Developing Playwright Framework for REST API Testing

Building Framework that people **actually want** to use

Who am I

- Civil Engineer with over 7 years of experience
- QA Engineer for over 4 years of experience
- Interests
  - Simplifying Complex Topics
  - Solving Problems
  - Improving products and processes
  - Helping others to become better professionals

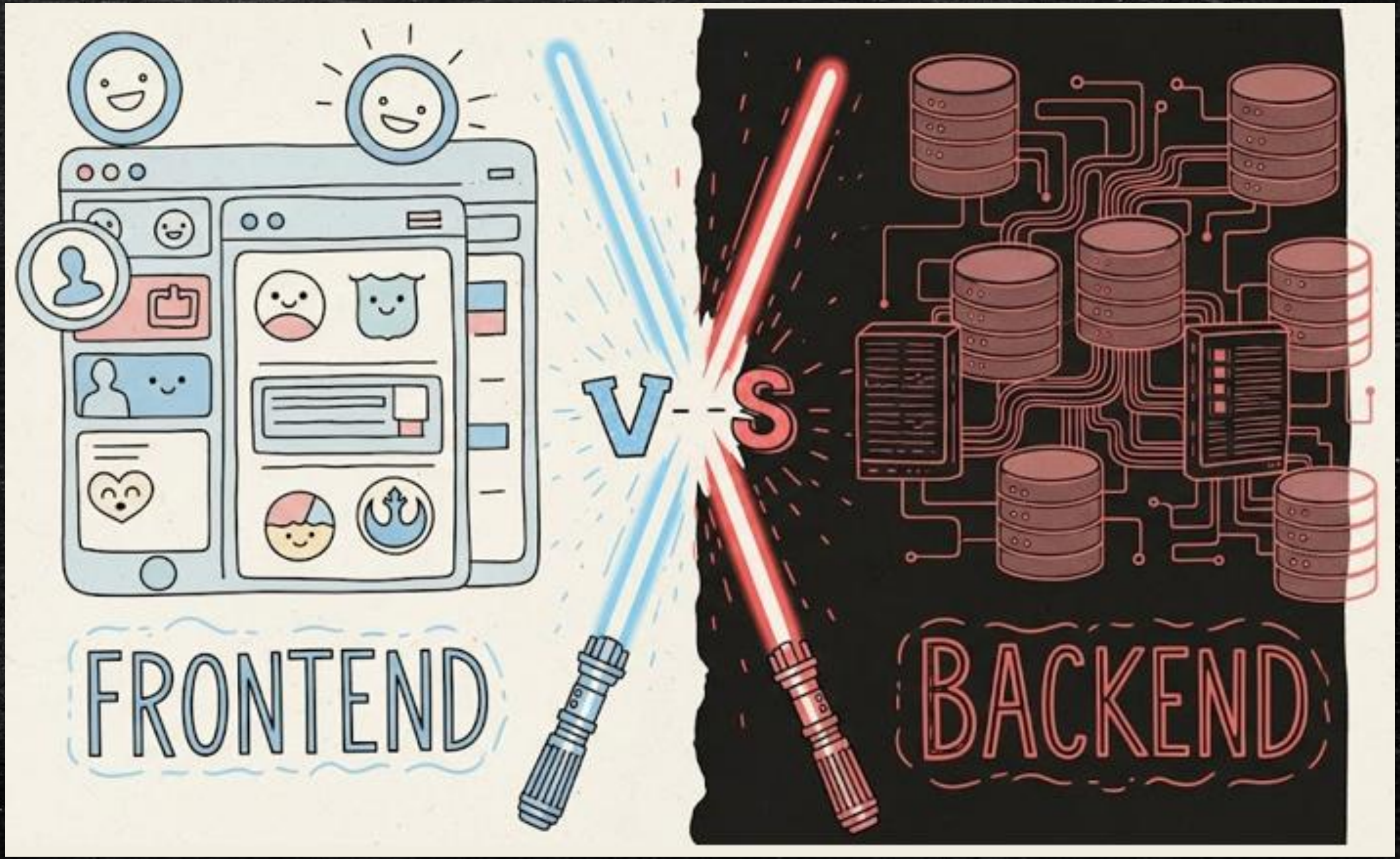
Agenda

- The Two Sides of an Application
- Where and What Do The Bugs Hide
- API Testing
  - What is API
  - Types of API Testing
- The Power of a Unified Tool
- Developing Playwright Framework for REST API Testing
  - Improving Developer Experience (DX)
  - The Abstraction Layer
  - The Magic – Custom Fixtures
  - Bulletproof – Zod Schema Validation

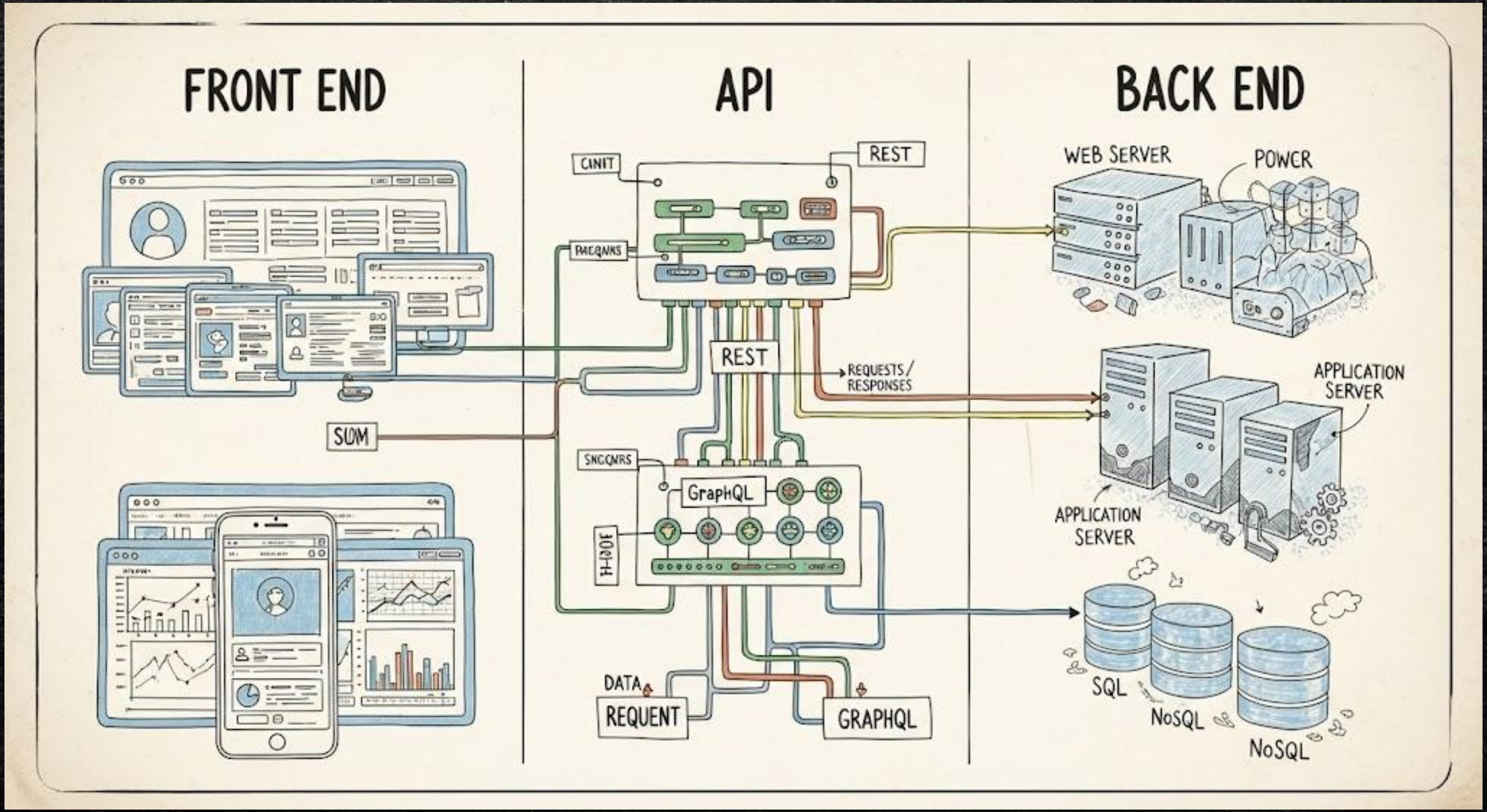
The Two Sides of a Web Application

High-level Architecture of Web App

- Front-End – Responsible for UI/UX
- Back-End – Responsible for Business Logic

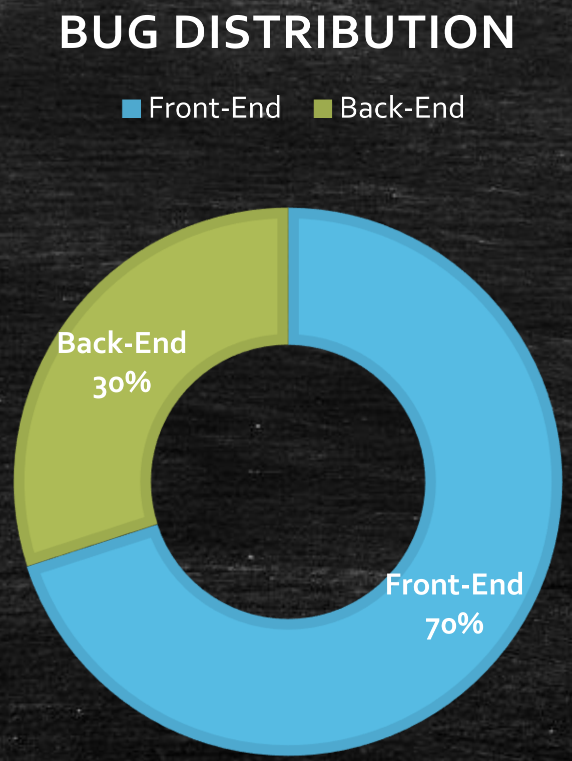


Communication Between the Two Sides



Where and What Do the Bugs Hide?

Front-End vs Back-End Bug Distribution

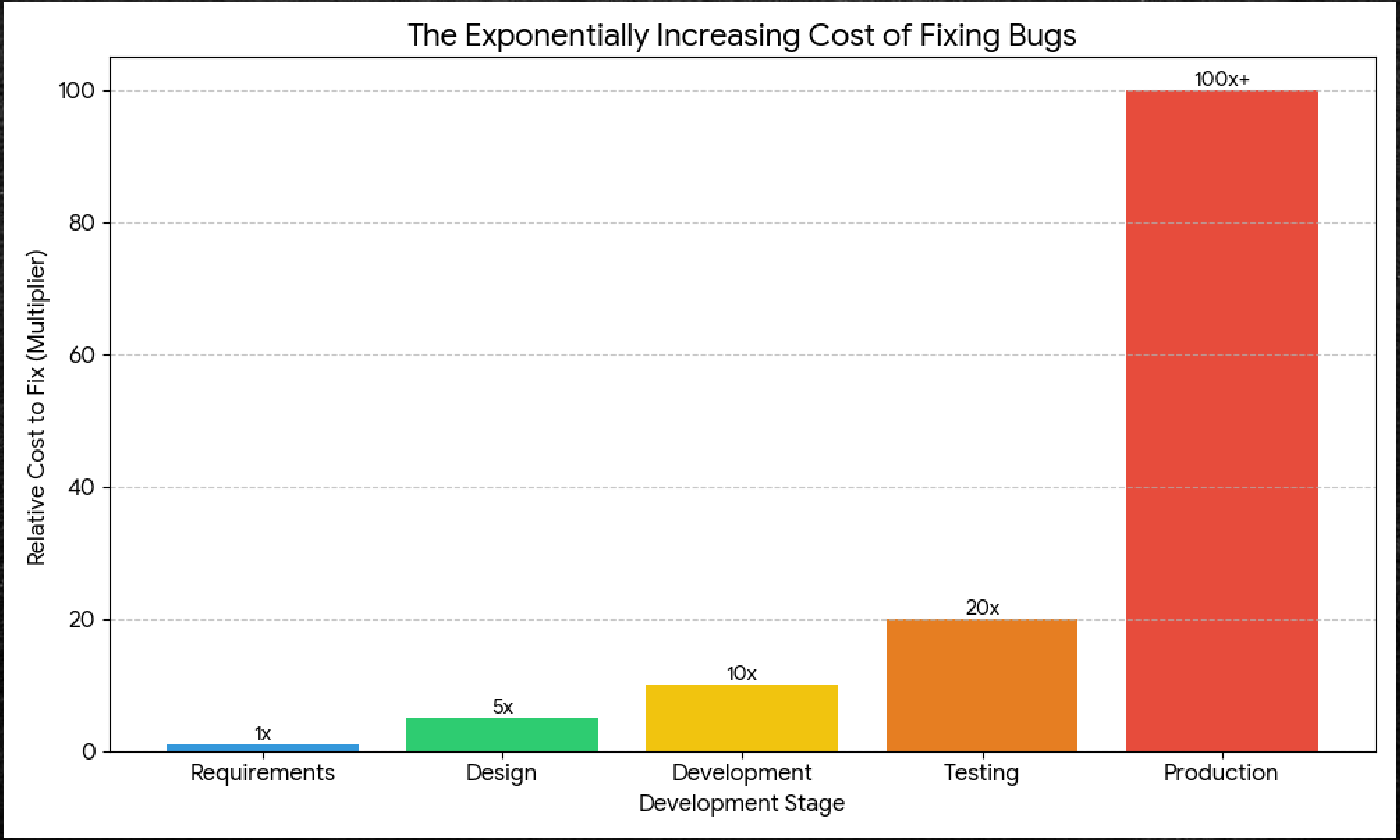


NOTE: The Data can vary significantly due to different factors

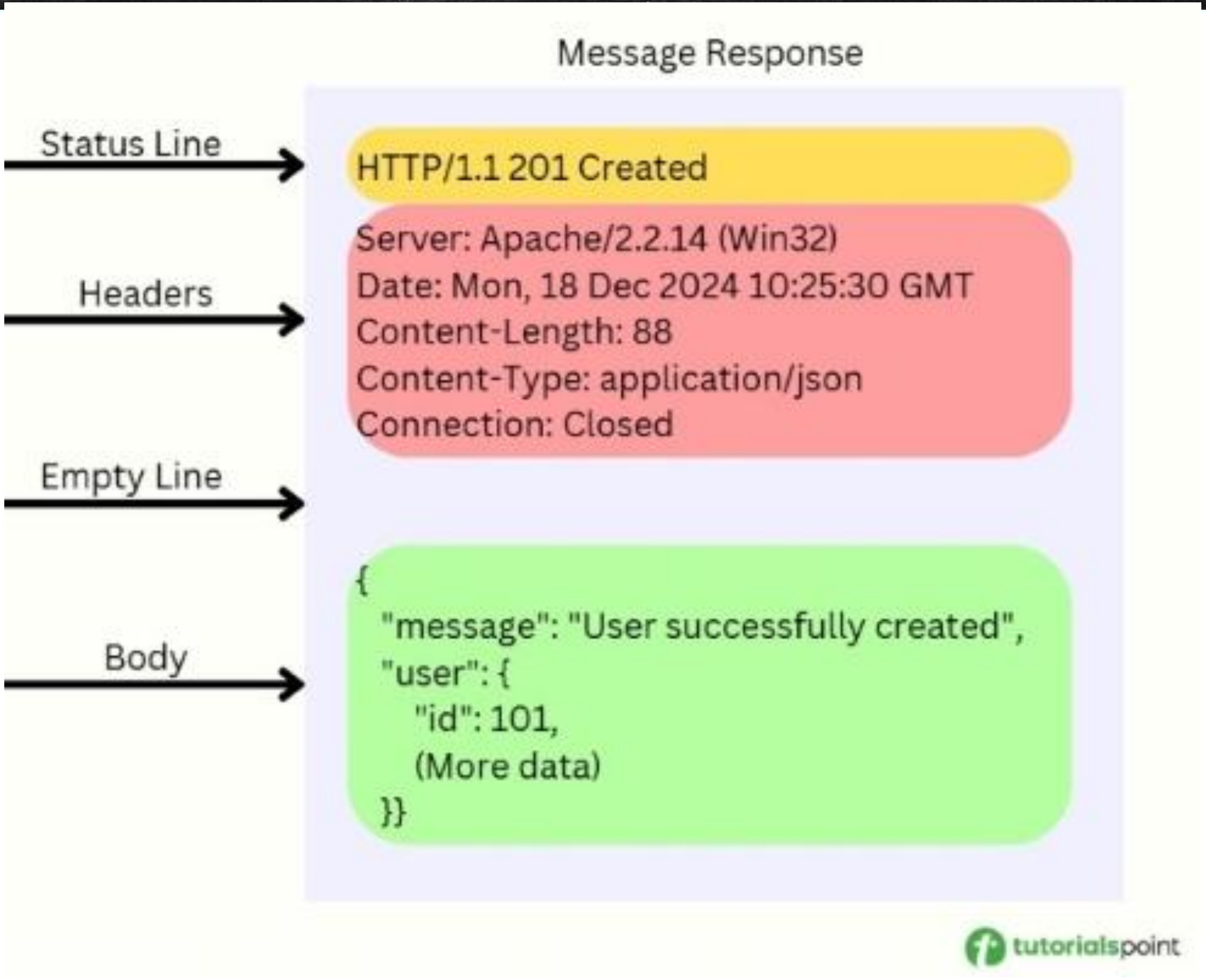
The Nature of Bugs: A Volume vs. Severity Profile

	LOWER SEVERITY	HIGHER SEVERITY
HIGH VOLUME	**FRONT-END BUGS** *(e.g., Visual glitches, CSS misalignments, minor usability issues)*	**FRONT-END BUGS** *(e.g., Broken "Checkout" button, major browser incompatibility)*
LOW VOLUME	**BACK-END BUGS** *(e.g., Minor API performance lag, inefficient internal query)*	**BACK-END BUGS** *(e.g., Security vulnerabilities, data corruption, system-wide outages)*

The Escalating Cost of Bug Remediation by Development Stage



API Testing



```
HTTP/1.1 201 Created
Server: Apache/2.2.14 (Win32)
Date: Mon, 18 Dec 2024 10:25:30 GMT
Content-Length: 88
Content-Type: application/json
Connection: Closed

{
  "message": "User successfully created",
  "user": {
    "id": 101,
    (More data)
  }}
```

The Importance

- Functional Testing
- Security Testing
- Integration Testing
- Performance Testing

The Power of a Unified Tool

Playwright

- World-Class Tool for UI Testing
- Robust API Testing
- Power of a Unified Tool
  - Reliability and Speed
  - CI/CD Simplification
  - Seamless End-to-End Testing
  - Developer Experience (DX)

Developing Playwright Framework for REST API Testing

What we need?

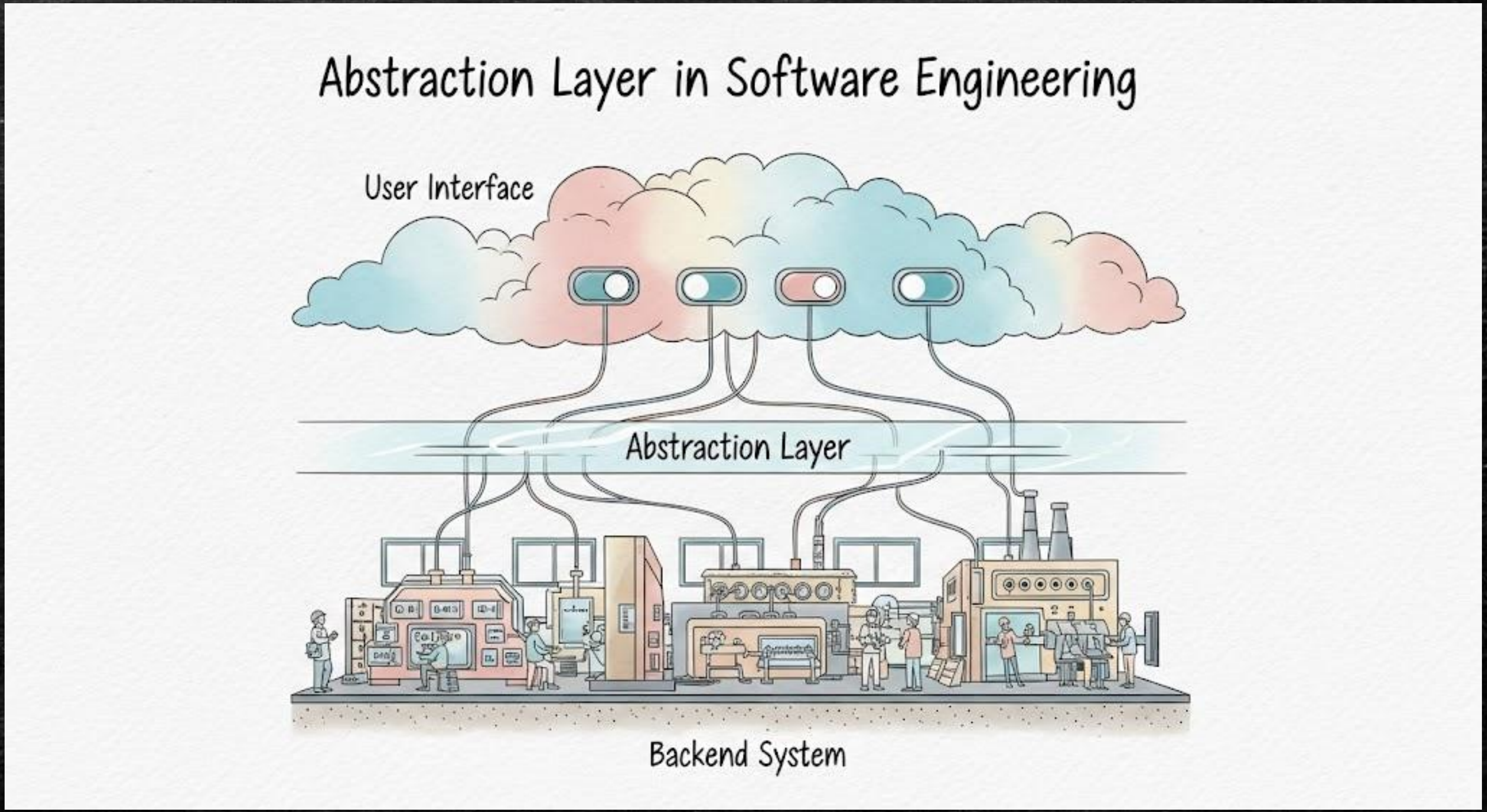
- Materials – GitHub Link - https://github.com/idavidov13/SoftUni-PW-API-Framework-Materials
- IDE – Cursor (Windsurf, VS Code)
- Application Under Test (AUT) - https://conduit.bondaracademy.com/
- Final Repository - https://github.com/idavidov13/SoftUni-PW-API-Framework-25.08.2025

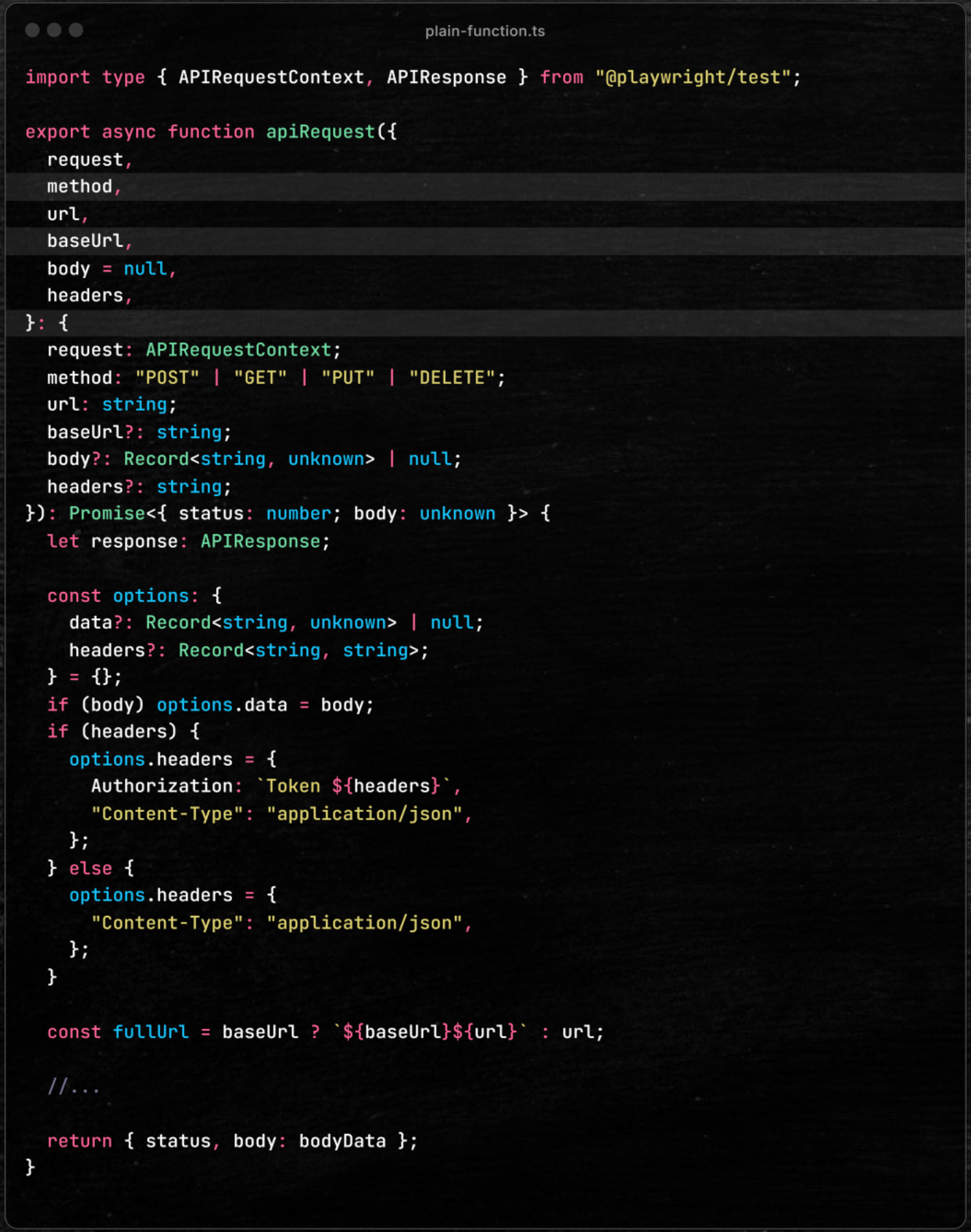
Improving Developer Experience (DX)

- Developer Experience – why it is important?
  - Developer Productivity - how quickly or simply a change can be made to a codebase
  - Developer Impact - how frictionless it is to move from idea to production
  - Developer Satisfaction - how the environment, and tools affect developer happiness
- How can be Improved
  - Productivity - using Cursor IDE
  - Impact - Playwright enables reliable end-to-end testing for modern web apps
  - Satisfaction - Implementing User Snippets and Custom Fixtures

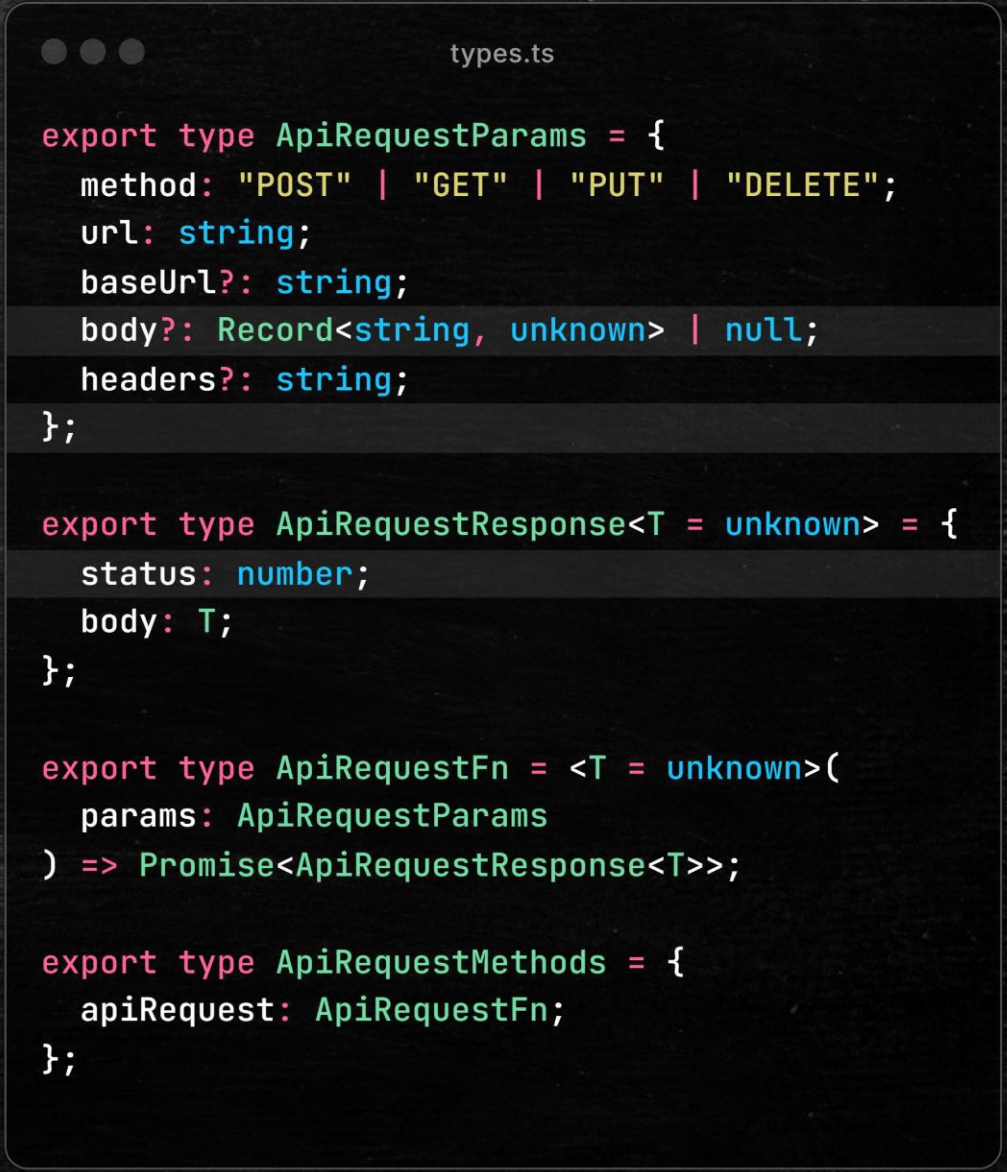
The Abstraction Layer

1. Create abstracted layer of API request, which unifies the work process
2. Remove unnecessary checks in the tests
3. Return only what is needed in the test




plain-function.ts

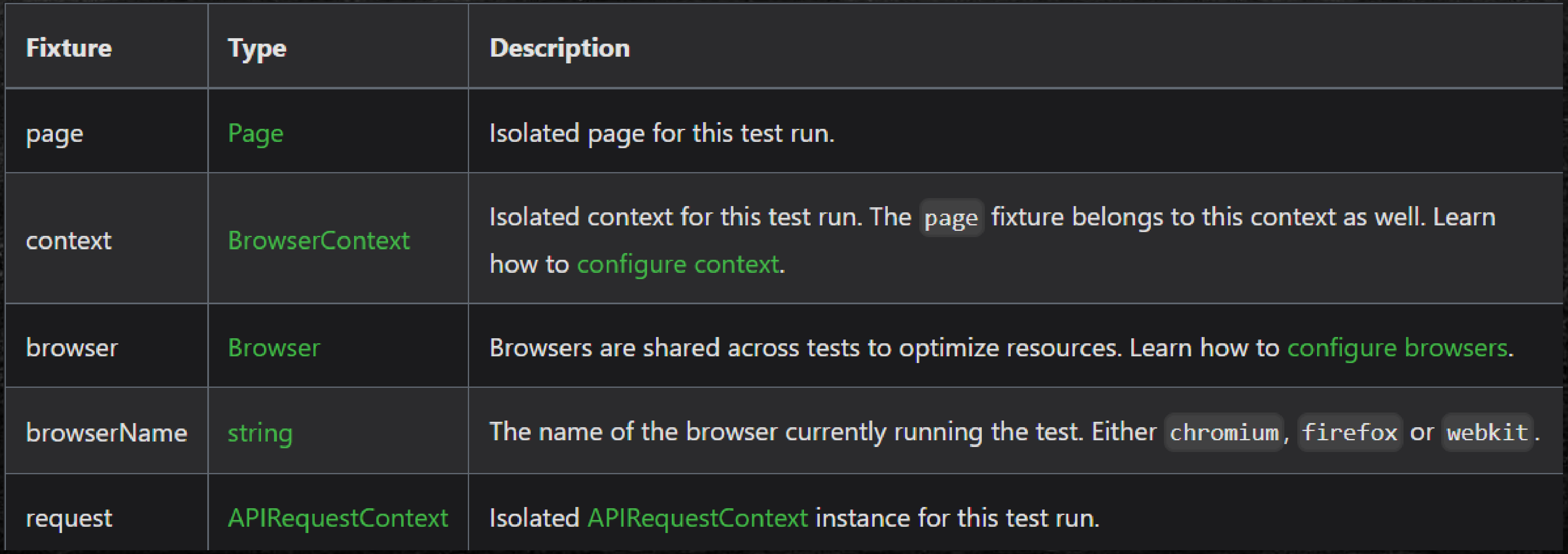
```ts
import type { APIRequestContext, APIResponse } from "@playwright/test";

export async function apiRequest({
  request,
  method,
  url,
  baseUrl,
  body = null,
  headers,
}: {
  request: APIRequestContext;
  method: "POST" | "GET" | "PUT" | "DELETE";
  url: string;
  baseUrl?: string;
  body?: Record<string, unknown> | null;
  headers?: string;
}): Promise<{ status: number; body: unknown }> {
  let response: APIResponse;

  const options: {
    data?: Record<string, unknown> | null;
    headers?: Record<string, string>;
  } = {};
  if (body) options.data = body;
  if (headers) {
    options.headers = {
      Authorization: `Token ${headers}`,
      "Content-Type": "application/json",
    };
  } else {
    options.headers = {
      "Content-Type": "application/json",
    };
  }

  const fullUrl = baseUrl ? `${baseUrl}${url}` : url;

  //...

  return { status, body: bodyData };
}
```


types.ts

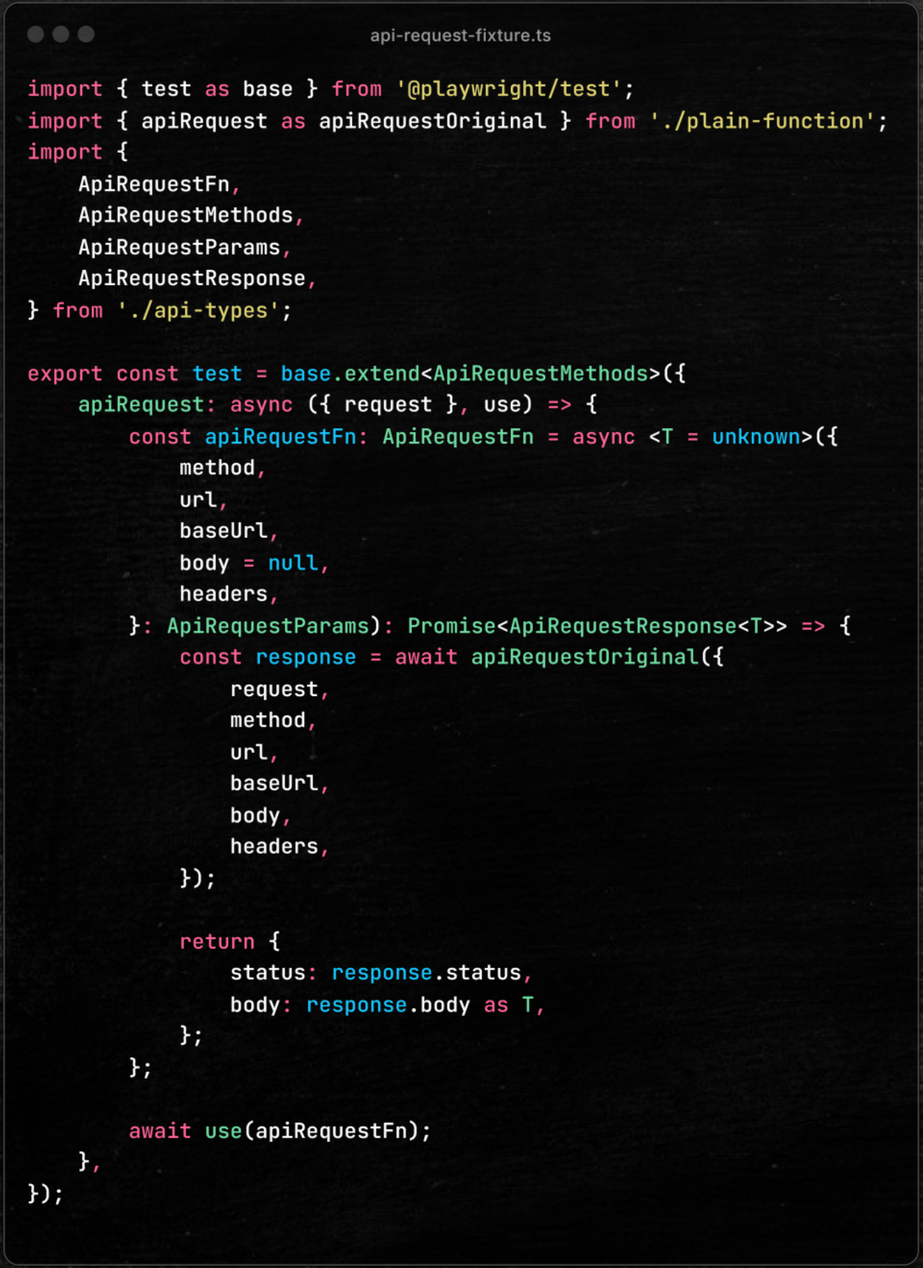
```ts
export type ApiRequestParams = {
  method: "POST" | "GET" | "PUT" | "DELETE";
  url: string;
  baseUrl?: string;
  body?: Record<string, unknown> | null;
  headers?: string;
};

export type ApiRequestResponse<T = unknown> = {
  status: number;
  body: T;
};

export type ApiRequestFn = <T = unknown>(
  params: ApiRequestParams
) => Promise<ApiRequestResponse<T>>;

export type ApiRequestMethods = {
  apiRequest: ApiRequestFn;
};
```

The Magic – Custom Fixtures

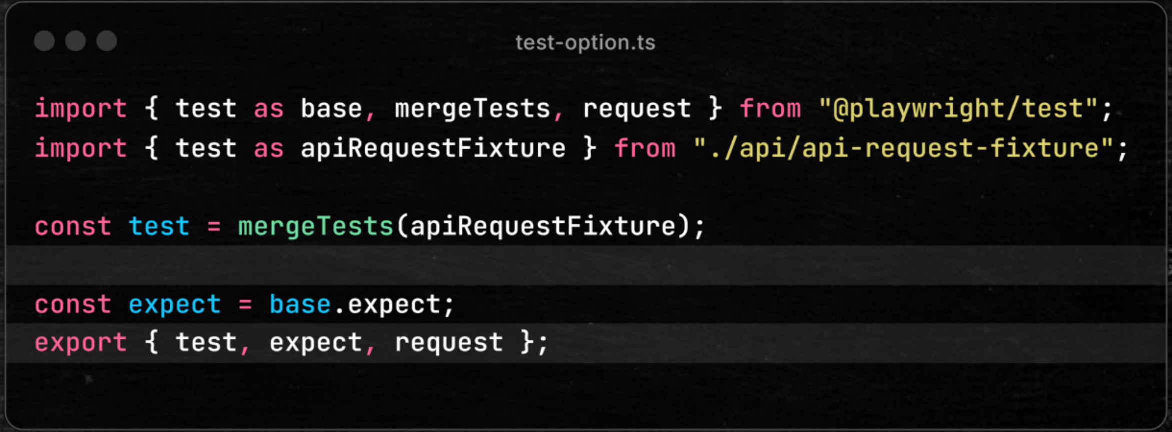
- Test fixtures are used to establish the environment for each test, giving the test everything it needs and nothing else.
- List of Built-in Playwright Fixtures



Fixture	Type	Description
page	Page	Isolated page for this test run.
context	BrowserContext	Isolated context for this test run. The `page` fixture belongs to this context as well. Learn how to configure context.
browser	Browser	Browsers are shared across tests to optimize resources. Learn how to configure browsers.
browserName	string	The name of the browser currently running the test. Either `chromium`, `firefox` or `webkit`.
request	APIRequestContext	Isolated APIRequestContext instance for this test run.


api-request-fixture.ts

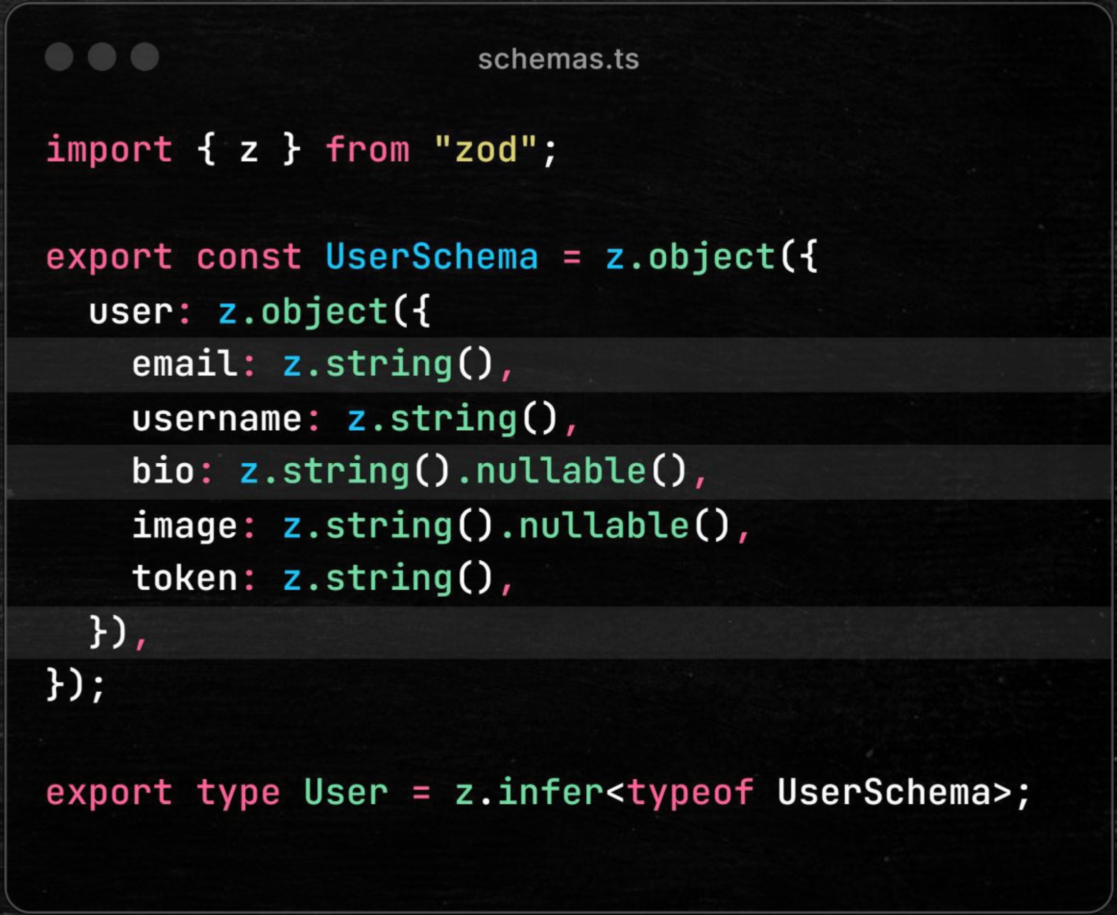
```ts
import { test as base } from '@playwright/test';
import { apiRequest as apiRequestOriginal } from './plain-function';
import {
    ApiRequestFn,
    ApiRequestMethods,
    ApiRequestParams,
    ApiRequestResponse,
} from './api-types';

export const test = base.extend<ApiRequestMethods>({
    apiRequest: async ({ request }, use) => {
        const apiRequestFn: ApiRequestFn = async <T = unknown>({
            method,
            url,
            baseUrl,
            body = null,
            headers,
        }: ApiRequestParams): Promise<ApiRequestResponse<T>> => {
            const response = await apiRequestOriginal({
                request,
                method,
                url,
                baseUrl,
                body,
                headers,
            });

            return {
                status: response.status,
                body: response.body as T,
            };
        };

        await use(apiRequestFn);
    },
});
```


test-option.ts

```ts
import { test as base, mergeTests, request } from "@playwright/test";
import { test as apiRequestFixture } from "./api/api-request-fixture";

const test = mergeTests(apiRequestFixture);

const expect = base.expect;
export { test, expect, request };
```

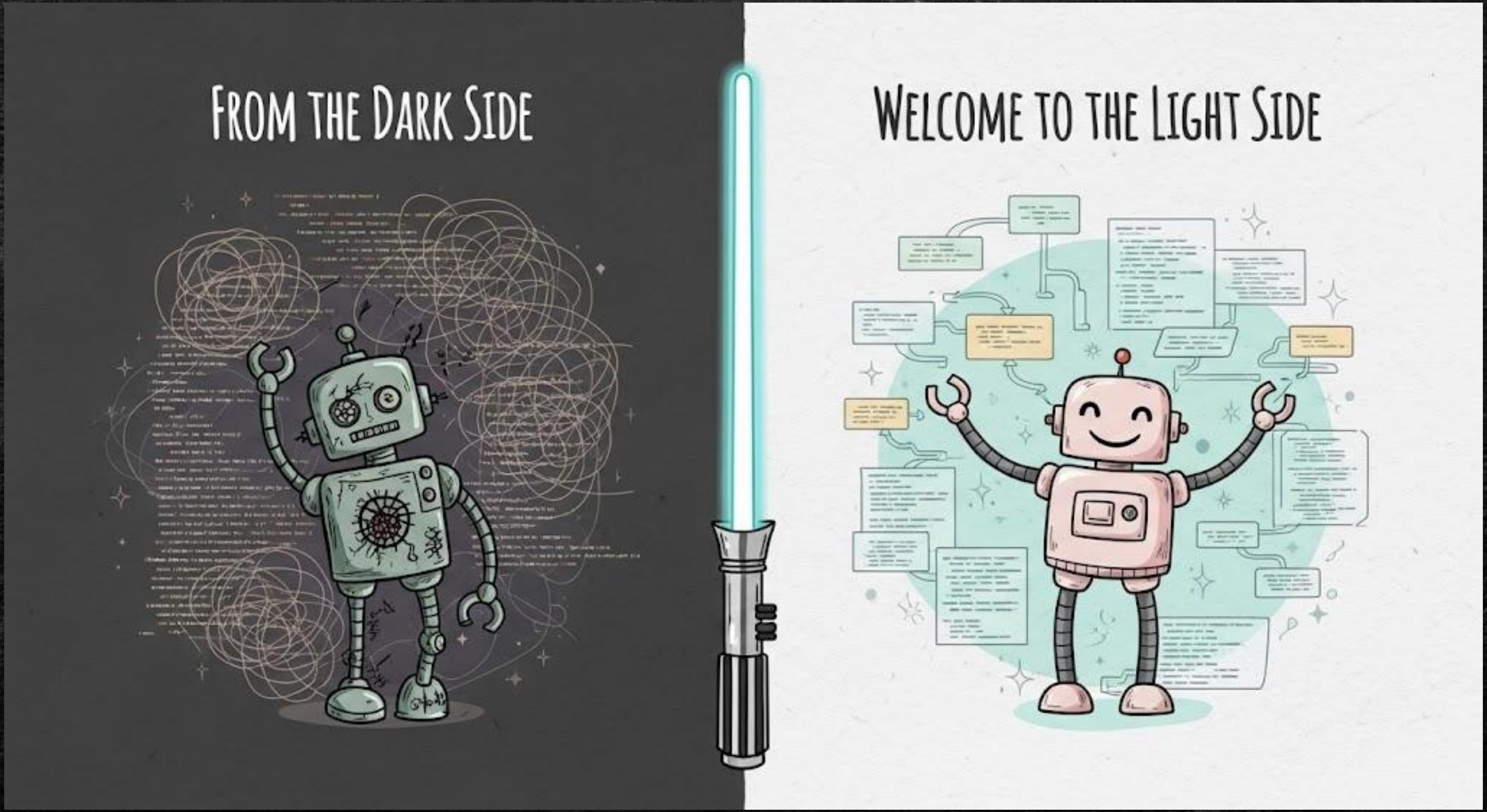
Bulletproof – Zod Schema Validation

- Zod is a TypeScript-first validation library
- By defining schemas, you can validate data, from a simple string to a complex nested object.



```ts
import { z } from "zod";

export const UserSchema = z.object({
  user: z.object({
    email: z.string(),
    username: z.string(),
    bio: z.string().nullable(),
    image: z.string().nullable(),
    token: z.string(),
  }),
});

export type User = z.infer<typeof UserSchema>;
```

Q&A Session



Thank you!

Contacts

- LinkedIn – https://www.linkedin.com/in/ivdavidov/
- GitHub – https://github.com/idavidov13
- Blog – https://www.idavidov.eu