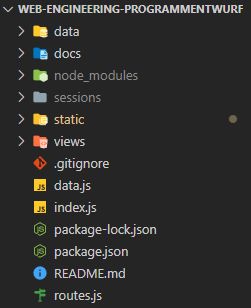
# Verzeichnisstruktur

Auf root Ebene befinden sich folgende Ordner:



* Data: Darin befinden sich alle JSON-Dateien, die für die Speicherung von Daten benötigt werden.
* Sessions: Hier werden die Session Daten gespeichert.
* Static: Dort befinden sich alle statisch eingebundenen Dateien wie Bilder, clientseitige JavaScript- und CSS-Dateien.
* Views: Dort liegen alle ejs-Dateien, die von der View-Engine „EJS“ verwendet werden können

Alle anderen Dateien sind im root-Ordner selbst abgelegt.

# package.json

In dieser Datei werden unterschiedliche Metadaten über eine Node.js App bzw. Projekt gespeichert. Dazu gehören beispielsweise der Name, die Version, eine Beschreibung und der Autor der App.

Außerdem wird hier mit dem keyword „main“ die Einstiegsdatei der App definiert. In unserem Fall ist dies die „index.js“, welche sich im root Verzeichnis befindet.

Im Block „scripts“ können Befehle, verbunden mit einem key, gespeichert werden. Zur leichteren Entwicklung verwenden wird beispielsweise den Befehl „nodemon index.js“, um die App lokal zu hosten. Das dazugehörige keyword ist „start“. Wenn man nun in der Konsole den Befehl „npm run start“ ausführt, wird der Befehl „nodemon index.js“ aufgerufen.

Ein weiterer wichtiger Block sind die „dependencies“. Hier sind alle npm-Pakete aufgelistet, die im Projekt verwendet werden. Mit einem einfach „npm install“ Befehl im Terminal, werden automatisch all diese Pakete installiert.

# Index.js

Die index.js dient als Einstiegsdatei für das Node.js Projekt.

In den ersten acht Zeilen, werden alle benötigten Node-Module (npm-Pakete) eingebunden.

In Zeile 10 werden vier Funktionen eingebunden, die zur besseren Verwendbarkeit in eine externe Datei ausgelagert wurden.

In Zeile 12 wird der Port festgelegt, auf welchen die App lauschen soll.

In der der darauffolgenden Zeile wird eine neue Express-Anwendung erstellt. Diese wird dann in der Konstante „app“ gespeichert.  
Express ist ein Web-Framework für Node.js. Es bietet eine Reihe von Funktionen, die das Erstellen von Webanwendungen und APIs erleichtern. Mit Express können unter anderem Routen definiert werden, um Anfragen an bestimmte URLs zu verarbeiten und Middleware verwendet werden, um Anfragen vor der Verarbeitung zu bearbeiten.

In Zeile 14 wird eine Instanz der „urlencoded“-Middleware erstellt und in der Konstante „encodeUrl“ gespeichert. Diese wird später in verschiedenen Routen verwendetet, um Anfragedaten im URL-codierten Format zu verarbeiten.

Weiterhin werden in Zeile 15 Optionen für den Session-Store gesetzt und in der Konstante „fileStoreOptions“ gespeichert. Dabei wird das Verzeichnis angegeben, in dem die Sessions gespeichert werden und die Anzahl, wie oft ein Speicherversuch durchgeführt werden soll.

In Zeile 21 werden alle vorhandenen Benutzer in die Variable „users“ gespeichert.

In Zeile 24 wird die „session“-Middleware zur Express Anwendung hinzugefügt. Hierbei werden verschiedene Option gesetzt. Wichtig zu erwähnen ist hierbei die „store“ und „genid“ Optionen. Als store wird hier eine neue Instanz vom FileStore zusammen mit den zuvor festgelegten Option (fileStoreOptions) erstellt.  
Die Option genid gibt eine Funktion an, die verwendet wird, um neue Sitzungs-IDs zu generieren. In unserem Fall wird die Funktion uuid() verwendet, um UUIDs als Sitzungs-IDs zu verwenden.

In Zeile 41 wird ebenfalls eine Middleware hinzugefügt. Diese überprüft, ob in der aktuellen Session ein Benutzer gespeichert ist. Wenn ja, wird die res.locals.authenticated Variable auf true gesetzt, andernfalls auf false.

In Zeile 51 wird als Middleware das Favicon festgelegt.

Zeile 54 setzt die View Engine auf „ejs“.

Zeile 56 fügt einige Routen, die in einer externen Datei definiert sind, hinzu.

In Zeile 57 wird das Verzeichnis „/static“ statisch in die Anwendung eingebunden. Alle Dateien, die sich darin befinden, können dann durch Pfadangabe an den Client gesendet werden.

## Routen

Im Folgenden werden die verwendeten Routen beschrieben.

### Post: /auth

Hier werden zuerst der übergebene Benutzername und das Passwort aus dem Request-Body gespeichert. Anschließend wird versucht den Benutzer in der Variable „users“ zu finden.  
Wird er gefunden, wird mit der Hilfe der becrypt.compare Methode getestet, ob das eingegebene Passwort mit dem gehashten Passwort im User-Objekt übereinstimmt. Wenn ja, wird eine Session erstellt und der Benutzer wird darin gespeichert. Danach wird der User zu Startseite geleitet. Wenn nein, bekommt der Benutzer eine Warnmeldung und wird zurück auf die Login-Seite gesendet.  
Wird der Benutzer nicht gefunden, bekommt der Nutzer ebenfalls eine Meldung und wird zurück geleitet.

### Post: /newUser

Zuerst werden hier alle übergebenen Eigenschaften aus dem Request-Body gespeichert. Danach wird geschaut, ob der Benutzername bereits existiert. Wenn ja wird dem Benutzer eine Meldung ausgegeben und er muss sich einen neuen Benutzernamen überlegen.  
Danach wird überprüft, ob das Passwort und die Wiederholte Eingabe übereinstimmen. Wenn nein, wird dem Benutzer ein Warnmeldung ausgegeben.  
Wenn ja, kann der Benutzer nun gespeichert werden. Dazu wird zuerst mit Hilfe von bcrypt das Passwort gehasht und gesalzen. Anschließend werden alle vom Benutzer angegebenen Daten und das gehashte Passwort in der Users-JSON gespeichert.

### Post: /newComment

Zu Beginn werden alle Artikel aus der Artikel-JSON geladen. Danach wird die übergebene URL aus dem Request-Body gespeichert.  
Nun wird überprüft, ob in der aktuellen Session ein Benutzerobjekt gespeichert ist. Wenn ja, wird anhand der übergebenen URL der Artikelname gespeichert. Anhand dieses Namens wird dann der richtige Artikel gesucht. In dieses Artikelobjekt werden dann der Benutzername, aktueller Zeitstempel und der Kommentarinhalt innerhalb des Kommentararrays gespeichert.  
Ist in der aktuellen Session kein Benutzer gespeichert, bekommt der Nutzer den Hinweis, dass er sich anmelden muss.

### Post: /logout

Wenn eine Session existiert, wird zuerst das Cookie des Benutzers gelöscht. Anschließen wird Session gelöscht und bei Erfolg wird der Nutzer zur Startseite geleitet.

## Funktionen

### generateTimestamp

Diese Funktion erstellt zuerst ein Datumsobjekt mit aktuellem Datum und Uhrzeit. In der Variable options werden verschiedene Eigenschaften zu Formatierung dieses Objekts gespeichert.  
Mit der Methode toLocaleTimeString wird das Datumsobjekt zusammen mit den Optionen dann in einem lesbaren deutschen Format ausgegeben.

Zum Schluss wird in der index.js die app.listen()-Methode aufgerufen. Damit wird der Webserver auf dem zuvor festgelegten Port gestartet.

# Kommentarfunktion

## HTML (comments.ejs)

Die Kommentarsektion besteht aus zwei großen Bereichen: Ein Verfassungs- und ein Anzeigebereich.

Der Verfassungsbereich besteht aus einer HTML-Form. Diese Form sendet beim Absenden einen Post-Request an die Route /newComment. Die Form beinhaltet eine Textarea, in der der Benutzer seinen Kommentar verfassen kann. Das Textfeld hat eine maximale Zeichenlänge von 1000 und eine minimale Zeichenlänge von 10.  
Außerdem gibt es ein unsichtbaren Inputfield mit dem namen „url“. Hier wird über clientseitiges JavaScript die aktuelle URL gespeichert.  
In einem Container unterhalb des Textfeldes wird außerdem angezeigt, wie viele Zeichen man bereits eingegeben hat.  
Am Ende befindet sich ein submit-Button, mit dem der Kommentar abgesendet werden kann.

Der Anzeigebereich besteht aus einem Informationsblock, der den Kommentarverfasser, einen Zeitstempel anzeigt und den Kommentar an sich.  
Der Anzeigebereich wird dynamisch mit Hilfe von ejs generiert und erweitert sich je nach Anzahl der Kommentare. Mit ejs wird zuerst geprüft, ob das Kommentarobjekt vorhanden ist. Wenn ja, wird zusätzlich geprüft ob darin Kommentare vorhanden sind. Trifft auch das zu wird der Anzeigebereich so oft ausgegeben, wie es Kommentare gibt. Zusätzlich wird auch der Inhalt dieser Kommentare (Benutzername, Zeitstempel und Kommentarinhalt) dynamisch aus dem Kommentarobjekt ausgelesen und ausgegeben.

## CSS (comments.css)

Folgend werden nur komplexere Style-Eigenschaften beschrieben. Eigenschaften wie padding oder margin sind in mehreren Elementen definiert und bestimmen dabei den Innen- bzw. Außenabstand zwischen den einzelnen Elementen. Auch die Eigenschaften height und width bestimmen lediglich die Höhe und die Breite der Elemente und haben sonst keine tiefere Bedeutung.

Comments-section-wrapper:  
Dieser Style umfasst die ganze Kommentarsektion und setzt deren Breite auf 100%. Display: flex bewirkt, dass sich alle untergeordneten Elemente flexibel anordnen. Mit flex-wrap: wrap wird festgelegt, dass sich die Elemente bei Bedarf umgebrochen werden.  
Bei fast allen untergeordneten Elementen die Breite auf 100% gesetzt, damit eine Vertikale Anordnung entsteht.

Comment-write-wrapper:  
Dieser Style umfasst den Verfassungsbereich. Eine wichtige Eigenschaft ist hier border-bottom. Damit wird unterhalb des Containers eine Linie erzeugt, die eine sichtbare Abgrenzung zwischen Verfassungs- und Anzeigebereich darstellt.

Comment-form:  
Bezieht sich auf die HTML-Form. Hier wird die Eigenschaft text-align: left gesetzt, damit alle sich darin befindlichen Textelemente linksbündig angezeigt werden.

CommentInput:  
Beschreibt das Aussehen der HTML-textarea. Mit resize: none wird festgelegt, dass die Größe des Textfelds nicht durch den Benutzer verändert werden kann. Außerdem wird mit font-family die Schriftart gesetzt. Mit border-radius werden an den Ecken des Textfelds kleine Rundungen hinzugefügt.  
Ist das Textfeld im Fokus (commentInput:focus) wird die border auf 2px vergrößert, um dem Nutzer visuell zu zeigen, dass er gerade das Textfeld benutzt. Die Eigenschaft outline wird auf none gesetzt, da sonst noch eine zusätzliche border angezeigt wird.

Comment-button:  
Beschreibt das Aussehen des Buttons zum Absenden des Kommentars. Mit color: black wird die Schriftfarbe auf Schwarz gesetzt. Mit der Eigenschaft transition: 0.3s wird die Dauer von Übergängen festgelegt. Dieser kommt zum Einsatz, wenn der Benutzer seine Maus über den Button bewegt (comment-button:hover). Dann wird die Hintergrundfarbe des Buttons etwas dunkler gemacht. Durch den transition-Effekt dauert diese Veränderung 0.3 Sekunden.

Comments-wrapper:  
Umfasst die Styles für den Anzeigebereich. Mit display: flex und flex-wrap: wrap wird hier wieder eine flexible Anordnung der Unterelemente bewirkt, die bei bedarf automatisch einen Umbruch auslösen. Bei fast allen untergeordneten Elementen die Breite auf 100% gesetzt, damit eine Vertikale Anordnung entsteht.

Comment-info:  
Beinhaltet den Benutzernamen und den Zeitstempel. Mit display: flex und justify-content: space-between werden diese beiden Elemente jeweils ganz links und ganz rechts platziert.

Comment:  
Beinhaltet den Kommentarinhalt. Hier wird zur besseren Abgrenzung eine border von 1px gesetzt.

## JS (comments.js)

Zu Beginn werden das Kommentarfeld (commentInput) und die Anzeige für die Anzahl der Zeichen (character-count) jeweils in eine Konstante gespeichert.

Dann wird ein Event-Listener zu dem Kommentarfeld hinzufügt. Der Event-Listener wird jedes Mal ausgelöst, wenn der Benutzer etwas in das Kommentarfeld eingibt ("input"-Event).  
Wenn das Event ausgelöst wird, wird eine Funktion ausgeführt, die das Event-Objekt als Argument erhält. Innerhalb dieser Funktion wird das Ziel des Events (e.target) in einer Variablen gespeichert.  
Dann wird das maxlength-Attribut des Ziel-Elements abgerufen und in einer Variablen namens maxLength gespeichert. Anschließend wird die aktuelle Anzahl der Zeichen im Ziel-Element gezählt und in einer Variablen namens currentLength gespeichert.  
Zum Schluss wird der Inhalt des Elements character-count aktualisiert, um die aktuelle Anzahl der Zeichen und die maximale Anzahl der Zeichen anzuzeigen (z.B. “25/1000”).

Die letzte Code-Zeile speichert die aktuelle URL als value im „url“-Input der HTML-Form.

# Datenspeicherung

Alle persistenten Daten werden in JSON-Dateien gespeichert. Diese JSON-Dateien liegen im Verzeichnis „./data“.

## Articles.json

Im Objekt „articles“ werden hier die Namen aller Artikel inklusiver ihrer Kommentare gespeichert. Der Name wird im Wert des Schlüssels „articleName“ gespeichert. Der Wert des Schlüssels "comments" ist ein Array von Objekten, wobei jedes Objekt einen Kommentar darstellt. Jeder Kommentar hat dabei die Schlüssel „username“ für den Benutzernamen, „timestamp“ für den Zeitstempel und „comment“ für den Inhalt des Kommentars.

## Users.json

Im Objekt „users“ werde hier alle Benutzer gespeichert. Als Werte werden hier pro Benutzer der Benutzername, Vorname, Nachname, das Passwort als Hash und das Geschlecht gespeichert.

# Registrierung

## HTML (registration.ejs)

Die HTML besteht hauptsächlich aus einer Form, die von einem Container umgeben wird.

Die Form beginnt mit einer Überschrift (h2). Anschließen folgen mehrere Input-Container mit der Klasse „input-wrapper“, die sich alle ähnlich sind. Sie bestehen aus einem Label und einem nachfolgenden Input.  
Alle input-Felder haben das Attribut „required“, einen Platzhaltertext und einen entsprechenden Namen.  
Die Inputs für den Benutzernamen, Vornamen und Nachnamen haben zusätzlich ein Attribut „maxLenght=20“, um die maximale Zeichenanzahl auf 20 zu beschränken. Außerdem wird mit dem Attribut „pattern“ mit Hilfe von regulären Ausdrücken beschrieben, dass nur bestimmte Zeichen (z.B. a-Z) zulässig sind.

Für die Wahl des Geschlechts wurden Radio-Buttons verwendet. Da beide Buttons den gleichen Namen haben, sind diese miteinander verbunden. Wir also ein Button ausgewählt, wird der andere automatisch abgewählt.

Für das Akzeptieren der Nutzungsbedingung wird eine einfach Checkbox verwendet.

Zum Schluss kommt ein Button vom Typ „submit“, um die HTML-Form abzusenden. Dabei wird ein POST an die Route „/newUser“ mit allen eingetragenen Daten gesendet.

## CSS (registration.css)

Beim body wird zuerst die Schriftart gesetzt. Mit display: flex kann das untergeordnete Element flexibel platziert werden. Die height wird auf 100vh gesetzt, um die volle Bildschirmhöhe einzunehmen. Zusätzlich wird das margin auf 0px gesetzt, um einen Overflow zu vermeiden. Als Hintergrund wird ein Bild verwendet. Dieses soll nicht wiederholt werden (no-repeat) und den gesamten Platz einnehmen (cover).

Registration-wrapper:  
Um die Breite festzulegen wird hier max-width (550px) statt width benutzt. Das bewirkt, dass, bei geringerer Breite vom Browser, der Container automatisch kleiner wird. Mit margin: auto platziert sich der Container horizontal mittig. Die Hintergrundfarbe wird mit rgba(255, 255, 255, 0.6) auf Weiß mit einer Transparenz von 0.6 festgelegt. Der border-radius rundet die Ecken ein wenig ab und durch das padding von 8px haben alle sich darin befindlichen Elemente einen Abstand zum Rand.

H2:  
Mit einer Breite von 100% und text-align: center wird die Überschrift horizontal mittig platziert.

Input-block:  
Mit display: flex und flex: wrap warden die untergeordneten Elemente flexibel angeordnet und bei Bedarf umgebrochen. Die Breite ist auf 100% gesetzt und die minimale Höhe beträgt 50px.   
Justify-content: space-between wird dann benötigt, wenn zwei Input-Elemente nebeneinander dargestellt werden sollen.

Double-column:  
Durch flex-basis: 48% wird die Breite dieser Container festgelegt. Da wie zuvor erwähnt im Elternelement die justify-content Eigenschaft gesetzt ist, werden die Container link und rechts mit Platz dazwischen angeordnet.

Input:  
Mit width: 100% wird immer die maximale Breite genutzt. Margin-top: 4px sorgt für einen Abstand zwischen Input und darüberliegenden Label. Die standardmäßige Border wird durch eine neue mit 1px Breite ersetzt. Um alle Abstände richtig zu berechnen wird box-sizing auf border-box festgelegt.  
Margin-Bottom sorgt für einen Abstand zum nächsten input-block.

Block-header:  
Wird als Überschrift für das Geschlecht benötigt und bekommt eine Breite von 100% und wird fett gedruckt.

Label:  
Alle Label bekommen eine Breite von 100%, um einen Umbruch auszulösen (flex-wrap).

Input[type=text & type=password]:  
Border-Radius für das Abrunden der Ecken und Padding, damit der eingegebene Text einen kleinen Abstand zur Border hat.

Input[type=radio & type=checkbox]:  
Margin um einen Abstand zu den angrenzenden Elementen zu generieren. Width: auto, damit die Inputs auf einer Höhe mit dem Label sind.

Agb-wrapper:  
Alle Eigenschaften von diesem Container sorgen dafür, dass die untergeordneten Elemente flexibel, rückwärts (row-reverse) und beginnend am Ende (flex-end) platziert werden. Das darin befindliche Label wird mit width: auto auf die selbe Höhe wie die checkbox gesetzt.

Button-wrapper:  
Mit align-items: center wird der Button vertikal mittig platziert.

Button:  
Breite wird auf 100% gesetzt und die Höhe auf 40px. Die Border wird entfernt und die Schrift wird auf 1em und fett gesetzt. Hinzu kommt eine Transition von 0.3s, die dann einen Sinn bekommt, wenn wir uns die Hover-Eigenschaft genauer anschauen.  
Hier wird die background-color auf Grau gesetzt. Durch die Transition dauert diese Veränderung nun 0.3. Außerdem wird der Cursor zum Pointer verändert.