Weekopgaven week 1

Lasse Benninga & Andre Nanninga

Opgave 1

***1. Waarom is het van belang dat er een protocol is dat door zowel de servers als de***

***clients wordt ondersteund?***

Omdat server en client elkaar anders niet kunnen "verstaan", het protocol zorgt ervoor dat informatie niet verkeerd wordt geiinterpreteerd of verloren gaat.

***2. Het http is nu in versie 1.1. Wat zijn de belangrijkste verschillen tussen deze versie***

***en versie 1.0?***

Http 1.1 vereist toevoeging van **Host Headers** in de request, om onderscheid te maken tussen hosts met hetzelfde IP adres.

Daarnaast maakt HTTP 1.1 het mogelijk om een **persistente connectie** te hebben: het is mogelijk om meer dan één request en respons tegelijkertijd te hebben op dezelfde HTTP connectie. In HTTP 1.0 moest je een nieuwe connectie openen voor elke nieuwe request/respons, die vervolgens werd gesloten.

Vervolgens heeft HTTP 1.1 **Options methods** toegevoegd: daarmee kan een HTTP client de mogelijkheden van de server opvragen.

Ook zijn er in HTTP 1.1 verbeteringen toegevoegd voor **Caching**, bijvoorbeeld entity tags die zorgen voor het categoriseren van resources.

Een andere toevoeging in HTTP 1.1 is **chunked encoding**, de content wordt in delen (chunks) verstuurd en kan dus al content weergeven zonder dat alle data is opgehaald.

***3. Installeer een plugin in je browser zodat je de http-headers kunt zien (bijvoorbeeld***

***live http-headers voor Chrome of FireFox, ieHttpHeaders voor Internet Explorer)***

***en ga vervolgens met je browser naar Blackboard. Beschrijf en verklaar een aantal***

***van de headers die langskomen.***

Eerste GET Request Header van BlackBoard

GET /webapps/blackboard/execute/announcement?method=search&context=course\_entry&course\_id=\_12403\_1&handle=announcements\_entry&mode=view HTTP/1.1

**De url van de GET request**

Host: blackboard.hanze.nl:443

**De serverhost (ip adres en poort)**

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

**Media types die de host accepteerd als response**

Accept-Encoding: gzip, deflate, sdch

**Geaccepteerde encoding als response**

Accept-Language: nl-NL,nl;q=0.8,en-US;q=0.6,en;q=0.4

**Geaccepteerde taal voor de response**

Referer: **Het adres (URI) waar de request vandaan kwam**

Upgrade-Insecure-Requests: 1

**Host geeft aan dat Content Security Policy wordt ondersteund (voor beveiliging)**

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36

**Informatie over de client en wat voor browserkit er wordt gebruikt.**

Een POST request van BlackBoard

**Nieuwe velden:**

Cookie: GMAIL\_AT=AF6bupPB00mKg9-oo7DaTcvwJ4qPffj7hw...

**Een cookie (informatie) die naar de client wordt gestuurd om de staat van informatie die de client heeft over de website op te slaan. (Zodat een sessie niet verloren gaat)**

Origin: [https://mail.google.com](https://mail.google.com/)

**Begint een request voor cross-origin resource sharing**

HTTP/1.1 304 Not Modified

**HTTP versie**

Cache-Control: public

**Geeft aan of de server het object (request/response) mag cachen.**

Cneonction: close

**Verbindingsstatus tussen host en client**

Date: Tue, 08 Sep 2015 11:29:28 GMT

**Datum en tijd van de server**

ETag: W/"3012-1414751840000"

**Entity Tag voor het object (Request). Zodat client eventueel de lokale kopie van de respons kan gebruiken als de request al eerder is gedaan.**

Expires: Thu, 08 Sep 2016 11:29:28 GMT

**TIjd wanneer de Entity Tag verstreikt.**

Server: Apache-Coyote/1.1

**Naam van de server**

Vary: Accept-Encoding,User-Agent

**Geeft aan de proxys door of een gecachede response kan worden doorgegeven of er een nieuwe response moet worden aangevraagd van de server.**

***4. Wat is de rol van al die cookies die in de headers heen en weer gestuurd worden?***

Cookies zijn een string die tussen de host en client worden verwisseld. Ze bevatten informatie over de client (gebruiker) en worden lokaal bij de client opgeslagen. Zo kan de host bepaalde (op de specifieke gebruiker afgestemde) informatie weergeven aan de client doordat deze zijn cookie aanbied aan de server.

***5. Waarom is het logisch dat Apache werkt als een deamon?***

Er is geen interactie tussen de gebruiker en Apache, omdat Apache communiceert met services en achtergrondprocessen, dit is de reden dat Apache een deamon is.

***6. In de tekst is sprake van MPMs. Wat zijn dat en wat is hun functie?***

MPM staat voor Multi Processing Module. Dit is een abstractie van de server basis logica naar een module in Apache 2.0, waardoor Apache vloeiender loopt op Windows besturingssystemen (er is een specifieke module voor Windows).

***7. Wat zijn de voordelen van de modulaire opzet van Apache?***

Door de modulaire opzet kan de gewenste functionaliteit voor de webserver gekozen worden. Dit maakt de server kleiner, scheelt geheugenverbruik en maakt de kans op fouten door misconfiguratie kleiner. Daarnaast kunnen modules simpelweg geiinstalleerd worden en hoeft apache niet compleet herschreven worden wanneer een nieuwe opzet wordt gemaakt.

***8. Wat is het centrale punt van dit artikel?***

Dat software in op alle fronten in de maatschappij aanwezig is en dat Andreessen (de schrijver) verwacht dat het aantal softwarebedrijven alleen maar gaat groeien. De bedrijven die groot worden zullen erg sterk in de markt staan. Mensen die op dit moment geen vruchten kunnen plukken van deze modernisering, of werkloos raken hierdoor kunnen alleen maar door onderwijs hier verandering in brengen. Tenslotte moeten de nieuwe bedrijven hun waarde laten zien.

***9. Andreessen beschrijft twee processen die ertoe leiden dat er in het komende decennium veel bedrijven verstoord (disrupted) zullen worden door software. Welke twee processen zijn dat?***

-Software kan eindelijk op grote schaal ingezet worden door de groei van apparaten per hoofd van gebuikers.

- Steeds meer grote industrieen draaien op software en leveren hun diensten via het internet aan. Daarnaast zijn de startupkosten voor een internetbedrijf erg laag en hoeft er niet veel geïnvesteerd worden in personeel en infrastructuur.

***10. Geef zelf drie voorbeelden van bedrijven of activiteiten die toenemende mate software-gedreven worden.***

- Informatie opvragen/afspraken maken van de overheid/gemeente.

- Afspraken maken met dienstverleners (kappers, monteurs).

- Bankieren (internetgedreven).

Opgave 2

Zie uitwerking

Opgave 3

***2. Wat wil het zeggen dat git een gedistribueerd systeem is?***

In tegenstelling tot een gecentraliseerd systeem is er een database server waarvan de client de gegevens haalt. In een gedistribueerd systeem worden de databasegegevens opgeslagen bij alle clients, daardoor kan er doorgewerkt worden als er een “server” uitvalt en blijven de database gegevens bestaan.

***3. Wat is een file based delta storage system en waarin verschilt dit van de manier waarop git werk?***

In FBDSS wordt er gewerkt met Logs die de wijzigingen steeds aanpassen. Git werkt met het hashen van de waarde van de content en controleert deze wanneer de content wordt opgevraagd. In Git wordt ook gewerkt met een manifest waar de pointers naar de content gezet worden. Delta compressie wordt alleen gebruikt wanneer er met het netwerk contact wordt gemaakt, bij Git Pull/Push. Niet bij lokale wijzigingen.

***4. Wat is de rol van de het index-bestand in de .git-directory?***

Het index bestand stelt een virtuele versie van de Working State Tree voor met paths en objectnamen. Deze bevat een lijst van de path-namen met bijbehorende permissies en de SHA1 (codering) van het object. Alleen de huidige inhoud van de index-file wordt gecommit, niet wat er in je working directory staat.

***5. Bekijk de onderstaande situatie***

***a. Wat betekenen die hexadecimale getallen in de verschillende groene blokjes?***

Versies van commits, dit zijn de hexadecimale hashes voor de versiecodes.

***b. Is een fast-foward merge op dit moment mogelijk bij git merge i18n? Beargumenteer je antwoord.***

Ja, dit is mogelijk. Er komt dan een nieuwe Master versie die ontstaat uit een merge tussen de twee branches.

***c. Is vervolgens een fast-forward merge mogelijk bij git merge lem? Beargumenteer je antwoord.***

Nee door de voorgaande merge bestaat de fastforward merge die eerst mogelijk was niet meer. (niet zeker?)

***6. Maak een directory aan op je locale file system, zet daar wat bestanden in en voeg die toe aan een git-repository. Wijzig vervolgens één van deze bestanden en leg uit wat je ziet wanneer je git status doet. Doe nu een git add op het bestand dat je net hebt gewijzigd en opnieuw een git status. Leg het verschil tussen deze twee uit.***

Na de add komt er een overzicht van de bestanden die kunnen worden gecommit en toegevoegd aan de server repository.

***7. Haal de bestanden die je in de vorige stap hebt gemaakt weg uit de directory doe vervolgens git commit. Wat valt je op aan deze werkmethode?***

De bestanden worden niet van de server verwijderd omdat de wijziging niet is ge-add. Je kan dus niet perongeluk iets verwijderen en dat committen.

***8. Maak ergens anders op je file system een clone van de repository uit de vorige opgaven en vergelijk de inhoud van deze clone met het origineel. Wat is het opvallende verschil?***

De clone bevat niet de commit informatie die het origineel heeft. De kloon is identiek aan wat op de server staat.

Opgave 4

Zie uitwerking

Opgave 5

***1. Waarom zijn sessie noodzakelijk in een webapplicatie die bijvoorbeeld de mogelijkheid van een winkelwagen bevat? Gebruik de statelessness HTTP in het antwoord.***

Sessies zijn nodig om content te kunnen serveren die specifiek voor de gebruik zijn. Bijvoorbeeld de winkelwagen bevat de producten die gebruiker heeft uitgekozen, een GET request op /winkelwagen moet alleen de winkelwagen van de gebruiker terug geven. In stateless HTTP zou dit echter altijd het zelfde resultaat geven. Voor elke gebruiker zou /winkelwagen dus het zelfde er uit zien. Dit is natuurlijk niet handig wanneer Piet schoenen wil komen maar Marte een broek.

***2. Je kunt deze statelessness als een nadeel, maar wat zijn de mogelijke voordelen?***

Een groot voordeel is is de zekerheid dat wanneer twee keer eenzelfde request wordt gedaan dat ook twee keer dezelfde response terug komt. Dit is makkelijker om te testen omdat het resultaat voorspelbaar is.

De gebruiker kan ook makkelijk elke webpagina met vrienden delen en zeker zijn dat zij dezelfde pagina zien.

***3. Beantwoord de volgende vragen over de toegewezen programmeeromgeving:***

***a) Waar worden de sessiegegevens bewaard?***

Sessiegegevens worden serverside opgeslagen in een Flask.Session object. Dit is een key-map object waarin data specifiek voor een gebruiker wordt bijgehouden. Op basis van een cookie met een key wordt de sessie met de gebruiker gekoppelt.

***b) Hoe wordt een sessie gestart?***

Allereest moet een secret\_key wordt gezet op volgende wijze:   
app.secret\_key = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'  
de secret\_key wordt gebuikt voor de key in de cookie.

Wanneer een gebruiker de website bezoekt wordt automatisch de cookie met de key gezet waarna bij elke request de session kan worden geupdate, bijvoorbeeld:  
session['username'] = request.form['username']

***c) Hoe wordt een sessie beëindigd?***

De sessie kan niet serverside worden beëindigd omdat deze gebaseerd is op een cookie. Pas wanneer de cookie wordt overschreven of verwijderd is de sessie 'echt' beëindigd.

Wel kan op de server de methode session.clear() worden aangeroepen waarbij alle data die is opgeslagen in een sessie wordt verwijdert.

***d) Op welke manier kunnen client-sided gegevens server side in een sessie opgenomen worden?***

De simpelste manier is via een formulier:  
session['username'] = request.form['username']

***4. Wat is een cookie in de context van webapplicaties?***

Een cookie is clientside key,value data voor een website. Deze dat wordt opgeslagen in de webbrowser van gebruiker en wordt meegestuurd bij een request.

Bijvoorbeeld: een gebruiker komt voor het eerst op een website en er wordt een special bericht getoond 'Maak kennis met onze website'. Om te voorkomen dat het bericht elke keer getoond wordt wordt een cookie gezet: 'new\_visitor': false. Bij elke volgende request ziet de server aan deze cookie dat de gebruiker geen nieuwe gebruiker is en dus niet het bericht hoeft te zien.

***5. Waar worden cookies opgeslagen?***

In de webbrowser van de gebruiker.

***6. Maak een pagina waarin een bezoeker via een invoerveld en een knop een tekst aan de sessie toevoegt. Laat de action parameter in de form tag verwijzen naar dezelfde pagina. Ook wordt op dezelfde pagina een lijst getoond van wat er in de sessie staat.***

Zie uitwerking.

***7. Zet de plugin die in opgave 1.3 is geïnstalleerd aan en beschrijf wat er specifiek met betrekking tot sessiemanagement wordt verstuurd.***

Zoals eerder beschreven wordt een sessie serverside bijgehouden en gekoppeld aan gebruiker op basis van een key. De cookie 'session' bevat deze key en wordt naar de server gestuurd. Op basis van deze key wordt koppeling met sessie en gebruiker gelegd.

De cookie header ziet er als volgt uit:

Cookie:session=.eJwdi0sKgCAUAO\_y1i40rbTLhL3PJlJRW0V3T1oMAwPzALYqe88nJ9hgIZxXpOCRtIkTOhkczmoMWqI1QZCi9QwKOrc-jl8K7sY1xYtHKZkLvB\_oeBwC.CNWRbw.W5Hlz7vmzovPMW4VP\_Dss3cJMws

***8. Welke rol spelen cookies in de implementatie van sessiemanagement?***

Zie antwoord op vraag 3 en 7.

***9. Het is belangrijk om doel (sessiemanagement) en middel (cookies als transportmechanisme) in deze context uit elkaar te houden. Zorg ervoor dat cookies in de browser niet meer geaccepteerd worden en verifieer dat het sessiemanagement in vraag h) het niet meer doet.***

De cookie session wordt niet langer gezet en verzonden en zodoende kan geen koppeling gelegd worden tussen een sessie en gebruiker. Het sessie object kan daarom niet gevuld worden.

***10. Is er een manier om sessiemanagement mogelijk te maken zonder cookies? Zoek uit hoe dat werk en pas het script van vraag h) aan. Verifieer dat het nu wel werkt, ongeacht of je cookies wel of niet aan hebt staan in je browser.***

Zie uitwerking.