reactivdevs

UKÁZKA Z KURZU

Production Ready Frontend Developer

Řemeslo, které po tobě firma chce. Kompletní osnova a ukázky ze čtyř lekcí.

@reactivdevs · reactivdevs.com

NEŽ ZAČNEME

Odkud se tenhle kurz vzal

Odvedl jsem přes sto 1:1 mentoringových lekcí s více než dvaceti vývojáři, od úplných začátečníků po seniory. Pokaždé se ukázalo to samé: kód nějak napsat umí, ale nikdo jim neukázal, jak vypadá projekt vedený pořádně.

Chyběl git, chyběly testy, chyběla pipeline, chyběla struktura. A hlavně chyběla jistota to obhájit na pohovoru nebo v code review. Tutoriály tohle neřeší, protože se v nich **nikdy nic nepokazí**.

Kurz Production Ready Frontend Developer je ten mentoring nahraný. Stejná témata, stejné pořadí, stejný formát. Příklady jsou v Reactu, principy platí i na Angularu a Vue. Tenhle dokument je ukázka: dostaneš celou osnovu a ukázky ze čtyř lekcí, ať víš, co kupuješ.

Formát, ať víš, do čeho jdeš:

- Kde na tom záleží, je to video: **sdílím obrazovku** a mluvím na jednoho konkrétního člověka, ne do publika.
- Reálný kód přímo v repu, včetně chvil, kdy něco nejede.
- Vysvětluje se **proč**, ne jen jak. To je to, co se z tutoriálu nedozvíš.
- Zbytek jsou hutné materiály ke stažení, ke kterým se vracíš, když si potřebuješ něco ověřit.
- Na konci kapitoly homework, na začátku další se koukáme na výsledek.

11 lekcí | Hromada materiálů | Certifikát po dokončení | Celoživotní přístup

CO TĚ ČEKÁ

Deset lekcí práce

Průběžný projekt je aplikace na vyúčtování cestovních výdajů. Dostaneš ji hotovou a funkční, od setupu po nasazení s CI, a lekce tě jí provedou: tady je to, takhle to funguje, a tohle by se stalo, kdyby to tam nebylo. Backend nahrazuje verzovaný OpenAPI kontrakt a MSW, přesně jako když backend ještě není hotový. Cvičení nejsou postav to, ale rozšiř to nebo najdi, co jsem rozbil.

01 **Repo jako od firmy.** Průchod hotovým repem soubor po souboru: tsconfig se strict módem řádek po řádku, ESLint a Prettier, které se nehádají, skripty jako rozhraní projektu, husky a commitlint. U každého uvidíš, co by se stalo, kdyby tam nebyl.

02 **Nástroje a prostředí.** Co si doladit sám: format on save, absolutní importy, debugger na Vite, React Query devtools, klik v prohlížeči do komponenty. Plus VS Code pluginy rozdělené na nutné, užitečné a ty, které si nezapínej.

03 **Git doopravdy.** Co se reálně děje při commitu, rebase versus merge, konflikty naživo a pull request, který kolega přečte za deset minut.

04 **React řemeslně.** Kde vede hranice komponenty, custom hooky, které jdou testovat, a generická komponenta přes useController.

05 **Formuláře.** react-hook-form a Zod jako jediný zdroj pravdy pro validaci i typy, podmíněná validace přes superRefine a dynamická pole.

06 **API vrstva.** Swagger jako kontrakt, generovaný klient, React Query, query keys a invalidace, MSW místo čekání na backend. Nula ručně psaných fetchů.

07 **Autentizace.** Login a logout flow na síti, httpOnly cookie versus localStorage a útoky, které to řeší, chráněné routy a refresh token.

08 **Architektura.** Kam co patří, state management a kdy žádný, modularizace, která se škáluje, a čistý kód na konkrétních místech v projektu.

09 **Testy a CI/CD.** Co vlastně testovat, Vitest, E2E v Playwrightu, GitHub Actions od nuly, quality gate a preview deploy pro každý PR.

10 **Pohovor a peníze.** Portfolio repo, které zaujme za třicet sekund, technický pohovor, coding task a jak si říct o přidání.

LEKCE 03

Pull request, který se dá zreviewovat

Tohle je nejlevnější věc, kterou můžeš pro svoji pověst v týmu udělat. Nikdo si nepamatuje, jak elegantně jsi vyřešil ten filtr. Všichni si pamatují, kdo posílá PR se 47 změněnými soubory a popisem „fixes".

Recenzent má omezenou pozornost. Prvních deset minut hledá chyby, potom už jen odklikává. Tvoje práce je vejít se do těch deseti minut.

Pravidla, která fungují:

1. **Jeden PR, jedna věc.** Když v popisu potřebuješ „a taky", máš dva PR.
2. **Do 400 řádků diffu.** Nad tím kvalita review prokazatelně padá.
3. **Refaktor odděl od změny chování.** Přejmenování půlky souborů schované v bugfixu je nejjistější způsob, jak si nechat něco proklouznout.
4. **Self-review před odesláním.** Projdi si vlastní diff v prohlížeči, ne v editoru. Uvidíš ho očima recenzenta a půlku poznámek si napíšeš sám.
5. **Popis říká proč, ne co.** Co udělal, vidí recenzent v diffu. Proč to muselo být takhle, ví jenom ty.

ŠABLONA POPISU, KTEROU POUŽÍVÁM

```
## Proč
Filtrování se dělalo na klientovi, u 5k záznamů to zamrzalo.

## Co se změnilo
- filtr přesunut na backend, nový query param `status`
- na FE zůstal jen debounce a URL sync

## Jak to vyzkoušet
1. `npm run dev`
2. /pets?status=available
3. přepni filtr, zkontroluj že se mění URL i request

## Co záměrně NENÍ součástí
- stránkování, řeším v samostatném PR
```

> **Poslední odstavec je ten nejcennější.** Když sám napíšeš, co jsi vědomě neudělal, recenzent to neřeší jako nedodělek. Ušetří to jedno celé kolo komentářů.

Úkol na příště:

- Vezmi svůj poslední PR a rozděl ho na dva podle pravidla „jedna věc".
- Napiš popis podle šablony výše a přečti si vlastní diff v prohlížeči.

LEKCE 06

API vrstvu si ručně nepíšeš

Tohle je lekce, po které mi lidi nejčastěji napíšou, že jim spadla čelist. Ne proto, že by to bylo složité. Proto, že do té doby psali ručně kód, který za ně může napsat generátor.

JAK TO VYPADÁ, KDYŽ SE TO PÍŠE RUČNĚ

```
const [pets, setPets] = useState<any[]>([]);
const [loading, setLoading] = useState(false);
const [error, setError] = useState(null);

useEffect(() => {
  setLoading(true);
  fetch(`${API}/pet/findByStatus?status=available`)
    .then(r => r.json())
    .then(setPets)
    .catch(setError)
    .finally(() => setLoading(false));
}, []);
```

Co je na tom špatně:

- ✕ **any[].** Backend změní pole a ty se to dozvíš od uživatele.
- ✕ **Žádná cache.** Přepneš tab a stahuje se to znovu.
- ✕ **Žádný retry, žádný stale handling.**
- ✕ **Po mutaci refetchuješ ručně** a někde na to zapomeneš.
- ✕ **Ten samý blok napíšeš u každého endpointu znovu.**

JAK TO VYPADÁ S KONTRAKTEM A GENERÁTOREM

```
// hook vygenerovaný orvalem ze swaggeru
// typy sedí s backendem, protože z něj pocházejí
const { data: pets, isLoading, error } = useFindPetsByStatus({
  status: 'available',
});
```

Jeden řádek. Cache, deduplikace requestů, retry, loading i error stavy jsou uvnitř. A hlavně: když backend změní kontrakt, **rozbije se ti build, ne produkce.**

QUERY KEYS BUILDER, BEZ KTERÉHO SE INVALIDACE ZVRHNE

```
export const petKeys = {
  all: ['pets'] as const,
  lists: () => [...petKeys.all, 'list'] as const,
  list: (status: PetStatus) => [...petKeys.lists(), status] as const,
  detail: (id: number) => [...petKeys.all, 'detail', id] as const,
};

// po vytvoření mazlíčka zneplatni všechny seznamy,
// ale nech detaily být
queryClient.invalidateQueries({ queryKey: petKeys.lists() });
```

> **Bez builderu se query klíče píšou jako stringy na dvaceti místech.** Pak přijde překlep, invalidace neprojde a ty hodinu hledáš, proč se seznam neaktualizuje. Viděl jsem to na každém druhém projektu.

Úkol na příště:

- Vygeneruj klienta ze Swaggeru svého projektu a nahraď jeden ruční fetch.
- Napiš keys builder pro jednu entitu a použij ho v invalidaci po mutaci.

LEKCE 09

Co vlastně testovat

Většina lidí testy nepíše ne proto, že by to neuměli, ale proto, že netuší, co má být v testu. Tak napíšou test na to, že se komponenta vyrenderovala, coverage jde nahoru a nikdo nic nechytil.

TEST, KTERÝ NECHYTÍ NIC

Ověřuje, že se komponenta vykreslila. Kontroluje interní stav. Rozbije se pokaždé, když přejmenuješ proměnnou, a projde pokaždé, když rozbiješ funkčnost.

TEST, KTERÝ MÁ CENU

Ověřuje chování, které uživateli slibuješ. Když ho rozbiješ, něco reálně přestalo fungovat. Přežije refaktor, protože nezná vnitřek.

Pravidlo, kterým se řídím:

Testuj to, co by tě probudilo ve tři ráno. Objednávka projde. Přihlášení funguje. Platba se nestrhne dvakrát. Zbytek je bonus.

Rozdělení, které dává smysl:

- **Unit** na čistou logiku: výpočty, formátování, custom hooky. Rychlé, jich je nejvíc.
- **Integrační** na komponentu s daty: vyplním formulář, odešlu, uvidím výsledek.
- **E2E** jen na kritické cesty. Přihlášení, objednávka, platba. Tři až deset scénářů, ne sto.

KONVENCE TESTID, NA KTERÉ SE DOMLUVÍME HNED

```
// blok__prvek, žádné náhodné stringy
<button data-testid="checkout__submit">Objednat</button>
<input data-testid="checkout__email" />

// v testu pak čteš, co se testuje, bez hledání v DOMu
await page.getByTestId('checkout__email').fill('a@b.cz');
await page.getByTestId('checkout__submit').click();
```

> **Storybook je na tohle často overkill.** S jedním klientem jsme místo něj postavili debug obrazovku: jedna routa, výčet komponent a jejich stavů. Pár desítek řádků, vidíš všechno pohromadě a neudržuješ druhou aplikaci vedle té svojí.

Checklist před tím, než test commitneš:

- [ ] Když rozbiju funkčnost, tenhle test spadne?
- [ ] Když jen přejmenuju proměnnou, projde?
- [ ] Testuju chování, nebo implementaci?
- [ ] Je z názvu jasné, co se pokazilo, když spadne?
- [ ] Běží bez připojení k reálnému backendu?

LEKCE 10

Jak si říct o peníze

Tohle je lekce, která se ti vrátí nejrychleji ze všech. Vývojáři umí obhájit volbu state managementu, ale u vlastního platu se rozklepou a nechají si to odbýt větou „teď na to není prostor".

Čtyři kroky, které projdeme:

1. **Sesbírej fakta, ne pocity.** Ne „makám dost", ale „převzal jsem CI, snížil čas buildu o polovinu, zaučil dva lidi". Konkrétní věci s dopadem.
2. **Zjisti trh.** Ne z jednoho inzerátu. Tři až pět reálných nabídek na tvoji roli a lokalitu. Bez toho střílíš naslepo.
3. **Řekni číslo první a řekni ho nahlas.** Kdo řekne číslo první, určuje pole. „Představuju si X" je celá věta, po které se mlčí.
4. **Domluv termín, ne příslib.** Když to nejde teď, ptej se: co konkrétně musí platit a kdy se k tomu vrátíme. Zapiš si to a pošli shrnutí mailem.

Co nikdy neříkat:

- ✕ „Potřebuju přidat, protože mám hypotéku." Tvoje náklady nejsou argument pro firmu.
- ✕ „Kolega bere víc." Rozjedeš tím vyšetřování, ne jednání.
- ✕ „Nějak se domluvíme." Domluvíte se na tom, co navrhne druhá strana.

Realita, kterou je fér říct nahlas: někdy je odpověď ne a je to informace o firmě, ne o tobě. S jedním klientem jsme přesně tohle zažili. Nedostal navýšení, protože ve firmě nikdo neřídil seniority. Ta lekce ho nasměrovala jinam a vyplatilo se to.

Úkol na příště:

- Napiš pět konkrétních věcí, které jsi za poslední rok dodal, každou i s dopadem.
- Najdi tři reálné nabídky na svoji roli a spočítej medián.
- Nahlas si vyzkoušej větu s číslem. Fakt nahlas, ne v hlavě.

KURZ

Co dostaneš

- **Hromada materiálů** ke stažení, plus video tam, kde má smysl vidět živou práci, sdílím obrazovku.
- **Hotové produkční repo** se setupem a quality gates.
- **Referenční projekt** vyúčtování výdajů od setupu po nasazení s CI.
- **Checklisty** na code review, pre-launch a přípravu na pohovor.
- **Certifikát** po zvládnutí závěrečného testu, sdílitelný důkaz, že to fakt umíš.
- **Celoživotní přístup**, vracíš se kdykoliv a bez omezení.

~~9 900 Kč~~ **6 900 Kč**

Zaváděcí cena, pak 9 900 Kč. Jednorázová platba, žádné předplatné, celoživotní přístup.

Osnova, repo i většina materiálů jsou hotové. Zapiš se na čekací listinu na reactivdevs.com/course/production-ready-frontend-developer. Termín spuštění se dozvíš jako první a zaváděcí cenu ti držím. Zápis je zdarma a k ničemu tě nezavazuje.

A když chceš víc:

AI Developer Druhý kurz: jak tohle řemeslo dělat násobně rychleji s AI. Připravuje se.

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

Konzultace Rychlá 1:1 pomoc s konkrétním zásekem.

Kód, co funguje, umí každý. Kód, co vydrží a umíš ho obhájit, je rozdíl mezi juniorem a člověkem, kterého chtějí.