GIVE ME A FUCKING

FRONTEND DEVELOPMENT LES 1

WAAROM DIT VAK?

Nou?

Vraag dit aan je studenten, schrijf het eventueel op het bord. Benadruk al deze punten ook.

- Materiaalkennis
- Hoe weet je wat je kunt ontwerpen als je niet weet wat er wel en niet kan?
- Hoe weet je of iets goedkoop of duur is om te maken?
- Waar houden je collega's zich mee bezig?
- Het is tof!

FRONTEND DEVELOPMENT WAS EEN ROTVAK

- Slechte browsers
- Slechte ontwerpers

Tot niet zo lang geleden was je als Frontend developer vooral iemand die de idiote quirks van de verschillende browsers kende. Bovendien kende je allemaal hacks om de onmogelijke designs van fantasierijke ontwerpers tóch mogelijk te maken.

FRONTEND DEVELOPMENT IS TOF

- Browsers die normaal doen
- Betere ontwerpers
- Meer, toffere mogelijkheden
- Slechte browsers zijn bijna dood
- Slechte ontwerpers zijn managers geworden

Tegenwoordig kan je aan *browser spitting* doen: als code geschreven is volgens de specificatie, dan kan je er vanuit gaan dat het werkt in alle browsers. Natuurlijk nog niet helemaal waar, maar inmiddels véél beter dan vroeger. Ontwerpers zijn ook beter geworden. En er is meer mogelijk: browsers zijn beter, sneller, er kan veel meer met CSS.

HOE KAN DAT?

Wat is er veranderd?

Kort geschiedenislesje vanaf de volgende slide. Je zou nu op ongeveer 5 minuten van je hoorcollege moeten zitten. Niet veel langer!

VROEGER...

1. Browsermonopolie
2. Device monopolie
3. Weinig standaarden
4. Het web werd niet begrepen

Toen het web nog zwart/wit was, hadden we IE als browsermonopolie. We hielden alleen rekening met vaste schermformaten en manieren van input (muis). Standaarden werden slecht ondersteund. Hierdood werd het web niet echt begrepen: het was iets met computers. En met computers kan alles, was de gedachte.

XHTML

Theoretisch perfect

Een poging tot verbetering kwam in de vorm van XHTML. Het idee was om alles weg te gooien en helemaal opnieuw te beginnen met een soort Bijlmereske utopie. Theoretisch perfect, zoals een Rietveldstoel…

XHTML

Draconische foutafhandeling

…maar er zaten een paar zeer negatieve kanten aan: drakonische error handling. Hier een oude uitleg: http://annevankesteren.nl/2005/11/draconian

XHTML

Niet backwards compatible

En niet backwards compatible. Alles wat er tot dan toe op het web bestond zou kapot zijn.

HTML

- Praktisch
- Tolerant
- Backwards compatible

Als tegenreactie kwam de WHATWG (bonuspunten voor wie weet waar dat voor staat). Met een veel pragmatischere aanpak

WHATWG

Veel pragmatischer

Gericht op standaardiseren van alles

Het web is misschien wel stuk, maar het werkt wel. En er zijn gewoon teveel mensen die het gebruiken. Beter om datgene wát ze gebruiken te standaardiseren.

STANDAARDISERING

- Hoe moet code werken in een browser
- Hoe moet *slechte* code werken in een browser

Er wordt gestandaardiseerd hoe valide code moet renderen in de browser. Maar er wordt ook gestandaardiseerd hoe invalide code moet renderen. WHATWG geniet de steun van alle browsers. Dit is belangrijk: dit stelt ons in staat om te *browser spitten*. We kunnen er vanuit gaan dat alles overal hetzelfde rendert. Wat staads vaker ook zo is. Hierdoor wordt ons vak weer leuk. Lees dit hoofdstuk even om je geheugen op te frissen: http://html5forwebdesigners.com/design/index.html

DESIGN PRINCIPES HTML5

- Do not reinvent the wheel
- Pave the cowpaths
- If it ain't broke, don't fix it

Geen dingen opnieuw uitvinden die al werken. Bijvoorbeeld drag and drop (hoewel dat wellicht wél opnieuw uitgevonden had moeten worden). Als mensen bepaalde patronen herhalen, standaardiseer ze dan, zoals veelvoorkomende classnames vervangen door semantische elementen.

IN CASE OF CONFLICT

1. users
2. authors
3. implementers
4. specifiers
5. theoretical purity

Hoe los je iets op als er een conflict van belangen bestaat bij een bepaald feature. Dit is een pracht van een beslisboom natuurlijk. Die zouden meer organisaties moeten gebruiken. Dan kregen we bijvoorbeeld geen termen als V1 voor tweedejaars, of V2 voor derdejaars. Users (bezoekers), authors (designer, developers, content mensen), implementers (browser bouwers), specifiers (mensen die de HTML spec schrijven).

PAVE THE COWPATHS

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
```

```
<meta charset=utf-8>
```

Hoe werken sommige dingen nu echt? Zoals bijvoorbeel DOCTYPE. Of Meta Charset. Het bleek dat mensen die bizarre string niet begrijpen. Niet alleen de mensen die HTML typen, ook de mensen die de browsers maken niet. Browsers zoeken heel pragmatisch naar de string charset=utf-8 en negeren de rest.

PAVE THE COWPATHS

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR
```

```
<!DOCTYPE html>
```

```
<!doctype html>
```

Browsers snappen, net zoals mensen, niks van al die bizarre letters. Alleen doctype html doet er toe. Schreeuwen doen we alleen op CAPSLOCK DAY. Fijn met kleine letters

NIEUWE ELEMENTEN

- Header
- Footer
- Aside
- Nav
- Main

Vraag of iemand weet waar deze dingen voor zijn. Ze zijn nogal vanzelfsprekend. Leg uit dat ze niet alleen op pagina-niveau gebruikt kunnen worden, maar ook op element-niveau: een article kan prima een footer en een aside hebben. Er is discussie over het main-element, of dat maar één keer of vaker op een pagina mag staan, en of het überhaupt wel nodig is. Je kan ze beschouwen als specifieke sectioning elementen. Lees de precieze details hier nog even door: http://html5forwebdesigners.com/semantics/index.html Video en Audio behandelen we later, bij de lessen over responsive, bij het concept dat er veel verschillende verschijningsvormen zijn.

NIEUWE ELEMENTEN

- Article
- Section

Deze zijn lastiger, en vaak inwisselbaar. Een article kan uit meerdere sections bestaan (bijvoorbeeld de secties in een lang artikel) en een section kan meerdere articles hebben (bijvoorbeeld een thematische sectie op een homepage met daarin een aantal items). Lees dit er nog even op na: http://html5forwebdesigners.com/semantics/index.html#article en dit http://html5forwebdesigners.com/semantics/index.html#section Noem een paar voorbeelden. Noem ook de sectioning flowchart die helpt bij het kiezen van het juiste sectioning-element: http://www.html5doctor.com/downloads/h5d-sectioning-flowchart.pdf

FORMULIEREN

De HTML5 spec begon als reactie op het feit dat formulieren niet krachtig genoeg waren. Vandaar een heleboel nieuwe attributen. Minder elementen hier.

NIEUWE INPUT TYPES

```
<input type="text">
<input type="email">
<input type="url">
<input type="number">
<input type="tel" required>
<input type="text" pattern="[1-9][0-9]{3}\s?[a-zA-Z]{2}">
```

Nieuwe types. Lekker makkelijk. Handig ook. In de volgende slides staan twee filmpjes gemaakt op een iPhone. De ene zonder deze attributen, de andere met.

SECTIONS IN FORMULIEREN

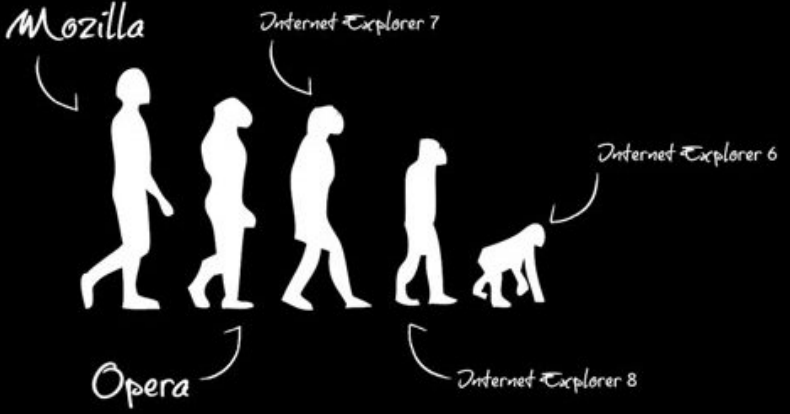
```
<form>
        <fieldset>
            <legend>Contactgegevens</legend>
            …
        </fieldset>

        <fieldset>
            <legend>Voorkeuren</legend>
            …
        </fieldset>
</form>
```

Dit zouden ze moeten weten, vraag het even voor je het laat zien. Goede vraag voor tijdens het mondeling.

BROWSERS EN DEVICES

Alles wat op het web staat moet op alle mogelijke apparaten werken. Kleine schermen, grote schermen, goede schermen, slechte schermen, snelle en trage computers en alle mogelijke vormen van input (muis, vinger, toetsenbord, stem, etc). Er zijn twee manieren om dit te doen: graceful degradation, progressive enhancement.

GRACEFUL DEGRADATION



Eerst iets bouwen wat werkt in de beste browser en daarna, zo nodig, aanpassen zodat het werkt in oude browsers.

Evolution of browsers

PROGRESSIVE ENHANCEMENT

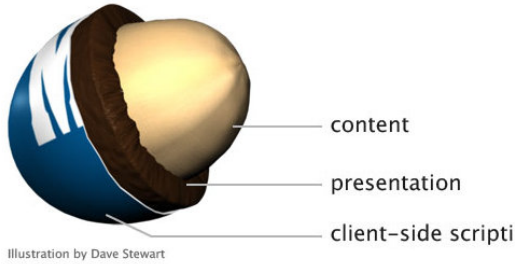


Illustration by Dave Stewart

Iets opbouwen in lagen. Als een browser/apparaat een feature ondersteunt, dan is dat leuk, maar nooit noodzakelijk. Mooi voorbeeld van PE is een roltrap. Als ie niet rolt dan doet ie het nog. Einde van het hoorcollege. Dit zou in een half uur moeten passen.