UNICHECK
by Turnitin

Ім'я користувача:
приховано налаштуваннями конфіденційності

Дата перевірки:
16.06.2023 10:32:52 EEST

Дата звіту:
16.06.2023 10:38:03 EEST

ID перевірки:
1015622765

Тип перевірки:
Doc vs Library

ID користувача:
100011372

Назва документа: Гімза КН-320

Кількість сторінок: 72 **Кількість слів:** 10418 **Кількість символів:** 82047 **Розмір файлу:** 1.56 MB **ID файлу:** 1015269882

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

9.84% Схожість

Найбільша схожість: 4.85% з джерелом з Бібліотеки (ID файлу: 1004042525)

Пошук збігів з Інтернетом не проводився

9.84% Джерела з Бібліотеки 719 .... **Сторінка** 74

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0% Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 13 сторінок

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

Фаховий молодший бакалавр
(освітньо-професійний ступінь)

на тему Автоматизація пакування Python бібліотек за допомогою GitHub Actions

Виконав студент III курсу, групи КН-*320*
ОПП *«Комп'ютерні науки»*
Спеціальності *122 Комп'ютерні науки*
Гімза Дмитро Петрович
(прізвище, ім'я по батькові)

Керівник (підпис) Богдан *Бугиль* (ім'я прізвище)

Нормоконтролер (підпис) Любомира *Кужій* (ім'я прізвище)

Рецензент (підпис) (ім'я прізвище)

Голова ЕК (підпис) Михайло *Свистун* (ім'я прізвище)

Члени ЕК (підпис) Володимир *Шумлянський* (ім'я прізвище)

(підпис) Любомира *Кужій* (ім'я прізвище)

Дипломна робота захищена в ЕК «___» _________ 20__ р.
з оцінкою «________»

Львів 2023

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Циклова комісія ***Комп'ютерні науки***
Освітньо-професійний ступінь ***Фаховий молодший*** бакалавр
Освітньо-професійна програма ***Комп'ютерні науки***
Спеціальність ***122 Комп'ютерні науки***

ЗАТВЕРДЖУЮ
Завідувач відділення
«Комп'ютерних наук та
інформаційної діяльності»
________ Володимир Шумлянський
« ____ » ____________ 2023 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Гімзі Дмитру Петровичу
(прізвище, ім'я та по батькові)

1. Тема роботи Автоматизація пакування *Python* бібліотек *за допомогою GitHub Actions*

керівник роботи к.т.н Бугиль Богдан Анатолійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом директора від “*22*” *лютого* *2023* року № *16 - ст*

2. Строк подання студентом роботи “ ____ ” __________ 2023 року

3. Вихідні дані до роботи

3.1 Написання тестового ПЗ

3.2 Створення Python бібліотеки її конфігурація

3.3 Публікація бібліотек на PYPi.org, автоматизація процесу пакування, тегування та створення GitHub Release

4. Зміст розрахунково-пояснювальної записки

4.1 Збір та опис теоретичних відомостей по створенню python бібліотек

4.2 Налаштування робочого середовища для створення python бібліотек та автоматизації даного процесу

4.3 Створення власної бібліотеки та автоматизація процесу її пакування

4.4 Охорона праці та техніка безпеки оператора з обробки інформації та програмного забезпечення

4.5 Техніко-економічне обгрунтування

5. Перелік графічного матеріалу

5.1.	*Загальна блок-схема пакування бібліотек на PyPI.org*
5.2.	*Блок-схема автоматизації пакування бібліотек за допомогою GitHub Actions*
5.3.	*Блок-схема тегування та створення GitHub Release для PyPI*
5.4.	*Дані економічного обґрунтування проєктного рішення*

6 Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		Завдання видав	Завдання отримав
Техніко-економічне обґрунтовування	*Мар'яна Смук*		
Охорона праці та безпека життєдіяльності	Олена Мельникова		

7. Дата видачі завдання « 01» березня *2023* року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання	Примітка
1	*Опрацювання наявної інформації про застосування пошукової системи*	10.03.2023	
2	*Створення Python бібліотеки*	25.03.2023	
3	*Пакування бібліотеки*	09.04.2023	
4	*Завантаження бібліотеки на PyPI.org*	01.05.2023	
5	*Завантаження бібліотеки на GitHub*	10.05.2023	
6	*Автоматизація пакування бібліотеки за допомогою GitHub Actions*	20.05.2023	
7	*Реліз бібліотеки за допомогою GitHub Releases*	10.03.2023	

Студент ______ Дмитро Гімза
(підпис) (ім'я, прізвище)

Керівник роботи ______ Богдан Бугиль
(підпис) (ім'я, прізвище)

РЕФЕРАТ

Текстова частина дипломної роботи: 65 ст., 27 рис., 3 табл., 18 джерел, 2 додатки.

Об'єкт проектування – пакування, автоматизація пакування та реліз Python бібліотеки, використовуючи GitHub Actions та GitHub Releases.

Мета виконання дипломної роботи полягає в створенні тестового ПЗ, а саме Python бібліотеки, її пакування в ручну, автоматизації пакування цієї ж бібліотеки за допомогою GitHub Actions та її реліз за допомогою GitHub Releases.

Галузь використання – Python проекти.

Проведено аналітичний аналіз інтернет ресурсів. Проаналізовано та обгрунтовано проблематику автоматизації Python бібліотеки. Реалізовано Python бібліотеку, яка робить конкатенацію стрічок щоб зробити емейл адресу.

БІБЛІОТЕКА, АВТОМАТИЗАЦІЯ, РЕЛІЗ, PYPI, PYTHON, GITHUB, GIHUB RELEASE, GITHUB ACTIONS

«___» __________ 202__ року. ______________

Підпис Власне ім'я ПРІЗВИЩЕ

ЗМІСТ

ВСТУП

Автоматизація пакування Python бібліотек за допомогою GitHub Actions - це процес автоматичного створення пакетів для Python-бібліотек із використанням GitHub Actions. GitHub Actions - це сервіс, що дозволяє автоматизувати робочі процеси на базі подій, що відбуваються у репозиторії на GitHub. Створення Python бібліотек - це коли ви створюєте набір функцій або класів для використання в інших програмах на Python. Автоматизація пакування бібліотеки - це процес автоматичного створення файлу з вашою бібліотекою, щоб люди могли легко використовувати вашу бібліотеку в своїх програмах. Для цього можна використовувати інструменти, які дозволяють автоматично створювати пакет з вашої бібліотеки. Такі інструменти можуть бути дуже корисними, оскільки вони дозволяють легко і швидко розповсюджувати вашу бібліотеку серед інших розробників, що забезпечує більш швидку та ефективну розробку програм на Python. Для того, щоб вашу Python бібліотеку можна було легко використовувати в інших програмах, потрібно забезпечити доступ до неї у вигляді пакету. Пакет - це спеціальний формат архіву, який містить всі файли вашої бібліотеки, а також необхідну інформацію про версію, залежності та інші метадані.

Один з найпоширеніших інструментів для автоматизації пакування Python бібліотек - це setuptools. Це набір інструментів, який дозволяє автоматично створювати Python-пакети з вашої бібліотеки за допомогою спеціального файла setup.py. У цьому файлі ви можете вказати всю необхідну інформацію про вашу бібліотеку, таку як ім'я, версію, залежності, авторів, опис та інші параметри.

Мета роботи - навчитись створювати бібліотеки з використання мови Python та виконати автоматизацію процесу взаємодії з репозиторієм публічних бібліотек pypi.org.

Обєкт дослідження - бібліотеки на мові програмування Python.

Предмет дослідження - автоматизація процесу пакування Python бібліотек та їх завантаження у репозиторій pypi.org

1 ЗБІР ТА ОПИС ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ ПО СТВОРЕННЮ PYTHON БІБЛІОТЕК

1.1 Поняття бібліотек Python та їх пакування

Python-бібліотека - це збірка підпрограм, які написані на мові програмування Python і містять функції, методи та класи для виконання різноманітних завдань. Бібліотеки Python зазвичай містять функції, які можна використовувати для обробки даних, роботи з мережами, графікою, шифруванням, створенням ігор, обробки зображень, роботи з базами даних та багатьма іншими завданнями.

Python має велику кількість бібліотек, які можна встановити з використанням менеджера пакетів pip, включаючи стандартну бібліотеку Python, яка включає багато корисних функцій та модулів, таких як math, datetime, os, json та інші.

Додаткові бібліотеки можуть бути створені користувачами та розміщені у відкритих репозиторіях, наприклад на GitHub. Також, бібліотеки можуть бути розповсюджені як окремі файли або у вигляді Python-пакетів, що включають всі необхідні файли та метадані.

Python-бібліотеки дозволяють розширити функціональність Python і скоротити час написання коду, дозволяючи використовувати готові функції та модулі замість того, щоб писати код з нуля. Це дозволяє програмістам більше уваги приділяти розв'язанню більш складних завдань, замість того, щоб витрачати час на написання коду для рутинних задач.

Python - це мова програмування, яка надає велику кількість стандартних функцій та модулів, а також відкриту систему для створення та використання додаткових бібліотек.

Python-бібліотеки зазвичай складаються з файлів і модулів, які можуть бути імпортовані у Python-програму та використовуватися для різних завдань.

Наприклад, модуль "os" містить функції, які дозволяють взаємодіяти з операційною системою, модуль "json" дозволяє роботу з JSON-даними, а модуль "requests" дозволяє взаємодіяти з веб-сайтами та отримувати дані з них.

Створення бібліотеки зазвичай включає створення документації та тестів, що допомагає користувачам легше зрозуміти, як користуватися функціями бібліотеки та переконатися, що вони працюють правильно. Документація може бути створена з використанням спеціальних інструментів, таких як Sphinx, а тести можуть бути написані з використанням Pytest або інших тестових фреймворків.

Для розповсюдження бібліотек Python зазвичай використовуються пакетні менеджери, такі як pip, що дозволяють легко встановлювати та оновлювати бібліотеки на комп'ютерах користувачів. Також, розповсюдження може бути здійснене через відкриті репозиторії на платформах, таких як GitHub, або через спеціальні репозиторії, такі як PyPI (Python Package Index).

GitHub Actions дозволяє автоматизувати процес пакування бібліотек Python та розповсюдження їх на різних платформах. Це зменшує час та зусилля, необхідні для ручного створення та розповсюдження бібліотеки, а також забезпечує її стабільність та надійність завдяки автоматичному тестуванню.

Для автоматизації пакування бібліотек Python за допомогою GitHub Actions необхідно створити спеціальний файл, який містить інструкції для автоматичного пакування та розповсюдження бібліотеки. Цей файл може містити інформацію про те, які версії Python підтримує бібліотека, які залежності необхідні для її роботи, які тести потрібно запустити перед пакуванням, та які дії потрібно виконати для розповсюдження бібліотеки на різних платформах.

Після створення файлу з інструкціями необхідно додати його до репозиторію на GitHub, та налаштувати GitHub Actions для автоматичного виконання пакування та розповсюдження бібліотеки. GitHub Actions дозволяє налаштовувати різні варіанти тестування та розповсюдження бібліотеки на різних платформах, таких як Windows, Linux та MacOS.

Після налаштування GitHub Actions, кожен коміт до репозиторію буде автоматично тестуватися та пакуватися в бібліотеку Python, яка може бути легко розповсюджена на різних платформах.

Застосування GitHub Actions для автоматизації пакування та розповсюдження бібліотек Python зменшує час та зусилля, необхідні для створення та розповсюдження бібліотек, забезпечує їх стабільність та надійність, та дозволяє зосередитися на розробці функцій бібліотеки.

Пакування (англ. packaging) - це процес створення пакетів (англ. packages), які містять програмне забезпечення, бібліотеки або інші компоненти програмного забезпечення для розповсюдження та встановлення на інших системах.

У випадку Python, пакування виконується з метою створення бібліотек або пакетів (англ. packages), які можуть бути легко встановлені та використані на різних системах. Бібліотеки Python зазвичай містять декілька файлів, які можуть бути розповсюджені в різних форматах, наприклад, у вигляді джерела (англ. source), виконуваних файлів (англ. executable), бінарних файлів (англ. binary), або пакетів, які містять різноманітні файли та метадані.

Стандартна бібліотека Python містить модуль пакування (англ. packaging), який надає набір інструментів для створення та розповсюдження пакетів. Найпоширенішим форматом пакування Python-бібліотек є формат Distutils, який дозволяє створювати пакети в різних форматах, таких як sdist, bdist, та wheel.

sdist - це формат, який містить джерело коду бібліотеки, тобто файли з розширенням .py. Такий формат зазвичай використовується для розповсюдження бібліотеки через мережу Інтернет.

bdist - це формат, який містить виконуваний код бібліотеки. Цей формат зазвичай використовується для розповсюдження бібліотеки на різних платформах, таких як Windows, Linux та MacOS.

wheel - це формат, який містить бінарний код бібліотеки та інші файли, які необхідні для її роботи. Цей формат є більш ефективним для розповсюдження бібліотек через мережу Інтернет, оскільки він дозволяє швидко та легко

встановлювати бібліотеку на різних системах, а також забезпечує більш ефективне управління залежностями.

Для пакування бібліотек Python можна використовувати різні інструменти, такі як Distutils, Setuptools, і Flit. Вони дозволяють створювати пакети, залежності та метадані для бібліотеки.

Setuptools - це інструмент для створення та розповсюдження бібліотек Python, який дозволяє легко додавати залежності, метадані та інші файли до пакету. Він є розширенням Distutils та дозволяє автоматизувати багато процесів, пов'язаних з пакуванням та розповсюдженням бібліотек.

Flit - це інструмент для створення та розповсюдження бібліотек Python, який дозволяє створювати пакети у форматі wheel та залежності у форматі requirements.txt. Він є більш простим у використанні, ніж Setuptools, тому часто використовується для пакування простих бібліотек.

Пакування Python-бібліотек - це важливий етап в процесі розробки програмного забезпечення, оскільки воно дозволяє легко та швидко розповсюджувати та встановлювати бібліотеки на різних системах.

1.2 Структура проекту для бібліотеки Python

Структура проекту для бібліотеки Python повинна бути організованою та стандартизованою, щоб забезпечити зручність використання та розповсюдження бібліотеки. Основні елементи, які повинні бути у структурі проекту для того, щоб він міг бути бібліотекою, наступні:

- Назва проекту та його версія. Це допоможе ідентифікувати бібліотеку та відрізняти її від інших проектів.
- Каталог з кодом бібліотеки. У цьому каталозі повинен знаходитися весь код, який складає бібліотеку. Він може бути організований за модулями, які містять функції та класи, пов'язані між собою.
- Файл README.md. Це файл, в якому описується бібліотека та її функціональні можливості. Він допоможе користувачам швидко зрозуміти, чим саме бібліотека може бути корисною.
- Файл setup.py або pyproject.toml. Ці файли містять метадані проекту та інформацію, необхідну для пакування та розповсюдження бібліотеки. Наприклад, в них вказуються назва та версія бібліотеки, залежності та інші параметри.
- Файл LICENSE. Це файл, в якому вказується ліцензія, на якій розповсюджується бібліотека. Ліцензія визначає права користувачів на

використання бібліотеки та розповсюдження її змінених версій.

- Файл .gitignore. Цей файл вказує Git, які файли та каталоги повинні бути ігноровані при збереженні проекту у репозиторії.
- Каталог з тестами.

Крім вище згаданих елементів, структура проекту для бібліотеки Python може включати й інші складові, залежно від потреб та вимог проекту. Деякі з них:

- Каталог з документацією. У цьому каталозі може знаходитися документація проекту, наприклад, документація API або документація для користувачів.
- Каталог з прикладами використання. У цьому каталозі можуть бути приклади використання бібліотеки, щоб допомогти користувачам зрозуміти, як використовувати бібліотеку у своїх проектах.
- Файл CHANGELOG.md. Цей файл містить список змін, які внесені до бібліотеки на різних версіях. Це допоможе користувачам визначити, які зміни були внесені до бібліотеки та як це може вплинути на їх проекти.
- Каталог з ресурсами. У цьому каталозі можуть знаходитися ресурси, такі як шрифти, зображення, дані тощо, які використовуються у проекті.
- Файл MANIFEST.in. Цей файл містить список файлів, які потрібно включити до дистрибутиву бібліотеки, але які не входять до каталогу з кодом бібліотеки.

Структура проекту для бібліотеки Python може бути створена вручну або згенерована за допомогою автоматичних інструментів, таких як Cookiecutter.

Головне, щоб вона була чіткою та стандартизованою, щоб забезпечити зручність використання та розповсюдження бібліотеки. Декілька деталей щодо структури проекту для бібліотеки Python:

Каталог з тестами. У цьому каталозі можуть знаходитися модульні, інтеграційні та приймочні тести для бібліотеки. Тести допоможуть переконатися, що бібліотека працює правильно та не містить помилок.

Файл setup.py. Цей файл містить інформацію про проект та дозволяє збирати та розповсюджувати бібліотеку за допомогою pip. У файлі можна вказати назву бібліотеки, версію, автора, опис та залежності.

Файл requirements.txt. У цьому файлі перелічуються залежності, які необхідні для встановлення та використання бібліотеки. Файл допоможе іншим користувачам встановити необхідні залежності та використовувати бібліотеку без проблем.

Каталог з модулями бібліотеки. У цьому каталозі знаходяться файли з кодом, які створюють модулі бібліотеки. Кожен модуль може містити класи, функції, константи та інші елементи, необхідні для роботи бібліотеки.

Файл README.md. Цей файл містить опис проекту, його функціональність, призначення, встановлення та використання. README.md є одним з найважливіших файлів у структурі проекту, оскільки він допомагає користувачам зрозуміти, як використовувати бібліотеку та що вона робить.

Важливо зазначити, що структура проекту може відрізнятися в залежності від потреб та вимог проекту, але зазвичай стандартизована структура забезпечує зручність використання та розповсюдження бібліотеки. Для додаткової зручності можна скористатися іншими стандартами та бібліотеками, наприклад, pytest для написання тестів, Sphinx для генерації документації, black для форматування коду, flake8 для перевірки стилю коду та інші.

Також, щоб бібліотека була зручною у використанні, важливо розробити чітке та докладне API (інтерфейс програмування застосунків). API повинно бути зрозумілим та простим для користування, а також добре задокументованим.

За допомогою GitHub Actions можна автоматизувати процеси, пов'язані з пакуванням та тестуванням бібліотеки, щоб полегшити її розповсюдження та забезпечити якість коду.

1.3 Принципи системи pypi.org

PyPI (Python Package Index) - це централізоване сховище для розповсюдження бібліотек Python. Це офіційний репозиторій для публікації та встановлення сторонніх бібліотек у Python.

PyPI дозволяє розробникам публікувати свої бібліотеки, щоб інші користувачі Python могли легко встановлювати їх у свої проекти. Крім того, PyPI забезпечує механізм версіювання, що дозволяє розробникам випускати оновлення своїх бібліотек та вирішувати проблеми безпеки.

Для публікації бібліотек на PyPI необхідно виконати певні кроки, такі як створення аккаунту та налаштування пакету. Якщо бібліотека проходить тестування та відповідає вимогам PyPI, вона може бути опублікована та доступна для завантаження.

PyPI також підтримує деякі інструменти, що спрощують публікацію бібліотек, такі як Twine - інструмент для завантаження пакетів на PyPI, або pip - утиліту для встановлення та управління пакетами з PyPI. Кілька деталей про PyPI:

- PyPI дозволяє не тільки публікувати власні бібліотеки, але і завантажувати та встановлювати бібліотеки сторонніх розробників у свої проекти.
- PyPI дуже популярне серед розробників Python та зазвичай є першим місцем, куди звертаються для пошуку та завантаження бібліотек. Це забезпечує легку інтеграцію та розповсюдження бібліотек серед інших користувачів Python.
- PyPI підтримує використання різних форматів для пакування бібліотек, включаючи wheels та source distributions. Wheels - це скомпільовані пакети, що містять готовий до виконання код, тоді як source distributions - це вихідний код бібліотеки, який можна компілювати на машині користувача.
- Під час публікації бібліотеки на PyPI можна додавати метадані, такі як опис, ключові слова, автори, ліцензії та інші важливі деталі про бібліотеку, що допомагають іншим розробникам зрозуміти, що вона робить та як її використовувати.
- PyPI також забезпечує механізм контролю версій, який дозволяє розробникам випускати оновлення своїх бібліотек та вирішувати проблеми безпеки. Користувачі можуть встановлювати конкретні версії бібліотек або встановлювати їхні останні доступні версії.
- PyPI є відкритим проектом, що дозволяє розробникам вносити свої внески та допомагати покращувати платформу. Також існує декілька альтернативних репозиторіїв, таких як Anaconda Cloud та Test PyPI.

Для того, щоб опублікувати свою бібліотеку на PyPI, потрібно створити обліковий запис на сайті PyPI та встановити пакет twine. Далі потрібно зберегти свою бібліотеку як файл tar.gz або zip та за допомогою команди twine upload завантажити пакет на сервер PyPI.

Процес пакування та публікації можна автоматизувати за допомогою GitHub Actions, як я розповідала раніше. Кожен коміт в гіт-репозиторії може тригерити GitHub Action, який буде автоматично пакувати та публікувати нову версію бібліотеки на PyPI. Таким чином, розробник може вести розробку бібліотеки у звичайному репозиторії, а завантаження та релізи бібліотеки будуть автоматично керуватися GitHub Actions.

1.4 Автоматизація припакуванні бібліотек

Автоматизація припакуванні бібліотек - це процес створення скриптів та налаштувань, які дозволяють автоматично пакувати та публікувати нові версії бібліотеки. Це дозволяє розробникам більш ефективно та швидко вести розробку, оскільки вони не витрачають час на ручне пакування та публікацію кожної нової версії.

Для автоматизації процесу припакування та публікації можна використовувати різні інструменти та технології. Наприклад, можна використовувати спеціальні бібліотеки для пакування, такі як setuptools, або використовувати засоби автоматизації, такі як GitHub Actions.

При автоматизації припакування бібліотек можна налаштувати автоматичне створення тестових середовищ для перевірки бібліотеки, автоматичне визначення версії бібліотеки за допомогою Git тегів або визначення номеру версії залежно від вмісту змін в Git-комітах, автоматичне створення файлу setup.py, який необхідний для пакування бібліотеки, автоматичне створення файлу README.md для показу інформації про бібліотеку на сторінці PyPI та багато іншого.

Автоматизація припакуванні бібліотек дозволяє ефективно вести розробку, допомагає запобігти помилкам та збільшує швидкість випуску нових версій бібліотеки.

Автоматизація припакуванні бібліотек полягає у створенні автоматизованого процесу, який дозволяє з легкістю створювати нові версії бібліотеки та завантажувати їх на репозиторій PyPI без необхідності виконувати весь процес вручну.

Для цього можна використовувати різні інструменти автоматизації, такі як скрипти на мовах програмування, Makefile або CI/CD сервіси, наприклад, GitHub Actions, Travis CI або GitLab CI. Ці інструменти дозволяють визначити правила для автоматичного пакування бібліотеки, створення її документації та завантаження на PyPI.

Наприклад, з використанням GitHub Actions можна налаштувати процес автоматичного пакування бібліотеки та завантаження її на PyPI за допомогою спеціального скрипту, який буде виконуватися кожного разу при злитті нових змін у гілку master. Це дозволить автоматизувати весь процес та зменшити ризик виникнення помилок в ручному режимі.

Також, важливо забезпечити належну тестування бібліотеки перед її публікацією на PyPI. Для цього можна використовувати автоматизовані тести,

які допоможуть переконатися в тому, що бібліотека працює належним чином і не містить помилок.

1.5 Поняття та призначення GitHub Actions

GitHub Actions - це послуга автоматизації розробки, яка дозволяє автоматично виконувати різні дії при зміні коду в репозиторії на GitHub. Вона дозволяє налаштовувати та запускати різноманітні процеси автоматизації, такі як тестування, збирання та пакування програмного коду, створення звітів, нотифікації та багато іншого.

GitHub Actions побудована на базі віртуальних машин, які дозволяють виконувати різні дії над кодом. Для налаштування цих дій використовуються так звані "workflow" - набір команд, які виконуються послідовно. Workflow може бути активований різними подіями, наприклад, зміною коду в гілці, відкриттям пул-реквесту або тегуванням релізу.

GitHub Actions надає безкоштовний доступ до певної кількості хвилин на місяць для виконання workflow. Якщо потрібна додаткова кількість хвилин або потрібні більш потужні віртуальні машини, можна підключити сплатні плани.

GitHub Actions має ряд різноманітних застосувань в автоматизації розробки програмного забезпечення. Ось деякі з найпоширеніших призначень:

- Тестування: GitHub Actions дозволяє автоматично запускати тестові скріпти при кожній зміні коду в репозиторії. Це дозволяє виявляти проблеми на ранніх етапах розробки та забезпечувати стабільність розроблюваного ПЗ.
- Збирання та пакування коду: GitHub Actions дозволяє автоматично збирати та пакувати програмний код для різних платформ та операційних систем. Це дозволяє значно спростити процес розгортання програмного забезпечення.
- Аналіз коду: GitHub Actions дозволяє автоматично проводити аналіз коду на предмет виявлення потенційних проблем з безпекою, ефективністю та якістю коду.
- Створення документації: GitHub Actions дозволяє автоматично створювати документацію для розроблюваного ПЗ на основі коментарів у коді.
- Релізи: GitHub Actions дозволяє автоматично створювати релізи ПЗ, робити тегування версій та повідомляти про це команду розробників та користувачів.
- Інтеграція з зовнішніми сервісами: GitHub Actions дозволяє інтегруватися з різноманітними зовнішніми сервісами, такими як AWS, Google Cloud, Azure, Docker та багатьма іншими, що робить автоматизацію розробки ще більш ефективною та потужною.

GitHub Actions - це інструмент автоматизації, що дозволяє створювати та виконувати різноманітні задачі на основі певних подій в Git-репозиторії, таких як коміти, відгалуження (branches), пул-реквести (pull requests) та інші.

GitHub Actions заснований на концепції workflows, тобто послідовності задач, які потрібно виконати, та на різних подіях, що стаються в Git-репозиторії.

Workflows можна налаштовувати за допомогою YAML-файлів, що знаходяться в директорії .github/workflows у Git-репозиторії.

GitHub Actions дозволяє виконувати різні операції в автоматичному режимі, такі як збирання проекту, запуск тестів, пакування бібліотеки, розгортання додатку на серверах та багато іншого. Для цього можна використовувати стандартні дії (actions), які надає GitHub, або створювати власні.

GitHub Actions забезпечує зручний інтерфейс для налаштування workflows та моніторингу їх стану. Крім того, GitHub Actions інтегрований з іншими сервісами GitHub, такими як GitHub Package Registry, що дозволяє зберігати та розповсюджувати пакети бібліотек, а також з іншими зовнішніми сервісами, такими як AWS, Google Cloud та інші.

Загалом, GitHub Actions дозволяє зробити процес розробки та релізу програмного забезпечення більш ефективним та автоматизованим.

2 НАЛАШТУВАННЯ РОБОЧОГО СЕРЕДОВИЩА ДЛЯ СТВОРЕННЯ PYTHON БІБЛІОТЕК ТА АВТОМАТИЗАЦІЇ ДАНОГО ПРОЦЕСУ

2.1 Встановлення програм та створення акаунту для роботи

Щоб встановити Python на свій комп'ютер, потрібно виконати наступні кроки:

- Завантажте встановлювач Python з офіційного сайту (https://www.python.org/downloads/). Зазвичай на головній сторінці ви побачите посилання на останню стабільну версію Python.
- Запустіть завантажений встановлювач. На першому екрані обов'язково встановіть галочку "Add Python to PATH", щоб мати доступ до Python з командного рядка.
- Встановлюйте Python за замовчуванням з усіма компонентами.
- Завершивши встановлення, ви можете запустити Python з командного рядка, введіть "python" та натисніть "Enter". Відкриється інтерактивна оболонка Python.
- Щоб написати та виконати свій перший Python-код, відкрийте текстовий редактор (наприклад, IDLE, PyCharm або Visual Studio Code) та напишіть код. Збережіть файл з розширенням ".py" та виконайте його з командного рядка за допомогою команди "python file_name.py".

Це допоможе встановити Python на ваш комп'ютер та почати роботу з цією мовою програмування.

Щоб встановити IntelliJ IDEA на свій комп'ютер, потрібно виконати наступні кроки:

- Завантажте встановлювач IntelliJ IDEA з офіційного сайту
- (https://www.jetbrains.com/idea/download/).Зазвичай на головній сторінці ви побачите посилання на останню стабільну версію.

- Запустіть завантажений встановлювач та слідуйте інструкціям на екрані.
- Під час встановлення вам можуть бути запропоновані додаткові компоненти, які можуть бути корисні для роботи з IntelliJ IDEA. Ви можете вибрати тільки ті компоненти, які потрібні вам.
- Після завершення встановлення запустіть IntelliJ IDEA. Вам можуть бути запропоновані деякі налаштування, які можуть бути важливими для вашої роботи.
- Після запуску ви можете створити новий проект або відкрити існуючий проект для редагування. Для створення нового проекту слід вибрати тип проекту та налаштувати параметри проекту.

Це допоможе встановити IntelliJ IDEA на комп'ютер та почати роботу з цією інтегрованою середовищем розробки.

Щоб створити акаунт на GitHub, виконайте наступні кроки:

- Перейдіть на головну сторінку GitHub (https://github.com/).
- Натисніть на кнопку "Sign up" (Зареєструватися), розташовану зверху справа на головній сторінці.
- Введіть свій електронний адрес, пароль та ім'я користувача відповідно до інструкцій на екрані. Придумайте ім'я користувача, яке буде легко запам'ятати та ідентифікувати вас.
- Натисніть кнопку "Create account" (Створити обліковий запис).
- Після створення акаунту, ви будете перенаправлені на сторінку "Welcome to GitHub" (Ласкаво просимо на GitHub), де вам буде запропоновано вибрати план тарифів та вказати свої зацікавлення.
- Після того, як ви виберете план та вкажете свої зацікавлення, вам буде запропоновано початкову сторінку, де ви зможете створювати репозиторії та взаємодіяти з іншими користувачами на GitHub.

Це допоможе створити акаунт на GitHub та почати розробку проектів в цьому сервісі

2.2 Локальна інтеграція з GitHub та реєстрація на pypi

Інтеграція локального репозиторію з GitHub допомагає зберігати та керувати версіями коду. Ось декілька кроків для локальної інтеграції з GitHub:

- Створіть репозиторій на GitHub.
- Встановіть Git на свій комп'ютер.
- Відкрийте командний рядок або термінал та перейдіть до директорії, де ви зберігаєте свій проект.
- Ініціалізуйте локальний репозиторій за допомогою команди **git init**.
- Додайте всі файли до локального репозиторію за допомогою команди **gitadd .** або **git add [назва файлу]**.
- Збережіть зміни в локальному репозиторії за допомогою команди **git commit -m** "повідомлення про коміт".
- Додайте віддалений репозиторій GitHub до локального репозиторію за допомогою команди **git remote add origin [посилання на репозиторій GitHub]**.
- Завантажте свій локальний репозиторій на GitHub за допомогою команди **git push -u origin master**.

Тепер локальний репозиторій зберігається на GitHub і можна керувати версіями свого коду, робити зміни та зберігати їх у своєму віддаленому репозиторії на GitHub. Також можна згодом змінити свій код та завантажити оновлення за допомогою команд **git add, git commit** та **git push**.

2.3 Створення тестового проекту

Щоб створити тестовий проект для бібліотеки Python, слід дотримуватися наступних кроків:

- Створіть нову директорію для проекту.
- Відкрийте командний рядок або термінал в цій директорії.
- Створіть віртуальне середовище за допомогою команди python -m venv

env. Ця команда створить нову директорію env, в якій будуть зберігатися всі пакети, необхідні для вашого проекту.

- Активуйте віртуальне середовище за допомогою команди, відповідної для вашої операційної системи. Наприклад, на Windows це може бути команда **.\env\Scripts\activate**, а на Linux або macOS - **source env/bin/activate**.
- Встановіть setuptools та wheel за допомогою команди **pip install setuptools wheel**.
- Створіть файл setup.py у кореневій директорії проекту та заповніть його відповідно до документації setuptools. Заповнений файл setuptools має вигляд як на рисунку 2.1.

```python
from setuptools import setup

setup(
    name='mylibrary',
    version='0.1.0',
    description='My test library',
    author='John Doe',
    author_email='johndoe@example.com',
    packages=['mylibrary'],
    classifiers=[
        'Development Status :: 3 - Alpha',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.6',
        'Programming Language :: Python :: 3.7',
        'Programming Language :: Python :: 3.8',
        'Programming Language :: Python :: 3.9',
    ],
    python_requires='>=3.6',
)
```

Рисунок 2.1 – Конфігурація файлу setuptools.

- Створіть папку mylibrary та в ній створіть файл __init__.py. Цей файл буде пустим, але він потрібен для того, щоб Python розпізнав цю директорію як пакет.
- Створіть папку tests та в ній створіть файл test_mylibrary.py. У цьому файлі ви можете написати тести для вашої бібліотеки.

На рисунку 2.2 показано тести для бібліотеки у файлі test_mylibrary.py.

Рисунок 2.2 – Тести для бібліотеки у файлі test_mylibrary.py

- Створіть файл README.md та опишіть у ньому свою бібліотеку.
- Запустіть команду **python setup.py sdist bdist_wheel**, щоб зібрати дистрибутив вашої бібліотеки. Ця команда створить два файлу в директорії dist: файл архіву джерела .tar.gz та файл бінарного архіву .whl.
- Завантажте бібліотеку на test.pypi.org за допомогою команди **twine upload --repository-url https://test.pypi.org/legacy/ dist/***. Ця команда завантажить вашу бібліотеку на тестовий сервер PyPI.
- Встановіть вашу бібліотеку з test.pypi.org за допомогою команди **pip install --index-url https://test.pypi.org/simple/ mylibrary**. Ця команда встановить вашу бібліотеку з тестового сервера PyPI.
- Імпортуйте вашу бібліотеку та скористайтеся функціями, щоб переконатися, що вони працюють.
- Якщо ви бажаєте опублікувати свою бібліотеку на основний сервер PyPI, то слід пройти реєстрацію та автентифікацію на сайті та виконати аналогічні команди завантаження і публікації бібліотеки, які описані в документації PyPI.

2.4 Налаштування тестового GitHub екшену

Настройка тестового GitHub Action (екшену) передбачає наступні кроки:

- Створіть файл .github/workflows/test.yml в кореневій папці вашого репозиторію.
- В цьому файлі визначте назву екшену та подію, за якою він буде виконуватися. Рисунок 2.3 показує назву екшену та подію за якою він буде

виконуватись.

Рисунок 2.3 – Назва екшену та подія, за якою він буде виконуватись

Визначте виконувача екшену. Це може бути контейнер з образом Docker або скрипт на мові, яку підтримує GitHub Actions. Для тестування Python-проектів можна використовувати офіційний образ Python. Для цього встановіть потрібну версію Python та додайте до PATH шлях до інтерпретатора Python. Як виглядає виконання подій можна побачити на рисунку 2.4.

Рисунок 2.4 – Виконання подій з файлу test.yml

- Визначте кроки, які будуть виконуватися при запуску екшену. Наприклад, виконайте тестові фреймворки для вашого проекту. На рисунку 2.5 зображено тестові фреймворки для проекту.

Рисунок 2.5 – Код для виконання тестових фреймворків

- Збережіть файл та закомітьте його в ваш репозиторій. Після цього GitHub автоматично запустить екшен при наступному push в гілку, на якій ви налаштували цей екшен.

- Переконайтеся, що ваш екшен успішно виконується та не містить помилок. Це можна перевірити у розділі Actions в вашому репозиторії на GitHub.

Якщо тестовий екшен успішно виконується, то ви можете перейти до наступних кроків:

- Розробіть вашу бібліотеку та додайте в неї тести. Використовуйте стандартні тести для Python, такі як unittest або pytest.
- Збережіть зміни, закомітьте їх та пушніть до вашого репозиторію на GitHub.
- Завантажте вашу бібліотеку на PyPI. Для цього ви можете використовувати setuptools та twine.
- Напишіть документацію до вашої бібліотеки та завантажте її на GitHub Pages або використовуйте інші сервіси для хостингу документації, наприклад ReadTheDocs.
- Після завантаження вашої бібліотеки на PyPI ви можете встановити її за допомогою pip на будь-якому іншому проекті. Використовуйте команду pip install <назва вашої бібліотеки> для встановлення вашої бібліотеки.
- Надайте доступ до вашого репозиторію та бібліотеки іншим розробникам, які можуть внести свій внесок до вашого проекту. Для цього ви можете використовувати систему контролю версій Git та налаштувати права доступу на вашому репозиторії на GitHub.
- Продовжуйте розробляти та підтримувати вашу бібліотеку, виправляйте баги та додавайте нові функції, щоб ваша бібліотека стала ще кращою.

3 СТВОРЕННЯ ВЛАСНОЇ БІБЛІОТЕКИ ТА АВТОМАТИЗАЦІЯ ПРОЦЕСУ ЇЇ ПАКУВАННЯ

3.1 Пакування бібліотеки у pypi

Pip — система керування пакунками, яка використовується для встановлення та управління програмними пакетами, які написані на Python.

Процес пакування бібліотеки зображено на рисунку 3.1.

Рисунок 3.1- Загальна блок-схема пакування бібліотек на PYPi.org

Для пакування своєї бібліотеки у pypi я зробив наступні кроки:

1. Переконався що у мене встановленна остання версія pip.

На рисунку 3.2 показано виконання команди **py –m pip install –upgrade pip** та її результат.

Рисунок 3.2 – Результат виконання команди **py –m pip install –upgrade pip**

Версія pip - це конкретна версія програмного забезпечення pip, яка використовується для керування пакетами Python. Версія pip може впливати на наступні аспекти:

- Функціональність: Кожна версія pip може мати різні функції та можливості. Нові версії pip можуть включати покращення, нові команди або інші функціональні зміни, які полегшують роботу з пакетами Python.
- Сумісність: Деякі пакети Python можуть мати залежності від

конкретних версій pip. У випадку, коли ви використовуєте старішу версію pip, деякі пакети можуть відмовитися встановлюватися або працювати належним чином. Оновлення до новішої версії pip може допомогти уникнути таких проблем і забезпечити сумісність з оновленими пакетами.

- Безпека: Постійні оновлення версій pip мають на меті поліпшити безпеку та захист користувачів. Нові версії можуть включати виправлення потенційних уразливостей та важливі покращення безпеки. Тому, використовуючи останню версію pip, ви забезпечуєте себе оновленнями безпеки.
- Продуктивність: Нові версії pip можуть мати оптимізації та покращення продуктивності, що дозволяють швидше та ефективніше встановлювати, оновлювати та керувати пакетами.

Узагалі, рекомендується використовувати останню стабільну версію pip, оскільки вона має найновіші функції, виправлення безпеки та покращення продуктивності.

2. Створив таку файлову структуру та файли пакетів. Як фиглядає файлова структура показано на рисунку 3.3.

Рисунок 3.3 – Вигляд файлової структури

Файлова структура Python бібліотеки має виглядати відповідно до рекомендацій та стандартів, щоб забезпечити належне функціонування бібліотеки та легкість використання для користувачів. Ось деякі причини, чому важливо дотримуватися рекомендованої файлової структури:

- Логічна організація: Відповідна файлова структура допомагає логічно організувати бібліотеку. Файли та каталоги розташовані таким чином, щоб було легко зрозуміти, де знаходяться різні частини бібліотеки, модулі, тести, документація та інші компоненти.

- Читабельність та зрозумілість: Коли користувачі чи інші розробники переглядають бібліотеку, зрозуміла структура допомагає їм швидко зорієнтуватися. Вони можуть легко знайти необхідний модуль, клас або функцію у відповідному місці.
- Інтеграція з іншими інструментами: Багато інструментів, таких як системи збирання документації, тести автоматизації, інструменти контролю версій, очікують певну файлову структуру, щоб працювати належним чином з бібліотекою. Дотримання стандартів допоможе уникнути проблем та спростити інтеграцію з цими інструментами.
- Сумісність з іншими розробниками: Якщо бібліотека буде використовуватися іншими розробниками або спільнотою, дотримання рекомендованої файлової структури забезпечить сумісність з їхніми очікуваннями та стандартами.

Загалом, дотримання рекомендацій щодо файлової структури допоможе забезпечити належне функціонування, читабельність та сумісність вашої бібліотеки з іншими інструментами та розробниками.

3. Налаштував конфігурацію файла pyproject.toml.

Конфігурацію файла pyproject.toml показано на рисунку 3.4.

```
pyproject.toml
[build-system]
requires = [
    "setuptools>=42",
    "wheel"
]
build-backend = "setuptools.build_meta"
```

Рисунок 3.4 – Конфігурація файла pyproject.toml

Файл pyproject.toml є текстовим файлом, який використовується в проектах Python для конфігурування та опису проекту, його залежностей та інших налаштувань. Цей файл став популярним з появою інструменту poetry, який дозволяє зручно управляти проектами та їх залежностями.

4. Налаштував конфігурацію файла setup.cfg.

На рисунку 3.5 показано конфігурацію файла setup.cfg.

Рисунок 3.5 – Конфігурація файла setup.cfg

Файл setup.cfg є конфігураційним файлом, який використовується для налаштування параметрів пакування та розповсюдження бібліотеки Python. Він зазвичай використовується разом з файлом setup.py для налаштування пакета та його метаданих.

5. Переконався, що встановлено останню версію збірки PyPA :

Результат виконання коду **py –m pip install –upgrade build** показано на рисунку 3.6.

Рисунок 3.6 – Результат виконання коду **py –m pip install –upgrade build**

Переконання, що встановлена остання збірка PyPA (Python Packaging

Authority), є важливим з кількох причин:

- Нові функції та покращення: Остання версія збірки PyPA може включати нові функції, покращення та виправлення помилок. Встановлення останньої версії дозволить скористатися оновленими можливостями та поліпшеннями.
- Сумісність з іншими інструментами: Інші інструменти, такі як pip, setuptools, twine та інші, можуть мати залежності від певних версій збірки PyPA.
- Виправлення безпеки: Оновлення до останньої збірки PyPA може включати виправлення відомих безпекових проблем. Це дозволить забезпечити більш високий рівень безпеки для бібліотеки та іншого програмного забезпечення.
- Оновлення стандартів та рекомендацій: PyPA вносить оновлення до стандартів та рекомендацій, які стосуються пакування та розповсюдження Python-проектів. Використання останньої збірки PyPA допоможе залишатися синхронізованим з найновішими стандартами та кращими практиками розробки.

6. Запустив команду для створення каталогу dist та двох файлів у ньому.

Результат виконання команди для створення каталогу dist та двох файлів у ньому показано на рисунках 3.7, 3.8, 3.9 та 3.10.

Рисунок 3.7 – Перша частина результату виконання команди **python –m build**

Рисунок 3.8 – Друга частина результату виконання команди **python –m build**

Рисунок 3.9 – Третя частина результату виконання команди **python –m build**

Рисунок 3.10 – Створенні файли у результаті виконання команди **python –m build**

Команда **python -m build** створює розподільчий архів проекту, який може бути використаний для пакування та розповсюдження бібліотеки Python. При виконанні цієї команди, буде створений каталог dist/:, в якому будуть збережені створені розподільчі архіви. Це можуть бути файли формату .tar.gz, .tar.bz2 або .whl (wheel), залежно від налаштувань проекту.

7. Зареєструвався на test.pypi.org (те ж саме що і pypi.org але надає

можливість спробувати засоби і процеси, не впливаючи на справжній реєстр.).

Як виглядає зареєстрований аккаунт на PyPI можна побачити на рисунку 3.11.

Рисунок 3.11 – Зареєстрований аккаунт на PyPI

8. Додав API-токен.

Вигляд створеного API-токену зображено на рисунку 3.12.

Рисунок 3.12 – Створений API-токен

API Token (також відомий як API ключ або токен доступу) є унікальним рядком символів, який надається користувачам або розробникам для автентифікації та авторизації при взаємодії з API (Application Programming Interface).

API Token використовується для ідентифікації особистості або додатка, що взаємодіє з API, і забезпечення доступу до відповідних ресурсів або функціональності. Цей токен може бути використаний для авторизації запитів, обмеження доступу до конкретних операцій або ресурсів, а також для відстеження використання API.

API Token є чутливою інформацією, яку необхідно зберігати в безпечному місці і не розголошувати непов'язаним особам або додаткам. Користувачі часто отримують API Token шляхом реєстрації акаунта на платформі, яка надає API, і створення спеціального ключа доступу для взаємодії з цим API.

API Token дозволяє розробникам отримати доступ до функціональності платформи, обмінюватися даними, створювати, зчитувати, оновлювати або видаляти ресурси, виконувати запити та отримувати відповіді з API. Кожна платформа може мати власні правила та процедури для отримання та використання API Token, тому слід дотримуватись документації та рекомендацій постачальника API.

9. Встановив Twine.

На рисунку 3.13 зображено результат виконання команди **py –m pip install –upgrade twine**.

Рисунок 3.13 – Результат виконання команди **py –m pip install –upgrade twine**

Twine - це інструмент командного рядка для завантаження пакетів Python на платформу PyPI (Python Package Index). Він дозволяє розробникам легко публікувати свої бібліотеки та пакети на PyPI, що дозволяє іншим користувачам легко встановлювати ці пакети за допомогою `pip`.

Основні функції та використання Twine:

- Завантаження пакетів на PyPI: Twine надає зручний спосіб завантажувати ваші пакети на PyPI. Ви можете використовувати Twine для відправки збудованих розподільчих архівів (`.tar.gz`, `.tar.bz2`, `.whl`) на PyPI для подальшого поширення.
- Аутентифікація: Twine дозволяє вам аутентифікуватися на PyPI, використовуючи ваші облікові дані. Це забезпечує конфіденційність і забезпечує, що тільки авторизовані розробники можуть завантажувати пакети на PyPI.
- Перевірка пакетів перед завантаженням: Twine автоматично перевіряє ваші пакети перед завантаженням на PyPI. Це включає перевірку на відповідність стандартам пакування, наявність необхідних метаданих та інші перевірки, щоб забезпечити якість та цілісність пакетів.
- Керування версіями: Twine допомагає управляти версіями вашого пакету. Ви можете вказати версію пакету відповідно до стандартів версіонування Python і переконатися, що нові версії пакету правильно ідентифікуються та оновлюються на PyPI.

10.Запустив Twine, щоб завантажити всі архіви в dist.

Результат виконання команди **py –m twine upload –repository testpypi dist/*** зображено на рисунку 3.14.

Рисунок 3.14 – Результат виконання команди **py –m twine upload –repository testpypi dist/***

Основні кроки, які Twine виконує під час завантаження архівів:

- Twine перевіряє наявність усіх архівів в каталозі dist. Це можуть бути архіви, згенеровані за допомогою команди python -m build або будь-яким іншим інструментом для пакування.
- Twine перевіряє наявність усіх архівів в каталозі dist. Це можуть бути архіви, згенеровані за допомогою команди **python -m build** або будь-яким іншим інструментом для пакування.
- Twine використовує ваш API Token PyPI (який ви надаєте) для автентифікації на платформі PyPI.
- Після автентифікації Twine починає відправку кожного архіву на сервер PyPI за допомогою захищеного протоколу передачі даних (наприклад, HTTPS).
- Після завантаження архівів на сервер PyPI, Twine автоматично перевіряє їх валідність та відповідність вимогам PyPI. Це включає перевірку правильності метаданих, наявності необхідних файлів та відповідність іменування стандартам PyPI.
- Після успішної перевірки архівів, Twine оновлює метадані вашого пакета на сервері PyPI. Це включає оновлення версії, опису, ключових слів та інших важливих даних, які допомагають користувачам знаходити та розуміти ваш пакет.
- Після завершення завантаження кожного архіву Twine отримує відповідь від сервера PyPI, яка підтверджує успішне завантаження архіву.
- Цей процес повторюється для кожного архіву, який знаходиться в каталозі dist.

11. Перевірив чи завантажилась бібліотека на pypi.

На рисунку 3.15 зображено завантажену на pypi.org бібліотеку.

Рисунок 3.15 – Завантажена на pypi.org бібліотека

3.2 Автоматизація GitHub Actions для пакування бібліотеки та її релізу у pypi

Автоматизація GitHub Actions для пакування бібліотеки та її релізу у PyPI дозволяє спростити та прискорити процес розробки та поширення вашої бібліотеки. Завдяки цій автоматизації, ви можете встановити послідовність дій, які будуть виконуватися автоматично при кожному коміті, забезпечуючи актуальну версію пакету та його належну публікацію на PyPI.

На рисунку 3.16 зображено блок-схему, яка показує процес автоматизаї GitHub Actions для пакування бібліотеки, а на рисунку 3.17 зображено блок-схему процесу релізу бібліотеки у pypi.

Рисунок 3.16 – Блок-схема автоматизації пакування бібліотек за допомогою GitHub Actions

Рисунок 3.17 – Блок-схема тегування та створення GitHub Release для PyPI

Для автоматизації GitHub Actions для пакування та релізу у pypi було виконано наступні кроки:

1. Завантажив бібліотеку на GitHub.

На рисунку 3.18 зображено бібліотеку у GitHub репозиторії.

Рисунок 3.18 – Бібліотека у репозиторії на GitHub

2. Через GitHub Actions створив у рипозиторії .github/workflows/ python-publish.yml.

Створений у рипозиторії github/workflows/ python-publish.yml. зображено на рисунку 3.19.

Рисунок 3.19 – github/workflows/ python-publish.yml. у рипозиторії

Файл python-publish.yml у каталозі .github/workflows в репозиторії на GitHub є частиною конфігурації GitHub Actions і використовується для автоматизації процесу публікації бібліотеки Python на PyPI (Python Package Index).

Основна мета цього файла - налаштувати робочий процес, який буде автоматично виконувати певні дії після події, такої як коміт або створення тегу в репозиторії. В цьому конкретному випадку, python-publish.yml налаштовує процес пакування та публікації бібліотеки на PyPI за допомогою інструментів, таких як setuptools, twine та PyPI.

Нижче наведено деякі основні етапи, які можуть бути включені в python-publish.yml:

- Встановлення необхідних залежностей: Зазвичай, на початку робочого процесу будуть виконуватись дії для встановлення необхідних пакетів Python та інших залежностей, які потрібні для пакування та публікації бібліотеки.
- Збирання бібліотеки: Цей етап включає виконання команд для збирання бібліотеки, наприклад, використання setuptools для створення розподільчого архіву (.tar.gz, .tar.bz2, .whl) з кодом бібліотеки.
- Перевірка бібліотеки: Можуть бути включені кроки для перевірки бібліотеки, такі як запуск тестів, аналіз стилю коду, перевірка покриття тестами тощо.
- Публікація на PyPI: Після успішної перевірки бібліотеки можна виконати команди для автоматичної публікації архіву на PyPI за допомогою twine або аналогічних інструментів. Цей етап включає введення API-токену PyPI, який дозволяє автентифікувати публікацію.
- Інші додаткові дії: Залежно від потреб, можуть бути включені додаткові кроки, такі як створення Git-тегу, оновлення документації, створення релізу на GitHub тощо.

3. Додав PYPI_API_TOKEN у Repository secrets.

На рисунку 3.20 зображено PYPI_API_TOKEN у Repository secrets.

Рисунок 3.20 – PYPI_API_TOKEN у Repository secrets

У GitHub Repository secrets додається PYPI_API_TOKEN (або будь-яке інше ім'я, яке ви обираєте) для зберігання API-токену PyPI у безпечному місці. Це потрібно з метою забезпечення безпеки вашого API-токену та уникнення його викриття у відкритому вигляді.

API-токен PyPI використовується для автентифікації вас як користувача під час взаємодії з PyPI (Python Package Index). Він дозволяє вам завантажувати, оновлювати або видаляти пакети з PyPI.

Додавши PYPI_API_TOKEN до GitHub Repository secrets, ви можете використовувати цей токен у своєму робочому процесі GitHub Actions або інших автоматизованих діях. Наприклад, ви можете використовувати його для автоматичної публікації вашої бібліотеки Python на PyPI за допомогою інструментів, таких як twine або setuptools.

Важливо зберігати API-токен PyPI у секреті, оскільки він надає доступ до вашого облікового запису PyPI. GitHub Repository secrets забезпечують безпечне зберігання цього токену, а також дозволяють його використання у ваших автоматизованих робочих процесах, не розкриваючи його значення в публічних налаштуваннях вашого репозиторію.

4. Створив новий реліз бібліотеки тим самим завантажив її на pypi.org.

Успішне завантаження бібліотеки на pypi.org через GitHub Realeases та завантажену бібліотеку зображено на рисунках 3.21 та 3.22.

Рисунок 3.21 – Успішне завантаження бібліотеки на pypi.org через GitHub Realeases

Рисунок 3.22 – Завантажена бібліотека на pypi.org

Новий реліз бібліотеки відноситься до випуску нової версії бібліотеки з певними змінами, вдосконаленнями або виправленнями. Кожен реліз має унікальний номер версії, що дозволяє ідентифікувати конкретну версію бібліотеки.

Релізи бібліотеки використовуються для наступних цілей:

- Інформування користувачів: Коли ви випускаєте новий реліз бібліотеки, ви надаєте користувачам інформацію про оновлення, зміни, виправлення помилок та нові функції, які включені в цю версію. Це дозволяє користувачам бібліотеки бути в курсі оновлень та використовувати новий функціонал.
- Версіонування: Використання релізів дозволяє здійснювати версіонування вашої бібліотеки. Зміна номера версії (наприклад, від 1.0.0 до 1.1.0) вказує на наявність нових змін або вдосконалень у вашій бібліотеці. Версіонування допомагає керувати сумісністю між різними версіями бібліотеки та дозволяє користувачам вибирати певну версію, яка відповідає їх потребам та вимогам.
- Управління залежностями: Релізи бібліотеки дозволяють вам керувати залежностями в інших проектах. Інші розробники можуть вказувати певну версію вашої бібліотеки як залежність у своїх проектах. Це спрощує управління залежностями та дозволяє забезпечити сумісність між версіями бібліотек.
- Зручність для розробників: Релізи бібліотеки надають зручний спосіб оновлення та використання вашої бібліотеки розробниками. Вони можуть оновлювати версію вашої бібліотеки на своїх проектах, отримувати виправлення помилок або новий функціонал безпосередньо через нові релізи.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

4.1 Основні вимоги до користувачів ПК з охорони праці, техніки безпеки, пожежної безпеки

При роботі користувач ПК зобов'язаний виконувати умови інструкції з експлуатації ПК.

До роботи з ПК допускаються працівники та учні, з якими був проведений вступний інструктаж та первинний інструктаж (на робочому місці) з питань охорони праці, техніки безпеки, пожежної безпеки та зроблений запис про їх проведення у спеціальному журналі інструктажів.

Працівники та учні при роботі з ПК повинні дотримуватися вимог техніки безпеки, пожежної безпеки.

При виявленні в обладнанні ПК ознак несправності (іскріння, пробоїв, підвищення температури, запаху гару, ознак горіння) необхідно негайно припинити роботу, відключити все обладнання від електромережі і терміново повідомити про це відповідних посадових осіб, спеціалістів.

Вміти діяти в разі ураження інших працівників, учнів електричним струмом або виникненні пожежі.

Знати місця розташування первинних засобів пожежогасіння, план евакуації працівників з приміщення в разі виникнення пожежі.

4.2 Вимоги охорони праці до приміщення для роботи з ПК

Стіни приміщень для роботи з ПК мають бути пофарбовані чи обклеєні шпалерами спокійних кольорів з коефіцієнтом відбиття 40% – 60%. Вікна повинні обладнуватися сонцезахисними пристроями.

Для освітлення приміщень з ПК необхідно використовувати люмінесцентні світильники. Освітленість робочих місць у горизонтальній площині на висоті 0,8 м від підлоги повинна бути не менше 400 лк. Вертикальна освітленість у площині екрану не більше 300 лк.

У приміщеннях для роботи з ПК необхідно проводити вологе прибирання та регулярне провітрювання протягом робочого дня. Видалення пилу з екрану необхідно проводити не рідше одного разу на день.

4.3 Вимоги охорони праці до робочого місця користувача ПК

Робочі місця для працюючих з дисплеями необхідно розташувати таким чином, щоб до поля зору працюючого не потрапляли вікна та освітлювальні прилади. Відео термінали повинні встановлюватися під кутом 90°-105° до вікон та на відстані не меншій 2,5 м від стін з вікнами. До поля зору працюючого з дисплеєм не повинні потрапляти поверхні, які мають властивість віддзеркалювання. Покриття столів повинне бути матовим з коефіцієнтом відбиття 0,25-0,4.

Відстань між робочими місцями ПК повинна бути не меншою 1,5-му ряду та не меншою 1м між рядами. ПК повинні розміщуватися не ближче 1м від джерела тепла, з метою забезпечення вільного доступу, комфорту та безпеки працівників.

Відстань від очей користувача до екрану повинна становити 500 мм - 700 мм, кут зору 10°-20°, але не більше 40°, кут між верхнім краєм відео терміналу та рівнем очей користувача повинен бути меншим 10°. Найбільш вигідним є розташування екрану перпендикулярно до лінії зору користувача, щоб забезпечити оптимальний комфорт та знизити навантаження на очі та шию під час роботи з комп'ютером.

4.4 Вимоги до режиму охорони праці

Загальний час роботи з відео терміналом не повинен перевищувати 50% тривалості робочого дня, щоб не нашкодити здоров’ю. Для робіт, які виконуються з великим навантаженням, слід робити 10-15 хвилинну перерву через кожну годину, для мало інтенсивної роботи такі перерви потрібно робити через 2 години. Кількість мікро пауз (до 1 хвилини) слід визначити індивідуально.

4.5 Вимоги техніки безпеки до користувачів ПК

Перед початком роботи на ПК користувач повинен: - пересвідчуватися у цілості корпусів і блоків (обладнання) ПК; перевірити наявність заземлення, справність і цілісність кабелів живлення, місця їх підключення. Забороняється вмикати ПК та починати роботу при виявлених несправностях.

Під час роботи пересвідчившись у справності обладнання, увімкнувши електроживлення ПК, розпочинати роботу, дотримуючись умов інструкції з його експлуатації.

Забороняється:

- замінювати і знімати елементи або вузли та проводити перемонтаж при ввімкненому ПК;
- з'єднувати і роз'єднувати вилки та розетки первинних мереж електроживлення, які знаходяться під напругою;
- знімати кришки, які закривають доступ до струмопровідних частин мережі первинного електроживлення при ввімкнутому обладнанні;
- користуватися паяльником з незаземленим корпусом;
- замінювати запобіжники під напругою;
- залишати ПК у ввімкнутому стані без нагляду.
- По закінченню робочого дня:
- послідовно вимкнути ПК, кнопкою „ВИМК" відключити електроживлення ПК згідно з інструкцією по експлуатації, і вийняти вилку кабелю живлення з розетки.
- впорядкувати робоче місце користувача ПК, прибрати використане обладнання та матеріали у відведені місця.
- при виявленні недоліків у роботі ПК протягом робочого часу необхідно повідомити посадовим особам та спеціалістам.
- залишаючи приміщення після закінчення робочого дня, дотримуючись встановленого режиму огляду приміщення, необхідно:
- зачинити вікна, квартирки, штори;

- перевірити приміщення та переконатися у відсутності тліючих предметів;
- відключити від електромережі всі електроприлади, електрообладнання та вимкнути освітлення;
- здати приміщення черговому вахтеру під охорону.

4.6 Основні вимоги до користувачів засобів оргтехніки з охорони праці, техніки безпеки, пожежної безпеки

Користувачі засобів оргтехніки зобов'язані завжди виконувати інструкції, що поставляються разом з засобами оргтехніки:

- підключати ксерокопіювальне обладнання підключати тільки до заземленої розетки живлення, неправильне підключення може привести до ураження електричним струмом;
- дотримуватися обережності при переміщенні засобів оргтехніки, машини, обладнані сортером, не повинні переміщуватися без нагляду спеціаліста по технічному обслуговуванню;
- завжди використовувати матеріали, спеціально призначені для відповідних засобів оргтехніки;
- використання непридатних матеріалів може призвести до поганої роботи оргтехніки або аварійним ситуаціям та залишати вільними вентиляційні отвори, призначені для запобігання перегрівання;

Засоби оргтехніки встановлюються в приміщеннях, що добре вентилюються і мають достатню площу для обслуговування.

4.7 Дії працівників у разі ураження людини електричним струмом

Терміново звільнити потерпілого від електричного струму (через відключення електроживлення в кабінеті, загального електроживлення на роздільному щиті або іншим способом):

- За показниками стану здоров'я потерпілого викликати швидку медичну допомогу (подзвонивши за міським телефоном – 03);
- При відсутності ознак життя до прибуття лікарів потерпілому необхідно робити штучне дихання.

4.8 Дії працівників у разі виникнення пожежі

У разі виникнення пожежі в приміщеннях кожен працівник і зобов'язані:

- оцінити обстановку;
- негайно повідомити про виникнення пожежі керівництво;
- вжити заходів з використанням вогнегасників для гасіння вогню і локалізації пожежі;
- при сигналі оповіщення про пожежу здійснити евакуацію з приміщення згідно загального „Плану евакуації людей на випадок виникнення пожежі;
- при необхідності повідомити пожежну охорону за міським телефоном – 101; при цьому необхідно назвати адресу , вказати поверх будівлі, обстановку на пожежі, наявність людей, а також повідомити своє прізвище;
- у разі необхідності, самому або через вахтера викликати інші аварійно- рятувальні служби: медичну, електромережі.

4.9 Дії працівників при необхідності отримання медичної допомоги

Для забезпечення невідкладної медичної допомоги для працівників та учнів у разі гіпертонічної кризи, серцево-судинній недостатності, порізів, та інших ушкоджень, тримати в кабінеті набір необхідних лікарських засобів.

При виникненні серйозної небезпеки для здоров'я працівників та учнів викликати невідкладну медичну службу за телефоном - 103.

5 ТЕХНІКО-ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

Завданням дипломної роботи є розробка програмного забезпечення для пошуку найкоротших шляхів у мережах зв'язку. Ефект від використання проєктного рішення полягає у пришвидшені часу отримання результатів. Економічний ефект від використання проєктного рішення полягає в зменшенні часових витрат на роботу з кінцевим споживачем.

5.1 Розрахунок витрат на розробку та впровадження проєктного рішення

Витрати на розробку та впровадження програмних засобів (К) включають:

$$K=K_1+K_2, \tag{5.1}$$

де К1 – витрати на розробку програмного продукту, грн.; К2 – витрати на налагодження та дослідну експлуатацію програмного засобу на ПК, грн.

Витрати на розробку програмного засобу включають в себе:

- Витрати на оплату праці розробників (З);
- Нарахування на зарплату (НЗ);
- Витрати на куповані витрати (ВК);
- Накладні витрати (НВ);
- Інші витрати (Інші).

Для розробки програмного продукту потрібні чотири спеціалісти-розробники, а саме:

- Керівник проєкту (К);
- Студент дипломник (СД);
- Консультант з охорони праці (КОП);
- Консультант з економічної частини (КЕ).

Згідно з штатним розписом Відокремленого структурного підрозділу «Фахового коледжу інформаційних технологій Національного університету «Львівська політехніка» місячною стипендією студента-дипломника є 1510 грн, а 1 година робочого навантаження становить для:

- Керівника проєкту – 87,06 грн;
- Консультанта з економічної частини – 106,96 грн;
- Консультанта з охорони праці – 93,70 грн.
- Денна оплата студента дипломника визначається:

$$1510/173 = 8{,}73 \text{ (грн)},$$

де 173- місячний фонд робочого часу, годин.

Розрахунок витрат на оплату праці всіх спеціалістів проєкту визначається за формулою:

$$В_{оп} \sum_{i=0}^{N} n_j * n_n * ЗП\ г_i\ ; \qquad (5.2)$$

де n_n – чисельність розробників проєкту 1-ої спеціальності чол.;n_n – час, витрачений на розробку проєкту працівником 1-ої спеціальності, дні; ЗП $г_i$ – погодинна заробітна плата розробника 1-ої спеціальності, грн;

Таким чином, витрати на оплату праці розробників складають:

- Зк = 1* 14 * 87,06= 1218,84 (грн);
- Зке = 1 * 1 * 106,96 = 106,96 (грн) ;
- Зкоп = 1 * 1 *93,70 = 93,70(грн).
- Зст = 1 * 200 * 8,73 = 1746,00(грн).

Сумарні витрати на оплату праці:

$$Воп= Зк+ Зке+ Зст+ Зкоп.$$

Відповідно,

$$Воп=1218{,}84 + 106{,}96 + 93{,}70 + 1746{,}00=3165{,}5 \text{ (грн.)}$$

Розрахунок витрат на оплату праці розробників наведено у таблиці 5.1.

Таблиця 5.1 – Розрахунок витрат на оплату праці

Спеціальність розробника	Кількість розробників роб.	Час роботи, год.	Погодинна заробітна плата розробника, грн.	Витрати на оплату праці, грн.
Керівник проєкту	1	14	87,06	1218,84
Консультант з економічної частини	1	1	106,96	106,96
Консультант з охорони праці	1	1	93,70	93,70
Студент-дипломник	1	200	8,73	1 746,00
Всього	4	216	-	3165,5

Нарахування на зарплату становить згідно з нормативом 22% від фонду оплати праці:

$$Нз= Воп*22,0/100, \quad (5.3)$$

де Воп – витрати на оплату праці, тис.грн.; 22 – норматив нарахувань на зарплату,%.

$$Нз= 3165,5* 0,22 = 696,41 \text{ (грн)}.$$

5.2 Розрахунок витрат на куповані вироби

Рахування витрат на куповані вироби в дипломній роботі має декілька цілей:

- Оцінка фінансової ефективності
- Доцільність використання виробів
- Документація та облік
- Оцінка ефективності роботи

Рахування витрат на куповані вироби допомагає керувати фінансами та

Для того, щоб порахувати витрати на куповані вироби (папір, друк) ми знаємо, що розрахунки визначаються за їх фактичними цінами з врахуванням найменування, номенклатури та необхідної кількості в проєкті. Транспортно-заготівельні витрати становлять 10% від суми витрат на куповані вироби. Розрахунок витрат на куповані вироби наведено в табл. 5.2.

Таблиця 5.2 – Розрахунок витрат на куповані вироби

Найменування купованих виробів	Марка, тип	Кількість на розробку, шт.	Ціна за одиницю, грн.	Сума витрат, грн.
Папір, пачок	Формат А4	1	198	198
Роздрук пояснювальної записки	Формат А4	51	1,5	76,5
Зшивання диплому в тверду палітурку		51	180	180
Транспортно-заготівельні витрати (10%)	–	–	–	45,45
Разом	–	–	–	454,5

Витрати на куповані вироби становлять:

Вк = 198+ 76,5 + 180 + 45,45 = 499,95 (грн.).

5.3 Розрахунок накладних та інших витрат

Накладні витрати (Нв) розраховуються за встановленими відсотками (30%) до витрат на оплату праці:

Нв= 3165,5* 30/100 = 949,65 (грн).

Інші витрати розраховуються по їх питомій вазі у структурі собівартості (10%):

Він = (Воп+Нз+Нв+Вк)*10/100; (5.4)

Він = (3165,5+ 696,41 + 949,65 + 499,95)*10/100 = 531,15 (грн);

Витрати на розробку проєктного рішення визначаються за формулою:

К1=Воп+Нз+Нв+Вк+Він;

(5.5)

$$К1= 3165,5 + 696,41 + 949,65 + 499,95 + 531,15 = 5842,66 \text{ (грн)};$$

5.4 Розрахунок витрат на налагодження проєктного рішення

Програма була розроблена протягом 50 днів із розрахунком 4 год на день (t=200 год.).

Потужність обчислювальної техніки (P) включає ноутбук, який споживає 0,47 кВт*год. Отже, вартість однієї машино-години роботи визначається за формулою:

$$Sm.r.= P*Тф, \tag{5.6}$$

де P – потужність комп'ютерної техніки, кВт; Тф - вартість 1 кВт-год електроенергії, грн. (Тф=5,43).

$$Sm.r=0,47* 5,43= 2,55 \text{ (грн/год)}.$$

Для того, щоб отримати розрахунок витрат на налагодження та дослідну експлуатацію програмного продукту на ПК потрібно скористатися нижче наведеною формолою:

$$К2= Sm.r * t, \tag{5.7}$$

де Sm.r. – вартість однієї машино-години роботи, грн\год; t – машинний час, витрачений на налагодження та дослідну експлуатацію програмного продукту, год.

$$К2 = 2,55 * 200 = 510,00 \text{ (грн)}.$$

Звідси нам відомо, що витрати на розробку та впровадження програмного продукту становлять:

$$К = 5842,66 + 510,00 = 6352,66 \text{ (грн)}.$$

Кошторис витрат на розробку та впровадження проєктного рішення наведений в таблиці 5.3.

Таблиця 5.3 Кошторис витрат розробки та реалізації програмного продукту.

Найменування елементів витрат	Сума витрат, грн..
Витрати на оплату праці	3165,5
Нарахування на зарплату	696,41
Витрати на куповані вироби	454,5
Накладні витрати	949,65
Інші витрати	531,15
Витрати на налагодження та дослідну експлуатацію	510,00
Всього (К=К1+К2)	6352,66

Таким чином, загальні витрати на розробку та впровадження проєктного рішення становлять 6352,66 (грн.).

ВИСНОВКИ

Створення Python бібліотеки - це важливий процес, який дозволяє розробникам створювати та ділитися своїм кодом з іншими користувачами. Бібліотека може бути використана для вирішення різноманітних задач, зокрема для розробки веб-додатків, машинного навчання, обробки даних та багатьох інших.

Для створення бібліотеки необхідно спочатку визначити функціональність бібліотеки та API, який буде використовуватися користувачами. Потім можна перейти до розробки коду, написання тестів, документації та налаштування інтеграції з системами контролю версій та автоматичними тестами.

Крім того, є кілька важливих етапів, які потрібно пройти для успішного створення бібліотеки, такі як завантаження бібліотеки на PyPI та налаштування доступу для інших розробників.

Узагалі, створення Python бібліотеки може зайняти деякий час та потребувати деякої експертизи, але це може бути дуже корисно для розробників, які хочуть ділитися своїм кодом та зробити його доступним для інших користувачів.

У цій дипломній роботі було створено просту, тестову бібліотеку, яка буде використана в цілях виконання автоматизації процесу її пакування. Функціонал бібліотеки виконує базову операцію з конкатенації текстових стрічок для створення емейл адреси. Тобто, завдяки цій бібліотеці можна буде створити емейл адресу, ввівши ім'я та прізвище людини, після чого користувач отримає готову емейл адресу. Також я підготував бібліотеку до завантаження на PyPI.org , завантажив її через командний рядок. Після цього я зробив процес завантаження на PyPI.org автоматичним, створивши конфігураційний файл з кроками які повторюють всі виконані вручну операції. У репозиторій GitHub були завантажені всі необхідні файли для створення бібліотеки Python та використано GitHub Actions для автоматичного завантаження та Gitub Releases для тегування репозиторію з бібліотекою. Якщо порівняти процес створення та завантаження бібліотеки в ручну і автоматизований процес, то очевидним вибором буде автоматизований, оскільки він не займає багато часу та не потребує виконувати одні й ті самі кроки введення команд кожного разу. Завдяки GitHub Actions та GitHub Releases все що потрібно, це всього навсього натиснути кілька кнопок, і все буде виконано автоматично, без додаткових зусиль.

У даній дипломній роботі також техніку безпеки та охорону праці при роботі з комп'ютером. Підвищення свідомості користувачів щодо користування ПК безпечним способом є важливим аспектом охорони праці. Навчання правильним технікам роботи з ПК, а також інформування про можливі ризики та способи їх запобігання можуть значно знизити ймовірність виникнення травм, розладів здоров'я та небажаних ситуацій. Крім цього було проведено розрахунки усіх витрат на розробку та впровадження проєктивного рішення, витрат на куповані вироби та витрат на наголодження проєктивного рішення, та проведено розрахунок накладних та інших витрат.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Creating a Python library" - документація Python з поясненням процесу створення бібліотек:
https://docs.python.org/3/tutorial/stdlib.html#creating-modules
2. "Packaging Python Projects" - документація Python про те, як упакувати свій проект для розповсюдження:
https://packaging.python.org/
3. "Python Packaging User Guide" - посібник для розробників, який описує різні методики пакування Python-проектів:
https://python-packaging-user-guide.readthedocs.io/
4. "How to Write a Python Library" - огляд кроків та кращих практик щодо створення бібліотек на Python:
https://realpython.com/python-libraries/
5. "Python library example" - приклад створення Python-бібліотеки з детальним описом кроків:
https://github.com/hassanhabib/python-library-example
6. "Creating a Python Package" - покроковий огляд створення Python-пакету, який включає в себе бібліотеку:
https://www.learnpython.org/en/Modules_and_Packages#Creating_a_Python_Pa ckage
7. "How to Publish an Open-Source Python Package to PyPI" - опис кроків для опублікування бібліотеки на PyPI:
https://www.twilio.com/blog/how-to-publish-an-open-source-python-package-to-pypi
8. "A Guide to Python's Virtual Environments" - огляд того, що таке віртуальні середовища та як їх використовувати для розробки та тестування Python бібліотек:
https://docs.python-guide.org/dev/virtualenvs/
9. "GitHub Actions Documentation - GitHub Docs" - офіційна документація GitHub Actions:
https://docs.github.com/en/actions
10. "Towards Data Science" - посібник у блозі Towards Data Science про створення

Python-бібліотеки з GitHub Actions:
https://towardsdatascience.com/how-to-write-and-publish-a-python-library-on-pypi-using-github-actions-8c9f2dfb2fa4

11. "actions/python-versions: Python builds for Actions Runner Images" – офіційний репозиторій GitHub Actions з багатьма прикладами використання для Python проектів:
https://github.com/actions/python

12. "Continuous Integration With Python: An Introduction – Real Python" - блог-пост на Real Python про автоматизацію тестування Python-бібліотек з GitHub Actions та Codecov:
https://realpython.com/python-continuous-integration/

13. "sdras/awesome-actions: A curated list of awesome actions to use on GitHub" репозиторій з прикладами GitHub Actions для Python-проектів: https://github.com/sdras/awesome-actions

14. "Python Packaging User Guide — Python Packaging User Guide" – офіційна документація Python Packaging:
https://packaging.python.org/

15. "Automating Python package publishing with GitHub Actions" - стаття, яка показує, як налаштувати GitHub Actions для автоматичного публікування Python-пакетів:
https://dev.to/danialmalik/automating-python-package-publishing-with-github-actions-3jfg

16. "Automating your Python project with GitHub Actions" - практичний посібник щодо автоматизації проекту Python з використанням GitHub Actions:
https://medium.com/swlh/automating-your-python-project-with-github-actions-9a82dbad35a3

17. "Continuous Integration for Python Projects with GitHub Actions" - огляд використання GitHub Actions для неперервної інтеграції Python-проектів: https://www.toptal.com/developers/github-actions/continuous-integration-python

18. "Automate Python package releases with GitHub Actions" - стаття, яка

розповідає про автоматизацію релізів Python-пакетів за допомогою GitHub Actions: https://levelup.gitconnected.com/automate-python-package-releases-with-github-actions-78da0a57cf26

ДОДАТОК А

Перелік кодів основних файлів

Python код бібліотеки яка робить конкатенацію стрічок щоб зробити емейл адресу

```
def create_email_address(name, domain):
    return name + "@" + domain
```

Код файлу python-publish.yml, який автоматизує пакування бібліотеки через GitHub Actions

```
name: Publish to PyPI.org
on:
  release:
    types: [published]
jobs:
  pypi:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
        with:
          fetch-depth: 0
      - run: python3 -m pip install --upgrade build && python3 -m build
      - name: Publish package
        uses: pypa/gh-action-pypi-publish@release/v1
        with:
          password: ${{ secrets.PYPI_API_TOKEN }}
```

Код файлу setup.cfg

```
[metadata]
name = mypackage_himza_itcollege_lviv_ua
version = 0.0.2
author = Dmytro Himza
author_email = dhimza2020@itcollege.lviv.ua
description = Module for creating an email address
long_description = file: README.md
long_description_content_type = text/markdown
url = https://github.com/pypa/sampleproject
project_urls =
    Bug Tracker = https://github.com/pypa/sampleproject/issues
classifiers =
    Programming Language :: Python :: 3
    License :: OSI Approved :: MIT License
    Operating System :: OS Independent

[options]
package_dir =
    = src
packages = find:
python_requires = >=3.6

[options.packages.find]
where = src
```

ДОДАТОК Б

Перелік кодів тестових файлів

Код тестового файлу setup.cfg

```
from setuptools import setup

setup( name='myli
 bary',
 version='0.1.0',
 description='My test library',
 author='Jhon Doe',
 author_email='jhondoe@exapmle.com'
 packeges=['mylibrary']
 classifiers=[
  'Development Status :: 3 - Alpha',
  'Programming Language :: Python :: 3',
  'Programming Language :: Python :: 3.6',
  'Programming Language :: Python :: 3.7',
  'Programming Language :: Python :: 3.8',
  'Programming Language :: Python :: 3.9',
 ],
 python_requires='>=3.6',
)
```

Код тестового файлу test.yml

```
jobs:
 test:
runs-on: ubuntu-latest
 container:
```

```
 image: python 3.9
steps:
 - name: Checkout code
   uses: actions/checkout@v2
- name: Setup Python
run:
   python -m pip install --upgrade pip
python -m pip install -r requirements.txt
env:
        PATH: ${{pythonLocation}}:$PATH
```

КОПІЇ ОБОВ'ЯЗКОВИХ КРЕСЛЕНЬ

					Автоматизація пакування Python бібліотек за допомогою GitHub Actions			
					Загальна блок-схема пакування бібліотек на PyPI.org	Літера	Маса	Масштаб
Зм	*Арк*	*№ докум.*	*Підпис*	*Дата*				

Розроб.	Гімза Д. П.			
Керівник	Бугиль Б. А.			
Реценз.				
Н.контр.	Кужій Л. І.			
Затверд.	Шумлянський В.О.			

Аркуш	Аркушів
1	4

					Автоматизація пакування Python бібліотек за допомогою GitHub Actions			
					Блок-схема автоматизації пакування бібліотек за допомогою GitHub	Літера	Маса	Масштаб
Зм	Арк	№ докум.	Підпис	Дата				

				Actions		
. .						
Розроб.	Гімза Д. П.					
Керівник	Бугиль Б. А.					
Реценз.					Аркуш 2	Аркушів 4
Н.контр.	Кужій Л. І.					
Затверд.	Шумлянський В.О.					

					Автоматизація пакування Python бібліотек за допомогою GitHub Actions		
Зм	Арк	№ докум.	Підпис	Дата	Блок-схема тегування та створення GitHub Release для PyPI	Літера	Маса	Масштаб

. .					
Розроб.	Гімза Д. П.				
Керівник	Бугиль Б. А.				
Реценз.					Аркуш 3 | Аркушів 4
Н.контр.	Кужій Л. І.				
Затверд.	Шумлянський В.О.				

Розрахунок витрат на куповані вироби

Найменування купованих виробів	Марка, тип	Кількість на розробку, шт.	Ціна за одиницю, грн.	Сума витрат, грн.
Папір, пачок	Формат А4	1	198	198
Роздрук пояснювальної записки	Формат А4	51	1,5	76,5
Зшивання диплому в тверду палітурку		51	180	180
Транспортно-заготівельні витрати (10%)	–	–	–	45,45
Разом	–	–	–	454,5

Кошторис витрат розробки та реалізації програмного продукту.

Найменування елементів витрат	Сума витрат, грн..
Витрати на оплату праці	3165,5
Нарахування на зарплату	696,41
Витрати на куповані вироби	454,5
Накладні витрати	949,65
Інші витрати	531,15
Витрати на налагодження та дослідну експлуатацію	510,00
Всього (К=К1+К2)	6352,66

Розрахунок витрат на оплату праці

Спеціальність розробника	Кількість розробників роб.	Час роботи, год.	Погодинна заробітна плата розробника, грн.	Витрати на оплату праці, грн.
Керівник проєкту	1	14	87,06	1218,84
Консультант з економічної частини	1	1	106,96	106,96
Консультант з охорони праці	1	1	93,70	93,70
Студент-дипломник	1	200	8,73	1 746,00
Всього	4	216	-	3165,5

					Автоматизація пакування Python бібліотек за допомогою GitHub Actions			
					Дані економічного обґрунтування проєктного рішення	Літера	Маса	Масштаб
Зм	*Арк*	*№ докум.*	*Підпис*	*Дата*				

Розроб.	Гімза Д. П.					
Керівник	Бугиль Б. А.					
Реценз.					Аркуш 4	Аркушів 4
Н.контр.	Кужій Л. І.					
Затверд.	Шумлянський В.О.					

Схожість

Джерела з Бібліотеки 719

1	**Студентська робота**	ID файлу: **1004042525**	**Навчальний заклад**: National University of Life and Envir	66 Джерело	4.85%
2	**Студентська робота**	ID файлу: **949845**	**Навчальний заклад**: Yuriy Fedkovych Chernivtsi National Univ	6 Джерело	4.67%
3	**Студентська робота**	ID файлу: **106408**	**Навчальний заклад**: National University of Life and Environm	41 Джерело	4.62%
4	**Студентська робота**	ID файлу: **5968120**	**Навчальний заклад**: Lviv Polytechnic National University		4.58%
5	**Студентська робота**	ID файлу: **2048856**	**Навчальний заклад**: Lviv Polytechnic National University		4.56%
6	**Студентська робота**	ID файлу: **1584576**	**Навчальний заклад**: Yuriy Fedkovych Chernivtsi National University		4.5%
7	**Студентська робота**	ID файлу: **991024**	**Навчальний заклад**: Yuriy Fedkovych Chernivtsi National University		3.71%
8	**Студентська робота**	ID файлу: **1087296**	**Навчальний заклад**: National University of Life and Environm	3 Джерело	2.84%
9	**Студентська робота**	ID файлу: **1086830**	**Навчальний заклад**: National University of Life and Environmental Sc...		1.71%
10	**Студентська робота**	ID файлу: **3377004**	**Навчальний заклад**: Lviv Polytechnic National University	2 Джерело	1.54%
11	**Студентська робота**	ID файлу: **1087228**	**Навчальний заклад**: National University of Life and Environ	10 Джерело	1.13%
12	**Студентська робота**	ID файлу: **1087008**	**Навчальний заклад**: National University of Life and Environmental Sc...		1.12%
13	**Студентська робота**	ID файлу: **1004036162**	**Навчальний заклад**: National University of Life and Environmenta...		1.06%
14	**Студентська робота**	ID файлу: **102262**	**Навчальний заклад**: Lviv Polytechnic National University	55 Джерело	0.62%
15	**Студентська робота**	ID файлу: **1050095**	**Навчальний заклад**: Lviv Polytechnic National University	2 Джерело	0.57%
16	**Студентська робота**	ID файлу: **1050290**	**Навчальний заклад**: Lviv Polytechnic National University	21 Джерело	0.57%
17	**Студентська робота**	ID файлу: **1000680961**	**Навчальний заклад**: Lviv Polytechnic National University	10 Джерело	0.57%
18	**Студентська робота**	ID файлу: **12243694**	**Навчальний заклад**: Lviv Polytechnic National University	13 Джерело	0.54%
19	**Студентська робота**	ID файлу: **1005679378**	**Навчальний заклад**: Lviv Polytechnic National University		0.54%
20	**Студентська робота**	ID файлу: **1086126**	**Навчальний заклад**: Lviv Polytechnic National University	4 Джерело	0.53%

21	Студентська робота	ID файлу: 1052928	Навчальний заклад: Lviv Polytechnic National University	6 Джерело	0.52%
22	Студентська робота	ID файлу: 1015216381	Навчальний заклад: Lviv Polytechnic National University	10 Джерело	0.51%
23	Студентська робота	ID файлу: 1341244	Навчальний заклад: Lviv Polytechnic National University	2 Джерело	0.49%
24	Студентська робота	ID файлу: 1069183	Навчальний заклад: Lviv Polytechnic National University		0.48%
25	Студентська робота	ID файлу: 1057463	Навчальний заклад: Lviv Polytechnic National University		0.48%
26	Студентська робота	ID файлу: 1060311	Навчальний заклад: Lviv Polytechnic National University		0.47%
27	Студентська робота	ID файлу: 1052663	Навчальний заклад: Lviv Polytechnic National University	20 Джерело	0.42%
28	Студентська робота	ID файлу: 1087026	Навчальний заклад: Lviv Polytechnic National University	2 Джерело	0.4%
29	Студентська робота	ID файлу: 1000718505	Навчальний заклад: Lviv Polytechnic National University	5 Джерело	0.4%
30	Студентська робота	ID файлу: 3377973	Навчальний заклад: Lviv Polytechnic National University	4 Джерело	0.38%
31	Студентська робота	ID файлу: 1030613	Навчальний заклад: Lviv Polytechnic National University	6 Джерело	0.37%
32	Студентська робота	ID файлу: 48990	Навчальний заклад: Lviv Polytechnic National University	3 Джерело	0.36%
33	Студентська робота	ID файлу: 1029890	Навчальний заклад: Lviv Polytechnic National University		0.36%
34	Студентська робота	ID файлу: 46339	Навчальний заклад: Lviv Polytechnic National University	2 Джерело	0.35%
35	Студентська робота	ID файлу: 3514463	Навчальний заклад: Lviv Polytechnic National University	21 Джерело	0.33%
36	Студентська робота	ID файлу: 46343	Навчальний заклад: Lviv Polytechnic National University		0.32%
37	Студентська робота	ID файлу: 3442670	Навчальний заклад: Lviv Polytechnic National University		0.32%
38	Студентська робота	ID файлу: 1014737345	Навчальний заклад: Lviv Polytechnic National University	30 Джерело	0.3%
39	Студентська робота	ID файлу: 1011467927	Навчальний заклад: Lviv Polytechnic National University		0.25%
40	Студентська робота	ID файлу: 46429	Навчальний заклад: Lviv Polytechnic National University	2 Джерело	0.25%
41	Студентська робота	ID файлу: 1005722044	Навчальний заклад: Zaporizhzhya National University	4 Джерело	0.24%
42	Студентська робота	ID файлу: 1013105696	Навчальний заклад: Yuriy Fedkovych Chernivtsi Nationa	29 Джерело	0.2%

43	**Студентська робота**	ID файлу: 1009328892	Навчальний заклад: National University of Life and Environmenta...		0.19%
44	**Студентська робота**	ID файлу: 1064711	Навчальний заклад: Lviv Polytechnic National University		0.19%
45	**Студентська робота**	ID файлу: 52802	Навчальний заклад: Lviv Polytechnic National University	30 Джерело	0.19%
46	**Студентська робота**	ID файлу: 1015200586	Навчальний заклад: Lviv Polytechnic National University	18 Джерело	0.19%
47	**Студентська робота**	ID файлу: 1004191128	Навчальний заклад: Lviv Polytechnic National University	2 Джерело	0.19%
48	**Студентська робота**	ID файлу: 49318	Навчальний заклад: Lviv Polytechnic National University		0.18%
49	**Студентська робота**	ID файлу: 46342	Навчальний заклад: Lviv Polytechnic National University		0.18%
50	**Студентська робота**	ID файлу: 1101450	Навчальний заклад: Lviv Polytechnic National University	4 Джерело	0.17%
51	**Студентська робота**	ID файлу: 1070918	Навчальний заклад: Lviv Polytechnic National University		0.17%
52	**Студентська робота**	ID файлу: 1008279032	Навчальний заклад: Lviv Polytechnic National University		0.17%
53	**Студентська робота**	ID файлу: 47558	Навчальний заклад: Lviv Polytechnic National University		0.17%
54	**Студентська робота**	ID файлу: 47034	Навчальний заклад: Lviv Polytechnic National University	12 Джерело	0.17%
55	**Студентська робота**	ID файлу: 50552	Навчальний заклад: Lviv Polytechnic National University	2 Джерело	0.17%
56	**Студентська робота**	ID файлу: 1000022015	Навчальний заклад: National Technical University of Ukr	28 Джерело	0.17%
57	**Студентська робота**	ID файлу: 12192162	Навчальний заклад: Lviv Polytechnic National University	3 Джерело	0.17%
58	**Студентська робота**	ID файлу: 5950856	Навчальний заклад: Lviv Polytechnic National University	3 Джерело	0.17%
59	**Студентська робота**	ID файлу: 1000077430	Навчальний заклад: Lviv Polytechnic National University	2 Джерело	0.16%
60	**Студентська робота**	ID файлу: 1015224955	Навчальний заклад: Lviv Polytechnic National University	14 Джерело	0.16%
61	**Студентська робота**	ID файлу: 51086	Навчальний заклад: Lviv Polytechnic National University	3 Джерело	0.16%
62	**Студентська робота**	ID файлу: 1000102096	Навчальний заклад: Lviv Polytechnic National University	13 Джерело	0.16%
63	**Студентська робота**	ID файлу: 3668707	Навчальний заклад: Lviv Polytechnic National University		0.15%
64	**Студентська робота**	ID файлу: 1015225679	Навчальний заклад: National Technical University of Ukr	30 Джерело	0.12%

65	**Студентська робота**	ID **файлу**: **1005736576**	**Навчальний заклад**: **National Aviation University**	5 Джерело	0.12%
66	**Студентська робота**	ID **файлу**: **1000051665**	**Навчальний заклад**: **National Technical University of Ukr**	24 Джерело	0.11%
67	**Студентська робота**	ID **файлу**: **1003992043**	**Навчальний заклад**: **Lviv Polytechnic National University**	30 Джерело	0.11%
68	**Студентська робота**	ID **файлу**: **1011511400**	**Навчальний заклад**: **Lviv Polytechnic National University**	4 Джерело	0.11%
69	**Студентська робота**	ID **файлу**: **1015193257**	**Навчальний заклад**: **National Technical University of Ukraine "Kyi...**		0.1%
70	**Студентська робота**	ID **файлу**: **1014814382**	**Навчальний заклад**: **National Technical University of Ukraine "Kyi...**		0.1%
71	**Студентська робота**	ID **файлу**: **1015264478**	**Навчальний заклад**: **Lviv Polytechnic National University**	4 Джерело	0.1%
72	**Студентська робота**	ID **файлу**: **3639476**	**Навчальний заклад**: **Lviv Polytechnic National University**		0.1%
73	**Студентська робота**	ID **файлу**: **1003742435**	**Навчальний заклад**: **National Technical University of Ukraine "Kyi...**		0.09%
74	**Студентська робота**	ID **файлу**: **1004081494**	**Навчальний заклад**: **National Technical University of Ukr**	2 Джерело	0.09%
75	**Студентська робота**	ID **файлу**: **1011180272**	**Навчальний заклад**: **National Aviation University**		0.09%
76	**Студентська робота**	ID **файлу**: **1015223434**	**Навчальний заклад**: **National Technical University of Ukr**	2 Джерело	0.09%
77	**Студентська робота**	ID **файлу**: **1015229532**	**Навчальний заклад**: **National Technical University of Ukr**	2 Джерело	0.09%
78	**Студентська робота**	ID **файлу**: **1015226722**	**Навчальний заклад**: **National University Ostroh Academy**	30 Джерело	0.09%
79	**Студентська робота**	ID **файлу**: **1978071**	**Навчальний заклад**: **Lviv Polytechnic National University**		0.08%
80	**Студентська робота**	ID **файлу**: **1002848253**	**Навчальний заклад**: **National University of Life and Envir**	2 Джерело	0.08%
81	**Студентська робота**	ID **файлу**: **109110**	**Навчальний заклад**: **Lviv Polytechnic National University**	30 Джерело	0.08%
82	**Студентська робота**	ID **файлу**: **2054980**	**Навчальний заклад**: **National University of Water Manageme**	4 Джерело	0.08%
83	**Студентська робота**	ID **файлу**: **1005790901**	**Навчальний заклад**: **Poltava National Technical Yuri Kondratyuk U...**		0.08%
84	**Студентська робота**	ID **файлу**: **2084620**	**Навчальний заклад**: **National University of Water Manageme**	2 Джерело	0.08%
85	**Студентська робота**	ID **файлу**: **1012978945**	**Навчальний заклад**: **Zaporizhzhya National University**		0.08%
86	**Студентська робота**	ID **файлу**: **3378814**	**Навчальний заклад**: **Lviv Polytechnic National University**		0.08%

87	**Студентська робота**	ID **файлу**: **1004032374**	**Навчальний заклад**: **National University of Water Management an...**	**0.08%**
88	**Студентська робота**	ID **файлу**: **1004033220**	**Навчальний заклад**: **Lviv Polytechnic National University**	**0.08%**
89	**Студентська робота**	ID **файлу**: **1003749668**	**Навчальний заклад**: **National University of Life and Environmenta...**	**0.08%**