API and WebSocket TESTING

Поліна Супранович / Світлана Дудченко / Савелій Уснич
Senior QA Automation Senior Test Analyst Senior QA

Про що поговоримо

- Що таке API. Історія виникнення
- Структура та компоненти HTTP-запитів
- Використання ШІ в тестуванні API
- Основи тестування API
- Чому з’явився WebSocket
- Демо: відстеження помилки
- Особливості тестування WebSocket
- Інструменти для тестування API та WebSocket
- AI технології в тестуванні

Що ми пропонуємо

- Актуальні технології та сервіси
- Досвід роботи на різних проектах
- Цікаві практичні приклади

Як ми працюємо разом

- Активне слухання
- Участь в інтерактивній частині презентації
- Запитання та коментарі максимально вітаються
- Зворотний зв'язок
- Запити на нові теми

Які асоціації у вас викликає поняття API?

menti.com

6903 0741

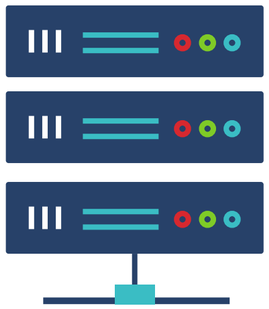
Waiting for participants

Як працювали веб-застосунки до появи API?

Користувач

Запит

HTML сторінка



Сервер

Користувач заповнює форму для пошуку книги

Форма надсилає запит на сервер через протокол HTTP

Сервер запускає скрипт, який виконує запит до бази даних

Результати запиту вставляються в HTML-сторінку, а сервер повертає її клієнту

Браузер відображає готову сторінку з результатами

Недоліки старого підходу

1. Сильне зв’язування даних та інтерфейсу
Сервер формував HTML, поєднуючи дані й оформлення. Зміни дизайну вимагали правок серверного коду.

2. Складність повторного використання даних
Дані були «вшиті» в HTML. Щоб використати їх в іншому форматі, доводилося парсити код або дублювати логіку.

3. Труднощі інтеграції із зовнішніми сервісами
Не було стандартів обміну даними. Доводилося скрапити сторінки або писати програми-посередники, ускладнюючи інтеграцію.

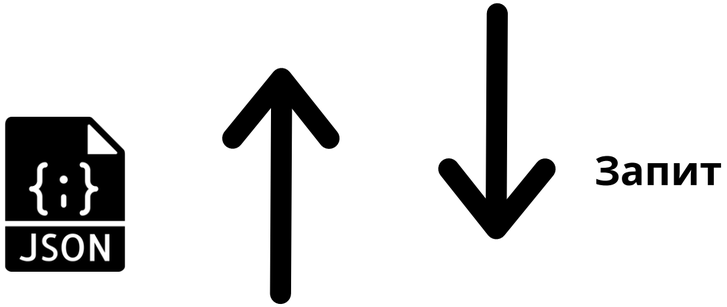
4. Велике навантаження та надмірність під час передавання даних
Сервер надсилав повний HTML за кожного запиту, навіть за мінімальних змін. Це збільшувало трафік і сповільнювало роботу.

5. Обмежені можливості масштабування
Генерація сторінок для різних пристроїв ускладнювала систему. Зростання аудиторії або поява нових платформ вимагали серйозних доопрацювань.

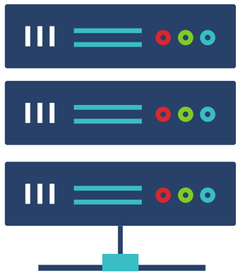
Як же все змінилося?

Як працюють web-застосунки з API
API - application programming interface

Користувач



Запит



Сервер

Користувач заповнює форму для пошуку книги

Форма надсилає запит на сервер через протокол HTTP

Сервер отримує запит і обробляє його через API.
API викликає серверний метод, який виконує запит до бази даних

Сервер повертає лише дані (у форматі JSON) через API

Клієнт отримує JSON-відповідь. Браузер або застосунок відображає тільки дані, потрібні користувачу, без повного перезавантаження сторінки

Переваги використання API

1) Розділення даних та інтерфейсу
Сервер передає лише «чисті» дані (JSON, XML тощо), а візуальна частина обробляється на боці клієнта

2) Стандартизований формат взаємодії
API чітко визначає протоколи та формати запитів/відповідей, що спрощує обмін даними між системами

3) Зручність інтеграції
Різні застосунки (веб, мобільні, IoT тощо) можуть підключатися до єдиного API та використовувати одну логіку без дублювання

4) Менше трафіку та вища продуктивність
Сервер передає лише потрібні дані, а не всю HTML-сторінку, що економить ресурси та пришвидшує відгук

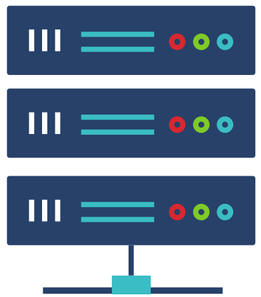
5) Гнучкість і масштабованість
API легко адаптувати до нових вимог і зростаючого навантаження, не зачіпаючи дизайн та користувацький інтерфейс

Клієнт-серверна архітектура

Користувач

HTTP відповідь

HTTP запит



Сервер

запит

відповідь

База даних

- **Клієнт —** це інтерфейс, з яким взаємодіє користувач. Клієнт відображає дані та надсилає запити на сервер

- **Сервер** — це центральна частина системи, яка:
  1) зберігає дані
  2) обробляє запити клієнта
  3) повертає дані клієнту через API

Переваги клієнт-серверної архітектури

Розділення завдань: Клієнт відповідає за відображення даних і користувацький інтерфейс, а сервер — за зберігання даних, бізнес-логіку та обробку запитів. Це спрощує розробку та підтримку системи

Легкість оновлення: Клієнт і сервер можуть оновлюватися незалежно один від одного. Наприклад, можна внести зміни на сервері, не зачіпаючи клієнтські застосунки

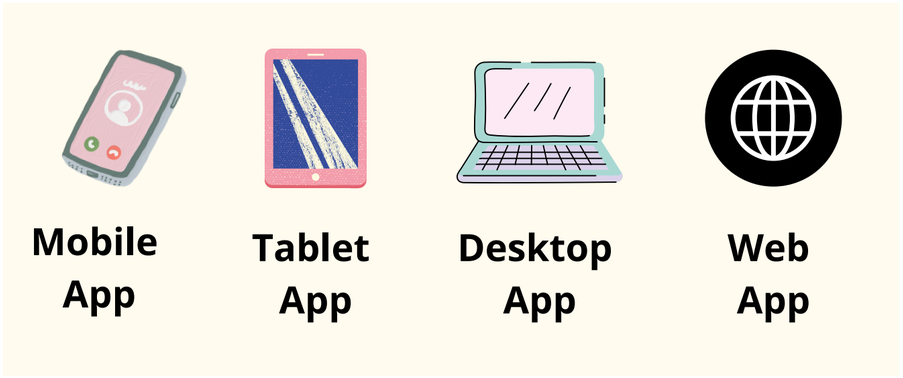
Масштабованість: Сервер може обслуговувати багато клієнтів одночасно, що дозволяє створювати системи, доступні з різних пристроїв (веб, мобільний застосунок, сторонні сервіси)

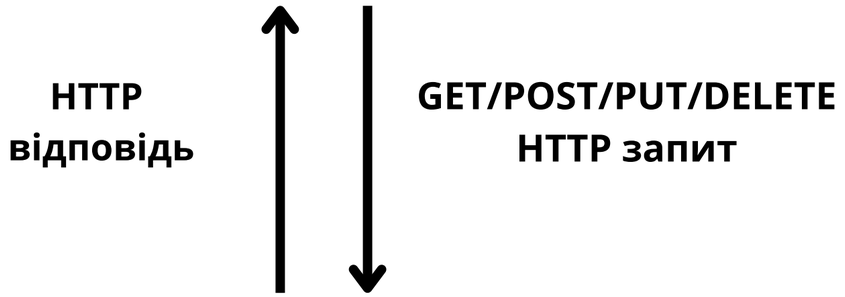
Підвищена безпека: Оскільки логіка й дані зберігаються на сервері, доступ до них можна обмежити та захистити. Клієнт бачить лише ті дані, які йому потрібні

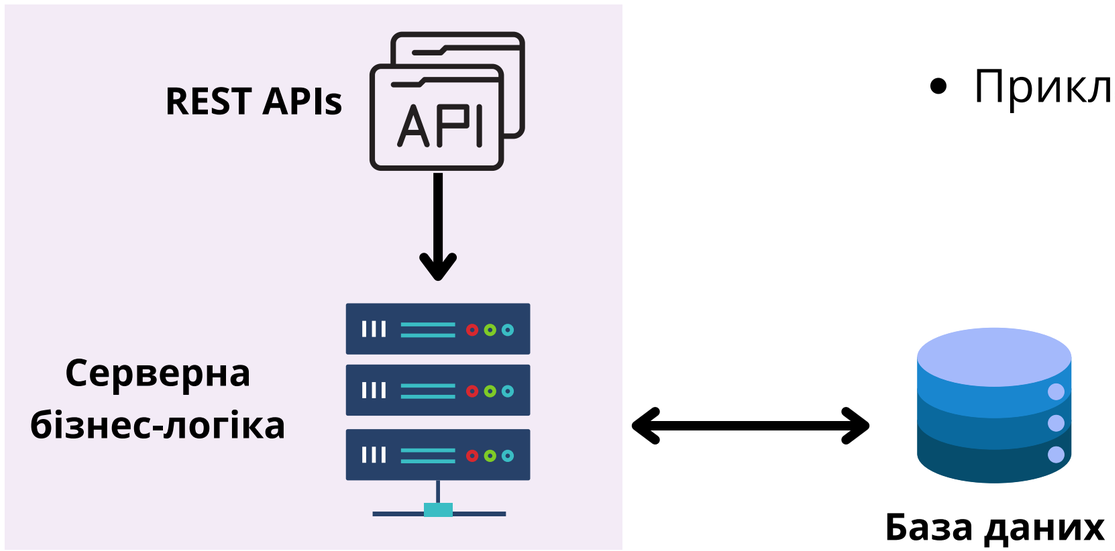
Підтримка різних платформ: Клієнтом може бути будь-який застосунок — веб, мобільний, десктопний. Усі вони можуть працювати з одним сервером через API

Зручність інтеграції: Сторонні сервіси можуть підключатися до сервера через API для отримання даних або виконання операцій

Типи API:
REST API (Representational State Transfer)







- Особливості: Простий, гнучкий, використовує протокол HTTP та стандартні методи (GET, POST, PUT, DELETE)
- Формат даних: JSON, XML
- Застосування: Веб-розробка, мобільні застосунки, мікросервіси
- Приклад використання: API для Google, X, Spotify

Основи роботи API: формати даних

Формати даних — це способи подання інформації, які використовуються для обміну даними між клієнтом і сервером. Найпопулярніші формати включають JSON, XML і бінарні дані

JSON (JavaScript Object Notation)

json

```json
{
  "id": 1,
  "name": "John Doe",
  "email": "john@example.com"
}
```

JSON — це простий текстовий формат обміну даними, його легко читати людині, і він підтримується більшістю мов програмування. Він чудово підходить для API, оскільки легко перетворюється на об’єкти або словники.

XML (eXtensible Markup Language)

xml

```xml
<user>
  <id>1</id>
  <name>John Doe</name>
  <email>john@example.com</email>
</user>
```

XML — це текстовий формат із тегами, який дозволяє описувати структуру та зв’язки даних. Він потужний, але більш громіздкий і складніший в обробці порівняно з JSON

Бинарные данные:

Бінарні дані — це двійковий формат, не призначений для читання людиною, але зручний для передавання мультимедіа: зображень, аудіо, відео. Вони компактніші за текстові формати, але потребують спеціальної обробки.

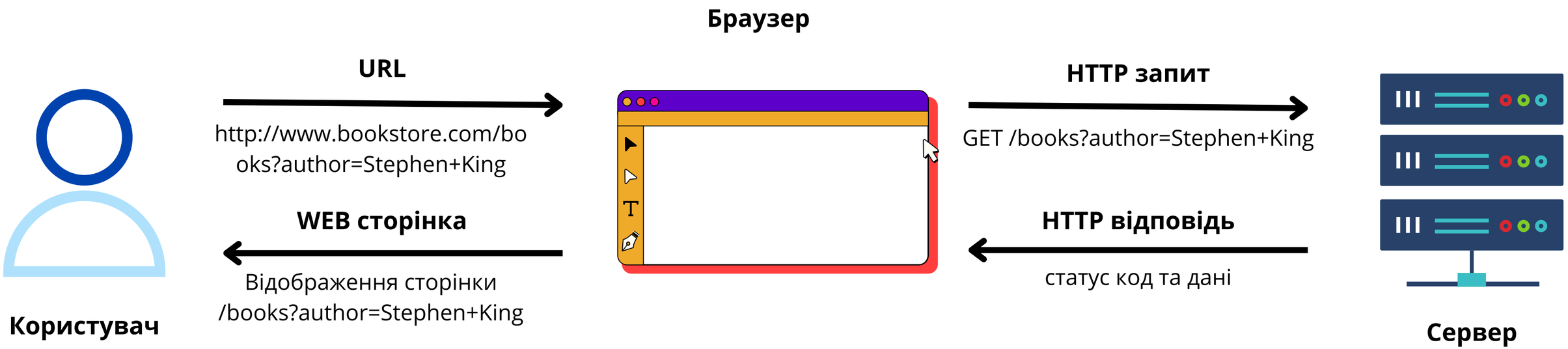
Основи роботи API: HTTP та пов’язані технології

HTTP (HyperText Transfer Protocol) — це протокол, який використовується для передавання даних у мережі Інтернет. Він є основою обміну інформацією між клієнтом і сервером, дозволяючи запитувати, надсилати та отримувати дані, такі як веб-сторінки, зображення, відео та інші ресурси

• **Запити та відповіді:** HTTP базується на принципі клієнт-сервер, де клієнт надсилає запити на сервер, а сервер відповідає, надсилаючи запитані ресурси.

• **Методи запиту:** В HTTP існують різні методи запиту, такі як GET (для отримання даних), POST (для надсилання даних), PUT (для оновлення даних), DELETE (для видалення даних) та інші.

• **Статус-коди:** Кожна відповідь від сервера супроводжується статус-кодом, який вказує на результат обробки запиту. Наприклад, 200 OK означає успішне виконання запиту, а 404 Not Found — що запитаний ресурс не знайдено.



Основи структури HTTP-запиту

HTTP-запит — це повідомлення, яке клієнт (наприклад, браузер або API-клієнт) надсилає серверу для отримання або передавання даних. Він складається з методу (наприклад, GET, POST), стартового рядка (URL і версії протоколу), заголовків (метаданих запиту) та, за потреби, тіла з даними, що передаються.

Структура HTTP-запиту:

Стартовий рядок: Вказує, куди надсилається запит, і містить метод (HTTP Method), який визначає, що саме клієнт хоче зробити.

Заголовки (Headers): Передають додаткову інформацію про запит, наприклад, формат даних

Тіло (Body): Містить дані, які надсилаються на сервер (використовується в запитах типу POST, PUT і PATCH)

Приклад HTTP запиту

```
1  POST /api/users HTTP/1.1          Метод + шлях + версія HTTP
2  Host: example.com                 До якого хосту йде запит
3  Content-Type: application/json    Дані у тілі запиту у форматі JSON
4  Authorization: Bearer abc123xyz   Токен авторизації
5  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   Інформація про клієнта
6  Content-Length: 60                Довжина тіла запиту
7  {
8    "name": "John"
9    "email": "johny@example.com",                Передані дані
10   "password": "securepassword123"
11 }
```

Основні типи HTTP-методів

HTTP request methods (або HTTP-методи) — це різні типи дій, які клієнт може виконати на сервері з використанням протоколу HTTP. Кожен метод визначає, яку дію потрібно виконати з ресурсом, до якого надсилається запит

GET	Запитує ресурс із сервера. Це найчастіше використовуваний метод для отримання даних. Запити з методом GET зазвичай не змінюють стан даних на сервері та можуть кешуватися.
POST	Надсилає дані на сервер, наприклад, для створення нового ресурсу. Цей метод використовується, коли потрібно передати дані в тілі запиту (наприклад, форма на сайті).
PUT	Використовується для оновлення наявного ресурсу на сервері. Зазвичай PUT надсилає дані, які мають повністю замінити старий вміст ресурсу.
DELETE	Видаляє ресурс із сервера. Цей метод використовується для видалення даних.
PATCH	Частково оновлює ресурс. На відміну від PUT, який замінює весь ресурс, PATCH застосовує лише зміни, зазначені в запиті.
HEAD	Схожий на GET, але не повертає тіло відповіді. Використовується для отримання метаданих про ресурс, наприклад заголовків, без завантаження вмісту.
OPTIONS	Запитує інформацію про підтримувані методи для ресурсу. Це корисно, щоб визначити, які операції дозволені для певного ресурсу на сервері.

Основи структури HTTP-відповіді

HTTP-відповідь — це повідомлення, яке сервер надсилає клієнту у відповідь на HTTP-запит. Вона складається з трьох основних частин:
Стартовий рядок (Status Line): містить код стану та опис результату запиту.
Заголовки (Headers): містять метадані про відповідь, такі як тип вмісту, довжина вмісту, інформація про сервер та інші дані, які допомагають клієнту зрозуміти, як обробити відповідь.
Тіло відповіді (Body): містить дані, які сервер надсилає у відповідь на запит. Наприклад, це може бути HTML-сторінка, JSON-дані або зображення.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 98
Date: Mon, 01 Jan 2025 12:00:00 GMT
Server: Apache/2.4.41 (Ubuntu)

{
  "status": "success",
  "message": "Request processed successfully",
  "data": {
    "id": 123,
    "name": "John",
    "email": "john@example.com"
  }
}
```

Приклад HTTP-відповіді

Стартовий рядок (Status Line)
• HTTP/1.1 200 OK → означає, що запит виконано успішно.

Заголовки (Headers)
• Content-Type: application/json → вказує, що дані в тілі відповіді у форматі JSON.
• Content-Length: 98 → довжина тіла відповіді (у байтах).
• Date: Mon, 01 Jan 2025 12:00:00 GMT → дата і час відповіді.
• Server: Apache/2.4.41 (Ubuntu) → інформація про сервер.

Тіло відповіді (Body)
• JSON-об’єкт із полями status, message і data.
• У data міститься інформація про користувача з id, name та email.

Коди відповіді HTTP

HTTP response codes (коди відповіді HTTP) — це числові коди, які сервер надсилає у відповідь на запит клієнта. Вони показують статус обробки запиту та можуть вказувати на успішне виконання операції, помилку або потребу в подальших діях.

1xx – Інформаційні (Informational)

- **100 Continue**
  Сервер отримав частину запиту й очікує продовження
- **101 Switching Protocols**
  Сервер змінює протокол за запитом клієнта

Сервер отримав запит і обробляє його, але відповідь ще не готова

2xx – Успішні (Success)

- **200 OK**
  Запит виконано, дані повернено
- **201 Created**
  Ресурс створено (наприклад, у POST)
- **204 No Content**
  Запит виконано, без тіла відповіді

Запит успішно оброблено, і сервер повернув запитані дані

3xx – Перенаправлення (Redirection)

- **301 Moved Permanently**
  Ресурс переміщено назавжди
- **302 Found**
  Ресурс тимчасово переміщено
- **304 Not Modified**
  Дані не змінилися

Запит потребує подальших дій, наприклад, перенаправлення (редиректу)

4xx – Помилки клієнта (Client Errors)

- **400 Bad Request**
  Некоректний запит
- **401 Unauthorized**
  Потрібна авторизація
- **403 Forbidden**
  Доступ заборонено, навіть якщо автентифікацію пройдено
- **404 Not Found**
  Ресурс не знайдено

Помилка на стороні клієнта (наприклад, невірний запит, відсутні права)

5xx – Помилки сервера (Server Errors)

- **500 Internal Server Error**
  Внутрішня помилка сервера
- **502 Bad Gateway**
  Сервер отримав некоректну відповідь
- **503 Service Unavailable**
  Сервер тимчасово не працює
- **504 Gateway Timeout**
  Минув час очікування відповіді

Помилка на боці сервера (наприклад, проблеми з обробкою запиту)

API авторизація та автентифікація

Аутентифікація — це процес перевірки автентичності користувача перед входом у додаток, тобто підтвердження, що він той, за кого себе видає. Авторизація, своєю чергою, визначає, які дії або ресурси доступні користувачу після успішної аутентифікації.

Аутентифікація

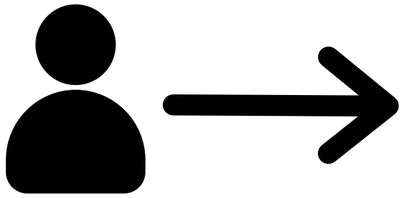
- Підтвердження особи користувача.
- Відповідає на запитання: **Хто ти?**
- Відбувається до авторизації

Приклади:
– Введення логіна і пароля
– Відправка токена
– Вхід через Google / GitHub (OAuth)

Авторизація

- Визначення прав доступу користувача.
- Відповідає на запитання: **Що тобі дозволено?**
- Виконується після автентифікації.

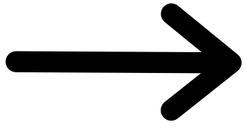
Приклади:
– Доступ до розділу «адмін»
– Можливість видалити дані
– Перегляд закритої інформації



Користувач увійшов у свій акаунт.

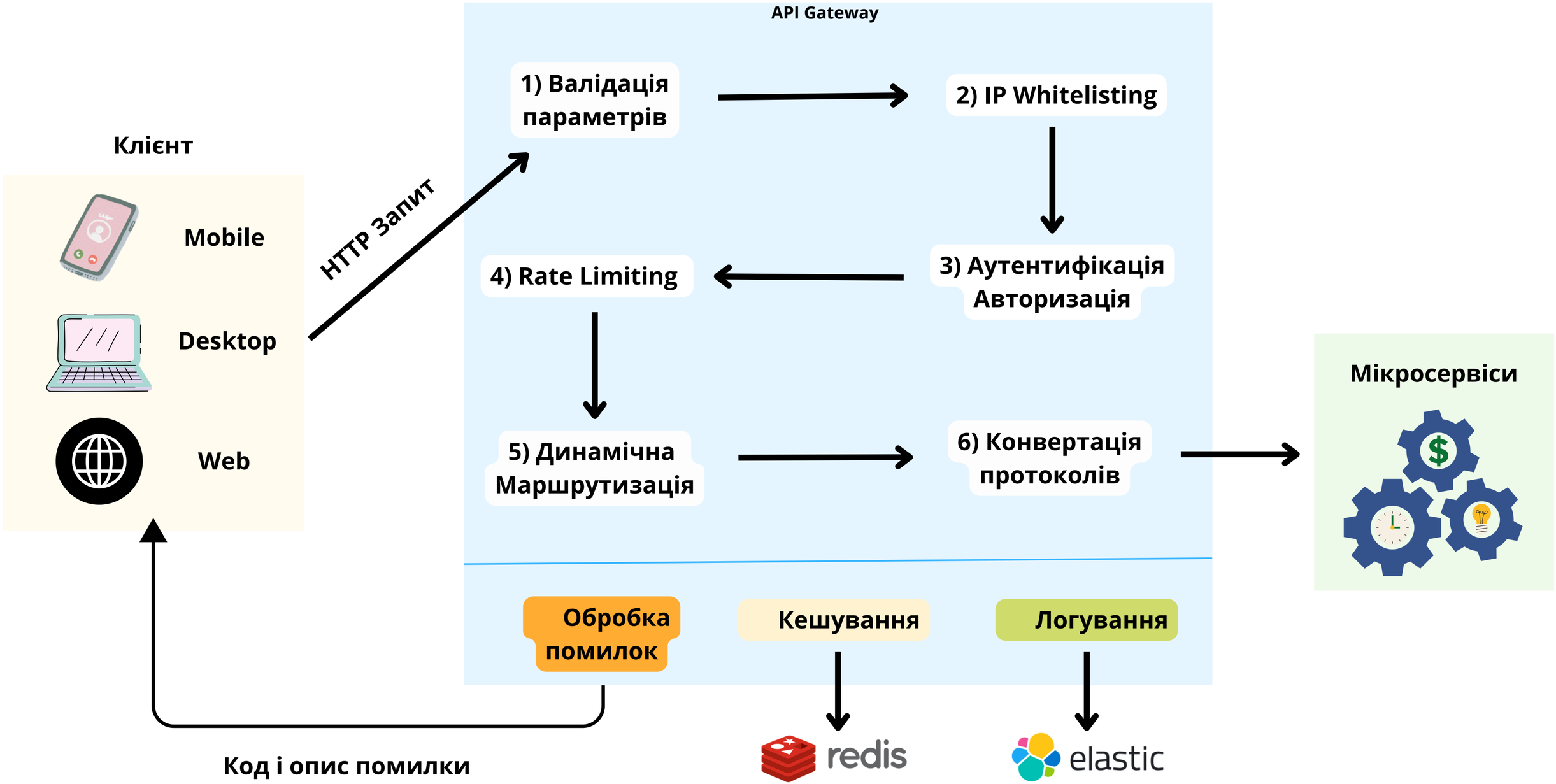
Автентифікація успішно пройдена

Користувач хоче видалити чужий документ.



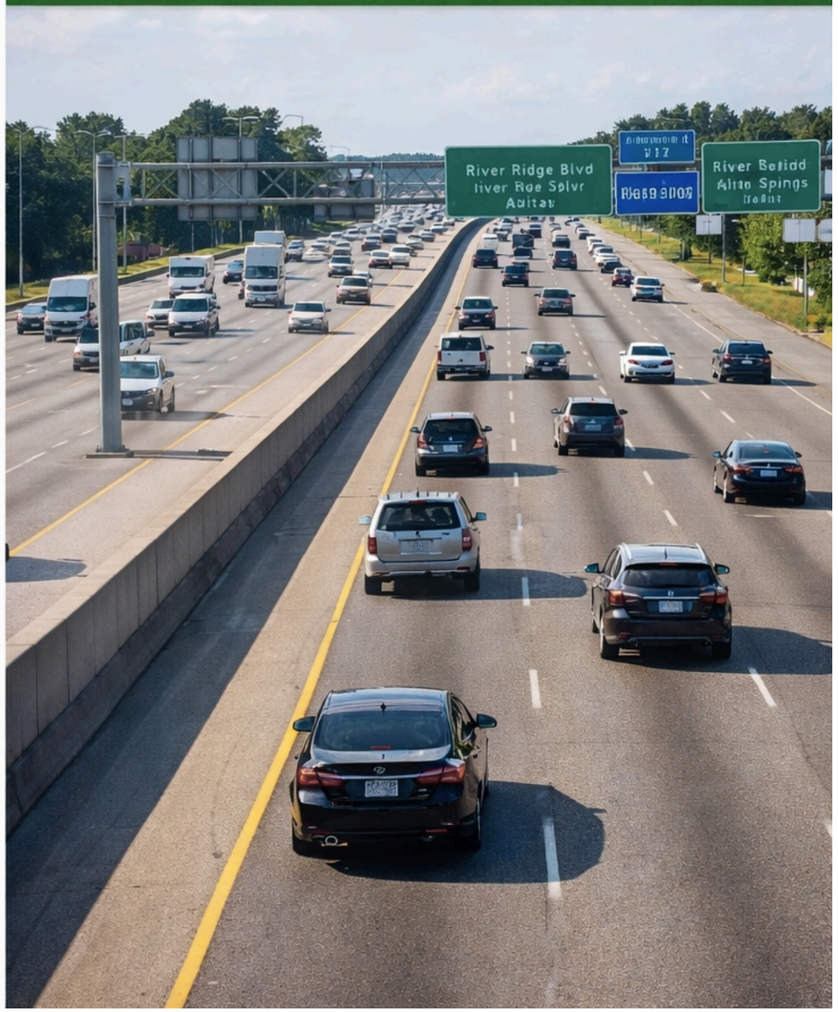
Авторизація не пройдена

Що таке API Gateway і навіщо він потрібен?



Request / Response:

Open Connection:



WebSockets

Що таке WebSocket. Коротка історія

Що було ДО:

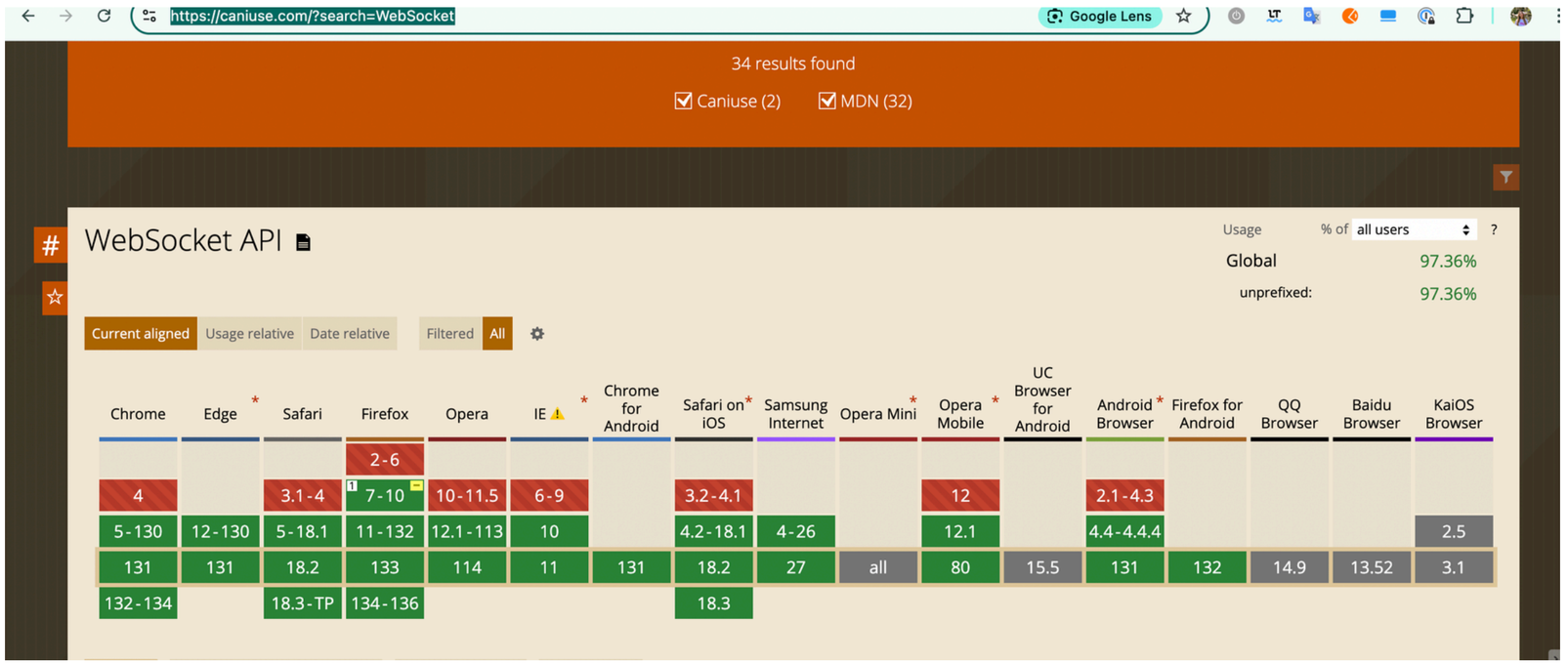
- **Ajax** — це метод асинхронного обміну даними з сервером у фоновому режимі та оновлення частин вебсторінки без необхідності її повного перезавантаження.
- **Comet** — це модель проєктування вебзастосунків, яка дозволяє вебсерверу надсилати дані в браузер (технологія «штовхання» даних або server push).

Порівняння:

Ajax ініціюється виключно клієнтом (HTTP Request -> HTTP Response).

Comet — це скоріше парасольковий термін для хаків на кшталт Long Polling, які імітували постійне з'єднання до появи стандартів HTML5.

У 2010 році Google Chrome 4 став першим браузером із підтримкою WebSockets, проклавши шлях для інших браузерів



Браузер	Поддержка WebSocket?
Google Chrome	(починаючи з версії 16)
Mozilla Firefox	(починаючи з версії 11)
Microsoft Edge	
Apple Safari	(починаючи з версії 7)
Opera	(починаючи з версії 12.1)
Internet Explorer	(починаючи з версії 10, але з обмеженнями)

Mentimeter

У яких сервісах WebSocket має перевагу над звичайним API?

menti.com

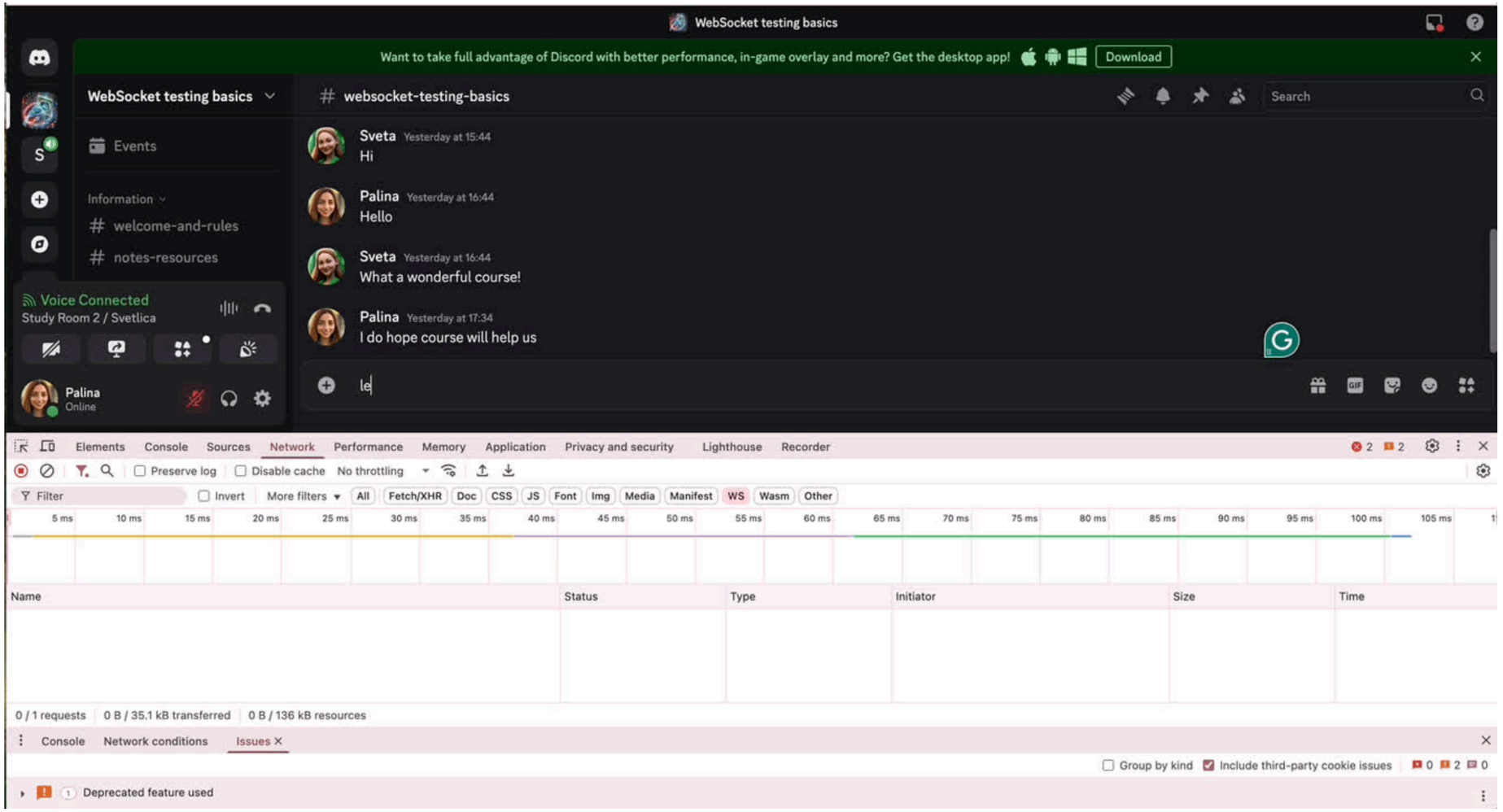
6903 0741

Waiting for participants

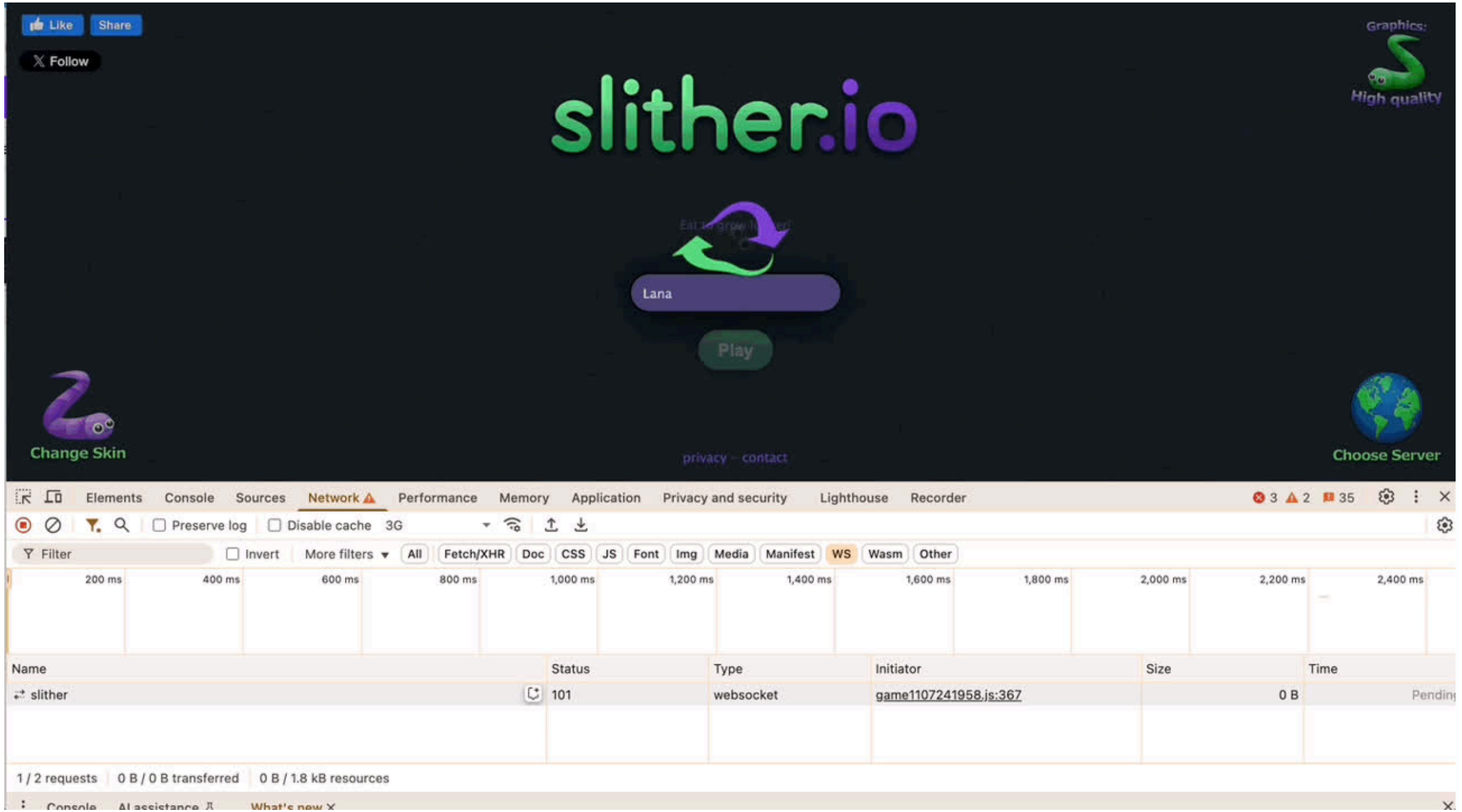
Next slide

Області застосунку

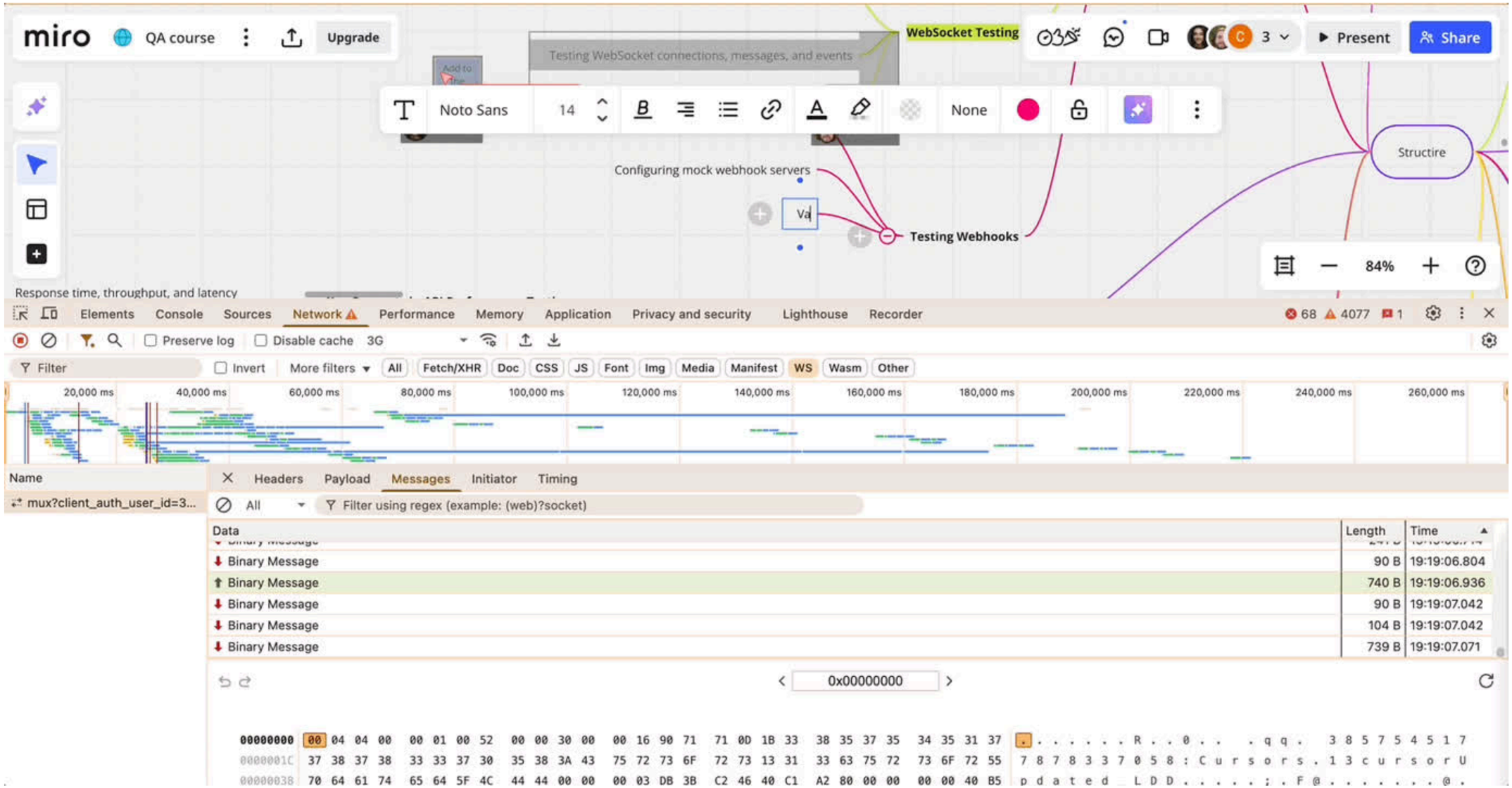
1. Чат-додатки - Передача повідомлень та синхронізація редагування тексту між багатьма користувачами одночасно:



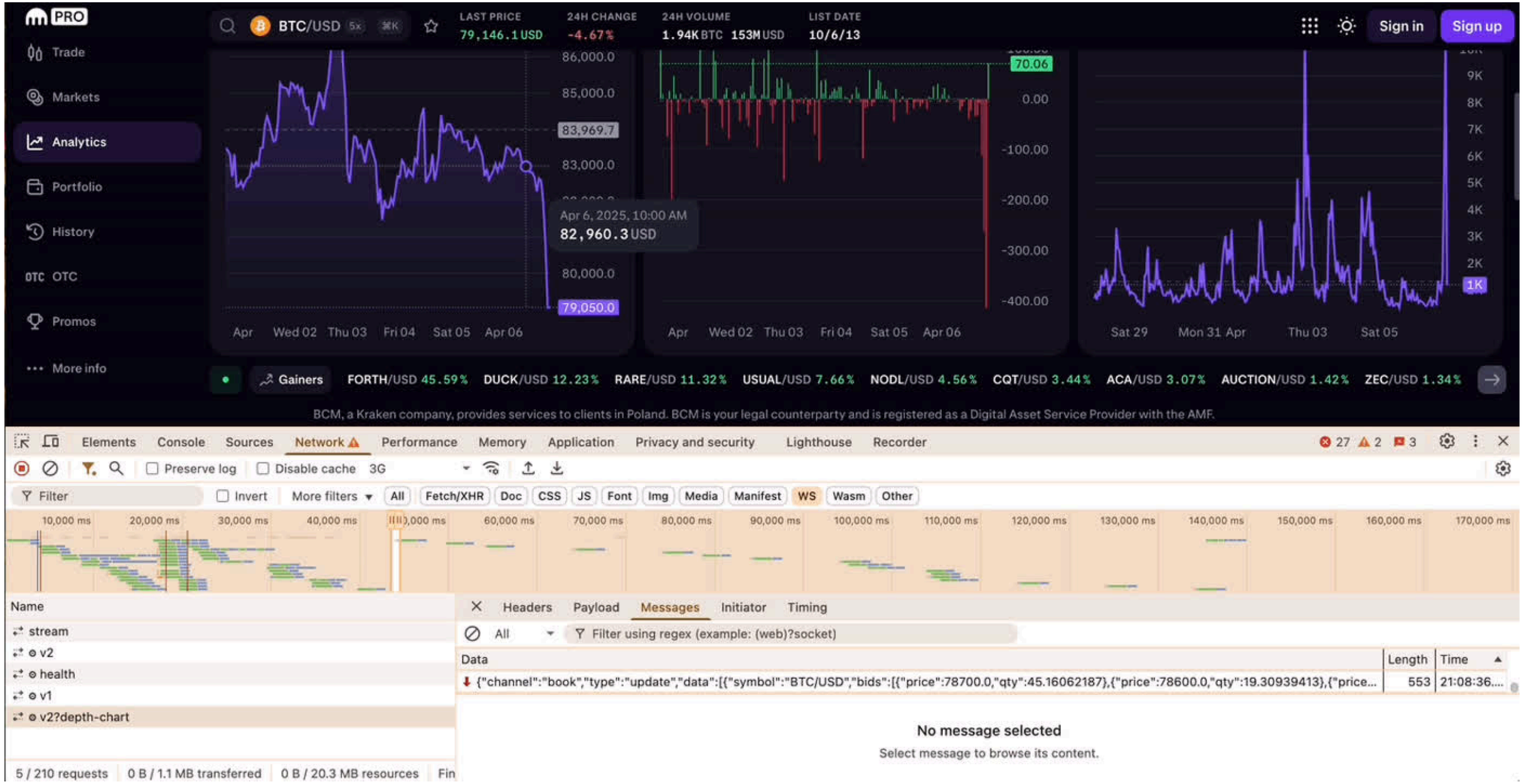
2. Інтерактивні онлайн ігр



3. Спільне редагування



4. Фінансові застосунки та меаркетинг.

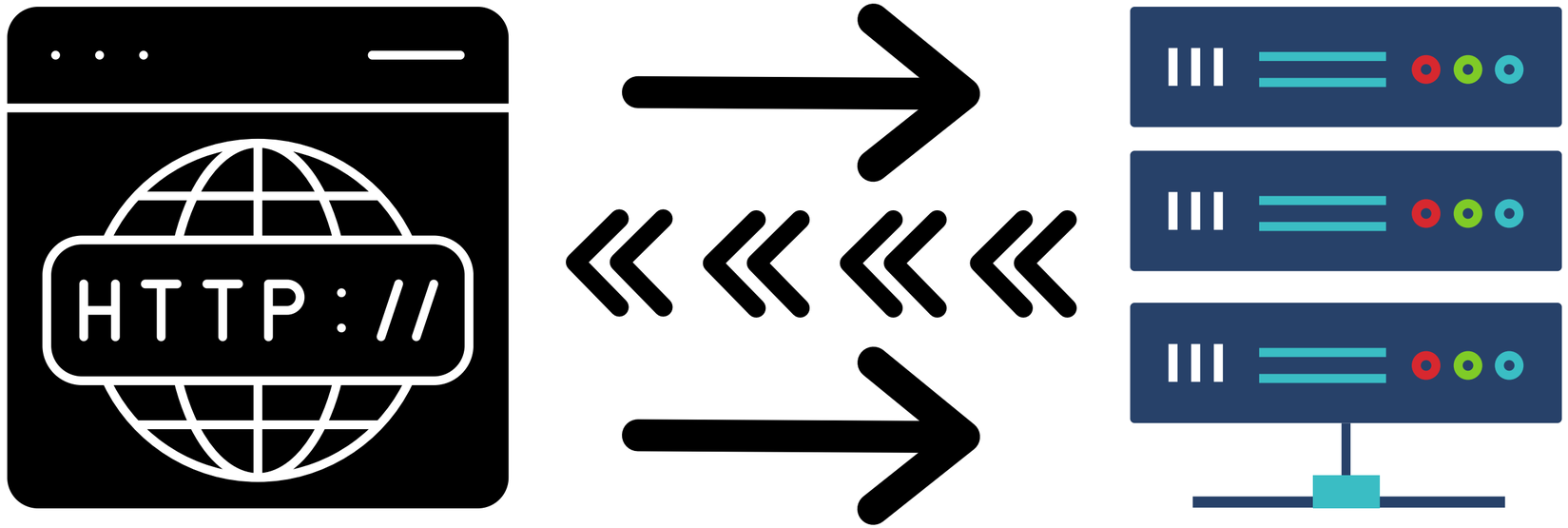


5. Соціальні мережі.

6. Системи моніторінгу

Знайомство з WebSocket

WebSocket (веб-сокет) - це протокол для двостороннього зв'язку між клієнтом і сервером через веб-з'єднання. Він надає можливість передавати дані в режимі реального часу без необхідності постійного запиту до сервера.



Як працює WebSocket

WebSocket починається з **HTTP-рукопожаття** (handshake), яке потім підвищується до постійного WebSocket-з'єднання. При встановленні з'єднання клієнт надсилає спеціальний запит із заголовками, а сервер відповідає статусом 101 (Switching Protocols) для узгодження переходу на протокол WebSocket.

1. **Процес рукопожаття.**

Приклад реквеста:

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
```

Респонс:

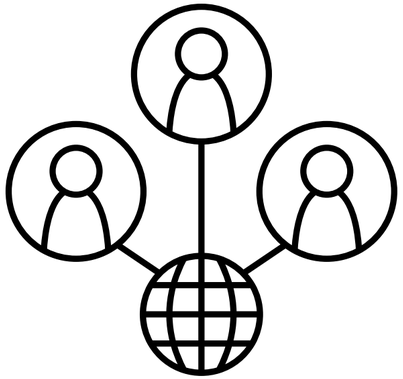
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
```

2. **Формат даних.**

Приклад:

{"message": "Hello, Web Socket!"}

3. Встановлення з’єднання.

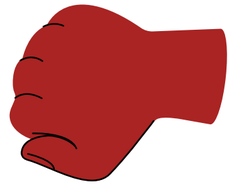


Приклад закриття WebSocket з’єднання:

```
js
socket.close(1000, "робота закінчена"); // ініціюється закриття з кодом
1000 и причиною
socket.onclose = event => {
 console.log(event.code); // 1000
 console.log(event.reason); // "робота закінчена"
};
```

Переваги і недоліки WebSocket

- Двосторонній зв'язок у реальному часі
- Мінімальна затримка.
- Кросплатформність.
- Ефективність.
- Масштабованість.



- Підтримка браузерів.
- Складність масштабування.
- Проблеми з SEO.
- Безпека.

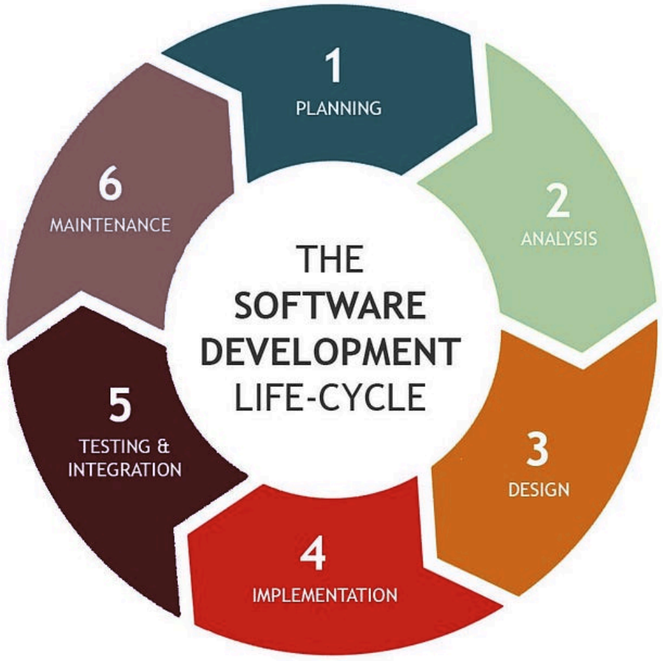
Все працювало ідеально… поки не прийшли користувачі

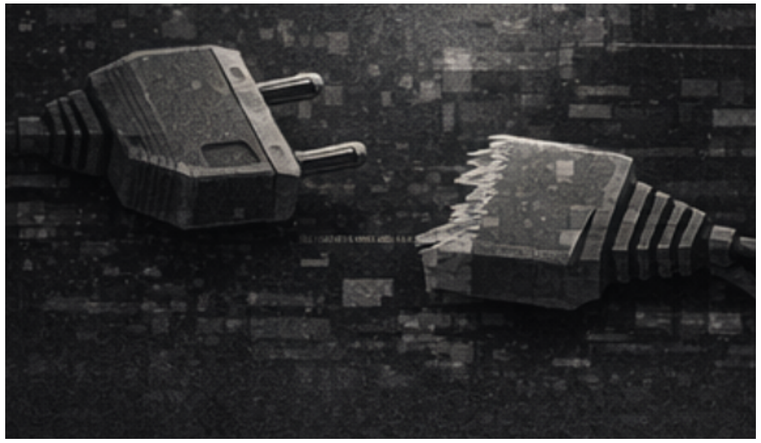
API

- 1000+ запитів за секунду
- неочікувані payload’и
- збої під навантаженням

WebSocket

- обриви з’єднання
- гонки повідомлень
- розсинхронізація





Користувач

- клікає швидше, ніж ти думав
- робить “неможливі” дії
- ламає все

Емуляція API endpoint’ів для розробки та тестування

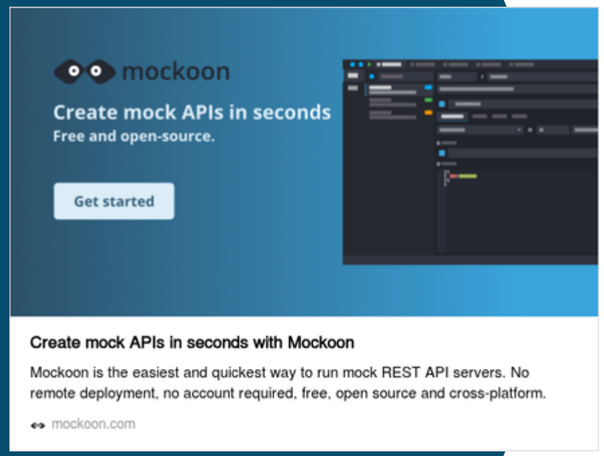
Дозволяє емулювати відповіді API без реального backend’у

Використовується:

- розробниками (локальна розробка)
- QA (тестування сценаріїв)

Корисно, коли:

- backend ще не готовий
- потрібно протестувати помилки / edge-case
- потрібна стабільна передбачувана відповідь



“При переході на сторінку профілю відображаються основні дані користувача — ім’я та роль...”

```
endpoint:
 - request:
   method: GET
   path: /api/user/profile/<profile_id>
  response:
   status: 200
   headers:
    Content-Type: application/json
   body:
    id: 123
    name: "Demo User"
    role: "tester"
```

Практика WebSocket: підключення та запити

- Протестуємо замоканий endpoint
- Розглянемо типові помилки
- Автоматизуємо тестування
- Використаємо штучний інтелект для автоматизації

Практика WebSocket: підключення та запити

- Підключимось до WebSocket у реальному проєкті
- Розглянемо, як відбувається обмін повідомленнями
- Відправимо тестові запити
- Проаналізуємо відповіді сервера
- Розглянемо типові сценарії використання

https://ws-playground.netlify.app

Особливості тестування

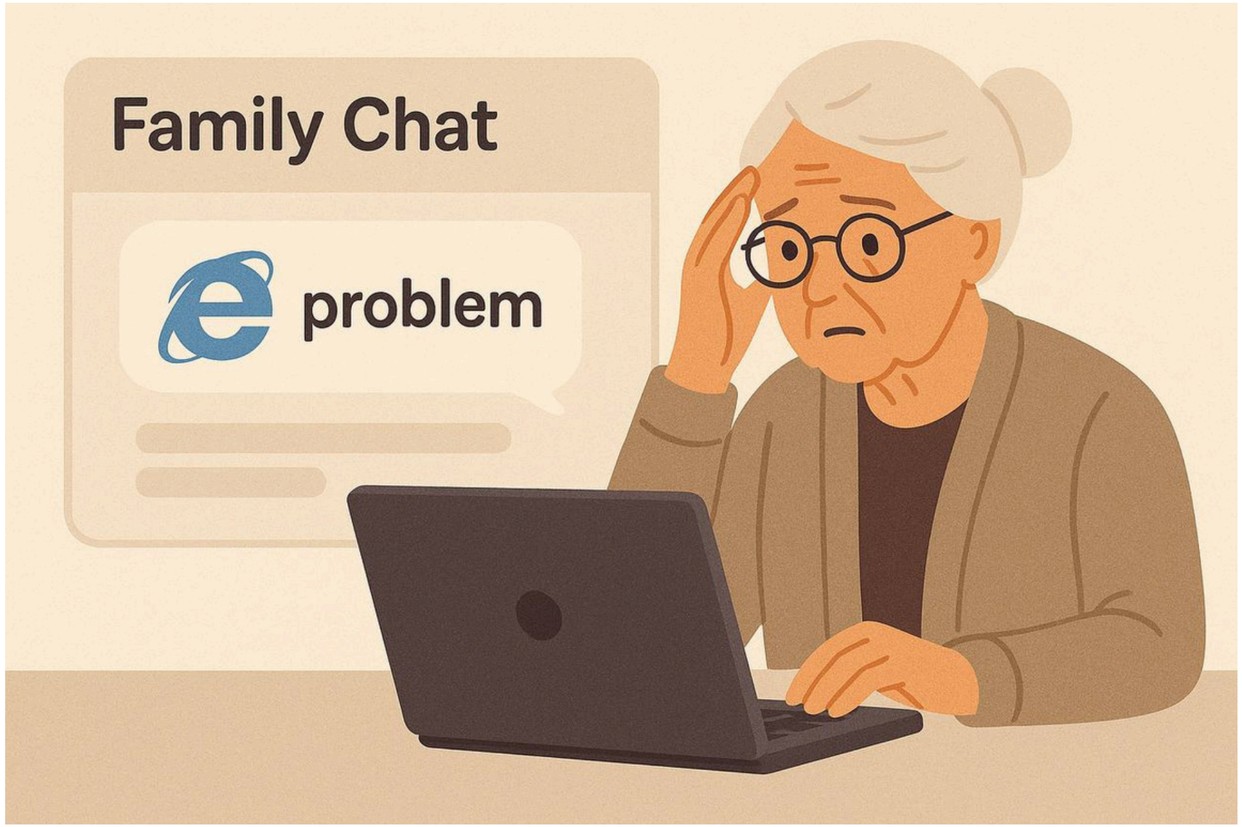
1. **Двосторонній зв'язок та швидкий обмін повідомленнями**	Повідомлення, надіслані сервером, не надходять на клієнтську сторону в реальному часі, що призводить до затримок в обміні даними.

Особливості тестування

2. **Тестування запитів і відповідей.**	Відповідь сервера неправильно форматована або не надсилається, коли клієнт надсилає запит.

Особливості тестування

3. **Проблеми з сумісністю**	Застосунок не працює у старих браузерах, таких як Internet Explorer, який не підтримує протокол WebSocket

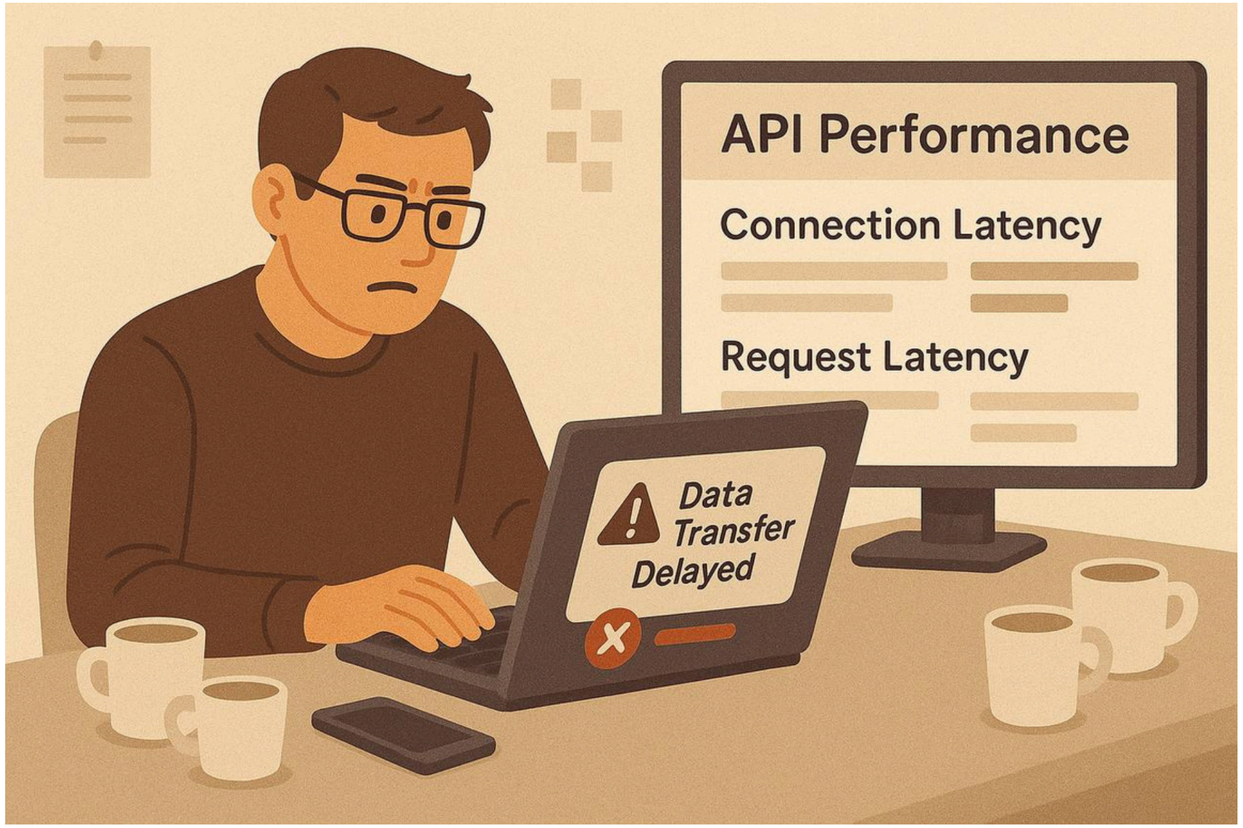


Особливості тестування

4. **Навантажувальне тестування.**	При високій кількості одночасних підключень сервер не може обробити всі запити і починає розривати з'єднання.

Особливості тестування

5. **Моніторинг метрик у реальному часі.**	Моніторинг дозволяє відстежувати стан активних з'єднань, швидкість передачі повідомлень та загальне «здоров'я» сервера безпосередньо під час роботи.

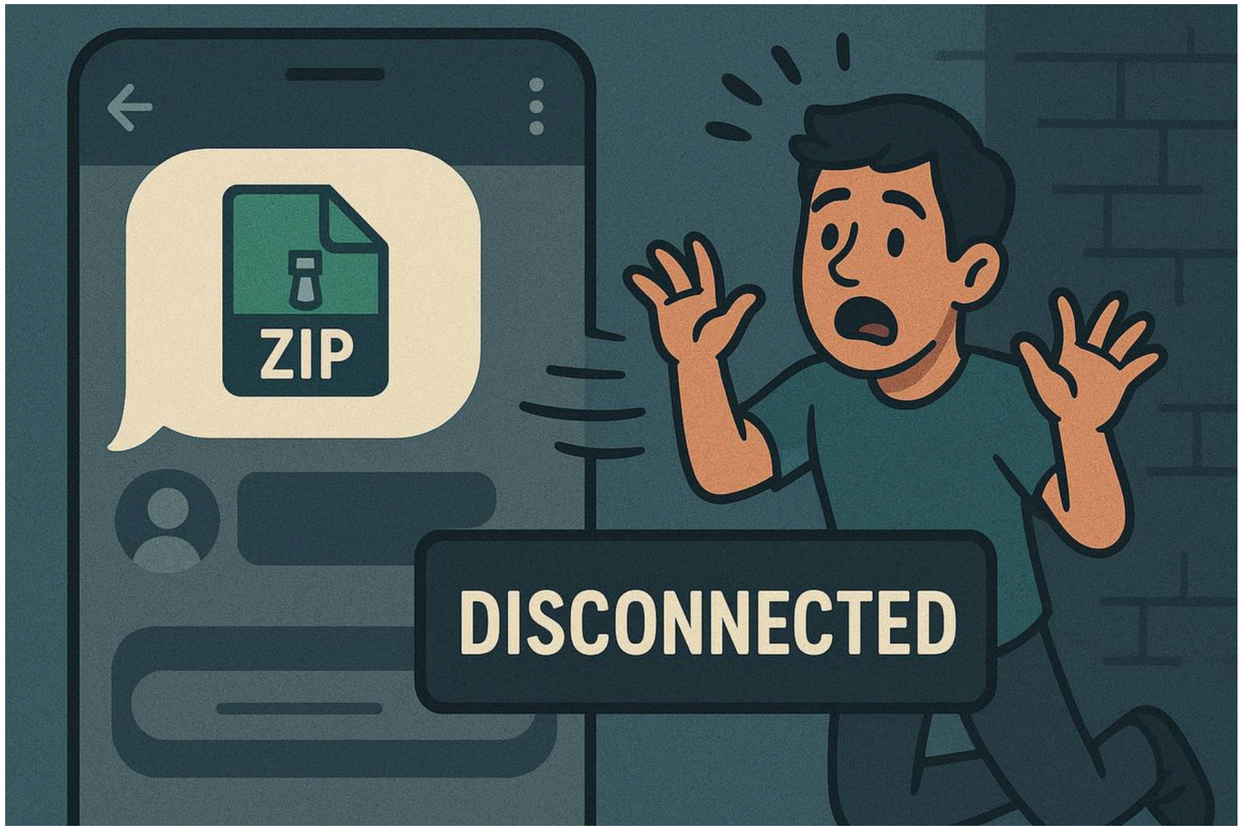


Особливості тестування

6. Перевірка безпеки	Передача даних без шифрування через незахищене з'єднання дозволяє зловмисникам перехоплювати інформацію.

Особливості тестування

7. **Обробка помилок**	Після втрати з’єднання сервер не відновлює з’єднання автоматично, і клієнт залишається в неактивному стані.



Особливості тестування

8. Відстеження часу з’єднання	З’єднання не розривається коректно після завершення сесії, що призводить до витоку ресурсів на сервері.

Інструменти тестування API and WebSocket

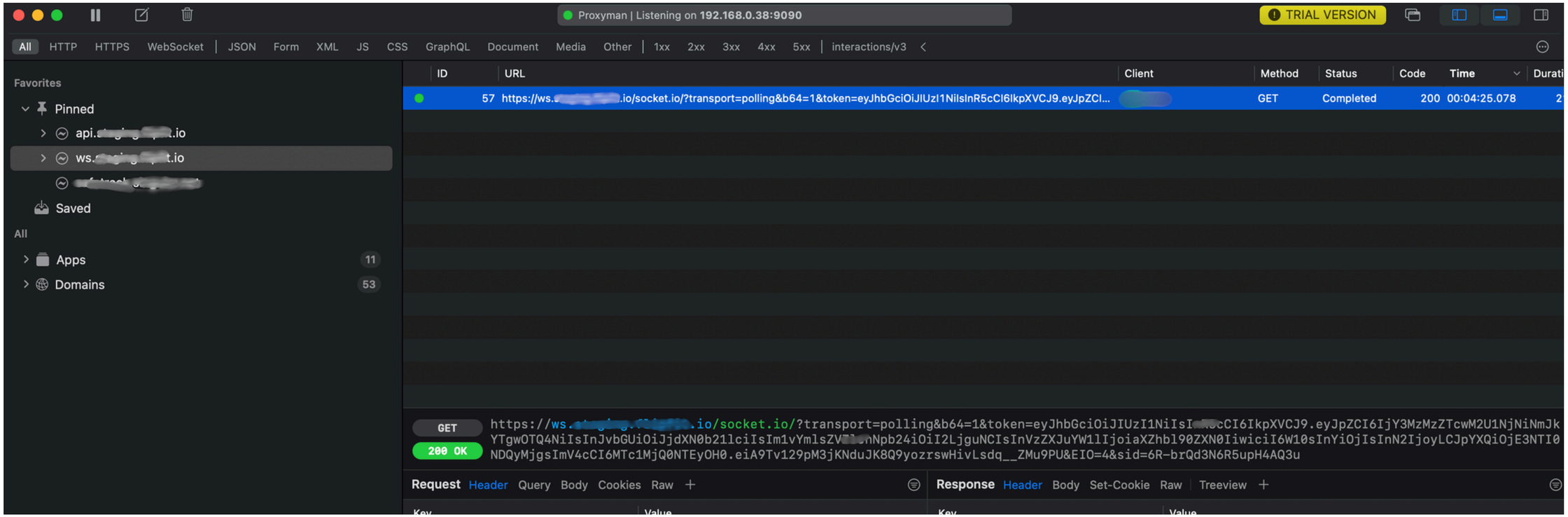
- **Postman** - це потужна платформа для тестування API, яка має повноцінну підтримку WebSocket для ручної перевірки та налагодження.

- **WebSocket King** - це максимально простий і доступний браузерний клієнт для моментального тестування WebSocket-з'єднань без встановлення зайвого ПЗ.
-

- **PieSocket** — це хмарний сервіс (WebSocket-as-a-Service), який надає готову інфраструктуру для розгортання, масштабування та захисту каналів зв'язку в реальному часі.

PIE SOCKET

- **Proxyman** — це нативний інструмент для macOS, який дозволяє легко перехоплювати, переглядати та аналізувати весь трафік (HTTP/HTTPS/WebSocket) між клієнтом і сервером.



- **Fiddler** — це кроссплатформенний інструмент для налагодження вебтрафіку, який дозволяє відстежувати, аналізувати та редагувати HTTP(S) і WebSocket трафік на будь-якій операційній системі.

- **OWASP Zed Attack Proxy (ZAP)** — це інструмент, який дозволяє тестувати безпеку застосунку, навіть якщо у вас немає спеціального бекграунду в кібербезпеці.

Мануал по тестуванню із Zap — https://www.zaproxy.org/getting-started/

Безкоштовні короткі уроки — https://www.zaproxy.org/zap-in-ten/

Додаткові інструменти:

- **Apache JMeter** — ідеально підходить для навантажувального тестування WebSocket. Дозволяє симулювати тисячі одночасних підключень.
- **Wireshark** — потужний інструмент для аналізу мережевого трафіку.
- **Beeceptor** — зручний веб-інструмент для налаштування мок-ендпоїнтів.

.

Що ми сьогодні розібрали

Теорія

- основи тестування API
- основи тестування WebSocket
- історія створення та роботи API і WebSocket

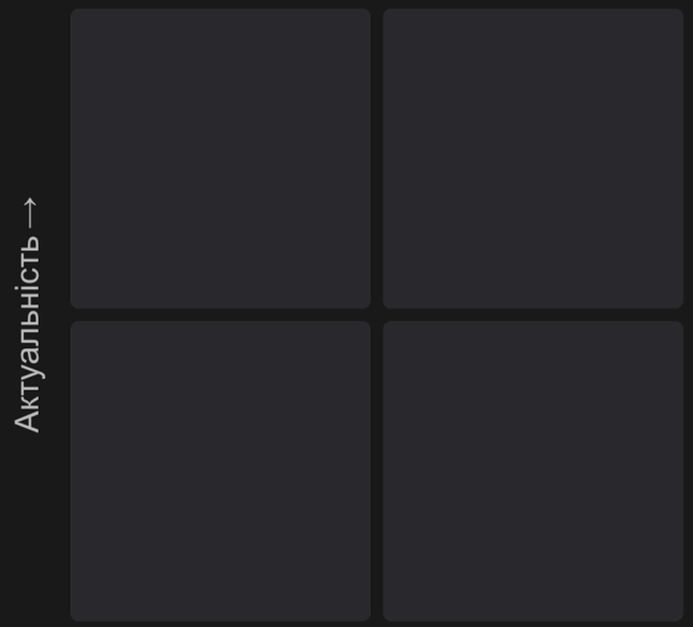
Практика

- спробували надсилати запити
- замокали endpoint та протестували його
- подивилися, як працює WebSocket у реальному часі

Результат

- протестували API та WebSocket
- розібрали типові помилки
- автоматизували тестування

Де б ви розмістили ці теми на матриці актуальність-складність?



1. Розуміння API
2. Тестування API
3. Розуміння Websocket
4. Тестування Websocket
5. AI в тестуванні API
6. AI в тестуванні Websocket
7. Перформанс тестування

Mentimeter

Задавайте свої питання!

menti.com

6903 0741

Waiting for participants

Start voting

ASIN: B0GHL4P2J8

Дякуємо за увагу!

Для запитань і співпраці:

plaza.publisher@gmail.com

Поліна Супранович | Світлана Дудченко | Савелій Уснич

ТЕСТУВАННЯ В ЕПОХУ ШІ: ЩО ЗМІНЮЄТЬСЯ?

Як штучний інтелект розмиває межі між розробкою та тестуванням

Основные навыки в пересечении ролей

Важные навыки, такие как **написание тестов** и **автоматизация**, становятся общими.

78%	фахівців вже застосовують ШІ у тестуванні
66 %	спеціалістів стурбовані майбутнім своєї професії через ШІ

PractiTest — State of Testing 2026

Якість — це не випадковість,
а результат розумних зусиль.»
– Джон Стюарт Мілль

Важливо розуміти тестування всім — розробникам і тестувальникам

- **Які тренди є важливими?**
- **Хто відповідає за якість?**
- **Яку роль відіграє ШІ у тестуванні?**