МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕСТУВАННЯ ПРОГРАМНИХ ПРОДУКТІВ ЗАСОБАМИ PYTHON НА ПРИКЛАДІ WEB-СЕРВІСУ ХДУ 24

Кваліфікаційна робота

на здобуття ступеня вищої освіти “бакалавр”

Виконала: студентка 4 курсу
Спеціальності: 121 Інженерія програмного забезпечення
Освітньо-професійної програми: Інженерія програмного забезпечення
Кісельгоф Анна Євгеніївна
Керівник: кандидат фізико-математичних наук, доцент Вейцбліт О.Й.
Співкерівник: кандидат фізико-математичних наук, доцент Ермолаев В.А.
Рецензент: директор IT компанії «InStandart», Толстопят Констянтин Вікторович

Херсон - 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.. 4

ВСТУП 6

РОЗДІЛ 1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО ТЕСТУВАННЯ 8

1.1 Тестування ПО та його переваги.. 8

1.2 Ціль тестування ПЗ 9

1.2.1 Поняття QA та QC 9

1.3 Класифікація тестування.. 10

1.3.1 Рівні тестування 10

1.3.2 Види тестування.. 17

РОЗДІЛ 2. СПОСОБИ ТЕСТУВАННЯ.. 19

2.1 Ручне тестування... 19

2.2 Автоматизоване тестування... 20

2.3 CI/CD 21

2.4 Безперервне тестування 22

2.3.1 Етапи безперервного тестування.. 22

2.3.2 Інструменти для безперервного тестування 23

2.3.3 Переваги безперервного тестування.................................... 24

2.3.4 Недоліки безперервного тестування ... 25

РОЗДІЛ 3. ТЕСТУВАННЯ API ... 26

3.1 Визначення тестування API ... 26

3.2 Типи API тестування ... 27

3.3 Інструменти для дизайну та тестування API ... 29

3.3.1 Swagger ... 29

3.3.2 Postman ... 30

3.3.3 API Platform ... 31

3.3.4 SAP Integration Suite ... 32

3.3.5 Workato ... 33

3.3.6 UUID ... 34

3.4 REST API ... 36

3.4.1 HTTP ... 38

3.4.2 CRUD ... 41

3.5 Розробка тест-кейсів для ХДУ 24 ... 42

ВИСНОВКИ ... 47

ДОДАТКИ ... 49

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ... 50

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. **API** - програмний інтерфейс додатку (application programming interface).
2. **JSON** - текстовий формат обміну даними на основі синтаксису ECMAScript (Javascript object notation).
3. **JWT** - коротке повідомлення з криптографічними перетвореннями, що використовується зокрема для аутентифікації користувачів (JSON WebToken).
4. **REST** - передача репрезентативного стану системи, підхід до організації API (Representational state transfe).
5. **SOAP** - протокол обміну структурованими повідомленнями, базується на XML (Simple Object Access Protocol).
6. **XML** - розширювана мова розмітки (Extensible Markup Language).
7. **Авторизація** - підтвердження користувачем права на виконання тих чи інших дій.
8. **Аутентифакація** - підтвердження користувачем своєї особи.
9. **Ідентифікація** - представлення користувачем себе.
10. **AUT** - Application under testing.
11. **Тест Кейс** (Test Case) - сукупність кроків, конкретних умов та параметрів, необхідних для перевірки реалізації функції, що тестується чи її частини.
12. **Тест план** (Test plan) - це документ, який описує весь обсяг робіт з тестування.
13. **UI** - інтерфейс користувача.
14. **Endpoint** - це інтерфейс, виставлений стороною, що спілкується, або каналом зв'язку.

15. **Реліз** (release) - випуск остаточної версії програми готового для використання продукту.

16. **QA** (quality assurance) - процес управління якістю продукту.

17. **QC** (quality control) - використовується для перевірки якості кінцевого продукту.

18. **DevOps** - методологія автоматизації технологічних процесів складання, налаштування та розгортання програмного забезпечення.
19. **GUI** (graphical user interface) - графічний інтерфейс користувача - тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації.
20. **Датасет** (data set) - набір логічно впорядкованих даних.
21. **OAuth** (Open Authorization) - відкритий стандарт авторизації, який дозволяє користувачам відкривати доступ до своїх приватних даних, що зберігаються на одному сайті, іншому сайту, без необхідності вводу імені користувача та паролю.

ВСТУП

Кожна команда розробників програмного забезпечення ретельно тестує свою продукцію, але на етапі поставки програмне забезпечення до користувачів завжди має дефекти. Інженери-тестувальники намагаються помітити їх до того, як продукт буде випущений, але вони часто з’являються знову, навіть за допомогою найкращих процесів ручного тестування. Програмне забезпечення для автоматизації тестування - це найкращий спосіб підвищити ефективність, дієвість та покриття тестування функціоналу програмного забезпечення.

Актуальність даної теми зумовлена тим, що вичерпне тестування веб-сервісу на хоча б найбільш вірогідні тест-кейси за якими користувачі будуть використовувати продукт призведе до раннього виявлення та виправлення дефектів до того, як сервіс буде представлено до широкого використання. Це зробить його використання більш приємним у використанні та не змусить користувачів обирати між вашим сервісом та конкурентів.

Метою курсової роботи є розробка API-тест-кейсів для web-сервісу ХДУ 24 - комплексної системи збору, обробки та зберігання інформації про індивідуальну траєкторію навчання студентів, навчальний процес, контроль успішності здобувачів та працівників освіти.

Об’єкт дослідження - засоби мови програмування Python для тестування програмних продуктів.

Предмет дослідження - API тест-кейси для індивідуального профілю студентів та працівників, їх взаємодія з існуючими сутностями та перевірки безпеки персональних даних.

Завдання дослідження:

- проаналізувати основні теоретичні поняття що стосуються методів, рівнів та способів тестування веб-сервісів;
- проаналізувати інфраструктуру, моделі та рівні доступу існуючого програмного забезпечення;
- розробити набір API тест-кейсів для модулів «Авторизація», «Користувач», «Працівник», «Студент», «Дисципліна», «Факультет» та інших згідно з таблицею [1];
- імплементувати найбільш пріоритетні тест-кейси;

Робота складається з 3-х розділів, містить 12 рисунків та 1 додаток.

РОЗДІЛ 1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО ТЕСТУВАННЯ

1.1 Тестування ПО та його переваги

Тестування програмного забезпечення - це процес оцінки та перевірки того, що програмний продукт, додаток або сервіс виконує те, що він повинен робити. Переваги тестування включають запобігання помилкам, зниження витрат на розробку та підвищення продуктивності команди та продукту.

Контроль якості при розробці програмного забезпечення необхідний тому, що затримка релізу або дефекти програмного забезпечення можуть зашкодити репутації бренду, що у подальшому призведе до втрати довіри та клієнтів. У окремих випадках помилка або дефект може призвести до погіршення роботи взаємопов'язаних систем або спричинити серйозні несправності [2].

Аналогію можна провести з автомобільною компанією, що змушена відкликати понад 1 мільйон автомобілів через програмний дефект датчиків подушки безпеки або програмна помилка, яка спричинила провал запуску військового супутника вартістю 1,2 мільярда доларів США [3]. У 2016 році збої програмного забезпечення в США коштували економіці 1,1 трильйона доларів США, окрім того, вони вплинули на 4,4 мільярда клієнтів [4].

Хоча на саме тестування також необхідно закладати чималий бюджет та час, компанії можуть заощадити значну кількість коштів на розробці та підтримці, якщо у них є хороша техніка тестування та процеси забезпечення якості. Раннє тестування програмного забезпечення виявляє проблеми до того, як продукт виходить на ринок. Чим швидше команди розробників отримають відгуки про тестування, тим швидше вони зможуть вирішити такі проблеми, як[2]:

- архітектурні недоліки;
- невдалі дизайнерські рішення;
- застаріла або неправильна функція;
- вразливості безпеки;
- масштабування.

1.2 Ціль тестування ПЗ

Ціль тестування або мета розробки та виконання тестів:

- забезпечити очищення ПЗ від помилок до прийнятного рівня (неможливо надати 100% покриття, але необхідно зробити все можливе і гарантувати, що очевидні дефекти виправлені) [6];
- переконатися, що ПЗ відповідає вимогам та специфікації;

1.2.1 Поняття QA та QC

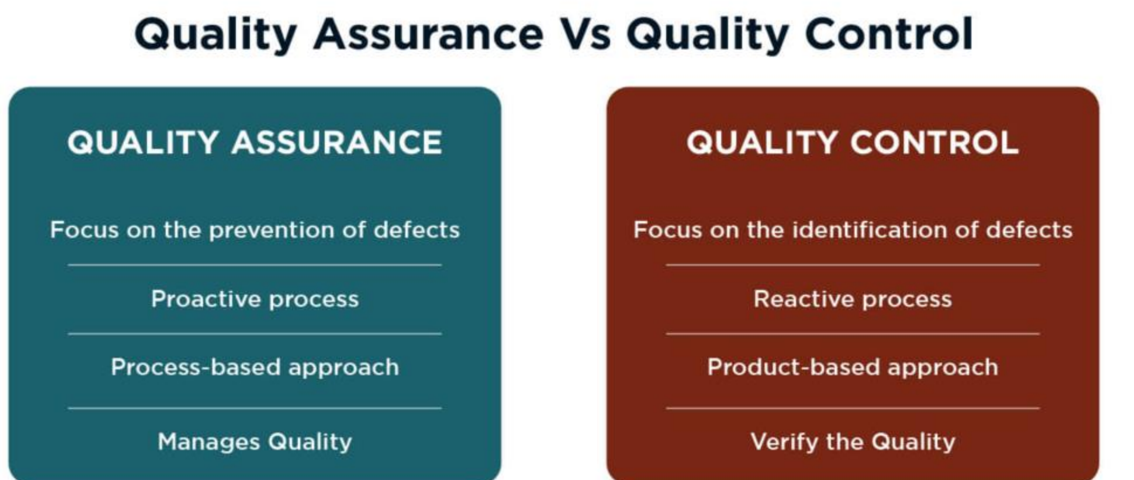


Рис. 1.2.1.1 Різниця QA та QC

QA спеціаліст аналізує продукт і надає команді QA зворотний зв'язок. Мета команди QA - знайти основну проблему, чому виникають проблеми з кодом [5].

Розрізненість між QC та QA допомагає командам не тільки зосередитися на функціонуючому продукті, але й підтримувати найкращі методи роботи в команді. Якщо команда QA добре аналізує

та виправляє процеси, команда QC матиме менше проблем для вирішення в довгостроковій перспективі. Хоча на практиці майже не використовують відокремлення цих двох команд, узагальнено людей з команди тестувальників прийнято називати QA.

На QA лежить відповідальність за розробку та впровадження процесів та стандартів для покращення життєвого циклу розробки ПЗ, та забезпечення впевненості у тому, що ці процеси виконуються (рис. 1.2.1.1). Фокусом QA є запобігання дефектам на всіх етапах його реалізації та постійне його вдосконалення. Займається такими питаннями як "а які види та методи тестування ми будемо використовувати?". Тут дуже підходить термін Validation із питанням "Are we building the right product?" - чи правильний продукт ми робимо, чи задовольняє продукт потреб користувача [5].

1.3 Класифікація тестування

Процес пошуку багів у програмних продуктах відрізняється залежно від кінцевої мети. Алгоритм виявлення дефектів сайту при перекладі сторінки іноземною мовою та визначенні граничного навантаження буде відрізнятися методами, інструментами та фахівцями, що залучаються до процесу.

1.3.1 Рівні тестування

Тестування на різних рівнях проводиться протягом усього життєвого циклу розробки та супроводу програмного забезпечення. Рівень тестування визначає те, над чим виконуються тести: над окремим модулем, групою модулів чи системою загалом (рис. 1.3.1.1). Проведення тестування на всіх рівнях системи - запорука успішної реалізації та здачі проекту [6].

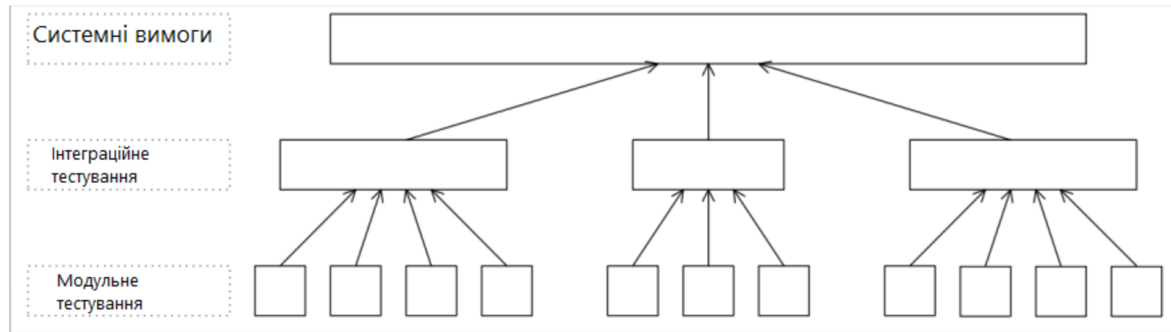


Рис. 1.3.1.1 Ієрархія рівнів тестування

Модульне тестування (Unit testing) - тестується мінімально можливий для тестування компонент, наприклад, окремий клас або функція, невеликі бібліотеки, окремі частини програми. Як правило, їх можна досліджувати ізольовано один від одного.

Для виконання модульного тестування потрібно розгорнути код в тестовому середовищі. Таким чином, наступне можна вважати критерієм для можливості написання та виконання модульного тестування:

В деяких випадках написання модульних тестів - надлишкове:

- код досить простий і не має розгалужень;
- код погано структурований і важко виділити окремий модуль;
- якщо код модуля базується на випадкові(рандомні) дані;
- не потрібно тестувати роботу сторонніх сервісів. В такому випадку доречні інтеграційні тести, де тестується написаний саме нами код;
- якщо перевірку успіху/невдачі так важко визначити кількісно, що неможливо достовірно виміряти. Наприклад, стеганографія непомітна для людей;

- якщо сам тест значно складніше написати, ніж код;
- якщо цей код - тимчасова заглушка до моменту повноцінної розробки.

Часто модульне тестування здійснюється розробниками програмного забезпечення [5]. При розробці веб-сервісу ХДУ 24 саме такий підхід і використовується - одразу після написання розробниками модулю відбувається покриття його логіки тест кейсами. Після внесення будь-яких змін до логіки модулю - зміни також вносяться до тест кейсів. Таким чином більше не потрібно витрачати багато часу на ручне тестування. Якщо будь-які зміни надалі будуть внесені - їх вплив одразу буде помічено.

Інтеграційне тестування (Integration testing) - тестуються інтерфейси між компонентами, підсистемами чи системами. Направлено на перевірку взаємодії між декількома частинами програми (кожну з яких перевірено на модульній стадії тестування). За наявності резерву часу на цій стадії тестування ведеться ітераційно, з поступовим підключенням наступних підсистем [5].

Системне тестування (System testing) - тестується інтегрована система її відповідність вимогам. Направлено на перевірку всієї програми, як єдиного цілого, зібраного з частин, перевірених на модульному та інтеграційному рівнях [6].

Приймальне тестування (Acceptance Testing) - це формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться, з метою визначення, чи задовольняє система приймальним критеріям та винесення рішення замовником або іншою уповноваженою особою приймається додаток чи ні [6].

Приймальне тестування виконується на підставі набору типових тестових випадків і сценаріїв, розроблених на підставі вимог до цього додатка. Рішення про проведення приймального тестування приймається, коли: [6]

1. продукт досягнув необхідного рівня якості;
2. замовник ознайомлений з Планом приймальних робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов'язаних з проведенням приймального тестування, дата проведення, відповідальні.

Фаза приймального тестування триває доти, доки замовник не виносить рішення про відправлення програми на доопрацювання або безпосередній прийом програми [7].

Для кожного рівня тестування можуть використовуватись різні види тестування, для кожного з яких, у свою чергу, можуть використовуватись різні типи тестових випробувань.

1.3.1.1 Альфа-тестування

Альфа-тестування - це остаточне внутрішнє приймальне тестування програмного забезпечення, тестування білого ящика - команда точно знає, як поводиться програмне забезпечення. Метою є тестування кожного окремого потоку користувачів від кінця до кінця. Ідея полягає в тому, щоб переконатися, що програмне забезпечення не має помилок, стабільно функціонує, як очікувалося [7].

Альфа-тестування необхідно узгодити з командою продукту. Вони є експертами щодо того, як програмне забезпечення має виглядати,

відчувати і вести себе. Вони зможуть забезпечити повний набір шляхів та взаємодій користувача. Використовуючи їх, можна розробити відповідні плани тестування, які будуть забезпечувати перевірку кожного шляху користувача з відповідними даними.

Альфа-тестування є важливим етапом життєвого циклу доставки програмного забезпечення. До цього етапу тестування було зосереджено на тому, щоб переконатися, що окремі частини програмного забезпечення працюють правильно. Тепер тестується, чи дійсно частини програмного забезпечення правильно взаємодіють між собою. Аналогією може бути створення автомобіля. До цього етапу тестувався двигун, трансмісія та шасі. Можливо, автомобіль навіть зібрали і випробували на рухомій дорозі. Але тепер потрібно взяти його на тестовий трек, щоб перевірити такі речі, як гальмування, керованість та функції безпеки [7].

1.3.1.2 Бета-тестування

Основна перевага бета-тестування полягає в тому, щоб програмне забезпечення надається реальним користувачам, які не мають інсайдерських знань про те, як все має працювати. Це також перший шанс перевірити, як програмне забезпечення веде себе в реальних умовах. Особливо це важливо для програм для мобільних телефонів. Мобільні телефони часто мають нестабільні мережеві з’єднання з високою затримкою. Це може створити додаткові проблеми для програмного забезпечення. Інша річ, яку робить бета-тестування, це піддає ваш бекенд реалістично високим навантаженням. До моменту бета-тестування повинно було проходити стрес-тестування на бекенд.

Часто результати тестових випробувань та реальних навантажень мають суттєві відмінності [8].

Багато компаній, що займаються програмним забезпеченням, використовують бета-тестування, щоб дізнатися, чи буде нова функція популярною серед користувачів. Хорошим прикладом є Apple, яка попередньо переглядає нові функції iOS у своїй бета-програмі. Багато з цих функцій доступні до нового релізу, але деякі - ні, а інші можуть бути відкладені до майбутнього незначного релізу. Інші компанії можуть проводити альфа- чи бета-тестування функцій, перш ніж вирішити, що саме остаточно залишити [7].

Усі великі компанії-виробники програмного забезпечення використовують бета-тестування. Google, зокрема, примітний тим, що вони часто зберігають продукти в бета-версії протягом багатьох років, але випускають програмне забезпечення для використання всіма. Тут вони мають на увазі, що програмне забезпечення є потенційно нестабільним, тому його слід використовувати з обережністю. Наприклад, Gmail був випущений у 2004 році, спочатку для обмеженої кількості запрошених користувачів, але з часом доступний для всіх. Розширена бета-версія тривала 5 років, і нарешті була офіційно випущена в липні 2009 року [8].

1.3.1.3 Відмінності альфа- та бета-тестувань

У таблиці нижче наведено основні відмінності між альфа-тестуванням і бета-тестуванням:

	Альфа	**Бета**
Головна мета	Тестуються користувацькі шляхи, чи працює додаток як очікується.	Перевіряється як реальні користувачі взаємодіють з ПЗ, як воно працює у справжніх робочих умовах.
Виконується	Внутрішньою командою, може потенційно замовником.	Звичайні користувачі або запрошені гості, що не зацікавлені у будь-якій зі сторін (замовник чи виконавець).
Рівень доступу	Білий ящик.	Чорний ящик.
Рівень організації	Ретельно структуроване, кожен шлях протестован, усі результати записані та проаналізовані.	Повністю неструктуроване, користувачі можуть робити будь-що без обмежень. Зворотній зв'язок запитується але необов’язковий.
Продуктивність	Не зацікавлені у продуктивності ПЗ або бекенду, перевіряється тільки функціональність.	Надійність та продуктивність - ключові аспекти бета-тестування разом із безпекою та стабільністю роботи.
Дії після тестування	Ідентифікуються та усуваються усі дефекти. Аналізуються та можуть бути прийнятими будь-які пропозиції до незначних змін.	Результати зворотнього зв’язку використовуються для аналізу ПЗ, також впливають на подальші версії.
Тривалість	Процес може бути довгим процесом, хоча перевага надається настільки короткому, наскільки це можливо.	Зазвичай, короткий процес, що триває декілька тижнів, хоча деякі компанії виконують бета протягом років.

1.3.2 Види тестування

Всі види тестування програмного забезпечення, залежно від цілей, можна умовно розділити на наступні групи [8]:

- функціональні;
- нефункціональні.

Функціональні тести базуються на функціях та особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування [8]:

- компонентному або модульному (Component/Unit testing);
- інтеграційному (Integration testing);
- системному (System testing);
- приймальному (Acceptance testing).

Функціональні види тестування розглядають зовнішню поведінку системи. Найпоширеніші види функціональних тестів [7]:

- функціональне тестування (Functional testing);
- тестування безпеки (Security and Access Control Testing);
- тестування взаємодії (Interoperability Testing).

Нефункціональне тестування описує тести, необхідні визначення характеристик програмного забезпечення, які можна виміряти різними величинами. Загалом це тестування того, як система працює. Далі перераховані основні види нефункціональних тестів [8]:

Види тестування продуктивності:

- навантажувальне тестування (Performance and Load Testing);

- стресове тестування (Stress Testing);
- тестування стабільності чи надійності (Stability / Reliability Testing);
- об'ємне тестування (Volume Testing);
- тестування установки (Installation testing);
- тестування зручності користування (Usability Testing);
- тестування на відмову та відновлення (Failover and Recovery Testing);
- конфігураційне тестування (Configuration Testing).

РОЗДІЛ 2. СПОСОБИ ТЕСТУВАННЯ

Тестування може проводитися трьома способами [6]:

1. Ручне тестування - найбільш практичний тип тестування, який в певний момент використовується кожною командою. Найбільш яскравий недолік у швидкоплинному життєвому циклі розробки програмного забезпечення - ручне тестування важко масштабувати.
2. Автоматизоване тестування - використовує тестові сценарії та спеціалізовані інструменти для автоматизації процесу тестування програмного забезпечення.
3. Безперервне тестування - йде ще далі, застосовуючи принципи автоматизованого тестування в масштабованому, безперервному вигляді для досягнення найнадійнішого охоплення тестами для підприємства.

2.1 Ручне тестування

Ручне тестування - це процес, у якому аналітики з контролю якості виконують тести один за одним в індивідуальному порядку. Метою ручного тестування є виявлення помилок і проблем з функціями до того, як програма буде випущено для широкого загалу [8].

Під час тестування спеціаліст вручну перевіряє ключові функції програмного додатка. Аналітики відтворюють тест-кейси та розробляють підсумкові звіти про дефекти без використання спеціальних засобів автоматизації. Ручне тестування програмного забезпечення виконується зусиллями живої людини, яка сидить перед комп'ютером, уважно переглядає програму, перевіряє різні комбінації

використання та введення, порівнює результати з очікуваною поведінкою та записує свої спостереження. Ручні тести часто повторюються під час циклів розробки для зміни вихідного коду та інших ситуацій, таких як кілька операційних середовищ та конфігурацій обладнання [9].

2.2 Автоматизоване тестування

Автоматизоване тестування - це процес, у якому тестувальники використовують інструменти та сценарії для автоматизації тестування[9].

Автоматичне тестування допомагає тестувальникам виконувати більше тестів і збільшувати покриття сценаріїв. Якщо порівнювати ручне тестування з автоматизованим, перше займає більше часу, друге - ефективніше [8].

Автоматизований інструмент тестування здатний відтворювати попередньо записані та заздалегідь визначені дії, порівнювати результати з очікуваною поведінкою та повідомляти інженеру з тестування про успіх або невдачу цих тестів. Після створення, їх можна легко повторити і розширити для виконання завдань, неможливих при ручному тестуванні. Через це автоматизоване тестування програмного забезпечення є важливою складовою успішних проектів розробки [8].

Автоматизоване тестування програмного забезпечення довгий час вважалося критичним для великих організацій, що розробляють програмне забезпечення, але часто вважається занадто дорогим або важким для невеликих компаній [8].

2.3 CI/CD

CI/CD - концепція, яка реалізується як конвейер, полегшує зв'язок тільки що закоміченого коду в основну кодову базу. Концепція дозволяє запускати різні типи тестів на кожному етапі (виповнення інтеграційного аспекту) і завершити його запуск із розгортанням закоміченого коду в фактичний продукт, який можуть бачити кінцеві користувачі (виконання доставки) [10].

CI/CD необхідні для розробки програмного забезпечення з застосуванням Agile-методології, яка рекомендує використовувати автоматичне тестування для швидкого налаштування робочого програмного забезпечення. Автоматичне тестування дає зацікавленим особам доступ до новостворених функцій і забезпечує швидкий зворотний зв'язок [11].

CI, або безперервна інтеграція - процес постійної розробки програмного забезпечення з інтеграцією в основну галузь. Автоматично збирає софт, тестує його та сповіщає, якщо щось йде не так.

CD, або безперервна доставка - процес постійної доставки ПЗ до споживача. Забезпечує розробку проекту невеликими частинами та гарантує, що його можна віддати в реліз у будь-який час без додаткових ручних перевірок.

Нажаль на даному етапі розробки проекту ХДУ 24, практика CI/CD не може використовуватись у повному масштабі - можливо забезпечити інтегрування CI частини після реалізації принаймні

базових API тестів, проте користувацький дизайн ще занадто нестабільний та не сформований для інтегрування CD.

2.4 Безперервне тестування

Безперервне тестування - це еволюція автоматичного тестування, використовує автоматизовані тести.

Головна перевага безперервного тестування - захист репутації бренду та користувацького досвіду без шкоди для релізу. Випуск нестандартного програмного забезпечення є ризиком для прибутку бізнесу [11].

Автоматичне тестування значно покращує результати ручного тестування. Але безперервне тестування доводить тестування програмного забезпечення до кінцевої фінішної прямої.

Результати, які можна отримати, якщо є можливість реалізувати безперервне тестування стабільним і здатним до масштабування [12]:

- більш швидкий зворотній зв'язок від користувачів;
- можливість швидкого відкату релізів у випадку знаходження критичних дефектів.

2.3.1 Етапи безперервного тестування

Можна виокремити основні етапи, які проходить безперервне тестування [13]:

- використання інструментів генерації автотестів з вимог та зворотного зв'язку від користувачів;
- створення тестового оточення;

- копіювання та анонімізування даних з продакшену для створення тестового датасету;
- застосування віртуальних сервісів (заглушки) для тестування API;
- паралельне тестування навантаженням.

2.3.2 Інструменти для безперервного тестування

Найбільш популярними інструментами для безперервного тестування є [14]:

1. Experitest - платформа безперервного тестування для мобільних та веб-додатків, що дозволяє проводити тести на 2000 реальних мобільних пристроях та браузерах. Вона повністю інтегруються з екосистемою розробки, тестування та безперервної інтеграції та сумісна з Appium, Selenium, Jenkins, Travis CI і т.д.
2. QuerySurge - DevOps-засіб для безперервного тестування даних. До основних функцій відносяться надійний API, аналітика даних, інтеграція в DevOps-процес для безперервного тестування і швидка перевірка великих обсягів даних.
3. Jenkins - інструмент безперервного тестування, написаний на мові програмування Java. Його можна налаштувати через графічний інтерфейс чи консольні команди.
4. Travis - інструмент безперервного тестування, розміщений на GitHub. Містить у собі безліч різних мов та розгорнуту документацію.
5. Selenium - інструмент тестування з відкритим кодом. Він підтримує всі провідні браузери: Firefox, Chrome, IE та Safari. Selenium WebDriver використовується для автоматизації веб-додатків.

2.3.3 Переваги безперервного тестування

Однією з найголовніших переваг є те, що тестування проводиться на всіх етапах розробки продукту (рис. 2.3.3.1).

Рис. 2.3.3.1 Етапи розробки програмного продукту, на яких проводиться безперервне тестування

Перевагами безперервного тестування є [14]:

- прискорення релізу продукту;
- поліпшення якості коду;
- оцінка покриття бізнес-ризиків;
- інтеграція з процесом DevOps;
- створення бази для гнучких та надійних процесів за кілька годин;
- безперервний механізм фідбеку;
- об'єднання традиційно розрізнених команд для покриття сучасних корпоративних потреб;

- усунення прогалини між розробкою, тестуванням та операціями;
- автоматизоване тестування допомагає досягти узгодженості, підтримуючи однакову конфігурацію для всіх відповідних тестів.

2.3.4 Недоліки безперервного тестування

Проблеми та недоліки з якими можна зіткнутися під час безперервного тестування [15]:

- важко відійти від звичного традиційного процесу командам розробників та QA;
- нестача навичок DevOps та правильних інструментів для тестування в Agile та DevOps середовищах;
- тестові середовища, які ніколи не відображатимуть продакшн-середовища;
- погано визначене управління тестовими даними;
- більш тривалі цикли інтеграції коду створюють проблеми з інтеграцією та запізне виправлення помилок;
- неефективні ресурси та тест-середовища;
- складна архітектура програми та бізнес-логіка, що обмежує DevOps-прийняття.

РОЗДІЛ 3. ТЕСТУВАННЯ API

3.1 Визначення тестування API

API тестування - це тип тестування програмного забезпечення, який перевіряє інтерфейси програмного забезпечення (API). Метою тестування API є перевірка функціональності, надійності, продуктивності та безпеки інтерфейсів програмування. У тестуванні API замість стандартних користувацьких вводів (клавіатури) та виводів використовується програмне забезпечення для надсилання викликів до API, отримання результатів та запису відповіді системи. Тести API дуже відрізняються від тестів GUI і не зосереджуються на зовнішньому вигляді програми [16]. Такі тести зосереджені на рівні бізнес-логіки архітектури програмного забезпечення (рис. 3.1.1).

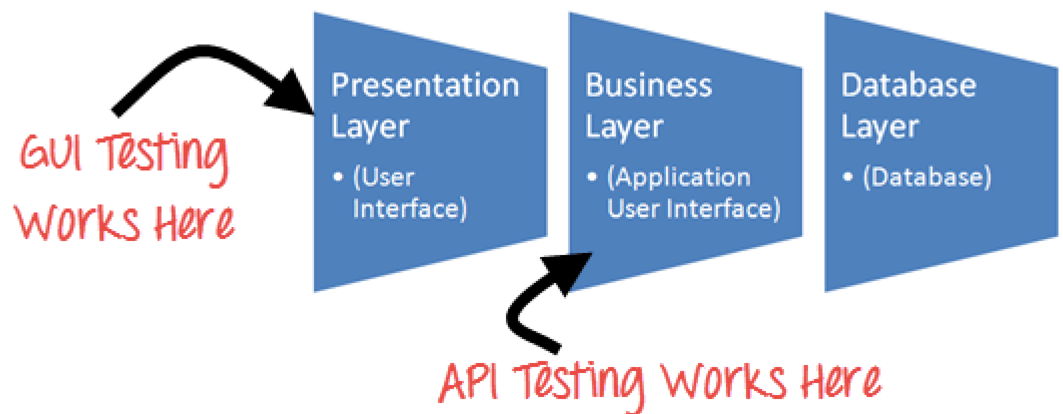


Рис. 3.1.1 Рівні, на яких тестується GUI та API

Для автоматизації тестування API потрібна програма, з якою можна взаємодіяти через API. Для цього можна використовувати [17]:

- готовий інструмент тестування для керування API;
- власний код для тестування API.

3.2 Типи API тестування

В цілому, до тестування API застосовуються наступні типи тестів[18]:

- Функціональне тестування - тести мають виконати набір викликів, задекларованих в API, щоб перевірити загальну працездатність системи.
- Юзабіліті-тестування - перевіряє, чи э функціональним API і маэ зручний інтерфейс, а також перевіряється інтеграція з іншими API.
- Тестування безпеки - перевіряє використовуваний тип аутентифікації та шифрування даних за допомогою HTTP.
- Автоматизоване тестування - створення скриптів, програм або налаштування додатків, які можуть тестувати API на регулярній основі.
- Тестування документації - перевіряється повний опис функцій API, його зрозумілість.

При тестуванні API необхідно аналізувати запити, адже це можливість виявити прихований дефект набагато швидше, ніж здійснюючи його пошук в інтерфейсі. Для цього потрібно уважно слідкувати за кодами станів та які методи використовуються для тих чи інших запитів.

За допомогою тестування API можна знайти наступні типи помилок [19]:

- збій обробки помилкових умов при передачі коректних і некоректних даних в запитах;

- зайві прапори, що не використовуються, в параметрах запитів;
- відсутня або дублююча функціональність;
- надійність серверів: труднощі при підключенні та отриманні відповіді від API;
- проблеми з безпекою;
- проблеми багатопоточності;

Для забезпечення повного тестового покриття необхідні тестові випадки для всіх можливих комбінацій вхідних даних.

Також гарна практика використання таких загальноприйнятих технік, як аналіз граничних значень і розбиття на класи еквівалентності. У запитах API в явному вигляді можуть передаватися значення параметрів. Це слушний привід виділити межі вхідних і вихідних значень і перевірити їх. Навіть у невеликого API є безліч варіантів використання та безліч комбінацій вхідних і вихідних змінних [20].

Тестування API має деякі переваги перед звичайним тестуванням через UI [20]:

- точне розуміння де відбувається помилка і чим вона викликана;
- витрачається менше часу на підготовку тестових даних за рахунок того що, більшість даних генеруються автоматично;
- можливе виконання тестів на великих обсягах даних з прийнятною швидкістю;
- можна провести тестування на ранніх етапах, коли ще немає користувацького інтерфейсу;

3.3 Інструменти для дизайну та тестування API

Автоматизований інструмент тестування API можна інтегрувати до конвеєра безперервної інтеграції. Ця інтеграція є відмінним вибором для покращення якості коду шляхом виявлення помилок на ранньому етапі життєвого циклу розробки ПЗ. Наявність правильних процесів та інструментів є критичною для тестування API [13].

3.3.1 Swagger

Інтерфейс Swagger дає змогу команді розробників візуалізувати та взаємодіяти з ресурсами API. Він автоматично генерується зі специфікації OpenAPI, а візуальна документація полегшує реалізацію серверної частини та використання на стороні клієнта [21].

Swagger дозволяє описати структуру API, щоб ПЗ мало змогу його прочитати. Читаючи структуру API, Swagger може автоматично створювати зручну та інтерактивну документацію API. Також може автоматично створювати клієнтські бібліотеки для API багатьма мовами та досліджувати інші можливості, наприклад автоматичне тестування. Swagger робить це, просить API повернути YAML або JSON, який містить детальний опис усього вашого API. Цей файл є списком ресурсів API, який відповідає специфікації OpenAPI. До специфікації можна включити інформацію, як-от [21]:

- які операції підтримує API;
- які параметри API і що він повертає;
- чи потрібна авторизація для доступу до API;
- додаткові поля з такою інформацією як умови, контакти та ліцензія на використання API.

Можна написати специфікацію Swagger для API вручну або створити її автоматично з анотацій у вихідному коді. У відкритому доступі є список інструментів, які дозволяють генерувати Swagger з коду.

При розробці бекенду для веб-сервісу ХДУ 24, були використані анотації для автоматичної генерації документації для полегшення використання API (рис. 3.3.1.1).

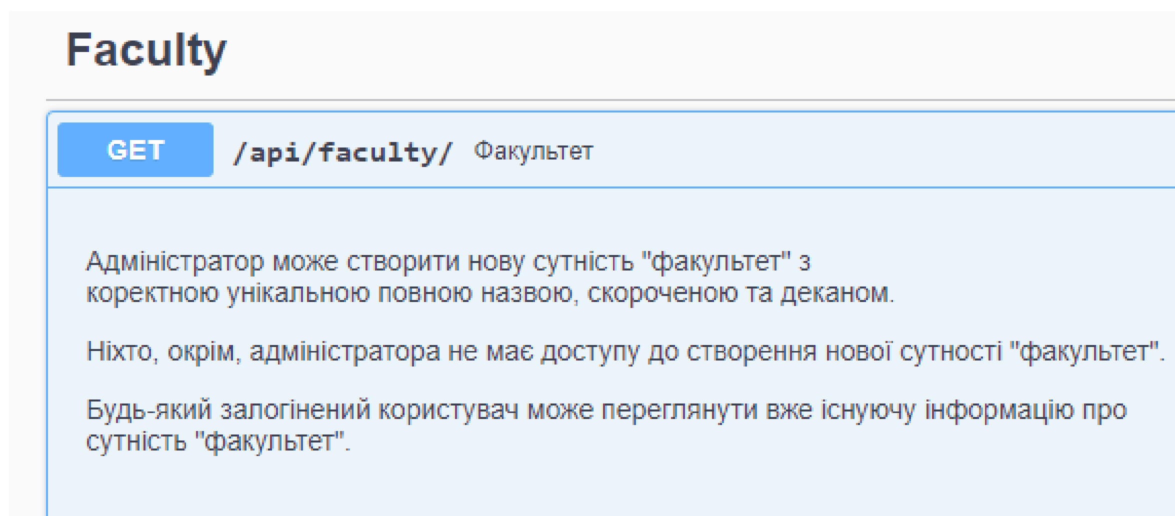


Рис. 3.3.1.1 Приклад створення документації за допомогою Swagger

3.3.2 Postman

Спочатку Postman вийшов на ринок як плагін Google Chrome. Його головною метою було тестування сервісів API. Зараз Postman розширив свої послуги як для Windows, так і для Mac [22].

За допомогою Postman можна контролювати API, створювати автоматичні тести, виконувати налагодження та виконувати запити. Postman має такі характеристики:

- інтерфейс дозволяє користувачам отримувати дані веб-API;

- дозволяє писати логічні тести і не базується на командному рядку;
- містить вбудовані інструменти, колекції та робочі області;
- підтримує різні формати, включаючи Swagger;
- загалом безкоштовний інструмент, однак є і додаткові платні функції [22].

Головним недоліком користувачі вважають те, що можливе одночасне тестування тільки одного API, проте це не грає ніякої ролі при виборі інструментів для тестування ХДУ 24.

3.3.3 API Platform

API Platform - це веб-фреймворк, розроблений для легкого створення проектів із використанням API та не ускладнює процес розширюваності та гнучкості [23]:

- можна створювати власну модель даних як звичайні старі класи PHP або імпортуйте існуючу зі словника Schema.org;
- надавати за лічені хвилини гіпермедіа REST або API GraphQL з розбиттям сторінок, перевіркою даних, контролем доступу, вбудовуванням зв’язків, фільтрами та обробкою помилок;
- вміст, який підтримується за замовчуванням - GraphQL, JSON-LD, Hydra, HAL, JSONAPI, YAML, JSON, XML і CSV;
- API (Swagger/OpenAPI) документація створюється автоматично;
- можливість додавання інтерфейсу адміністрування Material Design;
- підтримка створення середовища розробки за допомогою Docker та Kubernetes;
- підтримка автентифікації OAuth;

- специфікація та тести за допомогою інструменту тестування API для розробників.

Серверний компонент платформи API побудований на основі фреймворку Symfony, а клієнтські компоненти використовують React (також доступний варіант Vue.js). Це означає, що можна [23]:

- використовувати тисячі пакетів Symfony і компонентів React з платформою API;
- інтегрувати платформу API в будь-яку існуючу програму Symfony або React;
- перевикористовувати доступні бібліотеки та навчики, здобуті на Symphony та React.

Проте, як видно з опису API Platform, сервіс працює на Symphony та React, тож не підходить для використання на проекті ХДУ 24, оскільки останній розробляється за допомгою мови Python (конченый абзац, надо или переделать или вообще убрать этот пункт про апи платформ).

3.3.4 SAP Integration Suite

SAP Integration Suite - це інтеграційна платформа, яка дозволяє безперешкодно інтегрувати локальні та хмарні додатки та процеси з інструментами та попередньо створеним вмістом, яким керує SAP [24].

Можливості інтеграції додатків [24]:

- розробка хмарної та гібридної інтеграції з підтримкою AI;
- попередньо вбудовані інтеграції, якими керує та оновлюється SAP;

- доступ до популярних хмарних програм сторонніх розробників;
- проектування, публікація та керування API;

Інтеграційна платформа як послуга (iPaaS) надає хмарний сервіс для інтеграції додатків, даних, процесів і сервісно-орієнтованої архітектури (SOA). Це багатофункціональна платформа, яка підтримує інтеграцію «хмара-хмара», «хмара-локальний», «локальний-локальний» та B2B-інтеграцію. Сервіс підтримує інтеграцію в реальному часі та може бути масштабованим, щоб задовольнити потреби мобільного зв’язку великого обсягу; витяг, перетворення та завантаження (ETL); і середовища електронного обміну даними (EDI) [24].

3.3.5 Workato

Workato є платформою автоматизації підприємства, дозволяє як бізнес-командам, так і IT-командам інтегрувати свої програми та автоматизувати робочі процеси без шкоди для безпеки та управління. Це дозволяє компаніям отримувати результати від ділових заходів у реальному часі [25].

Написання коду не вимагається, а платформа використовує машинне навчання та запатентовану технологію, щоб зробити створення та впровадження автоматизації швидше, ніж традиційні платформи [25].

Суттєвим є те, що є тільки безкоштовний пробний період, після закінчення якого у будь-якому випадку необхідно буде сплачувати за

підписку на сервіс. У випадку з ХДУ 24 це є ключовим фактором, а отже, сервіс не є пріоритетним для вибору.

3.3.6 UUID

UUID (Universal Unique Identifier) - це 128-бітове значення, яке використовується для однозначної ідентифікації об’єкта або сутності в Інтернеті. Залежно від використовуваних конкретних механізмів, UUID або гарантовано відрізнятиметься, або, принаймні, дуже ймовірно, буде відрізнятися від будь-якого іншого UUID, створеного до 3400 року нашої ери [26].

UUID можна створювати, щоб посилатися на майже все, що можна уявити. Наприклад, вони можуть ідентифікувати бази даних, екземпляри системи, первинні ключі, профілі Bluetooth або об’єкти з коротким терміном життя.

UUID - це термін, аналогічний GUID. Спочатку GUID посилався на варіант UUID, який використовується Microsoft, але терміни стали синонімами в специфікації RFC 4122. UUID був стандартизований Open Software Foundation (OSF), ставши частиною розподіленого обчислювального середовища (DCE). Різні версії UUID відповідають специфікації RFC 4122 [26].

UUID генеруються за допомогою алгоритму на основі позначки часу та інших факторів, таких як мережева адреса (рис. 3.3.6.1). Безкоштовні інструменти для створення UUID включають UUIDTools або Online UUID Generator.

На проекті ХДУ 24 унікальні ідентифікатори UUID використовуються та назначаються автоматично при створенні екземпляру будь-якого класу, тож усі об’єкти можна однозначно ідентифікувати, маючи лише номер UUID.

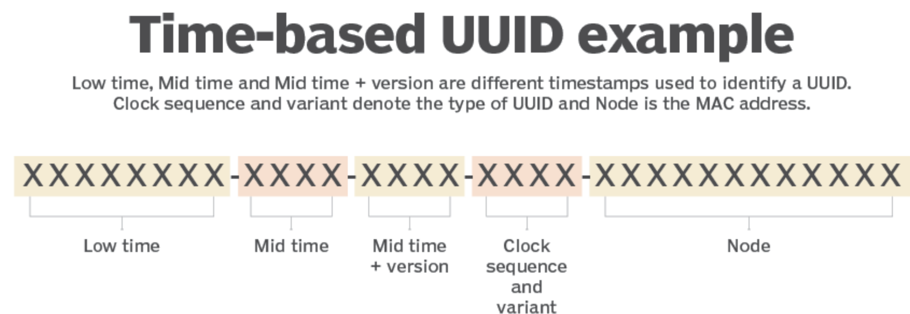


Рис. 3.3.6.1 Приклад генерації UUID, що сприається на позначку в часі

Існують такі версії UUID [26]:

1. v1 (унікальний) генерується за допомогою комбінації MAC-адреси хост-комп’ютера та поточної дати та часу. На додаток до цього, він також вводить ще один випадковий компонент, щоб бути впевненим у його унікальності.
2. v4 (випадковий) - біти, що містять UUID v4, генеруються випадковим чином і без логіки. Через це неможливо визначити інформацію про джерело за допомогою UUID. Однак тепер є ймовірність, що UUID може бути продубльований. Хоча тут нема причин для хвилювання, отже величезною кількістю можливих комбінацій (2^{128}) є майже неможливим створити дублікат, тільки якщо не генеруються трильйони ідентифікаторів кожну секунду протягом багатьох років.

3. v5 (не випадковий) - На відміну від v1 або v4, UUID v5 генерується шляхом надання двох частин вхідної інформації:
    a. вхідний рядок: будь-який рядок, який можна змінити у програмі;
    b. простір імен: фіксований UUID, який використовується в поєднанні з вхідним рядком, щоб розрізняти UUID, згенеровані в різних програмах, і для запобігання злому таблиць;

    Ці дві частини інформації перетворюються в UUID за допомогою алгоритму хешування SHA1.

    Важливо зауважити, що UUID v5 є послідовним. Це означає, що будь-яка дана комбінація введення та простору імен призведе до того самого UUID щоразу.

За попередньою оцінкою передбачається зберігання великого об’єму даних та генерування близько 10^6 об’єктів у базі даних ХДУ 24 [27], яким буде присвоєно унікальний ідентифікатор UUID v4. І хоча UUID v4 генерується повністю випадковим чином, усі комбінації можна вичерпати тільки за умови, що генеруватися буде один трильйон ключів кожної наносекунди протягом десяти мільярдів років [27].

3.4 REST API

REST API (також відомий як RESTful API) - це інтерфейс програмування прикладних програм (API або веб-API), який відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з веб-сервісами RESTful. REST розшифровується як

репрезентаційна передача стану і був створений комп’ютерним науковцем Роєм Філдінгом [10].

REST - це набір архітектурних обмежень, а не протокол чи стандарт. Розробники API можуть реалізувати REST різними способами.

Коли запит клієнта здійснюється через API RESTful, він передає уявлення про стан ресурсу запитувачу або ендпоінту. Ця інформація надається в одному з кількох форматів через HTTP: JSON, HTML, XLT, Python, PHP або звичайний текст. JSON є найбільш популярним форматом файлів, оскільки він не залежить від мови, а також він є досить легким для людського розуміння, проте і комп’ютери його також розуміють.

Слід мати на увазі ще дещо: заголовки та параметри також важливі в методах HTTP HTTP-запиту RESTful API, оскільки вони містять важливу інформацію про ідентифікатор щодо метаданих запиту, авторизації, єдиного ідентифікатора ресурсу (URI), кешу, файлів cookie та більше. Існують заголовки запиту та відповіді, кожен із власною інформацією з’єднання HTTP та кодами стану [11].

Для того, щоб API вважався RESTful, він повинен відповідати цим критеріям [14]:

- клієнт-серверна архітектура, що складається з клієнтів, серверів і ресурсів, із запитами, які керуються через HTTP;
- клієнт-серверний зв’язок без стану - це означає, що інформація про клієнта не зберігається між запитами на отримання, і кожен запит є окремим і не залежним;

- дані з кешуванням, які спрощують взаємодію клієнт-сервер;
- єдиний інтерфейс між компонентами, щоб інформація передавалася у стандартній формі;
- багатошарова система, яка організовує кожен тип серверів (відповідальних за безпеку, навантаження тощо), передбачала отримання запитуваної інформації в ієрархії, невидимі для клієнта;
- code-on-demand (опціонально): можливість надсилати виконуваний код із сервера клієнту за запитом, що розширює функціональні можливості клієнта;

Хоча API REST має відповідати цим критеріям, він все ще вважається легшим у використанні, ніж встановлений протокол, як-от SOAP (Протокол доступу до об’єктів), який має конкретні вимоги, такі як обмін повідомленнями XML, а також вбудовану безпеку та відповідність транзакціям, які роблять його повільніше і важче [13].

На відміну від цього, REST - це набір рекомендацій, які можна запровадити за потреби, що робить REST API швидшими та легшими, з підвищеною масштабованістю - ідеально підходить для Інтернету речей (IoT) та розробки мобільних додатків.

3.4.1 HTTP

Протокол передачі гіпертексту (HTTP) є основою всесвітньої павутини і використовується для завантаження веб-сторінок за допомогою гіпертекстових посилань. HTTP - це протокол прикладного рівня, призначений для передачі інформації між мережевими пристроями і працює поверх інших рівнів стеку мережевих протоколів.

Типовий потік через HTTP передбачає, що клієнтська машина робить запит до сервера, який потім надсилає повідомлення-відповідь [13].

Кожен запит HTTP, зроблений через Інтернет, несе з собою серію закодованих даних, які несуть різні типи інформації. Типовий HTTP-запит містить [12]:

- тип версії HTTP;
- URL;
- метод HTTP;
- заголовки запиту HTTP;
- необов'язкове тіло HTTP.

Метод HTTP вказує на дію, яку запит HTTP очікує від запитуваного сервера. Наприклад, два з найпоширеніших методів HTTP - це «GET» і «POST»; запит «GET» очікує повернення інформації (зазвичай у формі веб-сайту), тоді як запит «POST» зазвичай вказує, що клієнт подає інформацію на веб-сервер (наприклад, інформацію про форму, наприклад, надіслане ім'я користувача та пароль) [11].

Заголовки HTTP містять текстову інформацію, що зберігається в парах ключ-значення, і вони включені в кожен запит і відповідь HTTP (рис. 3.4.1.1). Ці заголовки передають основну інформацію, наприклад, який браузер використовує клієнт, які дані запитуються [15].

```
▼ Request Headers
  :authority: www.kspu.edu
  :method: GET
  :path: /RssAggregator.ashx?Name=News&lang=uk
  :scheme: https
  accept: application/rss+xml
  accept-encoding: gzip, deflate, br
  accept-language: en,en-GB;q=0.9,uk-UA;q=0.8,uk;q=0.7,ru-UA;q=0.6,ru;q=0.5,en-US;q=0.4
  origin: https://ksu24.kspu.edu
  referer: https://ksu24.kspu.edu/
  sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
  sec-ch-ua-mobile: ?1
  sec-ch-ua-platform: "Android"
  sec-fetch-dest: empty
  sec-fetch-mode: cors
  sec-fetch-site: same-site
  user-agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gec
  ko) Chrome/99.0.4844.82 Mobile Safari/537.36
```

Рис. 3.4.1.1 Приклад заголовків HTTP-запитів сайту ХДУ

Коди статусу HTTP - це 3-значні коди, які найчастіше використовуються для вказівки на те, чи успішно виконано запит HTTP. Коди стану розбиті на наступні 5 блоків [16]:

1. 1xx - інформаційний;
2. 2xx - успіх;
3. 3xx - перенаправлення;
4. 4xx - помилка клієнта;
5. 5xx - помилка сервера,

де «xx» відноситься до різних чисел від 00 до 99.

Коди статусу, що починаються з цифри «2», вказують на успіх. Наприклад, після того, як клієнт запитує веб-сторінку, найчастіше зустрічаються відповіді мають код статусу «200 OK», що вказує на те, що запит було виконано належним чином.

Якщо відповідь починається з «4» або «5», це означає, що сталася помилка, і веб-сторінка не відображатиметься. Код статусу, який починається з «4», вказує на помилку на стороні клієнта (дуже часто зустрічається код статусу «404 NOT FOUND (не знайдено)», коли є помилка в URL-адресі). Код стану, що починається з «5», означає, що на стороні сервера щось пішло не так. Коди стану також можуть починатися з «1» або «3», що вказує на інформаційну відповідь і переадресацію відповідно.

3.4.2 CRUD

CRUD - це абревіатура, яка відноситься до чотирьох функцій, які вважаються необхідними для реалізації програми постійного зберігання [17]:

- create (створення) - процедури генерують нові записи за допомогою операторів INSERT;
- read (читання) - зчитують дані на основі вхідних параметрів. Аналогічно, процедури RETRIEVE захоплюють записи на основі вхідних параметрів;
- update (оновлення) - змінюють записи, не перезаписуючи їх;
- delete (видалення) - видаляють дані там, де зазначено.

Організації, які відстежують дані клієнтів, облікові записи, платіжну інформацію, дані про стан здоров’я та інші записи, потребують обладнання та програм для зберігання даних, які забезпечують постійне зберігання. Ці дані зазвичай організовуються в базу даних, яка є просто організованою колекцією даних, які можна переглядати в електронному вигляді. Існує багато типів баз даних: ієрархічні бази даних, графічні бази даних та об’єктно-орієнтовані

бази даних тощо. Найпоширенішим типом бази даних є реляційна база даних, яка складається з даних, представлених у рядках і стовпцях і з'єднаних з іншими таблицями з додатковою інформацією за допомогою системи ключових слів, яка включає первинні та зовнішні ключі [17].

Абревіатура CRUD визначає всі основні функції, які притаманні реляційним базам даних і додаткам, які використовуються для керування ними, до яких належать Oracle Database, Microsoft SQL Server, MySQL та інші.

CRUD можна зіставити з протоколами DDS, SQL і HTTP. Протоколи HTTP є сполучною ланкою між ресурсами в архітектурі RESTful, що є основною частиною основи REST [18].

Відображення принципів CRUD на REST означає розуміння того, що GET, PUT, POST і CREATE, READ, UPDATE, DELETE мають разючу схожість, оскільки перше групування застосовує принципи останнього.

У сервісі ХДУ 24 використовується база даних PostgreSQL, яка э реляційною, а отже всі принципи CRUD та архітектурні рішення REST можуть та є використаними на проекті.

3.5 Розробка тест-кейсів для ХДУ 24

Перш ніж починати тестування API, важливо зрозуміти, що робить API і які його обов'язки. Цей крок має початися з отримання доступу до документації API. Слід вручну здійснити деякі виклики ендпоінтів

API, щоб отримати чітке розуміння того, як вони працюють і які дані повертають (рис. 3.5.1).

Це також допомагає виявити неточності та незрозумілі аспекти документації API, оскільки іноді вона читається вперше. Це може бути чудовою можливістю вдосконалити документи API. Документація ХДУ 24 не є розгорнутою, частіше дуже стисла, отже цей аспект вимагає більшої подальшої уваги та розвитку.

Маючи розуміння API та приклади його використання, можна починати тестування окремих ендпоінтів на їх очікувану поведінку (наприклад, GET/person/2 повинен повертати JSON, що описує особу з ідентифікатором 2 (рис.3.5.2)).

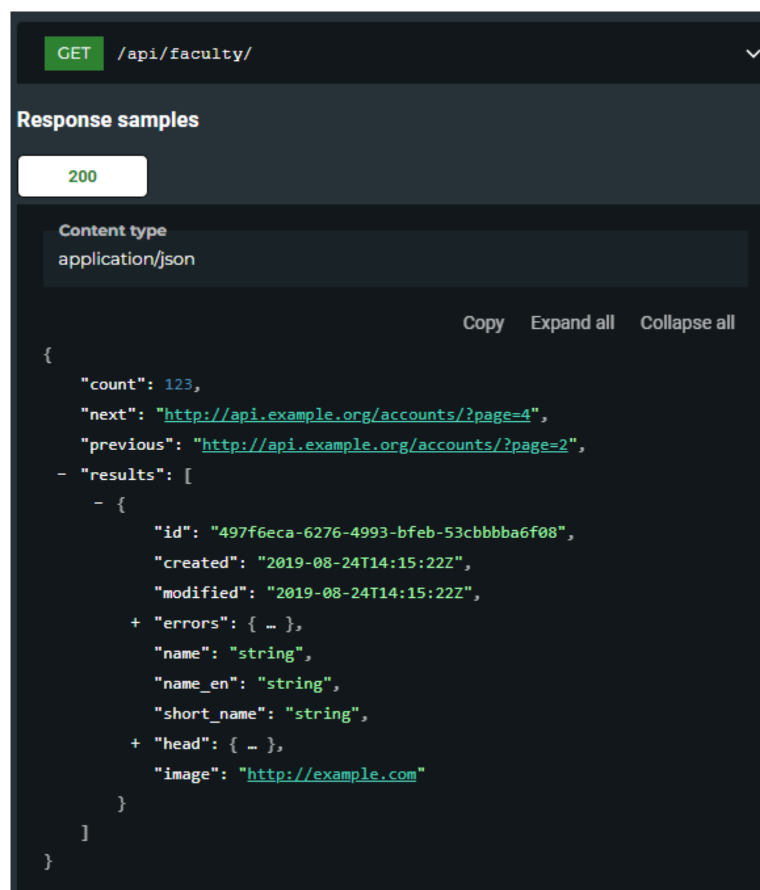


Рис. 3.5.1 Приклад API відповіді на GET запит

Корисні тести повинні мати кілька ознак [19]:

- атомний: кожен тест повинен бути автономним і не повинен покладатися на інші тести, які виконуються до/після нього, або на інші дані.
- переносний: будь-які дані, конфігурації та особливо URL-адреси повинні передаватися як параметри, щоб забезпечити переносимість між середовищами.
- повторюваний: повинна бути можливість часто повторювати тести без будь-яких процесів між запусками (наприклад, очищення даних вручну).

```python
def test_person_can_be_retrieved_via_api(self):
    alice, alice_token = self.create_and_login_user(
        "alice@example.com", role=ROLE_CHOICES.RoleAdmin
    )
    self.client.force_authenticate(user=alice.user)
    user = self.create_user(username='bob@example.com')
    json = {
        'user_id': '2',
        'user': user,
        'surname': user.last_name,
        'name': user.first_name,
        'image': ''
    }

    response = self.client.post(self.url, json)
    response = self.client.get(self.url + response.json()['id'] + '/')

    self.assertEqual(response.status_code, 200)
```

Рис. 3.5.2 Тест-кейс API GET/person/2

Також дуже важливо приділити достатню увагу тому, як API веде себе правильно в крайніх і негативних випадках, наприклад:

- чи повертається правильний код помилки (відповідь 404) на запит до неіснуючого об’єкту (рис. 3.5.3)?

```python
def test_retrieve_not_existed_user_via_api(self, ):
    alice, alice_token = self.create_and_login_user(
        "alice@example.com", role=ROLE_CHOICES.RoleAdmin)
    self.client.force_authenticate(user=alice.user)

    response = self.client.get(self.url + '10' + '/')

    self.assertEqual(response.status_code, 404)
```

Рис. 3.5.3 Тест-кейс із відповіддю Not Found (не знайдено)

- як працює API , якщо надано аргумент неправильного типу даних (відповідь 400) (рис. 3.5.4)?

```python
def test_faculty_can_not_be_created_with_wrong_data_via_api(self):
    alice, alice_token = self.create_and_login_user(
        "alice@example.com", role=ROLE_CHOICES.RoleAdmin
    )
    self.client.force_authenticate(user=alice.user)

    json = {
        'name': "Факультет комп'ютерних наук, фізики та математики",
        'short_name': 7,
        'head_id': alice.worker_set.first().id,
    }
    response = self.client.post(self.url, json)

    self.assertEqual(response.status_code, 400)
```

Рис. 3.5.4 Тест-кейс із відповіддю Bad Request (невірний запит)

- що станеться (рис. 3.5.5), якщо спробувати отримати доступ до об’єкта, на який у поточного користувача немає дозволів (відповідь 403)?

Щоб полегшити створення негативних тест-кейсів доцільним є перегляд списку кодів помилок HTTP (особливо діапазон 4xx – помилки користувача) і спроба створити сценарії для створення кожного з цих кодів.

Розробники, які використовують API, очікують, що будуть повернуті правильні помилки. Важливо забезпечити належну поведінку API під час виникнення помилок.

```
def test_person_can_not_be_retrieved_with_no_permission_via_api(self):
    alice, alice_token = self.create_and_login_user(
        "alice@example.com", role=ROLE_CHOICES.RoleAdmin
    )
    self.client.force_authenticate(user=alice.user)
    bob, bob_token = self.create_and_login_user(
        "bob@example.com", role=ROLE_CHOICES.RoleStudent
    )
    self.client.force_authenticate(user=bob.user)

    response = self.client.get(self.url + str(alice.user.id) + '/')

    self.assertEqual(response.status_code, 403)
```

Рис. 3.5.5 Тест-кейс із відповіддю Forbidden (заборонено)

Створення тест-кейсів для ХДУ 24 було розпочато із складання таблиці всіх ендпоінітів та пріоритезації кожного з них. В результаті проведеного огляду отримано 82 ендпоінта, та 4 категорії пріорітетів - від 0 до 3, де 0 - тестування не потрібне, 1 - найвищий пріоритет, а 3 - найнижчий.

Загалом, було складено та реалізовано 53 тест-кейса до 14 ендпоінтів.

ВИСНОВКИ

Для виконання поставлених завдань було проведено аналіз основних теоретичних понять, що стосуються методів, рівнів та способів тестування веб-сервісів. При підготовці до розробки було приділено увагу особливостям обробки даних та складання тест-кейсів.

На основі проведеного аналізу розроблено базове тестування можливостей API додатку.

Суттєву частку роботи приділено аналізу існуючих технологій всіх рівнів для тестування та менеджменту API веб-додатків. Детально досліджено тестування роботи клієнт-серверних додатків.

Відповідно до створених вимог розроблено тест-план для веб-додатку, зокрема його API частини. Спроектовано та реалізовано найбільш пріоритетні тест-кейси для таких модулів, як «Авторизація», «Користувач», «Працівник», «Студент», «Дисципліна», «Факультет» та інших згідно з таблицею [1] з використанням засобів Python.

Важливо зауважити, що, вже на початковому етапі створення тест-плану та аналізу існуючих моделей, було виявлено, а згодом і усунено, 4 суттєвих дефекти, пов’язані з рівнями доступу користувачів до даних, а отже і з безпекою всього застосунку. Також були зроблені зауваження щодо реалізації окремих модулів.

При написанні ключових частин використано спеціальну форму коментарів, що забезпечують інтеграцію опису функцій та їх параметрів в підказки популярних IDE (інтегрованих середовищ розробки). Останнє є корисним при подальшій розробці, особливо при

використанні існуючої кодової бази сторонніми розробниками, що є цілком можливим, зважаючи на модульність проекту.

Була реалізована подальша траєкторія розробки API-тестів для застосунку ХДУ 24.

При розробці проекту використовується система контролю версій git з приватним репозиторієм на сервісі GitHub[25], що дозволяє використовувати сучасні методи сумісної роботи та, одночасно з тим, дозволяє використовувати результати проведеного дослідження всім, у кого є доступ до репозитарію.

ДОДАТКИ

Додаток А

**КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИХЕРСОНСЬКОГО
ДЕРЖАВНОГО УНІВЕРСИТЕТУ**

Я, *Кісельгоф А.Є.*

учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

–не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні йполітичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

–не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

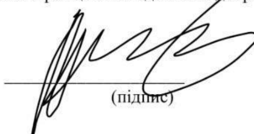
–не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

–не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

–не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальностій до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

10.03.2021		*Анна Кісельгоф*
(дата)	(підпис)	(ім'я, прізвище)

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тест-кейси ХДУ-24 URL: https://ksu24.kspu.edu/s/apitestcase (дата звернення: 28.01.2022).
2. Bicevska Z., Bicevskis J. Applying Self-Testing: Advantages and Limitations //Databases and Information Systems V-Selected Papers from the Eighth International Baltic Conference, DB&IS 2008. – 2008. – С. 2-5.
3. How did a software bug caused the failure of 1.2 billion military satellite launch in April 1999 (Cape Canaveral launch)? URL: https://www.quora.com/How-did-a-software-bug-caused-the-failure-of-1-2-billion-military-satellite-launch-in-April-1999-Cape-Canaveral-launch (дата звернення: 24.01.2022).
4. Report software failures cost 1.1 trillion URL: http://servicevirtualization. com/report-software-failures-cost-1-1-trillion-2016/ (дата звернення: 24.01.2022).
5. Harrold M. J. Testing: a roadmap //Proceedings of the Conference on the Future of Software Engineering. – 2000. – С. 61-72.
6. Certified Tester Foundation Level Syllabus https://www.istqb.org/downloads /send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html (дата звернення: 24.01.2022).
7. Алексеев Д. М., Иваненко К. Н., Убирайло В. Н. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ //СОВРЕМЕННЫЙ ВЗГЛЯД НА БУДУЩЕЕ НАУКИ. – 2016. – С. 6-7.
8. Шарафиев Д. Е. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ //ББК 1 А28. – 2020. – С. 88.

9. Рыбалко М. А., Иванова Е. А. Тестирование программного обеспечения, методы тестирования //Информационное общество: современное состояние и перспективы развития. – 2017. – С. 320-322.
10. Лямкин Д. Г. REST API АРХИТЕКТУРА ВЗАИМОДЕЙСТВИЯ СИСТЕМЫ ПРОЕКТИРОВАНИЯ С МЕХАНИЗМАМИ ФУНКЦИОНАЛЬНО-СТОИМОСТНОГО АНАЛИЗА //ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА (ИВТ-2020). – 2020. – С. 125-129.
11. Fielding R., Reschke J. Hypertext transfer protocol (HTTP/1.1): Semantics and content. – 2014.
12. Jestratjew A., Kwiecien A. Performance of HTTP protocol in networked control systems //IEEE Transactions on Industrial Informatics. – 2012. – Т. 9. – №. 1. – С. 271-276.
13. Skvorc D., Horvat M., Srbljic S. Performance evaluation of Websocket protocol for implementation of full-duplex web streams //2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). – IEEE, 2014. – С. 1003-1008.
14. Murphy L. et al. Preliminary analysis of REST API style guidelines //Ann Arbor. – 2017. – Т. 1001. – С. 48109.
15. Smith F. D. et al. What TCP/IP protocol headers can tell us about the web //Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems. – 2001. – С. 245-256.
16. Niveditha B., Kumbar M. HTTP Error Codes of Inaccessible and Retrieved URLs in LIS Journal Articles //Library Philosophy and Practice. – 2021. – С. 0_1-15.

17. Singjai A. et al. Patterns on Designing API Endpoint Operations. – 2021.
18. Corradini D. et al. Restats: A test coverage tool for RESTful APIs //2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). – IEEE, 2021. – С. 594-598.
19. Bennett B. E. A practical method for API testing in the context of continuous delivery and behavior driven development //2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). – IEEE, 2021. – С. 44-47.
20. Гардт В. Е. АНАЛИЗ API И АВТОМАТИЗАЦИЯ ИХ ТЕСТИРОВАНИЯ //XXIII Всероссийская студенческая научно-практическая конференция Нижневартовского государственного университета. – 2021. – С. 213-216.
21. API Documentation & Design Tools for Teams URL: https://swagger.io/ (дата звернення: 01.04.2022).
22. Postman API Platform URL: https://www.postman.com/ (дата звернення: 01.04.2022).
23. API Platform URL: https://github.com/api-platform/api-platform (дата звернення: 01.04.2022).
24. SAP Integration Suite URL: https://www.sap.com/products/integration-suite.html (дата звернення: 01.04.2022).
25. The Modern Leader in Automation | Workato URL: https://www.workato.com/ (дата звернення: 01.04.2022).
26. Aftab H. et al. Analysis of identifiers in IoT platforms //Digital Communications and Networks. – 2020. – Т. 6. – №. 3. – С. 333-340.

27. Сенчишен Д. О. Проєктування та розроблення сервісної архітектури управління бізнес-процесами університету. Сервіс «Відділ кадрів».