

Semsei Izsák
Fejlesztő, AI-al épített rendszerek

BULLETPROOF CODING.

Miért írj teszteket a vibe-kódolt appodra, és hogyan?

Ingyenes útmutató azoknak, akik Claude Code-dal, Cursorral vagy Lovable-lel építenek appot, és szeretnék, ha az élesben is működne.

5 tesztszint
unittól a CI/CD-ig

5 kész prompt
Claude Code-hoz és Cursorhoz

Valódi hibák
élesből, tanulsággal

1 oldalas lista
élesítés előtt, nyomtatható



KI VAGYOK, ÉS MIÉRT ÍRTAM EZT.

Szia! Izsák vagyok, a Simon's Burger fejlesztője. Az elmúlt egy évben a cég rendszereinek nagy részét Claude Code-dal építettem.

Amit AI-al építettem

- Weboldal
- Vendégelégedettségi kérdőív
- Tréningrendszer
- LED-fal show-vezérlő

Miért írtam

Ebben az útmutatóban azt írtam le, amit közben a tesztelésről megtanultam, néha a saját káromon. A tesztelést nem könyvből tanultam, hanem abból, amikor valami élesben nem úgy működött, ahogy a gépemen.

Nem kell hozzá fejlesztőnek lenned. Ha AI-al építesz appot, és szeretnéd, ha holnap is működne, ez neked szól.



Ezzel a jellel találkozol majd
Tapasztalatból

Ahol ez a kép van, ott egy valódi hibáról vagy döntésről olvasol a saját projektjeimből, tanulsággal együtt.

Amit szeretnék, hogy elvigyél

NE CSAK AZT KÉRDEZD, HOGY ZÖLD-E. KÉRDEZD MEG, MIRE KÉRDEZETT RÁ A TESZT.

A ZÖLD PIPA NEM GARANCIA.

Az AI pár perc alatt megír egy funkciót, és magabiztosan közli, hogy kész. Hogy tényleg jól működik-e, azt egy teszt tudja megmondani. Pontosabban: azt, amire rákérdez.

Az AI gyorsan ír

Egy délután elkészül, amire régen egy hét kellett. A hibák is ugyanilyen gyorsan születnek, és ugyanilyen gyorsan kerülnek élesbe.

És magabiztosan jelent

„Kész, működik", akkor is, ha senki nem próbálta ki. Ez nem rosszindulat: a modell a legvalószínűbb választ adja, nem a leellenőrzöttet.

Te nem olvasol el minden sort

Vibe-kódolásnál az eredményt nézed, nem a kódot. Ezért kell valami, ami helyetted figyeli, hogy a régi dolgok is működnek-e még.

A teszt egy megállapodás

Leírod, mit kell tudnia az appnak, és a gép minden változtatás után ellenőrzi. Nem fárad el, és nem felejt.

Mi az a teszt?

egy teszt anatómiája

Egy kis program, ami a te programodat ellenőrzi. Mindig három részből áll.

```
// 1. Előkészítés: két burger, darabja 2 500 Ft
const cart = [{ price: 2500, qty: 2 }];

// 2. Végrehajtás
const total = cartTotal(cart);

// 3. Ellenőrzés: ennyinek kell lennie
expect(total).toBe(5000);
```

1. **Előkészítés** — Bemenő adat, amit te választas
2. **Végrehajtás** — Meghívod, amit tesztelsz.
3. **Ellenőrzés** — Kimondod, minek kell kijönnie.

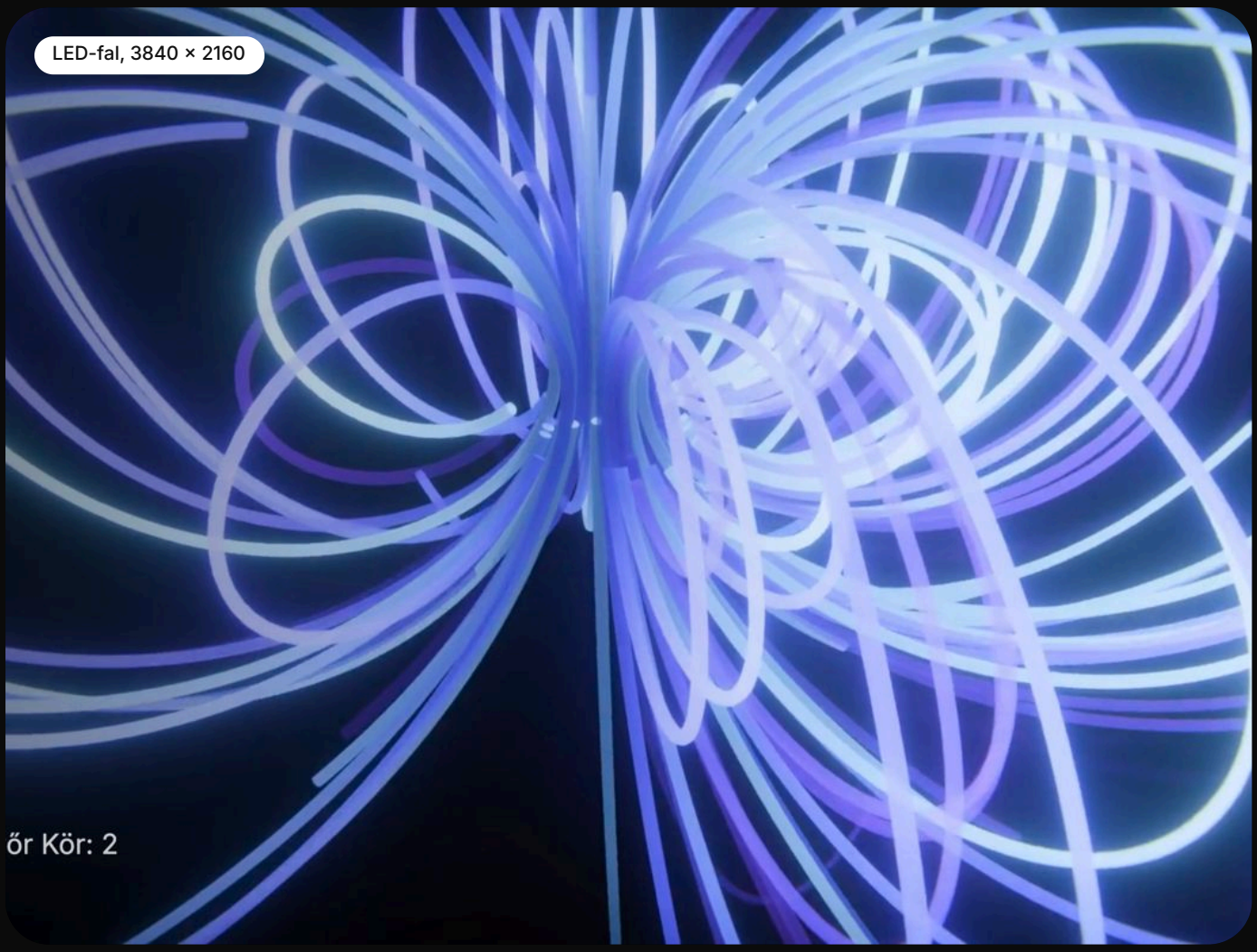
Ha az eredmény eltér, a teszt **piros lesz**, és pontosan megmondja, mit várt és mit kapott.

```
✗ cartTotal
  Elvárt   5000
  Kapott   2500
```

A lényeg: a teszt csak azt bizonyítja, amit ellenőriz. Ha rossz kérdést teszel fel, a zöld pipa semmit nem ér. Erről szól a következő történet.

A TESZT ZÖLD VOLT. A SORREND ROSSZ.

2026. szeptember 30-án volt a Smash Champs döntője, a Simon's smash burger-versenye. A LED-falon futó show-t én építettem Claude Code-dal, és én is vezéreltem.



A show-vezérlő egyik várakozó animációja (demó-felvétel). Erre a képre lehetett bármilyen szöveget kiírni.

Mi történt

- **A döntő napja.** Kérés érkezett: az esti fordulóban a sorrendet az idő és a büntetés összege adja.
- **Gyorsan megépítettem.** Claude Code-dal ez nem volt nagy munka, a szabály bekerült a pontozásba.
- **Egy teszt a saját gépemen.** Előre kitöltött számokkal lefuttattam. Átment.
- **Élesben derült ki**, hogy a tesztadatban a büntetés soha nem változtatott a sorrenden. A teszt akkor is zöld lett volna, ha a program a büntetést figyelmen kívül hagyja.
- **A helyszínen javítottuk.** Korábban beépítettem egy funkciót, amivel bármilyen szöveget ki tudok írni a várakozó animációra. Ezzel került ki a falra a helyes sorrend.

MIÉRT NEM VETTE ÉSZRE?

Mert a tesztadatban a büntetés túl kicsi volt ahhoz, hogy bármit megfordítson. A jó és a rossz program ugyanazt a sorrendet adta.

A tesztadat

ugyanaz a sorrend



Büntetéssel és nélküle is: **1. A, 2. B**

Ilyen adat kellett volna

megfordul



Büntetéssel: **1. B**, nélküle: 1. A

idő (s) büntetés (s)

Szemléltető számok, nem a verseny adatai.

Amit azóta máshogy csinálok

- Minden szabályhoz van olyan teszteset, ahol a **rossz megoldás más eredményt adna**, mint a jó.
- Ha az AI azt mondja, „ellenőrizve”, megkérdezem: **milyen adaton futott?**
- Élő eseményen mindig van egy kézi tartalék, amivel bármit ki tudok javítani a közönség előtt.



Tapasztalatból
Ez nem az AI hibája volt

A kód azt csinálta, amit kértem. A teszt is azt, amit kértem. Én kérdeztem rosszul, és siettem. Pont az ilyen napokon van a legnagyobb szükség jó tesztre: amikor nincs idő mindent kézzel végignézni.

A következő oldalakon az öt tesztszintet veszem végig, mindegyikhez egy kész prompttal, ami pont az ilyen hibákat kéri számon az AI-tól.

A tanulság

A TESZT NEM HAZUDOTT, CSAK ROSSZ KÉRDÉST TETT FEL.

ÖT SZINT, EGY RENDSZER.

Minden szint mást fog meg. A gyors, olcsó tesztekből legyen a legtöbb. Felfelé lassabbak, de egyre közelebb vannak ahhoz, amit a felhasználód lát. A CI/CD gondoskodik róla, hogy mindegyik le is fusson.



UNIT	Egyetlen függvény jól számol? Pontszám, ár, sorrend, áfa.	ezredmásodperc	08
INTEGRÁCIÓS	A kód és az adatbázis együtt is jól működik?	másodperc	11
END-TO-END	Egy ember végig tud menni a fő folyamaton?	másodpercek	14
TELJESÍTMÉNY	Elég gyors, és gyors marad, ha nő az adat?	perc	17
CI/CD	Minden feltöltésnél lefut az összes, és piros teszttel nem megy élesbe.	automatikus	20

A sáv azt mutatja, nagyjából mennyi teszted legyen az adott szintből a többihez képest.

TARTALOM.

Minden szintnél ugyanaz a felépítés: egy nyitóoldal, egy működő kódpélda jelölésekkel, és egy prompt, amit bemásolhatsz Claude Code-ba vagy Cursorba.

02	Ki vagyok, és miért írtam ezt	02
03	A zöld pipa nem garancia	03
04	A történet: Smash Champs döntő	04
06	Öt szint, egy rendszer	06
	1. Unit teszt	08
	2. Integrációs teszt	11
	3. End-to-end teszt	14
	4. Teljesítményteszt	17
	5. CI/CD	20
	Hogyan kezdj hozzá ma	23
	Puska: az egész egy oldalon	24
	Élesítés előtti ellenőrzőlista	25
	Biztonság, amit a teszt nem lát	26
	A szerzőről	27

Jelek a könyvben

Prompt — bemásolható, a <> részt cseréld

1 — jelölés a kódban, magyarázat mellette

| — a sor, ami tényleg ellenőriz

 — tapasztalatból: valódi hiba, tanulsággal

A kódpéldák

TypeScript (Vitest, Playwright) és Python (pytest). Nem kell érteni minden sorát. Az a cél, hogy lásd, mit kérj az AI-tól, és felismerd, ha rosszat ad.

Ha csak egy dolgot viszel el

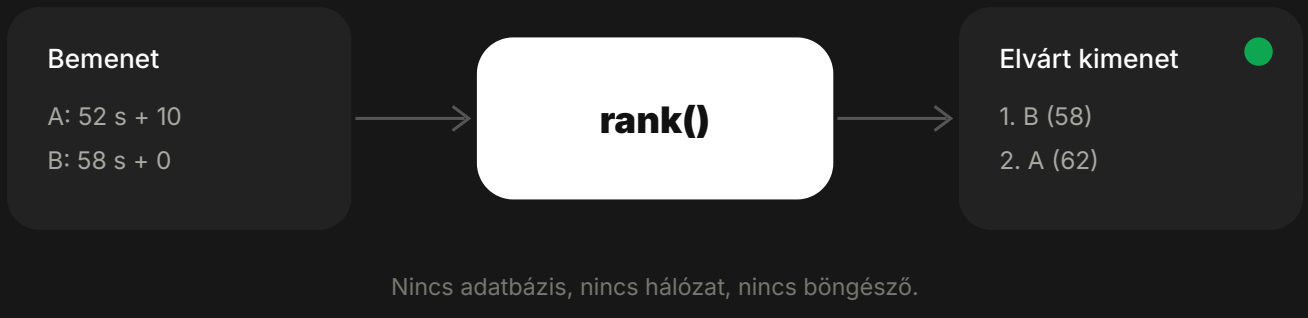
KÉRD MEG AZ AI-T, HOGY MUTASSA MEG: A TESZT ELBUKIK, HA SZÁNDÉKOSAN ELRONTOD A KÓDOT.

01

UNIT TESZT.

Egyetlen függvény jól számol? Pontszám, ár, sorrend, áfa. Ezredmásodpercek alatt kiderül.

ezredmásodperc | ebből legyen a legtöbb | Vitest, pytest



Nincs adatbázis, nincs hálózat, nincs böngésző.

Mit fog meg

- Rossz számolás: ár, áfa, pontszám
- Rossz sorrend és rendezés
- Elfelejtett szabály, mint a büntetés
- Határesetek: üres, nulla, holtverseny

Mit nem

- Hogy az adat tényleg elmentődik-e
- Hogy a gomb megnyomható-e

A RANGSOR, TESZTELVE.

A legkisebb egység: egyetlen függvény, kívülről semmi. Adatot kap, és megnézed, jó-e, amit visszaad.

Példa: a versenyrangsor, TypeScript és Vitest

npx vitest

ranking.ts

```ts
export type Result = { name: string; timeSec: number; penaltySec: number };

export function rank(results: Result[]): Result[] {
  // másolatot rendezünk, az eredeti lista nem változik
  return [...results].sort( 1
    (a, b) => a.timeSec + a.penaltySec - (b.timeSec + b.penaltySec), 2
  );
}
```

ranking.test.ts

```ts
import { describe, it, expect } from "vitest";
import { rank } from "./ranking";

describe("rank", () => {
  it("a büntetés megfordíthatja a sorrendet", () => { 3
    const results = [
      { name: "A", timeSec: 52, penaltySec: 10 }, // összesen 62 4
      { name: "B", timeSec: 58, penaltySec: 0 },  // összesen 58
    ];
    expect(rank(results).map((r) => r.name)).toEqual(["B", "A"]); 5
  });

  it("üres listára üres listát ad", () => { 6
    expect(rank([])).toEqual([]);
  });
});
```

1. **Másolat** — A [...] miatt az eredeti lista érintetlen marad.
2. **A szabály** — Idő + büntetés. Egy rossz megoldás pont ezt a felét felejti ki.
3. **Beszédes név** — A teszt neve kimondja, mit ellenőriz.
4. **Szándékos adat** — A büntetés itt elég nagy, hogy megfordítsa a sorrendet.
5. **Konkrét elvárás** — Leírt eredmény, nem az, amit a függvény épp visszaad.
6. **Határeset** — Üres lista: a legegyszerűbb, mégis gyakran elfelejtik.

```
✓ rank > a büntetés megfordíthatja a sorrendet
✓ rank > üres listára üres listát ad
```

2 sikeres

Minden számolásra

Ár, pontszám, időpont, készlet.

Minden új szabályra

Főleg, ha az utolsó pillanatban jön.

Minden élesben talált hibára

Hogy ugyanaz ne jöjjön vissza.

Futtasd gyakran

Másodpercek, bármikor mehet.

JÓ KÉRDÉST TEGYÉL FEL.

Ugyanaz a függvény, két teszt. Mindkettő zöld. Csak az egyik venné észre, ha valaki (vagy az AI) kiveszi a büntetést a számolásból.

Gyenge teszt

nem veszi észre

```
const results = [
  { name: "A", timeSec: 52, penaltySec: 0 },
  { name: "B", timeSec: 58, penaltySec: 2 },
];
expect(rank(results).map((r) => r.name))
  .toEqual(["A", "B"]);
```

helyes kód	büntetés nélkül
A, B ✓	A, B ✓

A gyorsabb versenyző a büntetéssel együtt is előrébb van. Ha a büntetés kimarad a kódból, ez a teszt **továbbra is zöld.**

Erős teszt

észreveszi

```
const results = [
  { name: "A", timeSec: 52, penaltySec: 10 },
  { name: "B", timeSec: 58, penaltySec: 0 },
];
expect(rank(results).map((r) => r.name))
  .toEqual(["B", "A"]);
```

helyes kód	büntetés nélkül
B, A ✓	A, B ×

Itt a büntetés megfordítja a sorrendet. Ha kimarad a kódból, a teszt **piros lesz.** Ilyen kellett volna a döntő napján.

Prompt Claude Code-hoz és Cursorhoz

A <> közötti részt cseréld ki

```
Írj unit teszteket a <rank() függvényhez a src/ranking.ts fájlban> Vitesttel.

Mielőtt kódot írsz, sorold fel a teszteseteket egy-egy mondattal: mit
ellenőriz, és miért fontos. Utána írd meg őket.

Szabályok:
- Minden üzleti szabályhoz legyen olyan eset, ahol a hibás megvalósítás
  (pl. ha a büntetés kimarad) MÁS eredményt adna, mint a helyes.
- Vedd bele a határeseteket: üres lista, egy elem, nulla, negatív érték,
  holtverseny, nagyon nagy szám.
- A tesztelt függvényt ne mockold. Csak a külső dolgokat (idő, hálózat).
- Konkrét elvárt értékeket írj, ne azt, amit a függvény épp visszaad.
- Bizonyítsd, hogy a tesztek védenek: rontsd el szándékosan a függvényt,
  futtasd le, mutasd meg a piros eredményt, aztán állítsd vissza és
  futtasd zölden.
- Ha egy teszt piros, ne a tesztet írd át, hanem szólj, mi a gond.
- A végén egy mondatban írd le, mit NEM ellenőriznek ezek a tesztek.
```

Előbb piros

Ha soha nem láttad elbukni, nem tudod, hogy működik.

Határesetek

Üres, nulla, holtverseny, egyetlen elem.

Ne mockold

Amit tesztelsz, az igazából fusson le.

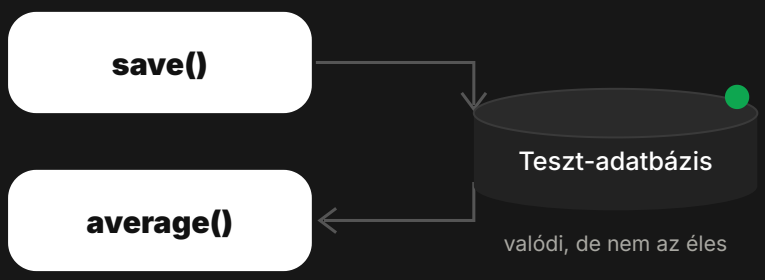
Beszédes név

A teszt neve mondja meg, mit ellenőriz.

02 INTEGRÁCIÓS TESZT.

Több rész együtt: a kódod és egy valódi adatbázis vagy szolgáltatás. Itt derül ki, hogy a külön-külön jó darabok együtt is működnek-e.

másodperc | valódi teszt-adatbázissal | pytest



1. üres adatbázis
2. tesztadat mentése
3. ellenőrzés
4. eldobás

Minden teszt üres adatbázissal indul, és a végén eldobja.

Mit fog meg

- Hibás lekérdezés, rossz szűrés
- Keveredő adatok két hely között
- Mást kapsz vissza, mint amit mentettél
- Sorlimit, mezőhossz, jogosultság

Négy szabály

- **Teszt-adatbázis:** külön a fejlesztéshez, sosem az éles.
- **Tiszta lappal:** minden teszt ismert állapotból indul.
- **Nagy adag:** teszteld a limitek felett is.
- **Két felhasználó:** egymás adatát ne lássák.

KÓD ÉS ADAT, EGYÜTT.

Integrációs

Értékelések mentése és átlaga, valódi SQLite adatbázissal. Minden teszt friss, üres adatbázist kap a memóriában.

Példa: értékelések mentése és átlaga, Python és pytest

pytest

store.py

```
import sqlite3

def init(conn):
    conn.execute("CREATE TABLE submissions (restaurant TEXT, score INTEGER)")

def save(conn, restaurant, score):
    if not 1 <= score <= 5: (1)
        raise ValueError("score must be between 1 and 5")
    conn.execute("INSERT INTO submissions VALUES (?, ?)", (restaurant, score))

def average(conn, restaurant):
    row = conn.execute(
        "SELECT AVG(score) FROM submissions WHERE restaurant = ?", (restaurant,) (2)
    ).fetchone()
    return row[0]
```

test_store.py

```
import sqlite3, pytest
from store import init, save, average

@pytest.fixture
def conn():
    c = sqlite3.connect(":memory:")   # minden teszt új, üres adatbázist kap (3)
    init(c)
    yield c
    c.close()

def test_average_counts_only_own_restaurant(conn):
    save(conn, "buda", 5)
    save(conn, "buda", 3)
    save(conn, "pest", 1)               # ez nem számíthat bele (4)
    assert average(conn, "buda") == 4 (5)
```

1 Őrkapu
Hibás pontszám be se juthat.

2 A szűrés
Ha kimarad, a teszt piros.

3 Tiszta lap
Valódi adatbázis, de eldobható.

4 Zavaró adat
Egy másik étterem 1-es pontja.

5 Elvárás
(5 + 3) / 2 = 4. A pesti 1-es nélkül.

következő

Következő teszt, amit érdemes kérni: hibás pontszám (például 6) nem kerül be az adatbázisba, és az átlag nem változik.

A CSENDES HIBA A LEGROSSZABB.

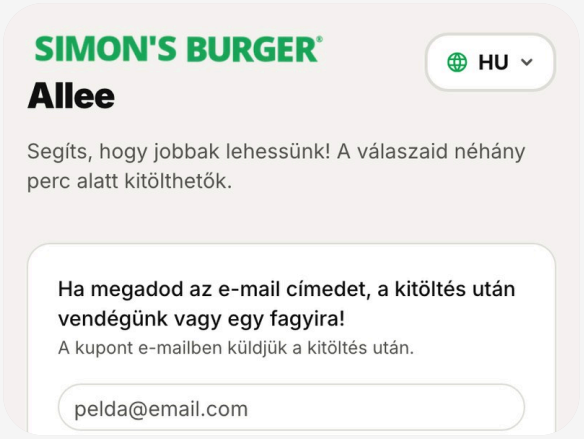
Az a hiba, ami hibaüzenetet dob, hamar kiderül. Az, amelyik csak kevesebb adatot mutat, hetekig is észrevétlen maradhat.



Tapasztalatból
A vendégelégedettségi kérdőívünk

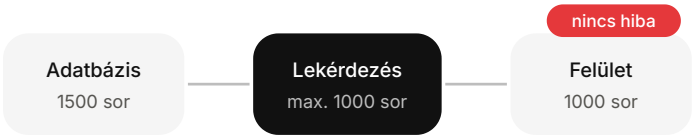
Az adatbázis-szolgáltatónk egy lekérdezésre alapból legfeljebb 1000 sort ad vissza, hibaüzenet nélkül. Amíg kevés volt a kitöltés, ez nem látszott. Aztán a legforgalmasabb éttermünknél a felület a kitöltéseknek csak a felét mutatta.

Unit teszt ezt soha nem látja, mert ott nincs adatbázis. Egy integrációs teszt, ami 1500 sort ment el, és mind az 1500-at visszavárja, piros lett volna.



A kérdőív, élesben. A vendég a blokkon lévő QR-kóddal jut ide.

Így vágja le csendben



Az 1500 soros teszt itt bukik el, és pont ezt akarod.

Ennyi látszott

57 a 113 kitöltésből

A javítás után mind a 113 megjelent.

Prompt Claude Code-hoz és Cursorhoz

A <> közötti részt cseréld ki

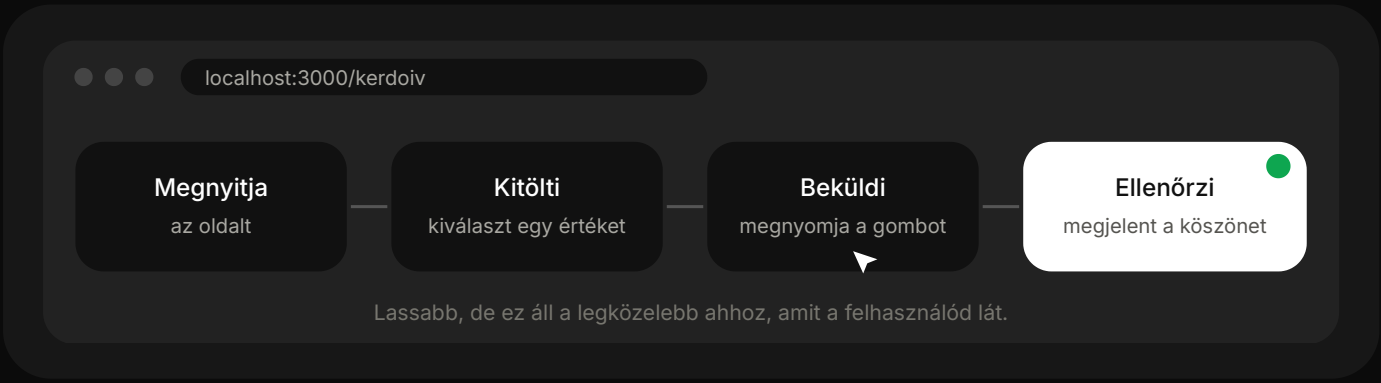
```
Írj integrációs teszteket a <kitöltések mentéséhez és lekérdezéséhez>.

Környezet:
- Valódi adatbázissal teszteld, ne mockkal: <helyi Postgres / Supabase
  branch / SQLite memóriában>. Az éles adatbázishoz SOHA ne nyúlj.
- Minden teszt üres, ismert állapotból induljon, és utána takarítson.

Amit mindenképp ellenőrizz:
- Amit elmentek, pontosan az jön vissza (ékezetes szöveg is).
- Két étterem vagy két felhasználó adata nem keveredik.
- Hibás adat nem kerül be, és érthető hibát kapok.
- Sok adat: ments el 1500 sort, és ellenőrizd, hogy mind visszajön.
  (Sok szolgáltatás csendben levágja a választ egy limitnél.)
- Ha van jogosultságkezelés, egy másik felhasználó ne lássa az adatot.

Mutasd meg, hogy legalább egy teszt elbukik, ha a lekérdezésből
kiveszem a szűrést, aztán állítsd vissza.
```

03

END-TO-END TESZT.

Úgy használja az appot, mint egy ember: megnyitja a böngészőt, kattint, gépel, és megnézi, mi jelenik meg.

másodpercek | 3–5 teszt a fő folyamatokra | Playwright



Mit fog meg

- Beragadt gomb, ami nem csinál semmit
- Rossz átirányítás, üres oldal
- Bejelentkezés és kijelentkezés hibái
- Ami mobil méreten eltűnik vagy kilóg

Mit nem

- Minden számolási határesetet: arra lassú és drága
- A ritka utakat: e2e-ből csak a fő folyamatokra írj

Jó kiindulás: 3–5 e2e teszt, a legfontosabb dolgokra, amiért az appod létezik.

KATTINTS ÚGY, MINT A VENDÉG.

End-to-end

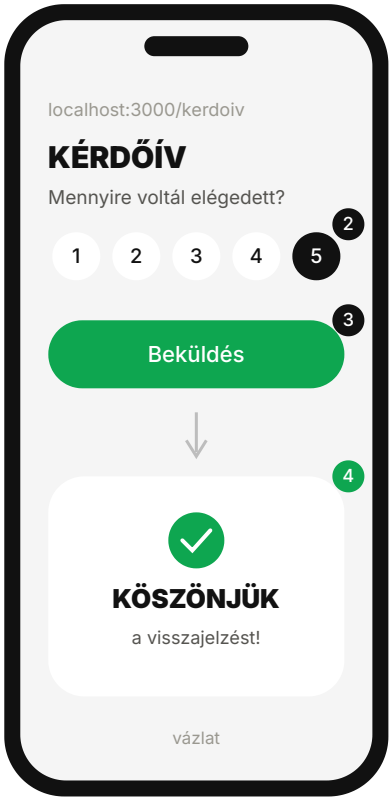
A teszt ugyanazt csinálja, amit egy vendég a telefonján: kiválaszt egy értéket, beküldi, és megnézi, megjelent-e a köszönet.

Példa: kérdőív kitöltése, Playwright

npx playwright test

tests/e2e/kerdoiv.spec.ts

```
import { test, expect } from "@playwright/test";

test("a vendég be tudja küldeni a kérdőívet",
  async ({ page }) => {
  await page.goto("http://localhost:3000/kerdoiv"); (1)

  // úgy keresünk elemet, ahogy egy ember látja:
  // szerep és felirat alapján
  await page.getByRole("radio", { name: "5" })
    .check(); (2)
  await page.getByRole("button", { name: "Beküldés" })
    .click(); (3)

  // nem az a kérdés, hogy elment-e a kérés,
  // hanem hogy a vendég mit lát
  await expect(page.getByText("Köszönjük")).toBeVisible(); (4)
});
```



1 Megnyitja
A helyben futó appot, nem az éleset.

2 getByRole
Túléli a dizájncserét, a CSS-osztály nem.

3 Kattint
Nincs fix várakozás, a Playwright magától vár.

4 Amit a vendég lát
Ez az egy sor a teszt lényege.

```
$ npx playwright test
✓ kerdoiv.spec.ts > a vendég be tudja küldeni a kérdőívet
```

1 sikeres

Szerep szerint
getByRole túléli a dizájncserét.

Nincs fix várakozás
A sleep lassú és ingadozó.

Mobilon is
A vendégeid többsége telefonról jön.

Kép hibánál
Azonnal látod, mi ment félre.

HAGYD, HOGY AZ ÜGYNÖK KATTINTSON.

End-to-end

Claude Code és Cursor is tud böngészőt vezérelni, például a Playwright MCP-vel vagy a Claude in Chrome bővítménnyel. Először kattintsa végig ő, aztán írja meg ugyanezt ismételhető tesztként.



Tapasztalatból
„Beküldés…"

A kérdőívünk beküldése után a gomb „Beküldés…" állapotban ragadt, pedig az adat rendben elmentődött. A vendég azt hitte, nem ment el, és újrapróbálta vagy feladta.

A mentés tesztje zöld volt, hiszen a mentés működött. Egy e2e teszt, ami a köszönő szöveget várja, piros lett volna.

a mentés tesztje	amit a vendég látott
adat elmentve	Beküldés…
zöld	e2e: piros

Két lépés, két haszon

1 **Kattintsa végig az ügynök.** Gyorsan megtalálja, ami szemmel látható: üres oldal, kilógó gomb, konzolhiba.

→

2 **Írja meg tesztként.** Amit egyszer végigkattintott, az ezután minden változtatásnál magától lefut, ügynök nélkül is.

Prompt Claude Code-hoz és Cursorhoz

A <> közötti részt cseréld ki

```
Indítsd el az appot helyben, és a böngészőben kattintsd végig ezt a
folyamatot úgy, ahogy egy felhasználó tenné: <regisztráció, kosárba
tétel, fizetés előtti összesítő>. Asztali és mobil méretben is.

Írd le, mit láttál minden lépésnél, és ami furcsa (lassú, kilóg,
konzolhiba), azt külön listában. Még ne javíts semmit.

Utána írd meg ugyanezt Playwright tesztként a tests/e2e mappába:
- Elemet szerep és látható felirat alapján keress (getByRole, getByText),
  ne CSS-osztály alapján.
- Ne várj fix ideig (nincs waitForTimeout), a Playwright magától vár.
- A végén azt ellenőrizd, amit a felhasználó lát, ne csak a kérést.
- Teszteld a sikertelen utat is: hibás adat, üres mező.
- Hiba esetén mentsen képernyőképet.
- Futtasd le háromszor egymás után. Ha egyszer is elbukik, keresd meg,
  miért ingadozik, és ne tegyél bele újrapróbálkozást, hogy elfedd.
```

04

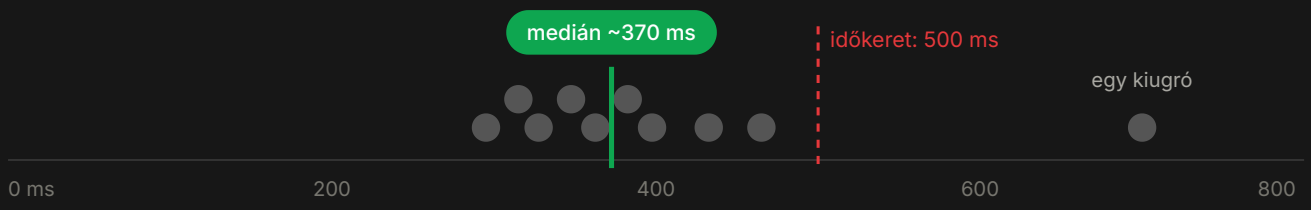
TELJESÍTMÉNY-TESZT.

Nem az a kérdés, hogy működik-e, hanem hogy elég gyorsan-e. A lassú app is hibás app, csak a felhasználó nem hibaüzenetet lát, hanem bezárja.

perc | medián, nem egy szám | pytest, k6, Locust

10 mérés ugyanarra az oldalra

szemléltető számok



Egyetlen mérés semmit nem mond. A középső érték igen, és az időkerettel összevetve el is bukhat.

Mit fog meg

- Sok egymás utáni lekérdezés, ahol egy is elég lenne
- Rossz régió: kód és adat két kontinensen
- Lassulás, ahogy nő az adat
- Túl nagy képek, túl sok betöltött adat

Két fajtája van

Sebességmérés: egy kérés vagy oldal mennyi idő alatt jön meg. Ezt bármikor futtathatod.

Terheléses teszt: mi történik, ha egyszerre százan jönnek (például k6 vagy Locust eszközzel). Ezt csak teszt-környezeten futtasd, élesen ne.

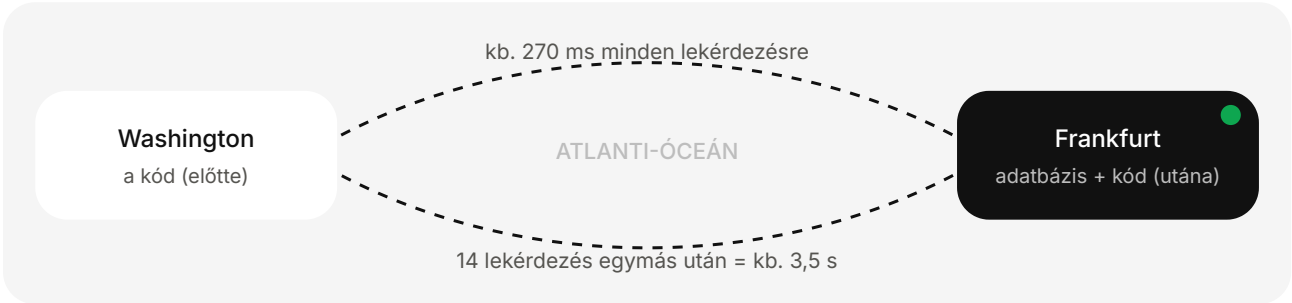
WASHINGTON, FRANKFURT.

Egy lassú dashboard, és egy ok, amire senki nem gondolt: a kód és az adat két kontinensen volt.



Tapasztalatból
A kód Washingtonban volt, az adatbázis Frankfurtban

A vendégelégedettségi felületünk lassú volt, és először az adatmennyiségre gyanakodtunk. Csakhogy 93 kitöltés semmi. A mérés mást mutatott: a szerverfüggvények Washingtonban futottak, az adatbázis Frankfurtban van. Minden lekérdezés átment az óceánon, egyenként kb. 270 ms alatt, és a dashboard 14 lekérdezést futtatott egymás után.



93 kitöltés
Ennyi adat nem lehet lassú. Nem is az volt.

270 ms
Egyetlen lekérdezés ideje az óceánon át.

14 lekérdezés
Egymás után, egy dashboard betöltésénél.

Egy oldal betöltése

Előtte 0,99–1,27 s

Utána 0,30–0,51 s

A javítás egyetlen sor volt: a függvényeket az adatbázis mellé, Frankfurtba költöztettük. A sötétebb rész a legjobb, a világosabb a leglassabb mért érték.

Eredmény

~3x gyorsabb

Nem kellett hozzá új kód, csak mérés. Ha a gyorsításba kezdünk mérés nélkül, rossz helyen keresünk.

ELŐBB MÉRJ, AZTÁN JAVÍTS.

Teljesítmény

Egyetlen mérés semmit nem mond: a hálózat ingadozik, a gép épp mást is csinál. Mérj többször, és a középső értéket (mediánt) figyeld.

Példa: időkeret egy végpontra, Python és pytest

pytest

test_speed.py

```
import statistics, time, urllib.request

URL = "http://localhost:3000/api/health"   # a saját végpontod

def test_median_response_is_under_500_ms():
    times = []
    for _ in range(10): (1)
        start = time.perf_counter()
        urllib.request.urlopen(URL, timeout=5).read()
        times.append((time.perf_counter() - start) * 1000) (2)
    assert statistics.median(times) < 500, f"medián: {statistics.median(times):.0f} ms" (3)
```

vercel.json (a mi javításunk, verziókövetve)

```
{ "regions": ["fra1"] } (4)
```

1 10 mérés
Ugyanúgy, egymás után.

2 Ezredmásodperc
Pontos óra, ms-ra váltva.

3 Időkeret
A medián 500 ms fölött: piros.

4 Egy sor
Kód az adat mellé, Frankfurtba.

Prompt Claude Code-hoz és Cursorhoz

A <> közötti részt cseréld ki

```
<A dashboard oldal> lassú. Még ne javíts semmit, először mérj.

1. Mérd meg az oldal betöltését legalább 10-szer. Add meg a mediánt
   és a leglassabbat. Előtte nézd meg, nem terhelt-e a gép.
2. Bontsd fel: hány lekérdezés vagy API-hívás fut, mennyi idő
   egyenként, egymás után vagy párhuzamosan futnak-e.
3. Nézd meg, melyik régióban fut a kód és melyikben az adatbázis.
4. Írd le a 3 legvalószínűbb okot a mérések alapján, a legnagyobbal
   kezdve. Csak utána javasolj javítást, és várd meg a jóváhagyásom.
5. A javítás után mérj újra ugyanígy, és tedd egymás mellé a számokat.
6. Írj egy tesztet, ami elbukik, ha a medián <500 ms> fölé megy.
```

Medián, nem egy szám
Legalább 10 mérés, ugyanúgy.

Nyugodt gép
Ha más is fut, a mérés hamis.

Időkeret ms-ban
Ami fölötte van, az piros.

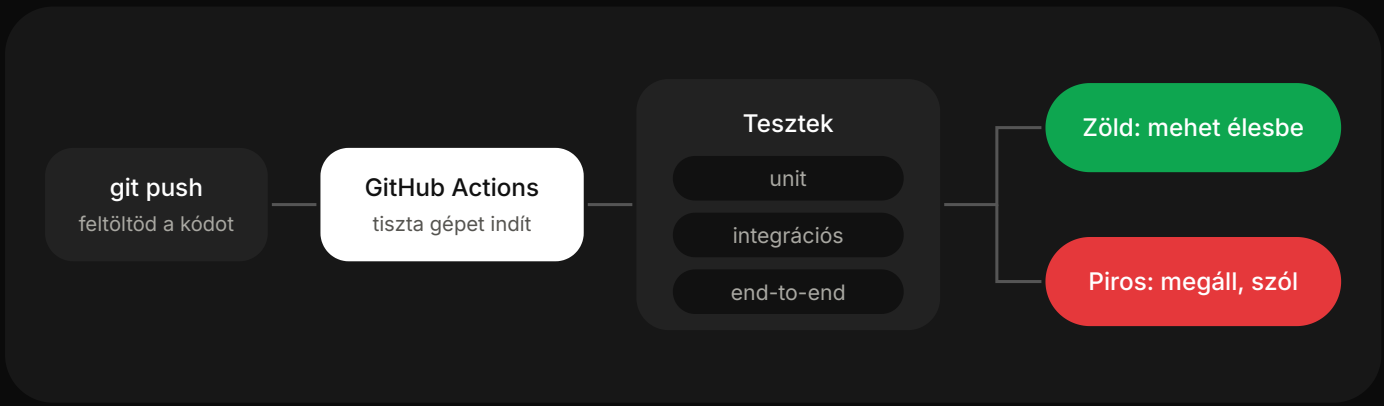
Előtte és utána
Javítás csak számokkal.

05

CI/CD.

A teszt csak akkor ér valamit, ha le is fut. A CI (folyamatos integráció) minden feltöltésnél lefuttatja az összeset helyetted. A CD (folyamatos élesítés) csak akkor viszi ki az új verziót, ha minden zöld.

automatikus · minden push-ra · GitHub Actions



Amit a CI ad neked

- **Tiszta gép.** Nem számít, mi van a te gépeden telepítve. Ha ott átmegy, máshol is fog.
- **Az AI munkáját is ellenőrzi.** Ha egy ügynök tölt fel kódot, az is ugyanazon a kapun megy át.
- **Nem lehet elfelejteni.** Nem neked kell eszedbe jutnia, hogy lefuttasd.
- **Ingyenes** kisebb projektekhez, a GitHub havi ingyenes kerettel.

EGY FÁJL, ÉS A GÉP FIGYEL.

Egy minimális GitHub Actions workflow. Feltöltöd a repóba, és onnantól minden push után magától lefut.

Példa: egy minimális workflow

.github/workflows/tests.yml

```yaml
name: tests

on: [push, pull_request] 1  # minden push és pull request

jobs:
  test:
    runs-on: ubuntu-latest 2
    steps: 3
      - uses: actions/checkout@v5
      - uses: actions/setup-node@v5
        with:
          node-version: 22
          cache: npm
      - run: npm ci
      - run: npm test 4   # unit és integrációs tesztek
      - run: npx playwright install --with-deps chromium
      - run: npx playwright test 5   # end-to-end tesztek
      - uses: actions/upload-artifact@v4
        if: failure() 6   # csak ha valami elbukott
        with:
          name: playwright-report
          path: playwright-report/
```

1. **on: mikor fusson**
   Itt minden feltöltésnél és pull requestnél.
2. **runs-on: milyen gépen**
   Egy friss Linux gép, amit a GitHub ad.
3. **steps: mit csináljon**
   Sorban, ha egy lépés elbukik, megáll.
4. **A kapu**
   Ha itt piros, a többi le sem fut.
5. **E2E**
   Valódi böngészővel, a CI gépén.
6. **Riport**
   Csak hibánál, letölthető.

Ha az e2e teszted a saját appodat nyitja meg, a Playwright configban a **webServer** beállítás indítja el a tesztek előtt. Ezt is megkérheted az AI-tól a következő oldal promptjával.

Az Actions fülön ✓ checkout ✓ npm ci ✓ npm test × playwright test riport feltöltve

vázlat

PIROSSAL NEM MEGY ÉLESBE.

A workflow magától lefut, de még nem állít meg semmit. Ahhoz, hogy piros teszttel tényleg ne lehessen élesíteni, két beállítás kell.

Bekapcsolás lépésről lépésre

1. Tedd fel a **tests.yml** fájlt, és nézd meg a GitHubon az **Actions** fülön, hogy lefut.
2. Rontsd el szándékosan egy teszt elvárt értékét, töltsd fel, és nézd meg, hogy piros lesz. Aztán állítsd vissza.
3. **Settings → Branches:** védd a main ágat, és kapcsold be, hogy csak sikeres tesztekkel lehessen bele olvasztani.
4. Az éles verzió csak a main ágból menjen ki (Vercelnél és Netlifynél ez az alapbeállítás).

Így néz ki bekapcsolva

vázlat

tests / test
1 teszt elbukott

main védett ág
sikeres teszt kell

Beolvasztás nem lehetséges

Prompt Claude Code-hoz és Cursorhoz

A <> közötti részt cseréld ki

```
Állíts be GitHub Actions CI-t ehhez a projekthez.

- Hozd létre a .github/workflows/tests.yml fájlt, ami minden push-ra és
  pull requestre lefuttatja az összes tesztet: <npm test, és a Playwright
  e2e teszteket>.
- Előtte nézd meg a package.json-t, és pontosan azokat a parancsokat
  használd, amik helyben is működnek. Futtasd le őket helyben is.
- Ha az e2e tesztekhez futnia kell az appnak, állítsd be a Playwright
  config webServer részét, hogy a teszt előtt elinduljon.
- Titkot (API-kulcs, jelszó) SOHA ne írj a fájlba. Ha kell, nevezd meg,
  milyen néven vegyem fel GitHub Secretsként, és onnan hivatkozz rá.
- Ha egy teszt elbukik, a Playwright riportja kerüljön fel artifactként.
- A végén írd le lépésről lépésre, hogyan kapcsolom be a main ág
  védelmét, hogy piros teszttel ne lehessen beolvasztani.
```

Minden push
Nem csak a mainre, minden ágra.

Titok a Secretsbe
Soha nem a YAML-be.

Riport hibánál
Letölthető képekkel együtt.

Legyen gyors
10 perc alatt fusson le.

HOGYAN KEZDJ HOZZÁ MA.

Nem kell egyszerre mind az öt szint. Kezdd ott, ahol most tartasz, és lépj egyet feljebb, ha az már megy.

Kezdő — ma, kb. 30 perc
Haladó — ezen a héten
Profi — ebben a hónapban

Kezdő

ma, kb. 30 perc

1. Válaszd ki az appod **egyetlen számolását**, ami ha elromlik, pénzbe vagy bizalomba kerül: ár, pontszám, időpont, készlet.
2. Másold be a **10. oldal promptját**, és írass rá unit teszteket.
3. Kérd meg, hogy **rontsa el szándékosan**, és nézd meg a piros eredményt a saját szemeddel.
4. Kérdezd meg tőle: „Milyen adaton futott a teszt, és mit nem ellenőriz?”

Haladó

ezen a héten

1. Hozz létre egy **teszt-adatbázist**, és írass integrációs teszteket a mentésre és a lekérdezésekre.
2. Írass **3 e2e tesztet** a fő folyamatokra, mobil méretben is.
3. Írd be a szabályokat a projekt **CLAUDE.md** (Cursorban .cursor/rules) fájljába, hogy minden munkamenet tudja.

```
# CLAUDE.md
## Tesztek
- Minden új funkcióhoz írj tesztet,
  és futtasd le, mielőtt készneknek mondod.
- Legyen olyan eset, ahol a rossz
  megoldás más eredményt adna.
- Éles adatbázison soha ne futtass tesztet.
- Ha egy teszt piros, ne a tesztet írd át,
  hanem szólj.
```

Profi

ebben a hónapban

1. Kapcsold be a **GitHub Actions** CI-t és a main ág védelmét (21–22. oldal).
2. Adj **időkeretet** a legfontosabb oldalakra, és mérd őket minden élesítés után.
3. Ahol a kockázat nagy (pénz, élő esemény), legyen **valós körülmények közti próba** is, ne csak unit teszt: valódi eszközön, valódi adattal, filléres tranzakcióval.
4. Minden élesben talált hibából legyen **egy új teszt**, ami ugyanezt a hibát legközelebb megfogja.

Mai feladat

MELYIK AZ A FUNKCIÓ AZ APPODBAN, AMITŐL A LEGJOBBAN FÉLSZ, HOGY EGYSZER CSENDBEN ELROMLIK? AZ AZ ELSŐ JELÖLT.

PUSKA.

Az egész útmutató egy oldalon. Tedd a monitor mellé.

Szint	Kérdés	Eszköz	Fő szabály
UNIT	Egyetlen függvény jól számol?	Vitest, pytest `npx vitest`	Legyen eset, ahol a rossz megoldás más eredményt adna.
INTEGRÁCIÓS	A kód és az adatbázis együtt is jól működik?	pytest + teszt-adatbázis `pytest`	Valódi adatbázis, sosem az éles. Teszteld 1500 sorral is.
END-TO-END	Egy ember végig tud menni a fő folyamaton?	Playwright `npx playwright test`	getByRole, nincs fix várakozás, azt ellenőrizd, amit a felhasználó lát.
TELJESÍTMÉNY	Elég gyors, és gyors marad, ha nő az adat?	pytest, k6, Locust	Legalább 10 mérés, medián, időkeret ms-ban.
CI/CD	Minden feltöltésnél lefut az összes?	GitHub Actions `.github/workflows/tests.yml`	Piros teszttel nem lehet a main ágba olvasztani.

Mondatok, amik minden promptomba bekerülnek

- Mielőtt kódot írsz, sorold fel a teszteseteket.
- Rontsd el szándékosan, mutasd meg a piros eredményt.
- Ha egy teszt piros, ne a tesztet írd át, hanem szólj.
- Az éles adatbázishoz SOHA ne nyúlj.
- Még ne javíts semmit, először mérj.
- Titkot SOHA ne írj a fájlba.
- A végén írd le, mit NEM ellenőriznek a tesztek.

Egy teszt három része

1 Előkészítés

2 Végrehajtás

3 Ellenőrzés

Határesetek

üres · egy elem · nulla · negatív · holtverseny · nagyon nagy szám · ékezet · 1000 sor felett

Két kérdés az AI-nak, minden „kész" után

MILYEN ADATON FUTOTT A TESZT? MIT NEM ELLENŐRIZ?

ÉLESÍTÉS ELŐTT.

Nyomtasd ki, vagy tedd a monitor mellé. Nem kell mindig minden pont, de mindegyiknél tudd, miért hagyod ki.

Projekt _____ Dátum _____ Ellenőrizte _____

Logika

- ◯ Van teszt a legfontosabb számolásokra.
  Ár, pontszám, sorrend, dátum.
- ◯ Minden szabálynál van eset, ahol a rossz megoldás más eredményt adna.
- ◯ Láttam a tesztet elbukni, amikor szándékosan elrontottam a kódot.
- ◯ Megvannak a határesetek.
  Üres, nulla, negatív, holtverseny, nagyon nagy.

Adat

- ◯ Ami elmentődik, pontosan az jön vissza, ékezettel együtt.
- ◯ Két felhasználó vagy két hely adata nem keveredik.
- ◯ Sok adattal is teszteltem, a limitek felett.
  Például 1500 sor.
- ◯ A tesztek nem az éles adatbázison futnak.

Felhasználó

- ◯ A fő folyamatokra van e2e teszt, asztalon és mobilon.
- ◯ A végén azt ellenőrzöm, amit a felhasználó lát.
- ◯ A hibás utat is kipróbáltam: rossz adat, üres mező, nincs net.

Sebesség

- ◯ Megmértem a fő oldalakat, legalább 10-szer, mediánnal.
- ◯ A kód és az adatbázis ugyanabban a régióban fut.
- ◯ Van időkeret, és a teszt piros, ha átlépjük.

Automatika

- ◯ Minden push-ra lefut az összes teszt (CI).
- ◯ Piros teszttel nem lehet a main ágba olvasztani.
- ◯ Élesítés után megnézem, hogy él-e.
  Oldal betölt, egy próbakérés sikerül.

Biztonság

- ◯ Adatbázis-módosítás előtt van friss mentés, és tudom, hol.
- ◯ Nincs titok a kódban, a promptban és a mentett engedélyekben.
- ◯ Törlést és éles adatmódosítást ember hagy jóvá.
- ◯ Megkérdeztem: ha most minden elromlik, honnan állítom vissza?

A végén

- ◯ Megkérdeztem az AI-t: „Mit NEM ellenőriznek a tesztjeid?”

BIZTONSÁG, AMIT A TESZT NEM LÁT.

A teszt azt mondja meg, hogy a kód jó-e. Azt nem, hogy mi történik, ha az AI rossz parancsot futtat. Ezeket a szokásokat a saját munkámban használom.

Mentés minden adatbázis-módosítás előtt

mielőtt az AI hozzányúl

Mielőtt az AI töröl, migrál vagy újrahúz egy adatbázist, készüljön helyi mentés. Csak te olvashasd, és írd le a projektnaplóba, hol van.

Én egy régi teszt-adatbázist így ürítettem ki egy új projekt kedvéért, és egy nagy séma-átalakítás előtti napon is mentettem a kitöltéseket.

```
# a jelszó env-ből jön, nem a parancsból
pg_dump "$DATABASE_URL" -f mentes-2026-11-12.sql (1)
chmod 600 mentes-2026-11-12.sql (2)
```

(1) mentés (2) csak te olvashatod



Tapasztalatból
Titok soha a promptban

Jelszót, API-kulcsot ne másolj a chatbe, és ne írj be parancsba. Ami a promptban van, az a beszélgetés része marad.

Az engedélylista is emlékszik. Ha egy parancsra azt mondod, „mindig engedélyezd”, a teljes parancs elmentődik, benne a jelszóval, ha ott volt. A saját gépemen is találtam ilyet. Ilyenkor a kulcsot le kell cserélni, és onnantól csak env-fájlból dolgozni.

Visszafordíthatatlan lépést ember hagy jóvá

- Törlést és tömeges módosítást az éles adatbázison én futtatok, nem az ügynök.
- Az AI mindent helyben készít elő, és élesítés előtt **egyetlen jóváhagyást** kér.
- Ügyféladatnál előbb próbafuttatás, ami csak kiírja, mit csinálna. Utána élesben.
- Ha az AI valamit nem enged, nem kerülöm meg: kézzel futtatom.

Írd le a csapdákat

A projekt CLAUDE.md fájljának legelejére. Ami nincs leírva, azt az ügynök nem tudja, és a következő munkamenet újra elköveti.

```
# CLAUDE.md
## Ezeket soha
- Éles adatbázison ne törölj és ne módosíts
  tömegesen. Írd meg az SQL-t, és add át nekem.
- Mentés nélkül ne migrálj.
- Titkot ne írj ki, és ne tegyél parancsba.
- Élesítés előtt kérj jóváhagyást.
```

Éles és teszt külön

Külön adatbázis a fejlesztéshez. Amit élesnek hívnak (prod, production), azt az ügynök is érzékenynek kezelje.

Nézd át az engedélyeket

Havonta egyszer nézd végig, mire mondtál „mindig engedélyezd”-et, és vedd ki a veszélyeseket.

Kérdés magadnak, minden nagyobb változtatás előtt

HA MOST MINDEN ELROMLIK, HONNAN ÁLLÍTOD VISSZA, ÉS MENNYI IDŐ ALATT?



KÖSZÖNÖM, HOGY ELOLVASTAD.

Remélem, a következő funkciódnál már te kérdezed meg az AI-tól: milyen adaton futott a teszt?



Semsei Izsák
Fejlesztő, Simon's Burger

LinkedIn linkedin.com/in/semseiizsak

A Simon's Burger fejlesztője vagyok. Egy év alatt a Wix-oldalakról, Google Formsból és táblázatokból saját rendszerekre álltunk át: weboldal, vendégelégedettségi kérdőív a blokkokon lévő QR-kóddal, tréningrendszer, címkenyomtató az éttermekben, és most egy közös platform az egész cégnek. Szinte mindet Claude Code-dal építettem.

A tesztelést nem könyvből tanultam, hanem abból, amikor valami élesben nem úgy működött, ahogy a gépemen. Ebből lett ez az útmutató.

HA HASZNOS VOLT, MEGHÍVHATSZ EGY KÁVÉRA.

Az útmutató ingyenes, és az is marad. Ha szeretnéd támogatni, hogy legyen folytatás, itt megteheted. Ha nem, az is teljesen rendben van.

Meghívlak egy kávéra

buymeacoffee.com/semseizsak

Küldd tovább

Ha ismersz valakit, aki AI-al épít appot, nyugodtan add oda neki.

Írj, ha elakadtál

LinkedInen szívesen válaszolok, ha egy tesztnél nem tudod, hogyan tovább.

Mondd el, mi hiányzott

A visszajelzésekből lesz a következő kiadás.

A TESZT NEM HAZUDIK. CSAK ARRA VÁLASZOL, AMIT KÉRDEZEL.