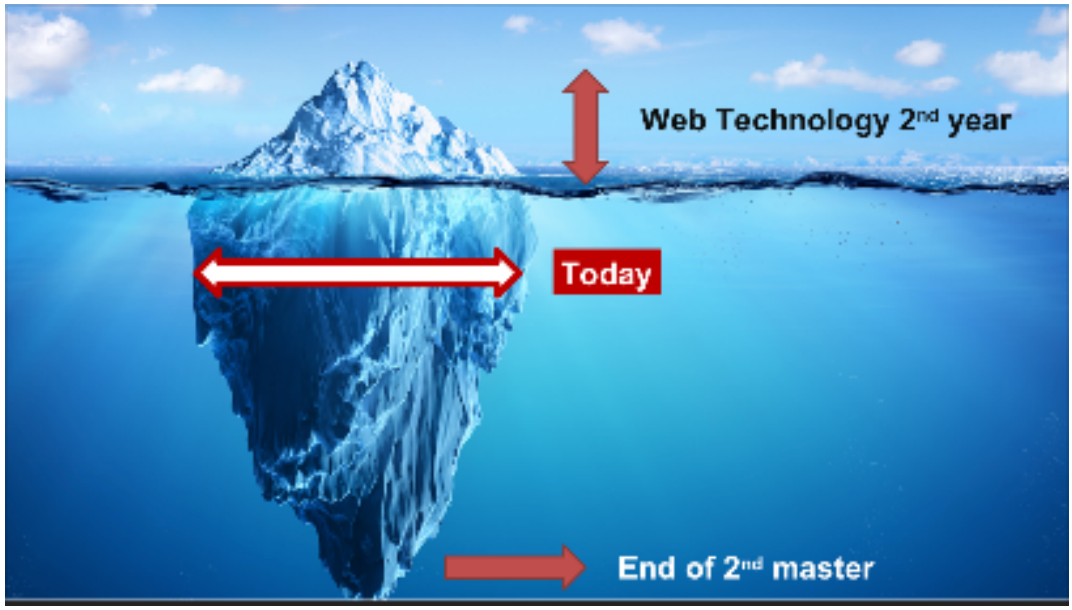
Web development
25/02/2021

Slides kom online voor de modern web development

Web development bij web was zeer weinig informatie.
Twas redelijk basis



Twas eigenlijk alles behalve voldoende

- Complex topic
- Difficult voor new programmers

Frontend en Backend zijn veel vager nu.
We gaan niet een full coding tutorial zijn.

Het is niet echte code → pseudocode

Alle topics

Frontend → client side interface en javascript en dergelijke
Backend → Waar de data vandaan komt (database)

Niet zo'n goed idee om een directe weg hebben tussen deze 2
→ data aanpassen, implementatie details weten, (mag allemaal niet)

Php is wel een goede code de dag van vandaag
→ afscherming!

Je voegt iets nieuws to aan je database:
Je moet een nieuwe pagina herladen

We kunnen het misschien
- Dynamischer
- Flexiber
- Aangenaam

Maken

Ajax was hier de oplossing voor
- Zonder page refresh
- Wow zo geweldig toen

Fetch() is eigenlijk beter en nieuwer (vervangen)

DOM → document object model → effectief renderen en je kan ook manipulaties doen

Rise of javascript

→ dualiteit (2 plaatsen waar de verantwoordelijkheid ligt om uw html code aan te passen)

jQuery → ken je ookwel
Laat toe om query'en van de dom veel gemakkelijker te maken
Abstractielayer die het gemakkelijker maakte om de dom te manipuleren

querySelector is ontstaan door jQuery omdat zij dit eerst hadden

(Nog altijd dualiteit maar veel gemakkelijker gewoon door jQuery)

Is dit beter?

Backend genereert html en dom insert doen

Backend → maak data
Frontend →

NodeJS → server side javascript
Op uw computer of software op de server

Server side en backend zijn javascript
→ isomorphic javascript (verliest de dualiteit)

NodeJS → Node package manager
We gaan een gigantische repository samenhouden met heel veel code snippits zodat we die kunnen afhalen
Je wil niet 20 libraries binnentrekken maar je kan ook 1 grote file binnentrekken

PHP-era
Frontend → ook php
(Handlebars)

Front-end frameworks
Fat-clients

Data komt van de server
Alles via AJAX opvragen
Weg dualiteit

Moderne frameworks (single page applications)
→ weinig refreshen (alleen stukjes vervangen die nodig zijn)

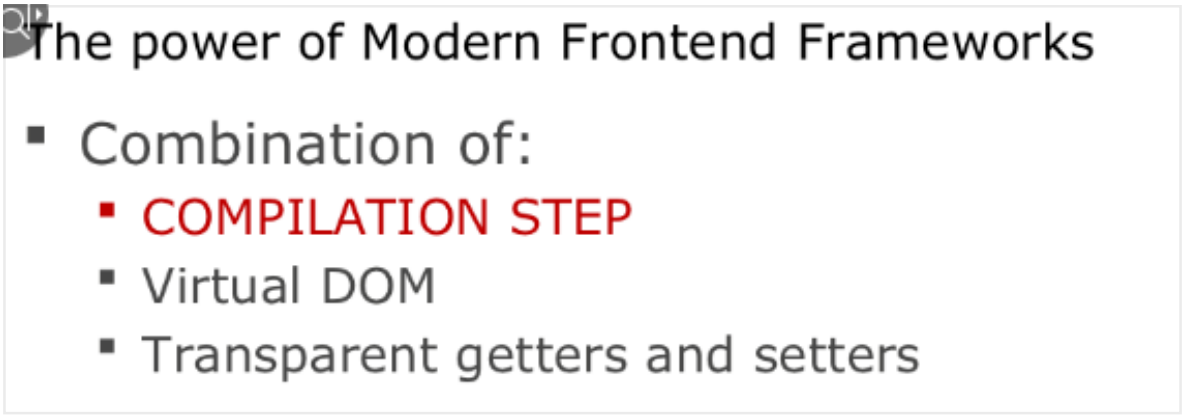
Als je een element aanpast in de DOM moet je de hele DOM refreshen eig
Vb: reddit → als je voor elk single element een update moet doen → epileptisch reageren → puur traag

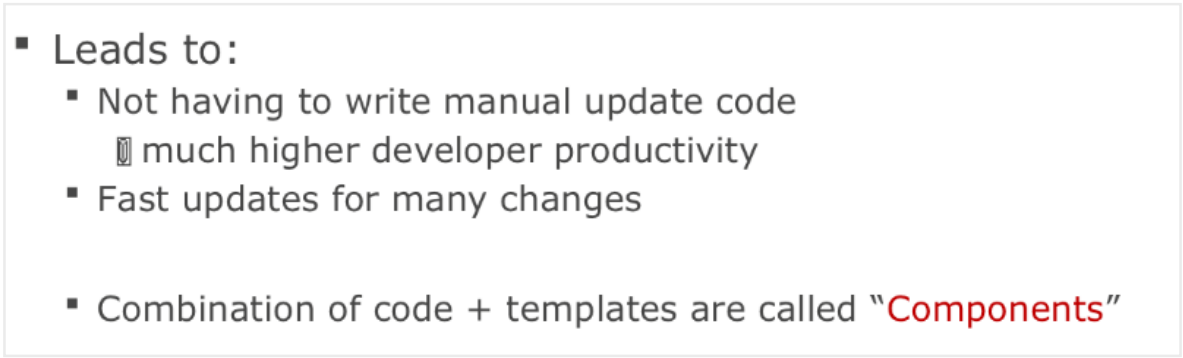
Reactive data binding
→ hoe werkt dit
Je hebt eig niet super veel nood om dit echt te weten
Maar er zijn enkele excepties

Achterliggend → transparant getter en setters gemaakt
Als je iets aanpast of iets toevoegt → setter wordt opgeroepen
Notify wordt gegeven aan de watchers en die zal dan de datarepresentatie refreshen

Uw DOM gaat eigenlijk dan altijd aangepast worden (en flikkeren)
Virtual DOM is een kopie in uw dom → als je wijzigen doet zie je dit niet

1. Doe een heel tal wijzigen
2. Kijken welke wijzigen er zijn geweest (echte dom zal dit dan bekijken op een bepaald interval)
3. Dan wordt het echt verandert





Manueel moet je html aanpassen, Ajax, errorcode, …
Super veel overhead die je niet wilt hebben

Frameworks vermijden dit door het zelf te doen

Alles samen
.view → css, template, ...

State stores (Redux, Vuex)
Je mag niet dat elk component elke backend communicatie de hele tijd doet
Je moet 1 grote staat hebben

Singleton of Abstractielaag

Je moet hierbij niet je Ajax doen :)

(Full circle)
Steep learning curve → online resources help a lot
Make sure you look at the recente stuff → (2jaar is al te lang geleden)

Het maakt het makkelijker maar ook moeilijker

Single page application is heel resource intensive

Frameworks zijn heel belangrijk bij interactie bij de backend
Als je niet super veel interactie nodig heb moet je niet een framework gebruiken (super niet toepasselijk)

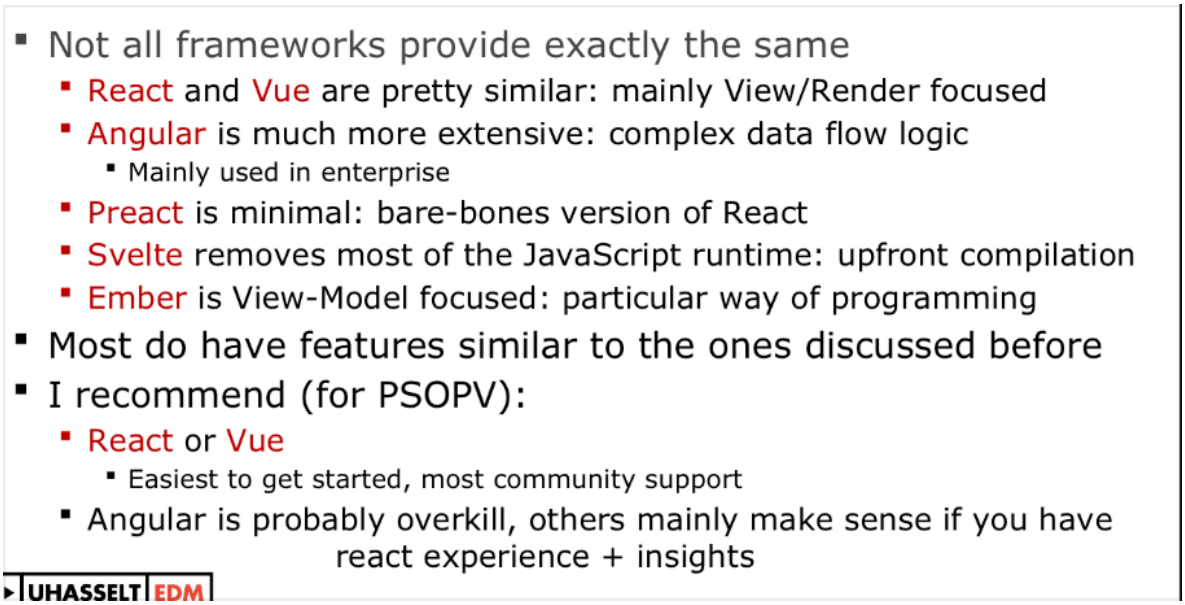
Client is wel heel snel? Toch stom
Terug op de server proberen hahaha

Server side rendering (SSR)
Virtual dom doorsturen (gerenderd op de server) naar de client
Je moet in de frontend, buttonlistener (moet lokaal)

Je kan heel snel displayen
Maar niet snel interactie

(2 keer renderen)
→ inladen → maar je kan niet interactie

Niet optimaal → maar komt wel vaak voor



React en Vue zijn goed voor het project
Preact (bare bones versie van react)
Svelte → removes most of javascript runtime (initial time to load issues)

Is het nodig om geavanceerde frameworks te gebruiken?
→ als je je analyse doet
(jQuery, ...)

Als je zeer complexe interactie hebt je pagina (Framework toch)
→ WAT?
→ WAAROM? Dit is de belangrijkste
Zorg dat de dingen die je kiest van taal, juist zijn

Part 2

Typescript and friends
Javascript is untyped: super flexible but very error-prone

- TypeScript (TS) is “JavaScript with types”

Heeft een full type system
- Enkel in compilatie stap wordt dit gecheckt
- Flexible is er nogsteeds: “any” type

Not executed by the browser
- Er komt javascript uit!

PROTIP: fantsoenlijke logging (in de console peren)

Er zijn nog andere
- CoffeeScript
- Flow
- Dart

→ zijn in het algemeen minder supported, minder powerful, less easy

WebAssembly
→ low level programming for het web
(Zet c en c++, Rust, TypeScript om in javascript)
→ Je kan niet aan de dom
(Je kan eigenlijk alleen maar met de data werken)

Gebruik zeker typescript
WebAssembly gaat waarschijnlijk niet van toepassing zijn

Babel
→ magische stap van framework naar een ding wat de browser kan compileren
Transpiler → van een taal naar een andere taal omzetten

Javascript wordt geüpdatet maar de browsers niet
→ babel zorgt dat je dingen nog kan supported

CSS: The CS student’s bane
Gebruik templates → niet zelf maken als je er niets van bakt
Gebruik **bootstrap, materialize**

CSS Reset → predefined lijst van alle elementen (uniform trekken naar dezelfde stijl)

Sass Less

Kan je lekker variabelen define

Als je een preprocessor hebt wil
Moet niet samen met een bootstrap gebruiken
(Zelf iets designer, sure gebruik maar een preprocessor)

Unit testen → heel erg handig
Maar moet niet bij dit PSOPV