

Aktuelle Entwicklungen im Bereich
Onlinemedien

Webentwicklung mit Go

Simon Storl-Schulke

Medieninformatik Bachelor, 251669

Dozent: Prof. Dr. Dirk Eisenbiegler

Inhaltsverzeichnis

Inhaltsverzeichnis	II
Abbildungsverzeichnis	III
1 Überblick	1
1.1 Einleitung	1
1.1.1 Entstehung	1
1.1.2 Warum Go?	1
2 Sprachkonzepte	3
2.1 Objektorientierte Programmierung [7]	3
2.1.1 Kapselung	3
2.1.2 Funktionen, Methoden und Closures	3
2.1.3 Interfaces	5
2.2 Exception Handling	5
3 Umgesetzte Webanwendung	7
3.1 Zielformulierung	7
3.2 Entwicklungsumgebung	7
3.3 Benutzeroberfläche und Frontend	7
3.3.1 Hauptseite	7
3.3.2 Registrierung und Anmeldung	9
3.3.3 Abgabe	9
3.4 Frontend	11
3.5 Backend	12
3.5.1 Programmstruktur	12
3.5.2 Handlerfunktionen und Routing [9]	12
3.5.3 HTML Templates [13]	13
3.5.4 Dateiverwaltung	15
3.5.5 JSON und Structs [11]	16

3.6	Konfiguration	17
3.6.1	Benutzer Authentifikation	18
3.7	Externe Bibliotheken	20
3.7.1	Blackfriday Markdown Processor [2]	20
3.7.2	Passwort Hasing	20
3.7.3	gorilla/mux Router [12].	22
3.8	Fazit	23
	Literaturverzeichnis	25

Abbildungsverzeichnis

Abbildung 1: Das Maskottchen der Programmiersprache - "Go Gopher"[8]	1
Abbildung 2: Closure	4
Abbildung 3: Interface	5
Abbildung 4: Hauptseite	8
Abbildung 5: Registrierung	9
Abbildung 6: Abgabeformular	10
Abbildung 7: Vereinfachte Darstellung des Backends	12
Abbildung 8: Handlerfunktion	13
Abbildung 9: Template System	13
Abbildung 10: JSON Umwandlung (Userstruct vereinfacht dargestellt) .	17
Abbildung 11: Blackfriday Markdown Parser	20

1 Überblick

1.1 Einleitung

1.1.1 Entstehung

Go ist eine quelloffene systemnahe Programmiersprache, die von der Firma Google entwickelt wurde und 2012 in die erste stabile Version kam. Das grundlegende Sprachdesign wurde 2007 von Robert Griesemer (Programmierer bei Google), Rob Pike und Ken Thompson entwickelt [15]. Letztere beiden sind unter anderem für die Entwicklung von UTF-8 und maßgebliche Beteiligung an der Entwicklung von Unix bekannt, sowie Ken Thompson für seinen Einfluss auf die Entstehung der Programmiersprache C.

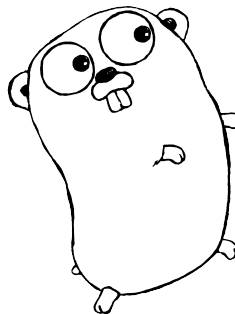


Abbildung 1: Das Maskottchen der Programmiersprache - "Go Gopher"[8]

1.1.2 Warum Go?

Im Folgenden einige Gründe, warum habe ich mich als Thema für diese Veranstaltung für die Webentwicklung mit Go entschieden habe und warum man sich als Webentwickler dafür entscheiden könnte.

- **Performance:** Aufgrund Go's systemnaher Natur, lässt sich sehr performanter Code schreiben. Zudem ist Nebenläufigkeit einfach umzusetzen.
- **Effizienter Compiler:** Der bereitgestellte Compiler ist Plattformunabhängig und um ein vielfaches schneller als die vergleichbaren Compiler anderer Sprachen. "...Go sollte schnell sein: Es sollte maximal ein

paar Sekunden brauchen eine große Programmdatei auf einem einzelnen Computer zu kompilieren.”[6]

- **Einfache Syntax:** Go hat eine sehr flache Lernkurve, was unter anderem an der in Einfachheit mit Python vergleichbaren Syntax liegt.
- **Flexible Programmstruktur:** Go’s Vererbung, Interfaces, Closures etc. erlauben eine sehr flexible Gestaltung des Codes.
- **Kontrolle** Die systemnahe Natur von Go erlaubt bei Bedarf genaue Kontrolle über den Code.
- **Mächtige Standardlibrary:** Die Standardlibrary beinhaltet die meisten zur Webentwicklung benötigten Werkzeuge und macht das benutzen eines Frameworks optional.
- **Bewährt:** Go wird von vielen großen Firmen, Webseiten und Anwendungen bereits eingesetzt u.a. Google, YouTube, SoundCloud, BBC, Docker, Canonical...
- **Aktive Entwicklung:** Go ist eine junge Sprache und wird kontinuierlich weiterentwickelt.

Im Vorfeld der Veranstaltung hatte ich mich schon kurz mit dem Thema Go befasst und einen einfachen Raytracer (3D Rendering Engine) geschrieben, die Kugeln und Dreiecke mit sehr einfachem Shading darstellen kann. Dabei habe ich jedoch kaum Go spezifische Sprachkonzepte verwendet und habe die Sprache nur oberflächlich kennengelernt und keine Features zur Webentwicklung verwendet. Go habe ich deshalb auch gewählt, um mich weiter in dieses Thema zu vertiefen.

2 Sprachkonzepte

2.1 Objektorientierte Programmierung [7]

Go bietet sowohl Werkzeuge für objektorientierte, als auch für funktionale Programmierung. Anders als strenger objektorientierte Sprachen, wie zum Beispiel Java und C#, verfügt Go nicht über herkömmliche Klassen. Das schließt objektorientierte Programmierung jedoch nicht aus. Stattdessen gibt es sogenannte `Structs`, die als Datentyp einsetzbar sind und für die sich auch Methoden definieren lassen. Structs bestehen lediglich aus einer Ansammlung von Variablen und sind dementsprechend leichtgewichtiger als vollständige Klassen. Structs können auch verschachtelt werden – sowohl mit Instanzen bereits bestehender-, als auch mit neu definierten Structs. Aufgrund des vergleichbaren Grundaufbaus, können Structs zum Beispiel problemlos direkt in JSON formatierte Strings umgewandelt werden (siehe 3.5.5).

2.1.1 Kapselung

Der Zugriffstyp wird anders als in den meisten anderen Sprachen nicht mit den Schlüsselwörtern `public` und `private` festgelegt. Stattdessen gelten alle großgeschriebenen als öffentlich und alle kleingeschriebenen Typen, Methoden und Funktionen als geschützt. Die Kapselung erfolgt nicht auf Ebene eines Structs, sondern innerhalb eines Paketes. Ein Paket kann eine beliebige Anzahl von Structs und Typen enthalten. Pakete lassen sich von anderen Paketen aus importieren. Geschützte Teile des Paketes sind von dort aus dann nicht aufrufbar, öffentliche Teile über die klassische Schreibweise `paketname.MethodeX()`.

2.1.2 Funktionen, Methoden und Closures

Funktionen und Methoden in Go erlauben neben den üblichen typischen Eigenschaften auch einige ungewöhnlichere Features, zum Beispiel:

- Beliebige Anzahl an Rückgabewerten- und Typen.
- Als Rückgabewerte und Parameter verwendbare Funktionen

- Closures (siehe unten)
- Annahme einer variablen Anzahl von Parametern.

2.1.2.1 Methoden

Da Structs in Go nur aus einer Anzahl von Werten bestehen, werden Methoden nicht in deren Rumpf definiert, sondern außerhalb. Nicht nur Structs, sondern jeder selbst definierte Typ kann Methoden haben. Um eine Funktion einem Typen als Methode zuzuordnen, wird dieser nach dem `func` Schlüsselwort definiert. Um Werte des Typs zu verändern, muss ein Pointer `*Typ` übergeben werden.

```
func (a *Auto) Schneller() {  
    a.Geschwindigkeit += 5  
}
```

Im Beispiel oben fügt die Methode `Schneller()` des Structs `Auto`, dessen Variable `Geschwindigkeit` 5 hinzu.

2.1.2.2 Closures [4]

Closures sind Funktionen, die Zugriff auf Variablen außerhalb ihres eigenen Funktionsrumpfes haben. So wie in diesem Beispiel (Abb. 2) kann eine Mutterfunktion als Rückgabewert eine Kindfunktion geben. Diese Kindfunktion hat dann beim Aufrufen der Mutterfunktion Zugriff auf die Variable `X` und erhöht diese um 1. Der Wert von `X` wird somit zur Laufzeit gespeichert.

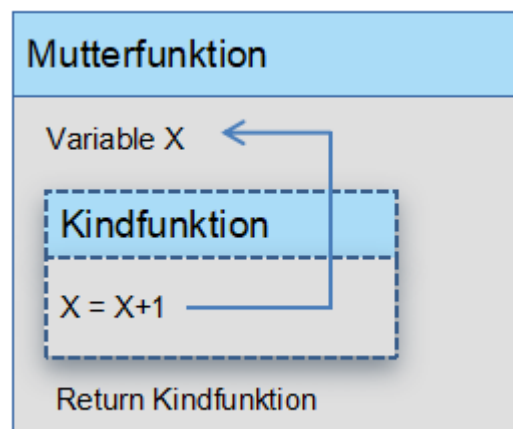


Abbildung 2: Closure

2.1.3 Interfaces

Interfaces in Go sind Strukturen, die aus einer Ansammlung von Methodenköpfen bestehen. Jedes Struct, das alle Methoden eines Interfaces mit den selben Parametern und Rückgabewerten implementiert, implementiert automatisch dieses Interface (Abb. 3). Anders als bei Java oder C#, muss dies nicht in die Deklaration des Structs geschrieben werden. Interfaces können genau wie Structs auch Methoden haben.

In Go wird häufig ein leeres `Interface{}` z.B. als Parameter oder Rückgabewert für Funktionen verwendet, um einen beliebigen Typen darzustellen (weil ein leeres Interface automatisch von allen vorhandenen Typen implementiert wird).

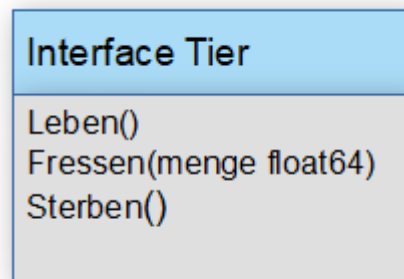


Abbildung 3: Interface

(Abb. 3) Das Interface `Tier` wird automatisch von allen Typen implementiert, die die darin definierten Methoden haben.

2.2 Exception Handling

Anstatt des üblichen Umgangs mit Exceptions mit `try` und `catch` gibt es in Go nur den eingebauten Typ `error`. Jede Funktion, die eventuell Fehler verursachen wird, erhält zusätzlich einen Rückgabewert vom Typ `error`. Die Fehlerbehandlung erfolgt dann nach einer `if error != nil` Abfrage.

```
func (a Auto) fahren(geschwindigkeit float32) error {  
    if a.Geschwindigkeit > geschwindigkeit {  
        fmt.Printf("Auto fährt")  
        return nil  
    }  
    return fmt.Errorf("Auto ist dafür zu langsam")  
}  
  
err := meinAuto.fahren(300)  
if err != nil {  
    log.Fatal(error)  
}
```

in diesem einfachen Beispiel wird davon ausgegangen, dass der Wert `geschwindigkeit` der Instanz `meinAuto` niedriger als 300 ist. Dementsprechend wird von der Funktion ein `error`objekt ausgegeben. `Fatal.log(err)` gibt nach der folgenden `if` Abfrage den in `err` gespeicherten Wert in der Konsole aus und beendet dann das Programm.

Error Handling in Go ist eines der kontroverseren Features der Sprache - es gibt sowohl passionierte Befürworter [5], als auch Kritiker [14] des Systems.

3 Umgesetzte Webanwendung

3.1 Zielformulierung

Ziel des Projektes war die Umsetzung einer konfigurierbaren Abgabepattform für Studierende mit moderner Benutzeroberfläche. Dozenten sollten Aufgabenformulierungen für ihre Veranstaltung veröffentlichen können, die dann von den Studierenden bearbeitet und von Tutoren und Dozent bewertet werden. Zudem sollte die Plattform konfigurierbar sein, um beispielsweise Kurse öffentlich oder geschlossen zu gestalten.

3.2 Entwicklungsumgebung

Als IDE für die Entwicklung der Anwendung wurde Visual Studio Code verwendet. Die dafür bereitgestellte Erweiterung für Go bietet alle Werkzeuge für die schnelle Programmierung in Go - Syntax Highlighting, automatische Code Formatierung und Imports, Intellisense, Error Highlighting und Debugging.

Das als Testserver verwendeten Rasperry Pi lief unter Ubuntu ohne Desktop Umgebung. Die Bedienung erfolgte ferngesteuert über den SSH Client PuTTY. Ein dafür angelegtes Makefile lädt mit einem Befehl die neuste Version der Webanwendung von GitHub, Kompiliert das Programm für Linux und startet den Server.

3.3 Benutzeroberfläche und Frontend

3.3.1 Hauptseite

Die Hauptseite der Webanwendung ist in vier Bereiche aufgeteilt (Abb. 4)

1. **Navigationsleiste:** Hier wird der Kursname angezeigt, sowie Schaltflächen für **Kursinfo** (öffnet im Anzeiger (4) Informationen zum Kurs des Dozenten), **Login/Logout/Registrieren** (Je nach Status des Nutzers) und **Post** (öffnet das Abgabeformular (Abb. 6))

2. **Seitenleiste:** Hier werden alle (oder bei geschlossenen Kursen der aktuell angemeldete) Nutzer angezeigt und können ausgewählt werden.
3. **Aufgabenwähler:** Auswahl der Aufgabe - wird `Alle` ausgewählt, werden alle Abgaben des in der Seitenleiste (3) ausgewählten Nutzers im Anzeiger (4) angezeigt.
4. **Anzeiger:** Zeigt Je nach ausgewähltem Nutzer und Aufgabe die Abgabe oder Aufgabenstellung an.

In Zukunft soll für angemeldete Tutoren und Admins unter dem Anzeiger noch ein Feedback- und Bewertungsbereich angelegt werden.

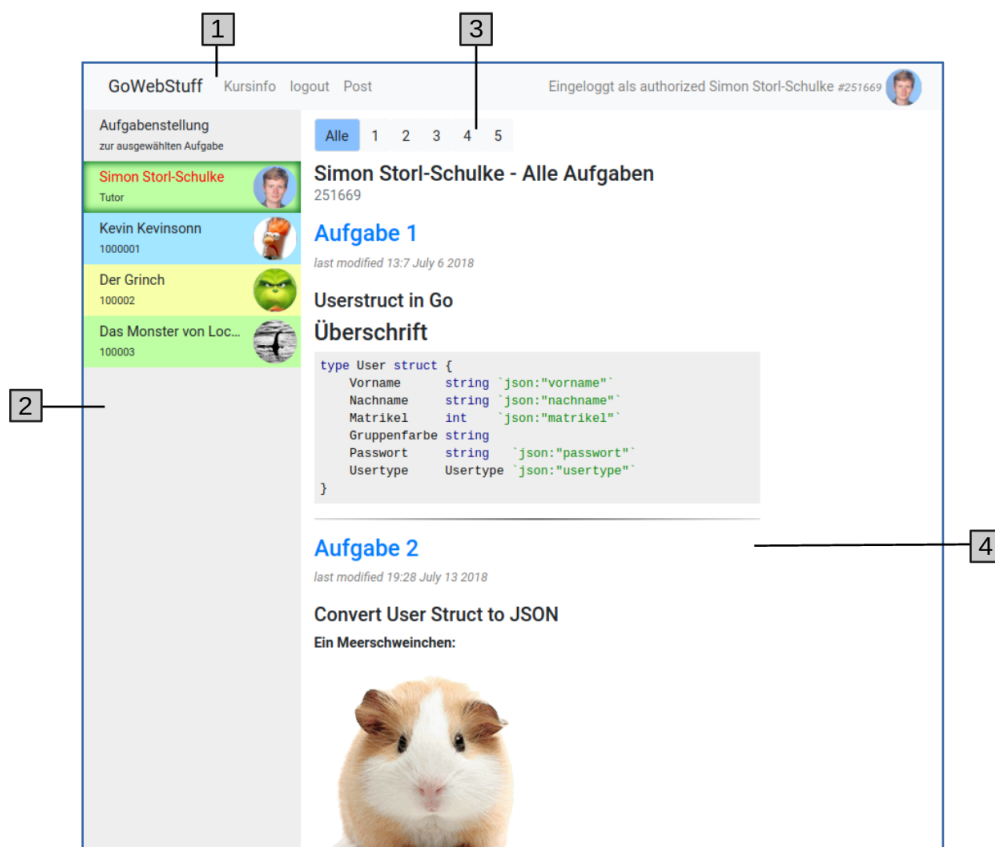
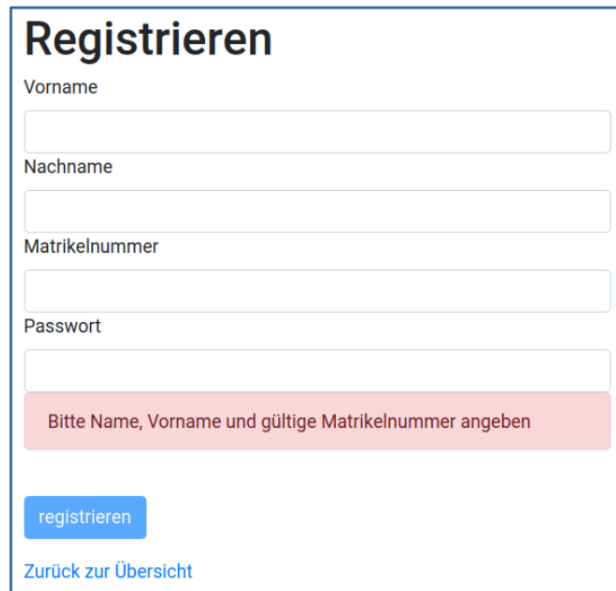


Abbildung 4: Hauptseite

3.3.2 Registrierung und Anmeldung

Die Registrierung (Abb. 5) und Anmeldung ist nach dem gängigen System aufgebaut. Anstelle des Benutzernamens erfolgt der Login jedoch mit Matrikelnummer und Passwort.



Registrieren

Vorname

Nachname

Matrikelnummer

Passwort

Bitte Name, Vorname und gültige Matrikelnummer angeben

[registrieren](#)

[Zurück zur Übersicht](#)

Abbildung 5: Registrierung

3.3.3 Abgabe

Über das Abgabeformular (Abb. 6) lassen sich Posts (Aufgabenstellungen und Abgaben) erstellen. Zudem ist es möglich Bestehende Aufgaben in den Textbereich zu laden um diese zu editieren. Auf der rechten Seite sind die wichtigsten Markdown Befehle aufgelistet um Studierenden die Nutzung zu vereinfachen.

In Zukunft wird hier eventuell noch eine Warnung eingebaut, falls für die gewählte Aufgabennummer schon ein Post existiert, um unfreiwilliges Überschreiben der eigenen Arbeit zu verhindern.

Abgabe

Simon Storl-Schulke 251669

[Zurück zur Übersicht](#)

Aufgabe Nr. [Bestehende Abgabe editieren](#)

Meine Abgabe

```
**User Konstruktor in Go**
```go
func NewUser(Vorname, Nachname string,
Matrikel int, Passwort string) User {
 return User{Vorname, Nachname, Matrikel, "",
Passwort, STUDENT}
}
```
```

Post

Zur Formatierung des Textes kann [Markdown](#) verwendet werden:

Überschriften:
Ich werde eine sehr dicke Überschrift
Ich mehr so mittel
eher kleiner
....

Textformat:
ich werde italic Text
ich werde dicker Text

Links & Bilder einbinden:
[klick mich](https://www.hs-furtwangen.de/)
![bild-titel](https://irgendein-Link-zu-irgendeinem-bild)

Listen
* Das hier werden

* Aufzählungen

1. Geht auch
2. Mit Nummern

Abbildung 6: Abgabeformular

3.4 Frontend

Das Frontend der Webanwendung wurde hauptsächlich mit reinem Javascript und Bootstrap 4 [3] umgesetzt. Kein weiteres Javascript Framework wurde verwendet, um das Projekt übersichtlich zu halten.

3.4.0.1 Styling

Beim Styling mit CSS gestaltete sich die Seitenleiste am kompliziertesten, da eine Lösung gefunden werden musste, Namen, Matrikelnummer und Portrait in den Nutzerschaltflächen auf wenig Platz - sowohl in der Desktop- als auch in der Mobilen Ansicht darzustellen.

3.4.0.2 Javascript

Javascript wurde eingesetzt, um die anzuzeigenden Inhalte im Anzeiger (Abb. 4) dynamisch via AJAX zu laden. Dabei werden die Inhalte aus dem URL-Pfad / [Matrikelnummer] / postraw / [Aufgabennummer] geladen. Diese werden in einer dafür geschriebenen Handlerfunktion (3.5.2) im Backend bereitgestellt.

Javascript wird zudem verwendet, um bei der Registrierung dynamisch anzuzeigen, wenn nicht genug Informationen eingegeben wurden (Abb. 5).

3.5 Backend

3.5.1 Programmstruktur

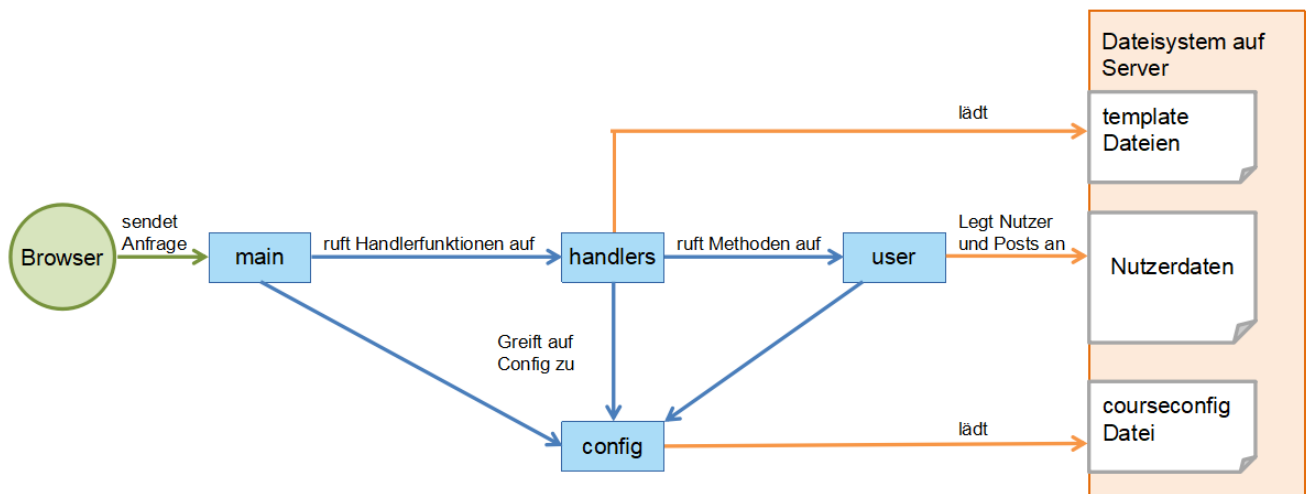


Abbildung 7: Vereinfachte Darstellung des Backends

Der Aufbau der Beispielanwendung ist aufgeteilt in vier Pakete (Abb. 7 blaue Kästen):

- **main:** beinhaltet die main Funktion.
- **handlers:** Handlerfunktionen für Haupt- und Unterseiten der Webseite.
- **user:** User Struct und Methoden für Registrierung, Posts, Lesen und schreiben in Ordnerstruktur etc.
- **config:** Config Struct und Methode zum Lesen der courseconfig.json Datei.

3.5.2 Handlerfunktionen und Routing [9]

Die Programmiersprache Go verfügt in der Standardlibrary über viele Werkzeuge zur Entwicklung von Webanwendungen. Den Kern bildet hierbei das Routing System. Hierbei wird für jede Seite der vordefinierten Funktion `http.HandleFunc` jeweils ein Pfad übergeben auf dem die entsprechende Seite erreichbar sein soll, sowie eine Handlerfunktion als Closure die als Parameter `http.ResponseWriter` und `*http.Request` empfängt. Innerhalb der Handlerfunktion kann die Webseite mit dem übergebenen `http.ResponseWriter` (hier `w`) beschrieben werden, zum Beispiel mit

`fmt.Fprint(w, "Hallo Welt")` Meist werden hierzu jedoch HTML-Templates (3.5.3) verwendet.

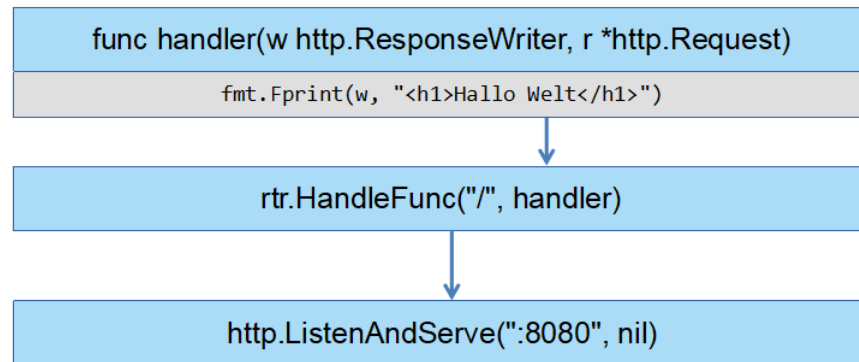


Abbildung 8: Handlerfunktion

Informationen über die Anfrage des Browsers - wie zum Beispiel gespeicherte Cookies (3.6.1) - werden über die übergebene `*http.Request` (hier `r`) gelesen.

Für eine Verbindung mittels https wird statt `ListenAndServe` die Funktion `ListenAndServeTLS` verwendet. Diese nimmt als zusätzliche Parameter noch Pfade zu Zertifikat- und Schlüsseldatei an.

3.5.3 HTML Templates [13]

Die Generierung von dynamischen Webseiten wird in Go hauptsächlich über sog. HTML-Templates (engl. Vorlagen) umgesetzt. Dabei wird eine HTML-formatierte Textdatei mit Schlüsselwörtern einer speziell dafür geeigneten "Template-Sprache" versehen, um zur Laufzeit Inhalte einzubinden.

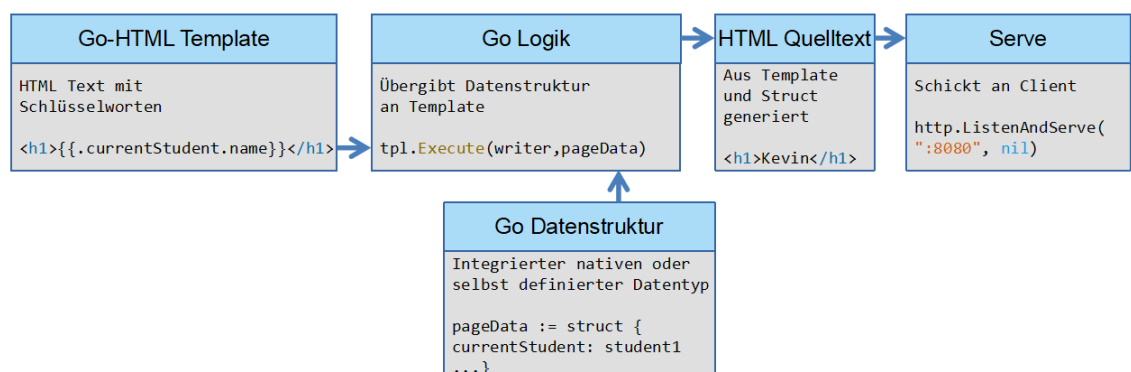


Abbildung 9: Template System

Das dem Nutzer anzuzeigende HTML Dokument wird dabei immer aus zwei Komponenten generiert: Eine Template Datei und eine beliebige Go-

Datenstruktur (meist Struct). Zunächst wird in der zur jeweiligen URL passenden Handlerfunktion die Funktion `template.Parsefiles` ausgeführt, mit der sich eine oder mehrere Textdateien zu eine Templateobjekten parsen lassen. Der Zugriff auf Elemente der Datenstruktur aus der Template Datei heraus erfolgt dann in doppelt geschweiften Klammern: Ein einfacher Punkt wird dann bei der Ausführung der Template zu dem übergebenen Datentyp in Textform. Auf verschachtelten Datenstrukturen wird mit Bezeichnung des gewünschten Kindelements nach dem Punkt zugegriffen. In diesem Beispiel (Abb. 9) wird zum Beispiel mit der Zeile

```
<h1>{{.currentStudent.name}}</h1>
```

auf das Kindelement `currentStudent.name` des übergeordneten Structs `pageData` zugegriffen und in eine HTML-Überschrift eingefügt. Der Ausführenden Methode `Execute` des Templates wird dabei die einzufügende Datenstruktur (hier `pageData`) und der `http.ResponseWriter` (aus der umschließenden Handlerfunktion) übergeben.

3.5.3.1 Template Logik

In der Templatesprache sind auch einfache Logikoperatoren möglich. So kann zum Beispiel mit dem Schlüsselwort `{{range .meinArray}}` über einen übergebenen Array iteriert werden. Der folgende HTML Code wird dann so oft wiederholt, wie sich Elemente im Array befinden. Das Schlüsselwort `{{.}}` wird in diesem Bereich dann zum aktuellen Element des Arrays, bis die Schleife mit `{{end}}` beendet wird. Diese Technik wird bei meinem Projekt zum Beispiel verwendet, um in der Seitenleiste alle registrierten Nutzer anzuzeigen (Abb. 4).

Zudem sind `if` Abfragen möglich:

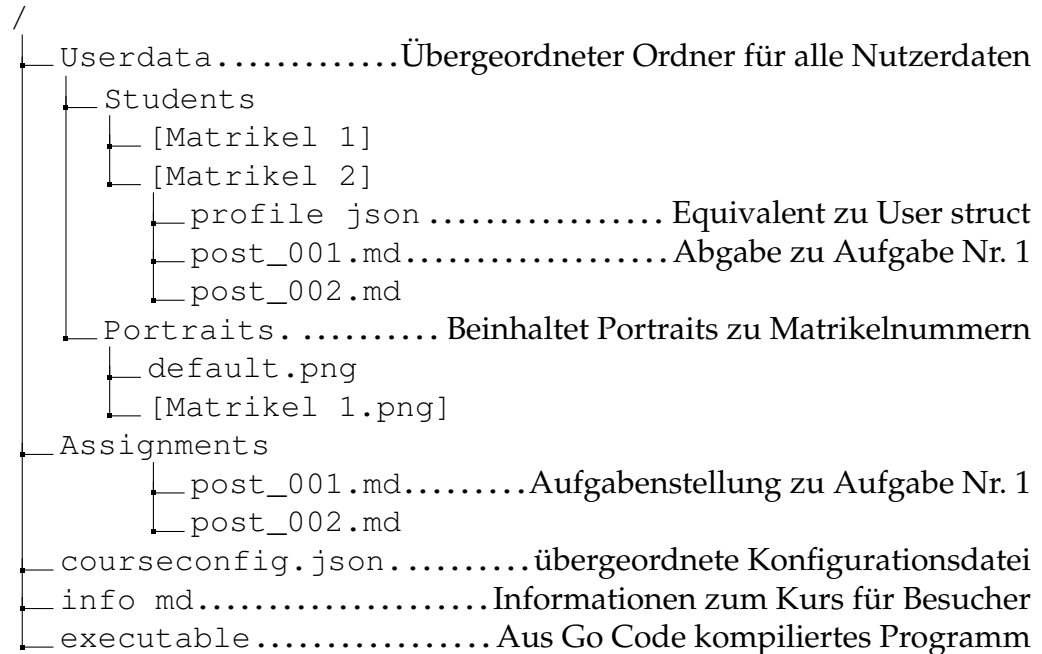
```
{{if .currentStudent.IsAuthorized}}  
<p>Nutzer ist authorisiert</p>  
{{else}}  
<p>Nutzer ist nicht authorisiert</p>  
{{end}}
```

Im oben stehenden Beispiel hat `currentStudent` eine boolsche Methode `isAuthorized`. Gibt diese `true` zurück, wird der erste Paragraf angezeigt, ansonsten der zweite. Viel komplexere `if` Abfragen sind in der Template-Sprache jedoch nicht möglich.

3.5.4 Dateiverwaltung

3.5.4.1 Ordnerstruktur

Die Nutzer- und Kursdaten werden auf dem Server in folgender Ordnerstruktur gespeichert:



Um das Programm leichtgewichtig und übersichtlich zu halten und aufgrund der einfachen Möglichkeiten in Go Dateien und Ordner zu laden und mit JSON Dateien zu arbeiten, habe ich mich für dieses einfache Dateisystem anstelle einer Datenbank (z.b. MongoDB) entschieden.

3.5.4.2 Dateien lesen und schreiben [10]

Für das Lesen und schreiben von Dateien wird in der Standardbibliothek das Paket `ioutil` bereitgestellt.

Das **Lesen** erfolgt über die Funktion `ReadFile(Dateiname)`, die einen `Bytearray` mit dem gelesenen Dateiinhalt, sowie ein `errorobjekt` zurückgibt (dass bei Erfolg den Wert `nil` hat). Diese Funktion wird in meinem Projekt zum Beispiel von der Method `GetPost(PostNummer)` des Usertyps verwendet, um den Inhalt eines Posts anhand gegebenem Nutzer und Nummer zu lesen.

Das **Schreiben** von Dateien erfolgt über die Funktion `WriteFile(Dateiname, Dateiinhalt, Zugriffsrechte)`, die als Rückgabewert um Fehler aufzufangen ein `errorobjekt` liefert, dass bei Erfolg `nil` ist. In meinem Projekt wird diese Funktion zum Beispiel genutzt um mit der Methode `PostNr(postinhalt,`

`postnummer`) des User Structs, Markdown-Dateien anhand einer Postnummer und des Inhalts in das Ordnersystem zu schreiben. Dabei wird zudem die gegebene Nummer in einen String mit drei Ziffern umgewandelt (5 -> 005) um das Sortieren der Dateien zu vereinfachen.

3.5.4.3 Fileserver

um Dateien auf dem Server bereitzustellen, wird die eingebaute Handlerfunktion `FileServer()` verwendet. Diese empfängt den Pfad zu einem Ordner auf dem Server. Mit der Funktion `http.Handle`, sind die enthaltenen Dateien dann unter der gewünschten URL erreichbar. In der im Projekt umgesetzten Webanwendung wird diese Funktion zum Beispiel verwendet, um eingebundene CSS und Javascript Dateien, sowie die Nutzerportraits bereitzustellen.

3.5.5 JSON und Structs [11]

Structs können in Go sehr leicht in JSON Dateien umgewandelt werden und umgekehrt. Für die Umwandlung von Struct zu Json, müssen die umzuwandelnden Variablen des Structs öffentlich sein (Großgeschrieben). Zusätzlich können in der Deklaration des Structs Namen für die zugehörigen JSON Werte festgelegt werden. Wird dies nicht getan, wird stattdessen der Name der entsprechenden Variablen selbst verwendet.

Für die Umwandlung werden die in der Standardbibliothek enthaltenen Funktionen `json.Marshal` und `json.Unmarshal` verwendet. `Marshal` empfängt einen beliebigen Typen - dargestellt durch ein leeres `interface{}` (ein leeres Interface wird automatisch von allen vorhandenen Typen implementiert). Als Rückgabewert liefert die Funktion einen Bytearray, der dann in eine Datei geschrieben werden kann, sowie ein Errorobjekt um Fehler abzufangen.

`Unmarshal` erhält einen (hier aus einer Datei gelesenen) Bytearray und einen beliebigen Typen (hier als Referenz zu einer Instanz des Structs `User` - `&us2`), der mit den den gelesenen Daten gefüllt wird.

In meiner umgesetzten Webanwendung wird dieses System für die Registrierung und das Auslesen von Nutzern verwendet. Deren Profile werden als JSON Dateien in der Ordnerstruktur gespeichert (siehe Abb. 7). Beim Aufruf der Hauptseite, werden alle Nutzer ausgelesen und in der Seitenleiste dargestellt (sofern der Kurs als öffentlich konfiguriert ist und der zugreifende Nutzer angemeldet ist). Zudem wird die Kurskonfiguration bei

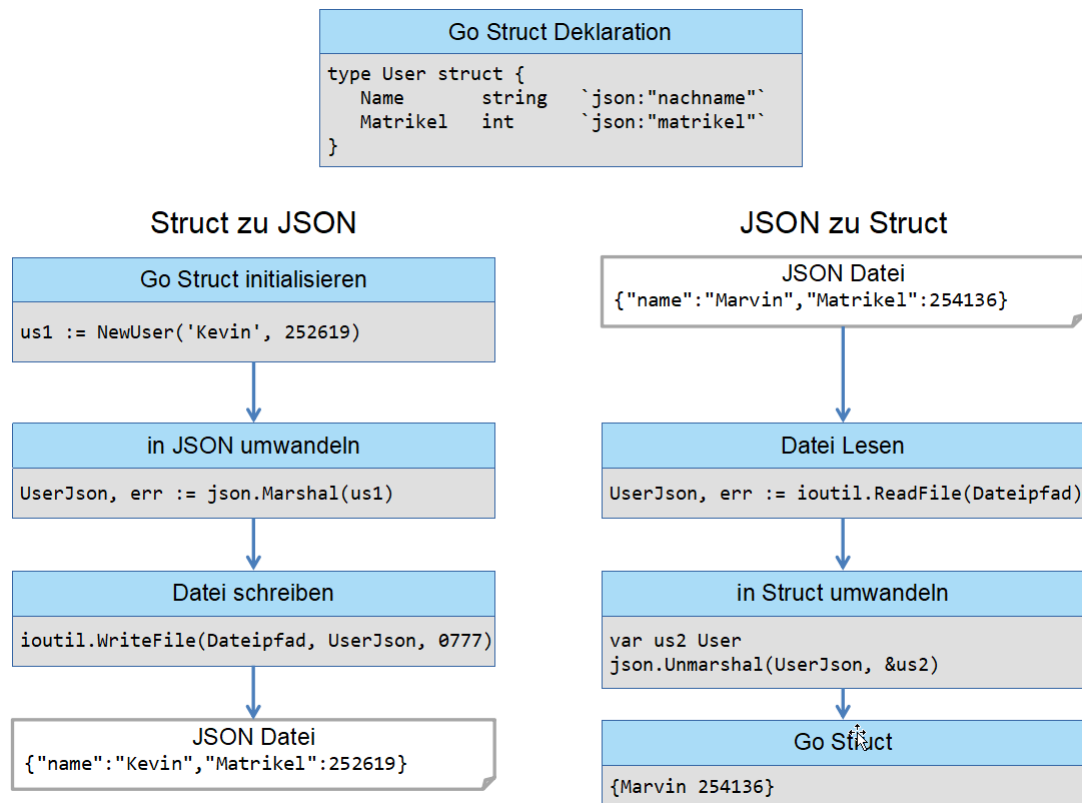


Abbildung 10: JSON Umwandlung (Userstruct vereinfacht dargestellt)

jedem Seitenaufruf aus einer JSON Datei gelesen.

3.6 Konfiguration

Die Konfiguration der Plattform erfolgt über die Datei `courseconfig.json`. Darin sind folgende Variablen enthalten:

- **port:** Bestimmt unter welchem Port auf dem Server die Anwendung läuft.
- **course_name:** Bestimmt den angezeigten Name des Kurses.
- **root_url:** Bestimmt den Pfad der URL.
- **group_number:** Bestimmt die Anzahl der Gruppen, in die die Kursteilnehmer aufgeteilt werden. Die Aufteilung dient zum Beispiel der Arbeitsteilung von Tutoren und Dozent bei der Abgabekontrolle.
- **open_course:** Bestimmt ob die Abgaben des gesamte Kurses für alle angemeldeten Benutzer sichtbar sind ist oder ob für Studierende jeweils nur das eigene Profil und Abgaben sichtbar sind.

- **tutors_can_post:** Bestimmt ob Tutoren auch als Profil auf der Kursseite angezeigt werden und Abgaben posten können.
- **master_password:** Passwort dass für die autorisierte Registrierung und Anmeldung von Tutoren und Dozenten benötigt wird.
- **classes:** Liste der teilnehmenden Studiengänge (MIB, MKB...)

Das Paket `config` enthält die Funktion `GetConfig`, die zur Laufzeit die JSON Datei ausliest und als Struct des Typen `Config` zurückgibt. Die Methode wird bei jedem Aufruf der Seite in der entsprechenden Handler-Funktion aufgerufen - dementsprechend muss der Server nicht neu gestartet werden um Änderungen vorzunehmen (ausgeschlossen davon sind natürlich die Optionen `port` und `root_url`).

Momentan ist die Konfiguration nur mit dem Editieren der JSON Datei auf dem Server möglich - später soll für diesen Zweck noch eine Benutzeroberfläche für den Administrator der Plattform gebaut werden. Zudem sind noch einige weitere Konfigurationsmöglichkeiten geplant, sowie die autorisierte Registrierung mittels Masterpasswort über die Webanwendung (momentan auch nur mit Editieren der jeweiligen `profile.json` Datei möglich)

3.6.1 Benutzer Authentifikation

3.6.1.1 Login

Für das Einloggen von Nutzern werden nach Vergleich der eingegebenen- und der auf dem Server gespeicherten Logininformationen (siehe 3.7.2) Cookies gesetzt, die jeweils Matrikelnummer und gehashtes Passwort beinhalten. Der im Paket `http` der Standardbibliothek enthaltene Typ `Cookie` besteht dabei wie das Browser-Equivalent aus einem Namen und einem Wert. Hier wurde als Name "Session" verwendet und als Wert Matrikel und Passwort. Der Cookie kann dann mit der Funktion `http.SetCookie(w, c)` gesetzt werden, die als Parameter den in der umschließenden Handlerfunktion übergebenen `http.ResponseWriter`, sowie den Cookie empfängt. Nach erfolgreichem Login wird der Nutzer mit der Funktion `http.Redirect` auf die Hauptseite weitergeleitet.

3.6.1.2 Session Check

Die Handlerfunktionen der Haupt- und Abgabeformular überprüfen zu jedem Seitenaufruf ob der Cookie "Session" mit gespeicherter Matrikelnum-

mer und Passwort übereinstimmt. Für diesen Zweck habe ich die Funktion `loggendIn(r) (bool, int)` geschrieben, die als Parameter die HTTP-Anfrage (hier `r`) erhält und als Rückgabe einen boolschen Wert, sowie bei Erfolg die Matrikelnummer des angemeldeten Nutzers liefert. Über den Befehl `r.Cookie("session")` lässt sich der Wert des Cookies aus einer HTTP-Anfrage (`r`) auslesen. Dieser Wert wird dann mit den gespeicherten Benutzerinformationen verglichen.

Ist die Überprüfung erfolgreich, wird der Nutzer auf der Hauptseite als Eingeloggt dargestellt und die Schaltflächen Post und Logout werden angezeigt. Versucht der Nutzer uneingeloggt auf das Abgabeformular zuzugreifen, wird er auf die Loginseite weitergeleitet.

3.7 Externe Bibliotheken

Bei der Umsetzung des Projektes habe ich, neben den Werkzeugen in der Standardbibliothek, zusätzlich drei externe Bibliotheken verwendet:

3.7.1 Blackfriday Markdown Processor [2]

Diese Bibliothek ermöglicht die Umwandlung von Markdown formatiertem Text zu html. Dies ermöglicht die einfache Formatierung von Aufgabenstellungen und abgegebenen Aufgaben.

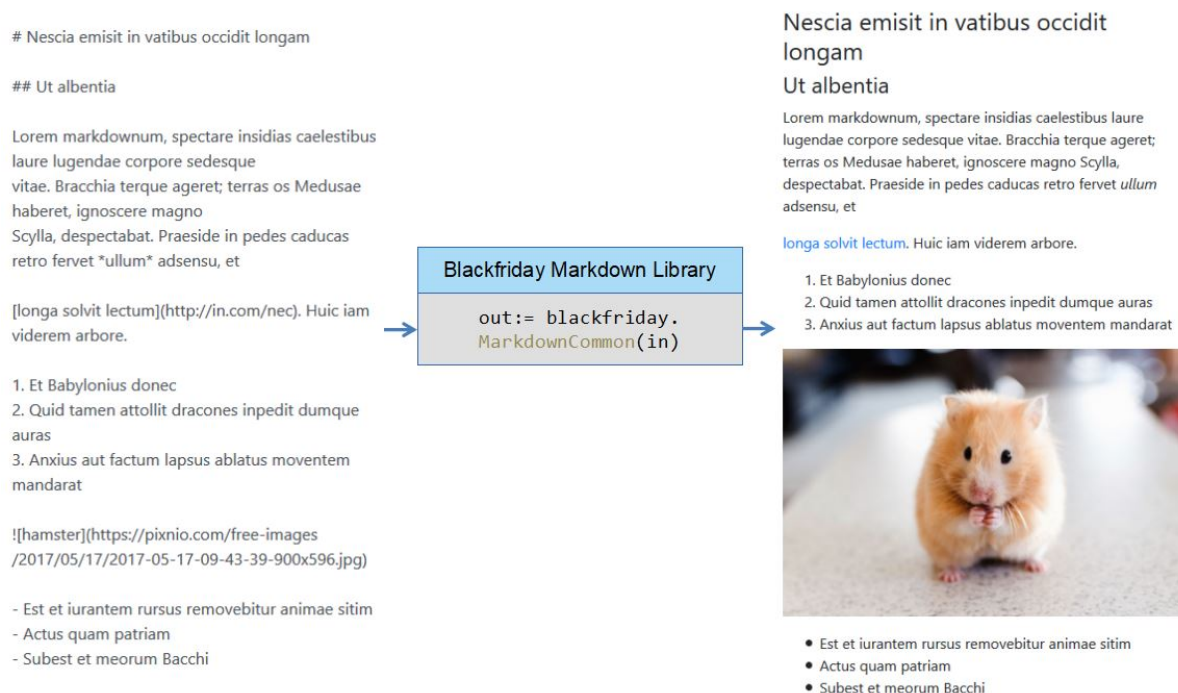


Abbildung 11: Blackfriday Markdown Parser

Mit der Zeile

```
out := blackfriday.MarkdownCommon(in)
```

wird in diesem Beispiel der html-formatierte String `out` aus dem Markdown formatiertem String `in` erzeugt. Auch Bilder lassen sich so problemlos einbinden.

3.7.2 Passwort Hasing

Für das Hashing gespeicherte Passwörter habe ich mich zuerst für den bcrypt Algorithmus entschieden [1], der als einer der Standards für das Hashen

von Passwörtern gilt. Da dieser auf dem Raspberry Pi, dass als Server zum Testen verwendet wurde jedoch viel zu langsam war und das Einloggen mehrere Minuten pro Nutzer beanspruchte, habe ich für diese Testversion zeitweise zu dem MD5 Algorithmus gewechselt. Dieser sollte heutzutage eigentlich nicht mehr zum Hashen von Passwörtern verwendet machen, unter anderem weil er Brut-Force-Angriffe erleichtert und weil es theoretisch möglich ist (wenn auch sehr unwahrscheinlich), dass verschiedene Passwörter denselben Hash erzeugen. Für die Raspberry Pi Testseite erfüllt er jedoch seinen Zweck. In Zukunft ist geplant, wieder auf den Bcrypt Algorithmus umzusteigen.

Für das Hashing mit MD5 bei der Registrierung wird zuerst eine Hash-Objektinstanz erzeugt:

```
hasher := md5.New()
```

Diese wird dann mit dem Passwort beschrieben:

```
hasher.Write([]byte(password))
```

Der resultierende Bytearray wird in einen String umgewandelt:

```
encodedPassword := hex.EncodeToString(hasher.Sum(nil))
```

Der resultierende String wird dann in der `profile.json` Datei des entsprechenden Nutzers gespeichert. Beim Einloggen wird der selbe Prozess mit dem eingegebenen Passwort des Nutzers ausgeführt und anschließend mit als gehasht gespeicherten Passwort verglichen.

Für den Das Passwort-Hashing mit Bcrypt gibt es eine vorgefertigte Funktion:

```
bcrypt.GenerateFromPassword([]byte(password), cost)
```

Wobei sich über die Integer `cost` die Komplexität des Algorithmus einstellen lässt.

3.7.3 gorilla/mux Router [12]

Diese Bibliothek diene mir dazu schnell und einfach Variablen aus der abgefragten URL auszulesen. Die Funktion `mux.Vars` empfängt hierbei den in der übergeordneten Handlerfunktion empfangenen Pointer zu einer http Anfrage (`*http.Request`) und liefert ein Assoziatives Datenfeld (map) der in der URL enthaltenen Parameter zurück. Folgendermaßen lässt sich beispielsweise die als URL-Parameter übergebene Variable `postnr` auslesen um die Nummer des anzuzeigenden Posts festzustellen.

```
postNr := mux.Vars(request) ["postNr"]
```

3.8 Fazit

Generell habe ich mich schnell in die Programmierung mit Go eingefunden. Aufgrund von Go's lösungsorientiertem Design, gab es kaum lange Zeiträume in denen ich mit einem Problem nicht weiterkam. Das Template-System erleichtert das Zusammenarbeiten von Front- und Backend, ist leicht verständlich und führt schnell zu gewünschten Ergebnissen. Tatsächlich war der frustrierendste Abschnitt des ganzen Projektes wohl nicht die Entwicklung mit Go selbst, sondern das Styling mit CSS.

Einige Features, die in anderen Sprachen nützliche Abkürzungen bieten habe ich in Go dennoch vermisst:

- Kein ternary Operator
- Kein Operator Overloading - was zum Beispiel für das rechnen mit Vektoren nützlich wäre.
- Exception Handling mit `if err != nil...` plustert code teilweise auf.
- `private` / `public` Kennzeichnung mit Groß- und Kleinschreibung spart zwar Code, ist aber teilweise gewöhnungsbedürftig. Zum Beispiel müssen Variablen im ganzen Code unbenannt werden um die Zugriffsart zu ändern (meist erledigt das natürlich die IDE)

Da ich insgesamt sehr überzeugt von den in Go bereitgestellten Werkzeugen zur Webentwicklung war, werde ich mich auf jeden Fall weiter mit dem Thema beschäftigen

Vielen Dank an Tobias Faller für die Bereitstellung eines Raspberry Pi als Testserver.

Literaturverzeichnis

- [1] bcrypt library. godoc.org/golang.org/x/crypto/bcrypt.
- [2] blackfriday markdown. github.com/russross/blackfriday.
- [3] Bootstrap. getbootstrap.com/.
- [4] closures. gobyexample.com/closures.
- [5] error handling in go. davidnix.io/post/errorhandlingingo/.
- [6] goFAQ New Language. golang.org/doc/faq#creating_a_new_language.
- [7] goFAQ OOP.
golang.org/doc/faq#Is_Go_an_objectoriented_language.
- [8] Gopher.
[upload.wikimedia.org/wikipedia/commons/
6/6f/Go_gopher_mascot_bw.png](https://upload.wikimedia.org/wikipedia/commons/6/6f/Go_gopher_mascot_bw.png).
- [9] http go dokumentation. golang.org/pkg/net/http/.
- [10] ioutil go dokumentation. golang.org/pkg/io/ioutil/.
- [11] json go dokumentation. golang.org/pkg/encoding/json/.
- [12] mux library. github.com/gorilla/mux.
- [13] templates go dokumentation. golang.org/pkg/html/template/.
- [14] why I dont like error handling in go. opencredo.com/why-i-dont-like-error-handling-in-go/.
- [15] Wikipedia Go. [wikipedia.org/wiki/Go_\(Programmiersprache\)](https://wikipedia.org/wiki/Go_(Programmiersprache)).