НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут Прикладного Системного Аналізу
Кафедра Системного Проектування

До захисту допущено:
Завідувач кафедри
Вадим МУХІН
«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Інтелектуальні сервіс-орієнтовані розподілені обчислення»
спеціальності Комп'ютерні науки на тему:
«Мікросервісна архітектура системи перевірки студентських робіт з програмування»

Виконав:
студент IV курсу, групи ДА-91
Сєров Іван Сергійович
_____

Керівник:
доцент, к.т.н.
Булах Богдан Вікторович
_____

Рецензент:
Зав. каф. ММСА,
к.т.н., доц. Тимощук Оксана Леонідівна
_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра системного проектування

Рівень вищої освіти: перший (бакалаврський)

Спеціальність: 122 «Комп'ютерні науки»

Освітньо-професійна програма:

«Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Вадим МУХІН

«___»_____2023 р.

ЗАВДАННЯ
на дипломну роботу студенту
Сєрова Івана Сергійовича

1. Тема роботи «Мікросервісна архітектура системи перевірки студентських робіт з програмування», керівник роботи Булах Богдан Вікторович, доцент, затверджені наказом по університету від «___»_____ 20__ р. №_____
2. Термін подання студентом роботи – «___»_____ 20__ р.
3. Вихідні дані до роботи
4. Зміст роботи:
    1. Вступ.
    2. Існуючі системи автоматизації перевірки навчальних завдань з програмування
    3. Загальна відомість про архітектурні підходи

4. Реалізація системи перевірки студентських робіт з програмування

5. Оцінка ефективності

6. Приклади використання

7. Висновки

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Презентація до захисту роботи.

6. Дата видачі завдання «___»_____ 20__ р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	02.12.2022	
2	Пошук опису існуючих систем автоматизації перевірки навчальних завдань з програмування	01.03.2023	
3	Формулювання функціональних вимог до системи	15.03.2023	
4	Підбір методології та інструментарію розробки	08.04.2023	
5	Опис основних сутностей реалізації	19.04.2023	
6	Реалізація системи перевірки студентських робіт з програмування	30.04.2023	
7	Порівняння та підведення підсумків	05.05.2023	

Студент

Сєров І. С.

Керівник

Булах Б. В.

АНОТАЦІЯ

до дипломної роботи Сєрова Івана Сергійовича на тему
«Мікросервісна архітектура системи перевірки студентських робіт з програмування»

Структура дипломної роботи: Загальний обсяг пояснювальної записки: 87 сторінки, 9 рисунків, 6 таблиць, 27 посилань.

Проблема автоматизованої перевірки студентських робіт дуже актуальна тема. Все більше людей починає вчитися програмуванню і є необхідність задовольнити потреби користувачів для перевірки таких робіт. Мікросервіси дають змогу розділити функціонал додатку і забезпечити безперебійну роботу його складових. Крім того, це дозволяє продвинутим користувачам додати необхідний функціонал власноруч, використовуючі ті інструменти, якими вони володіють. В ході роботи було розроблено додаток для перевірки студентських робіт з програмування на мові програмування Python, а також розглянуто архітектурні підходи до його створення і порівняння існуючих рішень з даної проблеми.

Метою роботи є розробка застосунку для перевірки студентських робіт з програмування.

Предмет дослідження – перевірка студентських робіт з програмування

Об’єкт дослідження – система перевірки студентських робіт з програмування.

Ключові слова: “Мікросервісна архітектура”, “Python”, “API”, “мікросервіси”

ANNOTATION

bachelor's thesis of Sierov Ivan Serhiyovych on “Microservice architecture of the system for evaluation of student assignments on programming course”

Structure of the thesis: Total volume of the explanatory note: 87 pages, 9 figures, 6 tables, 27 references.

The problem of automated verification of student work is a very relevant topic. More and more people are starting to learn programming and there is a need to meet the needs of users to check such works. Microservices allow you to divide the functionality of an application and ensure the smooth operation of its components. In addition, it allows advanced users to add the necessary functionality on their own, using the tools they have. In the course of this work, we developed an application for checking student assignments in Python programming language, as well as considered architectural approaches to its creation and compared existing solutions to this problem.

The purpose of the work is to develop an application for checking student assignments.

Subject of research – checking student assignments.

The object of research is a system for checking student assignments.

Keywords: “Microservice architecture”, “Python”, “API”, “microservices”

ЗМІСТ

ВСТУП

Автоматизація процесу перевірки студентських робіт з програмування – спосіб вдосконалити систему освіти. Використання таких інструментів у навчальному закладі також свідчить про те, що він є сучасним і використовує найкращі практики.

Наразі багато викладачів, які викладають програмування додатків, витрачають багато часу на перевірку студентських робіт. Тому як ніколи необхідно автоматизувати цей процес. Це економить час як студентам, так і викладачам, зменшує кількість помилок та покращує навчання.

Метою даного проекту є створення додатку з мікросервісною архітектурою для автоматизації процесу перевірки студентських робіт з програмування. Архітектура мікросервісів дозволяє розподіляти сервіси між декількома комп'ютерами (серверами). Це дозволяє балансувати навантаження. Такі сервіси, як перевірка на плагіат або перевірка компіляції коду, вимагають багато ресурсів, але за рахунок вищезазначених факторів можна досягти значного приросту продуктивності.

Практична цінність розробки такої системи полягає в тому, що вона може бути використана для кафедральних та університетських цілей, а також у тому, що всі користувачі мають доступ до робочого простору.

1 ОГЛЯД ІСНУЮЧИХ ОСВІТНІХ ПЛАТФОРМ

Сучасний світ вимагає від програмістів не тільки глибоких знань та навичок у сфері програмування, але й вміння швидко та точно вирішувати завдання. У навчальних закладах, де викладається програмування, ефективна та надійна оцінка студентських завдань є важливим аспектом навчального процесу. Однак, з ростом кількості студентів та завдань, викладачам стає складно проводити детальну та швидку перевірку робіт кожного студента.

1.1 Moodle

Наразі є ряд рішень, які широко використовують в Україні і світі. Деякі з них напряму пов’язані із університетською структурою, а деякі ж мають часткову або повністю відсутню інтеграцію з нею. Одним з найвідоміших платформ для управління навчанням є Moodle.

Moodle (Modular Object-Oriented Dynamic Learning Environment) - це відкрита система управління навчанням (Learning Management System), розроблена для підтримки навчального процесу та електронного навчання. Moodle надає набір інструментів та функціональності для створення, управління та взаємодії з навчальними курсами та студентами.

Мета використання системи Moodle полягає в полегшенні організації навчального процесу та забезпеченні зручного доступу до навчальних матеріалів, завдань, тестів та спілкування між учасниками курсу. Вона дозволяє викладачам створювати інтерактивні курси, завдання та тестові завдання, вести відстеження активності студентів, а також спілкуватися з ними через форуми та повідомлення.

Одним з важливих аспектів Moodle є можливість автоматизації перевірки навчальних завдань з програмування. Кодранер (CodeRunner) є одним з популярних плагінів для системи Moodle, який забезпечує

автоматизовану перевірку навчальних завдань з програмування. Цей плагін дозволяє викладачам створювати завдання, які вимагають написання програмного коду, та автоматично перевіряти цей код на правильність.

Основна функціональність Кодранера включає наступні можливості:

- Підтримка різних мов програмування: Кодранер підтримує широкий спектр мов програмування, таких як Java, Python, C++, PHP тощо. Це дає викладачам можливість створювати завдання з програмування у відповідності до вимог конкретного курсу чи програми навчання.
- Автоматична перевірка коду: Кодранер забезпечує автоматичну перевірку написаного студентом коду на основі певних тестів чи вхідних даних. Викладач може визначити набір тестових випадків, на яких буде проводитися перевірка, і встановити правила оцінювання для кожного завдання.
- Зворотний зв'язок для студентів: Після перевірки коду, Кодранер може надати студентам детальний зворотний зв'язок, включаючи інформацію про помилки, результати виконання тестових випадків та загальний бали. Це допомагає студентам зрозуміти їхні помилки та покращити свої програмувальні навички.
- Розширена конфігурація завдань: Кодранер надає викладачам гнучкі налаштування для кожного завдання, включаючи обмеження часу виконання, обмеження розміру вхідних даних, встановлення власних тестових випадків та оцінювання за складністю завдання. [1]

1.2 Gradescope

Gradescope - це платформа для оцінювання та зворотного зв'язку з оцінювання для навчальних закладів. Вона спеціалізується на автоматизованому оцінюванні різних видів завдань, включаючи програмування, математичні задачі, есе та інші. Gradescope надає

інструменти для зручного та ефективного проведення оцінювання, спрощує процес збору та перевірки студентських робіт та надає докладний зворотний зв'язок для студентів.

Основні особливості та функціонал Gradescope включають:

- Завантаження та управління роботами студентів: Викладачі можуть завантажувати студентські роботи на платформу, створювати завдання та курси, та легко управляти потоком робіт.
- Автоматична перевірка та оцінювання: Gradescope дозволяє налаштовувати правила перевірки для різних типів завдань. Наприклад, для завдань з програмування ви можете встановити автоматичну перевірку коду, яка оцінює правильність роботи за допомогою тестів чи інших критеріїв.
- Зручність перегляду та оцінювання: Викладачі можуть легко переглядати та оцінювати студентські роботи, використовуючи вбудований онлайн-редактор, який дозволяє позначати та коментувати код, математичні формули та інші елементи завдань.
- Зворотний зв'язок для студентів: Gradescope надає можливість надати докладний зворотний зв'язок для студентів, включаючи оцінки, коментарі та розбалікування роботи.
- Аналітика та статистика: Платформа надає викладачам інструменти для аналізу даних оцінювання, статистики та звітності, що допомагають зрозуміти результати та вдосконалити процес оцінювання.

З використанням Gradescope викладачі можуть ефективно оцінювати та надавати зворотний зв'язок студентам з програмування та інших видів завдань. Це полегшує процес оцінювання, сприяє об'єктивності та швидкості зворотного зв'язку, а також покращує якість навчання.

Gradescope є популярною платформою, яку використовують багато університетів та вищих навчальних закладів по всьому світу, зокрема:

- Стенфордський університет (Stanford University) – Gradescope використовується для оцінювання та зворотного зв'язку в багатьох курсах, включаючи ті, що пов'язані з програмуванням та інформатикою. [2]
- Массачусетський технологічний інститут (Massachusetts Institute of Technology, MIT) – Gradescope використовується в різних дисциплінах, включаючи науку про комп'ютери та інженерію. [3]
- Гарвардський університет (Harvard University) – Gradescope використовується для оцінювання та зворотного зв'язку в багатьох курсах, включаючи програмування, математику та науку про дані. [4]
- Карнегі-Меллонський університет (Carnegie Mellon University) – Gradescope використовується для оцінювання завдань з програмування та інших курсів, пов'язаних з комп'ютерними науками. [5]
- Йельський університет (Yale University) – Gradescope використовується для оцінювання та зворотного зв'язку в різних курсах, включаючи науку про комп'ютери, математику та гуманітарні науки. [6]

Багато інших навчальних закладів також активно використовують цю платформу для полегшення процесу оцінювання та забезпечення зворотного зв'язку для своїх студентів. [7]

1.3 Coursera

Окрім інтегрованих в університетську інфраструктуру існують платформи більш незалежні. Одна з них – Курсера (Coursera). Курсера – це платформа для онлайн-навчання, яка співпрацює з різними університетами та організаціями по всьому світу. Вона надає доступ до широкого спектру курсів з різних галузей знань, включаючи програмування. Сама платформа має вбудований інструмент для роботи з кодом і відповідно автоматичної перевірки його. Особливості та можливості дійсно унікальні, зокрема:

- Різноманітність курсів: Курсера має багатий вибір курсів від різних університетів та провідних спеціалістів у галузях. Крім того, провідні ІТ компанії також публікують свої курси там, що сприяє розвитку впізнаваності бренду і власне створення бази кваліфікованих фахівців з необхідних і поширених технологій.
- Гнучкість навчання: Курси на платформі Курсера можна проходити у власному темпі, відповідно до власного розкладу. Більшість курсів доступні в асинхронному форматі, що дозволяє студентам вивчати матеріал в зручний для них час.
- Відеолекції та матеріали: Курсера надає доступ до відеолекцій, матеріалів для самостійного опрацювання, домашніх завдань та інших ресурсів, що допомагають усвідомити та закріпити навчальний матеріал. А також інтегровано систему перекладу (автоматичного) на більшість мов, що сприяє більшому охопленню аудиторії з усіх куточків світу.
- Оцінювання та сертифікація: Після успішного завершення курсу студентам пропонується оцінювання, яке може включати виконання завдань, тестів або проектів. Також студенти можуть отримати сертифікат про успішне завершення курсу за певну плату або на базі гранту (Курсера відкрила доступ до українців до більшості курсів, а також допомагала студентам і унверситетам по всьому світу за часів пандемії).
- Спільнота та підтримка: На Курсері можна спілкуватися з іншими студентами та викладачами через форуми, дискусійні групи та засоби спілкування, що сприяє обміну думками, підтримці та колективному навчанню.

Курси, пройдені студентами, на курсері можна інтерпретувати в певну кількість кредитів, залежно від програми університету і курсу. Це полегшує процес навчання і для викладачів і для студентів, що позитивно впливає на якість знань здобувачів освіти.

Більше того, є окреме рішення Курсера для Кампусів (Coursera for Campus), який допомагає закладам освіти створювати окремі каталоги курсів, які найбільш відповідні для тої чи іншої освітньої програми, а співробітники і студенти можуть отримати доступ до LMS свого закладу.

Курсера для Кампусів також надає навчальним закладам інструменти для відстеження прогресу та залучення слухачів, а також можливість видавати сертифікати про проходження курсів. Крім того, навчальні заклади можуть використовувати контент Coursera для доповнення власних курсів і програм, надаючи студентам більш комплексну освіту. [8]

1.4 HackerRank

HackerRank - це сучасна платформа для студентів, щоб практикувати свої навички кодування та готуватися до технічних співбесід. HackerRank for School - це версія платформи, спеціально розроблена для навчальних закладів, що дозволяє вчителям створювати завдання з кодування та відстежувати прогрес своїх учнів. Ось деякі переваги використання HackerRank for School:

- Налаштовувані завдання з кодування: викладачі можуть створювати завдання з кодування відповідно до своєї навчальної програми, гарантуючи, що учні практикують відповідні навички.
- Відстеження прогресу: можна відстежувати прогрес своїх учнів та визначати сфери, де їм може знадобитися додаткова підтримка чи ресурси.
- Співпраця: студенти можуть працювати разом над завданнями з кодування, створюючи середовище для спільного навчання та заохочуючи одногрупникове навчання.
- Вирішення реальних проблем: HackerRank for School знайомить з реальними проблемами кодування, допомагаючи розвивати

критичне мислення та навички вирішення проблем, необхідні для успішної кар'єри в галузі програмної інженерії.

- Безперервне навчання: платформа пропонує широкий спектр завдань з кодування, що дозволяє постійно вдосконалювати свої навички та бути в курсі новітніх мов і технологій програмування.
- Управління часом: студенти можуть практикувати роботу в умовах дефіциту часу, що є важливою навичкою для інженерів-програмістів, які повинні дотримуватися дедлайнів і вчасно надавати високоякісне програмне забезпечення.
- Забезпечення якості: працюючи над проблемами кодування та отримуючи відгуки про свої рішення, студенти можуть розвинути глибоке розуміння забезпечення якості та важливості ретельного тестування та налагодження коду.

Таким чином, HackerRank for School є чудовим ресурсом як для вчителів, так і для учнів, надаючи платформу для навчання, практики та вдосконалення навичок кодування в атмосфері співпраці та залучення. [9]

1.5 Платформа «Сікорський»

Платформа «Сікорський» – це відкрите віртуальне навчальне середовище КПІ ім. Ігоря Сікорського, яке надає адміністраторам, викладачам та студентам широкий спектр можливостей для застосування сучасних технологій дистанційного навчання, розробки веб-ресурсів навчальних дисциплін, організації інтерактивної взаємодії викладачів і студентів, управління процесом дистанційного навчання.

Платформа базується на спеціалізованих веб-середовищах, таких як Moodle та Google Workspace for Education (пакет хмарних додатків від Google для побудови інформаційно-освітньої структури навчального закладу), а

також інших програмних продуктах, призначених для дистанційного навчання.

Платформа дистанційного навчання КПІ ім. Ігоря Сікорського пропонує різноманітні освітні ресурси, які поділені на п'ять груп:

- Дистанційні курси, розроблені в Moodle або Google Workspace for Education для здобувачів вищої освіти відповідно до їхніх навчальних планів під керівництвом і контролем викладачів. Доступ до цих курсів надається після реєстрації кожного учасника дистанційного навчання.
- Дистанційні курси, розроблені в Moodle або Google Workspace for Education для слухачів курсів підвищення кваліфікації. Доступ до цих курсів надається після реєстрації кожного учасника.
- Дистанційні курси з вільним доступом, які кожен може пройти без реєстрації та без можливості тестування.
- Дистанційні курси, які кожен може пройти після реєстрації. Вони мають можливість пройти тестування та отримати Сертифікати, які можуть бути зараховані в КПІ ім. Ігоря Сікорського відповідно до вимог "Тимчасового положення про порядок визнання результатів навчання, здобутих здобувачами вищої освіти КПІ ім. Ігоря Сікорського за програмами неформальної/інформальної освіти" (тестування та отримання Сертифікату є платною послугою).
- Відеокурси у вільному доступі.

Загалом платформа забезпечує гнучкий і зручний спосіб доступу студентів до освітніх ресурсів та взаємодії з викладачами й іншими студентами у віртуальному навчальному середовищі. [10]

1.6 Висновок до розділу

Наразі є багато готових систем автоматизації перевірки навчальних завдань з програмування і кожна з таких систем має свої переваги та недоліки.

Головна проблема таких систем це їх функціонал. Вони мають різний функціонал і щоб покрити всі необхідні вимоги треба комбінувати їх, що може викликати труднощі через їх низьку сумісність між собою. Також такі системи більш зосереджені на універсальності, тож функціонал саме для перевірки завдань з програмування потребує деякого покращення.

Існує інша проблема – перевірка робіт на плагіат. Таких перевірок немає інтегрованих у вже існуючі системи, тому є необхідність в розробці застосунку, який включає в себе усі перелічені пункти і відповідає актуальним потребам для перевірок робіт.

2 АРХІТЕКТУРНІ ПІДХОДИ ДО СТВОРЕННЯ ПРОГРАМНИХ СИСТЕМ

Архітектурні підходи в розробці програмного забезпечення відносяться до фундаментальних принципів, патернів і стратегій, що використовуються для проектування і структурування загальної системи та її компонентів. Ці підходи забезпечують високорівневу структуру для організації та з'єднання різних частин програмного забезпечення, визначення їх взаємодії та керування процесом розробки.

Архітектурні підходи вирішують такі проблеми, як масштабованість, продуктивність, підтримуваність, модульність, перевикористання та гнучкість. Вони визначають структуру програмної системи, потік даних, механізми зв'язку та моделі розгортання, а також надають рекомендації щодо їхнього проектування. Ці підходи допомагають гарантувати, що програмне забезпечення спроектоване таким чином, що відповідає бажаним функціональним і нефункціональним вимогам, а також зможе ефективно будуватися, розгортатися і підтримуватися.

Архітектурні підходи діють як шаблони або фреймворки, які розробники та архітектори використовують для прийняття обґрунтованих рішень щодо структури та організації системи. Вони забезпечують спільну мову та набір найкращих практик, які допомагають командам ефективно спілкуватися та співпрацювати під час процесу розробки. Дотримуючись архітектурних підходів, програмні системи можна розробляти та впроваджувати таким чином, щоб вони відповідали стандартам та оптимізували використання ресурсів, а також забезпечували майбутнє зростання та масштабованість.

2.1 Мікросервісна архітектура

Мікросервісна архітектура – це архітектурний підхід, який структурує додаток як набір невеликих, незалежних і мало пов'язаних між собою сервісів. Кожен сервіс фокусується на конкретній бизнес-функції і може бути розроблений, розгорнутий і масштабований незалежно. До особливостей такої архітектури можна віднести:

- Незалежність сервісу: Мікросервіси – це автономні одиниці, які інкапсулюють певну бізнес-функцію. Кожен сервіс працює незалежно, з власним кодом, даними та розгортанням. Така незалежність дозволяє розробляти, тестувати, розгортати та масштабувати сервіси автономно.
- Децентралізоване управління даними: В архітектурі мікросервісів кожен сервіс має власну базу даних або сховище даних. Такий підхід до децентралізованого управління даними допомагає забезпечити автономність сервісів і уникнути сполучень даних між сервісами. Однак він також створює проблеми, пов'язані з узгодженістю та синхронізацією даних, які необхідно вирішувати за допомогою ретельного проектування та шаблонів взаємодії.
- Комунікація та сумісність: Мікросервіси взаємодіють один з одним за допомогою простих механізмів, таких як HTTP API, протоколи повідомлень або архітектури, керованої подіями. Сервіси можуть взаємодіяти синхронно або асинхронно, залежно від вимог. Чітко визначені інтерфейси забезпечують вільний зв'язок і дозволяють сервісам розроблятися незалежно, не впливаючи на всю систему.
- Масштабованість та стійкість: Мікросервіси пропонують переваги масштабованості та відмовостійкості. Сервіси можуть бути незалежно масштабовані на основі попиту, що дозволяє ефективно використовувати ресурси. Якщо один сервіс виходить з ладу або зазнає високого навантаження, це не впливає на всю систему. Крім того, ізоляція

збоїв і стійкість до збоїв можуть бути досягнуті шляхом впровадження відповідних механізмів обробки помилок і резервного копіювання.

- Гнучкість розгортання: Мікросервіси можна розгортати за допомогою різних моделей розгортання, включаючи контейнеризацію (наприклад, Docker), безсерверні обчислювальні платформи або традиційні віртуальні машини. Така гнучкість дозволяє організаціям обирати модель розгортання, яка найкраще відповідає вимогам, інфраструктурі та операційним уподобанням.
- Управління та моніторинг: При наявності декількох сервісів в архітектурі мікросервісів, ефективне управління та моніторинг стають критично важливими. Інструменти та фреймворки для відстеження сервісів, балансування навантаження, централізованого ведення логів та розподіленого трейсингу можуть допомогти відстежувати та керувати станом, продуктивністю та доступністю системи. [11]

2.1.1 Приклади успішного впровадження мікросервісної архітектури в сучасній розробці програмного забезпечення

Одним з прикладів успішного впровадження мікросервісної архітектури є компанія Netflix. Вони перейшли на мікросервіси з 2009 року і досягли значних успіхів у використанні цього підходу. [12]

Іншим прикладом є компанія Amazon. Вони використовують мікросервісну архітектуру для свого хмарного сервісу Amazon Web Services (AWS). Вони розробили платформу, що базується на мікросервісах, яка надає різноманітні хмарні послуги для клієнтів з усього світу. [13]

Компанія Uber, провайдер послуг таксі та каршерингу, використовує мікросервіси для своєї глобальної платформи. Вони поділили свою архітектуру на набір незалежних мікросервісів, що дозволяє їм гнучко

масштабувати окремі компоненти та забезпечувати високу доступність своїх сервісів. [14]

Сервіс потокового музики Spotify також базується на мікросервісній архітектурі. Вони розділили свою систему на багато окремих сервісів, що дозволяє їм швидко впроваджувати нові функції та забезпечувати гнучке масштабування. [15]

Airbnb, платформа для бронювання помешкань, також використовує мікросервісну архітектуру для своєї інфраструктури. Вони розбили свою систему на набір окремих сервісів, що дозволяє їм зручно масштабувати та підтримувати різні функціональні компоненти. [16]

Twitter також перейшов на мікросервісну архітектуру для своєї платформи мікроблогінгу. Вони поділили свою систему на набір сервісів, що дозволяє їм зберігати великий обсяг даних та забезпечувати високу доступність своїх послуг. [17]

2.2 Монолітна архітектура

Монолітна архітектура - це архітектурний підхід, при якому весь додаток будується як єдиний, самодостатній блок. При такому підході всі компоненти і функції програми тісно пов'язані між собою і виконуються в рамках одного процесу або середовища виконання. Ось деякі ключові характеристики монолітної архітектури:

- Єдиний код: У монолітній архітектурі весь додаток розробляється і підтримується в рамках єдиного коду. Всі компоненти, такі як інтерфейс користувача, бізнес-логіка та доступ до даних, взаємозалежні та тісно пов'язані між собою.
- Тісний взаємозв'язок: Компоненти в моноліті тісно пов'язані між собою, а це означає, що зміни в одній частині системи можуть мати

каскадний вплив на інші частини. Модифікація або додавання нових функцій часто вимагає повторного тестування та перерозгортання всієї програми.

- Єдина одиниця розгортання: Монолітний додаток зазвичай розгортається як єдине ціле. Всі компоненти та залежності пакуються разом і розгортаються в одній інфраструктурі, що спрощує розгортання та створення версій порівняно з розподіленими системами.
- Спільна пам'ять і ресурси: У моноліті компоненти зазвичай мають спільний простір пам'яті та ресурси. Вони мають прямий доступ до спільних бібліотек, баз даних та інших ресурсів програми.
- Продуктивність: Монолітні архітектури часто пропонують хорошу продуктивність, оскільки всі компоненти працюють в рамках одного процесу і можуть ефективно взаємодіяти один з одним. Виклики функцій у пам'яті можуть бути швидшими порівняно з міжсервісним зв'язком у розподілених системах.
- Простота розробки: На початковому етапі монолітні архітектури легше розробляти та налагоджувати, оскільки вся система міститься в єдиній кодовій базі. Це вимагає менше управління інфраструктурою та конфігурацією порівняно з розподіленими системами.

Однак монолітні архітектури мають певні обмеження та проблеми, особливо в міру того, як додатки зростають у розмірі та складності:

- Масштабованість: Масштабування монолітного додатку може бути складним завданням. Оскільки весь додаток розгортається як єдине ціле, масштабування окремих компонентів незалежно неможливо. Це може обмежити масштабованість і використання ресурсів.
- Обслуговування та розширюваність: Великі монолітні додатки з часом можуть стати складними в обслуговуванні та розширенні. Додавання нових функцій або внесення змін вимагає модифікації всієї кодової бази, що збільшує ризик появи помилок і впливу на інші частини системи.

- Складність розгортання: Розгортання оновлень або виправлень у монолітному додатку часто вимагає розгортання всієї системи. Це може призвести до довшого часу розгортання, збільшення часу простою та підвищеного ризику помилок.
- Технологічна неоднорідність: у монолітній архітектурі всі компоненти повинні використовувати один і той самий стек технологій та середовище розробки. Це обмежує можливість впровадження нових технологій або мов програмування для певних частин системи. [18]

2.3 Сервіс-орієнтована архітектура

SOA (англ. Service-Oriented Architecture - сервіс-орієнтована архітектура) – це архітектурний підхід, який наголошує на створенні слабко пов'язаних і багаторазово використовуваних сервісів для побудови програмних систем. У SOA додаток декомпозується на набір сервісів, які взаємодіють один з одним через чітко визначені інтерфейси. Ось деякі ключові аспекти SOA:

- Орієнтація на сервіси: SOA обертається навколо концепції сервісів, які є самодостатніми одиницями функціональності програмного забезпечення. Сервіси розроблені так, щоб бути незалежними і автономними, надаючи конкретні можливості або бізнес-функції. До них можуть отримати доступ інші сервіси або додатки через стандартизовані інтерфейси, як правило, з використанням таких технологій, як SOAP (Simple Object Access Protocol) або REST (Representational State Transfer) через HTTP.
- Слабкий зв'язок: У SOA сервіси розроблені так, щоб бути вільно пов'язаними, тобто вони є незалежними і мають мінімальну залежність один від одного. Сервіси взаємодіють через контракти, визначені їхніми інтерфейсами, що дозволяє їм розвиватися незалежно, не впливаючи

на інші сервіси. Вільний зв'язок сприяє гнучкості, масштабованості та легшому обслуговуванню.

- Можливість багаторазового використання: Однією з ключових переваг SOA є орієнтація на повторне використання сервісів. Сервіси призначені для надання конкретних можливостей, які можуть бути використані кількома додатками або іншими сервісами. Це сприяє модульному дизайну і зменшує надмірність, оскільки сервіси можуть використовуватися в різних частинах системи або спільно використовуватися кількома додатками.
- Сумісність: SOA сприяє взаємодії між різними системами і технологіями. Використовуючи стандартизовані протоколи зв'язку і формати даних, сервіси можуть взаємодіяти один з одним незалежно від базових платформ або технологій. Це уможливлює інтеграцію та співпрацю між розрізненими системами як всередині організації, так і за її межами.
- Композиція сервісів: SOA дозволяє компонувати та організовувати послуги для створення більш складних бізнес-процесів. Сервіси можуть бути об'єднані для формування процесів вищого рівня або складених додатків, які представляють комплексні бізнес-процеси. Це дозволяє створювати гнучкі та адаптивні системи, які можна легко модифікувати або розширювати в міру зміни вимог.
- Управління та життєвий цикл сервісів: SOA зазвичай включає механізми управління та чітко визначений процес управління життєвим циклом послуги. Управління забезпечує дотримання архітектурних стандартів, політик і найкращих практик. Управління життєвим циклом послуг охоплює виявлення, проектування, розробку, розгортання, версіювання та виведення з експлуатації послуг.
- Масштабованість та продуктивність: SOA може забезпечити масштабованість, дозволяючи сервісам незалежно масштабуватися на основі попиту. Сервіси можуть бути розгорнуті на різних серверах або розподілені

між декількома вузлами, щоб впоратися зі збільшеним робочим навантаженням. Крім того, використання таких технологій, як черги повідомлень або подієво керовані архітектури, може допомогти підвищити продуктивність і забезпечити асинхронний зв'язок між сервісами. [11]

2.4 Подієво-орієнтована архітектура

EDA (Event-Driven Architecture - подієво-орієнтована архітектура) – це архітектурний підхід, який фокусується на створенні, виявленні та обробці подій у програмній системі. В EDA компоненти системи взаємодіють один з одним, створюючи та обробляючи події, що дозволяє їм реагувати на зміни або тригери. Ось деякі ключові аспекти архітектури, керованої подіями:

- Події: Події представляють собою певні події або зміни в системі або зовнішньому середовищі. Вони можуть генеруватися діями користувача, системними подіями або зверненнями від інших систем. Події, як правило, представлені у вигляді структурованих даних і несуть інформацію про подію, таку як тип події, мітка часу та відповідне навантаження даних.
- Виробники подій: Компоненти або сервіси в системі діють як виробники подій, генеруючи події, коли виникають певні умови або дії. Ці виробники публікують події в центральну шину подій або брокер повідомлень, роблячи їх доступними для обробки іншими компонентами.
- Споживачі подій: Споживачі подій підписуються на певні типи подій або шаблони подій, які їх цікавлять. Коли подія, що їх цікавить, публікується на шині подій, споживачі отримують цю подію і можуть вжити відповідних заходів або запустити подальші процеси на основі даних про подію.
- Слабкий зв'язок: EDA сприяє вільному зв'язку між компонентами, відокремлюючи їх від прямих залежностей і дозволяючи їм

взаємодіяти посередництвом подій. Компоненти повинні знати лише типи подій та пов'язані з ними дані, щоб взаємодіяти один з одним, що забезпечує гнучкість та незалежність у проектуванні системи.

- Асинхронний зв'язок: Системи, керовані подіями, зазвичай використовують асинхронні механізми зв'язку, де виробники та споживачі подій не повинні бути доступними або активними одночасно. Це забезпечує масштабованість, відмовостійкість і швидкість реагування, оскільки компоненти можуть обробляти події незалежно, не чекаючи негайної реакції.
- Джерела подій та журнали подій: У деяких випадках системи, керовані подіями, використовують джерела подій, що означає зберігання подій як першоджерело істини про стан системи. Журнали подій фіксують всі події, що відбуваються в системі, і можуть бути використані для аудиту, відтворення подій або відновлення стану системи в будь-який момент часу.
- Масштабованість та відмовостійкість: EDA за своєю суттю підтримує масштабованість і відмовостійкість, роз'єднуючи компоненти і дозволяючи їм масштабуватися незалежно. Системи, керовані подіями, можуть обробляти різні робочі навантаження, розподіляючи обробку подій між декількома екземплярами або вузлами, забезпечуючи ефективне використання ресурсів і відмовостійкість. [11]

2.5 Безсерверна архітектура

Безсерверна архітектура, також відома як Function as a Service (FaaS), – це архітектурний підхід, при якому розробники зосереджуються на написанні та розгортанні окремих функцій або невеликих блоків коду, які виконуються у відповідь на певні події або тригери. У безсерверній моделі хмарний провайдер керує базовою інфраструктурою, автоматично масштабуючи функції за потреби. Ось деякі ключові аспекти безсерверної архітектури:

- Виконання функцій: У безсерверній архітектурі основна увага приділяється розробці та розгортанню функцій, які виконують конкретні завдання або операції. Ці функції, як правило, короткочасні і не мають стану, тобто вони не підтримують жодного постійного стану між викликами. Функції виконуються в безсерверному середовищі, що надається хмарними платформами, такими як AWS Lambda, Microsoft Azure Functions або Google Cloud Functions.
- Подієво-керована модель: Безсерверні функції запускаються певними подіями або тригерами, такими як HTTP-запити, зміни в базі даних, завантаження файлів або заплановані таймери. Коли відбувається подія, пов'язана з нею функція виконується автоматично. Ця модель, керована подіями, дозволяє асинхронну обробку на основі подій, що дає змогу системам реагувати та відповідати на події в реальному часі або дії користувача.
- Автоматичне масштабування: Безсерверні архітектури забезпечують автоматичне масштабування та управління ресурсами. Хмарний провайдер управляє розподілом ресурсів на основі вхідного робочого навантаження. Функції масштабуються горизонтально, тобто додаткові екземпляри функції автоматично створюються і управляються в міру зростання попиту. Таке еластичне масштабування дозволяє ефективно використовувати ресурси та усуває потребу в ручному налаштуванні масштабування.
- Спрощення операцій: Безсерверні архітектури абстрагуються від управління базовою інфраструктурою, включаючи забезпечення, конфігурацію та масштабування серверів. Розробники можуть зосередитись на написанні логіки функцій, не турбуючись про проблеми інфраструктури. Таке спрощення операцій дозволяє командам швидше виконувати ітерації та розгортати код, скорочуючи час виходу на ринок та операційні витрати.

- Сторонні сервіси та інтеграції: Безсерверні архітектури часто використовують різні сторонні сервіси та інтеграції для розширення функціональності функцій. Ці сервіси можуть включати бази даних, черги, сервіси зберігання, служби автентифікації тощо. Інтеграція з цими сервісами дозволяє розробникам використовувати існуючі можливості і зосередитися на основній бізнес-логіці.
- Операції з зовнішніми сервісами у стані стану: Хоча безсерверні функції самі по собі не мають стану, вони можуть взаємодіяти із зовнішніми сервісами, такими як бази даних або кеші, для зберігання та отримання даних. Використовуючи зовнішні сервіси, безсерверні функції можуть отримувати доступ до постійних даних або зберігати інформацію про сеанси під час викликів. [11]

2.6 Висновок до розділу

Для системи перевірки студентських робіт з програмування найкраще підходить мікросервісна архітектура.

Основними перевагами можна виділити безперебійну роботу у випадку непрацездатності одного чи кількох сервісів, крім того завдяки своїй розподіленій структурі така система легка до покращення і індивідуального використання (користувач може розробити свої модулі відповідно під специфічні вимоги).

Також така система буде зручна в розгортанні як локально так і у хмарі, що також дозволяє використовувати її крос-платформенно і легко інтегрувати у вже існуючі системи.

3 ВИМОГИ ДО СИСТЕМИ ТА ПІДБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

Розробка додатку для перевірки студентських робіт з програмування актуальне завдання для сьогоденних вимог. Такий додаток має вміщувати усі необхідні функції для викладачів, щоб мінімізувати час для перевірки робіт, а також з чітко описаними принципами роботи, щоб користувачі мали змогу додати існуючий функціонал за власними потребами.

3.1 Формулювання функціональних вимог до системи

Формулювання вимог будемо створювати в форматі User Story.

User Story 1. Як користувач, я хочу мати можливість завантажувати роботи студентів з Git, щоб опрацьовувати їх іншими сервісами.

User Story 2. Як користувач, я хочу мати можливість перевірки робіт студентів на плагіат, щоб виявляти схожіть робіт між собою.

User Story 3. Як користувач, я хочу мати можливість перевіряти стиль оформлення робіт, щоб надавати поради з покращення коду.

User Story 4. Як користувач, я хочу мати можливість отримувати доступ до окремих компонентів системи, щоб користуватися нею в разі проблемної роботи окремих компонентів.

User Story 5. Як користувач, я хочу мати можливість покривати програму тестами автоматично, щоб виявляти проблеми програми при зміні коду.

Подібний підхід широко застосовується на реальних проектах і дозволяє чітко визначити завдання які треба виконати для забезпечення роботи програми.

Відповідно для того щоб забезпечити кожну User Story нам необхідні сервіси для них.

Сервіс інтеграції з Git. Система повинна підтримувати інтеграцію з репозиторієм Git, щоб користувач завантажував роботи для подальшої перевірки.

Сервіс перевірки на плагіат. Система повинна інтегруватися зі сторонніми сервісами перевірки на плагіат, які дозволяють виявляти схожість програмного коду з іншими джерелами. Серед способів перевірки MOSS використовується широко багатьма університетами по всьому світу і показує чудові результати роботи.

Сервіс перевірки синтаксису і стилю оформлення коду. Для мови програмування Python використовують стандарт PEP 8. IDE вже забезпечують подібний аналіз при створенні коду і використовують алгоритми ШІ (статичного аналізу коду). Ми ж потребуємо сервіс, який буде аналізувати інші файли. Такі інструменти називаються lint (або linter) і часто спеціалізуються на конкретних мовах програмування, в нашому випадку Python.

Сервіс створення автоматичних тестів. Для цього використовуються сервіси які працюють на генетичних алгоритмах. Ми створюємо нові тести, відбираючі найкращі, послідовно схрещуючи їх і мутуючи. Потім відбувається нова популяція і повторюємо даний процес поки точність не зупиниться (зазвичай коли останні кілька сотень ітерацій не покращують її). Що важливо усі тести утворені таким чином унікальні, тож важливо створити їх попередньо для завдань.

3.2 Підбір методології та інструментарію розробки

Для забезпечення якісної розробки системи треба підібрати відповідний інструментарій та методологію. Подібна система складається з великої кількості інструментів, зокрема кожен мікросервіс має якісь особливі свої складові.

Як основну мову програмування будемо використовувати Python, завдяки своїй стабільній роботі на різних системах. Використовуємо дві версії: 3.10.9 та 3.11.1.

Для роботи з мовою програмування використаємо IDE PyCharm.

Також будемо використовувати систему керування версіями Git. Як в якості власної системи керування версіями, так і для подальшого завантаження студентських робіт.

Для створення власне мікросервісів будемо використовувати фреймвор Flask.

Для контейнеризації сервісів використовуємо Docker.

Для перевірки на плагіат використовуємо модуль mosspy для MOSS.

Для перевірки синтаксису використовуємо модуль Flake8.

Для автоматичної генерації тестів використовуємо модуль pynguin.

Розглянемо необхідний інструментарій детальніше.

3.2.1 Python

Python — це високорівнева, інтерпретована мова програмування загального призначення. Вона була створена у 1991 році Гвідо ван Россумом і має простий і зрозумілий синтаксис, що полегшує розробку програм. Python набув популярності завдяки своїй простоті, ефективності і широкому спектру застосувань.

Основні особливості Python:

- Синтаксис: Python має простий і зрозумілий синтаксис, який нагадує англійську мову. Це полегшує читання та розуміння коду, а також сприяє швидкому вивченню мови для початківців. використання пробілів для відступів (індентування) визначає блоки коду, що робить код більш читабельним.

- Інтерпретованість: Python є інтерпретованою мовою програмування, що означає, що виконання програми відбувається за допомогою інтерпретатора без необхідності компіляції в машинний код. Це дозволяє швидко випробовувати та розробляти програми без необхідності додаткових кроків компіляції.
- Розширюваність: Python надає можливість використовувати модулі та бібліотеки, що розширюють функціональність мови. У Python існує велика кількість стандартних модулів, які вже включені до мови, а також сторонніх бібліотек, які можна встановлювати і використовувати за допомогою інструментів, таких як pip (утиліта керування пакетами). Це сприяє повторному використанню коду, прискорює розробку програм і дозволяє розв'язувати широкий спектр завдань.
- Багатофункціональність: Python має широкий набір стандартних бібліотек, що надають готові рішення для різних задач. Наприклад, у стандартній бібліотеці Python є модулі для роботи зі строками, регулярними виразами, операціями введення-виведення, мережевим програмуванням, обробки XML-даних, роботи з базами даних і багато іншого. Це значно спрощує розробку програм і дозволяє ефективно виконувати різні завдання.
- Крос-платформеність: Python підтримується на різних операційних системах, таких як Windows, macOS і Linux. Це означає, що ви можете розробляти програми на Python на одній платформі і виконувати їх на іншій платформі без необхідності внесення значних змін у коді. Це робить Python універсальним і доступним для використання на різних системах.
- Велика спільнота: Python має активну та велику спільноту розробників, яка постійно працює над покращенням мови, розробкою нових бібліотек та інструментів, а також надає підтримку та допомогу іншим розробникам. У Python є багато ресурсів, таких як документація, форуми,

блоги та конференції, що допомагають розвиватися та вивчати мову програмування.

Python використовується в різних областях, таких як веб-розробка, наукові обчислення, штучний інтелект, аналіз даних, автоматизація завдань, розробка ігор та багато іншого. Його простота, ефективність та широкий спектр застосувань роблять Python однією з найпопулярніших мов програмування у світі. [19]

3.2.2 PyCharm

PyCharm – це інтегроване середовище розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. Воно надає широкий набір інструментів і функцій, спеціально призначених для покращення продуктивності розробки програм на Python.

Основні особливості PyCharm:

- Редагування коду: PyCharm має потужний редактор коду з розумним автодоповненням, підсвіткою синтаксису, перевіркою правильності коду та іншими корисними функціями. Він також підтримує рефакторинг коду, що дозволяє легко вносити зміни в структуру коду без пошкодження його функціональності.
- Налагодження: PyCharm надає потужний інструмент для налагодження програм на Python. Ви можете ставити точки зупинки, спостерігати за значеннями змінних, виконувати код крок за кроком і аналізувати стек викликів. Це допомагає виправляти помилки та вдосконалювати функціональність програм.
- Управління проектами: PyCharm дозволяє зручно керувати вашими проектами. Ви можете створювати нові проекти, додавати файли, керувати залежностями, налаштовувати середовище виконання і багато

іншого. Є можливість роботи з різними типами проектів, такими як консольні додатки, веб-додатки, наукові проекти тощо.

- Підтримка систем контролю версій: PyCharm інтегрується з популярними системами контролю версій, такими як Git, SVN і Mercurial. Ви можете використовувати всі основні функції систем контролю версій, такі як коміти, синхронізацію, відгалуження та злиття, прямо з інтерфейсу PyCharm.
- Підтримка розширень: PyCharm має велику кількість плагінів і розширень, які дозволяють розширити його функціональність. Ви можете встановлювати розширення для роботи з конкретними фреймворками, бібліотеками чи іншими інструментами, що полегшує розробку програм в певних областях.
- Інструменти для тестування: PyCharm має вбудовану підтримку для тестування коду. Ви можете легко створювати, запускати та аналізувати результати тестів безпосередньо в середовищі PyCharm.

Компанія JetBrains допомагає студентам з вивченням мов програмування і пропонує безкоштовне користування деякими платними функціями середовища.

PyCharm є дуже популярним середовищем розробки для Python завдяки своїм потужним можливостям, зручному інтерфейсу та підтримці великої спільноти розробників. [20]

3.2.3 Git та GitHub

Git – це розподілена система контролю версій, яка дозволяє вам відстежувати зміни в коді вашого проекту. Git працює локально на вашому комп'ютері і зберігає повну копію всіх версій вашого проекту, усіх гілок і комітів. Ви можете створювати нові гілки для відокремленої розробки нових функцій або виправлень помилок, а потім злити їх в основну гілку після завершення. Git дозволяє вам працювати незалежно від мережі, здійснювати

резервне копіювання вашого коду і легко переміщуватись між різними версіями проекту. [21]

GitHub – це хостингова платформа, яка надає веб-інтерфейс та додаткові функціональні можливості для спільної роботи над проектами, які зберігаються в Git-репозиторіях. GitHub дозволяє вам завантажувати ваші Git-репозиторії на цю платформу та зберігати їх у віддаленому репозиторії на серверах GitHub. Це дозволяє вам спільно працювати з іншими розробниками над проектами, вносити зміни до коду, обговорювати завдання, відстежувати проблеми та злиття гілок.

GitHub надає багато корисних функцій, зокрема:

- Issues (питання): GitHub дозволяє вам створювати питання, в яких можна обговорювати завдання, проблеми або питання щодо вашого проекту. Ви можете призначати питання розробникам, додавати мітки та встановлювати пріоритети.
- Pull requests (запити на злиття): Ви можете запропонувати зміни в коді, створивши pull request. Це дозволяє іншим розробникам переглянути ваші зміни, обговорити їх та внести коментарі. Після перевірки ваш код може бути злитий (merged) з основною гілкою проекту.
- Колаборація: GitHub дозволяє декільком розробникам спільно працювати над проектом. Ви можете запрошувати інших користувачів до вашого проекту, надавати їм доступ для спільного редагування коду, створення гілок та інших операцій.
- Управління версіями: GitHub зберігає повну історію змін вашого проекту, усі гілки і коміти. Ви можете переглядати, порівнювати і відкатувати зміни, а також стежити за внесками кожного розробника.
- Інтеграція з іншими сервісами: GitHub підтримує інтеграцію з різними інструментами, такими як системи з неперервної інтеграції (CI), системи відстеження помилок, сервіси розгортання та інші. [22]

Загалом, Git і GitHub співпрацюють, дозволяючи розробникам ефективно керувати версіями свого коду, спільно працювати над проектами та забезпечувати зручну спільну роботу з використанням розподіленої системи контролю версій.

3.2.4 Flask

Flask – це веб-фреймворк для мови програмування Python, який дозволяє розробляти веб-додатки швидко і ефективно. Він вважається одним з найпопулярніших фреймворків для веб-розробки на Python.

Ось деякі основні риси Flask:

- Простота використання: Flask має простий і лаконічний синтаксис, що полегшує розробку веб-додатків. Він пропонує базовий набір функціональності, але дозволяє розширювати його за потребою.
- Мінімалістична структура: Flask не нав'язує жорстких правил стосовно структури проекту. Ви можете організовувати свій проект за власними уподобаннями, що дозволяє більшу гнучкість.
- Розширюваність: Flask має багато розширень (extensions), які додають додаткові функціональні можливості до фреймворку. Це дозволяє легко і швидко додавати підтримку для баз даних, автентифікації, форм, апі і багатьох інших компонентів.
- Вбудований сервер розробки: Flask має вбудований сервер розробки, що дозволяє вам швидко перевірити свій веб-додаток без необхідності налаштування окремого веб-сервера.
- Підтримка шаблонів: Flask надає підтримку для використання шаблонів, таких як Jinja2, що спрощує розробку веб-сторінок і відокремлення логіки від представлення.

- RESTful підтримка: Flask добре підходить для розробки RESTful API. Ви можете легко створювати маршрути, обробляти запити HTTP і передавати дані у форматі JSON або іншому.

Flask є дуже гнучким і простим у використанні фреймворком, який забезпечує широкий набір можливостей для розробки веб-додатків на Python. Він підходить як для розробників які шукають швидку та ефективну розробку веб-проектів. [23]

3.2.5 Docker

Docker – це відкрите програмне забезпечення, що дозволяє упаковувати, розгортати та управляти програмами у віртуалізованих контейнерах. Він забезпечує стандартизований спосіб упаковки програм та їх залежностей, що дозволяє розробникам створювати ізольовані середовища, які легко переносити та встановлювати на будь-якому комп'ютері.

Основні принципи та поняття Docker включають:

- Контейнери: Docker використовує контейнери для ізоляції та упаковки програм та їх залежностей. Контейнери використовують різні рівні віртуалізації, що дозволяє їм працювати на різних операційних системах та серверах. Контейнери забезпечують консистентне середовище, у якому програми можуть працювати незалежно від конфігурації сервера або інших програм.
- Dockerfile: Dockerfile - це текстовий файл, у якому описується конфігурація та налаштування контейнера. В Dockerfile вказуються кроки для побудови образу контейнера, включаючи базовий образ, установку залежностей, копіювання файлів та конфігурацію середовища. З Dockerfile можна автоматизувати процес побудови контейнера і забезпечити його відтворюваність.

- Образи: Docker використовує образи для побудови та розгортання контейнерів. Образ - це виконуваний пакет, який містить всі необхідні компоненти, включаючи операційну систему, програми та залежності. Образи можна завантажувати з центрального реєстру Docker (Docker Hub) або створювати самостійно за допомогою Dockerfile.

Docker Desktop - це офіційна версія Docker для робочих станцій (комп'ютерів) з операційною системою Windows або macOS. Docker Desktop забезпечує зручний спосіб встановлення та управління Docker-середовищем на вашому комп'ютері. Він включає в себе Docker Engine, Docker CLI та інші компоненти, необхідні для роботи з Docker.

Docker Desktop дозволяє вам побудову, запуск та управління контейнерами прямо на вашому робочому столі. Ви можете використовувати Docker CLI або графічний інтерфейс Docker Desktop для взаємодії з Docker-середовищем та виконання операцій з контейнерами, образами та іншими компонентами Docker. [24]

3.2.6 MOSS

MOSS – це інструмент, який використовується для виявлення плагіату в програмному коді.

MOSS – це веб-сервіс, призначений для автоматичного порівняння програмного коду з метою виявлення схожості або плагіату. MOSS використовує алгоритми, які аналізують структуру коду та порівнюють його з великою базою даних інших програмних робіт. Він повертає звіт, в якому вказані схожі частини коду та їх відсоткове співпадіння. [25]

MOSSPY є пакетом Python, який надає зручний інтерфейс для взаємодії з MOSS. Він дозволяє розробникам використовувати функціональність MOSS у своїх програмах на Python. MOSSPY надає можливість відправляти код на аналіз до MOSS, отримувати результати порівняння та обробляти їх у зручному форматі.

MOSS та MOSSPY широко використовуються в академічних та дослідницьких галузях для виявлення плагіату в програмному коді. Вони можуть бути корисними інструментами для викладачів, які перевіряють роботи студентів, а також для дослідників, які аналізують схожість програмних робіт у певній галузі.

3.2.7 Flake8

Flake8 – це інструмент для перевірки стилю та якості коду у мові програмування Python. Він поєднує кілька незалежних пакетів: pycodestyle (також відомий як pep8), pyflakes та McCabe. Цей інструмент допомагає забезпечити дотримання рекомендацій PEP 8, який є стандартом для стилю коду в Python.

Основні функції та можливості Flake8 включають:

- Перевірка стилю коду: Flake8 перевіряє ваш код на відповідність рекомендаціям PEP 8, які включають правила щодо розташування пробілів, іменування змінних, розташування дужок, довжину рядків та багато іншого. Він допомагає забезпечити єдність стилю у вашому проекті та полегшує читання та розуміння коду.
- Виявлення потенційних проблем: Flake8 також перевіряє код на потенційні проблеми та помилки, які можуть призвести до неправильної роботи програми. Наприклад, він виявляє невикористовувані змінні, неправильне використання операторів, неправильну обробку винятків та інші типові проблеми програмування.
- Інтеграція з іншими інструментами: Flake8 легко інтегрується з іншими інструментами розробки Python, такими як редактори коду або системи неперервної інтеграції (CI). Ви можете налаштувати Flake8, щоб автоматично перевіряти код під час написання або виконання певних дій у вашому редакторі або CI-системі.

- Налаштування правил: Flake8 надає можливість налаштувати правила перевірки, включаючи додавання чи виключення конкретних правил. Це дозволяє вам пристосувати перевірки до своїх потреб та вимог проекту.

Flake8 є популярним інструментом серед розробників Python, оскільки допомагає підтримувати високу якість коду та однорідність стилю в проектах. Використання Flake8 може поліпшити читабельність, зрозумілість та підтримку вашого коду, сприяючи кращій розробці та підтримці проектів на Python. [26]

3.2.8 Pynguin

Pynguin – це інструмент для автоматичної генерації тестових сценаріїв (тест-кейсів) для програмного коду у мові програмування Python. Він використовує методи генетичного програмування для створення тестів, які покривають різні шляхи виконання коду та можуть виявити помилки та недоліки.

Основні можливості та функції Pynguin включають:

- Автоматичне створення тестових сценаріїв: Pynguin може автоматично створювати тестові сценарії для класів, функцій або модулів у вашому програмному коді. Він аналізує код та його структуру, і на основі цього генерує тестові дані та послідовності викликів, які можуть покривати різні гілки в коді.
- Покриття коду: Pynguin спрямований на забезпечення високого покриття коду тестами. Він створює тестові сценарії, які максимально можливо покривають логіку в коді, включаючи різні гілки, умови та особливості програми.
- Еволюційне вдосконалення: Pynguin використовує генетичне програмування для вдосконалення тестових сценаріїв. Він запускає

ітеративний процес, в якому генерується початкова популяція тестів, а потім застосовуються оператори схрещування, мутації та відбору, щоб покращити тестові сценарії з кожною ітерацією.

- Інтеграція з іншими інструментами: Pynguin може інтегруватися з іншими інструментами тестування, такими як unittest або pytest, щоб автоматично виконувати створені тестові сценарії та аналізувати результати. Ви можете легко інтегрувати Pynguin у вашу існуючу систему тестування та автоматично оновлювати тестові сценарії з кожним зміненням коду.

Pynguin допомагає автоматизувати процес створення тестів та покриття коду, дозволяючи швидше виявляти та виправляти помилки в програмному коді. Він може бути корисним для розробників, які прагнуть поліпшити якість свого коду та забезпечити більш широке покриття тестами. Остання версія не коректно працює з Python 3.11+, тому для нього використовуємо Python 3.10.9. [27]

3.3 Висновок до розділу

Було обрано стек технологій, який дозволяє нам покрити усі наші вимоги, а також легкий в інтеграції і розгортанні.

- Python
- PyCharm
- Git та GitHub
- Flask
- Docker
- MOSSPY
- Flake8
- Pynguin

А також було описано основні шляхи використання додатку (User Story) і описано загальні поняття про сервіси які мають забезпечувати його роботу.

4 РЕАЛІЗАЦІЯ СИСТЕМИ ПЕРЕВІРКИ СТУДЕНТСЬКИХ РОБІТ З ПРОГРАМУВАННЯ

Основними сутностями в програмі є сервіси. В результаті роботи було розроблено систему мікросервісів для забезпечення роботи додатку. Зокрема сервіс інтеграції з Git, сервіс перевірки програм на плагіат, сервіс перевірки синтаксису і стилю оформлення та сервіс створення автоматичних тестів.

4.1 Сервіс інтеграції з Git

Увесь сервіс інтеграції з Git реалізовано за допомогою пакету Flask, який дозволяє створювати веб-додатки. Код створює Flask-додаток і створює маршрут '/download', який прослуховує POST-запити.

При POST-запиті на маршрут '/download' сервіс очікує отримати JSON-об'єкт у тілі запиту, де міститься поле 'repo_url' з URL репозиторію на GitHub, який потрібно завантажити. Далі, сервіс виконує наступні кроки:

1. Отримує URL репозиторію з запиту.
2. Формує шлях до вихідної папки, де буде розміщений клонований репозиторій. Для цього з URL репозиторію видаляються початкова частина 'https://github.com/' і розширення '.git'.
3. Виконує команду git clone для клонування репозиторію за допомогою Git. Репозиторій з клонованими файлами зберігається у вихідній папці.
4. Повертає вихідний шлях до папки, який є ідентифікатором клонованого репозиторію.

У разі успішного клонування репозиторію сервіс повертає відповідь зі статусом 200 і вихідним шляхом до папки з клонованим репозиторієм. У випадку помилки під час клонування репозиторію, сервіс повертає відповідь зі статусом 500 і повідомленням про помилку.

Реалізація цього сервісу передбачає встановлення Flask і Git на сервері, на якому буде запущений код. Крім того, перед запуском сервера потрібно встановити необхідні залежності за допомогою pip install або будь-яким іншим інструментом для керування пакетами Python.

Вхідні дані:

- POST-запит на маршрут '/download'.
- JSON-об'єкт у тілі запиту з полем 'repo_url', що містить URL репозиторію на GitHub.

Вихідні дані:

- Успішна відповідь:
- Статус 200.
- Відповідь зі статусом 200 і вихідним шляхом до папки з клонованим репозиторієм.

Помилкова відповідь:

- Статус 500.
- JSON-об'єкт з полем 'error', що містить повідомлення про помилку.

4.2 Сервіс перевірки на плагіат

Flask-сервіс, який використовує бібліотеку Moss для перевірки плагіату між різними текстовими файлами.

У сервісі використовується словник LANGUAGES, який містить співставлення між назвами мов програмування та їх розширеннями файлів.

Реалізація включає наступні кроки:

1. Створення Flask-додатку та визначення маршруту /check_plagiarism для POST-запитів.

2. У POST-запиті виконується отримання JSON-об'єкта з даними, які включають шлях до папки directory, ідентифікатор користувача userid та назву мови програмування language.
3. Створення об'єкта Moss з використанням переданого ідентифікатора користувача та назви мови програмування.
4. Проходження по всіх файлах у папці directory та її підпапках, та додавання файлів, які відповідають обраній мові програмування, до об'єкта Moss.
5. Відправлення файлів до сервісу Moss для перевірки на плагіат та отримання URL результатів.
6. Повернення JSON-об'єкта з полем result_url, що містить отриманий URL для результатів перевірки на плагіат.

Вхідні дані:

- POST-запит на маршрут /check_plagiarism.
- JSON-об'єкт у тілі запиту з полями:
    - directory: шлях до папки, де знаходяться файли для перевірки на плагіат.
    - userid: ідентифікатор користувача, який використовується для доступу до сервісу Moss.
    - language: назва мови програмування, для якої виконується перевірка на плагіат.

Вихідні дані:

- JSON-об'єкт з полем result_url, яке містить URL для результатів перевірки на плагіат.

4.3 Сервіс перевірки синтаксису і стилю оформлення коду

Даний код представляє собою Flask-сервіс для перевірки синтаксису і стилю оформлення коду за допомогою бібліотеки flake8.

У сервісі використовується маршрут '/flake8' для POST-запитів.

Реалізація сервісу включає наступні кроки:

1. Створення Flask-додатку та визначення маршруту '/flake8' для POST-запитів.

2. Отримання шляху до папки, що містить файли для перевірки, з JSON-об'єкта запиту.

3. Визначення допустимих типів файлів у змінній `TYPE_FILES`

4. Створення об'єкта `style_guide` з використанням `flake8.api.legacy.get_style_guide()`, ігноруючи будь-які правила, вказані в `ignore=[]`.

5. Проходження по всіх файлах у заданій папці та її підпапках.

6. Перевірка, чи файл має один з допустимих типів змінної `TYPE_FILES`, і якщо так, то виклик `style_guide.check_files()` для перевірки синтаксису і стилю оформлення цього файлу.

7. Завершення перевірки та повернення JSON-об'єкта з полем 'status', що містить 'success'.

Вхідні дані:

- POST-запит на маршрут '/flake8'.
- JSON-об'єкт у тілі запиту з полем 'directory', що містить шлях до папки, в якій знаходяться файли для перевірки синтаксису і стилю оформлення коду.

Вихідні дані:

- JSON-об'єкт з полем 'status', яке містить значення 'success'.

4.4 Сервіс створення автоматичних тестів

Даний код представляє собою Flask-сервіс для створення автоматичних тестів за допомогою інструменту pynguin.

У сервісі використовується маршрут '/pynguin' для POST-запитів.

Реалізація сервісу включає наступні кроки:

1. Створення Flask-додатку та визначення маршруту '/pynguin' для POST-запитів.

2. Отримання шляху до вхідної папки з JSON-об'єкта запиту, використовуючи поле 'input_path'.

3. Отримання шляху до вихідної папки з JSON-об'єкта запиту, використовуючи поле 'output_path'.

4. Отримання назви модуля з JSON-об'єкта запиту, використовуючи поле 'module_name'.

5. Побудова команди для виконання pynguin з використанням отриманих шляхів і назви модуля.

6. Виклик команди за допомогою `subprocess.run()` для генерації автоматичних тестів.

7. Повернення шляху до папки згенерованих тестів у відповідь, якщо команда успішно виконалася.

8. Повернення повідомлення про помилку в разі виникнення помилки під час виконання команди або іншої помилки.

Вхідні дані:

- POST-запит на маршрут '/pynguin'.
- JSON-об'єкт у тілі запиту з полями:
  - 'input_path': шлях до вхідної папки, що містить код для генерації тестів.
  - 'output_path': шлях до папки, де будуть збережені згенеровані тести.
  - 'module_name': назва модуля, для якого генеруються тести.

Вихідні дані:

- Якщо команда успішно виконалася, то повертається рядок, який містить шлях до папки згенерованих тестів, додавши префікс 'test_' до назви модуля.

- Якщо під час виконання команди сталася помилка, повертається повідомлення про помилку, яке містить код помилки або опис помилки.

4.5 Розгортання додатку

Процес розгортання цих сервісів в Docker передбачає наступні кроки:

1. Створення Docker-контейнера для кожного сервісу, використовуючи Dockerfile.

2. Збірка Docker-образів для кожного сервісу.

3. Запуск Docker-контейнерів зі створеними Docker-образами.

Реалізація процесу розгортання цих сервісів в Docker включає наступні кроки:

1. Створення Dockerfile для кожного сервісу, в якому описується середовище та залежності для запуску сервісу.

2. Збірка Docker-образу для кожного сервісу на основі відповідного Dockerfile. Це можна зробити за допомогою команди `docker build`.

3. Запуск Docker-контейнера для кожного сервісу на основі відповідного Docker-образу. Це можна зробити за допомогою команди `docker run`.

Вхідні дані:

- Docker-файли (Dockerfile) для кожного сервісу.
- Вхідні дані для кожного сервісу (наприклад, конфігураційні файли, вхідні параметри).

Вихідні дані:

- Запущені Docker-контейнери для кожного сервісу, що працюють на визначених портах або мережевих інтерфейсах.

Процес розгортання в Docker може здійснюватися через Docker Compose, який дозволяє керувати та координувати розгортання кількох

контейнерів разом. Це особливо зручно, коли потрібно розгорнути декілька сервісів, які взаємодіють між собою.

В Docker Compose можна описати конфігурацію для кожного сервісу, включаючи Docker-файли, залежності та параметри запуску. Після цього можна запустити всі сервіси одночасно за допомогою команди `docker-compose up`.

Таким чином, для розгортання кожного з цих сервісів в Docker потрібно створити відповідний Dockerfile для кожного сервісу, сконфігурувати Docker Compose для керування цими контейнерами та вказати необхідні вхідні дані для кожного сервісу (наприклад, порти, шляхи до директорій тощо).

4.6 Тестування додатку

Додаток тестувався на роботах студентів за минулі роки, зокрема було обрано роботу по скрапінгу сайтів в пошуках даних про навчальні курси і їх рейтинг.

Першим перевіримо сервіс по клонуванню репозиторію. Для цього запустимо наступний скрипт:

```
repo_url = 'https://github.com/SulfurTech/scrape'

# Create the request payload
payload = {
    "repo_url": repo_url
}

# Send the POST request to the microservice
```

```
    response =
requests.post("http://localhost:1111/download",
json=payload)

    # Check the response status code
    if response.status_code == 200:
        output_path = response.content.decode('utf-8')
        print('Directory with repo:', output_path)
    else:
        print("Error:", response.text)
        output_path =
repo_url.replace('https://github.com/',
'').replace('.git', '')
```

Як результат маємо виведений шлях:

```
git_api
Directory with repo: SulfurTech/scrape
```

Рисунок 4.1 – результат роботи сервісу інтеграції з Git

Запустимо скріпт для перевірки сервісу з flake8:

```
    directory = output_path
    # Create the request payload
    payload = {
        "directory": directory
    }

    # Make a POST request to the /flake8 endpoint
    response =
requests.post("http://localhost:2222/flake8",
json=payload)

    # Check the response status code
```

```
if response.status_code == 200:
    # Print the response content
    data = response.json()
else:
    print("Error:", response.text)
```

В результаті бачимо список стилістичних помилок, які містить код:

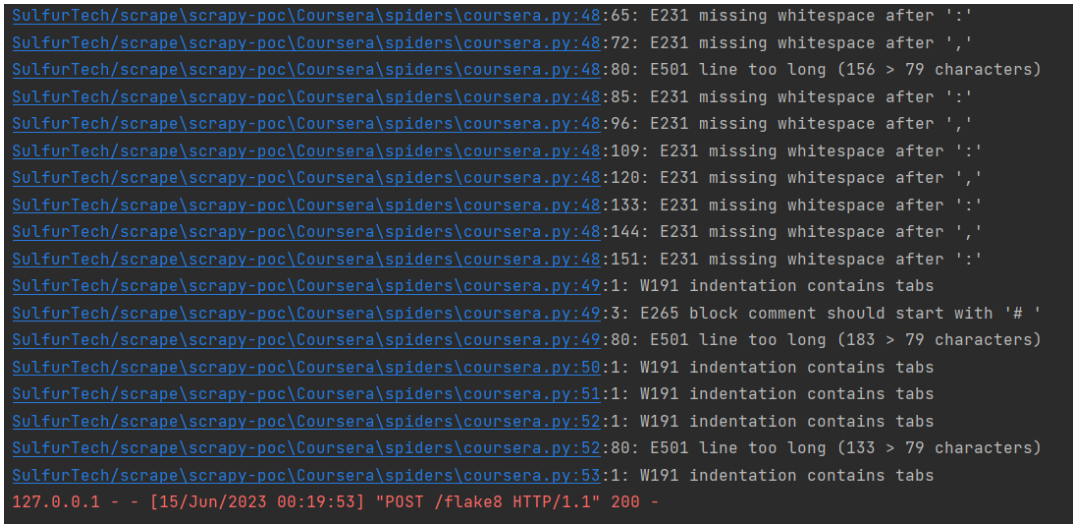


Рисунок 4.2 – результат роботи сервісу перевірки синтаксису і стилю оформлення коду

Перевіримо сервіс перевірки на плагіат:

```
userid = ‘ID’
directory = output_path
language = 'python'

# Create the request payload
payload = {
    "directory": directory,
    "userid": userid,
    "language": language
}
```

```
    # Make a POST request to the /check_plagiarism
endpoint

    response =
requests.post(f"http://localhost:3333/check_plagiarism"
, json=payload)

    # Check the response status code
    if response.status_code == 200:
        result = response.json()
        result_url = result['result_url']
        print("Plagiarism check complete. Results
available at:", result_url)
    else:
        print("Error:", response.text)
```

В результаті маємо посилання на сайт Стенфордського університету, який власне і забезпечує MOSS перевірку:

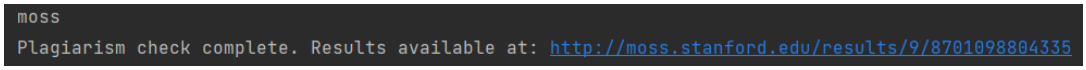
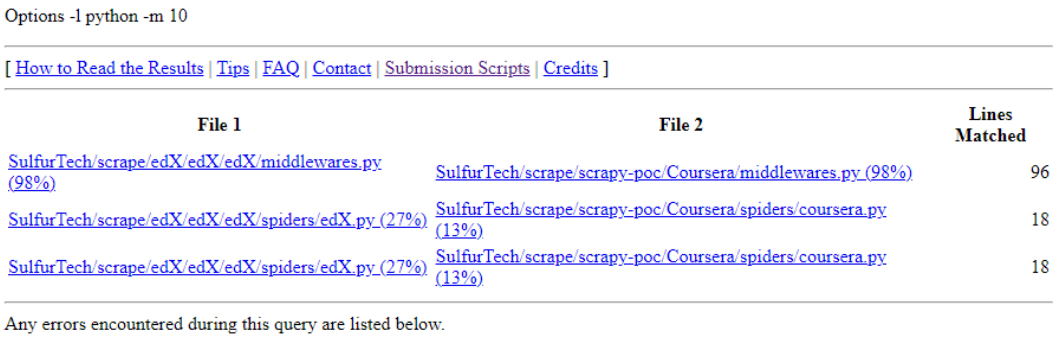


Рисунок 4.3 – результат роботи сервісу перевірки на плагіат (посилання)

За посиланням можемо знайти наступне:


Options -l python -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
SulfurTech/scrape/edX/edX/edX/middlewares.py (98%)	SulfurTech/scrape/scrapy-poc/Coursera/middlewares.py (98%)	96
SulfurTech/scrape/edX/edX/edX/spiders/edX.py (27%)	SulfurTech/scrape/scrapy-poc/Coursera/spiders/coursera.py (13%)	18
SulfurTech/scrape/edX/edX/edX/spiders/edX.py (27%)	SulfurTech/scrape/scrapy-poc/Coursera/spiders/coursera.py (13%)	18

Any errors encountered during this query are listed below.

Рисунок 4.4 – результат роботи сервісу перевірки на плагіат (сторінка)

Відсоток плагіату досить високий, що недивно для програм скраперів. Зокрема плагіат виявлено в блоках:

./spiders/edX.py

```
def parse_courses(self,response):
     courses = response.css('.d-card-wrapper')
     for course in courses:
          name = course.css('.d-card-body
h3::text').extract_first()
```

./spiders/coursera.py

```
def parse_courses(self,response):
     courses = response.css('.anchor-wrapper')
     for course in courses:
          name = course.css('.headline-1-
text::text').extract_first()
```

Власне ці блоки і працюють за однаковим принципом і відрізняються лише стилями CSS коду, який позначає складові сторінки.

Перевіримо так само генерацію тестів на модулі test.py, який вміщує в собі код для калькулятора з основними операціями:

```
payload = {
    "input_path": output_path,
    "output_path": output_path + '_test',
    "module_name": 'test'
}

response =
requests.post(f"http://localhost:4444/pynguin",
json=payload)
```

```
    # Check the response status code
    if response.status_code == 200:
        result = response.json()
        result_url = result['result_url']
        print("Test genered. Results available at:",
output_path + '_test')
    else:
        print("Error:", response.text)
```

В результаті маємо:

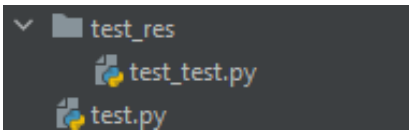


Рисунок 4.5 – результат роботи сервісу створення автоматичних тестів

Де скрипт файлу test_test.py:

```
    @pytest.mark.xfail(strict=True)
    def test_case_0():
        bool_0 = False
        var_0 = module_0.div(bool_0, bool_0)
        none_type_0 = None
        var_1 = module_0.sub(bool_0, bool_0)
        module_0.add(bool_0, none_type_0)


    @pytest.mark.xfail(strict=True)
    def test_case_1():
        float_0 = -2250.221463
        var_0 = module_0.div(float_0, float_0)
        assert var_0 == pytest.approx(1.0, abs=0.01,
rel=0.01)
        set_0 = {float_0, float_0}
        module_0.sub(set_0, float_0)
```

```
@pytest.mark.xfail(strict=True)
def test_case_2():
    complex_0 = -395.3 - 2986j
    tuple_0 = ()
    list_0 = [complex_0, tuple_0]
    module_0.mult(list_0, complex_0)

@pytest.mark.xfail(strict=True)
def test_case_3():
    complex_0 = -2384.9824 + 94.506436j
    none_type_0 = None
    module_0.add(complex_0, none_type_0)

@pytest.mark.xfail(strict=True)
def test_case_4():
    int_0 = 854
    none_type_0 = None
    module_0.sub(int_0, none_type_0)
```

4.7 Висновок до розділу

Було реалізовано чотири мікросервіси:

- Сервіс інтеграції з Git
- Сервіс перевірки на плагіат
- Сервіс перевірки синтаксису і стилю оформлення коду
- Сервіс створення автоматичних тестів

Було застосовано роботу додатку для перевірки робіт з програмування.

Виділено деякі проблеми додатку, зокрема відсутність інтерфейсу для легкої взаємодії, підтримку лінтерів для інших мов програмування, необхідність створення інструменту для звітування з результатів.

Зробити подібний додаток однією особою складно, тому не дивно що він має недоліки. Варто краще продумати бізнес задачі додатку та реалізувати бракуючий функціонал.

5 ФУНКЦІОНАЛЬНО ВАРТІСНИЙ АНАЛІЗ РОЗРОБЛЕНОГО ДОДАТКА

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

5.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

5.2. Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє класифікувати дані за обмеженої кількості маркованих екземплярів. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір самої програми.

F_2 – вибір методу впровадження алгоритмів.

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) мова програмування Python.

б) мова програмування C++.

Функція F_2.

а) використання готових бібліотек.

б) написання алгоритмів вручну.

Функція F_3:

а) Python Notebook.

б) IDE Jetbrains.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1).

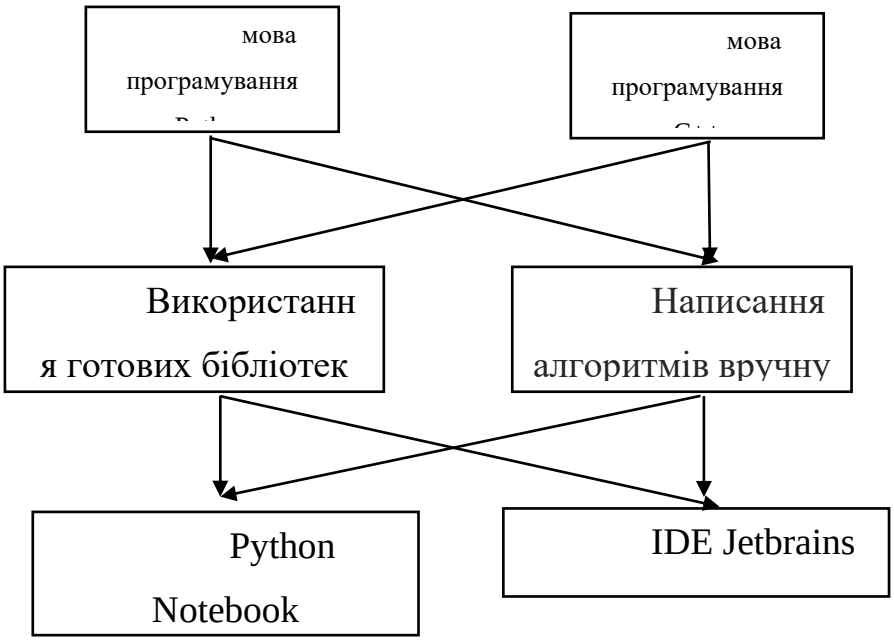


Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в таблиці 5.1.

Таблиця 5.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*A*	Кросплатформеність, доступність бібліотек, займає менше часу при	Динамічна типізація

		написанні коду	
	Б	Код швидко виконується	Більший час написання коду
F_2	*А*	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв'язати
	Б	Ідеально описують усі необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	*А*	Широкий вибір можливостей, легко запускається на будь-якому сервері	Відсутність відлагодження коду
	Б	Доступна можливість відлагодження коду	Необхідна локальна інсталяція

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу даємо гнучкості та швидкодії. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Існуючі бібліотеки мови програмування Python є надійними і оптимізованими для різних задач, тому написання алгоритмів вручну не є оптимальним варіантом для даної задачі, отже відкидаємо варіант Б. Функція F_3:

Середовище розробки не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти А та Б гідними розгляду.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2а – F_3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.3 Обґрунтування систем параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 5.2.

Таблиця 5.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	с	18	8	4
Об’єм пам’яті	X2	Мб	320	128	64

Потенційний об’єм програмного коду	X3	кількість рядків коду	3000	1600	800

За даними таблиці 5.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.4.

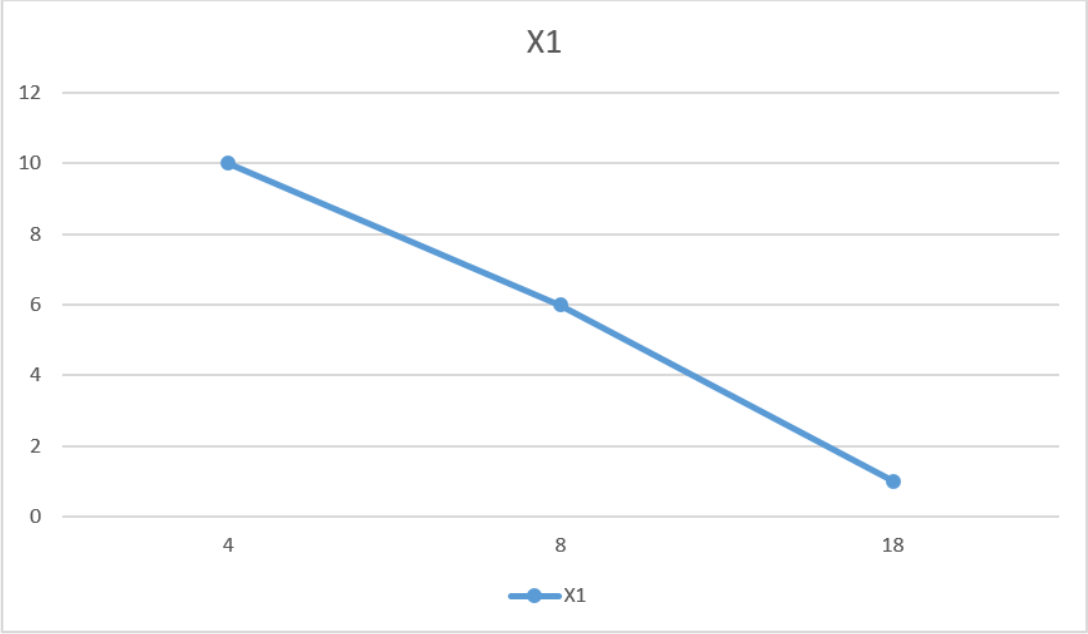


Рисунок 5.2 – X1, швидкодія мови програмування

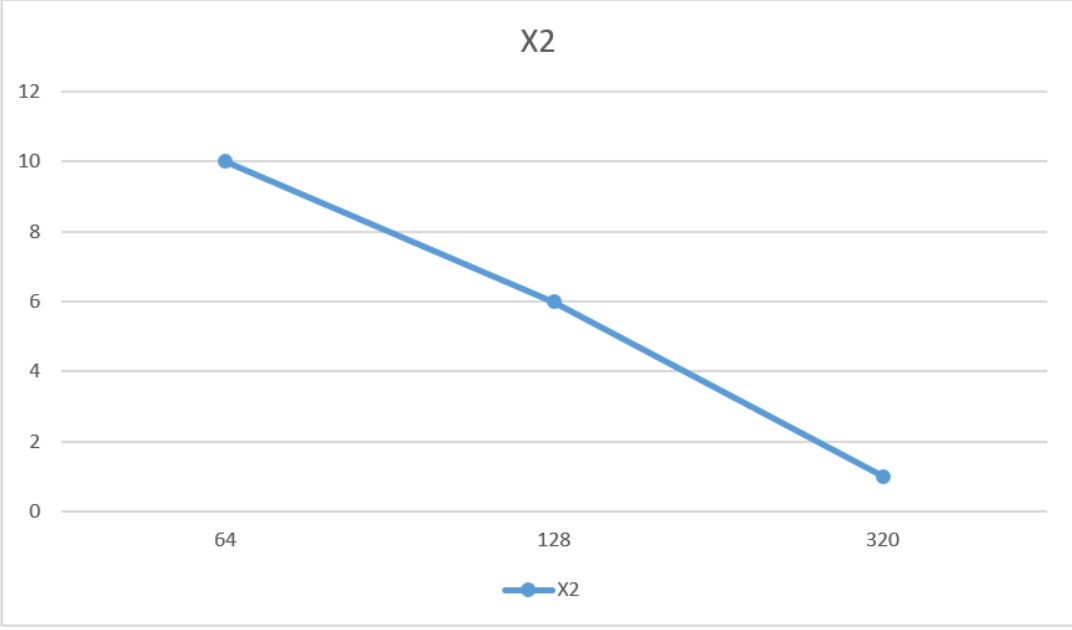


Рисунок 5.3 – X2, об’єм пам’яті

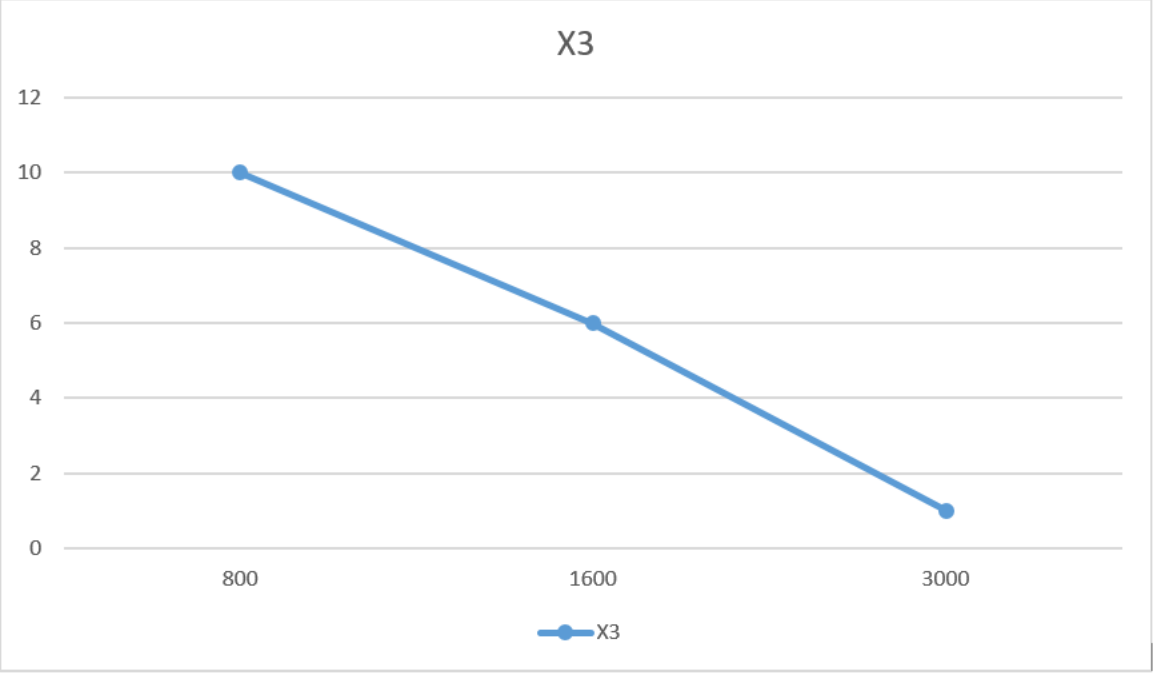


Рисунок 5.4 – X3, потенційний об'єм програмного коду

5.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	2	3	2	2	2	14	0	0
X2	Об’єм пам’яті	Mb	2	1	1	1	1	1	1	8	-6	36
X3	Потенційний об’єм програмного коду	Кількість рядків коду	3	3	3	2	3	3	3	25	7,5	56,25
	Разом		6	6	6	6	6	6	6	42	0	72

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (5.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{5.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{5.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \tag{5.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 72}{7^2(3^3 - 3)} = 0{,}734 > W_k = 0{,}67. \tag{5.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	>	0,5

X1 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (5.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (5.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (5.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (5.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \quad (5.10)$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j			Перша ітер.		Друга ітер.	
	X1	X2	X3	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$
X1	1	1,5	0,5	3	0,333	8	0,320
X2	0,5	1	0,5	2	0,222	5,5	0,220
X3	1,5	1,5	1	4	0,444	11,5	0,460
Всього:				9	1	25	1

5.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо (табл.5.6).

Абсолютні значення параметрів *X2* та *X1* відповідають технічним вимогам умов функціонування даного ПП.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (5.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 5.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	8	6	0,32	1,92
F2	А	X2	87	5	0,46	2,3
F3	А	X3	1200	9	0,22	1,98
F3	Б	X3	1800	7	0,22	1,54

За даними з таблиці 5.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (5.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}92 + 1{,}1 + 1{,}98 = 6{,}2 ;$$

$$K_{K2} = 1{,}92 + 1{,}1 + 3{,}22 = 4{,}56.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (5.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 37 \cdot 1.8 \cdot 0.9 = 59{,}94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =29 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$T_2 = 29 \cdot \ 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (59{,}94 \ + 20.88 + 4.8 + 20.88) \cdot 8 = 852 \quad \text{людино-годин.}$$

$$T_{II} = (59{,}94 \ + 20.88 + 6.91 + 20.88) \cdot 8 = 868{,}88 \quad \text{людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці бере участь один програміст з окладом 25000 грн., один аналітик в області даних з окладом 27000. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.,} \qquad (5.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{25000 + 27000}{3 \cdot 21 \cdot 8} = 103{,}18 \ \text{ грн.} \qquad (5.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_{ч} \cdot Т_{i} \cdot К_{Д}, \quad (5.16)$$

де $С_{ч}$– величина погодинної оплати праці програміста;

$Т_{i}$ – трудомісткість відповідного завдання;

$К_{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 103{,}18 \cdot 852 \cdot 1.2 = 105491{,}23$ грн.

II. $С_{ЗП} = 103{,}18 \cdot 868.88 \cdot 1.2 = 107581{,}25$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 105491{,}23 \cdot 0.22 = 23208{,}07$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 107581{,}25 \cdot 0.22 = 23667{,}88$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_{М}$)

Так як одна ЕОМ обслуговує одного програміста з окладом 25000 грн. та аналітика даних з окладом 27000 з коефіцієнтом зайнятості 0,2, то для однієї машини отримаємо:

$$С_{Г} = 12 \cdot М \cdot К_{З} = 12 \cdot 25000 \cdot 0{,}2 + 12 \cdot 27000 \cdot 0.2 = 124800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_{Г} \cdot (1 + К_{З}) = 124800 \cdot (1 + 0.2) = 149760 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД}= С_{ЗП} \cdot 0.22 = 149760 \cdot 0,22 = \quad 32947,2 \quad грн.$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 20000 = 7000 \quad грн.,$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 20000 \cdot 0.08 = 2240 \quad грн.,$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.35 = \\ = \quad 627,2 \quad години,$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \cdot 0,7 \cdot 0,2 \cdot 4,79 = 420,6 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 20000 \cdot 0,67 = 13400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (5.17)$$

$$С_{ЕКС} = 149760 + 32947,2 + 7000 + 2240 + 420,6 + 13400 = 205767,80 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 205767,80 / 627,2 = 328,07 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \quad (5.18)$$

I. $С_М = 328,07 \cdot 852 = 279518,76$ грн.

II. $С_М = 328,07 \cdot 868,88 = 285053,46$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (5.19)$$

I. $С_Н = 279518,76 \cdot 0,67 = 187277,57$ грн.

II. $С_Н = 285053,46 \cdot 0,67 = 190985,82$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (5.20)$$

I. $С_{ПП} = 105491,23 + 23208,07 + 279518,76 + 187277,57 = 595495,63$ грн.

II. $С_{ПП} = 107581,25 + 23667,88 + 285053,46 + 190985,82 = 607288,41$ грн.

5.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (5.21)$$

$$К_{ТЕР1} = 6,2 / 595495,63 = 1,041 \cdot 10^{-5},$$

$$К_{ТЕР2} = 4,56 / \ 607288,41 = 0,751 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1,041 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,041 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– Вибір програмного продукту – Python;

– Використання готових бібліотек;

– Середовище розробки Python Notebook.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

5.8 Висновки до розділу

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній дипломній роботі було розроблено мікросервісну архітектуру системи для перевірки студентських робіт з програмування. Розглянуті існуючі рішення з даної теми виявили проблему, яку необхідно було вирішити. Для реалізації такого додатку було проаналізовано різні архітектурні підходи, методології та інструментарій.

У ході роботи було розроблено наступні сервіси:

1. Сервіс інтеграції з Git: Цей сервіс дозволяє користувачеві завантажувати свої роботи з програмування з репозиторію Git для подальшої перевірки.

2. Сервіс перевірки на плагіат: Цей сервіс інтегрується зі сторонніми сервісами перевірки на плагіат, зокрема з MOSS (Measure Of Software Similarity), який широко використовується університетами по всьому світу. Він дозволяє виявляти схожість програмного коду з іншими джерелами.

3. Сервіс перевірки синтаксису і стилю оформлення коду: Цей сервіс аналізує файли з кодом на мові програмування Python згідно стандарту PEP 8. Він використовує алгоритми статичного аналізу коду для забезпечення відповідності синтаксису та стилю оформлення.

4. Сервіс створення автоматичних тестів: Для створення тестів використовуються сервіси, що працюють на генетичних алгоритмах. Вони генерують нові тести, шляхом схрещування і мутації, і відбирають найкращі з них. Цей процес повторюється до досягнення заданої точності. Важливою особливістю цього сервісу є створення унікальних тестів для кожного завдання.

Розроблений додаток відповідає поставленим завданням у дипломній роботі, забезпечуючи можливість інтеграції з Git, перевірку на плагіат, аналіз синтаксису і стилю оформлення коду, а також автоматичне створення тестів для завдань. Ці сервіси сприяють покращенню процесу перевірки

студентських робіт з програмування, забезпечуючи актуальність, безпеку та стійкість продукту для користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційний сайт платформи Moodle URL: https://moodle.org/ (дата звернення 01.06.2023)

2. Офіційний сайт Стенфордського центру для викладання і вивчення URL: https://gradescope.stanford.edu/ (дата звернення 01.06.2023)

3. Офіційний сайт Массачусетського технологічного інституту URL: http://kb.mit.edu/ (дата звернення 01.06.2023)

4. Офіційний сайт Гарвардського університету, відділу Академічних технологій URL: https://atg.fas.harvard.edu/gradescope (дата звернення 01.06.2023)

5. Офіційний сайт Карнегі-Меллонського університету URL: https://www.cmu.edu/canvas/gradescope/ (дата звернення 01.06.2023)

6. Офіційний сайт Йельського університету URL: https://www.yale.edu/ (дата звернення 01.06.2023)

7. Офіційний сайт платформи Gradescope URL: https://www.gradescope.com/ (дата звернення 01.06.2023)

8. Офіційний сайт платформи Coursera URL: https://www.coursera.org/ (дата звернення 01.06.2023)

9. Офіційний сайт платформи HackerRank URL: https://www.hackerrank.com/products/school/ (дата звернення 01.06.2023)

10. Сайт платфоми «Сікорський» URL: https://www.sikorsky-distance.org/ (дата звернення 01.06.2023)

11. Офіційний блог розробників IBM про мікросервісну архітектуру URL: https://www.ibm.com/ (дата звернення 07.06.2023)

12. Офіційний блог розробників Netflix URL: https://netflixtechblog.com/ (дата звернення 07.06.2023)

13. Офіційний блог розробників Amazon URL: https://aws.amazon.com/blogs/aws/ (дата звернення 07.06.2023)

14. Презентація системного архітектора компанії Uber з конференції QCon London 2015 URL: https://www.infoq.com/presentations/uber-market-platform/ (дата звернення 07.06.2023)

15. Презентація віце-президента Spotify з конференції goto Berlin 2015 URL: http://gotocon.com/berlin-2015/presentation/Microservices%20@%20Spotify (дата звернення 07.06.2023)

16. Офіційний блог розробників Airbnb на платформі Medium URL: https://medium.com/airbnb-engineering (дата звернення 07.06.2023)

17. Офіційний блог розробників Twitter URL: https://blog.twitter.com/engineering/en_us/topics/infrastructure/ (дата звернення 07.06.2023)

18. Порівняння мікросервісної структури та моноліту від Atlasian URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (дата звернення 07.06.2023)

19. Офіційна сторінка засновників мови програмування Python URL: https://www.python.org/ (дата звернення 10.06.2023)

20. Офіційна сторінка продукту PyCharm URL: https://www.jetbrains.com/pycharm/ (дата звернення 10.06.2023)

21. Офіційна сторінка засновників Git URL: https://git-scm.com/ (дата звернення 10.06.2023)

22. Офіційна сторінка платформи GitHub URL: https://github.com/ (дата звернення 10.06.2023)

23. Документація до фреймворку Flask URL: https://flask.palletsprojects.com/en/ (дата звернення 10.06.2023)

24. Офіційна сторінка інструменту Docker URL: https://www.docker.com/ (дата звернення 10.06.2023)

25. Офіційна сторінка сервісу MOSS URL: http://moss.stanford.edu/ (дата звернення 10.06.2023)

26. Документація до лінтеру Flake8 URL: https://flake8.pycqa.org/ (дата звернення 10.06.2023)

27. Офіційна документація до фреймворку Pynguin URL: https://pynguin.readthedocs.io/ (дата звернення 10.06.2023)

ДОДАТОК А

./git_api.py

```
import os
import shutil
import requests
from flask import Flask, request, jsonify, send_file

app = Flask(__name__)


@app.route('/download', methods=['POST'])
def download_repo():
    # Get the repository URL from the request body
    repo_url = request.json.get('repo_url')
    output_path =
repo_url.replace('https://github.com/',
'').replace('.git', '')
    try:
        # Clone the repository using git
        os.system(f'git clone {repo_url}
{output_path}')

        # Send the ZIP file as a response
        return output_path

    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=1111)
```

ДОДАТОК Б

./flake_api.py

```
import os
import glob
from flask import Flask, request, jsonify
import flake8.api.legacy

app = Flask(__name__)

@app.route('/flake8', methods=['POST'])
def check_files_in_directory():
    directory = request.json['directory']  # Retrieve
the directory from the request
    TYPE_FILES = ['.py']
    style_guide =
flake8.api.legacy.get_style_guide(ignore=[])
    for root, dirs, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)
            if any(file.endswith(file_type) for
file_type in TYPE_FILES):
                report =
style_guide.check_files([file_path])

    return jsonify({'status': 'success'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=2222)
```

ДОДАТОК В

./moss_api.py

```
LANGUAGES = {
        'c': '.c',
        'cc': '.cc',
        'java': '.java',
        'ml': '.ml',
        'pascal': '.pas',
        'ada': '.ada',
        'lisp': '.lisp',
        'scheme': '.scm',
        'haskell': '.hs',
        'fortran': '.f90',
        'ascii': '.ascii',
        'vhdl': '.vhdl',
        'verilog': '.v',
        'perl': '.pl',
        'matlab': '.m',
        'python': '.py',
        'mips': '.mips',
        'prolog': '.pl',
        'spice': '.spice',
        'vb': '.vb',
        'csharp': '.cs',
        'modula2': '.mod',
        'a8086': '.asm',
        'javascript': '.js',
        'plsql': '.sql'
    }
```

```
import os
from flask import Flask, request
from mosspy import Moss

app = Flask(__name__)

@app.route('/check_plagiarism', methods=['POST'])
def check_plagiarism():
        # Get directory and user ID from the request
    data = request.get_json()
    directory = data['directory']
    userid = data['userid']
    language = data['language']
    type_files = LANGUAGES[language]

    # Create Moss object with your Moss user ID
    m = Moss(userid, language)

    # Add all files in the directory and its
subdirectories to Moss
    for root, _, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)
            if any(file.endswith(file_type) for
file_type in type_files):
                if os.path.getsize(file_path) > 0:
                    m.addFile(file_path)

    # Send files to Moss and obtain the URL for the
results
```

```
    url = m.send()

    return {'result_url': url}

if __name__ == '__main__':
  app.run(host='0.0.0.0', port=3333)
```

ДОДАТОК Г

./pynguin_api.py

```
import subprocess
from flask import Flask, request

app = Flask(__name__)

@app.route('/pynguin', methods=['POST'])
def test_generator():
    input_path = request.json.get('input_path')
    output_path = request.json.get('output_path')
    module_name = request.json.get('module_name')
    command_to_run = f"pynguin \
    --project-path {input_path} \
    --output-path {output_path} \
    --module-name {module_name}"
    try:
        subprocess.run(command_to_run, shell=True,
check=True)
        return
f"{output_path}\\test_{module_name}"
    except subprocess.CalledProcessError as e:
        return f"Command execution failed with
error code {e.returncode}."
    except Exception as e:
        return f"An error occurred: {str(e)}"


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=4444)
```