НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК

«ПОГОДЖЕНО»
Декан факультету
інформаційних технологій
Болобот І.М., д.т.н., професор
_____
____ _____ 202_ р.

«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»
Завідувач кафедри комп'ютерних наук
Голуб Б.Л., к.т.н., доцент
_____
____ _____ 202_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему ____Автоматизована система підтримки прийняття рішень при тестуванні програмного забезпечення_____

Спеціальність _____122 «Комп'ютерні науки » _____

_____
(код і назва)

Освітня програма __Комп'ютерний еколого - економічний моніторинг_____

_____
(назва)

Орієнтація освітньої програми _____
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

_____ _____ __Кириченко В.В._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

_____ст. викладач_____ _____ __Ніколаєнко Д.В._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ ___Світлак А. Ю. _____
(підпис) (ПІБ студента)

КИЇВ-2024

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) _____

ЗАТВЕРДЖУЮ
Завідувач кафедри_____
_____
_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
“____” _____20_____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____Світлак Артем Юрійович_____
(прізвище, ім’я, по батькові)

Спеціальність ____ « 122 Комп’ютерні науки» _____
(код і назва)

Освітня програма ___Комп’ютерний еколого-економічний моніторинг_____
(назва)

Орієнтація освітньої програми_____
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи_____
_____
_____

затверджена наказом ректора НУБіП України від “____”_____20___р. №_____

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи _____
_____
_____
_____
_____
_____

Перелік питань, що підлягають дослідженню:

1. _____
2. _____
3. _____

Перелік графічного матеріалу (за потреби) _____
_____

Дата видачі завдання “_____”_____20___ р.

Керівник магістерської кваліфікаційної роботи _____Ніколаєнко Д.В._____
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ ____Світлак А. Ю._____
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ВСТУП

Сучасне прорамне забезпечення є невід’омною складовою багатьох галузей діяльності, забезпечуючи автоматизацію процесів, підвищення продуктивності та зручність використання. Разом із цим, через зростаючу складність створення веб-додатків з кожним днем стає все складніше контролювати за його якістю та надійністю через що витрачається все більше часу не тільки на розробку застосунків, а і на їх тестування. Ефективне тестування дозволяє виявити помилки на ранніх етапах розробки, знизити ризики збоїв та підвищити довіру користувачів до продукту.

У зв’язку із динамічним розвитком IT-індустрії, створення системи яка дозволить зменшити використання людського ресурсу при тестуванні програмного забезпечення стає все актуальніше. Так система дозволить скоротити час, необхідний для виконання рутинних задач, забезпечить систематичний підхід до перевірки програмного забезпечення та збільшить час, який може використати спеціаліст із тестування на більш важливі етапи тестування.

Саме тому розробка автоматизованої системи підтримки прийняття рішень із тестування програмного забезпечення є валживим напрямом із покращення результатів тестування програмногозабезпечення. Така система дозволить не лише скоротити процес тестування, а і зробити його більш інтелектуальним, забезпечуючи аналіз даних, та можливість прогнозування результатів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Сучасні програмні системи стають все більш складними. Через це збільшується і обсяг тестування, однак через обмеження у часі зазвичай не вдається провести ретельне тестування через що системи випускаються ненадійними. Такі системи можуть поставити під загрозу дані користувачів. Через це створення системи яка допоможе у тестування програмного забпезчення стає все необхіднішою.

Причинами для створення такої системи можна назвати:

1. Зростаючий обяг програмного забезпечення, через що збільшується і час тестування.
2. Зростаючий обсяг програмного забезпечення також впливає і на кількість помилок які спеціаліст із тестування не зможе знайти через обмеження у часі.
3. Автоматизація деяких процесіву тестування дозвоить більш ефективно розподіляти ресурси тестувальника.
4. Також розробникам потрібно адаптуватися та використовувати наві технології, такі як штучний інтелекто та машинне навчання.

1.2 Що таке тестування програмного забезпечення

Основна мета тестування- це процес перевірки програмного забезпечення з метою виявлення помилок, недоліків перед випуском на ринок або в експлуатацію. Тестування відбувається у всіх сферах програмування, зазвичай декілька разів у процесі розробки продутку. У деяких випадках, такі як розробка ігор тестування відбувається майже постійно. Альфа-тестування та бета-тестування це ті види тестування, на які можуть запрошувати гравців, однак і під час розробки гри він тестування відбувається спеціалістами.

Процес тестування охоплює запуск системи із різними вхідними станами та умовами, а також аналіз реакції системи на ці дані. Окрім цього перевіряються різні стани та умови, за яких запусається програмне забезпечення. Наприклад, тестування веб-застосунків зазвичай проводиться із використанням декількох браузерів [1].

Якщо це звичайний застосунок, його тестують із різними версіями операційної системи.

Тестування програмного забезпечення може відбуватися як розробниками продукту, так і спеціалістами з тестування або звичайними користувачам. Для звичайних користувачів доступ надається зазвичай на фінальних стадіях розробки застосунку.

1.2.1 Методи тестування програмного забезпечення

Існують два типи тестування, які проводять для перевірки програмного забезпечення: ручне тестування та автоматизоване [2].

При ручному тестуванні тестувальних виконує всі тести вручну, не використовуючи ніяких засобів автоматизації. Це низькорівневий та простий тип тестування, який не вимагає великої кількості додаткових знань. Також такий вид тестування вимагає аналітичних навичок.

Автоматизоване тестування передбачає застосування спеціального програмного забезпечення для виконання тестів та порівняння фактичних результатів роботи програмного забезпечення із очікуваними. Цей підхід дозволяє автоматизувани ті дії, що часто повторюються, забезпечуючи при цьому максимальне тестове покриття завдання. Для цього виду тестувань потрібно знання мови програмування та додаткових бібліотек для підключення, налагодження та запуску тестового випадку. Однак деякі типи тестування можуть бути виконуватись автоматизоване без знання програмування завдяки спеціалізованим інструментам, що дозволяють створювати й керувати тестами.

Не можна сказати что ручне тестування краще автоматизованого, адже воно необхідне через неможливість повної автоматизації тестувань.

У табл. 1.1 можна побачити порівняння ручного і автоматизованого тестування.

Табл. 1.1

	Ручне тестування	Автоматизоване тестування
Задання вхідних значень	Зазвичай використовуються різні значення створенні індивідуально	Вхідні значення чітко задані
Перевірка результатів	Дозволяє тестувальнику оцінювати нечітко сформульовані критерії	Оскільки вхідні параметри чітко задані, перевірка досить вузька
Повторюваність	Через складність повторного набору значень, повторити результат буде складно	Можливість повторення однакових вхідних значень
Чутливість на зміни у продукті	Незначні зміни не впливають на роботу тестувальника	Незначні зміни можуть призвести до необхідності переробки тесту
Швидкість виконання тестових випадків	Потрібно багато часу на виконання навіть одного тестового випадку	Виконання тестового випадку відбувається дуже швидко

1.2.2 Види та типи тестування програмного забезпечення

Залежно від цілей тестування, програмне забезпечення перевіряють за допомогою таких основних видів:

1. Функціональне тестування. Цей вид тестування зосереджений на перевірці зовнішньої поведінки системи, тобто виконуваних нею функцій. До нього належать:
   - Функціональне тестування. Забезпечує оцінку коректності виконання системою своїх функцій відповідно до визначених специфікацією. Може застосовуватись на всіх етапах тестування.
   - Тестування безпеки. Орієнтоване на перевірку здатності систем захищати конфіденційні дані та протистояти несанкційованому доступу .
   - Тестування взаємодії. Визначає здатність програмного забезпечення взаємодіяти із зовнішніми компонентами чи стстемами. Також включає інтеграційне тестування та тестування сумісності.
2. Нефункціональне тестування. Спрямоване на оцінку нефункціональних характеристик системи, зокрема специфічних властивостей програмного забезпечення. Сюди входять.
   - Конфігураційне тестування. Воно перевіряє роботу системи у різних конфігураціях програмного забезпечення та обладнання, які вона підтримує
   - Тестування на відмову і відновлення. Аналізує здатність системи залишатися працездатною в разі збоїв і відновлюватсь після них.

- Тестування зручності використання. Вивчає простоту взаємодії користувача із системою, а також зрозумілість та привабливість інтерфейсу.
- Тестування навантаження. Досліджує стабільність роботи системи під час різних рівнів навантаження. Зокрема:
    a) Тестування навантаження. Тестування додатку увиконується в умовах навантаження.
    b) Стресове тестування. Тестування додатку виконується в умова перевантаження.
    c) Тестування стабільноті і надійності. Спрямоване на вивчення поведінки системи в умовах навантаження при тривалому функціонуванні.
    d) Об’ємне тестуваня. Виконується для оцінки поведінки системи за умови збільшення обсягу даних.

3. Види тестування, які пов’язані зі змінами:
    - Димне тестування. Такий вид тестування використовується для виявлення явних помилок , які можуть виникнути на початкових етапах розробки прогармного забезпечення.
    - Регресивне тестування. Такий вид тестування проводиться після внесення змін чи виправлень для перевірки, що вони не викликали нових помилок у існуючому функціоналі.
    - Тестування збірки. Такий вид тестування направлений на перевірку відповідності версій програмного продукту критеріям якості, необхідним для початку тестування.
    - Тестування справності. Проводиться для того, щоб підтвердити, що окрема функція працює відповідно до заявлених специфікацій.

- Альфа-тестування. Імітація реальної роботи з системою. Може бути проведена як разробниками, так і потенційними користувачами.
- Бета-тестування. Розширенній тип альфа-тестування, який проводится із більшою кількістю користувачів.

1.2.3 Рівні тестування

Програмне забезпечення у наші часи використовує модульну структуру що призвело до появи різних рівнів тестування:

1. Модульне тестування перевіряє окремі частини програмного забезпечення, які можна протестувати незалежно завдяки їхній автономії.
2. Інтеграційне тестування спрямоване на оцінку коректності взаємодії між різними компонентами програми.
3. Системне тестування досліджує функціональні та нефункціональне характеристики програми як цілісної системи.
4. Приймальне тестуваннявиконується на фінальному етапі розробки для підтвердження того, що система відповідає вимогам. Приймальне тестування виконується згідно з планом приймальних робіт [3].

1.3 Принципи тестування

Існують принципи тестування, які є загальною інструкцією для тестування в цілому:

1. Тестування показує наявність дефектів, але не їх відсутність.
2. Вичерпне тестування неможливе.
3. Чим раніше розпочато тестування, тим дешевше та ефективніше виявляються дефекти.

4. 80% дефектів знаходиться у 20% модулів.
5. Парадокс пестициду. Якщо використовувати одні і ті тестові випадки, то вони перестають виявляти нові дефекти.
6. Види та методи тестування залежать від контексту завдання.
7. Якщо система відповідає специфікація замовника, це не означає що вона відповідає потребам користувачів [4].

1.4 Висновки за розділом

Основна мета тестування програмного забезпечення- впевнетися що програмне забезпечення працює згідно вимогам. Цей процес охоплює тестування різними видами тестування, залежно від застосунку який потрібно протестувані.

Є два методи тестування: ручне тестування та автоматизоване. Ручне тестування вимагає більше часу на тестування оскільки виконання тестових сценарії проводится вручну. Автоматизоване тестування використовую різні програмні засоби для автоматичного виконання тестів. Однак автоматизація всіх видів тестування неможлива, тому рунче тестування все ще необхідне.

Існують два основі види тестування: функціональне та нефункціональне. Окрім цього є види тестування, які пов’язані зі змінами. Для кожного застосунку використовуються різні види тестування.

Також існує декілька рівнів тестування: модульне, інтеграційне, системне та приймальне тестування.

Окрім цього існують сім принципів тетсування, які є основоположними для забезпечення якості та ефективності процесу тестування.

2 ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз вимог

Це перша і фундаментальна фаза STLC. На цьому етапі команда тестувальників отримує вимоги до програмного забезпечення котрі визначають специфіку продукту [5].

Основа мета цього кроку- точно визначити, що перевіряти та за якими стандартами. Найкращий час для цього- паралельно з фазою збору вимог у процесі розробки програмного забезпечення. Також під час цієї фази команди з тестування зустрічаються із зацікавленими сторонами для прояснення будь яких неоднозначностей у вимогах.

Тестування не етапі аналізу вимог проводятся, щоб переконатися що вимоги правильно інтерпретовані, зрозумілі та послідовні. Така діяльність включає:

1. Аналіз вимог тестування.
2. Визначення вимог до процесу тестування.
3. Визначення видів тестування та типів тестів.
4. Визначення пріоритетів та дедлайнів.
5. Підготовка матриці трасировки.
6. Підготовка тестового середовища.

2.2 Планування тестування

Ключовим етапом життєвого циклу тестування програмного забезпечення є планування, на якому визначаються всі аспекти процесу тестування. На цьому етапі керівник тестування разом із командою оцінюють необхідні ресурси, пронозують зусилля та визначають вартість проведення тестування. Планування розпочинається після завершення етапу збору вимог

Основні завдання, які виконуються на етапі плануваття тестування:

1. Підготовка плану тестування.
2. Вибір тестових інструментів.
3. Оцінка часу та витрат, необхідних для тестування.
4. Планування ресурсів.
5. Визначення ролей і обов’язків у команді тестувальників.
6. Розгляд та затвердження плану тестування.

Наприкінці цього етапу команда тестування розробляж детальний план заходів, які необхідно виконати. Водночас вони повинні чітко розуміти цілі, обсяг і очікувані результати тестування. Це забезпечує ефективну організацію процесу тестування та дозволяє команді досягти високої якості результатів.

2.3 розробка тестових випадків

Далі група тестувальників розробляє тестові випадки. Це набори даних, які використовуються для перевірки того, що система або функція працює належним чином.

Кожен тест базується на вимогах, зібраних на першому кроці. Іноді команди використовують шаблони тестів для швидкості та послідовності. Якщо шаблону немає, тестувальники пишуть докладні тестові випадки.

Тестовий випадок включає:

1. Ідентифікатор тестового випадку: унікальник номер, який часто автоматично генерується інструментом відстеження дефектів.
2. Опис функції: короткий опис того, що функція має виконувати.
3. Очікуваний результат: що програма потрібна робити.
4. Етапи перевірки: послідувність дій під час тестування.
5. Реальний результат: опис того, що вийшло під час тестування. Якщо вдалось знайти помилку, її фіксують та передають розробнику для виправлення.

2.4 Налаштування тестового середовища

Наступним етапом підготовки до тестування програмного забезпечення є налаштування середовища. Це включає в себе підготовку апаратних і програмних умов, за яких ви будете виконувати тести.

Для цього команда тестувальників виконує такі завдання:

1. Налаштування апаратного забезпечення.
2. Налаштування програмного забезпечення.
3. Димний тест на готовність тестового середовища
4. Налаштування тестових даних.

Важливо, щоб при налаштуванні тестового середовища використовувались ті самі засоби, які і при розробці програмного забезпечення.

Наприклад, якщо під час розробки продукту використовувалась мова програмування Python версії 3.13.0 із додатковими бібліотеками, команді тестувальників потрібно звернутись до розробників для виялвення на якій саме версії мови програмування та бібліотек розроблявся продукт.

2.5 Виконання тестів

Після всіх налаштувань починають проводити тести.

На цьому етапі проводиться така діяльність:

1. Виконання тестових випадків
2. Порівняння фактичних результатів з очікуваними
3. Перевірка того, що програмне забезпення працює належним чином
4. У разі знайденої помилки, фіксують її та відправляють команді розробників із вказівками щодо повторення помилки.

Зазвичай команда розробників виправляє помилку та повертає її на тестування. Таку помилку потрібно буде додатково протестувати щоб переконатися, що помилка виправлена без виникнення нових проблем. Ця фаза триває, доки не будуть завершені всі заплановані тести та не вирішено критичні проблеми.

Оскільки цей процес циклічного повторення тестів може бути дуже виснажливим, команди часто використовують засобі автоматизації тестування .

2.6 завершення циклу тестувань

Після того, як продукт було випробувано та визнано готовим до виробництва, цикл тестування формально завершується.

На цьому етапі генеруються звіти із тестування для замовника. проводится оцінка:

1. результатів тестування;
2. витрачений час;
3. відсоток знайдених помилок;
4. кількість знайдених помилок;
5. позитивні результати тестування;

2.7 Висновки за розділом 2

У даному розділі був розглянутий життєвий цикл тестування- STLC.

Основним етапом у життєвому циклі тестування можна назвати планування тестування. У цьому етапі тестування команда тестувальників підготовлює відповідні вимоги як до самій команді тестувальників, так і до їх винагороди за проведення створення та проведення тестових випадків.

3 ОСНОВНІ РІШЕННЯ ЩОДО ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Вибір мови програмування та бібліотек для розробки системи

Вибір мов програмування є важливим етапом розробки застосунківЮ оскільки саме від цього залежить функціональість, масштабованість і продуктивність системи.

Для створення front-end будуть використані HTML, CSS та Javascript.

HTML- це стандартна мова розмітки, яка використовується для створення та структурування веб-сторінок. Вона визначає, як відображати контент на веб-сторінкаї, текст, зорбраження та інші елементи. HTML є основою більшості веб-сайтів [6].

CSS- це форма стилів, яка використовується для оформлення та дизайну веб-сторінок, створених за допомогою HTML [7].

Javascript- це високорівнена мова програмування, яка використовується для додавання динамічнності, інтерактивності та функціональності веб-сторінок [8].

Для створення back-end існує багато мов програмування такі як Python, Java, Ruby та інші. Кожна з цих мов програмування має свої переваги при їх використанні для створення бекенду які можна побачити у табл. 3.2.

Табл. 3.2

Python	Ruby	Java
Має легкий і зрозумілий синтаксис	Фокусується на зручності для програміста, що робить код прости і зрозумілим	Має високу продуктивність для систем із великим навантаженням
Найпопулярніші фреймворки для бекенду- Flask та Django	Найпопулярніший фреймворк для бекенду- Ruby on Rail	Можна зробити бекенд із мінімальним набором бібліотек

Закінчення табл. 3.2

1	2	3
Підтримує багато бібліотек для роботи з даними, машинним навчанням тощо	Має багато інтегрованих функцій таких як маршрутизація, системаперевірки тощо	Великий вибір фреймворків які забезпечать потужний набір інструментів для розробки бекенду
Працює на всіх основних операційних системах, що робить його універсальним	Забезпечує високий рівень гнучкості та підтримує метапрограмування	Має вбудовані механізми захисту, що робить її надійною для корпоративних систем

Для створення бекенд було вирішено вибрати мову програмування Python, оскільки вона лега у розробці та інтеграції фреймворків і бібліотек.

Окрім вибору мови програмування також потрібно вибрати фреймворк. На Python найпопулярнішими є Flask та Django. Кожен з цих фреймоврік має свої особлівості які можна побачити у табл. 3.3 [9].

Таблиця 3.3

Flask	Django
Підходить для невеликих проєктів	Підходить для великих проєктів з великою кількістю функцій
Надає базові функціх для створення веб-додатків	Має повний набір інструментів і компонентів “З коробки”
Потрібно вибирати і інтегрувати сторонні інструменти	Швидка розробка завдяки наявності багатьох готових інструментів
Немає вбудованих бібліотек ORM	Використовує вбудований ORM, який дозволяє працювати з базами даних

Мною було прийнято рішення використовувати FLASK, оскільки він більше підходить для створення невеликих проєктів та легкість у інтегруванні сторонніх бібліотек .

Окрім цього, також було вибрано фреймворк Openai API яка дозволить нам яка дозволить нам підключити велику мовну модель ChatGPT. За допомогою цього фреймворку можна буде створити застосунок, який допоможе нам із автоматизоцією деяких процесів у тестуванні що дозволить краще розподіляти ресурси тестувальника [10].

3.2 Створення Front-end

Інтерфейс у веб-додатку орієнтований на інтерактивний аналіз завантаженого коду, надаючи різні способи перевірки зручним способом.

Є форма для завантаження файлів у яку потрібно буде завантажувати код для аналізу. Після завантаження файлу також потрібно вибрати спосіб аналізу. Для цього передбачено кілька кнопок для різних видів аналізу:

- Перевірка на помилки.
- Пояснення коду.
- Перевірка на синтаскичні помилки.
- Перевірка на помилки обробки даних.
- Перевірка на помилки взаємодії.

Кожна кнопка викликає функцію analyze, яка надсилає запит до сервера. Ця функція асинхронно надсилає файл та вибраний тип аналізу на сервер. Оскільки при отриманні результату, текст відображається у незручний для читання вид, було додане фаорматування яке буде покращувати вивід результату. Також зона результату має прокрутку, якщо вміст аналізу перевищує її висоту.

Додавання або заміна кнопок була реалізована так, щоб це було просто та швидко зробити оскільки для тестувальників потрібно швидкість. Для того

щоб додати нову кнопку потрібно скопіювати минулу строку та в атрибуті onclick замінити назву для аналізу. Також можна змінити назву кнопки, але це не впливатиме на роботу кнопки.

Приклад коду з кнопками можна побачити на рис. 1.

```
<form id="uploadForm" method="post" enctype="multipart/form-data">
    <input type="file" name="file" required>
    <button type="button" onclick="analyze('review')">Перевірка на помилки</button>
    <button type="button" onclick="analyze('explain')">Пояснення коду</button>
    <button type="button" onclick="analyze('check_syntax')">Перевірка на синтаксичні помилки </button>
    <button type="button" onclick="analyze('check_data')">Перевірка на помилки обробки даних </button>
    <button type="button" onclick="analyze('check_interaction')">Перевірка на помилки взаємодії</button>
</form>
```

Рис. 3.1- форма із кнопками для аналізу

За допомогою CSS було розроблено вуло розроблено візуальну частину, окільки без неї веб-додаток виглядає не достатньо гарно для користувача.

Вигляд веб-додатку можна побачити на рис. 2.

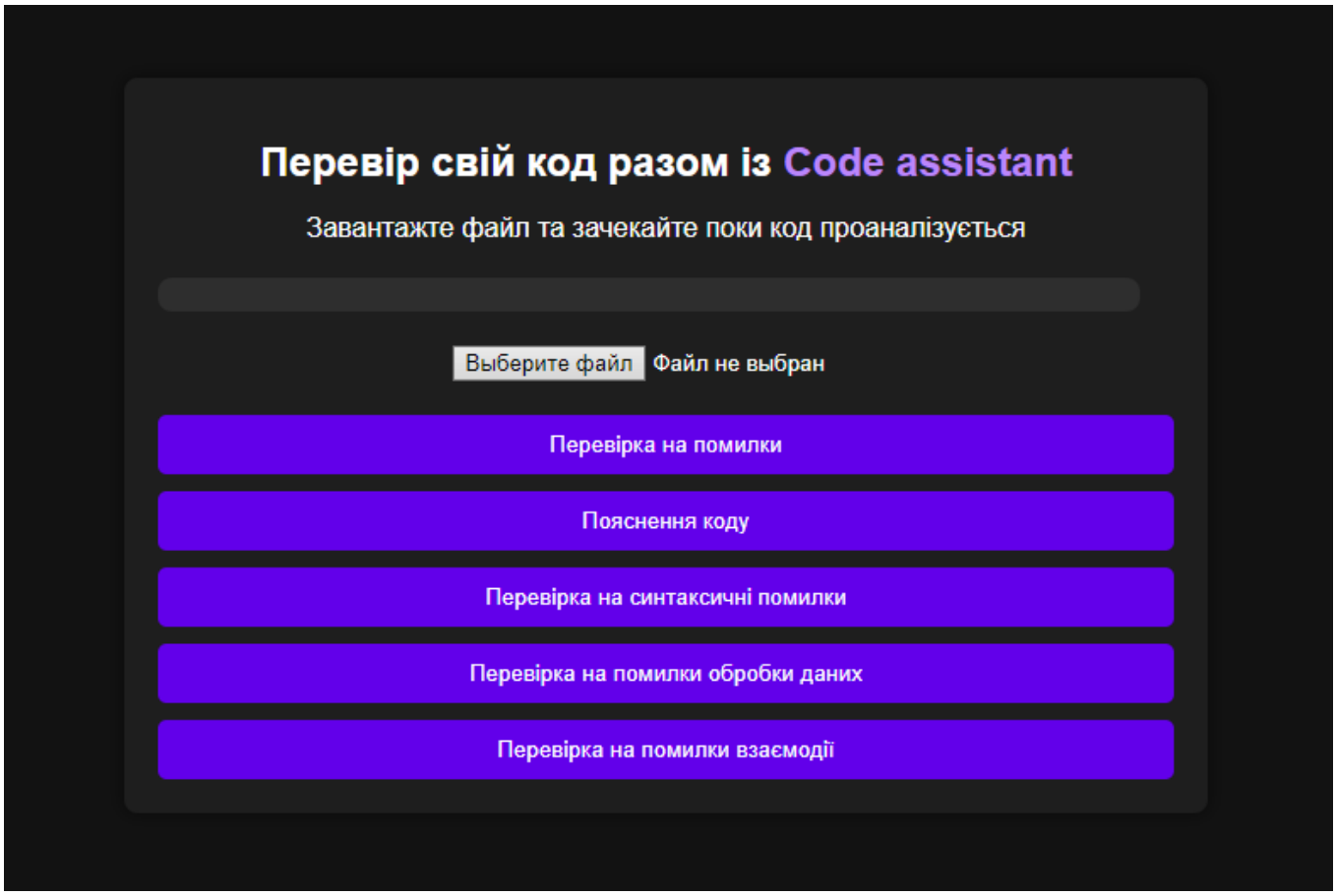

Рис. 3.2- вигляд веб-додатку

На рис.3 можна побачити роботу прокрути. Оскільки результати аналізу досить часто перевищує висоту зони результату, її створення має велику значимість для веб-додатку.

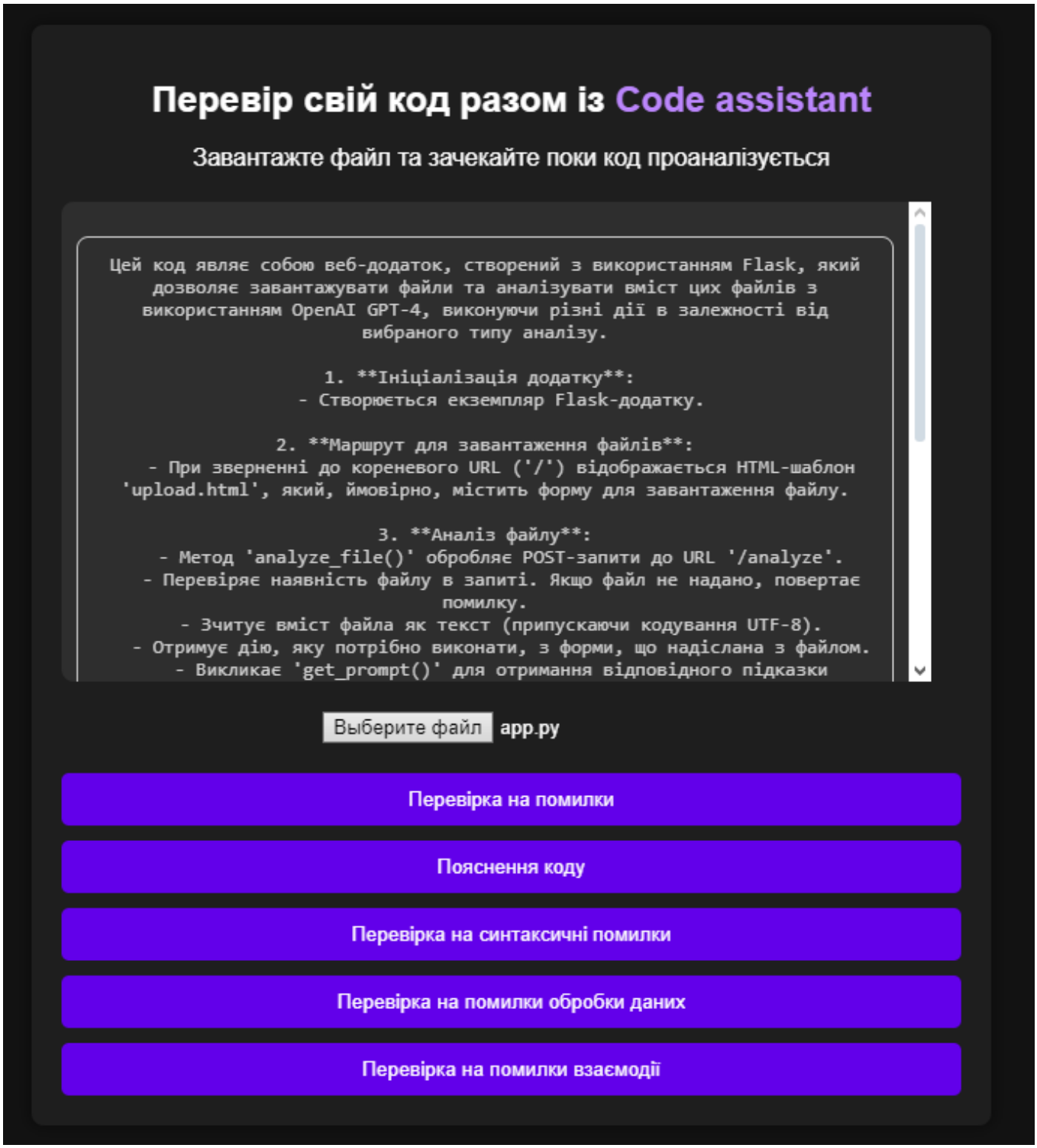

Рис. 3.3- результат аналізу веб-додатка

3.3 Створення back-end

Для створення бекенду спочатку потрібно завантажити та підключити фреймворк FLASK. Також потрібно підключити бібліотеку requests яка використовується для надсилання HTTP- запитів до API OpeanAI. І останнє що потрібно, це підключити HTML файл із створеним фронтендом.

Далі потрібно налаштувати два маршрути. Перший маршрут і буде підключати HTML файл. Другий маршрут буде обробником аналізу. Цей обробник виконує декілька функцій:

- Отримує завантажений файл із запиту.
- Перевіряє, чи файл присутній та чи вибрано дію.
- Зчитує вміст файлу.
- обирає відповідний запит залежно від дії.
- Надсилає вміст файлу разо із запитом до ChatGOT
- Повертає результат аналізу.

Також додана функція у якій знаходятся prompt які відсилаються разом із вмістом файлу на аналіз. Для того, щоб prompt працював, потрібно щоб співпадав атрибут front-end та ключ у словнику.

Приклад функції у якій розташовані prompt можна побачити на рис. 4.

```
def get_prompt(action):
    prompts = {
        "review": "Перевір на наявність помилок. Відповідь дай українською мовою.",
        "explain": "Поясни ТІЛЬКИ як працює код. Відповідь дай українською мовою..",
        "check_syntax": "Перевір код  на наявність ТІЛЬКИ синтаксичних помилок. Якщо зн
        "check_data": "Перевір код на наявність ТІЛЬКИ помилок обробки даних. Якщо знай
        "check_interaction": "Перевір код на наявність ТІЛЬКИ помилок взаємодії. Відпов

    }
    return prompts.get(action)
```

Рис. 3.4- функція із prompt

Також є функція, яка під’єднує Openai API до додатку. У цій функції потрібно визначити ключ API користовуча та модель за допомогою якої буде проводитись аналіз.

На рис. 5 можна побачити веб-додаток, на якому можна отримати власний API key.

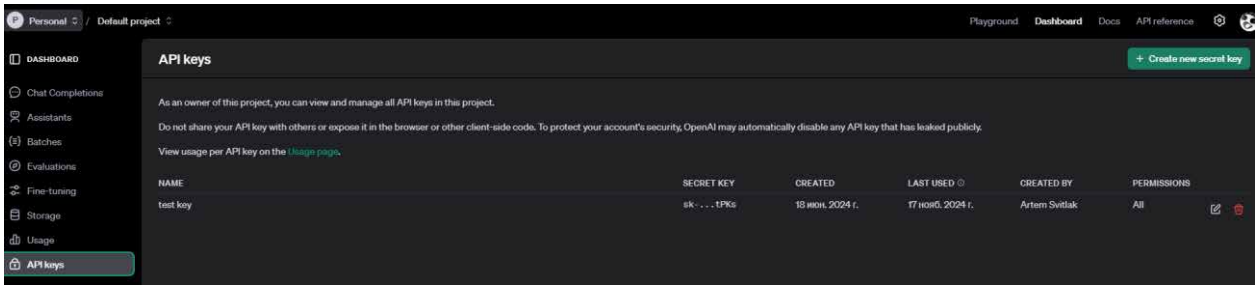

Рис. 3.5- місце розташування ключа

Для того, щоб отримати ключ потрібно зареєструватись на сайті OpenAI та у відповідній вкладці створити власний ключ.

Далі у функції потрібно буде створити форму для запиту у якій буде вміст файлу та promt. Після обробки на повернення результату аналізу повертається у форматі JSON.

3.4 Робота веб-додатку

Після того, як завершили створення фронтенду та бекенду потрібно буде запустити файл. Якщо при компілюванні коду проблем не було виявлено, у консоль буде виведено веб адресу на якій буде працювати веб-додаток.

На рис. 6 можна побачити коректний запуск веб додатку.

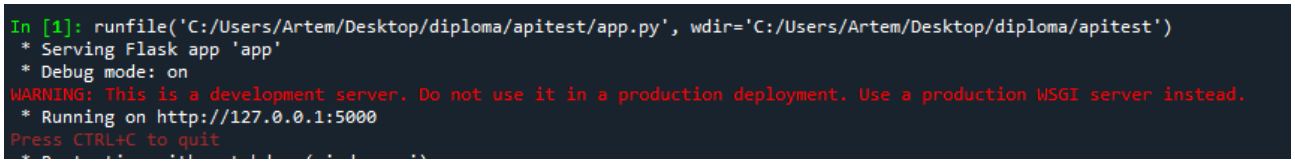

Рис. 3.6- запуск веб-додатку

Після вдалого запуску веб-додатку потрібно буде скопіювати та вставити веб-адресу яка була представлена у консолі.

Вигляд роботи веб-додатку у браузері можна побачити на рис. 7.

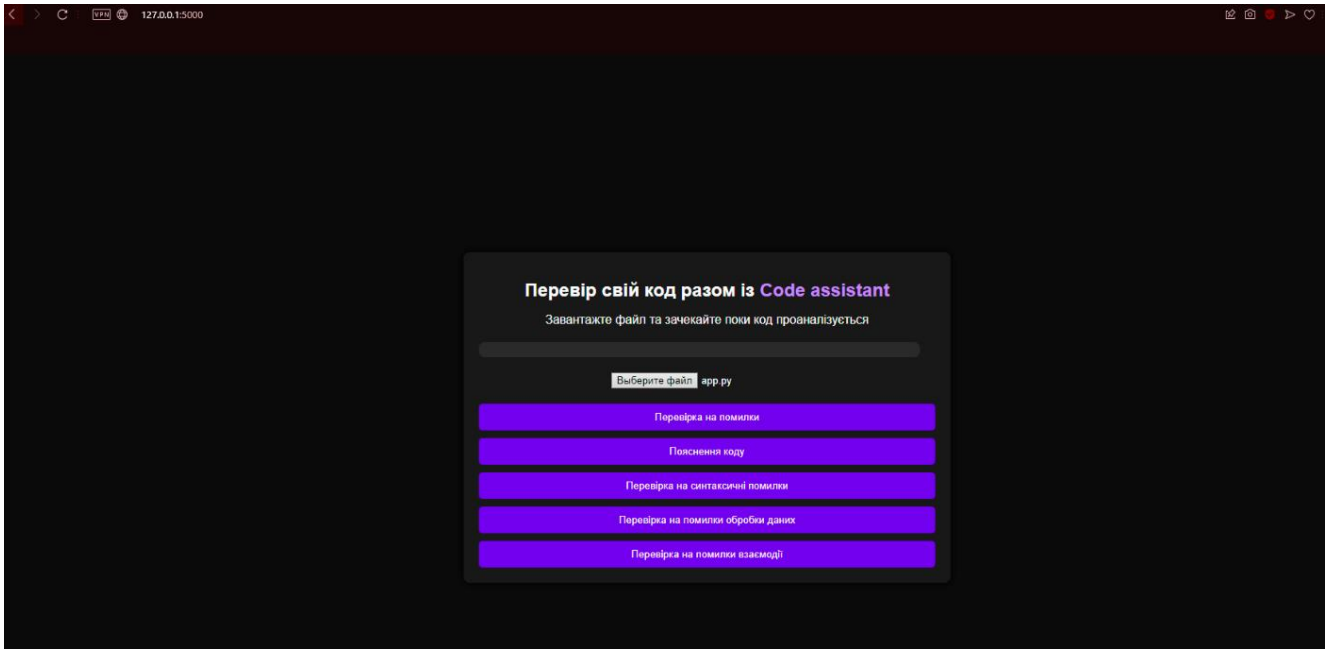

Рис. 3.7 - запущений веб-додаток

Тепер, єдине що залишилось щоб упевнитись у роботі веб-додатку це вибрати файл і відправити його на аналіз. Для цього я завантажив безкошовний шаблон і вибравши кнопку “Пояснення коду” отримав результат, завдяки якому не витрачаючи час на аналіз коду можу зрозуміти базовий принцип його роботи який представлений на рис. 8.

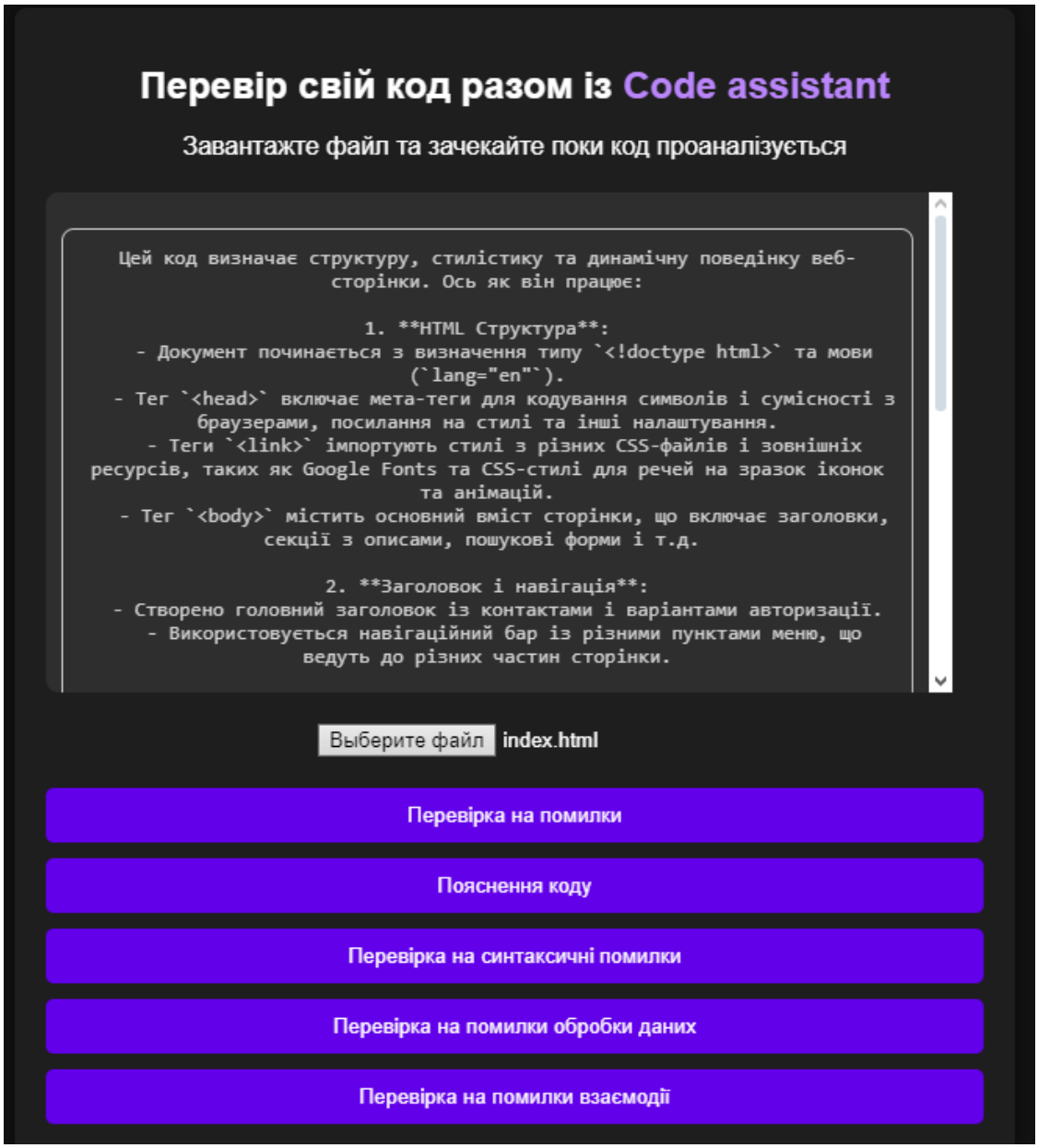

Рис. 3.8- результати роботи веб-додатка

Також, на рис. 9 можна побачити діаграму послідовності роботи веб-додатку

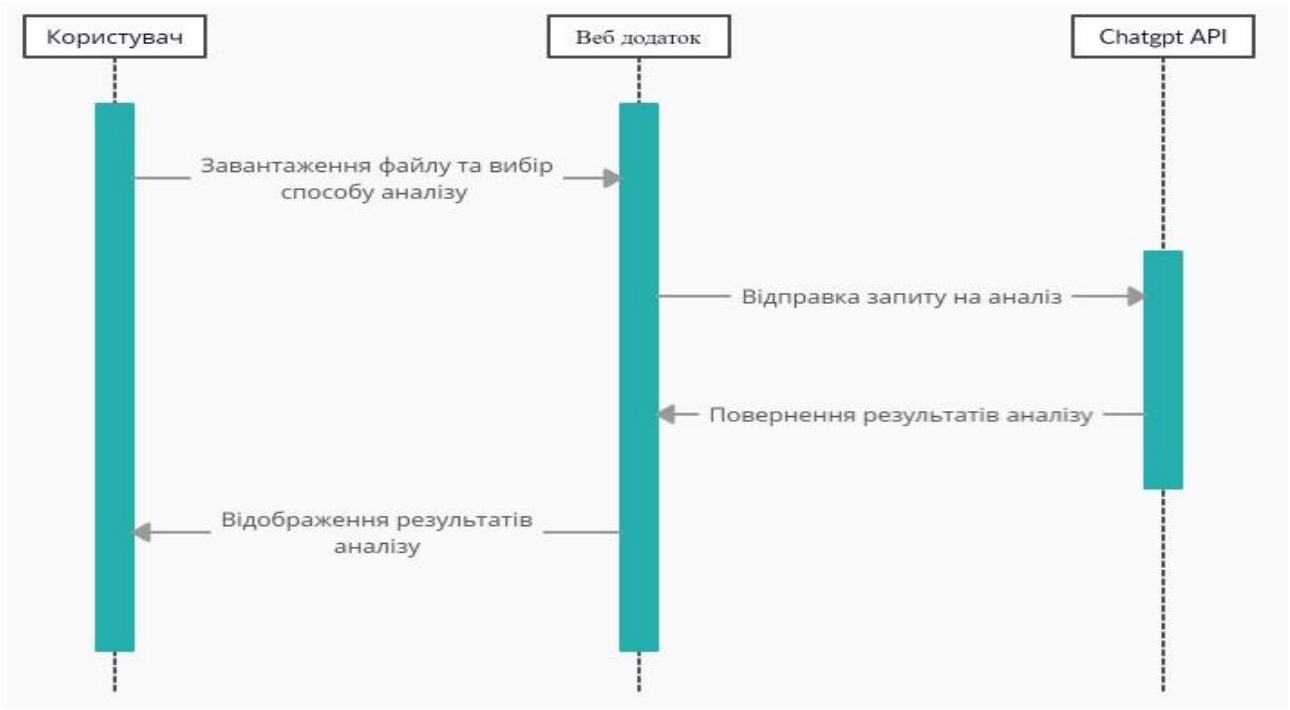

Рис. 3.9- діаграма послідовності веб-додатку

Спочатку користувач завантажує файл та вибирає спосіб аналізу. До кожної кнопки буде прив’язаний свій промпт який буде відправлятися разом із файлом на аналіз. Після аналізу результат буде повернений та відображений у веб-додатку.

3.5 Переваги та недоліки веб-додатку

Через простоту зміни або додавань нових кнопок, цей веб-додаток буде ефективним та гнучким інструментом який буде швидко виконувати свою роботу.

Використання великої мовної моделі дозволить швидко аналізувати та повертати результати аналізу користувачу через що користовач зможе швидко пристосовуватись до змін у коді додатку, який потрібно буде протестувати.

Також через постановки веб-додатку на сервер користувачу не потрібно буде зупиняти роботу застосунку оскільки при зміні веб-додаток сам буде перезавантажуватись.

Веб-додаток не було поставлено на сервер і він працює на сервері розробки. Це означає, що працювати із цим веб-додатком можна буде лише на комп’ютері із запущеним веб-додатком.

Також перевагою можна назвати відсутність бази даних. Оскільки це інструмент який допомагає із роботою, а не замінює важливий компонент у розробці, додавання бази даних для цього веб-додатку непотрібне. Для створення та оперування тестовими випадками а також фіксування помилок та дефектів існують корпоративні застосунки які більше підходять для цього. Також якщо базу даних такого веб-додатку взломлять, це поставить під загрозу інші веб-додатки або застосунки які були протестовані цим веб-сайтом. І останнє, база даних не була додана через можливість дефектів у великим мовних моделях. Такі моделі зазвичай запам’ятовують минулі запити і навіть якщо нові запити не мають нічого спільного із старими, вони все ще будуть псуватися. Отже, якщо випадково буде витік даних із цієї бази даних, результати буду псуватися через що, якщо аналіз буде проводити тетсувальник, який погаю вміє аналізувати інформацію, це призведе до хибних результатів.

Через це, додавання бази даних до такого застосунку буде більш шкідливим ніж його відсутнітсть.

3.6 Висновки за розділом 3

У даному розділі було розглянуто створення веб-додатку.

Для створення front-end було використано HTML, CSS та Javascript.

Для створення Back-end було використано Python, FLASK та OpenAI API.

Були розглянуті етапи розробки веб-додатку а також продемонстровано його роботу.

Також були обґрунтовані переваги та недоліки веб-додатку. Отдразу перевагою та недоліком веб-додатка можна назвати відсутність бази даних. У деяких випадках, існування бази даних може бути корисною, однак через ризик

для витоку даних та можливості псування варіантів аналізу від великої мовної моделі,

4 МОЖЛИВІСТЬ ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Використання великих мовних моделей для тестування

Великі мовні моделі(LLM)- це найрозповсюдженіша модель штучного інтелекту, оскільки вона булай найпершою представлена широкій публіці. Цей тип штучного інтелекту може розпізнавати та генерувати текст через що являється корисним інструментом який і був використаний у створенні веб-застосунку [11].

Однак, у цього типу штучного інтелекту є свої проблеми через які ефективно використовувати його у тестуванні програмного забезпечення можно лише у деяких типах тестування. Головною його проблемою являється спосіб навчання. Такий вид штучного інтелекту навчається на великий обсягах даних які знаходяться у відкритому доступі. Великі мовні моделі беруть великі обсяги даних та створюють щось “середнє”. Через це ефективно генерувати код цей штучний інтелект не здатен. Також, через складніть регулювання коректних даних, які використовуються для навчання штучного інтелекту, створюється така проблема: штучний інтелект навчається на сгенерованих штучним інтелектом даних, які і без того погані. Очевидно, якщо штучний інтелект навчається на своїх же “середніх” даних, такі моделі із часом стануть генерувати все гірший варіант відповіді. Через це використовувати великі мовні моделі для генеруваття коду, навіть невелику створює ризик для продукту.

Але такі моделі можна викорстовувати для виконання рутинних задач, таких як створення тестових випадків або отримання опису роботи коду. Такі результати не будуть втручатися у розробку коду і у випадку некоректності відповіді можна легко замінити.

Для великих мовних моделей у інших областях тестування потрібно спочатку перевіряти коректність роботи цього штучного інтелекту на наборах

даних, однак це ще не означає що при правильності відповідей, всі наступні відповіді будуть теж коректними. Для цього краще навчати власні моделі.

Також несуттєвою проблемою можна назвати те, що при тривалому використанні такого штучного інтелекту відповідь стає все гірше. Це пов’язано із тим, що штучний інтелект запам’ятовює минулі відповіді та при новому запиту може їх використати, навіть якщо теми не пов’язані між собою. Але таку проблему можна вирішити створивши нове діалогове вікно.

4.2 Використання генеративного штучного інтелекту зображень для тестування

Генеративний штучний інтелект зображень це той тип штучного інтелекту, який за допомогою вказівок створює зображення. Також цей тип штучного інтелект один із найвідоміших, оскільки зазвичай розробники таких видів штучного інтелекту дають можливость безкоштовного генеруват изображення.

Він може бути корисний для веб-дизайнерів, однак проблемою є те, що такий штучний інтелект використовує “свою” мову, через що після сгенерованого зображення веб-дизайнерам потрібно буде самому дороблювати вид веб-додатку.

Для використання у тестуванні такий штучний інтелект не підходить через специфіку роботи самого штучного інтелекту.

4.3 Використання інших видів штучного інтелекту для тестування

Окрім використання великих мовних моделей для тестування, можливо використати машинне навчання для створення та навчання власної моделі. Такі моделі краще робити вузьконаправленими для зменшення вірогідності виникнення помилок під час тестування. Цей вид штучного інтелекту може

допомогти у більшої кількості видів тестування, ніж великі мовні інтелекти. Єдиною проблемою може бути пошук даних для навчання такої моделі, однак команди розробників можуть для цього використовувати власні результати тестування.

Також у тестуванні може допомогти комп’ютерне бачення. Великі мовні моделі також можуть оброблювати інформацію із зображень, але використання спеціально навчаного штучного інтелекту може бути ефективніше. Такий вид штучного інтелекту може бути корисним у тестуванні користувальницького інтерфейсу.

4.3 Висновки за розділом

У даному розділі було розглянуто можливість використання різних видів штучного інтелекту для допомоги у тестуванні.

Окрім великих мовних моделей, у тестуванні може допомогти комп’ютерний зір або можна натренерувати власну модель за допомогою машинного навчання.

Генеративний штучний інтелект зображень може допомогти із розробкою веб-додатку, але не із його тестуванням.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи, проведено детальне ознайомлення із предметною областю – «Автоматизована система підтримки прийняття рішень при тестуванні програмного забезпечення».

Під час роботи із першим розділом було розглянуто методи, види, типи, рівні тестування та принципи тетсування. Існують два методи тестування: ручне та автоматизоване тестування. Під час кожного сетоду тестування використовуються свої види та типи тестування. Принципи тестування являють собою загальну інструкцію із тестування.

У другому розділі розглянуто життєвий цикл тестування програмного забезпечення. Життєвий цикл тестування складається із шести етапів: аналіз вимог, планування тестування, розробка тестових випадків, налаштування тестового середовища, виконання тестів і завершення циклу тестувань.

У третьому розділі було описано створення веб-застосунку. Було розглянуто процес створення веб-застосунку та технічні рішення які використовувались під час розробки застосунку.

У четвертому розділі було розглянуто можливість використання штучного інтелекту у тестуванні. Великі мовні моделі можуть бути корисні лише у деяких типах тестування. Для інших типів тестування потрібно буде навчати власну модель за допомогою машинного навчання, а також можливе використання комп’ютерного зору у тестуванні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке тестування ПЗ, його етапи, види, інструменти: веб-сайт. https://university.sigma.software/what-is-software-testing/

2. Ручне та автоматизоване тестування: веб-сайт. https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya/

3. Авраменко В.С., Авраменко А.С., Косенюк Г.В. Тестування програмного забезпечення: навчальний посібник. Черкаси, 2017, 284 с.

4. Принципи тестування: веб-сайт. https://qalight.ua/baza-znaniy/printsipi-testuvannya/

5. Software Testing Life Cycle (STLC): веб-сайт. https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/

6. HTML basics: веб-сайт. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

7. What is CSS: веб-сайт. https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

8. What is Javascript: веб-сайт. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

9. Differences Between Django vs Flask: веб-сайт. https://www.geeksforgeeks.org/differences-between-django-vs-flask/

10. OpenAi developer platform: веб-сайт. https://platform.openai.com/docs/overview

11. What is large language model(LLM): веб-сайт https://www.cloudflare.com/ru-ru/learning/ai/what-is-large-language-model/

12. Types of AI Explained: from narrow to super AI: веб-сайт https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/types-of-artificial-intelligence