**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**Факультет комп'ютерних наук, фізики та математики**
**Кафедра інформатики, програмної інженерії та економічної кібернетики**

# ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ «СЕРВІС ДОСТАВКИ ЇЖІ»

**Кваліфікаційна робота (проєкт)**

на здобуття ступеня вищої освіти "бакалавр"

Виконав: студент 4 курсу
Спеціальності 122 Комп'ютерні науки та інформаційні технології
Освітньо-професійної програми «Комп'ютерні науки та інформаційні технології»
першого (бакалаврського) рівня освіти
Чистяков Олександр Дмитрович
Керівники: старший викладач
Черненко Ірина Євгенівна,
кандидат фізико-математичних наук,
доцент Єрмолаєв Вадим Анатолійович
Рецензент: кандидат фізико-математичних наук, доцент
Бистрянцева Анастасія Миколаївна

Херсон – 2020

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| | |
|---|---|
| RUP | Rational Unified Process |
| ПЗ | Програмне забезпечення |
| СУБД | Система управління бази даних |
| GUI | Graphical User Interface |
| SCL | Script Control Language |
| WAPT | Web Application Testing |

# ВСТУП

**Актуальність теми**. Процес тестування при розробці сучасного програмного забезпечення займає значну частину часу і зусиль. Найчастіше він зводиться до аналізу поведінки програми на спеціально підібраному наборі тестів. Оскільки перебір всіх можливих ситуацій неможливий на практиці, намагаються вибрати невелике число тестів, за допомогою яких можна побачити поведінку програми при використанні тесту та судити про її поведінку в цілому. Використовуються різні метрики покриття коду для кількісної оцінки якості набору тестів. Якісним вважається набір тестів, що задовольняє деякому критерію повноти. Часто критерій повноти визначається за допомогою обраної метрики покриття коду. Автоматизація процесу побудови набору тестів дозволить істотно скоротити витрати на тестування. Існує безліч підходів до автоматизованої побудови набору тестів. Однак лише деякі з них мають програмну реалізацію, наприклад Microsoft Pex. Всі ці реалізації націлені на модульне тестування, при якому покриття фрагментів програми відбувається незалежно одне від одного. Однак внаслідок цього велика кількість згенерованих тестів містять неприпустимі значення для тестованих фрагментів коду, оскільки не враховується внутрішня структура програми. Як бачимо, у сучасній розробці якісного програмного забезпечення тестування програмних систем, вибір оптимальних технологій тестування, автоматизація тестуючого процесу для забезпечення високої якості програмних продуктів є актуальним.

**Метою** даного дослідження є реалізація ручного та автоматизованого тестування веб-застосунку «сервіс доставки їжі».

Відповідно до мети дослідження визначено такі **завдання**:

1. Проаналізувати існуючі методи тестування.

1. Розглянути детально методи тестування.

2. Знайти найбільш ефективні інструменти для тестування даного веб-сервісу.

3. Протестувати веб-сервіс.

4. Прослідкувати запобігання дефектів у роботі прогромного засобу.

**Об'єктом** дослідження є система автоматизованого та навантажувального тестування веб-застосунків.

**Предметом** дослідження є процес тестування систем веб-застосунку «сервіс доставки їжі».

**Методи** дослідження:

- тестування програмного забезпечення;
- визначення якості продукту;
- верифікація програмного забезпечення;
- автоматизація тест-кейсів.

**Практичне значення** отриманих результатів полягає у залученні досліджуваного матеріалу для підготовки до тестування веб-сервісу. Матеріали кваліфікаційної роботи можуть бути використані при написанні курсових робіт, рефератів та інших наукових розвідок.

**Структура** дослідження. Кваліфікаційної робота складається з вступу, двох розділів, висновків та списку використаних джерел.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1. Процес тестування у життєвому циклі продукту

Тестування є процесом аналізу програмного забезпечення, націлений на знаходження відмінностей між існуючими і необхідними властивостями[5].

Відповідно до ГОСТ Р МЕК ІСО 12207-99 у життєвому циклі програмного забезпечення ідентифікуються серед інших процесів перевірки, сертифікації та спільного аналізу такі процеси як процес перевірки і процес атестації.

*Процес перевірки* - це процес визначення функціювання програмного продукту у відповідності з вимогами або умовами, реалізованими на попередньому етапі роботи. Даний процес може включати аналіз, перевірку та випробування (тестування).

*Процес атестації* - це процес, який визначає повноту відповідності програмного продукту до вимог, встановлених до системи або її функціональному призначенню.

*Процес спільного аналізу* - це процес оцінки стану і, при необхідності, результатів роботи проєкту.

*Процес аудиту* - це процес визначення відповідності вимогам, планам і умовами контракту.

У цілому ці процеси становлять те, що зазвичай називають тестуванням.

Тестування опирається на тестові процедури з зазначеними початковими умовами, вхідними даними, очікуваним результатом, розробленим для визначеної мети, наприклад, перевірка окремої програми або верифікація відповідності означеним вимогам. Процедури

тестування можуть перевіряти різні концепції функціонування кінцевої продукції - від правильної роботи відокремленої функції до адекватного виконання зазначених бізнес-вимог.

Під час реалізації проєкту також необхідно мати на увазі, відповідно до яких стандартів і вимог буде проводитися тестування цього продукту. Які інструменти та засоби застосовувалися для пошуку та ведення документації по знайденим дефектам. Якщо не забувати про тестування з самого початку виконання проєкту, тестування продукту, який розробляється, не завдасть неприємних сюрпризів. А тоді й якість продукту, швидше за все, буде більш досконалою[5].

Все частіше використовуються ітеративні процеси розроблення програмного забезпечення, зокрема, технологія RUP - Rational Unified Process. На рисунку 1.1 можна побачити життєвий цикл продукту по RUP. При використанні такого підходу тестування перестає бути процесом «на відчепі» після того, як програмісти закінчили весь необхідний код. Робота над тестами завжди починається з етапу виявлення вимог, сформованих замовником до майбутнього продукту і тісно інтегрується з поточними завданнями. І це висуває нові вимоги тестувальникам. Їх роль не зводиться лише до виявлення помилок у можливій повноті. Вони повинні приймати участь в загальному процесі пошуку та усунення найбільш істотних ризиків проєкту якомога раніше. Для цього на кожну ітерацію визначається мета та методи тестування. А в кінці кожної ітерації визначається, наскільки ця мета досягнута, чи потрібні додаткові випробування або ні, і чи не потрібно змінити інструменти та принципи проведення тестів. У свою чергу, кожен виявлений недолік повинен пройти через свій життєвий цикл.


Рис. 1.1 Життєвий цикл продукту по RUP

Зазвичай тестування проводиться циклами, кожен з яких має конкретний список завдань та цілей. Цикл тестування може збігатися з ітерацією або відповідати її певній частині. Як правило, цикл тестування проводиться для конкретної збірки системи. Життєвий цикл програмного продукту, який зображений на рисунку 1.2 складається з певної кількості коротких ітерацій.

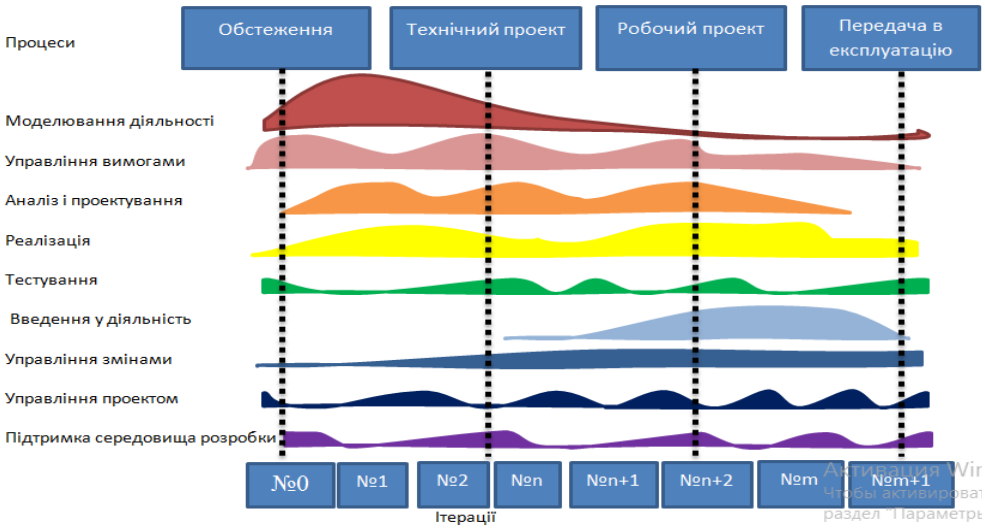



Рис. 1.2 Ітерації життєвого циклу програмного продукту

*Ітерація* - є закінченим циклом розроблення, який приводить до випуску кінцевого продукту або деякої його скороченої версії, яка розширюється від однієї ітерації до іншої, щоб, врешті-решт, стати завершеною системою. Кожна ітерація включає, як правило, аналіз та завдання планування робіт проєктування, реалізацію тестування та оцінку досягнутих результатів. Однак ці завдання можуть суттєво змінюватися. У відповідності до поставленої задачі, в ітерації вони групуються у фази.

Першою фазою є - *Початок* - основна увага приділяється завданням аналізу.

Другою фазою є - *Розроблення* - основна увага приділяється проєктуванню і випробуванню ключових проєктних рішень.

У третій фазі - *Побудова* - найбільш велика частка задач розроблення та тестування.

Кінцевою фазою є - *Передача* - вирішуються в найбільшій мірі завдання тестування і передача замовнику системи.

Фаза вважається виконаною коли всі цілі досягнуті за своєю специфікою в життєвому циклі продукту. По заверщенню всіх ітерацій, крім, ітерацій фази Початок, сворюються функціонуючої версії розроблюваної системи[6].

## 1.2. Місце тестування у структурі розроблення програмного забезпечення

Побудова фірми - розробника програмного забезпечення - відображає етапи життєвого циклу програмного засобу. Виконання робіт по одному або декільком етапам життєвого циклу ПЗ забезпечує один із підрозділів.

*Аналітичний відділ*. Аналітичний відділ забезпечує:

- визначення задуму і функціонального спрямування розвитку програмного продукту;
- проведення передпроєктного обстеження;
- визначення, на що здатна система;
- визначення (спільно з розробниками) технічних вимог до системи;
- опис в термінах бізнес-процесів предметної області, постановки завдань і специфікації на розроблення;
- при зміні законодавства, вимог клієнтів, розширенні функціональних можливостей продукту, написання постановок задач і специфікацій на доопрацювання програмного засобу;
- контроль процесу реалізації нових можливостей в програмних продуктах компанії.

*Відділ документації.* Зокрема даний відділ не виділяється в відокремлену структуру, він може входити до складу аналітичного відділу. Завданням відділу зазвичай є написання технічної документації для кінцевого користувача, відстеження змін в програмному засобі і актуалізація в документації.

*Відділ розроблення*. Є ключовим відділом для фірми. Якщо без інших відділів зазвичай можна обійтися, то без відділу розроблення ні.

У його завдання входять:

- дефініція технічних вимог до системи (спільно з аналітиками);
- розширення переліку функцій програмного засобу (реалізація доробок);
- виправлення помилок;
- реалізація основних функцій програмного засобу;
- адаптивість продукту для функціонування в інших умовах (перехід на нову СУБД, нову мову програмування та ін.)

- збільшення швидкодії, надійності та ін. в продукті.

*Відділ тестування*. До завдань відділу входять:

- контроль якості;
- підготовка документації тестування (плани та ін.);
- виявлення та локалізація помилок у функціонуванні програмних продуктів;
- фіксування і відстеження помилок у функціонуванні програмних засобів;
- перевірка відповідності документації програмного продукту стандартам і реально реалізованим функціям;
- участь в розробці та впровадженні системи якості;
- автоматизація тестування;
- оцінка продуктивності розроблювальних програмних засобів на різних програмно-апаратних платформах і їх специфічних змінах.

У деяких компаніях на відділ тестування покладаються збірка та випуск програмного забезпечення, в інших цим займається відділ розроблення[7].

Всі відділи компанії працюють як єдиний механізм, дані взаємодії впорядковані між собою і представляють виробничі технологічні процеси. Технологічні процеси, як правило, регламентовані внутрішніми документами або внутрішньокорпоративними стандартами та в сукупності представляють собою технологічний цикл виробництва програмного засобу.

Типових технологічних ланцюжків всередині компанії - тобто розробника програмного забезпечення, велика кількість. Як приклад розглянемо схему взаємодії відділу тестування програмного забезпечення з іншими відділами при виявленні помилки під час функціонування програмного забезпечення (рисунок 1.3).

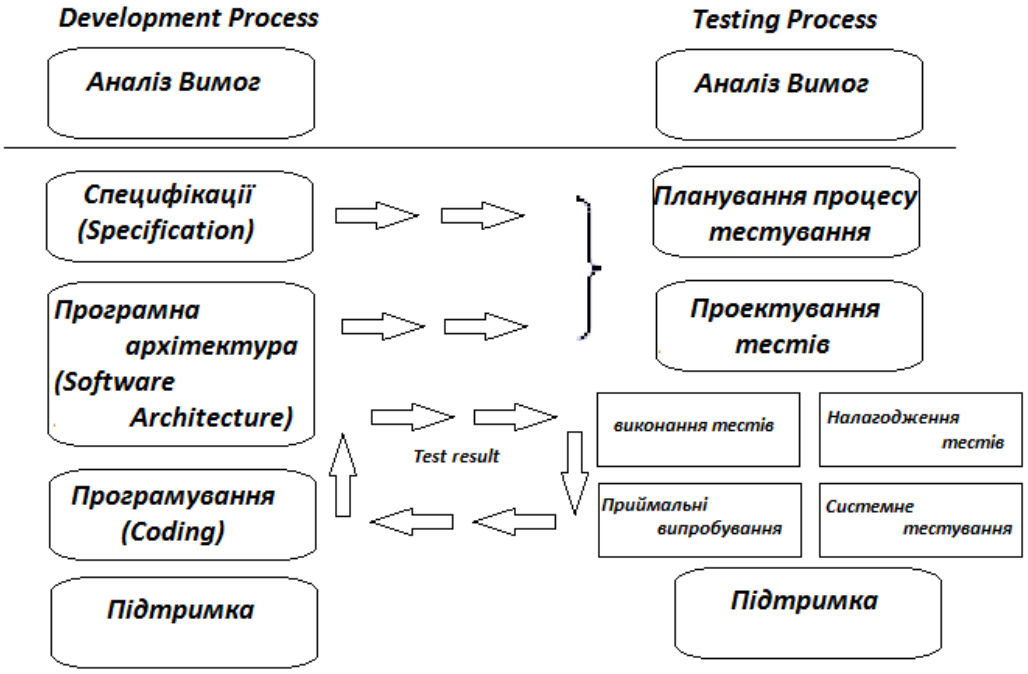



Рис. 1.3 Схема взаємодії відділу тестування програмного забезпечення з іншими відділами

## 1.3. Види та рівні тестування

### *Види тестування*

Існує систематизування видів тестування, яке засноване на тому рівні деталізації робіт проєкту, на який воно спрямоване.

На малюнку 1.4 зображені рівні тестування. Ці ж різновиди тестування можна пов'язати з фазою життєвого циклу, на якій вони виконуються.

*Модульне тестування* або (Unit-testing) - рівень тестування, на якому тестується мінімально ймовірний компонент для тестування, наприклад, певний клас або функція. На цьому рівні застосовуються методи «білого ящика». У сучасних проєктах модульне тестування («юніт-тестінг») здійснюється розробникам[9].

Модульне тестування необхідне для перевірки коректності деяких модулів, незалежно від їх середовища. "При цьому перевіряється, що якщо модуль отримує на вхід дані, що задовольняють певним критеріям коректності, то й коректними можна вважати результати. Для описання критеріїв правильності вхідних та вихідних даних часто використовують програмні контракти - передумови, що описують для кожної операції, на яких вхідних даних вона призначена, постумови, що описують для кожної операції, співвідношення вхідних даних з вихідними результатами, і інваріанти, що визначають критерії цілісності внутрішніх даних модуля." Основний недолік модульного тестування полягає у тому, що проводити його можна, тільки коли перевіряється вже закінчений елемент програми. Готуючи тести можна знизити вплив цього обмеження, (а це - найбільш трудомістка частина тестування) наперед на основі вимог, коли вихідного коду ще немає. Важливою часткою налагоджувального тестування, для вдосконалення написаного розробниками написаного коду є модульне тестування.

*Інтеграційне тестування* (Integration testing) - рівень тестування, на якому окремі модулі програмного забезпечення об'єднуються і тестуються в групі. Частіше за все, інтеграційне тестування проводиться вже після модульного тестування (юніт-тести для модулів повинні бути виконані і знайдені помилки виправлені) [9].

Системне тестування (System testing) - це тестування програмного забезпечення, яке проводиться для перевірки відповідності системи оригінальним вимогам до повної інтегрованої системи. Тестування системи відноситься до методів тестування "чорної скриньки", не вимагає знання внутрішнього пристрою системи. Тестування системи здійснюється за допомогою зовнішніх програмних інтерфейсів і тісно пов'язане з тестуванням користувальницького інтерфейсу (або через інтерфейс користувача), що проводиться шляхом моделювання дій користувача на елементах цього інтерфейсу. Окремими випадками цього

виду тестування є тестування графічного інтерфейсу користувача (Graphical User Interface, GUI) і призначеного для кінцевого користувача інтерфейсу Web-додатків (WebUI). Системне тестування виконується інженерами по тестуванню.

*Приймальне тестування* (Acceptance testing) - є тестуванням готового продукту кінцевими користувачами на реальному оточенні, в якому буде функціонувати тестований додаток. Приймальні тести розробляються користувачами, зазвичай, у вигляді сценаріїв. Для того, щоб знайти більше помилок рекомендується планувати не тільки системне тестування і приймальне, але і модульне і інтеграційне[5].

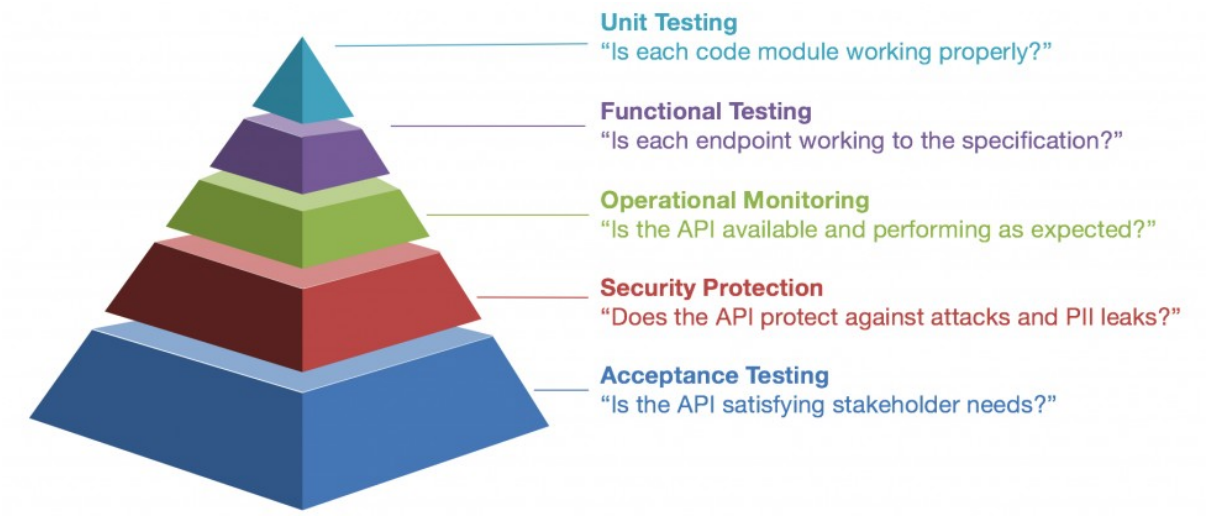



Рис. 1.4 Рівні тестування

*Статичне тестування* (Static testing) - тестування, в ході якого тестована програма (код) не виконується (запускається). Аналіз програми здійснюється на основі кінцевого вихідного коду, який вираховується вручну, або аналізується за допомогою спеціальних інструментів.

Приклади статичного тестування:

- огляди (Reviews); інспекції (Inspections);
- наскрізні перегляди (Walkthroughs);

- аудити (Audits).

*Динамічне тестування* (Dynamic testing) - тестування, в ході якого тестована програма (код) виконується (запускається). Альфа-тестування - тестування в процесі розроблення. Бета-тестування - тестування виконується користувачами (end-users). До того як випускається програмне забезпечення, воно має пройти стадії альфа (або внутрішнього пробного використання) та бета (пробного використання продукту із залученням зовнішніх вибраних користувачів) версій. Звіти про знайдені помилки, що надходять від вибраних користувачів поданих версій продукту, обробляються відповідно до визначених процедур, які включають тести (будь-якого рівня), які проводяться фахівцями групи розроблення. Такий вид тестування заздалегідь не може бути спланований.

«Смок-тест» (Smoke Testing, «димове тестування») в тестуванні це мінімальний набір тестів на виявлення явних помилок. Димовий тест зазвичай виконується програмістом; якщо програма не проходить цей тест, то її не має сенсу віддавати на глибше тестування[5].

***Види тестування*:**

*Функціональне тестування* (Functional testing) - мета даного тестування полягає у тому, щоб переконатися у якісному та повному функціонуванні об'єкта тестування:

- кожна функціональна вимога транслюється в тест-кейси (використовуючи техніки «чорного ящика») для того, щоб перевірити, що система функціонує в точності, як і описано в специфікації (функціональних вимогах до системи);
- перевіряється, чи всі функціональні вимоги дійсно запрограмовані реалізовані.

*Тестування продуктивності* (Perfomance testing) - тестування, яке виконується із метою виявлення, як швидко працює під певним

навантаженням частина системи або система у цілому. Також може використовуватись для перевірки та підтвердження атрибутів якості системи, таких як надійність масштабованість та споживання програмним засобом ресурсів.

Існує особливий вид тестів, коли робиться спроба досягнення кількісних границь, обумовлених характеристиками системи і її оточення:

- продемонструвати, що при заданих умовах система задовольняє критеріям продуктивності;
- виміряти, яка частина системи є причиною «поганої» продуктивності системи;
- виміряти час реакції на дію користувача, час відгуку на запит, і т.д. [11].

*Тестування навантаження* (Load testing) - це найлегша форма для тестування продуктивності. Зазвичай тестування навантаження проводиться для оцінки поведінки самої програми під заданим навантаженням. Цим навантаженням може бути, кількість одночасно працюючих користувачів програмного засобу, що здійснюють вказану кожному кількість транзакцій за відрізок часу. Поданий вищу тип тестування зазвичай дозволяє отримати середній час відгуку усіх важливих операцій. У разі спостереження за мережею, базою даних або сервером додатків[11].

*Тестування стабільності* (або Stability testing) проводиться завжди з метою переконатися, що додаток протягом тривалого відрізку часу витримує виділене навантаження. При проведенні цього виду тестування здійснюється спостереження за споживанням програмним засобом ліміту пам'яті, для того щоб виявити потенційні витоки. Таке тестування може виявити деградацію продуктивності програмного засобу, яке

виражається у більщій мірі у зниженні швидкості обробки інформації або збільшенні часу відповіді програмного засобу після тривалої.

*Тестування зручності використання* (Usability testing) - дослідження, яке виконується з метою визначення, чи зручний деякий штучний об'єкт (такий як вебсторінка, призначений для користувача інтерфейс чи пристрій) для його передбачуваного застосування. Таким чином, перевірка ергономічності вимірює ергономічність систем або об'єкта. Перевірка ергономічності програмного засобу зосереджена на вибраному об'єкті або невеликому наборі таких об'єктів[11].

*Тестування інтерфейсу користувача* (UI testing) - тестування графічного інтерфейсу користувача для того, щоб запевнитись, що він відповідає прийнятим стандартам і їх вимогам. Перевіряється, як додаток обробляє ввод з клавіатури і «мишки» і як відображаються елементи графічного інтерфейсу (текст, кнопки, меню, списки та інші елементи).

*Тестування безпеки* (security testing) - оцінка уразливості програмного забезпечення до врідоносних атак. Комп'ютерні системи нерідко є цілью незаконного проникнення в систему. Під таким проникненням говориться про великий діапазон дій: спроби групи невідомих проникнути в систему з невідомими цілями, помста службовців, злом шахраями для наживи. Тестування безпеки виявляє фактичну реакцію програмного засобу та його захисних механізмів, вбудованих у нього, на проникнення. Під час процесу такого тестування безпеки випробувач відіграє роль хакера.

Хакеру дозволено все:

- дізнатися пароль за допомогою будьяких зовнішніх засобів;
- атака на систему за допомогою утиліт, які аналізують захист;
- пригнічення, приголомшення системи (в надії, що вона відмовить, або відмовиться обслуговувати інших користувачів);

- цілеспрямоване введення хакером помилок в надії протрапити в систему в ході її оновлення;
- перегляд даних в надії знайти ключ або доступ для входу в систему.

*Тестування локалізації* (Localization testing) - багатогранна річ, що має на увазі перевірку безлічі аспектів, пов'язаних з адаптацією продукту для користувачів з інших країн. Наприклад, тестування локалізації для користувачів з Кореї може полягати в перевірці того, чи не видасть система помилку, якщо цей користувач на сайті знайомств введе розповідь про себе символами Kanji, а не англійською шрифтом.

*Тестування сумісності* (або Compatibility testing) - є видом нефункціонального тестування, метою якого є перевірка коректної роботи програмного засобу в певному оточенні. Оточення може включати в себе такі елементи:

- мережеві пристрої;
- апаратна платформа;
- системне програмне забезпечення (файрвол, веб-сервер, антивірус, ...);
- периферія (CD / DVD-приводи, принтери, веб-камери, ...);
- операційна система (Windows, Unix, MacOS, ...);
- браузери (Internet Explorer, Safari, Firefox, Opera, Yandex, Chrome, Safari).
- бази даних (Oracle,Realm, MS SQL, MySQL, ...).

## 1.4. Тест план

***Тест план*** (або Test Plan) - являє собою документ, який описує велику кількість роботи з напрямку тестування, починаючи з стратегії, опису об'єкта, розкладу, критеріїв та початку закінчення тестування, до

необхідного в процесі тестування роботи самого обладнання, знань та оцінки ризиків із варіантами їх вирішення.

"Тест план містить:

1) перелік рекомендованих тестових ресурсів;

2) перелік функцій та підсистем, які підлягають тестуванню;

3) стратегію тестування:

- аналіз функцій та підсистем для виявлення слабких сторін, що вимагають вичерпного тестування, тобто функціональних можливостей, де найбільш імовірно виникає дефект;
- визначення стратегії вибору вхідних даних для тестування (оскільки у реальних додатках набір вхідних даних програмного продукту є нескінченним, вибір остаточної підмножини для тестування є складним завданням, для вирішення якого можна застосувати методи покриття класів вхідних та вихідних даних, аналізу екстремальних значень, охоплення випадків використання тощо; обрана стратегія тестування має бути обгрунтована та задокументована);
- визначення необхідності автоматизації процесу тестування (слід обґрунтувати рішення щодо використання існуючої або створення нової автоматичної системи тестування, а також оцінку вартості для створення нової системи або впровадження існуючої);

4) графік (або розклад) циклів тестування;

5) вказівку конкретних параметрів апаратного та програмного оточення;

6) визначення тестових метрик, які необхідно складати і проводити аналіз, таких як покриття коду, обсяг тестового коду, покриття набору вимог, кількість і рівень серйозності дефектів.

Тест план повинен відповідати на такі питання:

1) *що тестувати*? Опис об'єкта тестування: додатки, системи, обладнання та ін.;

2) *що тестувати*? Список функцій і опис тестованої системи та її компонент;

3) *як тестувати*? Стратегія тестування: методології, види тестування та їх застосування по відношенню до об'єкту тестування, пріоритети, автоматизація та ін .;

4) *коли тестувати*? Послідовність робіт: фази, цикли тестування, процедура тестування - підготовка (Test Preparation), тестування (Testing), аналіз результатів (Test Result Analisys) у запланованих фазах розроблення;

5) де тестувати?

- оточення тестованої системи - її опис;
- необхідне обладнання для тестування та програмні засоби.

6) *хто буде тестувати*?

- ролі та обов'язки;
- імена і прізвища.

7) *критерії початку тестування*:

- готовність тест кейсів;
- готовність оточення;
- закінченість розроблення необхідного функціоналу;
- виконання юніт-тестів;
- білд побудований і задовольняє певним критеріям

8) *критерії закінчення тестування*:

- результати тестування задовольняють певним критеріям;
- вимоги до кількості відкритих багів виконані (визначаються заздалегідь).

9) *критерії передачі системи для приймального тестування*:

- приймальні випробування - де зберігаються і коли виконуються;
- процедура приймання.

10) *які ризики існують і як ми ними керувати*?

11) *метрики і звіти*:

- продуктові метрики - хто збирає, з якою періодичністю;
- звіти - хто створює, кому відправляє, і т.п.

12) *розклад білдів*.

Відповівши в тест-плані на ці запитання, можна вважати, що є документ з плануванням тестування. Щоб документ набув більш серйозний вигляд, пропонується доповнити його такими пунктами:

- оточення тестованої системи;
- опис програмно-апаратних засобів;
- необхідні для тестування програмні засоби та обладнання (тестовий стенд та його конфігурація, програмні продукти для автоматизованого тестування та ін.);
- ризики та виріанти їх вирішення. [4]

# РОЗДІЛ 2
# МЕТОДИКА ТЕСТУВАННЯ

## 2.1. Тестовий випадок (Test case). Види. Структура.

***Тестовий випадок*** (Test Case) – це:

- мінімальний елемент тестування (всього одна верифікація \ перевірка);
- сукупність конкретних умов, кроків і атрибутів, необхідних для перевірки тестування реалізації певної функції або їх частин;
- опис певних дій і умов, які необхідні для того, щоб виявити той чи інший баг.

Під терміном тест кейсом розуміють структуру виду:

Action > Expectd Results > Test Results.

У таблиці 2.1 показаний приклад test case.

*Таблиця 2.1 Приклад тестового* випадку

| ***Action*** | ***Expected Result*** | ***Test Result*** |
|---|---|---|
| Open web page "Login" | Login web page is opened | Passed... |

***Види Тестових Випадків***

Тест кейси діляться за очікуваним результатом на позитивні та негативні:

- позитивний тест кейс бере тільки валідні дані і перевіряє, що програмний продукт вірно здійснює функцію, що викликається;
- негативний тест кейс використовує як вірні так і не вірні дані (повинен бути хоча б один некоректний параметр) і ставить на перевірку виняткові ситуації (спрацьовування валідаторів), також

перевіряє, щоб функція яка викликається програмним засобом не виконувалася при спрацьовуванні такого валідатора.

***Структура Тестових Випадків (або Test Case Structure)***

Кожен тест кейс має складатися з трьох частин. У таблиці 2.2 показані ці частини.

*Таблиця 2.2 Структур*а Test case

| | |
|---|---|
| Pre conditions | Список послідовних дій, які призводять програмний засіб до такого придатного для проведення перевірки стану. Або список таких умов, виконання яких говорить про те, що система знаходиться в оптимальному для проведення тесту стані. |
| Test case description | Список таких дій, які переводять програмний засіб з одного стану в інший для того щоб отримати результат, на підставі якого можна зробити висновок про задоволення реалізації поставленим замовником вимогам |
| Post conditions | Список дій, які переводять систему у початковий стан (до проведення тесту - initial state) |

Примітка: Post Conditions є не обов'язковою частиною. Ця частина актуальна при автоматизованому тестуванні, тобто за один прогін можливо наповнити базу даних *n*-кількістю некоректних документів.

*Приклад тест кейсу*

do N1, verify D1

do N2, verify D2

do N3, verify D3

У наведеному прикладі відображена кінцева перевірка - D3. Це означає, що вона є ключовою. Отже, N1 і N2 - це дії, які призводять систему до стану тестопридатності, а D1 і D2 - умови того, що система знаходиться у стані, придатному для тестування. Таким чином, у таблиці 2.3 показано умову тест кейсу.

*Таблиця 2.3 Умова тест кейсу*

| Action | Expected Result | Test Result (is passed/is failed/is blocked) |
|---|---|---|
| Preconditions | | |
| do N1 | verify D1 | |
| do N2 | verify D2 | |
| Test Case. Description | | |
| do N3 | verify D3 | |
| Postconditions | | |

PostConditions в даному прикладі описані не були, але за логікою прийнято виконати кроки, які повернули б систему в початковий стан. (Наприклад, скасувати зміни, зроблені в документі, або видалили створений запис).

Потрібно відповісти на питання: "Чому розбиття такого виду зручно використовувати?". Відповідь у таблиці 2.4: кінцева перевірка одна, тобто в разі якщо тест провалений (test failed) буде відразу ясно, через що. Якщо виявляться провальними перевірки D1 або D2, то тест-case буде заблокований (test will be blocked), тому що функцію неможливо привести у придатний для тестування стан, але це не означає, що тестована функції не працює.

*Таблиця 2.4 Один з варіантів р*езультату

| Action | Expected Result | Test Result (is passed/is failed/is blocked) |
|---|---|---|
| Preconditions | | |
| do N1 | verify D1 | is passed |
| do N2 | verify D2 | is failed |
| Test Case Description | | |
| do N3 | verify D3 | is blocked |
| Postconditions | | |

### 2.2. Атрибути тест кейсу

У таблиці 2.5 представлені часто використовувані атрибути тест кейсу.

*Таблиця 2.5 Атрибути тест кейсу*

| Атрибут тест кейса | Опис |
|---|---|
| Test Case ID | Унікальне значення в межах не тільки документа, але і всієї фірми |
| Test Case Priority | Пріоритет. Вимірюється від 1 до n<br>- найвищий 1<br>- найнижчий n (для не дуже великих проєктів раціонально вибирати n = 4) |
| IDEA | Опис ідеї, що перевіряється тест кейсом |
| SETUP and ADDITIONAL INFO | Вхідні параметри |
| Revision History | Історія редагування. Приклад формату:<br>"Created on <date> by <name><br>Modified on <date> by <name><br>Change - які зміни і навіщо вони"[2] |

| | |
|---|---|
| Execution Part | "Здійснена частина тест кейсу.<br>Приклад формату:<br>Action - список команд<br>EXPECTED RESULT - або очікуваний результат<br>TEST RESULT - (blocked, passed, failed)"[2] |

Приклад тест кейсу: нехай для тестування перегляду публікацій на сторінці необхідно виконати наступний алгоритм:

1) відкрити веб-застосунок “сервіс доставки їжі”;

2) натиснути кнопку «Пошук»;

3) заповнити поле пошуку;

4) перейти на сторінку;

5) знайти модуль з публікаціями;

6) перейти на перше посилання;

7) запросити базу даних "select res from public where id = <номер публікації>".

Очікуваний результат: "Сторінка з інформацією та посиланням на публікацію"

У таблиці 2.6 можна побачити як для даного прикладу буде виглядати тест кейс. Для проходження одного і того ж сценарію на N наборах тестових даних створюється N тестів. Цьому правилу необхідно слідувати. Таким чином, щоб перейти на іншу публікацію, або перегляну інформацію, необхідно скопіювати вміст тест кейсу VV12345, наприклад, у тест кейс VV12346 і переписати тільки вхідні параметри. Такий вид тест кейсу називають керованим даними (data-driven).

Проблеми сценарію в прикладі:

- пункти 1-4 можуть змінюватися в зв'язку з міграцією сайту на новий домен, зміною інтерфейсу і т.д.;
- пункти 5-7 можуть бути легко змінені за рахунок нового дизайну сайту.

Отже, вбудь-якому разі доведеться переписувати весь тест кейс, щоб заново протестувати первинне завдання.

*Таблиця 2.6 Тест кейс для пр*икладу

| Test Case ID/Priority | VV1234567890 | 1 |
|---|---|---|
| 1) відкрити веб-застосунок “сервіс доставки їжі”;<br>2) натиснути кнопку «Пошук»;<br>3) заповнити поле пошуку;<br>4) перейти на сторінку;<br>5) знайти модуль з публікаціями;<br>6) перейти на перше посилання;<br>7) запросити базу даних "select res from public where id = <номер публікації>". | | |
| Revision History | | |
| Created on 23/01/2020 by Салат «Цезарь» | Новий тест кейс | |
| Execution Part | | |
| ACTION | EXPECTED RESULT | TEST RESULT |
| 1) відкрити веб-застосунок “сервіс доставки їжі”;<br>2) натиснути кнопку «Пошук»;<br>3) заповнити поле пошук;<br>5) перейти на сторінку; | Сторінка з інформацією та посиланням на публікацію | |

| | | |
|---|---|---|
| 6) знайти модуль з публікаціями;<br>7) перейти на перше посилання;<br>8) запросити базу даних "select res from public where id = <номер публікації>". | | |

Якщо ж розбити задачу на кілька тест кейсів, наприклад, за пункт 1 відповідатиме тест кейс "Вхід в систему", за 2-4 - "Пошук страви", 5-8 - "Вибір публікації" (на внутрішньому рівні кожного з них можливий ще більш детальний поділ), то можна переписати тест кейс так, як вказано в таблиці 2.7.

*Таблиця 2.7 Спрощення тест кейсу*

| | | |
|---|---|---|
| Test Case ID/Priority | VV12345 | 1 |
| IDEA: Пошук страви<br>SETUP and ADDITIONAL INFO: Аккаунт: user / password<br>Інформація стосовно страви: Морепродукти<br>Телефон для підтвердження: +38 (0595) 00-54-72<br>E-mail: Restaurant@gmail.com<br>Публікації: Дивитися<br>Запит SQL: select res from public where id = <номер публікації> | | |
| Revision History | | |
| Created on 01/01/2020 by Морепродукти | Новий тест кейс | |
| Modified on 01/01/2020 by Салати | Спрощення тест кейсу | |
| Execution Part | | |
| ACTION | EXPECTED RESULT | TEST RESULT |

| | | |
|---|---|---|
| 1) увійти до системи;<br>2) знайти страву;<br>3) знайти розділ «Публікації»;<br>4) перейти за посиланням;<br>5) запросити базу даних "select res from public where id = <номер публікації>". | Сторінка з інформацією та посиланням на публікацію | |

### 2.3. Структура баг репорту

Різні системи відслідковування помилок, пропонують різні поля для заповнення та відмінні структури опису помилок. У таблиці 2.8 вказано шаблон баг репорту.

*Таблиця 2.8 Шаблон баг репор*ту

| Шапка(Hat) | |
|---|---|
| Короткий опис (Summary) | Короткий опис існуючої проблеми, (вказує на причину та тип помилкової ситуації) |
| Проєкт (Project) | Назва продукту тестування |
| Компонент додатку (Component) | Назва функції або частини продукту тестування |
| Номер версії (Version) | Версія на якій була зафіксована помилка |
| Серйозність (Severity) | Поширена п'ятирівнева система градації серйозності дефекту:<br>• p1 Блокуючий (або Blocker); |

| | |
|---|---|
| | • p2 Критичний (або Critical);<br>• p3 Значний (або Major);<br>• p4 Незначний (або Minor);<br>• p5 Тривіальний (або Trivial). |
| Пріоритет (Priority) | Пріоритети дефекту:<br>• k1 Високий (High priority);<br>• k2 Середній (Medium priority);<br>• k3 Низький (Low priority). |
| Автор (Author) | |
| Призначено на (Assigned To) | |
| Оточення | |
| OS/Service-pack / Browser та Version/… | Інформація про оточення, у якому знайшли баг або помилку: OS/Service-pack / Browser та Version/…- ім'я та версія браузера |
| Опис | |
| Кроки відтворення багу (Steps to Reproduce) | Кроки, відтворюючі ситуацію, яка призвела до помилки. |
| Фактичний результат процесу (Result) | Результат, отриманий після проходження кроків до відтворення |
| Очікуваний результат процесу (Expected Result) | Очікуваний результат |
| Доповнення | |
| Приєднаний файл (Attachment) | Файл з логами процесу, Screenshot або інший документ, який проясняє причину помилки та може вказати на спосіб вирішення проблеми |

## 2.4. Написання баг репорту

Баг репорт - є технічним документом, у зв'язку з чим мова опису проблеми має бути технічною. Також має використовуватися правильна термінологія при створені назв елементів призначеного для користувача інтерфейсу (PopUp, Menu, DataPicker, TextBox, Label, ViewController, Combobox, DropDown ), дій користувача (Gest, click, hold, move, touch) і отримані результати (View is opened, crash report, info view).

У таблиці 2.9 представлені основні поля баг репорту і роль працівника, відповідального за заповнення даного поля.

*Таблиця 2.9 Заповнення полів баг репорту*

| Шапка | |
|---|---|
| Короткий опис (Summary) | Автор репорту помилки (зазвичай тестувальник) |
| Проєкт (Project) | Автор репорту помилки (зазвичай тестувальник) |
| Компонент додатку (Component) | Автор репорту помилки (зазвичай тестувальник) |
| Номер версії (Version) | Автор репорту помилки (зазвичай тестувальник) цей атрибут може бути змінений менеджером |
| Серйозність(Severity) | Менеджер проєкту або відповідальний менеджер за розроблення компоненту, до якого написаний баг репорт |
| Пріоритет (Priority) | Автор репорту помилки (зазвичай тестувальник), але багато систем баг трекінгу виставляють статус за замовчуванням |
| Автор (Author) | Встановлюється за |

| | |
|---|---|
| | замовчуванням, якщо немає, то вказується ім'я автора баг репорту |
| Призначено на (Assigned To) | Менеджер проєкту або менеджер відповідальний за розроблення компоненту, на який написаний баг репорт |
| ОС / СервісПак тощо / Браузер + версія /… | Автор баг репорту (зазвичай тестувальник) |
| Кроки відтворення (Steps to Reproduce) | Автор баг репорту (зазвичай тестувальник) |
| Фактичний результат (Result) | Автор баг репорту (зазвичай тестувальник) |
| Очікуваний результат (Expected Result) | Автор баг репорту (зазвичай тестувальник) |
| Приєднаний файл (Attachment) | Автор баг репорту (зазвичай це Тестувальник), а також будь-який член командної групи, який вважає, що прикріплені дані допоможуть у виправленні бага |

### 2.5. Автоматизоване навантажувальне тестування

Щоб обговорювати підходи до тестування навантаження і проблеми, які вирішуються з його допомогою, потрібно почати з основ - термінології [9].

1) віртуальний користувач (Virtual User) - програмний процес, циклічно виконує моделюються операції (приклад: простий користувач, який хоче скористатися системою);

2) ітерація (Iteration) - є одиним повтором виконуваної в циклі операції (приклад ітерації: вхід в систему -> проведення аналізу -> отримання результату аналізу -> вихід з системи);

3) інтенсивність виконання операції (Operation Intensity) - частота виконання операції в одиницю часу, в тестовому скрипті задається інтервалом часу між ітераціями (приклад: три користувача, які входять в систему через хвилину після того, як зайшов попередній користувач);

4) навантаження (Loading) - виконання операцій на загальному ресурсі (приклад: загальна кількість виконаних операцій в системі трьома користувачами за певний проміжок часу);

5) продуктивність (Performance) - кількість виконуваних операцій за виділиний період часу (якщо система не може виконати певну кількість операцій за даний період часу, то вона починає гальмувати, що означає брак продуктивності);

6) масштабованість додатку (Application Scalability) - пропорційне зростання продуктивності при збільшенні навантаження;

7) профіль навантаження (Performance Profile) - набір операцій з заданими інтенсивностями, отриманий на основі збору статистичних даних або певним шляхом аналізу вимог до тестованої системи (також називається сценарієм (скриптом));

8) навантажувальна точка - розрахована (або задана вимогами) кількість віртуальних користувачів у групах, що виконують операції з певними інтенсивностями.

Потрібно розглянути як ці сутності пов'язані між собою. Виразивши інтенсивність через інтервал часу між ітераціями, можна побачити, що зростання інтенсивності виконуваних операцій - це скорочення інтервалу часу. Зростання навантаження пропорційне зростанню інтенсивності. Також, при збільшенні інтенсивності зростає продуктивність. При цьому збільшується ступінь використання (завантаженості) ресурсів. З якогось моменту зростання продуктивності припиняється (а навантаження може продовжувати рости), відбувається насичення і потім деградація системи. На додаток можна помітити, що

при тестуванні зміна інтенсивності операцій може підкорятися якомусь закону (наприклад, Пуассона) або бути рівномірним протягом усього тесту.

***Огляд програм для навантажувального тестування***

Здаючи веб-сервер у повсякденну експлуатацію потрібно бути впевненим, що він витримає заплановане навантаження. Тільки створивши умови, наближені до бойових, можна оцінити, чи достатня потужність системи, чи правильно встановлені ті додатки, які беруть участь у створенні веб-контенту, та інші чинники, що впливають на роботу веб-сервера. У цій ситуації на допомогу прийдуть спеціальні інструменти, які допоможуть дати якісну і кількісну оцінку роботи як веб-вузла в цілому, так і окремих його компонентів.

***OpenSTA*** (рисунок 2.1) - більше ніж додаток для тестів, це відкрита архітектура, спроєктована навколо відкритих стандартів. Проєкт створений в 2001 році групою компаній CYRANO, яка підтримувала комерційну версію продукту, але CYRANO розпався, і зараз OpenSTA поширюється як додаток з відкритим кодом під ліцензією GNU GPL. Для роботи вимагає Microsoft Data Access Components (MDAC), який можна завантажити з сайту корпорації [12].

Поточний інструментарій дозволяє провести навантажувальне випробування HTTP / HTTPS сервісів, хоча його архітектура здатна на більше. OpenSTA дозволяє створювати тестові сценарії на спеціалізованій мові SCL (Script Control Language). Всі параметри запиту піддаються редагуванню, можлива підстановка змінних. Реалізована можливість організації розподіленого тестування, що підвищує реалістичність, або коли з одного комп`ютера не виходить навантажити потужний сервер. Кожна з машин такої системи може виконувати свою групу завдань, а repository host здійснює збір і зберігання результатів.

Після установки на кожній системі, що тестується, запускається сервер імен, робота якого обов'язкова. Результати тестування, які включають час відгуків, кількість переданих байт в секунду, коди відповіді для кожного запиту і кількість помилок виводяться у вигляді таблиць і графіків. Використання великого числа фільтрів дозволяє відібрати необхідні результати. Результат можна експортувати в CSV-файл. Можливості щодо виведення звітів дещо обмежені, але за посиланнями на сайті можна знайти скрипти і плагіни, що спрощують аналіз отриманої інформації.

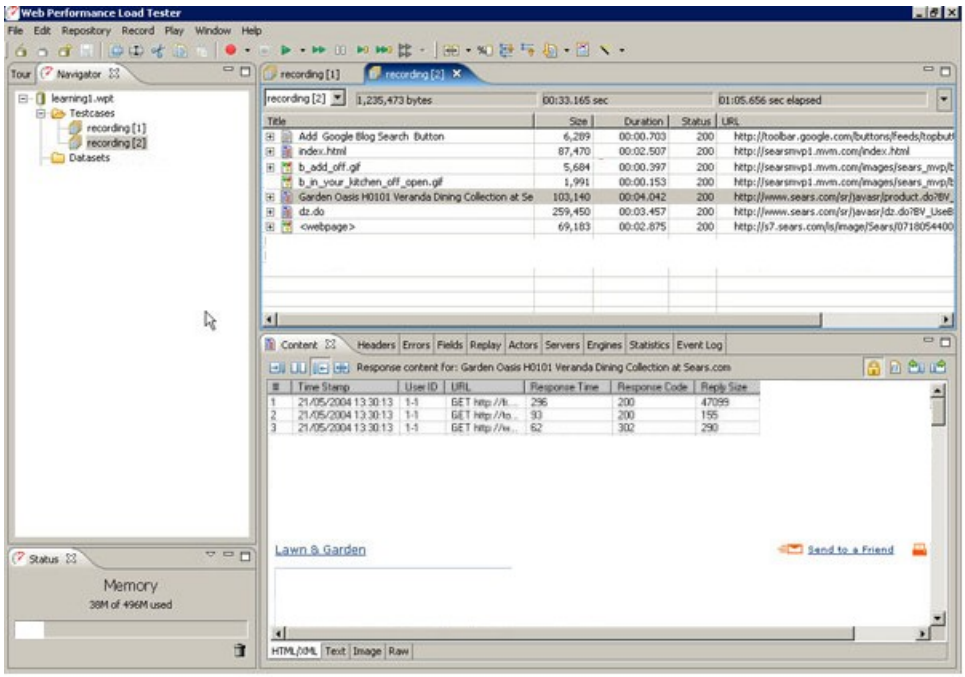



Рис. 2.1 OpenSTA

***Apache Jmeter*** (рисунок 2.2) - це єдиний інструмент для навантажувального тестування, який з однієї сторони безкоштовний і open source, а з іншої, досить розвинений і має можливість створення тестового навантаження одночасно з декількох PC. [12].

*З чого складається тест*

При створенні тесту JMeter пропонує кілька типів компонент:

1) samplers - основні елементи, які безпосередньо спілкуються з тестованим додатком, наприклад http sampler для звернення до веб-додатку;

2) logic controllers - елементи, що дозволяють групувати інші елементи в цикли, групи паралельного запуску;

3) assertions - елементи, що виконують контроль.

За їх допомогою ви можете перевірити текст, який ви очікуєте на веб-сторінці, або, наприклад, вказати, що ви очікуєте відповідь від сервера не більше ніж 2 секунди. Якщо який-небудь Assert не буде задоволений, тест буде мати негативний результат.

*Як виглядає звіт*

Що буде міститися в звіті, користувач налаштовує сам. Зручно те, що до звіту можна включати таблиці та графіки.

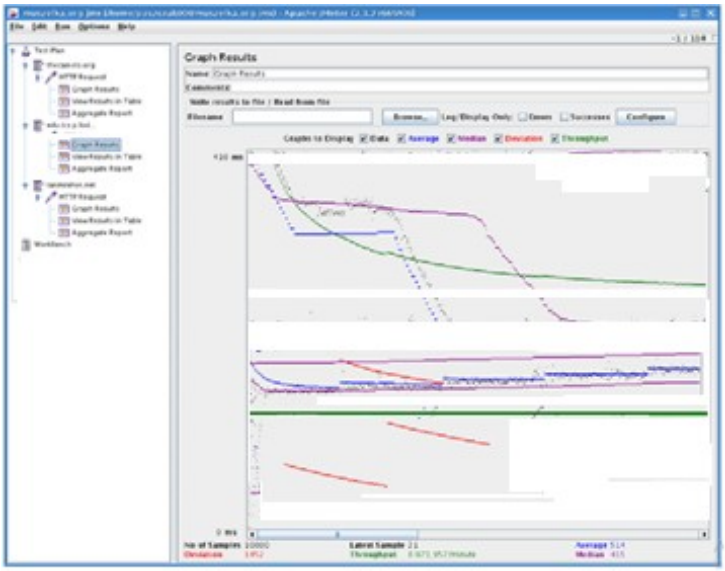



Рис. 2.2 JMeter

***Чи можна розширювати JMeter.***

Jmeter є зручним інструментом для запуску та моніторингу тестів, а також для перегляду звітів. Тому, якщо користувачеві потрібні особливі види тестів, які він може написати на Java, то все, що йому потрібно, це написати код самого тесту. Далі його потрібно оформити як

Sampler, після чого JMeter допоможе йому зробити всю іншу роботу по організації, конфігурації і виконанні тестів (в тому числі і з декількох комп'ютерів).

***WAPT*** (Web Application Testing) (рисунок 2.3) - дозволяє випробовувати стійкість веб та інших додатків, що використовують веб-інтерфейс, до реальних навантажень.

Розробляється новосибірської компанією SoftLogica LLC. Це одна з найпростіших у використанні програм огляду. Для проведення простого тесту навіть не потрібно заглядати в документацію, інтерфейс простий, але не локалізований. Для перевірки WAPT може створювати безліч віртуальних користувачів, кожен з індивідуальними параметрами. Підтримується кілька видів аутентифікації і куки. Сценарій дозволяє змінювати затримки між запитами і динамічно генерувати деякі випробувальні параметри, максимально імітуючи таким чином поведінку реальних користувачів. У запит можуть бути підставлені різні варіанти HTTP-заголовку, у налаштуваннях можна вказати кодування сторінок. Параметри User-Agent, X-Forwarded-For, IP вказуються в настройках сценарію. Значення параметрів запиту можуть бути розраховані кількома способами, у тому числі, визначені відповіддю сервера на попередній запит, використовуючи змінні і функції.

Підтримується робота з захищеним протоколом HTTPS (і всі типи проксі-серверів). Створені сценарії, які зберігаються в файлі XML-формату, можна використовувати повторно. Окрім стандартних Performance і Stress, у списку присутні кілька інших тестів, що дозволяють визначити максимальну кількість користувачів і тестувати сервер під навантаженням протягом довгого періоду.

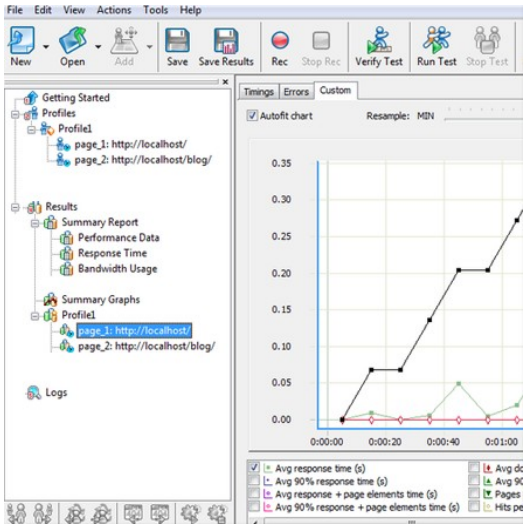



Рис. 2.3 WAPT

***NeoLoad*** (рисунок 2.4) - система, що дозволяє провести тестування навантаження веб-додатків.

Написана на Java. У звіті можна отримати детальну інформацію по кожному завантаженому файлу. NeoLoad вельми зручний для оцінки роботи окремих компонентів (AJAX, PHP, ASP, CGI, Flash, аплетів і ін.). Можлива установка часу затримки між запитами (thinktime) глобально і індивідуально для кожної сторінки. Тестування проводиться як з використанням дуже зручної графічної оболонки, так і за допомогою командного рядка (використовуючи заздалегідь підготовлений XML-файл). Підтримує роботу з протоколом HTTPS, з HTTP і HTTPS проксі, basic веб-аутентифікацію і cookies, автоматично визначаючи дані під час запису сценарію, і потім програє під час тесту. Для роботи з різними профілями для реєстрації користувачів можуть бути використані змінні. При проведенні тесту можна задіяти додаткові монітори (SNMP, WebLogic, WebSphere, RSTAT і Windows, Linux, Solaris), що дозволяють контролювати і параметри системи, на якій працює веб-сервер. [12] За допомогою NeoLoad можна проводити і розподілені тести. Один з комп'ютерів є контролером, на інші встановлюються генератори навантаження (loadGenerator). Контролер розподіляє навантаження між

loadGenerator і збирає статистику. Дуже зручно реалізована робота з віртуальними користувачами. Користувачі мають індивідуальні настройки, потім вони об'єднуються в Populations (повинна бути створена як мінімум одна Populations), в Populations можна задати загальну поведінку (наприклад, 40% користувачів популяції відвідують динамічні ресурси, 20% читають новини). Віртуальні користувачі можуть мати індивідуальну IP-адресу, смугу пропускання і свій сценарій тесту. Використовуючи утиліти навантажувального тестування, можна отримати інформацію щодо роботи веб-сервісу, вжити необхідних заходів щодо усунення виявлених недоліків і гарантувати необхідну продуктивність.

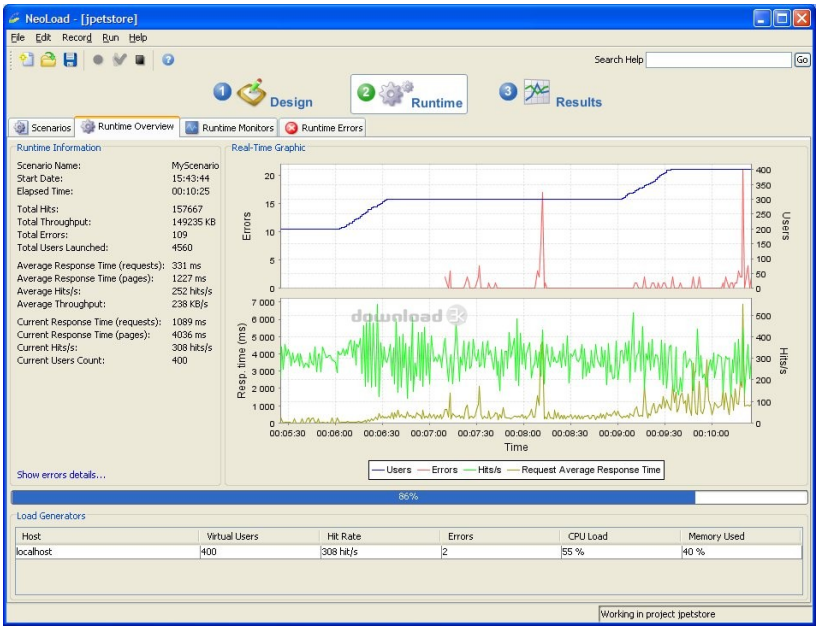



Рис. 2.4 NeoLoad

## 2.6. Автоматизоване функціональне тестування

***Автоматизоване тестування ПЗ*** (Automation Testing) - "це процес, у якому відбувається верифікація ПЗ, в котрому основні функції та кроки (запуск програмного засобу, його ініціалізація, його виконання,

його аналіз і виведення результату) також виконуються автоматично за допомогою інструментів для автоматизованого тестування [6].

З автоматизацією тестування, як і з іншими вузьконаправленими IT-напрямами, пов'язано багато неправдивих уявлень. Для того, щоб уникнути застосування автоматизації, яка є неефективною, слід обходити недоліки та використовувати всі переваги.

Переваги автоматизації тестування:

- повторюваність - всі створені тести мають виконуватися одноманітно, тобто виключений «людський фактор» (тестувальник не може пропустити тест з необережності);
- швидке виконання автоматизованого скриптового коду (не потрібно кожен раз звірятися з інструкціями та документаціями, що економить час на виконання);
- менші витрати на підтримку продукту - коли скрипти вже написані, для їх підтримки та аналізу результатів потрібен менший час, ніж на проведення того ж обсягу тестування але вручну;
- звіти - розсилаються автоматично та зберігаються репорти про результати тестування;
- виконання без втручання - під час виконання скриптових тестів тестувальник може займатися корисними справами.

Недоліки автоматизації тестування:

- повторюваність - написані тести будуть виконуватися одноманітно. Це одночасно є і недоліком, оскільки тестувальник може звернути увагу на деякі деталі і знайти дефект. Автоматичний процес цього зробити не зможе;
- витрати на підтримку – у разі автоматизованих тестів витрати менші, ніж на ручне тестування того ж функціоналу;

- великі витрати на розроблення - розроблення автоматизованих тестів дуже складний процес, оскільки йде розроблення програмного засобу, який тестує інший програмний засіб (у складних автоматизованих тестах є утиліти, бібліотеки та фреймворки, які потрібно тестувати і настроювати, що вимагає часу);
- вартість інструменту для автоматизації - при використанні ліцензійного програмного забезпечення, його вартість завжди досить висока (вільні безкоштовні інструменти відрізняються більш скромним функціоналом і значно меншою зручністю роботи);
- пропуск дрібних помилок - автоматизований скрипт може пропускати багато малих помилок, на перевірку яких він не запрограмований.

Оскільки для тесту потрібно перейти на сторінку «Увійти»,то ми можемо вивести це все в один клас.

В рамках роботи було розроблено методи тестування:

- Automatic test suits (додаток А).
- General test rate (додаток Б).
- Smoke test check list (додаток В).
- Test cases automation (додаток Г).
- Map Module (додаток Д).

# ВИСНОВКИ

У ході виконання роботи було вивчено процес тестування програмного забезпечення в ІТ-компаніях, розглянуто більшість видів дефектів в ПЗ, причини виникнення та способи їх відстеження, за допомогою систем стеження за вадами. Також вивчені способи створення і застосування тест кейсів для чорного ящика. Даний матеріал можна використовувати як для навчальних цілей, так і для успішного проходження технічного співбесіди при влаштуванні на роботу в якості тестувальника.

У другій частині роботи розглянуто програми для навантажувального тестування. За допомогою програми JMeter проведено тестування навантаження веб-сервісу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Библиотека MSDN. Источник информации для разработчиков, использующих средства, продукты, технологии и службы корпорации Майкрософт. [Электронный ресурс]. – Режим доступу: http://msdn.microsoft.com

2. КиберФорум. Форум начинающих и профессиональных программистов, системных администраторов, администраторов баз данных. [Электронный ресурс]. – Режим доступу: http://cyberforum.ru

3. ИНТУИТ. Интернет университет информационных технологий. [Электронный ресурс]. – Режим доступу: http://intuit.ru

4. Клуб программистов. [Электронный ресурс]. – Режим доступу: http://www.cyberguru.ru/

5. ВикипедиЯ. Свободная энциклопедия. [Электронный ресурс]. – Режим доступу: http://ru.wikipedia.org/wiki

6. Тестирование и качество ПО. [Электронный ресурс]. - Режим доступу: http://software-testing.ru/

7. Про тестинг. [Электронный ресурс]. Режим доступу: http://www.protesting.ru/

8. SQA Days. [Электронный ресурс]. Режим доступу: http://sqadays.com/

9. Software testing training and stoftware testing services. [Электронный ресуср]. Режим доступу: http://www.rbcs-us.com/

10. Уиттакер Д., Арбон Д., Каролло Д. Как тестирует Google.: Пер. с англ. - Спб.: Издательский дом "Питер", 2014.-320с

11. Савин Р. Тестирование Дот Ком, или Пособие по жесткому обращению с багами в интернетстартапах. - М.: Дело, 2007. - 312с

12. Хабрахабр. [Электронный ресурс]. Режим доступу: http://habrahabr.ru/

13. Code Project. Сообщество разработки программного обеспечения. [Электронный ресурс]. – Режим доступу: http://codeproject.com

14. Apache Software Foundation. [Электронный ресурс]. - Режим доступу: http://jmeter.apache.org/

15. SeleniumHQ Browser Automation. [Электронный ресурс]. - Режим доступу: http://docs.seleniumhq.org/

16. Полівой В. Як автоматизувати тестування ПЗ? // Cnews, 2007. – 235 с.

17. Дастін Е., Рєшка Д., Пол Д. \ Ї. Молодцова, М. Павлов. Автоматизоване тестування программного забеспечення. Іздательство «ЛОРИ», 2003 – 435 с.

18. Котляров В.П., Коликова Т.В. Основи тестування программного забезпечення., 2006. – 321 с.

19. Автоматизація процессів тестування, І. Винниченко, 2005 .

20. Система мультикритеріального тестування веб-додатків / В.Р. Лабо, К.М. Березька // Матеріали VI Всеукраїнської школи-семінару молодих вчених і студентів «Сучасні комп'ютерні інформаційні технології». – Тернопіль: ТНЕУ. – 2016 – с. 95

21. Хабрахабр. CodedUI или Ranorex? Автоматизация функционального тестирования .NET приложений. [Електронний ресурс]: В.В. Джигда. – 2013. – Режим доступу : https://habrahabr.ru/company/2gis/blog/188302/

22. HP QuickTest Professional. [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу : https://en.wikipedia.org/wiki/HP_QuickTest_Professional

23. Хабрахабр . Descriptive Programming в QuickTest Pro. [Електронний ресурс]: В.В. Ягилюк. – 2009. – Режим доступу : https://habrahabr.ru/post/69138/

24. 5 Best Test Automation Tools [Електронний ресурс] // automated-360 blog . – Режим доступу : http://automated-360.com/automation-tools/5-best-testautomation-tools

25. 10 REGRESSION/FUNCTIONAL WEB TESTING TOOLS [Електронний ресурс] // TECH BLOG. – Режим доступу : http://www.hurricanesoftwares.com/10-regressionfunctional- web-testing-tools/

26. Тестування програмного забезпечення. [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу : https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D 0%BD%D0%BD%D1%8F

27. Тестувальник з нуля: побірка безкоштовних сервісів. – Режим доступу : https://allstars-it.com/ukraine/a-tester-from-scratch-a-selection-of-free-services/ .

28. Techniques and tools of web applications and networks penetration testing - Режим доступу : http://diit.edu.ua/sites/tempus/files/full/1%20(12).pdf .

29. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ WEB-СТОРІНОК ЗА ДОПОМОГОЮ ДРАЙВЕРА БРАУЗЕРА SELENIUM WEBDRIVER - Режим доступу : http://molodyvcheny.in.ua/files/journal/2015/6/02.pdf .

30. . Лайза Криспин, Джанет Грегори. Гибкое тестирование. Вильямс, 2010. – 251 с.

# ДОДАТКИ

## Додаток А. Automatick Test Suits

| Module | ID | Name |
|---|---|---|
| **Login** | LG01 | Remember password |
| | LG02 | Security test |
| **Main** | ST01 | Alerts test |
| | ST02 | ChangeMain |
| | ST03 | Change password |
| **Schedule** | AD01 | Create/edit/deleteWeek A |
| | AD02 | Create/edit/delete Week B |
| | AD03 | Create/edit/deleteschedule |
| | AD04 | Create/edit/delete group |
| | AD05 | Create/edit/deleteday of week |
| **News** | SC01 | Create/edit/ deletepictures |
| | SC02 | Create/edit/deletenews |
| | SC03 | Create/Edit/Delete maintenance |
| | SC04 | Check Filters Show all/Overdue/due/Ok |
| **Map** | MP1 | Checkmainbuilding |
| | MP2 | CheckHousing |
| | MP3 | Check map attributes |

**3.** Smoke test suit

| Module | ID | Name |
|---|---|---|
| **Login** | LG01 | Remember password |
| | LG02 | Security test |
| **Main** | ST01 | Alerts test |
| | ST02 | ChangeMain |
| | ST03 | Change password |
| **Map** | MP1 | Checkmainbuilding |
| | MP2 | CheckHousing |
| | MP3 | Checkmapattributes |

# Додаток Б. General Test Rate

General Test Rate
1- most important
2- can do later

| **Name** | **Rate(1…5)** |
|---|---|
| Login | 5 |
| Main | 5 |
| News | 1 |
| Schedule | 4 |
| LoadTable | 4 |
| Map | 5 |

## Додаток B. Smoke test Checklist_release

### Smoke test Checklist

Prepared by: Alexandr Chistyakov 2/04/2020

| Site | Module | Website: nilsen-app.herokuapp.com | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Food | Key Functionality | | | | | | | |
| | | Icon | Table content | Food schedule | Courses taught | Publication | Export to kml | Knowledge of languages | Note |
| | | pass | pass | pass | pass | pass | fail | pass | |
| | LoadTable | Key Functionality | | | | | | | |
| | | Report | Terms | Table content | Export PDF | Export CSV | Print | Detailed | Filter | Note |
| | | A | Pass | pass | fail | fail | pass | pass | pass | |
| | | B | Pass | Pass | fail | fail | Pass | Pass | Pass | |
| | | Duration | pass | Pass | fail | fail | Pass | Pass | - | |
| | | Connection with food | fail | fail | fail | fail | fail | fail | - | |
| | | Fleet safety | Pass | Pass | fail | fail | Pass | Pass | | |
| | News | Key Functionality | | | | | | | | |
| | | OK | Due | Overdue | Show All | History | | | | | |
| | | | | | | History | | | More Details | | |
| | | | | | | Export PDF | Export CSV | Print | Export PDF | Export CSV | Print |
| | | Pass | fail | fail | fail | fail | Pass | Pass | fail | fail | Pass |
| | | Report | Table content | Export pdf | Export csv | Print | Note | Other | | | |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Publications | pass | pass | pass | pass | slowly | | | | | |
| | | Service type | - | - | - | - | | pass | | | | |
| | | Scheduls | - | - | - | - | | pass | | | | |
| | | Vehicle list | Pass | Pass | - | pass | | | | | | |
| | Map | Key Functionality | | | | | | | | | | |
| | | zoom | Publications Search | Info widget | | Pop up balloon | | Dashboard widgets | | | | |
| | | | | Status time | Driver | Status time | Geofence | ADBI | Eco run status | Eco run savings | Alerts | Maintenance |
| | | Pass | Pass | Pass | Pass | Pass | Pass | Pass | Pass | Pass | Pass | Pass |
| | Schedule | Key Functionality | | | | | | | | | | |
| | | Week A (zoom, edit) | | Division Select | Page Layout | Other | Landing page | | | | | |
| | | - | | pass | pass | pass | pass | | | | | |
| | Main | Key Functionality | | | | | | | | | | |
| | | Measurement | | Time zone | Page layout | Other | Feedback | | | | | |
| | | pass | | pass | pass | pass | Fail | | | | | |

# Додаток Г. Test cases automation

## 1. General test rate

| Name | Rate (from 1 to 5) |
|---|---|
| Login | 4 |
| Settings | 5 |
| Admin | 3 |
| News | 1 |
| Reports | 2 |
| Food | 3 |
| Map | 1 |

## 2. Login

### 2.1. Remember password case

| Properties | |
|---|---|
| Test case ID | LG01 |
| Test case Version | 1.0 |
| Date | 07/03/2020 |
| Author | AlexandrChistyakov |
| **Details** | |
| Test Steps | 1. Geturl 'nilsen-app.herokuapp.com'<br>2. Enter email<br>3. Click button "Submit"<br>4. Check text 'A message has been sent to your email address, follow link to reset password' |
| Note | Check email for letter |

### 2.3. Security test case

| Properties | |
|---|---|
| Test case ID | LG02 |
| Test case Version | 1.0 |

| | |
|---|---|
| Date | 07/03/2020 |
| Author | AlexandrChistyakov |
| Details | |
| Test Steps | 1. Get url ‘nilsen-app.herokuapp.com’<br>2. Enter ‘test1’ in login field, enter ‘testtest1’ to password field<br>3. Click button “Submit”<br>4. Enter ‘test2’ in login field, enter ‘TestTest1’ to password field<br>5. . Click button “Submit”<br>6. Enter ‘test1’ in login field, enter ‘testTest1’ to password field<br>7. Click button “Submit”<br>8. Check text ‘3 consecutive failed login attempts. Delay 5 minutes.’<br>9. Enter ‘test1’ in login field, enter ‘TestTest1’ to password field<br>10. Click ‘Submit’<br>11. Check link ‘Forget password?’ |
| Note | You can enter only 3 times incorrect login or password. Then access blocks for 5 minutes |

## 3. Settings

3.1 Alerts test case

| | |
|---|---|
| Properties | |
| Test case ID | ST01 |
| Test case Version | 1.0 |
| Date | 07/03/2020 |
| Author | AlexandrChistyakov |
| Details | |
| Test Steps | 1. Geturl ‘nilsen-app.herokuapp.com’<br>2. Enter‘TestTest1’ topasswordfieled.<br>3. Click button “Submit”<br>4. Click ‘Settings page’<br>5. Chose alert type<br>6. Enter 5 totimeexeeds<br>7. Enter automation test +date<br>8. Chose all in notify by fields<br>9. Click Confirm |

| | |
|---|---|
| | 10. Delete last 2 alerts<br>11. edit last alert<br>12. Click ‘Cancel’<br>13. edit last alert<br>14. select next alert in select box ‘Alert type’<br>15. Add 1 to field time exceeds<br>16. Add word ‘edit’ to description field<br>17. Choose next notify<br>18. 3 times do 15-18<br>19. Repeat steps from 1 to 19 for all alert types |
| Note | Check email for letter |

4.2.Change password

| Properties | |
|---|---|
| Test case ID | ST03 |
| Test case Version | 1.0 |
| Date | 08/03/2020 |
| Author | AlexandrChistyakov |
| **Details** | |
| Test Steps | 1. Geturl ‘nilsen-app.herokuapp.com’<br>2. Enter ‘test1’ to login field<br>3. Enter‘TestTest1’ topasswordfieled.<br>4. Click button “Submit”<br>5. Click ‘Settings page’<br>6. Enter old password<br>7. Enter new password<br>8. Enter new password to confirm it<br>9. Click ‘Submit’<br>10. Logout<br>11. Login with new password<br>12. Repeate 6-9 stepstochangepasswordfodefault(‘TestTest1’) |

## *4. Schedule*

### *4.1 Create/edit/delete Schedule*

| Properties | |
|---|---|
| Test case ID | AD01 |
| Test case Version | 1.0 |
| Date | 02/03/2020 |
| Author | AlexandrChistyakov |
| **Details** | |
| Test Steps | Geturl ‘‘nilsen-app.herokuapp.com’<br>1. Enter ‘test1’ tologin field<br>2. Enter‘TestTest1’ topasswordfieled.<br>3. Click button “Submit”<br>4. Click ‘Admin” page’<br>Create Schedule<br>5. ClickbuttontodrawSchedule<br>6. DrawSchedule<br>Week a,week b, Lecture week,Meeting of the department.<br>7. Clickbuttotstopdrawing<br>8. Dismiss alert<br>9. Repeat step 6-8<br>10. Accept alert<br>11. Click button cancel<br>12. Repeat step 6-8<br>13. Accept alert<br>14. Entername+date<br>15. SelectinturnallScheduletypes<br>16. Click “Submit”<br>Edite Schedule<br>17. Clickweek a<br>18. Get new coordinates<br>19. Drag and drop<br>20. Close pop up window<br>21. Clickbuttonrefresh(cancel)<br>22. Repeat steps 18-20 |

| | |
|---|---|
| | 23. Click “save” inSchedulelist<br>Edite Schedule in Schedule list<br>24. Clickeditbutton<br>25. Click cancel<br>26. Click edit image<br>27. EnternameofSchedule (“edit”+Schedulename +date)<br>28. SelectinturnallScheduletypes<br>29. Click ‘Submit’<br>30. Repeatsteps 27-30 forallScheduletyps<br>Delete Schedule<br>31. Clickbuttondelete<br>32. Dismiss<br>33. Clickbuttondelete<br>34. Accept |
| Note | |

### *4.2. Create/edit/delete Schedule type*

| Properties | |
|---|---|
| Test case ID | AD03 |
| Test case Version | 1.0 |
| Date | 02/03/2020 |
| Author | AlexandrChistyakov |
| **Details** | |
| Test Steps | Geturl ‘‘nilsen-app.herokuapp.com’<br>1. Enter ‘test1’ tologin field<br>2. Enter‘TestTest1’ topasswordfieled.<br>3. Click button “Submit”<br>4. Click ‘Admin” page’<br>Create Schedule type<br>5. Clickbutton “CreateScheduletype”<br>6. Click button “cancel”<br>7. Clickbutton “CreateScheduletype”<br>8. EnterSchedulename +date<br>9. Enter color number |

| | |
|---|---|
| | 10. Click “commit”<br>Edit Schedule type<br>11. Clickedit button<br>12. Click cancel button<br>13. Click edit button<br>14. Enter “Edit”+Schedule_name+date<br>15. Enter color number<br>16. Click button “Commit”<br>Delete Schedule type<br>17. Click on delete button<br>18. Dismiss alert<br>19. Click on delete button<br>20. Accept alert |
| Note | |

### *4.3. Create/edit/delete group*

| Properties | |
|---|---|
| Test case ID | AD04 |
| Test case Version | 1.0 |
| Date | 02/03/2020 |
| Author | AlexandrChistyakov |
| **Details** | |
| Test Steps | Geturl ‘‘nilsen-app.herokuapp.com’<br>1. Enter ‘test1’ tologin field<br>2. Enter‘TestTest1’ topasswordfieled.<br>3. Click button “Submit”<br>4. Click ‘Admin” page’<br>Create group<br>5. Click button “Create Group”<br>6. Click button “cancel”<br>7. Click button “Create Group”<br>8. Enter Group name +date<br>9. Add food tothegroup<br>10. Remove food fromgroup |

| | |
|---|---|
| | 11. Click “commit”<br>Edit Group<br>12. Click edit image<br>13. Click cancel button<br>14. Click edit button<br>15. Enter “Edit”+group_name+date<br>16. Add food tothegroup<br>17. Remove food fromthegroup<br>18. Click button “Commit”<br>Delete group<br>19. Click on delete button<br>20. Dismiss alert<br>21. Click on delete button<br>22. Accept alert |
| Note | |

## *5. News module*

### *5.1 Create/edit/ delete*

| Properties | |
|---|---|
| Test case ID | FL01 |
| Test case Version | 1.0 |
| Date | 09/03/2020 |
| Author | AlexandrChistyakov |
| **Details** | |
| Test Steps | Geturl ‘‘nilsen-app.herokuapp.com’<br>1. Enter ‘test1’ tologin field<br>2. Enter‘TestTest1’ topasswordfieled.<br>3. Click button “Submit”<br>4. Click ‘News” page’<br>Create New stype<br>5. Clickbutton “CreateNewstype”<br>6. Click button “cancel”<br>7. Clickbutton “CreateNewstype”<br>8. EnterNewstypename+ datetonamefield<br>9. Click on checkbox Distance |

| | |
|---|---|
| | 10. Enter value to distance field<br>11. Click checkbox Time<br>12. Enter value to the time field<br>13. Click checkbox Engine Hours<br>14. Enter value to the engine hours field<br>15. Click button create<br>16. Click button “cancel”<br>17. Click button create<br>18. Click button “confirm”<br>Edit News type<br>19. Click on “Edit” icon<br>20. Click cancel button<br>21. Click on “Edit ” icon<br>22. Entername+edit+date<br>23. Click on checkbox time<br>24. ClickoncheckboxHours<br>25. Clickonbuttoncreate<br>26. Click on button cancel<br>27. Click on button create<br>28. Click on button Confirm<br>29. Steps 20-30 for 7 combinations<br>Delete food<br>30. Click on “Delete” icon<br>31. Dismiss alert<br>32. Click on delete button<br>33. Accept alert |
| Note | |

### *5.2. CheckFiltersShowall/Overdue/due/Ok*

| Properties | |
|---|---|
| Test case ID | FL04 |
| Test case Version | 1.0 |
| Date | 09/03/2020 |

| Author | AlexandrChistyakov |
|---|---|
| Details | |
| Test Steps | Geturl ‘‘nilsen-app.herokuapp.com’ |
| 1. | Enter ‘test1’ tologin field |
| 2. | Enter ‘TestTest1’ to password field. |
| 3. | Click button “Submit” |
| 4. | Click ‘News” page’ |
| | Show all |
| 5. | Click on “Show all” |
| 6. | Make screenshot |
| 7. | Click on “Print” |
| 8. | Go to print page |
| 9. | Press key “Escape” |
| 10. | Make screenshot |
| | OK |
| 11. | Click on “Overdue” |
| 12. | Make screenshot |
| 13. | Click on “Print” |
| 14. | Go to print page |
| 15. | Press key “Escape” |
| 16. | Make screenshot |
| | DUE |
| 17. | Click on “Overdue” |
| 18. | Make screenshot |
| 19. | Click on “Print” |
| 20. | Go to print page |
| 21. | Press key “Escape” |
| 22. | Make screenshot |
| | Overdue |
| 23. | Click on “Overdue” |
| 24. | Make screenshot |
| 25. | Click on “Print” |
| 26. | Go to print page |
| 27. | Press key “Escape” |
| 28. | Make screenshot |
| Note | There shouldn’t be create schedule in the table. |

### 6.1 Check map attributes

| Properties | |
|---|---|
| Test case ID | MP3 |
| Test case Version | 1.0 |
| Date | 11/03/2020 |
| Author | AlexandrChistyakov |
| **Details** | |
| Test Steps | 1. Geturl ‘‘nilsen-app.herokuapp.com’<br>2. Enter ‘TestTest1’ to password field.<br>3. Click button “Submit” |
| Note | |

# Додаток Д. Testpass_map

**Map module**

Map buttons

| Elements | Action | Expected result |
|---|---|---|
| Show All | Click On | You should see all  buildings  on map |
| View. Street View | activate Checkbox | Street View window appers and its functionality is working |
| View.Buildingslist | activate Checkbox | Buildings list appears |
| View.Labels | activate Checkbox | Labels is displaying under Buildings image |
| Legend | Click On | See legend with different colours for different locations |
| 1 hostel/Main buildings | activate Checkbox | You should see 1 hostel or Main buildings |
| Traffic button | Click On | You should see traffic on map |
| Help Button | Click On | See help window with FAQ |
| Full screen Button | Click On | Map fullscreen |
| Map/satelite | activate Checkbox | Different map views |
| Zoom bar | Click On/wheel up down | Zoom should changed |

Note

Info icon should have black baground and black border for Buildings in Buildings list when you click on it. But there shouldn't be green background
Zoom icon should have black baground and green background for food in food list. There shouldn't be black border

| Buildings Info windows | | |
|---|---|---|
| Elements | Action | Expected result |
| Speed | Check matching with Buildings status table | Displaying current speed. And speed limit in icon |
| Odomentr | Check matching with Buildings status table | Displaying current odometr |
| Batteries | Check matching with Buildings status table | Displaying all batteris with correct values |
| Fuel level | Check matching with Buildings status table | Displaying fuel level or n/a |
| Solar/Wifi icong | Check matching with Buildings status table and in battery table | Displaying if Buildings have Solar or Wifi on board |
| Status time/status | Check matching with Buildings status table | Displying correct time and status |
| | | |
| Location | Check matching with Buildings status table | Current lovation |
| Main buildings | Check matching with Buildings status table | Current Main buildings if Buildings is in it |
| Station | Check matching with Buildings table | Buildings station |
| Buildings information | Check matching | Should be displaying information about Buildings |

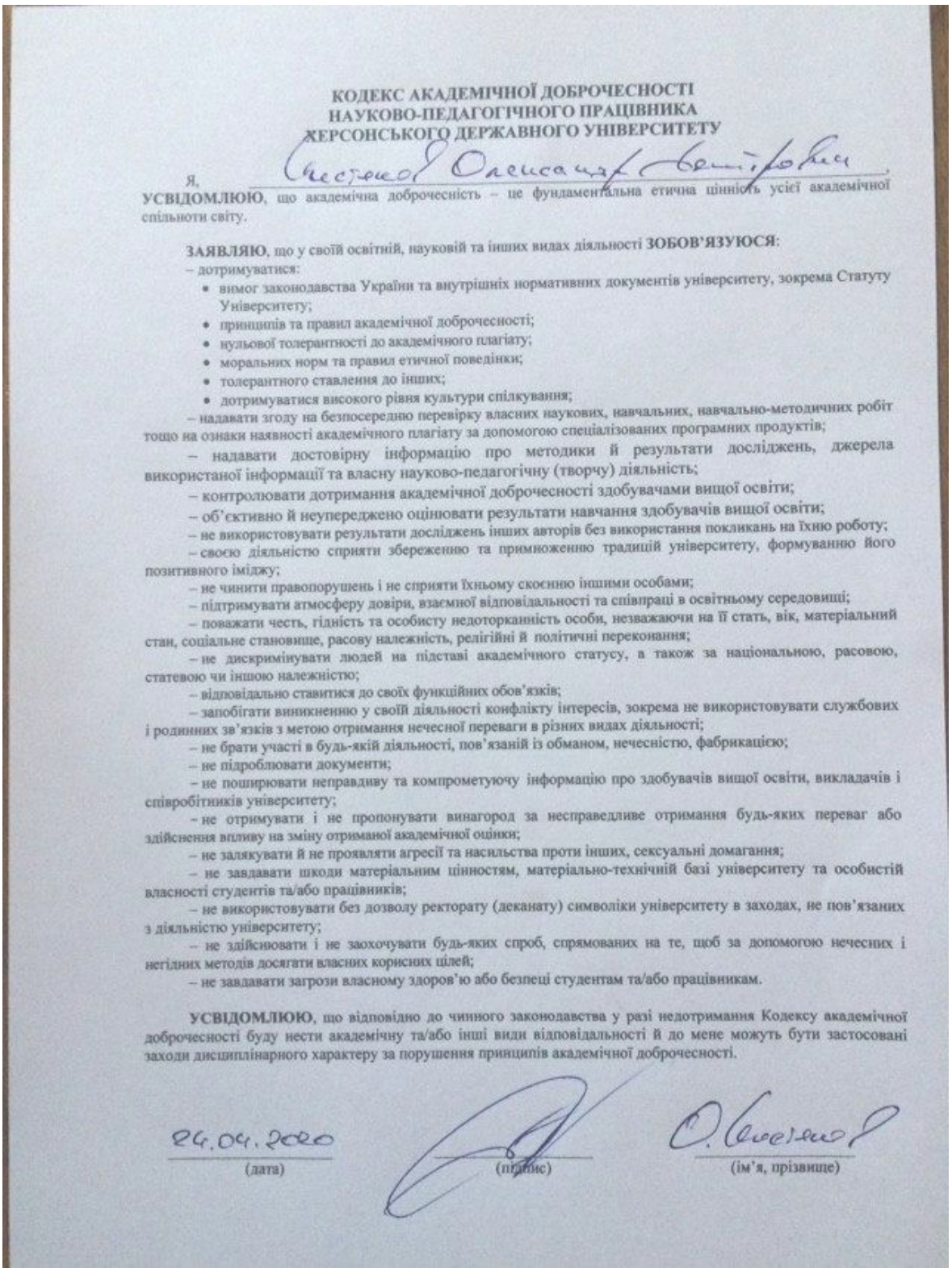
| Buildings icon on map | | |
|---|---|---|
| Elements | Action | Expected result |

Dashboard panel

| Elements | Action | Expected result |
|---|---|---|
| Adbi | Mouse over | Value in box match with value in icon |
| Adbi | Click On | Get News safety report. Adbi match with Adbi value on icon and in KPI wi |
| flletEco widget | Mouse over | Number of idle and ecorun active  food match with its number on map and |
| EcoRun widget.. FuelSpent/Eco Run savings | Mouse over | Value in window match with value in icon |
| EcoRun widget.Fuel cost | Mouse over | Display cost savings if user set cost in Setting, alse displaying Massege |
| EcoRun widget | Click On | Get News Eco Run report. Values match with  values on icon and cost matc |
| Alert widget | Mouse over | Number of  food with alerts match with its number on map and Buildings l |
| Maintenance widget | Mouse over | Number of due/overdue maintenances match with Its on Maintenance widg |
| Maintenance widget | Click On | News Page |

Note: please rewiev design  for map elemnts.

# Додаток Е. Кодекс академічної доброчесності



**КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ**
**НАУКОВО-ПЕДАГОГІЧНОГО ПРАЦІВНИКА**
**ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ**

Я, Шестопал Олександр Євгенович,

**УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

**ЗАЯВЛЯЮ**, що у своїй освітній, науковій та інших видах 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 власних наукових, навчальних, навчально-методичних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;

– надавати достовірну інформацію про методики й результати досліджень, джерела використаної інформації та власну науково-педагогічну (творчу) діяльність;

– контролювати дотримання академічної доброчесності здобувачами вищої освіти;

– об'єктивно й неупереджено оцінювати результати навчання здобувачів вищої освіти;

– 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 й політичні переконання;

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

– відповідально ставитися до своїх функційних обов'язків;

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

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

– не підроблювати документи;

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

– 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 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 студентам та/або працівникам.

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

24.04.2020 (дата) (підпис) О. Шестопал (ім'я, прізвище)