# **Модуль №2**

### **Теоретичні питання:**

1. Як Docker може полегшити процес розгортання програмного забезпечення, і чому він є популярним інструментом для цієї мети?

Docker полегшує процес розгортання програмного забезпечення завдяки контейнеризації, яка дозволяє створювати ізольовані середовища для запуску додатків. Контейнери містять усі необхідні залежності, бібліотеки та середовище виконання, що забезпечує стабільну роботу додатку незалежно від платформи чи інфраструктури. Це спрощує процес перенесення програм між середовищами розробки, тестування та продакшну. Завдяки легкості масштабування та відсутності конфліктів між залежностями, Docker став популярним інструментом для розгортання, особливо в контексті мікросервісної архітектури.

1. Які основні аспекти управління командою важливі в контексті розробки програмного забезпечення?

Управління командою в розробці програмного забезпечення базується на комунікації, плануванні, розподілі завдань та контролі прогресу. Важливими аспектами є формулювання чітких цілей, використання сучасних інструментів для трекінгу завдань (наприклад, Jira або Trello), регулярні зустрічі для синхронізації та забезпечення мотивації команди. Лідер команди має забезпечити підтримку та розвиток учасників, а також своєчасно вирішувати конфлікти й адаптувати процеси відповідно до змінних умов проекту.

1. Як система контролю версій Git допомагає управляти змінами в програмному коді та чому її використання є важливим для успішного проекту?

Git забезпечує управління змінами в програмному коді завдяки системі контрольованих комітів, гілок і об'єднань змін (merge). Це дозволяє командам працювати паралельно над різними функціями, зберігаючи контроль над змінами й можливість повернутися до попередніх версій у разі помилок. Використання Git важливе для успіху проекту, оскільки це сприяє організованості та прозорості, знижуючи ризики втрати даних і сприяючи більш ефективній співпраці розробників.

1. Які ключові стратегії деплойменту програмного забезпечення використовуються, і чому вони важливі для успіху проекту?

Ключові стратегії деплойменту включають ручний, автоматизований та безперервний деплоймент. Автоматизація за допомогою інструментів, таких як Jenkins або GitLab CI/CD, дозволяє швидше і безпомилково доставляти зміни на продакшн. Безперервний деплоймент (Continuous Deployment) мінімізує час між розробкою та доступністю нових функцій для користувачів. Ці стратегії важливі, оскільки вони зменшують ризики, підвищують швидкість розгортання та дозволяють оперативно реагувати на зворотний зв'язок.

1. Як проходить процес перевірки та валідації в розробці програмного забезпечення, і чому це важливо для забезпечення якості продукту?

Процес перевірки та валідації в розробці програмного забезпечення включає тестування функціоналу, інтеграції та продуктивності. Перевірка забезпечує, що програмне забезпечення працює відповідно до специфікацій, тоді як валідація гарантує, що продукт відповідає потребам користувачів. Цей процес важливий для забезпечення якості, зменшення помилок та створення позитивного досвіду для кінцевих користувачів. Автоматизовані та ручні тести разом із відгуками користувачів формують основу надійного програмного забезпечення.

### **Практичне завдання з Перевірки та валідації:**

**Мета завдання: Розробити функцію для обчислення факторіалу та напишіть юніт-тести для неї.**

1. Створіть функцію для обчислення факторіалу числа. Мова програмування може бути будь-якою (Python, JavaScript, Java, C++, тощо).

//js

function factorial(n) {

if (typeof n !== 'number' || !Number.isInteger(n)) {

throw new Error("Факторіал визначений лише для цілих чисел.");

}

if (n < 0) {

throw new Error("Факторіал не визначений для від'ємних чисел.");

}

if (n === 0 || n === 1) {

return 1;

}

return n \* factorial(n - 1);

}

module.exports = factorial;

1. Напишіть юніт-тести, що покривають основні випадки використання функції (наприклад, обчислення факторіалу для невід'ємного цілого числа).

// Тестування факторіалу для 1

test('Обчислення факторіалу 1 має дорівнювати 1', () => {

expect(factorial(1)).toBe(1);

});

// Тестування факторіалу для 5

test('Обчислення факторіалу 5 має дорівнювати 120', () => {

expect(factorial(5)).toBe(120);

});

// Тестування факторіалу для 10

test('Обчислення факторіалу 10 має дорівнювати 3628800', () => {

expect(factorial(10)).toBe(3628800);

});

// Тестування факторіалу для 15

test('Обчислення факторіалу 15 має дорівнювати 1307674368000', () => {

expect(factorial(15)).toBe(1307674368000);

});

// Перевірка на від’ємні числа

test('Має викидати помилку для від’ємних чисел', () => {

expect(() => factorial(-5)).toThrow("Факторіал не визначений для від'ємних чисел.");

});

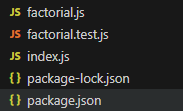
// Перевірка на дробові числа

test('Має викидати помилку для дробових чисел', () => {

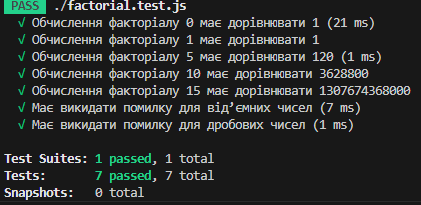
expect(() => factorial(5.5)).toThrow("Факторіал визначений лише для цілих чисел.");

});

1. Збережіть код вашої функції та тести в окремих файлах.



1. Запустіть тести та переконайтеся, що ваша функція працює коректно.



1. Створіть короткий звіт з результатами виконання тестів та вкажіть, чи пройшли всі тести успішно.

Функція factorial працює коректно для позитивних сценаріїв (числа 0, 1, 5, 10, 15) і правильно обробляє некоректні випадки (від’ємні та дробові значення). Тестування підтвердило стабільність і правильність роботи функції.

1. Завантажте ваш код та звіт на github репозиторій.

[Github](https://github.com/IDzuben/Modul-Factorial-Dzubenko-I-11)