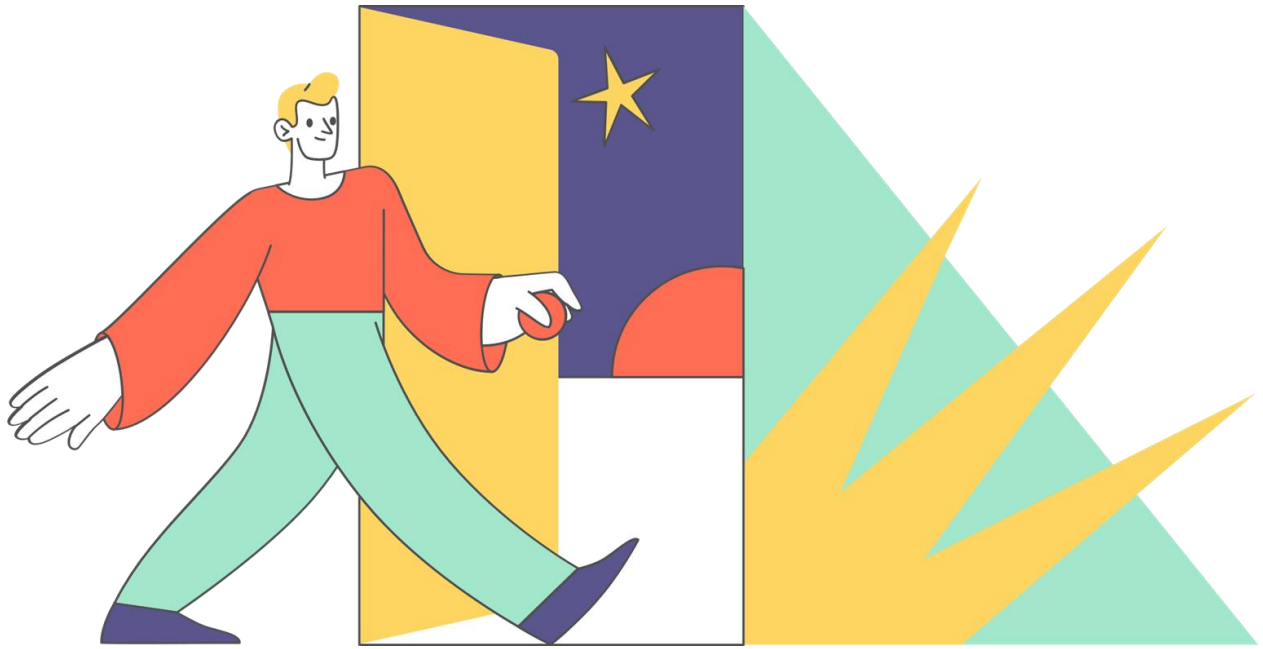


Урок 17. Тестування API. JSON. REST

Структура заняття

1. Клієнт-серверна архітектура
2. Принципи роботи AJAX
3. Протоколи передачі даних
4. REST архітектура

Архітектура клієнт-сервер
Client-server model

Один із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

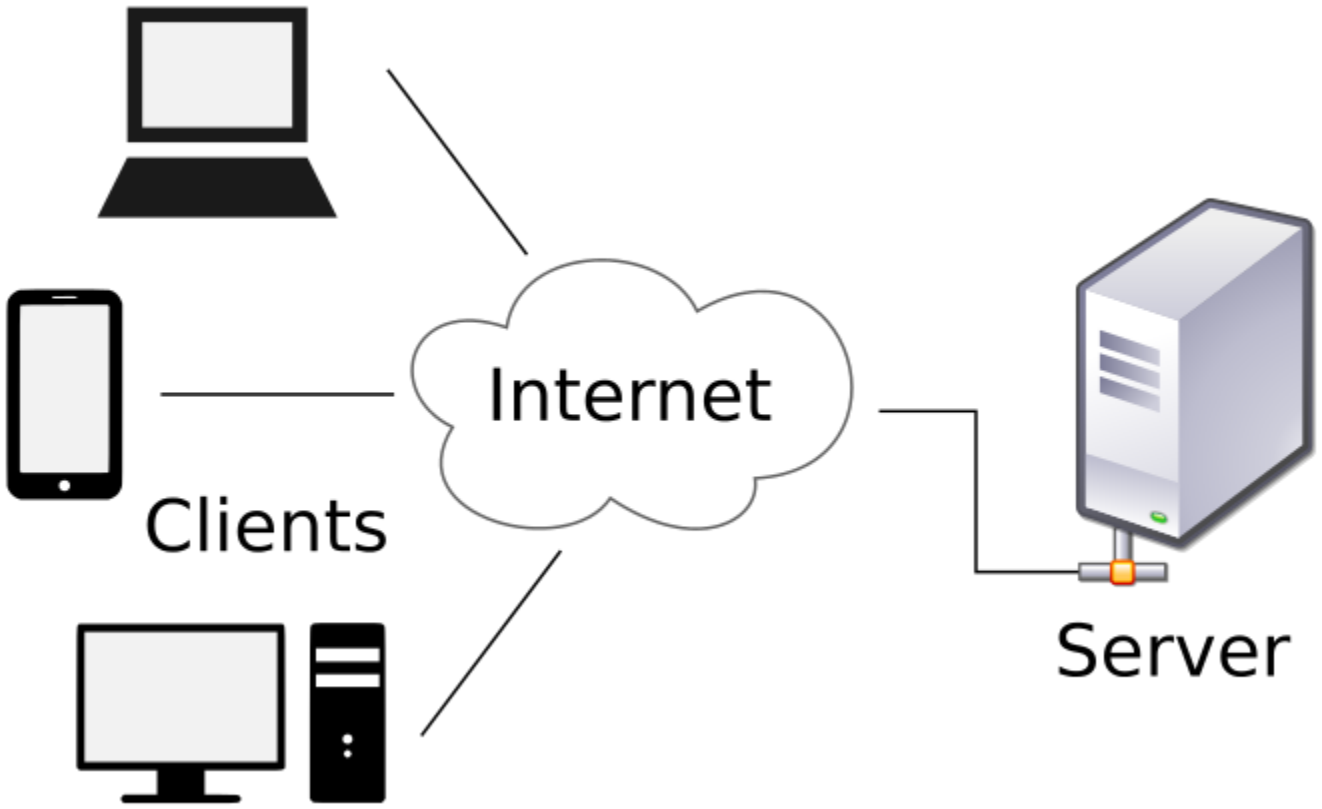
- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Дволанкова клієнт-серверна архітектура

Передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп’ютери, мобільні телефони та ін.

Дволанкова клієнт-серверна архітектура



Level 1 – FrontEnd (FE): HTML, SCC, JavaScript, Content.

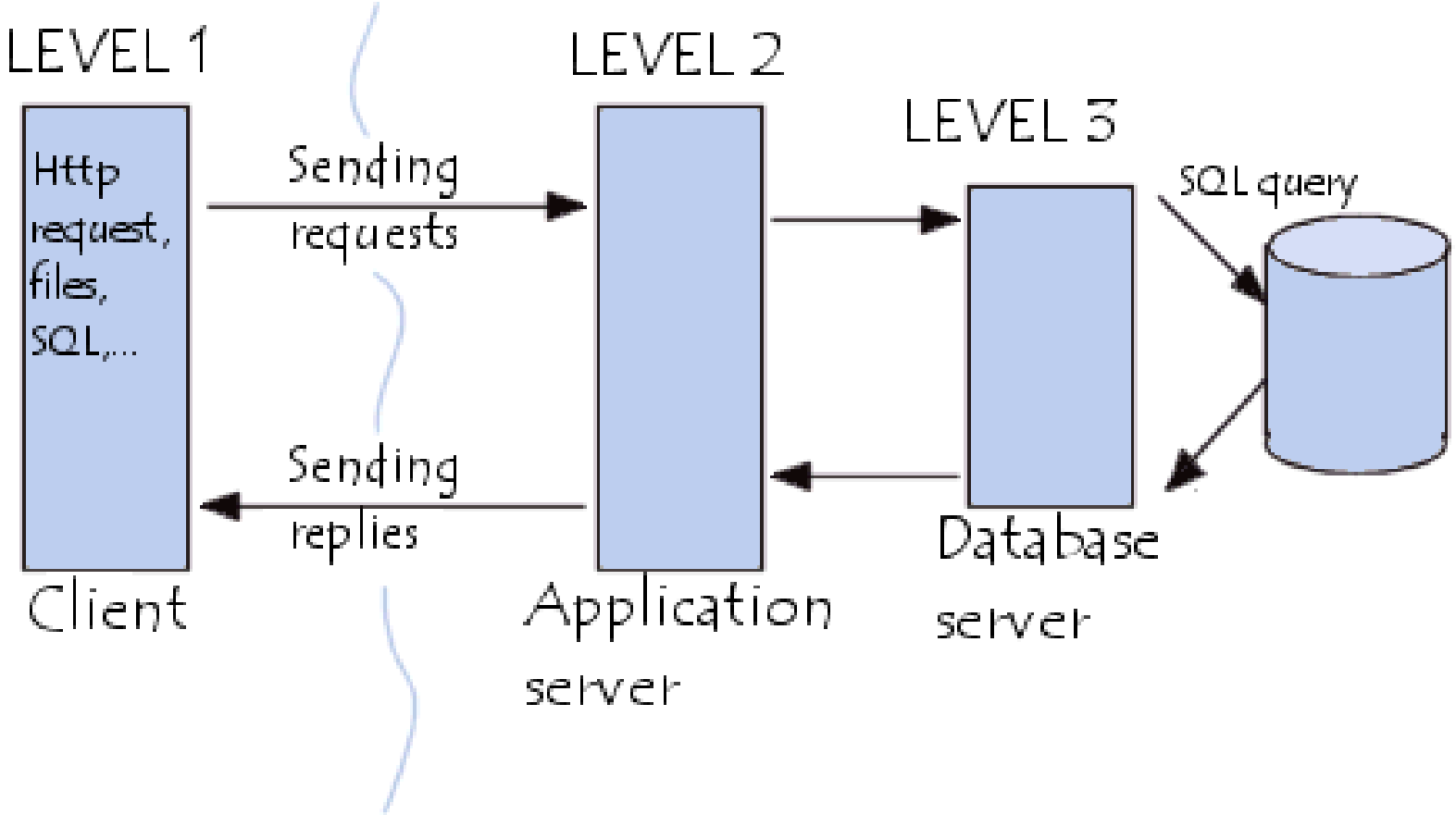
Level 2 – BackEnd (BE): Java, .Net, PHP, Pyton, Ruby

Триланкова клієнт-серверна архітектура

Передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка застосунку. Програми проміжного рівня можуть функціювати під управлінням спеціальних серверів застосунків, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

- високий ступінь гнучкості і масштабованості;
- висока безпека (тому що захист можна визначити для кожного сервісу або рівня);
- висока продуктивність (тому що завдання розподілені між серверами).

Триланкова клієнт-серверна архітектура



Level 1 – FrontEnd (FE): HTML, SCC, JavaScript, Content.

Level 2 – BackEnd (BE): Java, .Net, PHP, Pyton, Ruby

Level 3 – DataBase (DB): MySQL, Oracle, MongoDB

Ролі (функції) серверів

Веб-сервер (Web Server)

Сервер, що приймає HTTP-запити від клієнтів (веб-браузерів), видає їм HTTP-відповіді разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними.

Клієнти дістаються веб-сервера за URL-адресою потрібної їм веб-сторінки або іншого ресурсу.

Сервер застосунків (Application Server)

Сервер, що виконує деякі прикладні програми. Термін також відноситься і до програмного забезпечення, що встановлено на такому сервері і забезпечує виконання прикладного ПЗ.

Сервери баз даних (DB Server)

Сервери БД використовуються для обробки запитів користувача на мові SQL. При цьому СУБД знаходиться на сервері, до якого підключаються клієнтські додатки.

API (Application Programming Interface)

Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

API - контракт, який надає програма. «До мене можна звертатися так і так, я зобов'язуюсь робити те і це».

Контракт включає:

- саму операцію, яку ми можемо виконати,
- дані, що надходять на вхід,
- дані, що опиняються на виході (контент даних або повідомлення про помилку).

API поділяють по типам доступу

Публічні API

Розробники створили API та дають ним користуватись всім, хто цього бажає.

Приватні чи внутрішні API

Такі API розробляються для внутрішнього користування, щоб з'єднати свої програми. Доступ обмежений тільки спеціалістами, що працюють над цим API.

Партнерські API

API що пишуться для інтеграції з партнерськими програмами. Доступ обмежується спеціалістами власниками та партнерами.

AJAX (Asynchronous Javascript and XML)

Набір методів веб-розробки, які дозволяють веб-застосункам працювати асинхронно — обробляти будь-які запити до сервера у фоновому режимі.

JavaScript –керує динамічним контентом веб-сайту і дозволяє динамічно взаємодіяти з користувачем.

XML - варіант мови розмітки, що розширюється. Якщо HTML призначений для відображення даних, XML призначений для зберігання та перенесення даних.

І JavaScript, і XML працюють асинхронно в AJAX. В результаті будь-яка веб-програма, яка використовує AJAX, може надсилати та вилучати дані з сервера без необхідності перезавантаження всієї сторінки.

Порівняльна таблиця

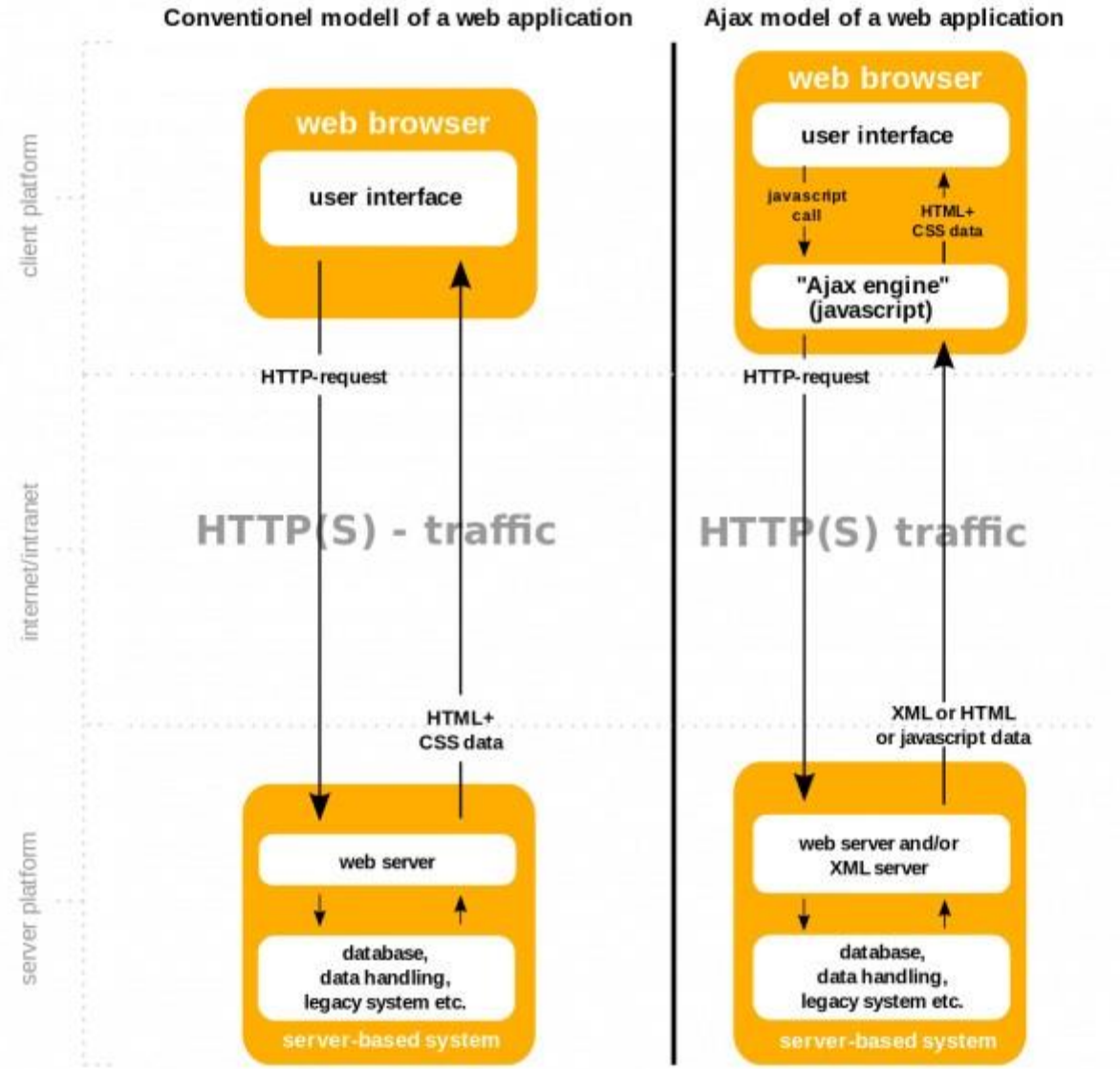
Звичайна модель

1. Запит HTTP надсилається з веб-браузера на сервер.
2. Сервер отримує та згодом витягує дані.
3. Сервер надсилає запитані дані до веб-браузера.
4. Веб-браузер отримує дані та перезавантажує сторінку для відображення даних.

Під час цього процесу користувачі не мають вибору, окрім як чекати, поки весь процес не буде завершений. Це не тільки забирає багато часу, а й створює непотрібне навантаження на сервер.

AJAX модель

1. Браузер створює виклик JavaScript, який активує XMLHttpRequest.
2. У фоновому режимі веб-браузер створює запит HTTP до сервера.
3. Сервер отримує, витягує та відправляє дані назад у веб-браузер.
4. Веб-браузер отримує запитані дані, які безпосередньо відображатимуться на сторінці. Перезавантаження не потрібне.


Conventionel modell of a web application
Ajax model of a web application
client platform
internet/intranet
server platform
web browser
user interface
HTTP-request
HTTP(S) - traffic
HTML+
CSS data
web server
database,
data handling,
legacy system etc.
server-based system
web browser
user interface
javascript
call
HTML+
CSS data
"Ajax engine"
(javascript)
HTTP-request
HTTP(S) traffic
XML or HTML
or javascript data
web server and/or
XML server
database,
data handling,
legacy system etc.
server-based system

Практика

1. **Структура запиту та його основні елементи в Developer Tools та будь-який запит в гуглі.**
2. **Принцип роботи технології AJAX – пагінація** https://hotline.ua/

API (Application Programming Interface)

API обмінюються даними та функціями, а для цього потрібні чіткі протоколи та архітектури — правила, за якими працюватиме API.

Протоколи

- **HTTP** – протокол прикладного рівня передачі, спочатку — як гіпертекстових документів у форматі HTML, нині використовується передачі довільних даних.
- **HTTPs** – розширення протоколу HTTP для підтримки шифрування з метою підвищення безпеки.
- **FTP** – протокол передачі файлів по мережі, що з'явився в 1971 задовго до HTTP і навіть до TCP/IP, завдяки чому є одним з найстаріших прикладних протоколів
- **SMTP** – це широко використовуваний мережевий протокол, призначений передачі електронної пошти у мережах TCP/IP.

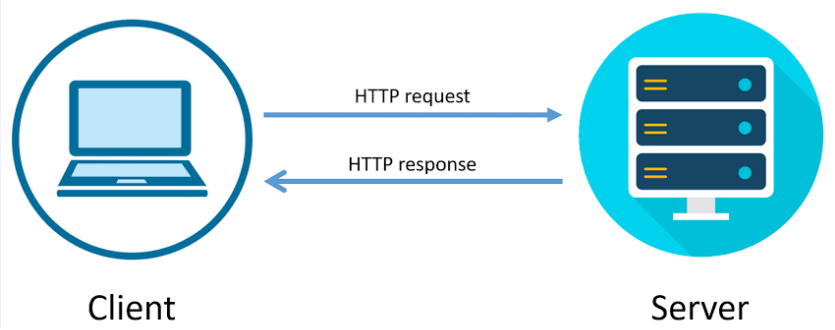
HTTP (HyperText Transfer Protocol)

Протокол передачі гіпертекстових документів в форматі HTML.

Обмін повідомленнями йде за звичайною схемою «запит-відповідь».

Основою HTTP являється технологія ‘клієнт-сервер’, тобто існує

- **Отримувач (клієнт), що ініціює з’єднання та відправку запиту**
- **Постачальник (сервер), що ініціює з’єднання для отримання запиту, проводять необхідні маніпуляції та відправляють відповідь з необхідним результатом.**



HTTP request

Method

Визначає операцію, яку необхідно здійснити з вказаним ресурсом

Request URI

Уніфікований ідентифікатор ресурсу. Шлях до конкретного ресурсу, над яким необхідно здійснити операцію.

Protocol version

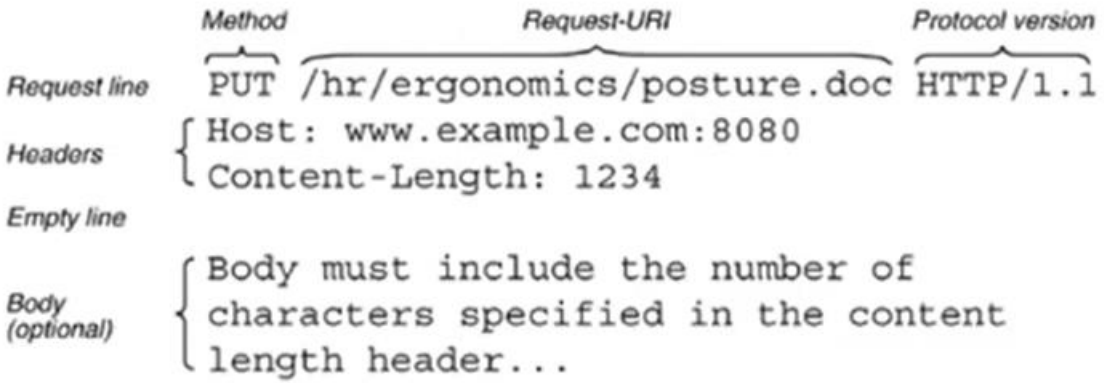
Визначає версію стандарту HTTP

Header

Основна частина HTTP-запитів і відповідей, що несуть інформацію про браузер клієнта, сторінки, сервера тощо

Body

Інформація, яку клієнт передає на ресурс. Поле не обов'язкове.



HTTP methods

GET

Запитує вміст вказаного ресурсу

POST

Передає призначені для користувача дані (наприклад, з HTML-форми) заданому ресурсу. При цьому передані дані включаються в Тіло запиту (Request body)

PUT

Завантажує вказаний ресурс на сервер.

PATCH

Завантажує певну частину ресурсу на сервер.

DELETE

Видаляє вказаний ресурс.

HTTP status code

1xx — інформаційний: запит прийнятий, продовжуй процес.

2xx — успіх: дія була успішно передана, зрозуміла, та прийнята.

3xx — перенаправлення: наступні дії мають бути успішно виконані для реалізації запиту.

4xx — помилка клієнта: запит містить синтаксичні помилки або не може бути виконаний.

5xx — помилка сервера: сервер не зміг виконати правильно сформований запит.

Найбільш поширені статуси:

200 OK — запит виконано успішно.

301 Moved Permanently — ресурс переміщено.

403 Forbidden — доступ до запитаного ресурсу заборонений.

404 Not Found — ресурс не знайдений.

503 Service Unavailable — сервіс недоступний.

Практика

За допомогою онлайн ресурсу https://reqbin.com/ побудувати запити POST та GET до наступного API: GO REST (/users endpoint).

Тестування API

Тестування спрямоване на перевірку функціонування перш за все бізнес логіки додатку.

Також потрібно враховувати, що API створюються багато в чому для інтеграції з іншими сервісами. І працюють з ними не люди, а інші програмні системи. Тому потрібно оцінювати API з позиції зручності його використання разом з іншими продуктами, з позиції легкої інтеграції з ними. Кожен API повинен бути гнучким, також мати зрозумілу і детальну документацію.

Переваги тестування API

1. Тестування основної функціональності програми на ранніх етапах створення додатку, ще до написання графічної оболонки. Це допомагає розкрити дрібні проблеми, до того, як вони стануть більшими.
2. Часова ефективність: API Testing потребує менше часу, ніж функціональне тестування графічного інтерфейсу (GUI testing); надає швидший зворотній зв'язок. Можна провести 30000 автоматизованих тестів API приблизно за 9 годин і стільки ж тестів графічного інтерфейсу за 50 годин. А це значно економить кошти.
3. Незалежність від мови програмування:
    - Модель передачі при проведенні тестування API не залежить від мови. Дані обмінюються за допомогою JSON або XML. Можна вибрати будь-яку мову під час автоматичного тестування.
    - Проста інтеграція з GUI: тести API можуть бути легко інтегровані, що особливо корисно, якщо треба виконувати функціональні тести графічного інтерфейсу користувача після тестування API.
    - Можливість виконувати тести на великих об'ємах вхідних даних. Тестування API (API Testing) допомагає зменшити різноманітні ризики.
    - Дуже ефективно перевіряє всі функціональні складові тестованої системи.

Типові помилки в API

- Збій обробки помилкових умов;
- Невикористані flag
- Відсутній або дублюється функціонал;
- Питання налаштування: труднощі при підключенні і отриманні відповіді від API;
- Проблеми з безпекою API;
- Питання по багатопоточності;
- Проблеми з продуктивністю: буває час відгуку API дуже високий;
- Помилки;
- Некоректна обробка валідних значень;
- Дані відповіді некоректно структуровані (JSON або XML).

Основні перевірки в API

- порожній запит;
- запит з невірним форматом даних, що передаються;
- запит з невірними параметрами;
- неправильне значення value для правильного параметра (порожнє, неправильні дані, неправильний формат даних, довжина);
- неправильне значення key (параметр);
- додаткова пара key:value;
- неправильні Headers (заголовки);
- неправильна авторизація (Headers)

Використання технік тестування

- **Оглядове дослідницьке тестування** — тести повинні виконати набір викликів, задекларованих в API, щоб перевірити загальну працездатність системи;
- **Перевірка документації** — перевіряється повнота описів функцій API, її зрозумілість і, в свою чергу, є фінальним результатом.
- **Аналіз граничних значень** — в API запитах в явному вигляді можуть передаватися значення параметрів. Це відмінний привід виділити кордону вхідних і вихідних значень і перевірити їх.
- **Розбиття на класи еквівалентності** — навіть у невеликого API є безліч варіантів використання і безліч комбінацій вхідних і вихідних змінних. Тому ми можемо зайвий раз використовувати наші навички виділення еквівалентних класів.
- **Юзабіліті-тестування** — перевіряє, чи є API функціональним і володіє зручним інтерфейсом, також перевіряється інтеграція з іншими;
- **Тестування безпеки** — перевіряє використовуваний тип аутентифікації і шифрування даних за допомогою HTTP;
- **Автоматизоване тестування** — створення скриптів, програм або настройка додатків, які зможуть тестувати API на регулярній основі

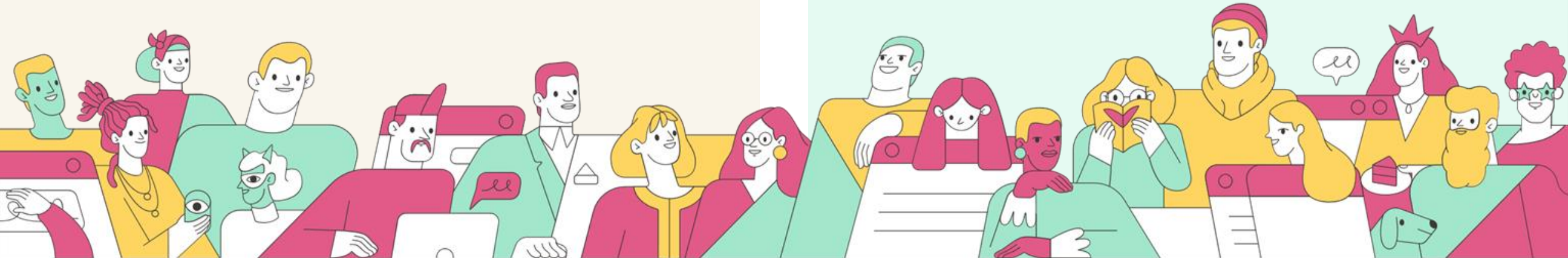
JSON vs XML

JSON (JavaScript Object Notation)

Текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передавання структурованої інформації через мережу.

XML (eXtensible Markup Language)

Стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. XML-документ складається із текстових знаків, і придатний до читання людиною.



JSON vs XML

JSON (JavaScript Object Notation)

JSON будується на двох структурах:

- Набір пар назва/значення. У різних мовах програмування це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список із ключем або асоціативним масивом.

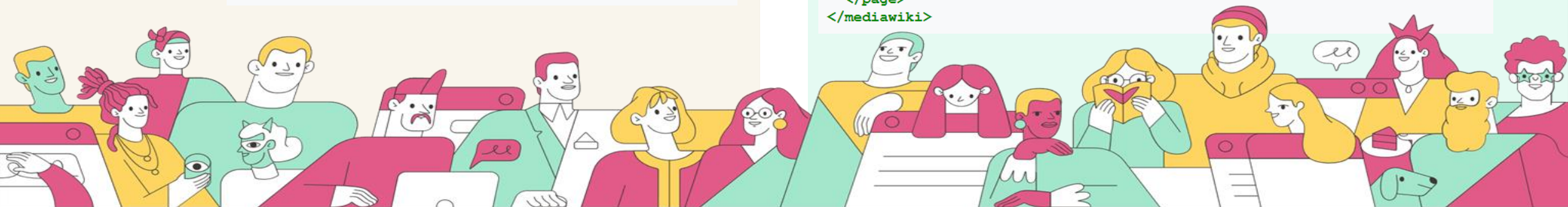
- Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність.

```
{
   "firstName": "Іван",
   "lastName": "Коваленко",
   "address": {
       "streetAddress": "вул. Грушевського 14, кв.101",
       "city": "Київ",
       "postalCode": 21000
   },
```

XML (eXtensible Markup Language)

Стандарт визначає набір базових лексичних та синтаксичних правил для побудови мови описання інформації шляхом застосування простих тегів.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.3/" xml:lang="uk">
  <page>
    <title>Фукідід</title>
    <id>1529</id>
    <revision>
      <id>4382</id>
      <timestamp>2006-09-18T22:11:53Z</timestamp>
      <minor />
      <comment>Interwiki</comment>
      <text xml:space="preserve">{{Wikipedia}}
      *Історія — це філософія в прикладах.
      </text>
    </revision>
  </page>
</mediawiki>
```



REST API

Representational state transfer API (передача стану уявлення) - це архітектурний стиль проектування API з використанням протоколу HTTP. Головна перевага REST – велика гнучкість. Застосовується скрізь, де є необхідність надання даних із сервера користувачеві веб-програми або сайту.

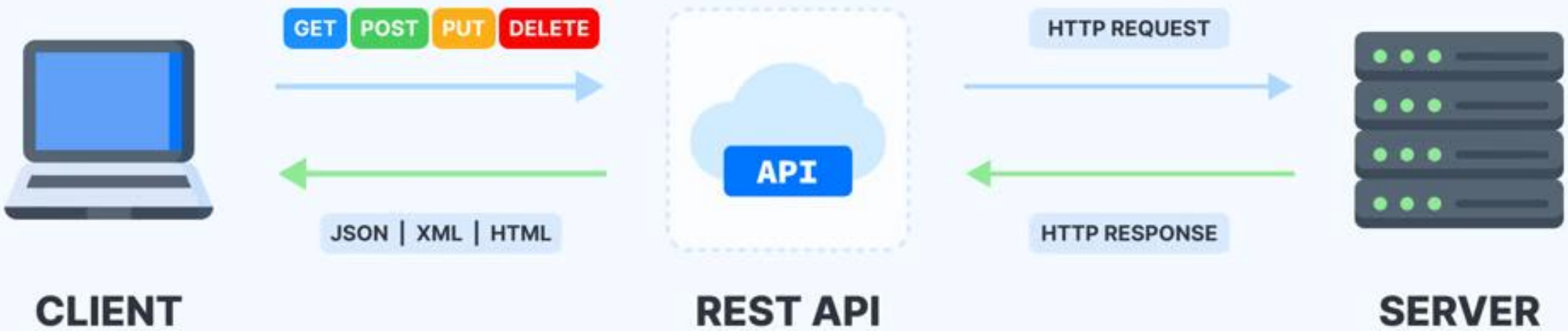
Головними компонентами REST API є:

Client - клієнт або програма, яка запущена на стороні користувача (на його девайсі) та ініціює комунікацію.

Server — сервер, який надає API як доступ до своїх даних та функцій.

Resource — ресурс є будь-яким видом контенту (відео, текст, картинка), який сервер може передати клієнту

REST API Model



Принципи REST API

1. Client-server

Програма REST має клієнт-серверну архітектуру. Клієнтом є той, хто надсилає запити на ресурси і ніяк не пов'язаний із сховищем даних. Зберігання даних залишається всередині сервера. Сервери ж не пов'язуються з інтерфейсом користувача. Іншими словами, клієнт та сервер незалежні один від одного і можуть розвиватися окремо, що робить REST API більш гнучким та масштабованим.

Принципи REST API

2. Uniform Interface

Передбачає наявність єдиного способу взаємодії з сервером незалежно від типу пристрою або програми.

- Identification of resources. Кожен ресурс REST має ідентифікатор, який не залежить від стану ресурсу. У ролі ідентифікатора виступає URL-адреса.
- Manipulation of resources through representations (маніпулювання ресурсами через уявлення). Клієнт має уявлення ресурсу, що містить дані його видалення чи зміни. Клієнт відправляє подання на сервер (об'єкт JSON), який хоче змінити, видалити чи додати.
- Self-descriptive messages (самодокументовані повідомлення) - кожне повідомлення має достатню інформацію для того, щоб сервер легко проаналізував запит.
- Hypermedia as the engine of application state (гіпермедіа як двигун стану програми)- використання посилань для кожної відповіді, щоб клієнт міг знайти інші ресурси. Таким чином, всі взаємодії в REST проходять через гіпермедіа.

Принципи REST API

3. Stateless (відсутність стану)

Сервер не містить жодної інформації про клієнта. Кожен запит включає всю необхідну інформацію для обробки. Інформація про сесію повністю зберігається за клієнта.

Принципи REST API

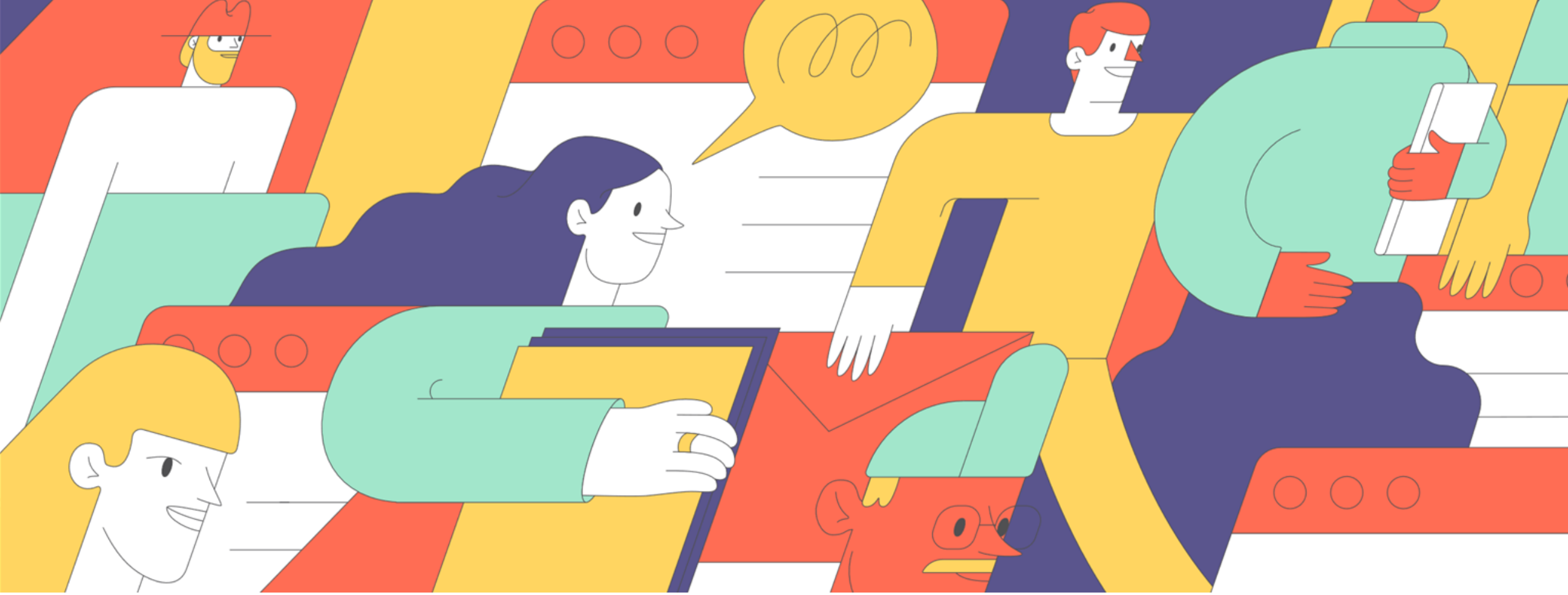
4. Cacheable (кешування)

Кожна відповідь повинна містити дані про те, чи вона кешується чи ні і протягом якого часу відповідь може бути кешована на стороні клієнта. Якщо відповідь може бути кешована, то в наступних схожих запитах клієнт може використовувати ті ж дані без повторного звернення на сервер. При правильному використанні це покращує продуктивність та доступність.

Принципи REST API

5. Layered system (багаторівнева система)

У REST використовується багаторівнева система - ієрархія шарів, що створює певні обмеження на поведінку компонентів. У багаторівневій системі компоненти можуть бачити тільки ті компоненти, які розташовані на найближчих рівнях і з якими вони взаємодіють.



Домашнє завдання