

Photo-Server

Von 9752762, 8093702 und 6539456

BusinessThemenProjekteDiversesAktuellPrint FriendlyGoogle DriveAdvent of Code


Home Photo-Server

Photo-Server

[Bilder hochladen](#)
[Zur Bestellliste](#)

Deine Bilder:

Auswahl zur Bestellliste hinzufügen



IMG_20181031_145654.jpg
09 Jan 21 21:55 CET
☐ Zur Bestellliste hinzufügen







Bild2.jpg
02 Nov 20 12:32 UTC
☐ Zur Bestellliste hinzufügen




20201101_202534.jpg
01 Nov 20 20:25 UTC
☐ Zur Bestellliste hinzufügen




IMG_20201031_202255.jpg
31 Oct 20 20:22 UTC
☐ Zur Bestellliste hinzufügen




20201028_110527.jpg
28 Oct 20 11:05 UTC
☐ Zur Bestellliste hinzufügen



IMG_20201027_131455.jpg
27 Oct 20 13:14 UTC
☐ Zur Bestellliste hinzufügen



IMG_20201027_123135.jpg
27 Oct 20 12:31 UTC
☐ Zur Bestellliste hinzufügen



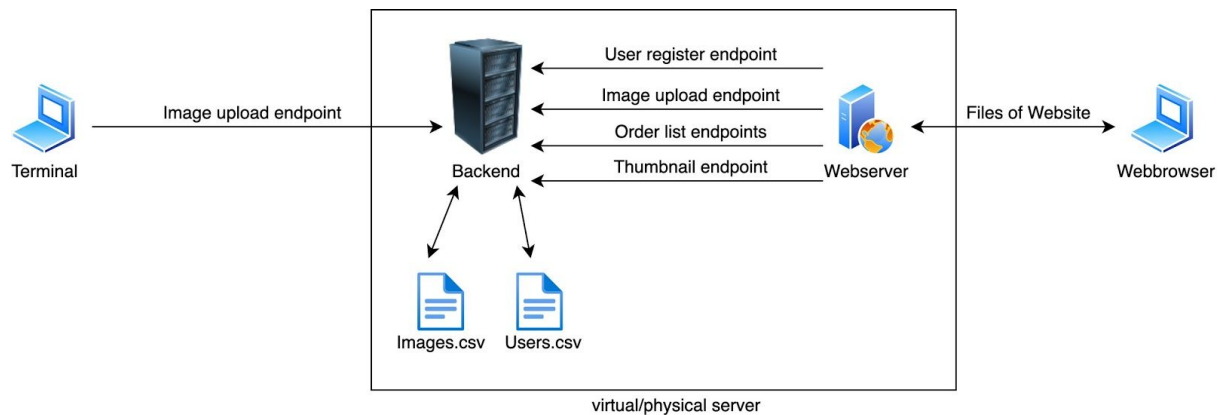
20201027_115022.jpg
27 Oct 20 11:50 UTC
☐ Zur Bestellliste hinzufügen

Architekturdokumentation	3
Kommandozeilen-Tool	4
Backend	4
Webanwendung	4
Anwenderdokumentation	5
Weboberfläche	5
Terminalanwendung - Batch Upload	8
Dokumentation des Betriebs	10
Erzeugen der Kompilate	10
Hinweise zur Verzeichnisstruktur	10
Webanwendung	10
Backend	11
Benutzer registrieren	11
Bild hochladen	12
Bilddaten abfragen	12
Vorschaubilder abfragen	12
Bestellliste abfragen	13
Bild zur Bestellliste hinzufügen	13
Bestellliste aktualisieren	13
Eintrag auf Bestellliste löschen	13
Gesamte Bestellliste löschen	13
Bestellliste herunterladen	14
Beiträge der Gruppenmitglieder	15
Beitrag von 9752762	15
Beitrag von 8093702	16
Beitrag von 6439456	17

Architekturdokumentation

Die Photo-Server-Anwendung ist in drei Abschnitte aufgeteilt. Das Backend bildet das Zentrum des Systemes. Mit dem Backend kommuniziert das Kommandozeilentool, welches den Upload eines ganzen Verzeichnisses ermöglicht, und die Webanwendung, welche das Registrieren, Hochladen und Anschauen ermöglichen.

In der folgenden Grafik sieht man die Komponenten und die Zusammenhänge.



Der Photo-Server nutzt die internen Packages `api`, `auth`, `cryptography`, `image`, `user` und `util`. Das Zusammenspiel der Packages ist teilweise in der Abbildung unten dargestellt.

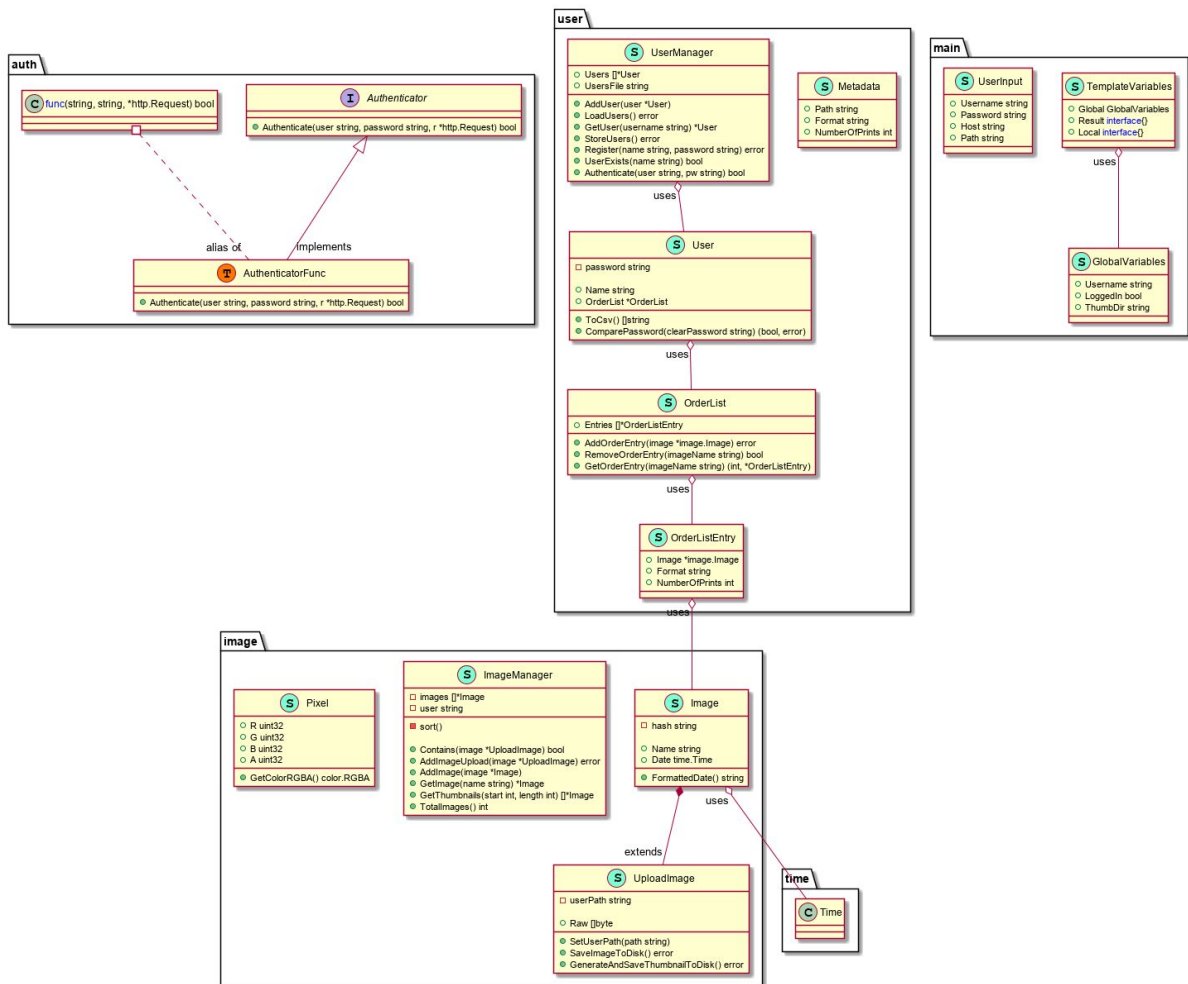
Das `api`-Package definiert die Input- und Output-Parameter für POST-Requests, sowie die Output-Parameter für GET-Requests. Außerdem ermöglicht es ein einfaches Dekodieren von JSON-Strings in Go-Objekte und umgekehrt.

Das `auth`-Package stellt dem Backend die Authentifizierung zur Verfügung. Hierfür stellt das Paket neben der `Authenticate`-Funktion unter anderem auch einen `AuthHandlerWrapper`, welcher eine Funktion vom Typ `http.HandlerFunc` zurückgibt. Dieser ermöglicht es, die `Authenticate`-Funktion als `http`-Handler zu verwenden.

Das `user`-Package ist verantwortlich für die Benutzerverwaltung. Die Benutzerdaten werden in einer `csv`-Datei verwaltet. Das `user`-Package verwaltet außerdem die Bestelllisten und stellt die Funktionen zum zippen einer Bestellliste zur Verfügung.

Das `image`-Package enthält die Funktionalitäten rund um Bilder im Photo-Server. Dazu zählt die Bildverwaltung. Für jeden Benutzer wird ein Ordner für die Bilder angelegt. Neben den Bildern werden in diesem Ordner auch die Vorschaubilder sowie eine `csv`-Datei gespeichert. Die `csv`-Datei enthält die Titel sowie Aufnahmedaten aller Bilder des Benutzers, absteigend nach dem Datum sortiert. Außerdem wird für jedes Bild ein Hashwert gespeichert. In dem `image`-Package sind außerdem die Funktionalitäten zum Erstellen der Vorschaubilder (Thumbnails) sowie der Exif-Parser enthalten.

Das `cryptography`-Package stellt Funktionen zum MD5-Hashing und Passwort-Salting zur Verfügung.



Kommandozeilen-Tool

Das Kommandozeilen-Tool ist eine Terminalanwendung, welche das Hochladen aller Bilder aus einem Ordner ermöglicht. Der Benutzer gibt hierfür seinen Benutzernamen, sein Passwort, den Host des Backends sowie den Pfad zu dem Ordner, in dem die Bilder liegen, an.

Backend

Das Backend verwaltet die Anwendungsdaten und stellt sie dem Kommandozeilen-Tool sowie dem Frontend zur Verfügung.

Webanwendung

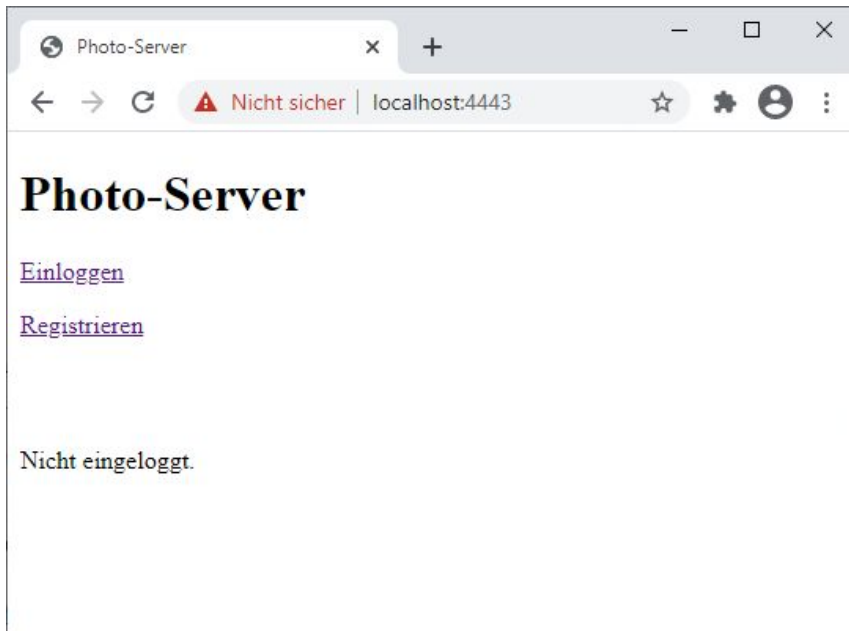
Die Webanwendung ermöglicht dem Benutzer die Interaktion mit der Anwendung über eine graphische Oberfläche im Browser. Hier kann sich ein neuer Benutzer registrieren, anmelden, Bilder hochladen, anschauen und Bilder zu einer Bestellliste hinzufügen. Der Webserver stellt die Bilder des aktuell angemeldeten Benutzers zur Verfügung.

Anwenderdokumentation

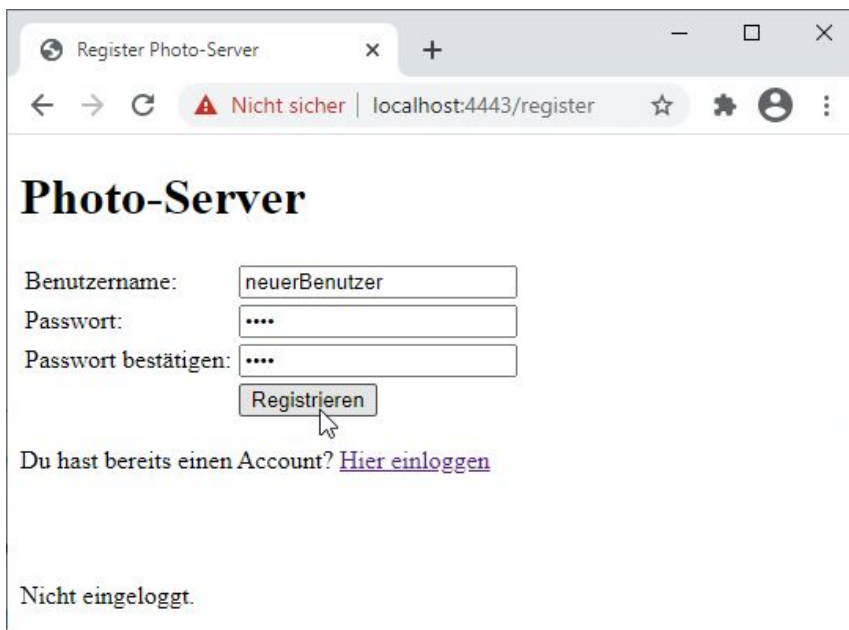
Weboberfläche

Das Frontend besteht aus fünf Ansichten:

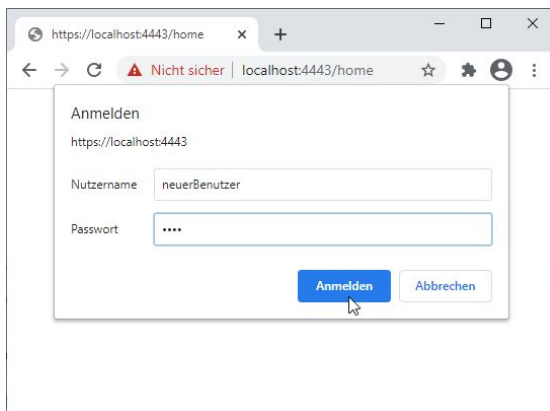
1. `/` - In der Root-Ansicht sind nur einfache Links zu `/register` und zu `/home` vorhanden.



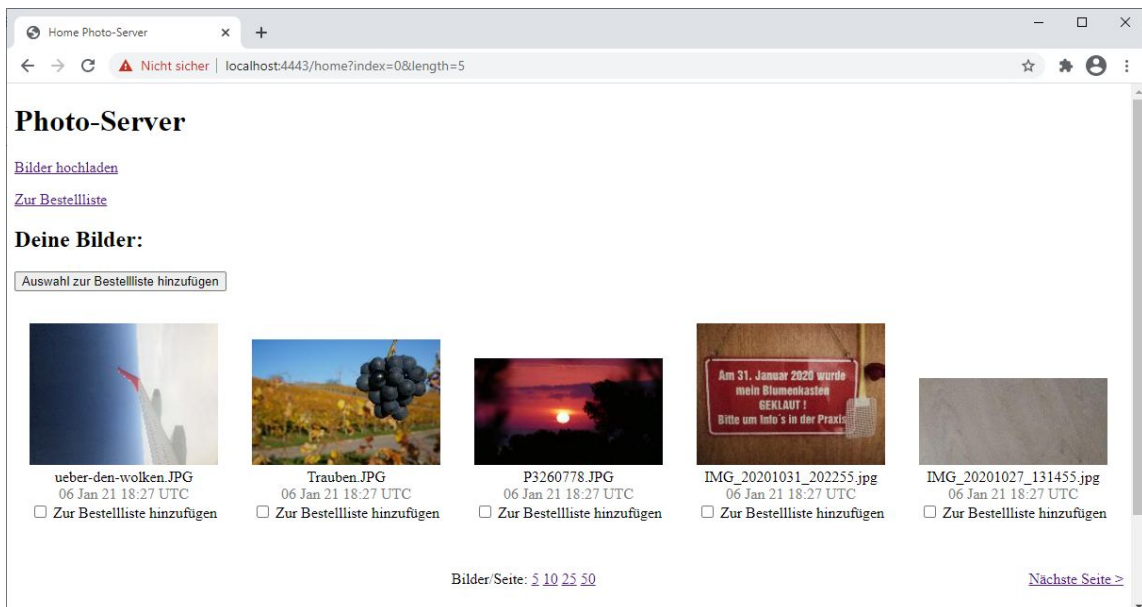
2. `/register` - Hier kann über ein Formular ein neuer Benutzer angelegt werden.



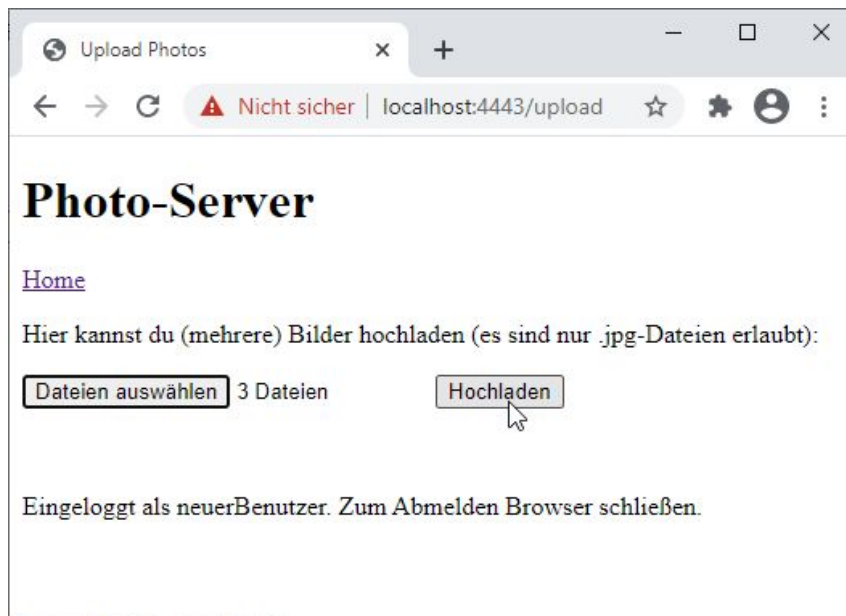
3. `/home` - Ist man nicht authentifiziert fragt der Browser nach Anmeldedaten.



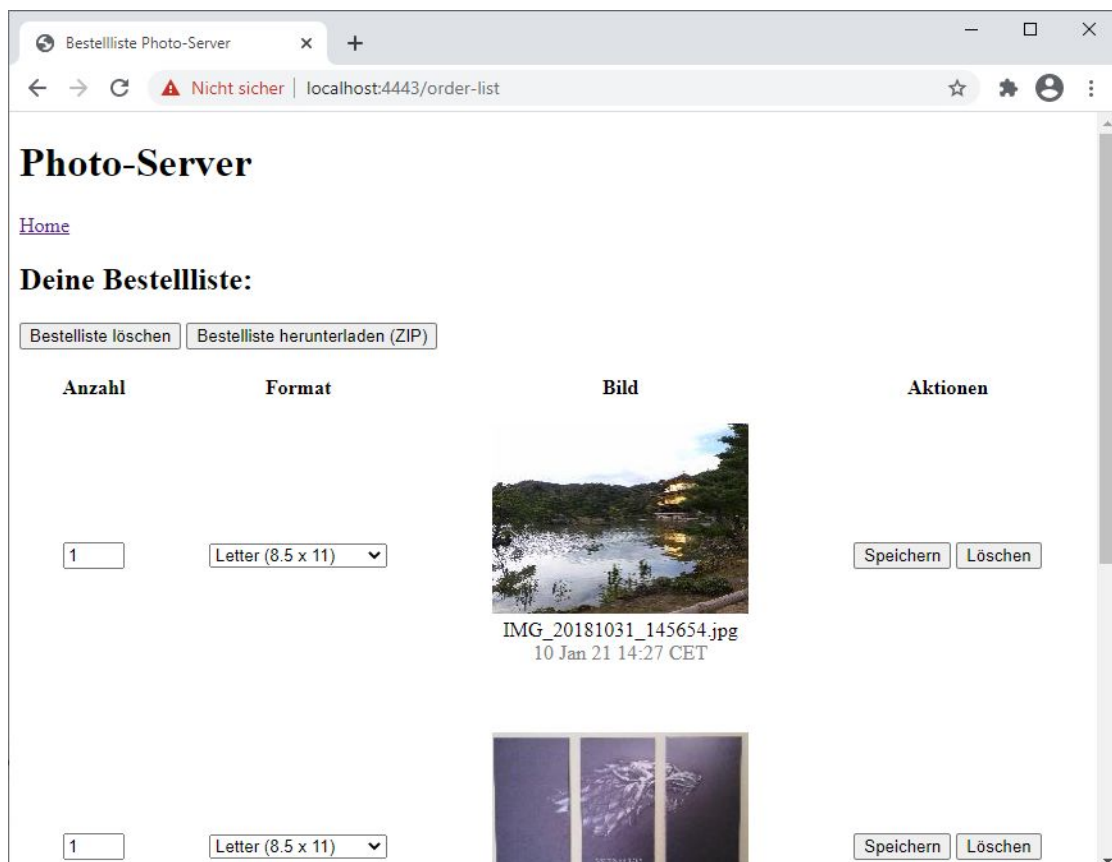
Nach der erfolgreichen Authentifizierung kann der Benutzer eine chronologisch sortierte Liste seiner bereits hochgeladenen Bilder mit Namen und Aufnahmedatum sehen. Diese Liste kann in mehrere Seiten aufgeteilt sein. Oben sind Links zu `/upload` und `/order-list` vorhanden. Unter jedem Bild befindet sich eine Checkbox mit der man die Bilder auswählen kann, die man mit dem entsprechenden Button zur Bestellliste des aktuell angemeldeten Benutzers hinzufügen möchte.



4. `/upload` - Hier können mehrere Bilder auf einmal hochgeladen werden. War jeder Upload der Bilder erfolgreich, landet der Benutzer wieder auf dieser Seite.



5. `/order-list` - Hier werden die Bilder angezeigt, die in `/home` zu der Bestellliste hinzugefügt wurden. Hier besteht die Möglichkeit die Anzahl und das Format der Abzüge eines Eintrags in der Bestellliste zu ändern. Es können einzelne Einträge oder sogar die komplette Bestellliste gelöscht werden. Zudem kann der Benutzer die Bestellliste als .zip-Datei herunterladen. In dieser Datei befinden sich dann die Bilder in Original-Auflösung zusammen mit einer .json-Datei, die die Anzahl und das Format der Bilder enthält.



Terminalanwendung - Batch Upload

In der Terminalanwendung werden alle Bilder aus einem angegebenen Ordner hochgeladen.

Wie in dem Bild zu sehen werden die hierfür benötigten vier Parameter (Benutzername, Passwort, Host des Backends sowie der Ordner) nacheinander vom Benutzer abgefragt.

Der Benutzer bekommt Rückmeldungen über den Programmablauf, zum Beispiel, wie viele Bilder aus dem Ordner eingelesen wurden.

[illegible]

Die Terminalanwendung ermöglicht es dem Benutzer mehrere Ordner hochzuladen, wobei er nur einmal Benutzernamen, Passwort sowie Host des Backends eingeben muss. Möchte ein Benutzer keinen weiteren Ordner hochladen, wird die Anwendung beendet.

[illegible]

Treten beim Hochladen Fehler auf, zum Beispiel wenn ein Bild versucht wird mehrfach hochzuladen, wird dies dem Benutzer zurückgemeldet.

Dokumentation des Betriebs

Erzeugen der Kompilate

Das Erzeugen der Kompilate kann über mehrere Wege erfolgen. Die hier beschriebene Variante ist über das Terminal. Wenn man sich im Terminal im Root-Verzeichnis des Projektes befindet, kann mit den folgenden Befehlen je ein Kompilat für das Backend, die Webanwendung und die Terminalanwendung erzeugt werden.

```
► go build -o backend ./cmd/backend
► go build -o web ./cmd/web
► go build -o terminal ./cmd/terminal
```

Die Namen für die Kompilate, sowie deren Dateierweiterung, kann an die eigenen Bedürfnisse und an das entsprechende verwendete Betriebssystem angepasst werden. Die oben genannten Kompilate (`backend`, `web`, `terminal`) sind für ein Linux-/Unix-basiertes Betriebssystem.

Für Windows könnte es wie folgt aussehen:

```
► go build -o backend.exe ./cmd/backend
► go build -o web.exe ./cmd/web
► go build -o terminal.exe ./cmd/terminal
```

Hinweise zur Verzeichnisstruktur

Die beiden Kompilate des Backends und der Webanwendung müssen im selben Verzeichnis platziert werden, da beide vom gleichen Ablageort systemrelevanter Dateien ausgehen.

Die **Zertifikate** (`key.pem` und `cert.pem`) für eine verschlüsselte Verbindung über HTTPS werden im gleichen Verzeichnis wie die Kompilate selbst erwartet. Die im Projekt abgelegten Zertifikate sind für "localhost" ausgestellt und für ein Jahr gültig. Für den Testbetrieb können diese genutzt werden, jedoch sollten für den Livebetrieb ein eigenes Zertifikat erstellt werden.

Für den produktiven Einsatz empfiehlt sich ein eigenes Verzeichnis außerhalb des Projektverzeichnisses zu erstellen.

Webanwendung

Die Webanwendung kann mit den folgenden Parametern konfiguriert werden. Eine Auflistung aller Parameter mit den zugehörigen Standardwerten kann ebenfalls über den Parameter `-h` (► `./web -h`) angezeigt werden.

- Port der Webanwendung (`port`), der Standardwert ist 4443
Beispiel: `./web -port=4443`
- Pfad und Name des Bildverzeichnisses (`imagepath`), der Standardwert ist "images".
Webanwendung und Backend müssen den gleichen Wert für diesen Parameter

erhalten!

Beispiel: `./web -imagepath="imagefolder"`

- URL zum Backend (`backendurl`), der Standardwert ist "<https://localhost>"

Beispiel: `./web -backendurl="https://127.0.0.1"`

- Port des Backends (`backendport`), der Standardwert ist 3000.

Beispiel: `./web -backendport=3001`

Beim Starten des Webservers können die oben genannten Parameter konfiguriert werden. Wenn beim Aufruf keine Angabe erfolgt werden die Standardwerte übernommen. Der Server gibt kurz nach dem Starten die URL und Port aus. Zum Beispiel:

2021/01/09 17:04:43 web listening on <https://localhost:4444>

Das Beenden des Servers erfolgt über die Standardtastenkombination zum Abbrechen eines Terminalbefehles **Strg + C**.

Backend

Das Backend kann mit den folgenden Parametern konfiguriert werden. Eine Auflistung aller Parameter mit den zugehörigen Standardwerten kann ebenfalls über den Parameter `-h` (► `./backend -h`) angezeigt werden.

- Port des Backends (`port`), der Standardwert ist 3000

Beispiel: `./backend -port=3001`

- Pfad und Name des Bildverzeichnisses (`imagepath`), der Standardwert ist "images". Webanwendung und Backend müssen den gleichen Wert für diesen Parameter erhalten!

Beispiel: `./backend -imagepath="imagefolder"`

Beim Starten des Backends können die oben genannten Parameter konfiguriert werden. Wenn beim Aufruf keine Angabe erfolgt werden die Standardwerte übernommen. Der Server gibt kurz nach dem Starten die URL und Port aus. Zum Beispiel:

2021/01/09 17:03:23 backend listening on <https://localhost:3000>

Das Beenden des Servers erfolgt über die Standardtastenkombination zum Abbrechen eines Terminalbefehles **Strg + C**.

Das Backend verfügt über mehrere Endpunkte, welche beispielsweise von der Terminalanwendung und der Webanwendung genutzt werden. Bei allen Aufrufen der Endpunkte muss über BasicAuth die Authentifizierung erfolgen.

Benutzer registrieren

Durch diese Anfrage wird ein neuer Benutzer zum Backend hinzugefügt. Hier ist keine BasicAuth notwendig, da der Benutzer noch nicht existiert.

POST-Request an `https://[url/ip:port]/register`

Inhalt der Anfrage als JSON:

- Gewünschter Benutzername (Username) (Darf nur a-z,A-Z,0-9,-,_ und . enthalten)
- Gewünschtes Passwort (Password)
- Gewünschtes Passwort wiederholen (PasswordConfirmation)

Bild hochladen

Durch diese Anfrage wird ein neues Bild für den Benutzer hinzugefügt, wenn dieses zuvor noch nicht hochgeladen wurde.

POST-Request an `https://[url/ip:port]/upload`

Inhalt der Anfrage als JSON:

- Name des Bilder (Filename)
- Erstellungsdatum des Bildes (CreationDate)
- Rohdaten des Bildes als Base64 kodiert (Base64Image)

Bilddaten abfragen

Hinweis: Dieser Endpunkt ist aktuell nicht freigeschaltet, da die Kommentar-Funktion noch nicht implementiert ist und somit dieser Endpunkt keine weiteren Funktionen anbietet. Durch eine Anfrage an diesen Endpunkt werden Details zu einem spezifischen Bild des Benutzers zurückgegeben.

GET-Request an `https://[url/ip:port]/image`

Benötigte Parameter:

- Name des Bildes (name)

Beispiel: `https://localhost:8080/image?name=myImage.jpg`

Als Ergebnis auf die Anfrage gibt die Schnittstelle ein JSON-Objekt mit dem Namen des Bildes (Name) und das Erstelldatum (Date) zurück.

Vorschaubilder abfragen

Durch eine Anfrage an diesen Endpunkt werden Details zu mehreren Thumbnails des Benutzers zurückgegeben.

GET-Request an `https://[url/ip:port]/thumbnails`

Benötigte Parameter:

- Startindex des Thumbnails (index)
- Anzahl der Thumbnails (length)

Beispiel: `https://[url/ip:port]/thumbnails?index=4&length=5`

Als Ergebnis auf die Anfrage gibt die Schnittstelle ein JSON-Objekt mit einem Array von Bildern (Name des Bildes Name und Erstelldatum Date) zurück und der Gesamtanzahl der Bilder (TotalImages) zurück.

Bestellliste abfragen

Durch eine Anfrage an diesen Endpunkt wird die Bestellliste des Benutzers zurückgegeben.

GET-Request an `https://[url/ip:port]/orderList`

Es werden keine weiteren Angaben benötigt.

Als Ergebnis auf die Anfrage gibt die Schnittstelle ein JSON-Objekt mit einem Array von Bestelllisteneinträgen zurück. Ein Eintrag besteht aus den Bilddaten (Name des Bildes `Name` und Erstelldatum `Date`), dem Bildformat (`Format`) und der Anzahl der Abzüge (`NumberOfPrints`).

Bild zur Bestellliste hinzufügen

Durch eine Anfrage an diesen Endpunkt wird ein Bild zur Bestellliste des Benutzers hinzugefügt.

POST-Request an `https://[url/ip:port]/addOrderListEntry`

Inhalt der Anfrage als JSON:

- Name des Bildes (`ImageName`)

Bestellliste aktualisieren

Durch eine Anfrage an diesen Endpunkt wird ein Eintrag in der Bestellliste des Benutzers geändert.

POST-Request an `https://[url/ip:port]/changeOrderListEntry`

Inhalt der Anfrage als JSON:

- Name des Bildes (`ImageName`)
- Bildformat (`Format`)
- Anzahl der Abzüge (`NumberOfPrints`)

Eintrag auf Bestellliste löschen

Durch eine Anfrage an diesen Endpunkt wird ein Eintrag aus der Bestellliste des Benutzers gelöscht.

POST-Request an `https://[url/ip:port]/removeOrderListEntry`

Inhalt der Anfrage als JSON:

- Name des Bildes (`ImageName`)

Gesamte Bestellliste löschen

Durch eine Anfrage an diesen Endpunkt wird die Bestellliste des Benutzers gelöscht.

POST-Request an `https://[url/ip:port]/deleteOrderList`

Es werden keine weiteren Angaben benötigt.

Bestellliste herunterladen

Durch eine Anfrage an diesen Endpunkt werden die Bilder in der Bestellliste in eine Zip-Datei gebündelt und zurückgegeben.

GET-Request an `https://[url/ip:port]/downloadOrderList`

Es werden keine weiteren Angaben benötigt.

Als Ergebnis auf die Anfrage wird ein JSON-Objekt mit der Base64-codierten Zip-Datei (`Base64ZipFile`).

Beiträge der Gruppenmitglieder

Beitrag von 9752762

Ich habe das Frontend, den dazugehörigen Webserver, die Kommunikation mit dem Backend, die Verwaltung der Benutzer, die Bestellliste und die Authentifizierung implementiert.

Im Frontend wird eine die HTML-Datei `layout.html` als Template-Wrapper für alle Frontend-Dateien verwendet. Dies hat den Vorteil nur einmal eine Überschrift oder einen Footer definieren zu müssen. Die Funktion `Layout` sorgt hierbei für das Zusammensetzen der Template-Dateien. Währenddessen werden Variablen in die Template-Dateien übergeben, wo diese ausgewertet werden können (Beispiel einer Syntax: `{{.Global.Username}}`). Außerdem werden hier eigens definierte Template-Funktionen übergeben, um z.B. im Frontend einfache Berechnungen vornehmen zu können (vgl. `templatefunctions.go`).

Beim Start des Webserver werden die vorhandenen Routen definiert und mit der zuständigen Handler-Funktion verknüpft. Dabei laufen manche Routen zuerst durch eine Authentifizierungs-Wrapper Funktion, um zu prüfen, ob der mitgeschickte HTTP Basic-Auth Header vorhanden ist und die darin gespeicherten Credentials korrekt sind. In jedem Handler wird die Funktion `CallApi` mit dem entsprechenden Request aufgerufen. Diese Funktion bereitet den Request für den API-Call vor, führt diesen durch und parst das JSON-codierte Ergebnis in ein übergebenes struct.

Neben dem Webserver werden in der `web/main.go` auch zwei `http.FileServer` gestartet. Einer stellt die Bilder aus dem `imagepath` zur Verfügung, der beim Start der Anwendung definiert wird. Dabei ist auch wieder ein Authentifizierungs-Wrapper im Einsatz. Dieser prüft, ob der aktuell angemeldete Benutzer auch wirklich auf die anfragende Datei zugreifen darf (vgl. `AuthFileServer()`). Außerdem wird bei jeder Anfrage an diesen Image-File-Server über den `CacheWrapper` in der Antwort der HTTP-Header `Cache-Control` gesetzt. Damit kann der Browser die Bilder cachen und muss diese nicht bei jedem Request neu vom Server anfordern. Der andere File-Server ist nur für andere statische Dateien zuständig, wie z.B. das Bereitstellen von CSS und JavaScript-Dateien. Dabei wird die `main.css`-Datei dem Frontend zur Verfügung gestellt.

Die Anmeldedaten der Benutzer werden in einer CSV-Datei gespeichert. Das Passwort ist dabei salted und hashed gespeichert. Der `UserManager` ist für die Verwaltung dieser Liste verantwortlich. Er umfasst Funktionen zum Laden und Speichern der Datei, zum Hinzufügen, Registrieren und Authentifizieren eines Benutzers.

Da bei fast jedem Request an den Webserver und an das Backend über HTTP Basic Authentication überprüft wird, ob der Benutzer in dieser CSV-Datei steht und das korrekte Passwort mitgeschickt ist, würde es zu sehr vielen Zugriffen auf das Dateisystem kommen. Deshalb wird die Datei zu Beginn einmal eingelesen und im RAM über eine Variable in `usermanagercache.go` gespeichert.

Die Bestellliste hängt über `OrderList` direkt am Benutzer und wird der Einfachheit halber nicht auf dem Dateisystem gespeichert, sondern befindet sich nur während der Laufzeit des Backends im RAM am Benutzer-Objekt. Wird der Backend-Server also neu gestartet, ist die Bestellliste wieder leer.

Beitrag von 8093702

Meine Beiträge zu dem Projekt sind das Kommandozeilen-Tool, welches das Hochladen eines Ordners mit Bildern ermöglicht, sowie der Exif-Parser.

Das Kommandozeilen-Tool ermöglicht das Hochladen eines Ordners mit Bildern. Die nötigen Parameter hierfür gibt der Benutzer ein (Benutzername, Passwort, Host des Backends sowie Pfad des Ordners mit Bildern). Anschließend werden aus dem angegebenen Ordner alle JPEG-Bilder ausgelesen. Für jedes Bild wird ein POST-Request an den Endpunkt `/upload` des Backends gestellt. Als Body wird unter anderem das Base64-kodierte Bild und der Dateiname des Bildes mitgeschickt.

Um die Requests effizient durchzuführen, habe ich das Hochladen eines Fotos als Go-Routine implementiert. Um die Abarbeitung der Go-Routinen abzuwarten, wurde eine WaitGroup implementiert.

Der Exif-Parser wird benötigt, um das Aufnahmedatum eines Bildes aus den Metadaten auszulesen.

In einer JPEG-Datei stehen die Exif-Daten nach dem APP1-Marker (0xFFE1). Die Daten folgen einer bestimmten Struktur. Die zwei Bytes direkt nach diesem Marker enthalten die Länge dieser APP1-Daten, also des Abschnitts, in dem die Exif-Daten stehen. Danach folgt ein Identifikator, ob die nachfolgenden Daten Exif-Daten sind. Dieser Identifikator sind das Wort „Exif“ in ASCII-Zeichen sowie zwei Bytes, welche den Wert 0x00 haben. Anschließend folgen die Exif-Daten.

Um das Aufnahmedatum eines Bildes auszulesen, muss zunächst der Abschnitt mit den Exif-Daten aus dem Bild ausgelesen werden. Hierfür wird der Marker (0xFFE1) in dem Bild gesucht und die nachfolgenden zwei Bytes ausgelesen. Diese enthalten die Länge des Datenabschnitts. Mit dieser Länge wird der Datenabschnitt ausgelesen.

Die Exif-Daten werden als Tags abgespeichert, wobei ein Tag aus einem Namen und dem dazugehörigen Wert besteht. Das Aufnahmedatum ist im Tag „DateTime“ gespeichert.

Um ein bestimmtes Tag auszulesen, müssen zunächst alle Tags ausgelesen, decodiert und in einer Map gespeichert werden. Hierbei müssen verschiedene Datentypen sowie die unterschiedlichen Längen der Werte beachtet werden. Wurden die Werte in das richtige Format konvertiert, können sie in einer Map gespeichert werden, wobei als Schlüssel der Name des Tags verwendet werden kann. Anschließend kann über diesen Namen der Wert eines bestimmten Tags abgerufen werden.

Diese Vorgehensweise ist mit viel Aufwand verbunden.

Der DateTime-Tag wird in einem bestimmten Format gespeichert (YYYY:MM:DD HH:MM:SS). Betrachtet man die Exif-Tags genauer, fällt auf, dass es lediglich zwei andere

Tags gibt, deren Werte im gleichen Format gespeichert werden. Dies sind *DateTimeOriginal* und *DateTimeDigitized*. In der Regel enthalten diese drei Tags den gleichen Wert, falls *DateTimeOriginal* und *DateTimeDigitized* einen Wert enthalten. Da für den Photo-Server keine anderen Tags aus den Exif-Daten ausgelesen werden, bietet es sich an, das Datum mit einem regulären Ausdruck aus den zuvor ausgelesenen Rohdaten zu ermitteln. Der reguläre Ausdruck, mit dem das Datum ausgelesen wird, ist `\d{4}\:(0[1-9]|1[012])\:(0[1-9]|12)[0-9]|3[01])`
`([01][0-9]|2[0-3]):([0-6][0-9]):([0-6][0-9])`. Für den regulären Ausdruck wird das Paket `regex` genutzt.

Beitrag von 6439456

Mein Beitrag in diesem Projekt liegt bei der Entwicklung der Logik für die Verwaltung und Speicherung der Bilder, sowie das Generieren der Vorschaubilder, im Backend. Die Konfigurierbarkeit der Webanwendung und des Backends über Flags ist ebenfalls von mir implementiert worden.

Die Entwicklung des Paketes `internal/image` ist der Hauptteil meines Beitrags. Eine Herausforderung war die Generierung des Thumbnails. Die Herangehensweise ist die Übernahme jedes x. Pixels in das Thumbnails. Das x wird durch das Verhältnis der Breite des originalen Bildes zur Zielbreite des Thumbnails gebildet.

Für die Implementierung der Hauptfunktionalität wurden sinnvolle Hilfsmethoden erstellt und teilweise in eigene Pakete wie `internal/cryptography` (für das Hashing eines Bildes) und `internal/util` (für das Lesen und Schreiben von binären Daten) ausgelagert. Um eine hohe Testabdeckung zu erreichen, sind Tests für alle Funktionen geschrieben worden. Zur Erkennung, ob ein Bild bereits für den Benutzer gespeichert wurde, wird einerseits der Dateiname genutzt. Andererseits wird aus den Rohdaten jedes Bildes über MD5 ein Hashwert generiert, welcher als eindeutige Identifikation des Bildes dient und nur einmal pro Benutzer vorkommen darf.

Das Hinzufügen der Logik für die Konfigurierbarkeit des Backends und der Webanwendung durch Flags beim Starten, sowie die zugehörigen Änderungen im Quellcode, damit die übergebenen Werte in allen Klassen verfügbar sind, waren meine Aufgabe.

Für die Versionsverwaltung des Projektes wird GitHub verwendet. Die dortigen "Actions" können für CI genutzt werden. Da die Abgabe über PushCI erfolgt, wo alle implementierten Tests erfolgreich durchlaufen müssen, habe ich eine Action für das Kompilieren und Ausführen der Tests mit Ausgabe der Testabdeckung erstellt. Das hat den Vorteil, dass bei jedem Push der Änderungen sichtbar ist, ob alle notwendigen Dateien im Repository enthalten sind, alle Tests erfolgreich durchlaufen und der Compiler keine Fehlermeldung zurückgibt. Somit konnte während der Entwicklungszeit das aktive Hochladen bei PushCI vermieden werden und ist nur zum Zeitpunkt der Abgabe notwendig.