

*Тестування WEB додатків*

## Вступ. Основні поняття та терміни

**Тестування програмного забезпечення** — процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності

**Ручне тестування (Manual testing)** — це процес ручної перевірки програмного забезпечення на помилки

**Автоматизоване тестування ПЗ** — частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення

**Техніка випробування «чорної скриньки»** - техніка тестування для виведення тестових кейсів на основі функціональних можливостей програми та не враховуючи внутрішню структуру системи

**Переваги:**

- Необов'язковість технічного досвіду
- Незалежність тестувальника від розробника
- Ефективність при тестуванні великих та складних додатків
- Виявлення дефектів та невідповідностей на ранній стадії тестування.

**Недоліки:**

- Існує ймовірність ігнорування можливих умов сценарію
- Пропуск всіх можливих входів та тестування їх результатів
- Неможливе повне випробування для великих та складних проектів

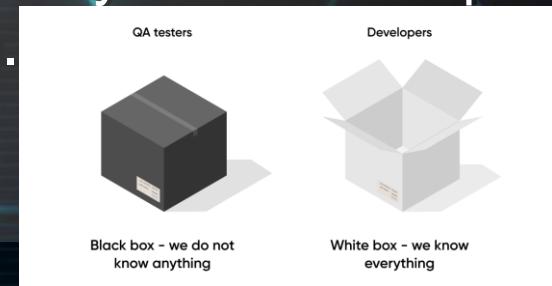
**Техніка тестування «білої скриньки»** - процедура виведення та/або відбору тестових випадків на основі аналізу внутрішньої структури компонента або системи.

### **Переваги:**

Це дозволяє видалити зайві рядки коду, що може привести до прихованих дефектів.

### **Недоліки:**

Автоматизовані тестові випадки можуть стати марними, якщо база коду швидко змінюється.



**Тестування методом «сірої скриньки»** – метод тестування програмного забезпечення, який передбачає комбінацію White Box і Black Box підходів.

**Переваги:**

- Об'єднує переваги тестування «чорної скриньки» та «білої скриньки»
- Покращує загальну якість продукції
- Зменшення витрат на тестування різних типів
- Звільняє більше часу для виправлення дефектів

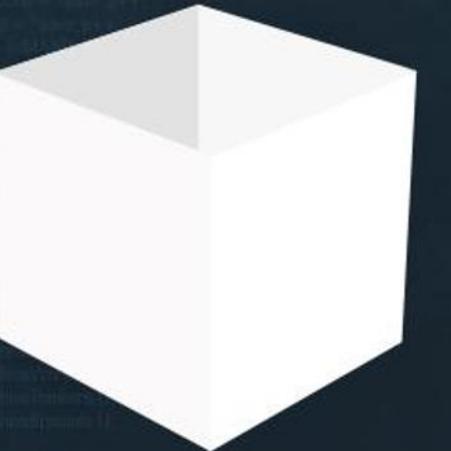
**Недоліки:**

- Часте переривання поточних операцій
- Коли тест виконується повністю, але зміст результату є неправильним

## Black box Testing



## White box Testing



## Gray box Testing



+

=

# Testing Techniques To Stay Bug-Free

# Основні означення

**Автоматизоване тестування пз** — частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення.

**Баг** - помилка, вада або дефект в комп'ютерній програмі або системі, що викликає в ній неправильний або неочікуваний результат чи поведінку.

**Верифікація** — це процес оцінки системи або її компонентів з метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу.

**Валідація** — це визначення відповідності розробленого програмного забезпечення очікуванням і потребам користувача, вимогам до системи.

**Програмне забезпечення** — сукупність програм системи оброблення інформації та програмних документів, необхідних для експлуатації цих програм.

**Фреймворк** — інфраструктура програмних рішень, що полегшує розробку складних систем.

**Рефакторинг** — процес редагування програмного коду, внутрішньої структури програмного забезпечення для полегшення розуміння коду та внесення подальших виправлень без зміни зовнішньої поведінки самої системи.

# Переваги та недоліки

Автоматизоване тестування — це симбіоз програмування і тестування. Тому тестувальнику обов'язково потрібно знати хоча б одну мову програмування.

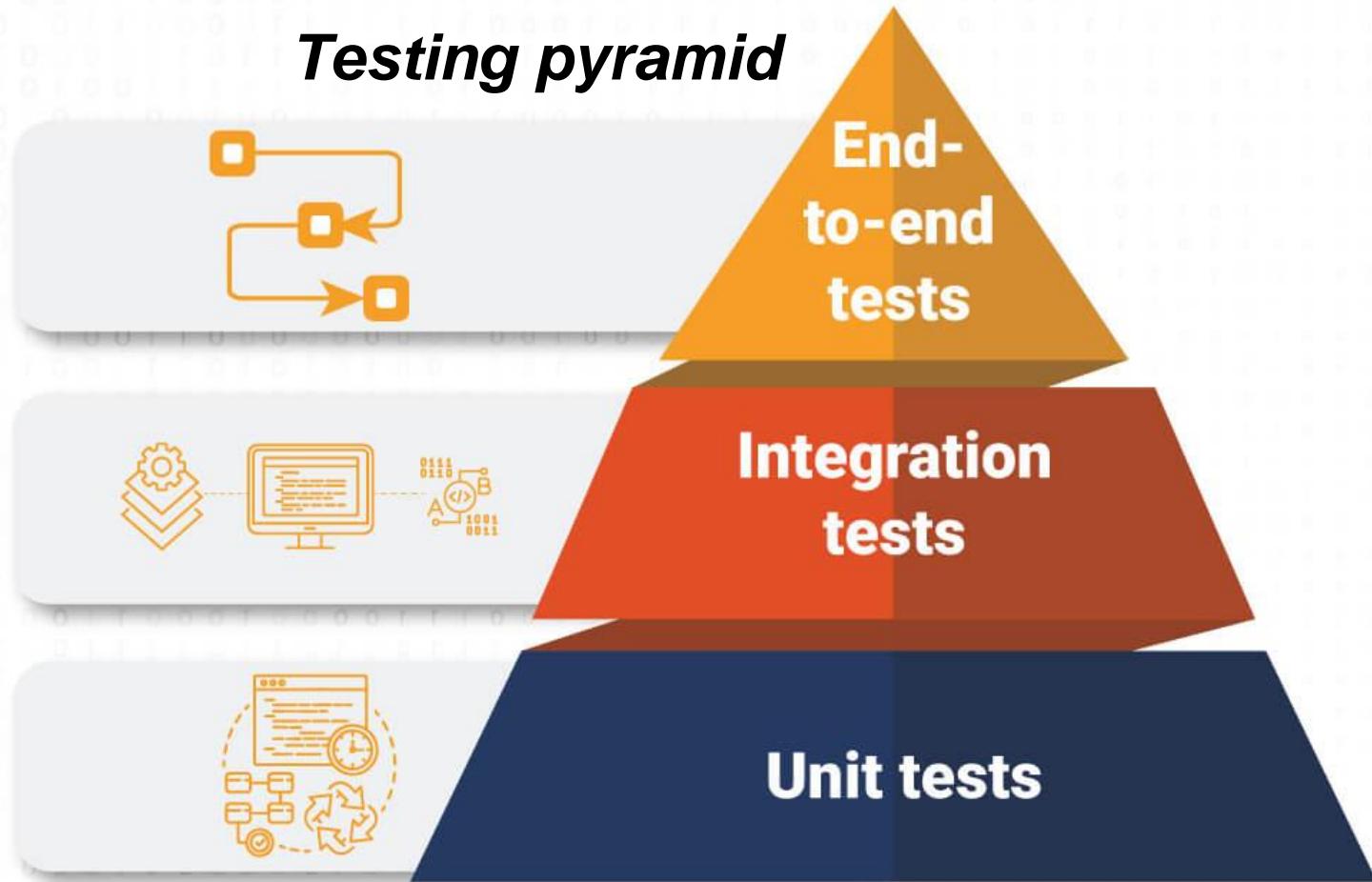
Переваги:

- Покращення якості та зменшення людського фактора
- Можливість виконувати продвинуті види тестування
- Прискорення отримання результату без втрати якості
- Виправданий результат

Недоліки:

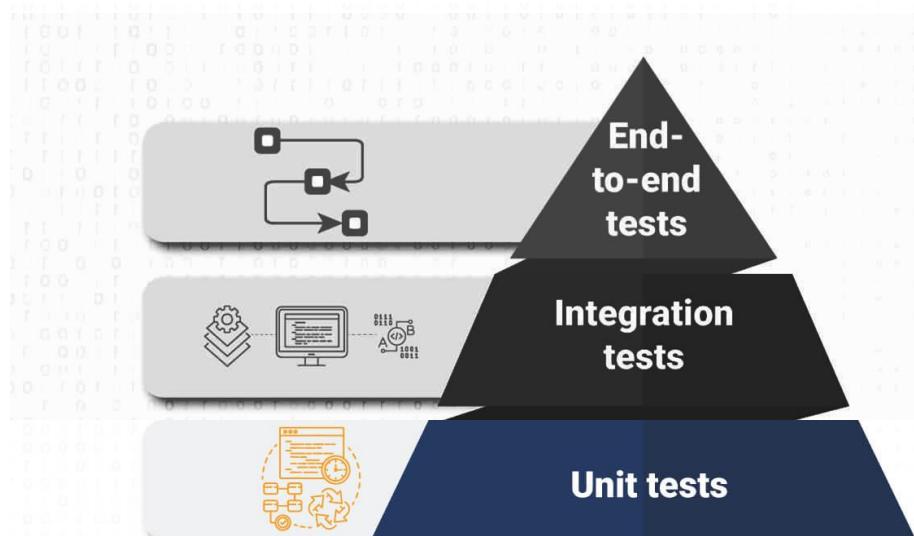
- Обмежена кількість тестових сценаріїв
- Не всі критерії підлягають тестуванню

# *Testing pyramid*

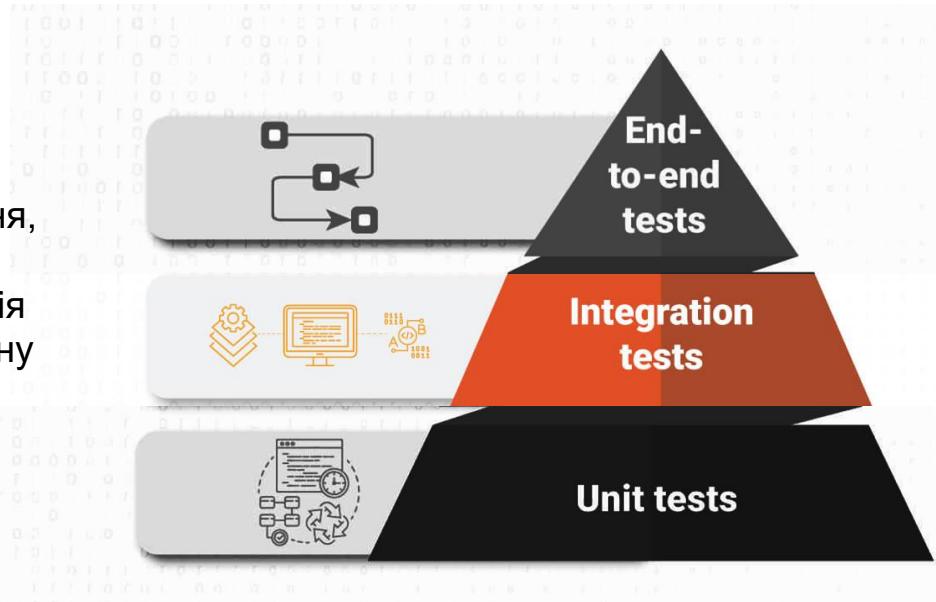


***Unit test*** - це програма, яка перевіряє роботу невеликої частини коду.

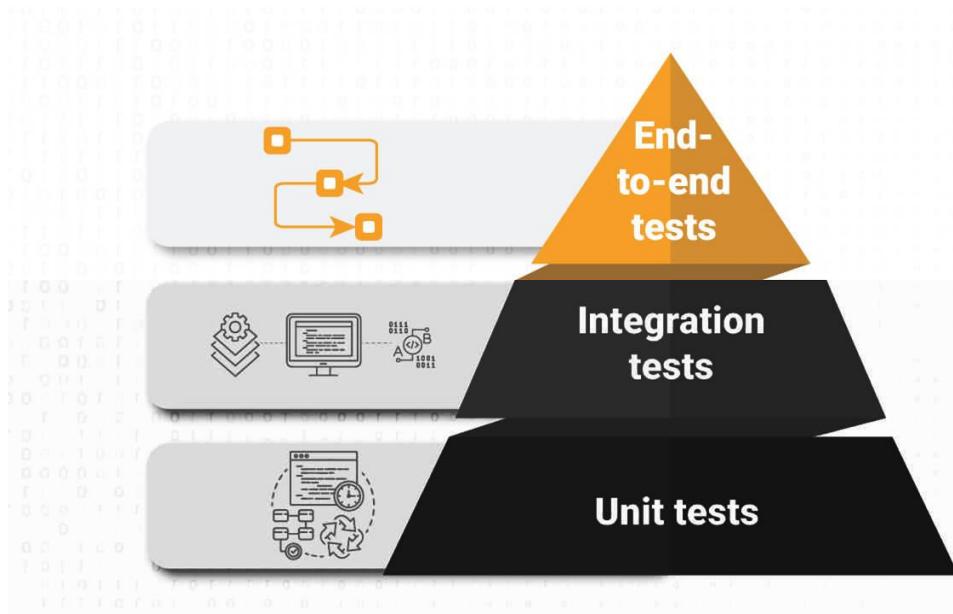
Розробники регулярно оновлюють сайти і додатки, додають новий функціонал, рефакторять код и вносят зміни, а потім перевіряють, як все працює



**Інтеграційне тестування** – вид тестування, при якому на відповідність вимог перевіряється інтеграція модулів, їх взаємодія між собою, а також інтеграція підсистем в одну загальну систему.



***End-to-end*** - це такі тести, метою яких є імітація реального сценарію користувача та перевірка системи, що перевіряється, та її компонентів для інтеграції та цілісності даних



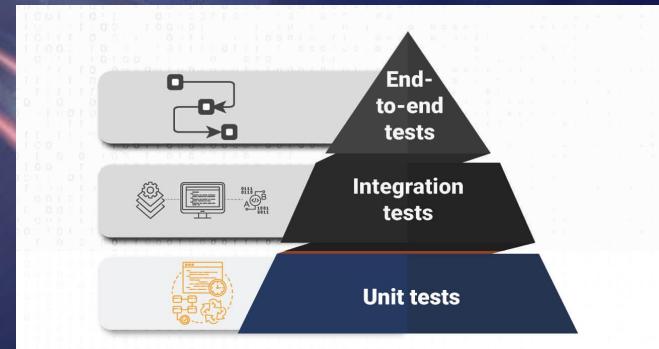
# Unit testing

**Модульне тестування** - це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду в програмі. Модулем ми можемо назвати найменшу частину програми , яка може бути протестована (функції, компоненти, модулі і т.д).

Такий вид тестування використовується з метою ізоляції кожного окремого модуля програми з подальшою перевіркою на коректність.

## Переваги:

- Легкий рефакторинг у подальшій розробці
- Додаткова документація
- Спрощення інтеграційного тестування



# Контракт модуля

Контракт - це короткий опис того, як модуль має працювати

```
generateRandomNumber(minNumber, maxNumber){  
    return Math.random() * (maxNumber - minNumber)  
}
```

Контракт:

→ Приймає 2 значення і використовує  
→ їх для генерування випадкового  
числа в цьому діапазоні

# Jest

Jest - це javaScript фреймворк для написання модульних та інтеграційних тестів. Написаний він був на Jasmine і підтримується компанією Meta.

## Початок роботи з Jest

1. Встановити за допомогою команди `npm install --save-dev jest`
2. Додати Jest в package.json `"scripts": { "test": "jest" }`
3. Створити файл з розширення ‘**назва нашого файлу**.test.js’

# Основні функції Jest

## функція для групування тестів

```
describe("назва блоку тестів", ()=> {
    ....
})
```

## функція в якій ми пишемо тест для нашого модуля

```
test("назва тесту", ()=>{
    ....
})
```

## функція яка повертає очікуваний результат

```
expect( «вираз для перевірки» )
.toBe( «очікуване значення» )
```

```
describe('sum module', () =>{
    Run | Debug
    test('should call func 1 time', () => {
        })
    Run | Debug
    test('should add two numbers', () =>{ ... })
})
test('should call func 1 time', () => {
    const randomMock = jest.fn().mockImplementation(() => 5)
    const wrapper = shallowMount(NumberGenerator)
    wrapper.setData({ minNumber: 5 })
    wrapper.setData({ maxNumber: 10 })
    expect(randomMock).toHaveBeenCalled(1)
})
```

```
expect(sum(1, 2)).toBe(3);
```

# Приклад написання unit тесту

sum.js

```
const sum = (a, b) => {
    return a + b
}
module.exports = sum;
```

sum.test.js

```
const sum = require('./sum')
Run | Debug
describe('sum module', () => {
    Run | Debug
    test('adds 1 + 2 to equal 3', () => {
        expect(sum(1, 2)).toBe(3);
    });
});
```

# ІНТЕГРАЦІЙНІ ТЕСТИ

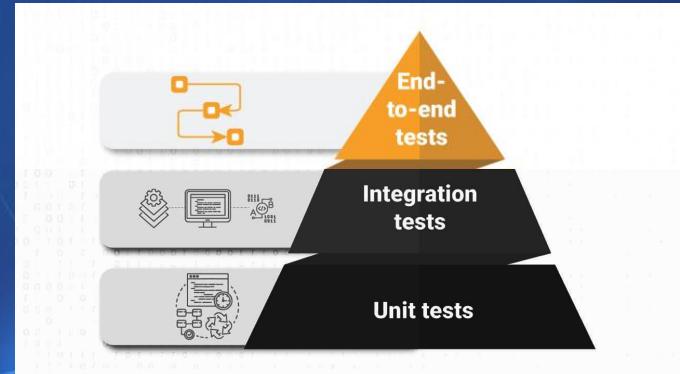
Це фаза тестування програмного забезпечення, під час якої окремі модулі програми комбінуються та тестиються разом, у взаємодії.

- перевірка інтерфейсу взаємодії між окремими модулями програми;
- контроль взаємозв'язків тестованого комплексу із сторонніми програмними рішеннями;
- тестування роботи зовнішніх компонентів рішення;
- контроль відповідності документації за проектом у частині взаємодії окремих модулів.

# End-to-end тести

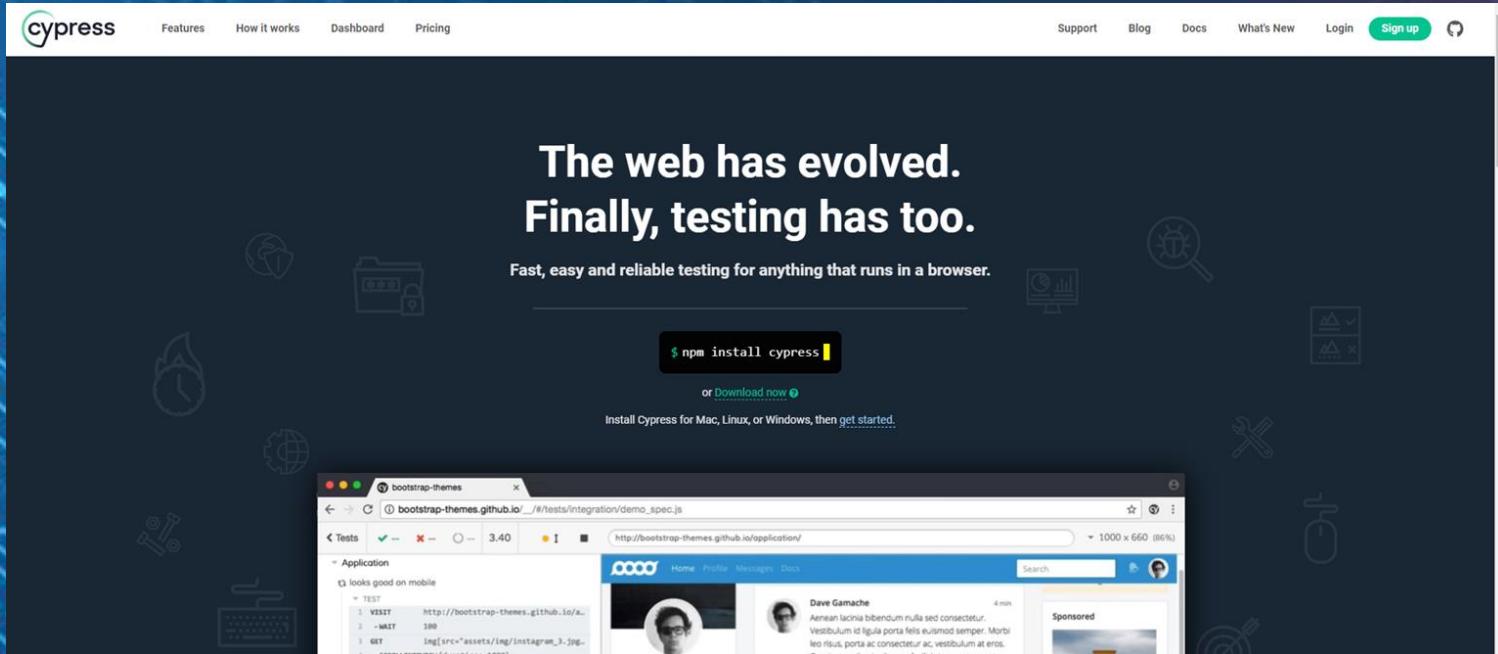
End-to-end тест - це фінальний тест проекта, у якому перевіряють весь функціонал. У цьому тесті проводиться відтворення роботи користувача з програмою і чи все правильно працює.

Цей тест є досить складним та затратним, тому його не завжди проводять. Так на малих проектах іноді пропускають цей тест, бо його можуть замінити коректні unit-тести та інтегровані тести.



# Cypress

Cypress - це фреймворк який дозволяє легко та зручно проводити E2E тести.



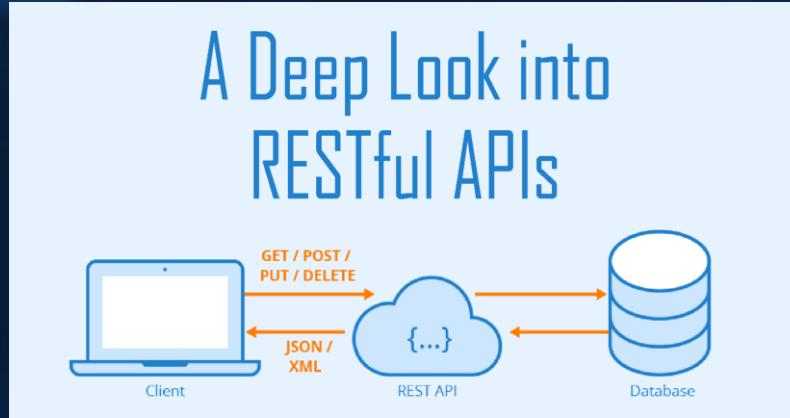
The screenshot shows the official website for Cypress. At the top, there's a navigation bar with links for Features, How it works, Dashboard, Pricing, Support, Blog, Docs, What's New, Login, and a green Sign up button. The main headline reads "The web has evolved. Finally, testing has too." Below it, a sub-headline says "Fast, easy and reliable testing for anything that runs in a browser." There are two download options: "\$ npm install cypress" and "or Download now". A note below says "Install Cypress for Mac, Linux, or Windows, then get started." On the left side, there's a sidebar with icons for mobile, browser, and application testing. On the right side, there are various icons representing different testing scenarios like databases, APIs, and user interactions. At the bottom, there's a screenshot of a browser window showing Cypress running tests against a GitHub page for Bootstrap themes. The test code in the browser's developer tools looks like this:

```
1 VISIT  http://bootstrap-themes.github.io/_/tests/integration/demo_spec.js
2 -WAIT  100
3 GET    img[src="assets/img/instagram_3.jpg"]
        .catch((err) => console.error(err))
        .timeout(1000)
```

The browser also displays a feed of posts from users like Dave Gamache.

# Тестування API

API – це набір процедур, протоколів і інструментів взаємодії для створення програмних додатків.



Тестове API

The screenshot shows a user interface for testing a REST API endpoint. At the top, there is a text input field labeled 'API endpoint' containing the URL [https://631f942258a1c0fe9f6c1e44.mockapi.io/api\\_test/:endpoint](https://631f942258a1c0fe9f6c1e44.mockapi.io/api_test/:endpoint). Below the input field are two buttons: 'NEW RESOURCE' on the left and 'GENERATE ALL' and 'RESET ALL' on the right. A large button labeled 'users' is visible at the bottom. In the center, there is a table with one row showing the number '52' in a blue box, with buttons for 'Data', 'Edit', and 'Delete' next to it.

# Checkly для тестування API

Checkly – це платформа для моніторингу продуктивності та коректної роботи API

### Create an API Check

ON  ACTIVATED  OFF MUTED [?](#)

CHECK NAME \*

CHECK TAGS

Type a tag, hit enter

Make an HTTP request

Create an HTTP request manually or import one. Use [global environment variables](#) to store secrets and reuse them in other checks.

METHOD \*  URL \*

[Run request](#)

[curl:// Import a cURL command](#) [Import from Swagger / OpenAPI](#)  Skip SSL [?](#)  Follow redirects [?](#)  This request should fail [?](#)

[BODY](#) [HEADERS](#) [QUERY PARAMS](#) [AUTHENTICATION](#)

Selecting the data type will automatically set the correct **Content-Type** header. You can override this by setting your own **Content-Type** header in the "add headers" section. You can also use variables in the body content, i.e. `{{{CUSTOMER_ID}}}`

No body  JSON  GraphQL  Form parameters  Raw data

1



## Переваги Checkly

- Графічний інтерфейс
- Гнучка настройка тестів
- Сповіщення о помилках на email
- Широка географія серверів
- Додаткові фічі (відстеження терміну дії SSL, подвійні перевірки тощо)

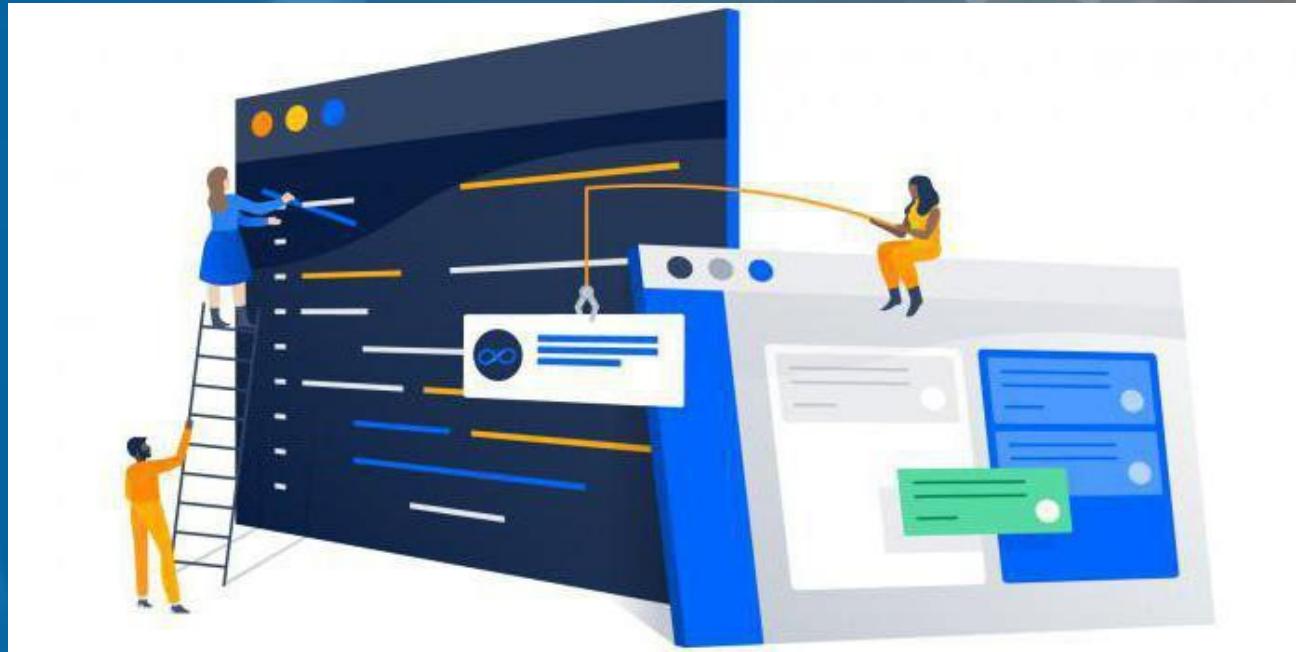
We already ran **3 274 026 178** API checks and **362 680 179** browser checks!

# Системи автоматизованого відстежування помилок

- Trello
- Jira Software
- Height
- Bugzilla
- Basecamp
- BugHerd

# Jira Software

Jira – система відстеження помилок, призначена для організації спілкування з користувачами, і для управління проектами.



- Summary - включає в себе лише загальний і короткий опис проблеми.
- Description - описуємо баг настільки широко, щоб це було зрозуміло навіть дитині. Цей пункт має включати такі ключові критерії: Steps to reproduce, Actual result, Expected result.
- Environment - тут вказуємо такі параметри як девайс, ос, браузер, і розширення. Не потрібно описувати девайс в Description, якщо в цьому немає потреби.
- Скріншоти і відео.
- Все що нижче (Platforms, Page, Feature, і т.д.) ставите інтуїтивно. Після вище вказаних пунктів створюємо баг репорт.
- Підзадач має бути дві - QA і Frontend/Backend(в залежності від баги).

# Створення баг репортів в Jira Software

The screenshot shows the Jira Software interface for the project "UzhnulInfoSys". The left sidebar includes sections for PLANNING (Roadmap, Board), DEVELOPMENT (Code), and other project management tools. The main area displays a Kanban board titled "UZINF board" with columns: TO DO, IN PROGRESS, and DONE. A red arrow points from the top center towards the "Create" button in the header menu bar.

Jira Software

Your work ▾ Projects ▾ Filters ▾ Dashboards ▾ People ▾ Apps ▾ Create

Search

UzhnulInfoSys  
Software project

Projects / UzhnulInfoSys

UZINF board

GROUP BY None ▾

TO DO

+ Create issue

IN PROGRESS

DONE ✓

+ Add

Roadmap

Board

Code

Project pages

Add shortcut

Project settings

You're in a team-managed project

Learn more

### Create issue

Import issues ...

Project\*

Issue type\*  Bug

Learn more

Summary\*

Description

Preconditions:  
The user is on [system.web.app/](https://system.web.app/)

Steps to reproduce:

1. Navigate to <https://system.web.app/>
2. Select filter button
3. Switch to lecturer side
4. Select random lecturer
5. Click "До розкладу"
6. Click lecturer surname in the table
7. Pay attention to the popup

Actual result:  
Popup is opening in the lower part of the screen.

Expected result:  
Expected that the popup would not be showing on the lecturer's schedule

Assignee

Assign to me

Labels

Reporter\*

Attachment

Linked Issues   
Select Issue

Create another issue

Add epic / INF-2

## Improper use of a hint

Attach Add a child issue Link issue ...

Description

Preconditions:

The user is on <https://system.web.app/>

Steps to reproduce:

1. Navigate to <https://system.web.app/>
2. Select filter button
3. Switch to lecturer side
4. Select random lecturer
5. Click "До розкладу"
6. Click lecturer surname in the table
7. Pay attention to the popup

Actual result:

Popup is opening in the lower part of the screen.

Expected result:

Expected that the popup would not be showing on the lecturer's schedule

Attachments (1)

chrome-capture-...-12.gif

Activity

Show: All  Comments  History Newest first ↗

Add a comment...

Pro tips press **M** to comment

To Do

Pinned fields

Click on the next to a field label to start pinning.

Details

Assignee Unassigned Assign to me

Labels None

Reporter Oleg

Created 9 seconds ago Updated 9 seconds ago