BFO Webprogrammierung

2022/2023

Yannick Rast, m29264

Vorgehen und Entwurfsentscheidungen:

Zu Beginn habe ich mich mit der Aufgabenstellung befasst. Ich las sie mir aufmerksam durch, markierte die zu erfüllenden Kriterien und fasste diese noch einmal in eigenen Worten zusammen. Daraufhin begann ich damit, ein neues Projektverzeichnis in meinem Git-Verzeichnis zu erstellen. Mit dem Befehl „go mod init Webprogrammierung“ erzeugte ich dann eine go.mod, damit das Verzeichnis als ein go-Modul anerkannt wird. Als Programmbasis kopierte ich mein „mongodb“-Projekt, welches ich innerhalb der Veranstaltung geschrieben hatte, in das neue Verzeichnis, machte ein paar Anpassungen und führte den Befehl „go mod tidy“ aus, um alle benötigten Abhängigkeiten zu aktualisieren.

Als Nächstes erstellte ich ein Git-Repository auf https://github.com/, welches ich anschließend innerhalb der go.mod mit dem Projektverzeichnis verknüpfte. Daraufhin wollte ich mit Hilfe des Vorlesungsbeispiels „ginstatic“ die Einrichtung des Dockers und der Ausführung des go-Programms als Container innerhalb dessen umsetzen. Leider konnte ich diesen erst nicht starten, obwohl Docker Desktop auf die neueste Version aktualisiert worden war und dieser zuvor während der Vorlesungszeit funktionierte. Also deinstallierte ich das Programm, installierte es erneut und startete meinen Rechner neu. Nun funktionierte Docker Desktop. Im nächsten Schritt erstellte ich eine Dockerfile sowie eine docker-compose.yaml und schrieb in diese die benötigten Angaben zur Ausführung meines Webserver-Programms mit mongodb als zwei separate Container im Docker.

Als das Erzeugen der Images beider Programmteile über den Befehl „docker compose build“ funktionierte und auch die Ausführung dieser als Container über den Befehl „docker compose up“ geklappt hat, konnte ich mich endlich mit dem Programmentwurf beschäftigen.

Zuallererst recherchierte ich im Internet, welche die beste Vorgehensweise bei der Speicherung von HTML-Inhalten in mongodb ist. Ich stieß darauf, dass es keine offizielle Vorgehensweise gibt und man mehrere Möglichkeiten hat, unter anderem die Speicherung als JSON-Dokumente, welche einer der beliebteren Methoden zu sein scheint. Dies kam mir sehr gelegen, da ich im vergangenen Jahr in der Veranstaltung „Programmierung 3“ schon ein wenig Erfahrung mit JSON machen durfte. Also entschied ich mich dazu, die Speicherung der HTML-Daten in dieser Form umzusetzen und recherchierte, ob HTML-Strings in JSON ohne weitere Vorkehrungen eingebunden werden können, oder ob ich auf eine alternative Form wie Markdown oder die Verschlüsselung von Sonderzeichen ausweichen muss. Erfreulicherweise konnte ich feststellen, das simples Abspeichern von HTML-Strings in JSON möglich ist, nur dass lediglich zweifache Anführungszeichen durch Hochkommas ersetzt werden müssen. Da es sich dafür meiner Meinung nach nicht lohnt, extra eine Methode zu schreiben, die HTML-Strings in der genannten Hinsicht anpasst, entschied ich mich dazu, diese Anpassungen für jede Webpage manuell vorzunehmen. Nun begann ich damit, mir Gedanken darüber zu machen, welche Attribute zu jeder Webpage gespeichert werden müssen.

Ich legte die Attribute Titel, Beschreibung, HTML-Inhalt, und Bilder für die Inhalte und Metadaten einer Webpage fest. Im Laufe der Entwicklung kamen dann noch fünf weitere Attribute zum Speichern von Referenzen auf spezifische CSS- und JS-Dateien, des Typen und dem URL-Tag einer Webpage, sowie der objektspezifischen ObjectID für das Speichern in der Datenbank hinzu. Also setzte ich dieses Konzept in Form einer Struktur im Code um und schrieb ein JSON-Dokument mit ersten Testdaten zum Erzeugen einer Webpage. Ich recherchierte das Einlesen und anschließende Einfügen von JSON-Dateien in go und mongodb, und implementierte dafür eine Funktion, welche die JSON-Dokumente entschlüsselt und deren Daten in die Datenbank einfügt.

Zudem importierte ich die Funktionen und Dateien für die Verwendung von Templates aus dem „ginstatic“-Vorlesungsbeispiel und machte ein paar Anpassungen an diesen, sodass die Templates meine Webpage-Daten darstellen konnten. Nachdem die Ausführung funktionierte, begann ich damit, ein Design-Konzept und die Struktur meiner Website zu entwickeln. Ich orientierte mich dabei an meiner Abgabe vom Kurs „Design interaktiver Oberflächen“ und nahm kleinere und größere Anpassungen an dieser vor. Ich entschied mich aus Zeitgründen dazu, auf die Skalierung der Webinhalte für Mobilgeräte im Rahmen der Aufgabe zu verzichten.

Da ein weiteres Kriterium der Aufgabenstellung lautet, dass das Programm neben Templates auch statische Webpages ausliefern können soll, kopierte ich die index.html und dazugehörige CSS in ein neues Verzeichnis für statische Dateien im go-Projekt. Mit Hilfe eines Fileservers konnte ich problemlos über den Pfad „/static/pages/“ die statische index.html im Browser öffnen. Also fügte ich dem Verzeichnis eine simple HTML-Datei mit dem Text „Static Page“ zum Testen der Ausführung von statischen Webpages hinzu und entfernte die index.html und die dazugehörige CSS wieder.

Nun erzeugte ich ein neues Verzeichnis für Dateien, in der später die Zip mit den Rohdaten abgelegt werden soll. Dort legte ich vorerst jedoch das JSON-Dokument sowie die CSS- und JS-Dateien aus meiner „DIO“-Abgabe, ohne diese in einer Zip-Datei zu komprimieren, hinein. Nun überlegte ich mir, welche HTML-Inhalte von allen Webpages geteilt werden, um diese in die base.template.html zu schreiben, um „Code-Redundazen“ zu vermeiden. Dazu gehören der Head, mitsamt CSS-Imports, der Body mit Wrapper, Header und Footer, wobei die beiden Letzteren die Haupt- und Footer-Navigation beinhalten, sowie die Script-Imports.

Da ich anhand des „ginstatic“-Beispiels einsehen konnte, wie die Einbindung von Arrays im Sinne der Website-Navigation innerhalb eines Templates umgesetzt wird, fiel mir auf, dass ich sowohl für die Haupt- und Footer-Navigation, die jeweiligen Webpages nach ihren Typen filtern und referenzieren können muss. Deshalb fügte ich an dieser Stelle den Webpages ein weiteres Attribut zum Speichern des Webpage-Typs hinzu und erstellte eine weitere Struktur sowie eine Datenbankkollektion, in der jeweilige Referenzen für jede Webpage gespeichert werden können. Ich entschied mich dazu, die Werte der Referenzen innerhalb der Initialisierungsfunktion der Datenbank mit den eingelesenen Daten der Webpages zu definieren. Dabei wird nach dem Einlesevorgang zu jeder Webpage aus dem Titel ein Tag generiert, welcher keinerlei Sonderzeichen enthält und als URL-Bezeichnung dienen soll. Dieser wird anschließend in der Webpage gespeichert. Des Weiteren wird der Tag neben dem Titel, dem Typen und dem daraus gebildeten URL-Pfad in der Referenz gespeichert. Beide Objekte werden daraufhin in die jeweiligen Datenbankkollektionen eingelesen.

Zum Übertragen der Webpage- und Navigationsdaten an die Templates erzeugte ich eine weitere Struktur, in der die Daten der aufgerufenen Webpage und die benötigten Referenzen in Form von Link-Arrays gespeichert werden können.

Als das Abrufen dieser Daten innerhalb der Templates funktionierte, machte ich mich daran, die Footer-Pages, die „Über mich“-Page sowie die Kontakt-Page mit dem Inhalt aus meiner „DIO“-Abgabe zu füllen. Ich fügte dazu die spezifischen CSS- und JS-Dateien dem Dateiverzeichnis hinzu und passte die Texte noch etwas an.

Im nächsten Schritt widmete ich mich dem zu erfüllenden Kriterium des Einlesens der Rohdaten aus einer Zip. Hierfür recherchierte ich wiedermals, welche Methode sich in Bezug auf Webserver dafür unter anderem am besten eignet. In diesem Fall erschien es mir am sinnvollsten, die Rohdaten, welche in der Datenbank gespeichert werden sollen, direkt aus der Zip zu lesen und an die mongodb-Datenbank zu übergeben. Für die weiteren Dateien entschied ich mich dafür, diese bei der Ausführung des Programms in ein temporäres Verzeichnis zu speichern, welches bei Beendigung des Containers wieder gelöscht wird. Also implementierte ich mehrere Methoden zum Lesen aus einer Zip anhand eines Beispiels, auf welches ich während meiner Recherche gestoßen bin (https://golang.cafe/blog/golang-unzip-file-example.html). Ich verknüpfte die bisherige JSON-Einlese-Funktion mit der Zip-Einlesefunktion und implementierte eine weitere Funktion zum Erzeugen und Schreiben der Temporären Verzeichnisse und Daten.

Zu guter Letzt fing ich damit an, die Startseite der Website, also die Portfolio-Page und das dazugehörige Template zu entwerfen. Damit auf der Portfolio-Webpage meine Projekte referenziert werden können, fügte ich der Übergabestruktur eine weiteres Array zum Speichern von Referenzen auf die Artikel-Webpages (Projekte) hinzu. Ich entschied mich für vier Projekte, an denen ich in den letzten Monaten gearbeitet habe und trug sie in das Webpage-JSON-Dokument mit dem Typen „article“ ein. Da es sich bei drei der Projekte um ein Video-Projekt handelt, habe ich den YouTube-Player mit den jeweiligen Videos innerhalb eines neuen Templates eingebettet und die Webpages um ein Attribut zum Speichern der Video-URL erweitert. Für das vierte Projekt erstellte ich ein weiteres Template, welches zur Abbildung von mehreren Bildern in einer Slideshow dienen soll. Da ich nur bei Projekten mit mehreren Bildern eine Slideshow dargestellt haben wollte, fügte ich dem Pagehandler eine Bedingung hinzu, die die Page darauf überprüft, ob es sich bei dem Artikel um eine Slideshow- oder Video-Page handelt.

Abschließend formulierte ich alle meine bisherigen Kommentare aus, fügte weitere zu, entfernte Redundanzen im Code und fügte den CSS-Dateien Anweisungen für die Darstellung für verschiedene Monitorauflösungen hinzu.

Ausführung:

1. **https://github.com/yannickrast/Webprogrammierung besuchen und über den Button „Code“ das Projekt herunterladen oder ggf. über GitHub Desktop öffnen**
2. **die heruntergeladenen Daten in ein neues Verzeichnis namens „Webprogrammierung“ einfügen**
3. **Docker Desktop ausführen**
4. **Terminal im Projektverzeichnis (mit Adminrechten) öffnen**
5. **Ausführen der Terminal-Eingabe: docker compose build**
6. **Ausführen der Terminal-Eingabe: docker compose up**
7. **Website über die Adresse http://localhost:9090/ aufrufen**
8. **statische Beispielpage lässt**

Quellen:

Golang:	**https://pkg.go.dev/archive**
go und JSON:	**https://astaxie.gitbooks.io/build-web-application-with-golang/content/en/07.2.html**
go und Zip:	**https://golang.cafe/blog/golang-unzip-file-example.html**
Map:	**https://openlayers.org/, https://www.openstreetmap.org/**
Kontakt:	**https://form.taxi/de/**
Rechtstexte:	**https://datenschutz-generator.de/impressum/**
YouTube-Einbettung:	**https://blog.hubspot.de/marketing/youtube-video-einbetten**
Slideshow:	**https://www.3schools.com.translate.goog/howto/howto.js.slideshow**

Eigenständigkeitserklärung:

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie die aus fremden Quellen direkt oder indirekt übernommenen Stellen/Gedanken als solche kenntlich gemacht habe.

Diese Arbeit wurde noch keiner anderen Prüfungskommission in dieser oder einer ähnlichen Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Hiermit stimme ich zu, dass die vorliegende Arbeit von der Prüferin/dem Prüfer in elektronischer Form mit entsprechender Software auf Plagiate überprüft wird.

Schöningen, 31.01.2023