Вкладка 1

**Level 1.**

1. Склади порівняльну таблицю функціонального, нефункціонального і пов’язаного зі змінами видів тестування.   
Порівняння має містити такі блоки:

* що перевіряється;
* коли застосовується;
* обмеження;
* особливості

| **Критерій** | **Функціональне тестування** | **Нефункціональне тестування** | **Тестування, пов’язане зі змінами** |
| --- | --- | --- | --- |
| **Що перевіряється** | Перевіряється, чи працює функціонал відповідно до вимог (use cases, сценарії користувача). | Перевіряються характеристики системи, які не стосуються функціоналу напряму: продуктивність, безпека, зручність. | Перевіряється, чи не зламалися існуючі функції після змін (оновлення, виправлення багів тощо). |
| **Коли застосовується** | На всіх етапах життєвого циклу розробки: від початку до релізу. Найчастіше — під час розробки і перед релізом. | Зазвичай на етапі завершення розробки або в рамках окремих фаз тестування. | Коли розробники змінюють код або налаштування програми: додають оновлення, виправляють помилки чи змінюють структуру коду |
| **Обмеження** | Обмежується лише функціональністю, не враховуючи інші аспекти як-от швидкість або масштабованість | Результати можуть бути залежними від середовища, важко виміряти об'єктивно; потребують спеціальних інструментів | Може не виявити нові баги, а тільки гарантувати, що старий функціонал не порушений |
| **Особливості** | Сюди входять перевірки, як працює функція для користувача, тестування, окремих частин коду та їх взаємодії. Такі тести часто автоматизують. | Сюди входять перевірки, як працює програма при великій кількості користувачів, тестування зручності інтерфейсу та безпеки. Потрібно імітувати велике навантаження, а іноді тестувати вручну. | Сюди входять перевірки, як працює програма при великій кількості користувачів, тестування зручності інтерфейсу та безпеки. Потрібно імітувати велике навантаження, а іноді тестувати вручну. |

2. Поясни, в чому різниця між регресією та ретестингом (5 речень).

Ретестинг — це процес повторного тестування певної функції після виправлення дефекту, щоб переконатися, що він справді усунутий. Регресійне тестування, натомість, перевіряє, чи не зламався існуючий функціонал після внесення змін у код. Ретестинг завжди фокусується на конкретному багу, тоді як регресія охоплює ширший набір тестів. Ретестинг не обов’язково автоматизується, а регресію часто автоматизують для швидкого запуску після кожного оновлення. Отже, ретестинг відповідає на питання: "Чи виправлено цю помилку?", а регресія — "Чи не зламалося нічого іншого?".

**Level 2.**

Як ти вважаєш, чи можливе для продукту проведення тільки функціонального тестування, без перевірки нефункціональних вимог?

* Якщо так – в яких випадках?
* Якщо ні – чому?
* Обґрунтуй свою відповідь.

**Так, можливо провести лише функціональне тестування без нефункціонального, але це допустимо лише в обмежених випадках.** Зазвичай це обґрунтовано специфікою продукту, проектними обмеженнями чи пріоритетами бізнесу. Проте, з точки зору якості — це завжди ризик.

***Це можливо у таких випадках:***

1. **Продукт на ранній стадії**

Якщо продукт ще тільки створюється для перевірки ідеї або концепції, функціональне тестування може бути достатнім, щоб переконатися, що основна логіка працює. Нефункціональні аспекти (продуктивність, безпека) тестують пізніше.

1. **Суворі обмеження часу або ресурсів**

Іноді бізнес приймає рішення випустити продукт із мінімальним тестуванням, концентруючись лише на критичній функціональності, відкладаючи нефункціональні перевірки на наступні ітерації.

1. **Прості продукти з низькими нефункціональними ризиками**

Наприклад, дуже простий калькулятор або скрипт, де швидкодія, масштабованість чи навантаження не є критично важливими.

***Цього недостатньо у таких випадках:***

1. **Комерційний продукт для кінцевих користувачів**

Продуктивність, безпека, юзабіліті є критично важливими для користувацького досвіду. Якщо їх не перевірити — продукт може працювати функціонально, але бути непридатним до використання (повільний, небезпечний, незручний). Наприклад, навіть якщо кнопка "Оплатити" працює, але сторінка вантажиться 10 секунд — користувач, скоріш за все, піде.

1. **Продукти з високими вимогами (напр., банківські системи)**

Тут нефункціональні вимоги (відмовостійкість, масштабованість, безпека) є критичними для бізнесу, і їх ігнорування — великий ризик.

1. **Регуляторні вимоги**

Продукти, які підпадають під сертифікацію або регуляторні акти вимагають обов’язкового тестування нефункціональних характеристик (захист даних, доступність).

### **Висновок:**

* **Можливо провести лише функціональне тестування, але лише там, де нефункціональні вимоги або відомо не критичні, або будуть протестовані пізніше.**
* **У більшості серйозних проектів повна відмова від нефункціонального тестування є серйозним ризиком, який може призвести до провалу продукту на ринку.**

Як ти розумієш необхідність проведення smoke (димового) тестування? Чи завжди воно є доречним?

## **Що таке Smoke (Димове) тестування?**

**Smoke Testing** — це поверхнева перевірка найкритичніших функцій системи, щоб упевнитися, що "все взагалі запускається і не падає". Це як перевірити, чи "йде дим з пристрою після ввімкнення", тобто чи продукт взагалі придатний для подальшого детального тестування.

**Необхідність Smoke тестування:**

Smoke тестування потрібне для того, щоб швидко перевірити, чи "жива" система після отримання нової версії програми. Воно дозволяє відразу виявити критичні помилки, які можуть зробити подальше тестування безглуздим. Сенс у тому, щоб не витрачати час і ресурси на детальні тести, якщо базові функції програми (наприклад, старт системи, логін, відкриття головної сторінки) не працюють взагалі.

Smoke тестування дає миттєвий фідбек команді розробки: можна швидко зрозуміти, чи придатна збірка до подальшого тестування, чи ні. Це особливо важливо у великих проектах з частими оновленнями, де тестування кожної збірки програми є критично важливим для якості продукту.

Коли тестування автоматизоване, smoke тести додають у спеціальні процеси, які самі запускаються після кожної нової збірки програми. Це допомагає одразу виявляти погані версії програми і не витрачати час на їхнє детальне тестування.

Таким чином, основна цінність smoke тестування — це економія часу, швидкий вияв "фатальних" багів і забезпечення стабільності процесу розробки.

**Чи завжди Smoke тестування доречне?**

Smoke тестування доречне майже завжди, особливо коли ми маємо справу з проєктами, де програму регулярно оновлюють і збірки виходять часто. Це стосується середніх і великих проектів, де важливо перевіряти, чи система взагалі працює після кожної зміни. Також smoke тести дуже важливі перед початком повного регресійного тестування, щоб не витрачати даремно час, якщо програма навіть не запускається.

Особливо актуальним smoke тестування є у процесах автоматизації, коли кожна нова версія програми автоматично перевіряється відразу після зборки.

Втім, є випадки, коли smoke тестування можуть не проводити. Наприклад, у дуже маленьких проектах, де зміни у коді мінімальні, і продукт тестується лише раз перед запуском. Або якщо немає достатньо часу чи ресурсів для цього, хоча це завжди ризиковане рішення. Але навіть у таких ситуаціях краще мати хоча б базові автоматизовані smoke тести.

**Level 3.**

Ти – засновник/ця стартапу і плануєш випустити на ринок мобільний застосунок для обміну світлинами котиків для iOS та Android пристроїв.

Користувачі можуть завантажувати фотографії котиків. Але не можуть завантажувати фотографії інших тварин/людей/об’єктів. Користувачі можуть додавати друзів, ставити “вподобайки”, залишати коментарі.  
  
Завдння: Напиши 5 функціональних тест-кейсів, які перевіряли б роботу застосунку.

### **Тест-кейс 1: Завантаження фото кота (позитивний сценарій)**

**Мета:** Перевірити, що користувач може успішно завантажити фотографію кота.  
**Передумови:** Користувач авторизований у застосунку.  
**Кроки:**

* 1. Відкрити сторінку завантаження фото.
  2. Вибрати фото кота з галереї.
  3. Натиснути кнопку "Завантажити".

**Очікуваний результат:** Фото успішно завантажено та з’являється у стрічці користувача.

### **Тест-кейс 2: Блокування завантаження фото, де немає кота (негативний сценарій)**

**Мета:** Перевірити, що користувач не може завантажити фото, де немає кота (людина, собака, предмет тощо).

**Передумови:** Користувач авторизований у застосунку.  
**Кроки:**

* 1. Відкрити сторінку завантаження фото.
  2. Вибрати фото собаки (або людини).
  3. Натиснути кнопку "Завантажити".

**Очікуваний результат:** Відображається повідомлення про помилку: "Будь ласка, завантажте фото кота".

### **Тест-кейс 3: Додавання друга у список друзів**

**Мета:** Перевірити, що користувач може додати іншого користувача у список друзів.

**Передумови:** Користувач авторизований.  
**Кроки:**

* 1. Перейти до профілю іншого користувача.
  2. Натиснути кнопку "Додати в друзі".

**Очікуваний результат:** Відображається повідомлення "Запит на додавання в друзі відправлено".

### **Тест-кейс 4: Ставлення вподобайки під фото**

**Мета:** Перевірити, що користувач може поставити "вподобайку" під фотографією.

**Передумови:** Користувач авторизований, у стрічці є фотографії котів.

**Кроки:**

* 1. Відкрити стрічку з фото.
  2. Натиснути на іконку "серце" під будь-яким фото.

**Очікуваний результат:** Іконка "серце" змінює колір, лічильник вподобайок збільшується на 1.

### **Тест-кейс 5: Додавання коментаря під фотографією кота**

**Мета:** Перевірити, що користувач може залишити коментар під фото.  
**Передумови:** Користувач авторизований, у стрічці є фотографії котів.  
**Кроки:**

* 1. Відкрити стрічку з фото.
  2. Вибрати будь-яке фото кота.
  3. Ввести текст коментаря у відповідне поле.
  4. Натиснути кнопку "Опублікувати".

**Очікуваний результат:** Коментар з’являється під фото у стрічці.

Напиши, які нефункціональні вимоги ти хотів/ла б застосувати для продукту твого стартапу.

Ось основні **нефункціональні вимоги**, які я б застосувала для мобільного застосунку обміну світлинами котиків:

**1. Продуктивність**

* Застосунок має завантажувати фото та оновлювати стрічку не довше ніж за 3 секунди при нормальному інтернет-з’єднанні.
* Відповідь на додавання вподобайки, коментаря або друга має бути миттєвою (до 1 секунди).

**2. Надійність (стабільність)**

* Застосунок не повинен аварійно закриватися при завантаженні фото, коментарів або при роботі з друзями.
* Відновлення після падіння має відбуватися без втрати введених даних.

**3. Безпека**

* Забезпечити аутентифікацію користувачів через надійний механізм (паролі, двофакторна аутентифікація).
* Захист від завантаження забороненого контенту (не котів).
* Шифрування персональних даних та фото під час передачі та зберігання.

**4. Юзабіліті (зручність користування)**

* Інтерфейс має бути простим і зрозумілим, адаптованим під різні розміри екранів iOS та Android.
* Користувач має швидко знаходити основні функції (завантаження фото, додавання друзів, вподобайки, коментарі).
* Повідомлення про помилки мають бути зрозумілими та допомагати користувачу виправити проблему.

**5. Масштабованість**

* Система має підтримувати одночасну роботу тисяч користувачів без погіршення швидкодії.
* Легко додавати нові функції без перебоїв у роботі.

Опиши перевірки, які б їх перевіряли (3-5 прикладів).

### **1. Перевірка продуктивності завантаження фото**

**Що перевіряємо:** Час завантаження фотографії кота.

**Як перевіряємо:** Вимірюємо час від моменту натискання кнопки “Завантажити” до появи фото у стрічці.

**Очікуваний результат:** Час завантаження не перевищує 3 секунд при стабільному інтернет-з’єднанні.

**2. Перевірка стабільності при завантаженні фото**

**Що перевіряємо:** Чи не відбувається аварійного завершення роботи застосунку.

**Як перевіряємо:** Повторюємо завантаження фото багаторазово (наприклад, 100 разів підряд) і слідкуємо за станом застосунку.

**Очікуваний результат:** Застосунок не завершується аварійно, усі фото завантажуються коректно.

### **3. Перевірка безпеки – блокування завантаження не котячих фото**

**Що перевіряємо:** Чи відхиляються фото, які не містять котів.

**Як перевіряємо:** Під час тесту завантажуємо різні фото (собак, людей, об’єктів) і перевіряємо реакцію системи.

**Очікуваний результат:** Відображається повідомлення про помилку і фото не зберігається.

### **4. Перевірка зручності інтерфейсу**

**Що перевіряємо:** Чи зрозумілі повідомлення користувачу.

**Як перевіряємо:** Виконуємо дії, що викликають помилки (наприклад, завантаження фото без кота) і аналізуємо текст повідомлень.

**Очікуваний результат:** Повідомлення прості, зрозумілі, допомагають користувачу виправити помилку.

### **5. Перевірка масштабованості (продуктивності під навантаженням)**

**Що перевіряємо:** Чи підтримує система одночасну роботу великої кількості користувачів.

**Як перевіряємо:** За допомогою інструментів навантажувального тестування імітуємо одночасну роботу тисяч користувачів, що завантажують фото, ставлять вподобайки та додають коментарі.

**Очікуваний результат:** Час відгуку системи залишається у межах прийнятних норм, помилок при обробці запитів немає.