Projektbericht für das Semesterprojekt im Modul

Internetanwendungen für mobile Geräte

SS 2018

Stand Montag, 04.07.2018

Inhaltsangabe

Projektbeschreibung

Arbeitsname:

mentionIT (in Anlehnung an to mention something)

Motivation:

Ich persönlich komme oft in die Verlegenheit, mir Notizen in einem gewissen Kontext machen zu wollen, um sie immer verfügbar zu haben. Häufig gestaltet sich dies kompliziert, da zum späteren Verständnis weitere Informationen erfasst werden müssen. Eine Lösung könnte sein, statt einer einfachen textlichen Nachricht, diese zusätzlich mit einem Bild oder einer Sprachnachricht zu kombinieren und für einen definierten Personenkreis einfach und zentral verfügbar zu machen.

Funktionsbeschreibung:

Ein Nutzer kann Einträge erzeugen, einsehen, ändern und löschen. Bei der Gestaltung ist er hierbei bis auf eine Überschrift frei in der Kombination der verfügbaren Bestandteile Bild, Text und Tonaufnahme.

Verbindet sich die Anwendung mit dem Internet, werden die vorhandenen Einträge vom Server abgerufen und deren Überschriften angezeigt. Wird ein Eintrag ausgewählt, ist er für die Bearbeitung frei. Die Anwendung soll sich responsive verhalten.

Wird die Verbindung unterbrochen, können neue Einträge angelegt und gelöscht werden. Wurde ein Eintrag neu angelegt oder zuvor angeschaut, kann auch dessen Inhalt bearbeitet werden. Bilder und Sprachnachrichten können für Einträge, die bereits angeschaut wurden, gelöscht werden. Ein Hinzufügen neuer Bilder oder Aufnahmen ist offline jedoch nicht möglich. Nach der Wiederherstellung der

Verbindung wird im Hintergrund eine Synchronisation durchgeführt. Die Anwendung zeigt an, wie der aktuelle Verbindungsstatus ist.

Über eine Registrierung können sich Nutzer registrieren. Registrierte Nutzer sehen nach einer Anmeldung die Einträge aller Nutzer. Dies soll auch so sein. Einträgen wird automatisch der jeweilige Nutzer hinzugefügt. Dieser kommt auch zur Anzeige.

Verwendete Techniken, Sprachen, Bibliotheken und Frameworks

Serverseitig kamen u.a. zum Einsatz:

- Betriebssystem Linux
- MongoDB (serverseitige DB)
- Node.js mit diversen npm Paketen
  - Javascript (Programmierung)
- Express.js mit den Paketen
  - bcrypt (Verschlüsselung)
  - body-parser (Zugriff auf übergebene Parameter)
  - cookie-parser (Zugriff auf Cookies)
  - multer (Fileupload, Speicherung, FormData)
  - mongodb (Treiber für MongoDB
  - mongoose (Zugriffsschicht für MongoDB)
  - mongo-sanitize (Säuberung von Mongo Eingaben)
  - escape-goat (einfaches es- und unescapen)
  - express-session (Session handling)
  - forever-service (Service Tool)
  - helmet (Absicherung des Servers)
  - pug (ehemals Jade als Template Engine)

Clientseitig kamen u.a. zum Einsatz:

- HTML5 (Struktur)
    - Web Worker (Synchronisation)
    - offline-manifest (Cachen von Ressourcen)
    - Web-App-Manifest (Layout Handy)
- CSS3 (Layout)
- Bootstrap 4 (CSS Framework)
- Javascript (Dynamik und Anwendung)
    - FormData (Datenübertragung)
    - Fetch API (statt xmlHTTPRequest)
    - Web Storage (local und session storage)
- jquery (Javascript Bibliothek)
- Indexed DB (clientseitige DB)
    - Dexie.js (Wrapper für Indexed DB)

Serverseitiger Aufbau

Serverseitig basiert die Anwendung auf einen Node.js Webserver. Zur Vereinfachung wird das Framework Express genutzt, welches viele Standardaufgaben übernimmt
bzw. vereinfacht. Für das Routing wird das neue Route Objekt von Express verwendet, welches gegenüber dem herkömmlichen Vorgehen flexibler ist und die Zuständigkeit weiter kapselt. Als Template-Engine kommt Pug zum Einsatz.

Zur Datenhaltung wird MongoDB mit dem npm Paket Mongoose eingesetzt. Diese bietet den Vorteil, einen modellbasierten Zugriff auf MongoDB zu haben und lässt sich gut in Express einsetzen. Zudem können die entsprechenden Modelle mit weiterer Logik zur Validierung, Vor- und Nachbereitung genutzt werden. Eingaben über Mongoose werden gecleant.

Zur Kommunikation des Servers mit der Anwendung existiert ein Satz an Funktionalitäten zum Anlegen, ändern und Löschen der verwalteten Objekte. Die

Aktion selbst wird hierbei wie üblich über die Art des Requests (Get, Post, Put, Delete) gehandelt. Rückgaben sind entweder entsprechende Seiten oder aber mindestens ein Status. Hierbei werden bewusst nur wenige Status berücksichtigt (200, 501, 401,..) um so wenig Informationen wie möglich auszugeben. Sämtliche Eingaben werden escaped. Bewusst erfolgt bei der Ausgabe kein unescapen. Weiter wird Helmet als Middleware eingesetzt. Das npm Paket sorgt bereits in seiner Grundeinstellung für eine möglichst sichere Konfiguration des Headers und wird nicht zuletzt auf der Express-Seite dringend als minimalen Schutz empfohlen.

Die Anwendung verfügt über eine eigene Benutzerverwaltung, welche mittels Session und Cookies umgesetzt wird. Für das Hashing der Passwörter wird das npm Paket bcrypt verwendet. Die Datenhaltung selbst erfolgt wiederum in MongoDB. Die Durchsetzung der Berechtigung erfolgt über eine kleine Middleware innerhalb von Express. Hierbei wird bei einem Login geschaut, ob der Nutzer in der Datenbank vorhanden ist und das verwendete Passwort gültig ist. In diesem Fall wird der Session die ID des Nutzer angehangen und er erhält Zugriff auf die Ressourcen.

Der Dateiupload wird über Multer mit seinem konfigurierbaren Storage Mechanismen unterstützt. Hierfür wird es als Middleware eingesetzt. Multer ist auch für den einfachen Zugriff auf übermittelte FormData Objekte zuständig und sehr nützlich.

Alle ausgelieferten Seiten liegen als pug-Templates vor, sämtliche clientseitigen Bibliotheken werden ebenso vom Server bezogen.

Clientseitiger Aufbau

Clientseitig ist die Anwendung als eine Single Page Application mit einem responsiven Verhalten konzipiert. Nachdem die Basisseite geladen wurde, wird nur noch deren Content aktualisiert. Unterstützend kommt das Framework Bootstrap 4 zur Gestaltung des Layouts zum Einsatz. Für die Dynamik sorgt Javascript zum Teil in Verbindung mit JQuery zur Vereinfachung der Selektionsvorgänge und

DOM-Manipulationen. Um die Funktionalität der Anwendung im Offlinemodus sicher zu stellen, wurden verschiedene Anstrengungen unternommen.

Alle notwendigen Ressourcen wurden in einem offline manifest aufgenommen, welches auf der Hauptseite im HTML-Tag mit der Eigenschaft manifest eingebunden ist. Hauptsächlich dient dies dazu, offline Zugriff auf die eingesetzten Skripte zu haben. Das Cachen der Seiten bzw. der einzelnen Inhalte selbst wurde hingegen mit einem eigenen Mechanismus umgesetzt. Sobald eine Seite angefordert wird, wird der Response in einem session storage abgelegt. Bei einem erneuten Aufruf wird dieser Eintrag überschrieben. Als Schlüssel dient die Route des Requests. Wird nun die Verbindung unterbrochen, so wird zunächst geschaut, ob für die aktuelle Route bereits ein Ergebnis vorliegt. Ist dies der Fall, so wird dies verwendet. Ist dies nicht der Fall, erfolgt ein entsprechender Hinweis. Namentlich bedeutet dies, dass in der Regel nur bereits besuchte Bereiche offline verfügbar sind. Ausgenommen hiervon sind triviale Seiten, namentlich z.B. das Impressum. Diese werden zu Anfang im local storage gecacht und auch primär aus diesen bezogen.

Offline durchgeführte Aktionen werden in einer IndexedDB protokolliert, sobald keine Verbindung zum Server besteht und die Oberfläche mit ihren Daten entsprechend angepasst. Nach einem Reconnect werden diese Aktionen dann sequentiell im Hintergrund abgearbeitet. Eine Notwendigkeit für eine explizite Aktualisierung besteht daher nicht. Zur Umsetzung wurde die Web Worker API eingesetzt, welche die Ausführung eines Skriptes ermöglicht ohne das Hauptscript zu blockieren.

Für die Umsetzung der notwendigen Manipulationen des DOMs zur Anpassung der Seiten im Offlinemodus (um Änderungen zu reflektieren) kommt Javascript und JQuery zum Einsatz. Für die Erstellung des HTML-Contents der offline neu erstellten Einträge wird das neue Template-Tag verwendet. Es wird vom Browser nicht gerändert, kann aber Bestandteil eines HTML-Dokumentes sein. Innerhalb der Anwendung wird es genutzt, um den Code für die Details eines Eintrags und dessen

“Card” für die Hauptseite zu erstellen. Unter anderen mit Hilfe der Funktion CreateDocumentFragment des Document-Objektes lässt sich so relativ einfach dynamisch neuer Content erstellen.

Der Zugriff auf die IndexedDB erfolgt im Kontext der Hauptseite mittels der Bibliothek Dexie.js, welche die native API kapselt und einen Promis-basierten Zugriff bietet. Innerhalb des Workers wird hingegen auf die Einbindung verzichtet und direkt die API der Datenbank verwendet. Der Grund hierfür ist, dass ich mich selbst mit dem nativen Zugriff auseinander setzen wollte, auch wenn ich ihn als eher unhandlich empfinde. Grundsätzlich kann man bei der Verwendung von Web Worker auch Skripte mittel import importieren.

Im Rahmen der Anwendung und Datenhaltung wird so verfahren, dass beim Verlassen der Seite oder dem Neuladen möglichst viel Informationen gelöscht wird. Hierzu gehört, das mit jedem Neuladen der Seite die genutzte IndexedDB neu erzeugt wird. Loggt sich ein Nutzer aus, so wird der Session storage geleert. Nur der local storage mit nicht relevanten Inhalt verbleibt auf unbestimmte Zeit.

Um Einfluss auf die Erscheinung zu nehmen, die meine Anwendung auf mobilen Devices hat, wenn sie dort “installiert” wurde, ist eine web app manifest Datei hinzugefügt worden. Hierbei handelt es sich um ein JSON-File, dass im HTML verlinkt ist.

Probleme und Lösungen

Zunächst möchte ich vorausschicken, dass ich außerhalb des Studium leider sehr wenig mit Webprogrammierung zu tun habe. Mein Arbeitsgebiet sind Datenbanken und die Programmierung in C#. So gebrauchte ich doch Zeit, um mich nach längerer Pause wieder in die Thematik und dem früher Gelernten einzufinden.

Auch die Serverseitige Programmierung war mir zumindest im Zusammenhang mit Node.js neu, ebenso wie Express, Bootstrap, IndexedDB, Dexie.js oder aber JQuery, um nur ein paar Dinge zu nennen, die ich erst im Verlauf des Semesters kennen lernte und in die ich mich zumindest grundsätzlich einarbeitete. Besonders bei Express und seinen Paketen gestaltete sich dies sehr zeitaufwändig.

Zwei Dinge beeindruckten mich während der Arbeit besonders. Dies war zum einen das sehr oft zu findende Konzept der Promises und die Art, wie ein Framework wie Express eingehende Anfragen verarbeitete und wie deren Verarbeitung mit Hilfe von (fertiger) Middleware, aber auch eigenen Code beeinflusst werden kann.

Besondere Schwierigkeiten machten mir die grundsätzliche Konzeption einer Webseite, die weniger Seite und vielmehr eine Anwendung sein sollte. Als ein Erfolg für mich sehe ich hier auch meine gewonnenen Kenntnisse und Einsichten über die Möglichkeiten an, auch clientseitig Daten auf verschiedene Art zu halten und unabhängig vom Server agieren zu können. Gerade die Möglichkeit mit Hilfe von Web Workern eine Art Multithreading mit Javascript umzusetzen bietet im Zusammenspiel mit IndexedDB viel Potential, hat aber auch viel Zeit für die Einarbeitung in Anspruch genommen.

Grundsätzlich musste ich auch feststellen, dass der clientseitige Code meiner Anwendung zwar noch überschaubar ist, er aber zur Grenze dessen ist, wo sich der Einsatz eines Strukturframeworks wie Angular oder React lohnen würde. In den Webseiten, die ich bisher kannte oder erstellte, wurde letztlich Inhalt nur angezeigt und erhielt mit Javascript etwas Dynamik. In einer Webanwendung wie der aktuellen hingegen kommen Objekte vor, die Eigenschaften haben. Die einen Status haben und die unter Umständen noch je nach Ihren Zustand anders innerhalb ihrer Umgebung agieren sollten. Aus meinen jetzigen Kenntnisstand heraus würde ich daher versuchen ein solchen Framework einzusetzen.

Etwas, dass weniger mit dem Code zu tun hat und mir zunächst etwas Schwierigkeit bereitete war die Art, für das Projekt Code zu schreiben und zu debuggen. Es ist eine große Umstellung, wenn man nur die Arbeit mit Desktopanwendungen gewohnt ist. Auch die lockere Bindung war ungewohnt und ich verbrachte als Beispiel viel Zeit damit, die Offline Synchronisation grundlegend hinzubekommen bzw. ein Gefühl dafür zu entwickeln.

Mögliche Verbesserungen

Wie ich schon angedeutet habe, konnte ich im Verlauf des Semesters für mich ziemlich viel mitnehmen. So würde ich heute von vornherein die gesamte Funktionalität mit einer offline-Funktionalität versehen, statt nur ein Teil, da ich die Thematik nun bessere verstehe und den Aufwand besser abschätzen kann. Auch fallen einen immer mehr Dinge auf, die nicht schön sind, aber dafür lernt man ja. Auf vier Punkte möchte ich kurz eingehen:

Zum einen ist dies die Bedienerführung. Man erkennt, dass ich eher gewohnt bin, im Desktop-Bereich GUIs zu entwickeln. Die aktuelle Oberfläche zeigt sich leider durchaus als mit einem Handy bedienbar, aber nicht sehr einfach und sehr effizient. Häufig sind als Beispiel unnötige Wege zu scrollen. Hier gibt es viel Raum, es besser zu machen.

Der nächste Punkt ist das Verhalten bei der Dateiauswahl und dem Upload. Das hier eine Offlinefunktionalität schön wäre, ergibt sich aus dem gesagten. Daneben ist die Dateiauswahl und das Verhalten bei größeren Dateien nicht schön. Beim ersten Punkt wäre ein Drag and Drop besser, beim Zweiten könnte die Nutzung eines Web Workers die UX verbessern.

Der dritte Punkt ist die Struktur der Anwendung auf der Clientseite. Die Logik funktioniert und bis auf ein paar kleine Bugs arbeitet sie auch korrekt. Als ich anfing, daran zu arbeiten, ging ich so vor, wie ich bei der Erstellung einer einfachen Seite

vorgehen würde. Das hat Nachteile, da die Inhalte der Anwendung nicht einfach statisch sind, sondern eher als eigene Objekte mit Eigenschaften angesehen werden können und sollten. Ein Strukturframework könnte sinnvoll sein.

Als letzten Punkt bin ich mir nach wie vor unsicher, ob man den Server nicht noch besser schützen könnte. Es gibt viele Information zu diesem Thema (die sich in Teilen widersprechen) und es wäre interessant, hier einen erfahrenen Entwickler beobachten zu können.