Houd de tijd bij met ESP32 en Toggl

werken met de M5Stack

Mathias Claußen (Elektor)

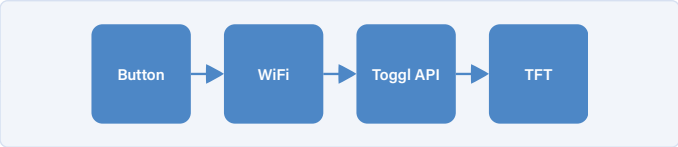
Het kan van pas komen de tijd bij te houden die u aan elektronicaprojecten besteedt. Dat kan met verschillende services. We gebruiken hier Toggl.com om te demonstreren hoe u web-requests kunt doen met HTTPS en hoe u basale GUI-functies kunt implementeren. Dat doen we allemaal met de M5Stack, een snel prototype- en ontwikkelplatform voor ESP32-gebaseerde projecten.

Figuur 1. M5Stack Core.

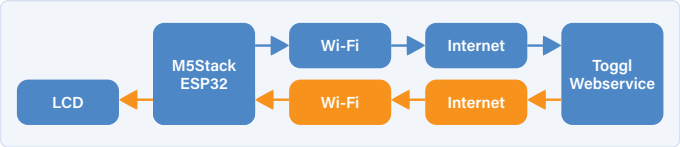
In deze moderne tijden, vooral als u van thuis uit werkt, kan het bijhouden van uw taken en de tijd die u eraan besteedt, u helpen uw aandacht erbij te houden. Het maakt het ook gemakkelijker om uw werk te bespreken – met collega's of met klanten. Er zijn veel oplossingen op de markt – zoals openTimetool, Toggl en Kimai – die tijd- en projectregistratie bieden die ook bruikbaar zijn om facturen voor klanten te genereren. Voor dit artikel heb ik de Toggl track-webservice gebruikt, een webgebaseerde oplossing die ook een plugin voor uw webbrowser of een app voor uw telefoon biedt om de tijd bij te houden. Het biedt een open API om met een tijdregistratiesysteem te interfacen, waardoor u uw eigen app of hardware kunt bouwen voor het bijhouden van tijd met behulp van de services van het bedrijf.

KISS

"Keep it simple stupid" (KISS). Voor een tijdregistratiesysteem is dit het beste wat u een gebruiker kunt bieden. Het bijhouden van de tijd is voor sommigen van ons meer iets wat we *moeten* doen dan iets wat we graag doen. We hoeven maar op één knop te drukken als we aan het werk gaan, en op een andere als we stoppen – eenvoudiger kan haast niet. Aangezien Toggl een online-service is, hebben we iets nodig dat verbinding kan maken met het internet (al dan niet draadloos) en onze request kan indienen. Voor zo'n taak denken we al snel aan de ESP32. Om een warboel van draden en componenten op een breadbord te vermijden, volgen we de KISS-route en pakken we een M5Stack Core Basic (**figuur 1**), waarvan de documentatie online beschikbaar is [1]. Naast de ESP32 krijgen we drie knoppen, een display, een batterij en wat andere periferie in een fraai compacte behuizing. Het enige dat we dan nog nodig hebben zijn een paar regels code.



Figuur 2. Datastroom naar de Toggl-service.



Figuur 3. Datastroom om de huidige status te verkrijgen.

Het basisconcept

Wat we nodig hebben is een apparaat dat toegang heeft tot de Toggl-webservice en dat op ten minste één knop kan reageren om aan te geven dat we aan het werk zijn, of juist niet natuurlijk. In het geval van de M5Stack betekent dit dat we WiFi gebruiken voor internetconnectiviteit, aangezien een access point binnen handbereik is en we ervan uit kunnen gaan dat we permanent online zijn. Zoals u in figuur 1 kunt zien, hebben we drie knoppen om uit te kiezen. En omdat er toch een LC-display is, maken we wat graphics om de status weer te geven (aan het werk of buiten spelen).
Hoe moeilijk kan de toegang tot Toggl zijn? In theorie niet zo moeilijk, aangezien de API goed gedocumenteerd is. Het komt erop neer dat op een knop moet worden gedrukt en dat een passend commando naar de Toggl-webservice moet worden gestuurd. Vervolgens wordt de status gewijzigd en dit krijgt de gebruiker dan weer te zien. Het klinkt eigenlijk eenvoudig. **Figuur 2** toont de datastroom naar de Toggl-API na een druk op een knop. De datastroom voor een update van het LCD met de actuele informatie is te zien in **figuur 3**: een datastroom naar de Toggle-server ergens in het net, en een datastroom terug naar de ESP32. Dit klinkt eenvoudig en ziet er ook zo uit.
Voor het WiFi-gedeelte is dit al vele malen gedaan en we gebruiken code die al in veel ESP32-projecten van Elektor is toegepast. Dit verschaft ons een modulaire webserver, met een mechanisme om tijd en OTA-updates af te handelen – dat laatste is overigens meer luxe dan noodzaak. Een protocol-stack wordt gebruikt voor de communicatie met de Toggl-service. **Figuur 4** toont de communicatie-stack en de betrokken bibliotheken. Het ESP32 Arduino-framework zorgt voor de netwerkcommunicatie met een WIFIClient-bibliotheek, die een verbinding mogelijk maakt met servers die bereikbaar zijn via de WiFi waarop de ESP32 is aangesloten. Daarbovenop hebben we de HTTPClient-bibliotheek die het HTTP-protocol en de HTTP-requests beheert, eveneens opgenomen in het ESP32 Arduino-framework. Aangezien Toggl vraagt om gegevens over HTTP uit te wisselen in JSON-formaat, hebben we een extra bibliotheek nodig om JSON af te handelen. In dit geval gebruiken we de ArduinoJSON-bibliotheek.
Naast de communicatie hebben we ook een bibliotheek nodig om het display in de M5Stack aan te sturen. Een al bekende bibliotheek die met het M5Stack-display, is de TFT_eSPI die een ruime keuze aan kant-en-klare functies levert voor het tekenen van graphics. Aangezien deze compatibel is, alle vereiste functies biedt om tekst en grafische elementen te tekenen en al in vorige projecten is gebruikt, is het zonde die niet te gebruiken. Een kort overzicht van de grafische stack van de TFT_eSPI is te zien in **figuur 5**.
Eerst echter gaan we nader op enkele belangrijke onderwerpen in, te beginnen met HTTP dat noodzakelijk is om toegang te krijgen tot de Toggl-service.

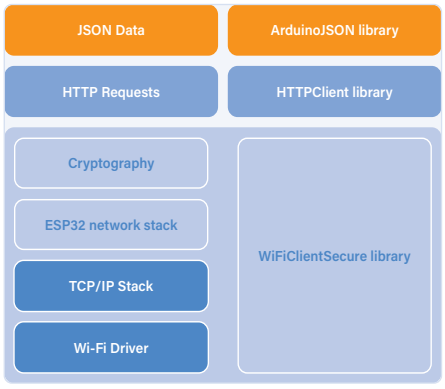
HTTP

Hypertext Transfer Protocol (HTTP), in 1989 bij CERN ontwikkeld, wordt gebruikt voor het aanvragen van een resource door een client (bijvoorbeeld uw webbrowser) en een antwoord van een webserver te krijgen. Als u bijvoorbeeld www.elektor.com wilt openen, stuurt uw client een request naar de webserver. Daartoe wordt een TCP/IP-verbinding tot stand gebracht en een request-string verzonden, zoals te zien in **listing 1**.
De eerste regel, een HTTP-header met `GET`, vertelt de server dat we iets vragen. De `/` vertelt de server dat we de standaard webpagina willen hebben. Met `HTTP/1.1` vertellen we de server dat we HTTP-versie 1.1 gebruiken. Er wordt gewerkt aan HTTP/3 (de derde protocolgeneratie), maar voor de meeste van onze toepassingen volstaat HTTP/1.1.

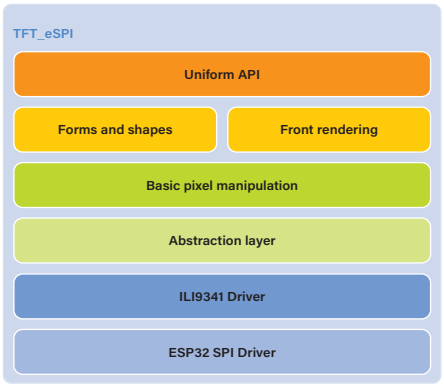
Listing 1. HTTP-request.

```
GET / HTTP/1.1
Host: www.elektor.com
User-Agent: curl/7.55.1
Accept: */*
```

Met betrekking tot de tweede regel, de Host, moet u weten dat de namen die u in uw browser intypt door een DNS-service worden vertaald naar een IP-adres. Dit betekent dat *elektor.com* vertaald zal worden naar *83.96.255.227*, het IP-adres dat wordt gebruikt voor de TCP/IP-verbinding. Achter dit IP-adres kan zich één webserver bevinden die niet alleen elektor.com afhandelt maar ook elektor.de of elektor.nl. Om de webserver te laten weten welke pagina u wilt, is de tweede regel met `Host` nodig.
Met `User-Agent` in regel 3, vertellen we de webserver wat voor HTTP-client we zijn, in dit geval *curl*. Hierdoor kan een webserver bijvoorbeeld pagina's presenteren die voor uw webbrowser geoptimaliseerd zijn, omdat Safari zich anders gedraagt dan Firefox. De laatste HTTP-header in regel 4 vertelt onze client om alle soorten inhoud te accepteren. Elk HTTP-request wordt afgesloten met een lege regel.
De webserver zal nu een antwoord op onze request formuleren, ook beginnend met



Figuur 4. Protocol-stack en bibliotheken.



Figuur 5. 'Stacked' tekenfuncties voor het LC-display.

Listing 2. HTTP-antwoord.

```
HTTP/1.1 200 OK
Server: unknown
Date: Tue, 17 Nov 2020 13:34:04 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=63072000
Pragma: no-cache
Expires: -1
Cache-Control: no-store, no-cache, must-revalidate, max-age=0
Accept-Ranges: bytes
Strict-Transport-Security: max-age=63072000
```

Index	Binair	Gecodeerd
0	000000	A
1	000001	B
2	000010	C
3	000011	D
4	000100	E
5	000101	F
6	000110	G
7	000111	H
8	001000	I
9	001001	J
10	001010	K
11	001011	L
12	001100	M
13	001101	N
14	001110	O
15	001111	P
16	010000	Q
17	010001	R
18	010010	S
19	010011	T
20	010100	U
21	010101	V
22	010110	W
23	010111	X
24	011000	Y
25	011001	Z
26	011010	a
27	011011	b
28	011100	c
29	011101	d
30	011110	e
31	011111	f
32	100000	g

Index	Binair	Gecodeerd
33	100001	h
34	100010	i
35	100011	j
36	100100	k
37	100101	l
38	100110	m
39	100111	n
40	101000	o
41	101001	p
42	101010	q
43	101011	r
44	101100	s
45	101101	t
46	101110	u
47	101111	v
48	110000	w
49	110001	x
50	110010	y
51	110011	z
52	110100	0
53	110101	1
54	110110	2
55	110111	3
56	111000	4
57	111001	5
58	111010	6
59	111011	7
60	111100	8
61	111101	9
62	111110	+
63	111111	/
OPVULLING		=

Tabel 1. BASE64-coderingstabel (bron: https://tools.ietf.org/html/rfc2045#section-6.8).

een reeks HTTP-headers zoals in **listing 2**. De eerste regel geeft de protocolversie en een antwoordcode. Aangezien we hebben gevraagd om HTTP/1.1, zal de server ook in deze protocolversie antwoorden. De antwoordcode die we krijgen is `200`, wat aangeeft dat ons request kon worden verwerkt. Dit wordt gevolgd door meer headers met aanvullende informatie.

Als we alleen geïnteresseerd zijn in de opgevraagde inhoud, kunnen we de header overslaan. De header wordt door een lege regel gescheiden van de inhoud die we hebben opgevraagd (niet getoond in listing 2). Voor het geval u zich dat afvraagt, de header zelf kan tot 8 KB of meer aan data bevatten. De body en de lengte kunnen worden bepaald door `Transfer-Encoding: chunked` of `Content-Length:<length>` waarbij de eerste wordt gebruikt voor inhoud van onbekende lengte, zoals dynamisch gegenereerde webpagina's, en de laatste voor inhoud waarvan de grootte van te voren bekend is, zoals plaatjes.

Er zijn meer http-methodes, zoals POST, waarbij gegevens worden overgebracht van de client naar de server. Deze wordt gebruikt als u informatie indient via een webformulier. Ook de PUT-methode kan worden gebruikt om gegevens over te brengen naar een webserver. Meer informatie is onder andere bij het Mozilla Developer Network [2] te vinden voor andere gedefinieerde methoden.

Van HTTP naar HTTPS

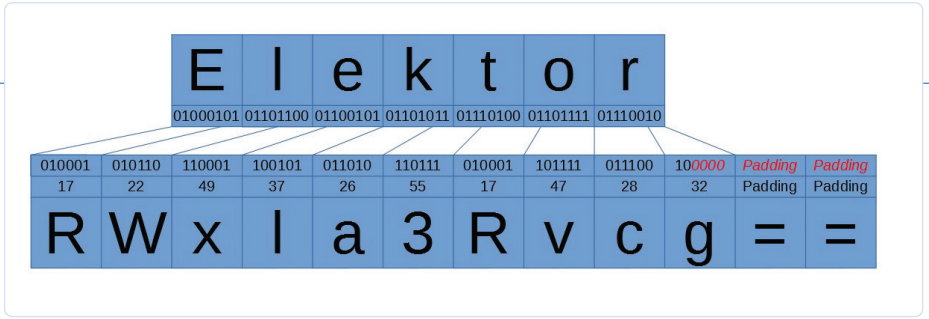
Toen HTTP werd ontwikkeld, is niet voorzien in geïntegreerde beveiliging voor het communicatieprotocol. De informatie wordt verzonden als platte tekst en iedereen met een beetje netwerkkennis kan die lezen of wijzigen. HTTPS, waar de 'S' voor *secure* staat, voegde een extra laag toe aan de communicatiestack. De server en de client praten nog steeds via het HTTP-protocol, maar in plaats van communicatie in platte tekst via een TCP/IP-verbinding, wordt al het verkeer tussen beide partijen eerst verwerkt door *Transport Layer Security* (TLS), dat de uitwisseling van cryptografische sleutels en encryptie beheert.

Dat betekent dat wanneer we HTTPS gebruiken, we meer rekenkracht, flash en RAM nodig hebben voor encryptie en decryptie, en de extra routines voor TLS in onze software moeten worden opgenomen. Op een moderne computer is voldoende rekenkracht voorhanden. Kleinere embedded systemen zullen communicatievertra-

gingen ondervinden als encryptie/decryptie en het beheer en de uitwisseling van sleutels uitsluitend in software worden gedaan. Daarom vindt u op moderne MCU's cryptografische accelerators die de encryptie van uw MCU overnemen. Bij de ESP32 is hardwareversnelling beschikbaar om dit proces te versnellen.

Request en antwoord

De Toggl-service biedt een open, gedocumenteerde API [3] om gegevens uit te wisselen. De API werkt via het indienen van POST-, PUT- en GET-requests naar een URL met gedefinieerde header en payload. Alle acties vereisen een authenticatie-sleutel die kan worden verkregen van de Toggle-webpagina. Met de verstrekte informatie kan het opbouwen van een request beginnen. We gebruiken de `WiFiClientSecure` klasse in combinatie met het http client-object. Dit is een bewezen manier om te communiceren met de server, zodat we niet opnieuw het wiel moeten uitvinden als het gaat om het verwerken van http-data.
We genereren van het authorisatie-token, een Base64-gecodeerde combinatie van gebruikersnaam en wachtwoord, met de hand. Als invoer voor de Base64-codering hebben we een API-sleutel en `:api_token` als een string. We kunnen de API-sleutel van de Toggl-service verkrijgen. Aangezien deze niet in platte tekst wordt verzonden, kan hij tekens bevatten die problemen veroorzaken met het gereserveerde teken voor de header zelf. Om gegevens van welke aard dan ook in deze requests op te nemen, is de oplossing een BASE64-codering van data. Gewoonlijk kunnen onze gegevens in een byte een waarde tussen 0 en 255 hebben. BASE64 gebruikt slechts 64 waarden (zie **tabel 1** voor een vertaaltabel) die zijn toegewezen aan het equivalent van de ASCII-karakters `A-Z`, `a-z`, `0-9`, `+`, `/` en `=` als opvulling. Als we bijvoorbeeld *Elektor* als een string in BASE64 coderen, krijgen we *RWxla3Rvcg==* (zoals te zien in **figuur 6**). Opgemerkt moet worden dat e-mail-attachments om vergelijkbare redenen ook worden gecodeerd met BASE64, en zoals u in het voorbeeld kunt zien, wordt een attachment hierdoor ongeveer 37% groter dan het origineel.
Zodra de authorisatie aan de HTTP-client is doorgegeven, kan het request worden ingediend. Het volgende voorbeeld laat zien hoe een GET-request wordt geïmplementeerd en hoe de geretourneerde informatie wordt verwerkt, zie **listing 3**. De



Figuur 6. Elektor, gecodeerd in BASE64.

`GET()`-functie retourneert de antwoordcode van de server op onze request, of als we een communicatieprobleem hebben, een waarde kleiner dan nul. De antwoordcode (response code) kan waarden in verschillende bereiken hebben, zoals [4] laat zien. De bekendste code is *404 Resource not found*. We verwachten hier code 200, Response okay en controleren daarop. Als er gegevens van de server worden ontvangen, kunnen deze worden opgehaald met `getString()` voor verdere verwerking.
Vergelijkbare code bestaat voor PUT en POST, met enkele kleine maar belangrijke verschillen, zoals te zien in **listing 4**. De `addHeader`-functie die voorafgaand aan `PUT()` wordt gebruikt zal twee extra headers toevoegen aan de request. De eerste is *Content-Type* die de webserver vertelt welk type gegevens we zullen versturen; de tweede is *Content-length*, en dit is waar HTTP plotseling een beetje anders wordt dan wat we eerder hebben gezien. Gewoonlijk zenden we na de header gegevens die tekst kunnen zijn, gescheiden door twee nieuwe regels en eindigend met twee nieuwe regels. Dit werkt voor tekst, maar bijvoorbeeld een JPEG-afbeelding

Listing 3. Code-snippet voor HTTP-request.

```
httpclient.begin(client,link);
   httpclient.setReuse(true);
   httpclient.setAuthorization((const char*)b64.c_str());
   int httpCode = httpclient.GET();
    if (httpCode > 0) { //Check for the returning code
        if(httpCode!=200){
            Serial.println("Response Error");
            String payload = httpclient.getString();
            Serial.println(httpCode);
            Serial.println(payload);
            *error = 1;
        } else{
            *Result = httpclient.getString();
        }
    } else {
        Serial.print("Request Error");
        Serial.println(httpclient.errorToString(httpCode));
    }
    httpclient.end();
```

Listing 4. HTTP-header toegevoegd.

```
httpclient.setAuthorization((const◦char*)b64.c_str());
   httpclient.addHeader("Content-Type","application/json");
   httpclient.addHeader("Content-length","0");
   int httpCode = httpclient.PUT((uint8_t*)(NULL),0);
      if (httpCode > 0) { //Check for the returning code

       . . .
```

Listing 5. Hergebruik van de verbinding.

```
httpclient.begin(client,link);
httpclient.setReuse(true);
httpclient.setAuthorization((const char*)b64.c_str());
httpclient.addHeader("Content-Type","application/json");
int httpCode = httpclient.POST(payload);
   if (httpCode > 0) {

. . .
```

Listing 6. Voorbeeld JSON-string.

```
{ "name": "Test", "value": 1351824120,"array": [ 123.45, 321.89 ] }
```

Lisitng 7. JSON genereren.

```
DynamicJsonDocument doc(capacity);
JsonObject time_entry = doc.createNestedObject("time_entry");
   if(description.length()>0){
       time_entry["description"] = description.c_str();
   }
time_entry["created_with"] = "ESP32";
serializeJson(doc, payload);
```

Lisitng 8. DeserializerJSON.

```
DeserializationError er = deserializeJson(doc, Result.c_str());
 if (er) {
      //something went wrong
         } else {
         if(false == doc["data"].isNull() ){
             JsonObject data = doc["data"];
             uint32_t data_id = data["id"];
             const char* data_start = data["start"];
            if(false == data["description"].isNull()){
                 const char* data_description = data["description"];
                  Element->description = String(data_description);
             } else {
                 Element->description = "";
             }
```

Listing 9. Toggl JSON-string.

```
{
"data":
   {
          "id":436694100,
          "pid":123,
          "wid":777,
          "billable":false,
          "start":"2013-03-05T07:58:58.000Z",
          "duration":1200,
          "description":"Brew some coffee",
          "tags":["billed"]
   }
}
```

kan bestaan uit elke combinatie van willekeurige bytes. Dit is waar *Content-length* in beeld komt. Headers worden gerangschikt en gescheiden van onze data door twee nieuwe regels. Met *Content-length* kunnen we elk type karakter of binaire data versturen. De webserver weet hoeveel bytes hij kan verwachten dankzij de opgegeven *Content-length*.

Voor de POST hoeven we alleen het content-type op te geven dat zal worden overgedragen. Er wordt geen eigenlijke payload verzonden. Als we een payload of inhoud zouden leveren die groter is dan 0 bytes, zou de http-client zelf een *Content-Length* header toevoegen. In dit speciale geval, waarbij de lengte op nul is gezet omdat we geen inhoud verzenden, wordt de header voor de content-lengte helemaal niet toegevoegd. Aangezien de server klaagt wanneer deze lengte ontbreekt, moeten we die zelf toevoegen met de `addHeader`-functie, zoals in listing 4.

Zoals u kunt zien in **listing 5**, is `setReuse()` op true gezet. Na dit request, hetzij POST, PUT of GET, zal de verbinding met de server normaliter door de client worden beëindigd. Als `setReuse()` true is, blijft de verbinding bestaan en wordt deze hergebruikt wanneer een volgende keer een request wordt ingediend. Dit spaart tijd omdat de verbinding niet opnieuw hoeft te worden opgebouwd als een volgende request wordt ingediend. Ook verbergt dit, op het moment van schrijven, een bug ergens in de netwerkstack, waardoor geheugen niet correct wordt vrijgegeven. Vroeg of laat kan dit ertoe leiden dat verbindingen mislukken omdat de ESP32 geen vrij geheugen meer heeft.

Gegevensuitwisseling met JSON

JSON staat voor *JavaScript Object Notation* en beschrijft een manier om efficiënt gegevens uit te wisselen tussen twee systemen. Alle gegevens worden verpakt in een string die er uit kan zien als die in **listing 6**. Deze string moet worden gegenereerd en weer gedecodeerd. De meeste omgevingen die gegevens voor webapplicaties verwerken en uitwisselen, gebruiken JSON en hebben ingebouwde functionaliteit of extra kant-en-klare bibliotheken voor genereren en opsplitsen van JSON. Dit geldt ook voor kleine apparaten zoals een Arduino of ESP32.

Een bibliotheek die met het Arduino Framework en de ESP32 kan worden gebruikt is ArduinoJSON; deze kan

JSON-strings ontleden en genereren voor uitwisseling met andere systemen. Aangezien Toggl JSON gebruikt om gegevens uit te wisselen, komt deze bibliotheek nu goed van pas. Van de Toggl-API kunnen we bepalen welke informatie verwacht wordt in JSON-strings. Deze documentatie beschrijft ook hoe gegevens moeten worden gegenereerd als informatie moet worden verzonden. Voor een nieuw tijd-item moeten we een JSON-string genereren met daarin een *time_entry* object. Binnen deze *time_entry* hebben we een *description*-string nodig en een optionele *created_with*-string voor de nieuwe entry.
De code in **listing 7** zal een nieuw `DynamicJsonDocument doc` op de heap van de ESP32 aanmaken met een vastgelegde capaciteit. Dit betekent dat geheugen in het niet-statisch gebruikte geheugen van de ESP32 wordt toegewezen met `malloc()` en later wordt vrijgegeven met `free()`. Het tegenovergestelde is StaticJsonDocument dat statisch in RAM wordt ondergebracht of, indien lokaal in een functie gebruikt, op de stack wordt geplaatst. StaticJsonDocument is sneller, maar omdat het lokaal gebonden is aan de stack betekent dit dat de grootte kleiner kan zijn dan wat we zouden kunnen gebruiken met DynamicJsonDocument. De functie `serializeJson()` zal de gewenste string creëren die kan worden verzonden. Het decoderen van JSON kan worden gedaan met `deserializeJson()`, dat elke fout retourneert die het tijdens het ontleden tegenkomt. Het codegedeelte staat in **listing 8**. Toegang tot de elementen binnen de JSON kan hierboven worden gezien. De string die door Toggl wordt geretourneerd, ziet eruit als in **listing 9**. De code controleert na het ontleden of het object *data* bestaat, dat wil zeggen of we een geldig resultaat hebben. Daarna benadert de code de elementen in *data* en pakt alleen de waarden die later nodig zijn voor weergave en gegevensverwerking. Dat zijn *id*, *start* en *description*. Die laatste is bijzonder, alleen als dit veld een waarde heeft binnen de Toggl-database wordt het geïntegreerd in *data*. Dit vereist dat met `data["description"].isNull()` moet worden gevalideerd dat de entry bestaat voordat we proberen er toegang tot te krijgen; anders zal de bibliotheek een exeption genereren.
De volledige gegevensuitwisseling is ingekapseld in de *toggleClient* bibliotheek. Hier ondersteunt het de basisbehoeften, het tonen van de huidige invoer, het starten van een nieuwe en het stoppen van de huidige lopende invoer. De code is niet perfect en iedereen is welkom om verbeteringen en foutcorrecties aan te dragen, aangezien dit meer bedoeld is als een uitgangspunt dan als een afgewerkte bibliotheek. Naast het verzamelen en verzenden van gegevens, moeten die ook worden weergegeven en dat is wat nu volgt.

Pixels tekenen

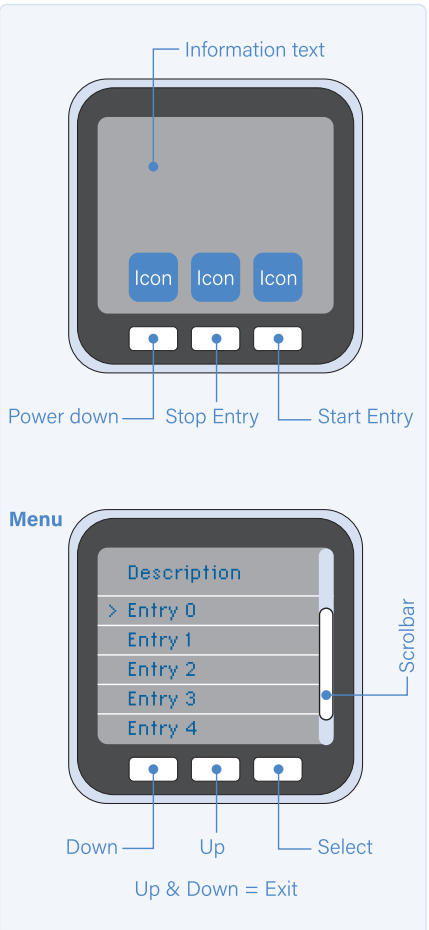
Voor het tekenen wordt de TFT_eSPI bibliotheek gebruikt, die u misschien nog kent van een ouder Elektor project, *Touch-GUI voor ESP32 en Raspberry & co* [5], en die met verschillende displays en ESP32-combinaties kan worden gebruikt. Omdat deze bibliotheek het display van de M5Stack ondersteunt kan sommige code uit andere projecten hergebruikt worden. In de bibliotheekmap moet het bestand *User_Setup_Select.h* aangepast worden om de ingesloten configuratie voor de M5Stack te gebruiken. Binnen *TogglButtonM5.ino* zal de regel `TFT_eSPI tft = TFT_eSPI();` een nieuw TFT_eSPI object aanmaken dat later gebruikt kan worden om te tekenen. Voor de initialisatie van het display hebben we slechts vier regels code nodig in onze `setup()`-functie, zoals te zien in **listing 10**. Het item `inverted display` moet op *true* worden gezet om de juiste kleurvolgorde voor het M5Stacks-display door te geven. Met `setRotation(1)` zal het display zijn onderkant hebben waar de knoppen van de M5Stack zich bevinden. Hierna kunnen alle mogelijke pixelbewerkingen gebruikt worden die de bibliotheek biedt.
Een kleine tip met betrekking tot de tekenprestaties, speciaal voor displays met de ILI9341. Deze displays zijn geoptimaliseerd voor het verwerken van volledige schermbeelden of grotere brokken schermdata. Pixelgewijs benaderen zal het tekenen minstens zeven keer trager maken. Dat komt door de manier waarop commando's en de toegang tot individuele pixels werken. De TFT_eSPI bibliotheek heeft het tekenen op de ESP32 al geoptimaliseerd en zal, waar mogelijk, de toegangstijden optimaliseren. Naast het instellen van pixels of het monochroom vullen van het scherm, voegt de bibliotheek ook andere tekenfuncties toe zoals lijnen, rechthoeken en printfuncties om strings op het scherm te zetten. En als iemand vraagt waarom u een bibliotheek gebruikt voor deze taak en geen driver schrijft, is het antwoord dat we het wiel niet opnieuw willen uitvinden.

Listing 10. LCD-initialisatie.

```
tft.init();
tft.invertDisplay( true );
tft.setRotation(1);
tft.fillScreen(TFT_WHITE);
```

De GUI, en een menu

De M5Stack heeft een display en drie knoppen. Time-tracking oplossingen hebben meestal slechts één knop voor het starten en stoppen van een entry. Aangezien de M5Stack ook op een batterij kan werken, kan een uitschakelknop erg handig zijn. **Figuur 7** toont het GUI-concept en ongeveer de positie van de enkele weer te geven items.
Aangezien de weergave van tekst vrij eenvoudig is, wordt de vraag natuurlijk *wat* we willen weergeven. De essentiële gegevens zijn daarbij de description (als we die hebben) en de begintijd. Voor de icoontjes gebruiken we die uit de open icon



Figuur 7. Ontwerptekening voor de GUI.

Figuur 8: Hoofdmenu van de GUI.

Figuur 9. Description-selectie in de GUI.

library [6]; het resulterende hoofdscherm is te zien in **figuur 8**. Hiermee kunnen we een nieuwe entry starten en stoppen, en het apparaat uitschakelen. Zo blijft het geheel gebruiksvriendelijk.

Als we een nieuwe Toggl-entry starten, zal die geen description bevatten. In principe hebben we die ook niet nodig, maar het zou fijn zijn als op zijn minst kunnen kiezen uit een aantal vooraf gedefinieerde descriptions. Het invoeren van descriptions met slechts drie knoppen is weliswaar mogelijk maar zeker niet gebruikersvriendelijk. Een totale herziening op basis van het project 'Wekker met drievoudige weergave' [7] resulteerde in een eenvoudig menusysteem dat met de TFT_eSPI-bibliotheek kan worden gebruikt. De schermresolutie en de oriëntatie van het menusysteem komen ook overeen met de configuratie die we hebben voor de Toggl-knop. Code kan in dit project worden getransplanteerd voor een menu zoals in **figuur 9**.

Dit is geen universeel, flexibel menu zoals u in een GUI-framework zou verwachten. Het is simpel, werkt met de TFT_eSPI en is gebouwd om met slechts een paar knoppen te worden gebruikt. Als ik meer tijd had gehad, zou ik de LVGL (*Light and Versatile Graphics Library*) [8] gebruikt hebben, omdat die ook werkt voor displays zonder aanraakfunctionaliteit. Nu we een zelfgemaakt menusysteem hebben, gaan we eens kijken hoe het werkt en hoe de graphics worden getekend. De logica voor het menu zit in een `RenderMenu()`-functie. Afhankelijk van de toestand, zal deze het menu tekenen en *true* retourneren. Als er geen menu wordt getekend, zal de functie *false* retourneren, wat aangeeft dat de inhoud van het scherm niet is veranderd. Als true wordt geretourneerd, moet worden vermeden om daarna nog meer te tekenen. Een vereenvoudigd stroomdiagram is te zien in **figuur 10**.

`RenderMenu()` bevat de menulogica, maar het tekenen doen we met `ShowMenuSettingsList()`. Deze functie tekent de menukop en maximaal vijf menu-items. De menulogica zorgt ervoor dat het geselecteerde item in een geldig bereik blijft. De tekenroutine berekent de startpositie en tekent vijf items als een lijst op het scherm. Ook wordt een schuifbalk (zoals bekend van veel Windows-gebaseerde systemen) berekend en getekend met een kleine indicator voor de positie binnen de lijst. De code van `ShowMenuSettingsList()` staat in **listing 11** en kan worden opgesplitst in drie delen. Het eerste is de koptekst. Om verwerkingstijd te besparen wordt deze uitsluitend opnieuw getekend wanneer dat nodig is.

Het centreren van de koptekst gebeurt handmatig door de halve lengte van de tekst in pixels te berekenen en die waarde af te trekken van de halve weergavebreedte. Dit

Listing 11. ShowMenuSettingsList.

```
void Menus::ShowMenuSettingsList(uint8_t selected_idx, bool refresh){

  /* Draw Headline */
  if(true == refresh){
    _lcd->fillRect(0,0,320,40,_lcd->color565( 45,47,50 ) );
    _lcd->setTextColor(_lcd->color565( 112,116,122 ),TFT_BLACK);
    _lcd->setFreeFont(FSB18);
    _lcd->setCursor(160- ( _lcd->textWidth("Description") / 2) ,30);
    _lcd->print("Description");
  }
  /* Headline done */

  /* Draw scrollbar */
  _lcd->fillRect(303,43,15,197,TFT_WHITE);
  _lcd->drawRect(300,40,20,200,_lcd->color565( 112,116,122 ));
  _lcd->drawRect(301,41,18,198,_lcd->color565( 112,116,122 ));
  _lcd->drawRect(302,42,16,196,_lcd->color565( 112,116,122 ));
  _lcd->fillRect(305,45 +(196/GetUsedEntryCount())*selected_idx
   ,10,(196/GetUsedEntryCount())-2 , TFT_DARKGREY);
  /* Scrollbar done */

  /* Menu entries */
  uint8_t startidx=0;
  uint8_t endindex=0;
  if(selected_idx>4){ //We can display 5 Elements
    startidx = selected_idx -4 ;
  }
  if((startidx+5)>GetUsedEntryCount()){ //Limit last drawn item
      endindex=GetUsedEntryCount();
  }else{
      endindex=startidx+5;
  }
  /*Draw up to five elements*/
  for(uint8_t i=startidx;i<endindex;i++){
          DrawSettingsMenuEntry( (40*(i-startidx))
   ,DescriptionArray[i].c_str() ,( i==selected_idx ) );
  }
  /* Menu entries drawn */
}
```

werkt als de tekst kleiner is dan 160 pixels en kan onbekende resultaten opleveren als de tekst langer is. De tweede is de scrollbar, die aan de rechterkant van het menu wordt getekend. De scrollbar bestaat uit een aantal rechthoeken. De lengte van de scrollbar wordt berekend uit het aantal items in het menu. Dit werkt zolang er minder dan 196 items in de lijst staan, anders kunnen er rare dingen gebeuren.

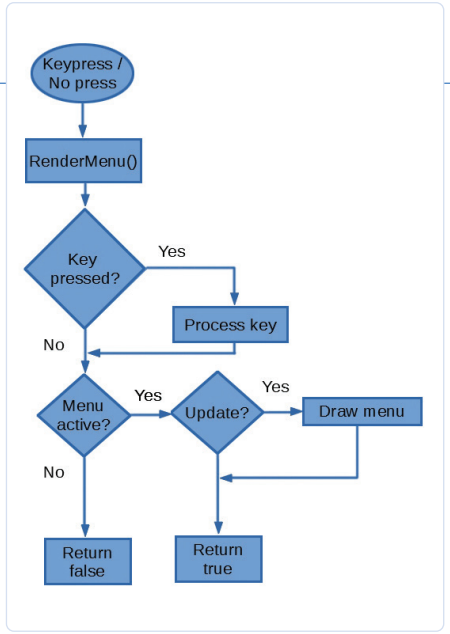
Voor het derde deel worden de stat-index en de eind-index voor de te tekenen lijstitems berekend. Aangezien we slechts vijf elementen kunnen tekenen, moeten we dat bereik dienovereenkomstig verschuiven ten opzichte van de door de gebruiker geselecteerde index. De drie delen zijn verantwoordelijk voor de lijst zoals die wordt weergegeven. Omdat elk item in de lijst er hetzelfde uitziet, wordt dit item getekend met zijn eigen functie die alleen een offset nodig heeft om op de juiste hoogte te tekenen, plus een vlag als dit item geselecteerd is. Dit genereert een slank menusysteem dat voor dit doel geschikt is en enkele basale menuconcepten demonstreert.

Webgebaseerde configuratie

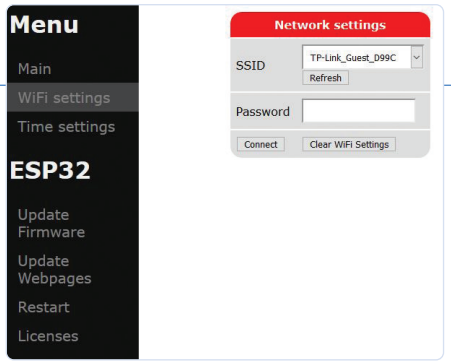
De software is niet vanaf nul geschreven. Ik heb delen van eerdere projecten hergebruikt. Dit geldt ook voor de webgebaseerde configuratie. Als de twee linkerknoppen worden ingedrukt tijdens het opstarten, zal de software een access-point starten waarmee u verbinding kunt maken met een computer of een draagbaar WiFi-apparaat. Anders zal de software proberen verbinding te maken met het WiFi-netwerk dat was geconfigureerd (indien aanwezig). Het starten van een webserver op een ESP32 kan eenvoudig zijn, en wordt gedaan door de software nadat WiFi is gestart en ons een webinterface presenteert, waarmee we het WiFi-netwerk kunnen instellen (**figuur 11**) en een Toggl-API-sleutel kunnen invoeren voor gebruikersauthenticatie (**figuur 12**).

Verdere ontwikkeling en wat we al weten

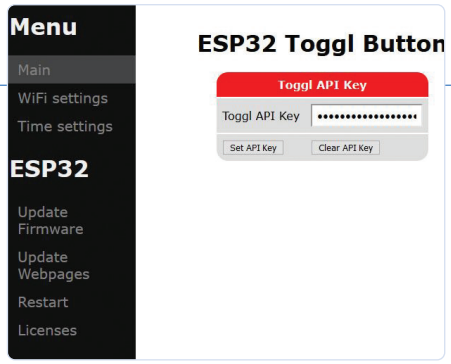
Hoewel de Toggl-knop werkt en het effectief bijhouden van de tijd mogelijk maakt, is er nog ruimte voor verbetering. In dit artikel heb ik slechts een paar van de vele aspecten van moderne IoT-apparaten aangestipt, vooral als ze interageren met een webserver of -browser. Termen als WiFi, HTTP-server, HTTP-client, TLS, JSON, JavaScript, HTML, GUI en C/C++ klinken wellicht modieus, maar het zijn in feite de essentiële ingredi-



Figuur 10. Stroomdiagram menusysteem.



Afbeelding 11. WiFi-configuratiepagina.



Figuur 12. Toggl API-sleutel invoeren.

enten voor dit eenvoudige project. Omgaan met HTTP en HTTPS en JSON aan de ene kant en GUI-ontwerp aan de andere, terwijl ook een webgebaseerde configuratie-interface wordt geboden, was ongebruikelijk in de begindagen van het IoT. Maar vandaag de dag is dit het absolute minimum voor een *connected* apparaat. Deze mengeling van verschillende vakgebieden – waaronder de basis van webontwikkeling, JSON, HTML, C/C++ en JavaScript – zou een heel boek kunnen vullen. Als zo'n boek verschijnt, zal Elektor daar melding van maken.

Tijdens het schrijven van dit artikel heb ik enkele ideeën opgedaan om de knop nog gebruiksvriendelijker te maken en om functies toe te voegen die de registratie zullen verbeteren. De eerste verbetering zou de overschakeling van de GUI naar LVGL zijn. Voor de GUI brengt dit ons snel naar wat we al weten. Als u een GUI aan uw project wilt toevoegen, zoek dan naar iets dat zich bewezen heeft en beschikbaar is. Er zijn zelfs oplossingen die een vriendelijke licentie hebben voor uw project. Uw eigen menusysteem implementeren voor het plezier van het leren kan leuk zijn, maar het wiel opnieuw uitvinden is dat niet.

Als u voor de ESP32 het Arduino-framework en de Arduino-bibliotheken gebruikt, kijk dan eens naar de GitHub-repository en de nog niet opgeloste problemen. Dit kan een goede bron zijn om problemen te vermijden of bekende beperkingen te omzeilen. De code voor deze Toggl-knop staat op de Elektor GitHub-pagina [9] en biedt een handige manier om de code te klonen en te verkennen omdat de geschiedenis bewaard blijft. Als u bugs vindt of suggesties hebt voor de code, kunt u de GitHub issues-functie gebruiken. Aangezien dit project veel onderwerpen raakt, kunt u me feedback geven als er onderwerpen zijn die in een toekomstig artikel meer in detail moeten worden uitgelegd.

200631-03

GERELATEERDE PRODUCTEN

- **M5Stack – ESP32 Basic Core Development Kit**
  www.elektor.nl/m5stack-esp32-basic-core-development-kit
- **M5Stack ESP32 Arduino MPU9250 Development kit**
  www.elektor.nl/m5stack-esp32-arduino-mpu9250-development-kit
- **W. Gay, *FreeRTOS for ESP32-Arduino* (Elektor 2020)**
  www.elektor.nl/freertos-for-esp32-arduino

Een bijdrage van
Ontwikkeling, tekst en diagrammen: **Mathias Claußen**
Illustraties: **Patrick Wielders**
Vertaling: **Jelle Aarnoudse**
Layout: **Giel Dols**

Vragen of opmerkingen?
Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via mathias.claussen@elektor.com of naar de redactie van Elektor, via redactie@elektor.com.

WEBLINKS

[1] **M5Core Basic**: https://docs.m5stack.com/#/en/core/basic
[2] **HTTP | MDN**: https://developer.mozilla.org/en-US/docs/Web/HTTP
[3] **Toggl.com API-documenten**: https://github.com/toggl/toggl_api_docs
[4] **HTTP Response Status Codes | MDN**: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
[5] **Touch-GUI voor ESP32 en Raspberry & co**: www.elektormagazine.nl/magazine/elektor-144/57038
[6] **Open Icon Library**: https://sourceforge.net/projects/openiconlibrary/
[7] **Wekker met drievoudige weergave**: http://bit.ly/elektor-3display-alarm-clock
[8] **Light and Versatile Graphics Library**: https://lvgl.io/
[9] **GitHub-projectpagina**: https://github.com/ElektorLabs/200631_ESP32_Toggl_Button