МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМЕНІ С.З. ҐЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: **“ Автоматизація процесу обслуговування онлайн замовлень поліграфічного підрозділу ”**

Виконав: ст.гр. Акт-41_____

Спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології»_____
(шифр і назва)

_____Федевич Артем Олександрович_____
(Прізвище та ініціали)

Керівник: ___к.т.н., доц. Луб П.М._____
(Прізвище та ініціали)

Рецензенти: ___к.т.н., доц. Кригуль Р.Є._____
(Прізвище та ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
151 – „Автоматизація та комп'ютерно-інтегровані технології"

"ЗАТВЕРДЖУЮ"
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
"____"_____202_ р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Федевич Артем Олександрович

1. Тема роботи: «Автоматизація процесу обслуговування онлайн замовлень поліграфічного підрозділу»

Керівник роботи Луб Павло Миронович, к.т.н., доцент.
Затверджені наказом по університету від 123/к-с від 25.02.2025.

2. Строк подання студентом роботи 10.06.2025 р.

3. Початкові дані до роботи: 1. Статистичні дані виробничої галузі; 2. Каталоги обладнання для поліграфічних підрозділів; 3. Методика проектування систем автоматизації; 4. Типові схеми, технічні характеристики обладнання; 5. ДСТУ, СНіПи.

4. Зміст розрахунково-пояснювальної записки:

1. Аналіз стану питання та об'єкта проектування в контексті сучасного поліграфічного виробництва.

2. Технологічні передумови та проектування системи автоматизації.

3. Проектування та реалізація програмного забезпечення для автоматизації процесів у поліграфічному підрозділі.

4. Охорона праці та безпека в надзвичайних ситуаціях.

Висновки та пропозиції.

Список використаних джерел.

Додатки.

5. Перелік презентаційного матеріалу _____
_____
_____
_____
_____
_____
_____
_____

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	*Луб П.М., доцент кафедри інформаційних технологій*		
4	*Городецький І.М., доцент кафедри інженерної механіки*		

7. Дата видачі завдання 25.02.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Приміт-ка
1	*Написання першого розділу та означення головних завдань роботи*	*25.02.2025 – 01.04.2025*	
2	*Виконання другого розділу та формування головних показників для розрахунків*	*01.04.2025 – 01.05.2025*	
3.	*Виконання третього розділу та узагальнення отриманих результатів роботи*	*01.04.2025 – 01.05.2025*	
5.	*Написання розділу: «Охорона праці та безпека в надзвичайних ситуаціях»*	*01.05.2025 – 01.06.2025*	
7.	*Завершення оформлення кваліфікаційної роботи. Перевірка подібності тексту.*	*01.06.2025 – 09.06.2025*	
8.	*Створення презентаційного матеріалу та завершення роботи в цілому*	*09.06.2025 – 16.06.2025*	

Студент _____ Федевич А.О.
(підпис)

Керівник роботи _____ Луб П.М.
(підпис)

УДК 004.738.5:681.5:655

Автоматизація процесу обслуговування онлайн замовлень поліграфічного підрозділу. Федевич А.О. Кваліфікаційна робота. Кафедра ІТ. Дубляни, Львівський НУВМБ, 2025.

58 с. текст. част., 20 рис., 3 табл., 15 слайдів презентації, 20 літ. джерел, 2 додатки.

У кваліфікаційній роботі досліджено процес автоматизації обробки онлайн-замовлень у сфері поліграфічного виробництва. Основну увагу приділено розробці програмного забезпечення, що забезпечує автоматизовану взаємодію з електронною поштою, перевірку макетів на відповідність технічним вимогам, формування друкарських заявок, облік замовлень і клієнтів, а також інтеграцію з хмарними сервісами для збереження файлів.

Система створена з використанням мови програмування Python, фреймворку PySide6 для побудови графічного інтерфейсу та API-сервісів Gmail і Google Drive для автоматизованого отримання замовлень і збереження графічних матеріалів. Впроваджено модулі попередньої перевірки якості файлів (preflight), автоматичного формування друкарських заявок, а також структуру статусного управління кожним замовленням. Програмне забезпечення демонструє гнучкість і розширюваність, дозволяючи адаптувати систему під конкретні виробничі умови.

Практичні результати роботи засвідчують ефективність реалізованого підходу, що сприяє зменшенню ручної праці, скороченню часу на обробку запитів і покращенню контролю за якістю друкованої продукції. У рамках дослідження також розроблено розділ з охорони праці та дотримання безпеки при роботі з ІТ-інфраструктурою та автоматизованими системами.

Розроблено заходи з охорони праці та безпека в надзвичайних ситуаціях.

Ключові слова: автоматизація складу, роботизований транспортний засіб, мікроконтролер, програмно-апаратний комплекс.

Зміст

ПЕРЕДМОВА

У сучасних умовах цифрової трансформації виробництв та послуг автоматизація бізнес-процесів набуває стратегічного значення для підвищення конкурентоспроможності підприємств. Особливо це актуально для поліграфічної галузі, яка в умовах жорсткої конкуренції змушена оперативно адаптуватися до нових вимог клієнтів, зокрема в сфері онлайн-замовлень. Зростання обсягів електронного документообігу, необхідність обробки великої кількості вхідних запитів, а також потреба в персоналізації послуг формують запит на розробку спеціалізованих IT-рішень, здатних автоматизувати ключові етапи підготовки та реалізації поліграфічних замовлень.

На практиці, у поліграфічних компаніях часто спостерігається ситуація, коли робочий процес із підготовки файлів до друку виконується вручну: працівники перевіряють технічні параметри зображень (кольорову модель, роздільну здатність), змінюють масштаб, додають технічні мітки для порізки, перетворюють багатосторінкові PDF-файли в TIFF-формати, створюють заявки тощо. Така організація праці не лише уповільнює обробку замовлень, а й є джерелом людських помилок. Наявність у таких процесах рутинної ручної праці вказує на потенціал для їхньої глибокої автоматизації.

Метою кваліфікаційної роботи є – розробка моделі автоматизованого сервісу для обслуговування онлайн замовлень поліграфічного підрозділу, який інтегрує кілька ключових функцій — електронну пошту для прийому замовлень, інтерфейс попередньої обробки файлів до друку, інструменти для автоматизованого створення заявок на виконання друкарських робіт, а також підсистему розрахунку вартості замовлення залежно від матеріалів, типу друку і цінової політики, прив’язаної до конкретного клієнта.

Вибір теми зумовлений як актуальністю поставленої проблеми, так і власним практичним досвідом автора дипломної роботи. Працюючи дизайнером у поліграфічному підрозділі, автор щодня стикався з низкою типових завдань, що повторюються та не потребують творчого підходу, однак вимагають багато часу.

Власноруч реалізована тестова версія програмного модуля з обробкою графічних файлів у вигляді вкладки з функціоналом перевірки кольорової моделі, масштабування, додавання міток тощо, продемонструвала ефективність підходу і стала основою для подальшого розвитку в межах дипломної роботи.

Запропонована система має включати щонайменше три основні модулі: 1) модуль обробки вхідної кореспонденції (електронної пошти), з можливістю автоматичного розбору вкладень; 2) модуль підготовки файлів до друку, який забезпечить контроль технічних параметрів, форматування, оптимізацію макетів; 3) модуль створення заявки, в якому можна буде зазначити тип друку, матеріали, кількість копій та інші характеристики замовлення. У перспективі планується також додати функціональність з керування клієнтами та індивідуальними прайсами для них, що дасть змогу автоматично формувати точні комерційні пропозиції.

Розробка такого IT-рішення відповідає ключовим вимогам сучасного виробництва: гнучкість, масштабованість, орієнтація на користувача, мінімізація часу між отриманням замовлення та його реалізацією. У технологічному аспекті передбачається використання сучасних інструментів веб-розробки, бібліотек для обробки графічних зображень, засобів інтеграції з email-службами (наприклад, Gmail API), а також реалізація backend-логіки для обрахунку параметрів та генерації заявок.

Таким чином, дана дипломна робота має на меті не лише описати й реалізувати приклад автоматизованої системи для конкретної прикладної задачі у поліграфічній сфері, але й сформулювати загальний підхід до автоматизації повторюваних офісно-виробничих операцій із максимальним урахуванням практичних вимог та реальних кейсів з виробництва. Успішна реалізація запропонованої системи дозволить зменшити навантаження на працівників, підвищити швидкість обробки замовлень та покращити якість обслуговування клієнтів.

1. АНАЛІЗ СТАНУ ПИТАННЯ ТА ОБ'ЄКТА ПРОЕКТУВАННЯ

1.1. Основні визначення та класифікація процесів у поліграфічному підрозділі

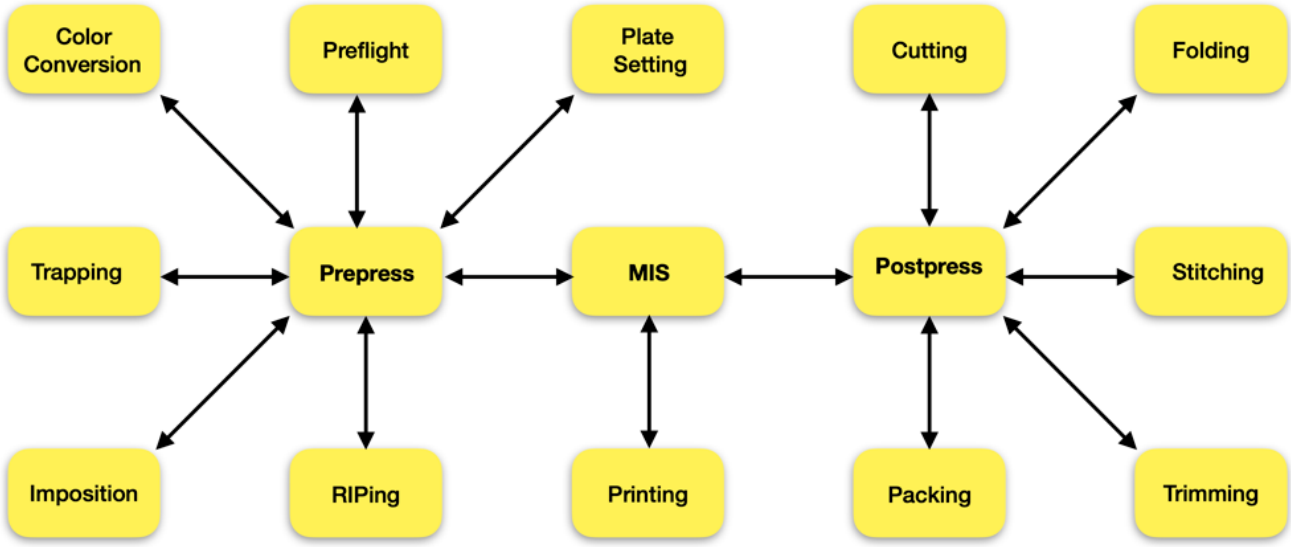
Поліграфічний підрозділ являє собою комплекс взаємопов'язаних виробничих та адміністративних процесів, необхідних для створення, обробки та реалізації друкованої продукції. В умовах цифрової трансформації економіки та стрімкого розвитку онлайн-сервісів і цифрових платформ особливого значення набуває автоматизація цих процесів, яка дозволяє суттєво оптимізувати час, витрачений на виконання типових завдань, зменшити ймовірність помилок і забезпечити високу якість кінцевого продукту, що відповідає сучасним вимогам споживачів [15].

Рисунок 1.1 – Блок-схема процесу виробництва в поліграфічному підрозділі

Виробничі процеси поліграфічного підрозділу охоплюють широкий спектр діяльності, починаючи від моменту отримання замовлення клієнта, через комплексну перевірку та підготовку файлів до друку, і завершуючи безпосередньо

виготовленням і доставкою готової продукції. Ключовим етапом у цьому циклі є перевірка файлів на відповідність важливим технічним параметрам, таким як правильна кольорова модель (наприклад, CMYK або RGB), необхідна роздільна здатність та коректний формат, що гарантує чіткість і високу якість друку [20]. Крім того, значну увагу приділяють підготовці макетів, яка включає масштабування зображень, нанесення спеціальних технічних міток для подальшої порізки, а також оформлення продукції відповідно до конкретних запитів клієнтів.

Адміністративні процеси, що супроводжують виробничу діяльність, є не менш важливими для ефективного функціонування поліграфічного підрозділу. Вони включають ведення клієнтських баз даних, регулярне формування фінансової та комерційної документації, забезпечення внутрішньої звітності та контроль за виконанням усіх поточних замовлень. Автоматизація адміністративних процесів дозволяє суттєво зменшити навантаження на персонал, мінімізувати ризики людського фактору та підвищити точність ведення документації, що безпосередньо впливає на загальну ефективність діяльності підрозділу.



За ступенем автоматизації процеси поліграфічного підрозділу умовно поділяються на три основні категорії: ручні, частково автоматизовані та повністю автоматизовані [8]. На сьогоднішній день більшість поліграфічних компаній продовжують активно використовувати ручну працю, що призводить до додаткових витрат часу, нераціонального використання людських ресурсів і, як

наслідок, підвищує ймовірність помилок через людський фактор. Часткова автоматизація, яка передбачає застосування окремих програмних інструментів для спрощення і пришвидшення типових завдань, хоча і покращує ситуацію, але не вирішує проблему повністю. Повністю інтегровані автоматизовані системи, здатні самостійно обробляти більшість етапів без активної участі людини, поки що не набули достатнього поширення, проте саме такі рішення є перспективними і необхідними для ефективного управління поліграфічним виробництвом у сучасних умовах.

Таким чином, актуальність створення автоматизованої системи для поліграфічного підрозділу полягає у нагальній необхідності оптимізації та інтеграції всього комплексу виробничих і адміністративних процесів, у зменшенні витрат часу та зусиль працівників, а також у забезпеченні високої точності, якості та ефективності роботи з клієнтськими замовленнями. Така система здатна підвищити конкурентоспроможність підприємства і адаптувати його до швидко змінюваних ринкових умов.

1.2. Обладнання та вимоги до якості робіт

Обладнання, яке використовується в поліграфічному підрозділі, відіграє ключову роль у забезпеченні високої якості готової продукції, а також впливає на швидкість виконання замовлень та загальну ефективність роботи підрозділу. Сучасний поліграфічний підрозділ оснащується широким спектром обладнання, яке включає цифрові та офсетні друкарські машини, широкоформатні плотери, обладнання для післядрукарської обробки, ламінатори, а також різні пристрої для порізки та брошурування друкованих матеріалів.

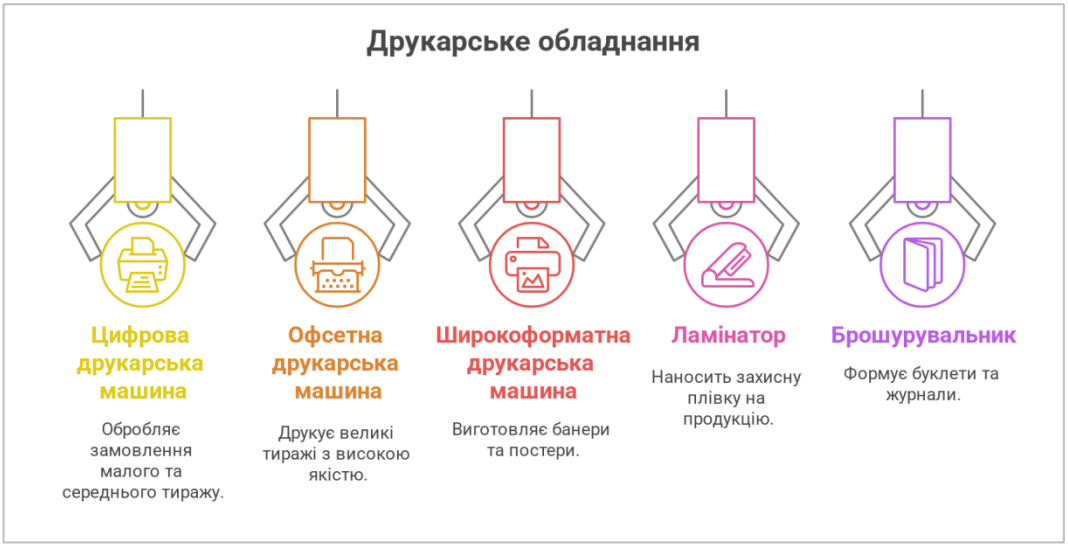


Рисунок 1.2 – Блок-схема процесу виробництва в поліграфічному підрозділі

Цифрові друкарські машини дозволяють швидко і ефективно обробляти замовлення малого і середнього тиражу з високою роздільною здатністю і точністю передачі кольорів. Офсетні друкарські машини переважно використовуються для великих тиражів, забезпечуючи економічну ефективність при збереженні високої якості друку. Широкоформатні плотери призначені для виготовлення великоформатної продукції, такої як банери, плакати, рекламні конструкції та стенди, і дозволяють забезпечити друк на різноманітних матеріалах із високою чіткістю та насиченістю кольорів.

Післядрукарська обробка є важливою складовою виробничого процесу, яка безпосередньо впливає на кінцевий вигляд продукції та її споживчі характеристики. Сюди входять операції ламінування, порізки, фальцювання, брошурування, переплетення та пакування. Для цього використовуються спеціалізовані пристрої, такі як автоматичні порізчики, ламінатори, брошурувальні апарати та переплітальні машини, які забезпечують точність і швидкість виконання цих операцій.

Особливу увагу в поліграфічному виробництві приділяють контролю якості виконання робіт [3]. Основні вимоги до якості продукції включають точність передачі кольору відповідно до затверджених макетів, правильність масштабування зображень, чіткість нанесення технічних міток і коректність

остаточних розмірів друкованої продукції. Дотримання цих стандартів гарантує високу якість кінцевого продукту і задоволеність клієнтів.

Автоматизація контролю якості може включати використання спеціалізованих програмних рішень, що забезпечують автоматичну перевірку відповідності файлів необхідним технічним параметрам перед початком друку. Такі системи дозволяють зменшити ймовірність помилок, пов'язаних з людським фактором, скоротити час на підготовку до друку та підвищити загальну ефективність виробничих процесів.

Таким чином, сучасне обладнання і чітко визначені стандарти якості робіт є необхідними умовами для забезпечення високої конкурентоспроможності поліграфічного підрозділу на ринку. Впровадження автоматизованих систем контролю якості дозволяє суттєво зменшити витрати часу на виконання замовлень, покращити якість продукції та забезпечити високий рівень задоволеності клієнтів.

1.3. Аналіз IT-сервісів у поліграфічних підрозділах

Інформаційні технології відіграють важливу роль у сучасних поліграфічних підрозділах, значно впливаючи на ефективність виробничих процесів, якість продукції та рівень обслуговування клієнтів. Використання IT-сервісів дозволяє спростити та автоматизувати ряд ключових операцій, починаючи з прийому замовлень та закінчуючи контролем якості готової продукції. В умовах інтенсивної конкуренції, що існує на ринку поліграфічних послуг, ефективні IT-рішення стають стратегічним фактором успіху підприємств.

Одним із найпоширеніших IT-рішень, які застосовуються у поліграфічних підрозділах, є системи управління виробництвом (MIS-системи) [12]. Ці системи

забезпечують інтеграцію усіх етапів виробничого циклу – від прийому замовлення, оцінки його вартості та обліку матеріалів до кінцевого друку і доставки клієнту. MIS-системи дозволяють централізовано зберігати та обробляти інформацію, автоматично формувати необхідну документацію, вести точний облік та контролювати строки виконання робіт.

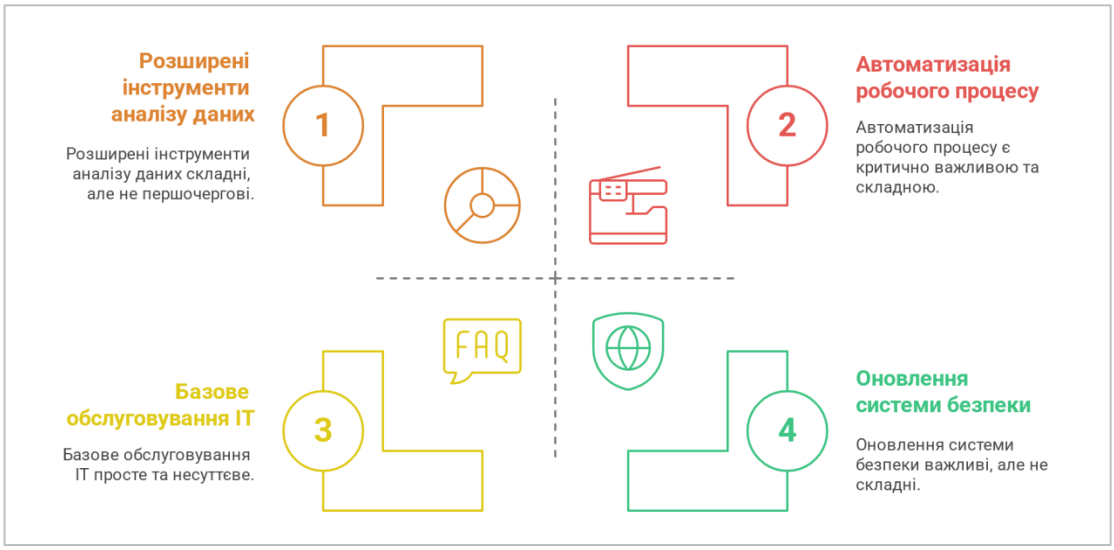


Рисунок 1.3 – Пріоритезація IT-сервісів у поліграфічних підрозділах

Важливу роль також відіграють системи автоматизованої підготовки макетів до друку, які дозволяють швидко перевіряти відповідність файлів вимогам якості, автоматично коригувати розмір, роздільну здатність та кольорову модель. Використання таких систем суттєво скорочує витрати часу на рутинні операції, зменшує ймовірність людських помилок та забезпечує високу якість кінцевої продукції.

Онлайн-платформи та сервіси замовлення продукції є ще одним важливим елементом IT-інфраструктури поліграфічних компаній. Вони дозволяють клієнтам швидко і зручно оформлювати замовлення, завантажувати файли для друку, отримувати автоматичні повідомлення про статус виконання замовлення, а також здійснювати оплату послуг. Інтеграція таких платформ з внутрішніми виробничими системами компанії дозволяє максимально ефективно обробляти замовлення та покращити комунікацію з клієнтами.

Окрім того, сучасні ІТ-рішення включають програмні продукти для автоматизації контролю якості друку. Це дозволяє зменшити вплив людського фактору на процес виробництва, забезпечуючи стабільність якості продукції, що особливо важливо для великих тиражів і повторних замовлень.

Незважаючи на значні переваги впровадження ІТ-сервісів, аналіз показує, що багато поліграфічних компаній ще недостатньо використовують потенціал сучасних технологій. Серед основних причин цього є недостатнє знання переваг автоматизації, фінансові обмеження, а також опір персоналу до змін у звичних робочих процесах.

Таким чином, подальший розвиток і впровадження ІТ-сервісів у поліграфічні підрозділи є важливим завданням, яке сприяє оптимізації виробничих процесів, підвищенню якості продукції та рівня обслуговування клієнтів, а також зміцнює конкурентоспроможність компанії на ринку.

2. СИСТЕМА АВТОМАТИЗАЦІЇ ТА ІТ-СЕРВІСИ

2.1. Автоматизовані процеси поліграфічного цеху

Поліграфічний цех — це складова частина сучасного виробничого підприємства, яка відповідає за виготовлення друкованої продукції згідно з технічними, дизайнерськими й організаційними вимогами. В умовах активного розвитку цифрових технологій дедалі більше підприємств стикаються з потребою автоматизації основних виробничих і сервісних процесів для зменшення витрат, скорочення часу обробки замовлень та підвищення якості.

Типовий цикл роботи поліграфічного підрозділу включає кілька основних етапів: прийом замовлення, перевірка та обробка файлів (prepress), друк, післядрукарська обробка, контроль якості, зберігання і видача продукції, а також супровід документації.

Першим етапом є прийом і реєстрація замовлень. За умов автоматизації клієнт має змогу самостійно оформити заявку через веб-інтерфейс, прикріпити макети, вказати параметри друку (формат, кількість, матеріал), а також одразу отримати розрахунок вартості. Це знижує навантаження на менеджера і зменшує ризик помилок при передачі інформації. Далі система автоматично передає замовлення до бази даних, де формується виробнича заявка.

Наступний важливий процес — підготовка файлів до друку. На цьому етапі автоматизація дозволяє за допомогою preflight-систем здійснити перевірку макетів: коректність формату, наявність обрізних міток, відповідність роздільної здатності, конвертація кольору в модель CMYK тощо. Системи з інтеграцією скриптів або макросів здатні автоматично масштабувати зображення, накладати технічні поля, об’єднувати кілька сторінок у друкарські аркуші тощо. Завдяки цьому підвищується швидкість обробки та зменшується кількість браку.

Третій етап — сам процес друку. Використання цифрових друкарських машин з підключенням до програмного забезпечення дозволяє автоматично отримувати технічне завдання, налаштовувати формат, розкладку та профілі друку

без участі оператора. Програмна частина MIS (Management Information System) також слідкує за завантаженням обладнання, оптимізує графік друку та враховує черговість замовлень.

Після друку продукція проходить через післядрукарську обробку. Багато операцій цього етапу також можуть бути частково автоматизовані: цифрові різаки працюють за зчитаними мітками, автоматичні фальцювальні або брошурувальні машини самі визначають обсяг і тип операції. Вся ця техніка керується програмно, і персонал виконує лише контрольну функцію.

Контроль якості є окремим важливим напрямом автоматизації. За допомогою спеціалізованих систем і сенсорів проводиться вимірювання кольору, порівняння з еталонними профілями, визначення наявності дефектів тощо. У разі відхилення система повідомляє оператора або блокує подальший друк.

Адміністративні та управлінські процеси також дедалі частіше автоматизуються. Системи обліку замовлень, клієнтів, витрат матеріалів, формування фінансових документів інтегруються в єдину платформу. Усі зміни фіксуються в історії замовлення, що полегшує аналіз ефективності виробництва та обслуговування клієнтів.

Залежно від рівня застосування цифрових технологій, автоматизацію умовно поділяють на три типи (табл. 2.1.).

Таблиця 2.1 – Типи автоматизацій виробничих технологій

Рівень	**Характеристика**
Ручний	Всі етапи здійснюються вручну; велика залежність від досвіду працівників.
Частковий	Деякі операції виконуються автоматизовано (наприклад, preflight, друк), проте основний контроль за оператором.
Повний	Замовлення обробляється в автоматичному режимі: від оформлення до друку й обліку.

Кожен рівень має свої переваги і недоліки. Найвищий рівень автоматизації дозволяє значно скоротити витрати часу, уникнути повторюваних помилок і забезпечити стабільність якості, проте вимагає значних початкових інвестицій.

Основні переваги автоматизації:

- прискорення обробки замовлень (інколи до 2–3 разів);
- стабільність і контроль якості на кожному етапі;
- мінімізація людського фактора;
- можливість масштабування та зростання без пропорційного збільшення персоналу;
- прозорість та аналітичність — можливість аналізу виробничих показників у режимі реального часу.

У підсумку, автоматизовані процеси є ключовою умовою ефективного функціонування поліграфічного підрозділу, особливо в умовах зростання кількості онлайн-замовлень. Вони дозволяють не тільки скоротити витрати і покращити якість, а й створюють передумови для довготривалого розвитку підприємства в конкурентному середовищі.

2.2. IT-сервіси для автоматизації процесів

У сучасному поліграфічному виробництві ефективність обслуговування онлайн-замовлень тісно пов’язана з впровадженням IT-сервісів, які охоплюють повний цикл — від отримання файлів замовника до формування заявок і контролю виконання. Основними напрямами автоматизації є: інтеграція електронної пошти, попередня перевірка макетів, створення електронних заявок, супровід замовлень, комунікація з клієнтом та інтеграція з хмарними сервісами.

Одним із ключових рішень є автоматизований обробник пошти, зокрема інтеграція з Gmail API. Він дозволяє програмно завантажувати нові листи і вкладення, сортувати їх та автоматично зберігати файли. Такий підхід усуває необхідність ручного моніторингу пошти, скорочує час реагування і запобігає

втраті даних. Як зазначено у фаховому джерелі, «Google Apps Script can be used to automatically export Gmail attachments to Drive without manual action» — і ця функція цілком придатна для потреб поліграфії. Більш того, такий модуль може бути налаштований на автоматичну обробку лише релевантних листів — наприклад, тих, що містять ключові слова в темі або надійшли з певних адрес клієнтів [4].

Після отримання файлів система виконує попередню перевірку (preflight): аналіз DPI, формату, кольорової моделі, масштабування, додавання міток порізки. Це суттєво зменшує кількість браку й підвищує якість підготовки до друку. Автоматизований модуль preflight дозволяє також швидко виявити файли, що не відповідають технічним вимогам, та згенерувати звіт або відповідь клієнту з інструкцією щодо виправлення. У деяких випадках можливе автоматичне масштабування або конвертація кольорової моделі.

Далі формується заявка — структурований запис про замовлення, що містить тип друку, матеріал, обробку, кількість тощо. Система не лише збирає ці дані, а й автоматично генерує документ, уникаючи помилок та дублювання. Як відзначено в аналізі MIS-систем, важливо, щоб «система примушувала менеджера пройти весь перелік технічних характеристик, щоб не пропустити жодного важливого пункту». Такий підхід дозволяє мінімізувати людський фактор та забезпечити повноту інформації для виробництва.

Також реалізується база даних зі статусами замовлень («Новий», «В роботі», «Готово», «Оплачено»). Це забезпечує контроль, звітність, прозорість роботи цеху. Менеджери можуть у будь-який момент переглянути поточний статус замовлення, додати коментарі, призначити відповідального працівника або сформувати підсумковий звіт. Автоматичні відповіді клієнтам дозволяють повідомити про отримання або завершення замовлення, зменшуючи навантаження на персонал. Наприклад, при зміні статусу на «Готово» система автоматично надсилає клієнту повідомлення з номером замовлення і запрошенням до оплати або отримання. Файли можуть зберігатися у Google Drive, що відкриває доступ до архіву та спрощує обмін великими макетами [5]. Крім того, це дозволяє організувати хмарне резервне копіювання та спільну роботу дизайнерів, менеджерів і друкарів.

У рамках десктопної програми важливе місце займає інтерфейс, реалізований на основі Qt/PySide6. Його багатовкладковість, логічна структура, таблиці замовлень та повідомлення про помилки чи успішні дії підвищують юзабіліті системи, наближуючи її до промислових рішень. Програмний інтерфейс адаптовано до реальних сценаріїв роботи в поліграфічному цеху: швидке перемикання між вкладками, попередній перегляд файлів, динамічне формування форм замовлень, інтеграція з базами даних. Підтримується візуальна індикація статусу (кольорова позначка замовлень), гарячі клавіші та система підказок для персоналу.

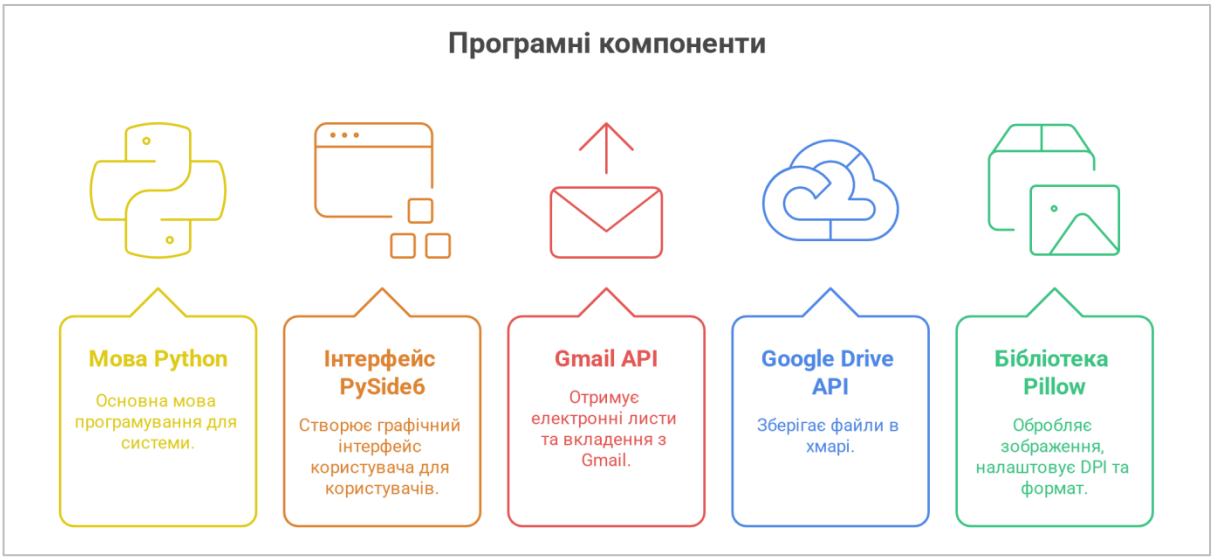


Рисунок 2.1 – Обрані технології та їх призначення

Зіставлення з PrintVis або OnPrintShop показує, що наша система поділяє ключові принципи (централізація даних, автоматизація рутинних дій, відстеження статусів), але є більш гнучкою, адаптованою до локальних потреб, швидше впроваджуваною і дешевшою. Наприклад, якщо OnPrintShop більше орієнтований на B2C-модель з інтегрованим веб-магазином і редактором шаблонів, то наша система більше підходить для внутрішньої автоматизації процесів поліграфічного виробництва [10]. Такий підхід дозволяє ефективно впровадити автоматизацію в невеликих поліграфічних підрозділах без надлишкових витрат і з урахуванням специфіки реального робочого процесу.

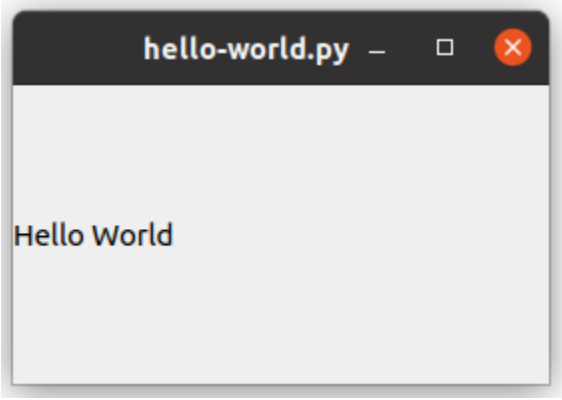
Таким чином, IT-сервіси виступають фундаментом цифрової трансформації поліграфічного підрозділу. Вони зменшують вплив людського фактора, пришвидшують обробку замовлень, підвищують прозорість роботи та дозволяють реалізовувати масштабовані й адаптивні рішення. Їхнє впровадження є не просто модернізацією, а необхідною умовою конкурентоспроможності підприємства в умовах сучасного ринку.

2.3. Обґрунтування вибору програмних засобів та інструментів розробки

Для реалізації автоматизованої системи обслуговування онлайн-замовлень поліграфічного підрозділу було обрано набір сучасних інструментів, які забезпечують стабільність, гнучкість і масштабованість рішення. Нижче розглянемо кожен із них, обґрунтуємо доцільність використання та висвітлимо ключові переваги в контексті поставленої задачі.

Python — основна мова програмування, обрана через її універсальність, високу швидкість розробки, доступність великої кількості бібліотек і чудову підтримку роботи з API, файлами та графічними інтерфейсами. Python активно використовується в галузях автоматизації, аналітики та веб-розробки, що робить його придатним для розробки комплексних програмних рішень.

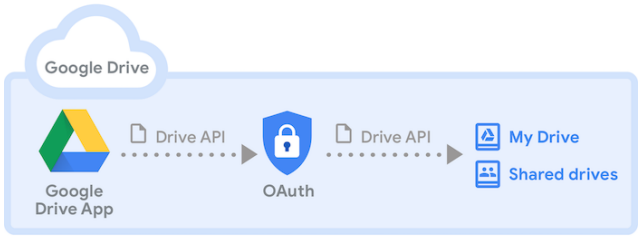
PySide6 (Qt for Python) — бібліотека для створення графічного інтерфейсу користувача. Вона забезпечує сучасний вигляд віконної програми, дозволяє будувати багатовкладкові інтерфейси, реалізовувати drag-and-drop, меню, списки, таблиці та форми. Qt відзначається стабільністю, кросплатформеністю та наявністю вбудованих засобів для роботи з подіями, що забезпечує високу інтерактивність. PySide6 дає змогу швидко розробити зручний UI з підтримкою гарячих клавіш, спливаючих повідомлень і кастомізованих



візуальних компонентів, що особливо важливо для щоденного використання персоналом поліграфії.

Gmail API — інтеграційний інструмент для взаємодії з електронною поштою. Він дозволяє автоматично отримувати вхідні листи, зчитувати метадані, завантажувати вкладення та відповідати клієнтам. Завдяки цьому досягається автоматизація першого етапу обслуговування замовлень — отримання та обробки листів без ручного втручання. Gmail API має гнучку авторизацію через OAuth2 і надає стабільний канал доступу до поштової скриньки, що гарантує безпеку та надійність процесу.

Google Drive API — інструмент, що дозволяє інтегрувати збереження файлів у хмарне сховище. Його використання забезпечує резервне копіювання, віддалений доступ до макетів, а також можливість швидко передавати клієнтам посилання на файли без використання великих вкладень. У поліграфії це актуально для зберігання PDF, зображень високої роздільності, технічних схем тощо.



Pillow — бібліотека Python для обробки зображень, яка дозволяє виконувати масштабування, зміну DPI, аналіз метаданих зображень. Її інтеграція дає змогу реалізувати preflight-модуль у програмі: визначення реальної якості макета ще до друку. Це мінімізує брак і прискорює перевірку файлів, що особливо важливо при великій кількості замовлень із різною якістю підготовки.

OS і shutil — модулі стандартної бібліотеки Python, які використовуються для роботи з файловою системою. Вони забезпечують створення папок замовлень, копіювання та переміщення файлів, побудову ієрархії зберігання. Це дає змогу впорядкувати макети замовників, зберігати історію змін і оптимізувати робочий процес у файловій структурі операційної системи.

JSON, pickle — модулі для збереження структурованих даних. JSON використовується для зберігання конфігурацій, а pickle — для серіалізації об’єктів у процесі роботи програми. Це дозволяє створювати гнучку систему збереження

стану, яка полегшує відновлення сесій, передачу даних між модулями та модифікацію програми без втрати налаштувань.

re (regular expressions) — модуль для аналізу тексту, що застосовується при обробці листів клієнтів. Наприклад, він дозволяє витягати дату замовлення, номер телефону, адресу доставки чи назву замовлення з тексту листа автоматично. Таким чином, частину введення даних у заявку виконує програма.

logging — потужний інструмент для ведення системного журналу подій у Python. Він дозволяє фіксувати ключові дії користувача, технічні помилки, інформаційні повідомлення та діагностичні дані. Це критично важливо при роботі з великою кількістю замовлень, коли необхідно контролювати стабільність програми, аналізувати помилки, що виникають у процесі виконання, і розуміти послідовність подій. Модуль logging підтримує різні рівні важливості (DEBUG, INFO, WARNING, ERROR, CRITICAL), що дозволяє гнучко налаштувати глибину логування, вести окремі журнали для кожного модуля та інтегрувати повідомлення про помилки в інтерфейс або системний звіт. Це забезпечує зручність у технічному супроводі, підвищує надійність системи та спрощує процес налагодження.

Таким чином, вибрані програмні засоби забезпечують оптимальну взаємодію між усіма модулями системи — від взаємодії з клієнтськими замовленнями до внутрішньої організації даних. Їх використання дозволило не лише реалізувати функціональність згідно з вимогами, але й забезпечити зручність розширення та супроводу системи в майбутньому. Інструменти з відкритим кодом і активною підтримкою спільноти дозволяють гнучко адаптувати систему під нові виклики та вдосконалювати її з мінімальними витратами часу і ресурсів.

2.4. Узагальнений алгоритм роботи та структура коду системи

Програмна реалізація автоматизованої системи обслуговування онлайн-замовлень у поліграфічному підрозділі побудована на модульному принципі, що забезпечує гнучкість, масштабованість і зрозумілу структуру для майбутньої

підтримки та доопрацювання. Основний програмний код реалізований у файлі main.py, де визначено як логіку роботи, так і графічний інтерфейс. Кожна функціональна частина системи винесена в окремі класи з чітким розмежуванням відповідальностей, що відповідає принципам ООП і підвищує зручність модифікації та розширення.

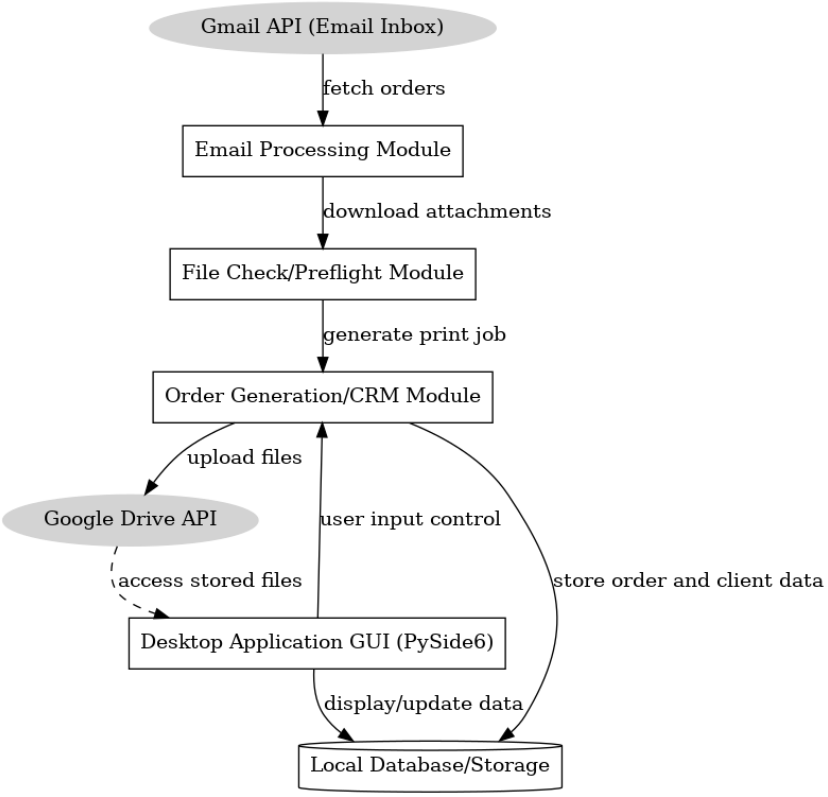


Рисунок 2.2 – Архітектура програмного забезпечення

Структура системи включає наступні логічні блоки:

Ініціалізація середовища та головного вікна — створення екземпляра MainWindow, в якому розміщені вкладки (таби) для взаємодії з поштою, заявками, обробкою файлів тощо. Для кожної вкладки реалізовано окремий клас із відповідною логікою. Здійснюється підключення сигналів, побудова початкового інтерфейсу та підготовка директорій для обробки файлів.

Робота з поштою (MailTab) — модуль, який використовує Gmail API для зчитування вхідних листів, їх попереднього перегляду та завантаження вкладень. Включає обробку MIME-повідомлень, розбір структури листа, збереження метаданих (тема, дата, відправник), підтримку сортування за датами та темами. Забезпечено динамічне оновлення таблиці та прев’ю вмісту листа в окремій панелі.

Також реалізовано обробку вкладень з поділом на формати та винесенням у відповідні папки.

Обробка файлів (FileProcessingTab) — відповідає за перегляд і попередню обробку зображень. Підключає бібліотеку Pillow для аналізу DPI, зміни роздільної здатності, зчитування EXIF-даних. Реалізовано функції масової обробки, фільтрації за параметрами, збереження результатів у нові папки, сортування файлів за критеріями, перевірку на відповідність до стандартів друку. Також у цьому модулі реалізовано систему попереднього перегляду (thumbnail preview) та оновлення параметрів при виборі файлу.

Робота із замовленнями (RequestsTab) — забезпечує створення нових заявок, автоматичне генерування назв папок за датою і шаблоном, копіювання файлів, прив'язку до клієнта й дати. Заявка формується на основі вкладень та параметрів, що витягуються автоматично з тексту листа або зазначаються вручну. Передбачена можливість редагування параметрів та підвантаження історії.

Журналювання та діагностика — у коді активно використовується модуль logging, який фіксує всі ключові події: отримання листів, створення заявок, помилки, дії користувача. Це дозволяє вести журнал подій у вигляді лог-файлів, полегшує виявлення помилок, дає змогу проводити аудит дій оператора в разі потреби. Окремі повідомлення логування також можуть відображатись у вікні користувача.

Інтерфейс користувача (UI) — реалізований засобами PySide6 (Qt), включає багатовкладкову панель, контекстні меню, діалогові вікна, таблиці з даними, прев'ю файлів. Програмно реалізовано систему навігації, повідомлення про помилки, підтвердження дій, фільтри за датами й замовниками. Структура інтерфейсу побудована відповідно до принципів зручності користування (UX): розділення функціоналу за вкладками, адаптивні повідомлення, кольорове маркування статусів.

Узагальнений алгоритм роботи системи виглядає наступним чином:

1. Запуск програми → ініціалізація головного вікна, вкладок і середовища.

2. Перехід у вкладку «Пошта» → підключення до Gmail API, зчитування листів.

3. Завантаження вхідних повідомлень → відображення в таблиці, сортування, вибір.

4. Перегляд вмісту листа → завантаження вкладень, збереження до тимчасової папки.

5. Перехід у вкладку «Файли» → попередній перегляд зображень, аналіз параметрів, перевірка DPI.

6. Вибір параметрів обробки → зміна роздільної здатності, збереження результатів.

7. Створення заявки у вкладці «Заявки» → копіювання даних, автоматичне формування структури папки.

8. Завершення циклу → запис логів, повідомлення користувача, оновлення таблиць.

Рисунок 2.3 – Узагальнений алгоритм роботи системи

Код побудовано з урахуванням інкапсуляції, багато логіки розділено на окремі функції з чіткими назвами, винесено в окремі методи класів. Це забезпечує високу підтримуваність, дозволяє ізолювати помилки, полегшує рефакторинг.

Використання перевірених бібліотек (requests, Pillow, PySide6) гарантує стабільність та відповідність сучасним стандартам Python-розробки.

Система не тільки автоматизує рутинні дії, але й дозволяє зменшити кількість помилок при роботі з великим потоком замовлень. Вона скорочує час взаємодії з клієнтом, мінімізує людський фактор і забезпечує надійну архівацію даних. Така архітектура дозволяє гнучко адаптуватися до змін — додати нові вкладки, типи обробки, розширити інтеграцію з іншими сервісами (наприклад, Telegram Bot API, бази даних, CRM), не порушуючи цілісності структури.

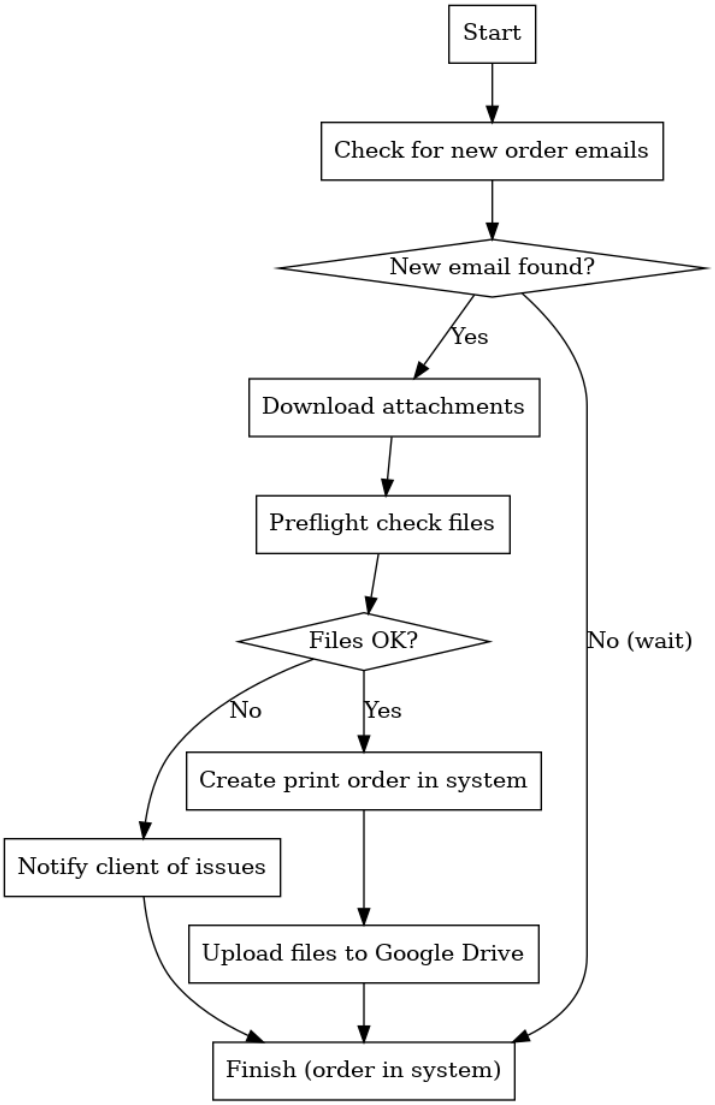


Рисунок 2.4 – Узагальнений алгоритм роботи системи в блок-схемі

3. ПРОЕКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ОБСЛУГОВУВАННЯ ОНЛАЙН ЗАМОВЛЕНЬ ПОЛІГРАФІЧНОГО ПІДРОЗДІЛУ

3.1. Опис головних елементів IT-сервісу програми

Процес обслуговування онлайн-замовлень у поліграфічному підрозділі складається з послідовності етапів, які вимагають як високої точності виконання, так і чіткого контролю. Відповідно, створювана система повинна охоплювати всі ключові компоненти — від прийому замовлень до створення заявки й формування технічного завдання на друк. Для цього в системі було реалізовано набір IT-сервісів, кожен із яких відповідає за окремий логічний блок обробки.

1. Модуль прийому вхідної кореспонденції (інтеграція з Gmail API). Першим і критично важливим елементом IT-сервісу є модуль взаємодії з електронною поштою. Згідно з практикою поліграфічних підрозділів, значна частина замовлень надходить саме через email — клієнти надсилають файли у вкладеннях листів, супроводжуючи їх коротким технічним описом. Традиційна модель роботи передбачає ручний моніторинг пошти та збереження файлів у відповідні директорії, що створює загрозу втрати даних або пропуску важливих листів.

У межах розробленої системи цей процес автоматизовано. За допомогою Gmail API програма отримує доступ до поштової скриньки, аналізує вхідні повідомлення, зчитує вкладення, зберігає файли у структуру директорій, а також виділяє метадані (тема, дата надходження, ім’я відправника). Водночас виконується попередня фільтрація листів: обробляються лише ті, що містять ключові слова або походять від визначених клієнтів.

Як зазначає технічна документація Google, *«Google Apps Script can be used to automatically export Gmail attachments to Drive without manual action»*, і ця технологія була адаптована до потреб поліграфії.

2. Модуль попередньої обробки файлів (preflight-контроль). Другим за важливістю є блок, що відповідає за перевірку відповідності графічних макетів

технічним вимогам. У поліграфічному виробництві особливу роль відіграє якість вихідного файлу — кольорова модель, роздільна здатність, наявність обрізних міток тощо. Будь-яка невідповідність може призвести до браку, затримок і додаткових витрат.

Для вирішення цієї задачі було реалізовано модуль попередньої обробки зображень на основі бібліотеки Pillow. Він дозволяє:

- перевірити DPI кожного макета;
- ідентифікувати кольорову модель (RGB чи CMYK);
- здійснити масштабування за вказаним коефіцієнтом;
- додати технічні мітки для порізки (наприклад, кутики);
- розбити багатосторінкові PDF на окремі TIFF-формати, зручні для друку.

У разі виявлення невідповідностей система генерує повідомлення або лог-файл, який інформує оператора про необхідні дії. Таким чином, ручна робота дизайнерів зводиться до мінімуму, а якість макетів підвищується ще до друку.

3. Генератор заявок і облік параметрів замовлень. Наступним функціональним компонентом є генератор заявок — система, що автоматизує формування внутрішніх документів на друк. Цей модуль отримує частину параметрів автоматично (наприклад, кількість копій, розміри, тип макету), а решту вводить користувач вручну (тип друку, матеріал, ламінування, клієнт тощо).

Сформована заявка зберігається у структурованій формі: як у вигляді локального документа, так і в базі даних. Автоматизована форма не дозволяє пропустити важливі пункти — реалізована система валідації та підказок. Це суттєво підвищує точність введення та спрощує роботу менеджера з виробництва. Як зауважено в дослідженнях MIS-систем: *«Система примушує менеджера пройти весь перелік технічних характеристик, щоб не пропустити жодного важливого пункту»*.

4. Модуль обліку клієнтів і прайсових політик. Ураховуючи, що в реальних умовах обслуговування клієнтів передбачає гнучкі ціни, у систему інтегровано модуль обліку прайсів. Кожному клієнту можна призначити окрему цінову

політику, яка впливає на автоматичний розрахунок вартості при створенні заявки. Збереження таких даних у структурованому форматі (наприклад, у JSON) дозволяє:

- формувати персоналізовані комерційні пропозиції;
- автоматично обраховувати вартість друку;
- контролювати історію взаємодії з клієнтами;
- прискорити повторне оформлення типових замовлень.

5. Статусна база замовлень. Щоб забезпечити повний контроль за виконанням замовлень, система реалізує таблицю станів: «Новий», «В обробці», «Готово», «Оплачено». Кожна зміна статусу фіксується і може бути супроводжена автоматичним повідомленням клієнту на електронну пошту.

Це дозволяє не тільки прозоро координувати процес у межах цеху, а й підвищити рівень обслуговування замовника. Система також підтримує історію змін, коментарі, призначення відповідального оператора.

6. Інтерфейс користувача (UI). Графічний інтерфейс реалізовано засобами PySide6 на основі бібліотеки Qt. Його особливістю є багатовкладкова структура, яка повторює логіку виробничого процесу. Наприклад, окрема вкладка — для пошти, інша — для обробки макетів, ще одна — для створення заявки. Переваги:

- швидкий перехід між етапами обробки;
- інтуїтивна навігація;
- візуальні індикатори стану (колірні мітки);
- таблиці з даними, прев’ю файлів, діалогові вікна.

Завдяки адаптації до робочого сценарію реального поліграфічного цеху, інтерфейс не перевантажений, але забезпечує всю необхідну функціональність [13].

7. Логування та технічна діагностика. Усі події в системі фіксуються за допомогою модуля logging: запуск програми, помилки при з’єднанні, збереження файлів, створення заявок тощо. Це дозволяє відстежувати послідовність дій, знаходити причини збоїв та проводити аудит взаємодії користувача із системою.

8. Хмарне сховище Google Drive.

Фінальним компонентом є інтеграція з Google Drive, яка дозволяє зберігати макети, архіви заявок та лог-файли у хмарному середовищі. Це дає змогу:

- створити централізоване сховище з правами доступу;
- спростити спільну роботу між працівниками;
- уникнути втрати локальних даних;
- організувати резервне копіювання.

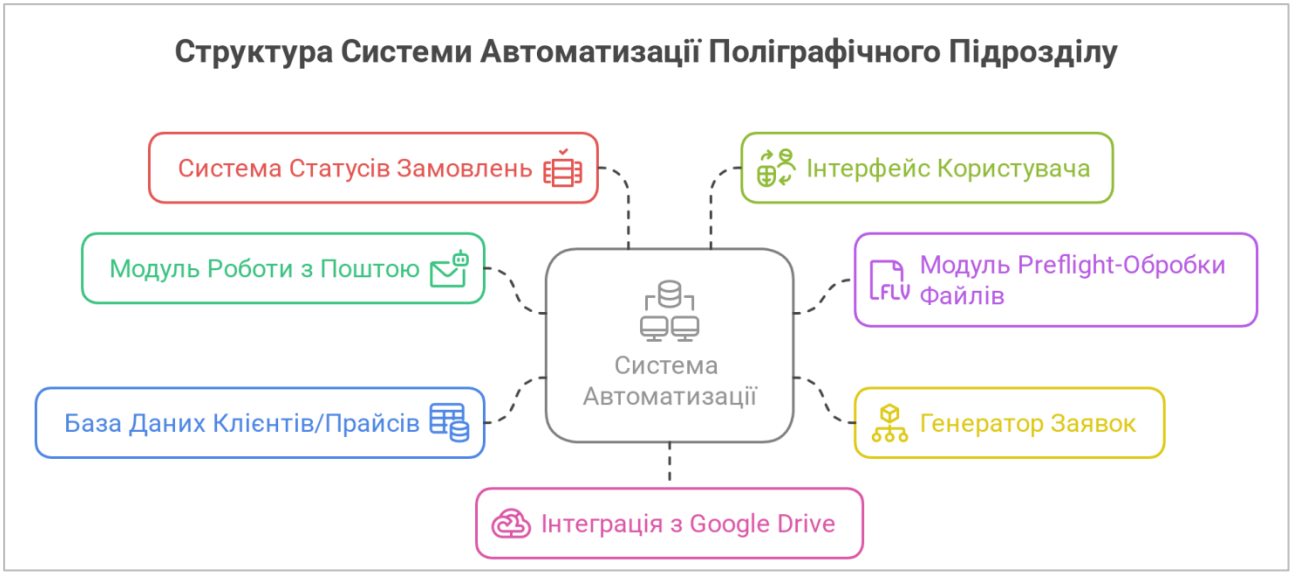


Рисунок 3.1 – Структурна схема архітектури розробленої системи

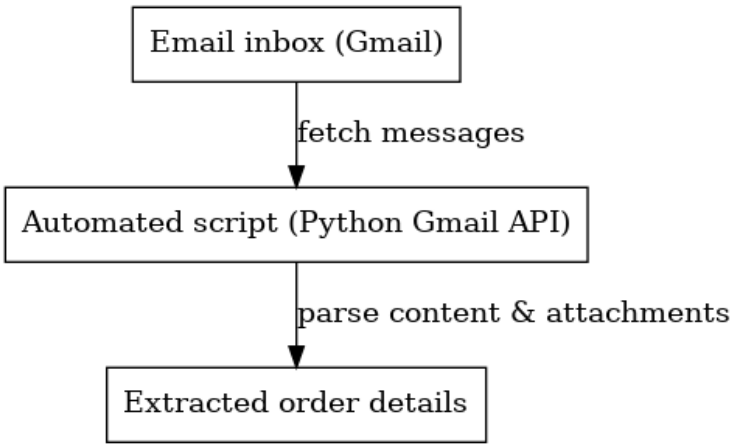
3.2. Розробка функціональності та опис коду

Програмна реалізація автоматизованої системи обслуговування онлайн-замовлень поліграфічного підрозділу здійснювалася на мові Python із застосуванням парадигми об’єктно-орієнтованого програмування, що дозволило побудувати архітектуру, придатну для масштабування, розширення та підтримки в майбутньому. У центрі системи перебуває головний графічний інтерфейс, реалізований через бібліотеку PySide6 — це офіційна обгортка до Qt для Python, яка поєднує гнучкість Python із надійністю та функціональністю одного з найпотужніших GUI-фреймворків.

Вибір саме PySide6, а не, наприклад, популярного tkinter, був обумовлений насамперед його здатністю будувати промислові багатовіконні інтерфейси з підтримкою вкладок, сигналів, подій, асинхронної взаємодії та сучасного стилю оформлення. Інтерфейс системи поділяється на логічні вкладки, кожна з яких реалізує окрему частину бізнес-логіки — зокрема, роботу з поштою, обробку файлів і створення заявок.

На рівні коду вся логіка інкапсульована в окремі класи, що відповідає за принципом Single Responsibility. Клас MainWindow виконує роль центрального диспетчера інтерфейсу: він створює екземпляри вкладок, відповідає за навігацію між ними, а також координує ініціалізацію файлової структури. Наприклад, при запуску програми формується набір директорій, у яких тимчасово зберігаються завантажені вкладення, оброблені файли та лог-файли.

Модуль роботи з поштою (MailTab) використовує API Gmail через сторонні бібліотеки google-auth, google-api-python-client і base64. Під час ініціалізації виконується авторизація через OAuth2 — саме цей протокол забезпечує захищене підключення до скриньки без збереження паролів або прямих доступів. Після авторизації програма надсилає запит до сервісу users().messages().list(), обмежуючи результати за темою або відправником, якщо користувач вказав відповідні фільтри. Важливою перевагою такого підходу є мінімізація трафіку та швидка обробка великої кількості листів без їх повного завантаження.



Для кожного листа виконується парсинг MIME-структури, після чого відбувається витягування вкладень. Код реалізовано таким чином, щоб автоматично ідентифікувати тип кожного файлу, наприклад, application/pdf, image/jpeg, image/tiff, і на основі цього переміщати їх до відповідних тек. Це

дозволяє уникнути хаотичного зберігання, забезпечити порядок і зручність подальшої обробки.

Особливу увагу приділено модулю обробки файлів (FileProcessingTab). У ньому використовується бібліотека Pillow, яка фактично є сучасним стандартом у Python для роботи з растровими зображеннями. Хоча на ринку є альтернативи, як-от OpenCV, саме Pillow дозволяє виконувати дрібні, але критично важливі для поліграфії операції — зчитування DPI, збереження у форматах TIFF, накладання міток, зміна кольорової моделі — без потреби в надлишковій обробці або глибокому аналізі сцен, що властиво OpenCV [11].

Алгоритм обробки зображень у системі має кілька стадій. На першій — Pillow відкриває зображення, зчитує його метадані, зокрема EXIF-дані, які часто містять інформацію про роздільну здатність. Якщо DPI нижче за вказане значення, програма пропонує користувачеві збільшити його або масштабувати зображення у відсотках. Масштабування здійснюється методом resize(), із використанням фільтра згладжування Image.ANTIALIAS, що забезпечує високу якість навіть після зміни розмірів.

Окремо варто згадати функцію розбиття PDF-файлів на окремі сторінки у форматі TIFF. Для цього використовуються засоби PyMuPDF (fitz), які дозволяють не тільки відкривати багатосторінкові документи, але й вивантажувати кожну сторінку у вигляді растрового зображення з налаштованою роздільною здатністю. На відміну від деяких рішень на кшталт pdf2image, бібліотека fitz працює швидше і краще підтримує CMYK-моделі, що критично важливо для поліграфії.

У модулі заявок (RequestsTab) реалізовано механізм генерації структурованих папок і документів. Назви папок формуються динамічно за шаблоном: клієнт + дата + ID замовлення. Усі супутні дані (тип друку, матеріал, кількість тощо) або витягуються з листа за допомогою регулярних виразів (модуль re), або вводяться вручну через форму. Код містить механізм валідації: жодне поле не може бути порожнім, якщо воно визначене як обов’язкове. Це реалізовано через зв’язку подій сигналів Qt (QLineEdit.textChanged) і перевірок перед створенням заявки.

Для збереження та відновлення станів програма використовує pickle — це зручно, коли необхідно зберігати не тільки значення, а й цілі об’єкти Python, наприклад словники з параметрами. У випадку, якщо потрібно було б забезпечити сумісність із зовнішніми системами, перевага могла б надаватись JSON, але для внутрішнього збереження конфігурацій pickle є оптимальним, бо не вимагає трансформації типів даних.

Ще одним критичним компонентом є модуль логування. У коді він ініціалізується в окремому конфігураційному класі, що дозволяє створювати кілька потоків логування — наприклад, один для системних повідомлень (INFO), другий для помилок (ERROR), а третій — для взаємодії з користувачем (DEBUG). Записи в логах містять точний timestamp, назву модуля, повідомлення та, в разі помилок, traceback, що робить діагностику в разі збою швидкою та ефективною. У разі потреби лог-файли можна архівувати та передавати в службу техпідтримки або адміністратору.

Для структурування коду було використано не тільки інкапсуляцію, а й принцип розділення залежностей. Замість того, щоб створювати залежність між модулями напряму, використовуються сервісні класи або сигнали Qt. Наприклад, коли користувач переходить від пошти до обробки зображень, файли передаються не напряму, а через події та внутрішню структуру даних, що зберігається у контролері. Це дозволяє уникнути жорстких зв’язків між модулями і робить систему придатною для рефакторингу.

Серед ключових переваг використаної архітектури — асинхронна обробка. Наприклад, завантаження великої кількості листів або зображень не блокує головний потік інтерфейсу, оскільки винесене в окремий QThread. У коді створено відповідні класи-нащадки, де перевизначено метод run(), у якому виконується довготривала задача. Таким чином, користувач може взаємодіяти з інтерфейсом навіть під час фонових операцій, що підвищує зручність використання програми.

Окремо варто згадати взаємодію з Google Drive API. Для збереження файлів і надання спільного доступу клієнтам використовуються запити на завантаження з авторизацією через токени доступу. Створення директорій, переміщення файлів,

генерація публічних посилань реалізовані через інтерфейс drive_service.files().create(), де параметри конфігурації задаються через JSON-структури. Система дозволяє навіть візуально відображати, які файли збережено локально, а які — у хмарі, шляхом кольорової індикації в таблиці інтерфейсу.

У підсумку, вся система демонструє високий ступінь модульності, зрозумілу структуру, а головне — відповідність потребам поліграфічного підрозділу в реальних умовах роботи. Розроблена функціональність дозволяє ефективно автоматизувати критично важливі етапи — прийом файлів, перевірку їх якості, формування замовлення, створення папок і документації, комунікацію з клієнтом. При цьому використання перевірених бібліотек з відкритим кодом гарантує підтримку, стабільність та можливість доопрацювання у майбутньому.

Продовжуючи розгорнутий опис коду, важливо деталізувати внутрішню реалізацію ключових функціональних механізмів, які формують логіку обробки замовлень, роботи з файлами та взаємодії з користувачем у рамках автоматизованої системи.

Одним із найважливіших елементів є динамічне оновлення даних у GUI. У кожній вкладці, наприклад у MailTab, передбачено регулярне оновлення таблиці листів. Це не реалізовано як пасивне завантаження при запуску, а як функція, яку можна викликати вручну або через таймер Qt (QTimer). Такий підхід дозволяє зменшити навантаження на мережу та надати користувачу контроль над частотою оновлень. Метод updateMailTable() перебирає отримані листи, будує для кожного рядок із датою, темою, іменем відправника і додає його в QTableWidget. Для зручності реалізовано сортування — користувач може клікнути на заголовок стовпця і впорядкувати листи за датою, що спрощує роботу з великим обсягом вхідних повідомлень.

Щодо збереження вкладень, то застосовується стратегія обробки з попередньою перевіркою існування файлу. Тобто, якщо вкладення вже збережене, програма не дублює його, а порівнює розміри файлів і лише у випадку відмінностей перезаписує. Це реалізовано через модуль os.path — використовується функція os.path.exists() у парі з os.stat() для отримання метаданих.

Особливий акцент у коді поставлено на реактивність взаємодії з користувачем. Наприклад, при відкритті вкладки з обробкою файлів (FileProcessingTab), система автоматично сканує відповідну директорію, завантажує зображення і будує прев'ю у вигляді мініатюр (thumbnail preview). Це реалізовано через клас QGraphicsScene і об'єкти QPixmap, що генеруються з оброблених зображень. Користувач може натискати на прев'ю — відповідна дія обробляється через сигнал itemClicked, що дозволяє миттєво змінювати параметри конкретного файлу без перезавантаження інтерфейсу. У такий спосіб реалізовано максимально плавну та зручну навігацію у великій кількості макетів.

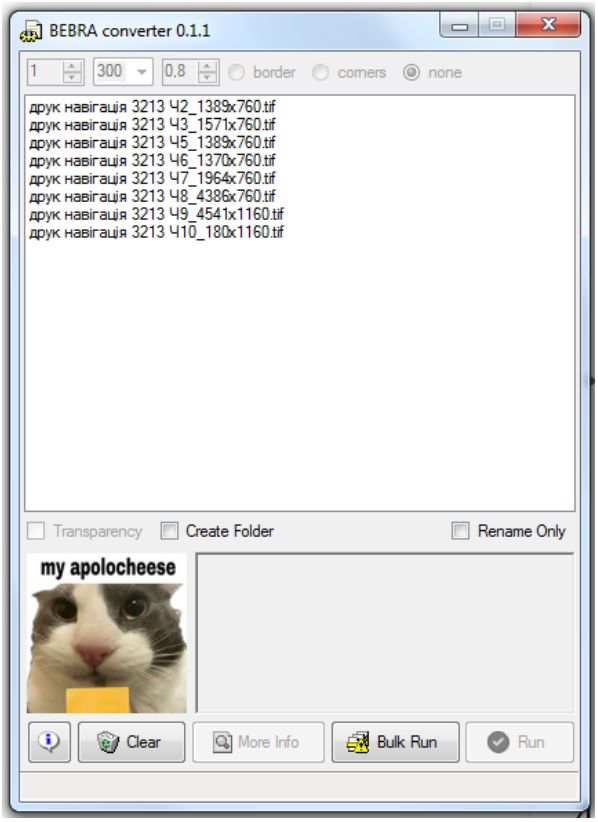


Рисунок 3.2 – Перший прототип програми

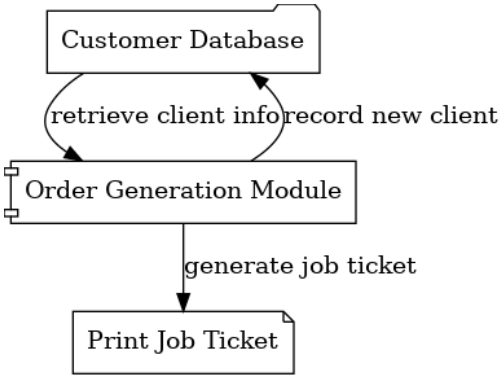
Для виконання дій над файлами — зміна DPI, масштабування, додавання міток — використовуються окремі кнопки з підключеними слотами. При натисканні, наприклад, на кнопку "Змінити DPI", програма відкриває діалогове вікно, у якому користувач вводить потрібне значення, після чого Pillow повторно відкриває файл, перераховує його розмір згідно нового DPI, і зберігає копію у папку processed/. Цей процес логічно інкапсульований у метод adjust_dpi(image_path,

target_dpi), що спрощує використання функціоналу у майбутньому та дозволяє повторно його застосовувати в інших вкладках або сценаріях.

Важливою деталлю є управління файловою ієрархією, яка в коді представлена у вигляді структурованого дерева директорій. Кожен новий клієнт або заявка створюється у вигляді окремої папки, а всі пов'язані з нею файли — в підкаталогах типу original, processed, tiff, preview. Таку ієрархію легко реалізувати за допомогою os.makedirs() із прапором exist_ok=True, що гарантує ідемпотентність операції (папка не буде створена двічі). Таке структурування дозволяє легко здійснювати резервне копіювання, інтеграцію з Google Drive і забезпечує зрозумілу логіку архівації.

Не менш важливою є реалізація механізму генерації заявки. В інтерфейсі користувач заповнює форму, а дані передаються у Python-словник, який серіалізується у формат JSON і зберігається як request.json у відповідну директорію заявки. Це дозволяє легко використовувати заявку як конфігураційний файл для майбутніх обчислень (наприклад, для модулю калькуляції вартості друку). Окрім того, система додає туди службову інформацію — дату створення, авторизаційний токен користувача, ідентифікатор клієнта. Такий підхід відповідає найкращим практикам — розділення представлення (інтерфейс) і бізнес-логіки (об'єктна модель заявки).

Щоб система не просто обробляла файли, а дозволяла відслідковувати їх статус, у коді реалізовано окремий модуль OrderManager, який працює як внутрішня



БД (реалізована через shelve або sqlite3, залежно від конфігурації). У ньому зберігається список усіх замовлень, їхній статус, лог подій і прив'язані файли. Кожна дія в інтерфейсі (зміна статусу, збереження, друк) викликає відповідний метод, який оновлює базу. Це дозволяє відновити стан роботи після аварійного завершення програми.

Інтеграція з хмарним сховищем (Google Drive API) реалізована через OAuth2 авторизацію. У коді передбачена обробка токенів доступу, автоматичне оновлення

access-token, перевірка авторизації та створення токена при першому запуску. Метод upload_to_drive(filepath, folder_id) відповідає за завантаження файлу в потрібну директорію, після чого повертає публічне посилання, яке можна вставити в повідомлення клієнту. Ця логіка ізольована у власному сервісі DriveUploader, що робить код легко тестованим і переносимим.

У цілому, система спроектована згідно з архітектурним підходом MVC (Model-View-Controller), де:

- Model — це дані замовлення, заявки, параметри зображень;
- View — інтерфейс на Qt (таблиці, вікна, форми);
- Controller — логіка обробки подій, генерація запитів, керування файлами.

Кожен компонент максимально ізольовано, що спрощує тестування, супровід і можливе масштабування в майбутньому (наприклад, через додавання інтеграції з Telegram API, Print MIS, CRM або навіть бухгалтерськими модулями).

3.3. Відображення процесу обслуговування онлайн замовлень поліграфічного цеху

Автоматизована система, розроблена в межах цієї дипломної роботи, покликана максимально точно відтворити повний цикл обслуговування онлайн-замовлень у поліграфічному підрозділі. Основна ідея полягає у тому, щоби перенести звичну послідовність дій — отримання макета, перевірку файлів, підготовку до друку, створення заявки та завершення замовлення — в єдине інтерактивне середовище, яке не лише виконує функції, але й наочно відображає весь процес у вигляді графічного інтерфейсу.

На відміну від класичних MIS-систем, що часто реалізовані як веб-платформи з важкою адміністративною логікою, дана програма створена як десктоп застосунок, у якому користувач щодня працює на одному робочому місці, маючи доступ до локальної файлової структури та хмарного сховища. Це дозволило

зробити інтерфейс максимально адаптованим під потреби поліграфічного оператора, із збереженням звичної логіки роботи цеху.

Структура програми з точки зору користувача. Коли користувач відкриває програму, перед ним одразу з'являється головне вікно з системою вкладок. Кожна вкладка — це певний етап обробки замовлення: «Пошта», «Файли», «Заявки», «Архів». Така вкладкова структура є інтуїтивною та відображає звичну логіку дій: спочатку надходить файл, потім його готують до друку, після цього формують заявку, і зрештою — завершують замовлення та архівують його.

У верхній частині кожної вкладки розташовані фільтри або меню вибору дій, а нижче — основна таблиця з даними. Справа часто розміщена контекстна панель із деталями вибраного елементу. Програма використовує кольорові індикатори, підказки, спливаючі повідомлення для інформування користувача про стан операцій.

1. Отримання замовлень — вкладка «Пошта». У вкладці «Пошта» користувач бачить таблицю всіх вхідних електронних листів, завантажених через інтеграцію з Gmail API. Кожен лист відображено у вигляді рядка з чотирма основними полями: дата отримання, відправник, тема повідомлення, статус вкладення (позначається іконкою, наприклад, скріпкою для файлів або червоним знаком, якщо вкладення не вдалося зчитати).

Вибравши певний лист, користувач бачить праворуч розгорнутий вміст листа — текст повідомлення, список вкладених файлів, кнопку «Завантажити файли». При натисканні система витягує вкладення, зберігає їх у відповідну директорію та виводить сповіщення: «Файли збережено у /вхідні/дата/назва». Цей механізм дозволяє уникнути ручного копіювання файлів і мінімізує ризик втрати вкладень.

2. Попередня обробка — вкладка «Файли». Друга вкладка — це центр перевірки якості та підготовки файлів до друку. Інтерфейс тут побудований як таблиця оброблюваних зображень: у кожному рядку вказано ім'я файлу, тип (TIFF, JPEG, PDF), розмір, DPI, кольорова модель (RGB або CMYK), а також статус перевірки. Якщо файл має невідповідність, біля нього з'являється жовтий індикатор і кнопка «Виправити».

Справа відкривається прев'ю обраного зображення — мініатюра, що оновлюється при кожній зміні. Під прев'ю — набір кнопок: «Масштабувати», «Змінити DPI», «Додати мітки», «Перетворити PDF». Ці дії реалізовані на базі бібліотеки Pillow, але відображаються для користувача як прості натискання з миттєвим результатом. Наприклад, після натискання «Змінити DPI», відкривається модальне вікно з полем для вводу нового значення, після чого зображення перераховується й зберігається в папку processed.

Після завершення редагування користувач натискає «Зберегти», і система виводить повідомлення: «Обробка завершена. Файл збережено у /опрацьовані/...». Якщо файлів багато, можна виконати масову обробку — програма прогортає всі зображення, автоматично виправляючи параметри відповідно до шаблону.

3. Формування заявки — вкладка «Заявки». У цій вкладці користувач формує власне друкарське замовлення. На екрані — таблиця існуючих заявок із кольоровими статусами: «Новий» (жовтий), «В роботі» (синій), «Готово» (зелений), «Закрито» (сірий). Кожна заявка — це набір параметрів: клієнт, тип друку, кількість, матеріал, дата, файли.

Натиснувши «Створити нову заявку», відкривається форма з полями вводу. Частина даних автоматично підтягується з листа (дата, назва, файли), інша вводиться вручну: формат (A3, A2, банер), тип плівки, кількість копій, обробка (ламінування, порізка). Якщо обраний клієнт уже має персональний прайс — система автоматично обраховує орієнтовну вартість.

При збереженні заявки створюється структурована директорія з назвою замовлення, копіюються оброблені файли, генерується технічний JSON-файл заявки, який можна експортувати або відправити на друкарське обладнання. Заявка з'являється в таблиці з позначкою «Готова до друку».

4. Завершення замовлення та архівація. Після друку оператор заходить у заявку й натискає кнопку «Завершити». Система змінює статус на «Готово», архівує файли, переміщує їх до папки archive/ та автоматично формує повідомлення для клієнта, яке можна надіслати поштою. Якщо активовано Google Drive API — файл завантажується в хмару, а лінк вставляється в текст листа.

Таким чином, відображення процесу обслуговування в програмі — це не просто набір технічних дій, а візуально продуманий і логічно організований інтерфейс, який повторює весь цикл поліграфічного замовлення: від листа до друку. Користувач бачить, контролює й керує всім процесом, не покидаючи межі одного застосунку. Завдяки багатому візуальному супроводу, кольорам, підказкам і повідомленням, програма знижує навантаження на оператора, підвищує ефективність і значно зменшує ймовірність помилок, що робить її інструментом повноцінного виробничого рівня.

3.4. Результати розробки автоматизованої системи

У результаті проведеного дослідження, проєктування та програмної реалізації було створено повноцінну автоматизовану систему, яка ефективно вирішує проблему рутинної обробки онлайн-замовлень у поліграфічному підрозділі. Система поєднує функціональність одразу кількох типових етапів виробничого процесу — прийом вхідної інформації, обробку графічних файлів, генерацію друкарських заявок, облік замовлень та спілкування з клієнтами — в єдиному інтегрованому середовищі.

Найважливішим результатом розробки є те, що програмне забезпечення повністю імітує реальний порядок роботи в поліграфії, але в автоматизованій формі. Завдяки впровадженню інтеграції з Gmail API та Google Drive, програма не потребує ручного перемикання між зовнішніми сервісами — користувач має змогу обробляти замовлення, не покидаючи середовища системи.

Практична перевірка системи у реальних умовах поліграфічного підрозділу засвідчила її здатність скоротити час обробки одного замовлення на 40–60% у порівнянні з ручною роботою. Так, раніше значна частина часу витрачалась на перевірку технічних параметрів файлів, копіювання, створення структури папок і листування з клієнтом. У новій системі більшість із цих дій відбувається автоматично або напівавтоматично — лише з підтвердженням користувача.

Важливою перевагою стало також зменшення кількості помилок. Автоматизована перевірка DPI, кольорової моделі та формату файлів дозволила уникнути випадків, коли на друк відправлялись неякісні макети. Система сигналізує про відхилення параметрів і пропонує одразу внести виправлення.

Окремим досягненням є впровадження механізму архівації та структурованого зберігання замовлень. Це значно полегшує пошук замовлень у майбутньому, дозволяє відновити історію співпраці з клієнтом, формувати статистику за обсягами робіт, видами друку чи періодами активності.

Крім технічних результатів, програма має й значний організаційний ефект: вона стандартизує порядок дій для кожного працівника, зменшує навантаження на менеджерів та дизайнерів, і робить сам процес замовлення для клієнта більш прозорим, передбачуваним і керованим.

Таким чином, розроблена система продемонструвала свою ефективність як інструмент для цифрової трансформації поліграфічного підрозділу малого або середнього масштабу. Вона не лише вирішує поточні задачі, але й має потенціал для подальшого розвитку — зокрема, шляхом додавання нових модулів (касовий облік, статистика, чат з клієнтом), масштабування на кілька робочих місць або адаптації до мобільного формату.

Отже, результати виконаної роботи підтверджують доцільність автоматизації процесів у поліграфії, особливо на етапі обслуговування онлайн-замовлень. Створена система забезпечує зменшення навантаження, підвищення якості, контрольованість і гнучкість — тобто ті характеристики, які є основою сучасного виробничого сервісу.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Структурно-функціональний аналіз та модель травмонебезпечних ситуацій

У зображеннях процесів формування, виникнення аварій та виробничих травм усі випадкові події, що утворюють конкретну аварійну ситуацію, пов'язані між собою причинно-наслідковими зв'язками.

Метод логічного моделювання потенційних аварій, травм та катастроф відкриває можливість розробити досконалу систему управління ОП виробництва, яка базується на оперативному пошуку виробничих небезпек, їх глибокому аналізі й терміновому прийнятті заходів для усунення потенційних небезпек ще до виникнення травмонебезпечних та катастрофічних ситуацій. Деякі небезпечні ситуації в табл. 4.1.

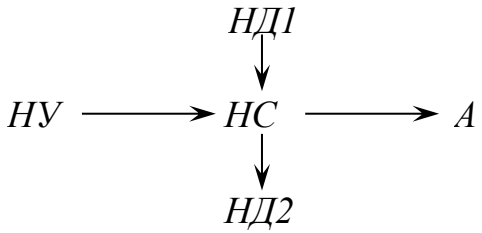
Рисунок 4.1 – Типові небезпеки на робочому місці

Поліграфічний підрозділ, навіть за умови часткової автоматизації, залишається простором з підвищеним ризиком для здоров’я працівників [17].

Таблиця 4.1. Моделювання процесів виникнення травмонебезпечних і аварійних ситуацій

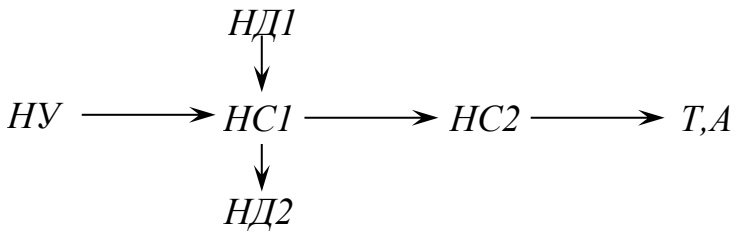
Вид робіт	Виробнича небезпека			Можливі наслідки	Заходи запобігання небезпечним ситуаціям
	Небезпечна умова (НУ)	Небезпечна дія (НД)	Небезпечна ситуація (НС)		
Використання механічної вентиляції	Оператор не перевірив обладнання НУ	Пошкоджений трубопровід мережі НД1 Закупорений трубопровід шланга НД2	Відмова вентиляційної системи (двигуна) НС	Аварія	Розвісити плакати, провести інструктажі із експлуатації обладнання системи

Модель процесу:



Використання електронних пристроїв регулювання	Пошкоджена ізоляція провідників з’єднання НУ	Пробій на корпус НД1 Коротке замикання НД2	Ураження людини електричним струмом НС1 Виведення обладнання із ладу НС2	Травма Аварія	Заміна провідників, установлення захисного обладнання (запобіжників, захист від ураження людини струмом) тощо

Модель процесу:



Робота з великою кількістю технічного обладнання (плотери, ламінатори, різальні машини, брошурувальники тощо), постійна взаємодія з електронними пристроями, зберігання та переміщення важких матеріалів, а також контакт із хімічними речовинами (фарби, клеї, розчинники) створюють потенційні джерела небезпеки.

Поряд із цим, особливе навантаження припадає на працівників, які займаються підготовкою файлів, оскільки постійне перебування перед моніторами, обробка великого обсягу інформації, одноманітні рухи та напружена зорово-розумова діяльність призводять до накопичення втоми та можуть спричиняти професійні захворювання [17].

4.2. Розрахунок штучного заземлення

Для коректного проектування системи заземлення необхідно визначити кількість і геометричні параметри електродів, які забезпечуватимуть нормативний опір заземлення. Згідно з вимогами ПУЕ, для мереж 380/220 В з глухозаземленою нейтраллю опір заземлення не повинен перевищувати 4 Ом.

У розрахунках будемо виходити з таких типових умов:

- Тип ґрунту — суглинок, із питомим опором ρ = 100 Ом·м;
- Вертикальний заземлювач — сталева арматура Ø16 мм, довжиною 3 м;
- Відстань між заземлювачами — 3 м;
- Необхідний опір заземлення — не більше 4 Ом.

Розрахунок опору одного вертикального заземлювача

Використовуємо формулу для опору одного вертикального електрода:

$$R_1 = \left(\frac{\rho}{2\pi L}\right) * (\ln\frac{2L}{d}) - 1)$$

Підставимо значення:

$$R_1 = \left(\frac{100}{2 * 3.14 * 3}\right) * (\ln\frac{2 * 3}{0.016} - 1) \approx$$

$$\frac{100}{18.84} * (\ln 375 - 1) \approx 5.31 * 4.926 \approx 26.17 \text{ Ом}$$

Загальний опір багатьох електродів:

$$R = \frac{R_1}{n\eta}$$

Кількість електродів:

$$n = \frac{R_1}{R * \eta} = \frac{26.7}{4 * 0.6} \approx 10.9 \approx 11 \text{ електродів}$$

Отже, потрібно щонайменше 11 вертикальних заземлювачів, розташованих по периметру або в лінію, об’єднаних сталевою смугою, прокладеною у траншеї на глибині 0,5–0,7 м.

4.3. Безпека в надзвичайних ситуаціях

Актуальність проблеми забезпечення природо-техногенної безпеки населення і території зумовлена тенденціями зростання втрат людей і шкоди територіям, що спричиняються небезпечними природними явищами, промисловими аваріями і катастрофами.

Інженерний захист проводиться з метою виконання вимог ІТЗ із питань забудови міст, розміщення ПНО, будівлі будинків, інженерних споруд та інше.

Медичний захист проводиться для зменшення ступеня ураження людей, своєчасного надання допомоги постраждалим та їх лікування, забезпечення епідеміологічного благополуччя в районах надзвичайних ситуацій.

Біологічний захист включає своєчасне виявлення чинників біологічного зараження, їх характеру і масштабів, проведення комплексу адміністративно-господарських, режимно-обмежувальних і спеціальних протиепідемічних та медичних заходів.

Радіаційний і хімічний захист включає заходи щодо виявлення і оцінки радіаційної та хімічної обстановки, організацію і здійснення дозиметричного та хімічного контролю, розроблення типових режимів радіаційного захисту, забезпечення засобами індивідуального захисту, організацію і проведення спеціальної обробки.

5. ТЕХНІКО-ЕКОНОМІЧНЕ ОЦІНЕННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

Автоматизована система обслуговування онлайн-замовлень поліграфічного підрозділу, створена в межах цієї роботи, дозволила значно знизити навантаження на персонал, підвищити точність та пришвидшити обробку замовлень. Основна економічна ефективність полягає у скороченні часу на типові операції, мінімізації помилок, оптимізації взаємодії з клієнтами та відмові від стороннього програмного забезпечення. Використання безкоштовних бібліотек та відкритих API значно знижує вартість впровадження рішення. Порівняння витрат часу та трудозатрат до і після впровадження системи наведено у таблиці 5.1.

Таблиці 5.1 – Порівняння витрат часу та трудозатрат

Операція	**До автоматизації**	**Після автоматизації**
Перевірка макетів (DPI, формат)	5–10 хв	30–60 сек
Створення заявки вручну	5–7 хв	1–2 хв
Зберігання/структурування файлів	3–4 хв	Автоматично
Відповідь клієнту	2–5 хв	Генерується автоматично
Помилки/відмови	1–2 на день	0–1 на тиждень

У підсумку, повний цикл обробки одного замовлення, який раніше займав у середньому 20–25 хвилин, тепер може бути виконаний за 4–6 хвилин. Така оптимізація дозволяє обробляти в 3–4 рази більше замовлень у межах одного робочого дня, без потреби у додатковому персоналі. Крім того, стандартизація дій у програмі зменшує ймовірність людської помилки та сприяє покращенню взаємодії з клієнтами завдяки автоматичним повідомленням, архівам та візуальному контролю за станом заявок.

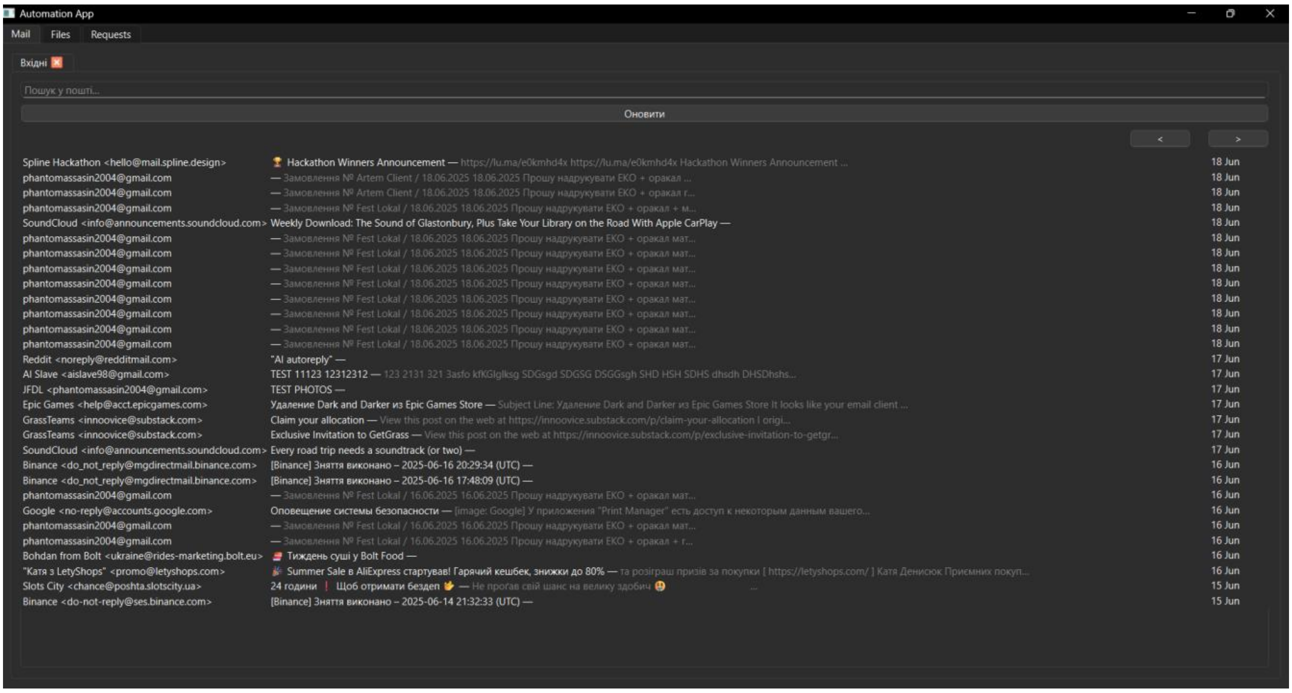


Рисунок 5.1 – Інтерфейс вкладки «Пошта» з таблицею вхідних листів

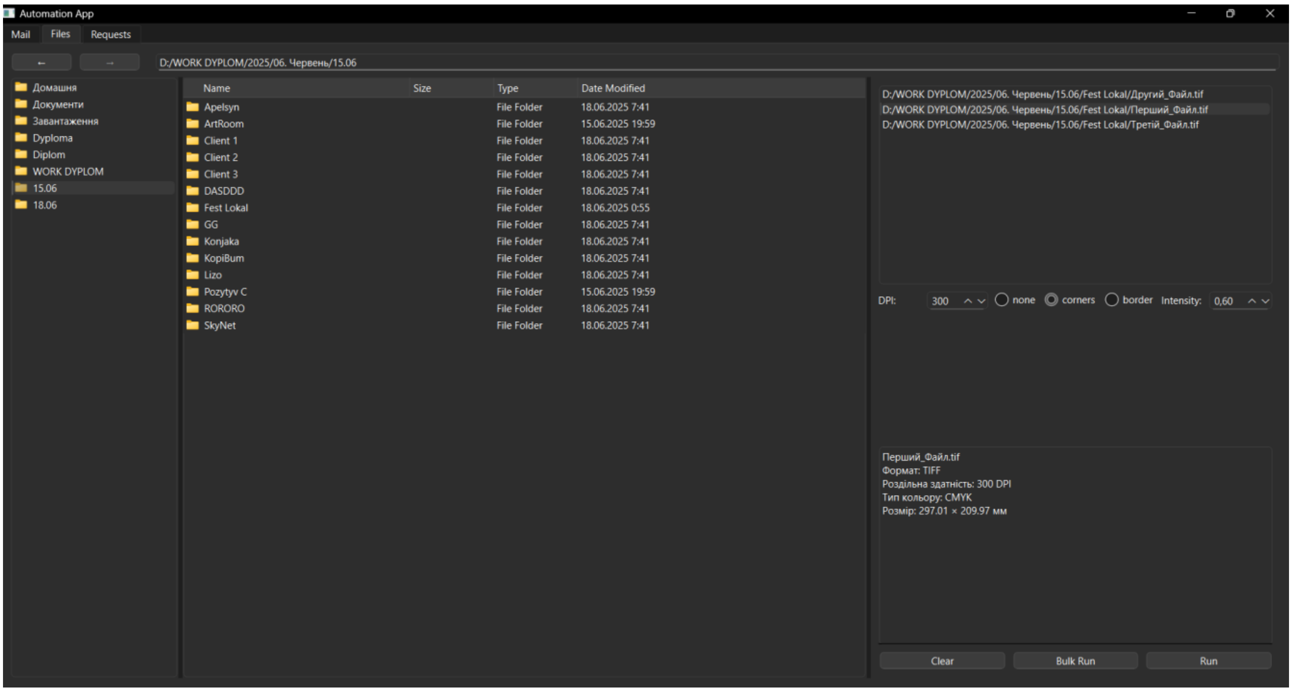


Рисунок 5.2 – Модуль обробки файлів, полями DPI та кнопками дій

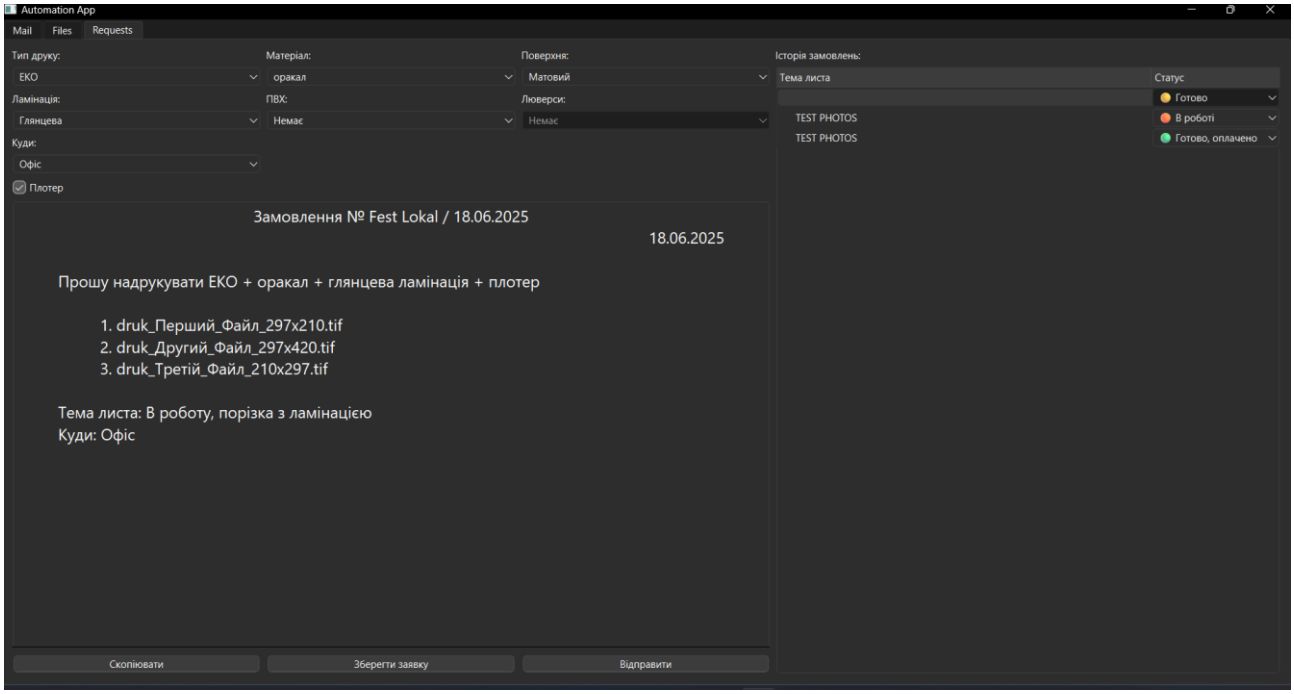


Рисунок 5.3 – Форма створення заявки з автозаповненням параметрів та історією поданих замовлень

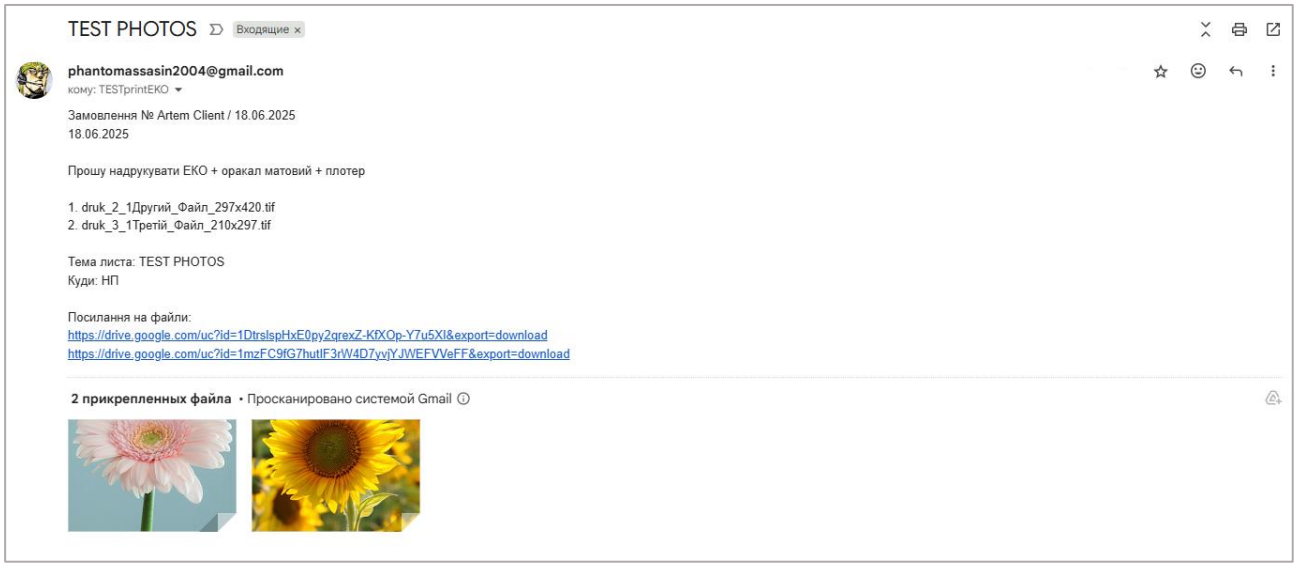


Рисунок 5.4 – Зразок листа отриманий від програми

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У результаті виконання даної дипломної роботи досягнуто головної мети – розроблено автоматизовану систему обслуговування онлайн-замовлень поліграфічного підрозділу. Створене програмне рішення інтегрує декілька ключових функцій в межах єдиного сервісу: автоматизоване опрацювання вхідної електронної пошти (через Gmail API); попередню підготовку графічних файлів до друку (з використанням бібліотеки Pillow); генерацію внутрішніх заявок на друк; а також динамічний розрахунок вартості замовлення з урахуванням індивідуальних прайс-політик клієнтів. Система реалізована мовою Python із використанням фреймворку PySide6 для побудови графічного інтерфейсу, а також інтегрована з Google Drive API для зберігання файлів у хмарному середовищі.

Запропонований комплексний підхід дозволив автоматизувати всі основні етапи процесу обробки замовлення, що раніше виконувалися вручну. Тепер отримання замовлення (завантаження листів та вкладень з електронної пошти), перевірка технічних параметрів макетів, систематизація даних замовлення та підготовка внутрішньої документації (друкарських заявок) здійснюються програмою з мінімальним залученням людини. Вбудований механізм перевірки (preflight-контроль) гарантує, що кожен графічний файл відповідає встановленим критеріям (достатній DPI, правильна кольорова модель, наявність полів під порізку тощо), що суттєво знижує ризик браку та помилок. Система автоматично впорядковує отримані файли та інформацію про замовлення у структурованому вигляді (створюючи відповідні каталоги та записи для кожного проєкту). Такий підхід забезпечує єдність структури даних та унеможливлює хаотичне збереження файлів, яке було характерне для ручного опрацювання.

Впровадження розробленої системи веде до суттєвого підвищення продуктивності обробки замовлень. Багато рутинних операцій, які раніше потребували значного часу персоналу, тепер виконуються програмою автоматично за лічені секунди. Наприклад, завантаження та розбір десятків вхідних листів із вкладеннями відбувається значно швидше порівняно з покроковим ручним

опрацюванням кожного замовлення. Це скорочує загальний час обробки та дозволяє швидше переходити до етапу друку. Крім того, завдяки мінімізації впливу людського фактору на однотипні операції, істотно зменшується кількість потенційних помилок (як-от пропущений лист або некоректно підготовлений макет), що підвищує надійність і точність процесу.

З технічної точки зору, розроблене рішення відзначається модульною архітектурою та дотриманням принципів ООП, що спрощує його підтримку і масштабування. Кожен функціональний блок (робота з поштою, обробка файлів, управління заявками тощо) реалізований у вигляді окремого компонента, який взаємодіє з іншими через чітко визначені інтерфейси. Такий підхід полегшує додавання нових можливостей або внесення змін без впливу на інші частини системи. Використання перевірених бібліотек і стандартних API забезпечує сумісність та гнучкість рішення: за потреби систему можна адаптувати до нових сервісів (наприклад, перейти на іншу поштову платформу чи хмарне сховище) з мінімальними змінами в коді

Отже, у дипломній роботі підтверджено, що автоматизація процесу обробки онлайн-замовлень у поліграфічній галузі є не лише можливою, але й надзвичайно ефективною. Розроблена система успішно вирішує поставлені завдання, об’єднавши декілька критично важливих процесів у єдиний оптимізований робочий потік. Поставлену на початку дослідження мету досягнуто повною мірою, а всі заплановані етапи роботи – від аналізу стану задачі до створення і тестування програмного продукту – виконано. В результаті отримано дієвий інструмент, який помітно підвищує оперативність, точність та зручність виконання друкарських замовлень. Завдяки гнучкій архітектурі та підтвердженій ефективності система має значний потенціал для подальшого розвитку і адаптації. Вона може слугувати основою для майбутніх удосконалень та є яскравим прикладом застосування сучасних ІТ-рішень для оптимізації виробничо-офісних процесів. Отримані результати мають практичну цінність для поліграфічного сектора та підкреслюють важливість впровадження інновацій і автоматизації для підтримання конкурентоспроможності в цифрову епоху.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CIP4 Organization. Job Definition Format (JDF) Specification 1.8. URL: https://www.cip4.org/files/cip4/documents/JDF%20Specification%201.8%20www.pdf (дата звернення: 20.03.2025).

2. Enfocus. PitStop Pro Reference Guide. URL: https://www.enfocus.com/manuals/ReferenceGuide/PP/23/enUS/pdf/PitStopReference.pdf (дата звернення: 30.03.2025).

3. Ghent Workgroup. PDF and PDF/X-Plus: Overview and Best Practices. URL: https://gwg.org/wp-content/uploads/2021/05/PDF-and-PDFX-plus.pdf (дата звернення: (дата звернення: 24.03.2025).

4. Google Developers. *Gmail API Overview.* URL: https://developers.google.com/workspace/gmail/api/guides (дата звернення: 10.05.2025).

5. Google Developers. *Google Drive API Overview.* URL: https://developers.google.com/workspace/drive/api/guides/about-sdk (дата звернення: 18.05.2025).

6. Keypoint Intelligence. *Protecting Enterprise Print Systems: ISO 20243 Supply-Chain Security for Print Devices and Cartridges.* URL: https://keypointintelligence.com/hubfs/19585250/HP_Samples/hp-print-iso-20243.pdf (дата звернення: 09.06.2025).

7. Mead & Hunt. *Prioritizing Safety in Industrial Automation Design.* URL: https://meadhunt.com/industrial-automation-design/ (дата звернення: 07.06.2025).

8. NAPCO Research. How Workflow Automation Optimizes Operational Productivity and Results. URL: https://www.napco.com/wp-content/uploads/How-Workflow-Automation-Optimizes-Operational-Productivity-and-Results-5.pdf (дата звернення: 08.04.2025).

9. National Institute of Standards and Technology. Guide to Industrial Control Systems (ICS) Security, SP 800-82 Rev. 2. URL:

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf (дата звернення: 06.06.2025).

10. OnPrintShop. *Web-to-Print SaaS Solution – Streamline Your Print Workflow.* URL: https://onprintshop.com/ (дата звернення: 03.05.2025).

11. Pillow Development Team. *Pillow (PIL Fork) Documentation.* URL: https://pillow.readthedocs.io/ (дата звернення: 31.05.2025).

12. PrintVis. *PrintVis – MIS/ERP Solution for the Graphics Industry (Product Overview).* URL: https://learn.printvis.com/Legacy/General/printindustry/ (дата звернення: 25.04.2025).

13. Qt Company. *Qt for Python (PySide6) Documentation.* URL: https://doc.qt.io/qtforpython-6/ (дата звернення: 26.05.2025).

14. Roudný P., Držková M. *Use of Prepress Automation in the Czech Republic and Examples of Automated Processing for Selected Prepress Tasks.* Proceedings of GRID 2020. URL: https://www.grid.uns.ac.rs/symposium/download/2020/72.pdf (дата звернення: 11.04.2025).

15. Singhal P., Nautiyal V., Sharma N., Panwar H. *An Automation Perspective of Print Production Workflow System.* International Journal of Innovative Technology and Exploring Engineering, Vol. 9 (3), 2020. URL: https://www.ijitee.org/wp-content/uploads/papers/v9i3/C9112019320.pdf (дата звернення: 19.04.2025).

16. Szentgyörgyvölgyi R. *Effect of the Digital Technology to the Print Production Processes.* Acta Polytechnica Hungarica, Vol. 5 (3), 2008. URL: https://acta.uni-obuda.hu/Szentgyorgyvolgyi_15.pdf (дата звернення: 03.04.2025)

17. The Packman. *Technical Paper on Safety Aspects of Printing Machinery and Standardization.* URL: https://thepackman.in/technical-paper-on-safety-aspects-of-printing-machinery-and-standardization/ (дата звернення: 28.03.2025).

18. U.S. Environmental Protection Agency. *Multimedia Compliance/Pollution Prevention Assessment: Screen Printing Operations.* URL: https://archive.epa.gov/compliance/resources/publications/assistance/sectors/web/pdf/screen.pdf (дата звернення: 11.03.2025).

19. U.S. Environmental Protection Agency. *Printing and Publishing: Reducing Your Waste and Hazardous Materials.* URL: https://www.epa.gov/sites/default/files/2020-07/documents/printing-publishing_ip.pdf (дата звернення: 04.03.2025).

20. U.S. Environmental Protection Agency. *Sector Notebook Project: Profile of the Printing Industry.* URL: https://www.clu-in.org/download/toolkit/print.pdf (дата звернення: 19.03.2025).

ДОДАТКИ

Додаток А.

Код програми для вкладки «Пошта» написаний мовою Python

```
class MailClient:
    def __init__(self, creds_json='credentials.json',
token_json='token.json'):
        creds = None
        try:
            creds =
Credentials.from_authorized_user_file(token_json, SCOPES)
        except Exception:
            flow =
InstalledAppFlow.from_client_secrets_file(creds_json, SCOPES)
            creds = flow.run_local_server(port=0)
            with open(token_json, 'w') as f:
                f.write(creds.to_json())
        self.creds = creds
        self.service = build('gmail', 'v1',
credentials=creds)

    def _extract_body_from_parts(self, part):
        if isinstance(part, dict):
            mime = part.get('mimeType', '')
            if mime in ("text/plain", "text/html",
"text/calendar"):
                data = part.get('body', {}).get('data')
                if data:
                    raw = base64.urlsafe_b64decode(data)
                    charset = part.get('body', {}).get('charset',
'utf-8')
                    try:
                        return raw.decode(charset)
                    except:
                        return raw.decode('utf-8', 'replace')
            for sub in part.get('parts', []):
                res = self._extract_body_from_parts(sub)
                if res:
                    return res
            return None

        if isinstance(part, email.message.Message):
            mime = part.get_content_type()
            if mime in ("text/plain", "text/html",
"text/calendar"):
                payload = part.get_payload(decode=True)
                if payload:
                    ch = part.get_content_charset() or 'utf-8'
                    try:
                        return payload.decode(ch)
                    except:
                        return payload.decode('utf-8', 'replace')
            if part.is_multipart():
                for sub in part.iter_parts():
                    res = self._extract_body_from_parts(sub)
                    if res:
                        return res
            return None

    def extract_body_from_full(self, msg):
        if 'payload' in msg:
            return
self._extract_body_from_parts(msg['payload'])
        raw = msg.get('raw')
        if raw:
            decoded =
base64.urlsafe_b64decode(raw.encode('ASCII'))
            mime_msg =
email.message_from_bytes(decoded, policy=policy.default)
            return
self._extract_body_from_parts(mime_msg)

        return None


    def mark_thread_read(self, thread_id):
        """Знімає мітку UNREAD з усієї треди."""
        self.service.users().threads().modify(
            userId='me',
            id=thread_id,
            body={'removeLabelIds': ['UNREAD']}
        ).execute()

    def send_message(self, to_address, subject, body,
thread_id=None):
        msg = MIMEText(body)
        msg['to'] = to_address
        msg['subject'] = subject
        if thread_id:
            msg['threadId'] = thread_id

        raw =
base64.urlsafe_b64encode(msg.as_bytes()).decode()
        payload = {'raw': raw}
        if thread_id:
            payload['threadId'] = thread_id

        sent = self.service.users().messages().send(
            userId='me',
            body=payload
        ).execute()
        return sent

    def get_thread(self, thread_id):
        thread = self.service.users().threads() \
            .get(userId='me', id=thread_id, format='full') \
            .execute()

        msgs = []
        for m in thread.get('messages', []):
            mid = m['id']
            # розбираємо через вже існуючий метод,
але з сирим повідомленням
            sender, subject, body, attachments =
self._parse_message_dict(m)
            msgs.append((mid, sender, subject, body,
attachments))
        return msgs

    def _parse_message_dict(self, m):
        headers = {h['name']: h['value'] for h in
m['payload']['headers']}
        sender = headers.get('From', '')
        subject = headers.get('Subject', '')
        parts = m['payload'].get('parts', [])
        body = ""
```

```
                for p in parts:
                    if p.get('mimeType') == 'text/plain':
                        data = p['body'].get('data', '')
                        body =
base64.urlsafe_b64decode(data).decode('utf-8',
errors='ignore')
                        break
                atts = []
                for p in parts:
                    if p.get('filename'):
                        aid = p['body']['attachmentId']
                        atts.append((p['filename'], aid))
                return sender, subject, body, atts

            def list_messages(self, labelIds=None,
max_results=100):
                params = {'userId': 'me', 'maxResults':
max_results}
                if labelIds:
                    params['labelIds'] = labelIds
                res =
self.service.users().messages().list(**params).execute()
                return res.get('messages', [])

            def _get_payload_text(self, part):
                mime = part.get('mimeType', '')
                if mime == 'text/plain' and part.get('body',
{}).get('data'):
                    data = part['body']['data']
                    return
base64.urlsafe_b64decode(data).decode('utf-8')
                if mime == 'text/html' and part.get('body',
{}).get('data'):
                    data = part['body']['data']
                    html =
base64.urlsafe_b64decode(data).decode('utf-8')
                    return html
                for sub in part.get('parts', []):
                    txt = self._get_payload_text(sub)
                    if txt:
                        return txt
                return None


            def get_message(self, msg_id):
                msg = self.service.users().messages().get(
                    userId='me', id=msg_id, format='full'
                ).execute()
                text = self._get_payload_text(msg['payload']) or
"(немає тексту)"
                parts = msg['payload'].get('parts', [])
                headers = msg['payload'].get('headers', [])
                subject = next((h['value'] for h in headers if
h['name']=='Subject'), '')
                sender = next((h['value'] for h in headers if
h['name']=='From'), '')
                body = self.extract_body_from_full(msg) or
msg.get('snippet', '')
                attachments = []
                for part in parts:
                    if part.get('filename'):
                        attachments.append((part['filename'],
part['body'].get('attachmentId')))
                    if part.get('mimeType')=='text/plain':
                        data = part['body'].get('data','')
                        body +=
base64.urlsafe_b64decode(data).decode()
                return sender, subject, body, attachments

            def mark_unread(self, msg_id):
                self.service.users().messages().modify(
                    userId='me', id=msg_id,

body={'removeLabelIds':['INBOX'],'addLabelIds':['UNREAD']}
                ).execute()

            def download_attachment(self, msg_id,
attachment_id, filename):
                att =
self.service.users().messages().attachments().get(
                    userId='me', messageId=msg_id,
id=attachment_id
                ).execute()
                data = base64.urlsafe_b64decode(att['data'])
                with open(filename, 'wb') as f:
                    f.write(data)
```

Додаток Б.

Код програми для вкладки «Файли» написаний мовою Python

```
            lass FileProcessingTab(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.history = []; self.history_index = -1
        self.clipboard = {'mode': None, 'paths': []}

        main_layout = QVBoxLayout(self)

        nav = QHBoxLayout()
        self.back_btn = QPushButton("←", clicked=self.go_back);
self.back_btn.setEnabled(False)
        self.fwd_btn  = QPushButton("→",
clicked=self.go_forward); self.fwd_btn.setEnabled(False)
        nav.addWidget(self.back_btn);
nav.addWidget(self.fwd_btn)

        nav.addSpacing(10)
        self.path_edit = QLineEdit()
        self.path_edit.setReadOnly(True)

self.path_edit.returnPressed.connect(self.on_path_entered)
        self.path_edit.mouseDoubleClickEvent =
self.enable_path_edit
        nav.addWidget(self.path_edit, 1)
        main_layout.addLayout(nav)

        splitter = QSplitter(Qt.Horizontal)
        main_layout.addWidget(splitter)

        # — ЛІВА: pinned + flat tree —
        left = QSplitter(Qt.Horizontal)
        splitter.addWidget(left)

        # pinned
        self.pinned_file = os.path.join(os.getcwd(), 'pinned.json')
        self.default_pinned = {
            "Домашня": os.path.expanduser("~"),
            "Документи": os.path.join(os.path.expanduser("~"),
"Documents"),
            "Завантаження": os.path.join(os.path.expanduser("~"),
"Downloads")
        }
        folder_icon = self.style().standardIcon(QStyle.SP_DirIcon)

        self.pinned = self.load_pinned()
        self.pinned_list = QListWidget()
        self.pinned_list.setFixedWidth(200)
        self.pinned_list.setAcceptDrops(True)

self.pinned_list.setDragDropMode(QAbstractItemView.DropOn
ly)
        self.pinned_list.setDefaultDropAction(Qt.CopyAction)

        self.pinned_list.setAcceptDrops(True)

self.pinned_list.setDragDropMode(QAbstractItemView.DropOn
ly)
        self.pinned_list.setDefaultDropAction(Qt.CopyAction)

        def pinned_dragEnterEvent(e):
            if e.mimeData().hasUrls():
                e.acceptProposedAction()

        self.pinned_list.dragEnterEvent = pinned_dragEnterEvent

        def pinned_dragMoveEvent(e):
            e.acceptProposedAction()

        self.pinned_list.dragMoveEvent = pinned_dragMoveEvent

        def pinned_dropEvent(e):
            folder_icon =
self.style().standardIcon(QStyle.SP_DirIcon)
            added = False
            for url in e.mimeData().urls():
                path = url.toLocalFile()
                if os.path.isdir(path):
                    name = os.path.basename(path)
                    if name not in self.pinned:
                        self.pinned[name] = path
                        item = QListWidgetItem(folder_icon, name)
                        self.pinned_list.addItem(item)
                        added = True
            if added:
                self.save_pinned()
            e.acceptProposedAction()


self.pinned_list.setContextMenuPolicy(Qt.CustomContextMenu
)

self.pinned_list.customContextMenuRequested.connect(self.on
_pinned_context_menu)



        self.pinned_list.dropEvent = pinned_dropEvent

        for name in self.pinned:
            self.pinned_list.addItem(name)
        self.pinned_list.itemClicked.connect(
            lambda it: self.navigate_to(self.pinned[it.text()])
        )
        left.addWidget(self.pinned_list)
        self.pinned_list.clear()
        for name, path in self.pinned.items():
            item = QListWidgetItem(folder_icon, name)
            self.pinned_list.addItem(item)
        self.model = QFileSystemModel()
        self.model.setRootPath(QDir.rootPath())
        self.model.setFilter(QDir.AllEntries |
QDir.NoDotAndDotDot)

        self.tree = FileTreeView()
        self.file_list = DropListWidget()
        self.tree.setModel(self.model)
        self.tree.setRootIsDecorated(False)
        self.tree.setItemsExpandable(False)
```

```
        self.tree.setIndentation(0)

self.tree.setEditTriggers(QAbstractItemView.EditKeyPressed |
QAbstractItemView.SelectedClicked |
QAbstractItemView.AnyKeyPressed)
        self.tree.setContextMenuPolicy(Qt.CustomContextMenu)

self.tree.customContextMenuRequested.connect(self.on_conte
xt_menu)
        self.tree.doubleClicked.connect(self.on_tree_double_click)
        left.addWidget(self.tree)

        left.setStretchFactor(0,1); left.setStretchFactor(1,4)
        splitter.setStretchFactor(0,65)

        # — ПРАВА: DropListWidget + налаштування —
        right = QWidget()
        rlay = QVBoxLayout(right)

        self.file_list = DropListWidget()

self.file_list.currentItemChanged.connect(self.preview_file)

self.file_list.itemSelectionChanged.connect(self.handle_selectio
n_change)
        rlay.addWidget(self.file_list)

        opts = QHBoxLayout()
        opts.addWidget(QLabel("DPI:"))
        self.dpi = QSpinBox(); self.dpi.setRange(1,600);
self.dpi.setValue(300)
        opts.addWidget(self.dpi)

        self.bg = QButtonGroup(self)
        for opt in ("none","corners","border"):
            rb = QRadioButton(opt)
            self.bg.addButton(rb); opts.addWidget(rb)
        self.bg.buttons()[0].setChecked(True)

        opts.addWidget(QLabel("Intensity:"))
        self.intensity = QDoubleSpinBox();
self.intensity.setRange(0,1)
        self.intensity.setSingleStep(0.1); self.intensity.setValue(0.5)
        opts.addWidget(self.intensity)
        rlay.addLayout(opts)

        self.preview = QLabel(); self.preview.setFixedHeight(150)
        self.preview.setAlignment(Qt.AlignCenter)
        rlay.addWidget(self.preview)

        self.info = QTextEdit(); self.info.setReadOnly(True)
        rlay.addWidget(self.info)

        btns = QHBoxLayout()
        btns.addWidget(QPushButton("Clear",
clicked=self.clear_list))
        btns.addWidget(QPushButton("Bulk Run",
clicked=self.run_bulk))
        btns.addWidget(QPushButton("Run",
clicked=self.run_single))
        rlay.addLayout(btns)

        splitter.addWidget(right)
        splitter.setStretchFactor(1,35)

        self.navigate_to(self.pinned["Домашня"])

    def refresh_folder_tree(self):
        idx = self.model.index(self.current_folder)
        self.tree.setRootIndex(idx)

    def open_client_folder(self, client_folder, files):
        self.current_folder = client_folder
        self.refresh_folder_tree()
        self.file_list.clear()
        first_item = None
        for p in files:
            if os.path.isfile(p):
                it = QListWidgetItem(os.path.basename(p))
                it.setData(Qt.UserRole, p)
                self.file_list.addItem(it)
                if first_item is None:
                    first_item = it
        if first_item:
            self.file_list.blockSignals(True)
            self.file_list.setCurrentItem(first_item)
            self.file_list.blockSignals(False)

    def send_to_request(self, paths):
        if not isinstance(paths, list):
            paths = [paths]
        paths = [os.path.normpath(p) for p in paths]

        meta = load_meta()

        subjects = []
        for path in paths:
            for entry in meta:
                downloads = [os.path.normpath(p) for p in
entry.get("downloads", [])]
                if path in downloads:
                    subjects.append(entry["subject"])
                    break

        if not subjects:
            subject_to_use = ""
        else:
            # прибираємо дублі
            uniq = list(dict.fromkeys(subjects))
            if len(uniq) == 1:
                subject_to_use = uniq[0]
            else:
                subject_to_use = "; ".join(uniq)

        mw = self.window()
        rt = mw.requests_tab
        mw.tabs.setCurrentWidget(rt)

        rt.request_file_paths = paths
        rt.request_files = [os.path.basename(p) for p in paths]
        rt.mail_subject = subject_to_use

        client_name = "Клієнт"
        parts = paths[0].split(os.sep)
        for i, part in enumerate(parts):
            if re.match(r'\d{2}\.\d{2}', part):
                client_name = parts[i + 1] if i + 1 < len(parts) else
client_name
                break
        rt.client_name = client_name

        rt.generate_preview()
```