DarwinAI
MESTERSÉGES EVOLÚCIÓ
Oktatás – Tanácsadás – Fejlesztés



VIBE CODING GYORSTALPALÓ · TANKÖNYV

Az ötlettől a működő szoftverig – kódolás nélkül

Gyakorlati tankönyv felnőtteknek, akik saját eszközt, alkalmazást vagy rendszert építenének AI-val – programozói tudás nélkül.

DarwinAI · Mesterséges evolúció 1.0 · 2026. október darwinai.hu/vibe-coding

MIELŐTT ELKEZDED

Hogyan használd ezt a könyvet?

Ez a könyv azoknak szól, akik már tudnak beszélgetni egy AI-chattel, és most szeretnék megtanulni, hogyan lesz egy ötletből **működő szoftver** – anélkül, hogy programozóvá válnának.

A vibe coding (AI-val, a saját szavaiddal végzett szoftverfejlesztés) nem a kódolásról szól, hanem a **gondolkodásról és a folyamatról**: mit akarsz, hogyan írod le, hogyan adod át az AI-nak, és hogyan ellenőrzöd, amit kaptál. Ezt tanítja ez a gyorstalpaló – lépésről lépésre, valódi példákon, a DarwinAI saját munkamódszere alapján.

Minden leckéhez tartozik egy rövid animációs videó a DarwinAI YouTube-csatornáján. A videó és a lecke ugyanazt tanítja: a videó gyors áttekintés, a könyv a részletes, visszalapozható változat.

Sorban haladj
A leckék egymásra épülnek. Az első kettő után már érteni fogod a teljes folyamatot.

Előbb videó, aztán könyv
A videó 3–4 perc alatt megmutatja a lényeget; a lecke elmélyíti.

Csináld közben
Minden lecke végén van egy „Próbáld ki" feladat. Ezektől lesz tudás az olvasottból.

A részletek hátul
A technikai magyarázatok a mellékletekbe kerültek. Csak akkor lapozz oda, ha érdekel.

Kinek szól?

Laikusoknak és más szakmák profijainak: vállalkozóknak, vezetőknek, tanároknak, orvosoknak, mérnököknek, irodai munkatársaknak – bárkinek, aki magának vagy a csapatának szeretne eszközt, alkalmazást, rendszert építeni. Előfeltétel csak egy: tudj egy AI-chatnek (ChatGPT, Claude) érthetően feladatot adni.

Mire lesz szükséged?

- Egy számítógép (Mac vagy Windows) – a telefon a tanuláshoz kevés.
- Egy AI-chat (ChatGPT vagy Claude) – a tervezéshez.
- Egy coding agent előfizetés (például Claude Code vagy OpenAI Codex) – az építéshez. A 3. lecke segít választani.
- Egy ötlet. Kicsi is lehet – sőt, az a legjobb.

A DARWINAI ONLINE ANYAGAI

A könyv a darwinai.hu Vibe Coding szekciójára épül: **darwinai.hu/vibe-coding** – ott találod a pipálható folyamatot, a szabálykönyvet, a szabály-összeállítót és a letölthető ellenőrzőlistákat.

Tartalomjegyzék

Hogyan használd ezt a könyvet? 1

I. RÉSZ Alapok

1 **Mi a programozás?** A program is egy nyelv – és az AI beszéli 4
2 **Mi a vibe coding?** Te mondod meg, mit – az AI megcsinálja, hogyan 6
3 **Az eszközök** Sima chat vagy coding agent – és melyik platform? 9
4 **Modell és erőfeszítés** Melyik „agy" dolgozzon – és mennyit gondolkodjon? 11

II. RÉSZ Tervezés

5 **A folyamat térképe** Kilenc lépés az ötlettől a működő programig 14
6 **Vázlatból specifikáció** A saját szavaidtól az átadható tervig 17
7 **UI-terv: így fog kinézni** Négy irány, egy választás – még kód nélkül 21
8 **A projektmappa** Egy mappa, benne minden, amit az AI-nak tudnia kell 24

III. RÉSZ Építés

9 **Indítás és közös munka az agenttel** Az első prompt, az első öt perc – és a jó visszajelzés 27
10 **Elkészült – mit kaptál?** Az átadás, a README és a localhost titka 30
11 **A terminál, félelem nélkül** Amikor az AI parancsot ad, és neked kell beírnod 33
12 **Tesztelés és hibajavítás** A „Kész." még nem kész 36

IV. RÉSZ Tovább

13 **Mentési pontok: Git és GitHub** Hogy bátran kísérletezhess – mindig legyen visszaút 39
14 **API, AI a programban és a titkok** Amikor a programod más rendszerekkel beszélget 41
15 **Adatbázis** A program tartós memóriája – és mikor kell 43
16 **PWA – app a telefonodon** Weboldalból telepíthető alkalmazás, app store nélkül 45
17 **Publikálás: hogyan lesz belőle élő oldal?** A képernyőképtől a saját domainig 47
18 **Haladó eszköztár** Szabálykönyv, projektmemória, skillek és MCP 49
19 **Az első saját projekted** Az egész egy oldalon – és a terv a következő hétre 51

MELLÉKLETEK Kézikönyv a gép mellé

A **Szószedet** A tankönyv fogalmai ábécérendben, egy-két mondatban. 53
B **Promptgyűjtemény** A tankönyv összes mintapromptja, sorrendben – másold, és igazítsd a saját projektedhez. 55
C **Terminál-puska** A leggyakoribb parancsok Macen és Windowson. A sor eleji jelet ($, >) soha nem kell beírni. 57
D **Hibaelhárító** A leggyakoribb elakadások, és mit tegyél velük. Ha egyik sem segít: B. melléklet, 8. prompt. 58
E **Technikai háttér** Részletesebb magyarázatok azoknak, akiket érdekel, mi történik a háttérben. A tanuláshoz nem szükséges. 59

I. RÉSZ

I.

Alapok

Mi ez az egész – és mivel dolgozunk?

1 Mi a programozás?

2 Mi a vibe coding?

3 Az eszközök

4 Modell és erőfeszítés

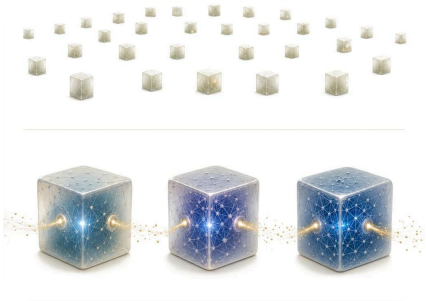


01

I. RÉSZ · ALAPOK · 1. LECKE

Mi a programozás?

A program is egy nyelv – és az AI beszéli



EBBEN A LECKÉBEN

- Megérted, mi egy program, egy hétköznapi hasonlattal.
- Meglátod, miért nyelv a programnyelv is.
- Megérted, miért tud az AI programot írni – és mi ebben a te szereped.

kb. 8 perc olvasás

Videó: 01. rész

3 kulcsgondolat

Naponta használsz programokat: a telefonod naptárát, a netbankot, egy webáruházat, a táblázatkezelőt. Mindegyik mögött ugyanaz az egyszerű dolog áll: **utasítások pontos sorozata**, amelyet a számítógép lépésről lépésre végrehajt.

A program olyan, mint egy recept

Egy recept is utasítások sorozata: „Vegyél két tojást. Verd fel. Ha habos, add hozzá a cukrot." Egy tapasztalt szakács kitalálja, amit a recept nem mond ki. A számítógép viszont semmit nem talál ki: pontosan azt csinálja, amit leírtak neki – se többet, se kevesebbet.

PÉLDA

Recept: „Ha a tészta aranybarna, vedd ki a sütőből."
Program: „Ha a kosárban lévő termékek ára meghaladja a 20 000 Ft-ot, a szállítás ingyenes."
Mindkettő feltételt vizsgál, és aszerint cselekszik. A program csak szigorúbb: minden esetet pontosan le kell írni benne.

A programnyelv is egy nyelv

A számítógép nem érti a magyart. Ezért találták ki a **programozási nyelveket** – Python, JavaScript, HTML és társaik. Ezeknek is van szókincsük és nyelvtanuk, mint a magyarnak vagy az angolnak; csak sokkal szigorúbbak: egy elfelejtett zárójel is hibát okoz.

TE MONDOD

„Írj egy kis programot, ami egy nettó összegből kiszámolja a bruttót, 27% ÁFÁ-val."

AI

AZ AI MEGÍRJA — PYTHON

```
# ÁFA-kalkulátor
def brutto(netto, afa=0.27):
    return round(netto * (1 + afa))

print(brutto(10000))
```

FUT

EREDMÉNY

12 700

10 000 Ft nettó + 27%

Ugyanaz a gondolat két nyelven: magyarul te mondod, Pythonban az AI írja le – és a gép lefuttatja.

A nagy nyelvi modellek – a ChatGPT, a Claude és a többiek – rengeteg szövegből tanultak, és ebben a szövegben rengeteg programkód is volt. Ezért úgy „beszélnek" Pythonul vagy JavaScriptül, mint angolul vagy magyarul. Nekik a **magyar mondatból programkódot írni ugyanolyan fordítási feladat**, mint magyarról angolra fordítani.

TIPP

Ez a vibe coding lényege: te a saját nyelveden mondod el, mit szeretnél – az AI lefordítja a gép nyelvére.

Miből áll egy szoftver?

Egy egyszerű weboldaltól a komoly üzleti rendszerig a legtöbb szoftvernek ugyanaz a négy része van. Ezeket nem kell megtanulnod – de jó, hogy léteznek, mert az AI is ezekről fog beszélni.

A felület
Amit látsz és kattintasz: gombok, űrlapok, táblázatok. Weboldalon HTML (tartalom), CSS (kinézet) és JavaScript (működés).

A logika
A szabályok: mit számol, mit ellenőriz, mi történik egy gombnyomásra.

Az adatok
Amit megjegyez: ügyfelek, számlák, beállítások. Ha tartósan kell, adatbázisba kerül.

A futtatás helye
Ahol a program fut: a saját gépeden, vagy egy szerveren az interneten.

A szoftver négy része. A tankönyv későbbi leckéi mindegyikhez visszatérnek.

JÓ TUDNI

Az E. mellékletben részletesebben is olvashatsz a fájltípusokról (HTML, CSS, JavaScript, Python) – de a tanuláshoz nincs rá szükség.

Mit jelent ez neked?

Eddig egy saját szoftverhez programozót kellett fogadni, vagy évekig tanulni. Ma a legfontosabb képesség a **pontos leírás**: tudni, MIT akarsz, és ezt érthetően elmondani. A kódot az AI írja. A te dolgod a cél, a döntések és az ellenőrzés – éppen az, amiben a saját szakmádban már most is jó vagy.

KULCSGONDOLATOK

1. A program utasítások pontos sorozata – mint egy nagyon szigorú recept.
2. A programnyelv is nyelv, és a mai AI-modellek folyékonyan beszélik.
3. A te nyelved a magyar: a feladatod a pontos leírás és az ellenőrzés.

PRÓBÁLD KI

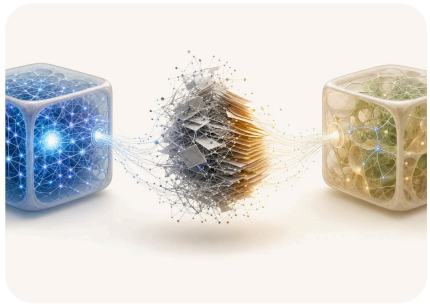
1. Nyiss meg egy sima AI-chatet (ChatGPT vagy Claude), és írd be: „Készíts egyetlen HTML-fájlt, amely egy nettó összegből kiszámolja a bruttó árat 27%-os ÁFÁ-val. Legyen szép és egyszerű.”
2. A kapott kódot másold be egy új szövegfájlba, mentsd el `afa.html` néven, majd nyisd meg dupla kattintással.
3. Gratulálunk: az első programod fut a böngésződben. Egyetlen sor kódot sem írtál.

02

I. RÉSZ · ALAPOK · 2. LECKE

Mi a vibe coding?

Te mondod meg, mit – az AI megcsinálja, hogyan



EBBEN A LECKÉBEN

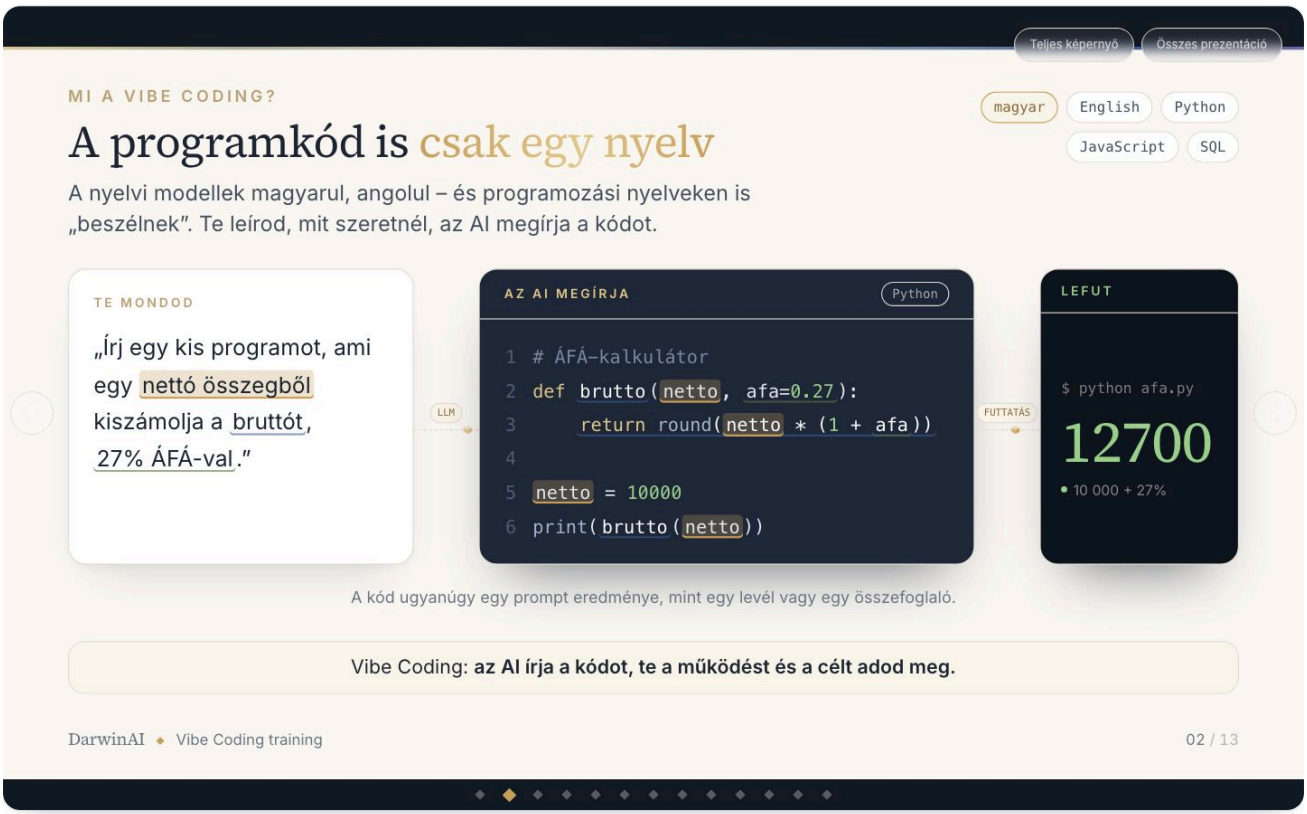
- Megtudod, mit jelent a vibe coding, és honnan jön a név.
- Látsz valódi példákat: mi készült így, mennyi idő alatt.
- Tisztázzuk, mi a te dolgod, és mi az AI-é.

kb. 9 perc olvasás

Videó: 02. rész

3 kulcsgondolat

A **vibe coding** (kiejtése: vájb kóding) azt jelenti, hogy **AI-val, a saját szavaiddal készítesz szoftvert**: te adod meg a működést és a célt, az AI írja meg a kódot. A kifejezést 2025 elején Andrej Karpathy, az egyik legismertebb AI-kutató tette népszerűvé: a fejlesztő a „hangulatot", az irányt adja meg – a gépelés az AI dolga.

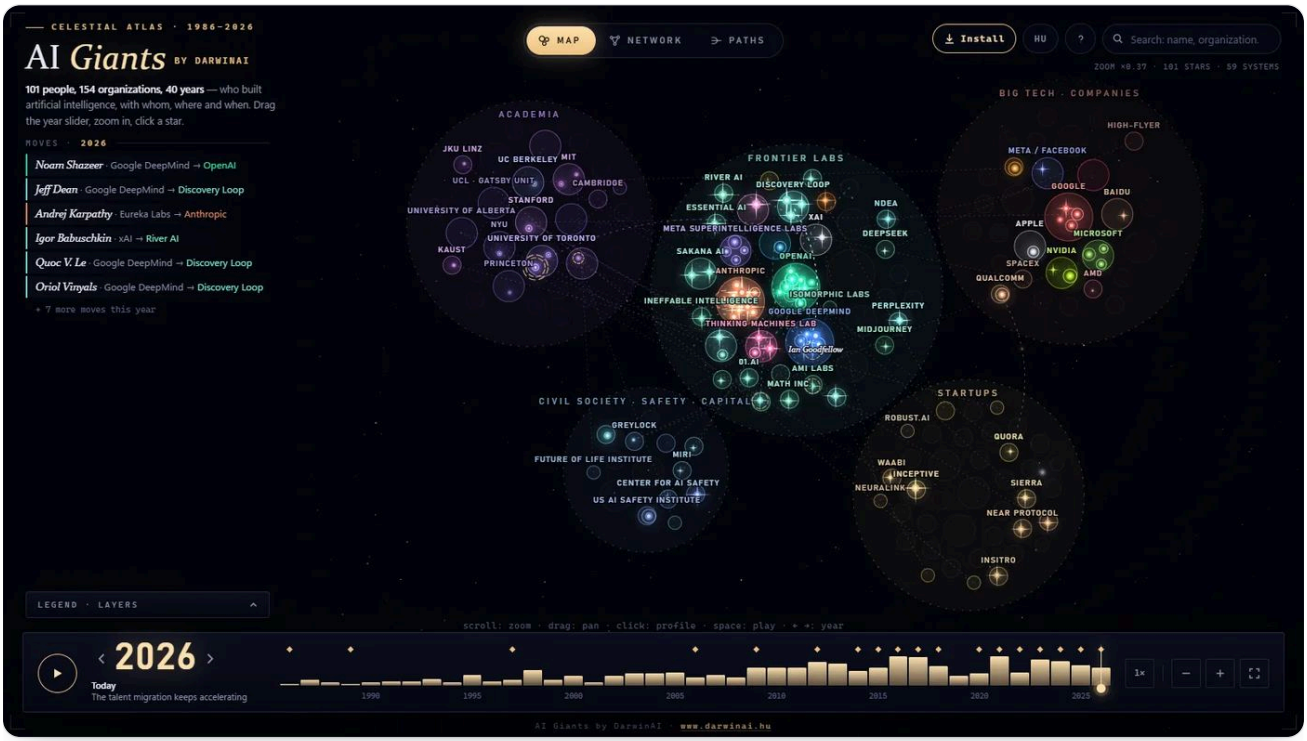


A DarwinAI Vibe Coding trainingjének 2. diája: a programkód is csak egy nyelv.

Mit lehet így építeni?

Szinte bármit, ami képernyőn fut. A DarwinAI saját projektjei között van prezentáció, belső munkaeszköz, védett webes rendszer, publikus weboldal, animáció és játék is. Néhány valódi példa, a ráfordított idővel:

PROJEKT	MI EZ?	RÁFORDÍTÁS
AI Giants	Interaktív „csillagtérkép” 101 AI-szereplő pályájáról, egyetlen HTML-fájlban	kb. 1 óra, 5 prompt
Leonardo	Bemutató weboldal és 20 diás prezentáció élő szimulátorral	kb. 2 óra, 6 prompt
Symmetria	Telepíthető, offline is működő 3D kristálytani webapp, magyarul és angolul	kb. 3 óra, 4 prompt
Pacsi	Kutyafajta-választó app: 124 fajta élő, animált felhőben	kb. 17 óra, 61 prompt
Polibeli	Párhuzamos szóbeli vizsgarendszer virtuális vizsgáztatóval – éles rendszer	hosszabb, iteratív fejlesztés



Az AI Giants – kb. egy óra aktív munkával, egyetlen fájlban.

Ki mit csinál?

A te dolgod

- Kitalálni, **mit** akarsz, és kinek
- Elmondani, mitől jó az eredmény
- Dönteni, ha az AI kérdez
- Kipróbálni és visszajelezni

ÉS

Az AI dolga

- Megírni a kódot
- Létrehozni a fájlokat, mappákat
- Elindítani és tesztelni a programot
- Megkeresni és kijavítani a hibákat

A munkamegosztás: te vagy a megrendelő és az ellenőr, az AI a kivitelező.

Reális elvárások

A vibe coding nem varázslat, hanem **munkamódszer**. Egy kis eszköz órák alatt elkészül; egy komolyabb rendszer napokig, hetekig tartó, sok körös munka. Az eredmény minősége szinte mindig azon múlik, mennyire jól írtad le az elején, mit akarsz – erről szól a tankönyv II. része.

- **Kiváló:** saját munkaeszközök, belső segédprogramok, prototípusok, weboldalak, bemutatók, oktatási anyagok, játékok.
- **Óvatosan:** ha érzékeny személyes adatot, pénzt vagy sok felhasználót kezel a rendszer. Ilyenkor a kész rendszert érdemes szakemberrel is átnézetni.

- **Nem helyettesíti:** a gondolkodást. Ha te sem tudod, mit akarsz, az AI sem fogja tudni.

> **JÓ TUDNI**
> A DarwinAI tapasztalata: a projekt idejének jó része a tervezés. Ami ott eldől, az később nem kerül pénzbe és órákba.

KULCSGONDOLATOK

1. Vibe coding: az AI írja a kódot, te adod meg a működést és a célt.
2. Valódi, működő termékek készülnek így – órák vagy napok alatt.
3. Te vagy a megrendelő és az ellenőr; az eredmény a leíráson múlik.

PRÓBÁLD KI

1. Nézz meg két projektet a darwinai.hu/projektek oldalon, és figyeld meg: mit csinál, kinek szól.
2. Írj fel három ötletet, amit a saját munkádban vagy otthon szívesen használnál. Nem kell okosnak lennie – a legjobb első projekt egy kis, bosszantó feladat megoldása.

03

I. RÉSZ · ALAPOK · 3. LECKE

Az eszközök

Sima chat vagy coding agent – és melyik platform?



EBBEN A LECKÉBEN

- Megérted, miért nem elég a sima chat egy igazi szoftverhez.
- Megismered a fő vibe coding platformokat.
- Látod, mit csinál egy coding agent, amíg dolgozik.

kb. 10 perc olvasás

Videó: 03. rész

3 kulcsgondolat

Kódot a sima chat is ír – az előző lecke gyakorlatában te is kipróbáltad. Egy kis HTML-oldalhoz ez elég. Amint viszont a program több fájlból áll, adatot ment, vagy többször kell módosítani, a sima chat nehézkessé válik: a kódot neked kell fájlokba másolgatnod, futtatnod, és a hibákat visszahordanod.

Sima chat

Kódot ír – de nem erre találták ki. A fájlokba másolás, a futtatás és a hibakeresés rád marad.

VS

Coding agent (vibe coding platform)

Direkt kódolásra készült: a fájlokat ő kezeli, futtatja a programot, tesztel és javít.

> A legfontosabb különbség: a coding agent nem a chatablakban „beszél" a kódról, hanem a projekt fájljaiban dolgozik.

Mitől jobb egy coding agent?

Kezeli a kódbázist
Ismeri a teljes projektet, ezért csak ott módosít, ahol kell – nem ír újra mindent.

Kódolásra írt alaputasítás
Tudja, hogyan épül fel egy felület, hogyan kell adatot és jelszót biztonságosan kezelni.

Tudja a dolgát
Megtervezi a lépéseket, létrehozza a fájlokat, futtatja a programot.

Tesztel és visszaellenőriz
Kipróbálja, amit írt; ha hibát talál, javítja és újra ellenőrzi.

> A DarwinAI training „Platform, ami tud kódolni" diájának négy érve.

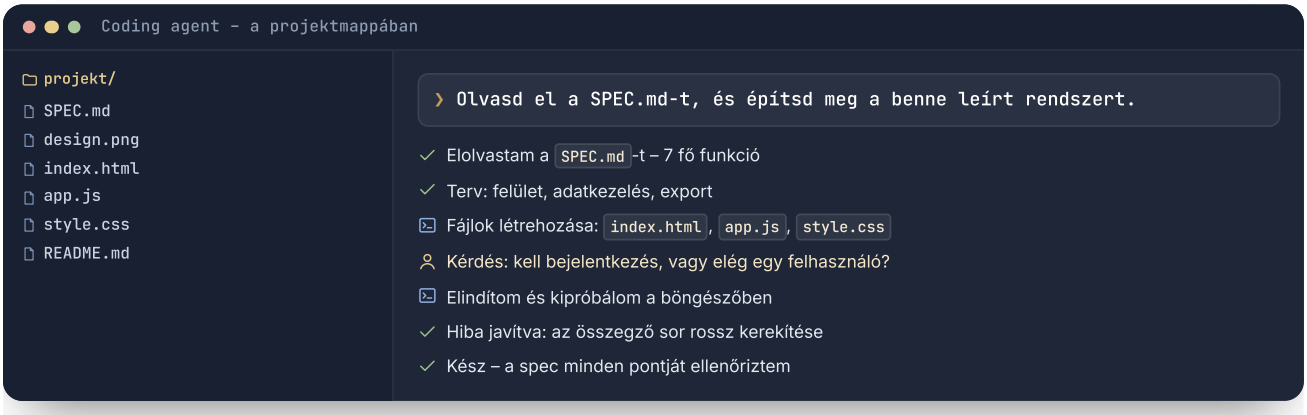
A fő platformok

PLATFORM	HOL FUT?	KINEK AJÁNLOTT?
Lovable	Böngészőben	A leggyorsabb kezdés: semmit nem kell telepíteni, a publikálás egy gomb.
Claude Code (Anthropic)	Asztali app, böngésző vagy terminál	Saját gépen, saját mappában dolgozó, nagyon erős agent – komolyabb projektekhez.
OpenAI Codex	Asztali app vagy böngésző	A ChatGPT-előfizetéshez kapcsolódó coding agent – szintén komolyabb projektekhez.
Cursor, VS Code	Letöltött kódszerkesztő	Ha később a kódba is bele akarsz nézni; bármilyen AI-modellel használhatók.

A tankönyv a **saját gépen, projektmappában dolgozó coding agentre** épül (Claude Code vagy Codex), mert ez adja a legtöbb szabadságot, és ezt használja a DarwinAI is. A lépések a többi platformon is nagyon hasonlók.

> **JÓ TUDNI**
> A coding agentekhez általában fizetős előfizetés kell (például Claude Pro vagy Max, ChatGPT Plus vagy Pro). Az árak és a csomagok gyakran változnak – mindig a szolgáltató oldalán nézd meg az aktuálisat.

Mit csinál az agent, amíg dolgozik?



Egy coding agent munkanaplója (szemléltetés). Balra a mappa fájljai, jobbra a lépések – és egy kérdés, amire neked kell válaszolnod.

> **FIGYELEM**
> Az agent fájlírás vagy parancsfuttatás előtt engedélyt kérhet. Mindig olvasd el, mit csinálna – erről a 9. leckében részletesen lesz szó.

KULCSGONDOLATOK

1. Igazi szoftverhez coding agent kell, nem sima chat.
2. Az agent a projekt fájljaiban dolgozik: létrehoz, futtat, tesztel, javít.
3. Kezdéshez jó a Lovable; saját gépes, komolyabb munkához Claude Code vagy Codex.

PRÓBÁLD KI

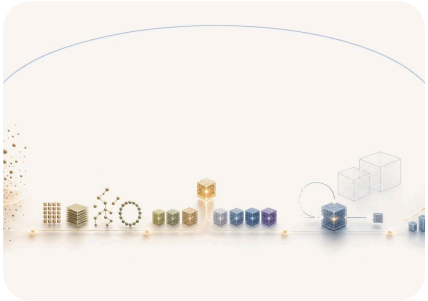
1. Döntsd el, melyik platformmal kezdesz. Ha bizonytalan vagy: Claude Code vagy Codex asztali app – ezekkel a tankönyv minden lépése végigcsinálható.
2. Telepítsd vagy nyisd meg, és jelentkezz be. Még ne indíts projektet – azt a 9. leckében tesszük meg.

04

I. RÉSZ · ALAPOK · 4. LECKE

Modell és erőfeszítés

Melyik „agy” dolgozzon – és mennyit gondolkodjon?



EBBEN A LECKÉBEN

- Megérted, mi a modell, és miért van belőle több.
- Megtudod, mit jelent az erőfeszítés (effort) beállítás.
- Kapsz egy egyszerű szabályt: mikor melyiket válaszd.

kb. 7 perc olvasás

Videó: 04. rész

3 kulcsgondolat

Minden AI-platformban kiválaszthatod, melyik **modell** dolgozzon. A modell az AI „agya”. A nagyobb modellek okosabbak és alaposabbak, de lassabbak és többe kerülnek; a kisebbek gyorsak és olcsók, egyszerűbb feladatra tökéletesek.

A másik beállítás az **erőfeszítés** (angolul *effort* vagy *reasoning*): mennyi ideig „gondolkodjon” a modell, mielőtt válaszol vagy cselekszik. Jellemző fokozatai: alacsony, közepes, magas, extra magas, maximum.

Modell = ki dolgozik
Nagyobb modell: okosabb, alaposabb, drágább. Kisebb: gyors és olcsó.

Effort = mennyit gondolkodik
Magasabb fokozat: alaposabb terv, kevesebb hiba – több idő és költség.

A két beállítás együtt dönti el a minőséget, a sebességet és a költséget.

Az egyszerű szabály

1. **A projekt első, nagy felépítése**
   Ez a legfontosabb pillanat: itt dől el a szerkezet. Használd a legerősebb modellt, magas vagy maximális erőfeszítéssel.
2. **Későbbi, jól körülhatárolt módosítások**
   „A gomb legyen zöld”, „adj hozzá egy dátummezőt” – ehhez gyorsabb, olcsóbb modell és közepes erőfeszítés is bőven elég.
3. **Független új funkció**
   Nyiss neki új beszélgetést (sessiont): átláthatóbb, és a régi, már nem releváns részletek nem zavarják össze az AI-t.

	MODELL	ERŐFESZÍTÉS	MIKOR?
Vázlatból specifikáció (chat)	**erős, „gondolkodó”**	**magas**	egyszer, a projekt elején
Első nagy építés (agent)	**a legerősebb**	**extra magas / max**	egyszer, az indításnál
Kisebb módosítás	gyorsabb, olcsóbb	közepes	sokszor, menet közben
Szövegjavítás, apróság	gyors	alacsony	bármikor

Melyik feladathoz mit válassz – általános ökölszabály.

PÉLDA

A DarwinAI ajánlása az első nagy építéshez (2026. október):
Codex – Sol 6.1, High vagy Extra High erőfeszítéssel.
Claude Code – Opus 5.5, Extra vagy Max erőfeszítéssel.
A modellek gyorsan változnak: mindig az aktuális kínálatból válassz. Jó kiindulópont a legnagyobb modell, a maximumnál eggyel alacsonyabb fokozattal.

A „rövid távú memória”

Egy beszélgetésen belül az AI emlékszik mindarra, ami addig elhangzott – ezt hívják **kontextusnak**. Ez hasznos, de egy idő után a sok régi részlet már inkább zavar. Ezért jó szokás minden független feladathoz új beszélgetést nyitni: tiszta asztal, friss figyelem.

FIGYELEM

Claude-ban a modellt beszélgetésenként választod. Ha menet közben váltasz, újra fel kell dolgoznia az addigi beszélgetést – ez plusz költség.

KULCSGONDOLATOK

1. Modell = ki dolgozik; erőfeszítés = mennyit gondolkodik.
2. Az első nagy építéshez a legerősebb modell kell, magas erőfeszítéssel.
3. Apró módosításhoz elég a gyorsabb, olcsóbb beállítás; független feladathoz új beszélgetés.

PRÓBÁLD KI

1. Nyisd meg a választott platformot, és keresd meg, hol állítható a modell és az erőfeszítés. Jegyezd fel, milyen lehetőségeid vannak.

II. RÉSZ

II.

Tervezés

Mielőtt egy sor kód is készülne – itt dől el a minőség.

5 A folyamat térképe

6 Vázlatból specifikáció

7 UI-terv: így fog kinézni

8 A projektmappa



05

II. RÉSZ · TERVEZÉS · 5. LECKE

A folyamat térképe

Kilenc lépés az ötlettől a működő programig



EBBEN A LECKÉBEN

- Áttekinted a teljes vibe coding folyamatot egyetlen térképen.
- Megérted, miért a tervezés a legfontosabb szakasz.
- Megtudod, hol találod a pipálható, részletes változatot.

kb. 7 perc olvasás
Videó: 05. rész
3 kulcsgondolat

Sokféle munkamódszer létezik, és idővel mindenki kialakítja a sajátját. Kezdőként viszont sokat segít egy kipróbált útvonal. Ez a kilenc lépés a DarwinAI saját folyamata – minden valódi projektünk így készül, a kis eszközöktől a komoly rendszerekig.



A vibe coding folyamat kilenc lépése. A szaggatott keretes lépések (UI-terv, publikálás) opcionálisak.

Három szakasz

I. Mit akarsz?
1–3. lépés: vázlat, specifikáció, látványterv. Még egy sor kód sem készül – és mégis itt dől el a legtöbb.

II. Építsd meg
4–7. lépés: projektmappa, indítás, figyelés. Az AI dolgozik, te irányítasz.

III. Használd és add tovább
8–9. lépés: átadás, első indítás, publikálás. Itt lesz a kódból használható eszköz.

Miért a tervezés a legfontosabb?

Mert **a specifikáción gondolkodni sokkal olcsóbb, mint kész kódot javítani**. Ha a terv jó, az AI első nekifutásra közel azt építi meg, amit szeretnél. Ha a terv homályos („csinálj egy jó cégértékelő eszközt"), az AI kitalálja helyetted a részleteket – és ritkán azt, amit te gondoltál.

× Homályos kérés
„Csinálj egy jó cégértékelő appot."
→ Az AI találgat: kinek, mit, hogyan.

VS

✓ Átgondolt specifikáció
Cél, célcsoport, folyamat, bemenet és kimenet, funkciók, mitől jó, mi nem cél.
→ Az AI azt építi, amit kértél.

A DarwinAI tapasztalata: amit a tervezésben megspórolsz, azt az építésben többszörösen fizeted meg.

Mennyi idő?

LÉPÉS	JELLEMZŐ IDŐ
Saját vázlat	akár 1–2 nap „érlelés” – közben más is lehet a fejedben
AI-upgrade, UI-terv	30–60 perc
Projektmappa, indítás	10–15 perc
Kis figyelés	kb. 5 perc
Az AI önálló munkája	pár perctől néhány óráig, a projekt méretétől függően
Átadás, kipróbálás, javítások	fél órától napokig – ez az iterációs szakasz



> A darwinai.hu/vibe-coding/folyamat oldalon a folyamat részletes, 15 lépéses, pipálható változata is megtalálható – mintapromptokkal és letölthető ellenőrzőlistával.

TIPP

Nyisd meg a folyamat oldalát az egyik ablakban, a projektedet a másikban, és pipáld, ami kész. Így semmi nem marad ki.

KULCSGONDOLATOK

1. Kilenc lépés: vázlat → AI-upgrade → UI-terv → mappa → indítás → figyelés → magára hagyás → átadás → publikálás.
2. A tervezés a legfontosabb: olcsóbb a terven gondolkodni, mint a kódot javítani.
3. A részletes, pipálható változat a darwinai.hu/vibe-coding/folyamat oldalon van.

PRÓBÁLD KI

1. Válassz egyet az 2. leckében felírt ötleteid közül. Ezen fogsz végigmenni a következő leckékben.
2. Nyisd meg a darwinai.hu/vibe-coding/folyamat oldalt, és fusd át a lépéseket.

06

II. RÉSZ · TERVEZÉS · 6. LECKE

Vázlatból specifikáció

A saját szavaidtól az átadható tervig



EBBEN A LECKÉBEN

- Megtanulod, mit írj bele az első, saját vázlatodba.
- Megkapod a promptot, amellyel az AI specifikációt csinál belőle.
- Megérted, mi a Markdown, és miért SPEC.md a fájl neve.

kb. 12 perc olvasás

Videó: 06. rész

3 kulcsgondolat

1. lépés: a saját vázlat

Az első lépés a legemberibb: **írd le a saját szavaiddal, mit szeretnél**. Nem kell szépnek, rendezettnek vagy teljesnek lennie. Lehet jegyzet, felsorolás, néhány bekezdés – akár hangüzenet leirata is. A lényeg, hogy benne legyen, amit tudsz.

Hét kérdés segít, hogy semmi fontos ne maradjon ki:

Mi a cél?
Mire való, milyen problémát old meg?

Kinek készül?
Ki használja, mennyire ért hozzá, milyen eszközön?

Mi a folyamat?
Mit csinál a felhasználó, és mit a program – lépésről lépésre?

Mi megy be, mi jön ki?
Milyen adatokkal, fájlokkal dolgozik, és mit ad vissza?

Mit tudjon?
A fő funkciók és képernyők. Kell-e belépés?

Mitől jó az eredmény?
Minek kell a végén mindenképpen működnie?

Mi nem cél?
Mit hagyjon ki szándékosan? A határok is a terv részei.

A jó specifikáció hét kérdése – a DarwinAI training „Specifikáció” diája alapján.

JÓ TUDNI

A MIT a lényeg, nem a HOGYAN. Adatbázis-táblákat, programnyelvet, technikai felépítést nem neked kell kitalálnod – azt a coding agent megtervezi.

PÉLDA

Így néz ki egy valódi, szándékosan rendezetlen első vázlat (részlet):

SAJÁT VÁZLAT – SZERZŐDÉS-ELLENŐRZŐ ahogy eszembe jutott

szerződés-ellenőrző app

kellene egy program amibe be lehet tölteni egy szerződést (pdf vagy word), és megmondja, mi a kockázatos benne. főleg kisvállalkozóknak, akik nem tudnak minden szerződéshez jogászt fizetni.

kiemeli a gyanús részeket: fizetési határidő, kötbér, felmondás, kizárólagosság. piros-sárga-zöld jelzés talán?

fontos, hogy egyszerűen magyarázza el, ne jogi nyelven!! és mit kérdezzek vissza a másik féltől.

a határidőket gyűjtse ki egy listába, mert azokat szoktam elfelejteni. emlékeztető talán?

több szerződést is el lehetne tárolni? nem tudom még.

belépés kell? ne lássa más a szerződéseimet, érzékeny adat!!

Látszik, hogy ismétlődik, néhol bizonytalan („talán?", „nem tudom még"), és hiányoznak belőle dolgok. **Pont ilyen vázlatból lesz a legjobb specifikáció** – mert benne van, ami fontos neked.

TIPP

Hagyd érni a vázlatot. A DarwinAI-nál egy-két napig is alakul: közben eszedbe jutnak új dolgok, és kiderül, mi a valóban fontos.

2. lépés: az AI-upgrade

A vázlatot most odaadod egy AI-nak **sima chatben** (ChatGPT vagy Claude) – lehetőleg erős, „gondolkodó" modellel. A feladata: rendszerezni, kiegészíteni, és **átadható specifikációt** írni belőle. Kódolni még nem szabad.

MINTAPROMPT – VÁZLATBÓL SPECIFIKÁCIÓ ChatGPT vagy Claude, erős modellel

```
Az alábbi vázlatot egy szoftverről írtam, amit egy coding agenttel szeretnék elkészíttetni.

1. Strukturáld és rendszerezd a leírásomat.
2. Jelezd az ellentmondásokat és a hiányzó pontokat – ha kell, kérdezz vissza.
3. Tedd hozzá azokat a funkciókat, amelyek szerinted szükségesek a cél eléréséhez.
4. Készíts belőle Markdown formátumú specifikációt, amelyet közvetlenül át tudok adni egy vibe coding eszköznek.

NE kezdj még kódolni. Adatbázis-táblákat és részletes technikai architektúrát nem kell meghatároznod – azt később a coding
agent megtervezi.

A vázlatom:
[ide másold be a vázlatodat]
```

Ha vannak mintafájlok, amelyekkel a kész programnak dolgoznia kell (egy minta Excel, egy PDF, egy dokumentum), **csatold őket** ehhez a beszélgetéshez is. Sokat segítenek abban, hogy a terv a valóságra épüljön.



Az AI visszakérdezései a folyamat legértékesebb részei. Válaszolj rájuk – a válaszaid bekerülnek a specifikációba.

Olvasd végig – és húzd ki, amit nem akarsz

Az AI hajlamos „túltervezni": olyan funkciókat is betesz, amelyek jól hangzanak, de neked nem kellenek. Olvasd végig a specifikációt, és **töröld vagy kérd kivenni**, ami felesleges. Minden kihúzott funkció gyorsabb, olcsóbb, megbízhatóbb programot jelent.

Mentsd el: SPEC.md

A kész specifikációt mentsd el egy **SPEC.md** nevű fájlba. Az **.md** a **Markdown** formátum jele: egyszerű szöveg, amelyben néhány jel (például `#` a címek előtt, `-` a felsorolásnál) megőrzi a szerkezetet. Az AI-k ezt a formátumot olvassák a legszívesebben. Egy sima `leiras.txt` is megteszi, ha az egyszerűbb.

Markdown – ahogy írod

```
# Szerződés-ellenőrző

## 1. Cél
Kisvállalkozók gyorsan lássák,
mi kockázatos egy szerződésben.

## 2. Funkciók
- PDF / Word feltöltése
- piros-sárga-zöld kockázatjelzés
- határidő-lista emlékeztetővel
```

→

Ahogy megjelenik

Szerződés-ellenőrző

1. Cél – Kisvállalkozók gyorsan lássák, mi kockázatos egy szerződésben.

2. Funkciók – PDF / Word feltöltése · kockázatjelzés · határidő-lista

A Markdown egyszerű szöveg, néhány jellel. Ember és gép is könnyen olvassa.

TIPP

A chatben általában van „Másolás" gomb a válasz alatt. Másold ki, illeszd be egy szövegszerkesztőbe (Macen TextEdit „egyszerű szöveg" módban, Windowson Jegyzettömb), és mentsd `SPEC.md` néven.

KULCSGONDOLATOK

1. Az első vázlat lehet rendezetlen – a lényeg, hogy benne legyen, amit tudsz.
2. A hét kérdés: cél, célcsoport, folyamat, bemenet–kimenet, funkciók, mitől jó, mi nem cél.
3. Az AI sima chatben specifikációt csinál belőle – kódolás nélkül. Az eredmény: SPEC.md.

PRÓBÁLD KI

1. Írd meg a saját vázlatodat a választott ötletedhez – 10–20 sor bőven elég. Használd a hét kérdést.
2. Add oda a mintaprompttal egy AI-chatnek (erős modell). Válaszolj a kérdéseire.
3. Olvasd végig az eredményt, húzd ki a felesleget, és mentsd SPEC.md néven.

07

II. RÉSZ · TERVEZÉS · 7. LECKE

UI-terv: így fog kinézni

Négy irány, egy választás – még kód nélkül



EBBEN A LECKÉBEN

- Megérted, mikor érdemes látványtervet készíteni.
- Megkapod a promptot, amellyel négy eltérő irányt kérhetsz.
- Megtanulod, hogyan választasz, és mit mentesz el belőle.

kb. 7 perc olvasás

Videó: 07. rész

3 kulcsgondolat

A **UI** (user interface, felhasználói felület) az, amit a program használója lát: a képernyők, a gombok, a színek, az elrendezés. Ha nem adsz róla útmutatást, az AI egy átlagos, „sablonos" felületet épít. Ha viszont **látványtervet** adsz mellé, az eredmény azonnal a te ízlésedet és céljaidat tükrözi.

JÓ TUDNI

Ez a lépés opcionális. Egy saját, belső eszköznél a kinézet kevésbé fontos; egy ügyfeleknek szóló weboldalnál vagy alkalmazásnál viszont sokat számít.

Négy irányt kérj – ne egyet

Maradj ugyanabban a chatben, ahol a specifikáció készült (így az AI ismeri a rendszert), és kérj tőle képgenerálással **négy kifejezetten eltérő** látványtervet. Négy irány közül sokkal könnyebb választani, mint egyetlen javaslatot értékelni.

MINTAPROMPT – UI-TERVEK — ugyanabban a chatben, képgenerálással

```
Tervezd meg a most létrehozott rendszer néhány fő képernyőjét, a core designját. Csinálj 4 kifejezetten eltérő
megközelítést, és mindegyikhez írd le röviden, mi a hangulata és kinek szól.
```



Valódi példa: a Leonardo projekt egyik látványterve a ChatGPT-ből. A kész oldal ebből épült.

Válassz – vagy keverj

- **Válaszd ki a legjobbat**, és mentsd el képként: **design.png**.
- **Keverd a kedvenc részeket**: „Az 1. terv színei, a 3. terv elrendezése.” Kérd, hogy készítsen ebből egy ötödiket.
- **Adj meg kulcsszavakat**, ha tudod, mit szeretnél: letisztult, prémium, játékos, sötét, magazinszerű – vagy a céged színeit.
- **Mutass példát**: egy weboldal, amelynek tetszik a hangulata, sokat segít.



A kész Leonardo-oldal: a látványterv hangulata és elrendezése egy az egyben átjött.

TIPP

A látványterv nem pixelpontos előírás, hanem **irány**. A coding agent a hangulatot, a színeket és az elrendezést veszi át belőle – és működő felületet épít hozzá.

KULCSGONDOLATOK

1. A látványterv az ízlésedet adja át az AI-nak – nélküle sablonos felület lesz.
2. Kérj négy kifejezetten eltérő irányt ugyanabban a chatben, ahol a specifikáció készült.
3. Válassz vagy keverj, és mentsd el design.png néven.

PRÓBÁLD KI

1. A specifikációs chatben add ki a UI-promptot.
2. Válaszd ki a kedvencet (vagy kérj keveréket), és mentsd el design.png néven a SPEC.md mellé.

08

II. RÉSZ · TERVEZÉS · 8. LECKE

A projektmappa

Egy mappa, benne minden, amit az AI-nak tudnia kell



EBBEN A LECKÉBEN

- Megtanulod, hogyan rendezd a projektjeidet a gépeden.
- Megtudod, mi kerüljön a projektmappába az építés előtt.
- Megismered a szabályfájlt (RULES.md), és tudod, mire figyelj a biztonságnál.

kb. 8 perc olvasás
Videó: 08. rész
3 kulcsgondolat

A coding agent mindig **egy mappában dolgozik**: azt olvassa, abban hoz létre és módosít fájlokat. Ezért minden projektnek saját mappa jár, és ebbe még az építés előtt mindent beteszel, amit az AI-nak tudnia kell.

A jó rendszer: egy gyűjtőmappa, benne projektek

Dokumentumok / AI Projects

```
AI Projects/                          gyűjtőmappa
  szerzodes-ellenorzo/                ez a projekt
    SPEC.md                           a specifikáció
    design.png                        a választott látványterv
    RULES.md                          általános elvárások (opcionális)
    minta-fajlok/                     teszt-dokumentumok
      minta-szerzodes.pdf
  ugyfelkezelo/                       egy másik projekt
  excel-elemzo/
```

Minden projekt külön mappában. Az építés előtt benne van a specifikáció, a látványterv, a szabályfájl és a mintafájlok.

TIPP

Mappanévnek használj kisbetűs, ékezet és szóköz nélküli nevet (pl. `szerzodes-ellenorzo`). Ez nem kötelező, de később – például a terminálban – sok apró bosszúságtól kímél meg.

FIGYELEM

Soha ne add oda az agentnek a teljes Dokumentumok mappádat, vagy olyan mappát, amelyben több, egymástól független projekt van. Az agent a kapott mappában módosíthat, létrehozhat és akár törölhet is fájlokat.

A szabályfájl: RULES.md

Vannak elvárások, amelyek **minden projektedre** igazak: legyen mobilon is használható, legyen magyar és angol nyelvű, legyen benne használati útmutató, a titkos adatok soha ne kerüljenek a kódba… Ezeket nem kell minden alkalommal újra leírnod: egyszer összeállítod őket egy **RULES.md** fájlba, és minden új projektmappába bemásolod.



A darwinai.hu/vibe-coding/szabaly-osszeallito oldalon pipálással állíthatod össze a saját RULES.md fájlodat – 12 területből, 46 kipróbált szabályból.

A szabálykönyv 12 területet fed le – a tervezéstől a dizájnon, a biztonságon és az üzemeltetésen át a minőségig. Kezdőként érdemes a javasolt alapcsomaggal indulni, és később bővíteni.

Mintafájlok és tudásbázis

Ha a programnak fájlokkal kell dolgoznia (szerződések, számlák, táblázatok), tegyél a mappába néhány **valódi vagy valósághű mintát**. Az agent ezeken teszteli a munkáját. Érzékeny adatnál használj kitalált vagy anonimizált példákat.

És a jelszavak, kulcsok?

Ha a programnak valamilyen külső szolgáltatáshoz kell hozzáférnie (például egy AI-szolgáltatáshoz), ahhoz **kulcs** kell. Ezt soha ne írd be a specifikációba vagy a chatbe. A biztonságos módszer a **.env** nevű fájl – a 14. leckében részletesen megmutatjuk. Kezdetben elég annyi, hogy az agent maga szól, ha kulcsra van szüksége.

KULCSGONDOLATOK

1. Minden projektnek saját mappa jár – és csak ezt a mappát kapja meg az agent.
2. Építés előtt a mappában van: SPEC.md, design.png, RULES.md és a mintafájlok.
3. A RULES.md a minden projektedre érvényes elvárásaid gyűjteménye – egyszer állítod össze.

PRÓBÁLD KI

1. Hozz létre egy „AI Projects" mappát, benne egy mappát a projektednek.
2. Tedd bele a SPEC.md-t és a design.png-t.
3. Állítsd össze a saját RULES.md-det a darwinai.hu/vibe-coding/szabaly-osszeallito oldalon, és mentsd ugyanide.

III. RÉSZ

III.

Építés

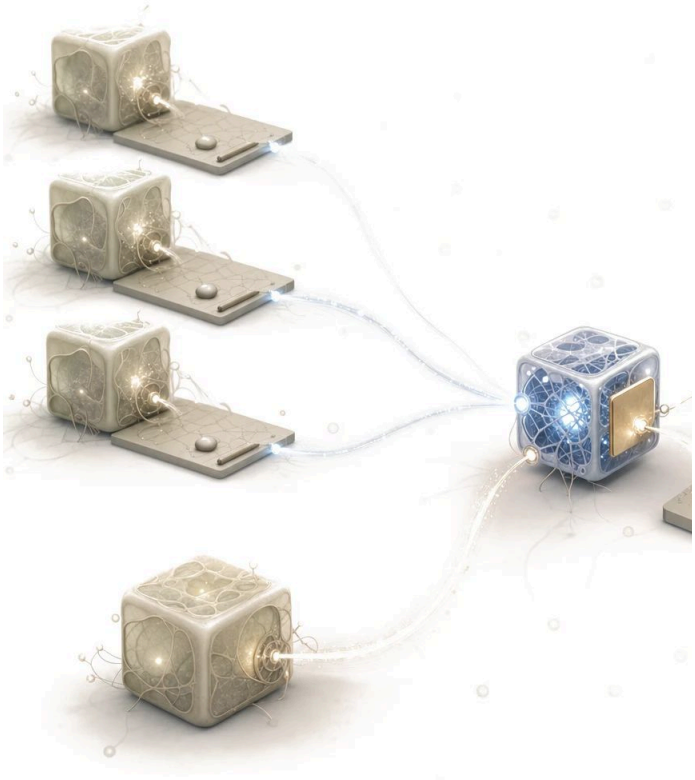
Az AI dolgozik, te irányítasz – és megtanulod használni, amit kaptál.

9 Indítás és közös munka az agenttel

10 Elkészült – mit kaptál?

11 A terminál, félelem nélkül

12 Tesztelés és hibajavítás



09

III. RÉSZ · ÉPÍTÉS · 9. LECKE

Indítás és közös munka az agenttel

Az első prompt, az első öt perc – és a jó visszajelzés



EBBEN A LECKÉBEN

- Elindítod az első igazi vibe coding munkamenetedet.
- Megtanulod, mit figyelj az első percekben, és mit jelentenek az engedélykérések.
- Megtanulod, hogyan adj jó visszajelzést, hogy az AI gyorsan javítson.

kb. 11 perc olvasás

Videó: 09. rész

3 kulcsgondolat

Elérkeztél a pillanathoz, amikor az AI kódolni kezd. Ha a II. rész lépéseit végigcsináltad, ez meglepően egyszerű lesz: a nehezén már túl vagy.

5. lépés: az indítás

1. **Nyisd meg a coding agentet**
   Claude Code vagy Codex – asztali appban vagy böngészőben.
2. **Indíts új projektet / munkamenetet, és válaszd ki a projektmappát**
   Innentől az agent ebben a mappában dolgozik.
3. **Állítsd be a modellt és az erőfeszítést**
   Az első nagy építéshez a legerősebbet, magas fokozattal (4. lecke).
4. **Add ki az indító promptot**
   Lásd lent. Ha még eszedbe jutott valami elvárás, írd hozzá.

AZ INDÍTÓ PROMPT — Claude Code, Codex, Lovable...

```
A projektmappában megtalálod a SPEC.md fájlt (a rendszer leírása), a design.png fájlt (a látványterv) és a RULES.md fájlt (az általános elvárások). Olvasd el mindhármat, és készítsd el a SPEC.md-ben leírt rendszert a design.png alapján, a RULES.md követelményei szerint. Csak akkor tekintsd késznek, ha a specifikáció minden pontját végigellenőrizted.
```

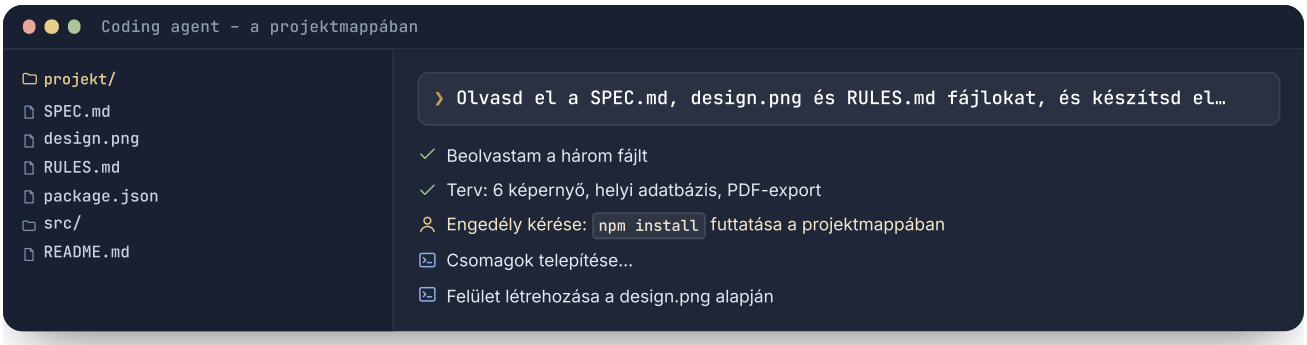
TIPP

A prompt utolsó mondata a legfontosabb: „Csak akkor tekintsd késznek, ha a specifikáció minden pontját végigellenőrizted.” Ezzel arra kéred az agentet, hogy ne álljon meg félúton.

6. lépés: az első öt perc

Ne menj el rögtön. Az első néhány percben figyeld, mit csinál az agent:

- **Visszakérdez?** Válaszolj röviden és egyértelműen.
- **Jó irányba indul?** Ha félreértett valamit, most szólj – később drágább.
- **Engedélyt kér?** Olvasd el, mit akar csinálni (lent részletesen).



Egy engedélykérés munka közben. Az agent megmondja, milyen parancsot futtatna – te döntesz.

Az engedélykérések

Az agent fájlírás vagy parancsfuttatás előtt gyakran **engedélyt kér**. Ez biztonsági funkció, nem hiba. Jellemző válaszlehetőségek: „Engedélyezem egyszer", „Mindig engedélyezem ebben a projektben", „Elutasítom".

AMIT KÉR	MIT JELENT?	MIT TEGYÉL?
Fájl létrehozása, módosítása a projektmappában	A munkája része.	Nyugodtan engedélyezd.
`npm install`, `pip install`	A projekthez szükséges csomagokat tölti le.	Általában rendben van.
`npm run dev`, **tesztek futtatása**	Elindítja, kipróbálja a programot.	Rendben van.
Bármi a projektmappán kívül	A géped más részeihez nyúlna.	Állj meg, kérdezd meg, miért kell.
`sudo`, **rendszergazdai jog, törlés (**`rm -rf`**)**	Komoly, nehezen visszafordítható művelet.	Csak akkor, ha érted és tényleg kell.

7. lépés: magára hagyod

Ha jó irányba indult, **hagyd dolgozni**. Egy kisebb projekt néhány perc, egy nagyobb akár egy-két óra. Közben nyugodtan csinálj mást. Ha kész, az agent összefoglalót ír – erről szól a következő lecke.

Iterálás: a jó visszajelzés

Az első változat ritkán tökéletes – ez természetes. A vibe coding **körökben** halad: kipróbálod, visszajelzel, az AI javít. A jó visszajelzés konkrét:

× Gyenge visszajelzés

- „Nem jó."
- „Csináld szebbre."
- „A gomb nem működik."

VS

✓ Jó visszajelzés

- „A Mentés gomb után nem jelenik meg az új tétel a listában."
- „A fejléc legyen sötétkék, a betűk nagyobbak – mint a design.png-n."
- „Képernyőkép csatolva: itt csúszik szét telefonon."

Mit csináltál, mit vártál, mi történt – és ha lehet, egy képernyőkép.

- **Egyszerre egy-két dolgot kérj.** Tíz kérés egy üzenetben = tíz félmegoldás.
- **Képernyőkép** sokat ér: a legtöbb agentbe egyszerűen be lehet húzni.
- **Független új funkcióhoz nyiss új munkamenetet** – tisztább, gyorsabb.
- **Nagy változtatás előtt kérj tervet:** „Mielőtt hozzányúlsz, írd le, mit változtatnál."

KULCSGONDOLATOK

1. Indítás: agent → projektmappa → modell → indító prompt, a végén „csak akkor kész, ha mindent ellenőriztél”.
2. Az első öt percben figyelj: kérdések, irány, engedélykérések. Utána hagyd dolgozni.
3. Iterálj konkrét visszajelzéssel: mit csináltál, mit vártál, mi történt – plusz képernyőkép.

PRÓBÁLD KI

1. Indítsd el a saját projektedet az indító prompttal.
2. Az első öt percben figyelj: válaszolj a kérdéseire, olvasd el az engedélykéréseket.
3. Ha elkészült, próbáld ki, és adj egy konkrét visszajelzést – figyeld meg, hogyan javít.

10

III. RÉSZ · ÉPÍTÉS · 10. LECKE

Elkészült – mit kaptál?

Az átadás, a README és a localhost titka



EBBEN A LECKÉBEN

- Megtanulod értelmezni az agent záróüzenetét.
- Megérted a különbséget: kész a kód – fut a program – mások is elérik.
- Megérted, mi a localhost, és miért nem működik a link másnál.

kb. 9 perc olvasás

Videó: 10. rész

3 kulcsgondolat

Az agent egyszer csak azt írja: „Elkészült." Gyakran még hozzáteszi: „Indítás: `npm run dev`. Megnyitás: `localhost:5173`." Kezdőként ez elsőre rejtélyes. Ebben a leckében megfejtjük.

Három külön állapot

1

A kód elkészült

A fájlok ott vannak a mappában.

→

2

A program fut

Elindítottad, a böngészőben kipróbálható.

→

3

Mások is elérik

Szerveren van, egy linkkel megnyitható.

> Az „elkészült" általában csak az elsőt jelenti. Egy projekt lehet kész úgy, hogy még el kell indítanod – és futhat nálad úgy, hogy másnak még nincs közzétéve.

Mit jelent a záróüzenet?

HA EZT ÍRJA…	…EZ AZT JELENTI	A TE DOLGOD
„Elkészült az alkalmazás."	Létrehozta vagy módosította a fájlokat.	Nézd meg, hol vannak, és hogyan használhatók.
„A tesztek sikeresek."	Bizonyos ellenőrzések lefutottak.	Kérdezd meg, mit ellenőrzött; a fő feladatot próbáld ki te is.
„Indítás: `npm run dev`"	Ezzel a paranccsal indul a program.	A projekt mappájában add ki (11. lecke).
„Megnyitás: `localhost:5173`"	Itt érhető el a felület, amíg a program fut.	Nyisd meg a böngészőben.
„README" vagy útmutató	Leírás az indításról és a működésről.	Mentsd el – holnap is kelleni fog.
„Még kell egy API-kulcs / fiók."	Egy funkcióhoz további beállítás kell.	Tisztázd, mi működik már, és mi vár erre.

Milyen eredményt kaptál?

Önálló .html fájl
Dupla kattintással megnyílik a böngészőben. A legegyszerűbb eset.

Helyben futó program
Előbb el kell indítani (pl. `npm run dev`), utána a kiírt címen nyílik meg.

Indítófájl
`start.command` (Mac) vagy `start.bat` (Windows) – dupla kattintás.

Platform-előnézet
„Preview" gomb – pl. a Lovable-ben. Kérdezd meg, ki érheti el.

Közzétett webcím
Már szerveren van, egy linkkel bárki megnyithatja.

Fájl az eredmény
PDF, videó, táblázat – megnyitod a hozzá való programmal.

Kérj rendes átadást

Ha az agent csak annyit írt, hogy „kész", kérj tőle **laikusoknak szóló átadási útmutatót**, és mentesd el a projekt **README.md** fájljába. Ez a projekted használati utasítása – holnap, jövő héten, vagy ha valakinek továbbadod, ebből fogsz dolgozni.

ÁTADÁSI ÚTMUTATÓ KÉRÉSE — a munkamenet végén

```
A fejlesztés végén adj laikusoknak szóló átadási útmutatót a ténylegesen elkészült projekthez.

Írd le:
1. Mi készült el, és hol találhatók a fájljai?
2. Milyen programokat kell először telepítenem?
3. Mi az első indítás pontos menete?
4. Melyik mappában, milyen parancsokat futtassak, és mit kell látnom utánuk?
5. Hogyan nyitom meg az eredményt, és melyik ablakot kell nyitva hagynom?
6. Hogyan állítom le, és hogyan indítom újra másnap?
7. Hová menti az adatokat, és hogyan készítek róluk biztonsági másolatot?
8. Mit teszteltél ténylegesen, és mi nincs még bekötve?
9. Mihez kell fiók, API-kulcs vagy fizetős szolgáltatás?

Különítsd el az egyszeri előkészítést a későbbi indítástól.
Mentsd az útmutatót a projekt README.md fájljába is.
```

A localhost titka

Ami böngészőben fut, az első körben **nincs fent az interneten**. A kód a te gépeden van, és egy kis helyi webszerver futtatja. A cím, amit kapsz – például `http://localhost:5173` – ezt jelenti:

`http://localhost:5173`

`http://`
a böngésző így kéri az oldalt

`localhost`
= ez a gép, ahol éppen vagy

`:5173`
a port – a program „ajtószáma"

Egy localhost cím három része. A port száma programonként más lehet (3000, 5173, 8000…) – mindig a kiírtat nyisd meg.

FIGYELEM
Ne küldd el a localhost linket másnak! Nála a „localhost" az ő saját gépét jelenti, ahol nincs ott a programod. Telefonon megnyitva is a telefonra mutat. Ha mások is használnák, publikálni kell (17. lecke).

A LEGEGYSZERŰBB KÉRÉS — ha nem tudod, hogyan nézd meg

```
Indítsd el és mutasd meg.
```

KULCSGONDOLATOK

1. Három külön állapot: kész a kód → fut a program → mások is elérik.
2. Kérj laikusoknak szóló átadási útmutatót, és mentsd a README.md-be.
3. A localhost = a saját géped. A link másnál nem működik – ahhoz publikálni kell.

PRÓBÁLD KI

1. Kérd az átadási útmutatót a saját projektedhez, és olvasd el.
2. Kérd: „Indítsd el és mutasd meg.” Nyisd meg a kapott localhost címet a böngésződben.

11

III. RÉSZ · ÉPÍTÉS · 11. LECKE

A terminál, félelem nélkül

Amikor az AI parancsot ad, és neked kell beírnod



EBBEN A LECKÉBEN

- Megérted, mi a terminál, és hogyan nyitod meg.
- Megtanulod, hogyan ismered fel és hogyan másolod be az AI által adott parancsokat.
- Megtanulod az első indítást, a leállítást és a másnapi újraindítást.

kb. 11 perc olvasás

Videó: 11. rész

3 kulcsgondolat

A **terminál** az a jellemzően sötét ablak, ahol villogó kurzor után szöveges parancsokat írhatsz be: programot indíthatsz, fájlokat kezelhetsz, szoftvert telepíthetsz. A filmekben hekkerek használják – a valóságban egy egyszerű eszköz, amelyből neked mindössze **négy mozdulatot** kell ismerned.

JÓ TUDNI

Sok agent maga is futtatja a parancsokat, így lehet, hogy soha nem kell terminált nyitnod. De ha egyszer azt kéri: „Futtasd ezt a terminálban”, jó, ha nem ijedsz meg.

Hogyan nyitod meg?

RENDSZER	ÍGY NYITOD MEG
Mac	Command + Szóköz → írd be: **Terminal** → Enter.
Windows	Start menü → írd be: **PowerShell** (vagy Windows Terminal) → kattints a találatra.

Amikor az AI parancsot ad

A vibe coding beszélgetésben gyakran kapsz ilyen választ: „Futtasd a következő parancsokat a terminálban.” A parancsok ilyenkor **külön, szürke vagy sötét kódblokkban** jelennek meg – gyakran egy „Másolás” gombbal a sarkukban.

Coding agent – a válasz

TE

Hogyan indítom el a programot a saját gépemen?

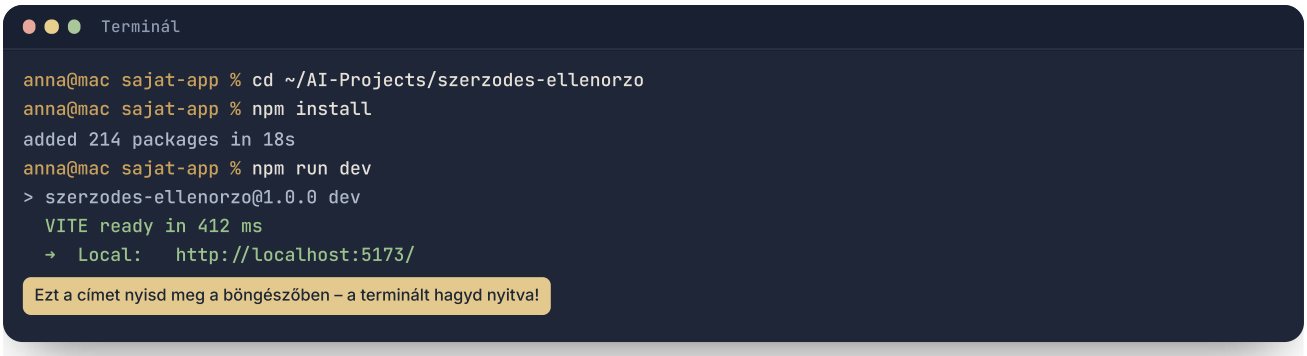
AI

Nyiss terminált, és futtasd egymás után ezt a három parancsot:

```
cd ~/AI-Projects/szerzodes-ellenorzo
npm install
npm run dev
```

Utána nyisd meg a kiírt címet a böngészőben.

Az AI által adott parancsok kódblokkban érkeznek. Ezeket kell – pontosan, soronként – bemásolnod.



Ugyanez a terminálban. A sor elején lévő jelet (itt: `anna@mac sajat-app %`, Windowson `PS C:\…>`) a terminál írja ki – nem kell beírnod.

Az aranyszabályok

1. **Soronként haladj**
   Egy parancs → Enter → várd meg, amíg újra megjelenik a parancssor (a villogó kurzor az új sor elején). Csak utána jöhet a következő.
2. **Pontosan azt másold, amit kaptál**
   A „Másolás" gombbal a legbiztosabb. A sor eleji `$` vagy `>` jelet ne másold be – az csak jelzés.
3. **Ismeretlen parancsot ne futtass vakon**
   Ha `sudo` (rendszergazdai jog) vagy `rm` (törlés) van benne, állj meg, és kérdezd meg az AI-t, mit csinál.
4. **Ha hibát kapsz, ne próbálgass**
   Másold ki a hibaüzenetet, és add vissza az AI-nak (lent a minta).

A négy mozdulat, ami elég

`cd` = lépj be a mappába
A terminál mindig egy mappában „áll". A parancsot a projekt mappájában kell kiadni: `cd "mappa-útvonala"`. Tipp: Macen a mappát a Finderből a terminálba húzva beíródik az útvonala.

`ls` / `dir` = mi van itt?
Kilistázza a mappa tartalmát (Mac: `ls`, Windows: `dir`). A jó mappában ott a `README.md` vagy a `package.json`.

Enter = futtatás
A beírt parancs csak Enterre indul el.

Ctrl + C = leállítás
Leállítja a futó programot (Macen is a Control billentyű!).

Indítás, leállítás, másnap

MIKOR?	MIT CSINÁLSZ?
Első alkalommal	Terminál → `cd` a projektmappába → `npm install` (telepíti a szükséges csomagokat, csak egyszer kell) → `npm run dev`.
Használat közben	A terminál „nem ér véget" – ez normális, a szerver fut. Hagyd nyitva, és válts a böngészőre.
Leállítás	Kattints a terminálba, és nyomj **Ctrl + C**-t. A böngészőlap bezárása nem állítja le a programot.
Másnap	Terminál → `cd` a projektmappába → `npm run dev`. Az `npm install` általában nem kell újra.

TIPP
Kényelmesebb indítást is kérhetsz: „Készíts az én rendszeremhez indítófájlt, hogy dupla kattintással induljon." Ilyenkor terminált sem kell nyitnod.

Ha elakadsz

HA HIBÁT KAPSZ – EZT KÜLDD VISSZA — a hibaüzenettel együtt

```
Mac / Windows rendszeren próbálom indítani. Ebben a mappában vagyok: […]. Ezt a parancsot futtattam: […]. Ezt az üzenetet kaptam: […]. Mondd meg a következő konkrét lépést, és azt is, milyen eredményt kell látnom, ha sikerült.
```

FIGYELEM

A hibaüzenetbe soha ne másolj jelszót vagy API-kulcsot – a legtöbb indítási hiba megértéséhez nincs rájuk szükség.

A leggyakoribb elakadások és megoldásuk a **D. mellékletben** (Hibaelhárító) találhatók.

KULCSGONDOLATOK

1. A terminál egy egyszerű eszköz: cd, ls/dir, Enter, Ctrl + C – ennyi elég.
2. Az AI parancsait kódblokkból, pontosan, soronként másold be; várd meg, amíg végez.
3. Hibánál ne próbálgass: add vissza az üzenetet az AI-nak, és kérd a következő lépést.

PRÓBÁLD KI

1. Nyiss terminált, lépj be a projektmappádba (`cd`), és listázd ki a tartalmát (`ls` vagy `dir`).
2. Indítsd el a programodat a README alapján, nyisd meg a böngészőben, majd állítsd le Ctrl + C-vel.
3. Indítsd el újra – most már egyedül.

12

III. RÉSZ · ÉPÍTÉS · 12. LECKE

Tesztelés és hibajavítás

A „Kész.” még nem kész



EBBEN A LECKÉBEN

- Megtanulod, hogyan teszteltesd végig a rendszert az AI-val.
- Kapsz egy egyszerű ellenőrzőlistát a saját kipróbáláshoz.
- Megtanulod, mit tegyél, ha az AI körbe-körbe javít.

kb. 8 perc olvasás
Videó: 12. rész
3 kulcsgondolat

Az első működő képernyő még nem jelenti, hogy a rendszer valóban kész. Az AI hajlamos optimista lenni: „Minden működik!” – miközben egy ritkábban használt gomb hibát dob. Ezért a tesztelés két részből áll: **az AI teszteli a teljes rendszert**, és **te kipróbálod a fő útvonalat**.

Teszteltesd végig az AI-val

TELJES TESZT KÉRÉSE — az első változat után

```
Most teszteld végig a teljes rendszert a specifikáció alapján. Ellenőrizd minden funkcióját, keresd meg a hibákat, javítsd ki őket, majd ellenőrizd újra. A végén írd le röviden: mit teszteltél, mit javítottál, és mi maradt nyitva.
```

A jó coding agent ilyenkor elindítja a programot, végigkattintja a funkciókat, és a talált hibákat kijavítja. Kérd meg, hogy **javítás után újra ellenőrizzen** – egy javítás néha újabb hibát okoz.

JÓ TUDNI

A „tesztek sikeresek” hasznos, de kérdezd meg, mit jelent: automatikus kódellenőrzést, böngészős kipróbálást vagy valódi külső szolgáltatással végzett próbát? Egy bemutatóadatokkal működő képernyő és egy teljesen bekötött funkció nem ugyanaz.

A saját kipróbálásod – hat kérdés

Megnyílik?
A megfelelő oldal vagy alkalmazás jelenik meg?

A fő művelet megy?
Be tudsz vinni, számolni, feltölteni, menteni?

Jó az eredmény?
A várt eredményt kapod, és érthető, hol találod?

Hibás bemenet?
Egy üres kötelező mező érthető üzenetet ad?

Újraindítás után?
Leállítás és újraindítás után megvannak az adataid?

Telefonon?
Ha mobilon is kell, ott is kipróbáltad?

> Próbálj ki egy teljes feladatot egyszerű, kitalált adatokkal – úgy, ahogy a felhasználó tenné.

Ha hibát találsz

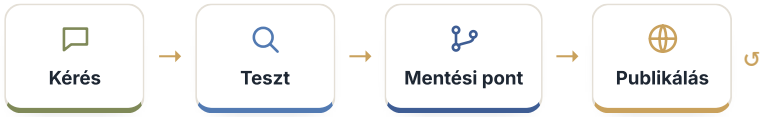
HIBAJELENTÉS képernyőképpel

```
Hibát találtam. Ezt csináltam: […]. Ezt vártam: […]. Ehelyett ez történt: […]. (Képernyőkép csatolva.) Keresd meg az okát, javítsd ki, és ellenőrizd, hogy a javítás mást nem rontott el.
```

Ha az AI körbe-körbe javít

Előfordul, hogy ugyanaz a hiba harmadszor is visszajön, vagy minden javítás mást ront el. Ilyenkor ne erőltesd tovább ugyanabban a beszélgetésben:

1. **Állj meg, és kérj magyarázatot**
   „Mielőtt újra javítasz: mi szerinted a hiba oka? Milyen megoldásokat látsz?"
2. **Nyiss új munkamenetet**
   A régi beszélgetés tele van félresikerült kísérletekkel. Egy friss munkamenet, rövid hibaleírással, gyakran elsőre megoldja.
3. **Válts erősebb modellre vagy magasabb erőfeszítésre**
   A makacs hibák gyakran több „gondolkodást" igényelnek.
4. **Térj vissza egy működő változathoz**
   Ha van mentési pontod (13. lecke), egy mozdulattal visszaállhatsz, és máshogy kérheted.



A vibe coding ciklusa: kérés → teszt → mentési pont → (publikálás) – és újra.

KULCSGONDOLATOK

1. A „Kész." nem kész: kérd az AI-t, hogy a specifikáció alapján tesztelje végig a rendszert.
2. Te is próbáld ki a fő útvonalat – hat egyszerű kérdéssel.
3. Ha körbe-körbe javít: állj meg, új munkamenet, erősebb modell, vagy vissza egy működő változathoz.

PRÓBÁLD KI

1. Add ki a teszt-promptot a saját projektedben, és olvasd el az összefoglalót.
2. Menj végig a hat kérdésen. Minden talált hibát a hibajelentés-mintával küldj vissza.

IV. RÉSZ

IV.

Tovább

Mentés, kapcsolatok, adatok, telefon, internet – amikor a projekt komolyra fordul.

13 Mentési pontok: Git és GitHub

14 API, AI a programban és a titkok

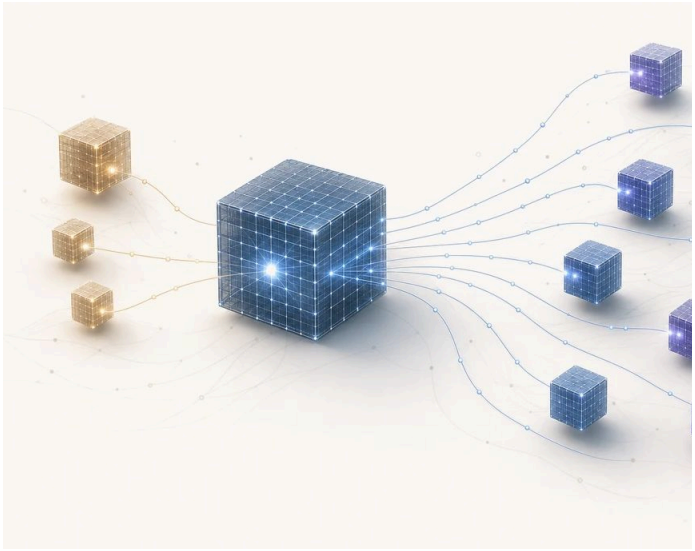
15 Adatbázis

16 PWA – app a telefonodon

17 Publikálás: hogyan lesz belőle élő oldal?

18 Haladó eszköztár

19 Az első saját projekted



13

IV. RÉSZ · TOVÁBB · 13. LECKE

Mentési pontok: Git és GitHub

Hogy bátran kísérletezhess – mindig legyen visszaút



EBBEN A LECKÉBEN

- Megérted, mi a Git és a GitHub, és miért fontos még kezdőként is.
- Megismered az öt alapfogalmat: repo, commit, push, pull, branch.
- Megtanulod, mikor és hogyan kérj mentési pontot az AI-tól.

kb. 9 perc olvasás

Videó: 13. rész

3 kulcsgondolat

Képzeld el, hogy a programod tökéletesen működik, és kérsz egy nagy változtatást. Az AI nekiáll – és utána semmi nem működik. Mi a megoldás? Ha van **mentési pontod**, egy mozdulattal visszaállsz a működő változatra. Ezt tudja a **Git**.

Git = verziókövetés a gépeden
Mint egy játék mentései: minden „commit" egy elmentett, visszaállítható állapot, névvel és időponttal.

GitHub = a kódod otthona a felhőben
A mentési pontjaid távoli példánya. Innen indul a publikálás, és innen tudsz más gépen is dolgozni.

Az öt alapfogalom

FOGALOM	EGYSZERŰEN
Repository (repo)	Maga a Git által kezelt projekt – a mappád, a teljes előzményeivel.
Commit	Egy elmentett állapot, rövid leírással: „Működik a PDF-export."
Push	A gépeden lévő commitok feltöltése a GitHubra.
Pull	A GitHubon lévő új változások letöltése a gépedre.
Branch	A projekt egy külön fejlesztési ága – kísérletezéshez, a fő változat érintése nélkül.

A jó hír: **a Git-parancsokat neked nem kell ismerned**. Az agent elvégzi őket, ha megkéred. A te dolgod annyi, hogy tudd, **mikor** kérj mentést.

Mikor kérj mentési pontot?

- **Ha valami jól működik** – mielőtt bármi újba kezdesz.
- **Nagy módosítás előtt** – új funkció, átszervezés, dizájncsere.
- **A munkanap végén** – hogy holnap tiszta lappal folytathasd.

MENTÉSI PONT KÉRÉSE — nagy módosítás előtt

```
A jelenlegi verzió működik. Készíts egy Git commitot erről az állapotról, és töltsd fel (push) a GitHubra. Csak utána kezdd el az új funkciót.
```

VISSZAÁLLÁS — ha valami elromlott

```
Az utolsó módosítás óta nem működik a program. Állítsd vissza a projektet az utolsó működő commitra, és mondd el, mi változott azóta.
```



A biztonságos ritmus: ami működik, azt elmented – utána jöhet az új.

GitHub-fiók és privát tároló

A GitHub ingyenes alapcsomaggal is használható. Regisztrálj a github.com-on, és kérd az agentet, hogy hozzon létre egy **privát** tárolót (repository) a projektnek – a privátot csak te látod. Az első összekapcsolásnál a böngészőben kell majd engedélyezned a hozzáférést; az agent végigvezet rajta.

> **FIGYELEM**
> A GitHubra a kód kerül, a titkok nem. Az API-kulcsok és jelszavak a .env fájlban maradnak, amely nem töltődik fel (14. lecke).

> **JÓ TUDNI**
> A GitHub nem klasszikus biztonsági mentés. Fontos projektnél – különösen, ha adatokat is tárol – legyen külön backup is.

KULCSGONDOLATOK

1. A commit egy visszaállítható mentési pont; a GitHub a projekted felhőbeli otthona.
2. A Git-parancsokat az agent végzi – a te dolgod tudni, mikor kérj mentést.
3. Ami működik, azt elmented; utána jöhet a nagy változtatás.

PRÓBÁLD KI

1. Regisztrálj a github.com-on (ha még nincs fiókod).
2. Kérd az agentet: „Hozz létre a projektnek privát GitHub-tárolót, és mentsd el a jelenlegi állapotot.”

14

IV. RÉSZ · TOVÁBB · 14. LECKE

API, AI a programban és a titkok

Amikor a programod más rendszerekkel beszélget



EBBEN A LECKÉBEN

- Megérted, mi az API, egy hétköznapi hasonlattal.
- Eldöntöd, kell-e a programodba „futás közbeni" AI.
- Megtanulod biztonságosan kezelni az API-kulcsokat (.env).

kb. 10 perc olvasás

Videó: 14. rész

3 kulcsgondolat

Mi az API?

Az **API** egy **kapu**, amelyen át a programod más rendszerekkel beszélget. Olyan, mint egy étterem pultja: nem mész be a konyhába, hanem leadod a rendelést egy szabványos formában, és megkapod az ételt. Így tud a programod levelet küldeni Gmailből, időpontot foglalni a naptárban, számlát kiállítani vagy árfolyamot lekérdezni – anélkül, hogy ezeket neked kellene megírnod.



Az API-hívás menete: kérés → kapu → másik rendszer → válasz.

AI a kész programban – kell-e?

Fontos különbség: attól, hogy a programot AI-val készítetted, **a kész programnak nem kell AI-t használnia**. Az AI lehetett kizárólag a fejlesztő. Egy kalkulátor, egy nyilvántartó vagy egy weboldal AI nélkül fut – olcsón és kiszámíthatóan.

Akkor kell **futás közbeni AI**, ha a program maga végez „értelmes" szöveges vagy képi munkát: összefoglal egy feltöltött szerződést, kategorizálja a leveleket, válaszol az ügyfelek kérdéseire, elemez egy képet. Ilyenkor a programod az API-n keresztül egy nyelvi modellnek küldi a feladatot.

AI csak fejlesztőként

- Kalkulátor, nyilvántartó, weboldal
- Nincs futási AI-költség
- Kiszámítható működés

VS

AI a programban is

- Összefoglal, kategorizál, válaszol
- Használatarányos költség
- API-kulcs kell hozzá

Az API-kulcs = jelszó

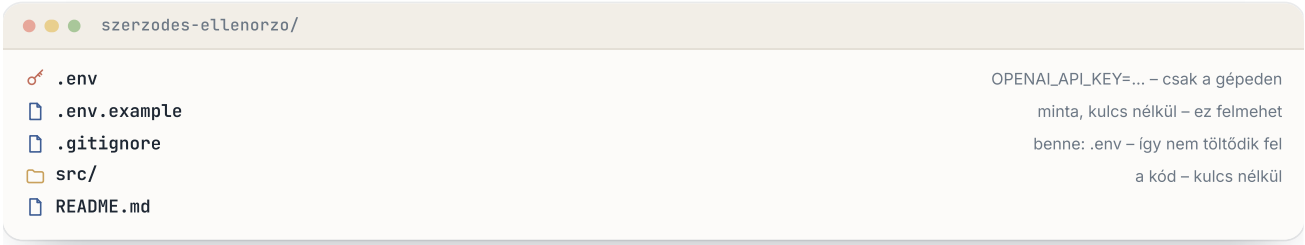
Az API-k többsége **kulccsal** azonosítja a programodat: ez egy hosszú, titkos karaktersor, amelyet a szolgáltató oldalán hozol létre (például OpenAI: platform.openai.com, Anthropic: platform.claude.com). A használat általában **külön számlázódik** az előfizetésedtől, használat alapján.

FIGYELEM

Az API-kulcs egy jelszónak megfelelő titok. Aki megszerzi, a te számládra használhatja. Soha ne küldd el senkinek, ne tedd a chatbe, ne írd a specifikációba, és soha ne kerüljön a böngészőben futó (frontend) kódba.

A biztonságos hely: a .env fájl

A titkos adatoknak egy külön fájl jár a projektmappában: a **.env**. A program innen olvassa ki őket, ezért a kódban nem szerepelnek. A **.gitignore** nevű fájl pedig gondoskodik róla, hogy a .env soha ne kerüljön fel a GitHubra.



A titkok helye. A .env a gépeden marad; a kód csak a kulcs nevére hivatkozik.

KÉRD ÍGY — mielőtt kulcsot használnál

```
Hozd létre a szükséges .env fájlt és a .env.example mintafájlt, és gondoskodj róla, hogy a .env szerepeljen a .gitignore-ban. Mondd meg, melyik sorba írjam be a kulcsomat – a kulcsot én magam írom be, ne kérd el a chatben.
```

Négy szabály

- **Projektenként külön kulcs** – ha egy kiszivárog, csak azt kell cserélni.
- **Költési limit** a szolgáltató oldalán – így egy hiba sem okozhat nagy számlát.
- **Publikált alkalmazásnál** a kulcs a tárhely titkos beállításai közé kerül (17. lecke), nem fájlba.
- **Ha kiszivárgott**: azonnal töröld a szolgáltató oldalán, és hozz létre újat.

KULCSGONDOLATOK

1. Az API egy kapu, amelyen át a programod más rendszerekkel beszélget.
2. A kész programnak nem kell AI – csak ha maga végez szöveges vagy képi „gondolkodó” munkát.
3. Az API-kulcs jelszó: .env fájlba, .gitignore mögé, projektenként külön, költési limittel.

PRÓBÁLD KI

1. Döntsd el a saját projektednél: kell-e futás közbeni AI? Ha nem, ez a lecke most kimaradhat.
2. Ha kell: hozz létre kulcsot a szolgáltatónál költési limittel, és kérd az agentet, hogy készítse elő a .env fájlt.

15

IV. RÉSZ · TOVÁBB · 15. LECKE

Adatbázis

A program tartós memóriája – és mikor kell



EBBEN A LECKÉBEN

- Megérted, mi az adatbázis, és mikor van rá szükség.
- Megismered a négy tárolási módot és a veszélyeiket.
- Megtudod, melyik megoldással érdemes kezdeni.

kb. 7 perc olvasás

Videó: 15. rész

3 kulcsgondolat

Az **adatbázis** a program tartós memóriája. Ami bekerül – egy ügyfél, egy számla, egy beállítás –, az a program bezárása után is megmarad. Rendezett táblákban tárolódik, ezért gyorsan kereshető, szűrhető és összesíthető.

Kell-e adatbázis?

Általában nem kell

- Kalkulátor, átváltó
- Bemutató vagy prezentációs oldal
- Egyszeri feldolgozó eszköz (fájl be, fájl ki)
- Játék pontszám-mentés nélkül

VAGY

Általában kell

- Tartósan tárolsz adatot
- Többen dolgoznak ugyanazzal
- Van bejelentkezés
- Több eszközről kell elérni
- Publikus webalkalmazás

> Ha bizonytalan vagy, kérdezd meg az agentet: „Kell ehhez adatbázis? Mi a legegyszerűbb megoldás?”

Hová mentődnek az adatok?

TÁROLÁSI MÓD	MIT JELENT?	MIRE FIGYELJ?
Csak a futó program memóriája	Amíg fut, megvan.	Újraindításkor elveszhet minden.
A böngészőben	A böngésző „zsebében” tárolja.	Másik böngészőben vagy gépen nem látszik; törléskor elvész.
Helyi fájl vagy adatbázis (pl. SQLite)	Egy fájl a projektmappában.	Erről neked kell biztonsági másolatot készíteni.
Felhős adatbázis (pl. Supabase)	Egy szolgáltató szerverén.	Fiók kell; a szolgáltató díjai és szabályai számítanak.

Mivel kezdj?



Az adatbázis-lehetőségek egyszerűtől a haladóig.

TIPP

A training-gyakorlatainkban ezt a mondatot tesszük minden vázlat végére: „A rendszer legyen egyszerű és gyorsan elkészíthető. Ha adatbázis kell hozzá, használj kis, helyi adatbázist (pl. SQLite).”

JÓ TUDNI

A felhős szolgáltatók ingyenes csomagjai korlátozottak, az árak gyakran változnak – mindig a szolgáltató oldalán nézd meg az aktuálisat.

Ha felhős adatbázist választasz, a hozzáférési adatokat (címet, kulcsot) ugyanúgy a **.env** fájlba kell tenni, mint az API-kulcsot. Innentől az agent úgy dolgozik az adatbázissal, mint a kód többi részével: megtervezi a táblákat, megírja a mentést és a lekérdezéseket.

Ne felejtsd: mentés

Az adat értékesebb a kódnál: a kódot az AI újraírja, az elveszett ügyféllistát nem. Kérd meg az agentet, hogy írja le a README-be, **hol vannak az adatok, és hogyan készítesz róluk biztonsági másolatot** – és próbáld is ki egyszer a visszaállítást.

KULCSGONDOLATOK

1. Az adatbázis a program tartós memóriája – nem minden programhoz kell.
2. Saját gépre SQLite, webalkalmazáshoz felhős adatbázis (pl. Supabase); a saját szerver haladóknak való.
3. Az adat értékesebb a kódnál: legyen róla mentés, és tudd, hogyan állítod vissza.

PRÓBÁLD KI

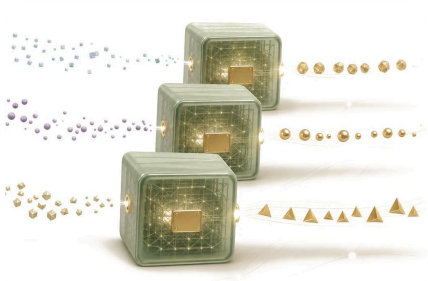
1. Kérdezd meg az agentet a saját projektedben: „Hová menti a program az adatokat? Mi történik, ha leállítom és újraindítom?”
2. Kérd, hogy írja le a README-be a biztonsági mentés menetét.

16

IV. RÉSZ · TOVÁBB · 16. LECKE

PWA – app a telefonodon

Weboldalból telepíthető alkalmazás, app store nélkül



EBBEN A LECKÉBEN

- Megérted, mi a PWA, és miben más, mint egy „igazi” app.
- Megtudod, mikor jó választás.
- Megkapod a promptot, és megtanulod a telepítést iPhone-on és Androidon.

kb. 6 perc olvasás

Videó: 16. rész

3 kulcsgondolat

A **PWA** (Progressive Web App, progresszív webalkalmazás) olyan weboldal, amely **úgy viselkedik, mint egy telefonos alkalmazás**: ikonja van a kezdőképernyőn, teljes képernyőn nyílik meg böngészősáv nélkül, és akár internet nélkül is elindul. Mindezt **app store nélkül** – nincs jóváhagyás, nincs fejlesztői fiók, nincs külön Android- és iPhone-változat.

Natív app (App Store, Google Play)

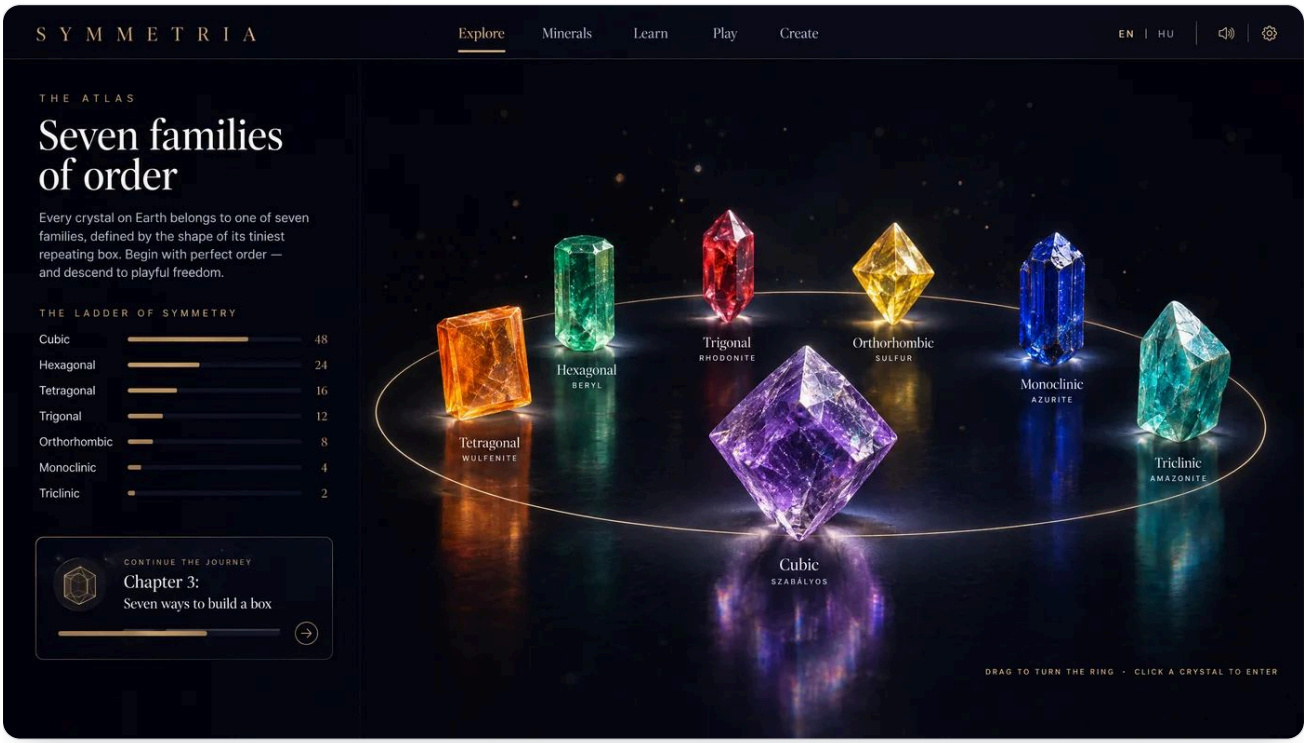
- Külön fejlesztés iPhone-ra és Androidra
- Áruházi jóváhagyás, fejlesztői fiók
- Minden telefonos funkcióhoz hozzáfér

VS

PWA

- Egyetlen weboldal, minden eszközre
- Egy link – a felhasználó maga telepíti
- Ikon, teljes képernyő, offline működés

> A legtöbb saját eszközhöz és belső alkalmazáshoz a PWA bőven elég.



> Valódi példa: a Symmetria 3D kristálytani alkalmazás PWA – telepíthető, és offline is működik.

Mikor jó választás?

- Ha a programot **telefonon is** használnák – terepen, úton, ügyfélnél.
- Ha **gyorsan, egy linkkel** akarod eljuttatni a felhasználókhoz.

- Ha fontos, hogy **gyenge vagy hiányzó internet mellett is** elinduljon.

> **JÓ TUDNI**
> Korlátok: néhány telefonos funkció (például bizonyos értesítések iPhone-on, vagy a háttérben futás) a PWA-ban korlátozott. Ha ilyenre van szükség, kérdezd meg az agentet, megoldható-e.

Így kérd

PWA-VÁ ALAKÍTÁS — a kész alkalmazáshoz

```
Alakítsd az alkalmazást telepíthető PWA-vá: legyen ikonja és neve a kezdőképernyőn, teljes képernyőn nyíljon meg, és az alapfunkciói internet nélkül is induljanak el. Teszteld, és írd le a README-be, hogyan telepíthető iPhone-on és Androidon.
```

> **FIGYELEM**
> A PWA-hoz a programnak **publikálva kell lennie**, biztonságos (https) címen – localhostról a telefon nem tudja telepíteni. Erről a következő lecke szól.

Telepítés a telefonra

TELEFON	LÉPÉSEK
iPhone (Safari)	Nyisd meg a címet Safariban → Megosztás gomb (négyzet felfelé nyíllal) → „Főképernyőhöz adás” → Hozzáadás.
Android (Chrome)	Nyisd meg a címet Chrome-ban → jobb felső menü (három pont) → „Alkalmazás telepítése” vagy „Hozzáadás a kezdőképernyőhöz”.

KULCSGONDOLATOK

1. A PWA telepíthető weboldal: ikon, teljes képernyő, offline indulás – app store nélkül.
2. Egy prompttal kérheted; a telepítéshez publikált, https-es cím kell.
3. A legtöbb saját eszközhöz bőven elég; különleges telefonos funkcióknál kérdezz rá.

PRÓBÁLD KI

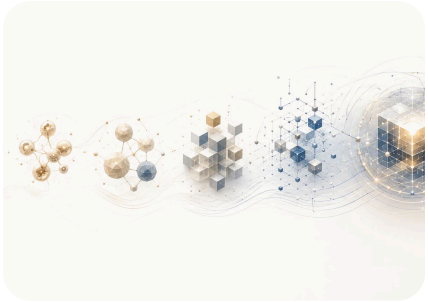
1. Ha a projekted telefonon is jól jönne, kérd a PWA-promptot – és a publikálás után (17. lecke) telepítsd a saját telefonodra.

17

IV. RÉSZ · TOVÁBB · 17. LECKE

Publikálás: hogyan lesz belőle élő oldal?

A képernyőképtől a saját domainig



EBBEN A LECKÉBEN

- Eldöntöd, mire van szüksége annak, akinek a programot átadnád.
- Megismered a publikálás lehetőségeit, az egyszerűtől a haladóig.
- Megérted a hosting, a domain és a https szerepét.

kb. 11 perc olvasás

Videó: 17. rész

3 kulcsgondolat

A programod a gépeden fut, a localhost címen. Ha mások is használnák, két kérdés van: **mire van szükségük**, és **hol fusson** a program.

Először: mit szeretne a másik ember?

HA A MÁSIK EMBER…	EZT ADD ÁT	MEGJEGYZÉS
…csak megnézné, mi készült	Képernyőképek, rövid videó vagy önálló bemutató HTML	Nincs telepítés, nincs szerver.
…a saját gépén használná	A teljes projekt (pl. ZIP) + README	Telepítési és indítási útmutatóval.
…linkkel, telepítés nélkül használná	Közzétett alkalmazás, működő címmel	Tisztázd, kell-e belépés, és ki fér hozzá.
…továbbfejlesztené	Forráskód (GitHub), README, .env.example	Írd le az ismert korlátokat is.

⚠ **FIGYELEM**

Átadás előtt a személyes kulcsokat (.env) és az átadáshoz nem szükséges adatokat hagyd ki a projektből.

A publikálás lehetőségei

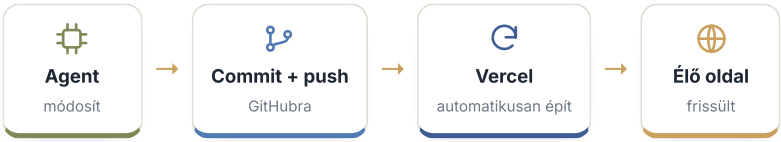


A publikálás lépcsője. Kezdőként a középső fokokkal érdemes indulni.

Statikus vagy „igazi” webalkalmazás?

A **statikus oldal** csak HTML, CSS és a böngészőben futó JavaScript: nincs saját szervere, adatbázisa, titkos kulcsa. Ilyen egy bemutató, egy kalkulátor, egy portfólió. Ezekhez a **GitHub Pages** ingyenes és egyszerű.

Ha a programnak **szerveroldali része, adatbázisa vagy API-kulcsa** van, webalkalmazás-tárhely kell: kezdőként a **Vercel** a legkényelmesebb. Összekötöd a GitHubbal, és minden mentési pont (push) után **magától** elkészíti és kiteszi az új verziót.



> Az automatikus publikálás: te csak a mentést kéred – a többi magától történik.

PUBLIKÁLÁS KÉRÉSE — ha mások is használnák

```
Szeretném publikálni az alkalmazást, hogy mások is elérjék egy linken. Javasold a projekthez illő legegyszerűbb megoldást (GitHub Pages vagy Vercel), és vezess végig lépésről lépésre. Az API-kulcsok a tárhely titkos beállításai közé kerüljenek, ne a kódba.
```

Hosting, domain, https – három szó

Hosting (tárhely)
Egy szerver, amely folyamatosan futtatja a programot – nem a te laptopodon kell futnia.

Domain
A cím, amit a böngészőbe írnak (pl. cegem.hu). Évente bérelhető, és beállítod, hogy a tárhelyre mutasson.

https
A biztonságos kapcsolat lakatja. A modern tárhelyek automatikusan adják.

A Vercel és a GitHub Pages ingyen is ad egy címet (például `sajatprojekt.vercel.app`). Saját domaint később is köthetsz hozzá.

Publikálás után

- **Teszteld az élő oldalt is** – ismételd meg a 12. lecke kipróbálását élesben, telefonon is.
- **Gondold át, ki fér hozzá.** Ha a programban érzékeny adat van, kell bejelentkezés.
- **A ciklus innentől:** kérés → teszt → mentési pont → automatikus publikálás.

KULCSGONDOLATOK

1. Előbb azt döntsd el, mire van szüksége a másiknak: megnézni, helyben futtatni, linken használni vagy továbbfejleszteni.
2. Statikus oldalhoz GitHub Pages, webalkalmazáshoz Vercel – a GitHubról automatikusan frissül.
3. Hosting = ahol fut, domain = a címe, https = a biztonságos kapcsolat.

PRÓBÁLD KI

1. Döntsd el, kinek és hogyan adnád át a projektedet.
2. Ha linkkel kell elérhetővé tenni: add ki a publikálás-promptot, és kövesd az agent lépéseit. Utána nyisd meg az élő címet a telefonodon is.

18

IV. RÉSZ · TOVÁBB · 18. LECKE

Haladó eszköztár

Szabálykönyv, projektmemória, skillek és MCP

EBBEN A LECKÉBEN

- Megismered azokat az eszközöket, amelyekkel a második, harmadik projekted már gyorsabb lesz.
- Megérted, mi a projektmemória-fájl, a skill és az MCP.
- Tudod, hol folytasd a tanulást.

kb. 8 perc olvasás

Videó: 18. rész

3 kulcsgondolat

Az első projekted után észre fogod venni, hogy sok mindent minden alkalommal újra elmondasz. A haladó eszközök arról szólnak, hogy **a tudásodat egyszer leírd, és az AI mindig magától használja**.

1. A szabálykönyv (RULES.md)

A 8. leckében már összeállítottad. A DarwinAI szabálykönyve 12 területen 46 szabályt gyűjt össze: terv és szerkezet, dizájn és márka, használhatóság, felhasználók és admin, üzemeltetés, biztonság és jog, mérés és költség, AI a termékben, láthatóság, kommunikáció, leadek, minőség és átadás. Ahogy tapasztalatot gyűjtesz, bővítsd a sajátodat.



A darwinai.hu/vibe-coding/szabalykonyv – mind a 46 szabály, magyarázattal.

2. A projektmemória (CLAUDE.md, AGENTS.md)

A coding agentek minden munkamenet elején elolvasnak egy speciális fájlt a projektmappában – Claude Code-ban ez a **CLAUDE.md**, Codexben az **AGENTS.md**. Ide kerül, amit az agentnek a projektről mindig tudnia kell: mire való, hogyan indul, mire figyeljen, mit ne csináljon. Így egy új munkamenet sem „felejt".

PROJEKTMEMÓRIA KÉRÉSE — a projekt egy későbbi pontján

```
Készíts a projekthez egy CLAUDE.md (vagy AGENTS.md) fájlt: röviden írd le, mire való a rendszer, hogyan indul, milyen fontos döntéseket hoztunk eddig, és mire figyeljen egy új munkamenet. Tartsd naprakészen, ha fontos dolog változik.
```

3. Skillek

A **skill** egy újrahasznosítható szakértelem-csomag az AI-nak: egy mappa leírással, szabályokkal és mintákkal. Egyszer leírod benne, hogyan kell valamit jól csinálni – például a céged arculatát követni, jelentést írni vagy weboldalt tervezni –, és az agent a megfelelő pillanatban magától előveszi. Így a saját módszered másolható és átadható.

4. MCP – csatlakozók a külvilághoz

Az **MCP** (Model Context Protocol) egységes csatlakozó az AI-ügynök és a külső eszközök között – olyan, mint az USB. Ezen át tud az agent például a naptáradba nézni, egy adatbázisba írni, böngészőt vezérelni vagy egy tervezőprogramot használni. Az agent beállításaiban kapcsolhatod be őket.

Szabálykönyv
Minden projektre érvényes elvárásaid.

Projektmemória
Amit az agentnek erről a projektről mindig tudnia kell.

Skill
Újrahasznosítható szakértelem: „így csináljuk mi".

MCP
Csatlakozó a naptárhoz, adatbázishoz, böngészőhöz.

A haladó eszköztár négy eleme. Mindegyik ugyanazt szolgálja: amit egyszer jól leírtál, azt az AI mindig használja.

Még néhány haladó szokás

- **Tervezési mód:** sok agentnek van „csak tervezz, ne módosíts" üzemmódja. Nagy változtatás előtt használd.
- **Párhuzamos munkamenetek:** független feladatokon egyszerre több agent is dolgozhat – de mindig külön ágon (branch).
- **Másik AI mint ellenőr:** egy fontos rendszer kódját egy másik modellel is átnézetheted: „Nézd át biztonsági szempontból."

KULCSGONDOLATOK

1. A haladó eszközök lényege: amit egyszer jól leírtál, azt az AI mindig magától használja.
2. RULES.md a közös elvárásokra, CLAUDE.md / AGENTS.md a projekt memóriájára, skill a módszeredre.
3. Az MCP csatlakozókkal az agent a külvilágot is eléri: naptár, adatbázis, böngésző.

PRÓBÁLD KI

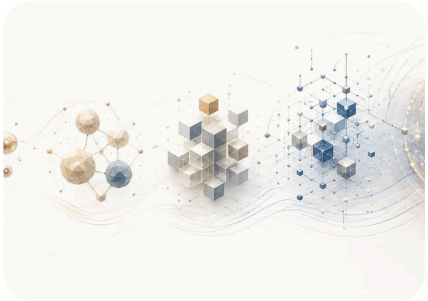
1. A projekted végén kérd a projektmemória-fájlt (CLAUDE.md vagy AGENTS.md).
2. Gondold végig: mit kellett a legtöbbször elmondanod az AI-nak? Írd be a RULES.md-be.

19

IV. RÉSZ · TOVÁBB · 19. LECKE

Az első saját projekted

Az egész egy oldalon – és a terv a következő hétre



EBBEN A LECKÉBEN

- Egy oldalon látod a teljes utat.
- Kapsz ötleteket és egy egyhetes tervet az első saját projektedhez.
- Megkapod a munkamenet-lezáró ellenőrzőlistát.

kb. 7 perc olvasás

Videó: 19. rész

3 kulcsgondolat

Végigértél. Most már érted, mi a vibe coding, és ismered a teljes folyamatot. Egy dolog hiányzik: **hogy csináld**. Ez a lecke ebben segít.

Az egész egy oldalon

1. Szándék
Az AI írja a kódot, te adod meg a működést és a célt.

2. Platform
Kódoló platform (coding agent), nem sima chat.

3. Specifikáció
A specifikáció dönti el, mit kapsz – emberi nyelven, átgondoltan.

4. Folyamat
Saját vázlat → AI-upgrade → UI-terv → mappa → indítás → átadás.

A DarwinAI training négy kulcsgondolata.

Ötletek az első projekthez

A legjobb első projekt **kicsi, a saját bosszúságodat oldja meg, és egy hét alatt kész**. Néhány ötlet a DarwinAI trainingjéből:

TERÜLET	ÖTLETEK
Üzleti alapok	Árajánlat-készítő · egyszerű cégértékelő · adatkinyerés PDF-ből · onboarding-folyamat
Irodai segédek	Meeting → teendőlista · számla-nyilvántartó · ügyfélválasz-segéd · heti riport-generátor · mini ügyfélkezelő
Tanulás	Kvíz-készítő · szókártya-gyakorló · vizsga-kikérdező · tanulási terv
Játék	Saját Flappy Bird · memóriakártya-játék · téglatörő a logóddal

TIPP

Írd minden vázlat végére: „A rendszer legyen egyszerű és gyorsan elkészíthető. Ha adatbázis kell hozzá, használj kis, helyi adatbázist (pl. SQLite).”

Egyhetes terv

NAP	FELADAT	LECKE
1. nap	Ötlet kiválasztása, saját vázlat (és hagyd érni)	6.
2. nap	AI-upgrade → SPEC.md; UI-tervek → design.png	6–7.
3. nap	Projektmappa, RULES.md, indítás, első öt perc figyelés	8–9.
4. nap	Átadás, első indítás, kipróbálás	10–11.
5. nap	Tesztelés, visszajelzések, javítások, mentési pont	12–13.
+1 nap	Ha kell: publikálás, PWA, és mutasd meg valakinek!	16–17.

A munkamenet lezárása – ellenőrzőlista

- [ ] Tudom, milyen eredmény készült, és hol találom.
- [ ] Megvan a projekt és az indítási útmutató (README.md).
- [ ] Értem, mi az egyszeri előkészítés, és mi a későbbi indítás.
- [ ] Meg tudom nyitni, és el tudom végezni benne a fő feladatot.
- [ ] Tudom, hogyan állítom le és indítom újra.
- [ ] Tudom, hová ment adatot, és hogyan őrzöm meg.
- [ ] Tudom, mely részeket tesztelték, és mi vár még ellenőrzésre.
- [ ] Értem, helyben, előnézetben vagy közzétett címen fut-e.
- [ ] Tudom, mit kell átadnom, ha más is szeretné használni.

Hol folytasd?

- **darwinai.hu/vibe-coding** – a pipálható folyamat, a szabálykönyv, az összeállító és a letöltések.
- **A DarwinAI YouTube-csatornája** – a sorozat videói és az új anyagok.
- **DarwinAI training** – ha csapatoddal, élőben, a saját projekteteken tanulnátok.

KULCSGONDOLATOK

1. Szándék – platform – specifikáció – folyamat: ez a vibe coding négy alapköve.
2. A legjobb első projekt kicsi, a saját bosszúságodat oldja meg, és egy hét alatt kész.
3. Egy munkamenetet akkor zártál le jól, ha az eredményt meg tudod nyitni, ki tudod próbálni, és később újra tudod használni.

PRÓBÁLD KI

1. Válaszd ki az első projektedet, és tűzz ki egy kezdőnapot a naptáradban.
2. Kövesd az egyhetes tervet – és ha kész, mutasd meg valakinek!

A. MELLÉKLET

Szószedet

A tankönyv fogalmai ábécérendben, egy-két mondatban.

Adatbázis	A program tartós memóriája: rendezett táblákban tárolt adatok, amelyek a program bezárása után is megmaradnak.
Agent (coding agent)	Kódolásra készült AI-ügynök, amely a projekt fájljaiban dolgozik: létrehoz, módosít, futtat, tesztel. Pl. Claude Code, OpenAI Codex.
API	Kapu, amelyen át egy program más rendszerekkel beszélget szabványos kérésekkel és válaszokkal.
API-kulcs	Titkos karaktersor, amely azonosítja a programodat egy API-nál. Jelszóként kezelendő.
Backend	A program szerveren futó, a felhasználó számára láthatatlan része: adatkezelés, titkok, számítások.
Branch (ág)	A projekt külön fejlesztési ága, ahol a fő változat érintése nélkül lehet kísérletezni.
Build	A kód „összecsomagolása” kiadható, gyorsan futó formába.
CLAUDE.md / AGENTS.md	Projektmemória-fájl, amelyet a coding agent minden munkamenet elején elolvas.
Commit	Egy elmentett, visszaállítható állapot a Gitben, rövid leírással.
Deploy (élesítés)	A program kitétele egy szerverre, hogy mások is elérjék.
Domain	Egy webcím (pl. cegem.hu), amelyet évente bérelsz, és a tárhelyre irányítasz.
Effort (erőfeszítés)	Beállítás, amely megszabja, mennyit „gondolkodjon” a modell, mielőtt válaszol vagy cselekszik.
.env	Fájl a titkos adatoknak (kulcsok, jelszavak). A gépeden marad, nem kerül fel a GitHubra.
Frontend	A program felhasználó által látott része, ami a böngészőben fut.
Git	Verziókövető rendszer: a projekt állapotainak mentése és visszaállítása.
GitHub	A Git-projektek felhőbeli otthona; innen indulhat az automatikus publikálás.
.gitignore	Lista azokról a fájlokról, amelyeket a Git nem menthet el és nem tölthet fel (pl. .env).
Hosting (tárhely)	Szerver, amely folyamatosan futtatja a programot.
HTML, CSS, JavaScript	A weboldalak három nyelve: tartalom és szerkezet, kinézet, működés.
https	Titkosított, biztonságos webes kapcsolat (a lakat a címsorban).
Iteráció	Egy kör a fejlesztésben: kérés → kipróbálás → visszajelzés → javítás.
Kontextus	Mindaz, amit a modell egy beszélgetésen belül „fejben tart”.
localhost	„Ez a gép.” A saját gépeden futó program címe; másnál nem működik.
Markdown (.md)	Egyszerű szövegformátum néhány jellel (#, -, **), amely megőrzi a szerkezetet.
MCP	Egységes csatlakozó az AI-ügynök és külső eszközök (naptár, adatbázis, böngésző) között.

Modell	Az AI „agya”; a nagyobb okosabb és drágább, a kisebb gyorsabb és olcsóbb.
npm	A Node.js csomagkezelője; ezzel telepíted (npm install) és indítod (npm run dev) a projekteket.
Node.js	Futtatókörnyezet, amely a JavaScriptet a böngészőn kívül is futtatja. Sok vibe coding projekt igényli.
Port	Szám a cím végén (pl. :5173), amely megmondja, a gép melyik programjához szól a kérés.
Prompt	Az AI-nak adott utasítás vagy kérés.
Push / pull	Feltöltés a GitHubra / letöltés a GitHubról.
PWA	Telepíthető weboldal: ikon, teljes képernyő, offline indulás, app store nélkül.
README.md	A projekt használati útmutatója: mi ez, hogyan indul, hol vannak az adatok.
Repository (repo)	Git által kezelt projekt, a teljes előzményeivel.
RULES.md	A minden projektre érvényes elvárásaid fájlja.
Skill	Újrahasznosítható szakértelem-csomag az AI-nak: leírás, szabályok, minták.
Specifikáció (SPEC.md)	A rendszer átgondolt, emberi nyelvű leírása: mit tudjon, kinek, hogyan, és mi nem cél.
SQLite	Kicsi, egyetlen fájlban tárolt adatbázis; helyi eszközökhöz ideális.
Supabase	Felhős adatbázis-szolgáltatás felhasználókezeléssel; népszerű vibe coding projektekhez.
Terminál	Szöveges parancsablak (Mac: Terminal, Windows: PowerShell).
Token	A szöveg apró egysége, amelyben a modellek „gondolkodnak” és számláznak.
UI	User interface – a felhasználói felület: képernyők, gombok, színek, elrendezés.
Vercel	Webalkalmazás-tárhely, amely a GitHubról automatikusan publikál.
Vibe coding	Szoftverkészítés AI-val, a saját szavaiddal: te adod a célt és a működést, az AI írja a kódot.

B. MELLÉKLET

Promptgyűjtemény

A tankönyv összes mintapromptja, sorrendben – másold, és igazítsd a saját projektedhez.

1. VÁZLATBÓL SPECIFIKÁCIÓ (6. LECKE)

AI-chat, erős modell

```
Az alábbi vázlatot egy szoftverről írtam, amit egy coding agenttel szeretnék elkészíttetni.

1. Strukturáld és rendszerezd a leírásomat.
2. Jelezd az ellentmondásokat és a hiányzó pontokat - ha kell, kérdezz vissza.
3. Tedd hozzá azokat a funkciókat, amelyek szerinted szükségesek a cél eléréséhez.
4. Készíts belőle Markdown formátumú specifikációt, amelyet közvetlenül át tudok adni egy vibe coding eszköznek.

NE kezdj még kódolni. Adatbázis-táblákat és részletes technikai architektúrát nem kell meghatároznod - azt később a coding
agent megtervezi.

A vázlatom:
[ide másold be a vázlatodat]
```

2. UI-TERVEK (7. LECKE)

ugyanabban a chatben

```
Tervezd meg a most létrehozott rendszer néhány fő képernyőjét, a core designját. Csinálj 4 kifejezetten eltérő
megközelítést, és mindegyikhez írd le röviden, mi a hangulata és kinek szól.
```

3. EGYSZERŰSÉG-ZÁRADÉK (A VÁZLAT VÉGÉRE)

opcionális

```
Fontos: a rendszer legyen egyszerű és gyorsan elkészíthető. Ha adatbázis kell hozzá, használj kis, helyi adatbázist (pl.
SQLite).
```

4. INDÍTÓ PROMPT (9. LECKE)

coding agent

```
A projektmappában megtalálod a SPEC.md fájlt (a rendszer leírása), a design.png fájlt (a látványterv) és a RULES.md fájlt
(az általános elvárások). Olvasd el mindhármat, és készítsd el a SPEC.md-ben leírt rendszert a design.png alapján, a
RULES.md követelményei szerint. Csak akkor tekintsd késznek, ha a specifikáció minden pontját végigellenőrizted.
```

5. TERV KÉRÉSE NAGY VÁLTOZTATÁS ELŐTT (9. LECKE)

coding agent

```
Mielőtt hozzányúlsz a kódhoz, írd le, mit és hogyan változtatnál, és milyen kockázatai vannak. Csak a jóváhagyásom után
kezdj bele.
```

6. INDÍTÁS ÉS BEMUTATÁS (10. LECKE)

coding agent

```
Indítsd el és mutasd meg.
```

7. ÁTADÁSI ÚTMUTATÓ (10. LECKE)

a munkamenet végén

```
A fejlesztés végén adj laikusoknak szóló átadási útmutatót a ténylegesen elkészült projekthez.

Írd le:
1. Mi készült el, és hol találhatók a fájljai?
2. Milyen programokat kell először telepítenem?
3. Mi az első indítás pontos menete?
4. Melyik mappában, milyen parancsokat futtassak, és mit kell látnom utánuk?
5. Hogyan nyitom meg az eredményt, és melyik ablakot kell nyitva hagynom?
6. Hogyan állítom le, és hogyan indítom újra másnap?
7. Hová menti az adatokat, és hogyan készítek róluk biztonsági másolatot?
8. Mit teszteltél ténylegesen, és mi nincs még bekötve?
9. Mihez kell fiók, API-kulcs vagy fizetős szolgáltatás?

Különítsd el az egyszeri előkészítést a későbbi indítástól.
Mentsd az útmutatót a projekt README.md fájljába is.
```

8. HA A TERMINÁLBAN HIBÁT KAPSZ (11. LECKE)

a hibaüzenettel

```
Mac / Windows rendszeren próbálom indítani. Ebben a mappában vagyok: […]. Ezt a parancsot futtattam: […]. Ezt az üzenetet kaptam: […]. Mondd meg a következő konkrét lépést, és azt is, milyen eredményt kell látnom, ha sikerült.
```

9. KÉNYELMES INDÍTÓ (11. LECKE)

coding agent

```
Az alkalmazás már működik. Tedd egyszerűvé a későbbi indítást: készíts az én rendszeremhez indítófájlt, és ellenőrizd a leállítás utáni újraindítást is.
```

10. TELJES TESZT (12. LECKE)

az első változat után

```
Most teszteld végig a teljes rendszert a specifikáció alapján. Ellenőrizd minden funkcióját, keresd meg a hibákat, javítsd ki őket, majd ellenőrizd újra. A végén írd le röviden: mit teszteltél, mit javítottál, és mi maradt nyitva.
```

11. HIBAJELENTÉS (12. LECKE)

képernyőképpel

```
Hibát találtam. Ezt csináltam: […]. Ezt vártam: […]. Ehelyett ez történt: […]. (Képernyőkép csatolva.) Keresd meg az okát, javítsd ki, és ellenőrizd, hogy a javítás mást nem rontott el.
```

12. MENTÉSI PONT (13. LECKE)

nagy módosítás előtt

```
A jelenlegi verzió működik. Készíts egy Git commitot erről az állapotról, és töltsd fel (push) a GitHubra. Csak utána kezdd el az új funkciót.
```

13. TITKOK ELŐKÉSZÍTÉSE (14. LECKE)

kulcs használata előtt

```
Hozd létre a szükséges .env fájlt és a .env.example mintafájlt, és gondoskodj róla, hogy a .env szerepeljen a .gitignore-ban. Mondd meg, melyik sorba írjam be a kulcsomat – a kulcsot én magam írom be.
```

14. ADATOK ÉS MENTÉS (15. LECKE)

coding agent

```
Hová menti a program az adatokat? Mi történik, ha leállítom és újraindítom? Írd le a README-be, hogyan készítek biztonsági mentést, és hogyan állítom vissza.
```

15. PWA (16. LECKE)

a kész alkalmazáshoz

```
Alakítsd az alkalmazást telepíthető PWA-vá: legyen ikonja és neve a kezdőképernyőn, teljes képernyőn nyíljon meg, és az alapfunkciói internet nélkül is induljanak el. Teszteld, és írd le a README-be, hogyan telepíthető iPhone-on és Androidon.
```

16. PUBLIKÁLÁS (17. LECKE)

ha mások is használnák

```
Szeretném publikálni az alkalmazást, hogy mások is elérjék egy linken. Javasold a projekthez illő legegyszerűbb megoldást (GitHub Pages vagy Vercel), és vezess végig lépésről lépésre. Az API-kulcsok a tárhely titkos beállításai közé kerüljenek, ne a kódba.
```

17. PROJEKTMEMÓRIA (18. LECKE)

a projekt későbbi pontján

```
Készíts a projekthez egy CLAUDE.md (vagy AGENTS.md) fájlt: röviden írd le, mire való a rendszer, hogyan indul, milyen fontos döntéseket hoztunk eddig, és mire figyeljen egy új munkamenet.
```

C. MELLÉKLET

Terminál-puska

A leggyakoribb parancsok Macen és Windowson. A sor eleji jelet ($, >) soha nem kell beírni.

MIT SZERETNÉL?	MAC (TERMINAL)	WINDOWS (POWERSHELL)
Terminál megnyitása	Command + Szóköz → Terminal	Start → PowerShell
Hol vagyok?	`pwd`	`pwd`
Mi van ebben a mappában?	`ls`	`dir`
Belépés egy mappába	`cd "mappa neve"`	`cd "mappa neve"`
Egy szinttel feljebb	`cd ..`	`cd ..`
A saját mappámba	`cd ~`	`cd ~`
Csomagok telepítése (egyszer)	`npm install`	`npm install`
Program indítása	`npm run dev`	`npm run dev`
Elérhető parancsok listája	`npm run`	`npm run`
Program leállítása	Control + C	Ctrl + C
Python-program futtatása	`python3 app.py`	`python app.py`
Node.js telepítve van?	`node -v`	`node -v`
Mappa megnyitása a fájlkezelőben	`open .`	`explorer .`
Előző parancs újra	↑ nyíl, majd Enter	↑ nyíl, majd Enter
A képernyő törlése	`clear`	`cls`

TIPP

Macen a mappát a Finderből a terminálablakba húzva beíródik a teljes útvonala – így nem kell begépelni. Windowson a Fájlkezelő címsorába kattintva másolhatod ki az útvonalat.

FIGYELEM

Óvatosan: `sudo` (rendszergazdai jog), `rm` / `del` (törlés), `rm -rf` (visszavonhatatlan, teljes törlés). Ilyen parancsot csak akkor futtass, ha pontosan érted, mit csinál – kérdezd meg az AI-t.

D. MELLÉKLET

Hibaelhárító

A leggyakoribb elakadások, és mit tegyél velük. Ha egyik sem segít: B. melléklet, 8. prompt.

TÜNET / ÜZENET	MI A GOND?	MIT TEGYÉL?
„Nem érhető el az oldal” / connection refused	A program nem fut, leállt, vagy más címen indult.	Indítsd el (`npm run dev`), és a kiírt címet nyisd meg.
`npm` **vagy** `node` **nem található**	A Node.js nincs telepítve.	Kérj telepítési segítséget az AI-tól; telepítés után nyiss új terminált.
`python` **nem található**	A Python nincs telepítve (Macen próbáld: `python3`).	Kérj telepítési útmutatót az AI-tól.
`package.json` **nem található (ENOENT)**	Rossz mappában vagy.	Lépj be a projekt mappájába (`cd`), és nézd meg: `ls` / `dir`.
`Missing script: dev`	Nincs ilyen nevű indítóparancs.	Futtasd: `npm run` – és add át az AI-nak a listát.
`port already in use` **(EADDRINUSE)**	A portot már használja valami – gyakran a saját, korábbi példányod.	Zárd be a régi terminált (Ctrl + C), vagy kérj segítséget.
A felület látszik, de a mentés vagy az AI-funkció hibázik	A háttérrész vagy egy külső kapcsolat nem működik.	Add át az AI-nak a hibaüzenetet (képernyőképpel).
API-kulcsot vagy .env beállítást kér	Egy külső szolgáltatás hozzáférése hiányzik.	Kérj útmutatót: hol kell beállítani, mihez kell, mennyibe kerülhet (14. lecke).
A link a kollégádnál nem működik	Localhost linket küldtél – nála az ő gépét jelenti.	Publikálj, vagy add át a projektet README-vel (17. lecke).
Az AI ugyanazt a hibát harmadszor „javítja”	Elakadt a beszélgetés.	Új munkamenet, erősebb modell, vagy vissza egy mentési pontra (12. lecke).
Telefonon szétcsúszik a felület	A felület nincs mobilra optimalizálva.	Képernyőképpel kérd: „Tedd mobilon is jól használhatóvá.” Tedd a RULES.md-be is.
Elfogyott a keret / „usage limit”	Az előfizetésed időszakos kerete elfogyott.	Várj a keret frissüléséig, vagy válts kisebb modellre az apró feladatokhoz.

JÓ TUDNI

A hibaüzenetet mindig szövegként (vagy képernyőképként) add vissza az AI-nak. Jelszót, API-kulcsot soha ne másolj bele.

E. MELLÉKLET

Technikai háttér

Részletesebb magyarázatok azoknak, akiket érdekel, mi történik a háttérben. A tanuláshoz nem szükséges.

E1. Fájltípusok, amelyekkel találkozni fogsz

KITERJESZTÉS	MI EZ?
`.html`	Weboldal tartalma és szerkezete. Önmagában is megnyitható a böngészőben.
`.css`	A weboldal kinézete: színek, betűk, elrendezés.
`.js` / `.ts`	JavaScript / TypeScript: a működés, a logika. A TypeScript a JavaScript szigorúbb, típusos változata.
`.jsx` / `.tsx`	React-komponensek – a modern webes felületek építőkövei.
`.py`	Python-program – népszerű adatfeldolgozáshoz, automatizáláshoz, AI-hoz.
`.json`	Strukturált adatformátum; pl. a `package.json` írja le a projekt csomagjait és parancsait.
`.md`	Markdown-dokumentum: SPEC.md, README.md, RULES.md.
`.env`	Titkos beállítások (kulcsok). Nem kerül fel a GitHubra.
`.sql` / `.db` / `.sqlite`	Adatbázis-utasítások, illetve adatbázisfájl.

E2. Node.js, npm és a node_modules

A webes vibe coding projektek többsége **Node.js**-re épül: ez a futtatókörnyezet teszi lehetővé, hogy a JavaScript a böngészőn kívül, a gépeden is fusson. Az **npm** a hozzá tartozó csomagkezelő. A projekt `package.json` fájlja felsorolja, milyen kész „csomagokra" (mások által írt kódrészekre) van szükség; az `npm install` ezeket letölti a **node_modules** mappába. Ez a mappa nagy (akár több száz MB), de bármikor újra letölthető – ezért nem kerül a GitHubra, és átadáskor sem kell elküldeni.

A `package.json` „scripts" része sorolja fel az indítóparancsokat: `dev` (fejlesztői indítás, automatikus frissítéssel), `build` (kiadható változat készítése), `start` vagy `preview` (a kész változat futtatása). Az `npm run <név>` ezeket futtatja.

E3. Frontend, backend, és hol fut mi

A **frontend** a böngésződben fut: amit ott van, azt a felhasználó elvileg ki tudja olvasni – ezért nem lehet benne titkos kulcs. A **backend** a szerveren fut: itt vannak a titkok, az adatbázis-kapcsolat, a védett számítások. Egy „statikus" oldalnak csak frontendje van; egy webalkalmazásnak mindkettő. Modern keretrendszerek (pl. Next.js) mindkettőt egy projektben kezelik.

E4. Localhost, IP-cím, port

Minden gépnek van egy „önmagára mutató" címe: `localhost`, számmal `127.0.0.1`. A **port** (pl. `:3000`, `:5173`, `:8000`) azt mondja meg, a gép melyik programjához szól a kérés – egy gépen egyszerre több program is futhat, mindegyik a saját portján. Ha egy port foglalt, a fejlesztői szerver gyakran automatikusan a következőt választja – ezért kell mindig a ténylegesen kiírt címet megnyitni.

E5. Git-parancsok, ha kíváncsi vagy

PARANCS	MIT CSINÁL?
`git init`	Git-tárolóvá alakítja a mappát.
`git status`	Megmutatja, mi változott az utolsó mentés óta.
`git add .`	Kijelöli az összes változást a mentéshez.
`git commit -m "üzenet"`	Elmenti az állapotot, leírással.
`git push`	Feltölti a mentéseket a GitHubra.
`git pull`	Letölti a GitHubról az új változásokat.
`git log --oneline`	Kilistázza a korábbi mentéseket.
`git switch -c uj-ag`	Új ágat (branch) nyit, és átvált rá.
`git restore .`	Elveti a még el nem mentett változásokat (óvatosan!).

E6. API-k működése

A legtöbb webes API **HTTP-kéréseket** fogad: `GET` (lekérdezés), `POST` (új adat küldése), `PUT` / `PATCH` (módosítás), `DELETE` (törlés). Az adatok jellemzően **JSON** formátumban utaznak. A válasz egy állapotkóddal érkezik: `200` = rendben, `401` / `403` = nincs jogosultság (gyakran hibás kulcs), `404` = nincs ilyen, `429` = túl sok kérés (keretkorlát), `500` = hiba a szerveren. Ha az AI ilyen kódot említ, ez segít megérteni, hol a gond.

E7. Környezeti változók és titkok élesben

A `.env` fájl sorai `NEV=érték` formájúak (pl. `OPENAI_API_KEY=sk-…`). A program „környezeti változóként" olvassa be őket. Publikált alkalmazásnál nincs .env fájl a szerveren: a tárhely (Vercel, Coolify stb.) beállításaiban adod meg ugyanezeket a neveket és értékeket („Environment Variables"). A `.env.example` ugyanezeket a neveket tartalmazza érték nélkül – ebből tudja más, milyen kulcsokra van szükség.

E8. Adatbázisok röviden

A legtöbb adatbázis **relációs**: táblákból áll (pl. ügyfelek, számlák), amelyek sorokban tárolják az adatokat, és egymásra hivatkozhatnak. A nyelvük az **SQL**. Az **SQLite** egyetlen fájl, szerver nélkül – kis, helyi eszközökhöz ideális. A **PostgreSQL** komoly, többfelhasználós adatbázis-szerver; a Supabase és a Neon is erre épül. Az adatbázis szerkezetének változásait **migrációk** írják le – ezeket az agent kezeli.

E9. Hosting, DNS és https

A domain és a tárhely közötti kapcsolatot a **DNS** teremti meg: a domain-szolgáltatónál egy rekorddal (jellemzően `A` vagy `CNAME`) megadod, hová mutasson a cím. A tárhelyszolgáltató megmondja, pontosan mit kell beírni. A **https**-hez tanúsítvány kell, amelyet a modern tárhelyek (Vercel, Coolify, GitHub Pages) automatikusan és ingyen kérnek (Let's Encrypt). A DNS-változás terjedése néhány perctől néhány óráig tarthat.

E10. A PWA technikai elemei

Egy PWA-hoz három dolog kell: egy **webmanifest** fájl (név, ikonok, színek, indítási mód), egy **service worker** (háttérben futó szkript, amely gyorsítótárazza a fájlokat, így offline is indul), és **https**. Az agent mindhármat létrehozza a 16. lecke promptjára.

E11. Tokenek és kontextusablak

A modellek a szöveget **tokenekre** – nagyjából szótöredékekre – bontják; a magyar szöveg jellemzően több tokenből áll, mint az angol. A modell egyszerre véges mennyiségű tokent tud „fejben tartani” – ez a **kontextusablak**. Hosszú munkamenetben a régi részletek kiszorulhatnak vagy összefoglalódhatnak; ezért jó a független feladatokat új munkamenetben kezdeni, és a fontos tudnivalókat a projektmemória-fájlba (CLAUDE.md / AGENTS.md) írni.



A DarwinAI magyar AI-tanácsadó, -oktató és -fejlesztő műhely. Abban segítünk, hogy az AI ne kísérlet maradjon, hanem mindennapi munkaeszköz legyen – egyéneknek és szervezeteknek.

Saját (és a szervezeted) projektjeihez és oktatási célra szabadon felhasználhatod. Továbbadáskor, közzétételkor vagy átdolgozott változatnál a forrás megjelölése kötelező: „DarwinAI – darwinai.hu”. A modell- és árinformációk 2026. októberi állapotot tükröznek; az AI-eszközök gyorsan változnak.

© 2026 DarwinAI · Budaházy Szabolcs – darwinai.hu