GO (golang): Schnelle & sichere Webanwendungen programmieren

Eine umfangreiche Einführung in die
GO Programmierung von Webanwendungen
für Einsteiger und fortgeschrittene Entwickler

Jens Schendel

Version 1.0.177, Oktober 2023

Inhaltsverzeichnis

Abschnitt 1 – Einführung

I don't know where I am going, but I am on my way..
Voltaire

Lektion 1 – Darf ich mich vorstellen? Das „whoami" für Udemy-Kurse

Willkommen zum Kurs „GO (golang): Schnelle und sichere Webanwendungen programmieren". Mein Name ist Jens Schendel, aber bekanntermaßen sind Namen ja nur Schall und Rauch und stellen eher das Äquivalenz zur Eingabe des Befehls whoami auf der bash oder einer anderen Linux Shell dar. Aber auch eine Shell gibt ja nur den Benutzernamen zurück, als den einen das System gerade erkennt. Daher lassen Sie mich kurz hier ein paar Worte über mich verlieren.

Ich bin von Hause aus gelernter Mediengestalter mit Fachrichtung Medien Design und Medien Operating, arbeite als Front-End Web Entwickler als auch als Linux Administrator und System Operator. Im letztgenannten Bereich habe in den letzten Jahren ein paar zusätzliche Zertifikate erworben. Wahrscheinlich haben Sie sich aber auch schon meine Dozentenseite auf Udemy oder mein LinkedIn-Profil angesehen.

Ich bin begeisterter Programmierer aus Leidenschaft, betreibe einen unbedeutenden englischsprachigen YouTube Channel für C-Programierung-bezogene Themen namens myCTalkthroughs. Als ich Googles noch relativ junge Programmiersprache GO für mich entdeckte, fand ich sofort großen Spaß an der Einfachheit, mit der man auch auch komplexer Themenbereiche programmiertechnisch bearbeiten kann. Und daraus entwickelte ein ebenso großer Spaß an der Vermittlung dieses Wissens in Form von Kursen auf Udemy einige in Deutsch, hauptsächlich aber in Englischer Sprache.

Meine Kurse (auch als kostenlose Tutorials) auf Udemy richten sich bisher eher an Anfänger, aber meine kleine ausgesuchte Gruppe von Studenten fragten eher dann auch Themen nach, die fortgeschrittene Fertigkeiten vermitteln. Dem komme ich in diesem Kurse gerne nach!

Und dabei belassen wir es mal mit der Selbstvorstellung, legen wir lieber mal los! Oder mit den Worten des großen amerikanischen Gegenwartsphilosophen Rick Sanchez:

„Wubba Lubba Dup Dup!" Halten Sie Ihre Portal Gun bereit - jetzt geht's los!

Lektion 2 – Warum ausgerechnet GO? Warum nicht Node.js oder eine andere Programmiersprache?

Ganz schnell gesagt: GO ist großartig.

Mit GO zu programmieren fühlt sich an, als hätte man es mit C zu tun, aber jemand hat alles entfernt, was sich wie als müsste man um den Schmerz herumarbeiten, und stattdessen ein paar Features hinzugefügt, die man ansonsten ohnehin selbst hätte programmieren müssen.

Wer hat's erfunden?

- Google (GO auf Wikipedia) namentlich:
- Rob Pike, Unix, UTF-8
- Robert Griesemer studierte bei dem Erfinder von Pascal, arbeitete an Googles V8 JavaScript engine
- Ken Thompson leitete die Implementierung von U**x, erfand die Programmiersprache B und damit den Vorgänger von C und war an der Entwicklung von C beteiligt.

Warum jetzt nochmal GO?

- Äußerst effiziente Kompilierung
- GO erzeugt kompilierte Programme
- Es gibt einen "Garbage Collector" (GC)
- Es gibt keine virtuelle Maschine, in der der Code ausgeführt wird, keine Ausführungsschicht oder Umgebung, keinen Emulator und keinen Interpreter
- Schnelle Ausführungszeiten
- Benutzerfreundlichkeit, "Ease of Programmierung".

Zusammenfassend kann man sagen, dass es drei Hauptmerkmale gibt, die GO so erfolgreich machen:

1. Einfache und sehr schnelle Kompilierung - selbst große Projekte werden in Sekunden und Minuten kompiliert, nicht in Stunden
2. Effiziente Ausführung mit sehr hoher Ausführungsgeschwindigkeit
3. Einfache Programmierung - Programmieren sollte einfach sein, nicht schmerzhaft für die Gehirnwindungen.

Brad Fitzpatrick hat dies 2014 in einigen Folien zusammengefasst und diese Folien sind auf go.dev verfügbar: https://go.dev/talks/2014/gocon-tokyo.slide#31

Wofür GO verwendet werden kann

- Einfach gesagt: Alles, "was Google macht" und alle Internetdienste, die höchsten Ansprüchen genügen und hoch skalierbar sein müssen.
- Networking/Vernetzung, Server-Client Programmierung
- http/https, tcp, udp
- Nebenläufigkeit/parallele Programmierung
- Systemprogrammierung
- Automatisierung, Befehlszeilentools
- Kryptographie/Ver- und Entschlüsselung
- Bildverarbeitung

Programmierprinzipien

- sinnvoll, verständlich, anspruchsvoll
- sauber, klar, leicht zu lesen

Unternehmen, die GO im Einsatz haben auch auf der Seite go.dev

> Google, YouTube, Netflix, Paypal, Meta, Microsoft, Twitter, Docker, Kubernetes, neben anderen auch InfluxDB, Twitter, Apple, Cloudflare, DropBox, viele andere mehr im Detail

Trivia:

> Der Erfinder von Node.js hat Node aufgegeben und sich stattdessen GO zugewandt
>
> GO-Programmierer sind derzeit die bestbezahlten Programmierer in den USA - 5. in der Welt (2017)

Lektion 3 – Installation von GO und Einrichtung einer kostenlosen Entwicklungsumgebung (IDE)

Die Installation und Einrichtung von GO ist heutzutage trivial und ich werde nur kurz darauf eingehen.

Kurzgesagt: Laden Sie GO für Ihr Betriebsystem und entsprechende Prozessorarchitektur herunter und und folgen Sie anschließend den Installationsanweisungen.

GO bringt seinen eigenen Compiler bereits mit, aber ein gutes IDE hat viele Vorteile. Sehr bekannt sind LiteIDE, GOSublime und GOland. Leider sind sie meines Wissens nicht kostenlos. Natürlich kann man auch mit einem guten Texteditor wie Atom, VIM oder Notepad++ und ein paar Erweiterungen durchaus seine eigene Entwicklungsumgebung bauen, die Features wie Syntax-Hervorhebung und automatische Compilierung und vieles mehr unterstützt. Doch für einen Kurs wie diesen empfehle ich Visual Studio Code. Es ist kostenlos, erfüllt für mich alle Anforderungen an ein gutes IDE und steht für Linux, macOS und Windows bereit.

Die Installation läuft wieder auf Herunterladen und Installieren hinaus. Fertig.

Bitte beachten Sie, dass Ihre Installation bestimmt etwas besser aussieht, ich habe hier ziemlich große Schriftarten eingestellt, damit Sie auch auf einem mobilen Endgerät den Code in den Videos noch gut lesen können.

Jetzt installieren wir noch eine Extension für GO, der wir vielleicht vertrauen müssen. Dazu öffnen wir die Voreinstellungen (hier Preferences) und dort die Erweiterungen (hier Extensions) und suchen nach GO. Die installieren wir. Eventuell müssen Sie Visual Studio Code neu starten. Wer will, kann noch die Github-Unterstützung für Visual Studio Code installieren. Danach sind wir startklar. Das war einfach, oder?

Falls irgendetwas mit der Installation von GO, bzw., von Visual Studio Code nicht geklappt hat, verweise ich auf die Webseiten go.dev und code.visualstudio.com. Ein häufiges Problem ist die fehlende Angabe einer sogenannten Umgebungsvariablen, damit Ihr Benutzer, bzw., Ihr Betriebssystem GO überhaupt finden kann. Das Internet sollte Ihnen weiterhelfen.

Lektion 4 – Lernhinweise zu diesem Kurs

Um das Beste aus diesem Kurs herauszuholen, empfehle ich Ihnen, die PDF mit der Kursübersicht herunterzuladen und kursbegleitend zur Hand zu haben. Bitte denken Sie an unsere Umwelt und verzichten Sie darauf, die PDF auszudrucken. Die PDF hat eine Inhaltsangabe mit internen Links, mit denen Sie schnell in der PDF zur aktuellen Lektion springen können. Machen Sie davon Gebrauch. Hier und da werden auch externe Quellen verlinkt.

Dieser Kurs wird auf Deutsch angeboten und ich versuche eigentlich ganz gerne auf den inflationären Einsatz von Anglizismen zu verzichten, aber viele Fachausdrücke haben sich in Englisch einfach durchgesetzt und Sie werden Probleme haben, wenn ich von Karten statt von Maps und von Schnittstellen statt von Interfaces spreche.

Ich habe mir Gedanken beim Aufbau der Kurses gemacht und grundsätzliche Themen und Konzepte an den Anfang gestellt. Bitte überspringen Sie keine Lektionen, außer das behandelte Thema ist Ihnen wirklich geläufig. Kommentare und weitere Erklärungen, wenn es welche gibt, stelle ich jeweils an das Ende von Lektionen.

Falls Sie ein Thema jedoch gar nicht verstehen, bekommen Sie in anderen Kursen den Tipp, die Lektion direkt noch einmal anzusehen. Ich persönlich halte das nicht für zielführend und empfehle Ihnen stattdessen Folgendes: Versuchen Sie zu dem Thema selbst zu recherchieren. Google, Webseiten wie gobyexample oder stackoverflow, oder auch die GO Webseite selbst bieten da genug Ansatzpunkte. Ist das Thema dann immer noch nicht klar, fahren Sie erst einmal im Kurs fort. Oft werden Konzepte in Rückblick klarer, wenn man sie in einem eindeutigen Kontext angewendet sieht.

Dann ganz wichtig: Machen Sie mit. Programmieren lernt man nicht dadurch, dass man anderen beim Programmieren zusieht. Vollziehen Sie die einzelnen Lektionen bei sich nach bevor Sie zur nächsten Lektion weitergehen. Und für diese Empfehlung gilt auch der Hinweis, dass Tippen, nicht Kopieren und Einsetzen der Weg zum Lernerfolg sind. Tippfehler sind weithin unterschätzte Lernhilfen, die helfen, die richtigen Gedächtnisstrukturen bei Ihnen zu formen. Wenn Sie ein fertiges Ergebnis kopieren, verzichten Sie auf diesen Fortschritt.

Jetzt laden Sie sich am Besten zuerst mal die Kursübersicht herunter, falls noch nicht geschehen.

Lektion 5 – Begleitende Kursübersicht als PDF (auch auf Github verfügbar)

Dieser Kurs wird mit einem PDF geliefert, das sich als begleitender Teil des Kurses versteht.

Lektion 6 – Kurzer Überblick über Abschnitte und Inhalte dieses Kurses

Ein kurzer Überblick über den Kurs, damit Sie einen ersten Eindruck von dem bekommen, was Sie erwartet.

Lektion 7 – Grundlegende Informationsquellen im Internet über GO und benutzte Software

Eine Sammlung von Quellen aus dem Internet, die im Kurs verwendet werden.

Über GO

- GO Website
- The GO Playground
- GO's Standard Library
- GO Packages
- GO Specifications
- GoByExample

Verwendete Entwicklungsumgebung

- Visual Studio Code

Abhängigkeiten dieses Projekts

- github.com/go-chi/chi/v5 | Router
- github.com/alexedwards/scs/v2 | Sessions
- github.com/justinas/nosurf | CSRF-Token
- github.com/asaskevich/govalidator | Validator (server-sided)
- github.com/jackc/pgx/v5 | PostgreSQL Driver & Toolkit
- github.com/xhit/go-simple-mail | Golang package for sending e-mail
- Caddy 2 | a powerful, enterprise-ready, open source web server with automatic HTTPS written in Go

Ebenfalls eine Rolle spielen

- github.com/twbs/bootstrap | Bootstrap - HTML, CSS, and JavaScript framework (no jQuery)
- RoyalUI-Free-Bootstrap-Admin-Template | Free Bootstrap 4 Admin Template
- github.com/fiduswriter/Simple-DataTables | DataTables but in TypeScript transpiled to Vanilla JavaScript
- github.com/postgres/postgres | PostgreSQL Server (mirror only)
- github.com/gobuffalo/pop | Soda/Migrations - standardization of database tasks
- github.com/dbeaver/dbeaver | Dbeaver - free multi-platform database tool
- github.com/mymth/vanillajs-datepicker | Vanilla JavaScript datepicker
- github.com/jaredreich/notie | unobtrusive notifications - clean and simple JavaScript
- github.com/jackc/pgx/v5 | SweetAlert2 - so many options for JavaScript popups
- github.com/mailhog/MailHog | MailHog - Web and API based SMTP testing

- Foundation for Emails 2 | Quickly create responsive HTML e-mails that work
- Cobra | A Framework for Modern CLI Apps in Go
- GoDotEnv | A Go port of Ruby's dotenv library

Abschnitt 2 – Schneller Einstieg in GO als Crashkurs

> *I'm trying to free your mind, Neo. But I can only show you the door. You're the one that has to walk through it.*
>
> *Morpheus*

Lektion 8 – Nehmen Sie diesen Hinweis und diesen, und diesen auch noch!

Und schon sind wir in Abschnitt 2, in dem ich Ihnen einen kleinen Überblick über in GO üblichen Programmiertechniken geben werde.

Jetzt haben wir im vorherigen Abschnitt eine integrierte Entwicklungsumgebung eingeführt und ich habe Ihnen in epischer Breite nahegelegt, doch bitte jeden Teil mitzumachen, keine Lektion auszulassen und den Code auch zu tippen, und schon brechen wir mal mit allen diesen Regeln.

Zuerst einmal soll unser IDE weniger profanen Aufgaben vorbehalten sein. Daher weiche ich in diesen Lektionen erst einmal auf den GO playground aus. Das hat mehrere Vorteile. Zum einen muss man nichts installieren. Dann steht der GO Playground praktisch überall zur Verfügung, wenn man online ist, und er bietet die Möglichkeit, Links zu erstellen und so Code einfach zu teilen. Diese Links wandern dann auch in die Kursübersicht. Also keine IDE und kein Abtippen.

Außerdem können Sie den ganzen Abschnitt überspringen, wenn Sie bereits Erfahrung mit GO oder meinen GO Einsteigerkurs durchgearbeitet haben. Natürlich können Sie die paar Lektionen auch als Wiederholung sehen.

Lektion 9 – Hello, World

Das ist also jetzt GO im Schnelldurchgang.

Wir gehen mal schnell auf den Spielplatz. Der ist erreichbar auf go.dev/play.

Bei Ihnen mag die Seite etwas anders aussehen, ich habe mir hier das GO Logo ausgeblendet und ich bevorzuge den Dark Mode in meinem Browser -- so wie ich Kaffee trinke und meine Seele tief im Innern.

Der Playground funktioniert auf jedem modernen Browser und kommt bei jedem Reload immer mit demselben Hello-World-Programm daher:

```
// You can edit this code!
// Click here and start typing.
package main
```

```
import "fmt"


func main() {
     fmt.Println("Hello, 世界")
}
```

An den chinesischen Zeichen hier, die soviel wie "Welt" bedeutet können Sie bereits eine besondere Eigenschaft von GO erkennen. GO st komplett in UTF-8 geschrieben und hat keine Probleme UTF-8 kompatible Zeichen zu verarbeiten und auszugeben. Sollte das auf der Shell ihres Computers scheitern, liegt das an der fehlenden Unterstützung Ihres Terminals, nicht an GO.

GO Programme selbst sind erst einmal ein Textdatei, deren Name auf .go endet. Jede Datei muss einem Package zugehörig sein. Ein Package kann aus mehreren Dateien bestehen. Das Package hier heißt `main`, weil es auch die Funktion `main()` einhält. Das muss nicht zwingend so sein, ist aber oft so.

Eine, und zwar genau eine, Funktion `main()` ist die Mindestvoraussetzung, um ein lauffähiges GO Programm zu erstellen, denn genau hier ist der Einstiegspunkt zum Programmablauf. Die Funktion `main()` wird deklariert durch das Keyword `func`, den Identifier, sagen wir mal den Namen der Funktion, und zweier Klammern, gefolgt von geschweiften Klammern, die den Beginn und das Ende der Funktion markieren.

Wir finden hier eine weitere Funktion namens `Println()`, die als Argument unter anderem einen String entgegen nimmt. Diese Funktion stammt aus einem anderen Package namens `fmt`. Wenn wir die Funktion nutzen wollen, müssen wir zuvor das package `fmt` importieren. Danach können wir aus dem Package `fmt` die Funktion `Println()`. Die Zuordnung machen wir durch den Punkt-Operator. Wir können nur Funktionen, Variablen und Konstanten aus importierten Packages nutzen, wenn diese in dem Package auch exportiert wurden. GO kennt dazu kein Keyword wie export, private oder public.

Vielleicht ist Ihnen aber aufgefallen, dass die der Identifier der Funktion `Println()` mit einem Großbuchstaben beginnt. Das ist keine Eigenheit des Programmierers, sondern die in GO gängige Methode, um eine Funktion, Variable oder Konstante außerhalb des Packages verfügbar zu machen. So einfach kann Programmieren sein.

Lektion 10 – Variablen - das Wichtigste zuerst

```
// You can edit this code!
// Click here and start typing.
package main


import "fmt"
```

```
func main() {
     fmt.Println("Hello, 世界")
}
```

GO ist eine strikt oder auch stark typisierende Programmiersprache mit statischen Typen. Da bedeutet erst einmal, dass alle Variablen mit einem Typ deklariert werden müssen und dieser Typ sich über die Laufzeit des Programms auch nicht mehr ändert. Werte können konvertiert werden, müssen dann aber in einer Variablen anderen Typs gespeichert werden.

Die Deklaration einer Variablen geht immer mit einer Initialisierung einher.

```
var x int
```

deklariert eine Variable mit dem Identifier x vom Typ Integer, also einer ganzen Zahl. Die Variable wird impliziert mit einem Nullwert initialisiert, wenn wir ihr nicht explizit einen Wert zuweisen. Dieser Wert ist nicht wie zum Beispiel in C irgendein Zufallswert, der gerade im Speicher der Variablen steht, sondern der sogenannte Zero-Value, ein für jeden Typ vordefinierter Nullwert, mit dem die Variable initialisiert wird.

Für Ganzzahlen ist der Zero-Value 0. Geben wir den doch mal aus und fügen hinzu:

```
fmt.Println(x)
```

Okay.

Wenn wir x mit dem Keyword `var` außerhalb einer Funktion deklarieren, hat sie packageweiten Geltungsbereich. Das nenn sich auch „package scope“. Wir können sie sogar sofort mit einem Wert initialisieren - vielleicht 23.

```
var x = 23
```

In diesem Fall legt GO den Typ für uns fest, den es aus dem zugeordneten Wert schließt.

Schauen wir uns den Typ an.

```
fmt.Printf("x is of Type %T\n", x)
```

Randbemerkung: `Printf()` dient zur formatierten Ausgabe eines Strings mit Platzhaltern, die durch %-Zeichen eingeleitet werden. Hier % Groß-T für den Abruf des Typs. Die Variable für die

der Platzhalter gilt, ist das nachgestellte durch Komma abgetrennte x. Backslash-n dient hier bei `Printf()` einfach als Mittel zum Zeilenumbruch.

Innerhalb und wirklich nur innerhalb einer Funktion gibt es die Deklaration und Initialisierung mit implizierter Typangabe eine Kurzschreibweise.

```
Y := 42.0
```

Damit erstellen wir eine Variable y und weisen Ihr den Wert 42.0 zu. GO sollte erkennen, dass wir nun eine Fließkommazahl erhalten wollen, doch die Ausführung schlägt fehl. Wir erhalten den Fehler:

```
y declared but not used
```

Warum das? Eine Variable, die nicht paketweiten Geltungsbereich hat, kann zwar in der Kurznotation mit Doppelpunkt und Gleichheitszeichen mit einem auf die Funktion beschränkten Geltungsbereich erstellt werden, aber eine solche Variable muss zwingend auch benutzt werden. Die minimale „Benutzung“, die ich mir vorstellen kann, ist die Ausgabe des Wertes von y. Bei der Gelegenheit geben wir auch mal den Typ für y aus.

```
fmt.Println(y)
fmt.Printf("y is of Type %T\n", y)
```

Jetzt klappt die Ausführung und auch der Typzuweisung hat geklappt ist klar.

Der Vollständigkeit halber erstellen wir jetzt auch noch einen String. Ein String ist in GO ein eigener Typ und die Länge des Strings ist ein Teil diesen Typs.

```
z := "Doh"
fmt.Println(z)
fmt.Printf("z is of Type %T\n", z)
```

Wir haben das Keyword `var` kennengelernt, packageweiten Geltungsbereich gesehen, und wir kennen jetzt implizierte Zuweisung des Zero-Values bei Typzuweisung, aber auch implizierte Typzuweisung bei Wertzuweisung, und haben den Short Declaration Operator, den Doppelpunkt Gleichheitszeichen benutzt.

Für Variablen belassen wir es erst einmal dabei und werfen einen Blick auf Funktionen.

Code auf dem Playgorund ansehen

Lektion 11 – Alles funktional?

Praktisch alle imperativen Programmiersprachen erlauben prozedurale Strukturen in Form von Funktionen. GO ist da nicht anders.

Funktionen kann man wo immer man möchte mit packageweitem Gültigkeitsbereich anlegen. Vereinfacht gesagt, nennt man die Deklaration ihre Signatur. Das sieht dann in etwas so aus:

```
// func (r receiver) identifier(arg argumentType, ...)
(returnType(s)) { Code }
```

Übrigens: die beiden Slashes hier sind eine übliche Notation für Kommentare in GO.

Gehen wir mal durch und erstellen eine einfache Ausgabefunktion.

Keyword func, receiver r stellen wir zurück, das kommt später. Wir können den Receiver problemlos weglassen.

Identifier `outputThat`

Drei durch Komma getrennte Argumente a, b, c vom Typ `int, float64` und `string`.

Wir wollen hier mal etwas zurückgeben und da kommt sofort einen weitere Besonderheit von GO zum Einsatz, die Ihnen wahrscheinlich unbekannt ist, falls Sie Erfahrungen in Java, C oder PHP haben: GO kann mehrere Rückgabewerte haben. Das ist ein Alleinstellungsmerkmal von GO, von dem wir ausführlich Gebrauch machen werden. Später mehr.

Hier wollen wir mal eine Statusmeldung `done` vom Typ `string` und vielleicht einen boolean Wert ausgeben.

```
func outputThat(a int, b float64, c string) (string, bool) { }
```

Jetzt wollen wir die Funktion auch ausführen und ihr x, y und z übergeben. Wir rufen `outputThat()` innerhalb von `main()` auf.

```
outputThat(x, y, z)
```

Aber wir werden zwei Rückgabewerte erhalten. Die müssen wir durch irgendetwas entgegennehmen lassen und das machen wir mittels des `:=` Operators.

```
done, trueOrNot :=
```

Hier zusammen in einer Zeile:

```
done, trueOrNot := outputThat(x, y, z)
```

Allerdings müssen wir diese Werte auch ausgeben. Sie erinnern sich. Jede innerhalb einer Funktion deklarierte Variable muss auch benutzt werden. Wertzuweisung reicht dazu nicht.

```
fmt.Println(done, trueOrNot)
```

Bevor wir das ausführen, muss unsere Funktion aber auch etwas machen und etwas zurück liefern. Implementieren wir das:

```
fmt.Printf("Integer = %d, FloatingPoint = %f, String = %s\n", a, b, c)
return "Okay.", true
```

Jetzt noch ein paar wichtige Anmerkungen.

Man kann auch Funktionen ohne Identifier erstellen. Das nennt man anonyme Funktionen, die man dann auch durch Anhängen von einfachen Klammern sofort zur Ausführung bringen kann.

Außerdem wir bei der Wertübergabe der Argumente nicht zwischen "call-by-value" und "call-by-reference" unterschieden, das bedeutet, dass die Betrachtungsweise immer call-by-value ist. Immer bei einem Funktionsaufruf, bei dem Werte übergeben werden, werden Kopien dieser Werte erstellt, die ausschließlich innerhalb und zur Ausführungszeit einer Funktion existieren.

Das bedeutet hier zum Beispiel, dass wir zwar unser a,b und c neue Werte zuweisen können, aber x, y und z davon völlig unbeeinflusst bleiben. Das gilt auch, wenn wir innerhalb unserer Funktion `outputThat()` die Identifier x, y und z benutzen würden. Auch wenn sie gleich aussehen und vom gleichen Typ sind, sind das komplett eigene Entitäten.

Wenn wir Werte außerhalb der Funktion ändern wollen, müssen wir Pointer übergeben. Über Pointer lernen wir in der nächsten Lektion mehr, aber soviel sei schon hier gesagt: Auch bei der Übergabe von Pointern wird wieder eine Kopie angelegt und die Betrachtungsweise der Funktionsaufrufs mit Übergabe von Argumenten bleibt gleich: call-by-value. Es wird eben nur ein Wert vom Typ Pointer übergeben. Bei anderen Programmiersprachen spricht man in diesem Fall von call-by-reference. Besser wir gewöhnen uns an call-by-value. Wir lassen das jetzt hier mal sein.

Und last but not least, die Regel mit der aus dem Package exportierten Variablen oder Konstanten durch Großschreibung des ersten Zeichens des Identifiers gilt auch für Funktionen! Das bedeutet, dass wenn jemand unser `package main` importieren würde, könnte er `outputThat()` nicht aufrufen mit

```
main.outputThat(23, 42.0, "Opps.")
```

Bei eine Funktion `OutputThat()` wäre aber ein Aufruf von

```
main.OutputThat(23, 42.0, "Yuppie.")
```

möglich.

Code auf dem Playground ansehen

Lektion 12 – Pointer – mit dem Finger auf andere zeigen

Wenn Sie von einer Programmiersprache wie PHP kommen, sind Pointer wahrscheinlich ein komplett neues Konzept für Sie. Pointer sind jedoch unglaublich nützlich.

Man könnte ein Video von einer Stunde Dauer machen, um Pointer im Detail zu erklären, und genau das habe ich auf meinen YouTube Channel für C bereits gemacht. Aber das wäre hier nicht zielführend, denn und das als Hinweis jedoch für alle, die bereits etwas mit Pointern anfangen können: Pointer in GO kommen ohne die teils schwer zu verstehende und daher gefürchtete Pointer-Arithmetik aus. Daher befassen wir uns hier ausschließlich mit dem fundamentalen Konzept eines Pointers wie er in GO zum Einsatz kommt.

Um Pointer zu erklären, nutze ich hier Strings. Dann haben wir das Thema auch schon mal erwähnt. Strings in GO sind ein eigener Datentyp! Im Grunde eine Aneinanderreihung von 32-bit breiten Ganzzahlen: `int32`. Jede dieser 32-bit Zahlen wird jedoch als UTF-8 Zeichen interpretiert. Und um Zahlen von UTF-8 Zeichen zu unterscheiden, benutzt man einen eigenen Datentyp für UTF-8-kompatible 32-bit Integers: Runes. Der Datentyp `rune` wiederum ist nichts anderes als ein Alias zu `int32`. Ein String ist ein eigener Datentyp, der eine Aneinanderreihung von solchen Runes darstellt. Das ist jetzt auch noch nicht ganz richtig, denn der Datentyp String beinhaltet nicht die ganzen Daten selbst, sondern einen Anfangspunkt im Speicher, an dem die erste Rune zu finden ist, eine Länge innerhalb des Speichers und eine Kapazität, bis zu der man ohne großen Zeitverlust Runes – also UTF-8-konforme Zeichen – zu einem String hinzufügen kann, ohne dass unter der Haube irgendeine Hin-und-Herkopiererei stattfinden muss, um zusammenhängenden Platz für unseren String im Speicher zu schaffen.

Uff – viel Information auf einmal über Strings. Ganz kurz nochmal: `string` ist ein Datentyp. Die Werte sind Aneinanderreihungen von UTF-8-Zeichen, die als `rune` bezeichnet werden. Strings haben eine Länge und eine Kapazität.

Und jetzt auf zum Spielplatz

Was haben Strings mit Pointern zu tun?

Augenscheinlich erst einmal nichts, jedenfalls nichts, was hier jetzt wichtig wäre. Ich nutze hier einen String, um Pointer zu erklären.

```
aCostume := "Clip Clop"
fmt.Printf("Roger is wearing a costume as %s.\n", acostume)
```

Wenn ich nun versuche, diesen String an eine Funktion zu übergeben, funktioniert das bis zu einem bestimmten Punkt erst einmal.

```
func changeTheCostume(c string) {
     fmt.Printf("Roger is wearing a costume as %s.\n", c)
     newCostume := "Ricky Spanish"
     c = newCostume
     fmt.Printf("Roger is wearing a costume as %s.\n", c)
}
```

aber, wenn ich den Inhalt hier ändere, und nach Aufruf der Funktion hier in `main()` ausgebe

```
changeTheCostume(aCostume)
fmt.Printf("Roger is wearing a costume as %s.\n", aCostume)
```

sehen Sie, dass sich rein gar nichts geändert habe. Selbst wenn ich innerhalb der Funktion nun denselben Identifier nutze, funktioniert das nicht. Um diese Behauptung zu beweise tausche ich mal eben `c` gegen `aCostume`.

Hinweis: An dieser Stelle im Video wird `c` kurz gegen `aCostume` ausgetauscht.

Wie ich Ihnen in der Lektion über Funktionen schon erklärt habe, wird bei einer Übergabe eines Wertes in eine Funktion immer eine Kopie dieses Wertes erstellt. Wir können den Wert dieser Kopie zwar ändern, aber unsere Änderung wirkt sich nur auf die Kopie aus. Und der Geltungsbereich dieser Kopie ist ausschließlich innerhalb der Funktion. Die Kopie existiert buchstäblich nur zur Laufzeit dieser Funktion.

Man könnte nun natürlich auf die glorreiche Idee kommen, einfach mal alle Variablen mir `var` außerhalb einer Funktion packageweit verfügbar zu machen, aber davor kann ich nur warnen. Jede Variable nimmt Speicherplatz ein. Das mag jetzt bei einer oder zwei int überhaupt nichts ausmachen, aber bei selbst erstellten Datentypen von vielleicht mehren Megabyte Größe müllen Sie sehr schnell den Speicher voll. Man sollte immer nur die Variablen in die Existenz rufen, die man auch handeln kann.

Genau hier kommen Pointer ins Spiel.

```
func changeTheCostume(c *string)
```

Mit den Asterisk Operator teilen wir unserer Funktion mit, dass wir als Typ nicht einen Wert vom Typ String erwarten, sondern einen Pointer zu einem String. Wir wollen also nicht den Wert

auslesen, sondern nur wissen, wo der Wert steht. Der Pointer zeigt auf Daten eines Typs. Merken: Sternchen mit Typ ist ein Pointertyp.

Aber nun dürfen wir nicht mehr den Wert des Strings übergeben, sondern nur noch die Adresse im Speicher, wo der Stringwert zu finden ist. Dafür nutzen wir das vorangestellte Ampersand (oder et, oder „Kaufmannsund“, wie immer Sie dieses Zeichen nennen mögen.

```
changeTheCostume(&aCostume)
```

Und in einem dritten Schritt müssen wir nun Rogers Kostüm noch einen neuen Wert zuweisen.

Doch wir haben ja nur einen Ort, wo dieser Wert im Speicher steht. Hier kommt das Sternchen wieder zum Einsatz, aber diesmal nicht mit dem Typ, sondern mit dem Identifier. Was eben eine Referenz war, wird nun dereferenziert, oder mit einfachen Worten: Wir wollen auf den Wert zugreifen, der an diesem Ort gespeichert ist. Und diesem weisen wir einen neuen Inhalt zu. Diesmal natürlich keinen Pointerwert, sondern einen echten String.

```
*c = newCostume
```

Innerhalb unserer Funktion ist ja auch `c` nun kein String mehr, sondern ein Wert vom Typ Pointer zu einem String. Wenn wir den dort stehenden Wert mit `Printf()` ausgeben wollen, müssen wir den ebenfalls dereferenzieren.

```
fmt.Printf("Roger is wearing a costume as %s.\n", *c)
```

Tadaa! Man sieht, dass nun Roger problemlos sein Kostüm wechseln kann.

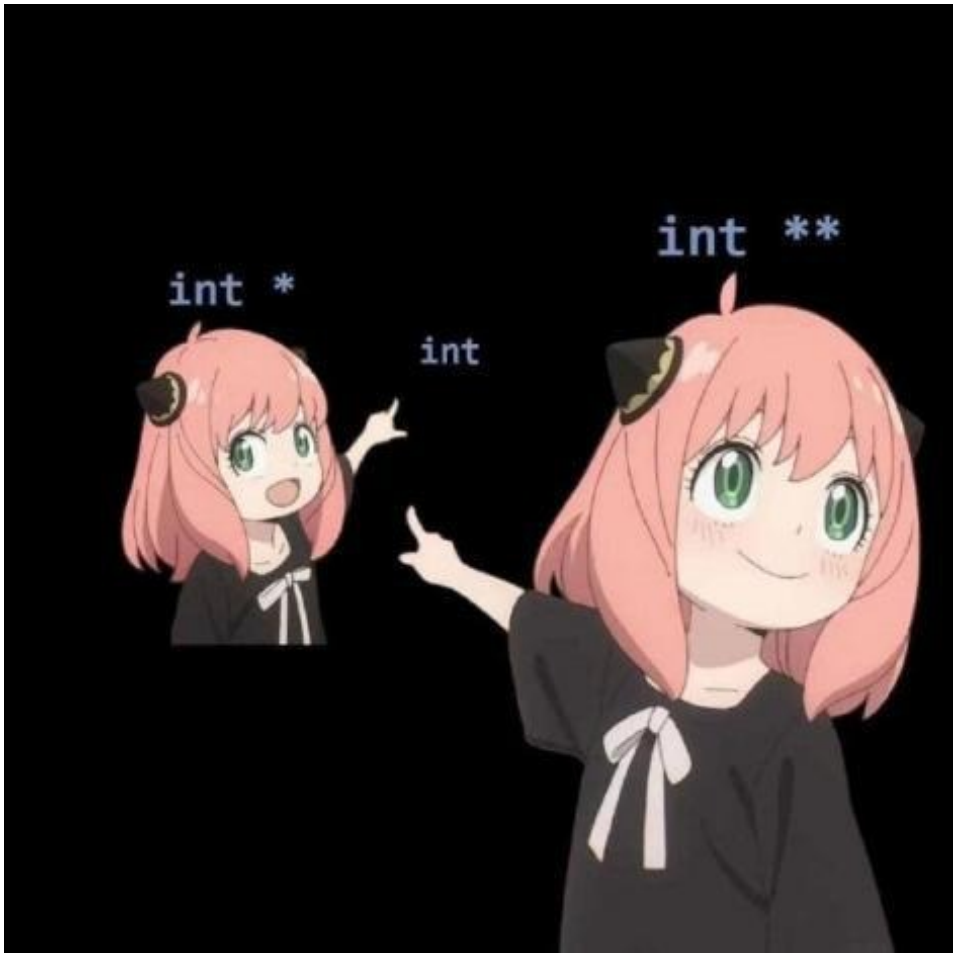


Bild 1: (Quelle: https://i.redd.it/r01sq96mnz691.jpg)

Jetzt sollte auch klar sein, woher der Name Pointer stammt. Sie sind Zeiger, die auf eine bestimmte Stelle im Speicher zeigen, wo eine bestimmter Wert eines bestimmten Typs gespeichert ist. Das Spiel kann man noch weiter treiben und Pointer zu Pointern erstellen, weil ein Pointer ja selbst wieder einen Wert darstellt, der irgendwo im Speicher liegt.

Code auf dem Playground ansehen

Zusammenfassung, was wir jetzt verstanden haben sollten:

Sternchen „*“ mit einem Typ zusammen definieren einen Pointer zu einem Typ.

Ampersand „&“ vor einer Variablen liefert die Adresse der Variablen, und stellt einen Wert dar, den ein Pointer enthält.

Sternchen „*“ mit Identifier, einem Variablennamen, zusammen stellen aber eine Dereferenzierung des Wertes dar, auf den der Pointer zeigt.

Alle klar?

Lektion 13 – Die Schattenwelt - es geht immer um Typen und Structs

Variable Shadowing

Bevor wir uns Typen und Structs zuwenden, werfen wir noch mal einen kurzen Blick auf das geheime Schattendasein, das Variablen führen können.

```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

var role1 = "Billionaire"

func main() {
	var role2 = "Inventor"
	fmt.Println("Bruce Wayne is", role1)
	fmt.Println("Bruce Wayne is", role2)

	tellMeWhoYouAre("Batman")
}

func tellMeWhoYouAre(role1 string) string {
	fmt.Println("Bruce Wayne is", role1)
	return role1
}
```

Tauscht man in diesem einfachen Beispiel den Namen des Arguments der Funktion und für den Rückgabewert von `role1` zu `role3` wird der global für das Package global definierte Wert gültig. Daraus können wir sehen, dass der Gültigkeitsbereich von Variablen, die innerhalb einer Funktion deklariert wurden, eine höhere Priorität haben als die globale Deklaration. Das muss man erst einmal sacken lassen, denn im Umkehrschluss bedeutet das auch, dass Variablen eine geheime Schattenexistenz führen können, von der man nichts weiß, weil man sich bei der globalen Deklaration nicht bewusst darüber ist, dass die in irgendeiner irgendwo importierten Funktion unter demselben Identifier, ein geheimes Schattendasein führen. Das nennt sich Variable Shadowing. Also mit der Benennung immer schön aufpassen. Das ist auch der Grund, warum in Funktionen oft eher generische Variablennamen mit einem Buchstaben benutzt werden, global oder im Hauptprogramm aber durchaus eher Aussagekräftige – vermute ich jetzt einfach mal so.

Variable Shadowing kann aber auch passieren, wenn ich innerhalb der Hauptfunktion `main()` erneut `role1` deklariere und ihr einen Wert zuweise. Ich mache das mal mit dem Short Declaration Operator.

```
role1 := "playboy"
```

Da ist total zulässig in `main()` und so wird aus `role1` “playboy”.

https://go.dev/play/p/AAtDF7mxhtd

So viel zu Variable Shadowing. Sieht man sich die Spezifikation von GO auf https://go.dev/ref/spec an, sieht man recht schnell: Es dreht sich alles im Typen und Typisierung. Neben den üblichen Verdächtigen, sind sogar Strings ein Typ ebenso wie sogar Funktionen selbst einfach nur Typen sind.

Ein Typ fällt da noch auf, den wir uns jetzt ansehen und das sind structs. Structs kennen Sie vielleicht aus einer Vielzahl der objektorientierten Programmiersprachen. Ob GO dazu gezählt werden sollte, daran scheiden sich noch die Geister. Jedenfalls bietet auch GO structs an.

Structs dienen der Strukturierung von Daten

Was ein struct ist lässt sich am Leichtesten erklären, wenn wir versuchen, die Familie Smith mit einfachen Eigenschaften zu beschreiben.

```
var familyName string
var firstName string
var age int
var birthdate time.Time
var species string
```

Allesdings gerät das schnell außer Kontrolle, wenn man so eine Gruppe für einen einzelnen Datensatz an eine Funktion übergeben möchte, oder ihn vielleicht in einer Datenbank speichern möchte.

Man kann solche Datensätze aber zusammenfassen in einer Struktur mittel des Keywords struct und üblicherweise legt man dafür gleich seinen eigenen Datentyp an, dem man seinen eigene Identifier gibt.

```
type FamilyMember struct {
     FamilyName string
     FirstName  string
     Age         int
     Birthdate  time.Time
     Species    string
}
```

Jetzt haben wir einen Type deklariert, jetzt deklarieren wir uns auch einen Wert. Das funktioniert sehr fein mit dem Short Declaration Operator innerhalb von `main()`.

```
francine := FamilyMember{
          FamilyName: "Smith",
          FirstName:  "Francine",
          Species:    "Human",
     }
```

Und schon können wir den kompletten Datensatz ausgeben, oder auch einzelne Elemente.

```
fmt.Println(francine)
fmt.Println(francine.FirstName)
```

Fügen wir noch einen Roger hinzu, dann ist das Prinzip schnell klar.

```
roger := FamilyMember{
          FamilyName: "Smith",
          FirstName:  "Roger",
          Age:        1600,
          Species:    "Alien",
     }

fmt.Println(roger)
fmt.Println("Roger is", roger.Age, "earth years old.")
```

Beim Benutzen von structs gibt es in GO nur wenig zu beachten.

Zum einen kann man einen selbst definierten Typ nur dann außerhalb eines Packages benutzen, wenn der Identifier des Typs groß geschrieben wurde. Zum anderen gilt das natürlich ebenso für jedes Element des Structs. Der Identifier eines Structs oder der Elemente ist da nicht anders als der einer Variablen, Konstante oder Funktion.

Zum anderen muss man bei der Wertzuweisung des Elemente darauf achten, dass man nicht zwingen des Elementbezeichner wiederholen muss. Das gehört aber zum guten Ton und Best Practice. Und, dass man alle Wertezuweisungen durch ein Komma voneinander trennen muss – auch die letzte. Ansonsten ist das einfach ein Syntax-Error.

Sie wissen also jetzt Bescheid über variable Shadowing und haben die den Einsatz von structs kennengelernt.

Code auf dem Playground ansehen

Lektion 14 – Receiver – der Wahnsinn bekommt Methode

Was ist eine Methode? Klassisch ist eine Methode eine Funktion, die ein Element eines struct ist.

Das, was Ihnen als Methode aus der Objekt-orientierten Programmierung vielleicht bekannt ist, hat auch in GO Einzug gehalten und ist nicht strikt als Element eines struct umgesetzt.

Was ist damit gemeint? In C++ zum Beispiel lassen sich Funktionen als Elemente in ein Struct integrieren und mit dem Punkt-Operator als Funktion mit Zugriff auf die anderen Struct-Elemente definieren. Auf diese Weise erhält man so etwas wie eine interne Funktion, das man wie ein Element aufrufen kann. Man kennt diesen Aufbau als eine Methode.

In GO nutzt man um eine Methode zu erstellen den Receiver r, den ich Ihnen in der Lektion mit der Einführung von Funktionen vorenthalten habe. Genau das holen wir jetzt nach. Ich habe das Beispiel aus der vorherigen Lektion mit ein paar zusätzlichen Werten versehen, um die Mitglieder der Familie Smith aus der Serie American Dad abzubilden. Das Element Geburtsdatum habe ich mal entfernt. Und gebe alle Werte einmal aus, weil man keine ungenutzten Variablen in GO erstellen kann.

Werfen wir ein Blick darauf: https://go.dev/play/p/BtgNcikw83A

Ich füge mal gleich hier unterhalb des Structs eine Funktion ein:

```
func (r FamilyMember) sayYourName() string {
     fullName := r.FirstName + " " + r.FamilyName
     return fullName
}
```

Die Funktion muss nicht zwingend unterhalb des structs stehen. Man kann sie im Prinzip überall im Package anlegen, denn der verbindene Teil ist hier das Element Receiver r.

Und jetzt wollen wir mal so eine Methode benutzen.

Sagen wir Roger will von Klaus seinen kompletten Namen erfahren. Wir machen das so:

```
fmt.Println("Hello my name is", roger.sayYourName(), "- show me
your ID card and I tell you who you are. Klaus says: ",
klaus.sayYourName())
```

Hier rufen wir gleich die Methode `sayYourName()` von zwei verschiedene Werten auf. So erstellt man Methoden in GO. Man erstellt einfach eine Funktion, die sich durch Setzen eines Receiver r auf einen als struct definierten Typ bezieht. Fertig.

Code auf dem Playground ansehen

Lektion 15 – Maps und Slices

Jetzt wollen wir mal ein bisschen auf die Tube drücken und auch in dieser Lektion wieder zwei Themen abhandeln – Maps und Slices. Schließlich wollen wir ja in absehbarer Zeit auch mal anfangen, eine Webanwendungen zu bauen. Und der aktuelle Abschnitt beschäftigt sich ja nur mit den Grundlagen der Programmiersprache GO.

Als Programmierer verbringen Sie einen großen Teil Ihrer der Zeit damit, Daten eine Struktur zu geben oder den passenden Datentyp für eine Struktur von Daten zu finden.

Maps

Um große Mengen an Werten von ungeordneten Datenpaaren verfügbar zu machen, gibt es sogenannte Hash Tables. Hash Tables im Detail zu erklären, sprengt bei Weitem den Rahmen dieser Lektion, aber es gibt einen sehr schönen Artikel auf medium.com, der die Grundlagen von Hash Tables und die konkrete Implementierung in GO in einfach zu verstehendem Englisch erklärt.

Hash Tables in GO werden Maps genannt. Maps sind ein zusammengesetzter Datentyp, der ungeordnete Datenpaare über einen Schlüsselwert auffindbar macht. Wir schauen uns ein einfaches Beispiel an. Sehr oft werden hier Benutzerdaten herangezogen, weil diese Daten in der Regel einmalig sind. Wir können uns einfach vorstellen, dass ein Service wie Google, den Bedarf hat, einzelne Benutzer aus vielen Millionen Benutzerkonten sehr schnell anhand Ihrer – sagen wir mal – E-Mail-Adresse zu identifizieren und dieses Daten ungeordnet und schnell durchsuchbar vorzuhalten. Das ziehen wir jetzt mal als Beispiel heran.

Zuerst erstellen wir und eine Map auf dem Playground. Das geht mit dem Short Declaration Operator und ist durch Nutzung des Keywords `make` recht einfach und verständlich.

```
//    mapIdentifier := make(map[keyType]dataType)
```

`make` selbst ist eine ummantelnder Befehl, der dafür sorgt, dass für einige Datentypen entsprechend Speicher reserviert wird, dieser Speicher erweitert und auch am Ende des Programmablaufs wieder freigegeben wird. Als Argument rufen wir einen Datentyp – hier `map` auf mit den Eigenschaften, die wir benötigen.

Der Datentyp `map` sieht so aus, dass die Datentypbezeichnung `map` gefolgt wird von eckigen Klammern, die den Typ eines Schlüsselwertes halten, gefolgt von einem Datentyp. Maps in GO sind keine Raketenwissenschaft.

```
users := make(map[string]string)
```

Und jetzt erstellen wir uns ein paar Daten.

```
users["peter@griffin.family"] = "Peter Griffin"
```

und anschließend machen wir mal die Probe und geben den Wert aus, den wir unter dieser E-Mail-Adresse gespeichert haben:

```
fmt.Println(users["peter@griffin.family"])
```

Funktioniert auf Anhieb. Wir fügen ein paar mehr Datenpaare hinzu.

```
users["lois@griffin.family"] = "Lois Griffin"
users["meg@griffin.family"] = "Meg Griffin"
users["chris@griffin.family"] = "Chris Griffin"
users["stewie@griffin.family"] = "Stewie Griffin"
users["brian@griffin.family"] = "Brian Griffin"
```

Können wir auch die komplette `map` ausgeben?

```
fmt.Println(users)
```

Sie sehen auch das funktioniert, sollte man aber mit Vorsicht benutzen, weil ein paar Millionen Datensätze abzurufen, auch mal eine Zeit dauern kann. Bitte beachten Sie, dass die einzelnen Datensätze nicht in der gleichen oder umgekehrten Reihenfolge ausgegeben werden, in der sie erstellt wurden.

Ein paar findige Programmierer dachten, sie könnten sich damit auch etwas Arbeit ersparen und diesen Umstand wie einen Index benutzen, doch das ist nicht der Weg, wie die GO Entwickler sich die Nutzung von Maps vorgestellt haben. Daher haben sie dafür Sorge getragen, dass bei der Ausgabe immer eine andere Reihenfolge gewährleistet ist. Maps sind eben ungeordnete Listen.

Was passiert, wenn wir jetzt einen weiteren Datensatz mit demselben Schlüsselwert hinzufügen wollen?

```
users["stewie@griffin.family"] = "Steward Griffin"
fmt.Println(users["stewie@griffin.family"])
```

Das funktioniert natürlich nicht, weil jeder Schlüsselwert einmalig ist. Eine erneute Erstellung eines Datenwertes mit einem schon existierenden Schlüssel überschreibt einfach nur den schon erstellten Wert.

Einen Wert zu löschen ist einfach und funktioniert mit der Funktion `delete()`, die den Identifier der Map und den Schlüsselwert des zu löschenden Elements entgegennimmt.

```
delete(users, "stewie@griffin.family")
fmt.Println(users["stewie@griffin.family"])
```

Und hier sehen, Sie auch, dass ein nicht in einer map vorhandener Schlüsselwert, nicht zu einer Fehlermeldung führt. Wenn Sie also prüfen möchten, ob ein Wert in einer Map vorhanden ist oder nicht, gibt es da eine andere Möglichkeit.

Zu guter Letzt steht die Frage im Raum, was den der Zero-Value einer Map ist. Also, was ist da drin, wenn die Map noch keine Werte hat? Wir fügen einfach an entsprechender Stelle hinzu:

```
fmt.Println(users)
```

Doch wir erhalten nur:

```
map[]
```

Stattdessen müssen wir entweder die Länge mit der Funktion `len()` feststellen oder wissen, dass der Pointer einer leeren Map vom Datentyp `nil` ist. Unter der Haube gibt es nämlich einen Pointer, der zum ersten Element einer Map zeigt. Ist das nicht vorhanden, hat er den Wert `nil`.

```
fmt.Println(len(users))
fmt.Println(users)
```

Es scheint hier so zu sein, dass `users` schon einen Wert ungleich `nil` hat. Das liegt daran, dass wenn wie den Short Declaration Operator und `make` benutzen, der Pointer bereits auf eine Speicheradresse gerichtet wird, an der GO plant die Daten der Struktur anzulegen. Auf eine leere Map mittels `nil` zu schließen ist nur möglich, wenn wir die Map mit `var` angelegt haben. Dann haben wir das auch mal gesehen:

```
var dummyMap map[string]string
fmt.Println(dummyMap == nil)
```

Dieser Zero-Value existiert neben für leere `maps` auch für die Datentypen `pointer, function, interface, channel` und `slice`. Im Grunde dient `nil` als Wert für einen leeren Pointer, damit der nicht auf die Zahl oder Adresse Null zeigen muss, und alle diese

Datentypen beschreiben unterschiedliche zusammengesetzte Strukturen, die irgendwo im Speicher abliegen. Und auf den Letztgenannten, `slice`, werfen wir jetzt einen Blick.

Code auf dem Playground ansehen

Slices

Ein Slice in GO ist im Grunde ein Datentyp, der auf ein Array aufgebaut wurde, um die Mängel im Umgang mit Arrays zu beheben und die Werte eines beliebigen Datentyps in sequenzieller Ordnung verfügbar macht. Weitere Datentypen bauen auf Slices auf. Der Datentyp string ist zum Beispiel ein Slice vom zugrundeliegenden Datentyp `byte`, was wiederum nur eine Reihe von unsigned 8-bit Werten ist. Im String werden die bei der Ausgabe, aber dann gemäß UTF-8 in 4er-Paketen als `rune` interpretiert. Das aber nur als Randbemerkung.

Um Slices zu erklären, erstellen wir uns ein typisches Array.

```
var meeseeks [3]string
fmt.Println(meeseeks)
```

Jetzt noch schnell ein paar Werte, auf die wir auch problemlos zugreifen können:

```
meeseeks[0] = "Meeseek#1"
meeseeks[1] = "Meeseek#2"
meeseeks[2] = "Meeseek#3"
fmt.Println(meeseeks)
fmt.Println(meeseeks[0])
```

Wollen wir einen weiteren Meeseek hinzufügen, schlägt das fehl.

```
meeseeks[3] = "Meeseek#4"
invalid argument: index 3 out of bounds [0:3]
```

Code auf dem Playground ansehen

Das bedeutet für uns, dass bei der Nutzung von Arrays in der Regel zur Compilierzeit schon die Größe des Arrays feststehen muss. Das ist aber selten der Fall. Man kann jetzt entweder ein viel zu großes Array annehmen und eine Menge Speicherplatz verschwenden, oder man muss dynamisch Speicherplatz reservieren und seinem Arrays hinzufügen. Das bringt aber Geschwindigkeitsprobleme zur Laufzeit mit sich. Jede Erweiterung oder Verkleinerung eines Arrays bedeutet neuen zusammenhängenden Speicher zu finden, die bestehenden Daten dorthin zu kopieren, die zusätzlichen Daten ebenfalls dort anzulegen, und schließlich den bisherigen

Speicherplatz zu löschen oder freizugeben. Wenn man das ab und zu machen muss, geht das vielleicht noch, aber wenn man das für jedes Element, das hinzugefügt wird, macht, wird dynamisch reservierter Speicher für Arrays schnell zur Kraftprobe für jeden Computer. Und dann stellen Sie sich noch vor, Sie wollen ein Element an einer bestimmten Stelle einfügen oder löschen ohne, dass eine Lücke entsteht und das mit Millionen von Elementen.

Im Grunde ist dynamisch zugewiesener Speicher für Arrays immer ein Kompromiss zwischen Ausführungsgeschwindigkeit und Flexibilität. Darüber hinaus muss man die Programmiertechniken dafür erst einmal beherrschen und dann auch mit Rücksicht auf die entsprechende Prozessorarchitektur zum Einsatz bringen. Für eine Workstation gelten da andere Regeln als für einen Raspberry Pi.

Die Lösung, die GO gefunden hat, um diese und andere Probleme im Zusammenhang mit Arrays erst gar nicht aufkommen zu lassen, sind die Slices-Typen.

Slices kann man auf verschiedene Arten erstellen.

1. Mit `var` genauso wie Arrays, in dem man die Anzahl in den vorangestellten Klammern weg lässt. Damit überlässt man erst einmal alles Grundeinstellungen und die Initialisierung dem Compiler
2. Mit Short Declaration Operator und einer direkten Wertzuweisung für zumindest ein Element.
3. Oder mit `make`. Das sollte die bevorzugte Methode sein, wenn man die Anfangsgröße und eine zu erwartende Größe zu Laufzeit bereits kennt oder zumindest sinnvoll abschätzen kann.

Schauen wir uns alle drei Möglichkeiten auf dem Playground an. Wir werden lernen, Slices zu erstellen, Werte zuzuweisen und wir werden zwei Eigenschaften kennenlernen, die ein Teil Ihres Typs darstellen: Größe und Kapazität

https://go.dev/play/p/nAKPG3ogHth

Ganz wichtig: Slices sind „immutable“ also sozusagen unveränderlich. Was bedeutet das? Man kann einem Slice natürlich einen Wert zuweisen und danach einen neuen Wert zuweisen. Und scheinbar ändert sich der Wert ja dann. Die Wahrheit ist aber, dass ein neues Slice angelegt wird mit den gewünschten Eigenschaften des neuen Slices. Sie erinnern sich, Größe und Kapazität sind Eigenschaften eines Slices und verschiedene Slices sind miteinander nicht kompatibel.

Daher ist es bei Operationen mit Slices nötig, Funktionen anzuwenden. `append()` haben wir kennengelernt, aber es gibt keine delete() Funktion! Wie gesagt ist es ja ohnehin nötig, ein Slice neu aufzubauen und den Wert neu zuzuweisen und das funktioniert mit `append().`

Wir machen das jetzt nur ganz kurz hier, denn etwas zu löschen, indem man etwas hinzufügt, klingt anfangs paradox, wenn man nicht weiß, dass man ein Slice problemlos weiter slicen kann.

Wir nehmen mal das Slice `someFloats:`

```
[42.42 0 0 23.23 123.456 555.555 888.888] 7 12
```

Wir wollen 123.456 aus dem Slice löschen. Dazu nehmen wir das Slice und weisen einen neuen Wert zu. Aber wir starten nur bis zu einem Teil vom Anfang einschließlich bis zu dem Teil, der entfernt werden soll ausschließlich. Wir nutzen einen Selektor, der durch einen Doppelpunkt getrennt innerhalb eckiger Klammern dem Identifier des Slices nachgestellt wird. Nach dem gleichen Prinzip fügen wir anschließend den Teil hinzu, der folgen soll. Denken Sie daran, das der Index bei Null beginnt.

```
someFloats = append(someFloats[0:4], someFloats[5:7])
```

was uns einen Fehler einbrockt:

```
cannot use someFloats[5:7] (value of type []float64) as type
float64 in argument to append
```

`append()` erwartet kein Slice, das angehängt wird, sondern eine beliebige Zahl von Einzelwerten des zugrundeliegenden Datentyps. Wir müssen die irgendwie „ausrollen“ oder „abspulen“. Und in GO gibt es dafür einen ganz einfachen Operator. Wir hängen drei Punkte an das Slice, das wir als Einzelwerte übergeben wollen, an.

```
someFloats = append(someFloats[0:4], someFloats[5:7]...)
```

Das funktioniert schon mal ganz toll, aber es gibt noch eine klarere, verkürzte Schreibweise, wenn wir „vom Anfang“ und das „bis zum Ende“ auswählen wollen. Wir können nämlich dann einfach den Index komplett wegfallen lassen:

```
someFloats = append(someFloats[:4], someFloats[5:]...)
```

Es gibt schon einige Funktionen, die einem das Leben erleichtern, und die auf den Umgang mit Slices hin und auf Geschwindigkeit optimiert und ausgiebig getestet sind. Beispiel ist hier das Sortieren, das ich anhand der Methode `Ints()` aus dem package `sort` demontieren will:

```
sort.Ints(someIntegers)
fmt.Println(someIntegers, len(someIntegers), cap(someIntegers))
```

Bam! Fertig.

Man braucht sich um Speicheraufteilung, Reservierung und Freigabe überhaupt nicht mehr zu kümmern. Slices sind eines der Features in GO, die „Ease of Programming“ überhaupt erst möglich machen. Wenn Arrays Programmierers Brot und Butter sind, sind Slices so eine Art „Super Food“

wir Goji Beeren auf Steroiden. Von den GO Entwicklern wird der Einsatz von Slices anstelle von Arrays ausdrücklich empfohlen.

Code auf dem Playground ansehen

Lektion 16 – Entscheidungen & Conditionals – if, else, else if, switch

Haben Sie jemals damit gerungen, eine Entscheidung zu treffen? Ich glaube wir alle haben das schon mal. Dabei treffen wir jeden Tag Hunderte vielleicht Tausende Entscheidungen – gute und weniger gute. Die Bedingungen, auf denen unsere Entscheidungen beruhen, sind vielfältig sowohl in der Art der Bedingung als auch im Wert und der Gewichtung dieses Wertes.

Computer haben es einfacher. Sie sind am Ende des Tages von sehr binärer Natur. Man könnte sie als sehr weit und weit entwickelten Lichtschalter betrachten, der genau zwei Zustände kennt: an und aus, 1 und 0, wahr und falsch.

Für einen Computer gibt es also auf die Frage „Ja oder Nein?“ keine Antwort wie „Vielleicht“, „ungefähr“ oder „so etwas in der Art“. Daher kann man sämtliche Zustände alle grundsätzlich auf „wahr“ oder „falsch“ prüfen.

Bestimmt aus anderen Programmiersprachen bekannt ist sind da Keywords wie `if, else, else if`.

Wir schauen uns die kurz an und lernen bei der Gelegenheit noch schnell `Println()` aus dem Paket `log` kennen. (Weitere Erläuterungen im Video)

Code auf dem Playground ansehen

Wichtig ist nur daran zu denken, dass die Logik der Vierschachtelung von `if, else if` und `else` Statements in GO durch Einrückungen dargestellt wird.

Mehr als eine `else if` Abfrage ist selten sinnvoll. Braucht man eine komplexere Logik, kann man meist auf ein eine `switch` Anweisung ausweichen.

Schauen wir uns auch dafür ein Beispiel an. (Weitere Erläuterungen im Video)

Code auf dem Playground ansehen

Lektion 17 – In da loop: „for“ und „range“ als Team

Keine anderen Loops, keine While, keine Do-while-Schleifen.

3 Schritte:

1. Initialisierung
2. Bedingung
3. Wertänderung

Für Datentypen, bei denen sich Länge dynamisch ändert, kann man sich nun endlich zunutze machen, dass die Länge ein Teil diverser Datentypen ist. Man kann die abfragen. Daraus basiert

Keyword range, das zum Iterieren über die Werte von verschiedenen Datentypen in den for-Loops dient.

https://go.dev/play/p/h7VkPNB4T38

https://go.dev/play/p/QqWrZ__fsvi

Lektion 18 – Polymorphismus: Interfaces sind Schnittstellen - der Name ist Programm

Interfaces sind eine Besonderheit von GO. Interfaces dienen dazu, eine Eigenschaft vieler Objekt-orientierter Programmiersprachen herzustellen: Polymorphismus.

Im Grunde können Werte durchaus von verschiedenen Typen sein. Zuerst einmal hat jeder Werte einen Datentyp, der ihm zugewiesen wurde, aber unter bestimmten Gesichtspunkten können verschiedene Werte als auch einem oder mehreren weiteren Datentypen zugehörig betrachtet werden. Das nennt man Polymorphismus und erreicht wird er in GO ausschließlich mit einem Datentyp namens Interface.

Ein Interface ist auch im wirklichen Leben ein Bindeglied zwischen verschiedenen Typen, der eine Verbindung zwischen beiden herstellt. Schauen wir uns ein Beispiel an.

https://go.dev/play/p/HQT40d_4DCB

Das bedeutet, dass wenn ein Typ als vom Typ interface deklariert wird, bedeutet das in etwa so etwas wie: Wenn irgendein Typ folgende Methoden mit folgende Rückgabewerten von bestimmten Typen implementiert, dann betrachtet GO ihn auch als vom Typ dieses Interfaces.

Man kann also einen übergeordneten Typ erstellen, von dem man ganz generisch nur verlangt, dass egal was er sonst so macht, er auch auch von eben diesem Typ ist, wenn er bestimmte Bedingungen erfüllt.

Jetzt wird auch klar, warum interface{} “empty interface” so etwas wie “alle Typen” bedeutet. Alles, an das keine Bedingung zur Erfüllung des Typs “empty interface” geknüpft ist, ist auch vom Typ “empty interface”. Es gibt also keine Einschränkung vom Typ “empty interface” zu sein und das trifft auf alle Datentypen zu.

Lektion 19 – GO modules

Eine kleine Übersicht über die Nutzung der eingebauten Paketverwaltung in GO: GO modules

Das Package-Management ist von entscheidender Bedeutung für den Entwicklungsprozess in jeder Programmiersprache. Mit der Einführung von GO-modules hat GO einen großen Schritt in Richtung Verwaltung von Paketen und Abhängigkeiten gemacht. GO-modules bietet eine Möglichkeit, die Abhängigkeiten Ihres Projekts zu verwalten, was es einfacher macht, Ihren Code mit anderen zu teilen und an Projekten zu arbeiten. Sie helfen auch dabei, sicherzustellen, dass die richtige Version eines Pakets verwendet wird und reduzieren das Risiko von Kompatibilitätsproblemen. GO-modules machen es einfach, den Überblick über Ihre Abhängigkeiten zu behalten, was das Upgraden oder Zurückkehren zu früheren Versionen einfacher

macht. Mit GO-modules können Sie sicher sein, dass Ihr Code wie erwartet funktioniert, unabhängig von Änderungen an Ihren Abhängigkeiten.

GO modules

Lektion 20 – Channels sind die Schlüssel zu nebenläufiger Kommunikation in GO

Wir erkunden die faszinierende Welt von Channels in GO, eines leistungsstarken Konzepts, das es Ihnen ermöglicht, Kommunikation zwischen mehreren GO-Routinen zu ermöglichen und zu synchronisieren.

In diesem Code haben wir einen Kanal c mit der Funktion make erstellt. Wir haben dann eine GO-Routine verwendet, um den Wert 42 an den Kanal c zu senden. Schließlich haben wir im Hauptprogramm den aus dem Kanal c empfangenen Wert mit dem Operator <- ausgegeben.

Wie Sie sehen, ermöglichen Channels es Ihnen, Daten sicher zwischen GO-Routinen zu übertragen. Aber das ist erst der Anfang! Channels ermöglichen es Ihnen auch, den Informationsfluss zu steuern, die Daten zu puffern und den Kanal sogar zu schließen, um das Ende der Kommunikation zu signalisieren.

Wenn Ihnen so ein simples Beispiel nicht reicht, um das zumindest das Konzept von Channels zu verstehen, werfen Sie einen Blick auf https://gobyexample.com/channels

Beim Verwenden von Channels in GO gibt es mehrere wichtige Überlegungen zu beachten:

- Kanalrichtung: Channels können bidirektional oder unidirektional sein. Es ist wichtig, die richtige Richtung für Ihren Anwendungsfall zu wählen, da sie den Datentransport beeinflusst.
- Puffergröße: Channels können gepuffert oder nicht gepuffert sein. Gepufferte Channels können Daten ohne Blockierung speichern, während nicht gepufferte Channels blockieren, bis die Daten empfangen werden. Es ist wichtig, die richtige Puffergröße für Ihren Anwendungsfall zu wählen, um eine effiziente Kommunikation zu gewährleisten.
- Schließen von Channels: Channels können geschlossen werden, um das Ende der Kommunikation zu signalisieren. Wenn ein Kanal geschlossen ist, führen alle Versuche, Daten an ihn zu senden, zu einer Panik. Es ist wichtig, das Schließen von Channels korrekt zu behandeln, um unerwartetes Verhalten zu vermeiden.
- Synchronisation: Channels können verwendet werden, um GO-Routinen zu synchronisieren. Es muss jedoch darauf geachtet werden, Deadlocks zu vermeiden, bei denen zwei oder mehr GO-Routinen aufeinander warten.
- Datentypen: Channels können in GO jeden Datentyp übertragen, aber es ist wichtig sicherzustellen, dass der richtige Datentyp übertragen und empfangen wird.

Insgesamt ist es wichtig, die Eigenschaften und das Verhalten von Channels in GO zu verstehen, um sie effektiv in Ihren Projekten verwenden zu können.

Code auf dem Playground ansehen

Lektion 21 – Import und Export von Daten im JSON Format

Der Austausch von Daten zwischen Programmen, die in verschiedenen Programmiersprachen geschrieben wurden, ist eine gängige Praxis in der Softwareentwicklung. Eine weit verbreitete Methode für diesen Zweck ist die Verwendung des JSON-Formats (JavaScript Object Notation), eines leichtgewichtigen und textbasierten Datenaustauschformats. JSON ist ein Standardformat zur Darstellung von Daten, das einfach zu lesen und zu schreiben ist und von fast allen modernen Programmiersprachen interpretiert und generiert werden kann.

In GO können Daten in verschiedenen Formaten wie CSV, JSON, XML und anderen importiert und exportiert werden. Ich zeige Ihnen zwei einfache Beispiele für den Import und Export von Daten im JSON-Format.

Importieren von Daten aus JSON

Um Daten aus einer JSON-Datei zu importieren, können wir das Paket `encoding/json` verwenden, das in der Standardbibliothek von GO enthalten ist.

Nehmen wir an, wir haben eine JSON-Datei namens `data.json` mit dem folgenden Inhalt:

```
[
  {
    "name": "Stan",
    "age": 42,
    "gender": "Male"
  },
  {
    "name": "Francine",
    "age": 37,
    "gender": "Female"
  }
]
```

Wir können diese Daten wie folgt in unser GO-Programm importieren:

```
package main

import (
     "encoding/json"
     "fmt"
     "os"
```

```
)

type Person struct {
     Name   string `json:"name"`
     Age    int    `json:"age"`
     Gender string `json:"gender"`
}

func main() {
     file, err := os.Open("data.json")
     if err != nil {
          panic(err)
     }
     defer file.Close()

     decoder := json.NewDecoder(file)
     var people []Person
     if err := decoder.Decode(&people); err != nil {
          panic(err)
     }

     fmt.Println(people)
}

    fmt.Println(people)
}
```

Hier öffnen wir zunächst die JSON-Datei mit der Funktion `os.Open` und überprüfen sie auf Fehler. Dann erstellen wir einen neuen `json.Decoder` unter Verwendung der geöffneten Datei und deklarieren ein Slice von Person structs, um die dekodierten Daten zu speichern. Schließlich dekodieren wir die JSON-Daten mit der Decode-Methode und geben das Ergebnis auf der Konsole aus. Beachten Sie, dass wir einen Verweis auf das Personen-Slice übergeben, damit die Decode-Methode es mit den dekodierten Daten auffüllen kann.

Exportieren von Daten nach JSON

Um Daten in eine JSON-Datei zu exportieren, können wir das Paket `encoding/json` verwenden, das in der Standardbibliothek von GO enthalten ist.

Angenommen, wir haben ein Slice von Structs, das einige Daten repräsentiert:

```
type Person struct {
     Name   string `json:"name"`
     Age    int    `json:"age"`
     Gender string `json:"gender"`
}

people := []Person{
     {"Peter", 44, "Male"},
     {"Lois", 43, "Female"},
}
```

Wir können diese Daten wie folgt in eine JSON-Datei namens `data.json` exportieren:

```
package main

import (
    "encoding/json"
    "fmt"
    "os"
)

type Person struct {
    Name   string `json:"name"`
    Age    int    `json:"age"`
    Gender string `json:"gender"`
}

func main() {
    people := []Person{
```

```
        {"John", 25, "Male"},
        {"Mary", 31, "Female"},
    }

    file, err := os.Create("data.json")
    if err != nil {
        panic(err)
    }
    defer file.Close()

    encoder := json.NewEncoder(file)
    encoder.SetIndent("", "  ")
    if err := encoder.Encode(people); err != nil {
        panic(err)
    }

    fmt.Println("Data exported to data.json")
}
```

Hier erstellen wir zunächst ein Slice von `Person` structs, das die Daten repräsentiert, die wir exportieren wollen. Dann erstellen wir eine neue Datei mit `os.Create` und prüfen auf Fehler. Wir erstellen einen neuen `json.Encoder` unter Verwendung der erstellten Datei und setzen die Einrückung für die Ausgabe mit der Methode `SetIndent`. Schließlich kodieren wir die Daten mit der Methode `Encode` und geben eine Meldung auf der Konsole aus, die den Export bestätigt.

Lektion 22 – Unit Tests

Hier ist ein Beispiel dafür, wie man Unit-Tests in GO mit dem eingebauten Testing-Package durchführt.

Nehmen wir an, wir haben eine einfache Funktion, die zwei ganze Zahlen addiert:

```
func Add(a, b int) int {
    return a + b
}
```

Um einen Unit-Test für diese Funktion zu schreiben, können wir eine neue Datei mit dem Namen `add_test.go` im gleichen Paketverzeichnis erstellen, mit folgendem Inhalt:

```
package main

import "testing"

func TestAdd(t *testing.T) {
     // Test cases
     cases := []struct {
          a, b, want int
     }{
          {1, 2, 3},
          {0, 0, 0},
          {-1, 1, 0},
          {-1, -1, -2},
     }
     // Iterate over test cases
     for _, c := range cases {
          got := Add(c.a, c.b)
          if got != c.want {
               t.Errorf("Add(%d, %d) == %d, want %d", c.a, c.b,
got, c.want)
          }
     }
}
```

Dieser Code definiert eine Testfunktion namens `TestAdd`, die eine Reihe von Testfällen ausführt, von denen jeder aus zwei ganzen Zahlen und der erwarteten Ausgabe der Funktion `Add` besteht. Die Testfunktion verwendet die Methode `t.Errorf`, um einen Fehler zu melden, wenn die Funktion ein unerwartetes Ergebnis liefert.

Um den Test auszuführen, können wir den Befehl `go test` im Terminal aus dem Paketverzeichnis verwenden:

```
go test .
```

Dadurch werden alle Testfunktionen des Pakets ausgeführt und die Ergebnisse gemeldet. In diesem Fall, wenn alles richtig funktioniert, sollten wir eine Ausgabe wie die folgende sehen:

```
PASS
ok      _/path/to/package    0.001s
```

Wenn ein Testfall fehlschlägt, meldet `go test` einen Fehler mit einer Meldung, die die Details des Fehlers enthält.

Das ist ein einfaches Beispiel dafür, wie man Tests in golang durchführt.

Zusammengefasst

golang hat ein eingebautes Testing-Package, das einen einfachen und konsistenten Rahmen für das Schreiben und Ausführen von Tests bietet.

Testdateien in golang folgen der Namenskonvention `*_test.go` und befinden sich im gleichen Paketverzeichnis wie der zu testende Code.

Testfunktionen in golang beginnen mit dem Präfix Test und nehmen ein einzelnes Argument des Typs `*testing.T`, welches verwendet wird, um Testfehler und Logmeldungen zu melden.

Testfunktionen sollten gut strukturiert und einfach zu lesen sein, mit klaren und beschreibenden Testfällen, die alle Aspekte des zu testenden Codes abdecken.

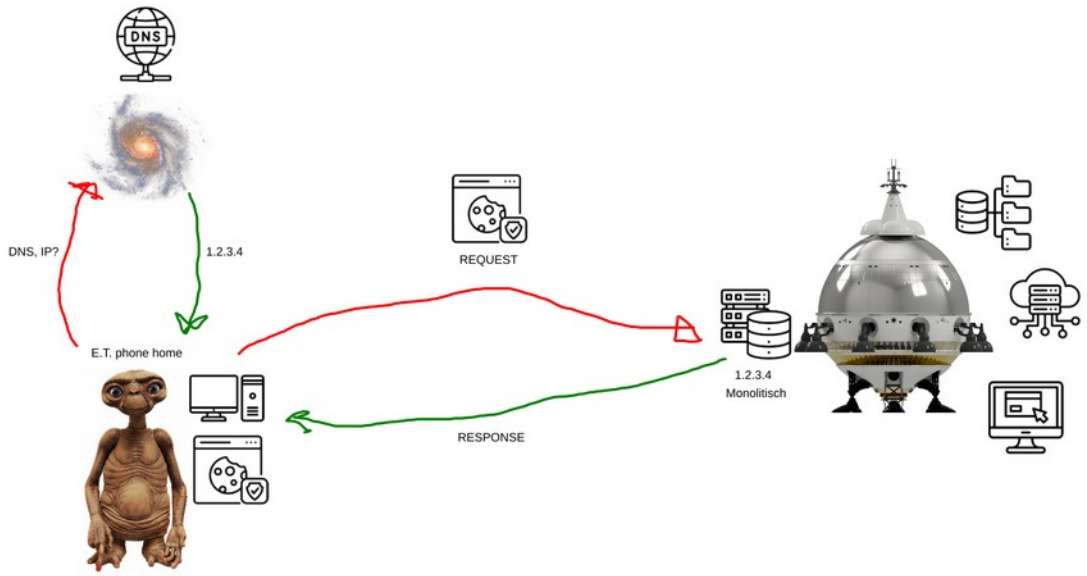
golang unterstützt eine Vielzahl von Testmethoden und -werkzeugen, einschließlich Unit-Tests, Integrationstests, Benchmarking und Code-Abdeckungsanalyse.

Wenn Sie diese Richtlinien befolgen, können Sie effektive Tests für Ihren golang-Code schreiben und sicherstellen, dass Ihre Software zuverlässig und robust ist.

Abschnitt 3 – Einfache Webanwendung - der Anfang

Things work in cycles.
Joan Jett

Lektion 23 – Der HTTP-Anforderungs-/Antwort-Zyklus



Lektion 24 – Die erste Webanwendung "Es lebt! Es lebt!"

Dieser Code in GO richtet einen einfachen HTTP-Server ein, der auf Port 8080 lauscht und auf jede HTTP-Anfrage an den Root-Pfad ("/") mit der Nachricht "Hello, 世界" (was "Hallo, Welt" auf Chinesisch bedeutet) antwortet.

Die Methode `http.HandleFunc()` richtet eine Handler-Funktion ein, die zwei Argumente annimmt: eine `http.ResponseWriter`-Schnittstelle und einen `*http.Request`-Zeiger. Wenn eine Anfrage an den Root-Pfad gestellt wird, wird diese Funktion aufgerufen und schreibt die Nachricht "Hello, 世界" in den `http.ResponseWriter`. Die Funktion `fmt.Fprintf()` wird verwendet, um die Nachricht in den Antwort-Writer zu schreiben und gibt die Anzahl der geschriebenen Bytes und mögliche Fehler während des Schreibvorgangs zurück.

Die letzte Zeile der `main()`-Funktion startet den HTTP-Server, indem die Methode `http.ListenAndServe()` aufgerufen wird, die auf dem angegebenen Port lauscht und

eingehende Anfragen mit der registrierten Handler-Funktion bedient. Das Unterstrich-Zeichen vor dem Aufruf von `http.ListenAndServe()` wird verwendet, um den zurückgegebenen Fehlerwert zu ignorieren, was eine häufige Methode ist, um Fehler zu behandeln, die nicht erwartet werden.

v1.0.24

Lektion 25 – Losgelassen: Handler arbeiten jetzt mit der Kraft von Funktionen!

Unser Webserver beantwortet jetzt auf Anfragen für zwei URLs: "/" und "/about".

Die URL "/" wird von der Funktion `Home` behandelt, die einfach einen String an den `ResponseWriter` schreibt. Die URL "/about" wird von der Funktion `About` behandelt, die die `getData`-Funktion aufruft, um einen Namen und einen Spruch zu erhalten, und die `addValues`-Funktion aufruft, um zwei Zahlen zu addieren. Die Ergebnisse dieser Funktionsaufrufe werden dann in einem formatierten String an den `ResponseWriter` geschrieben.

Die `main()`-Funktion richtet die Handler für die beiden URLs ein, startet den Webserver und gibt eine Meldung aus, die anzeigt, dass die Anwendung gestartet wurde.

v1.0.25

Lektion 26 – Errors: Fehler haben und sind ein Wert

Eine segensreiche Möglichkeit in der Programmiersprache GO ist die Eigenschaft von Funktionen, zwei oder mehr Rückgabewerte zu liefern. Das gab den Designern die Möglichkeit, Fehler dort zu behandeln, wo sie potentiell auftreten, nämlich während der eigentlichen Operation, anstatt jeden möglichen Fehler zu einem potentiellen Programmabsturz eskalieren zu lassen, und dann gesammelt abzuhandeln. Und um das zu erreichen, gibt es sogar einen eigenen Datentyp `error`, auf den man prüfen kann. Das schauen wir uns im nachfolgenden Beispiel an.

v1.0.26

Lektion 27 – HTML-Vorlagen: weil niemand die Zeit hat, das Rad neu zu erfinden!

Bisher haben wir ja nur einfachen Text ausgeliefert, aber GO wurde ja von Google entwickelt und ist mit einer umfangreichen Standardbibliothek ausgestattet um, - ja - um alles im Netz zu tun, was Google halt so macht. Und dazu gehört auch das Ausliefern von HTML, und dort wiederum gibt es ein Package, das Templates, also Vorlagen, dafür zur Verfügung stellt. Und wie man damit umgeht, schauen wir uns jetzt an.

Ein Hinweis: Die Einbindung des Package `text/template` diente hier jetzt eher Lehrzwecken, um die Vorgehensweise im Umgang mit HTML-Templates in GO zu verdeutlichen. Später benutzen wie dazu ein etwas fortgeschrittenes Package.

v1.0.27

Lektion 28 – Organisieren und Erobern: Lassen Sie uns aufräumen und unseren Platz optimieren!

Wir haben den Code aufgeräumt, auf verschiedene Dateien aufgeteilt und schließlich Bootstrap in unsere HTML-Templates integriert.

Bootstrap

v1.0.28

Lektion 29 – Restrukturierungsmaßnahmen – Struktur wie aus dem Lehrbuch

Wir erstellen eine Ordnerstruktur, die von `go modules` unterstützt wird, betrachten unsere Programmteile, die wir auf verschiedene Dateien verteilt haben, als Packages – wie die von Drittanbietern. Wir verschieben unsere Anwendung und unsere neuen Packages in die entsprechenden Ordner und nehmen die notwendigen Änderungen im Code vor, damit `go modules` sie entsprechend finden und importieren kann.

```
myGOWebApplication
├── cmd
│   └── web
│       └── main.go
├── go.mod
├── LICENSE
├── pkg
│   ├── handlers
│   │   └── handlers.go
│   └── render
│       └── render.go
├── README.md
└── templates
    ├── about-page.tpml
    └── home-page.tpml
```

v1.0.29

Lektion 30 – Layouts wie ein Boss

Wir erweitern unsere Templates und nutzen Layouts, um daraus generisch Webseiten für unsere Responses zu generieren.

v1.0.30

Lektion 31 – Dynamischer Cache für effektive Template-Verarbeitung

Lernen Sie eine Methode kennen, einen einfachen dynamisch wachsenden Cache aufzubauen.

v1.0.31

Lektion 32 – Statischer Cache #1: effiziente Template-Verarbeitung

Ein statischer Cache für Templates bietet Vorteile für unsere Webanwendung.

v1.0.32

Lektion 33 – Statischer Cache #2: Konfigurationsdatei für globale Variablen einführen

Wir nutzen eine globale Variable, die wir in einer Konfigurationsdatei verfügbar machen.

v1.0.33

Lektion 34 – Statischer Cache #3: Finaler Schritt zur Implementierung mit globaler Variable

Wir nutzen die in der Konfigurationsdatei definierte globale Variablen, um einen statischen Cache zu erstellen, den man ein- und ausschalten kann.

v1.0.34

Lektion 35 – Was man mit einer Konfigurationsdatei sonst noch anfangen kann

Hinweise zur Nutzung einer Konfigurationsdatei.

Lektion 36 – Geteilte Freude ist doppelte Freude: Daten teilen mit Templates

Lernen Sie, wie man Daten in Templates verfügbar macht und sie dort einsetzt.

v1.0.36

Abschnitt 4 – GO with the Flow: Eine Übersicht über Middlewares und Sessions in GO!

> *A horse is dangerous at both ends and uncomfortable in the middle.*
> *Ian Fleming*

Lektion 37 – Einführung von Middleware/Routing Packages in GO

https://en.wikipedia.org/wiki/Multiplexer

https://dev.to/karankumarshreds/middlewares-in-go-41j

https://www.alexedwards.net/blog/which-go-router-should-i-use

Lektion 38 – Implementierung eines einfachen Routers (bmizerany/pat)

Wir importieren unser erstes externes Package und erstellen damit Routen.

bmizerany/pat

v1.0.38

Lektion 39 – Entwicklers Liebling: go-chi/chi als neues externes Router Package

Wechsel von pat zu chi als Routing und Middleware Package.

go-chi/chi

v1.0.39

Lektion 40 – Middleware: Basteln Sie eigene und werden Sie der coolste Coder der Stadt

Lernen Sie, wie man Middleware in GO mit einer schnellen Übung erstellt und werden Sie in kürzester Zeit ein Middleware-Meister!

justinas/nosurf

v1.0.40

Lektion 41 – State Management mit Session(s)

Implementierung einer Middleware in Form eines Paketes zur Erstellung und Verwaltung von Sessiondaten.

alexedwards/scs

v1.0.41

Lektion 42 – Funktionstest für Sessiondaten

Ein kurzer Test, ob die Übergabe von in Sessions eingebetteter Daten funktioniert.

v1.0.42

Abschnitt 5 – Projektauswahl und Arbeiten mit Forms: Ein papierloser Traum!

May your choices reflect your hopes, not your fears.
Nelson Mandela

Lektion 43 – Gedanken zur Projektauswahl

Lernen Sie, ein Projekt grob zu skizzieren und den Umfang und Arbeitsaufwand zu schätzen.

Planung

I. Was soll unsere Webanwendung sein?

II. Wie groß ist der Umfang?

III. Was sind die grundlegenden Funktionalitäten?

I. Webanwendung

- Buchung/Reservierung
- Zwei Objekte/Services

II. Umfang der Webanwendung

- Landingpage oder Showcase-Seite
- Einzelseiten für die Vermietungsgsachen
- Verfügbarkeitsabfrage
- Buchungs-/Reservierungsausführung
- Benachrichtigungssystem

III. Funktionalitäten

- backend & Login
- Übersicht der aktuellen Buchungen
- Kalender der kommenden und/oder vergangener Buchungen
- Änderungen/Stornierung einer Buchung/Reservierung

Was müssen wir umsetzen?

- Sichere Authentifizierung (Login)

- Datenspeicherung, also Datenbank

- Mitteilungssystem (SMS oder E-Mail)

Lektion 44 – Kurzer Hinweis zu Github

Ein kurzer Hinweis, warum die Nutzung von github.com eine gute Idee sein könnte.

Lektion 45 – Statische Dateien: Halt still und lass dich einbinden!

Statische Dateien alle aus einem bestimmten Ordner servieren ist einfach:

Pixabay

v1.0.45

Lektion 46 – HTML - Ausflug in die 1990er

Ein simples Beispiel wie HTML funktioniert.

v1.0.46

Lektion 47 – Punktlandung! Wir erstellen eine Landing Page

Wir richten eine serverlose Entwicklungsumgebung ein, um HTML-Seiten zu erstellen, die wir später in GO Templates umwandeln werden. Wir starten mit der Landing Page.

v1.0.47

Lektion 48 – Erstellung der HTML-Seiten der Bungalows

Mit einer einfachen Landing Page sind die Bungalow-Seiten 1-2-3-Fertig!

v1.0.48

Lektion 49 – Erstellen und Aufpeppen einer HTML-Seite zur Verfügbarkeitsprüfung

Erstellung einer HTML-Seite mit einem Formular, um Daten vom Benutzer zu bekommen.

v1.0.49

Lektion 50 – make-reservation.html ist unsere Antwort auf: "Haben Sie eine Reservierung?"

Ein HTML-Formular, um eine Reservierung zu machen.

v1.0.50

Abschnitt 6 – Code-Kaboom! JavaScript und CSS kommen ins Spiel!

> *Design is not just what it looks like and feels like. Design is how it works.*
>
> *Steve Jobs*

Lektion 51 – JavaScript: Freund oder Feind?

v1.0.51

Lektion 52 – Mühelose Datumsauswahl: Holen Sie sich jetzt ein Vanilla JavaScript Datumsauswahl-Paket!

Wir übernehmen die Kontrolle über die Datumsauswahl durch Import eines (Vanilla) JavaScript Pakets eines Drittanbieters.

Vanilla JavaScript Datepicker

v1.0.52

Lektion 53 – Notie by Nature: Einfache Mitteilungen einblenden

Implentierung des JavaScript Pakets „notie“.

notie

v1.0.53

Lektion 54 – Sweetalert: Zeit für Süßkram

Implementierung des JavaScript Pakets „sweetalert2“.

sweetalert2

v1.0.54

Lektion 55 – Sweetalert Candystore - ein eigenes JavaScript Modul

Wir erstellen unser eigenes JavaScript Modul.

JavaScript modules:
developer.mozilla.org
www.freecodecamp.org

v1.0.55

Lektion 56 – Vom langweiligen Button zum Superstar: Die Implementierung einer neuen Funktion in unserem JavaScript-Modul!

Wir implementieren eine neue Funktion in unserem JavaScript-Modul und versuchen, diese mit einem Button bereitzustellen.

v1.0.56

Lektion 57 – CSS: Webseiten weniger hässlich machen seit 1996

Eigene CSS in eine lokale Datei auslagern, CSS Importe sortieren und ein kurzer Blick auf die Funktionsweise von CSS.

v1.0.57

Abschnitt 7 – HTML in GO Templates verwandeln, serverseitige Validierung und noch mehr Handler

> *Coming together is a beginning, staying together is progress, and working together is success.*
>
> *Henry Ford*

Lektion 58 – Kurze Übersicht, was in diesem Abschnitt passiert

v1.0.58 (Sie können mit diesem Code in die nächste Lektion starten, der bereits von einigen Altlasten aus den vorherigen Lektionen befreit wurde)

Lektion 59 – Konvertierung zu GO Templates: Von HTML zu "Glücklich bis ans Ende der Zeit"

Erfahren Sie wie Sie aus einfachem HTML Templates zur Nutzung in GO erstellen.

v1.0.59

Lektion 60 – CSRF Token – Implementierung

Wir sorgen für Sicherheit gegen Cross-Site Request Forgery (CSRF), indem wir mit der Middleware NoSurf, für jeden potentiellen POST-Request einen Token generieren.

v1.0.60

Lektion 61 – Die Macht von JSON in golang entfesselt: Ein Handler, gibt Daten in JSON zurück!

Erfahren Sie, wie Sie mit golang einen Handler erstellen können, der Daten im JSON-Format zurück gibt.

v1.0.61

Lektion 62 – Vorbereitungen für die Übertragung und Verarbeitung von AJAX-Requests

In dieser Lektion lernen Sie, wie eine Anfrage via JavaScript an unsere Webanwendung eine Antwort von unseren JSON-Daten liefernden Handler auslösen kann und dessen Antwort zu interpretieren und die Daten in JavaScript zu verwenden!

v1.0.62

Lektion 63 – Von GET zu POST: Bringen wir den AJAX-Anfragen ein paar Manieren bei!

Darüber hinaus lernen Sie, die Funktion `custom()` in JavaScript anzupassen, damit sie mehr Allgemeingültigkeit hat.

v1.0.63

Lektion 64 – Pimp Your Code: Refaktorisierung leicht gemacht!

Halten Sie die Struktur Ihres Codes sauber - führen Sie eine Refaktorisierung durch!

v1.0.64

Lektion 65 – Serverseitige Validierung - das Was, das Wie und das Warum überhaupt!

Eine kleine Übersicht, was serverseitige Validierung ist und warum sie in vielen Fällen mehr Sinn macht als clientseitige Validierung (allein).

Client-side vs. Server-side Validation (StackOverflow)
Client-side vs. Server-side Validation (Medium)

Lektion 66 – Implementierung der serverseitigen Validierung I - Formularfelddaten und Fehler

Werten Sie Formulardaten aus und erkennen Sie Fehler.

v1.0.66

Lektion 67 – Implementierung der serverseitigen Validierung II - Forms Modell & Fehleranzeige

Erstellen Sie ein Packages `model`, das Datenmodelle hält und erstellen Sie neue Validatoren.

v1.0.67

Lektion 68 – Implementierung der serverseitigen Validierung III - Mehr Felder & Required Func

Erstellen Sie weitere Felder und machen Sie gleich mehrere davon zu „notwendigen“ Feldern.

v1.0.68

Lektion 69 – Implementierung der serverseitigen Validierung IV - mehr Validators & Package govalidator

Lernen Sie wie Sie ein externes Package nutzen können, das weitere Validatoren zur Verfügung stellt.

v1.0.69

Lektion 70 – Anzeige einer Übersicht der Reservierungsdaten (mittels Sessions)

Nutzen Sie Sessions, um die gesammelten Reservierungsdaten nach der Reservierung auf einer Übersichtsseite anzuzeigen.

v1.0.70

Lektion 71 – Kurze Rückmeldung: Alerts als Feedback an den Benutzer mittels Notie ausgeben

Nutzen Sie `notie`, um dem Benutzer Feedback zu seinen Aktionen zu geben.

v1.0.71

Lektion 72 – Alternative Template Engine: Nutzen Sie die Kraft eines Jets

Vorstellung einer alternativen Template Engine.

Templates in der Standard Library

Performance Vergleich einiger GO Template Engines (2020)

Die CloudyKit/jet Template Engine

Beispiel aus dieser Lektion kann auch in den Resourcen zu dieser Lektion heruntergeladen werden.

Beispielcode für die Nutzung der Jet Template Engine

Abschnitt 8 – Code auf Herz und Nieren prüfen: Tests retten den Tag (oder Tage!)

> *A good test is one that has a high probability of breaking the code if there is a problem*
>
> *Michael Bolton*

Lektion 73 – Testing in GO: Das Warum und das Wieso

Über den Sinn und Zweck, aber auch die Wichtigkeit von Tests in GO.

Mein GO/golang-Kurs für Beginner

Lektion 74 – Erfolgreich testen: Tests für das Package main unserer Webanwendung

Lernen Sie wie Sie das Package `main` Ihrer Webanwendung testeten können, wenn der Test selbst eine Funktion `main()` ausführt.

v1.0.74

Lektion 75 – Handlers-Tests I - der Anfang: Ersteinrichtung/Handhabung GET-Request-Handlers

Lernen Sie, wie Sie GET-Request-Handler testen.

v1.0.75

Lektion 76 – Handlers-Tests II - Fortsetzung: Handhabung POST-Request-Handlers

Lernen Sie wie Sie POST-Request-Handler testen.

v1.0.76

Lektion 77 – Render-Tests I - Erstellen der Testumgebung und Funktion TestAddDefaultData()

Um ein Package unter immer gleichen Umgebungen zu testen, ist es nötig eine solche Umgebung anzulegen.

v1.0.77

Lektion 78 – Render-Tests II - Erstellen von Tests für Funktion TestRenderTemplate() und den Rest

Lernen Sie wie sie auch für die anderen Funktionen im Package `render` Tests hinzufügen.

v1.0.78

Lektion 79 – Abdeckung der Tests von Package handlers und Package render

Werfen Sie einen Blick auf die Abdeckung des Tests für die Packages `handlers` und `render`.

Lektion 80 – Praxisübung: Schreiben Sie einen grundlegenden Test für Package forms

Schreiben Sie einen grundlegenden Test für das Package `forms`.

v1.0.80

Lektion 81 – Beispiellösung: [SOLVED] Testing für Package forms

Eine Lösung für Tests der Funktionen im Package `forms`.

v1.0.81

Lektion 82 – Abschließende Hinweise und Tipps für den Aufruf unserer Webanwendung

Gerne können Sie sich eine Batchdatei oder ein Script schreiben, um den Aufruf Ihrer Anwendung zu vereinfachen.

Abschnitt 9 – Streben nach Verbesserung: Error Handling

> *Better three hours too soon than a minute too late.*
>
> *William Shakespear*

Lektion 83 – Konsolidierung der Fehlerbehandlung in einem Package "helpers"

Lernen Sie, wie Sie einen Teil der Fehlercodeauswertung in einem Package `helpers` zusammenführen und zentral auswerten können.

v1.0.83

Lektion 84 – Anwendung von ClientError und ServerError und Updates der relevanten Tests

Lernen Sie wie man Fehlercodes auswertet und zwischen Client-Fehler und Server-Fehler unterscheidet.

v1.0.84

Abschnitt 10 – Datenbank I - Einführung in Datenbanknutzung und SQL mit PostgreSQL und DBeaver

> *You can have data without information, but you cannot have information without data.*
>
> *Daniel Keys Moran*

Lektion 85 – Kurze Abschnittsübersicht und Download/Installation von PostgreSQL und DBeaver

- My Free Training on Udemy Running a Simple Webserver Performing CRUD Actions on a PostgreSQL Database Server
- PostgreSQL Server Download
- Postgres.app Download (nur macOS)
- DBeaver Community Edition (CE) Download
- Vergleich PostgreSQL - mySQL

Lektion 86 – Linux: Installation von PostgreSQL und DBeaver und Verbindungserstellung

Hier Debian-basierte Distributionen, für SUSE-basierte Distributionen analog mit Paketverwaltung Yum:

```
sudo apt-get update
sudo apt-get install -y postgresql dbeaver

service postgresql start
service postgresql status
```

Und nun starten wir als User `postgres` ein Administrations-Tool namens PSQL. Im Grunde öffnen wir ein Terminal in den Postgres-Server. Ignorieren Sie die Fehlermeldung hier, die im Grunde nur besagt, dass der Benutzer Postgres in unserem Home-Verzeichnis keine Rechte besitzt.

```
sudo -u postgres psql
```

PostgreSQL-Terminal:

```
\password
exit
```

Starten Sie DBeaver (hier vom Terminal aus, natürlich können Sie das auch aus dem Startmenü wählen):

```
dbeaver
```

Neue Verbindung zu PostgreSQL erstellen unter Verwendung von User `postgres` und dem neu erstellten Passwort.

Lektion 87 – macOS: Installation von PostgreSQL und DBeaver und Verbindungserstellung

PostgreSQL: Download von standalone Postgres.App und Installation durch Drag&Drop in Applications.

Postgres.app download

Postgres.App starten, Passwort festlegen.

DBeaver: Installer herunterladen und ausführen, neue Verbindung zu PostgreSQL erstellen (eventuell notwendige Treiber herunterladen) unter Verwendung von User `postgres` und dem neu erstellten Passwort.

Lektion 88 – Windows: Installation von PostgreSQL und DBeaver und Verbindungserstellung

PostgreSQL: Installer für PostgreSQL herunterladen und ausführen. EULA abnicken, Standards akzeptieren, Zugriff erlauben, Passwort setzen. PostgreSQL-Server sollte im Hintergrund starten.

DBeaver: Installer herunterladen und ausführen, neue Verbindung zu PostgreSQL erstellen (eventuell notwendige Treiber herunterladen) unter Verwendung von User `postgres` und dem neu erstellten Passwort.

Lektion 89 – CRUD - Jetzt wird's schmutzig! SQL Statements in Aktion

Lernen Sie den grundsätzlichen Umgang mit den notwendigen Aktionen, die Sie in Datenbanken ausführen können möchten:

- **C**reate (Erstellen)
- **R**ead/**R**etrieve (Auslesen)
- **U**pdate (Ändern)
- **D**elete/**D**estroy (Löschen)

zusammen bekannt als **CRUD**.

Lektion 90 – SQL-Abfragen für Fortgeschrittene - nicht unbedingt komplizierter, aber komplexer

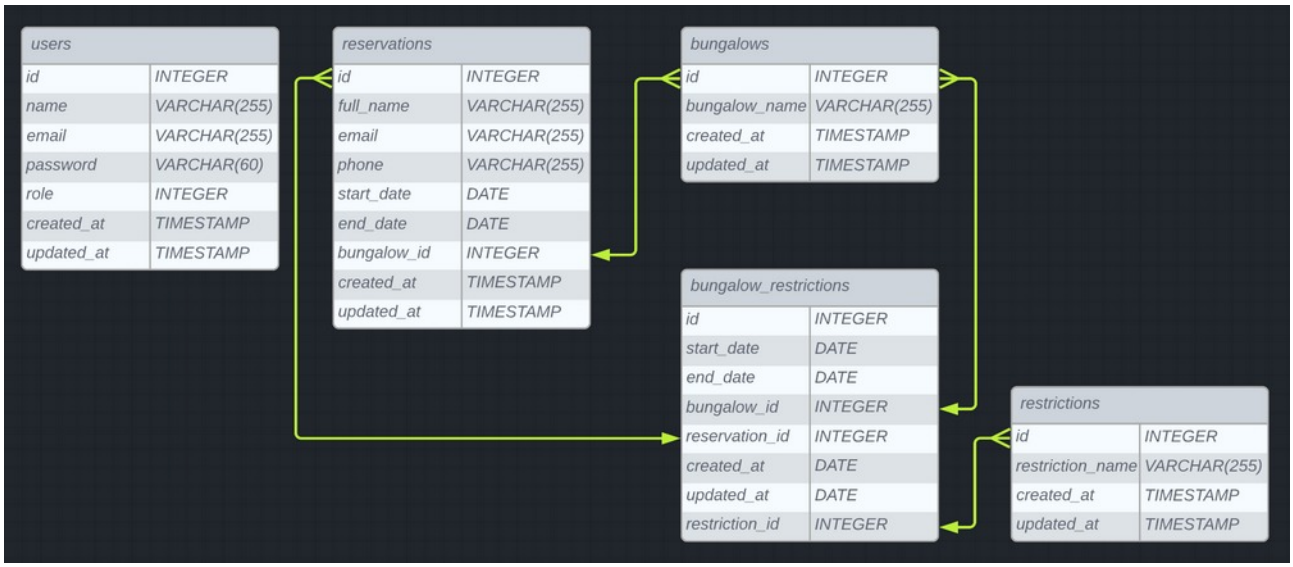
Lernen Sie über die CRUD-Anweisungen hinaus, Klauseln und Optionen kennen, um komplexere Datenbankabfragen zu erstellen.

Abschnitt 11 – Datenbank II - Erstellung und notwendige Strukturierung der Datenbank

> *Structure is more important than content in the transmission of information.*
>
> *Abbie Hoffman*

Lektion 91 – Faszination Datenbankstruktur: Entity Relationship Diagram erstellen!

Ein kurzer Einblick in die Erstellung eines Entity-Relationship-Diagram.



Lektion 92 – *pop* "Soda gefällig?" - Installation von gobuffalo/pop, auch Soda genannt!

Wir kompilieren und installieren ein Kommandozeilen-Tool zur Datenbank-"Druckbetankung" heraus gelöst aus dem Webdeveloper Framework gobuffalo!

Komplettes Framework gobuffalo für Webentwicklung in GO:

Installation für das komplette Framework (nicht nötig für den Umfang dieses Kurses)

Wir wollen aber nur einen Teil nutzen namens Migrations: Automatisierung und „Versionsverwaltung“ für Datenbanken.

Und damit wir nicht das komplette Framework installieren, greifen wir auf ein Tool für die Kommandozeile zurück, das im Buffalo Framework enthalten ist und dort „pop“ heißt.

Aber dieses „pop“-Kommando kann man sich auch als standalone-Version herunterladen, bzw. installieren, dann heißt es „Soda“. Und dieses Soda steht entweder als Binary für die gängigen Betriebssysteme zur Verfügung, oder aber, wenn Sie GO installiert haben, ist eine Sache von wenigen Minuten, es zu installieren.

Sie sollten Sie in der Lage sein, sich Ihre Werkzeuge – wenn Sie denn schon als GO Quellcode vorliegen - selber zu installieren. Sie sollten GO auch lauffähig zur Verfügung stehen haben.

Wenn Sie auf macOS Homebrew bevorzugen, oder auf welche, Betriebssystem auch immer ein Binary nutzen wollen, ist das beides ebenfalls völlig okay!

Windows

Binary herunterladen und ablegen in einem der folgenden Ordner:

```
C:\Windows
C:\Windows\System
C:\Windows\System32
C:\Users\Username\go\bin
```

Installation:

```
go install github.com/gobuffalo/pop/v6/soda@latest
```

Test:

[WIN + R]

```
cmd.exe
powershell.exe

soda.exe -v
```

macOS

Binary (Architektur beachten!) herunterladen und ablegen in einem der folgenden Ordner:

```
/usr/local/bin
/usr/local/go/bin
```

Installation:

```
go install github.com/gobuffalo/pop/v6/soda@latest
```

Test:

Terminal:

```
which soda
soda -v
```

Linux

Binary herunterladen und ablegen in einem der folgenden Ordner:

```
/usr/local/bin
/usr/local/go/bin
```

Installation:

```
go install github.com/gobuffalo/pop/v6/soda@latest
```

Test:

Terminal:

```
which soda
soda -v
```

Lektion 93 – Migrations I - Erstellung der Tabelle "users"

Erstellen Sie `soda generate` , um eine erste Tabelle in der Datenbank zu erstellen.

Database Setup

Migrations/FIZZ

FIZZ

`database.yml`:

```
development:
  dialect: postgres
  database: mygowebapp
  user: postgres
  password: [PUTYOURPASSWORDHERE]
  host: 127.0.0.1
  pool: 5
```

```
test:
  url: {{envOr "TEST_DATABASE_URL"
"postgres://postgres:postgres@127.0.0.1:5432/myapp_test"}}


production:
  url: {{envOr "DATABASE_URL"
"postgres://postgres:postgres@127.0.0.1:5432/myapp_production"}}
```

soda generate fizz CreateUsersTable

```
UP:
create_table("users") {
t.Column("id", "integer", {primary: true})
t.Column("full_name", "string", {"default": ""})
t.Column("email", "string", {})
t.Column("password", "string", {"size": 60})
t.Column("role", "integer", {"default": 1})
}

DOWN:
drop_table("users")
```

v1.0.93

Lektion 94 – Migrations II - Massenproduktion: Erstellung aller anderen Tabellen

Migrations sorgen dafür, dass Sie effizient alle notwendigen Tabellen für Ihre Webanwendung erstellen können.

soda generate fizz CreateReservationsTable

```
UP:
create_table("reservations") {
  t.Column("id", "integer", {primary: true})
  t.Column("full_name", "string", {"default": ""})
  t.Column("email", "string", {})
  t.Column("phone", "string", {"default": ""})
```

```
  t.Column("start_date", "date", {})
  t.Column("end_date", "date", {})
  t.Column("bungalow_id", "integer", {})
}

DOWN:
drop_table("reservations")
```

soda generate fizz CreateBungalowsTable

```
UP:
create_table("bungalows") {
  t.Column("id", "integer", {primary: true})
  t.Column("bungalow_name", "string", {"default": ""})
}

DOWN:
drop_table("bungalows")
```

soda generate fizz CreateBungalowRestrictionsTable

```
UP:
create_table("bungalow_restrictions") {
  t.Column("id", "integer", {primary: true})
  t.Column("start_date", "date", {})
  t.Column("end_date", "date", {})
  t.Column("bungalow_id", "integer", {})
  t.Column("reservation_id", "integer", {})
  t.Column("restriction_id", "integer", {})
}

DOWN:
drop_table("room_restrictions")
```

soda generate fizz CreateRestrictionssTable

```
UP:
```

```
create_table("restrictions") {
  t.Column("id", "integer", {primary: true})
  t.Column("restriction_name", "string", {"default": ""})
}

DOWN:
drop_table("restrictions")
```

v1.0.94

Lektion 95 – Migrations III - Erstellung eines Foreign Key für Tabelle "reservations"

Auch Beziehungen von Tabelleneinträgen, die sogenannten „Foreign Keys“ untereinander lassen sich mit Migrations bequem erstellen:

`soda generate fizz CreateFkForReservationsTable`

```
UP:
add_foreign_key("reservations", "bungalow_id", {"bungalows": ["id"]}, {
    "on_delete": "cascade",
    "on_update": "cascade",
})

DOWN:
drop_foreign_key("reservations", "reservations_bungalows_id_fk", {})
```

v1.0.95

Lektion 96 – Migrations IV – To Be Continued: Two Foreign Keys for "bungalow_restrictions"

Fortsetzung der Erstellung weiterer Foreign Keys.

`soda generate fizz CreateFkForBungalowRestrictionsTable`

```
UP:
add_foreign_key("bungalow_restrictions", "bungalow_id",
{"bungalows": ["id"]}, {
    "on_delete": "cascade",
    "on_update": "cascade",
})


add_foreign_key("bungalow_restrictions", "restriction_id",
{"restrictions": ["id"]}, {
    "on_delete": "cascade",
    "on_update": "cascade",
})


DOWN:
drop_foreign_key("bungalow_restrictions",
"bungalow_restrictions_restrictions_id_fk", {})


drop_foreign_key("bungalow_restrictions",
"bungalow_restrictions_bungalows_id_fk", {})
```

v1.0.96

Lektion 97 – Praxisübung: Fügen Sie "bungalow_restrictions" den fehlenden Foreign Key hinzu

Eine Übung für Sie: erstellen Sie den noch fehlenden Foreign Key in der Tabelle `bungalow_restrictions`.

Lektion 98 – Beispiellösung: [SOLVED] Der fehlende Foreign Key für "bungalow_restrictions"

Eine Lösung für die Erstellung des Foreign Key in der Tabelle `bungalow_restrictions`.

```
UP:
add_foreign_key("bungalow_restrictions", "reservation_id",
{"reservations": ["id"]}, {
    "on_delete": "cascade",
```

```
    "on_update": "cascade",
})
```

DOWN:

```
drop_foreign_key("bungalow_restrictions",
"bungalow_restrictions_reservation_id_fk", {})
```

v1.0.98

Lektion 99 – Migrations V - Nitro-Einspritzung: Index für "users" und "bungalow_restrictions"

Ein zuvor angelegter Index für häufig abgefragte und umfangreiche Datensätze beschleunigt die Datenbankabfragen ganz erheblich!

`soda generate fizz CreateIndexForBungalowRestrictionsTable`

UP:

```
add_index("bungalow_restrictions", ["start_date", "end_date"], {})
add_index("bungalow_restrictions", ["bungalow_id"], {})
add_index("bungalow_restrictions", ["reservation_id"], {})
```

DOWN:

```
drop_index("bungalow_restrictions",
"bungalow_restrictions_start_date_end_date_idx")
drop_index("bungalow_restrictions",
"bungalow_restrictions_bungalow_id_idx")
drop_index("bungalow_restrictions",
"bungalow_restrictions_reservation_id_idx")
```

v1.0.99

Lektion 100 – Praxisübung: Fügen Sie dem Table "reservations" sinnvolle Indexe hinzu

Überlegen Sie für welche Daten in der Tabelle `reservations` ein Index nützlich wäre und legen Sie diese an.

Lektion 101 – Beispiellösung: [SOLVED] Nützliche Indexe für den Table "reservations"

Ein Lösungsvorschlag für Migrations zur Erstellung je eines Index für die Spalte `email` und `full_name`.

`soda generate fizz CreateIndexForReservationsTable`

```
UP:
add_index("reservations", ["email"], {})
add_index("reservations", ["full_name"], {})

DOWN:
drop_index("reservations", "reservations_email_idx")
drop_index("reservations", "reservations_full_name_idx")
```

v1.0.101

Lektion 102 – Migrations VI - "Der Clou" für die Entwicklungsphase der Datenbank

```
soda reset
```

führt alle „down“-Migrations aus, bevor es alle „up“-Migrations ausführt. Die Datenbank wird so in einen ursprünglichen genau definierten Zustand gebracht.

Abschnitt 12 – Datenbank III - Anbindung einer PostgreSQL Datenbank an eine Webanwendung

> *You don't have to like someone [...] to make a connection.*
>
> *Jennifer Aniston (made up by chatgpt)*

Lektion 103 – Ein Beispiel: Wie eine Anwendung in GO mit einer Datenbank verbunden wird

Kostenloses Training (Udemy, 2h) Erstellung einnes Webservers und Verbindung zu PostgreSQL-Server

Beispiel (von github)

Lektion 104 – PostgreSQL-Verbindung: Wie beim Golf! Kein Driver, wenn man einen braucht!

Erstellung eines Packages „driver“ unter Zuhilfenahme der Packages github.com/jackc/pgx zur Erstellung/Zugriff von/auf PostgreSQL Datenbanken.

github.com/jackc/pgx

Referenz auf pkg.go.dev (Dokumentation)

v1.0.104

Lektion 105 – Integrationsarbeit: Einfügen der Datenbank/Treiber (Repository Pattern)

Verbindung des Package `drivers` mit unserer Webanwendung mittels des sogenannten Repository Pattern.

v1.0.105

Lektion 106 – Etwas Einfaches: Erstellung der benötigten Models

Hinzufügen von mehr „Modellen“ in Form von Datentypen, um Werte aus der Datenbank lesen und speichern zu können.

v1.0.106

Lektion 107 – Putz- und Flickstunde: Regelmäßige Wartung und Säuberung für Ihr Projekt

Insbesondere, wenn Sie auf sich selbst gestellt arbeiten, stellen Sie Notation und/oder Vorgehensweise immer wieder unter verschiedenen Gesichtspunkten infrage und ändern Sie den Quellcode dort, wo es Ihnen sinnvoll erscheint. Versetzen Sie sich immer in die Position eines fachlich versierten Kollegen. Könnte der Ihren Code ohne erheblichen Aufwand nachvollziehen?

v1.0.107

Lektion 108 – Zu viel für diesen Kurs - geht aber, wenn Sie wollen: Object–relational Mapping

Kurze Übersicht über Object-Relational Mapping (ORM), Definition, Nutzen, Vorteile/Nachteile und Verzichtserklärung.

ORM – Wikipedia

GORM

upper/db

Lektion 109 – Double Trouble: Reservierungserstellung und Speicherung in der Datenbank

Implementieren Sie die Datenbankfunktion, um die generierte Reservierung in der Datenbank zu speichern.

v1.0.109

Lektion 110 – Mal mit einem Stock anstupsen: Kurzer Funktionstest der Reservierungsfunktion

Probieren Sie Ihre neu implementierten Funktionen regelmäßig aus, um unliebsame Überraschungen zu vermeiden.

v1.0.110

Lektion 111 – Kleiner Schritt für ein Mensch... Datenbank-Eintrag in den BungalowRestrictions

Für das, was LastInsertID in MariaDB oder MySQL ist, können Sie „returning id“ nutzen.

v1.0.111

Lektion 112 – Verfügbarkeitsprüfung: Prüfen der Verfügbarkeit für einen Zeitraum pro Bungalow

Erstellen Sie eine Datenbankfunktion, die die Verfügbarkeit eines Bungalows für einen bestimmten Zeitraum prüft.

Die im Video verwendete Datenbankabfragen in SQL lauten:

```
-- check exact start and end date, expected outcome: 1
select
      count(id)
from
      bungalow_restrictions
where
      '2024-02-01' < end_date and '2024-02-04' > start_date;

-- check start date before restricted date and same end date, expected
outcome: 1
select
      count(id)
from
      bungalow_restrictions
where
      '2024-01-31' < end_date and '2024-02-04' > start_date;

-- check same start date and end date after restricted date, expected
outcome: 1select
      count(id)
from
      bungalow_restrictions
where
      '2024-02-01' < end_date and '2024-02-10' > start_date;

-- search dates outside of restricted dates but the period contains the
restricted period, expected outcome: 1
select
      count(id)
from
      bungalow_restrictions
where
      '2024-01-10' < end_date and '2024-02-15' > start_date;

-- search dates are completely inside and contained in the restricted
period, expected outcome: 1
select
      count(id)
from
      bungalow_restrictions
where
      '2024-02-02' < end_date and '2024-02-03' > start_date;

-- search dates are completely outside and not overlapping with the
restricted period, expected outcome: 0
select
      count(id)
from
      bungalow_restrictions
```

```
where
      '2024-02-07' < end_date and '2024-02-15' > start_date;
```

v1.0.112

Lektion 113 – Verfügbarkeitsprüfung: Verfügbarkeit für einen Zeitraum für alle Bungalows

Erstellen Sie eine Datenbankfunktion, die die Verfügbarkeit aller Bungalows für einen bestimmten Zeitraum prüft.

Die verwendete Datenbankabfrage in SQL lautet:

```
select
      b.id, b.bungalow_name
from
      bungalows b
      where b.id not in
      (select
            bungalow_id
      from
            bungalow_restrictions br
      where '2024-02-01' < br.end_date and '2024-02-04' > br.start_date);
```

v1.0.113

Lektion 114 – Zarte Bande: Erstellung einer Verbindung Datenbankfunktionen und Handlers

Rufen Sie Ihre Datenbankfunktionen in den Handlern auf und werten Sie die Information aus.

Lektion 115 – Was darf’s sein? Verbindung der Verfügbarkeitsprüfung zur Reservierungsseite

Stellen Sie eine Verbindung von der Verfügbarkeitsprüfung zur Reservierung her.

Lektion 116 – Mission erfüllt: Wir machen erfolgreich eine Reservierung!

Erfahren Sie, wie sie Handler anpassen müssen, um einer Reservierung zu erstellen.

v1.0.116

Lektion 117 – Nachlese: Übersichtsseite finalisieren, Datumsauswahl einschränken, Debugging

Überarbeiten Sie die Anzeige einer Übersichtsseite.

v1.0.117

Lektion 118 – Migrations VII - "Horsing Around" mit Datenbankeinträgen vorbeugen

Erstellen Sie mit `soda generate` Migrations für verschiedene Tabellen. Nutzen Sie den Export als SQL-Statements bereits bestehender Daten aus DBeaver heraus.

`soda generate sql SeedBungalowsTable`

DBeaver, Doppelklick auf die vorgesehene Tabelle, Auswahl der Registerkarte „Data“, Auswahl aller Zeilen, Öffnen des Kontextmenüs mit der rechten Maustaste, „advanced copy“ → „copy as SQL“

```
UP:
[Paste from Clipboard]

DOWN:
delete from bungalows;
```

`soda generate sql SeedRestrictionsTable`

DBeaver, Doppelklick auf die vorgesehene Tabelle, Auswahl der Registerkarte „Data“, Auswahl aller Zeilen, Öffnen des Kontextmenüs mit der rechten Maustaste, „advanced copy“ → „copy as SQL“

```
UP:
[Paste from Clipboard]

DOWN:
delete from restrictions;
```

soda generate sql SeedDevReservationsTable

DBeaver, Doppelklick auf die vorgesehene Tabelle, Auswahl der Registerkarte „Data“, Auswahl aller Zeilen, Öffnen des Kontextmenüs mit der rechten Maustaste, „advanced copy“ → „copy as SQL“

```
UP:
[Paste from Clipboard]

DOWN:
delete from reservations;
```

soda generate sql SeedDevBungalowRestrictionsTable

DBeaver, Doppelklick auf die vorgesehene Tabelle, Auswahl der Registerkarte „Data“, Auswahl aller Zeilen, Öffnen des Kontextmenüs mit der rechten Maustaste, „advanced copy“ → „copy as SQL“

```
UP:
[Paste from Clipboard]

DOWN:
delete from bungalow_restrictions;
```

v1.0.118

Lektion 119 – JavaScript auf'm Date mit JSON: Verfügbarkeitsprüfung mit JSON-Generierung

Werten Sie die Daten in JSON in Ihrem Handler aus, die von Ihrem JavaScript generiert werden.

v1.0.119

Lektion 120 – Anzeige des Ergebnis der Bungalow Verfügbarkeitsabfrage für den Benutzer

Fügen Sie einen Funktionsaufruf für ein Popup-Fenster hinzu und erstellen Sie den korrekten Link für einen GET-Request mit allen notwendigen Informationen zur Erstellung einer Reservierung.

v1.0.120

Lektion 121 – Sessionerstellung: Eine Verbinder zwischen Verfügbarkeit und Reservierung

Benutzen Sie Sessions, indem Sie Reservierungsdaten von der Verfügbarkeitsprüfung an das Reservierungsformular übergeben.

v1.0.121

Lektion 122 – Datenübertragung: JavaScript kopieren in Templates, Vorschlag Code Abstraktion

Die anderen Vorlagen für die Ferienhausseiten mit JavaScript befüllt und JavaScript aus dem Base Layout in einen lokalen Ordner umorganisiert und von dort importiert.

v1.0.122

Abschnitt 13 – Checkup: Tests aktualisieren, um Ihren Code gesund und munter zu halten

> *When debugging, novices insert corrective code; experts remove defective code.*
>
> *Richard E. Pattis*

Lektion 123 – Keine Datenbank für Ihr Tests-Setup? Faken Sie doch eine!

Um das verschachtelte Repository Pattern für das Datenbank Repository in `test_setup.go` für Package `handlers` nutzen zu können, erstellen Sie einen Fake-Datenbanktyp, dem Sie zur Zeit noch leere Datenbank-Methoden unterschieben.

v1.0.123

Lektion 124 – Reparieren der Tests für die Handlers - Reservation in Sessions als Kontext

Lernen Sie, `test_setup.go` und `handlers_test.go` in package `handlers` umzuschreiben. Einführung eines Request-Recorders, um Session-Daten in Tests zu nutzen und eine Instanz zu haben, die einen Client "vortäuscht", der Antworten empfängt. Grundsätzlich wird ein kompletter Anfrage/Antwort-Zyklus in Tests simuliert, der für jeden Handler individuell angepasst werden kann.

v1.0.124

Lektion 125 – Verbesserte Testabdeckung und verschiedene Testfälle für GET-Request Handler

Lernen Sie, wie Sie die Abdeckung des zu testenden Codes erhöhen und weitere Testfälle implementieren.

Lektion 126 – Ein Beispiel für das Schreiben von Tests für POST-Request Handler

Lernen Sie anhand eines Beispieles, wie man einen POST-Request-Handler testet.

v1.0.126

Lektion 127 – Besonderer Fall: Test von POST-Request Handler ReservationJSON

Erfahren Sie, warum der POST-Request-Handler `ReservationJSON` besonders getestet werden muss.

v1.0.127

Lektion 128 – Kurzer Blick auf den Rest der POST-Request Handler Tests, wenn's beliebt.

Überblick über die Anpassungen der Tests der POST-Request-Handler.

v1.0.128

Lektion 129 – Auswechselung und Typveränderung: Aus reqBody wird postedData vom Typ url.Values

Erfahren Sie, wie Sie `reqBody` (vom Typ `string`) gegen `postedData` vom Typ `url.Values{}` austauschen und wie Sie Ihren Request Body mit Hilfe der vorgefertigten Methoden schneller erstellen können.

v1.0.129

Lektion 130 – Houston, wir haben ein Problem! Notfall Debugging im Schnellverfahren!

Lernen Sie, wie Sie einen Fehler ausmerzen, sobald Sie ihn entdecken. Frühzeitige Beschäftigung mit einem Problem hilft langwierige und schwierige Fehlersuche und Beseitigung in der Zukunft zu vermeiden.

v1.0.130

Abschnitt 14 – Wenn der Postmann zweimal klingelt: E-Mail Integration in eine Webanwendung

> *I do love email. Wherever possible I try to communicate asynchronously. I'm really good at email.*
>
> *Elon Musk*

Lektion 131 – Wie war das nochmal? Wie E-Mail und das SMTP-Protokoll funktionieren …

Rufen Sie sich in Erinnerung, wie SMTP dafür sorgt, dass E-Mail an den Empfänger zugestellt wird.

Wie E-Mail funktioniert

Wie SMTP funktioniert

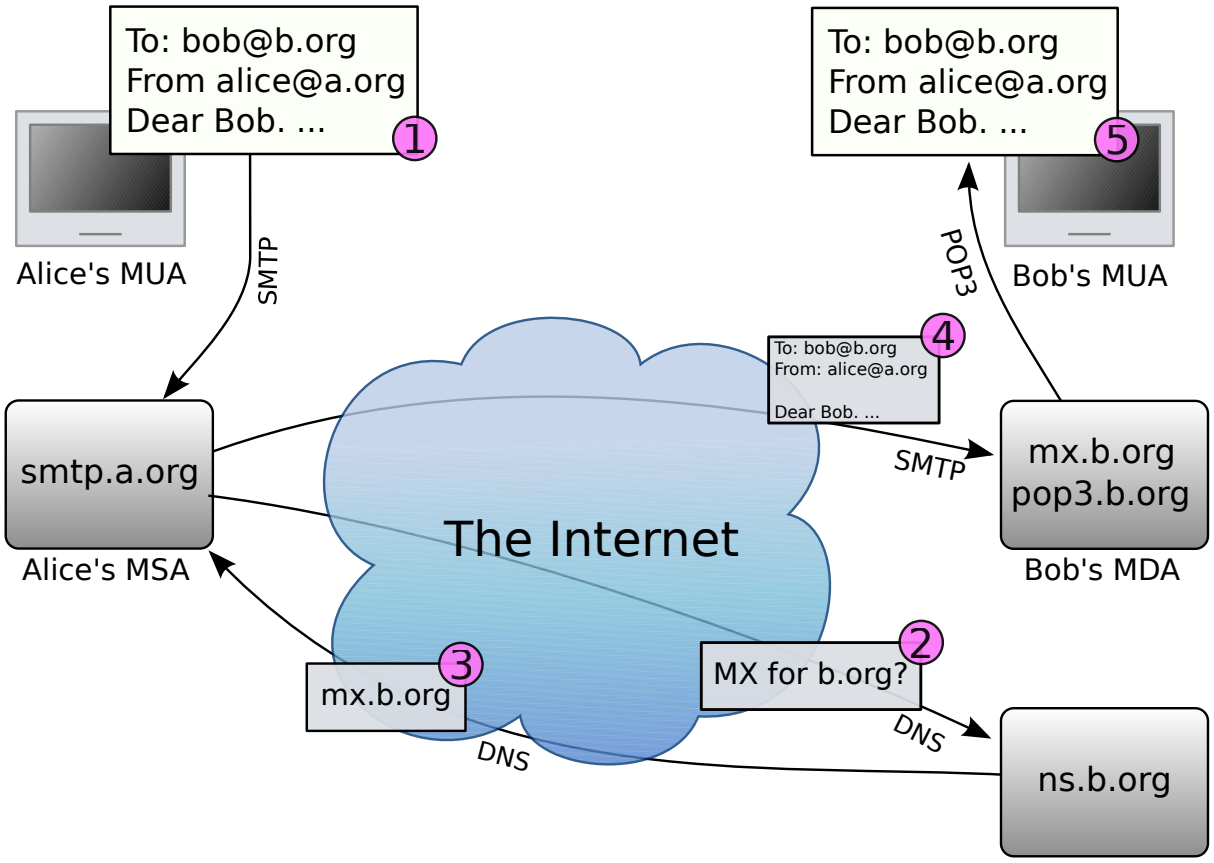


Figure 2: File:Email.svg. (2021, January 6). Wikimedia Commons. Retrieved 03:18, August 26, 2023 from https://commons.wikimedia.org/w/index.PHP?title=File:Email.svg&oldid=524450166.

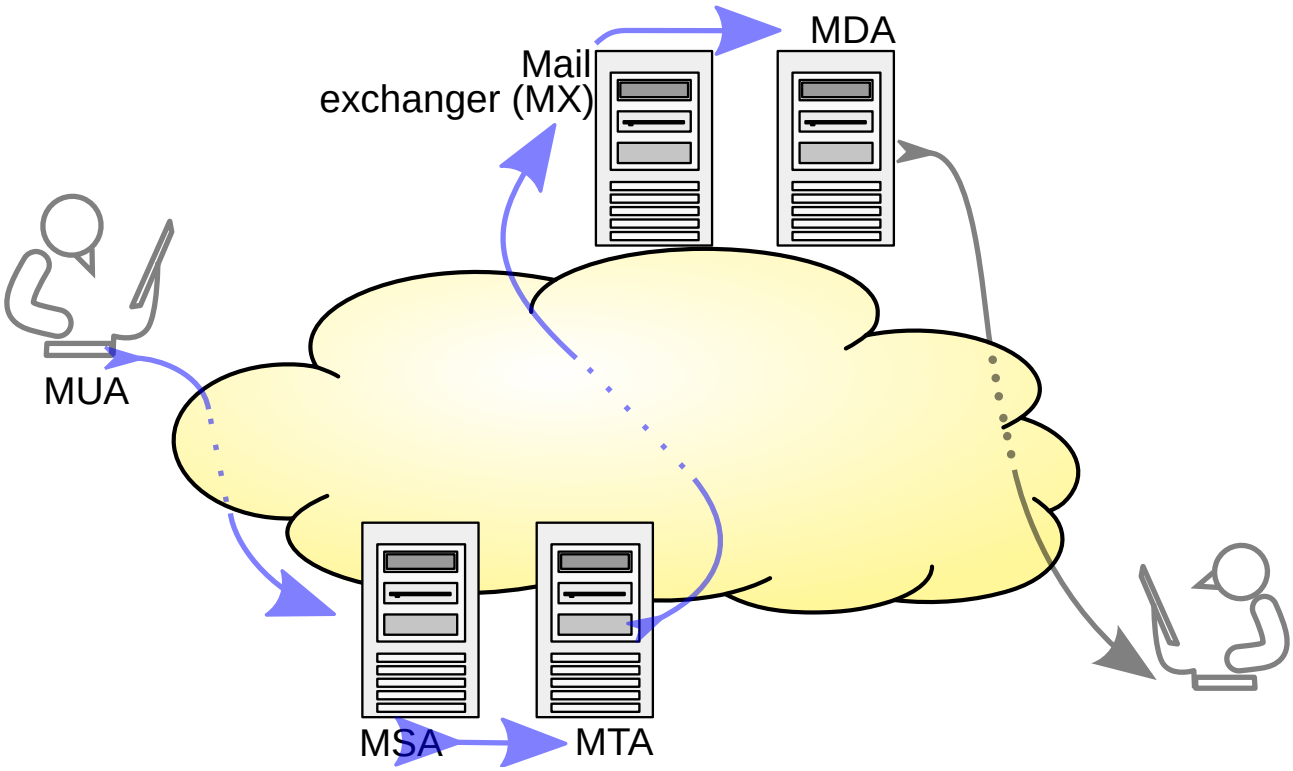


Figure 3: File:SMTP-transfer-model.svg. (2022, May 1). Wikimedia Commons. Retrieved 06:10, August 25, 2023 from https://commons.wikimedia.org/w/index.PHP?title=File:SMTP-transfer-model.svg&oldid=653069184.

Lektion 132 – Mailhog Installation: Mal so richtig die Sau rauslassen!

Die Software MailHog lässt sich auf verschiedenen Betriebssystemen mit wenigen einfachen Schritten installieren und nutzen.

Windows

Binary herunterladen (32 oder 64 Bit) und mit Doppelklick starten. Zum Beenden Fenster der Kommandozeile oder PowerShell schließen.

macOS

Binary herunterladen (Darwin/64 Bit) und ausführen.

Alternativ:

Homebrew installieren mit

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
)"
```

und im Terminal

```
brew update && brew install mailhog
```

ausführen. Anschließend kann man mit

```
brew service start mailhog
```

und

```
brew service stop mailhog
```

Mailhog als Service im Hintergrund starten, bzw, beenden.

Linux

Binary herunterladen (386/32 Bit, amd/64 Bit oder ARM), ausführbar machen und im Terminal ausführen. Mit STRG+C stoppen.

Alternativ:

```
sudo apt-get -y install golang-go
go install github.com/mailhog/MailHog@latest
```

Lektion 133 – E-Mail Versand mit der Standard Library - nur mal der Vollständigkeit halber!

Sie können in GO bereits mit der Standard-Library einfach E-Mails versenden:

```
from := "rick@c-137.universe"
auth := smtp.PlainAuth("", from, "", "localhost")
err = smtp.SendMail("localhost:1025", auth, from,
[]string{"morty@lost-in-roy.game"}, []byte("Come here, or  *burp*
I buy a new Morty!"))
if err != nil {
     log.Fatal(err)
}
```

v1.0.133

Lektion 134 – GO Simple Mail: Eröffnen Sie einen anwendungsweiten Kanal zum E-Mail-Versand

Lernen Sie, wie Sie eine GO-Routine als Listener auf einem Kanal lauschen lassen können, in den Sie von fast überall in Ihrer Webanwendung eine Mail absetzen können.

Sehen Sie,

- wie Sie das Package GO Simple Mail einbinden
- einen Channel einrichten, der E-Mails entgegen nimmt
- einen Listener erstellt, der parallel zur Anwendung auf Mails wartet und
- E-Mails unabhängig von der Webanwendung versendet

v1.0.134

Lektion 135 – #MEGA - Make E-Mail Great Again! - Erstellen/Versenden von E-Mail-Notifications

Lernen Sie, wie Sie eine E-Mail aus Ihrer Webanwendung heraus an den Benutzer versenden.

v1.0.135

Lektion 136 – Informiert bleiben: eine Lösung, um E-Mails an den Betreiber zu senden

Implementieren Sie den Versand von E-Mail an den Betreiber/Admin Ihrer Webanwendung.

v1.0.136

Lektion 137 – ... und jetzt in Schönschrift: Hübsch formatierte E-Mails mit Foundations

Nutzen Sie Foundations for Emails 2, um Ihre E-Mails auf jedem Gerät optimal anzuzeigen.

Foundation for Emails 2

v1.0.137

Lektion 138 – Update der Tests - Hilft ja nichts, es muss ja gemacht werden!

Eine Übersicht über die Tests, damit Ihr Code nachvollziehbar und überprüfbar bleibt.

v1.0.138

Abschnitt 15 – Authentifizierung: Beweisen Sie, dass Sie Sie sind und erhalten Sie Zugang

> *The most common form of despair is not being who you are.*
> *Søren Kierkegaard*

Lektion 139 – Verbessern Sie Ihre App: Gestalten Sie einen einfachen Anmeldebildschirm!

Erstellen Sie einen einfachen Login.

v1.0.139

Lektion 140 – Erfolgreich navigieren: Erstellen einer Route zum Login und einen Handler

Erstellen Sie Route und den GET-Request-Handler für den Login.

v1.0.140

Lektion 141 – Sicherheit freischalten: Authentifizierungs- und DB-Funktionen erstellen

Erstellen Sie alle notwendigen Datenbankfunktionen, um einen Benutzer zu authentifizieren.

v1.0.141

Lektion 142 – Nach dem Formular: Ein Login-Handler, der abliefert

Schreiben Sie den POST-Request-Handler für die Daten, die über das Login-Formular eingeliefert werden.

v1.0.142

Lektion 143 – Etwas Middleware: Zaubern mit Middleware-Magie!

Implementieren Sie Ihr eigenes kleines Stück Middleware, um Benutzer zu authentifizieren.

v1.0.143

Lektion 144 – Genial einfache: Einen Benutzer mit Migrationen anlegen!

Erstellen Sie einen Datenbankeintrag für einen Administrator mittels `migration`.

`soda generate sql SeedUsersTable`

`UP:`

```
INSERT INTO public.users
(full_name,email,password,role,created_at,updated_at) VALUES
('Patrick Star','patrick@bikini-
bottom.ocean','$2a$12$EfnbNqKehrvw7uUgswFR2OlHcE9XjmYaYH8x02JPXapK
2YFx5IX4i','1','2020-01-01 00:00:00.000','2020-01-01
00:00:00.000');
```

`DOWN:`

```
delete from users;
```

Einen Hashwert für ein Passwort erstellen

v1.0.144

Lektion 145 – Die Login-Seite auf dem Prüfstand: Erfolg wartet auf Sie!

Testen Sie, ob die Login-Seite so funktioniert, wie Sie es erwarten.

v1.0.145

Lektion 146 – Authentifizierte Benutzer finden und stilvoll abmelden!

Erstellen Sie eine Unterscheidungsmöglichkeit von authentifizierten und nicht authentifizierten Benutzern, die Sie in den Templates zur Verfügung steht, um ausgewertet zu werden.

v1.0.146

Lektion 147 – Sichern Sie Ihre App: Aufbau eines durch Middleware gesicherten Adminbereichs!

Erstellen Sie einen geschützten Bereich, indem Sie eine Route mit Hilfe von Middleware gegen nicht authentifizierten Zugriff schützen.

v1.0.147

Lektion 148 – Putz- und Flickstunde: Kleinere Säuberungsaktionen - nochmal schnell durchfegen!

Räumen Sie Ihren Code regelmäßig auf, um mögliche Probleme in der Zukunft zu vermeiden.

v1.0.148

Abschnitt 16 – Home, Sweet Home: Ein individualisiertes Backend für sichere Wartungsarbeiten

> *Home is a place you grow up wanting to leave, and grow old wanting to get back to.*
>
> *John Ed Pearce*

Lektion 149 – Ein Admin Dashboard in Fertigbauweise erstellen - Auswahl eines Templates

Erfahren Sie, wie Sie schnell und effizient einen Administrationsbereich erstellen, indem Sie sich ein Packages eines Drittanbieters zunutze machen.

Sie können überarbeitete `admin-layout.tpml` und `admin-dashboard-page.tpml` Dateien in den Ressourcen dieser Lektion auf Udemy finden.

https://github.com/BootstrapDash/RoyalUI-Free-Bootstrap-Admin-Template

v1.0.149

Lektion 150 – Wie am Fließband: Bulk Erstellung von Routen, Handlern und Templates

Erstellen Sie sofort in einem Rutsch alle Routen, Handler und Templates für bereits geplante Funktionalitäten.

v1.0.150

Lektion 151 – Alle Reservierungen anzeigen: Von der Datenbank in eine stilvollen Tabelle

Sie erfahren, wie Sie eine Datenbankfunktion implementieren, um alle vorhandenen Reservierungen auszulesen.

Simple DataTables

v1.0.151

Lektion 152 – Eine Copy-Paste-Orgie: Erstellung einer Auflistung aller neuen Reservierungen

Lernen Sie, wie Sie bereits erstellten Quellcode nutzen, um schnell und effizient eine weitere Funktionalität zu implementieren.

v1.0.152

Lektion 153 – Zwischenspiel: Makeover und kleine Fehler ausbügeln

Stellen Sie sich kleinen Fehlern und Nachlässigkeiten frühzeitig und sorgen Sie für ausreichend Einträge in der Datenbank während der Entwicklungsphase.

v1.0.153

Lektion 154 – Eine einzelne Reservierung anzeigen: Vorbereitung auf Weiteres

Lernen Sie, wie Sie die Datenbankfunktionen implementieren, um Reservierungsdaten zu speichern oder zu löschen.

v1.0.154

Lektion 155 – Neue Möglichkeiten: Erstellen der Funktionen für den Datenbankzugriff

Erstellen Sie weitere Datenbankfunktionen.

v1.0.155

Lektion 156 – Ganz konkret: Implementierung der Bearbeitungsfunktion

Erstellen Sie den POST-Request-Handler, um das Formular, das die zu bearbeitende Reservierung enthält, zu verarbeiten.

v1.0.156

Lektion 157 – Einen Gang hoch schalten: Status einer Reservierung ändern

Erstellen Sie einen Set-To-Processed-Button mit Sicherheitsabfrage, um den Status einer Reservierung zu ändern.

v1.0.157

Lektion 158 – Eine Reservierung löschen: Ist das Kunst oder kann das weg?

Erstellen Sie einen Delete-Button mit Sicherheitsabfrage, um eine Reservierung endgültig aus der Datenbank zu entfernen.

v1.0.158

Lektion 159 – Reservierungskalender I: Überschrift und Navigation

Erstellen Sie eine Überschrift mit Jahr und Monat und bauen Sie eine kleine Navigation, um monatsweise vor und zurück zu blättern.

v1.0.159

Lektion 160 – Reservierungskalender II: Bungalows, Tage und Checkboxen

Bauen Sie eine einfach Tabelle auf, um die Tage eines Monats für jeden Bungalow anzuzeigen und nutzen Sie Checkboxen als Platzhalter.

v1.0.160

Lektion 161 – Reservierungskalender III: Reservierungen und geblockte Tage

Fügen Sie sowohl bereits bestehende Reservierungen als auch Checkboxen für freie und geblockte Tage in Ihren Kalender ein.

v1.0.161

Lektion 162 – Reservierungskalender IV: Render das! Den Kalender anzeigen!

Bringen Sie dem Package `render` bei, den Kalender anzuzeigen.

v1.0.162

Lektion 163 – Reservierungsbearbeitung I: POST-Request, Route und Handler

Erstellen Sie den POST-Request-Handler und die zugehörigen Route und den Handler.

v1.0.163

Lektion 164 – Reservierungsbearbeitung II: Korrekte Rückkehr nach Bearbeitung

Sie müssen dafür sorgen, dass der POST-Request-Handler nach der Bearbeitung einer Reservierung zur richtigen Seite zurückkehrt.

v1.0.164

Lektion 165 – Reservierungsbearbeitung III: Handler, um Aktionen auszuführen

Erfahren Sie, wie die beiden Handler funktionieren, um geblockte Tage im Reservierungskalender zu verwalteten.

v1.0.165

Lektion 166 – Reservierungsbearbeitung IV: Datenbankfunktionen für Aktionen

Sie schreiben die beiden Datenbankfunktionen, um einen Tag für einen Bungalow in der Datenbank zu blocken und um einen geblockten Tag wieder freizugeben.

v1.0.166

Lektion 167 – Quo vadis? Korrektur der Redirects nach Bearbeitung

Sie erfahren, wie Sie die die Redirects, das Routing und die URL-Parameter anpassen müssen, damit der Anwender aus der Bearbeitung einer Reservierung korrekt zurück geleitet wird.

v1.0.167

Lektion 168 – Die Handler-Tests wieder lauffähig machen und ein paar Tests

Erhalten Sie einen Überblick, welche Maßnahmen notwendig sind, damit Ihre Handler-Tests wieder funktionieren.

v1.0.168

Abschnitt 17 – In Betrieb gehen: Bereitstellen Ihrer Webanwendung auf dem Server!

> *Tame birds sing of freedom. Wild birds fly.*
> *attributed to John Lennon*

Lektion 169 – Webanwendung flexibel starten: Nutzen Sie Command Line Flags

Sie lernen, wie Sie Ihre Parameter als Argumente zu Command Line Flags übergeben und so Ihre Webanwendung flexibel starten können. Sie nutzen das Package der Standard Library `flag`.

Cobra - A Framework for Modern CLI Apps in GO

Understanding golang Command Line Arguments

v1.0.169

Lektion 170 – Hinweis zur Nutzung von .env Dateien für Ihre Webanwendung

Erfahren Sie, wie Sie mit dem Package eines Drittanbieters auch .env-Dateien zur Definition Ihrer Umgebung nutzen können.

.env Datei mit GO nutzen

Lektion 171 – Texteditoren Nano and Vi/Vim: Kurze Bedienhinweise

Auf einem Linux-Server stehen Ihnen wahrscheinlich nur textbasierte Editoren wie `nano` und `vi/vim` über ein Terminal/Shell zur Verfügung. Sie können vielleicht ein paar Bedienhinweise gebrauchen.

Nano Editor Cheat Sheet

Vi Editor Cheat Sheet

Lektion 172 – Server besorgen und die notwendige serverseitige Software einrichten

In dieser Lektion werden Ihnen Möglichkeiten aufgezeigt, einen Server zu mieten und Linux als Betriebssystem mit ein paar Tools einzurichten. Auch die Serversoftware `Caddy2` gehört dazu.

Service Provider

- Digital Ocean
- Vultr
- Linode
- Azure
- AlibabaCloud (intl) / Aliyun (China)
- Amazon AWS

Server Software

- Nginx
- Apache
- Lighttpd
- Caddy
- Caddy Install (Binaries/Packages/Docker Image)

Lektion 173 – GO installieren und die Webanwendung auf den Server bekommen

Sie installieren GO auf einem Linux Betriebssystem und nutzen git/github.com oder einen einfachen Download, um Ihren Sourcecode auf Ihren Server zu bekommen.

Lektion 174 – Kein Mail Transfer Agent (MTA) verfügbar? Fake it till you make it!

Ohne einen Mail Transfer Agent (MTA) auf dem Zielsystem stürzt Ihre Anwendung Immer wieder ab. Damit die Dinge überhaupt funktionieren, installieren Sie übergangsweise `postfix`.

Lektion 175 – Supervisor – irgendjemand muss ja aufpassen, während Sie nicht da sind

Nutzen Sie die Anwendung `supervisor` (unter Linux), um Prozesse zu starten, zu überwachen und wenn notwendig neu zu starten.

Lektion 176 – Logo, Footer Content: Letzte Handgriffe bevor der Vorhand sich hebt!

Sie erfahren, wie Sie einfach ein Logo einbinden, den Footer und einige Seiten mit Inhalten füllen.

Abschnitt 18 – Abschiedsgruß & wie es von hier weitergehen könnte (und Platz für Bugfixes)

> *Education is the most powerful weapon which you can use to change the world.*
>
> *Nelson Mandela*

Lektion 177 – Auf Wiedersehen und wie es mit Ihrer Webanwendung weitergehen könnte

Zeit Abschied zu nehmen – Sie bekommen eine kleine Inspiration, wie Sie Ihre Anwendung verbessern können.

Latest Release