Тестування, верифікація та валідація ПЗ

Концидайло Андрій Михайлович

Quality Assurance Engineer
asp_kam1@student.ztu.edu.ua
@AndyFox96

Фронтенд (клієнтська сторона користувацького інтерфейсу)

- це все, що браузер може читати, виводити на екран та запускати. Це HTML, CSS та Javascript.
- HTML (HyperText Markup Language) говорить браузеру, з чого складається сторінка: текст, кнопка, зображення, заголовок, список
- CSS (Cascading Style Sheets) говорить браузеру, як зображати елементи: відступ вправо на 10 пікселів, текст чорного кольору шрифту Comic Sans
- JavaScript говорить браузеру, як реагувати на деякі взаємодії: наводите на кнопку і змінюється колір, натискаєте і з’являється вікно тощо

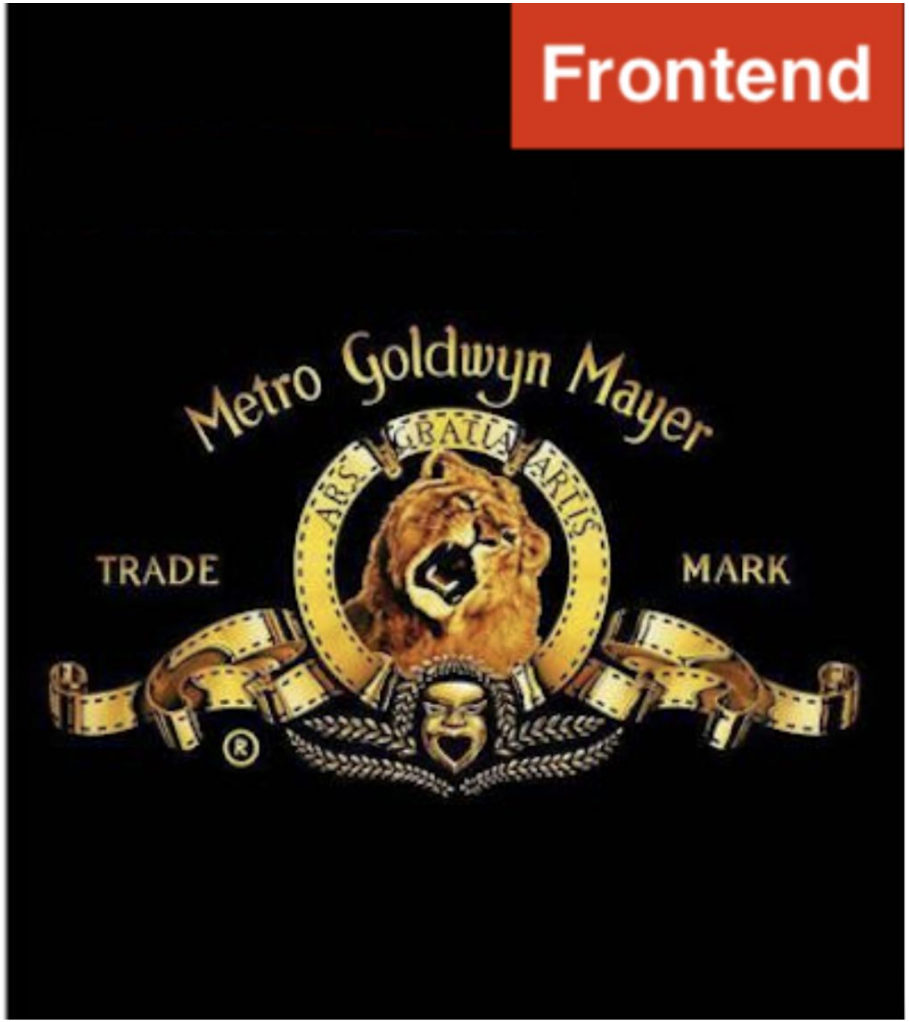
https://hackernoon.com/in-simple-terms-backend-code-frontend-code-and-how-they-interact-2485c5a1bbd2

Бекенд (програмно-апаратна частина проєкту)

- це все, що працює на сервері (на комп’ютері, що під’єднаний до мережі, який відповідає на повідомлення від інших комп’ютерів)
- це обробка бізнес-логіки
- Можна писати на багатьох мовах програмування: Java, C#, Python, Ruby, PHP, JavaScript
- Можна використовувати різні системи управління базами даних: MySQL, MsSQL, PostgreSQL, MongoDB

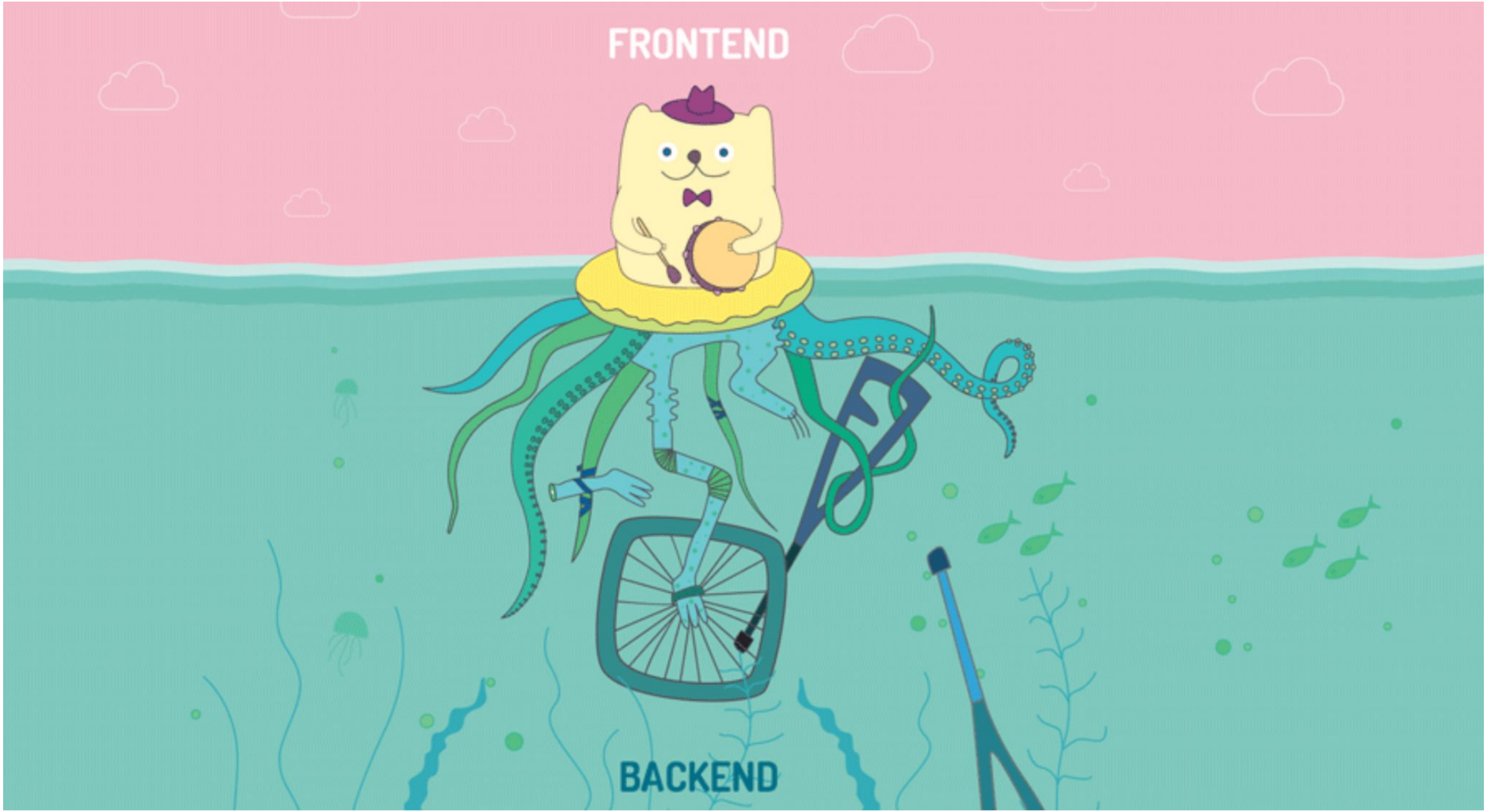
https://hackernoon.com/in-simple-terms-backend-code-frontend-code-and-how-they-interact-2485c5a1bbd2


Backend


Frontend
Metro Goldwyn Mayer
TRADE
MARK

Як взаємодіють фронтенд та бекенд?

- Один із найпопулярніших способів - Клієнт-серверна архітектура
- У цьому випадку HTTP-запити надсилаються безпосередньо на сервер програми, а сервер відповідає HTML-сторінкою.
- Між отриманням запиту та відповіддю сервер зазвичай шукає інформацію в базі даних.
- Коли сторінку завантажено в браузері, HTML визначає, що буде показано, CSS — як це буде виглядати, а JS — будь-які особливі взаємодії.


FRONTEND
BACKEND

https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

Як працює клієнт-серверна архітектура

- **Клієнт** – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.
- **Сервер** – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.
- Правила взаємодії між клієнтом і сервером називаються протоколом обміну (протоколом взаємодії)
- Мережевий протокол - це набір правил, за якими клієнт та сервер взаємодіють.

https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

Як працює клієнт-серверна архітектура

Програмне забезпечення з боку клієнта відправляє запит на сервер, де він обробляється і готовий результат відправляється клієнту.

Сервер виконує наступні функції:

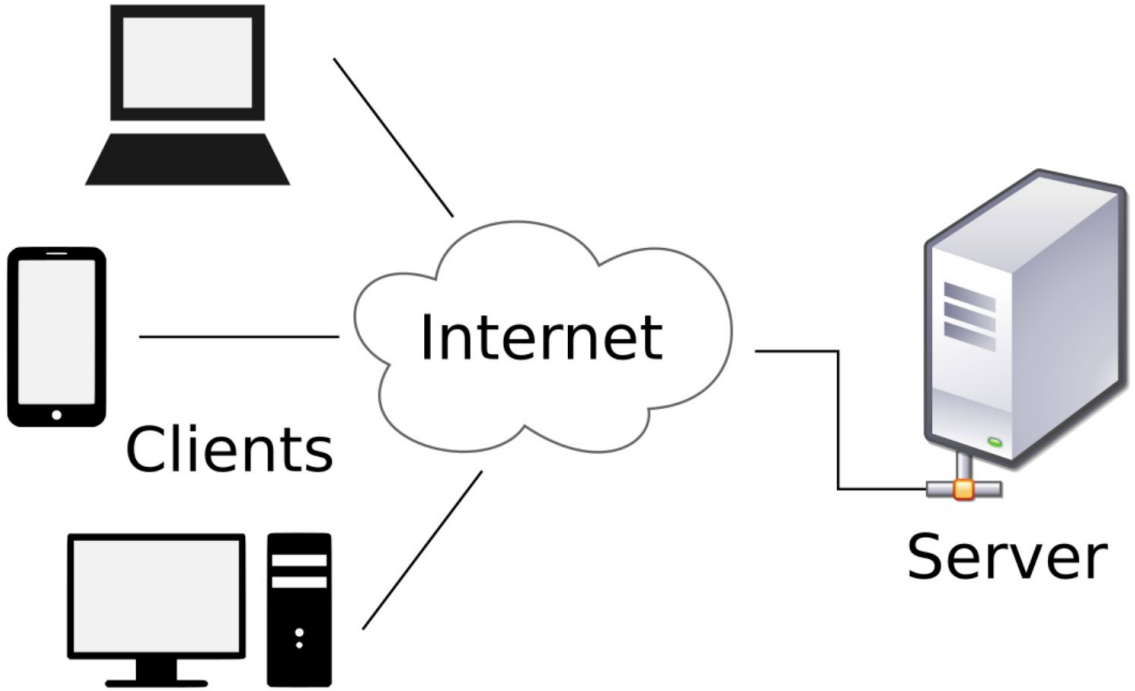
- збереження даних і доступ до них
- обробка запиту клієнта
- відправка відповіді клієнту

Функції, які реалізуються клієнтом:

- формування і відправка запиту до сервера
- отримання результатів і відправка додаткових запитів
- надання юзер інтерфейсу

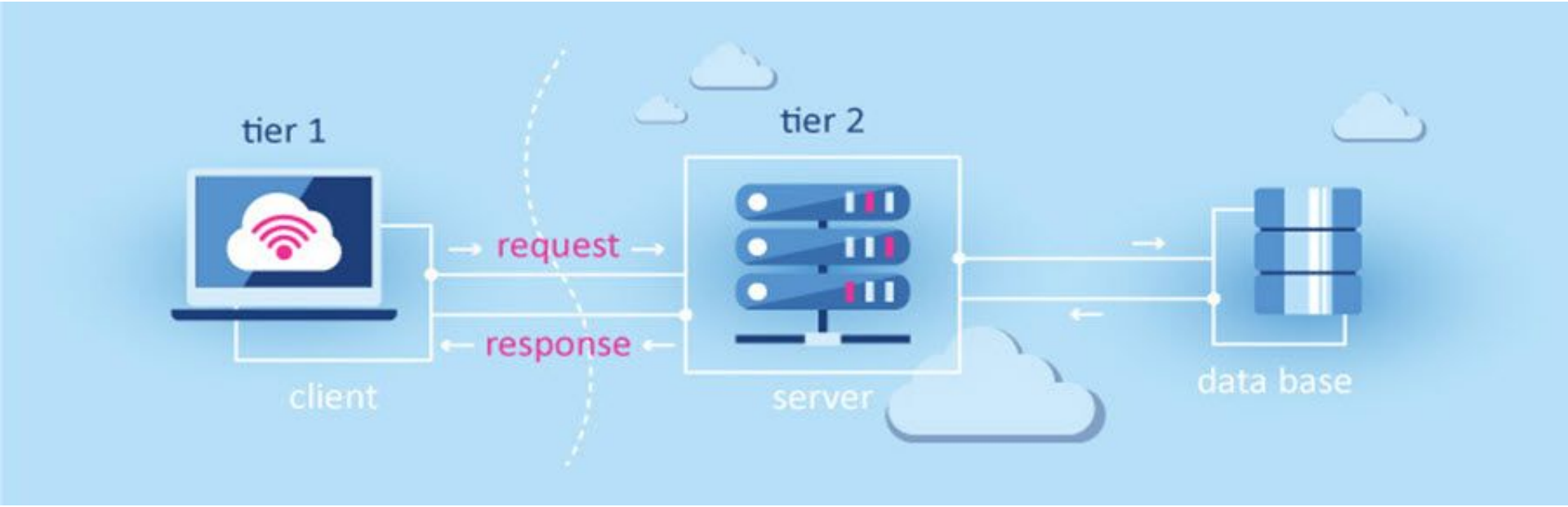
https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

Дворівнева архітектура



https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

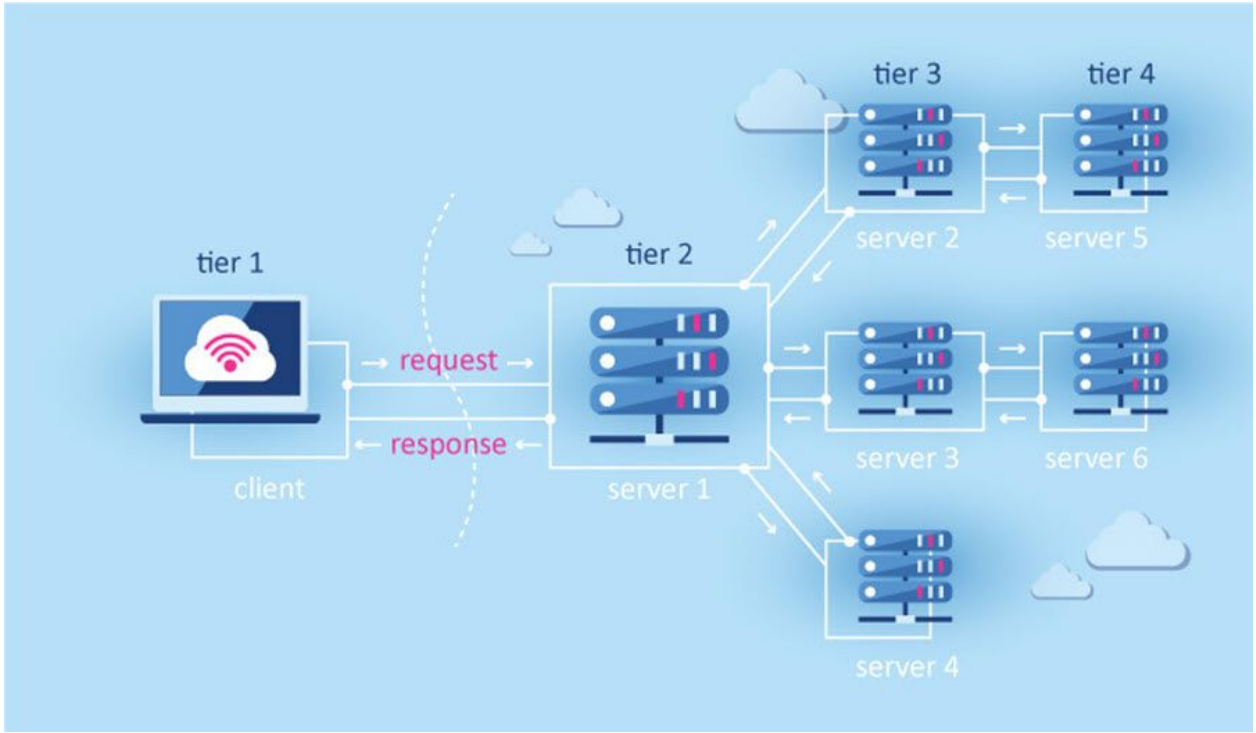
Трирівнева архітектура



https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

Багаторівнева архітектура

Досягається шляхом розширення трирівневої, способом встановлення додаткових серверів. Багаторівнева архітектура дозволяє підвищити ефективність роботи інформаційної системи, а також оптимізувати розподіл її програмно-апаратних ресурсів.

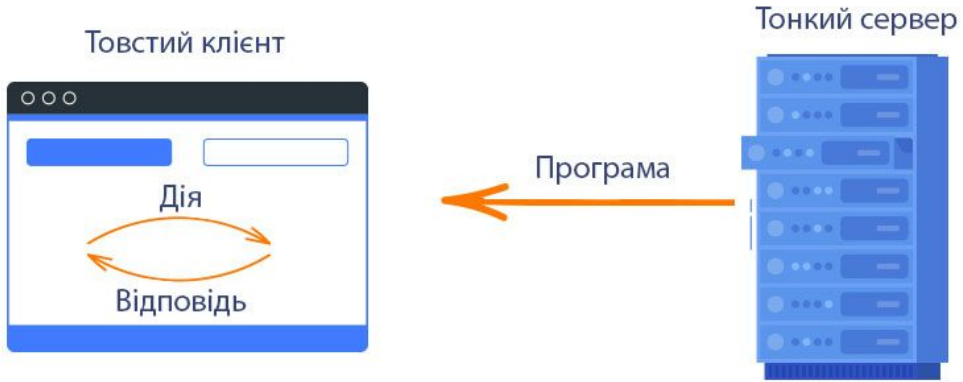


https://training.qatestlab.com/blog/technical-articles/thick-and-thin-client/

Товстий та тонкий клієнт

Товстий клієнт – клієнт, який проводить запитувані користувачем операції незалежно від центрального сервера. Центральний сервер в такому варіанті архітектури може використовуватися як сховище даних, обробка та надання яких переноситься на робочу машину клієнта.

Товстим клієнтом є робоча станція або ПК, які працюють під управлінням власної операційної системи і мають повний необхідний набір програмного забезпечення для реалізації завдань користувача.

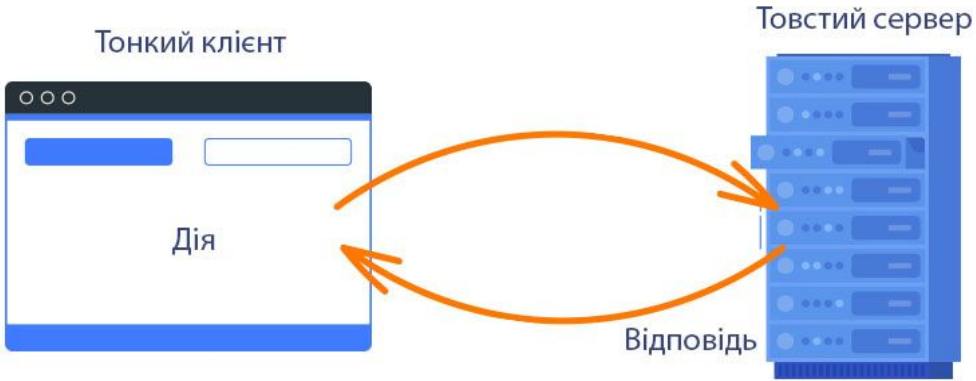


https://training.qatestlab.com/blog/technical-articles/thick-and-thin-client/

Товстий та тонкий клієнт

Тонкий клієнт – тип клієнта, який переносить завдання з обробки даних на сервер, не використовуючи свої обчислювальні можливості для їх реалізації. Обчислювальні ресурси такого клієнта дуже обмежені, їх повинно бути досить лише для запуску необхідного мережевого додатку, використовуючи, наприклад, web-інтерфейс.

Web-браузери і web-додатки, такі як WordPress, Google Docs та онлайн-ігри, є прикладами тонкого клієнта. Також до тонких клієнтів можна віднести і сайти-пошуковики Google і Yahoo.



Мережеві протоколи - HTTP & HTTPS

- HTTP – це протокол, в якому описано правила передачі даних в Інтернеті. Він допомагає браузеру завантажувати веб-сторінки, а серверу отримати інформацію, яку користувач ввів на сайті.
- HTTPS це той же протокол, але з надбудовою безпеки.
- За HTTP інформація передається у звичайному вигляді, а за HTTPS - у зашифрованому. Шифрувати дані потрібно, щоб хакери не змогли нічого прочитати, якщо їх перехоплять.
- Коли ви вводите щось на сайті, який працює за HTTPS, перед надсиланням даних на сервер браузер зашифровує інформацію. Щоб розшифрувати та прочитати її, потрібен спеціальний ключ, який зберігається лише на сервері. Таке шифрування називається криптографічним.

https://training.qatestlab.com/blog/technical-articles/client-server-architecture/
https://ssl.com.ua/blog/http-vs-https/

Browser cache

https://training.qatestlab.com/blog/technical-articles/cookies-cache-session-in-browser/

- Завдяки кешуванню даних, браузер зберігає частину інформації локально на комп'ютері користувача.
- Що зазвичай зберігається в кеші? Як правило, веб-сторінки одного сайту об'єднує загальний дизайн. Отже, існують веб-елементи, які повторюються на різних сторінках, наприклад, логотип сайту. Щоб не передавати одне і те ж зображення при кожному переході, логотип може бути локально збережений в кеші і при оновленні сторінки він буде завантажений з жорсткого диска користувача, а не з сервера. Крім зображень кешуватися може звук, відео, текст і т. ін.
- Кеш браузера має обмежений розмір. Ця величина може налаштовуватись. Коли розмір кешу досягає граничного значення, ті його частини, які не використовувалися найдовше, видаляються і звільняють місце новим елементам.

Cookies

https://training.qatestlab.com/blog/technical-articles/cookies-cache-session-in-browser/

- Куки (cookies) – це певний обсяг даних, який створюється веб-сервером після відвідування користувачем веб-сторінки і зберігається на комп'ютері користувача у вигляді окремого текстового файлу.
- Як правило, cookies містять ідентифікаційні дані, інформацію про користувача, про параметри і налаштуваннях, обраних під час роботи з веб-сторінкою, та іншу службову інформацію.

Для чого сайту ця інформація? Вона використовується, як правило, для:

- аутентифікації користувача (інакше довелося б вводити логін і пароль при кожному запиті);
- персоналізації – застосування обраних користувачем параметрів при подальших запитах (наприклад, мова сайту або товари в кошику);
- збору і аналізу статистики.

Cookies

https://training.qatestlab.com/blog/technical-articles/cookies-cache-session-in-browser/

- Куки (cookies) – це певний обсяг даних, який створюється веб-сервером після відвідування користувачем веб-сторінки і зберігається на комп'ютері користувача у вигляді окремого текстового файлу.
- Як правило, cookies містять ідентифікаційні дані, інформацію про користувача, про параметри і налаштуваннях, обраних під час роботи з веб-сторінкою, та іншу службову інформацію.

Для чого сайту ця інформація? Вона використовується, як правило, для:

- аутентифікації користувача (інакше довелося б вводити логін і пароль при кожному запиті);
- персоналізації – застосування обраних користувачем параметрів при подальших запитах (наприклад, мова сайту або товари в кошику);
- збору і аналізу статистики.

Ідентифікація, аутентифікація, авторизація

- Ідентифікація - отримання ідентифікатора користувача в системі.
  Ми кажемо системі: Я-Василь, система каже: Твоє ід - 987.
- Аутентифікація - підтвердження особи, перевірка автентичності.
  Ми кажемо: ось мій пароль, щоб ти знав що я точно Василь. Система каже: натисни “Ок” на твоєму телефоні і я повірю.
- Авторизація - система наділяє нас правами доступу, які повинен мати Василь :)

Що таке JSON

- **JavaScript Object Notation** – стандартний текстовий формат для представлення структурованих даних на основі синтаксису JavaScript. Він зазвичай використовується для передачі даних у веб-додатках (наприклад, відправлення деяких даних із сервера клієнту, таким чином, щоб це могло відображатися на веб-сторінці або навпаки).
- Об'єкт JSON може бути збережений у власному файлі, який в основному є текстовим файлом з розширенням .json

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Правила побудови Json файлів

JSON вимагає подвійних лапок, які будуть використовуватися навколо рядків та імен властивостей. Одинарні лапки недійсні.

Навіть одна недоречна кома або двокрапка можуть призвести до збою JSON-файлу і не працювати. Ви повинні бути обережні, щоб перевірити будь-які дані, які ви намагаєтеся використовувати (хоча згенерований комп'ютером JSON з меншою ймовірністю включає помилки, якщо програма генератора працює правильно).

Перевірити свій Json файл можна тут: https://jsonformatter.org/

Що таке API?

API (Application Programming Interface) — це набір правил, які дозволяють програмам спілкуватися між собою.

Якщо порівняти з реальним життям, то API — це як **офіціант у ресторані**:

- клієнт (додаток) → робить замовлення (запит до API),
- офіціант (API) → передає замовлення на кухню (сервер),
- кухня (база даних/логіка) → готує страву (результат),
- офіціант повертає клієнту готову страву (відповідь API).

Де використовуються API?

Чому тестування API важливе?

1. **Центральна частина системи.** Якщо API працює неправильно, не працюватиме і веб, і мобільний застосунок.
2. **Раннє виявлення помилок.** API створюють ще до інтерфейсу. Тестування дозволяє знайти баги до появи UI.
3. **Надійність інтеграцій.** Один API може використовуватися різними клієнтами (Android, iOS, веб, інші сервіси).
4. **Швидкість.** Тести API запускаються швидше за UI-тести, бо не потрібно відкривати браузер чи емулятор.
5. **Економія коштів.** Виправити помилку на рівні API дешевше, ніж після релізу.

1. REST API

REST (Representational State Transfer) – найпопулярніший тип API.

Використовує протокол **HTTP**.
Дані зазвичай передаються у форматі **JSON** (рідше XML).
Має стандартні методи:

- GET – отримати дані,
- POST – створити новий ресурс,
- PUT – оновити ресурс,
- DELETE – видалити ресурс.

2. SOAP API

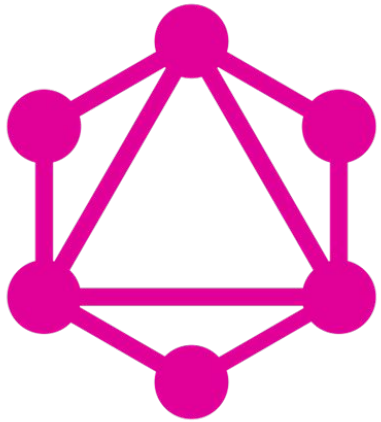
SOAP (Simple Object Access Protocol) – старіший стандарт, популярний у корпоративних системах.

- Працює поверх **HTTP**, але використовує **XML** для запитів і відповідей.
- Має сувору специфікацію і багато правил.
- Використовує **WSDL (Web Services Description Language)** для опису сервісу.

3. GraphQL API

GraphQL – сучасний стандарт від Facebook.

- Клієнт може сам вибирати, які дані отримати.
- Використовує єдиний **endpoint** (наприклад, /graphql).
- Працює через HTTP, зазвичай у форматі JSON.



GraphQL

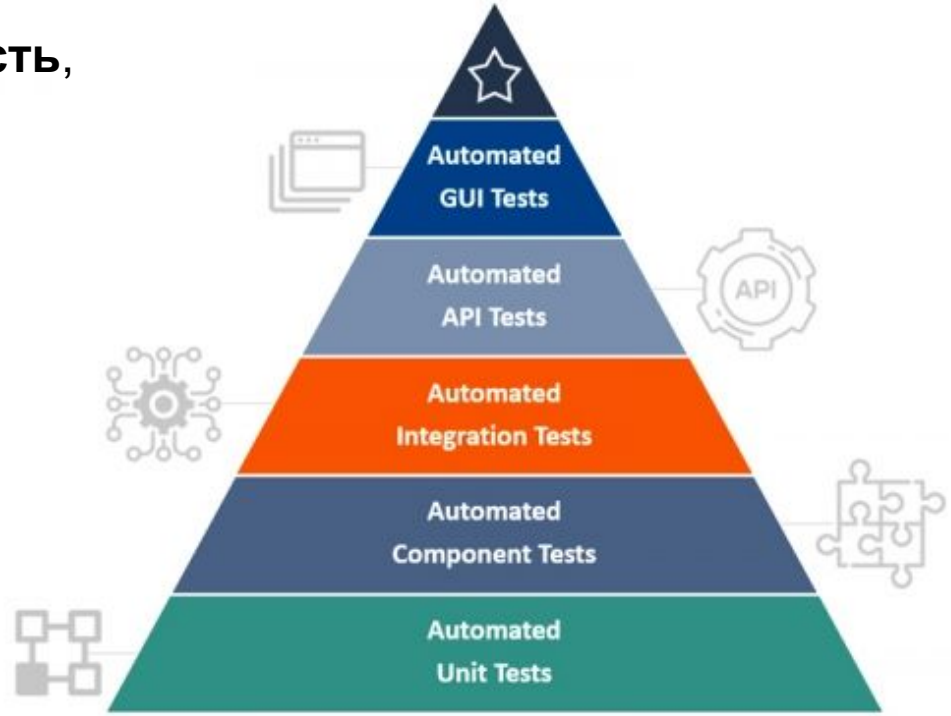
З чого складається HTTP (API) запит, що ми надсилаємо

- Request URL - це адреса, за якою ми посилаємо запит
- Headers (заголовки) - дозволяють клієнту і серверу передавати допоміжну інформацію
- Тип запиту або метод запиту
- Тіло запиту - файл з інформацією для запита (найчастіше JSON)

Що перевіряють у API тестуванні

API може працювати правильно тільки тоді, коли воно:

- виконує свою **функціональність**,
- правильно обробляє **дані**,
- захищене від зловмисників,
- працює **швидко й стабільно**.

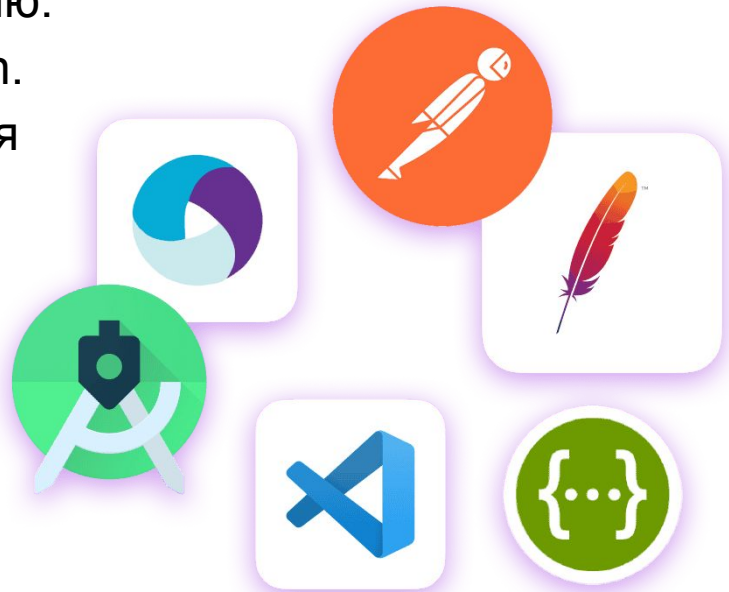


Етапи API тестування

1. Вивчити документацію (Swagger, OpenAPI).
2. Створити набір тестів: позитивні, негативні, boundary cases.
3. Виконати ручне тестування у **Postman/Insomnia**.
4. Перейти до автоматизації (Python, Java, JS).
5. Інтегрувати тести у CI/CD (GitHub Actions, Jenkins).

Інструменти для API тестування

- **Postman** – зручний для ручного тестування, підтримує автоматизацію.
- **Insomnia** – простий аналог Postman.
- **Swagger** – документація + генерація запитів.
- **SoapUI** – тестування SOAP і REST.
- **JMeter** – навантажувальне тестування.
- **K6** – сучасний інструмент для performance-тестів.



Види тестування продуктивності

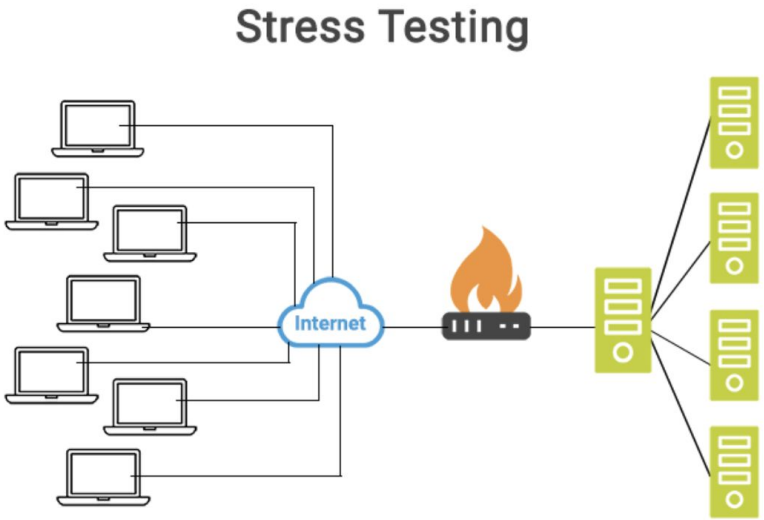
- навантажувальне тестування (Performance and Load Testing)
- стресове тестування (Stress Testing)
- тестування стабільності чи надійності (Stability / Reliability Testing)
- об'ємне тестування (Volume Testing)

Навантажувальне тестування (Performance and Load Testing)

Це автоматизоване тестування, яке імітує роботу певної кількості бізнес-користувачів на загальному (спільному для них) ресурсі.

Стресове тестування (Stress Testing)

Тестування на стрес (Stress Testing) дозволяє перевірити, наскільки працездатним є застосунок і система в цілому в умовах стресу, а також оцінити здатність системи до відновлення, тобто повернення до нормального стану після припинення впливу стресового навантаження. В даному контексті стресом може бути збільшення інтенсивності виконання операцій до дуже високих значень або аварійна зміна конфігурації сервера. Також одним із завдань під час стрес-тестування може бути оцінка деградації продуктивності, тому цілі стрес-тестування можуть перекриватися з цілями тестування продуктивності.

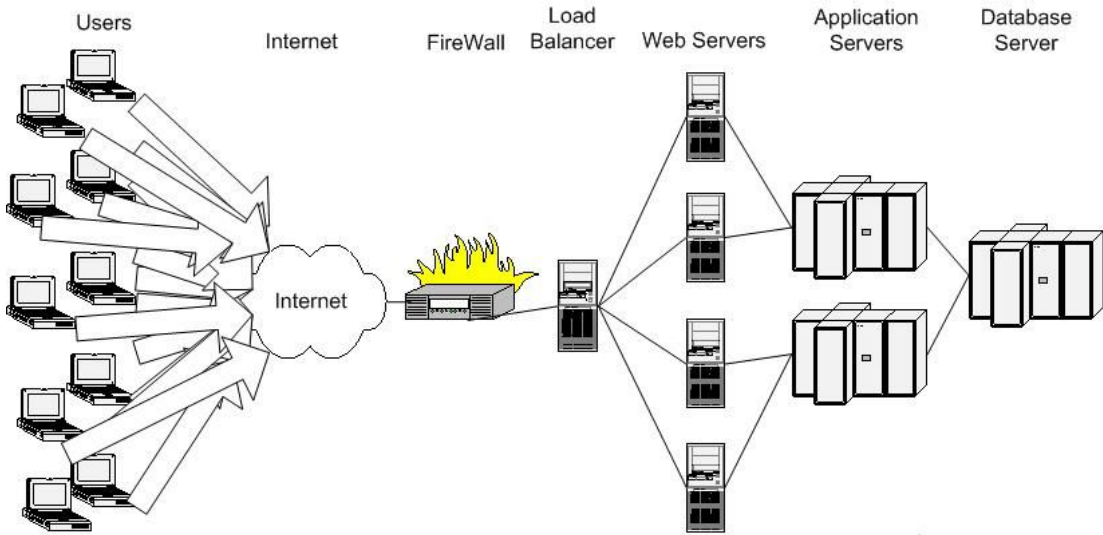


Тестування стабільності чи надійності (Stability / Reliability Testing)

Тестування стабільності або надійності (Stability / Reliability Testing) має на меті перевірку працездатності додатка під час тривалого (багатогодинного) тестування з середнім рівнем навантаження.

Об'ємне тестування (Volume Testing)

Обсягове тестування (Volume Testing) має на меті отримання оцінки продуктивності при збільшенні обсягів даних у базі даних застосунку.



6. Приклади тестових сценаріїв

Endpoint:
GET /users

- Очікуваний результат: 200 OK + список користувачів.

Endpoint:
GET /users/{id}

- Позитивний: існуючий ID → 200 OK + дані користувача.
- Негативний: неіснуючий ID → 404 Not Found.

Endpoint:
POST /users

- Валідні дані → 201 Created.
- Пропущене обов’язкове поле → 400 Bad Request.

Авторизація

- Без токена → 401 Unauthorized.
- Неправильний токен → 403 Forbidden.

7. Кращі практики

- Тести поділяти на **позитивні** та **негативні**.
- Використовувати **Mocks і Stubs**, щоб не залежати від сторонніх сервісів.
- Автоматизувати критичні кейси.
- Використовувати **data-driven testing** (один тест із різними наборами даних).
- Інтегрувати тести в CI/CD для постійної перевірки.