hogeschool vives

Web Development 1 : labo 1 - HTML

studiegebied **HWB**

bachelor in het **toegepaste Informatica**

campus **Kortrijk**

academiejaar **2024-2025**

Inhoudsopgave

1 Inleiding

In deze les wordt kort de werking van "het internet" toegelicht, vanuit het oogpunt van onze browser. De inhoud van dit vak gaat vooral over de client kant (lees: de browserkant) van het internet:

- Hoe er met de server kant gecommuniceerd wordt (deze les)
- Hoe we inhoud, visualisatie en gedrag van webpagina's kunnen vastleggen (verdere lessen)

1.1 Gebruikte software tools

- Google Chrome browser downloaden en installeren.
- Chrome developer tools (kun je zichtbaar maken in Chrome door op F12 te drukken).
- Webstorm downloaden (zie leeromgeving) en installeren.

1.2 Verslag

In dit deel van de cursus staan verschillende vragen die je moet beantwoorden en opdrachten om iets te maken of uit te proberen. Het is belangrijk dat je alle opdrachten zorgvuldig uitvoert!

Documenteer je werk in een verslag document (pdf of docx) “**verslag Internet en HTTP**” waarin je:

- Voor elke uitprobeeropdracht een entry maakt met screenshots ter staving van wat je deed.
- Je antwoorden op de gestelde vragen neerschrijft.

Oplossingen van 'grotere' opdrachten (met veel code) bewaar je in een Webstorm project "**Labo 1**". Per opdracht maak je in dit project een aparte folder waarin je de bestanden (en subfolders) plaatst.

2 Labo

2.1 Chrome Developer Tools

We gaan er in deze cursus van uit dat je Chrome als browser gebruikt, al je oplossingen dienen op Chrome te werken. Hierbij is het handig om als ontwikkelaar gebruik te maken van de Chrome Developer Tools. Je vindt een overzicht van de mogelijkheden online[1]. Kijk beslist eens op die pagina wat de verschillende tabbladen te bieden hebben.

2.2 HyperText Transport Protocol (HTTP)

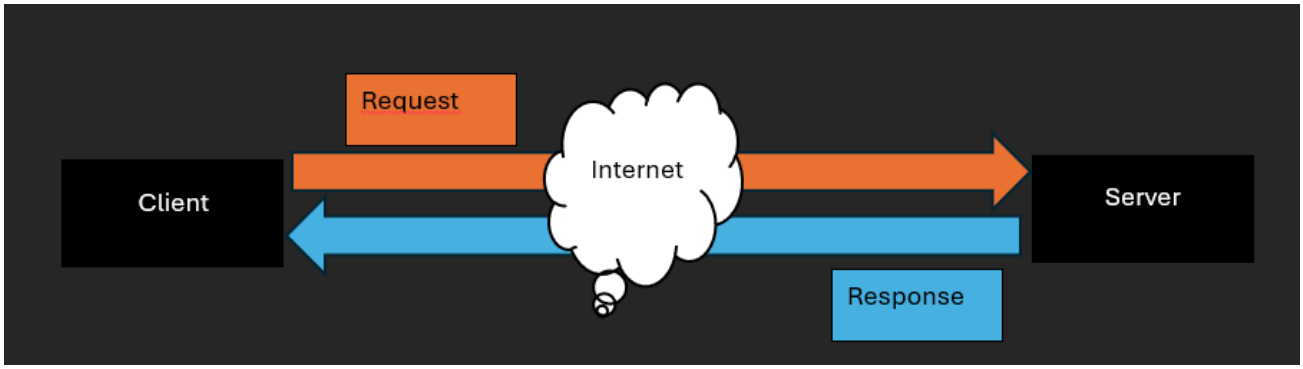
Telkens je met een browser naar een webpagina surft, wisselt de browser informatie uit met de webserver die de pagina moet aanleveren.

Het protocol waarmee deze beiden communiceren is HTTP, waarbij de browser de rol speelt van HTTP-client en de webserver de rol van HTTP-server. De communicatie tussen beiden bestaat uit een of meerdere request bericht(en) en (verstuurd door de browser) en met telkens (hopelijk) een bijbehorend response bericht van de server. Het protocol definieert hoe berichten moeten worden geformatteerd en verzonden, en hoe servers en browsers moeten reageren op verschillende verzoeken. Het protocol legt ook vast welke soorten berichten er mogelijk zijn en wat ze betekenen. De berichten zijn tekst gebaseerd, we zullen ze verderop bekijken.

2.2.1 Opdracht 1

Zoek op het internet naar afbeeldingen op basis van de zoektermen **http client server**. Teken hieronder een schematische voorstelling van de communicatie tussen een client en een server. Voor computers gebruik je rechthoeken, voor berichten pijltjes en in het midden teken je een mysterieus wolkje dat 'het internet' voorstelt. Zorg ervoor dat de volgende termen erop aangeduid staan: client, server, request, response, internet, tekst en geef met getallen de volgorde van de berichten aan.

Elk request dat de browser naar de server stuurt bevat een URL waarmee een resource geïdentificeerd wordt (bv. een document, een video, een afbeelding, ...) waarin de browser geïnteresseerd is. De respons wordt teruggestuurd op basis van het IP-adres van de client computer.



2.2.1.1 Voorbeeld

Een URL zou er zo kunnen uitzien

https://documents.example.com:8080/en/read?id=123&highlight=internet#section1

Die URL bestaat uit de (meestal niet verplichte) onderdelen:

[1] https://developer.chrome.com/devtools

- protocol: `https`
    - Dit is het protocol dat gebruikt moet worden om de resource op te vragen. Een protocol beschrijft de spelregels van de communicatie en de toegelaten berichten. Het verkeer op het internet gebeurt op basis van veel verschillende protocollen, bv. SMB voor fileshares, IMAP voor email, HTTP om documenten uit te wisselen, ... Je browser ondersteunt veel verschillende protocollen, maar bij het dagdagelijkse browsen is dit doorgaans HTTP.
- host: `example.com`
    - De host naam is doorgaans een domeinnaam, maar kan bv. ook een IP-adres zijn.
    - Je computer gebruikt het IP-adres van de server voor de eigenlijke communicatie, dit adres wordt opgezocht met een DNS lookup[2] van de hostname.
- subdomain: `documents`
    - Een subdomein is een techniek waarmee de beheerder van de domeinnaam, de inkomende berichten gericht naar bepaalde servers kan sturen
- port: `8080`
    - Een poortnummer identificeert voor welke service op de server het bericht bedoeld is. Een service is niks anders dan een programma dat op de server draait en luistert naar de inkomende trafiek voor een specifieke poort. Eenzelfde server kan immers vele rollen vervullen: fileserver voor network shares, een streaming server, een webserver, een databank server, etc.
- path: `en/read`
    - Dit identificeert een bepaalde resource op de server. Vaak wordt hier een logische indeling gebruikt die verband houdt met iets uit 'de realiteit', bv. `/opleidingen/handelswetenschappen-en-bedrijfskunde/apps-gamification` identificeert het document op de vives.be website met informatie over het keuzetraject “Apps & Gamification” uit de opleiding “Toegepaste Informatica binnen HWB”. Merk op dat bv. de opleiding zelf niet in de URL voorkomt.
- parameters
    - Parameters worden gebruikt om via de URL informatie mee te geven met het request (merk op dat er nog andere manieren zijn om dit te bewerkstelligen, zoals in de request body).
    - Uit de voorbeeld URL hierboven:
        - Parameter `id` heeft waarde `123`.

[2] DNS vertaalt een hostname naar een IP adres, e.g., zo weet je browser welk IP adres er achter `google.com` hangt.

        - Parameter `highlight` heeft waarde `internet`.

- fragment: `#section1`
    - Een fragment identificeert een onderdeel van de opgevraagde resource, bv. de sectie voor Hoofdstuk 7 in een webpagina met de tekst van een boek. Je browser zal na het inladen van deze pagina automatisch naar Hoofdstuk 7 scrollen.

Extra info https:

Staat voor de **H**yperText **T**ransfer **P**rotocol **S**ecure. Een mondvol! Gelukkig is er dus die afkorting om te gebruiken.

HTTPS is het veiligheidsprotocol waarmee je een versleutelde verbinding tussen een webserver en een browser opzet. Met het protocol van HTTPS-security kan jouw computer via een beveiligde verbinding connecteren met een website. Via encryptie, de eigenlijke versleuteling van de gegevens, kunnen internetcriminelen de informatie niet in handen krijgen.

Bij een SSL verbinding worden berichten beveiligd verzonden. Eenvoudig gezegd stuurt de server jou een doosje met een hangslotje. Je kunt een berichtje in dit doosje stoppen en het op slot doen. Je hebt echter geen sleutel om het doosje open te maken. Dit kan alleen de server.

SSL zorgt er dus voor dat verkeer beveiligd verzonden kan worden en niet onderschept kan worden. Maar hoe weet je of het doosje dat je opgestuurd krijgt van de juiste afzender afkomstig is? Wie zegt dat een hacker je geen doosje opstuurt?

Vandaar dat bij SSL geregistreerd wordt welke doosjes van welke website zijn. Dit doet de uitgever van het SSL certificaat. Dat is ook de reden dat je betaalt voor een SSL certificaat. De verstrekker van het SSL certificaat houdt daarvoor een lijst bij met doosjes en wie de doosjes mogen gebruiken. Jouw bezoeker weet zo zeker dat de doosjes die hij opgestuurd krijgt van jou afkomstig zijn en weet zeker dat de informatie die hij verstuurt bij jou aankomt.

2.2.2 Opdracht 2

Duid de verschillende onderdelen van de volgende URL:

https://www.bol.com/nl/p/hoe-werkt-dat-nou/9200000057347012/?country=BE&suggestionType=browse#product_alternatives

Protocol: https://
Domein: www.bol.com
Top Level Domain: .com
Pad: /nl/p/hoe-werkt-dat-nou/9200000057347012/
Query string: ?country=BE&suggestionType=browse
Fragment: #product_alternatives

Elk bericht dat de browser en server uitwisselen (zie bv. je afbeelding uit Opdracht 1) is een tekst

waarin we twee delen onderscheiden:

Body

Dit is de eigenlijke informatie.

- Een request body kan de data bevatten die de gebruiker in een invulformulier invulde.
- Een response body kan de tekst van de gevraagde webpagina bevatten

Headers

Dit is meta informatie (i.e. informatie over de informatie).

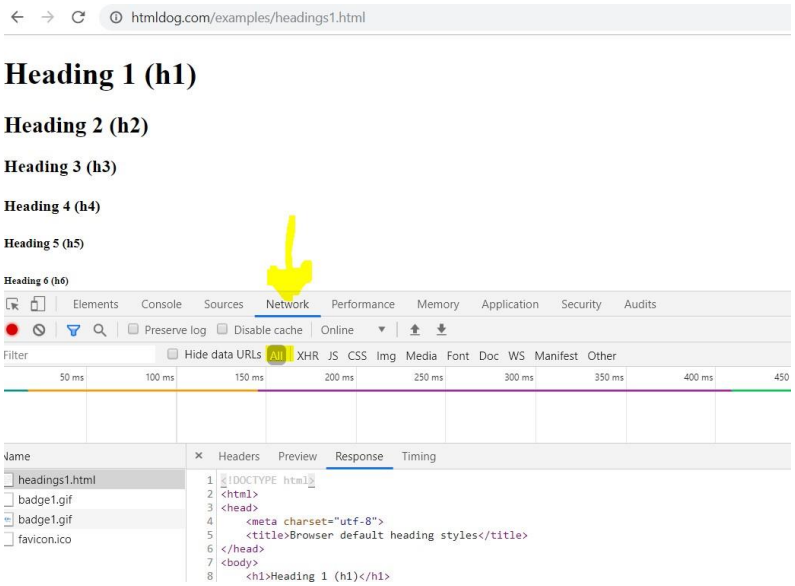
- In een request header kan staan in welk formaat de browser de resource kan ontvangen.
- In een response header kan het tijdstip staan waarop de webpagina het laatst werd gewijzigd.

Naast de url zal een request dus nog veel meer informatie (lees: tekst) bevatten. De server ontvangt deze request en moet op basis van deze informatie beslissen wat zijn response zal zijn. Heel vaak bevat de response de inhoud (lees: tekst) van een webpagina, e.g., een HTML document.

De browser ontvangt de response van de server en moet de bijbehorende informatie dan op een of andere manier presenteren aan de gebruiker.

2.2.3 Opdracht 3

Open de Chrome developer tools (F12), ga naar het Network tabblad en surf naar: https://www.htmldog.com/examples/headings1.html.



Figuur 1 screenshot htmldog.com en Chrome Developer Tools

Je kunt nu op het originele request klikken en de headers bekijken, alsook het response inspecteren. Kijk beslist eens naar de status code en response body. De response body bevat in dit geval de code voor een HTML document dat de inhoud en structuur van de opgevraagde `headings1.html` resource beschrijft.

Zo'n HTML document bevat naast de eigenlijke inhoud (bv. de tekst van een krantenartikel) heel vaak ook verwijzingen naar andere documenten:

- Bestanden met layout informatie (CSS bestanden).
- Bestanden met uitvoerbare programma code (javascript bestanden).
- Afbeeldingen.

- Andere resources, e.g., links naar andere webpagina's).

2.2.4 Opdracht 4

In het voorgaande voorbeeld heeft de browser niet alleen een request voor de `headings1.html` resource verstuurd. Welke resources heeft je browser nog meer opgevraagd? Hoe zie je dit?

Eronder zie je ook bsa.js, arcticFox, …

Kun je in het HTML document in het response body terugvinden waarom net die resources werden opgevraagd? Merk op dat browser plugins zoals anti-virus programma's en ad-blockers sommige requests kunnen tegenhouden en ook eigen requests kunnen versturen!

Je kan de HTML document niet zien waarom specifieke resources zijn opgevraagd. Komt door dat de opvraging vaak in code of scripts zit.

2.2.5 Opdracht 5

Surf naar https://www.vives.be. Welke andere soorten resources worden opgevraagd door het inladen van deze pagina? Xhr, fetch, gif, …

Werden alle requests naar dezelfde server verstuurd? Om dit te zien kun je best een 'domain' kolom toevoegen aan de tabel. Rechtsklik hiervoor op de hoofding van de tabel en vink 'Domain' aan.

Nee.

Rechts naast elk request zie je de timing informatie die weergeeft wanneer het request verstuurd werd en wanneer het response werd ontvangen.

Het opvragen van 2 resources van een webpagina kan normaliter onafhankelijk van elkaar gebeuren, de browser kan dus een pagina sneller kunnen inladen door een volgend request te versturen, nog voor het response op een vorig request werd ontvangen.

Hoe kun je dit uit de timing informatie afleiden?

Door te kijken wanneer de verzoeken zijn verstuurd en voltooid.

Lijkt het erop dat het aantal gelijktijdige ‘onafgewerkte’ requests beperkt is? (onafgewerkt, in de zin dat het request verstuurd is maar nog geen response werd ontvangen).

Dit komt door dat er limiet is aan hoeveel verzoeken tegelijk kunnen lopen. Sommige verzoeken starten pas nadat andere zijn voltooid.

2.2.6 Opdracht 6

Open het Netwerk tabblad van de Chrome developer tools, surf naar je favoriete webmail client, e.g., gmail, en log in.

Wacht tot 'alles' ingeladen is en zorg ervoor dat je niet met de muis over de eigenlijk webpagina gaat (beperk de bewegingen tot het network tabblad). Afhankelijk van welke webmail provider je

gebruikt, zul je zien dat er na verloop van tijd requests bijkomen ook al verandert er niks aan de webpagina.

Waarvoor zouden die 'spontane' requests dienen?

Webmail checkt of ik nieuwe mails hebt gekregen of niet. Ook zorgen ze ervoor dat de verbinding actief blijft zodat ik niet automatisch uitgelogd wordt.

2.3 HTTP response status codes

Elk response dat de server terugstuurt bevat een status code die aangeeft om wat voor soort response het gaat, e.g., *alles ok*, *server overbelast*, *toegang tot resource verboden*.

2.3.1 Opdracht 7

Surf naar https://www.vives.be/dezepaginabestaatniet.hml. Merk op dat er wel degelijk een response teruggestuurd wordt alhoewel de pagina niet bestaat. Hoe komt dit?

Wat betekent de status code 404 in de response header?

Dit betekent dat de website of pagina die ik probeer te bezoeken niet bestaat op de server. 404 wilt tonen dat de server, deze pagina niet kan vinden.

2.3.2 Opdracht 8

Probeer nu eens https://www.ditdomeinbestaatnogniet.be/blabla.html. Wat is het verschil met de vorige opdracht?

Dit betekent dat de domein niet bestaat. M.a.w. DNS-adres kan niet terug gevonden worden op de server.

2.3.3 Opdracht 9

Zoek onderstaande HTTP status codes op (bv. wikipedia) die in een response kunnen voorkomen en schrijf hun betekenis op. Omcirkel deze die je al eens bent tegengekomen bij het surfen. Schrijf ook een gepaste hoofding boven de kolommen.

Succes	Redirection	Client error	Server error
200 -> OK	301 -> bron is perm. Verplaatst naar nieuwe url.	400 -> server kon de request niet verwerken door bv. Syntaxfout van client	500 -> Server heeft een onverwachte fout ervaren
204 -> Request is succesvol maar geen inhoud	302 -> bron tijdelijk verplaats naar andere url	401 -> Authenticatie is vereist om bron te bekijken	503 -> Server tijdelijk niet beschikbaar.
	303 -> client wordt doorgestuurd naar een andere url	404 -> bron niet gevonden op de server	

2.4 HTTP request methods

Elke client request is van een bepaalde soort die met de request method wordt aangeduid.

2.4.1 Opdracht 10

Zoek op het internet welke HTTP request methods er bestaan en schrijf ze neer. Waarvoor dienen de vaak gebruikte GET en POST methods?

Get: Gebruikt om informatie van de server op te halen zoals een webpagina. Het verandert niks op de server.

Post: Wordt gebruikt om gegevens naar de server te sturen bv. Bij het invullen van een formulier.

Waar in een request staat aangegeven om welke request method het gaat, en hoe vind je dit terug in de Chrome developer tools (zie uitleg bij Opdracht 3)?

Het staat in het begin van het verzoek dat je browser naar de server stuurt.
Je opent de website en drukt op toets f12. Daarna ga je naar tabblad Network. Nadien klik je op een verzoek in de lijst en je ziet dan de details, ook de request method.

Als je een url in de adresbalk van je browser typt en op enter drukt, wat voor request method gebruikt de browser dan om die resource op te vragen bij de server?

De browser gebruikt een Get request om de webpagina op te vragen.

Als je in een webpagina op een gewone hyperlink klikt, welke request method wordt er dan gebruikt?

Dan gebruik de browser weer Get request om de gelinkte pagina op te halen.

Stel, iemand schrijft een webapplicatie om producten te beheren en realiseert de "wis productgegevens" functionaliteit door middel van een gewone hyperlink met 'wis' opschrift. Om een

product te wissen moet een gebruiker dus gewoon op de 'wis' link klikken bij dit product. Op een bepaald moment komt bv. de google-bot langs die (programmatorisch) alle links uitprobeert om te zien wat dit oplevert aan nieuwe pagina's om te indexeren. Of stel dat de browser een accelerator plugin bevat die proactief gelinkte pagina's inlaadt zodat de gebruiker niet hoeft te wachten bij het klikken op een link.

Wat zou er dan gebeuren met de productgegevens?

Productgegevens zouden per ongeluk gewist kunnen

worden zonder dat iemand dit bewust doet.

Dit is trouwens geen hypothetisch geval, zie bv.

- https://thedailywtf.com/articles/The_Spider_of_Doom
- https://thedailywtf.com/articles/WellIntentioned-Destruction
- https://betanews.com/2005/05/06/google-web-accelerator-draws-concern/

2.5 Caching door de browser

Veel resources wijzigen niet zo vaak, denk bv. aan de afbeelding met het bedrijfslogo die op bijna alle pagina's van hun website voorkomt. Een browser kan veel trafiek vermijden door resources lokaal bij te houden voor toekomstig gebruik i.p.v. telkens dezelfde resource aan de server te vragen. Deze techniek heet ‘caching’ en is voor iedereen een goeie zaak: de gebruiker krijgt sneller de pagina te zien en spaart bandbreedte, de beheerder van de server spaart bandbreedte, geheugen en CPU cycles op de server.

2.5.1 Opdracht 11

Ga naar het network tabblad van de Chrome developer tools en surf naar https://www.vives.be. Bekijk de vele requests die het inladen van die ene pagina heeft veroorzaakt. Hoeveel requests waren er in totaal? 36

2.5.2 Opdracht 12

Zorg dat de Chrome developer tools actief zijn. Open dan de vives.be pagina, rechtsklik op de refresh knop en kies '*Empty cache and hard reload*'.

Hiermee dwingen we de browser om bij het herladen van de pagina z'n cache te negeren en alle resources bij de server op te vragen.

Hoeveel kilobytes of megabytes aan data werd er verstuurd om alle nodige resources in te laden?

24.1 Kb

Hoe lang duurde het vooraleer alle resources van de pagina waren ingeladen?

Klik nu gewoon op de refresh knop.

Kijk nogmaals hoeveel data er werd verstuurd. Waarom is dit zoveel minder? Laadde de pagina sneller?

24 Kb. En de pagina laadde trager.

Waar kun je zien welke documenten daadwerkelijk verstuurd werden en welke niet?
Aan de code 200 -> betekent is OK.
Waar vindt de browser dan de inhoud van de documenten die niet bij de server werden opgevraagd? Je kan zien aan de code 304 -> bestand is up to date in de cache.
Hoe weet de browser welke documenten best opgevraagd (moeten) worden en welke niet? M.a.w. hoe lang mag de browser een bepaalde resource als 'vers' te beschouwen? (Hint: kijk eens naar de response

headers). Cache-Control geeft aan hoelang een bestand als vers kan worden beschouwd. Expires toont wanneer dat die niet meer als vers wordt gezien.

2.6 Web analytics en tracking

Web analytics is de techniek om informatie over de websurfers te verzamelen die je site bezoeken, om deze informatie vervolgens te analyseren om bepaalde aspecten te optimaliseren (bv. meer verkopen). Deze statistieken worden verzameld door Javascript programma's die met de pagina worden meegeladen, de gewenste data capteren en daarna deze data naar een server doorsturen om opgeslagen te worden voor verdere analyse.

Men houdt bv. bij over welke browser het gaat, welke versie, welke schermgrootte, via welke pagina de bezoekers op de website arriveerden, welk pad doorheen de website hebben ze gevolgd, hoeveel

bezoekers zijn terugkerende bezoekers, etc.).

Op basis van de verzamelde statistieken kunnen dan onderbouwde beslissingen worden genomen om het gekozen aspect (bv. meer verkopen) te optimaliseren. Enkele bekende aanbieders zijn Google Analytics en Statcounter.

- https://www.google.com/analytics
- https://statcounter.com/

Deze aanbieders verzamelen statistieken over miljarden webpagina bezoekjes via de websites van hun hun klanten.

- Statistieken over de bezochte website: populairste landing pages, vaak gevolgde paden doorheen hun website, land van bezoeker, ...
- Statistieken over de browsermogelijkheden van de bezoeker: browser, versie, schermgrootte, ...

Elke klant krijgt natuurlijk enkel gedetailleerde informatie over hun eigen website bezoekers te zien.

Men stelt echter vaak ook globale statistieken ter beschikking over de browsermogelijkheden van bezoekers, bv. eens per maand. Deze zijn handig om te beslissen welke mogelijkheden je kunt gebruiken. Bv. is het nog nuttig om extra moeite te steken in work-arounds voor een bepaalde oude browser versie, op welke minimale schermbreedte je moet mikken met je website, ...

Een voorbeeld vind je hier: https://gs.statcounter.com. Let er wel op dat het beter is om je eigen statistieken te tracken, want jouw doelpubliek is niet noodzakelijk hetzelfde als de gemiddelde

websurfer.

2.6.1 Opdracht 13

Installeer in Chrome de *Ghostery* extension. Bezoek nu de volgende webpagina's:

- https://www.nieuwsblad.be/
- https://www.cnn.com
- https://www.vives.be
- https://www.vrt.be/vrtnws/nl/

Schrijf bij elke webpagina hoeveel tracking scripts door Ghostery worden ontdekt. Welke soorten tracking scripts je bent tegengekomen (beacon, advertising, analytics, etc.)?

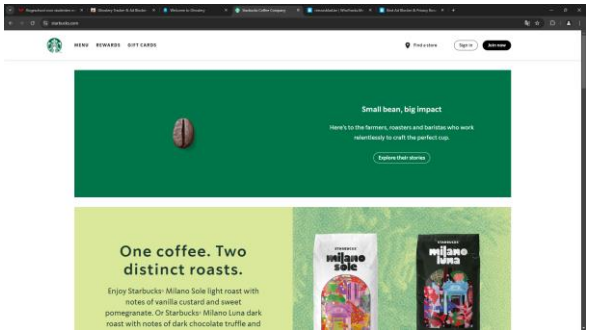
DoubleClick, Gemius, cXense, Didomi, aswpsdkus, shared.mediahuis, …

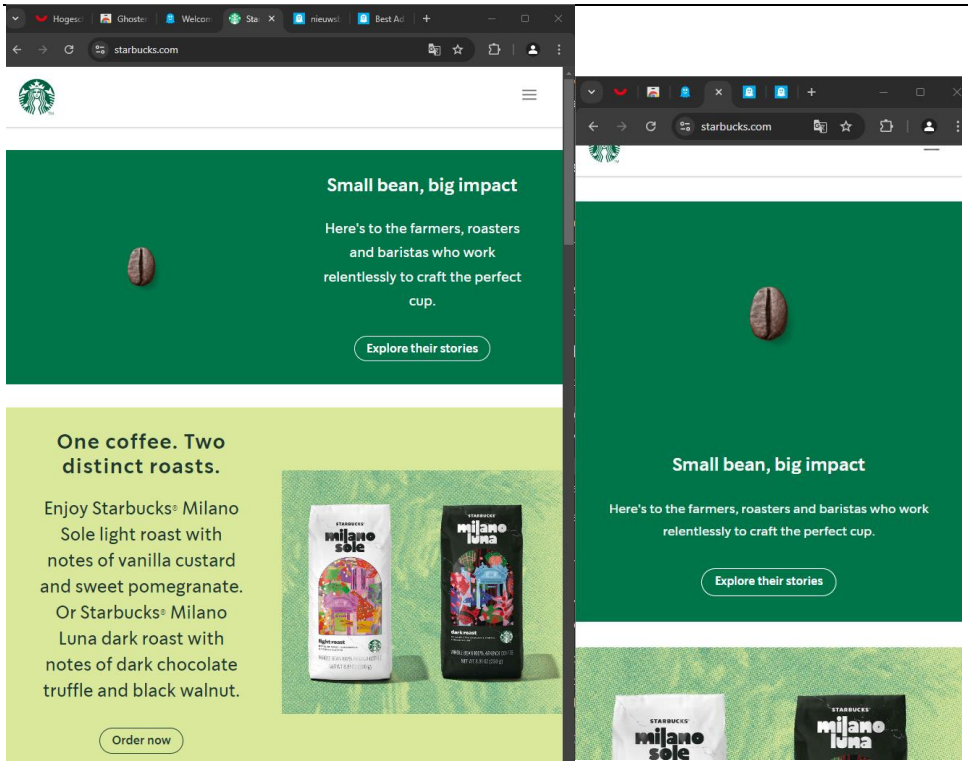
2.6.2 Opdracht 14

Zorg ervoor dat je browser venster gemaximaliseerd is en surf naar https://www.starbucks.com. Maak nu je browser venster gaandeweg minder breed en let erop hoe de inhoud van de webpagina niet enkel opschuift maar ook daadwerkelijk verandert!

Deze techniek heet **responsive design** en komt later nog aan bod in de labo's.

Doorgaans verschuiven elementen en/of verkleinen ze, maar op sommige breedtes is er echter een abrupte overgang naar een andere layout. Neem bij elke abrupte overgang een screenshot (bv. met het *Snipping Tool* in Windows). Hoeveel verschillende layouts tel je? 3




starbucks.com
Small bean, big impact
Here's to the farmers, roasters and baristas who work relentlessly to craft the perfect cup.
Explore their stories
One coffee. Two distinct roasts.
Enjoy Starbucks® Milano Sole light roast with notes of vanilla custard and sweet pomegranate. Or Starbucks® Milano Luna dark roast with notes of dark chocolate truffle and black walnut.
Order now
milano sole
milano luna
starbucks.com
Small bean, big impact
Here's to the farmers, roasters and baristas who work relentlessly to craft the perfect cup.
Explore their stories
milano sole
milano luna