Internet of Things – Gateway

Week 3-ev.

Auteur: Jorg Visch

Versie: 2.5

Datum: 30-aug-16

1 Inhoud

2 Inleiding

In de vorige lessen heeft het weerstation iedere keer zelf verbinding met internet om geregistreerde data op te sturen, bijv. naar een web service. Het is niet altijd mogelijk dat alle apparaten direct met internet verbinding kunnen of mogen maken. Om dit te omzeilen zou je een gateway kunnen inzetten die als een poort kan dienen naar buiten toe.

In deze weken ga je nog een keer een weerstation maken. In eerste instantie functioneert de Arduino als web server, maar is ook configureerbaar via die web server. Daarna gaat de Arduino data posten via een gateway naar een online storage.

3 Bronnen

Bronnen Internet HTTP protocols:

- http://www.w3.org/Protocols/
- RFC7230
  http://tools.ietf.org/html/rfc7230 (begindocument protocolbeschrijving van de berichten)
- Extra:
  - http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177
  - http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

4 Opdrachten

Lees eerst de hele opdrachten door, zodat je weet wat er van je verwacht wordt. Alle opdrachten in dit document moet je inleveren en zal beoordeeld worden (BP1 zie OWE-beschrijving op insite).

4.1 Opdracht 1: Parseren request-line

Om requests goed te kunnen verwerken is het noodzakelijk om de request-line[1] te parseren. Maak een methode die van een binnenkomend http-bericht de request line inleest, splitst in de drie element en dat teruggeeft. Een paar tips/suggesties:

- Lees de karakters één voor één in en als je een spatie tegenkomt, ga je over naar het inlezen van het volgende deel van de request line.
- Het inlezen van de request line is natuurlijk het inkomende bericht inlezen totdat je de eerste CRLF[2] tegen komt.
- Je kunt een struct gebruiken als returnwaarde van die methode waar de drie onderdelen van de request line in zitten. Of je kunt gebruik maken van pointer-argumenten om de waardes terug te geven. Let wel op dat op één of andere manier af te leiden moet zijn of het inlezen goed ging of niet.

4.2 Opdracht 2: GET en POST op de Arduino

Implementeer de 'Groet'-webservice op Arduino. In Week 2 heb je in de tutorial een oefening gedaan (in Visual Studio) en een webservice gemaakt. In deze opdracht moet je zelf de 'Groet'-webservice op de Arduino implementeren.

4.2.1 Laat de webservice een groet teruggeven (m.a.w. GET 'Groet')

Maak eerst de GET-implementatie, de definitie staat hieronder.

GET 'groet'	
Request om een vriendelijke groet terug te geven.	
Request	`GET http://10.0.0.45/groet/Arduinootje HTTP/1.1` `Host: 10.0.0.45`

Response	`HTTP/1.1 200 OK`
	`Content-Type: application/json; charset=utf-8`
	`Content-length: 19`
	`Connection: close`

[1] http://tools.ietf.org/html/rfc7230#section-3.1.1
[2] http://tools.ietf.org/html/rfc5234#appendix-B.1

	`"Hallo Arduinootje"`

4.2.2 Maak het mogelijk om de groet aan te passen (m.a.w. POST 'groet')

Met een POST naar de 'Groet'-webservice moet de groet in te stellen zijn. Hierboven staat nog een 'hard-coded' response '*Hallo <naam>*', maar middels een POST naar de webservice moet je de tekst 'Hallo' kunnen wijzigen. Hieronder de definitie.

POST 'groet'	
Request om de groet aan te passen	

Request	`POST http://10.0.0.45/groet HTTP/1.1`
	`Host: 10.0.0.45`
	`Content-Type: application/json; charset=utf-8`
	`Content-Length: 5`
	
	`"Hoi"`

Response	`HTTP/1.1 200 OK` `Connection: close`

4.3 Opdracht 3: Weerstation als webserver

4.3.1 Introductie

Maak een weerstation dat als webserver te benaderen is. De webserver moet zich als een echte webserver gedragen. Dat betekent dat als er verkeerde requests gedaan worden er een juiste response terugkomt (header met response code), zodat clients weer goed kunnen reageren. Om dit voor elkaar te krijgen moet je dus het http-protocol kennen. De volgende bronnen kun je gebruiken om het HTTP protocol verder te leren te kennen en gebruiken:

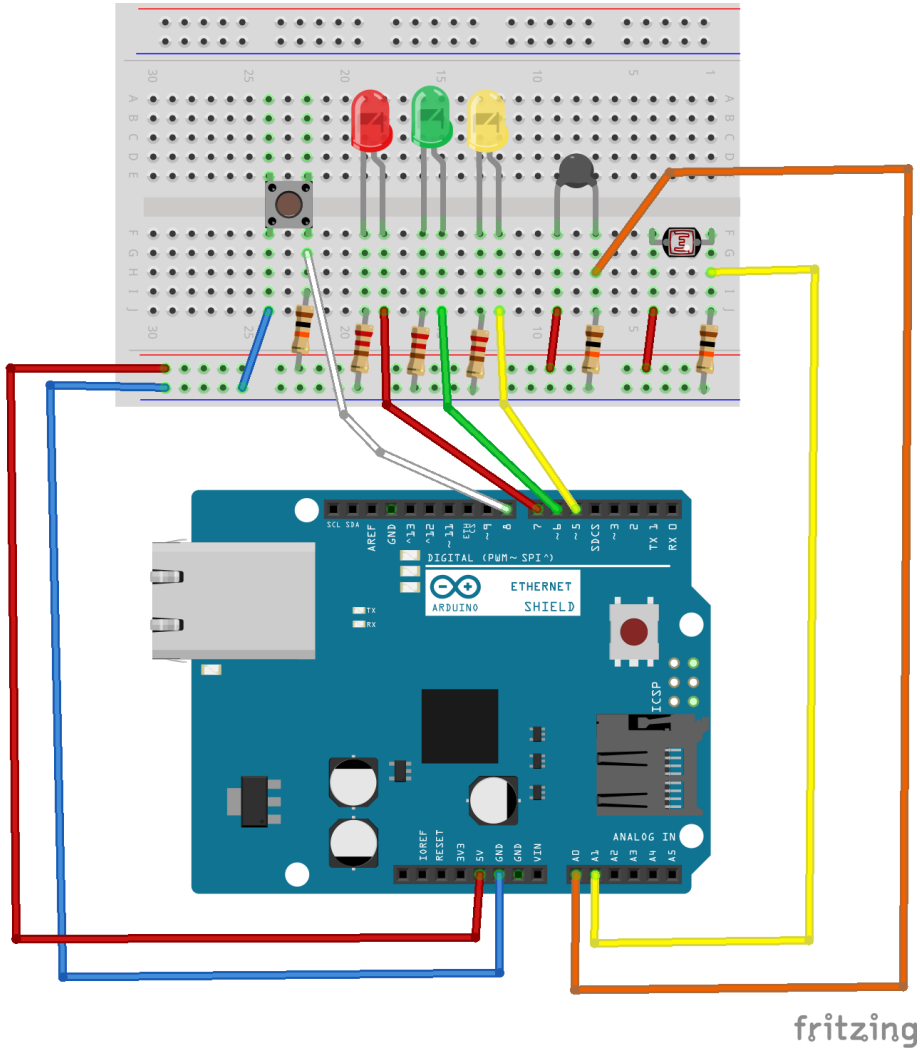
- http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html
  Een heldere uitleg hoe het HTTP protocol is samengesteld met mooie en duidelijke voorbeelden
- http://www.w3.org/Protocols/
  De officiële protocolbeschrijvingen van het W3C. De volgende twee documenten zijn van belang:
  - RFC7230 http://tools.ietf.org/html/rfc7230
  - RFC7231 http://tools.ietf.org/html/rfc7231

  Deze protocollen hoef je niet helemaal door te lezen en te onthouden, maar zoek de voorbeelden uit de eerste bron terug en herken waar deze 'vandaan komen'.

4.3.2 Benodigde hardware

- Arduino + usb cable
- Ethernet shield + ethernet cable
- LDR (light dependent resistor)
- Thermistor
- Button
- 3x resistor 10kΩ
- 3x resistor 220Ω
- 3 Leds (yellow, green, red)
- Wires

4.3.3 Fritzing schema



4.3.4 Deelopdracht A: Bouw de opstelling en test deze

Bouw bovenstaande opstelling en schrijf een test-sketch in Arduino om de aansluitingen te testen (laat de ledjes even knipperen, reageer op de knop, druk de uitgelezen waardes van de sensoren af naar de Seriële poort). De test-sketch is onderdeel van je oplevering.

4.3.5 Deelopdracht B: Weerstation als webserver

In voorgaande lessen heb je een weerstation gemaakt die steeds zelfstandig data opstuurt. Nu maak je een weerstation dat actief benaderd wordt en steeds correcte data teruggeeft. Verder is het een actief[3] weerstation, het houdt namelijk de temperatuur in de gaten of deze nog tussen (instelbare) grenzen ligt. Als de temperatuur binnen de grenzen ligt, dan brandt de groene led. Komt de temperatuur beneden het minimum, dan gaat de gele led branden. En als de temperatuur boven het maximum komt, dan gaat de rode led branden.

Voordat je aan dit project begint zul je helder op een rijtje moeten krijgen wat er nu exact opgeleverd moet worden. M.a.w. maak voordat je begint een lijst met functionaliteiten en overleg dat met de opdrachtgever (je docent). Maak weer een gedegen ontwerp, je kunt hiervoor diverse schematechnieken gebruiken die je in voorgaande courses geleerd hebt. Denk bijv. aan state-

[3] 'actief' betekent in deze context dat het weerstation een realtime systeem is, dus dat er direct op gebeurtenissen reageert (m.a.w. geen delays)

machines, sequence diagrams, activity diagrams, etc. etc. Leg daarnaast vooral het communicatieprotocol vast. Dus, beschrijf alle requests in tekstvorm en alle response in tekst, zodat je weet wat er in de requests komt te staan. Denk vooraf na over hoe je e.e.a. gaat oplossen, want je zult tegen de limieten van de Arduino aanlopen als je domweg alleen String gebruikt.

De webserver moet de volgende requests ondersteunen:

- Get / --> html pagina met alle statussen (HTML)
- Get /temp --> response temperatuur en, min en max (JSON)
- Get /lux --> response lux (JSON)
- Get /data --> alle data (JSON)
- Put of post? /temp --> instellen min en max (voor de kleurtjes van de leds) (JSON)

Leg per request in detail vast wat er verwacht wordt en wat de response zal zijn. Gebruik daarvoor onderstaand formaat. Werk eerst een definitie uit, bespreek deze eventueel en ga 'm daarna pas erbij bouwen.

Twee voorbeelden (let op, na de headers altijd een lege regel, ook als er geen content is):

01 Alle statussen (index pagina)	
Request om alle data en instellingen te presenteren.	
Request	`GET http://10.0.0.45/ HTTP/1.1` `Host: 10.0.0.45`
Response	`HTTP/1.1 200 OK` `Content-Type: text/html; charset=utf-8` `Connection: close` `<!doctype html>` `<html>` `  <head>` `  <meta charset="UTF-8">` `  <title>Weerstation</title>` `  </head>` `  <body>` `  Temperatuur: 24.5 graden Celcius` `  (min: 18, max 30) ` `  Lichtintensiteit: 473 lux ` `</body>` `</html>`
Opmerking	De HTML is hier mooi opgemaakt voor de leesbaarheid, maar bij de implementatie hoeft dat natuurlijk niet, mits het maar valide HTML is. Let op, enters (nieuwe regels) zijn twee karakters.

02 Vraag de temperatuur op	
Request om de huidige temperatuur en instellingen op te vragen	
Request	`GET http://10.0.0.45/temp HTTP/1.1` `Host: 10.0.0.45`

Response	`HTTP/1.1 200 OK`
	`Content-Type: application/json; charset=utf-8`
	`Connection: close`

	Content-length: 38 {"temperatuur":24.5,"min":18,"max":30}

4.4 Gateway: "van Arduino naar Gateway naar Internet"

In de voorgaande opdrachten konden alle onderdelen zelf verbinding maken met internet. In de praktijk is dat niet altijd zo, niet elk apparaat maakt zelf verbinding met internet. Een mogelijke oplossing kan het inzetten van een gateway zijn. Een 'gateway' betekent in het Nederlands zoiets als 'portaal', 'poort', etc. In deze weekopdracht ga je een gateway (een eigen webservice) maken die data van verschillende weerstations doorstuurt naar een webservice op internet die de data weer opslaat. Daarnaast biedt de webservice via de gateway toegang tot de weerstations om deze te configureren.

Hieronder vind je eerst de algemene opdrachtomschrijving en instructies. Daarna staat uitleg hoe om te gaan met de online webservice. Lees eerst alles door voordat je aan de slag gaat. In de les zal er nog aandacht besteed worden aan hoe om te gaan met de online webservice.

4.4.1 Eindopdracht

De opdracht luidt:

"Maak een netwerk van weerstations die hun meetwaarden naar een gateway sturen. De gateway stuurt deze data door naar een webserver. De webserver visualiseert de inkomende data (gemiddelden, grafieken, tijdsaspecten, etc.) in een website. De temperatuurgrenzen, voor de leds, zijn per weerstation te configureren via de ~~webserver~~ gateway."

Oftewel:

- De Arduino (het weerstation) registreert data en verstuurt dat naar een eigen te bouwen webservice, (de gateway).
- De Gateway stuurt de data weer door naar een webservice op internet.

Enkele voorwaarden:

- Gebruik als basis het weerstation dat je gemaakt hebt in Week 2 (weerstation post zelf actief de data naar de webservice, en nu dus naar de gateway)
- Er moeten meerdere weerstations via één gateway kunnen gaan.
- Alle communicatie tussen de onderdelen verloopt via REST-protocollen en standaard dataformaten (dus JSON, x-www-form-urlencoded, etc.).
- IP-nummers kunnen veranderen (denk maar aan het Wifi-netwerk op school). Dus een ip-nummer als identificatie is niet handig, verzin daar een oplossing voor. M.a.w. als een ip-nummer van een weerstation en/of een gateway verandert, moet de data nog steeds herleidbaar zijn naar dat specifieke weerstation/gateway.

Aandachtspunten:

1. Deze opdracht is een geheel project, pak dat ook als zodanig aan. De docent treedt op als opdrachtgever annex 'klant'. De opdrachtomschrijving is maar summier, zorg dat je in overleg met de docent de juiste zaken maakt.
2. Leg de opdrachtomschrijving vast
3. Analyseer de wensen en eisen en prioriteer deze.

4. Bedenk een globale architectuur, leg deze vast in een afbeelding. Beschrijf vooral waar welke componenten zich bevinden, hoe deze met elkaar communiceren en welke 'berichten' ze naar elkaar sturen.
5. Maak een planning van welke onderdelen je gaat maken en in welke volgorde.
6. Ga het project bouwen volgens de stappen die je bij het vorige onderdeel bedacht (en besproken) hebt:
   - Werk iteratief, m.a.w. ga niet alles volledig van te voren ontwerpen, maar pak steeds een onderdeel of functionaliteit.
   - Maak per onderdeel ontwerpen (code, schermen, database, etc.)
   - Beschrijf steeds het communicatieprotocol (zoals je ook bij de eerdere, dit voorkomt veel problemen bij het uitwerken naar code.
   - Test je werk geregeld of alles nog werkt.
7. Aan het eind van het project moet alles werken en er moet een rapport liggen waar alle analyse- en ontwerpproducten in vastliggen. Deze rapporten bevatten geen lange inleidende en beschrijvende teksten, maar vooral plaatjes, schema's, etc. die je werk verduidelijken.

4.4.2 De online webservice

De online webservice die als data store gebruikt gaat worden kun je vinden op:
http://iot.jorgvisch.nl

1. Maak op de site een account aan. Gebruik je HAN-mailadres als naam, zodat bekend is wie het account aangemaakt heeft. Op gezette tijden worden accounts die onbekend zijn geblokkeerd.
2. Log in op de site en roep met je webbrowser de 'groet'-service aan: http://iot.jorgvisch.nl/api/groet. Als het goed is krijg je een JSON of XML-bericht terug met als content "*Hoi <mailadres>*".
3. Log uit en roep de 'groet'-service nog eens aan. Als het goed is, krijg je nu de respons "*Jou ken ik niet*".
4. Roep de 'groet'-service aan met Fiddler, ook nu moet je de response "*Jou ken ik niet"* krijgen.
5. De online webservice maakt gebruik van zogenaamde Token based authentication. Bestudeer de volgende tutorial:
   http://www.asp.net/web-api/overview/security/individual-accounts-in-web-api
   Bestudeer vooral de HTTP requests en repsonses van de paragrafen 'Get an Access Token' en 'Send an Authenticated Request'. Het registeren van een gebruiker hoeft niet middels een HTTP request, want dat hebt je al in stap 1 (hierboven) gedaan.
   Voer de volgende stappen uit:
   a. Roep met Fiddler http://iot.jorgvisch.nl/token aan om een access token te krijgen.
   b. Roep de 'Groet'-service aan met het verkregen token. Als je het juiste token hebt gebruikt krijg je de response "*Hoi <mailadres>"*

Opvragen van een Token

Request (let op dat je geen enter achter het password zet):

```
POST http://iot.jorgvisch.nl/Token HTTP/1.1
Host: iot.jorgvisch.nl
Content-Type: application/x-www-form-urlencoded
Content-Length: 64

grant_type=password&username=jorg.visch%40han.nl&password=foutje
```

Response (een voorbeeld):

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 687
Content-Type: application/json;charset=UTF-8
Expires: -1
Server: Microsoft-IIS/10.0
X-SourceFiles: =?UTF-
8?B?YzpccHJvamVjdHNcanZpc2NoLnZpc3VhbHN0dWRpby5jb21cSW50ZXJuZXQgT2YgVGhpbmdzXGlvdC5
qb3JndmlzY2gubmxcaW90Lmpvcmd2aXNjaC5ubFxUb2tlbg==?=
X-Powered-By: ASP.NET
Date: Mon, 28 Sep 2015 20:11:29 GMT

{"access_token":"rbKTZgKdbYqdbu04ZHUFyG7UiTslqL1hRU……1nrtU","token_type":"bearer"
,"expires_in":7199,"userName":"jorg.visch@han.nl",".issued":"Mon, 28 Sep 2015
20:11:29 GMT",".expires":"Mon, 28 Sep 2015 22:11:29 GMT"}
```

Aanroepen met een token

Request

```
GET http://iot.jorgvisch.nl/api/greetings HTTP/1.1
Host: iot.jorgvisch.nl
Authorization: Bearer rbKTZgKdbYqdbu04ZHUFyG7UiTslqL1hRU……1nrtU
```

Response (een voorbeeld):

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-SourceFiles: =?UTF-
8?B?YzpccHJvamVjdHNcanZpc2NoLnZpc3VhbHN0dWRpby5jb21cSW50ZXJuZXQgT2YgVGhpbmdzXGlvdC5
qb3JndmlzY2gubmxcaW90Lmpvcmd2aXNjaC5ubFxhcGlcZ3JvZXRc?=
X-Powered-By: ASP.NET
Date: Mon, 28 Sep 2015 20:20:55 GMT
Content-Length: 23

"Hoi jorg.visch@han.nl"
```