Kaffeeverwaltung Dokumentation

Kaffeeverwaltung.doc

Version: 1.0

Projektbezeichnung	coffeesnitch	
Projektleiter	Maik Scherr	
Verantwortlich	Konstantin Fein	
Erstellt am	26.07.2017	
Zuletzt geändert	01/09/2017 10:26	
Bearbeitungszustand		in Bearbeitung
	X	vorgelegt
		fertig gestellt
Dokumentablage	lokal	
V-Modell-XT Version	Version 1.1.0	

Inhalt

1 Autostart

Beim Start des Raspberry Pi werden automatisch Skripte ausgeführt, welche den Node.js Server und Nginx Server starten. Diese Autostarts werden standardmäßig in der rc.local Datei unter ***/etc/rc.local/*** gespeichert. In dieser Datei werden Befehle als ***root*** ausgeführt. In der Datei wird mittels ***su*** der User zu pi gewechselt, um die lokalen Benutzerdaten zu nutzen. Im Folgenden werden Skripte auf dem Desktop des Benutzers aufgerufen. Diese Skripte rufen wiederrum andere Skripte auf und setzen diese in den Hintergrund. Dies hat den Sinn, dass der Pfad einfach als Benutzer bzw. durch SSH Zugriff geändert werden kann. Diese Autostarts werden nur beim Starten des Raspberrys ausgeführt. Sollen Autostarts bei jedem SSH Zugriff ausgeführt werden, so müssen die Befehle unter der ***/home/pi/.bashrc*** Datei gespeichert werden. Wollen Startvorgänge abhängig vom eingestellten Runlevel ausgeführt werden, müssen sie in der ***/etc/rc*/*** Datei aufgerufen werden. Beim Kaffeeverwaltung Projekt wurde nur die *rc.local* Datei als Autostartdatei verwendet. Im Folgenden finden Sie eine Übersicht zu den aufgerufenen Skripten:

Autostart Dateiname	Funktion
startApp.sh	Startet den Nginx Server, welcher die Angular Applikation dem Benutzer bereitstellt
startCoffeeMgmgServer	Startet den Node Server, der den allge-

	meinen Plug-In Betrieb gewährleistet
startRfidScanner1	Startet den 1. Rfid Scanner
startRfidScanner2	Startet den 2. Rfid Scanner

2 Node Server

Der Node Server ist unter dem Dateipfad ***/home/pi/git/coffeesnitch/*** zu finden. Dort ist zum einen das Plug-In gespeichert, welches nur für den Benutzer wichtig ist, aber hier nochmal zur Sicherheit abgespeichert ist. In dem ***database*** Ordner befindet sich die Kundendatenbank mit den Informationen der Benutzer. In dem Ordner ***rfid_adapter*** befinden sich die Skripte für die Rfid-Lesegeräte. Der Hauptteil befindet sich im ***Server*** Ordner. Dort befindet sich unter dem Ordner ***Adapter*** welcher die Datei ***JsonAdapter.js*** beinhaltet. Dieser Adapter hört auf http-Requests und legt diese Anfrage auf den internen Event Bus. Der Event Bus ist eine Topologie, die alle Module des Servers miteinander verbindet und Kommunikation untereinander ermöglicht. Http- Anfragen sind standardmäßig Json Nachrichten, welche vom Adapter verarbeitet werden. Somit ist es möglich auf die einzelnen Inhalte eines Objektes zuzugreifen.

In der ***system.json*** ist der Dateipfad der Module angegeben über diese Datei wird der Server gestartet und gestoppt. Dazu wird in der Bash der Befehl: ***pm2 start/stop system.json*** verwendet.

In dem Ordner ***Modules*** befindet sich mit den Modulen das Kernstück des Servers.

2.1 BrowserNotificationModule

Im BrowserNotificationModule kommunizieren die Module mit dem Benutzer. Nur hier werden Benachrichtigungen und Events an den Benutzer gesendet und sonst nirgendwo. Die Verbindung mit dem Benutzer wird über einen WebSocket hergestellt. Konstanten mit der Bezeichnung ***NOTIFICATION_EVENT*** sind werden als Subscripe Type angegeben während Konstanten mit dem Namen ***EVENT*** für die sendMessage-Methoden verwendet werden. Über diesen Namen werden die Nachrichten beim Benutzer erfragt. Der Event-Handler („client.start“) wird in jedem Modul benötigt, um untereinander kommunizieren zu können.

2.2 CountDownModule

Das Countdownmodul startet einen Timer, der die Zeit bis zur Fertigstellung der Kaffeekanne zählt. Im ersten Teil werden die Minuten und Sekunden aus der Zeit herausgeholt, um auf dem Raspberry die verbleibende Zeit anzuzeigen. In der Switch-Anweisung werden in jedem Viertel Statusnachrichten an den Benutzer gesendet (0%, 25%, 75% und 100%). Diese dienen dazu die Statusbar auf Seite des Benutzers zu ak-

tualisier. Außerdem wird gesondert ein Event an den Benutzer gesendet, wenn die Zeit abgelaufen ist.

2.3 MappingModule

Das MappingModul arbeite mit den Lesegeräten zusammen und erhält von diene die RFID-Tags. Über eine simple Switch-Abfrage werden die Tags auf die Stockwerke zugewiesen. Die erneute Abfrage wird gesperrt und mehrere Events werden ausgeführt. Das erste Event gibt den Datenbankzugriff zum Abholen der Kaffeekannen wieder frei. Das zweite Event informiert über dem Benutzer über das Eintreffen einer Kaffeekanne. Das dritte Event Startet den Timer mit einer Zeit von 480 Sekunden also 8 Minuten und übergibt die Kaffeekannen ID Das letzte Event kontrollier den Brauvorgang.

2.4 DatabaseModule

Als Datenbank wird die Dokumentenbasierte Datenbank **lowDB** verwendet, welche die Daten in einem Json File speichert. Diese Open-Source Datenbank hat den Vorteil, dass sie leichtgewichtig ist und keine große Rechnerauslastung beansprucht, was sehr wichtig für den Betrieb auf dem Raspberry ist. Durch die beiden Konstanten ***db*** und ***dbCoffee*** werden zwei Unterschiedliche Datenbanken erstellt. Zum einen eine Benutzerdatenbank, welche ausschließlich Kundendaten enthalten (Gebäude,

Stockwerk, Name …). Mit low wird die Konstante als Datenbank gekennzeichnet und einen Speicherplatz zugewiesen. Defaults setzen die Datenbanksturkutur vergleichbar mit Spaltennamen in einer SQL Datenbank. Das Datenbankmodul ist das einzige Modul, dass Nachrichten vom Benutzer erhalten. Dies geschieht über die Funktion ***server.on(„Message“, …*** Darin wird zuerst die Anzahl an Benutzern gezählt, um die ID für den neuen Benutzer zu bestimmen. Unter der Funktion ***addUser*** werden zuerst Daten für die zweite Datenbank, die Angular Datenbank gesammelt. Zuerst wird für die Historie und den Status (siehe Angular Website) das aktuelle Datum mit Zeit erfasst und gespeichert. Danach wird an die Kundendatenbank weitergeleitet. Hier wird geprüft ob ein Benutzer mit dem Benutzernamen x im Gebäude y auf Stockwerk z bereits existiert und gibt bei Existenz false als Rückgabewert zurück. Andernfalls wird true zurückgegeben und ein neuer Benutzer erstellt. In der Funktion ***changeData*** wird über den Übergabeparameter ***field*** die Benutzerdaten geändert. Dabei gibt field folgendes an:

Nummer	Funktion
1	Ist der Benutzername
2	Ist das Gebäude
3	Ist das Stockwerk
4	Ist der Kaffeeverbrauch
5	Ist die Anzahl an Abholvorgänge

6	Ist die gesamte Brautzeit

Dabei wird immer zuerst überprüft, ob der Benutzer überhaupt existiert

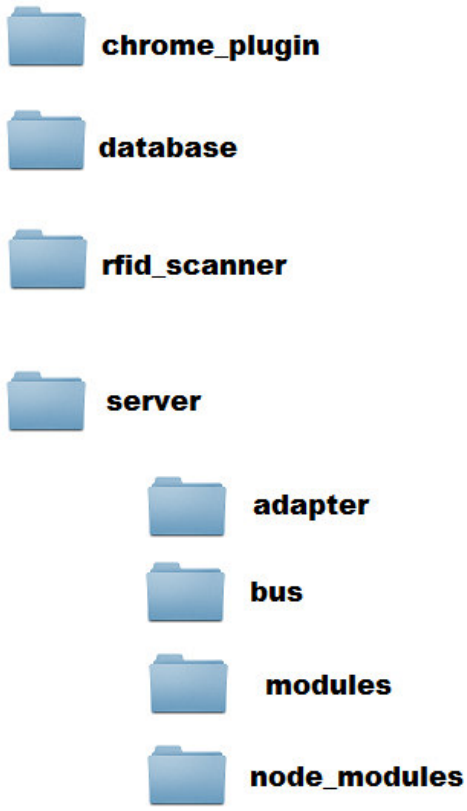
Die Funktion ***returnData*** gibt Kundendaten nach obigen Schema zurück.

Die Funktion ***UpdateUser*** prüft den Rückgabewert von der Funktion ***addUser,*** falls der Rückgabewert true ist, wird ein neuer Benutzer angelegt und nichts upgedated, bei false werden die Werte entsprechend über die Funktion **changeData** aktualisiert.

Die letzte und größte Funktion ***updateAngular*** aktualisiert die Angular Datenbank, darin werden die Benutzerdaten durchgelaufen und sammelt die Daten und trägt sie in die Angular Datenbank ein. Werden Daten nicht gefunden, so wird ein Standardwert in die Datenbank eingetragen.

Zuletzt werden Benutzernachrichten empfangen und nach dem Inhalt geprüft. Benutzerdaten sind ähnlich CSV-Dateien durch ein Komma getrennt und werden als String übergeben. Aus diesem String werden die einzelnen Daten gelesen, jenachdem wie der String aufgebaut ist (Länge des Strings ausschlaggebend). Je nach Inhalt des Strings werden unterschiedliche Operationen durchgeführt. Zuletzt besitz das Modul einen

Event-Handler, um mit anderen Modulen zu kommunizieren.



3 Raspberry Pi Allgemein

Der aktuelle Raspberry ist der Raspberry Pi der ersten Generation mit dem ARMv6 Chipsatz, der noch sehr langsam ist. Die ***Netzwerkeinstellunge*** sind in der Datei ***/etc/network/interfaces*** zu finden. Sowohl Lan als auch WLan-Verbindung wurden als statisch deklariert, damit man beim Plug-In stets eine konstante IP-Adresse angeben kann. Zudem wurde ein DNS-Dienst eingestellt. Auf dem Raspberry läuft die ***Node Version v6.9.0*** und die ***NPM Version v3.10.8***. Damit kann man Angular

4 auf dem Raspberry Pi betreiben. Angular 4 ist global auf dem Raspberry installiert, kann also überall genutzt werden.

4 Angular Anwendung und Nginx Server

Auf dem Raspberry Pi findet man unter ***/home/pi/git/Website/*** den Ordner ***Kaffeeverwaltung*** oder auch wahlweiße in der Git Repo. Dieser Ordner beinhaltet alle Elemente für die Website ist jedoch nicht die auf dem Raspberry laufende Anwendung, sondern nur für die Entwicklung der Anwendung notwendig. In dem ***Src-Ordner*** findet man allgemeine Dateien zur Angular Anwendung wie das Icon und das globale Stylesheet ***styles.css.*** Das HTML-File ***index.html*** dient als Einstieg für die Anwendung und enthält unteranderem den allgemeinen HTML-Aufbau, der somit in den darunterliegenden Dateien nicht mehr benötigt wird und den Ladescreen, falls das Laden der Website länger dauert. Der Ladescreen ist ein einfaches Pixel Art, welches einen Text symbolisiert. Durch dieses Verfahren wird kein zusätzlicher Ladeaufwand durch aufwendige Bilder oder Animationen beansprucht und trotzdem genießt der Anwender eine optisch ansprechende Gestaltung. Unter ***Assets*** befinden sich Bilder und Icons für die Website und zudem die ein Json-Datenbankdokument, welches alle Daten für die Website beinhaltet und von Angular in den HTML Körper geladen wird. Im Angular Ordner ist dieses Datenbank Dokument nicht an den Server gebunden, sondern dient lediglich zur Entwicklung. Unter ***App*** befinden sich die Komponenten der Website. Zentraler Punkt sind die App Komponenten in der ***app.component.html*** sieht man den Aufbau der Website. Der Header,

die Navigationsbar und der Flotter bleiben immer identisch nur der ***<router-outlet>*** Bereich ändert sich beim Drücken auf die Knöpfe der Navigationsbar. Aufgrund von Nginx wird hier erst das Website Icon geladen. Der ***<router-outlet>-***Tag steht in Verbindung mit der ***header*** Komponenten, welches in der HTML jeweils nur auf andere Komponenten verweist. Das heißt bei der Navigation auf der Website ändern sich der Header-, Navigation und Fotter -bereich nicht! Lediglich der Seitenkörper wird jeweils neu geladen. Dies verringert Ladezeiten, die auf dem Raspberry länger ausfallen können. Ansonsten findet man in den folgenden Ordnern den Inhalt der Seiten. Besonders wichtig sind die Ordner ***header, nav*** und ***fotter.*** Diese enthalten die zuvor erwähnten statische Websitekomponenten.

4.1 Header Komponente

Das Header Komponente ist der kleine Bereich über der Navigationsbar, dass das Firmenlogo, einen Schriftzug und Social Media Knöpfe beinhaltet. Alle Komponenten passen sich der Bildschirmgröße an, deswegen ist das CSS-Dokument recht groß jedoch aufgrund der Kommentare gut zu lesen.

4.2 Navigation Komponente

Das Nav Komponente ist die Navigationsleiste und besteht aus einem simplen HTML-Code der nur eine ungeordnete Liste mit Verweisen, die anstelle des href-Tags ein ***Router Link-Tag*** haben, um mit dem übergeordneten App Komponenten zu interagieren. Mit CSS wird anschließend die Ungeordnete Liste vertikal angeordnet und mit der Farbkombination versehen.

4.3 Startpage Komponente

Das Startpage Komponente ist für die Startseite zuständig. Die HTML Seite besteht aus einem Textbereich, einem Konfigurationsbereich und einem Bildbereich. Der Textbereich ist im <main>-Tag enthalten und teilt sich in einen <articel> und einem <aside>-Tag auf. Unter dem Tag lässt sich eine Diashow von Bilder zum Projekt finden! Auch diese Komponente passt sich an die Bildschirmgröße an. Ansonsten besteht die CSS-Datei nur aus Style Anweisungen für die Diashow.

4.4 Komponenten: statistic, person-statistic und coffee-supplier

Diese Komponenten zeichnen sich dadurch aus, dass sie Statistiken enthalten. Hierfür wird NG2 Charts verwendet. Statistiken sind durch <canvas …> Tags gekennzeichnet. Für ausgefallenere Statistiken empfiehlt sich das bereits einsatzbereite D3 Js. Diese Komponenten enthalten allesamt einen Service, der die Daten aus dem JSON Datei rausliest

und bereitstellt. In der component.ts werden diese Informationen dann in die Statistiken eingetragen. Die Styles für die Statistiken werden größtenteils von der API bereitgestellt, daher befinden sich hauptsächlich Styles für die Knöpfe in der CSS-Datei! Die Statistiken passen sich nicht der Bildschirmgröße an, sondern bleiben immer gleich groß.

4.5 Suggestion Komponente

Diese Komponente nimmt Vorschläge zum Projekt entgegen. Dabei wird der Name, die E-Mail, der eigentliche Vorschlag und die Priorität entgegengenommen und geprüft. Falls die Eingabedaten inkorrekt sind, wird unter dem betreffenden Feld eine Fehlernachricht angezeigt. Diese Informationen werden ausgelesen und stehen im component.ts als String zur Verfügung und können im Service verarbeitet werden.

4.6 History Komponente

Diese Komponente zeigt die letzten Ereignisse der Kaffeekannen an. Dabei werden jeweils die drei aktuellsten Benachrichtigungen angezeigt. Diese werden durch *** -***Tags gekennzeichnet. Darin wird über das ***[innerHTML]-***Attribute der String aus der TypeScript-Datei in die HTML-Datei geladen. Die Typescript-Datei bekommt die Benachrichtigungen aus dem JSON-File! Die Benachrichtigung besteht aus zwei Teilen dem Kopfbereich und den Körperbereich. Der Kopf die Art des

Vorgangs, also „Auffüllen“, „Abholen“ … und die Kaffeekanne, die durch Gebäude und Stockwerk deutlich gekennzeichnet ist.

4.7 Komponenten: status und consumption

Diese beiden Komponenten sind durch die großen Tabellen gekennzeichnet. In der HTML Datei ist der Standard Aufbau einer Tabelle zu finden, die wieder durch das [innerHTML]-Tag die spezifischen Tabellendaten bekommt. Die spezifischen Tabellendaten werden aus der JSON-Datenbank durch einen Service bekommen und in Typescript in das HTML Dokument übertragen. Die Berechnungen zum Beispiel für den durchschnittlichen Tagesverbrauch werden von der Datenbank übernommen und stehen bereits fertig in der Datenbank. Das Stylesheet enthält die CSS-Befehle der Tabelle und die dazugehörigen Befehle zum Responsive Design.

4.8 Angular Anwendung auf einem NGINX Server betreiben

Die Entwicklungsumgebung kann mit dem Befehl ***ng build*** in eine Auslieferung Version kompiliert werden. Dadurch entsteht ein ***dist-***Ordner, welcher im Folgenden in ***/home/pi/git/Website/Kafffeeverwaltung_NGINX*** kopiert werden sollte, um die Angular Anwendung auf dem NXINX Server betreiben zu können. Dieser Ordner ist unter ***/etc/nginx/sites-available/default*** als Root-

Verzeichnis angegeben. Danach ist der NGINX Server unter der IP: 172.27.10.180 erreichbar.

5 Chrome Plug-In

Der zentrale Punkt des Chrome Plug-Ins ist die manifest. json. Diese enthält die Plug-In Beschreibung, gibt Berechtigungen an und bindet die einzelnen Skripte. Das Aussehen des Plug-Ins wird durch die HTML-Datei ***popup.html*** bestimmt. Diese besteht aus einem einfachen Field-set, das eine Tabelle beherbergt, um die korrekte Positionierung der Elemente sicherzustellen. Der Stil wird durch die CSS-Datei ***styles-heet.css*** festgelegt. Dies bestimmt jedoch nur das Aussehen des Plug-Ins und nicht die der Optionsseite. Die HTML Seite wird über die ***po-pup.js*** Skriptdatei in ihrer Funktionalität erweitert. Dies kümmert sich nur um die HTML-Datei und wird deaktivier sobald die aktive HTML-Anwendung geschlossen wird. Danach tritt die ***background.js*** Datei in Kraft diese führt, wie der Name verrät, die Hintergrundanwendungen aus.

5.1 Popup JavaScript Datei

Diese Datei ist nur solange aktiv, wie die HTML Anwendung offen ist.
In dieser Datei werden zu Beginn die HTML-Elemente ausgelesen und Variablen zugewiesen, um diese später manipulieren zu können. Im Folgenden werden diese HTML Elemente geprüft und bei Korrektheit in den

lokalen Browserspeiche gespeichert. Dies dient dazu, damit die Hintergrundseite die HTML Daten ebenfalls auslesen kann. Knöpfen wir hier einen EventListner zugewiesen, der auf einen Knopfdruck reagiert. Der Website Button öffnet einen neuen Tab und verweist auf die NGINX Website. Der Connect Button prüft die Verbindungsdaten und sendet der Background.js den Befehl zum Verbinden. Zum Schluss wird ein Block eingeschoben der Daten vom Background.js empfängt und daraus folgend das Aussehend der HTML Seite reguliert. Dazu wird geprüft ob sich das dazugehörige Element im lokalen Browserspeicher ändert!

5.2 Background JavaScript Datei

Diese Datei läuft unabhängig von der HTML Oberfläche und behandelt die Hintergrundvorgänge. Zuerst werden Variablen deklariert. Jede Zeile mit Variablen ist für einen Anwendungsfall bestimmt. Die erste Funktion ist ***tryConnect().*** Diese holt sich die IP-Adresse und den Port und versucht über einen Websocket auf den Raspberry Pi Server zuzugreifen. Zudem stellt diese Funktion andere Funktionen, die in Zusammenhang mit der Websocket Verbindung stehen, zur Verfügung. Die ***onOpen()*** Funktion prüft ob eine Verbindung besteht und passt aufgrund dessen das Aussehen der HTML Oberfläche an. Die größte Funktion ist ***onMessage***() diese wird bei Nachrichten von Server an den Client aufgerufen. Diese Funktion decodiert die JSON Nachricht vom JSON Adapter des Servers und prüft im Folgenden auf die unterschiedlichen Anwendungsfälle.

Abfragetyp	Funktion
INCOMMING_POT_EVENT	Wird Aufgerufen sobald der RFID Scanner eine Kaffeekanne erkennt. Es wird eine Benachrichtigung für den Benutzer erstellt.
POT_READY_EVENT	Wird beim Fertigstellen des Brauvorgangs aufgerufen. Ändert das Icon, prüft mit der Switch-Anweisung auf die gewünschte Anzeigezeit, stoppt den Plug-In Timer, gibt die Statusbar wieder frei, falls diese weggeklickt wurde und ruft die Funktion replyBtnClick() zum Überprüfen des Knopfdruckes auf.
POT_UPDATE_EVENT (1)	Wird je nach Auswahl in den Optionen eine Statusbar angezeigt und das Plug-In Icon geändert.
POT_UPDATE_EVENT (2)	Wie 1. Unterscheidet sich nur in den Einstellungen. Hier wird keine Statusbar angezeigt, nur das Plug-In Icon ändert sich.
CONNECTED_EVENT	Hiermit wird das Aussehen des Plug-Ins nach Erhalt der Server Message

	geändert.
INFORM_USER_EVENT	Hier wird dem Benutzer eine Benachrichtigung angezeigt, dass die Kaffeekanne bereits abgeholt wird.
CONTROLL_COFFEE_EVENT	Hier wird der Abholer der Kaffeekanne als Benachrichtigung angezeigt.

Die Funktion ***replyBtnClick(notifyId, btnIdx)*** wird beim Betätigen eines Knopfdruckes aufgerufen. Das erste Übergabeparameter verweist auf die Benachrichtigung, die den Knopfdruck ausgelöst hat. Über das zweite Übergabeparameter wird aus einer Vielzahl an Knöpfen der passende bestimmt. Beginnend bei 0 für den ersten Knopf! Im Folgenden wird eine zweite Verbindung zum Server zum Rücksenden von Nachrichten aufgebaut. Im Zeitintervall wird geprüft, ob die Verbindung aufgebaut ist oder nicht. Bei erfolgreichem Verbindungsaufbau werden zwei unterschiedliche Nachrichten versandt. In der ersten Nachricht werden die anderen Benutzer über das Abholen der Kaffeekanne vom aktiven Benutzer informiert und bei der zweiten Nachricht werden relevante Daten für die Datenbank übertragen. Schlussendlich wird die Verbindung wieder geschlossen.

Die Funktion ***removeAllClients()*** schließt alle Verbindungen die bestehen (gespeichert in dem Array sockets) und schließt diese, damit keine redundanten Verbindungen bestehen. Dies kann nämlich zur Folge haben, dass Benachrichtigungen mehrfach angezeigt werden.

Die Funktion ***updateIcon***(timer) bekommt die aktuelle Zeit in Zeitintervallen vom Server übertragen und spielt daraufhin eine zeitabhängige Animation zur Änderung des Plug-In Icons ab. Um den Nachrichtenaustausch zwischen Client und Server so gering wie möglich zu halten, bekommt der Client nur alle 120 Sekunden ein Statusupdate. Zwischenzeitlich zählt das Plug-In selbständig mit und vergleicht anschließend seinen erhaltenen Wert mit dem des Servers. Diese Funktion wurde zur Übersichtlichkeit in die Funktion ***setIcon***(…) ausgelagert, die lediglich das passende Bild auswählt und lädt.

Die Funktion ***updateIconAndBar(timer, crtTimer)*** erfüllt grundsätzlich die identische Aufgabe wie die obige Funktion. Zusätzlich zeigt sie jedoch noch die Statusbar an. Diese Funktion kann durch das klicken auf „x“ im Benachrichtigungsfenster abgebrochen werden. Damit das Icon aber weiterhin aktualisiert werden soll muss auf die obige Anwendung anschließend gewechselt werden. Dazu wird zusätzlich eine stopp Zeit mitgezählt. Dies ist notwendig, da diese Anwendung nicht wie die obige Funktion im Sekundentakt agiert, sondern im 800 Milisekundentakt, da dies für die Statusbar relevant ist. Für die Statusanzeige wird die Benachrichtigung im Takt gelöscht und neu erstellt.

Die Funktionen ***showInteractionNotification*** und ***showNotification*** zeigen den Abholen Knopf für die Kaffeekanne eine bestimmte Zeitdauer an. Dies bedarf eine Fallunterscheidung. Aufgrund dessen die beiden unterschiedlichen Methoden. Die erste zeigt die Benachrichtigung solan-

ge an, bis der Benutzer eine Interaktion durchführt. Über die zweite Methode ist eine bestimmte Zeit einstellbar.

Die Funktion ***loadOptions***() lädt die Einstellungen aus dem Optionsmenü in die Plug-In Anwendung. Diese werden im Folgenden überprüft, berichtigt und falls nicht vorhanden werden Standartwerte eingestellt.

Die Funktion ***connectedStyle***() und ***disconnectedStyle***() ändern das Aussehen des Plug-Ins in Kooperation mit dem popup.js File je nach Aktivität des WebSockets. Diese Anwendung kann stellenweiße fehlerhaft sein, da die Verbindung des Webservers nicht immer korrekt aufgebaut wird jedoch angezeigt wird.

5.3 Optionsmenü

Der Ausgangspunkt für das Optionsmenü ist die ***options.html*** sie enthält die HTML-Struktur für ein simples Formular, mit dem man einfache Einstellungen treffen kann. Diese HTML-Elemente werden von der ***options.css*** Datei graphisch aufbereitet. Die HTML-Elemente werden von der option.js Datei entgegengenommen. Diese JavaScript-Datei speichert die Einstellungen in Allgemeinen Browser Speicher und liest diese auf Wunsch wieder aus. Zudem wird hier eine kurze Animation zur visuellen Bestätigung der Änderung angezeigt. Im Weiteren bietet das Optionsmenü noch die empfohlenen Einstellungen für das Plug-In an. Zudem

wird eine erweiterte Plug-In Beschreibung angezeigt. Die Navigation erfolgt über eine Navigationsbar.

6 Firefox Plug-In

Das Firefox Plug-In ist im Grunde gleich wie das Chrome Plug-In aufgebaut und basiert auch auf diesem. Jedoch funktionierten viele Funktionen des Chrome Plug-Ins unter Firefox nicht. Firefox bietet zwar eine eigene Add-On API, diese ist aber zum aktuellen Stand veraltet und bietet nicht dieselben Funktionalitäten wie Chrome. Auch ähnliche Funktionalitäten stehen nicht zur Verfügung. Aufgrund dessen wurde entschieden die Anwendung auf Basis von HTML und CSS3 aufzubauen, diese Lösung funktioniert auf allen gängigen Browsern. Komplexere Benachrichtigungen wurden aus dieser Version entfernt und durch Basisbenachrichtigungen ersetzt. Die Anwendung hat eine neue HTML-Oberfläche bekommen, die weitaus größer ist. Diese überarbeitete Oberfläche besteht nun aus zwei Bereichen. Der erste ist identisch zum Chrome Plug-In, der zweite ersetzt die Funktionen des Chrome Plug-Ins. Dazu gehört eine Benachrichtigungsbox, die zusätzlich zu den grundlegenden Pop-up Benachrichtigungen eine Kopie oder je nach Nachricht eine Erweiterte Benachrichtigung darstellt, die nicht nur kurz unten aufblinkt, sondern auch über einen längeren Zeitraum nachzusehen ist. Darunter befindet sich die adaptierte Statusbar aus dem Chrome Plug-In. Diese wurde mit einfachen HTML Boxen erstellt. Danach wurde über CSS3 das Design einer Statusbar verliehen. Standardmäßig steht die Statusanzeige auf 100%.

Die Prozentanzeige(Text) und der Fortschrittsbalken werden in JavaScript angepasst. Dazu werden in JavaScript die HTML-Elemente ausgelesen und über dem Styling Attribute wird die Breite der inneren Box gemäß dem Status angepasst. Der Zugriff auf dem Styling Attribute erfolgt mithilfe des Punkt-Operators über den Befehl ***prg.style.width.*** Um die Benachrichtigungen auf der HTML Oberfläche anzuzeigen, erfolgt ein Austausch zwischen den ***popup.js*** und dem ***background.js.*** Die Hintergrunddatei sendet der Vordergrunddatei Benachrichtigungen bei Events vom Server. Die Vordergrunddatei ändert aufgrund dieser Benachrichtigung die Statusbar und die Benachrichtigungsbox. Danach wird die HTML Seite durch den Befehl ***location.reload()*** aktualisiert. Dieser Befehl ist notwendig, um HTML Seite mit den neusten Nachrichten zu füttern. Achtung nach diesem Befehl werden keine anderen Befehle mehr ausgeführt. Bestimmte Nachrichten werden auf der HTML-Seite nur eine bestimmte Zeitdauer angezeigt und danach durch andere Nachrichte ersetzt. Dies erfolgt in der Regel, um wieder in den Grundzustand zurück zu kehren. Zum Beispiel wird nach der Fertigstellung der Kaffeekanne nach einer Zeitdauer die Nachricht: „Kaffeekanne momentan voll befüllt“ angezeigt. Die Änderung der HTML Nachrichten werden durch Funktionen realisiert die in Großbuchstaben geschrieben wurden. Zudem ist auffällig das die background.js im Gegensatz zum Chrome Plug-In schlanker ist. Unnötige Funktionen wurden ausgefiltert.

Bekannte Auffälligkeiten:

Browser Erweiterungen werden falsch dargestellt und funktionierten fehlerhaft, wenn man einen zweiten Monitor verwendet, welcher eine unterschiedliche Auflösung hat wie der Hauptmonitor und man unter Windows bei Einstellungen/System/Bildschirm die Größe von Text, Apps und anderen Elemente auf eine unterschiedliche Prozentzahl einstellt. Lösen lässt sich dies, wenn 100% also 1:1 Größe einstellt.