <HBox>
                <TextField fx:id="Q" text="27" style="-fx-translate-x: 6;"/>
                <TextField fx:id="T" text="0.2" style="-fx-translate-x: 30;"/>
                <TextField fx:id="S" text="5" style="-fx-translate-x: 53;"/>
                <TextField fx:id="B" text="1.8" style="-fx-translate-x: 63;"/>
            </HBox>
            <HBox>
                <Label fx:id="Qlabel" text="Межа глобального підбору" style="-fx-translate-x: 3;"/>
                <Label fx:id="Tlabel" text="Крок глобального спрощення" style="-fx-translate-x: 18;"/>
                <Label fx:id="Slabel" text="Нижня межа закін. підбору" style="-fx-translate-x: 30;"/>
                <Label fx:id="Blabel" text="Нач. спрощення глоб. підбору" style="-fx-translate-x: 40;"/>
            </HBox>
        </VBox>
    </HBox>
    <HBox style="-fx-translate-y: 5; -fx-translate-x: 10;">
        <CheckBox fx:id="saveImageCheckBox" text="Зберегти фото Out"/>
        <CheckBox fx:id="saveStanokCheckBox" text="Зберегти фото Stanok"/>
        <CheckBox fx:id="saveVizualCheckBox" text="Зберегти фото Visual"/>
    </HBox>
    <Label fx:id="fileNameLabel" text=""/>
    <HBox id="resize" spacing="10">
        <VBox>
            <TextField fx:id="LabelM" text=""/>
            <Label fx:id="LabeM" text="Ширина"/>
```

Малюнок 4.1 Приклад виставлення елементів інтерфейсу у FXML

У верхній частині знаходиться *HBox* із кнопками, які відповідають за ключові функції програми. Як показано на цьому скріншоті, кнопки з *fx:id* на кшталт *Load*, *Synk*, *RemoveAll*, та інші, мають прив'язані обробники подій через атрибут *onAction*. Це означає, що при натисканні відповідної кнопки викликається метод у контролері, який обробляє певну дію. Наприклад, кнопка з текстом "Обрати фото" викликає метод *#handleButtonClick*, що ймовірно відповідає за завантаження зображення.

Окрім основних функціональни кнопок, як показано на скріншоті, у структурі є прихований блок налаштувань з *fx:id="Settings"*, який стає видимим за допомогою кнопки "Показати/Приховати параметри". Цей блок містить текстові поля для введення числових значень і мітки, які відображають межі та параметри глобального підбору. Наприклад, поле з *fx:id="Q"* дозволяє користувачеві вказати певне значення, а мітка поруч пояснює, що саме це поле означає.

Стилізація компонентів здійснюється через підключений CSS-файл *«(stylesheets="@Style.css")»*, що дозволяє налаштовувати зовнішній вигляд інтерфейсу, не змінюючи FXML-файл. Як видно, компоненти позиціонуються за допомогою атрибутів стилю *-fx-translate-x* і *-fx-translate-y*, що забезпечує точне налаштування їхнього розташування.

Таким чином, інтерфейс, створений за допомогою Java FXML у IntelliJ IDEA, демонструє добре організовану структуру з чітким поділом логіки та зовнішнього вигляду. Кожен елемент має свою функцію та легко інтегрується з бізнес-логікою через контролер, що робить розробку зрозумілою і зручною

Щоб цей інтерфейс можна було завантажити в програмі, у Java-коді створюється головний клас, який розширює базовий клас Application із JavaFX. У методі start(Stage primaryStage) зазвичай здійснюється завантаження FXML-файлу за допомогою класу FXMLLoader, після чого сцена прив’язується до створеного вікна, і відображається користувачеві. Такий підхід дозволяє зберігати основну логіку завантаження інтерфейсу у єдиному місці.

```
public void start(Stage stage) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource( name: "hello-view.fxml"));
    Scene scene = new Scene(fxmlLoader.load(), v: 1600, v1: 900);
    stage.setMaximized(true);
    stage.setTitle("Premier Socks");
    stage.setScene(scene);
    stage.show();
}
```

Малюнок 4.2 Метод старт, який створює сцену та завантажує файл FXML

Одним із важливих елементів є клас-контролер, який пов'язується з FXML-файлом через атрибут fx:controller у кореневому тегу. Контролер відповідає за обробку подій користувача та керування елементами інтерфейсу. Наприклад, при натисканні кнопки користувач може викликати певний метод, що реалізований у цьому контролері. Щоб мати доступ до елементів, оголошених у FXML, у відповідному Java-класі використовується анотація @FXML, що дозволяє зв’язати об’єкти інтерфейсу з відповідними змінними.

Як показано на скріншоті, кожен елемент інтерфейсу, наприклад кнопки, текстові поля чи зображення, має відповідне поле у контролері, яке дозволяє програмно взаємодіяти з цим елементом.

```
4 usages
private static Color selectedColor;
17 usages
private static BufferedImage Vizual;
3 usages
private static int Gheigh;
5 usages
private static int newGweight;
3 usages
private static int Gweight;
@FXML
private Button Load;
@FXML
private Label fileNameLabel;
@FXML
private ImageView imageView;
@FXML
private ImageView ImageStanok;
@FXML
private ImageView Zakathchick;
3 usages
private PhotoLoader photoLoader = new PhotoLoader();

@FXML
private ComboBox comboBox;
F0x1G
@FXML
private void handleButtonClick(ActionEvent event) { openFileChooser(); }
@FXML
private TextField LabelM;
@FXML
private TextField LabelN;
@FXML
private VBox RightPanel;
@FXML
private VBox Settings;
@FXML
private TextField Q;
@FXML
private TextField T;
```

Рисунок 4.3 Приклад коду у файлі HelloController

Наприклад, кнопка ***Load***, яка відповідає за завантаження фотографій, пов’язана з методом ***handleButtonClick***. Цей метод обробляє подію натискання кнопки та викликає функцію ***openFileChooser()***, яка, ймовірно, відкриває файловий діалог. Поля ***TextField***, такі як ***Q***, ***T*** чи ***B***, відповідають за введення параметрів, що впливають на роботу програми, і можуть використовуватися для передачі значень у логіку обробки.

Також важливою частиною є поля для зображень ***imageView***, ***ImageStanok*** та ***Zakatchchick***. Вони використовуються для відображення різних етапів обробки фотографій чи проміжних результатів. Використання цих полів у поєднанні з логікою програми дозволяє користувачеві взаємодіяти з графічним інтерфейсом у режимі реального часу.

Це забезпечує повний контроль над поведінкою інтерфейсу.

```
@FXML
private CheckBox autoResCheck;
@FXML
private Slider sliderContrast;
@FXML
private Slider sliderBrightness;
private Image resetImage;  6 usages
private Image fullResetImage;  3 usages
@FXML
private CheckBox saveImageCheckBox;
@FXML
private CheckBox saveStanokCheckBox;
@FXML
private CheckBox saveVizualCheckBox;
@FXML
private Button generate;
```

Малюнок 4.4 Приклад ініціалізації елементу інтерфейсу в основному коді в класі Controller

Окрему увагу слід звернути на правильну організацію структури проєкту. Як правило, FXML-файли зберігаються в окремій папці (наприклад, resources), тоді як класи-контролери і основна логіка розміщуються у відповідних Java-пакетах. Це дозволяє зберігати порядок у проекті та легко масштабувати його в майбутньому. Важливо також не забувати, що завантаження ресурсів має здійснюватися через правильний шлях у методі getResource, особливо якщо проєкт буде зібраний у форматі .jar.

Використання JavaFX та FXML забезпечує високу гнучкість при створенні інтерфейсів. Крім ручного редагування FXML, для його створення можна скористатися візуальним редактором Scene Builder, який дозволяє розміщувати елементи у графічному режимі та одразу призначати для них ID та методи обробки подій. Після збереження FXML-файлу його одразу можна підключити до Java-проєкту та продовжити реалізацію логіки в контролерах.

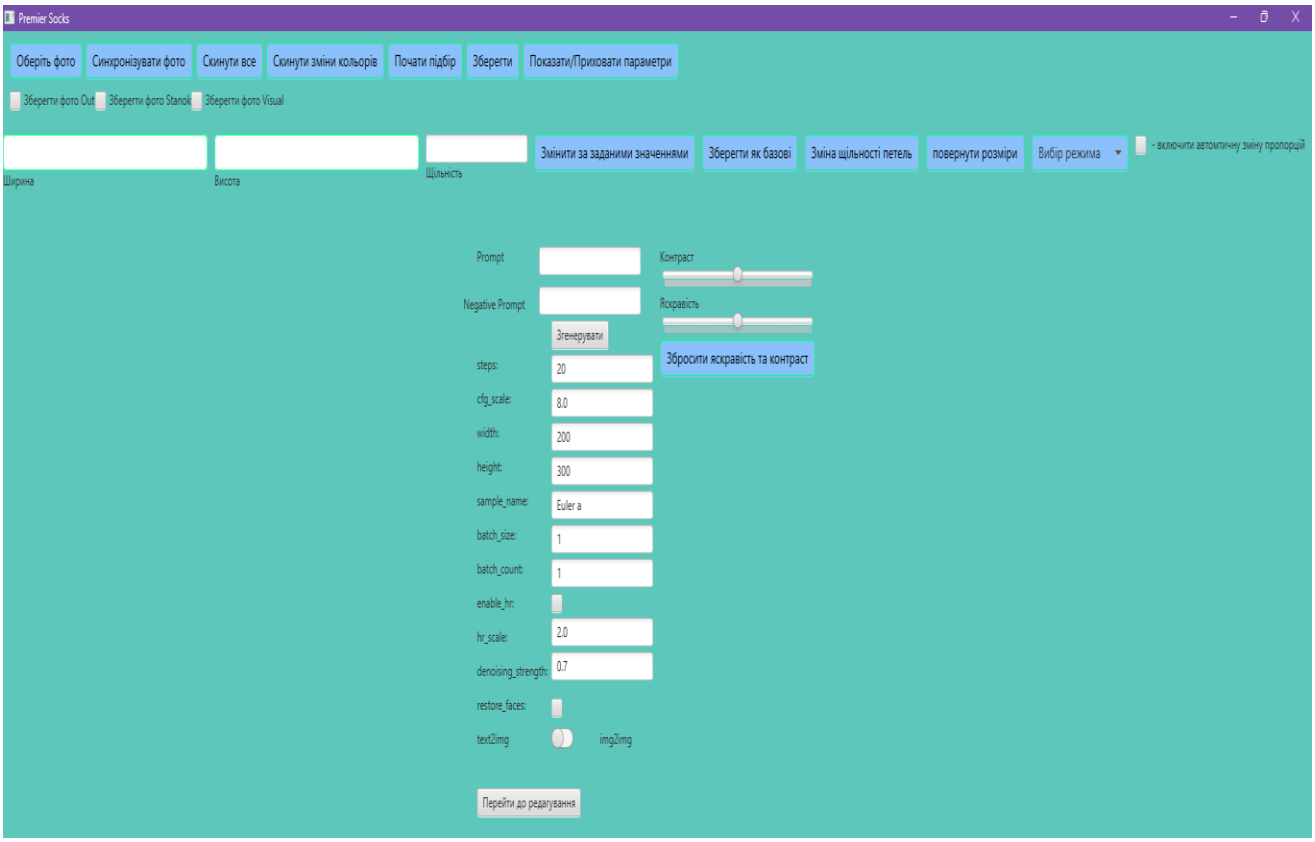
Загалом, поєднання Java-коду, FXML та механізму контролерів дозволяє створювати структуровані, зручні та масштабовані додатки з графічним інтерфейсом. Такий підхід є професійним і рекомендованим для розробки сучасного програмного забезпечення, де важливо розділити візуальну частину і логіку програми. Завдяки цьому підтримка та розвиток коду значно спрощуються.

Таким чином, програмна реалізація забезпечує взаємодію між графічними елементами інтерфейсу і внутрішньою логікою додатка, створюючи цілісну систему, яка відповідає за функціональність програми.

4.2 Розробка основних складових програми

Загальна структура програми

Створена програма складається з кількох ключових модулів, кожен із яких відповідає за окремий етап обробки зображення: завантаження файлу, налаштування алгоритму обробки, можливість створення картинки за допомого ШІ, робоча область програми для зміни самого фото, збереження результату після обробки. Основна взаємодія користувача відбувається через головне вікно інтерфейсу, яке включає робочу область та панель керування.



Малюнок 4.5 Вид програми, при першому запуску.

Програма розділена на такі функціональні частини:

- Завантаження робочого фото: підтримує формати BMP, PNG, JPG;Панель редагування: дозволяє регулювати основні параметри — яскравість, контраст, а також змінювати розмір зображення;
- Робоча зона: частина інтерфейсу на якій можна переглянути оброблений матеріал та змінити його кольорову палітру;
- Збереження фото: можливість зберегти одне. два або усі фото після обробки програмою.

Процес роботи з програмою можна розбити на 10 пунктів:

1. Запуск прорами
2. Завантаження робочого фото
3. Корекція яскравості та контрасту (при необхідності)
4. Зміна параметрів обробки алгоритму (при необхідності)
5. Запуск алгоритму
6. Корекція кольорів готового зображення продукту (при необхідності)
7. Збереження декількох вподобаємих результатів (а саме, фото, схема та емуляція на продукті)
8. Повторний прогон через алгоритм (при необхідності)
9. Збереження нових результатів
10. Очищення пам’яті програми або повне закриття програми

4.3 Інструкція роботи з програмою

1. Завантаження та попередній перегляд зображення

Коли програма запустилася, натисніть на кнопку "Оберіть фото" і знайдіть потрібний файл для обробки. Під цифрою 1 показана сама кнопка. Під цифрою 2 показано саме вікно проводника, через який знаходимо фото.

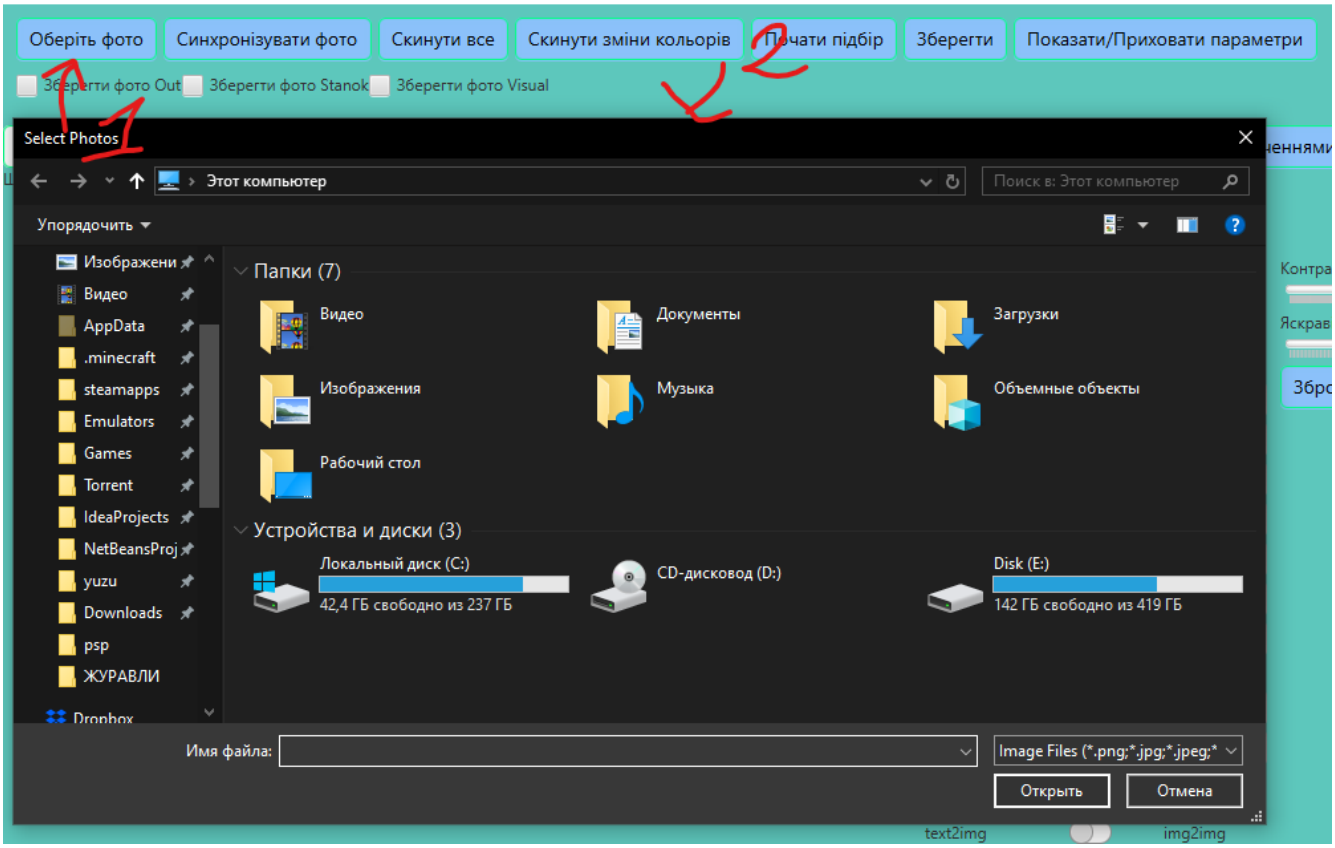


Рисунок 4.6 Проводник для завантаження фото

P.S. Рекомендуємо обробити фото, прибравши в ньому шуми. Для цього можна використовувати Photoshop або якийсь онлайн ресурс з ШІ. Приклад: з ліва не оброблена картинка, де є багато відтінків інших кольорів, які не зразу побачиш.

Справа — оброблений варіант, без шумів, без зайвих деталей по краям(на фотці зліва є білі полоски, справа і знизу. Їх краще прибирати.

Рисунок 4.7-4.8 Зліва фото з шумом, зправа без шуму

За потреби змініть параметри обробки фото натиснувши на кнопку (скрін) "Показати/Приховати параметри" (якщо вони закриті, то берутся базові) також можна скинути значення до базових нажавши 2 раза по кнопці.

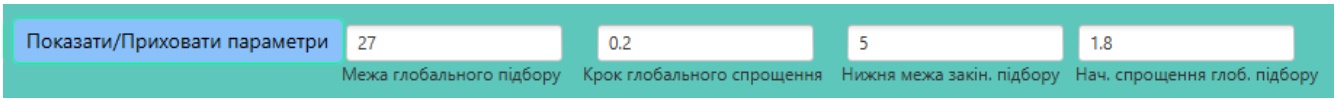


Рисунок 4.9 Параметри змін обробки

P.S. Робочі діапазони можуть варіюватися від фото до фото. Після змін параметр результат може бути кращим або гіршим.

2. Базове редагування зображення

Інтерфейс редагування містить слайдери для зміни яскравості, контрасту, а також кнопки для зміни розміру. Усі зміни відображаються на полотні після натискання на кнопку для підтвердження змін За потреби можна змінити розмір фото, кількість петель, контраст та яскравість та розтянути зображення можна відповідними кнопками, (щоб зміни розмір на свої параметри введіть відповідні значення в висоту та ширину та натисніть "Змінити за заданими значеннями").

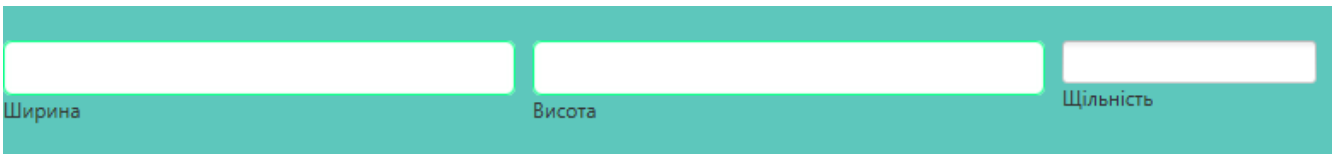


Рисунок 4.10 Поля для змін ширини, висоти та щільності

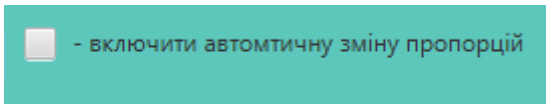


Рисунок 4.11 Опція для авто зміни пропорцій

Також можна змінити контрастність та яскравість робочого фото.

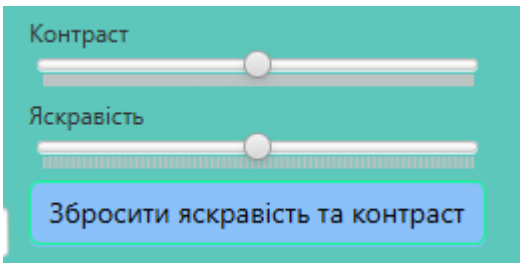


Рисунок 4.12 Повзунки контрасту та яскравості

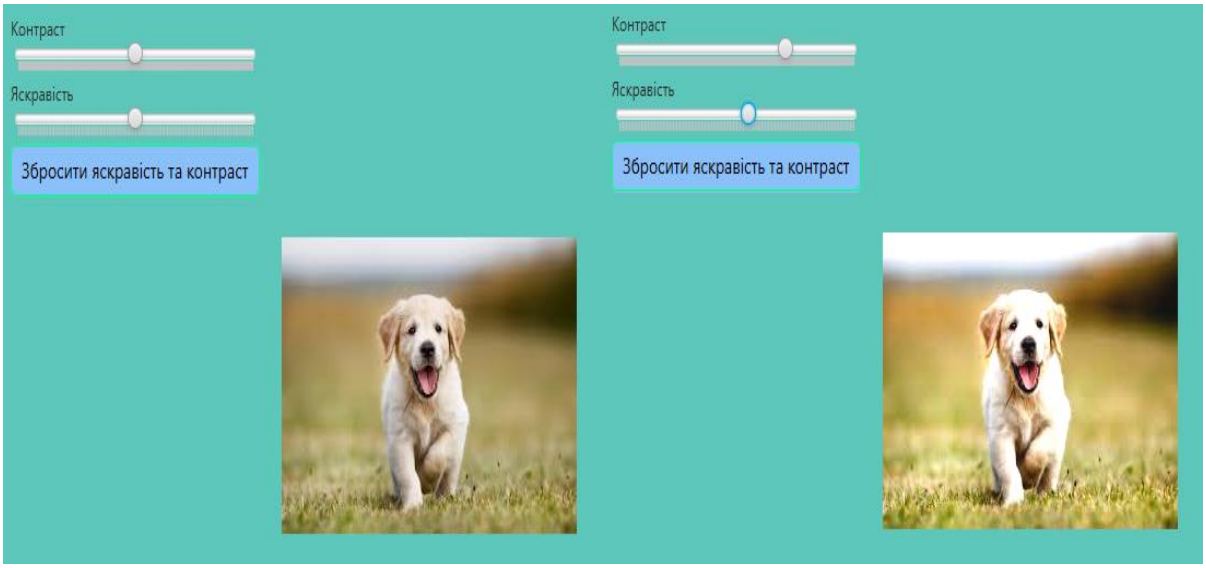
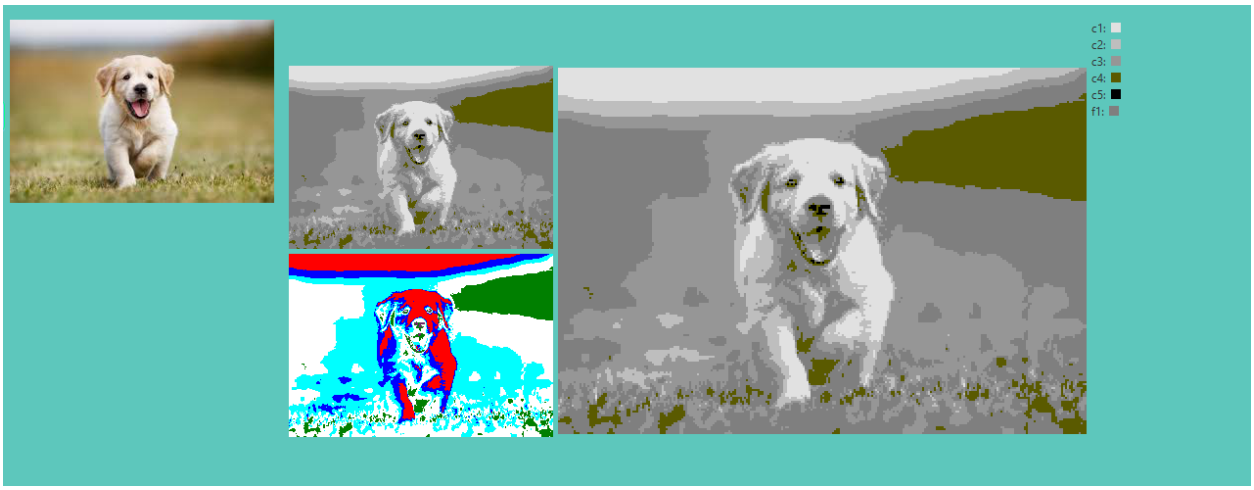


Рисунок 4.13 Приклад до та після змін контрасту та яскравості.

Після усіх змін. користувач може запустити алгоритм обробки фото натисканням на кнопку “Почати підбір



Малюнок 4.14 Приклад після прогону фото через алгоритм обробки

Перше фото це оригінал з яким працює дизайнер. Наступне фото, оброблена до допустимих кольорів зі спрощенням у градієнті. Під обробленим фото можна відразу побачити робочу схему для верстату

Якщо потрібно виключити нитководи, кількість робочих нитководів натиснувши "Вибір режима" і обравши без яких нитководів буде вишиватися носок.

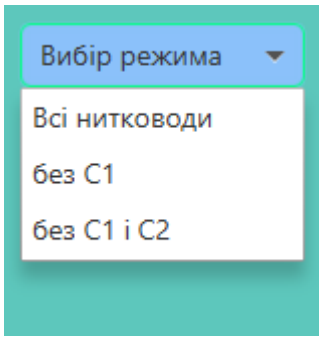


Рисунок 4.15 Викидний список для вибору режиму обробки

3. Адаптація зображення до вимог верстата

Після того, як усі зміни для обробки були задані, натисніть на "Почати підбір", після чого на екрані з'являтся 4 фото, верхнє ліве зображення до обробки, наступні 2 в стовпчик загальне фото та фото для станка, справа менш детальна візуалізація готового виробу.

P.S. (можна зразу змінити параметри та запустити нову обробку(Через використання випадкових чисел в групуванні кольорів, не змінюючи параметри, можна здобути різний результат (іноді гірший, іноді кращий).

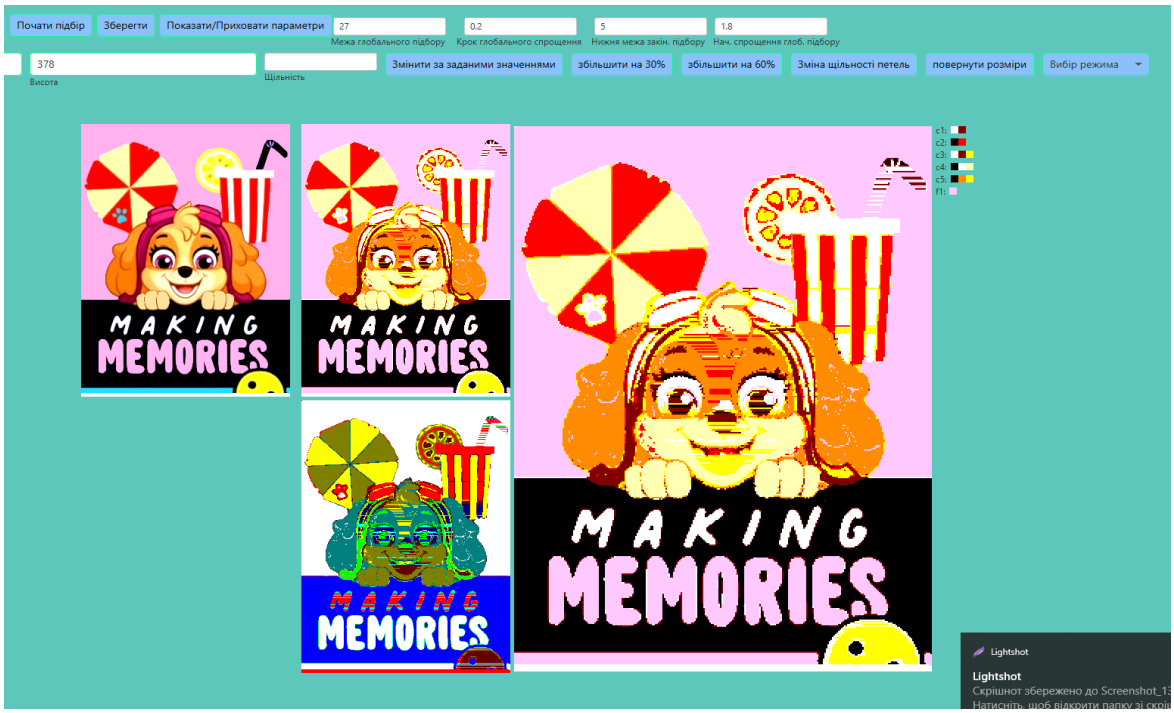


Рисунок 4.16 Приклад поганої обробки

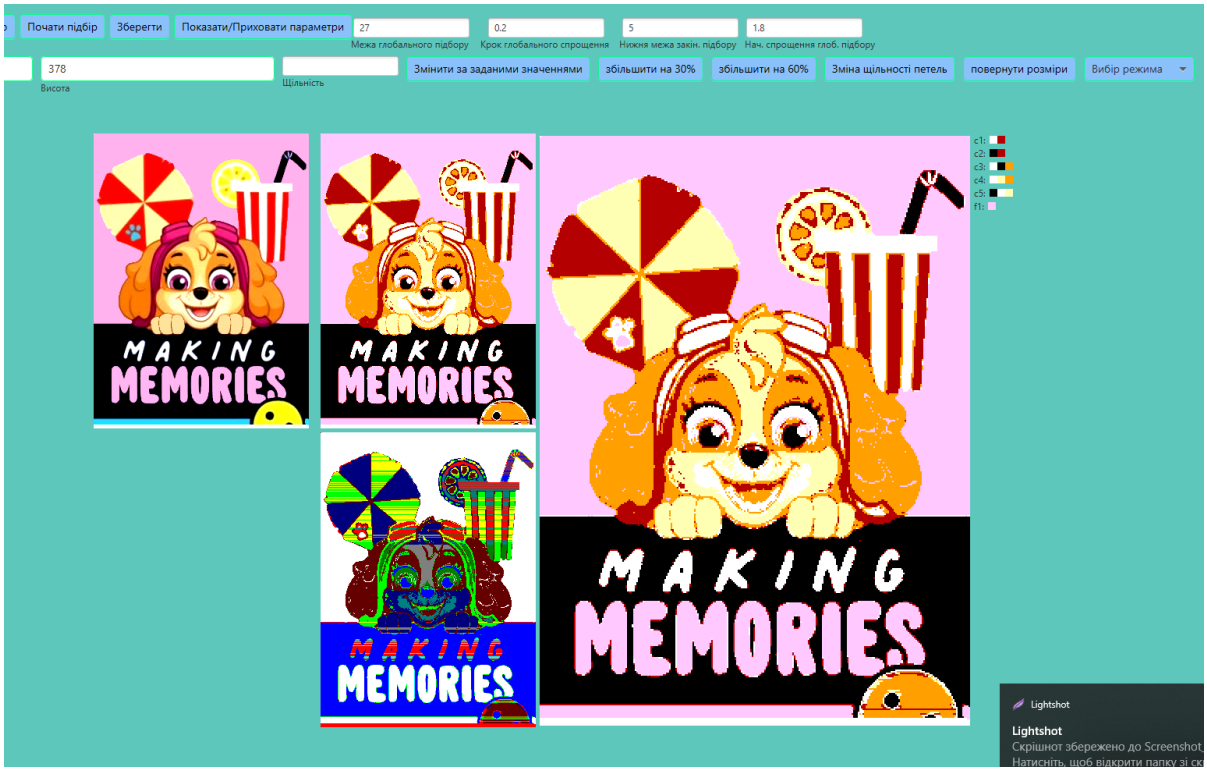


Рисунок 4.17 Приклад кращої обробки

Праворуч, від усіх попередніх фото, можна побачити збільшену фотокарточку, на якій, на перший погляд нічого не змінено. Якщо зберегти останнє фото та приблизити його, то можна побачити, що програма робить окремий візерунчатий матеріал для дизайнерів, який симулює картинку на готовому трикотажному продукту.

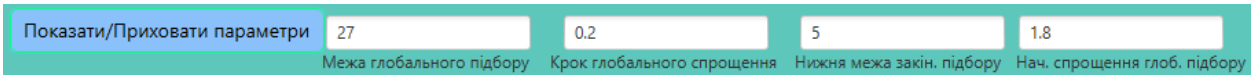


Рисунок 4.18 Параметри для більш точної обробки

Завдяки цим параметрам, обробку робочого фото можна вдосконалити не змінюючи код самої програми. Рекомендовано для користувачів, які знають як працює алгоритм обробки

Після обробки усіх трьох фото, справа від них буде панель, де показано робочі нитководи і які кольори, у якому порядку, повинно вишиватися.

Рисунок 4.19 Місце розташування змінних кольорів

Натиснувши на кольоровий квадрат, з'явиться вікно, де після натискання на кнопку "Choose color", відкриється палітра, де ви зможете змінити колір на картинці.

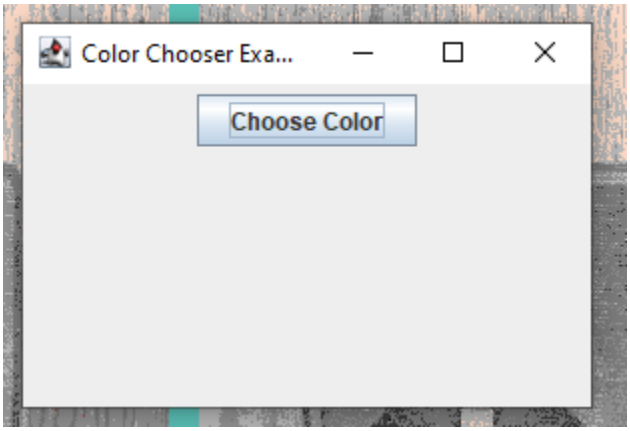


Рисунок 4.20 Вікно, яке відкривається при натисканні на кольоровий квадрат

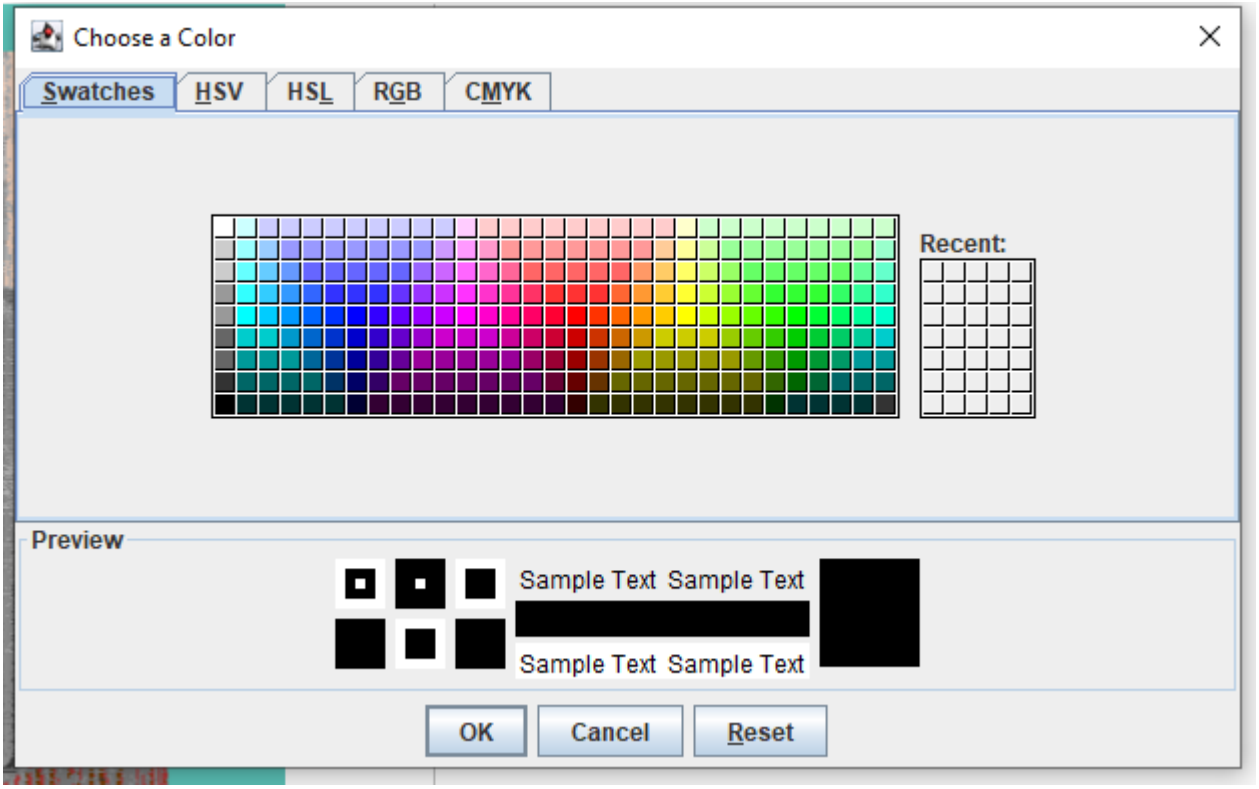


Рисунок 4.21 Вікно для зміни кольору

P.S. Не рекомендуємо закривати вікно з кнопкою "Choose color", бо тоді програма повністю закриється.

5.Збереження результату та експорт

Після обробки, змін, корегувань, результати можна завантажити з програми натиснувши "Зберегти". Процес збереження було реалізовано через такий самий спосіб, як і завантаження, через файловий провідник. Користувач може обрати які саме оброблені фото він або вона можуть зберегти у обраній папці для подальшої роботи. Для зберігання конкретного фото є можливість обрати потрібний матеріал поставивши галочки.

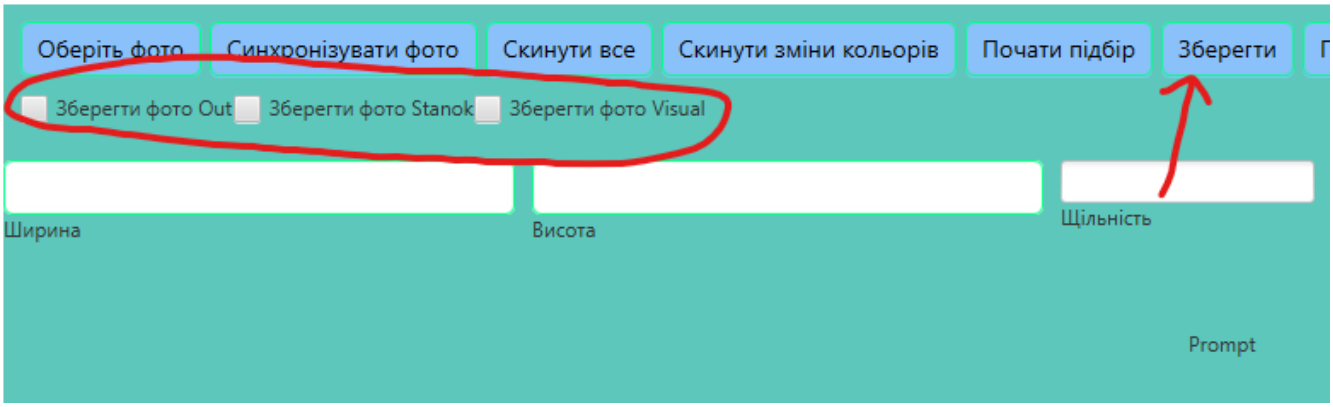


Рисунок 4.22 Розташування кнопки та галочок

Далі відкриється провідник через який обираємо місце збереження, потім в залежності від кількості, яку завантажуєте, буде відкриватися вікно провідника наново для кожного файлу.

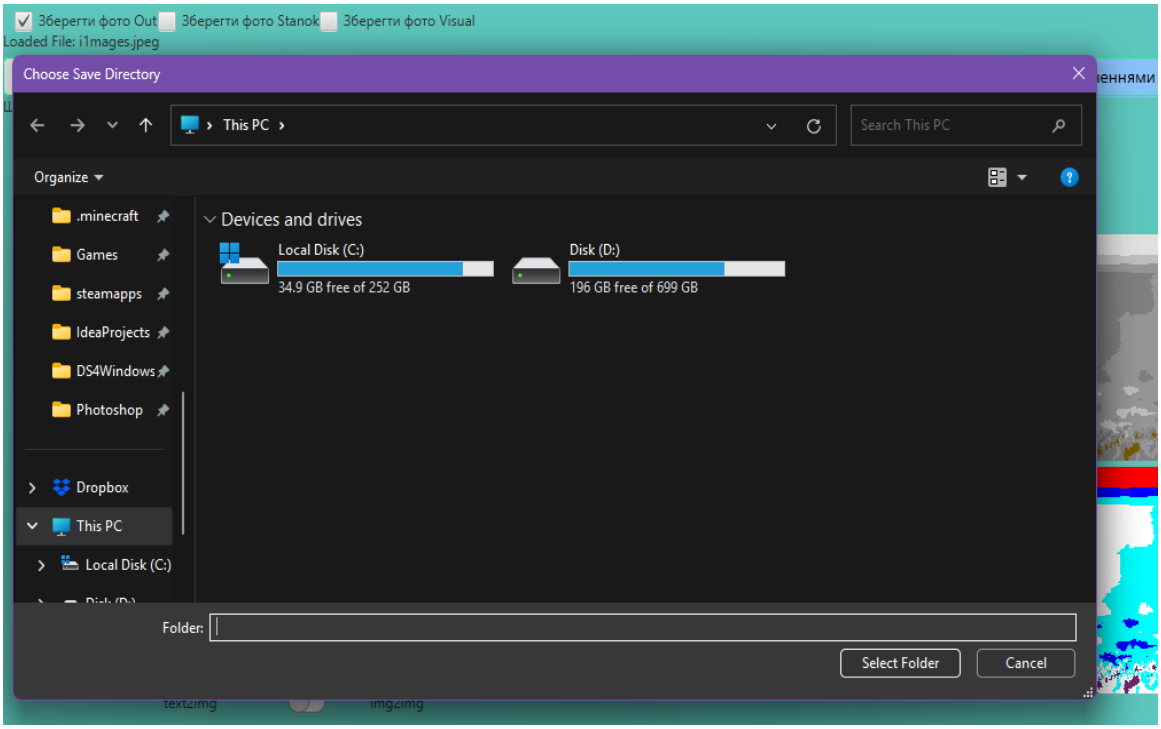
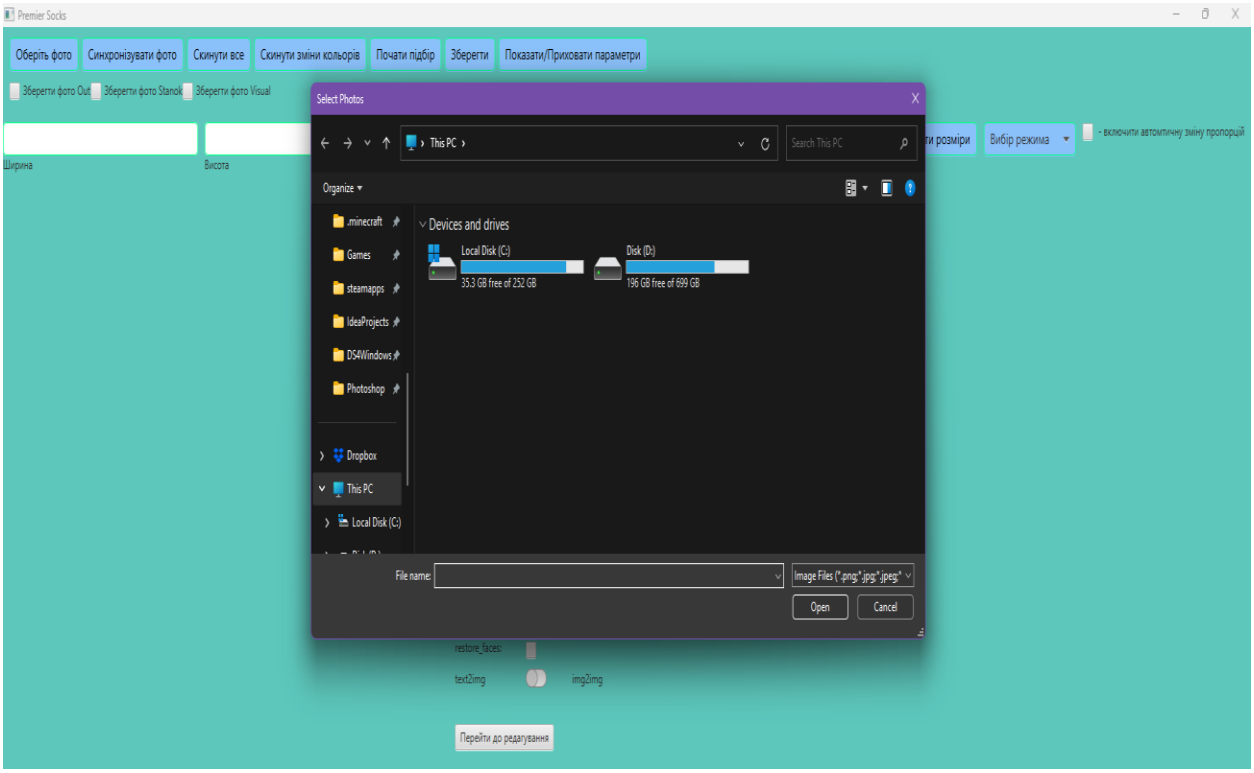


Рисунок 4.23 Вікно збереження файлу

Користувач має можливість завантажити зображення через кнопку «Оберіть файл», після чого відкривається вікно файлового провідника, де юзер може знайти фото і завантажити його у програму. Розмір самого фото буде, до зміни за допомогою програми, буде відображено 1:1.



Малюнок 4.24 Файловий провідник для завантаження робочого фото.

ВИСНОВКИ

У межах дипломного проєкту було розроблено інтерфейс спеціалізованого програмного забезпечення для підготовки зображень до друку на автоматизованих трикотажних верстатах. Основна мета полягала у створенні інтуїтивно зрозумілого, адаптивного та функціонального середовища, що відповідає потребам дизайнерів у сфері текстильного виробництва.

У процесі роботи:

- проаналізовано сучасні підходи до UX/UI-дизайну для професійного ПЗ та адаптовано їх до задач трикотажної галузі;
- реалізовано підтримку ключових операцій обробки графіки: зміна розмірів, корекція контрасту і яскравості, зменшення палітри до дозволеної кількості кольорів;
- впроваджено автоматизовану адаптацію зображень до технічних параметрів верстату з урахуванням шаблонів нитководів і обмежень на розмірність та кольоровість;
- забезпечено зручну навігацію та можливість збереження одного або декількох результатів обробки.
- для реалізації проекту було застосовано технології Java, JavaFX та FXML

Результатом є програмне забезпечення, яке спрощує робочий процес дизайнера, мінімізує потребу у перемиканні між вікнами інших ПЗ, які дизайнери використовують, та підвищує загальну ефективність виробничої підготовки.

Проєкт має практичне значення для підприємств, що використовують цифрові технології у виробництві трикотажних виробів, а також відкриває перспективи подальшого розвитку — зокрема, у напрямку розширення підтримки різних типів верстатів, інтеграції зі штучним інтелектом для генерації візерунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ольховська О.В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» / О.В. Ольховська, О.О. Черненко. – Полтава: ПУЕТ, 2022. – 71 с.

2. Аналіз інтерфейсів (Interface Analysis): Режим доступу: https://www.maxzosim.com/interface-analysis/

3. Самонюк, Т. В. Проектування інтерфейсу користувача веб-сайту з використанням функціонально-орієнтованого підходу : магістерська дис. : 122 Комп'ютерні науки / Самонюк Тетяна Вікторівна. – Київ, 2020. – 83 с.

4. Круг, С. (2014). Don't Make Me Think: A Common Sense Approach to Web Usability. New Riders.

5. Норман, Д. (2013). The Design of Everyday Things. Basic Books.

6. Маккей, Е. Н. (2013). UI is Communication: How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication. New Riders.

7. Морато, К. Р. (2021). JavaFX 17 by Example. Apress

8. Вівер, Дж., Ченг, В., Селлер, Л., та ін. (2021). Pro JavaFX 9 (оновлене видання). Apress.

9. Лоу, Д. (2022). JavaFX for Dummies (оновлене видання). For Dummies.

10. Блох, Дж. (2023). Modern JavaFX Development. O'Reilly Media.

11. Деа, К., Біті, М., та ін. (2014). JavaFX 8: Introduction by Example. Pearson Education.

12. Френкель, Н. (2015). Mastering JavaFX 8 Controls. Packt Publishing.

13. Вівір, Дж., Гао, В., та ін. (2017). Pro JavaFX 9. Apress.

14. Шаран, К. (2014). Learning JavaFX 8: Building User Experience and Interfaces with Java 8. Packt Publishing.

15. Тідвелл, Дж. (2010). Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media.

16. Гарретт, Дж. Дж. (2010). The Elements of User Experience: User-Centered Design for the Web. New Rider