reactivdevs

ZDARMA · UKÁZKA Z KURZU

Od vibe-codingu k produkčnímu softwaru s AI

Základní workflow pro vývojáře, kteří chtějí s AI stavět rychleji a bez špaget.

@reactivdevs · reactivdevs.com

NEŽ ZAČNEME

Pro koho to je a co si odneseš

Tenhle návod je pro vývojáře, kteří už něco naprogramují a chtějí přestat s AI ztrácet čas. Není to o tom naučit se nový jazyk ani nástroj. Je to o způsobu práce, díky kterému s AI postavíš reálný, produkční software rychleji a bez toho, aby ses k němu za měsíc bál vrátit.

Příklady píšu v TypeScriptu a Node, ale slouží jen jako ukázka. Jazyk ani technologie nejsou důležité, principy fungují všude stejně. Přečteš to za 20 minut. Ber to jako základ, na kterém stavíme celý kurz.

TERMINOLOGIE

Co je LLM a proč ti to mění vývoj

LLM (velký jazykový model, jako Claude nebo GPT) nevyhledává odpovědi v žádné databázi. Předpovídá nejpravděpodobnější pokračování textu na základě vzorců z obrovského množství dat. Všechno, vstup i výstup, se přitom počítá v **tokenech** (token je kousek slova). Právě proto občas sebejistě plácne nesmysl: nekontroluje pravdu, jen dopočítává, co je pravděpodobné.

> **Klíčový fakt:** model si mezi požadavky nic nepamatuje. Vidí jen to, co máš právě teď v **kontextovém okně**, jeho pracovní paměti.

Do toho okna se vejde systémová instrukce, dosavadní konverzace, tvůj dotaz i kód, všechno dohromady, a je omezené. Co v okně není, pro model neexistuje. Proto se kontext ukládá do souborů (třeba CLAUDE.md), jinak ho příště nemá.

Co si z toho odnést:

- Kvalitu výstupu určuje hlavně to, co dáš do okna. Není to kouzlo, je to kontext.
- Model není vševědoucí ani neomylný. Ověřuj, nespoléhej slepě.
- Krátké, cílené zadání s kontextem porazí dlouhý vágní prompt.

A záleží, kde s AI pracuješ. Chat je dobrý na dotazy a rozvahu. Pro reálný vývoj je ale klíčový nástroj přímo v editoru (Claude Code, Cursor), který vidí celý projekt, ne jen to, co zrovna nalepíš. K tomu se dostaneme dál.

REFERENCE

Slovníček pojmů

Pár slov, která se v tomhle světě motají pořád dokola. Ať v tom máš jasno.

- **LLM:** velký jazykový model. Předpovídá text po tokenech, nevyhledává v databázi.
- **Token:** nejmenší jednotka textu (kousek slova). Vstup i výstup se počítá v tokenech.
- **Kontextové okno:** pracovní paměť modelu. Co v něm není, model nevidí.
- **Prompt:** zadání pro AI. Čím lepší kontext, tím lepší výstup.
- **CLAUDE.md:** soubor s trvalým kontextem projektu, který AI čte při každém spuštění.
- **Plan mode:** režim, kdy AI nejdřív navrhne plán a kód píše až po schválení.
- **Agent:** AI, která odbaví celý úkol samostatně, ne jen vrátí odpověď.
- **Orchestrace:** řízení více agentů, kteří pracují paralelně.
- **MCP:** standardní protokol pro napojení AI na tvoje nástroje (Slack, git, databáze).
- **Halucinace:** sebejistě znějící, ale nesprávný výstup. Proto vždy ověřuj.

MINDSET

Proč jsi s AI pomalejší, než bys mohl být

Většina vývojářů dnes AI používá, ale používá ji špatně. Ne proto, že by byli líní nebo hloupí, ale proto, že jim to nikdo neukázal jinak. Typický postup vypadá takhle: otevřu ChatGPT, popíšu problém, zkopíruju odpověď do editoru, ono to nefunguje, zkopíruju chybu zpátky. A tak pořád dokola.

Vypadá to jako práce s AI. Ve skutečnosti je to **nejpomalejší možný způsob**, jak ji použít, a k tomu nejnebezpečnější pro kvalitu tvého kódu.

> **Ber AI jako šikovného juniora.** Když mu řekneš jen „udělej to", dostaneš náhodu. Když mu dáš kontext, cíl a pravidla, dostaneš seniorní výsledek.

Čtyři návyky, které tě brzdí nejvíc:

- ✕ Copy-paste smyčka mezi chatem a editorem
- ✕ Prompt typu „udělej mi appku" bez jakéhokoli kontextu
- ✕ Slepé přijetí prvního výstupu bez review
- ✕ Jeden obří soubor, protože „to zatím funguje"

V dalších kapitolách nahradíme každý z těchhle návyků něčím, co tě reálně zrychlí.

REALITA

5 mýtů o AI ve vývoji

Kolem AI se toho napovídá spousta. Tady je pět nejčastějších mýtů a jak to je doopravdy.

- **„AI mě nahradí."** Nahradí spíš toho, kdo ji neumí používat. Kdo ji umí vést, je násobně rychlejší.
- **„AI je neomylná."** Předpovídá pravděpodobné, ne pravdivé. Bez kontroly ti do kódu propašuje chyby.
- **„Stačí lepší prompt."** Nejde o kouzelnou formuli, ale o kontext, plán a malé kroky.
- **„S AI je appka za odpoledne."** Naklikat prototyp ano. Udržitelný produkt je o řemesle, ne o rychlosti.
- **„Kód od AI je vždycky špatný."** Špatný je kód bez vedení. S dobrým workflow je čistý a otestovaný.

ZÁKLAD

Tři pilíře rychlého a bezpečného AI vývoje

Celý zbytek tohohle návodu stojí na třech pilířích. Když si zapamatuješ jen tohle, už budeš před většinou lidí.

1. Kontext. AI není vědma. Kvalita výstupu se odvíjí od toho, kolik relevantních informací dostane: co stavíš, proč, jaká platí pravidla a jak vypadá zbytek projektu. Většina „hloupých" odpovědí není chyba modelu, ale chybějícího kontextu.

2. Iterace po malých krocích. Čím větší kus práce zadáš najednou, tím hůř se kontroluje a opravuje. Malé, ověřitelné kroky znamenají míň chyb a snadné vrácení, když se něco pokazí.

3. Kontrola. AI zrychluje psaní, ne přemýšlení. Zodpovědnost za kód je pořád tvoje. Každý krok si přečti, ověř a teprve pak pusť dál. Bez toho vzniká špageta, ke které se za měsíc nedá vrátit.

> Kontext říká AI, co má dělat. Malé kroky drží práci pod kontrolou. Kontrola zajistí, že výsledek vydrží. Vynech jeden pilíř a rozbije se celek.

PILÍŘ 1 · KONTEXT

Kontext místo úkolu: jak zadávat

Nejrychlejší způsob, jak zlepšit výstupy AI, není lepší model ani chytřejší trik. Je to dát jí pořádný kontext dřív, než ji o cokoli požádáš.

Porovnej si to. Špatné zadání: „Udělej mi přihlašování." AI netuší, jaký máš stack, jak řešíš stav, jestli chceš tokeny nebo session, ani jak vypadá zbytek kódu. Vygeneruje generický kód, který do projektu nezapadá, a ty pak dvě hodiny opravuješ.

Dobré zadání dá AI čtyři věci v tomhle pořadí:

- **Kontext:** co je projekt, jaký stack, jak vypadá okolí (relevantní soubory)
- **Cíl:** čeho chceš dosáhnout a proč
- **Pravidla:** konvence, omezení, čemu se vyhnout
- **Úkol:** teprve teď konkrétní zadání

> **Šablona promptu**
> Kontext: Pracuju na [typ projektu] v [stack]. Přikládám [soubory/části].
> Cíl: Potřebuju [co] tak, aby [proč / jak se to má chovat].
> Pravidla: Drž se [konvence]. Nepoužívej [čemu se vyhnout]. Uprav jen [rozsah].
> Úkol: [konkrétní krok]. Než začneš psát kód, navrhni krátce postup.

Vágní vs. strukturované, na reálném příkladu:

```
# Vágní zadání
Zrefaktoruj tuhle funkci.

# Strukturované zadání
Kontext: processPayment() v payments/service, complexity 18, 0 % testů.
Cíl: rozděl na menší funkce (max complexity 5), zachovej veřejné
rozhraní.
Pravidla: neměň signatury, každá nová funkce ať má test.
Úkol: nejdřív navrhni strukturu, počkej na schválení, pak teprve piš
kód.
```

NÁSTROJE

Z chatu do editoru

Chatovací okno je fajn na rychlý dotaz nebo vysvětlení konceptu. Na reálnou práci v kódu je to ale ten nejpomalejší nástroj, protože AI nevidí tvůj projekt. Vidí jen to, co jí zrovna nalepíš, takže pořád dokola vysvětluješ, co která část dělá.

Řešením je přesunout se do editoru s AI (Cursor, Claude Code). Rozdíl je zásadní: AI má přístup k celému projektu, čte skutečné soubory, drží kontext a umí zasáhnout na víc místech najednou.

Jednoduché pravidlo:

- **Chat** = ptám se, učím se, brainstormuju
- **Editor s AI** = stavím, refaktoruju, opravuju

Jakmile tenhle přechod uděláš, zmizí velká část copy-paste smyčky z první kapitoly úplně sama.

A nastav si projekt jednou. Soubor **CLAUDE.md** v kořeni repa je trvalý kontext, který AI čte při každém spuštění. Nemusíš pořád dokola vysvětlovat stack a pravidla.

```
# CLAUDE.md
## Stack
Node + TypeScript, PostgreSQL (Prisma).
## Konvence
Testy u důležité logiky. Malé commity. Kód anglicky.
## Zakázáno
Necommituj bez zelených testů. Neupravuj generované soubory.
```

```
# spuštění v projektu
cd muj-projekt
claude
> Přidej endpoint POST /users s validací a testy.
  ✓ 23 testů prošlo
```

PILÍŘ 2 · PLÁNOVÁNÍ

Plánuj, než začneš psát (plan mode)

Nejdražší chyby nevznikají při psaní kódu, ale při špatně pochopeném zadání. Proto se vyplatí nechat AI napřed naplánovat a teprve pak psát. Tomuhle přístupu se říká spec-driven: než vznikne první řádek kódu, existuje krátký plán, na kterém se shodnete.

Moderní nástroje to mají zabudované jako **plan mode**. Funguje takhle:

1. Zadáš, co chceš, a přepneš AI do režimu plánování.
2. AI místo psaní kódu vrátí návrh postupu: jaké soubory se dotkne, co změní, v jakém pořadí.
3. Ty plán přečteš a opravíš dřív, než se napíše jediný řádek.
4. Teprve po schválení AI plán provede.

Proč je to tak silné: opravit špatný plán je otázka jedné věty. Opravit špatně napsaný kód napříč deseti soubory je otázka hodin. Plan mode přesouvá kontrolu na nejlevnější možné místo.

> Pravidlo: u čehokoli většího než triviální změna si nejdřív vyžádej plán. Když v něm něco nesedí, je to signál, že bys stejnou chybu dostal i v kódu.

```
# zapnutí plan mode (Claude Code)
Shift + Tab   → přepíná Plan / Act
nebo v promptu: "plan only, neupravuj žádné soubory"

# místo změn dostaneš plán
1. middleware/ratelimit — token bucket
2. zapojit na /api routy
3. testy pro limity a hlavičky
```

PILÍŘ 2 · MALÉ KROKY

Anti-špageta: aby to za tři měsíce nešlo do koše

Naklikat něco, co „zatím funguje", umí dneska každý. Umění je postavit to tak, aby se do toho dalo za tři měsíce sáhnout. Rozdíl dělají tři návyky.

Malé diffy. Zadávej práci v malých, ověřitelných krocích. Jedna změna, jedna kontrola. Když je něco špatně, vrátíš pár řádků, ne celý den práce.

Architektura v malém. Nenech vzniknout jeden soubor o tisících řádcích. Ať AI drží věci oddělené: logika zvlášť, UI zvlášť, data zvlášť. Řekni jí to jako pravidlo, jinak nasype všechno na jednu hromadu.

Commituj často. Každý ověřený krok ulož do gitu. Máš tak vždy bod, kam se dá vrátit, a AI dostane čistý základ pro další krok.

PILÍŘ 3 · KONTROLA

Kvalita s AI

AI umí kód nejen psát, ale i hlídat. Většina lidí využívá jen tu první polovinu.

- **Typy a lint jako záchranná síť.** Nech si nastavit přísný typový systém, linter a formátování. AI se pak sama drží tvých pravidel a chyby vidíš hned, ne až v produkci.
- **Testy píše taky AI.** U důležité logiky si nech napsat testy a nech je proběhnout. Nejde o dogma, jde o to, aby se ti věci nerozbíjely potichu.
- **AI jako code reviewer.** Než něco pustíš dál, nech AI vlastní kód projít a hledat chyby, hrany a bezpečnostní díry. Druhý pár očí zadarmo.

> Quality gates (typy, lint, testy) nejsou brzda. Jsou to mantinely, díky kterým můžeš jet s AI rychle a přitom bezpečně.

```
# .claude/settings.json — kontrola po každé akci AI
"hooks": { "Stop": [{ "hooks": [
  { "type": "command", "command": "npm run verify" }
]}]}
```

Po každé akci se spustí tvoje kontrola. AI vidí výstup a chybu opraví ještě dřív, než se vrátí k tobě.

BEZPEČNOST

Nedej AI klíče od všeho

AI je mocný nástroj, ale neomezený přístup jí nedávej. Nastav přesně, co smí a co ne. Pár minut práce, co ti ušetří pořádný průšvih.

- **Allow list:** přesně dané příkazy, co AI smí spustit (git status, testy, build).
- **Deny list:** co nikdy nesmí číst. Klíče, certifikáty, tokeny, SSH.
- **Žádné produkční secrets.** AI nikdy nedávej reálné přístupy ani produkční data.

```
# .claude/settings.json
"permissions": {
  "allow": ["Bash(git status)", "Bash(npm run test)"],
  "deny":  ["Read(./.env)", "Read(**/*.pem)", "Read(~/.ssh/**)"]
}
```

> Pravidlo: AI má mít přesně tolik přístupu, kolik potřebuje k úkolu. Nic víc.

UKÁZKA

Stejná feature dvakrát

Vezměme jednoduchý úkol: přidat filtrování v seznamu.

ŠPATNĚ

Do chatu napíšeš „přidej filtrování". Dostaneš generický kód, který nepočítá s tvým stavem ani daty. Nalepíš ho, ono to napůl funguje, opravuješ, doptáváš se, po hodině to jakž takž jede a nikdo neví, jak přesně.

SPRÁVNĚ

V editoru dáš AI kontext, cíl a pravidla. Vyžádáš si plán, schválíš ho, necháš provést v malých krocích a každý zkontroluješ. Za pár minut máš čistou funkci, které rozumíš a která zapadá do projektu.

Stejný nástroj, stejný model. Rozdíl je jen ve způsobu práce, přesně v těch třech pilířích.

BONUS

Prompt patterns a checklist

Pět prompt patterns, co použiješ hned:

1. **Kontext napřed:** „Tady je projekt a relevantní soubory. Než něco navrhneš, řekni, co jsi z toho pochopil."
2. **Nejdřív plán:** „Nepiš kód. Navrhni postup a soubory, kterých se to dotkne."
3. **Malý krok:** „Uprav jen tohle jedno místo, nic dalšího neměň."
4. **Vysvětli mi to:** „Vysvětli, proč jsi to udělal takhle a jaké to má nevýhody."
5. **Zkontroluj se:** „Projdi vlastní kód a najdi chyby, hrany a bezpečnostní rizika."

Checklist produkčního setupu:

- [ ] Přísné typy a linter
- [ ] Automatické formátování
- [ ] Git a časté commity
- [ ] Testy u důležité logiky
- [ ] Pravidla pro AI (konvence projektu)
- [ ] Oddělená struktura (logika / UI / data)

KAM TO JDE DÁL

Agenti, MCP a automatizace

Až zvládneš základní workflow, čeká tě další patro. Tady už AI nejen píše kód, ale řídí celý tvůj pracovní tok. Krátký přehled, ať vidíš, co ti zatím schází za know-how.

- **MCP servery.** Napojí AI na tvoje nástroje (Slack, mail, Jira, databáze, git) přes jednotné rozhraní. AI pak nepracuje jen s kódem, ale s celým tvým světem.
- **AI agenti.** Samostatně odbaví úkol od začátku do konce, ne jen vrátí jednu odpověď.
- **Orchestrace.** Víc agentů běží paralelně, orchestrator je řídí a spojuje jejich výsledky.
- **Scheduled flows.** Naplánované běhy, co se spustí samy, třeba každé ráno nebo přes noc.

```
# .claude/ — vše v gitu, sdílené s týmem
.claude/
  skills/        # znovupoužitelné instrukce
  agents/        # specializovaní subagenti
  settings.json  # permissions, hooks, MCP
CLAUDE.md        # kontext projektu
```

Každá z těch věcí sama o sobě šetří čas. Teprve když je poskládáš dohromady, stane se něco většího. Jak to vypadá v praxi, ukazuje bonus na další straně.

TVŮJ DEN

Vývojář budoucnosti nepíše kód. Zadává a kontroluje.

Tohle není sci-fi, existuje to už dneska. Se správně poskládaným AI workflow se z vývojáře stává hlavně ten, kdo práci řídí a ověřuje. Podívej se na běžné ráno.

- **Zapneš počítač a jdeš si pro kafe.**
- **Naplánovaný flow už pracuje.** Mezitím prošel Slack a maily, vytáhl důležité a připravil ti plán dne s hotovými TODO.
- **Agenti odbavují práci.** Vrátíš se s kafem a úkoly z plánu už běží, orchestrator je řídí a rozděluje.
- **Ty jen verifikuješ.** Procházíš hotové změny a co je v pořádku, pustíš dál.
- **Slack za tebe.** Agent připravuje odpovědi a posílá je k tvému schválení, než je odešle.

Nenapsal jsi skoro jediný řádek kódu. A přesto se udělalo víc práce než za celý den ručního programování. Stal ses zadavatelem a kontrolorem. To je ten skutečný skok, o kterém většina lidí zatím jen slyšela.

Otázka není, jestli tahle změna přijde. Ale jestli u ní budeš ten, kdo řídí.

KURZY

Dva kurzy: řemeslo a rychlost

Tohle byl základ. ReactivDevs je praktické vzdělávání pro vývojáře, kteří chtějí psát production-ready software a efektivně používat AI. Stojí na dvou kurzech a oba mají stejný formát: **každá lekce je nahraný 1:1 call**, sdílená obrazovka, reálný kód, žádné slidy.

Production-ready Developer

Řemeslo, které po tobě firma chce. Vzniklo ze stovky odučených 1:1 lekcí. 13 modulů, 46 lekcí.

- **Setup a git doopravdy:** TypeScript, lint, husky, CI, rebase i konflikty naživo.
- **React, formuláře a API vrstva:** react-hook-form a Zod, Swagger jako kontrakt, React Query.
- **Architektura a testování:** clean code, design patterns, unit i E2E v Playwrightu.
- **CI/CD, soft skills a kariéra:** pipeline, secrets, jak si říct o peníze, pohovor.

AI Developer

To samé řemeslo, ale násobně rychleji s AI. 7 modulů, 35 lekcí, součástí je AI Dev Kit.

- **Spec a plán s AI:** kontext místo úkolu, plan mode, prompt patterns.
- **Reálný projekt end-to-end:** Prisma, backend, frontend, nasazení. Plus stejná feature ručně a s AI vedle sebe.
- **Kvalita s AI:** review, testy, bezpečnost a tři úrovně přístupu.
- **AI Dev Kit:** hotové repo se skills, agenty a nastavením, plus jak si napsat vlastní.

> **Bez řemesla ti AI jen rychleji vyrobí nepořádek.** Proto ty kurzy existují dva a proto dávají největší smysl v tomhle pořadí.

ZA 5 900 KČ

Co dostaneš v ceně

Každý kurz stojí **5 900 Kč** místo plných 8 900 Kč. Jednorázová platba, žádné předplatné, 14denní garance vrácení peněz.

- **Video lekce** ve formátu nahraného 1:1 callu, jedeš vlastním tempem
- **Startovací repo** s produkčním setupem a quality gates
- **Referenční projekt** stavěný poctivě od prázdného repa po nasazení s CI
- **AI Dev Kit** se skills a agenty (u kurzu AI Developer)
- **Checklisty a šablony:** code review, pre-launch, příprava na pohovor
- **Doživotní přístup** a aktualizace zdarma

> **Koupíš jednou, máš napořád.** Zlomek ceny bootcampu nebo mentoringu, a přitom vidíš, jak se produkční software staví doopravdy.

DALŠÍ KROK

Kam se posunout

Tohle byl základ. Naučil ses, jak s AI pracovat rychleji a bezpečně: kontext, plánování, malé kroky, kontrola. To samo o sobě tě posune před většinu lidí.

Production-ready Developer je obsahově hotový a natáčí se, takže se u něj dá **zapsat na čekací listinu**. Termín spuštění se dozvíš jako první a zaváděcí cena ti zůstane držená. Na AI Developer se pracuje. Když chceš vedení přímo na svém projektu, je tu mentoring, a na konkrétní zásek rychlá konzultace.

Production-ready Řemeslo od setupu po novou práci. 13 modulů, 46 lekcí. 5 900 Kč, čekací listina otevřená.

AI Developer Stavění s AI, agenti a AI Dev Kit. 7 modulů, 35 lekcí. 5 900 Kč, připravuje se.

Mentoring 1:1 vedení na tvém projektu, code review a zpětná vazba.

Konzultace Rychlá 1:1 pomoc s konkrétním problémem.

> **Odkaz najdeš v biu a na reactivdevs.com/course.** Pokud ti tenhle návod pomohl, ulož si ho a pošli ho někomu, komu by se hodil.