Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра
на тему: «Розробка програмного забезпечення для автоматизації розрахунків банківських депозитних лімітів, ОВДП та нормативу Н6»

Виконав: студент групи ІПЗ-21-2
Спеціальність 121 «Інженерія програмного забезпечення»
Вдовін Андрій Максимович
(прізвище та ініціали)

Керівник к.т.н., доц. Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

Доцент кафедри програмного забезпечення систем Дніпровського державного технічного університету
(посада)

к.т.н., доц. Косухіна О.С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Вдовін А.М. Розробка програмного забезпечення для автоматизації розрахунків банківських депозитних лімітів, облігацій внутрішньої державної позики та нормативу Н6.

Дипломна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення». – Університет митної справи та фінансів, Дніпро, 2025.

Дана кваліфікаційна робота присвячена розробці програмного забезпечення для автоматизації розрахунків депозитних лімітів, ОВДП (облігацій внутрішньої державної позики), які банки можуть включати в покриття обов'язкових резервів та нормативу Н6. У сучасних умовах зростаючої кількості фінансових операцій актуальним є створення інструментів, що підвищують швидкість та ефективність обробки даних та знижують ризик помилок, викликаних людським фактором. Було проведено проектування, розробку програмного забезпечення та її тестування. Передбачено інтерфейс користувача для зручного введення й перегляду результатів.

У роботі обґрунтовано вибір інкрементної моделі розробки програмного забезпечення, що дозволяє поетапно впроваджувати і тестувати функціональні компоненти. Проведено модульне та регресійне тестування для перевірки коректності розрахунків. Особливу увагу приділено питанням надійності, точності результатів і відповідності програмного забезпечення вимогам банківських регуляторів.

Результати дипломної роботи можуть бути використані в банківських установах для підвищення продуктивності та точності при формуванні фінансової і статистичної звітності.

Ключові слова: програмне забезпечення, автоматизація, розробка, бібліотеки, Python.

ABSTRACT

Vdovin A.M. Development of software for automating calculations of bank deposit limits, government bonds and the N6 standard.

Thesis for the degree of Bachelor in specialty 121 "Software Engineering". – University of Customs and Finance, Dnipro, 2025.

This qualification work is dedicated to the development of software for automating the calculations of deposit limits, domestic government bonds, which banks can include in the coverage of required reserves and the N6 standard. In today's conditions of a growing number of financial transactions, it is relevant to create tools that increase the speed and efficiency of data processing and reduce the risk of errors caused by the human factor. Software design, development and testing were carried out. A user interface is provided for convenient input and viewing of results.

The work justifies the choice of an incremental software development model, which allows for the gradual implementation and testing of functional components. Modular and regression testing is carried out to verify the correctness of calculations. Particular attention is paid to issues of reliability, accuracy of results and compliance of the software with the requirements of banking regulators.

The results of the thesis can be used in banking institutions to increase productivity and accuracy in the formation of financial and statistical reporting.

Keywords: software, automation, development, libraries, Python.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОВДП — облігації внутрішньої державної позики

НБУ — Національний банк України

СЕП — система електронних платежів

СУБД — система управління базами даних

МСФЗ — міжнародні стандарти фінансової звітності

ETL — Extract Transform Load

API — application programming interface

АРМ — спеціальне робоче місце

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах розвитку банківської сфери та збільшення обсягів фінансової інформації актуальним стає питання автоматизації розрахунків, пов'язаних з фінансовою звітністю. Зокрема, значну увагу приділяють точному та своєчасному обчисленню банківських депозитних лімітів, облігацій внутрішньої державної позики (ОВДП) та нормативу Н6, що регулюються вимогами Національного банку України. Обробка таких розрахунків без зручних інструментів потребує багато часу і супроводжується ризиком людських помилок. Тому створення програмного забезпечення, яке здатне автоматизувати ці процеси, є надзвичайно актуальним.

У багатьох банках ці розрахунки й досі виконуються вручну або з використанням застарілих інструментів, що не гарантує високої точності, а також вимагають значних часових і людських ресурсів. Це створює потенційні ризики, пов'язані з помилками, затримками у звітності та зниженням ефективності роботи фінансових аналітиків.

Особливої актуальності ця проблема набуває в умовах постійних змін у нормативно-правовому полі, коли необхідно швидко адаптувати програмні рішення до нових вимог. Тому створення гнучкого, надійного та масштабованого програмного забезпечення, яке дозволяє адаптуватися до змін і виконувати розрахунки з мінімальним втручанням користувача, є важливою задачею сучасної інженерії програмного забезпечення.

Метою роботи є автоматизація розрахунків банківських депозитних лімітів, ОВДП та нормативу Н6 з урахуванням нормативних вимог та потреб банківських установ.

Методи дослідження: аналіз бізнес-процесів, що полягав у вивченні та формалізації існуючого процесу розрахунків банківських нормативів. Зокрема був виконаний аналіз вихідних даних, вимог Національного банку України до звітності, а також дій користувача при заповненні звітів вручну. Моделювання бізнес-процесів, які використовувалися для побудови логіки роботи майбутнього

програмного забезпечення. Була використана інкрементна модель розробки, що передбачає поступове впровадження функціоналу. Це дало змогу тестувати окремі частики програмного забезпечення. Для коректності роботи окремих функцій застосовувалися тестування за допомогою Unit Testing.

У процесі дослідження ставилися та вирішувалися наступні завдання:

1. Аналіз вимог банківських регуляторів до обчислення депозитних лімітів, ОВДП та нормативу Н6;
2. Проєктування та реалізація структури програмного забезпечення;
3. Вибір мови програмування та бібліотек;
4. Розробка інтерфейсу для користувача;
5. Тестування на точність та надійність результатів.

Об'єктом дослідження є процес обробки та формування даних для розрахунків фінансової та статистичної звітності у банківських установах.

Предметом дослідження є програмне забезпечення для автоматизації розрахунків депозитних лімітів, облігацій внутрішньої державної позики, які банки можуть включати в покриття обов'язкових резервів та нормативу Н6.

Практичне значення одержаних результатів – створення інструменту, який дозволяє підвищити ефективність роботи працівників банків, зменшити ризики помилок та забезпечити швидкий доступ до результатів фінансових розрахунків.

Структура роботи:

Розділ 1. Аналіз предметної області. Постановка задачі.

Розділ 2. Аналіз і вибір методів реалізації та побудови ПЗ.

Розділ 3. Вирішення задачі реалізації програмного забезпечення з інтерфейсом для розрахунку депозитних лімітів, облігацій внутрішньої державної позики, які банки можуть включати в покриття обов'язкових резервів та нормативу Н6.

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 20 найменувань. Обсяг роботи 51 сторінка, 20 рисунків.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Проблематика і актуальність задачі

У сучасних умовах цифрової трансформації фінансового сектору та підвищення вимог до якості, точності та швидкості формування фінансової звітності, особливої актуальності набуває автоматизація аналітичних і розрахункових процесів у банківській сфері. Банки зіштовхуються з дедалі більшими обсягами даних та необхідністю суворого дотримання вимог фінансових регуляторів, зокрема Національного банку України (НБУ) . У цьому контексті критично важливим є забезпечення своєчасного та безпомилкового обчислення таких ключових показників, як депозитні ліміти, облігації внутрішньої державної позики (ОВДП), які можуть включатися в покриття обов'язкових резервів, та норматив Н6, що відображає обмеження кредитного ризику на одного контрагента [19].

У статті по аналізу цифрового банківського сектору автор приділяє увагу найактуальнішим тенденціям діджиталізації банків в Україні та досліджує наукові підходи щодо цифрової трансформації фінансової сфери [18]. Протягом останніх двох років цифровізації банківського сектору було присвячено чимало праць вітчизняних дослідників і практиків. Зокрема, у роботі згадуються наукові розвідки О. Копилової, Ю. Пічугіної, К. Гончар, які розглядають ключові чинники діджиталізації українських банків. Автори окреслюють поетапний цикл впровадження цифрових технологій, що включає такі етапи, як тригер, розробка, аналіз, адаптація, впровадження та удосконалення. Вони також акцентують увагу на ризиках та потенційних наслідках цифрової трансформації в банківській системі.

Інші дослідники, зокрема Л. Бондаренко та А. Подарин, акцентують увагу на зростанні цифрової конкурентоспроможності України у світових рейтингах, зокрема за показниками безконтактних платежів та використання цифрових фінансових інструментів. Ці тенденції свідчать про активне впровадження інновацій в українській банківській сфері. О. Стащук і Р. Мартинюк, у свою

чергу, підкреслюють економічну ефективність цифровізації, яка дозволяє банкам скорочувати витрати на фізичні відділення та персонал завдяки автоматизації процесів і впровадженню новітніх технологій. Автор підкреслює, що тема цифровізації банківського сектору залишається надзвичайно актуальною, особливо в контексті аналізу сучасного стану діджиталізації та оцінки перспектив її подальшого розвитку. Такий підхід підкріплює наукову цінність дослідження та визначає його мету — оцінити вплив цифровізації на банківську сферу, виявити її переваги та ризики, а також запропонувати напрями майбутнього вдосконалення цифрової трансформації банків.

Тому актуальною проблемою є створення спеціалізованого програмного забезпечення, яке буде включати в себе такі аспекти:

- Зменшить часові витрати на щоденні розрахунки: банки й досі використовують ручні методи обробки даних, наприклад, копіювання даних між різними джерелами (звіти, внутрішні бази, регуляторні форми), ведення розрахунки в Excel з використанням складних формул, ведення окремих журналів для контролю операцій. Через це фахівці витрачають багато часу на підготовку звітів, аналітики замість аналізу даних займаються рутинними обчисленнями, що є особливо критичним, виходячи із вимог Національного банку України;
- Знизить ризик помилок у звітності: неправильне введення параметрів, заповнення рахунків, неактуальні дати та людська неуважність можуть призвести до поганих наслідків.
- Забезпечить відповідність розрахунків вимогам НБУ: НБУ постійно оновлює вимоги до форматів звітності, вносить зміни до формул нормативів та строків подання звітності, тому важливо швидко адаптуватися;
- Дозволить оперативно оновлювати логіку роботи відповідно до змін нормативних документів: фінансове законодавство змінюється часто і банки повинні запобігати технічним помилкам через застарілі алгоритми та швидко реагувати на нові вимоги.

З точки зору розробника, така задача є викликом, що потребує не лише реалізації обчислювальної логіки, а й комплексного підходу до обробки даних. У банківській діяльності джерела інформації часто мають неоднорідну структуру: дані надходять з різних підрозділів, у різних форматах, іноді з порушенням стандартів. Програмне забезпечення повинне бути здатне не лише обчислювати за формулою, а й знаходити необхідну інформацію у великій кількості файлів, перевіряти правильність форматування, структури, цілісність даних. Це вимагає глибокого розуміння як фінансових процесів, так і сучасних інструментів програмування, зокрема роботи з бібліотеками для обробки електронних таблиць, регулярними виразами, структурованими виключеннями.

Програмне забезпечення повинне бути зручним для кінцевого користувача. Тому важливо передбачити інтуїтивно зрозумілий графічний інтерфейс, де усі дії, від вибору файлу до виводу результату, виконуються в кілька кроків. Використання бібліотек для інтерфейсу дозволяє зробити застосунок зрозумілим навіть для працівників без технічної підготовки. Це суттєво підвищує практичну цінність розробки та її готовність до використання в реальних умовах.

1.2. Аналіз існуючих рішень та методів розробки програмного забезпечення для роботи з фінансовою та статистичною звітністю

Сучасна розробка програмного забезпечення для роботи з фінансовою та статистичною звітністю вимагає не лише глибокого розуміння предметної області, але й володіння актуальними технологіями, архітектурними підходами та інструментами, що забезпечують ефективність, надійність і гнучкість систем. У банківській сфері, яка є однією з найбільш регламентованих та критичних з точки зору точності даних, вимоги до програмного забезпечення особливо високі.

З технічного погляду, першим важливим напрямком у побудові таких систем є використання ETL-пайплайнів — процесів, що забезпечують витяг

(Extract), трансформацію (Transform) та завантаження (Load) даних [17]. Це може бути особливо корисним у випадку, коли фінансова звітність формується на підставі багатьох джерел: Excel-файлів, SQL-баз даних, зовнішніх API або навіть вручну введених даних. У таких випадках розробники часто застосовують бібліотеки Python, зокрема pandas для обробки табличних даних, або інструменти типу Apache, Airflow чи Talend, які дозволяють будувати гнучкі та автоматизовані конвеєри даних [1]. Для українських банків особливо актуально вміти працювати з файлами у форматах НБ — наприклад, XLS, XML, DBF, які мають специфічну структуру, часто із зашифрованими умовами валідації.

Другим ключовим методом є побудова мікросервісної архітектури. На практиці це означає, що кожен елемент системи, наприклад розрахунок депозитного ліміту, генерація звіту за формою 611 або перевірка коректності звітного періоду, реалізується у вигляді окремого сервісу, що працює автономно, має власну логіку та інтерфейс взаємодії (наприклад, REST API). Такий підхід дозволяє масштабувати систему, легко оновлювати окремі її частини та організовувати паралельну обробку — що дуже важливо в періоди пікового навантаження, таких як кінець кварталу або року. Мікросервіси часто реалізовуються за допомогою Python-фреймворків FastAPI або Flask, що дозволяє створювати окремі незалежні сервіси з чітко визначеними інтерфейсами для обміну даними між ними.

Ще одним популярним рішенням є автоматизоване генерування звітів у відповідних форматах. Зокрема, це може бути автоматичне формування файлів Excel (через бібліотеки openpyxl, xlsxwriter), PDF (за допомогою reportlab, weasyprint) або навіть XML за заданим шаблоном [2]. У більш складних системах шаблони звітів зберігаються окремо – наприклад, у YAML або JSON-файлах, де задані формули, назви полів і перевірки. Це дозволяє змінювати структуру або логіку звіту без переписування коду, що значно підвищує гнучкість рішення.

Окрім цього, важливою складовою сучасних систем є візуалізація даних — створення зручних аналітичних панелей, де користувач може бачити динаміку зміни показників, ризики, прогнози. Тут часто використовують BI-платформи —

такі як Power BI, Tableau або Qlik, які інтегруються з джерелами даних і дозволяють створювати інтерактивні дашборди. Для банківської звітності це особливо актуально при роботі з показниками ліквідності, резервів, ризиків, де аналітика має не менш важливе значення, ніж звітність. Крім того, не варто забувати про готові програмні продукти, які широко використовуються в банківському середовищі. Зокрема, системи типу SAP S/4HANA, які підтримують інтеграцію з усіма фінансовими процесами, забезпечують відповідність стандартам МСФЗ (міжнародним стандартам фінансової звітності) і можуть працювати з великими обсягами даних у реальному часі. Такі рішення часто інтегруються з внутрішніми системами банку через API або сервіси обміну даними. Іншим прикладом є Microsoft Dynamics 365 Finance, що дозволяє створювати звіти, вести бюджетування та управляти фінансовими процесами як у традиційній бухгалтерії, так і за міжнародними стандартами. Його перевагою є гнучка інтеграція з Power BI, Office 365 та Azure.

Один із найкращих прикладів вдалого програмного забезпечення, створеного з метою автоматизації всіх ключових бізнес-процесів у банківській сфері: від обслуговування клієнтів до формування обов'язковою звітності перед регулятором — програмний комплекс SrBank та АРМ. SrBank — це сучасна автоматизована банківська система, розроблена українською компанією Soft Review [3]. На сьогоднішній день ця система широко використовується для в банківських установах України як на рівні центрального апарату, так і в регіональних відділеннях.

Система SrBank побудована на власній технологічній платформі SrCoreSystem із використанням фреймворку SrFTP, який дозволяє забезпечити уніфікований підхід до реалізації всіх бізнес-модулів системи. В основі бази даних використовується рішення від Oracle, що гарантує високу продуктивність, надійність і масштабованість.

Особливістю архітектури є модульність: кожен окремий напрям банківської діяльності реалізується через власний функціональний модуль, що

спрощує обслуговування та розширення системи, а також дозволяє банку швидко адаптувати систему до змін законодавства чи внутрішніх процесів.

Комплекс SrBank охоплює багато напрямів банківської діяльності:

- Обслуговування рахунків клієнтів;
- Депозитні та кредитні операції;
- Облік операцій банківськими картами;
- Операції на міжбанківському валютному та грошовому ринку;
- Облік діяльності з цінними паперами;
- Казначейські операції та управління ліквідністю;
- Внутрішня діяльність та підтримка обліку МСФЗ;
- Підтримка інтеграції з платіжними системами (SWIFT, REUTERS, СЕП НБУ).

SrBank є чудовим прикладом того, як має працювати добре структуроване ПЗ у сфері фінансів. Вона поєднує в собі використання сучасних СУБД, модульний підхід до розробки, можливість централізованої підтримки, ведення логування та аудиту всіх дій користувачів. Також можна відмітити централізацію обліку та контролю, адаптивність до змін законодавства, модульність та можливість швидкого впровадження нових функцій, підтримка декількох ролей користувачів з різними правами доступу та системи резервного копіювання та аварійного відновлення.

АРМ — це спеціалізоване програмно-апаратне рішення, що забезпечує співробітнику банку виконання конкретних функціональних задач у межах визначеної ролі або підрозділу [4].

У банківській сфері АРМ зазвичай розробляються для виконання рутинних і критично важливих операцій, пов’язаних із обробкою документів, звітністю, платіжними дорученнями, перевіркою нормативів, формуванням файлів для СЕП (Системи електронних платежів), підготовкою фінансової звітності, моніторингом ризиків тощо. Типовий приклад — АРМ бухгалтера, АРМ економіста або АРМ спеціаліста з фінансової звітності. Усі вони мають інтерфейс, що адаптований до задач конкретного користувача, доступ до

потрібних модулів, довідників і звітів, а також можуть включати інтеграцію з іншими банківськими інформаційними системами (як-от SrBank, внутрішні бази даних, СЕП НБУ, СУБД).

Архітектура АРМ побудована на основі клієнт-серверної моделі, де клієнтська частина — встановлюється безпосередньо на комп'ютер користувача, а серверна частина — відповідає за зберігання даних, обробку запитів, зв'язок з іншими системами. Дане програмне забезпечення підключене до електронного документообігу, формує регламентовані звітності та має зрозумілий інтерфейс.

Технології, які зазвичай використовуються для таких електронних систем:

- Мова програмування: C# (.NET Framework), Python;
- Графічний інтерфейс: WinForms/WPF, tkinter;
- База даних: MS SQL Server, Oracle або Access/SQlite;
- Інтеграція: SOAP/REST API та протоколи FTP/SFTP.

Перед розробкою програмного забезпечення було проаналізовано мови C# та Python та було виявлено сильні та слабкі сторони кожного з них. Підхід із використанням C# вирізняється високою продуктивністю та інтеграцією в екосистему Microsoft, що є значною перевагою для банків, де основні обчислення здійснюються на основі Excel-файлів. Завдяки наявності спеціалізованих бібліотек, таких як Interop або EPPlus, C# дозволяє працювати з Excel із збереженням усіх форматів, формул і макросів, що часто є критично важливим. Крім того, можливість створювати функціональні графічні інтерфейси з використанням WPF або WinForms дозволяє адаптувати програму під потреби кінцевого користувача та забезпечити зручність у роботі. Однак обмеженням цього підходу є його залежність від операційної системи Windows, складність розгортання та менша гнучкість у випадках, коли потрібно швидко змінити бізнес-логіку або структуру обробки даних без перекомпіляції.

З іншого боку, підхід на основі Python забезпечує значно вищу гнучкість, простоту розробки та можливість швидкого прототипування. Мова має велику кількість бібліотек для обробки табличних даних, зокрема pandas, openpyxl, NumPy, що робить її надзвичайно зручною для реалізації алгоритмів аналізу,

перевірки умов, обчислень та автоматичної генерації звітів. Python також дозволяє легко масштабувати обробку даних, реалізовувати ETL-процеси й інтегруватися з іншими системами через API. Крім того, Python є кросплатформеним, що відкриває можливість розгортання рішення не лише на Windows, а й на Linux чи macOS. Водночас варто визнати, що інтерпретований характер мови та залежність від зовнішніх бібліотек знижують її продуктивність при обробці великих обсягів даних порівняно з C#, а можливості побудови інтерфейсів за допомогою базових засобів (наприклад, Tkinter) не завжди відповідають корпоративним стандартам якості.

1.3. Висновок до першого розділу. Постановка задачі

У першому розділі було здійснено аналіз сучасних підходів до автоматизації фінансової та статистичної звітності в банківських установах. Зокрема, було розглянуто як загальні принципи побудови подібного програмного забезпечення, так і приклади реальних рішень, які вже використовуються в практиці банків, зокрема система SrBank та автоматизовані робочі місця (АРМ). Ці дослідження дозволили виявити як переваги, так і обмеження існуючих підходів, а також окреслити потребу в розробці гнучких, адаптивних та спеціалізованих інструментів, які краще відповідають специфіці повсякденної роботи з фінансовими даними.

Сучасна банківська діяльність вимагає високої точності обчислень, дотримання нормативних вимог, зокрема вимог Національного банку України та Міжнародних стандартів фінансової звітності (МСФЗ), а також здатності обробляти значні обсяги даних у стислий термін. У цьому контексті автоматизація процесів набуває не просто бажаного, а обов'язкового характеру. Важливу роль у цьому процесі відіграє саме програмне забезпечення, яке здатне значно зменшити ризик людських помилок, підвищити швидкість обробки даних та забезпечити надійність і валідацію результатів.

Однією з ключових проблем, що спостерігається в щоденній практиці, є необхідність обробки великих Excel-таблиць, які містять фінансові показники, нормативи, розрахункові формули тощо. У багатьох банках ці процеси досі виконуються вручну, що призводить до втрати часу, зниження продуктивності працівників та зростання ймовірності помилок. При цьому існуючі системи, як-от SrBank, хоч і мають модульну структуру та широкий функціонал, часто не дозволяють здійснювати нестандартні або індивідуальні розрахунки без додаткових програмних рішень. Саме тому виникає потреба у створенні допоміжного внутрішнього інструменту, який буде здатен гнучко адаптуватися до внутрішніх процедур банку та виконувати спеціалізовані завдання з мінімальними затратами часу та зусиль.

З технічної точки зору, сучасна розробка програмного забезпечення у банківській сфері базується на перевірених архітектурних підходах — таких як клієнт-серверна модель або мікросервісна архітектура. У межах обробки великих обсягів даних дедалі більшої популярності набувають ETL-процеси (Extract, Transform, Load), які забезпечують послідовний потік даних з джерела у цільову систему. Для реалізації таких процесів застосовуються спеціалізовані інструменти, зокрема Apache Airflow, Talend, а також більш гнучкі засоби на базі мови програмування Python, такі як бібліотеки pandas, openpyxl, NumPy, що дозволяють реалізувати як обчислення, так і автоматизовану валідацію результатів. Сам Python, як універсальний і простий у вивченні інструмент, демонструє високу ефективність у розробці швидких та надійних рішень для фінансових обчислень.

Отже, на основі проведеного аналізу сформульовано чітку задачу — розробити власне програмне забезпечення для автоматизованого розрахунку банківських лімітів (депозитних, облігаційних, періодичних) на основі даних із Excel-файлів. Це ПЗ повинно мати графічний інтерфейс, бути простим у використанні, надавати можливість вибору типу розрахунку, виконувати валідацію вхідних даних, а також формувати результати у вигляді, зручному для подальшої аналітики чи подачі до регуляторів. Обрана технологічна база —

Python разом із бібліотеками tkinter, pandas та openpyxl — дозволяє створити ефективне, стабільне та адаптивне рішення в межах реальних потреб банківської практики. Задача відповідає сучасним вимогам цифровізації, підвищення ефективності банківських процесів, а також демонструє, як за допомогою актуальних технологій можна вирішити конкретні прикладні завдання у сфері фінансів.

Загалом, підсумки першого розділу чітко окреслюють не лише актуальність теми, але й глибоке розуміння контексту проблеми. Вони стали підґрунтям для подальших етапів — проєктування, реалізації, тестування та впровадження ефективного прикладного рішення, що забезпечує високий рівень автоматизації, точності та продуктивності в обробці фінансових звітів. Таким чином, підготовлений аналітичний базис слугує логічною основою для переходу до наступного етапу — безпосередньої розробки програмного забезпечення.

РОЗДІЛ 2. АНАЛІЗ І ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ ТА ПОБУДОВИ ПЗ

2.1. Обґрунтування оптимального варіанта реалізації завдання кваліфікаційної роботи

Банки потребують гнучких, масштабованих та адаптивних рішень для автоматизації звітних процесів. Завдання, що поставлено в межах даної кваліфікаційної роботи — створення програмного забезпечення для автоматизованого розрахунку фінансових показників (депозитних, облігаційних, нормативних лімітів), — вимагає врахування кількох ключових аспектів: типу вхідних даних (переважно Excel), вимог до точності обчислень, потреби у зручному інтерфейсі та можливості масштабування в межах внутрішньої IT-інфраструктури банку. З огляду на ці умови, було прийнято зважене рішення реалізовувати ПЗ із використанням мови програмування Python та її відповідних інструментів і бібліотек.

Одним із головних чинників вибору Python стало те, що ця мова широко застосовується для автоматизації задач, обробки табличних даних та побудови інтерфейсів. На відміну від компільованих мов, таких як C++ чи навіть C#, Python має інтерпретований характер, що дозволяє швидко тестувати логіку, змінювати модулі без тривалого етапу компіляції, і легко масштабувати функціонал. Це надзвичайно важливо в умовах, коли банківські розрахунки потребують постійного оновлення формул, методик або логіки в залежності від змін регуляторної бази або бізнес-потреб.

Особливу роль у реалізації поставленого завдання відіграє робота з Excel-файлами, які залишаються основним способом обміну фінансовими даними в банківському середовищі. Багато вихідних таблиць мають складну структуру: з кількома листами, злитими комірками, формулами, заголовками, прихованими рядками чи колонками. Для коректної роботи з такими документами використовується бібліотека openpyxl, яка дозволяє зчитувати, змінювати та створювати Excel-файли у форматі .xlsx, зберігаючи всю структуру документа.

Це забезпечує високу точність і відповідність вихідному вигляду, що критично важливо для звітності, яка надалі використовується в аудиторських перевірках або подається до Національного банку України.

Для обробки і аналізу даних було обрано бібліотеку pandas — одну з найпотужніших бібліотек для роботи з табличними даними в Python. Вона дозволяє ефективно обробляти великі масиви інформації, виконувати фільтрацію, агрегацію, об'єднання таблиць, перетворення форматів, обчислення складних фінансових показників за формулами. У контексті банківської діяльності, pandas дозволяє реалізувати, наприклад, моделі обрахунку залишків, відсотків, середньозважених значень або виконання умовних перевірок — все це є типовими щоденними задачами фінансових аналітиків.

Оскільки одним із завдань є забезпечення зручної роботи користувача, було прийнято рішення реалізувати графічний інтерфейс користувача за допомогою Tkinter — стандартного інструменту побудови GUI у Python [5]. Незважаючи на свою простоту, Tkinter дозволяє створити повноцінне віконне середовище з кнопками, меню, вкладками, списками, повідомленнями про помилки, індикаторами завантаження тощо. У рамках кваліфікаційної роботи цей інструмент використовується для створення головного вікон програм, введення параметрів обчислень, завантаження Excel-файлів, виводу результатів, а також обробки логіки взаємодії користувача з системою.

Крім того, перевагою Python є широка підтримка логування, тестування та створення документації. У межах кваліфікаційної роботи передбачається впровадження системи логування на основі стандартного модуля logging, що дозволяє фіксувати події в роботі програми — від помилок користувача до системних винятків і результатів обчислень. Це значно полегшує відлагодження програми та підтримку в майбутньому.

Структурою програмного продукту, було обрано інкрементну модель розробки, що передбачає поетапне додавання функціоналу. Такий підхід дозволяє реалізовувати робочі версії на ранніх етапах розробки, тестувати їх на реальних даних і вносити зміни без ризику для всієї системи. Це особливо зручно

в умовах практичного банківського середовища, де кінцеві користувачі, працівники відділу фінансової звітності, можуть дати зворотний зв'язок і впливати на остаточну реалізацію логіки програми.

Варто підкреслити, що саме Python як основна мова реалізації обраної задачі забезпечує найбільшу гнучкість, низький поріг входження, велику кількість доступної документації та прикладів, а також активну спільноту, що постійно розвиває інструменти. Крім того, Python відкриває шлях до майбутнього розширення функціоналу, наприклад, інтеграції з базами даних (PostgreSQL, SQLite), створення веб-інтерфейсів (через Flask або Django), додавання елементів автоматичного аналізу чи валідації, побудови графіків, автоматизованого надсилання звітів тощо.

Результатом обґрунтованого аналізу наявних технічних альтернатив було визначено, що розробка ПЗ на основі Python із використанням бібліотек pandas, openpyxl, Tkinter є найоптимальнішим рішенням для реалізації цілей кваліфікаційної роботи. Це дозволяє досягти балансу між функціональністю, простотою підтримки, гнучкістю у зміні логіки обчислень і адаптацією до практичних потреб фінансового підрозділу банку.

2.2. Створення технічних завдань для обох програмних забезпечень

Технічне завдання для ПЗ, що розраховує депозитні ліміти та ліміт на купівлю ОВДП:

Назва проєкту: програмноне забезпечення для автоматизації розрахунків банківських депозитних лімітів, ОВДП та нормативу Н6.

Мета проєкту: створити ПЗ, що дозволяє знаходити відповідні файли звітності у папці, обробляти дані з файлів Excel відповідно до заданих критеріїв, виконувати фінансові розрахунки (депозитні/облігаційні ліміти та норматив Н6), забезпечити логування операцій та розрахувати значення певного періоду з автоматичним проходженням по робочим дням [16].

Загальні вимоги до функціоналу:

- Розрахунок дат для знаходження потрібних файлів за заданими шаблонами, урахування специфіки днів;
- Обробка Excel-файлів звітності, автоматичний пошук файлів у вказаній директорії за шаблоном назви файлу, що містить дату;
- Розрахунок депозитного ліміту на актуальну дату;
- Розрахунок депозитних лімітів за період;
- Розрахунок ліміту ОВДП;
- Логування.

Таблиця 2.1

Функціональні вимоги до програмного забезпечення

№	Функція	Вхідні дані	Вихідні дані	Опис
1	calculate_dates	(ДД.ММ.РРРР)	date_minus_3, date_minus_263	Обчислює дати для формул
2	file_finder_plus_ deposit_calculator	Папка, дати -3, -263	Число (депозитний ліміт)	Аналізує файли 21X/C5X
3	file_finder_plus_ bond_calculator	Папка, дата	Число (ліміт облігацій)	Аналізує звіти 20X
4	file_finder_plus_ deposit_limit_period	Папка, початок/кінець періоду	Папка, початок/кінець періоду	Генерує звіт лімітів за період

Нефункціональні вимоги:

- Мова реалізації: Python 3.10+;
- Бібліотеки: pandas, openpyxl, re, datetime, logging, os, tkinter;
- Формат вхідних файлів: .xlsx (Excel 2007+);
- Платформа запуску: Windows;

- Використання тільки локальних ресурсів (немає потреби у підключенні до інтернету);
- Швидкодія: Обробка одного дня — до 3 секунд на звичайному ПК.

Інтерфейс користувача повинен надавати можливість вибору папки з файлами, введення дати або періоду, результат розрахунку. Також програма повинна обробляти різні помилки, як розрахункові так і помилки користувача при введенні інформації. Будь-яка помилка повинна відображатися у файлі логів.

Структура проєкту наведена на рис. 2.1. Вона слугує базовим орієнтиром на який спирається побудова програмного забезпечення.

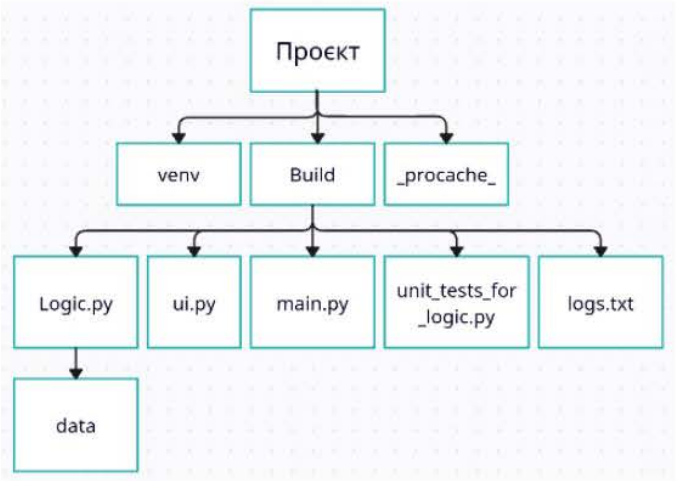

Рисунок 2.1 — Базова структура першого ПЗ

Технічне завдання для ПЗ, що розраховує банківський норматив Н6:

Назва проєкту: програмне забезпечення для автоматизації розрахунків нормативу Н6 з графічним інтерфейсом

Мета проєкту: Розробити програмне забезпечення, яке дозволяє автоматизувати обчислення нормативу Н6 на основі Excel-звітів. ПЗ повинно забезпечувати зручний графічний інтерфейс користувача (GUI), автоматичний пошук потрібних файлів за шаблоном, обробку даних, обробку помилок, а також підтримку вибору дати розрахунку.

Загальні вимоги до функціоналу:

- Визначення дати для розрахунку;

- Пошук Excel-файлів у вказаній директорії за шаблоном назв (що містять дату);
- Обробка вмісту Excel-файлів за вказаними формулами;
- Розрахунок нормативу Н6;
- Виведення результату в інтерфейсі;

Нефункціональні вимоги:

- Мова реалізації: Python 3.10+;
- Бібліотеки: pandas, openpyxl, re, datetime, os, tkinter;
- Формат вхідних файлів: .xlsx (Excel 2007+);
- Платформа запуску: Windows;
- Використання тільки локальних ресурсів (немає потреби у підключенні до інтернету);
- Швидкодія: Обробка одного дня — до 3 секунд на звичайному ПК.

Структура проєкту наведена на рис. 2.2. Вона слугує базовим орієнтиром на який спирається побудова програмного забезпечення.

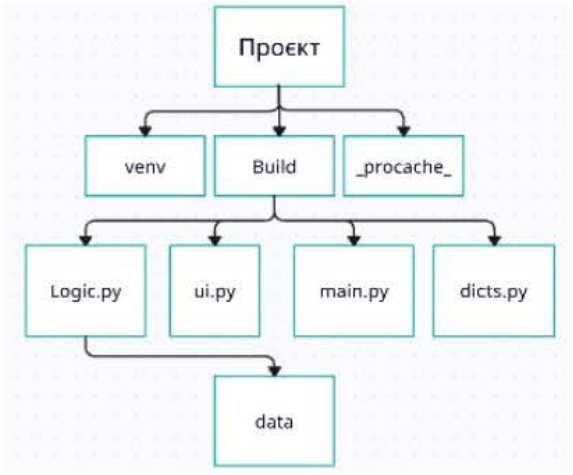

Рисунок 2.2 — Базова структура другого ПЗ

2.3. Висновок до другого розділу. Постановка задачі

У другому розділі було здійснено всебічне технічне обґрунтування вибору інструментів та підходів до реалізації програмного забезпечення для

автоматизації банківських розрахунків. Було розглянуто ключові вимоги до системи — точність обчислень, гнучкість налаштувань, зручність для кінцевого користувача, можливість масштабування, швидкодія та відповідність банківському середовищу, яке базується на роботі з Excel-файлами.

В результаті аналізу оптимальним варіантом було визнано використання мови програмування Python у поєднанні з бібліотеками pandas, openpyxl, re, datetime, logging та стандартними інструментами роботи з графічним інтерфейсом (tkinter). Python забезпечує швидку розробку, просте розширення функціоналу, зручне тестування та активну підтримку спільноти, що критично важливо в умовах змінного банківського та регуляторного середовища.

Особливу увагу приділено роботі з табличними даними, зокрема Excel-документами, які є основним джерелом фінансової звітності у банківській практиці. Застосування бібліотеки openpyxl дає змогу коректно працювати з різноманітними форматами таблиць. Використання pandas дозволяє реалізувати ефективну логіку обчислення фінансових показників на великих масивах даних, включаючи фільтрацію, агрегацію, об'єднання та складні математичні формули.

Розробка графічного інтерфейсу з використанням tkinter дозволяє створити інтуїтивно зрозуміле середовище взаємодії з користувачем, що сприяє зменшенню кількості помилок під час роботи та покращує загальний досвід експлуатації програми. Система логування на базі logging забезпечує відстеження усіх важливих подій, що є необхідним для діагностики проблем і підтримки стабільності системи [6].

Було сформовано 2 технічних завдання, які визначають функціональні та нефункціональні вимоги до програмного забезпечення, а також встановлюють чіткі критерії ефективності роботи. Інкрементна модель розробки, обрана як методологія, дозволяє реалізовувати програмне забезпечення поступово [14].

Проведене дослідження і підготовлені технічні завдання є надійною основою для практичної реалізації програмного продукту.

РОЗДІЛ 3. ВИРІШЕННЯ ЗАДАЧІ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РОЗРАХУНКІВ БАНКІВСЬКИХ ДЕПОЗИТНИХ ЛІМІТІВ, ОВДП ТА НОРМАТИВУ Н6

3.1. Опис програмного забезпечення, що розраховує депозитні ліміти за актуальну дату або період та ліміт на купівлю ОВДП. Опис бібліотек та взаємодії між модулями. Аналіз документації, структури звітів та формул НБУ.

На етапі підготовки до розробки програмного забезпечення було приділено особливу увагу вивченню нормативної документації Національного банку України, що регламентує порядок подання банківської звітності у вигляді електронних таблиць. Основна мета цього етапу полягала у розумінні структури, формату та логіки заповнення звітних форм, а також у точному відтворенні алгоритмів обчислення ключових фінансових показників [20].

Було приділено увагу Excel-файлам з назвами, які починаються на шаблони 21X, C5X та 20X. Ці документи містять звітні дані, що подаються банками до НБУ, і охоплюють широкий спектр інформації, включаючи залишки за рахунками, типи інструментів, види операцій, дати операцій тощо. Вивчення структури цих файлів було необхідним для подальшого розрахунку депозитних лімітів та лімітів на купівлю ОВДП.

Файли з префіксом 21X містять звітність щодо залишків за депозитами, обсягів коштів за видами рахунків і використовуються для розрахунку депозитних лімітів [7]. Файли C5X застосовувались у старіших періодах, і хоча їх структура близька до 21X, певні елементи відрізняються, що було враховано в логіці обробки [8]. Файли 20X призначені для відображення операцій із цінними паперами, у тому числі державними облігаціями, і слугують джерелом для розрахунку лімітів на купівлю облігацій відповідно до встановлених нормативів [9].

В ході аналізу було розшифровано призначення ключових колонок, що містять коди операцій (наприклад, R020, R030, S170, T070, T100 тощо), а також

особливості структури рядків, які відповідають тим чи іншим видам банківських операцій [15]. Важливим аспектом стала ідентифікація конкретних умов, при яких певні значення мають враховуватись у фінансових розрахунках (наприклад, для обчислення депозитного ліміту враховуються лише записи з певними кодами валют, рівнями підпорядкованості тощо).

Саме такий попередній аналітичний етап дозволив сформувати точне уявлення про правила обробки даних та створити надійні програмні модулі, які відповідають актуальним вимогам банківської звітності та враховують специфіку періодів подання файлів.

Детальний опис керуючого модуля main.py:

У програмному файлі main.py використовуються три ключові бібліотеки, кожна з яких відіграє свою специфічну роль у функціонуванні програмного забезпечення (Рисунок 3.1).

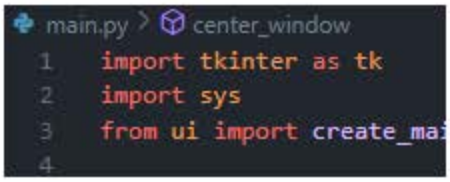

Рисунок 3.1 — Три ключові бібліотеки

Перша з них — це tkinter, стандартна бібліотека Python, яка є основою для побудови графічного інтерфейсу користувача. Завдяки їй створюється головне вікно програми, а також усі дочірні вікна, що відповідають за різні види розрахунків. tkinter надає розробнику можливість візуально структурувати застосунок: налаштувати розміри вікон, кольорову гаму, обробку подій при натисканні кнопок, закритті вікон тощо. У файлі main.py вона застосовується для створення головного вікна за допомогою tk.Tk(), створення додаткових вікон через tk.Toplevel(), а також для управління зовнішнім виглядом — фоновим кольором, заголовками вікон і положенням на екрані. Вся логіка перемикання між вікнами, приховування або знищення вже неактуальних елементів також реалізується за допомогою можливостей tkinter.

Друга бібліотека — це sys, яка також входить до складу стандартної бібліотеки Python. Вона використовується в програмі для системного завершення процесу, коли користувач закриває застосунок. Це дозволяє не лише знищити графічні елементи, але й гарантувати, що процес програми повністю завершиться і не залишиться у фоні. У main.py вона використовується в функції on_closing, яка викликається при закритті будь-якого з вікон, і забезпечує коректне завершення роботи всього застосунку.

Третій блок імпорту — це модуль ui, який представляє собою користувацький (тобто створений у рамках цього програмного проєкту) модуль, що містить функції побудови інтерфейсу. З нього імпортуються функції create_main_window, create_calculation_window, create_deposit_calculation_window, create_bond_limit_window та create_deposit_limit_period_window. Кожна з цих функцій відповідає за формування конкретного вікна або інтерфейсного елемента, згідно з логікою застосунку. Таким чином, ui дозволяє логічно відокремити структуру графічного інтерфейсу від основного коду, що керує поведінкою застосунку. Це підвищує модульність і читабельність програмного коду. У сукупності ці три бібліотеки забезпечують повноцінну реалізацію взаємодії користувача з програмним забезпеченням, включно з візуальним оформленням, логікою переходів між різними режимами обчислень і завершенням роботи програми.

Розроблене програмне забезпечення є десктопною прикладною системою, призначеною для автоматизації розрахунків депозитних лімітів, облігацій внутрішньої державної позики (ОВДП), а також нормативу Н6 згідно з вимогами до фінансової звітності в банківській сфері. Програма надає зручний користувацький інтерфейс для завантаження Excel-файлів, виконує автоматизовану обробку даних, виконує розрахунки за складними алгоритмами та зберігає результати для подальшого використання.

Програмне забезпечення побудоване за модульним принципом та реалізоване мовою програмування Python 3.10+ з використанням ряду перевірених бібліотек. Архітектура системи передбачає поділ логіки на такі

компоненти, як логіка обчислення розрахунків, інтерфейс користувача, керуючий файл, який відповідає за запуск програми та взаємодіями між графічним інтерфейсом. Також програма має додатковий модуль, який є окремим файлом для перевірки працездатності різних функцій, а саме файл з юніт-тестами.

Файл main.py у розробленому програмному забезпеченні відіграє ключову роль, оскільки є головною точкою запуску програми та координує взаємодію між різними графічними інтерфейсами (вікнами), що відповідають за обчислення банківських показників. Його основне призначення — ініціалізувати програму, створити головне вікно, керувати навігацією між підвікнами, забезпечити правильне закриття додатка, а також передавати дані між компонентами через глобальний стан додатка.

На початку файлу імпортуються необхідні модулі: стандартна бібліотека tkinter, що використовується для побудови інтерфейсу, та sys, яка забезпечує системні виклики (наприклад, завершення процесу). Також з файлу ui.py імпортуються функції, які відповідають за створення кожного з вікон користувацького інтерфейсу — це create_main_window, create_calculation_window, create_deposit_calculation_window, create_bond_limit_window та create_deposit_limit_period_window.

Далі йде допоміжна функція center_window, яка відповідає за візуальну зручність користувача — вона динамічно обчислює координати розміщення вікна по центру екрана, враховуючи поточну роздільну здатність монітора. Завдяки цьому всі вікна (незалежно від розміру) автоматично відкриваються по центру екрана, що підвищує ергономіку програми.

Функція main() є ядром модуля. Вона створює головне вікно програми root із заголовком «Банківські ліміти», фоном у фірмовому помаранчевому кольорі (#ff8732), та викликає початкову функцію create_main_window, яка відображає стартовий інтерфейс користувача.

Одна з ключових частин main() — це створення словника app_state (Рисунок 3.2), який є своєрідним глобальним сховищем стану додатка. У цьому

словнику зберігаються всі активні вікна (як main_window, calculation_window, так і три типи розрахункових вікон), а також значення обраного користувачем типу ліміту (selected_limit_type). Це дозволяє програмі легко відстежувати, які вікна відкриті в даний момент, і забезпечувати правильну навігацію між ними без дублювання.

```
def main():

    root = tk.Tk()
    root.title("Банківські ліміти")
    center_window(root, 600, 400)
    root.config(bg='#ff8732')

    app_state = {
        "main_window": root,
        "calculation_window": None,
        "deposit_calculation_window": None,
        "bond_calculation_window": None,
        "deposit_limit_period_window": None,
        "selected_limit_type": None,
    }
```

Рисунок 3.2 — Словник app_state

Усередині main() визначено декілька вкладених функцій, які керують навігацією між вікнами:

- open_calculation_window(limit_type) — відкриває проміжне вікно для вибору виду розрахунку, приймає аргумент типу ліміту, який обрав користувач. При цьому головне вікно приховується (withdraw), а нове створюється як підвікно Toplevel;
- open_deposit_calculation_window() — створює вікно для розрахунку депозитного ліміту, приховуючи вікно вибору типу розрахунку;
- open_bond_calculation_window() — створює інтерфейс для обчислення ліміту на купівлю облігацій;
- open_deposit_limit_period_window() — формує інтерфейс для обчислення депозитного ліміту за певний період, з розширеним вікном.

Кожна з цих функцій перевіряє, чи вікно вже створене — якщо так, то старе знищується, щоб не дублювати ресурси. Потім створюється нове вікно із заданими параметрами, фоном і заголовком, і передається у відповідну функцію побудови інтерфейсу з файлу ui.py. Для прикладу можна ознайомитися з кодом функції open_calculation_window(limit_type) (Рисунок 3.3).

```
def open_calculation_window(limit_type):

    app_state["selected_limit_type"] = limit_type
    if app_state["calculation_window"]:
        app_state["calculation_window"].destroy()
    app_state["main_window"].withdraw()
    app_state["calculation_window"] = tk.Toplevel(root)
    center_window(app_state["calculation_window"], 600, 400)
    app_state["calculation_window"].config(bg='#ff8732')
    app_state["calculation_window"].title("Розрахунок")
    create_calculation_window(
        app_state["calculation_window"],
        app_state,
        open_deposit_calculation_window,
        open_bond_calculation_window,
        open_deposit_limit_period_window
    )
    app_state["calculation_window"].protocol("WM_DELETE_WINDOW", on_closing)
```

Рисунок 3.3 — Функція open_calculation_window(limit_type)

Особливу увагу приділено обробці повернення користувача назад. Наприклад, функція go_to_main_window() закриває всі відкриті вікна та повертає користувача до головного меню. Аналогічно, go_to_calculation_window() дає змогу повернутися до вікна вибору типу розрахунку. Це забезпечує логічну навігацію та послідовність взаємодії, уникаючи плутанини між етапами обчислення.

Була створена функцію on_closing(), яка забезпечує коректне завершення роботи програми. Вона знищує всі вікна, які могли бути відкриті, та викликає sys.exit() — таким чином програма гарантовано звільняє ресурси і не залишає процес у фоновому режимі (Рисунок 3.4).

```
def on_closing():

    if app_state["calculation_window"]:
        app_state["calculation_window"].destroy()
        app_state["calculation_window"] = None
    if app_state["deposit_calculation_window"]:
        app_state["deposit_calculation_window"].destroy()
        app_state["deposit_calculation_window"] = None
    if app_state["bond_calculation_window"]:
        app_state["bond_calculation_window"].destroy()
        app_state["bond_calculation_window"] = None
    if app_state["deposit_limit_period_window"]:
        app_state["deposit_limit_period_window"].destroy()
        app_state["deposit_limit_period_window"] = None
    root.destroy()
    sys.exit()
```

Рисунок 3.4 — Функція go_to_main_window()

Завершується виконання файлу запуском циклу root.mainloop(), який запускає основний цикл обробки подій інтерфейсу. Сам виклик main() здійснюється через стандартну конструкцію if __name__ == "__main__":, що гарантує, що код буде виконаний лише при безпосередньому запуску файлу, а не при імпорті як модуля.

Як результат, файл main.py виступає головним контролером поведінки інтерфейсу додатка. Він пов’язує воєдино створення вікон, переходи між ними, збереження стану та завершення програми. Його структура чітко організована, забезпечує розширюваність і підтримує гнучке доповнення нових функцій без порушення основної логіки.

Детальний опис модуля логіки розрахунків лімітів — logic.py:

У модулі logic.py використовується кілька зовнішніх і стандартних бібліотек, кожна з яких відіграє важливу роль у забезпеченні функціональності програми, що обробляє фінансові Excel-документи. Ось розгорнутий текстовий опис кожної з цих бібліотек:

Бібліотека pandas використовується для роботи з табличними даними та датами. У цьому модулі вона виконує дві ключові функції: по-перше, допомагає формувати діапазони робочих днів із використанням методу pd.bdate_range, що

критично важливо для правильного розрахунку фінансових показників у будні дні; по-друге, вона дозволяє зручно працювати з датами як об'єктами, що легко перетворюються, форматуються та обчислюються. Хоча основне зчитування Excel-файлів відбувається через іншу бібліотеку, pandas відіграє допоміжну роль у роботі з календарем.

Бібліотека datetime — це стандартний інструмент Python для роботи з датами й часом. Вона використовується для парсингу дат, які вводить користувач (наприклад, у форматі день.місяць.рік), а також для форматування дат у такий вигляд, який дозволяє легко шукати файли з відповідними назвами. Крім того, datetime дає змогу обчислювати попередні або наступні дати, що є важливим під час створення логіки обчислень фінансових лімітів у різні періоди.

Бібліотека openpyxl забезпечує можливість відкривати, читати та обробляти файли Excel формату .xlsx без потреби встановлювати Microsoft Excel. У модулі вона використовується безпосередньо через функцію load_workbook, яка відкриває вказаний Excel-файл для подальшого аналізу. За допомогою цієї бібліотеки програма ітерується по рядках таблиці, зчитує значення з конкретних клітинок і проводить обчислення згідно із заданими умовами.

Модуль os використовується для навігації по файловій системі. Зокрема, він дозволяє отримувати список файлів у зазначеній директорії, об’єднувати шлях до папки з іменами файлів, а також перевіряти наявність файлів, що відповідають заданому шаблону. Завдяки цьому можна динамічно знаходити потрібні файли за датами та назвами.

Бібліотека re, яка забезпечує роботу з регулярними виразами, використовується для пошуку та фільтрації файлів за шаблоном імені. За допомогою регулярних виразів програма ідентифікує, які саме файли відповідають потрібним датам (наприклад, починаються з "21X" або "C5X" і містять дату). Це дозволяє автоматизувати пошук потрібної звітності без ручного втручання.

Модуль logging відповідає за ведення журналу подій. У ньому налаштовано логування у файл logs.txt, причому з використанням кодування

UTF-8. Усі ключові дії, зокрема обчислення дат, знаходження файлів, обробка даних і результати обчислень, записуються в лог-файл. Це дозволяє не тільки відслідковувати хід виконання програми, але й зберігати історію розрахунків і можливих помилок для подальшого аналізу (Рисунок 3.5).

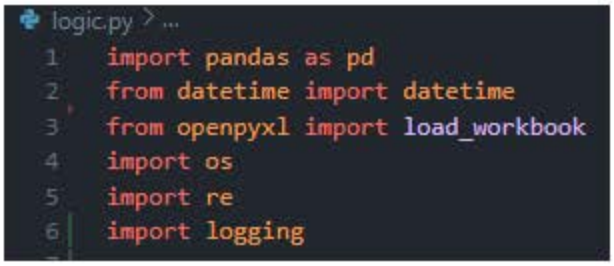

Рисунок 3.5 — Бібліотеки модуля logic.py

Модуль logic.py реалізує обчислювальну логіку програмного забезпечення, яке призначене для автоматизованої обробки звітних Excel-файлів та обчислення ключових фінансових показників для банківської діяльності. Він імпортує такі бібліотеки, як pandas, datetime, openpyxl, os, re та logging. pandas використовується для створення діапазону робочих днів і роботи з часовими мітками, datetime — для парсингу та форматування дат, openpyxl дозволяє зчитувати дані з Excel-файлів, os забезпечує навігацію по файловій системі, re дозволяє формувати регулярні вирази для пошуку потрібних файлів, а logging використовується для фіксації процесів обчислення, помилок і проміжних результатів у лог-файл logs.txt.

На початку модуля налаштовується система логування, яка веде журнал подій у файл із кодуванням UTF-8. Далі створюється глобальний словник dates, який зберігає ключові дати, зокрема дату мінус три робочих дні, мінус двісті шістдесят три робочі дні та шлях до робочої теки з файлами.

Функція calculate_dates приймає поточну дату у форматі рядка, перетворює її у об'єкт дати, обчислює робочі дні у відповідному діапазоні, і визначає дві цільові дати — одну за три дні до поточної, іншу — за 263 робочі дні до неї (Рисунок 3.6). Якщо поточна дата припадає на четвер, застосовується додаткове зміщення, що враховує вихідні або нестандартні робочі дні. Отримані значення

зберігаються у глобальному словнику dates і використовуються далі у функціях, які працюють з файлами. Всі ці обчислення записуються до логів.

```
def calculate_dates(current_date_str):
    global dates
    end_date = pd.Timestamp(datetime.strptime(current_date_str, '%d.%m.%Y'))
    workdays = pd.bdate_range(end_date - pd.DateOffset(days=1000), end_date)

    weekday = end_date.weekday()

    if weekday == 3:  # четверг
        dates['date_minus_3'] = (workdays[-4] - pd.Timedelta(days=2)).strftime('%d.%m.%Y')
        dates['date_minus_263'] = (workdays[-264] - pd.Timedelta(days=2)).strftime('%d.%m.%Y')
    else:
        dates['date_minus_3'] = workdays[-4].strftime('%d.%m.%Y')
        dates['date_minus_263'] = workdays[-264].strftime('%d.%m.%Y')

    logging.info(f'Вызвана calculate_dates: текущая дата = {current_date_str}, '
                 f'date_minus_3 = {dates["date_minus_3"]}, '
                 f'date_minus_263 = {dates["date_minus_263"]}')

    return dates['date_minus_3'], dates['date_minus_263']
```

Рисунок 3.6 — Функція calculate_dates

Функція file_finder_plus_deposit_calculator використовується для обчислення депозитного ліміту. Вона приймає шлях до теки з файлами, дату мінус три дні та дату мінус 263 дні. Виходячи з того, яка з цих дат припадає до чи після граничної дати 11.04.2024, вибирається шаблон для імені файлу. За допомогою регулярних виразів відбувається пошук відповідних файлів у вказаній теці. Після цього з обраних файлів зчитуються значення з конкретних стовпців: зокрема враховуються тільки ті рядки, де значення певних клітинок відповідають потрібним кодам та умовам. У разі відповідності даних умові, значення з поля T070 збираються і підсумовуються. Далі за заданою формулою виконується розрахунок депозитного ліміту. У цій формулі враховуються два значення — основне та базове (для старішої дати), а також коефіцієнти k і m. Результат округлюється до сотих і повертається як результат. У лог-файл записуються вхідні дати, результати читання з файлів та остаточне значення.

Функція file_finder_plus_bond_calculator виконує схожу операцію, але вже для розрахунку ліміту на облігації. Вона приймає дату від користувача, форматує

її у потрібний вигляд, і за допомогою регулярного виразу шукає файл, ім'я якого починається з «20X» і містить цю дату. Після знаходження відповідного Excel-файла функція зчитує дані з конкретних стовпців: коди for20, значення t100, коди ekb, а також значення дати в полі q007. Проводиться логіка перевірки умов — наприклад, чи є код B20055, чи дорівнює for20 значенням «008», «062», «011» або «055» — і відповідно до цих умов збираються значення. Вони накопичуються в окремі списки або змінні, після чого застосовується фінальна формула для обчислення облігаційного ліміту. Формула включає множення на коефіцієнт, віднімання накопичених значень та округлення. У випадку помилки вся інформація записується в лог.

Функція file_finder_plus_deposit_limit_period дозволяє автоматично обробити діапазон дат і вивести депозитні ліміти для кожної робочої дати в цьому діапазоні. Вона приймає початкову і кінцеву дати, обчислює всі робочі дні між ними, і для кожного дня викликає calculate_dates, а потім file_finder_plus_deposit_calculator. Усі отримані результати акумулюються у список, який потім об'єднується в єдиний текстовий звіт. Для кожної дати проміжний результат логуються.

Модуль logic.py виступає в ролі центрального ядра розрахункової логіки. Його функції пов'язані між собою через обмін параметрами: спочатку за допомогою функції calculate_dates генеруються дати, які потім передаються у функції, що шукають і обробляють відповідні Excel-файли. Всі результати фіксуються в лог-файлі для можливості аудиту або відстеження помилок. У підсумку, цей модуль забезпечує повністю автоматизовану логіку для складних розрахунків, базуючись на файлах звітності банку, при цьому зберігаючи прозорість і відтворюваність усіх дій завдяки логуванню.

Детальний опис інтерфейсного модуля ui.py:

У графічному інтерфейсі програмного забезпечення використовуються бібліотеки tkinter, tkinter.messagebox та tkinter.filedialog. Бібліотека tkinter є стандартною для Python і використовується для створення графічного інтерфейсу користувача. З її допомогою реалізовано основне вікно програми,

елементи управління (мітки, поля введення, кнопки, текстові поля), а також функціонал взаємодії користувача з додатком. Змінні типу StringVar зв'язуються з полями введення для зчитування та передачі даних до функцій логіки. Крім того, використано методи bind для додавання ефектів при наведенні на кнопки, що покращує зручність користування.

Модуль tkinter.messagebox застосовується для відображення діалогових вікон із повідомленнями про помилки, які можуть виникнути під час некоректного введення даних або обробки файлів. Це дозволяє оперативно інформувати користувача про неправильні дії, як-от відсутність введення дати, помилковий шлях до папки або некоректний формат даних.

Модуль tkinter.filedialog використовується для виклику стандартного вікна вибору папки, де зберігаються файли для обробки. Обраний шлях зберігається у змінну і надалі передається у функції, що відповідають за розрахунки.
Графічний інтерфейс інтегрований з функціями обробки даних, які реалізовані в окремому модулі логіки. Зокрема, використовуються функції file_finder_plus_bond_calculator та file_finder_plus_deposit_limit_period, які здійснюють обробку Excel-файлів та виконують розрахунки відповідно до обраного режиму. Отримані результати автоматично виводяться в інтерфейс для перегляду користувачем.

У цьому файлі реалізовано повноцінний графічний інтерфейс для трьох ключових розрахункових функцій, що стосуються банківських операцій: розрахунку депозитного ліміту на актуальну дату, розрахунку ліміту на придбання ОВДП та розрахунку депозитного ліміту за вказаний період. Усі три функціональні модулі реалізовані у вигляді окремих вікон на основі бібліотеки Tkinter, і пов'язані між собою через об'єкт app_state, який зберігає посилання на головне вікно програми. Це дозволяє легко перемикатися між вікнами, не втрачаючи контекст.

Кожне вікно містить поля для введення даних (дати або шляху до папки), кнопки керування, а також відображення результату. Дані з полів зчитуються за допомогою StringVar, після чого передаються у відповідну функцію логіки (яка

розташована в окремому модулі, не показаному у цьому файлі). Для розрахунку депозитного ліміту на конкретну дату використовується функція file_finder_plus_deposit_calculator, для розрахунку по ОВДП — file_finder_plus_bond_calculator, а для розрахунку депозитного ліміту за період — file_finder_plus_deposit_limit_period. Вони приймають вхідні параметри (дату або період і шлях до папки) та повертають результат, який виводиться у відповідне поле.

Кнопка "Розрахувати" в кожному вікні виконує відповідну функцію після перевірки коректності введених даних. Якщо дані некоректні або виникає помилка (наприклад, якщо відсутні файли чи неправильна дата), з'являється вікно з повідомленням про помилку (messagebox.showerror). Кнопка "Назад" у кожному вікні закриває поточне вікно і повертає до головного меню, яке було приховано (withdraw) під час відкриття підвікна. Це реалізовано через ключ app_state["calculation_window"].

Кожна кнопка має ефекти наведення (<Enter>, <Leave>) для візуального відгуку користувачу — фоновий колір змінюється при наведенні. Це покращує UX, робить інтерфейс динамічним і чуйним.

Дизайн інтерфейсу витриманий у помаранчевих і сірих тонах, із шрифтами Arial Rounded MT Bold, що забезпечує зручність зчитування тексту. Всі елементи розміщені вручну через place, що забезпечує точний контроль над позиціонуванням. Для виводу довших результатів (у випадку розрахунку за період) використовується Text з вертикальним скролбаром.

3.2. Опис програмного забезпечення, що розраховує банківський норматив Н6. Опис бібліотек та взаємодії між модулями. Аналіз документації, структури звітів та формул НБУ.

Програмне забезпечення для розрахунку нормативу Н6 базується на вимогах та регламентах Національного банку України, зокрема на Постанові №351 від 30 червня 2016 року (із змінами), яка регламентує порядок розрахунку

економічних нормативів комерційними банками. Норматив Н6 визначає максимальний розмір кредитного ризику на одного контрагента і розраховується як відношення загального обсягу заборгованості контрагента до регулятивного капіталу банку, помножене на 100%. Таким чином, Н6 є показником концентрації ризику, що дозволяє контролювати, наскільки великий ризик банк несе при роботі з окремими контрагентами. За нормативними вимогами, значення Н6 зазвичай не повинно перевищувати 25%, хоча цей поріг може коригуватись залежно від рішень регулятора.

Для розрахунку цього показника програмне забезпечення використовує два основні звітних файли: 01Х та А7Х. Файл 01Х (відповідно до форми 351) містить перелік активних операцій з контрагентами банку. У ньому відображені ключові дані щодо кожного контрагента, включаючи його код, найменування, загальний обсяг заборгованості, а також інші параметри, необхідні для фільтрації та класифікації ризиків. Цей файл, як правило, представлений у форматі Excel (.xlsx) з іменуванням, що включає дату звітності. Програма зчитує таблицю з цього файлу, аналізує кожен рядок та визначає релевантні суми заборгованості, які беруть участь у розрахунку нормативу.

Інший файл — А7Х (форма 611) — містить дані про регулятивний капітал банку [10]. Саме це значення використовується як знаменник у формулі обчислення Н6. Звіт А7Х також представлений у форматі Excel, і програма шукає в ньому відповідну клітинку або рядок, де вказане числове значення капіталу, яке відповідає звітній даті. Після отримання обох складових — суми заборгованості та значення регулятивного капіталу — програма розраховує Н6.

Програмне забезпечення реалізоване на мові Python 3.10 з використанням ряду бібліотек, таких як pandas для обробки табличних даних, openpyxl для роботи з Excel-файлами на рівні клітинок, datetime для коректного обрахунку дат, re для використання регулярних виразів при пошуку шаблонів у назвах файлів, os для доступу до файлової системи та tkinter для створення графічного інтерфейсу. Модулі програми взаємодіють між собою через чітко структуровану архітектуру: модуль інтерфейсу дозволяє користувачу вибрати папку з файлами

та ввести дату розрахунку, після чого модулі логіки обробки викликають відповідні функції для зчитування, аналізу та обчислення показника. Усі дії користувача та можливі винятки фіксуються в лог-файлі, що дозволяє відслідковувати роботу програми та діагностувати помилки.

Детальний опис керуючого модуля main.py:

Цей файл — це класичний вхідний скрипт (entry point) для запуску програми з графічним інтерфейсом, яка реалізована в модулі ui. Це основний запускний файл, який використовується для старту всього застосунку. Саме його відкриває користувач для початку роботи з програмою

Детальний опис модуля інтерфейсу ui.py:

У цьому файлі використовується бібліотека tkinter, яка є стандартною бібліотекою мови Python для створення графічного інтерфейсу користувача. Вона дозволяє формувати вікна з елементами управління, такими як кнопки, текстові поля, мітки тощо. Завдяки tkinter можна легко створити простий GUI-додаток без додаткових залежностей.

Файл реалізує основне вікно програми, яка призначена для розрахунку банківського нормативу Н6. У графічному інтерфейсі користувач бачить поле для введення дати, поле для вибору папки, кнопку для відкриття діалогу вибору папки, а також кнопку для запуску розрахунку та поле для виводу результату.

При запуску функції start_app створюється головне вікно програми з фіксованими розмірами, яке розміщується по центру екрана. Додаються мітки з відповідним текстом та естетичним оформленням, а також текстові поля для введення дати та шляху до папки. Кнопка вибору папки дозволяє відкрити вікно провідника, після чого шлях автоматично вноситься до відповідного поля. Для покращення взаємодії з користувачем реалізовано зміну кольору кнопок при наведенні курсора.

При натисканні кнопки "Розрахувати" викликається функція calculate, яка передає дані з полів введення до логіки розрахунку. Результат обчислення нормативу Н6 автоматично з’являється у відповідному полі. Програма також

обробляє закриття вікна, використовуючи стандартну подію WM_DELETE_WINDOW, що забезпечує коректне завершення роботи програми.

Детальний опис модуля з логікою розрахунку Н6 — logic.py:

На початку модуля здійснюється підключення необхідних бібліотек. Зокрема, використовується os для роботи з файловою системою, datetime — для обробки дат, re — для застосування регулярних виразів, openpyxl — для відкриття та обробки Excel-файлів, а також pandas для ефективної роботи з табличними даними. Ці бібліотеки дозволяють реалізувати гнучку та масштабовану логіку обробки документів у різних форматах і з різними схемами розміщення даних.

Основна частина модуля полягає у пошуку файлів у вказаній директорії, які відповідають шаблонам, залежно від обраної користувачем дати. Застосовуючи регулярні вирази, модуль ідентифікує відповідні Excel-документи, які містять дані про активи, пасиви, залишки на рахунках тощо. Кожен з файлів обробляється поетапно, відкривається через openpyxl, після чого з конкретних аркушів зчитуються значення комірок згідно з заданими координатами або умовами. Дані збираються в словники — dict_liabilities, dict_assets та moneybox_L, які структурують фінансову інформацію за категоріями.

На основі цих словників обчислюються дві ключові змінні: passiv і activ. Змінна passiv представляє собою сукупний обсяг банківських зобов'язань, який розраховується як сума низки позицій із різних категорій пасивів: кошти клієнтів, депозити, кредити, міжбанківські ресурси та інші джерела. При цьому враховуються як основні статті, так і специфічні комбінації вкладених підкатегорій, що вказані через багаторівневі індекси.

Змінна activ відображає обсяг активів банку, які вважаються прийнятними для покриття зобов'язань згідно з методологією нормативу Н6. До неї входять грошові кошти, кредити, інвестиції та інші ліквідні активи, визначені в словнику dict_assets, а також залишки на рахунках із moneybox_L. Як і у випадку з пасивами, розрахунок ведеться через складання числових значень за вказаними ключами та вкладеними рівнями структури.

Після отримання значень активів і пасивів виконується фінальна частина обчислення — розраховується сам норматив Н6 як відношення активів до пасивів, помножене на 100, що дає у відсотках значення нормативу. Отриманий результат як кінцевий вихід функції.

3.3. Результат розробки програмного забезпечення, інтерфейс програм. Створення інсталятора програми, яка обчислює ліміти за допомогою ПЗ Inno Setup Compiler

Інтерфейс програми для розрахунку депозитних лімітів та лімітів на купівлю ОВДП:

Головне вікно програми, яке дозволяє користувачу вибрати тип ліміту, який буде розраховуватися (Рисунок 3.7).

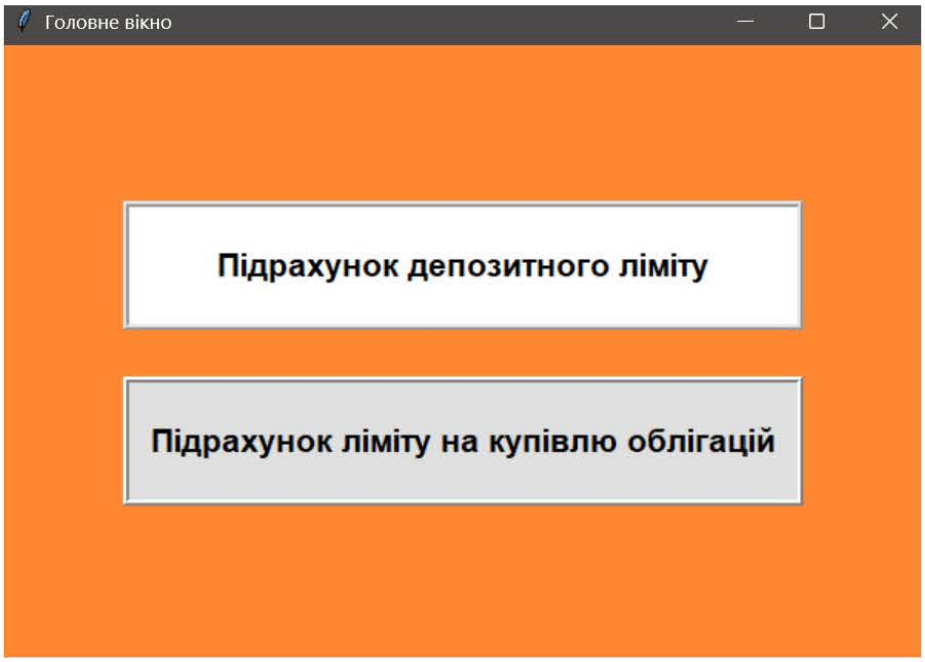

Рисунок 3.7 — Головне вікно програми

Вікно «Розрахунок», яке дозволяє користувачу розрахувати депозитний ліміт на актуальну дату або за будь який проміжок часу (Рисунок 3.8).

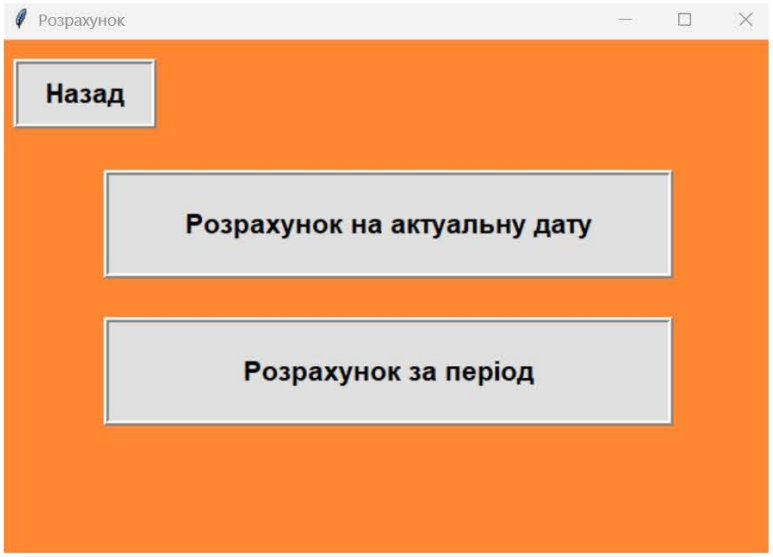

Рисунок 3.7 — Вікно «Розрахунок» для вибору типу розрахунку

Воно дозволяє користувачеві ввести актуальну дату, отримати інформацію про те, файли з якими датами мають бути в папці для правильного розрахунку депозитного ліміту. Для розрахунку депозитного ліміту використовуються 2 типи файлів: 21X та C5X. Елемент Textbox поряд з елементом Label з текстом «-3 дні» показує те, яка має бути дата в назві файлу 21X, а Елемент Textbox поряд з елементом Label з текстом «-263 дні» показує те, яка має бути дата в назві файлу C5X (Рисунок 3.8).

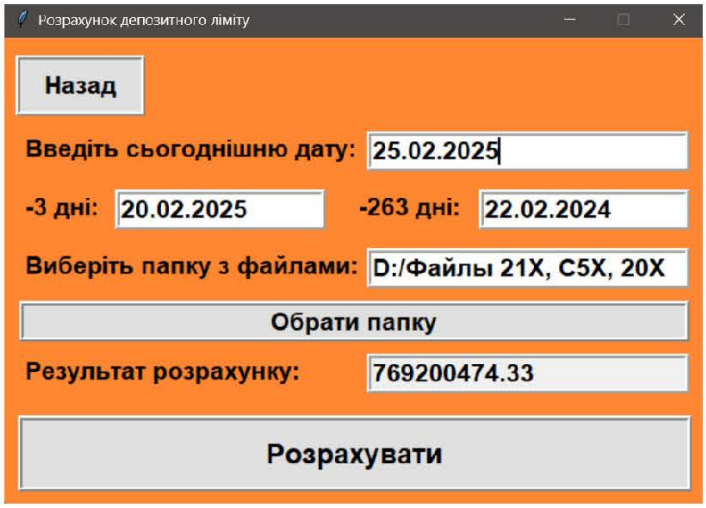

Рисунок 3.8 — Вікно розрахунку депозитного ліміту на актуальну дату

Вікно «Розрахунок депозитного ліміту» за період. Воно дозволяє

користувачеві початкову дату та кінцеву та отримати інформацію про депозитні ліміти за весь проміжок часу. Це вікно програми використовує практично ті ж самі елементи та логіку, як и вікно для розрахунку депозитного ліміту на актуальну дату. Користувач отримує інформацію в елементі Textbox зі Scrollbar у вигляді списку (див. Рисунок 3.9).

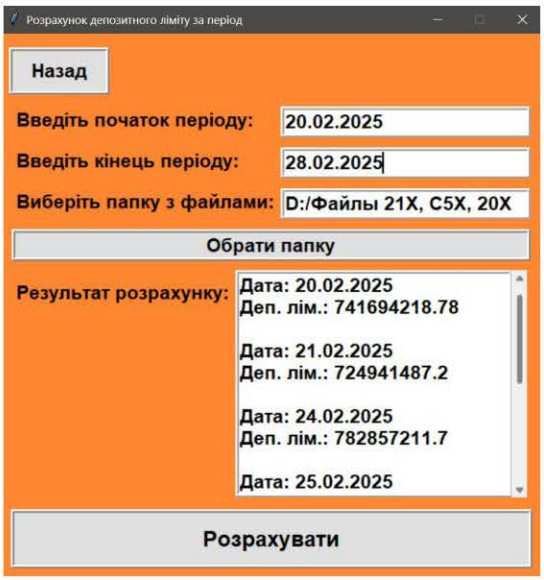

Рисунок 3.9 — Вікна для розрахунку депозитного ліміту за період

Вікно «Розрахунок ліміту на купівлю облігацій» дозволяє користувачу розрахувати ліміт на купівлю облігацій шляхом введення дати да вибору потрібної папки з файлом 20X, який використовується при розрахунку (Рисунок 3.10).

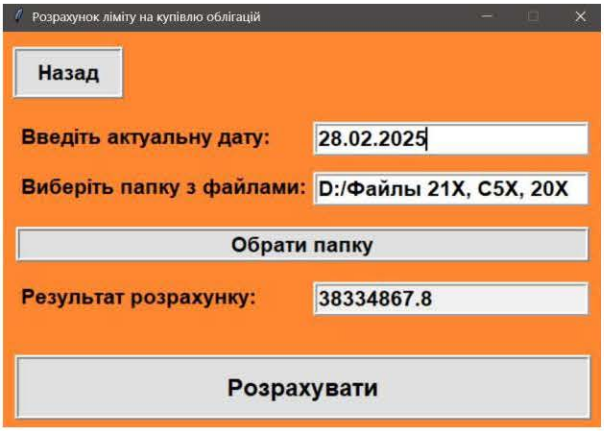

Рисунок 3.10 — Вікна для розрахунку ліміту на купівлю ОВДП

Програма обробляє різні види помилок та допомагає користувачу зорієнтуватися. Наприклад, якщо користувач не ввів дату, ввів її не у коректному форматі, забув обрати папку або в папці не знайшлося потрібних файлів для розрахунку, то програма динамічно підкаже користувачу, що потрібно зробити для коректного розрахунку. Також в окремих місцях використовується конструкція try/except для виявлення помилок при обробці стрічок в Excel-файлів. Деяки помилки будуть записуватися до логів, що корисно для розробника, тому що можна легко побачити які саме винятки перехоплюються. Приклад обробки помилок за допомогою messagebox у файлі ui.py (Рисунок 3.11).

```
def calculate_bond_limit():
    start_period = st_date.get()
    finish_period = fin_date.get()
    folder = folder_path.get()

    if not folder:
        messagebox.showerror("Помилка", "Будь ласка, оберіть папку.")
        return

    if not start_period or not finish_period:
        messagebox.showerror("Помилка", "Будь ласка, введіть дату.")
        return

    try:
        if file_finder_plus_deposit_limit_period(folder, start_period, finish_period) != None:
            formula = file_finder_plus_deposit_limit_period(folder, start_period, finish_period)
            res_textbox.config(state=tk.NORMAL)
            res_textbox.delete(1.0, tk.END)
            res_textbox.insert(tk.END, str(formula))
            res_textbox.config(state='disabled')

    except IndexError:
        messagebox.showerror("Помилка", "У вибраній папці недостатньо потрібних файлів для розрахунку за цей період.")
    except ValueError:
        messagebox.showerror("Помилка", "Введена некоректна дата")
    except Exception as e:
        messagebox.showerror("Помилка", "Шлях до папки з файлами некоректний")
```

Рисунок 3.11 — Обробка різних типів помилок за допомогою messagebox.showerror

Інтерфейс програми для розрахунку банківського нормативу Н6:

Інтерфейс програми для обчислення нормативу Н6 виконаний у вигляді простого та зручного вікна з використанням бібліотеки Tkinter. Головне вікно має фіксований розмір 500 на 260 пікселів, розташоване по центру екрану, з приємним світлим фоновим кольором.

У верхній частині розташовані текстові поля з підписами: перше — для введення дати, друге — для вибору папки з файлами. Поруч із полем для вибору папки є кнопка «Обрати папку», що відкриває діалог вибору каталогу. Нижче розміщено поле для відображення результату розрахунку — воно є доступним лише для читання, щоб користувач бачив обчислене значення, але не міг змінити його вручну.

У самому низу є велика кнопка «Розрахувати», яка запускає обчислення нормативу Н6 на основі введених даних.

Кнопки мають інтерактивний ефект підсвічування при наведенні, що робить інтерфейс більш приємним і зручним для користувача.

Варто зазначити, що програма наразі знаходиться на стадії розробки. Попередньо реалізовано головне вікно з базовою функціональністю обчислення нормативу Н6. У майбутньому можуть бути додані додаткові вікна або функціональні блоки для розширення можливостей програми.

На цей момент програма здатна чітко та коректно обчислити норматив Н6, надаючи користувачу зручний інтерфейс для введення даних і перегляду результатів (Рисунок 3.12).

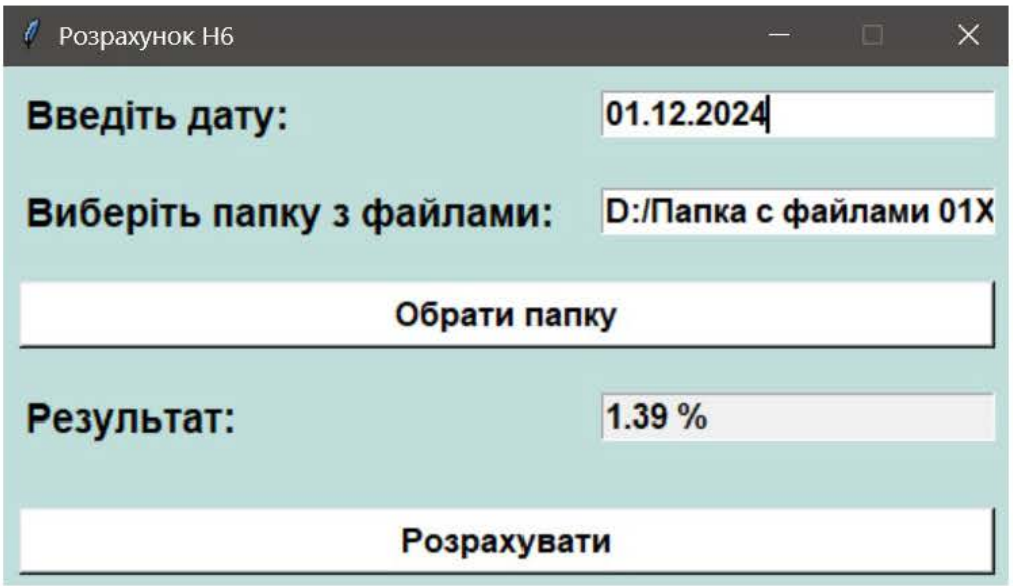

Рисунок 3.12 — Інтервейс головного вікна програми для обчислення банківського нормативу Н6

Інтерфейс програми також включає продуману обробку помилок, що реалізована аналогічно до попереднього програмного забезпечення. Якщо користувач, наприклад, не вибрав папку з файлами або не ввів дату, програма відразу динамічно відображає відповідне повідомлення про помилку через вікна повідомлень (messagebox). Це допомагає користувачу швидко зорієнтуватися, яка саме інформація відсутня або введена некоректно, і як це виправити.

Також у коді передбачено обробку можливих виключень — зокрема, помилок доступу до файлів, некоректного формату дати чи відсутності потрібних файлів у вибраній папці. Завдяки конструкції try-except розробник отримує інформативні повідомлення, що дозволяє уникнути збоїв у роботі програми.

Створення інсталятора для програмного забезпечення, що розраховує депозитні ліміти таліміт на купівлю ОВДП за допомогою ПЗ Inno Setup Compiler:

ІнноСетам Компайлер — це спеціалізоване програмне забезпечення, розроблене для автоматизації процесу компіляції та обробки даних в інформаційних системах. Воно забезпечує швидкий і надійний переклад вихідного коду, налаштування параметрів компіляції та інтеграцію з іншими модулями для подальшої обробки [11].

Головні особливості ІнноСетам Компайлера включають:

- Автоматизація процесів — дозволяє запускати компіляцію безпосередньо з інтерфейсу користувача або в автоматичному режимі за розкладом;
- Підтримка різних мов програмування — забезпечує гнучкість у роботі з різноманітними кодовими базами;
- Інтелектуальна обробка помилок — аналізує помилки під час компіляції і надає користувачеві детальні рекомендації для їх виправлення;
- Інтеграція з іншими інструментами — легко поєднується з системами контролю версій, засобами тестування та відлагодження.

ІнноСетам Компайлер знаходиться на стадії активного розвитку, і в майбутньому планується додавання нових функцій, таких як підтримка

додаткових платформ, розширені можливості для налаштування процесів компіляції та покращення користувацького інтерфейсу.

Загалом, цей компайлер сприяє підвищенню продуктивності розробників і якості кінцевих програмних продуктів за рахунок оптимізації та спрощення складних процесів компіляції, тому і був обраний, як інструмент для створення зручних інсталяторів.

Інтерфейс ІнноСетапа — це зручне та інтуїтивно зрозуміле середовище для налаштування і створення інсталяційних пакетів програмного забезпечення. Він розроблений таким чином, щоб користувач міг швидко і без зайвих зусиль створювати інсталяційні файли з необхідними параметрами (Рисунок 3.13).

Основні елементи інтерфейсу ІнноСетапа:

- Головне вікно — центральна область, де відображаються основні налаштування інсталяції, такі як назва програми, версія, каталог встановлення, інформація про авторські права;
- Меню налаштувань — розташоване збоку або зверху, дозволяє переключатися між різними розділами: файли для встановлення, ярлики, умови встановлення, параметри інтерфейсу;
- Редактор скриптів — вкладка або окреме вікно, де користувач може редагувати або створювати інсталяційні скрипти для більш тонкої настройки процесу встановлення;
- Редактор скриптів — вкладка або окреме вікно, де користувач може редагувати або створювати інсталяційні скрипти для більш тонкої настройки процесу встановлення.
- Кнопки керування — «Створити інсталятор», «Перевірити проект», «Зберегти» та інші, що дозволяють контролювати процес підготовки пакету.

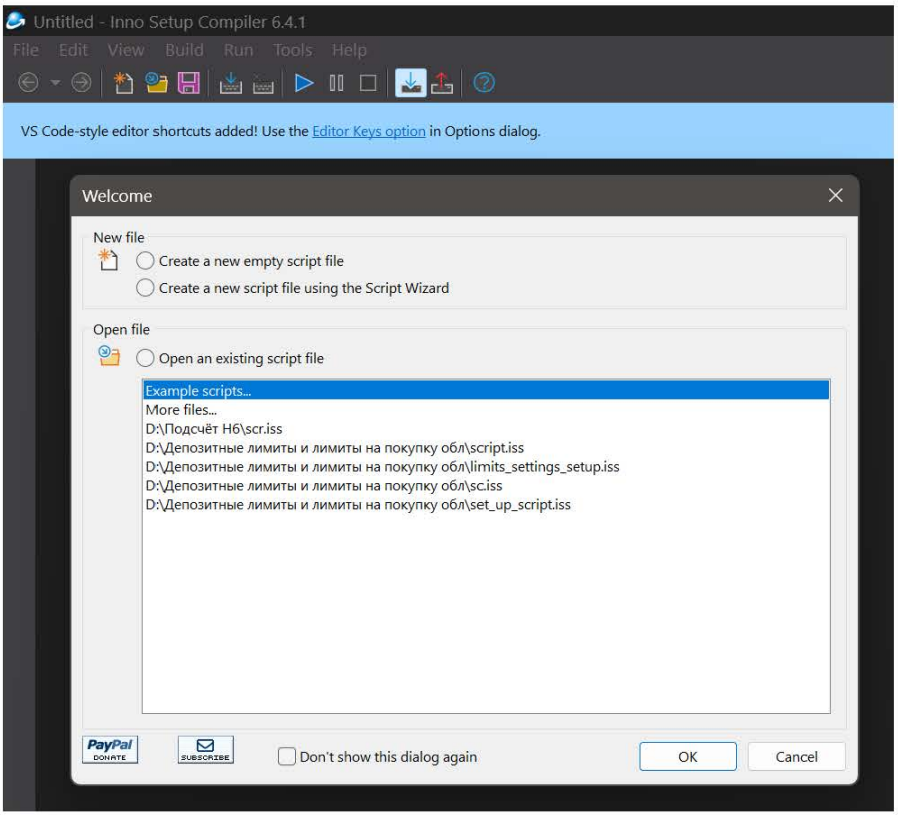

Рисунок 3.13 — Інтерфейс ПЗ InnoSetup Compiler

Для перетворення Python-скриптів у самостійні виконувані файли (.exe на Windows) використовується бібліотека PyInstaller, яка дозволяє запускати скрипт без встановленого Python [13]. Це дуже зручно, коли потрібно поширювати програму користувачам, які не мають Python або не хочуть його встановлювати. PyInstaller збирає всі необхідні файли та залежності в один або декілька файлів, створюючи готову до запуску програму.

Для того, щоб встановити цю бібліотеку потрібно спочатку активувати середовище розробки Python та ввести команду (pip install pyinstaller). Коли бібліотека буде встановлена потрібно виконати команду, яка створить .exe файл програми. В цьому випадку було введено таку команду: pyinstaller --onefile main.py --windowed --name=Limits --icon=L_icon.ico (Рисунок 3.14).

Пояснення, щодо параметрів:

- --onefile — об’єднати всі файли в один виконуваний;
- --windowed — вимкнути консольне вікно при запуску (корисно для графічних програм);

- --name=Limits — задати ім’я готового .exe файлу;
- --icon=icon.ico — додати іконку до програми;
- main.py — назва керуючого Python-скрипта.

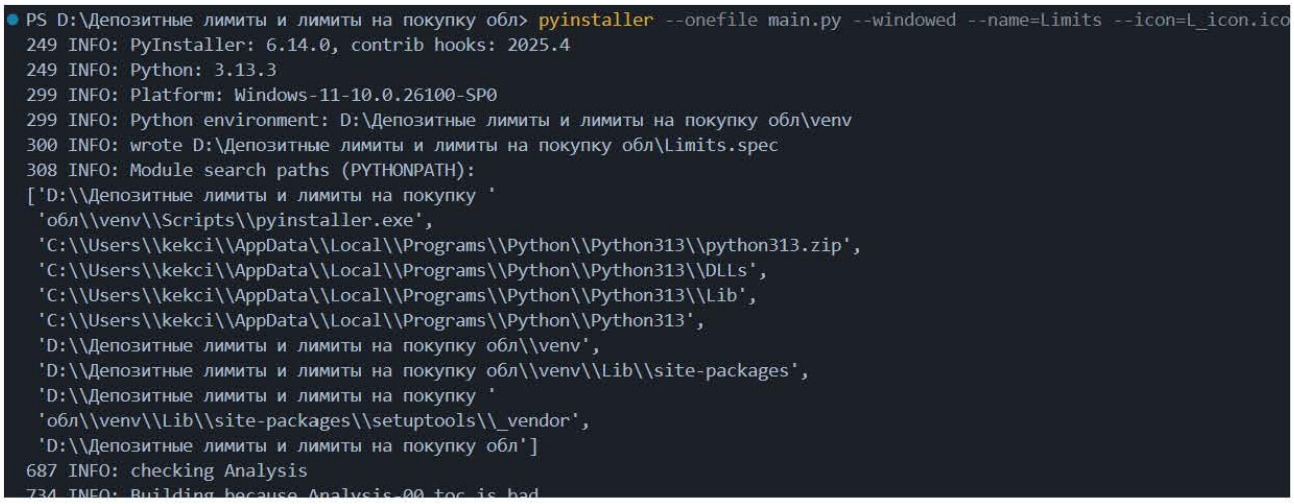

Рисунок 3.14 — Створення .exe файлу

Потрібно дочекатися завершення процесу, в процесі зборки в терміналі можна побачити багато повідомлень про збірку. Після завершення у папці з проєктом з’явиться нова папка dist. В цій папці буде знаходитися файл Limits.exe — це і буде готова програма. Для того щоб переконатися, що все спрацювало коректно потрібно запустити файл, який отримали після збірки та впевнитися, що немає технічних помилок.

Тепер переходимо до офіційного сайту ПЗ InnoInstaller Compiler та встановлюємо останню версію програми. Коли програма встановилася потрібно її запустити та обрати «Create a new script file». Відкриється вікно, де потрібно буде ввести назву програми, версію, посилання на сайт або гіт-репозиторій та розробника (Рисунок 3.15) [12].

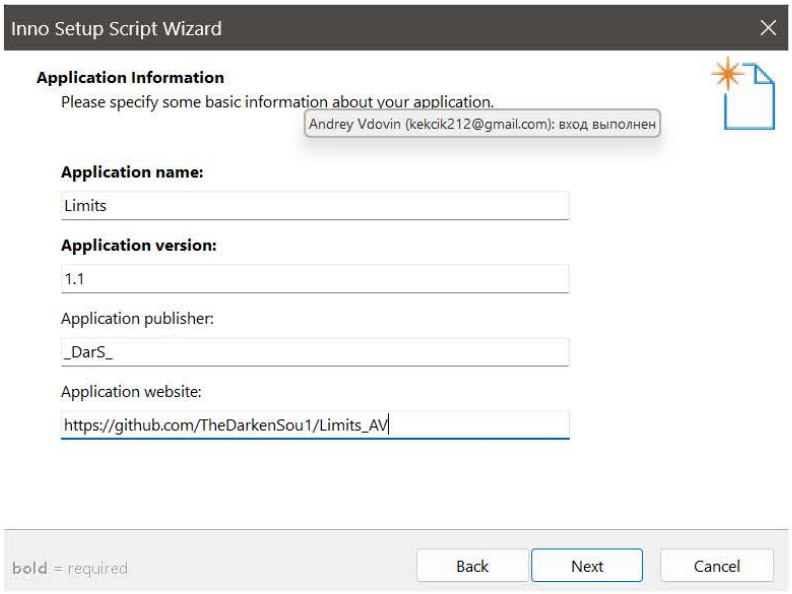

Рисунок 3.15 — Назва програми, версія, розробник і посилання на гіт-репозиторій

Далі потрібно вказати головний виконуваний файл (.exe) програми, який буде запускатися після встановлення та за потреби встановити шляхи до додаткових файлів або папок (Рисунок 3.16).

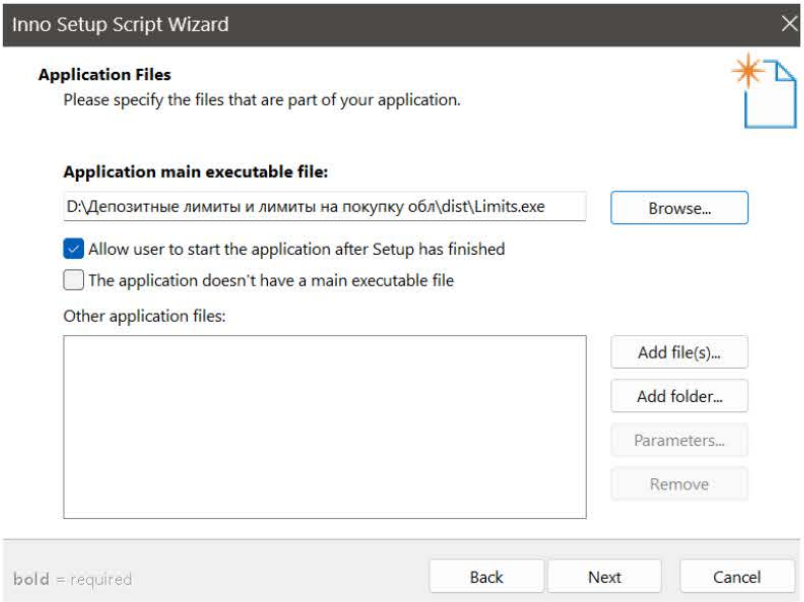

Рисунок 3.16 — Налаштування виконуваного файла

Далі потрібно перейти до налаштування шляху встановлення програми. У полі, яке з’являється на цьому етапі, необхідно вказати директорію, в яку буде

встановлено програму. За замовчуванням Inno Setup пропонує шлях {pf}\Limits, що означає, що програма буде встановлена в стандартну папку Program Files на комп'ютері користувача, у підкаталог з назвою Limits. Якщо користувач бажає, він може дозволити майбутнім користувачам змінювати цей шлях під час встановлення, або ж заборонити це, залишивши встановлення лише у вказану директорію. Після вибору параметра переходять далі, натискаючи кнопку Next.

На наступному етапі майстер пропонує налаштувати ярлики. З'являється поле з назвою Start Menu folder name, куди потрібно ввести назву папки в меню Пуск, яка буде створена для програми — у нашому випадку це Limits. Також є можливість дозволити або заборонити створення ярлика на робочому столі, поставивши або знявши відповідну галочку з пункту "Allow user to create a desktop shortcut". Якщо checkbox активний, користувач, який інсталюватиме програму, зможе обрати — створити ярлик на робочому столі чи ні. Після цього переходять далі, натискаючи Next.

Далі відкривається вікно, де можна вказати іконку програми. Якщо користувач уже має готовий .ico файл, його слід вибрати через кнопку огляду навпроти поля Application icon file. Обрана іконка буде використана для всіх ярликів, які створюватимуться інсталятором — і в меню Пуск, і на робочому столі. Це надає програмі індивідуального вигляду та полегшує її візуальне сприйняття. Після встановлення іконки переходять до наступного вікна.

У наступному вікні майстра користувач обирає мови, які підтримуватиме інсталятор. За замовчуванням увімкнено англійську мову, але за бажанням можна додати українську або інші мови зі списку, поставивши відповідні галочки. Цей вибір визначить, якою мовою буде інтерфейс вікна інсталяції для кінцевого користувача.

Завершивши налаштування, користувач переходить до останнього вікна майстра. Тут з'являється кнопка Finish, яка завершує роботу Script Wizard. Після натискання відкривається згенерований автоматично інсталяційний скрипт у вигляді .iss-файлу. Це повноцінний інструкційний файл для Inno Setup, який можна при бажанні відредагувати вручну (Рисунок 3.17).

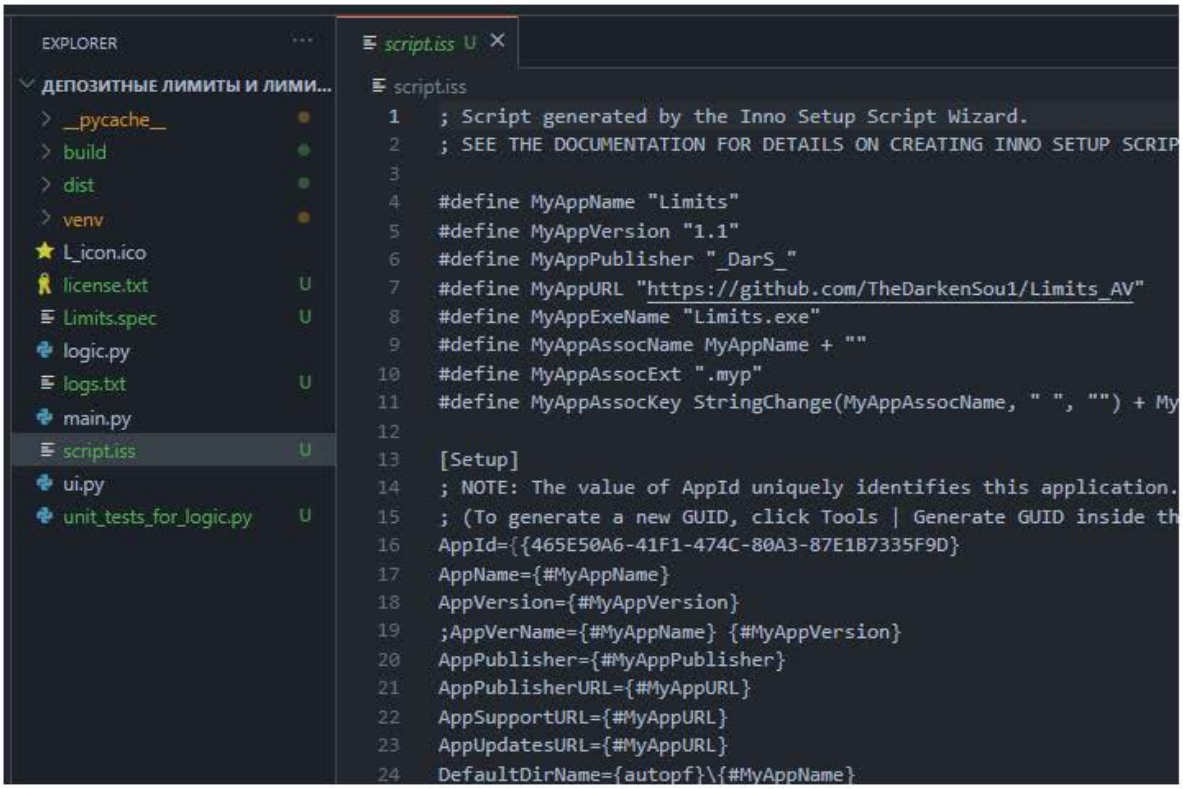

Рисунок 3.17 — Створений файл script.iss

Після відкриття скрипту користувач переходить до етапу побудови інсталятора. Для цього достатньо натиснути кнопку Compile у верхній панелі програми або просто натиснути клавішу F9. В процесі компіляції Inno Setup збере всі вказані файли, структуру встановлення, і створить один виконуваний файл-інсталятор у папці Output. Цей файл зазвичай має назву типу SetupLimits.exe. Користувач може передати цей файл іншим людям для встановлення своєї програми. Після запуску створеного інсталятора з'явиться знайомий інтерфейс установника, програма встановиться у вказану директорію, створить ярлики з заданою іконкою, а при запуску самої програми термінал не відкриватиметься, оскільки було використано параметр --windowed під час створення .exe за допомогою PyInstaller. Результат створеного інсталятора, іконки програми та відкритої програми показано на рис. 3.18.

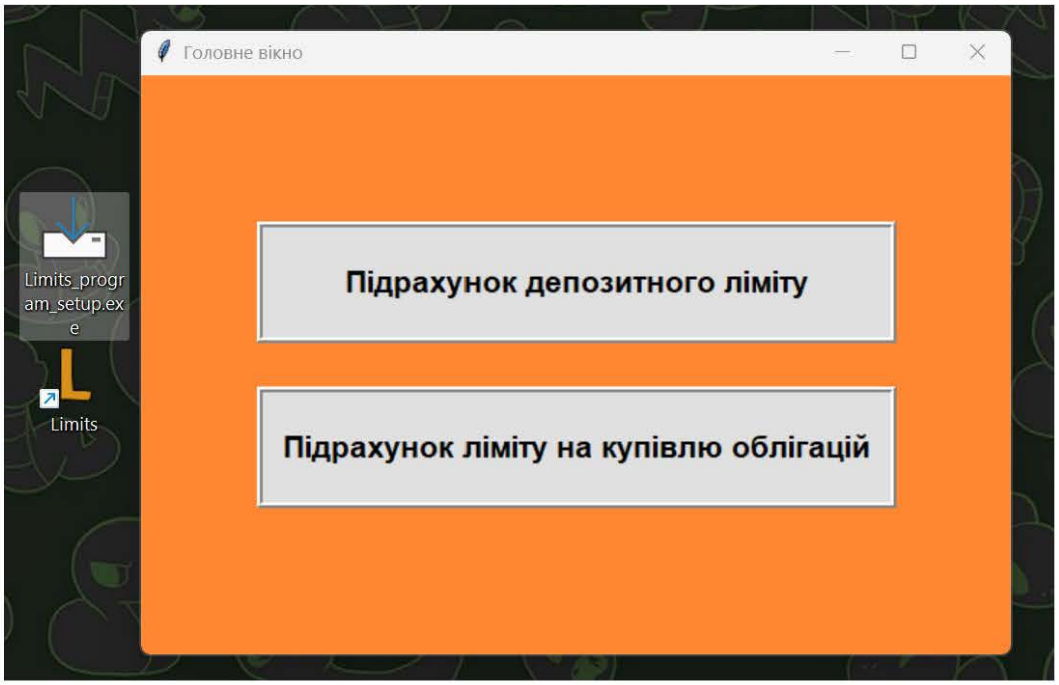

Рисунок 3.18 — Інсталятор, ярлик та програмне забезпечення

3.4. Висновки до третього розділу

У третьому розділі було повністю реалізовано завершальний етап створення програмного забезпечення для автоматизованого розрахунку банківських лімітів та нормативу Н6. Програми пройшла усі необхідні етапи — від реалізації логіки обробки Excel-файлів до створення зручного графічного інтерфейсу з елементами управління. Особливу увагу було приділено обробці помилок — система реалізована так, щоб динамічно реагувати на можливі помилки користувача та надавати підказки через повідомлення, що дозволяє уникати типових проблем і спрощує користування додатком.

Інтерфейси обох програми був оформлений у стилі, зручному для користувача: він дозволяє ввести дату, вибрати папку з файлами та швидко отримати результат обчислень.

Після цього було здійснено важливий етап — створення .exe-файлу за допомогою PyInstaller. Це дозволило перевести програму з розробницького формату у зручний вигляд для користування користувачам без потреби у встановленому Python. Завдяки параметру --windowed програма запускається без

терміналу, що робить її вигляд професійним і зручним для кінцевого користувача.

Останнім кроком стало створення повноцінного інсталятора за допомогою Inno Setup Compiler. Шлях встановлення, створення ярликів, мову інсталятора та візуальну іконку. Після компіляції було згенеровано виконуваний файл-установник. Таким чином, завершено формування готового ПЗ, яке може встановлюватися і використовуватися сторонніми користувачами без складнощів, що підтверджує завершення всіх ключових етапів життєвого циклу програмного продукту.

Як результат, програмне забезпечення, що розраховує депозитній ліміти за актуальну дату та за період, а також ліміт на купівлю ОВДП — повністю готовий та відтестований продукт, який може використовуватися відділом фінансової та статистичної звітності банку. Програмне забезпечення, що розраховує банківський норматив Н6 наразі знаходиться на стадії доопрацювання та тестування, але вже може коректно розрахувати формулу нормативу та виводити результат, а також має зрозумілий інтерфейс.

ВИСНОВКИ

У результаті виконаної роботи було повністю реалізовано програмне забезпечення для автоматизованого розрахунку ключових банківських показників, таких як депозитні ліміти на конкретну дату, ліміти на придбання облігацій внутрішньої державної позики (ОВДП), депозитні ліміти за певний період, а також розрахунок нормативу Н6. Розробка двох ПЗ здійснювалась поетапно з дотриманням принципів інкрементної моделі, що дозволило по черзі реалізовувати і тестувати окремі функціональні компоненти, забезпечуючи стабільність і контроль якості на кожному етапі. На початкових етапах були визначені функціональні вимоги, побудовано загальну архітектуру системи та розроблено алгоритми обробки банківських показників. Подальші ітерації включали реалізацію логіки пошуку та аналізу Excel-файлів, інтеграцію словників для активів і пасивів, а також створення інтерфейсів, які об'єднали ці компоненти в єдине рішення.

Основна мета — автоматизація обчислень, що раніше виконувались вручну, була досягнута. Реалізоване рішення дозволяє значно зменшити кількість помилок, скоротити час обробки Excel-файлів, що містять фінансову звітність, та спростити взаємодію співробітників банківського підрозділу з великою кількістю структурованих і напівструктурованих даних. Програма здатна працювати з різними шаблонами вхідних файлів, виявляючи потрібну інформацію навіть за умов відхилення від стандартної структури. Завдяки використанню бібліотек pandas, openpyxl, регулярних виразів та розроблених логічних алгоритмів вдалося досягти високої точності розрахунків, навіть для складних вкладених формул, що охоплюють десятки показників з активів і пасивів банку. Також було враховано часові формати, особливості збереження чисел у Excel, та виключення ситуацій з пропущеними даними.

Графічний інтерфейс користувача був реалізований за допомогою бібліотеки tkinter. Він містить окремі модулі для кожного з видів розрахунків, включаючи вибір дати, завантаження папки з файлами, запуск розрахунків та

відображення результатів. Інтерфейс побудований таким чином, щоб бути інтуїтивно зрозумілим навіть для користувача без глибоких технічних знань, завдяки зручній навігації, візуальним підказкам та чітким повідомленням про помилки. Зокрема, кольорове маркування полів, спливаючі повідомлення, підказки та елементи керування (кнопки, випадаючі списки) сприяють зменшенню кількості помилок з боку користувача. Програма також запам'ятовує останні введені параметри, що пришвидшує повторне використання.

Особливу увагу було приділено взаємозв'язкам між інтерфейсними модулями та логічними функціями. Усі дані, введені користувачем, передаються до окремих функцій обробки, які повертають результат обчислень або повідомлення про виявлені помилки. Такий підхід значно підвищив гнучкість та підтримуваність проєкту: логіку обробки можна змінювати окремо від інтерфейсу, що спрощує подальшу розробку. Ця модульність дозволяє легко оновлювати або замінювати окремі компоненти програми без ризику порушення роботи інших частин. Також це полегшує тестування — логічні блоки можна перевіряти незалежно від GUI, що важливо для забезпечення стабільної роботи програми та пришвидшує процес виявлення помилок.

Крім того, був реалізований інсталяційний пакет за допомогою системи Inno Setup, що дозволяє зручно розгортати програмне забезпечення на будь-якому комп'ютері без необхідності ручної установки бібліотек та залежностей. Завдяки цьому ПЗ, що розраховує ліміти, набуває ознак повноцінного прикладного продукту, готового до використання у виробничому середовищі банку. Інсталятор забезпечує швидке встановлення програми, створення ярликів, а також підтримку деінсталяції, що робить роботу з ним зручною та професійною. Крім того, інсталятор дозволяє налаштовувати параметри запуску, шлях до робочої директорії та інші корисні опції, що важливо для корпоративного середовища.

Під час розробки було опрацьовано низку викликів, які вимагали уважного підходу до структури Excel-файлів, формату збережених дат, особливостей обліку банківських показників та потреби в оптимізації часу обробки. У процесі

вдалося розробити універсальні шаблони обробки даних, що дозволяють швидко адаптувати програму до змін у структурі вхідних документів. Також особливу увагу було приділено обробці винятків та помилок — користувач завжди отримує чітке повідомлення про те, що сталося, без «падіння» програми. Це забезпечує впевненість у працездатності програми навіть у складних або нестандартних сценаріях, а також підвищує довіру до використання ПЗ у щоденній роботі банку.

Окремо варто відзначити ефективне використання словників (dict_liabilities, dict_assets, moneybox_L тощо), які значно спрощують логіку доступу до потрібних показників. Це дало змогу уникнути дублювання коду та зробити його більш читабельним і гнучким до змін. Додавання нових фінансових показників або редагування існуючих потребує лише оновлення відповідного словника без зміни основної логіки. Подібна структура зменшує ризики виникнення помилок при модифікації програми та дозволяє ефективно масштабувати її для обробки додаткових показників або нових типів фінансових документів.

Таким чином, реалізовані ПЗ стали результатом комплексної роботи, що поєднала аналіз фінансових вимог, розробку алгоритмів, створення інтерфейсу користувача, тестування, розробку установника та забезпечення зручності користування. ПЗ є прикладом сучасного інструменту для автоматизації бізнес-процесів, орієнтованого на точність, зручність і ефективність. У перспективі можливе його розширення: додавання нових функцій, підтримка інтеграції з базами даних або хмарними сервісами, масштабування на інші типи звітності, а також впровадження системи аналітики та візуалізації результатів для подальшого стратегічного планування. Реалізовані рішення також можуть бути адаптовані для інших фінансових установ, що відкриває шлях до створення універсального продукту. Виконана робота підтвердила важливість підходу системної розробки, що базується на поетапному впровадженні, модульності, орієнтації на користувача та гнучкості при внесенні змін. Реалізоване ПЗ успішно виконує поставлені завдання та є готовим для використання.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Pandas: управління даними проєкту | by Olegh Bondarenko: https://medium.com/stinopys/pandas-управління-даними-проєкту-e41290b1a18f.

2. OpenPyXL – Read the Docs: https://openpyxl.readthedocs.io/.

3. Комплексна повнофункціональна система автоматизації фінансової діяльності банку SrBank — https://soft-review.com.ua/ourcases/sab-srbank/.

4. Технологія роботи в середовищі АРМ: https://pidru4niki.com/14860110/bankivska_sprava/tehnologiya_roboti_seredovischi_arm.

5. Tkinter — Python interface to Tcl/Tk: https://docs.python.org/uk/3.13/library/tkinter.html.

6. Logging facility for Python: https://docs.python.org/uk/3.13/library/logging.html.

7. Перелік файлів з показниками статистичної звітності. Документація НБУ файл 21X: https://bank.gov.ua/files/Taxonomy/Description_21X.docx.

8. Перелік файлів з показниками статистичної звітності. Документація НБУ файл C5X: https://bank.gov.ua/files/Taxonomy/Description_C5X.docx.

9. Перелік файлів з показниками статистичної звітності. Документація НБУ файл 20X https://bank.gov.ua/files/Taxonomy/Description_20X.docx.

10. Перелік файлів з показниками статистичної звітності. Документація НБУ файл A7X: https://bank.gov.ua/files/Taxonomy/Description_A7X.docx.

11. Inno Setup Documentation: https://jrsoftware.org/ishelp.php.

12. Як розвернути репозиторій Git | База знань Hostinger: https://support.hostinger.com/uk/articles/1583302-як-розгорнути-репозиторій-git

13. PyInstaller Manual — Pyinstaller 6.14.0 documentation: https://www.pyinstaller.org/.

14. Інкрементна модель життєвого циклу розробки ПЗ: https://surl.li/seaqxu.

15. НБУ. Параметри для розрахунку ліміту на придбання депозитних сертифікатів: https://bank.gov.ua/ua/legislation/Resolution_08042024_40.

16. НБУ: Положення про обов'язвокі економічні нормативи: https://bank.gov.ua/admin_uploads/article/proekt_2021-12-24.pdf.

17. ETL-процеси: написання вимог: https://www.artofba.com/uk/post/etl-process-for-business-analyst.

18. Д. Кретов, Автоматизація процесів кредитного скорингу та рішень про видачу кредитів, що скорочує час на розгляд заявок та знижують ризики. 2024 — 2 с.

19. Я.І. Чайковський, Банківські технології і продукти. 2021 — 105 с.

20. НБУ, про внесення зміни до постанови Правління Національного банку України від 24 лютого 2022 року №22, щодо придбання банком лімітованих депозитних сертифікатів: https://bank.gov.ua/ua/legislation/Resolution_08042024_40.

ДОДАТОК А

1. Посилання на git-репозиторій для перегляду коду програмного забезпечення, що розраховує депозитні ліміти на актуальну дату, за період та ліміту на купівлю ОВДП: https://github.com/TheDarkenSou1/Limits_AV.

2. Посилання на git-репозиторій для перегляду коду програмного забезпечення, що розраховує банківський норматив Н6: https://github.com/TheDarkenSou1/H6_prog.