**Rest API**

**Get Request**

**Lab Exercise**

**Felix Brandstetter**

**Lars Weber**

# Setup Steps

* Mache ein git clone von https://github.com/F1103BR/VS\_RestAPI\_GET.git
* Importiere Template\_VS\_RestAPI in dein CCS
* Lade dir die mongoose ws version 6.18 von git herunter (im Reiter „Realeases“)
* Entpacke die Zip und kopiere die mongoose.h und .c file in dein Projektverzeichnis
* WICHTIG : Passe die HEAP- und Stack-Size an:



* Properties -> ARM linker -> Basic Options
* Erstelle eine mongoose\_config.h setzte in der config dieses define

**#define** CS\_PLATFORM CS\_P\_TM4C129

* Jetzt includiere diese config in die mongoose.h datei

Mit „“

* Suche in der mongoose.h datei nach:

**#if** CS\_PLATFORM == CS\_P\_TM4C129

* Füge in die Includes **#include** <stdbool.h> hinzu
* Da wir einen anderen Compiler verwenden müssen wir noch in der mongoose.c datein eine for-schleife anpassen (Ca. Zeile: 15677)

A computer screen shot of a math equation

Description automatically generated

* Das int i muss vor der for-Schleife deklariert werden

# LWIP Setup:

* Passe deine Include options an. Beachte hierbei auch dass du auf deinen workspace verweist. Im Projekt entspricht

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

* „{workspace\_loc:/$(ProjName)}“ verweist auf das Projekt im workspace
* SW\_ROOT ist der Pfad zum Tivaware Ordner
* Erstelle einen utils ordner in deinem Projektverzeichnis und kopiere die Dateien lwiplib.c uartstdio.c und ustdlib.c hinzu ( TivaWare\_C\_Series-2.2.0.295-> utils)

A screenshot of a computer

Description automatically generated

* Erstelle jetzt noch einen ordner mit drivers und kopiere aus dem tivaware ordner unter example -> boards -> tm4c129exl unter drivers die datei pinout.c hinein

# Anpassen der startup.ccs

**#include** "inc/hw\_nvic.h"

**#include** "inc/hw\_types.h"

**extern** **void** **lwIPEthernetIntHandler**(**void**);

**extern** **void** **SysTickIntHandler**(**void**);

SysTickIntHandler, // The SysTick handler

lwIPEthernetIntHandler, // Ethernet

# Konfigurieren und initialisieren von mongoose in der main

Am Anfang der main.c ist eine mongoose struktur defineirt

**struct** mg\_mgr g\_mgr;

Diese Struktur ist der HAUPT-Mongoose Event Manager

Das folgende muss im Code bei “//Configure mongoose and initialize it here” implementiert warden.

Wir müssen folgende Funktionen aufrufen um mongoose zu initialiseren und zu konfigurieren

mg\_mgr\_init(&g\_mgr, NULL);

Diese initialisiert den Mongoose Event Manager.

* &g\_mgr ist ein Zeiger auf die mg\_mgr Struktur, die als globaler Manager für alle Verbindungen dient.
* Der zweite Parameter ist ein optionaler Benutzerkontext, der hier nicht verwendet wird (daher NULL).

Jetzt konfigurieren wir den mongoose manager mit einem err char damit wir falls etwas schief läuft, wissen was genau der Fehler war (debugging).

**const char** \*err;

**struct** mg\_bind\_opts opts = {};

opts.error\_string = &err;

Jetzt erstellen wir eine mg\_connection struktur von mongoose und übergeben unsere konfigurationen.

Fügt das in den Code ein und füllt die funktion mg\_bind\_opt() mit den richtigen Übergabewerten.

**struct** mg\_connection \*nc = mg\_bind\_opt();

**if** (nc == NULL){

**UARTprintf**("Failed to create listener:%s\r\n", err);

**return** 1;

}

**struct** mg\_connection \*mg\_bind\_opt(**struct** mg\_mgr \*mgr, **const** **char** \*address,

MG\_CB(mg\_event\_handler\_t callback,

**void** \*user\_data),

Füllt die vier input Parameter selbst aus.

Tips:

1. **struct** mg\_mgr \*mgr : übergebt die Adresse des Haupt Mongoose Event Handler
2. **const** **char** \*address: Übergebt hier den Port für eine Standard http Verbindung
3. MG\_CB(mg\_event\_handler\_t callback: Hier muss der Event Handler übergeben werden, den ihr nachher Initialisiert. Übergabe: ev\_handler
4. Übergebt hier die struct von opts bei der wir den error string initialisiert haben.

nachdem wir die mg\_connection Struktur erstellt haben müssen wir mongoose noch mitteilen welches protokoll wir verwenden wollen.

mg\_set\_protocol\_http\_websocket(nc);

Jetzt ist mongoose initialisiert und wir können zum Event Handler weiter gehen.

# Der Mongoose Event Handler:

**void** **ev\_handler**(**struct** mg\_connection \*nc, **int** ev, **void** \*ev\_data);

## input Erklärung:

* **struct mg\_connection \*nc**:
* Dies ist ein Zeiger auf die mg\_connection Struktur, die die Verbindung darstellt, die das Ereignis ausgelöst hat.
* Diese Struktur enthält Informationen über die Verbindung, wie z.B. den Socket, den Verbindungsstatus und benutzerdefinierte Daten, die an die Verbindung angehängt werden können.
* Über diesen Zeiger können Sie auf Details der aktuellen Verbindung zugreifen und mit der Verbindung interagieren (z.B. Daten senden oder empfangen).
* **int ev**:
* Dies ist ein ganzzahliger Wert, der den Ereignistyp darstellt.
* Mongoose verwendet vordefinierte Konstanten für verschiedene Ereignistypen (z.B. MG\_EV\_ACCEPT für neue Verbindungen, MG\_EV\_HTTP\_REQUEST für HTTP-Anfragen).
* Dieser Wert ermöglicht es dem Event Handler, zu bestimmen, welche Art von Ereignis aufgetreten ist, und entsprechend zu reagieren.
* **void \*ev data:**
* Dies ist ein Zeiger auf ereignisspezifische Daten, die zusätzliche Informationen über das Ereignis enthalten.
* Der genaue Typ und Inhalt der Daten hängt vom Ereignistyp (ev) ab. Beispielsweise kann es sich um eine HTTP-Anfrage, eine WebSocket-Nachricht oder einen anderen Datentyp handeln.
* Dieser Zeiger muss in den richtigen Typ umgewandelt werden, um die spezifischen Ereignisdaten zu nutzen.

Die Funktion ist in der main.c datei schon als template implementiert erstelle hier jetzt ein switch case der die verschiedene events handeln soll (die verschiedenen Gruppen).

Benutze den übergabeparameter int ev um die Events zu handeln.

Wir wollen in unserem Fall

MG\_EV\_ACCEPT,

MG\_EV\_HTTP\_REQUEST und

MG\_EV\_CLOSE

Handeln.

# Implementierung des MG\_EV\_ACCEPT-Cases:

Um diesen Case in einem Mongoose Event-Handler zu implementieren, folgen Sie diesen Schritten:

1. **Deklarieren Sie einen Buffer für die Adresse:**
   * Dies dient zum Speichern der String-Repräsentation der IP-Adresse und des Ports.
   * Verwende 32 als Arraygröße.
2. **Verwenden Sie die mg\_sock\_addr\_to\_str-Funktion:**
   * Diese Funktion wandelt die Adresse der eingehenden Verbindung in einen String um.
3. **Geben Sie die Verbindungsinformationen aus:**
   * Nutzen Sie eine Funktion wie UARTprintf, um die Informationen der neuen Verbindung anzuzeigen.

Tipps:

**int** mg\_sock\_addr\_to\_str(**const** **union** socket\_address \*sa, **char** \*buf, size\_t len,

**int** flags)

* Das ist die deklaration in der mongoose.c  
  die socket adresse bekommen wir durch die mg\_connection struktur hierbei kann man auf die adresse wie folgt zugreifen:

&nc->sa

* Der buffer ist das erstellte array
* und für size\_t len sollte man die länge des buffers übergeben das kann mit der funktion

sizeof(„Dein erstelltes array“) machen.

* Als letztes können flags übergeben werden um der Funktion zu sagen wie es die adresse speichern soll: wir wollen IP und Port speichern also übergeben wir:

MG\_SOCK\_STRINGIFY\_IP | MG\_SOCK\_STRINGIFY\_PORT

In UARTPrintf kann man mit %s und dann , „pointer to array“ eine variable übergeben werden

Vergiss am Ende des Case nicht das **break**;

# Implementierung des MG\_EV\_HTTP\_REQUEST-Cases:

Hierfür benötigen wir jetzt eine Struktur für die Informationsverwaltung der http Anfrage

**struct** http\_message \*hm = (**struct** http\_message \*) ev\_data;

definiere wieder eine Adresse für die Speicherung der ip Adresse hierfür kannst du einfach den code aus MG\_EV\_ACCEPT kopieren.

Über UART kannst du jetzt ausgeben dass eine http anfrage eingegangen ist

Jetzt können wir ein if else Konstrukt aufbauen um auf verschiedene URI anfragen zu reagieren

Hierfür erstellen wir ein if Statement und in das argument kann die funktion gegeben werden

**int** mg\_vcmp(**const** **struct** mg\_str \*str1, **const** **char** \*str2);

Der erste Übergabe Parameter bekommen wir aus unserer erstellten: http\_message \*hm

hierbei kann auf die URI folgendermaßen zugegriffen werden: &hm->uri

als zweiten Übergabe Parameter übergibt man einen String mit der URI auf die reagiert werden soll: "/YourURI"

Die Funktion gibt 0 zurück, wenn die beiden übergebenen Parameter gleich sind also vergleichen wir mit == 0.

Was ihr genau in diesem if-Block schreibt ist für jede Gruppe individuell und wird weiter unten für jede Gruppe einzeln erklärt.

Was für jede Gruppe jedoch gleich bleibt, ist ein else Block am Ende in diesem else Block handeln wir alle Anfragen auf die wir keine spezielle Antwort senden wollen.

Wenn wir keine spezielle Antwort haben, dann senden wir den Error Code 404 zurück

In mongoose gibt es dafür Funktionen die das senden vereinfachen

**void** mg\_send\_response\_line(**struct** mg\_connection \*nc, **int** status\_code,

**const** **char** \*extra\_headers)

* mg\_send\_response\_line(nc, 404, "Content-Type: text/html\r\nConnection: close");

**int** mg\_printf(**struct** mg\_connection \*conn, **const** **char** \*fmt, ...)

* mg\_printf(nc, "\r\n<h1>Nicht gefunden</h1>\r\n");

kopiert diese beiden Funktionen in den else Block um einen Error Code zu senden und dem Benutzer anzuzeigen dass die Seite nicht gefunden wurde.

Als letztes in dem Case MG\_EV\_HTTP\_REQUEST müssen wir mongoose mitteilen, dass die Daten, die wir vorher durch die Funktionen in den if Statements zugewiesen haben gesendet werden sollen und die Verbindung geschlossen werden soll.   
Dies geht mit: nc->flags |= MG\_F\_SEND\_AND\_CLOSE;

Zum Schluss fügt noch diese beiden Zeilen hinzu damit wir visuell sehen wann es zu einer http request kommt

led\_value ^= CLP\_D1;

**LEDWrite**(CLP\_D1, led\_value);

vergesst auch hier nicht wieder das **break**; am ende des Case

# Implementierung des MG\_EV\_CLOSE-Cases:

Da wir gerade die Flag für connection close gesetzt haben kommen wir zum letztes Case in dem Event Handler MG\_EV\_CLOSE

In diesem Case geben wir einfach per UART aus, dass die Verbindung geschlossen wurde.

**UARTprintf**("Verbindung geschlossen\n");

**break**;

Jetzt sind wir fertig mit dem mongoose Event Handler und ihr könnt jetzt eure spezifischen Gruppenfunktionen implementieren.

# Gruppe 1

Ihr programmiert eine Reaktion auf Get für die URI /button

Fügt einen if block in euren switch case hinzu der auf die URI /button reagiert

(„/button“ = : "/YourURI")

Erstellt ein **const** **char** \*response;

Ruft dann eure gruppenfunktion auf : funktion\_gruppe1(&response);

Die Funktionen der Gruppen wurden von uns gemacht hierbei setzen wir die Werte für die Antwort. In eurer Funktion wird keine html in response geschrieben, sondern nur reiner Text.

Um jetzt mongoose zu sagen was man senden will kann man die funktion mg\_send(); verwenden.

Bevor wir aber diese Funktion verwenden müssen wir noch einen header senden. Das geht mit

**void** mg\_send\_head(**struct** mg\_connection \*c, **int** status\_code,

int64\_t content\_length, **const** **char** \*extra\_headers)

Überlege welche parameter übergeben werden müssen und rufe die funktion nach deiner Gruppenfunktion auf.

Tips:

* statuscode =200 , übergebe als extra headers: "Content-Type: text/plain"
* Jetzt rufe diese Funktion auf und übergebe die Parameter:

**void** mg\_send(**struct** mg\_connection \*nc, **const** **void** \*buf, **int** len)

Zusatz: bei **const** **void** \*buf, muss die response übergeben werden.

Ergänzend müsst ihr euer display noch in der while Schleife in der int main blinken lassen

Ihr könnt diese beiden Funktionen verwenden (mit anderen Farben) um ein blinken auf dem Display des Microcontrollers zu erreichen.

CFAF128128B0145T\_clear();

CFAF128128B0145T\_clear();

# Gruppe 2

Ihr programmiert eine Reaktion auf Get für die URI /blink und /blink/%d wobei %d = 0 1 2

Fügt einen if block in euren switch case hinzu der auf die URI /blink reagiert

(„/blink“ = : "/YourURI")

Erstellt ein **const** **char** \*response;

Ruft dann eure gruppenfunktion auf mit funktion\_gruppe2(&response,5,sys\_clk);

Die Funktionen der Gruppen wurden von uns gemacht hierbei setzen wir die Werte für die Antwort. In eurer Funktion wird keine html in response geschrieben, sondern nur reiner Text.

Um jetzt mongoose zu sagen was man senden will kann man die funktion mg\_send(); verwenden.

Bevor wir aber diese Funktion verwenden müssen wir noch einen header senden. Das geht mit

**void** mg\_send\_head(**struct** mg\_connection \*c, **int** status\_code,

int64\_t content\_length, **const** **char** \*extra\_headers)

Überlege welche parameter übergeben werden müssen und rufe die funktion nach deiner Gruppenfunktion auf.

Tips:

* statuscode =200 , übergebe als extra headers: "Content-Type: text/plain"
* Jetzt rufe diese Funktion auf und übergebe die Parameter:

**void** mg\_send(**struct** mg\_connection \*nc, **const** **void** \*buf, **int** len)

Zusatz: bei **const** **void** \*buf, muss die response übergeben werden.

Füge jetzt noch einen esle if block hinzu der auf /blink/%d reagiert

Hierbei verwenden wir jetzt die sscanf funktion um auf %d zu scannen

**sscanf**(hm->uri.p, "/blink/%d", &index)

so kannst du den index auf den wert von %d setzten

die Funktion gibt 1 zurück wenn die uri gleich ist

jetzt erstelle wieder einen char

**const** **char** \*response;

Rufe deine Gruppenfunktion auf jedoch dieses mal mit dem index als übergabeparameter.

funktion\_gruppe2(&response,index,sys\_clk);

zuletzt machst du das gleiche wie bei /blink 🡪 mg\_send\_head und mg\_send

# Gruppe3

Ihr programmiert eine Reaktion auf Get für die URI /display

Fügt einen if block in euren switch case hinzu der auf die URI /display reagiert

(„/display“ = : "/YourURI")

Erstellt ein **const** **char** \*response;

Ruft dann eure gruppenfunktion auf : funktion\_gruppe3(&response);

Die Funktionen der Gruppen wurden von uns gemacht hierbei setzen wir die Werte für die Antwort. In eurer Funktion wird keine html in response geschrieben, sondern nur reiner Text.

Um jetzt mongoose zu sagen was man senden will kann man die funktion mg\_send(); verwenden.

Bevor wir aber diese Funktion verwenden müssen wir noch einen header senden. Das geht mit

**void** mg\_send\_head(**struct** mg\_connection \*c, **int** status\_code,

int64\_t content\_length, **const** **char** \*extra\_headers)

Überlege welche parameter übergeben werden müssen und rufe die funktion nach deiner Gruppenfunktion auf.

Tips:

* statuscode =200 , übergebe als extra headers: "Content-Type: text/plain"
* Jetzt rufe diese Funktion auf und übergebe die Parameter:

**void** mg\_send(**struct** mg\_connection \*nc, **const** **void** \*buf, **int** len)

Zusatz: bei **const** **void** \*buf, muss die response übergeben werden.

# Gruppe 4

Ihr programmiert eine Reaktion auf Get für die URI /tone und /getmyparty

Fügt einen if block in euren switch case hinzu der auf die URI /tone reagiert

(„/tone“ = : "/YourURI")

Erstellt ein **const** **char** \*response;

Ruft dann eure gruppenfunktion auf : funktion\_gruppe4(&response);

Die Funktionen der Gruppen wurden von uns gemacht hierbei setzen wir die Werte für die Antwort. In eurer Funktion wird keine html in response geschrieben, sondern nur reiner Text.

Um jetzt mongoose zu sagen was man senden will kann man die funktion mg\_send(); verwenden.

Bevor wir aber diese Funktion verwenden müssen wir noch einen header senden. Das geht mit

**void** mg\_send\_head(**struct** mg\_connection \*c, **int** status\_code,

int64\_t content\_length, **const** **char** \*extra\_headers)

Überlege welche parameter übergeben werden müssen und rufe die funktion nach deiner Gruppenfunktion auf.

Tips:

* statuscode =200 , übergebe als extra headers: "Content-Type: text/plain"
* Jetzt rufe diese Funktion auf und übergebe die Parameter:

**void** mg\_send(**struct** mg\_connection \*nc, **const** **void** \*buf, **int** len)

Zusatz: bei **const** **void** \*buf, muss die response übergeben werden.

Für den Block für /getmypart benutze diesen else if block:

**else** **if** (mg\_vcmp(&hm->uri, "/getmypart") == 0) {

**char** \*html\_response;

**int** response\_length = mg\_asprintf(&html\_response, 0,

"<html><body><h1>Responses Ziffer und Index</h1>"

"<h2>Ziffer</h2><p>%s</p>"

"<h2>Index</h2><p>%s</p>"

"<p>Um weitere Informationen zu erhalten, finde den Server, der auf <a href=\"/display\">/display</a> reagiert.</p>"

"</body></html>",

responses\_ziffer[4], responses\_index[4]);

mg\_send\_head(nc, 200, response\_length, "Content-Type: text/html");

mg\_send(nc, html\_response, response\_length);

**free**(html\_response);

}

# Gruppe 5

Ihr programmiert eine Reaktion auf Get für die URI /finish

Fügt einen if block in euren switch case hinzu der auf die URI /init reagiert

(„/finish“ = : "/YourURI")

Erstellt ein **const** **char** \*response;

Ruft dann eure gruppenfunktion auf : funktion\_gruppe5(&response);

Die Funktionen der Gruppen wurden von uns gemacht hierbei setzen wir die Werte für die Antwort. In eurer Funktion wird keine html in response geschrieben, sondern nur reiner Text.

Um jetzt mongoose zu sagen was man senden will kann man die funktion mg\_send(); verwenden.

Bevor wir aber diese Funktion verwenden müssen wir noch einen header senden. Das geht mit

**void** mg\_send\_head(**struct** mg\_connection \*c, **int** status\_code,

int64\_t content\_length, **const** **char** \*extra\_headers)

Überlege welche parameter übergeben werden müssen und rufe die funktion nach deiner Gruppenfunktion auf.

Tips:

* statuscode =200 , übergebe als extra headers: "Content-Type: text/plain"
* Jetzt rufe diese Funktion auf und übergebe die Parameter:

**void** mg\_send(**struct** mg\_connection \*nc, **const** **void** \*buf, **int** len)

Zusatz: bei **const** **void** \*buf, muss die response übergeben werden.

# Gruppe 6

Ihr programmiert eine Reaktion auf Get für die URI /init

Fügt einen if block in euren switch case hinzu der auf die URI /init reagiert

(„/init“ = : "/YourURI")

Erstellt ein **const** **char** \*response;

Ruft dann eure gruppenfunktion auf : funktion\_gruppe6(&response);

Die Funktionen der Gruppen wurden von uns gemacht hierbei setzen wir die Werte für die Antwort. In eurer Funktion wird keine html in response geschrieben, sondern nur reiner Text.

Um jetzt mongoose zu sagen was man senden will kann man die funktion mg\_send(); verwenden.

Bevor wir aber diese Funktion verwenden müssen wir noch einen header senden. Das geht mit

**void** mg\_send\_head(**struct** mg\_connection \*c, **int** status\_code,

int64\_t content\_length, **const** **char** \*extra\_headers)

Überlege welche parameter übergeben werden müssen und rufe die funktion nach deiner Gruppenfunktion auf.

Tips:

* statuscode =200 , übergebe als extra headers: "Content-Type: text/plain"
* Jetzt rufe diese Funktion auf und übergebe die Parameter:

**void** mg\_send(**struct** mg\_connection \*nc, **const** **void** \*buf, **int** len)

Zusatz: bei **const** **void** \*buf, muss die response übergeben werden.

# Gruppe 7

Ihr programmiert eine Reaktion auf Get für die URI /snake

Fügt folgedne Funktion zu eurer main.c datei hinzu:

**int generate\_snake\_response(char** \*\*response) {

**return** mg\_asprintf(response, 0, snake\_html\_template, responses\_index[1], responses\_ziffer[1]);

}

Fügt einen if block in euren switch case hinzu der auf die URI /init reagiert

(„/snake“ = : "/YourURI")

Erstellt ein **char** \*response;

Rufe dann diese Funktion auf:

**int** snake\_response\_length = generate\_snake\_response(&response);

Die Funktionen der Gruppen wurden von uns gemacht hierbei setzen wir die Werte für die Antwort. In eurer Funktion wird Html skript in response geschrieben

Um jetzt mongoose zu sagen was man senden will kann man die funktion mg\_send(); verwenden.

Bevor wir aber diese Funktion verwenden müssen wir noch einen header senden. Das geht mit

**void** mg\_send\_head(**struct** mg\_connection \*c, **int** status\_code,

int64\_t content\_length, **const** **char** \*extra\_headers)

Überlege welche parameter übergeben werden müssen und rufe die funktion nach deiner Gruppenfunktion auf.

Tips:

* statuscode =200 , übergebe als extra headers: "Content-Type: text/html"
* Jetzt rufe diese Funktion auf und übergebe die Parameter:

**void** mg\_send(**struct** mg\_connection \*nc, **const** **void** \*buf, **int** len)

Zusatz: bei **const** **void** \*buf, muss die response übergeben werden.

# Passwort erraten:

Lade das Python-Skript von git hub herunter und installiere die benötigten Bibliotheken

Führe das Python Skript aus und gebe deine Get-Anfragen ein.

git clone <https://github.com/F1103BR/VS_Python_Tool_RestAPI.git>

Startet eure Passwortsuche mit http://<ipaddr>/init

ipaddr müsst ihr selbst herausfinden welcher Server auf diese Anfrage keinen Error zurückgibt

Das Passwort wird vom Server immer so gesendet: Ziffer: %s Index:%d

Wobei dann Ziffer ein Teil des Passwortes ist und Index die Stelle an der diese Ziffer steht.