Les 14 — Docenttekst (Autocue)

Online les via Teams — letterlijk voorleesbaar. Slide-nummers tussen `[SLIDE N]`.

> **Setup voor de les:** - Monitor 1 (privé): dit autocue-document fullscreen - Monitor 2 (gedeeld in Teams): Cursor + browser + terminal + slides

>

> **Structuur:** ±1 uur live demo → kwartier pauze → ±1,5 uur "Nu jullie"

>

> **Voorbereiding — accounts:** - GitHub ■ — `gh` CLI ingelogd - Vercel ■ — CLI geïnstalleerd (`npm i -g vercel`) + ingelogd (`vercel login`) - Cursor ■ — Pro of Business plan, ingelogd

>

> **Voorbereiding — voorbereide bestanden:** - Een schone test-folder klaar (`~/web/demo-portfolio` of vergelijkbaar) - `.cursor/rules/general.mdc` snippet copy-paste klaar - `AGENTS.md` snippet copy-paste klaar - Twee feature-prompts copy-paste klaar (hero + about)

>

> **Voorbereiding — backup:** - Een eerder gemaakte demo op standby, voor als live demo vastloopt - Screenshots van Vercel preview-URL's voor het geval Vercel traag is

>

> **Voor uitgebreide handleiding zie `Les14-Docenttekst.md` (normale versie).**

BLOK 1 — Welkom + Vandaag

`[SLIDE 1]`

Welkom bij Les 14. Vandaag verlaten we localhost. We gaan een nieuw Next.js-project scaffolden, koppelen aan GitHub en Vercel, en daarna gaan we daar in Cursor features aan toevoegen. Steeds in een eigen feature-branch, met een eigen preview-URL.

Dit is een live demo van begin tot eind. Ik bouw alles voor jullie ogen op. Stel vragen onderweg — in de chat of zet je microfoon aan. Niet wachten tot het eind.

`[SLIDE 2]`

Wat we vandaag in één voorbeeld gaan zien:

Een vers Next.js-project. Naar GitHub. Naar Vercel. Vercel CLI gebruiken om env vars te beheren. Cursor in detail: chat, plan, build, inline edit, rules, project context, en background agents. Dan twee features

bouwen, elke in een eigen branch, met preview-URL. Eindigen met PR's die we mergen naar productie.

Werkwijze: ik bouw, jullie kijken. Na de pauze doen jullie hetzelfde voor je eigen project.

BLOK 2 — Scaffold + Git + Vercel

`[SLIDE 3]`

Stap één: scaffolden. We gebruiken `npx create-next-app`. Dat is het officiële commando van het Next.js team. Het stelt je een paar vragen — TypeScript ja, Tailwind ja, App Router ja, src-directory ja — en je hebt binnen 30 seconden een werkende app.

(Ik open een terminal, draai create-next-app, doorloop de prompts, en start de dev-server.)

Je krijgt: een werkende Next.js 16-app, op localhost. TypeScript en Tailwind preconfigured. ESLint klaar. En de juiste `.gitignore`, dus `.env.local`, `node_modules` en `.next` komen straks niet in Git terecht.

`[SLIDE 4]`

Stap twee: Git initialiseren en pushen naar GitHub.

Vijf commando's. `git init` maakt een lokale repo. `git add .` zet alles klaar. `git commit -m "init"` maakt de eerste snapshot. `git branch -M main` zorgt dat onze branch `main` heet. En dan `gh repo create mijn-portfolio --public --source=. --push` — dat is de GitHub CLI die in één commando een remote-repo aanmaakt, koppelt en pusht.

(Ik draai deze commando's. Browser naar github.com om te bevestigen dat de code er staat.)

Als je gh niet geïnstalleerd hebt: handmatig op github.com een lege repository maken, daarna `git remote add origin` en `git push -u origin main`. Werkt ook prima.

`[SLIDE 5]`

Stap drie: Vercel koppelen.

We openen vercel.com/new. Klikken **Import Git Repository**. Vercel ziet al onze GitHub-repos — we kiezen `mijn-portfolio`. Framework wordt automatisch gedetecteerd als Next.js. Build-commando en output-directory mag je standaard laten.

Klik Deploy. En dan wachten we ongeveer 45 seconden. Vercel klonet de repo, draait `npm install`, draait `npm run build`, deploy't naar het Edge Network, en geeft een URL terug.

(Ik wacht tot de deploy klaar is, open de URL in nieuwe tab.)

Daar staat ie. Een verse Next.js-app, live op het internet. Vanaf nu: elke push naar GitHub triggert automatisch een nieuwe deploy. Geen extra werk.

`[SLIDE 6]`

Twee soorten URL's krijg je gratis. Productie-URL — die hoort bij `main`. En preview-URL's voor elke andere branch.

Het formaat van een preview-URL is `-git--.vercel.app`. Hele lange URL's, maar je hoeft ze nooit te onthouden — Vercel post ze automatisch in je Pull Requests.

Voordeel van preview-URL's: iedereen kan op je werk-in-progress klikken zonder lokale setup. Geen npm install. Geen `.env.local`. Klikken en kijken.

`[SLIDE 7]`

Environment variables in de Vercel UI. Onze app heeft er nu nog geen nodig — we hebben geen database, geen API key. Maar voor straks: je gaat in deze opleiding apps bouwen met OpenAI-keys,

Supabase-URLs, Stripe-secrets. Die komen allemaal hier.

Belangrijk: drie environments. Production — actief bij deploys vanaf main. Preview — voor alle andere branches. Development — als je `vercel dev` lokaal draait.

En de `NEXT_PUBLIC_` prefix: variabelen met die prefix worden in de browser-bundle gestopt. Zonder prefix zijn ze server-only. Dat is het verschil tussen "iedereen kan dit zien" en "alleen mijn API routes kennen dit".

BLOK 3 — Vercel CLI

`[SLIDE 8]`

Naast het dashboard is er ook een CLI. Die wil ik even laten zien, want voor dagelijks werk is dat veel sneller dan klikken.

Wat je krijgt: `vercel link` om je lokale folder aan een Vercel-project te koppelen. `vercel env pull` om productie-env-vars naar lokaal te trekken. `vercel env add` en `rm` om vars te beheren zonder dashboard. `vercel logs` voor runtime logs. En `vercel --prod` voor een handmatige productie-deploy.

Installeren: `npm i -g vercel`, dan `vercel login`. Klaar.

`[SLIDE 9]`

`vercel link` doe je één keer per project. Het vraagt naar welk scope — je persoonlijke of een team — en welk project. Daarna maakt het een `.vercel/project.json` aan, en weten alle volgende commando's bij welk Vercel-project ze horen.

`vercel env pull .env.local` is mijn favoriete commando. Het trekt alle env vars uit je gekoppelde environment en schrijft ze naar `.env.local`. Eén commando — je hoeft je env vars nooit handmatig te kopiëren.

Tip: standaard pull't hij development env-vars. Wil je preview-vars? `vercel env pull --environment=preview`.

(Ik draai `vercel link` *en* `vercel env pull` *— laat de prompt en output zien.)*

`[SLIDE 10]`

Env vars vanuit terminal beheren. `vercel env add` vraagt om naam en waarde, en welke environment. Je kunt meerdere environments tegelijk geven — `production preview development` — voor één commando dat alles dekt.

`vercel env rm` om te verwijderen. `vercel env ls` om te zien wat je hebt.

Wanneer is dit handig? Als je een API-key roteert. Nieuwe key generen, oude verwijderen, nieuwe toevoegen. Drie commando's in de terminal — geen dashboard nodig.

BLOK 4 — Cursor in detail

`[SLIDE 11]`

Cursor. Nu het hoofdgerecht van vandaag.

Wat is Cursor: een VS Code-fork met AI native ingebouwd. Niet een extensie. Het IS de editor. Werkt met Claude, GPT, Gemini en lokale modellen — je kiest per chat welk model.

Wat krijg je boven gewone VS Code? Chat met je hele codebase als context. Plan-mode — eerst nadenken, dan doen. Build-mode — laat de AI autonoom uitvoeren. Inline edit met Cmd+K. Background agents voor parallel werk. Cursor Rules per project. En tab-completion die meekijkt met je hele bestand, niet alleen je laatste regel.

`[SLIDE 12]`

Cursor 3 heeft twee hoofdvensters. Belangrijk om hiermee te beginnen, want het is in versie 3 nieuw.

Aan de ene kant: het **Editor Window**. Dat is de klassieke VS Code-stijl IDE — file explorer links, editor in het midden, terminal beneden. Daar schrijf je code, review je diffs, gebruik je extensies. Dat is je hoofdwerkplek.

Aan de andere kant: het **Agents Window**. Dat is een nieuwe interface, een nieuwe interface waar alle agents zichtbaar zijn. Lokale agents, cloud-agents, remote agents, agents die je vanaf je telefoon hebt getriggerd. Alles op één plek.

Je wisselt tussen beide met **Cmd+Shift+N**. Dat is de belangrijkste sneltoets van vandaag — onthoud die.

Binnen het Editor Window heb je een chat-paneel — Cmd+L opent het. En in dat chat-paneel kun je kiezen welke modus de AI gebruikt. Eén van die modi is **Agent mode** — daar gaan we het zo over hebben. Plus Cmd+K voor snelle inline-edits in één bestand.

Gouden regel: code schrijven of één feature bouwen → Editor Window met Agent mode in chat. Meerdere features parallel → Agents Window met background agents.

`[SLIDE 13]`

Editor Window in detail. Dit is waar je 80 procent van je tijd doorbrengt.

Stap één: **Cmd+Shift+N** — als je in Agents Window zat, switcht hij naar het Editor Window. Je ziet nu de klassieke layout: file explorer links, editor in het midden, terminal onderaan.

Stap twee: dit voelt als VS Code. Cmd+P opent de file-search. Tab accepteert AI auto-completion suggesties. Bekende sneltoetsen werken allemaal.

Stap drie: voor een snelle inline edit selecteer je code in de editor en druk je **Cmd+K**. Er opent een input-balkje boven je selectie. Je typt wat je wilt — bijvoorbeeld "rename deze variabele naar firstName" — en Cursor toont de diff direct in de editor. **Tab** om te accepteren, **Esc** om te rejecten.

Stap vier: voor langere taken open je het chat-paneel met **Cmd+L**. Het opent rechts, naast je editor.

Wanneer gebruik je het Editor Window? Voor je standaard coding-flow. Als je veel files tegelijk wilt zien met split screens. Voor VS Code extensies — linters, formatters, debuggers — die allemaal gewoon werken.

(Ik open mijn Editor Window, doe een Cmd+K op een variabele-naam.)

`[SLIDE 14]`

Agent mode. Dit is de AI-modus *binnen* het chat-paneel in het Editor Window.

Stap één: ik ben in het Editor Window. **Cmd+L** opent het chat-paneel rechts. Stap twee: bovenaan de chat zie ik een mode-selector — een dropdown of een rij icoontjes. Ik kies **Agent**.

Stap drie: ik typ niet een specifieke kleine wijziging, maar de HELE taak. Bijvoorbeeld: "Bouw een /about pagina met een team-grid in onze huisstijl, baseer de styling op onze hero-section."

Stap vier: ik druk Enter. De agent start. Stap vijf: in het chat-paneel verschijnt een lopende takenlijst — de agent vinkt zijn eigen stappen af terwijl hij werkt.

Belangrijk om te weten: in Cursor 3 heeft Agent mode **geen limiet** op het aantal tool-calls per taak. De agent kan dus meerdere files editen, terminal-commando's runnen, en zelfs web-content browsen. Hij

vraagt wel altijd eerst voordat hij een terminal-commando uitvoert.

Stap zes: aan het einde verschijnt een samenvatting van alle wijzigingen, per bestand een diff. Stap zeven: **Keep** of **Accept all** om alles te accepteren, of je accepteert per file.

Wanneer kies je Agent mode? Voor grotere features die meerdere bestanden raken. Refactors. Nieuwe pagina's bouwen. API-routes opzetten. Als je het plan vertrouwt en aan het eind tijd hebt om goed te reviewen.

Belangrijke tip: hou je prompt specifiek. "Bouw een about-pagina met team-grid" is goed. "Bouw een mooie website" is veel te open. Dan krijg je een gok.

(Ik laat een Agent mode demo zien — kleine maar concrete taak.)

`[SLIDE 15]`

Background agents — het derde stuk. Dit is wat Cursor 3 echt vernieuwt.

Wat het is: async agents die in geïsoleerde cloud-VMs draaien. Niet in jouw editor — vandaar background. Ze werken parallel in eigen Git branches, en je beheert ze via het **Agents Window**.

Hoe kom je er? **Cmd+Shift+N** — switcht van Editor Window naar Agents Window. Daar zie je een lijst-overzicht met alle running en done agents.

Hoe maak je er een aan? Stap één: in het Agents Window klik je op **New agent** of het plusje bovenaan. Stap twee: kies een environment. Local — draait op je laptop. Cloud — draait op Cursor's servers in een geïsoleerde VM. Remote SSH — als je een eigen server hebt. Voor vandaag pakken we cloud, want dat is de meest interessante demo.

Stap drie: beschrijf de taak. Wees specifiek, alsof je een ticket aanmaakt voor een collega. Stap vier: kies optioneel een branch-naam. Stap vijf: klik **Start**.

Stap zes: de agent draait nu in een cloud-VM. Je kunt **Cmd+Shift+N** terug doen naar het Editor Window en gewoon verder werken. De agent babysit zichzelf.

Hoe haal je resultaten op? Stap één: **Cmd+Shift+N** terug naar Agents Window. Stap twee: klare agents staan op **Done**. Stap drie: ik klik op de agent — ik zie de diff, de commit message, en alle stappen die hij heeft genomen.

Stap vier: **Open in PR** maakt automatisch een Pull Request op GitHub. Stap vijf: Vercel pakt het op en maakt een preview-URL. Stap zes: review in browser, merge op GitHub.

Wanneer gebruik je dit? Voor twee tot vier parallelle features. Voor lange refactors die je niet wilt babysitten. Of als je vanaf je telefoon, Slack of Linear een taak triggert terwijl je in een vergadering zit.

(Ik open Agents Window met Cmd+Shift+N en demonstreer een lege agent.)

`[SLIDE 16]`

Context — hoe vertel je Cursor waar hij naar moet kijken.

In chat type je een at-teken en je krijgt een autocomplete. at-file:foo.tsx voor één bestand. at-folder:components voor een hele map. at-code:functionName voor een specifieke functie of class. at-docs voor externe documentatie. at-web voor een live web-search. at-git voor git history of blame.

Voorbeeld: "Refactor at-file UserCard.tsx zodat het at-code useUser uit at-folder hooks gebruikt." Cursor weet nu exact welke bestanden in scope zijn.

Hou de scope klein. Geen 50 bestanden in één chat — Cursor wordt dan slechter, niet beter.

`[SLIDE 17]`

Cursor Rules. Per-project instructies aan de AI.

Wat zijn ze: Markdown-bestanden in `.cursor/rules/`. Worden aan ELKE prompt toegevoegd als systeem-instructie. Met YAML front-matter kun je zeggen `alwaysApply: true` of een glob meegeven — bijvoorbeeld alleen actief op `**/*.test.ts` bestanden.

Voorbeeld inhoud: gebruik TypeScript strict mode. Tailwind voor styling, geen CSS modules. Server components by default, `use client` alleen waar nodig. Imports met de `@/` alias.

Effect: je hoeft niet elke keer "vergeet niet TypeScript te gebruiken" in je chat te typen. Cursor weet het al.

(Ik open `.cursor/rules/general.mdc` *en plak mijn voorbereide regels.)*

`[SLIDE 18]`

Project context. Naast Cursor Rules zet je ook een `AGENTS.md` of `PROJECT.md` in je repo-root.

Verschil: Cursor Rules zijn instructies — "doe het zo". AGENTS.md is documentatie — "zo zit deze codebase in elkaar". Tech stack en waarom je die keuze hebt gemaakt. Belangrijke patronen, bijvoorbeeld hoe auth werkt of hoe je errors afhandelt. Hoe je tests draait, hoe je deployed. Pointers naar belangrijke bestanden.

Mooi detail: AGENTS.md wordt ook gelezen door Claude Code en GitHub Copilot. Eén bestand, meerdere AI-tools.

(Ik laat mijn voorbereide AGENTS.md zien.)

`[SLIDE 19]`

Zeven tips die je van Cursor-amateur naar Cursor-power-user brengen.

Eén — Cmd+I voor Composer met multi-file context, niet alleen Cmd+L. Twee — at-everything gebruiken, ook at-recommended laat Cursor zelf bestanden voorstellen. Drie — plan in stappen accepteren, niet alles in één klap. Vier — `alwaysApply: false` rules voor opt-in conventies die je per-chat wilt activeren. Vijf — Auto-mode laat Cursor het beste model per taak kiezen.

Zes — Cmd+Shift+Enter geeft je een diff-preview voordat je accept. Zeven — Notepads zijn herbruikbare prompt-snippets per project. Handig voor "schrijf een React component met..." die je vaak gebruikt.

Anti-pattern: alles in één gigantische chat dumpen. Hou je chats focused per feature.

BLOK 5 — Live feature-workflow

`[SLIDE 20]`

Nu zetten we alles bij elkaar. De workflow voor één feature, van idee tot preview-URL.

Vijf stappen. Plan in Cursor — beschrijf wat je wilt. Branch — `git checkout -b feature/x`, en Cursor kan dit ook voor je doen. Build — laat Cursor de code schrijven via Plan + Apply. Commit en push — Vercel start automatisch een preview-deploy. En open de preview-URL — test, geef feedback, itereer.

Cyclus duurt ongeveer tien minuten per feature. Inclusief AI denkwerk. Dat is wat het bouwen versnelt — niet dat AI mooier code schrijft, maar dat de cycle korter is.

`[SLIDE 21]`

Live demo één: een hero-section voor onze homepage. Via Plan plus Build.

(Ik doe live:)

- `git checkout -b feature/hero`

- Cursor chat: *"Plan een hero-section voor de homepage. Gradient achtergrond, een grote titel, ondertitel, twee CTA-knoppen."*
- Plan verschijnt — we lezen het door, eventueel kleine tweaks vragen
- Apply — Cursor maakt de wijzigingen in `app/page.tsx`
- Cmd+K voor inline tweak — bijvoorbeeld kleur van de gradient
- `git add . && git commit -m "feat: hero section"`

Tot zover binnen Cursor. Nu pushen.

`[SLIDE 22]`

`git push -u origin feature/hero`.

Binnen 30 tot 60 seconden begint Vercel een preview-build. Het verschijnt in het dashboard. Als ik nu meteen een PR open op GitHub, post de Vercel-bot zelf de URL als comment.

(Ik open de preview-URL.)

Daar is mijn hero-section. Live, op een unieke URL. Ik kan deze URL nu sturen naar wie ik maar wil — een collega, een klant — en ze klikken erop en zien exact wat ik heb gebouwd.

`[SLIDE 23]`

Live demo twee: een about-pagina. Maar nu via een Background agent. Parallel werken.

(Ik doe live:)

- `git checkout -b feature/about`
- In Cursor: *"Open background agent: bouw een /about pagina met team-leden in een grid. Gebruik onze hero-section als stijlreferentie."*
- Agent start in een eigen tab — ik kan in mijn main editor blijven werken
- Ik laat zien dat ik bijvoorbeeld een nieuwe Inline edit doe op een ander bestand
- Na een paar minuten meldt de agent zich klaar
- Diff review — apply
- `git add . && git commit -m "feat: about page"`
- `git push -u origin feature/about`

Tweede preview-URL is binnen een minuut online. Twee features tegelijk, twee preview-URL's, en ik heb intussen nog wat anders gedaan ook.

`[SLIDE 24]`

PR's mergen.

We hebben nu twee branches op GitHub. Op github.com open ik beide PRs. In elke PR staat de Vercel-comment met de preview-URL. We klikken erop, controleren dat alles werkt zoals bedoeld.

Mergen naar `main`. Vercel detecteert de merge en bouwt automatisch een productie-deploy. Productie-URL update vanzelf.

Mentaliteit om mee te nemen: main is altijd deploy-baar. Feature branches zijn je experimenteer-ruimte. Niets in main wat niet door een preview is gegaan.

BLOK 6 — Recap + Nu jullie

[SLIDE 25]

Wat we vandaag gedaan hebben.

Een vers Next.js-project gescaffold met `npx`. Git repo aangemaakt en gepushed naar GitHub. Vercel-project gekoppeld en eerste deploy. Vercel CLI gebruikt om te linken en env vars te pullen. Cursor in detail bekeken — chat, plan, build, inline edit, rules, AGENTS.md, en background agents. Twee features gebouwd in eigen branches met eigen preview-URL's. PR's gemerged naar productie.

Wat je nu kunt: elke nieuwe app vanaf nul opzetten met deze flow. Cursor en Vercel zijn je standaard-werkbank voor alles wat je verder bouwt in deze opleiding en daarna.

[SLIDE 26]

Na de pauze gaan jullie hetzelfde doen. De opdracht staat in de Lesopdracht-PDF.

Kort: nieuw `npx create-next-app`-project voor jezelf — kies een onderwerp dat je leuk vindt. Push naar je eigen GitHub. Vercel project + eerste deploy. `vercel link` plus `vercel env pull`. Open in Cursor — schrijf één eigen `.cursor/rules/general.mdc` en één `AGENTS.md`. Bouw twee features in eigen branches via Cursor. Push beide → twee preview-URL's. Open één PR, deel beide URL's in de chat.

Je hebt anderhalf uur. Vraag in de chat als je vastloopt — ik kijk mee.

[SLIDE 27]

Volgende week Les 15: RAG en embeddings.

We bouwen een PDF Q&A-app.; Je laadt een PDF op, en je kunt de AI vragen stellen over de inhoud. Antwoorden komen uit jouw documenten, niet uit de training-data van het model. Dat gaan we from scratch bouwen met embeddings, vector search via pgvector, en context-injectie.

Voorbereiding: je Next.js-app van vandaag moet live staan op Vercel. Je hebt minstens twee PR's gemerged met preview-URL's. Cursor draait, en je AGENTS.md en `.cursor/rules/` zijn ingevuld.

Tot volgende week.

Bijlage A — `.cursor/rules/general.mdc` (copy-paste klaar)

```
---
description: Algemene project-conventies voor mijn-portfolio
alwaysApply: true
---

# Code Style
- Gebruik TypeScript strict mode
- Geen any-types; geef expliciete return types op functies
- Tailwind voor styling — geen CSS modules
- Server components by default; "use client" alleen waar nodig
- Imports: gebruik @/ alias voor src/

# Bestandsstructuur
- Componenten in src/components/<naam>/index.tsx
- Page-specifieke componenten in app/<route>/_components/
- Hooks in src/hooks/<naam>.ts

# Algemeen
- Geen console.log in committed code
- Errors: gebruik next/error of toast (geen alert)
- Async: prefer await over .then chains
```

Bijlage B — `AGENTS.md` (copy-paste klaar)

```
# Mijn Portfolio — Project Guide

## Stack
- Next.js 16 (App Router)
- TypeScript strict
- Tailwind CSS v4
- Vercel hosting

## Belangrijke conventies
- Server components by default
- "use client" alleen voor interactie / hooks
- @/ alias verwijst naar src/
- ESLint moet groen zijn voor merge

## Hoe lokaal draaien
\`\`\`bash
npm install
vercel env pull .env.local   # synct env vars
npm run dev                  # http://localhost:3000
\`\`\`

## Hoe deployen
- Productie: merge naar main → Vercel deploy't automatisch
- Preview: push naar feature branch → Vercel maakt preview-URL

## Belangrijke bestanden
- app/layout.tsx — root layout met fonts en theme
- app/page.tsx — homepage met hero
- src/components/ — herbruikbare UI-componenten
```

Bijlage C — Feature-prompts (copy-paste klaar)

Feature 1 (Plan + Build):

```
Plan een hero-section voor de homepage van mijn-portfolio.

Specs:
- Gradient achtergrond (subtiel, twee blauwtinten)
- Grote titel: "Tim Rijkse"
- Ondertitel: "AI Developer & Docent"
- Twee CTA-knoppen: "Projecten" (primair) en "Contact" (secundair)
- Responsive: stack op mobiel, naast elkaar op desktop
- Gebruik Tailwind, geen custom CSS
```

Feature 2 (Background agent):

```
Open background agent.

Bouw een /about pagina met team-leden in een grid. Specs:
- 6 team-leden (mock data is prima)
- Grid: 1 kolom op mobiel, 2 op tablet, 3 op desktop
- Per persoon: foto-placeholder, naam, rol, korte bio
- Stijl: matched onze hero-section (zelfde gradients en typografie)
- Header bovenaan met titel "Het team"
```