Державний торговельно-економічний університет

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка системи тестування Web-додатків»

Студентки 4 курсу, 8 групи,
спеціальності
122 «Комп'ютерні науки» _____ *підпис студента* — Буріна Валерія Олегівна

Науковий керівник
кандидат фізико-математичних наук _____ *підпис керівника* — Філімонова Тетяна Олегівна

Гарант освітньої програми
кандидат технічних наук, доцент _____ *підпис керівника* — Демідов Павло Георгійович

Київ 2022

АНОТАЦІЯ

Автоматизація тестування дозволяє значно скоротити витрати компаній-розробників, заощадити час і ресурси, що витрачаються на тестування, знизити ризики випуску на ринок неякісного продукту.

У випускній кваліфікаційній роботі розроблено Web-додаток «Help me» з набором функцій записної книги. Розроблено систему тестування, результатом роботи якої було покрито тестами 85% додатку і протестовано його функціонал.

Розроблена система для тестування Web-додатків готова для використання в реальних проектах.

Ключові слова: Web-додаток, автоматизація тестування, фреймворк, Python.

АНОТАЦІЯ

Automation of testing can significantly reduce the costs of development companies, save time and resources spent on testing, reduce the risks of releasing a poor-quality product to the market.

In the final qualification work, the Web-application «Help me» with a set of functions of the notebook was processed, the result of which was covered by tests of 85% of the application and its functionality was tested.

The developed system for testing Web-applications is ready for use in real projects.

Keywords: Web-application, automation of testing, framework, Python.

ЗМІСТ

ВСТУП

Актуальність роботи. Програмне забезпечення використовується практично у всіх сферах нашого життя, величезні кошти витрачаються на розробку різноманітних програм, що є затребуваними в промисловості і бізнесі, в індустрії розваг, в освіті та медицині. Завдання зниження вартості розробки програмного забезпечення і поліпшення якості продукції, що випускається, є одними з найбільш актуальних в індустрії інформаційних технологій. Автоматизація тестування дозволяє значно скоротити витрати компаній-розробників, заощадити час і ресурси, що витрачаються на тестування, знизити ризики випуску на ринок неякісного продукту. Тому технології автоматизації тестування набирають все більшої популярності серед компаній, пов'язаних з розробкою програмних продуктів.

Мета роботи: обґрунтування та розробка програмного засобу тестування Web-додатків.

Об'єкт дослідження: процес тестування Web-додатків.

Предмет дослідження: технології в системі тестування Web-додатків.

Для досягнення поставленої мети необхідно виконати **наступні завдання.**

1. Аналіз особливостей тестування Web-додатків.
2. Порівняння фреймворків тестування на основі Python.
3. Розробка системи тестування Web-додатків.

Для вирішення поставлених задач використовувалися такі методи:

- загальнонауковий аналітичний метод;
- метод порівняльного аналізу для вибору інструментів для розробки системи тестування;
- методи об'єктно-орієнтованого програмування для розробки системи тестування Web-додатків.

Практичне значення. Розроблена система для тестування Web-додатків готова для використання в реальних проектах.

РОЗДІЛ 1. АНАЛІТИЧНЕ ДОСЛІДЖЕННЯ СПЕЦИФІКИ РОБОТИ WEB-ДОДАТКІВ

1.1 Організація роботи Web-додатків

Під web-додатком розуміється прикладна програма, розроблена з архітектури «клієнт-сервер», що використовує в якості клієнта web-браузер і працює на стороні web-сервера (Рис. 1.1). При цьому взаємодія між клієнтом і сервером виконується з використанням протоколу HTTP [5].

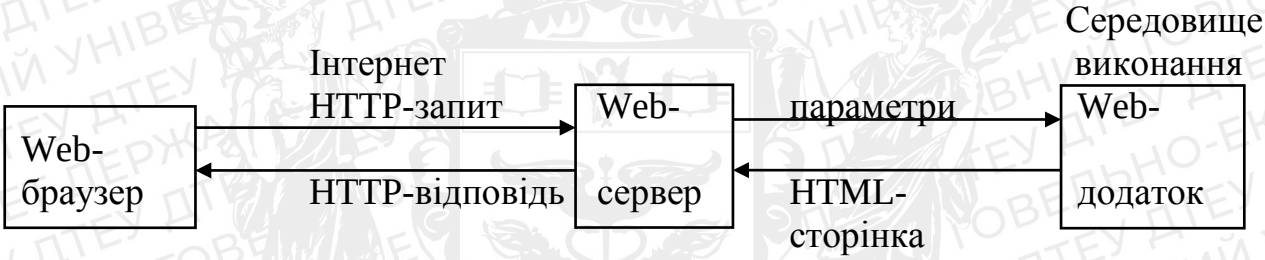


Рис. 1.1 Схема взаємодії користувача з web-додатком

Користувач за допомогою браузера відправляє HTTP-запит за певним URL-адресою, який зіставляється з ресурсами на web-сервері. Якщо даний URL-адреса вказує на динамічний ресурс, то сервер запускає деяку зовнішню програму (web-додаток), яка формує HTML-сторінку, яка може бути показана браузером, і повертає її клієнту. Основною частиною web-додатки є його логіка на стороні web-сервера.

Найчастіше web-додаток не є самостійною програмою, а виконується під управлінням деякої зовнішньої програми - середовища виконання. В якості такої програми можуть виступати: сервер додатків або контейнер додатків. Середовище виконання робить такі дії:

• приймає від web-сервера всю інформацію, пов'язану з HTTP- запитом;

• визначає, яка програма, створене для даного середовища, має бути виконано;

• створює всі допоміжні об'єкти, які можуть знадобитися для роботи цього додатка (наприклад, Application, Session, Request, Response, User і т. П.);

• завантажує запитувана додаток і передає йому управління;

• web-додаток виконується і створює HTML-сторінку, яку записує в спеціально створений для цього об'єкт (зазвичай об'єкт Response) і після цього повертає управління web-серверу;

• web-сервер формує HTTP-відповідь, включає в нього сформовану web-додатком HTML-сторінку і відправляє його клієнту, який надіслав запит до даного web-додатком.

Зазвичай HTML-сторінки, створені web-додатком, включають HTML-форми, з якими працює користувач. При натисканні користувачами на кнопки типу submit браузер відправляє новий HTTP-запит до того ж самому web-іріложенію. Послідовність викликів web- додатки користувачем становить сеанс його роботи. Даний сеанс завершується або в результаті вибору користувачем команди про закінчення його роботи з web-додатком, або в результаті тривалого проміжку часу між відправленнями наступних HTTP-запитів (зазвичай більше 20 хвилин).

Відзначимо, що web-додаток призначений для вирішення деяких завдань користувачів і обов'язково має користувальницький графічний інтерфейс.

Web-додатки у порівнянні з локальними програмами мають ряд наступних переваг [5].

1. Простота доступу до додатка. Будь-яка людина, що має комп'ютер, підключений до мережі Інтернет, може використовувати web-додаток.
2. Простота розгортання. На відміну від локальних додатків web-додатки після завершення розробки не вимагають установки на комп'ютерах користувачів. Досить тільки повідомити їм URL-адресу програми. При зміні додатки все користувачі відразу починають

працювати зі зміненою версією.

3. Наявність великої кількості навчених користувачів.
4. Високий рівень розвитку і надійності мережевих з'єднань і web-технологій.

Web-додатки мають наступні недоліки.

1. Відсутність підтримки стану сеансу роботи і затримка при перезавантаженні кожної сторінки. Кожна перевантаження (або оновлення сторінки) викликає помітну затримку, викликану необхідністю встановити HTTP- з'єднання, обробити запит на сервері, передати по мережі у відповідь HTTP-повідомлення і перезавантажити сторінку браузером. Це створює скачки та періодичною очисткою роботи користувача.
2. Обмежений набір елементів управління для проектування форм додатки. Поточна версія мови HTML підтримує тільки обмежений набір елементів управління (текстові елементи, радиокнопки, прапорці, списки, що випадають і командні кнопки).
3. Неузгоджені підходи до оформлення та способу взаємодії з користувачами. Хоча web-мережа надає розробникам і дизайнерам значну гнучкість і можливість проявити свої творчі здібності, що виходять в результаті цього розмаїття і неузгодженість призначених для користувача інтерфейсів і способів взаємодії з web-додатками часто створюють проблеми для користувачів. Це викликано тим, що користувачі стикаються з великою різноманітністю стилів візуальних інтерфейсів і способів взаємодії, кожен з яких пропонує свій словник об'єктів, дій і візуальних уявлень, змішаних в одному і тому ж додатку.

Для вирішення таких проблем проектування і пов'язаних з ними проблем використання багато компаній розробляють керівництва з проектування користувальницького інтерфейсу і стилю для впорядкування оформлення та роботи web-додатків.

4. Відкритість доступу до додатків. У зв'язку з відкритим доступом до web-додатків практично для всіх web-додатків потрібно підтримувати безпеку роботи користувачів з ними.

Процес розробки може бути розділений на наступні складові:

1. розуміння проблеми;
2. планування рішення;
3. виконання плану;
4. перевірка точності отриманого результату;
5. доопрацювання з метою видалення можливих помилок або неточностей.

Розглянемо основні етапи розробки локальних додатків.

У процесі розробки будь-якого програмного забезпечення можна виділити наступні основні види діяльності:

1. Визначення вимог до програмного забезпечення. Інжиніринг вимог призначений для розуміння розв'язуваної проблеми.
2. Проектування: призначене для планування вирішення проблеми.
3. Реалізація: перетворення плану в працюючий програмний код.
4. Перевірка і оцінка якості: призначене для виявлення помилок програмного коду або невідповідностей між певними вимогами і їх реалізацією.
5. Розгортання: надання користувачам можливості працювати зі створеним програмним забезпеченням.
6. Підтримка: призначена для відстеження використання діючої системи і збереження її працездатності.
7. Розвиток: призначений для поліпшення з часом розробленого рішення; надання нових вхідних даних для процесу розробки в формі нових вимог.

Розглянемо докладно етапи розробки програмного забезпечення.

Інжиніринг вимог призначений для розуміння необхідних можливостей і характеристик створюваного програмного продукту.

Даний аналіз спрямований на визначення функціональних вимог (які функції система повинна виконувати) і не функціональних вимог (якість пропонованого рішення). Інжиніринг вимог також передбачає виявлення загальної ідеї, яка стоїть за розробляється системою; основних зацікавлених осіб, яким потрібна нова система, і умови, в яких буде використовуватися система. Виявлені вимоги обробляються з метою створення високорівневих моделей даної системи, яка абстрагується від непотрібних подробиць даної проблемної області.

Проектування призначене для опису рішення, яке повинно відповідати функціональним вимогам і вимогам ефективності, а також обмеженням того середовища, в якій вона буде працювати. Раніше зібрані вимоги уточнюються і поліпшуються, щоб задовольняти можливим технологічним обмеженням. Проектування включає наступні дії:

- проектування схеми даних і класів;
- проектування компонент;
- проектування графічного інтерфейсу;
- проектування архітектури системи.

Допомагає краще сформувати специфічні особливості системи, такі як структура, поведінка, взаємодія, дані і потік управління. Дозволяє розділити сфери відповідальності, основний принцип програмної інженерії - вирішення проблеми шляхом поділу на різні підзадачі - може допомогти впоратися зі складністю і досягти необхідних технічних якостей, таких як адаптованість, простота підтримки, розширюваність і багаторазова використовуванність.

На етапі реалізації проекту розроблені проектні рішення перетворюються у відповідний програмний код. Можуть знадобитися бібліотеки програм, різні мови програмування, різні комунікаційні протоколи і технічні пристрої.

Перевірка і оцінка якості зазвичай проводиться паралельно з реалізацією, тому що правильність і надійність проміжних результатів, а не тільки кінцевого продукту є дуже важливим для гарантування якості всього

програми. Якість в значній мірі пов'язано з наступними критеріями:

- оцінкою функціональності, тобто правильності поведінки щодо заданих функціональних вимог;
- оцінкою продуктивності, так званого часу очікування відгуку програми в звичайних умовах і при пікових навантаженнях;
- оцінкою зручності використання, тобто легкості використання, комунікаційної ефективності та відповідності стандартам використання.

Розгортання додатка надає користувачам можливість використання цього додатка. Залежно від типу додатка процес розгортання може включати:

• встановлення прогрмного забезпечення на комп'ютерах клієнтів;

• встановлення центрального додатка і баз даних на сервері;

• конфігурація проміжного комунікаційного програмного забезпечення;

• інструктування і навчання майбутніх користувачів, особливо, якщо встановлюється зовсім новий додаток, а не нова версія вже наявної програми.

Підтримка розгорнутого і ефектів у програмному забезпеченні означає забезпечення його робочого стану, яке складається в гарантуванні його доступності та зменшенні збоїв. Може включати: періодичну перевірку файлів журналу; звітів про помилки та очищення тимчасових файлів; виправлення помилок; установку виправлень.

Розвиток програми. Додаток призначений для вирішення реальних завдань користувачів. Однак у зв'язку з розвитком організації, ускладненням розв'язуваних завдань, поліпшенням розуміння користувачем можливостей даного програмного забезпечення необхідно розвивати створене програмне забезпечення. Нові потреби користувачів з'являються тільки після тогопевного часу роботи із створеним програмним забезпеченням. Після цього вони починають давати свої пропозиції та коментарі. Поява нових вимог може викликати необхідність запустити весь процес розробки заново. Незважаючи на суворе дотримання рекомендацій правильної організації

процесу розробки програмного забезпечення, часто тільки після розгортання і накопичення деякого досвіду роботи користувачів з програмним забезпеченням стає зрозуміло, що деякі вимоги не були повністю виконані і додаток повинен бути доопрацьовано.

1.2 Особливості тестування Web-додатків

Тестування відіграє життєво важливу роль в процесі розробки і створення якісного програмного забезпечення. Необхідно серйозно ставитися до аналізу і проектування структурованого процесу, який забезпечить своєчасний і успішний випуск проекту.

Важливо пам’ятати, що довіру користувачів дуже просто втратити, а виправити допущені помилки може коштувати дорожче, ніж спочатку провести повну підготовку та тестування.

Етапи тестування web-додатків [1]:

1. *Підготовчий етап та вивчення документації.* В даний етап входить аналіз технічного завдання; вивчення кінцевих макетів; тест кейсів; матриці відповідності (для валідації покриття вимог щодо продукту тестами) і складання плану тестування.

2. Тестування верстки.

Візуальна частина:

- невірне відображення блоків, складових інтерфейсу, нестиковки колірної гами;
- тестування локалізованих версій (переклад сайту);
- відповідність макету (шари у PhotoShop);
- у разі зменшення/збільшення масштабів (75-150%) без візуальних недоліків;
- підсвічування полів з помилками;
- перевірка у дозволах (+ прокрутка);

Доступність і відсутність JS помилок:

- чи натискаються клікабельні елементи (внутрішні/зовнішні посилання, посилання на електронну пошту, кнопки, іконки);
- під час наведення на клікабельні елементи курсор змінюється, в іншому випадку – ні;
- підказки на незрозумілих клікабельних елементах;
- під час відключення зображень повинні бути підписи невеликим сірим кольором (у Web Developer -> Images -> Replace Image With Alt Attributes);
- працездатність при вимкненому JS. Критичні функції повинні бути доступні без JS (в Web Developer -> Disable -> Disable JS -> All JS)

Коректна робота, надійна верстка:

- перевірка роботи з даними (введення великої та малої кількості тексту у форму; блоки з контентом міняються місцями (Firebug (HTML -> Edit)));
- перевірка роботи стилів (введення тексту з заголовками, з абзацом і без, з картинками).

404-і запити:

чи є 404-і помилки (Firefox -> Tools -> Validate links).

3. *Функціональне тестування.* Вид тестування, у якому виявляється некоректна/неправильна робота функціоналу програми.

Необхідними перевірками є:

- коректність роботи головних функцій сайту;
- перехід за посиланнями;
- перевірка користувальницьких форм (валідація полів, обов’язкові/необов’язкові поля, повідомлення про помилки під час неправильного введення, додавання коментарів у блог, зворотний зв’язок);
- пошук і покупка товару, оформлення замовлення;
- звірка переданого замовником контенту з наявним на сайті;
- перевірка можливої авторизації/реєстрації;

- додавання, видалення та редагування даних користувачів, товарів і замовлень.

4. *Ad-hock тестування* – імпровізаційне тестування без підготовки.

Допомагає зрозуміти:

- чи зрозумілим є призначення форм;
- чи відзначені обов'язкові поля та чи всі обов'язкові поля відмічені;
- чи вбудована обов'язкова перевірка заповнених форм;
- чи відбувається перевірка правильності введення контактних даних.

З переваг даного тестування можна виділити:

- достатньо швидке знайомство з системою;
- специфічні несправності;
- масу питань і пропозицій;
- економію часу.

Негативне тестування, яке зазвичай називають тестуванням шляху помилок або тестуванням на збій, це процес застосування якомога більшої кількості креативних підходів та перевірки програми на предмет наявності невірних даних. Його призначення полягає в тому, щоб перевірити, чи показуються помилки користувачеві, де вони можуть бути, або більш витончено обробляти неправильні значення. Проводиться для забезпечення стабільності додатків.

Еквівалентні тести – це тести, які призводять до одного і того ж результату. Група тестів є класом еквівалентності, за умови дотримання наступних умов:

- всі тести написані для виявлення однієї помилки;
- якщо один з тестів виявить помилку, то інші теж її виявлять;
- зворотне теж вірно.

Еквівалентна область – частина області вхідних або вихідних даних, для яких поведінка компонентів або систем, ґрунтуючись на специфікації, вважаються однаковими.

5. *Exploratory testing*, також носить назву інтуїтивного тестування, має на увазі одночасне проєктування, виконання тестів і навчання продукту.
6. *Usability тестування* (User Experience).

Дозволяє перевірити комфортне використання сайту для користувача, наскільки легко знайти необхідну інформацію або виконати бажані дії.

7. *Навігаційне тестування сайту.* Чи всі сторінки, кнопки та поля на них, зрозумілі під час використання, доступ до головної сторінки та меню з усіх інших сторінок можливий, навігація проста та інтуїтивно зрозуміла.
8. *Тестування контенту.* Відсутність граматичних/орфографічних помилок, контент інформативний та структурований, зображення та заголовки мають відповідні розміри і розміщені вірно.
9. *Зручність використання.* Чи зрозуміла структура веб-додатку, яке враження справляє і чи наявні зайві компоненти на сторінках.
10. *Тестування UI* (User Interface). Відповідність стандартам графічних інтерфейсів й елементів дизайну, правильність локалізованих версій, тестування з різними дозволами, на смартфонах і планшетах.
11. *Тестування сумісності* (конфігураційне тестування).

Тип нефункціонального тестування програмного забезпечення, що дозволяє перевірити, чи може програмне забезпечення працювати на іншому обладнанні, операційних системах, додатках, мережевих середовищах або мобільних пристроях.

12. *Кросплатформенне (багатоплатформне) тестування сайту*. Окремі функції можуть мати проблеми з певними операційними системами, тому необхідно перевіряти роботу програми у різних версіях Windows, Unix, Mac, Linux, Solaris і ін.
13. *Кросбраузерні тестування сайту*. Також коректна робота залежить від типу браузера. Верстка повинна бути кросбраузерною, щоб забезпечити однакову візуальну частину, доступність,

функціональність і дизайн у всіх браузерах. Необхідно перевіряти масштабованість, розширюваність, рамки для елементів у фокусі, відсутність JS помилок (лівий нижній кут сторінки). Перевіряти роботу необхідно у таких браузерах, як: Internet Explorer, Firefox, Chrome, Safari, Opera, Edge різних версій.

14. *Перегляд на мобільних пристроях.* Незважаючи на перевірку роботи веб-додатків у різних роздільних здатностях на комп'ютері, найчастіше помилки на мобільних пристроях залишаються непоміченими. Отже, наполегливо рекомендується перевіряти коректне відображення та роботу вашого веб-додатку на мобільних пристроях різних операційних пристроях, а також на планшетах.
15. *Тестування баз даних*. Необхідно перевірити правильність встановлення зв'язку з сервером, перевірити сумісність сервера з програмним забезпеченням, апаратними засобами, базою даних і мережею. Також слід перевірити, що відбувається під час переривання будь-якої дії, під час наступного з'єднання з сервером під час виконання операцій.
16. *Тестування продуктивності.* Методика нефункціонального тестування, для вимірювання таких параметрів системи як чутливість та стабільність, за різних навантажень. Дозволяє досліджувати швидкодію сайту та можливості масштабованості додатку, наприклад, під час додавання нових користувачів. Проводиться з метою з'ясувати, яке навантаження сайт здатний витримати. Тестування продуктивності вимірює атрибути якості системи, такі як масштабованість, надійність і використання ресурсів.
17. *Навантажувальне тестування* – це метод тестування продуктивності, у якому реакція системи вимірюється в різних умовах навантаження. Відповідає за реакцію веб-додатка у разі збільшення робочого навантаження. Навантажувальні випробування

проводяться для нормальних і пікових навантажень (одночасна купівля товару або авторизація на сайті великої кількості користувачів).

Підхід навантажувального тестування:

- Оцінити критерії прийнятності продуктивності;
- Визначити критичні сценарії;
- Модель робочого навантаження;
- Визначити цільові рівні навантаження;
- Дизайн тестів;
- Виконати тести;
- Проаналізуйте результати.

Завдання навантажувального тестування: час відгуку, пропускна здатність, утилізація ресурсів, максимально призначене для користувача навантаження, бізнес-метрики.

Стрес-тестування (Stress Testing) перевіряє систему на її стійкість і обробку помилок в умовах надзвичайно високого навантаження (оцінює як система працює в екстремальних умовах, поза обмеженнями та лімітами). Стрес-тестування проводиться, щоб переконатися, що система не буде аварійно завершувати роботу у критичних ситуаціях.

Тестування стабільності/надійності (Stability/Reliability Testing) – тип тестування програмного забезпечення, який перевіряє, чи може програмне забезпечення працювати безвідмовно протягом певного періоду часу у вказаному середовищі.

18. *Об'ємне тестування* (Volume Testing) – тип тестування програмного забезпечення, яке проводиться для аналізу продуктивності системи за рахунок збільшення обсягу даних у базі даних.
19. *Тестування паралелізму* (Parallel Testing) – тип тестування програмного забезпечення, який перевіряє кілька додатків або підкомпонентів однієї програми одночасно, щоб скоротити час тестування. Під час паралельного тестування тестувальник запускає

дві різні версії програмного забезпечення одночасно з одним і тим самим вводом. Мета полягає в тому, щоб з'ясувати, чи ведуть себе колишня система і нова система однаково або по-різному.

20. *Тестування безпеки.* Спрямоване на оптимізацію безпеки системи під час проєктування, розробки, використання та обслуговування програмних систем і їх інтеграції з критично важливими для безпеки апаратними системами у виробничому середовищі.

Аспекти безпеки програмного забезпечення:

Функціональне програмне забезпечення не повинно створювати небезпек (наприклад: управління сучасним літаком не повинне спрямовувати його в океан).

Системи моніторингу повинні працювати без збоїв (наприклад: резервний комп'ютер повинен запускатися автоматично у разі збою на основному).

Цілі в тестуванні безпеки:

- у складних системах, де задіяні багато взаємодій, критично важливі для безпеки функції повинні бути ідентифіковані та ретельно проаналізовані;
- помилки визначені та усунені;
- кількість критичних помилок підтримується на низькому рівні, щоб уникнути непрацездатності системи;
- атрибути безпеки повинні розглядатися як частина всіх рівнів тестування ПЗ.

Принципи безпеки:

- Конфіденційність (обмеження або надання доступу до інформації).
- Цілісність (можливість відновити дані в повному обсязі у разі їх пошкодження; доступ до зміни інформації тільки певної категорії користувачів).
- Доступність (ієрархія рівнів доступу та їх чітке дотримання).
- Обробка помилок та регресійне тестування.

Після завершення розробки web-додатку слід провести оцінку й аналіз виявлених помилок для подальшого запобігання їх повтору. А також виконати регресійне тестування.

21. *Регресійне тестування.* Використовує техніку тестування чорного ящика (повторне виконання тестів), на які впливають зміни коду. Ці тести повинні виконуватися якомога частіше протягом всього життєвого циклу програмного забезпечення у разі змін коду для виправлення дефектів або для покращення роботи веб-додатків.

1.3. Рівні тестування web-додатків

1. *Приймальне тестування* – вид тестування, що проводиться на етапі здачі готового продукту (або готової частини продукту) замовнику. Метою такого тестування є визначення готовності продукту, що досягається шляхом проходу тестових сценаріїв та випадків, які побудовані на основі специфікації вимог до програмного забезпечення, що розробляється [4].

Результатом приймального тестування може стати:

- Відправка проекту на доопрацювання.
- Ухвалення його замовником, в якості виконаної задачі.

Це фінальний етап тестування продукту перед його релізом. При цьому, він не є дуже ретельним, всеохоплюючим та повним – тестується, зазвичай, тільки основний функціонал.

Приймальне тестування проводитися або самим замовником, або групою тестувальників, що представляють інтереси замовника, або тестувальниками компанії-розробника. Залежить від уподобань компанії-замовника.

2. *Системне тестування* – це тестування програмного забезпечення, що виконується на повній, інтегрованій системі, з метою перевірки відповідності системи вихідним вимогам, як функціональним, так і не функціональним (Рис. 1.2) [4].

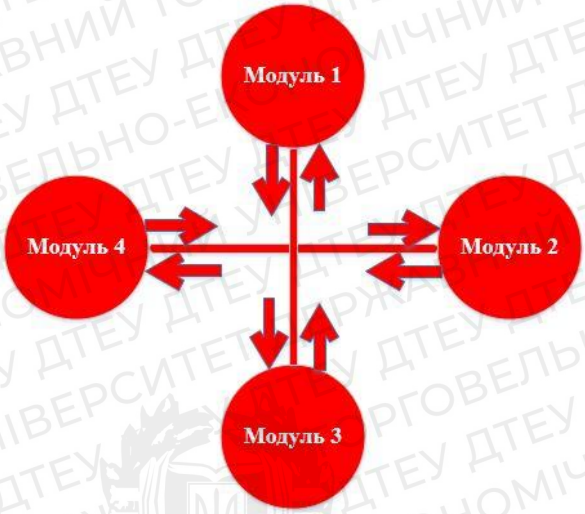


Рис. 1.2 Схема системного тестування

Об’єкт тестування виділений червоним кольором.

Виконуючи системне тестування, можна виявити наступні типи дефектів:

- Неправильне використання системних ресурсів.
- Непередбачені комбінації даних користувача.
- Проблеми з сумісністю оточення.
- Непередбачені сценарії використання.
- Невідповідність функціональним вимогами.
- Погана зручність використання.

Системне тестування виконується методом «Чорного ящика», тому як все те, що перевіряється є «зовнішніми» сутностями, які не вимагають взаємодії з внутрішнім складом програми. Також виконувати його рекомендується в оточенні, максимально наближеному до оточення кінцевого користувача.

Можна виділити 2 підходи до системного тестування:

1. *На базі вимог.* Тестування проводиться відповідно до функціональних або нефункціональних вимог, для кожного з яких пишеться test case (тестові прецеденти).

2. *На базі випадків використання.* Тестування відбувається відповідно до варіантів використання продукту, на основі яких створюються user cases (користувальницькі прецеденти). Для кожного з даних користувальницьких прецедентів створюються свої тестові прецеденти.

3. Інтеграційне тестування

Виходячи із відмінностей між модульним тестуванням і системним тестуванням, інтеграційне тестування є перехідним етапом між представленням програми у вигляді окремих модулів у вигляд повністю функціональної системи.

Інтеграційне тестування – вид тестування, при якому на відповідність вимог перевіряється інтеграція модулів, їх взаємодія між собою, а також інтеграція підсистем в одну загальну систему. Для інтеграційного тестування використовуються компоненти, вже перевірені за допомогою модульного тестування, які групуються у множини. Дані множини перевіряються відповідно до плану тестування, складеним для них, а об'єднуються вони через свої інтерфейси.

Так як модулі поєднуються між собою за допомогою передбачених реалізацією інтерфейсів і в процесі тестування у нас немає потреби розглядати внутрішню структуру компонентів, можна стверджувати, що інтеграційне тестування виконується методом «чорного ящику» (Рис. 1.3) [1].

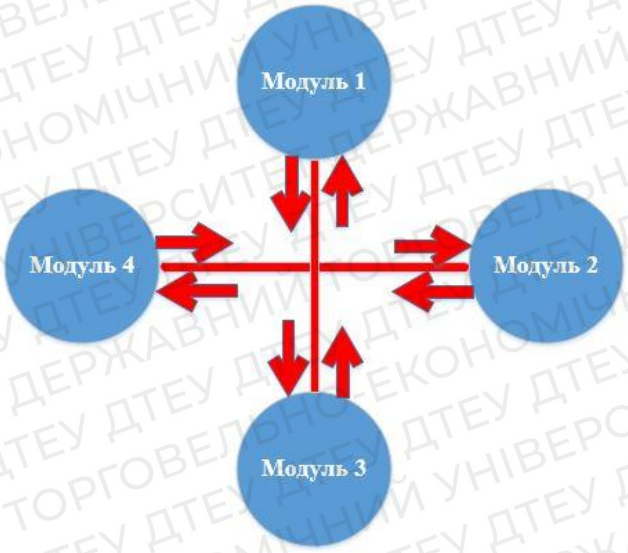


Рис. 1.3. Схема інтеграційного тестування

Об’єкт тестування виділений червоним кольором.

Існує декілька підходів до інтеграційного тестування:

- *Знизу вгору.* Спочатку збираються і тестуються модулі найнижчих рівнів, а потім по зростанню до вершини ієрархії. Даний підхід вимагає готовності усіх зібраних модулів на всіх рівнях системи.
- *Зверху вниз.* Даний підхід передбачає рух з модулів високого рівня вниз. При цьому використовуються заглушки для тих модулів, які знаходяться нижче за рівнем, але включення яких до тесту ще не відбулося.
- *Великий вибух.* Всі модулі всіх рівнів збираються разом, а потім тестується. Даний метод економить час, але вимагає ретельного опрацювання тест кейсів.

4. Модульне тестування

*Модульне тестування (Unit testing***)**– тестування кожної атомарної функціональності додатку окремо, в штучно створеному середовищі. Саме потреба у створенні штучної робочого середовища для певного модуля, вимагає від тестувальника знань в автоматизації тестування програмного забезпечення, деяких навичок програмування. Дане середовище для деякого юніта створюється за допомогою драйверів і заглушок (Рис. 1.4) [4].

Драйвер – визначений модуль тесту, який виконує елемент, що ми тестуємо.

Заглушка – частина програми, яка симулює обмін даними із компонентом, що тестується, виконує імітацію робочої системи.

Заглушки необхідні для:

− імітації відсутніх компонентів для роботи даного елементу;

− подачі або повернення модулю певного значення, можливості надати тестеру самому ввести потрібне значення;

– відтворення певних ситуацій (виключення або інші нестандартні умови роботи елемента).

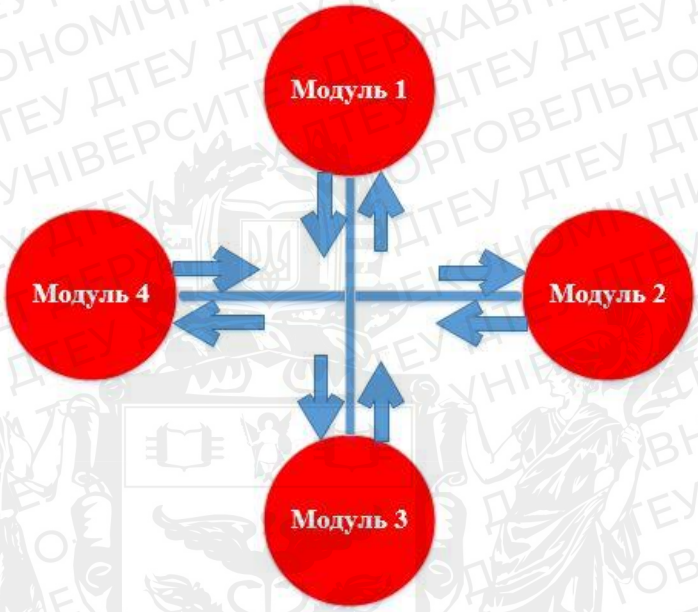


Рис. 1.4 Схема модульного тестування

Розглянемо переваги модульного тестування.

- модульне тестування мотивує програмістів писати код максимально оптимізованим, проводити рефакторинг (спрощення коду програми, не зачіпаючи її функціональність), так як за допомогою Юніт-тестування можна легко перевірити працездатність розглянутого компонента.
- необхідність відділення реалізації від інтерфейсу (зважаючи на особливості модульного тестування), що дозволяє мінімізувати залежності в системі.
- документація Юніт-тестів може служити прикладом «живого документа» для кожного класу, що тестується даним способом.

Модульне тестування допомагає краще зрозуміти роль кожного класу на тлі всієї програмної системи. Також, при «розробці через тестування», яка активно використовується в екстремальному програмуванні, модульне

тестування є одним з основних інструментів, що дозволяє розробляти продукт відповідно до вимог до даного модулю.

Висновки. В першому розділі проведено аналітичне дослідження роботи Web-додатків. Розглянуто сутність Web-додатків, принципи роботи, переваги і недоліки роботи Web-додатків. Проаналізовано особливості тестування, рівні тестування Web-додатків.

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

2.1 Сутність, принципи і переваги автоматизованого тестування

Автоматизоване тестування передбачає використання інструменту автоматизації для виконання набору тестів. У той час як ручне тестування виконується людиною, що сидить перед комп'ютером, ретельно виконує всі етапи тестування.

Автоматизація програмного забезпечення також може вводити тестові дані в систему, яку тестують, порівнювати очікувані та фактичні результати та генерувати детальні звіти про тестування. Однак воно вимагає значного вкладання коштів та ресурсів.

Цикл розробки вимагає багаторазового виконання одного й того ж набору тестів під час послідовності розробки. Використовуючи автоматизацію, можна написати набір тестів і відтворювати його повторно у разі необхідності. Як тільки набір тестів автоматизовано, втручання людини не потрібне. Також це допомагає поліпшити ROI (коефіцієнт окупності інвестицій). Метою автоматизації є скорочення кількості тестів, які потрібно запускати вручну, а не усунення ручного тестування в цілому.

Автоматизоване тестування програмного забезпечення є важливим з наступних причин:

- Ручне тестування усіх робочих процесів, усіх полів, усіх негативних сценаріїв вимагає багато часу та грошей.
- Доволі складно протестувати мультимовні сайти вручну.
- Автоматизація не вимагає втручання людини. Ви можете запустити автоматичний тест без нагляду (наприклад вночі).
- Автоматизація збільшує швидкість виконання тесту.
- Автоматизація допомагає збільшити покриття тестами (Test Coverage).

- Ручне тестування може бути нудним а, отже, веде до випадкових помилок [2, 3].

Існує кілька основних видів автоматизованого тестування:

– *Автоматизація тестування коду* (Code-driven testing) – тестування на рівні програмних модулів, класів та бібліотек (фактично, автоматичні юніт-тести);

– *Автоматизація тестування графічного інтерфейсу* користувача (Graphical user interface testing) – спеціальна програма (фреймворк автоматизації тестування) дозволяє генерувати користувальницькі дії – натиснення кнопок, кліки мишкою, та відслідковувати реакцію програми на ці дії – чи відповідає вона специфікації.

– *Автоматизація тестування API* (Application Programming Interface) – програмного інтерфейсу програми. Тестуються інтерфейси, призначені для взаємодії, наприклад, з іншими програмами або з користувачем. Тут, знову ж таки, як правило, використовуються спеціальні фреймворки.

Для складання автоматизованих тестів, QA-фахівець повинен вміти програмувати. Автоматичні тести – це повноцінні програми, просто призначені для тестування.

Коли, що і як автоматизувати і чи автоматизувати взагалі – дуже важливі питання, відповіді на які повинна дати команда розробки. Вибір правильних елементів програми для автоматизації більшою мірою визначатиме успіх автоматизації тестування в принципі. Потрібно уникати автоматизації тестування ділянок коду, які можуть часто змінюватися.

Порівняння ручного та автоматизованого тестування

Як ручне, так і автоматизоване тестування можуть використовуватися на різних рівнях тестування, а також бути частиною інших типів і видів тестування.

1. Автоматизація зберігає час, сили та гроші. Одного разу автоматизований тест можна запускати знову і знову, докладаючи мінімум зусиль.

2. Вручну можна протестувати практично будь-який додаток, в той час як автоматизувати варто тільки стабільні системи. Автоматизоване тестування використовується головним чином для регресії. Крім того, деякі види тестування, наприклад, ad-hoc або дослідницьке тестування можуть бути виконані тільки вручну.

3. Мануальне тестування може бути повторюваним та нудним. У той же час, автоматизація може допомогти цього уникнути – за вас все зробить комп'ютер.

В таблиці 2.1 можна подивитися перевагі і недоліки автоматизованого тестування.

Таблиця 2.1

Переваги і недоліки автоматизованого тестування

Переваги	*Недоліки*
Підвищує точність та швидкість виявлення помилок порівняно з ручним тестуванням	Вибір правильного інструменту вимагає значних зусиль, часу і планування.
Економія часу та зусиль.	Потреба в знаннях засобів тестування
Збільшує охоплення тестуванням, оскільки багато інструментів тестування можна використовувати одночасно, що дозволяє паралельне тестування та різні сценарії тестування.	Вартість придбання інструменту для тестування та тестового обслуговування.
Повторюваний сценарій.	Потрібна певна кваліфікація для написання сценарію.

Таким чином, на реальних проектах найчастіше використовується комбінація ручного і автоматизованого тестування, причому рівень автоматизації буде залежати як від типу проекту, так і від особливостей постановки виробничих процесів в компанії.

2.2 Підходи до автоматизованого тестування

Для реалізації ефективного автоматизованого тестування велике значення має підхід до побудови тестів, що залежить від самого процесу розроблення, бо ці речі мають бути взаємопов'язані.

Виокремлюють такі основні підходи [5, 7]:

− *розроблення на основі тестів* (англ. Test Driven Development). Цей підхід передбачає спочатку організацію автоматизованого тестування за допомогою написання модульних, функціональних та інтеграційних тестів, і лише після цього написання робочого коду продукту;

− *поведінковий підхід до тестування* (англ. Behaviour Driven Development), що є різновидом (доповненням) розроблення на основі тестів, із тією лише різницею, що цей підхід орієнтований на поведінку, яка покривається тестами (за тестового підходу основний фокус іде безпосередньо на сам код).

Суть поведінкового підходу полягає у розробленні тестів, які описують систему у термінах, зрозумілих нетехнічному спеціалісту;

− тестування на основі ключових слів (англ. Keyword Driven Testing). Цей підхід передбачає використання ключових слів, що описують набір дій, потрібних для виконання конкретного кроку тестового сценарію;

− тестування на основі даних (англ. Data Driven Testing). За цього підходу тестові дані зберігаються окремо від тестових сценаріїв, наприклад, у файлах або в базі даних.

Такий розподіл значно спрощує самі тести та їх об'єм.

Поведінковий підхід (BDD) як один з інструментів для організації автоматизації тестування Часто витрачають багато часу на комунікацію між клієнтом і командою із розроблення для того, щоб зрозуміти, що потрібно розробити, що вже зроблено і що хочуть бачити кінцеві користувачі [17]. Цей зв'язок часто є слабким місцем у процесі. Поведінковий підхід є сукупністю практик для процесу розроблення, спрямованих на зменшення деяких

ризиків, пов'язаних із комунікацією та зворотним зв'язком між командою з розробки та клієнтом. Саме за допомогою цього підходу є можливість імітувати, як програма має вести себе з погляду кінцевого користувача. Також тестові сценарії можна писати у реальному часі, спираючись на поведінку системи. Якщо команди з розробки працюють у змінному середовищі, яке базується на ітеративному доставленні програмного забезпечення до кінцевого користувача, то поведінковий підхід дає змогу проекту бути більш гнучким [5, 7]. Кожен тестовий сценарій буде реальним сценарієм для користувача, а не простим тестовим випадком, завдяки впровадженню матриці «Роль-Поведінка-Причина» та формули «Дано-Коли-Тоді», що базуються на синтаксисі Gherkin.

На ринку є декілька інструментів із відкритим доступом для реалізації цього підходу. Розглянемо їх порівняння у таблиці 2.2.

Таблиця 2.2

Порівняння інструментів BDD

Назва	«Жива» документація	Зручність у написанні тестів і документації	Налаштування та робота	Звітування	Інтеграція з CI/CD
Cucumber [6]	Наявна	Зручно, зі стандартами BDD	Зручний у використанні й нескладний у налаштуванні	Так, свій формат. У реальному часі	Так
Jbehave [9]	Частково	Зручно, зі стандартами BDD	Складний у налаштуванні	Так, але потребує налаштувань	Так
Gauge [8]	Частково	Не дуже зручно (без більш стандартного формату)	Складний у налаштуванні	Так, але потребує налаштувань	Так

2.3 Автоматизоване тестування з використанням Python

Автоматизоване тестування - це добре відомий контекст у світі тестування. Саме там плани тестування виконуються за допомогою сценарію, а не людини.

Python містить інструменти та бібліотеки, які підтримують автоматизоване тестування.

Тестові приклади Python порівняно легко писати. Зі збільшенням використання Python, основи автоматизації тестів на основі Python також стають популярними.

Зі зростанням використання Python популярність фреймворків для тестування на його основі також зростає. Розглянемо основні фреймворки, їх переваги та недоліки.

1. Robot Framework
2. Pytest
3. TestProject
4. PyUnit (Unittest)
5. Nose2
6. Behave
7. Lettuce
8. Testify

1. Робот фреймворк (RF)

Це відкрита структура автоматизації тестів для тестування на приймання, ATDD і RPA. Її ядро написане на Python, але може працювати в Jython (Java реалізація Python) і IronPython (Python для .NET framework).

Переваги:

1. Він заснований на підході тестування на основі ключових слів, який дозволяє нам легко створювати тестові випадки з використанням зручних для людини ключових слів (не вимагає досвіду кодування).

2. Підтримує всі операційні системи (Windows, Linux або macOS) і всі додатки (веб, мобільні та настільні додатки).

3. Надає чіткі та зручні дані звіту HTML (включаючи скріншоти).

4. Багата екосистема з безліччю API, що робить її добре розширюваною фреймворком і дозволяє інтегруватися з будь-яким іншим стороннім інструментом.

5. Ідеально підтримується спільнотою і має багато інтернет-ресурсів.

Недоліки:

1. Паралельне тестування не підтримується з коробки, але його можна використовувати з Selenium Grid (https://www.selenium.dev/documentation/en/grid/) або через Pabot (https://pabot.org/).

2. Краще чи гірше, це змушує вас працювати за заздалегідь визначеною методологією, тому спочатку крива навчання може бути довшою, ніж зазвичай.

3. Створення загальних ключових слів може зайняти більше часу, ніж звичайне кодування.

4. Важко створювати звіти.

2. Pytest

Pytest - це фреймворк тестування з відкритим вихідним кодом, який є одним з найбільш широко використовуваних в Python. Pytest також підтримує модульне, функціональне та API-тестування.

Переваги:

1. Дозволяє створювати компактні і прості набори тестів.

2. Легко розширюється за допомогою плагінів, таких як: *pytest-randomly, pytest-cov, pytest-django, pytest-bdd.*

3. Можна додати *плагін pytest html* до свого проекту для друку HTML-звітів лише з одним прапором у командному рядку.

4. Можна паралельно запускати тести за допомогою плагіна pytest-xdist.

Недоліки:

Сумісність не є ключовим фактором для Pytest, тому що, хоча легко писати тестові випадки на Pytest, їх неможна використовувати в інших рамках через використання процедур, унікальних для Pytest.

3. TestProject

TestProject - це безкоштовний фреймворк для автоматизованого тестування з хмарними та локальними HTML-звітами. За допомогою TestProject можна легко збирати автоматизацію для тестування мобільних, web- або інших додатків. Він підтримує Python версії 3.6 і вище, а також фреймворки Pytest і Unittest. TestProject включає в себе всі залежності, необхідні для надання єдиного кросплатформного агента тестування.

Переваги:

1. Агент, який включає в себе всі сторонні бібліотеки, необхідні для запуску та розробки автоматизації тестів для мобільних, веб- або загальних тестів.
2. Безкоштовні автоматичні звіти у форматах HTML/PDF (зі скріншотами).
3. Історія виконання доступна через API RESTful.
4. Завжди актуально завдяки останнім і стабільним версіям драйверів Selenium/Appium.
5. Уніфікований SDK для веб-тестів, Android, iOS та Generic.
6. Вбудований бігун і функціональність для звітності.
7. Кросплатформна підтримка Mac, Windows, Linux і Docker.
8. Велика спільнота та підтримка: форум, блог та вбудований чат.

Недоліки:

1. Агент проводить один тест за раз, тому вам знадобляться агенти Docker для паралельного тестування.
2. Функції співпраці з гібридної хмари обмежені під час роботи в автономному режимі.

4. PyUnit (Unittest)

PyUnit (Unittest) — це фреймворк тестування підрозділів Python, натхненний JUnit. Він є частиною стандартної бібліотеки Python, тому більшість розробників починають свій шлях до тестування з нею.

Переваги:

1. Так як цей фреймворк є частиною стандартної бібліотеки, ніяких додаткових модулів встановлювати не потрібно - все виходить з коробки.
2. Пропонує просте і гнучке виконання тестових кейсів.
3. Швидке створення тестових звітів у XML та unittest-sml-звітності.

Недоліки:

1. Призначення тестового коду іноді стає неясним через підтримку абстракції.
2. Велика кількість коду шаблону.

5. Nose2

Nose2 є наступником Nose, який заснований на PyUnit (Unittest), але з плагінами. Nose2 розширює можливості PyUnit за допомогою різних плагінів, які додають підтримку для виконання тестів, виявлення тестів, декораторів, фантастики, параметризації тощо.

Переваги:

1. Nose2 розширює фреймворк PyUnit (Unittest), який міститься в стандартній бібліотеці Python.
2. Включає в себе велику кількість вбудованих плагінів, які можуть спростити і прискорити тестування.
3. Підтримує паралельне тестування.
4. Автоматично створює тести.

Недоліки:

1. Відсутність документації.
2. Не так активно підтримується, як інші фреймворки.

6. Behave

Behave є одним з найпопулярніших Фреймворків тестування BDD (розробка, керована поведінкою) на Python.

Переваги:

1. Дозволяє писати тестові кейси читабельною мовою, що спрощує співпрацю між командами з подібним функціоналом.

2. Існує безліч документації та підтримки, які допоможуть вам розпочати роботу.

3. Повністю підтримує Gherkin, тому створення функціональних файлів не вимагає спеціальних технічних знань.

Недоліки:

1. Немає підтримки паралельного виконання.
2. Тільки тестування чорних ящиків.

7. Lettuce

Lettuce - це ще один фреймворк Python BDD на основі Cucumber.

Переваги:

1. Підтримує Gherkin, що дозволяє членам команди без технічного фону легко створювати тести природною мовою.

2. За аналогією з Behave, він в основному призначений для тестування чорних ящиків, але також може бути використаний для більшої кількості типів тестування. Наприклад, Lettuce може тестувати різні моделі поведінки і взаємодії серверів і баз даних.

Недоліки:

1. Деякої функціональності від інших фреймворків не вистачає, тому салат більше підходить для невеликих проектів.

2. Не схоже, що сама структура і її документація активно підтримуються.

3. Для перевірки успішності реалізації потрібно налагодити спеціальну комунікацію між усіма зацікавленими сторонами проекту: розробниками, тестувальниками і менеджерами.

8. Testify

Testify створений для заміни Unittest і Nose і має розширену функціональність в порівнянні зі стандартним Unittest.

Переваги:

1. Він використовується для модульного, інтеграційного та системного тестування.
2. Для тих, хто знайомий з Unittest, дуже легко працювати.
3. Наявність великої кількості плагінів.
4. Як і Nose2, Testify здатний виявляти тести.
5. Простий синтаксис світильників.

Недоліки:

1. Бракує великої кількості документації, тому новачкам доведеться шукати багато речей самостійно.
2. Комплексне впровадження паралельного тестування.

Висновки. В розділі 2 розглянуто сутність, принципи, переваги та недоліки автоматизованого тестування. Проаналізовано основні підходи до автоматизованого тестування. Розглянуто основні фреймворки тестування на основі Python, їх переваги і недоліки. Для розробки системи тестування вибрано фреймворк Pytest. І написання функцій тестування.

РОЗДІЛ 3. СИСТЕМА ТЕСТУВАННЯ WEB-ДОДАТКІВ

3.1 Розробка web-додатку «Help me»

Розробимо Web-додаток, для якого потім буде розроблено систему тестування.

Web-додаток, персональний помічник, має перелік наступних функцій:

1. Зберігає контакти з іменами, адресами, номерами телефонів, email і днями народження до книги контактів.
2. Перевіряє вірність веденого номеру телефону і електронної адреси під час створення та редагування запису. У випадку некоректного введення інформує користувача.
3. Здійснює пошук необхідного запису в книзі контактів.
4. Редагує і видаляє записи в книзі контактів.
5. Зберігає, редагує і видаляє замітки з довільною текстовою інформацією.
6. Додає «теги» та ключові слова, що описують тему та зміст замітки.
7. Здійснює пошук тег за ключовими словами.

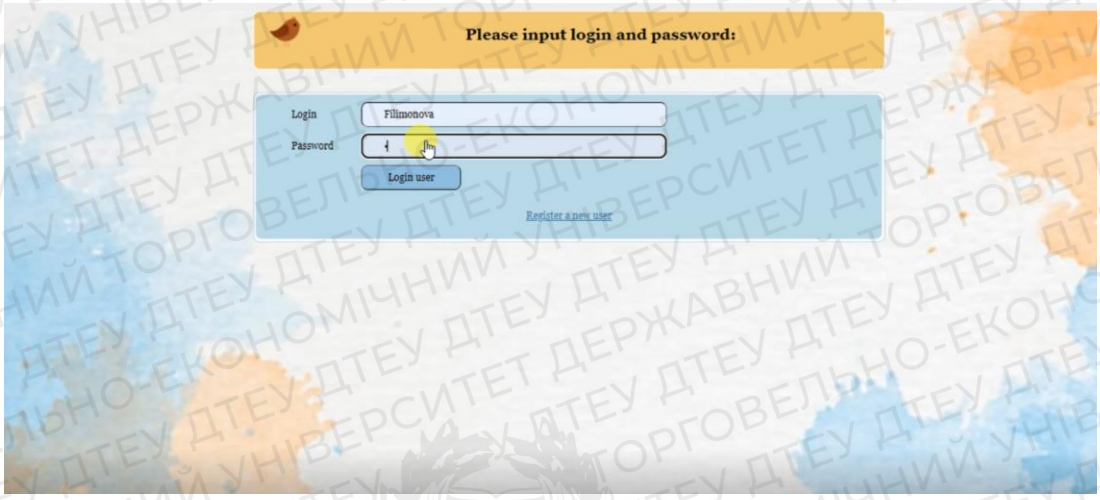
Технології, які застосовувались при розробці додатку (Табл. 3.1)

Технології для розробки Web-додатку

Таблиця 3.1

Technology List	**Utilites**
Python	VSCode/PyCharm
Keras/tensorflow	Pytest
HTML/CSS	Git/GitHub
JavaScript	
Flask	

На рис. 3.1 бачимо демонстрацію роботи додатку «Help me».


Please input login and password:
Login
Filimonova
Password
Login user
Register a new user


Please edit Contact details:
Return to BOT nest
Name
Peter Henry
Phone list
Email
stanley34@sanchez.biz
Birthday
17.11.1972
ZIP
68224
Country
Swaziland
Region
City
Port Kristen
Street
Megan Corners
House
216
Apartment
47
Save contact

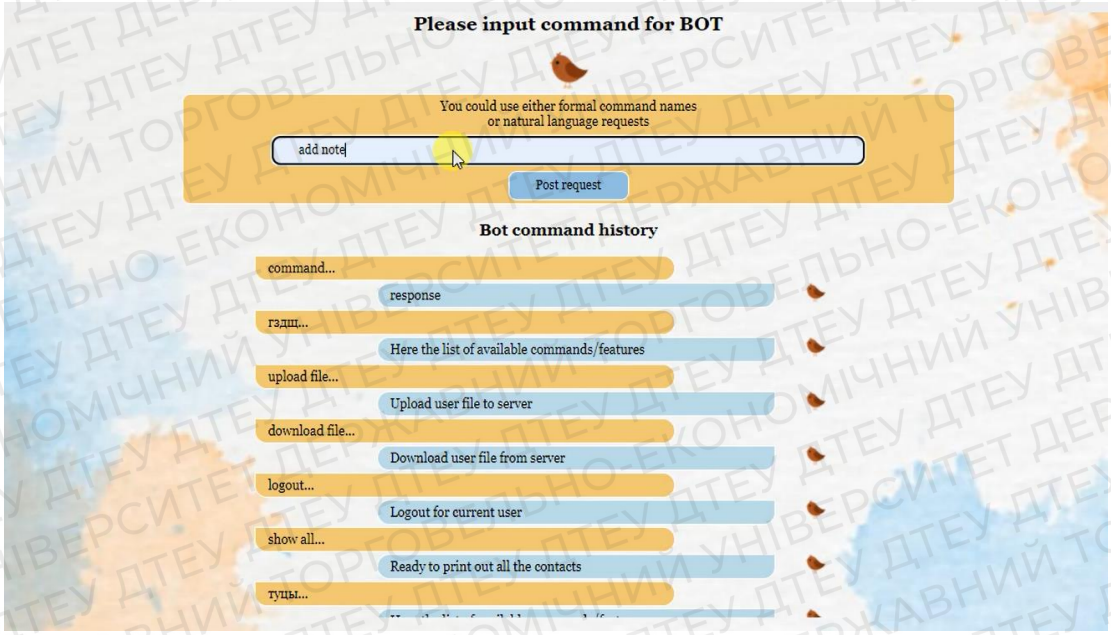

Please input command for BOT
You could use either formal command names
or natural language requests
add note
Post request
Bot command history
command...
response
гэдш...
Here the list of available commands/features
upload file...
Upload user file to server
download file...
Download user file from server
logout...
Logout for current user
show all...
Ready to print out all the contacts
тупы...

Рис. 3.1. Персональний помічник «Help my»

При розробці використовували мову програмування Python. Web-інтефейс реалізовано за допомогою фреймворку Flask. Інформація зберігається в базі даних PostgreSQL.

3.2 Розробка системи тестування для Web – додатку «Help me»

Розроблений додаток виконує багато функцій. Тому розробка автоматизованої системи тестування саме в цьому випадку була необхідна, тому що написання ручних тестів потребує багато часу. Та існує вірогідність, що певна частина коду не буде покрита тестами.

Тому було прийнято рішення розробити комбіновану систему, яка містить написання тестів з використанням фреймворка Pytest. І написання функцій тестування.

Система тестування включає в себе тести для всіх функцій, які виконує додаток (Рис. 3.2).

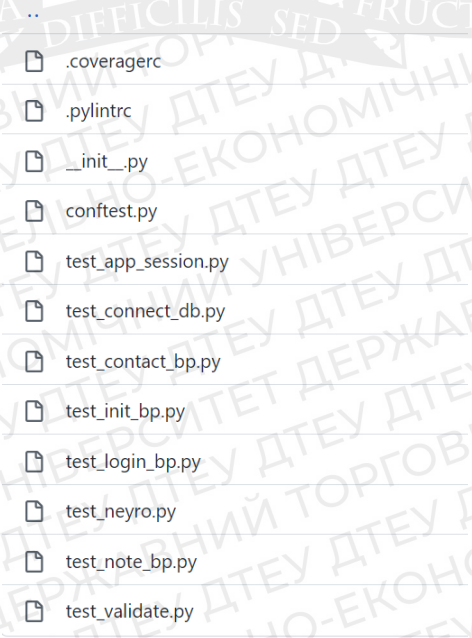


Рис. 3.2 Структура файлів, що містять тести

Наприклад, на рис. 3.3 можна побачити фрагмент файла *test_validate.py* (перевірка символів, що входять до номеру телефону (pythest)).

```
@pytest.mark.parametrize('phone_list', phones)
def test_clean_phone_str(phone_list):
    """
    Testing phone string cleaner
    :param phone_list: fixture
    :return:
    """
    res = clean_phone_str(phone_list)
    assert res.isdigit() is True


names = ['', '*' * 51, 34, None, False]
```

Рис. 3.3 Тестування коректності номеру телефону

Також наведемо фрагмент файла *test_login_bp.py* (рис. 3.4).

```
def test_logout(app, client):
    """
    Testing the /login/logout handler for mongo
    :param app: fixture
    :param client: fixture
    :return:
    """
    app.before_request_funcs[None] = [before_request]

    response = client.post('/login/login', data={'Login': 'test', 'Password': 'test'})
    assert response.headers['Location'] == 'http://localhost/bot-command'

    response = client.post('/bot-command', data={'BOT command': 'help'})
    assert response.headers['Location'] == 'http://localhost/help_'

    response = client.post('/login/logout')
    assert response.status_code == 405

    response = client.get('/login/logout')
    assert response.status_code == 302

    response = client.post('/bot-command', data={'BOT command': 'help'})
    assert response.headers['Location'] == 'http://localhost/login/login'
```

Рис. 3.4 Тестування логіну

В результаті роботи 85% відсотків коду було покрито тестами (Рис. 3.5).

	Stmts	Miss	Cover
\bot\SQL_alchemy_classes.py	105	15	86%
\bot__init__.py	0	0	100%
\bot\contact_bp.py	150	20	87%
\bot\contacts_data_classes.py	194	17	91%
\bot\db_postgres.py	20	1	95%
\bot\file_bp.py	54	5	91%
\bot\file_data_classes.py	99	11	89%
\bot\fill_db.py	95	80	16%
\bot\init_bp.py	65	14	78%
\bot\login_bp.py	57	4	93%
\bot\main.py	55	15	73%
\bot\neural_code.py	58	1	98%
\bot\news_bp.py	21	4	81%
\bot\news_feed__init__.py	0	0	100%
\bot\news_feed\app.py	16	2	88%
\bot\news_feed\covid_parsing.py	35	0	100%
\bot\news_feed\currency_parsing.py	56	1	98%
\bot\news_feed\get_content2.py	46	11	76%
\bot\news_feed\inflation_parsing.py	40	0	100%
\bot\news_feed\settings.py	1	0	100%
\bot\note_bp.py	140	23	84%
\bot\notes_data_classes.py	123	15	88%
	0	0	100%
	31	1	97%
\bot\users_data_classes.py	60	8	87%
\bot\validate.py	123	6	95%
	1648	254	85%

Рис. 3.5 Результат роботи автоматизованої системи тестування

Висновки. В розділі 3 було розроблено Web-додаток «Help me» і розроблено систему автоматизованого тестування, результатом роботи якої було покрито тестами 85% додатку і протестовано його функціонал.

ВИСНОВКИ

У випускній кваліфікаційній роботі розроблено систему тестування. Для цього було розроблено Web-додаток «Help me», що виконує багато корисних функцій. В результаті проведених досліджень були отримані такі висновки:

1. Розглянуто сутність, принципи роботи, переваги і недоліки Web-додатків.
2. Проаналізовано особливості тестування, рівні тестування Web-додатків.
3. Проаналізовано основні підходи до автоматизованого тестування.
4. Розглянуто основні фреймворки тестування на основі Python, їх переваги і недоліки.
5. Розроблено Web-додаток «Help me» з набором функцій записної книги.
6. Розроблено систему тестування, результатом роботи якої було покрито тестами 85% додатку і протестовано його функціонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бенюх Л. І., Глибовець А. М., Афонін А. О. Поведінковий підхід (BDD) як ефективний метод для організації автоматизованого тестування у безперевному доставленні продукту - Наукові записки НаУКМА. Volume 3. Комп'ютерні науки, 2020. P. 62-68
2. Буров Є.В. Інтелектуальна система автоматизованого тестування програмного продукту з використанням алгоритмічних моделей. Вісник Національного університету «Львівська політехніка». 2011. № 699: Інфор-маційні системи та мережі. С. 21–30
3. Кравчук С.О. Проблеми автоматизації тестування програмного забезпечення. Актуальні задачі сучасних технологій: матеріали VII міжнар. наук.-техн. конф. мол. учен. та студ., м. Тернопіль, 28-29 листопада 2018 р. Тернопіль, 2018. С. 95
4. Тестування веб-проєктів: основні етапи та поради. [Електроний ресурс]. Режим доступу: https://qalight.ua/baza-znaniy/testuvannya-veb-pro%d1%94ktiv-osnovni-etapi-ta-poradi/
5. Тузовський А. Ф. Проектування і розробка Web-додатків: навч. посібник, 2017. [Електроний ресурс]. Режим доступу: https://stud.com.ua/97571/informatika/proektuvannya_i_rozrobka_web-dodatkiv
6. Cucumber JVM – 2020 [Electronic recourse]. – Mode of access: https://github.com/cucumber/cucumber-jvm
7. Ferguson Smart John. BDD in Action / John Ferguson Smart. –Manning Publications Co, 2015. – P. 18–342.
8. Gauge – 2020 [Electronic recourse]. – Mode of access: https:// gauge.org
9. JBehave – 2020 [Electronic recourse]. – Mode of access: https://jbehave.org
10. The Art of Application Performance Testing / Ian Molyneaux – M. : O'Reilly Media. – 2009. – №1. – 158 с.

11. The Art of Software Testing / Glenford J. Myers, Revised and Updated by Tom Badgett, Todd M.Thomas, Corey Sandler. - 2nd ed. - Hoboken, New Jersey.: John Wiley & Sons, Inc., 2004 - 234 p.
12. The way of the web tester / Jonathan Rasmusson – M. Pragmatic Bookshelf;, 2016. – 258 с.
13. Top 10 Automation Testing Tools in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.netsolutions.com/insights/top-10-automation-testing-tools/
14. Performance Testing Guidance for Web Applications /– M. Microsoft Press;, 2007. – 288 с.
15. Web Load Testing for Dummies / Scott Barber – M. : John Wiley & Sons, Inc. – 2011. – 44 с.