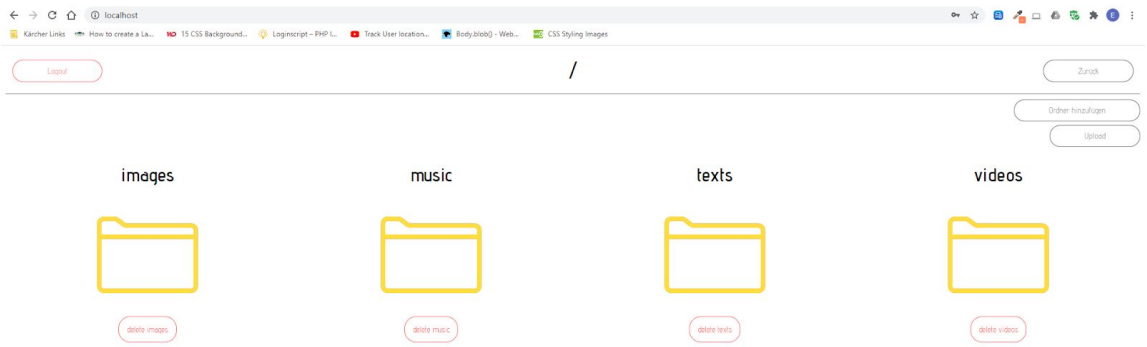
Dokumentation für Web Engineering

Einführung

Nr.	Kategorie	Anforderung	Punkte	
1	Laufzeitumgebung	Sprachen	3	X
2		Keine Bibliotheken **Achtung: Bei Missachtung werden die Punkte der betroffenen Features nicht gewertet!**	5	X
3	Sicherheit	Authentifizierung	5	X
4		Logout	5	X
5	Darstellung	Anpassung an Browserfenster	7	X
6		Usability	5	X
7		Fehlererkennung	4	X
8		Verzeichnisse & Dateitypen	4	X
9	Navigation	Verzeichnis öffnen	4	X
10		Ebene Wechseln	5	X
11		Browser-History	5	X
12		Wiedererkennung	4	X
13	Datei-Handling	MP4-Video-Datei abspielen	3	X
14		Audio-Datei abspielen	3	X
15		Bild-Datei ansehen	4	X
16		Text-Datei bearbeiten	6	
17		Datei herunterladen	6	X
18		Datei löschen	4	X
19		Text-Datei erstellen	5	
20		Datei hochladen	5	
21	Verzeichnis-Handling	Verzeichnis löschen	4	X
22		Verzeichnis erstellen	4	X
			70	

Das Fileservice Projekt für das Modul “Web Engineering 1” erforderte im Allgemeinen, mit einer von Herrn Spörl bereitgestellten Webservice API eine Art Windows Explorer oder Dropbox als Webseite nachzubauen. Hierfür gab es verschiedene Anforderungen an die Herangehensweise und das Ergebnis.
Bevor ich auf die verschiedenen Anforderungen näher eingehe, würde ich gerne kurz erklären, wie mein Code ungefähr aufgebaut ist.
Dieser besteht aus drei Dateien: die index.html, backend.js und style.css. In der HTML Datei ist das Grundgerüst des Programms festgelegt. Das Styling für die gesamte Seite übernimmt die style.css und die Abläufe im Hintergrund werden in der backend.js festgehalten und bestimmt. Die Startseite sieht folgendermaßen aus:



Laufzeitumgebung

Sprachen

Wie bereits erwähnt, wurden im Projektcode lediglich die Sprachen CSS, Javascript und HTML benutzt. Die Applikation ist in modernen Browsern ohne Plugins lauffähig. Alle Dateien sind kommentiert, besonders die backend.js Datei.

Bibliotheken

Es wurden ebenfalls weder Bibliotheken, Frameworks oder sonstige Hilfen bei der Implementierung, außer die API von Herrn Spörl und die History API, genutzt.

Sicherheit

Authentifizierung

Da das Projekt eine Single-HTML Page beinhaltet, musste der Login so gestaltet werden, dass er auf display=”none” gesetzt wird, wenn Eingabe von Passwort und Username übereinstimmen. Hierfür wurde die backend.js über den <head> Tag in der HTML Datei eingebunden und eine XMLHttpRequest Anfrage für den Login geschrieben. Falls dieser erfolgreich war, so wurde der Login-Container auf display=”none” gesetzt und die Mainpage, also die Seite, auf der man alle Inhalte sieht und die interaktiv ist, auf display=”inline” gesetzt (bzw. dessen Container, in der HTML Datei mit der id=”page”). Außerdem wird bei erfolgreichem Login der Authentifizierungs Token erstellt und in den Session Storage gespeichert. Das eingegebene Passwort des Users ist nicht sichtbar, sondern wird in kleinen Punkten dargestellt.

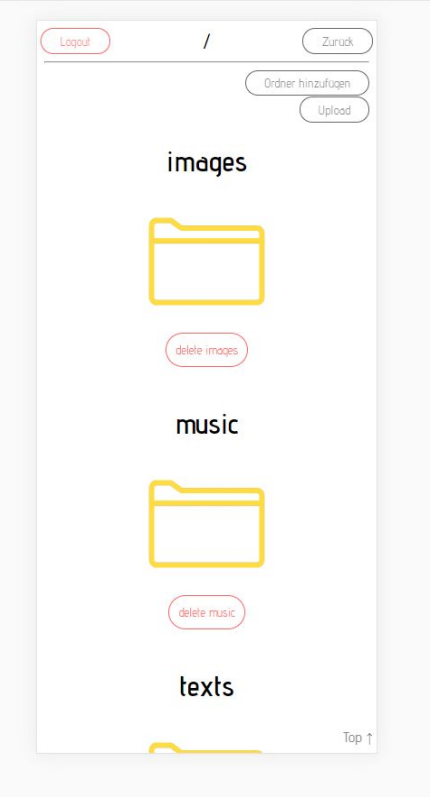
Logout

Für den Logout gibt es oben rechts auch einen fixed Button mit einer onclick Funktion für den Logout. Diese Funktion ist eine Anfrage an die API und loggt den User aus. Dem Requestheader wird der Token, der im Session Storage gespeichert wurde, mitgegeben. Falls der Logout erfolgreich war, so wird der gesamte Session Storage durch sessionStorage.clear() gelöscht. Der gesamte Main Container wird auf display=”none” gesetzt und der Login Container auf display=”inline” gesetzt. Ebenfalls wird der ordnercontent div geleert bzw innerHTML = “” gesetzt. Der Logout funktioniert lediglich auf der Startseite, sobald die Startseite verlassen wird, wird der Logout Button auf disabled gesetzt. Es gibt ebenfalls einen Timer, der resettet wird, sobald eine neue Anfrage aufgerufen wird. Läuft der Timer ab, so wird der User automatisch (nach 10 Minuten) ausgeloggt und erhält den zugehörigen alert.

Darstellung

Anpassung an Browserfenster

Durch das Styling mit Media Queries konnte die Seite an das Browserfenster in allen Größen angepasst werden. Je kleiner das Fenster wird, umso weniger Elemente (Ordner / Files) werden in einer Reihe angezeigt. Ist das Fenster auf Handygröße, so wird nur noch ein Element pro Reihe angezeigt:



Usability

Bei dem Usability wurde darauf geachtet, schlichtere Icons mit etwas schlichteren Farben und Buttons zu wählen und dem User eine übersichtliche Darstellung zu bieten. Wichtige Knöpfe wie Logout, Delete oder Login haben Farben wie bspw. rot oder grün bekommen, der Rest die Farbe weiß mit schwarzem Rand. Alle Buttons haben eine Transition, sodass ein schöner, smoother Hover Effekt entsteht. Damit die Buttons nicht zu sehr hervorstechen, wurden einigen Buttons eine Opacity von 0.6 gegeben, die dann beim Hovern auf 1 erhöht wird. Ordner werden mit einem schlichten, gelben Icon angezeigt und der Download Button einer Datei befindet sich direkt unter ihr. Er ist auch schlicht gehalten und soll nicht so sehr hervorstechen. Allgemein wurde darauf geachtet, nicht zu viele Farben zu mixen und das gesamte Design übersichtlich und schlicht zu halten.

Fehlererkennung

Funktioniert etwas nicht, wie der User es möchte, wie bspw. einen Ordner löschen oder hinzufügen, so werden ihm unmittelbar danach die jeweiligen alerts ausgegeben, dass etwas nicht funktioniert hat. Somit wird der User von allen Fehlern seitens der API informiert.

Navigation

Verzeichnis öffnen

Durch die richtige Implementierung und Setzung des Paths im Session Storage ist es für den User möglich, Ordner bzw. Verzeichnisse zu öffnen und diese anzusehen. Öffnet er ein Verzeichnis, so wird die Funktion getOrdner() aufgerufen, die den Response der API in ein JsonObjekt umwandelt und dann als Parameter der Funktion drawOrdner(JsonObjekt) übergibt. Dies ist ein rekursiver Aufruf, da in drawOrdner() gecheckt wird, ob ein Ordner angeklickt wird und wenn ja, dann wird wieder dieser Ordner durch getOrdner bekommen und alle anderen Elemente werden auf display=”none” gesetzt. Der User kann somit problemlos Verzeichnisse öffnen und auch Verzeichnisse in Verzeichnissen öffnen. Dies bildet die Grundlage für die Browser History und das Wechseln der Ebenen.

Ebenen wechseln

Durch die Implementierung eines Zurück Buttons, der oben rechts fixed auf der Seite ist, ist es für den User möglich, in sein Elternverzeichnis zu wechseln. Hierfür wurde der Befehl window.history.go(-1) verwendet, welcher auf Grundlage der History API basiert.

Browser History

Die History API sorgt im Backend dafür, dass jeder Path, auf dem man sich befindet, durch PushState gespeichert wird. Falls nun ein neues Verzeichnis aufgerufen wird, wird durch einen Popstate Eventlistener das Verzeichnis in der History geladen und dessen Ordner wird durch getPushStateOrdner() geladen. Hierfür konnte leider nicht die normale Funktion getOrdner() verwendet werden, da sonst die Dateien aus dem Kinderordner auch im alten Ordner, in den man zurück navigiert, angezeigt werden würden. Durch diese extra Funktion getPushstateOrdner() wurde dieser Fall verhindert und der User kann problemlos zwischen den Verzeichnissen hin- und hernavigieren.

Wiedererkennung

Der User weiß durch den Text, der ihm den Path des atuellen Verzeichnisses anzeigt immer, wo er sich befindet. Dies ist daher möglich, weil der Path im Session Storage gespeichert ist und die Überschrift über das JS mit .innerHTML immer durch den aktuellen Path ersetzt wird.

Datei-Handling

MP4-Video-Datei apsielen

Der User kann ebenfalls .mp4 Videos abspielen. Hierfür gibt es eine Funktion, die überprüft, welchen Typ das geklickte Element (div) hat. Ist dieses Element vom type “video”, so wird eine Dataurl mit dem Responsetext im Base64 Format erstellt und als source dem Video element hinzugefügt. Das Video Element hat ebenfalls das Attribut “controls”, sodass der User auch das Video stoppen, pausieren, die Lautstärke verändern und vieles mehr machen kann.

Audio-Datei abspielen

Parallel zum Video geschieht dasselbe für das Audio. Der User kann ebenfalls Audios abspielen und diese auch kontrollieren, sofern das geklickte Element ein Audio ist (Dies erfährt er durch das Icon, das ihm anzeigt, um welchem Typ es sich handelt. Hierfür gibt es die Funktion showType(type) im JS, die dementsprechend die Icons zuweist).

Bild-Datei ansehen

Der User kann ebenfalls ein Bild ansehen. Das Verfahren ist genau gleich wie bei Audio und Video.

Text-Datei bearbeiten

Der User kann leider nur Text-Dateien öffnen und downloaden. Durch die Funktion “atob” wird der Base64 verschlüsselte Text entschlüsselt angezeigt. Durch die mangelnde Zeit konnte die Bearbeiten Funktion leider nicht mehr implementiert werden.

Datei herunterladen

Der User kann jede beliebige Datei downloaden, die sich in den Verzeichnissen befindet. Jede Datei, die es gibt, bekommt durch das JS einen Download Button, bzw. <a> Element, mit dem Link der zugehörigen Dataurl zugeordnet, sodass jede beliebige Datei gedownloadet werden kann.

Datei löschen

Durch die Funktion deleteOrdner() können ebenfalls auch Dateien gelöscht werden, da die Delete Anfragen für das Löschen von Ordnern und Dateien gleich und unabhängig vom Dateityp sind. Jede Datei und jeder Ordner hat einen Delete Button, der beim Klicken die delete-Funktion aufruft und anschließend den Ordner oder die Datei löscht. Ordner können nur gelöscht werden, wenn sie leer sind. Auch hierfür gibt es einen Alert. Wird ein Element gelöscht, so wird es im HTML durch das JS auch direkt auf display=”none” gesetzt.

Text-Datei erstellen

Leider kann der User keine Text-Datei auf der Seite erstellen.

Datei hochladen

Der User kann leider auch keine Dateien hochladen. Lediglich die UI wurde hierfür implementiert, besitzt jedoch keine richtigen Funktionen. Die Anfrage für den Upload gibt es auch im JS, diese wurde aber leider auch noch nicht wegen mangelnder Zeit genutzt.

Verzeichnis-Handling

Verzeichnis löschen

Durch den delete Button an jedem Ordner ist es für den User möglich, ein ausgewähltes Verzeichnis im aktuellen Path löschen zu können. Dieses muss, wie bereits erwähnt, leer sein. Ist dies nicht der Fall und der User möchte das Verzeichnis dennoch löschen, so erhält er einen Alert, der ihm das Problem erklärt.

Verzeichnis erstellen

Der User kann durch den Button “Ordner hinzufügen” in dem aktuellen Path einen Ordner hinzufügen. Drückt er den Button, so öffnet sich ein Dialog mit einem Eingabefeld für den neuen Ordner- bzw Verzeichnisnamen. Dort kann der User den Namen des neuen Verzeichnisses im aktuellen Path eintippen und anschließend mit dem Button “Hinzufügen” den Ordner erstellen und anzeigen. Im Hintergrund läuft hierfür die Funktion addOrdnerElement(name = input.value), die mit dem Klick auf “Hinzufügen” aufgerufen wird.

Fazit

Das Projekt beinhaltet fast alle Punkte aus dem Anforderungskatalog bis auf Text-Datei erstellen, bearbeiten und Dateien hochladen. Bei der Entwicklung des Backends gab es einige Probleme durch die vielen Verknüpfungen der Funktionen, die jedoch mit Hilfe von Herrn Spörl, viel Denken und Recherche meist gut gelöst werden konnten. Für die Verknüpfung meiner Funktionen habe ich mit Stift und Papier eine kleine Skizze erstellt, die zeigen soll, wie der JS Code ungefähr aufgebaut ist: