(schönes Bild hier hin? ☺ )

PowerPoint nicht vergessen!

Projektarbeit

Projekt: Wetterapp

Teamname: Team Jacob

Lehrer: Bernd Schlütter

Klasse: E3FIAE

Tanja Weiser

Dominic Holzwarth

Max Krupp

Inhaltsangabe

[1. Zustandsvergleich 3](#_Toc437422750)

[2. Projektumsetzung 4](#_Toc437422751)

[3. Lessons Learned 11](#_Toc437422752)

[4. Schritte zum Starten des Servers 11](#_Toc437422753)

# 1. Zustandsvergleich

Soll-Zustand

* Aktuelles Wetter anzeigen
* Wettervorhersage anzeigen
* Customizing des Ortes

Ist-Zustand

* Aktuelles Wetter anzeigen
* Wettervorhersage anzeigen
* Customizing des Ortes
* Customizing der angezeigten Elemente möglich
* Customizing der Temperatureinheit (C° und Fahrenheit) möglich

# 2. Projektumsetzung

Projektplanung

Zu Beginn der Projektplanung wurde ein Konto bei [www.wunderground.com](http://www.wunderground.com) erstellt um einen API Key zu erhalten. Der API Key wird benötigt um aktuelle Wetterdaten von der Internetseite zu beziehen.

Nachdem es möglich war ein Request an den Wunderground-Server zu senden wurde die Struktur der in der Response enthaltenen JSON Daten studiert.

Diese JSON Daten wurden ebenfalls als Mockdaten abgespeichert und letztendlich zur Entwicklung und auch zum Testen der App genutzt.

Weiterhin wurden Icons ausgewählt, die zur Darstellung des Wetters dienen sollen.

Zur Versionsverwaltung soll Github verwendet werden, somit war es notwendig ein Repository zu erstellen.

Abschließend wurden erste Aufgaben zur Entwicklung des UI aufgeteilt.

Entwicklungsumgebung

Als Entwicklungsumgebung wurde Eclipse mit einem Git Plugin verwendet,

Benötigt wurde außerdem eine lokale OpenUI5 Bibliothek und ein Node.js Server zum Ausführen der App.

Entwicklung der App

Die Wetter-Webapp wurde mit der kostenlosen Version des JavaScript Framework SAP UI5 entwickelt. Sie ist aufgebaut mit einer index.html Seite, die das UI5 Framework lädt. Darauf folgt eine Script, der mit Hilfe des UI5 Framework eine UI5 App in den HTML Body einfügt. Eine App kann aus einen oder mehreren Views bestehen. Dem Model-View-Controller Konzept entsprechend hat ein View einen Controller und ein oder mehrere Modelle. Ein View kann als JavaScript oder XML View umgesetzt werden. Entsprechend der aktuellen SAP UI5 Best Practice entschieden wir uns für die Verwendung von XML Views. In XML Views können verschiedene UI5 Controls definiert und ihre Attribute gesetzt werden. Die Attribute können fest definiert werden oder aus einem Model ausgelesen werden. Ändert sich das Attribut im Model so wird das Controll automatisch geupdated. Ein UI5 Control ist ein JavaScript Objekt, das mit Hilfe der Attribute DOM Strukturen generiert, diese mit CSS ausstattet und einfache Funktionalitäten zur Verfügung stellt. Ein einfaches UI5 Control ist z.B. ein Button. Wenn er angeklickt wurde ruft er eine Funktion auf, die im XML View angegeben wurde und sich im JavaScript Controller des Views befindet.

Entwicklung des Servers:

Um den Verbrauch der durch den Wunderground-Service vorgeschriebenen Request-Anzahl für das kostenlose Konto zu minimieren, wurde zusätzlich zur App selbst ein Node.js-Server entwickelt, der die Daten vom Wetterservice zwischenspeichert und auch die Einstellungen der Anwendung verwaltet.

Der Server besteht aus zwei Dateien:

Die Datei server.js ist der Einstiegspunkt der Anwendung. Hier wird der Express-Server gestartet, um auf dem Port 3000 zu horchen und die Datei service.js eingebunden und instanziiert, um die anwendungsspezifische Logik auszulagern.

Der Service selbst besitzt als Membervariablen die folgenden Einstellungen:

|  |  |
| --- | --- |
| SERVICE\_URL | An diese URL wird das HTTP-Request geschickt, um Daten zu holen. |
| DEBUG\_SERVICE\_URL | An diese URL wird das HTTP-Request geschickt, wenn der Server im Debug-Modus gestartet wird. |
| API | Dies ist der API-Key vom Wunderground-Service |

Der Server nutzt das Node.js-Package „Express“, das es erlaubt, relativ einfach einen HTTP-Webservice mit benutzerdefinierten Routen zu erstellen.

Übersicht der wichtigsten Dateien

|  |  |
| --- | --- |
| Filename | Beschreibung |
| /webapp/  index.html | Lädt das SAP UI5 Framework und den „Display“ View. |
| /webapp/view/  Display.view.xml | Er enthält die Struktur der SAP UI5 Controls. |
| /webapp/view/  Formatter.js | Er enthält Funktionen, die aufgerufen werden um die Daten zum Anzeigen im XML View vorzubereiten. |
| /webapp/controller/  Main.controller.js | Er enthält Funktionen, die bei der Interaction mit der App aufgerufen werden. Die onInit Funktion wird automatisch aufgerufen und lädt das Konfigurations- und Wettermodel. |
| /webapp/view/  SettingsPopover.fragment.xml | Es enthält eine XML Struktur der Controls, die im Konfigurations-Popover angezeigt werden. |
| /webapp/style/  style.css | Custom CSS zum Verbessern des Aussehens. |

Testen

Um die Qualität und die Funktionalität der App zu testen wurden OPA5 und QUnit Tests implementiert.

OPA5

OPA5 (One Page Acceptance Tests) ist eine API für OpenUI5 Steuerelemente. Es wird genutzt um Benutzer Interaktionen, Navigation und Data Binding in Verbindung mit OpenUI5 zu testen.

OPA5 Tests werden, wie die App, mit JavaScript entwickelt. Das bietet den Vorteil schnell und einfach auf JavaScript Funktionen zugreifen zu können. Weiterhin ist somit auch eine gute Integration mit OpenUI5 gegeben.

Ein OPA5 Test besteht aus Arrangements, Actions und Assertions. Für die Actions und die Assertions wurde jeweils eine JavaScript Datei angelegt und die benötigten Funktionen implementiert. Unter die Kategorie der Actions fallen hier beispielsweise Funktionen wie das Klicken auf einen Button, wogegen eine Assertion eine Behauptung, wie z.B.: den Text auf einem Button überprüft.

|  |  |
| --- | --- |
| **Weather Tile** | |
| Check Labels | Überprüft alle Labels und deren Werte |
| Find current weather icon and check image name | Sucht das Icon per ID und überprüft den Icon Pfad |

|  |  |
| --- | --- |
| **Gust Tile** | |
| Check Labels | Überprüft alle Labels und deren Werte |
| Find compass | Überprüft die Sichtbarkeit des Kompasses |
| **Precip / Humidity Tile** | |
| Check Labels | Überprüft alle Labels und deren Werte |
| Find humidity circle progress | Überprüft die Sichtbarkeit des Humidity Circle Progress Controls und dessen Wert |
| Find rainometer | Überprüft die Sichtbarkeit des Rainometer Controls |

|  |  |
| --- | --- |
| **Forecast 1-3** | |
| Check Labels | Überprüft alle Labels und deren Werte |
| Find weather icon and check image name | Sucht das Icon per ID und überprüft den Icon Pfad |

|  |  |
| --- | --- |
| **Custom Settings** | |
| Find customize button by id, click on it and search for dialog by id | Überprüft ob der Dialog geöffnet wird nachdem auf das Icon geklickt wurde |
| Open dialog, change town and save | Ändert in den Einstellungen die Stadt und überprüft die Änderungen nach dem Speichern |
| Open dialog, click on radiobutton Fahrenheit and save | Ändert die EInheit der Temepratur und überprüft nach dem Speichern die Änderungen auf den Tiles |
| Open dialog, click on radiobutton Celcius and save | Ändert die EInheit der Temepratur und überprüft nach dem Speichern die Änderungen auf den Tiles |
| Open dialog, change state of forecast switch and save | Ändert die SIchtbarkeit der Wettervorhersage und überprüft nach dem Speichern die Änderungen auf dem UI |
| Open dialog, change state of humidity switch and save | Ändert die Sichtbarkeit der Luftfeuchtigkeit und überprüft nach dem Speichern die Änderungen auf dem UI |
| Intervall | Ändert die Intervallzeit, speichert und überprüft die Änderungen nochmals in den Einstellungen |

Die verwendeten Testdaten sind Echtdaten die als Antwort vom Wunderground-Server auf eine Anfrage kamen. Diese wurden im Projekt in einem JSON-Modell gespeichert.

Ausgeführt werden die Tests nach dem Start des Node.js Servers über folgende URL <http://localhost:3000/test/opa/opa5.html?test=true>

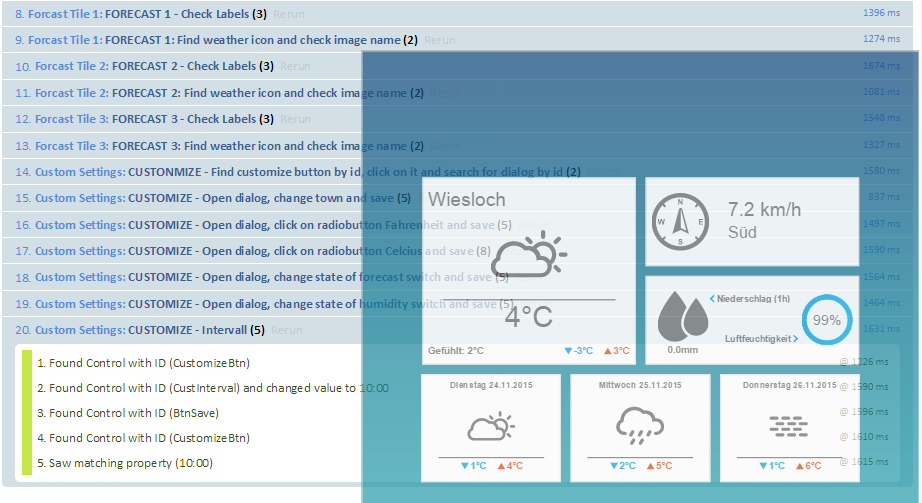


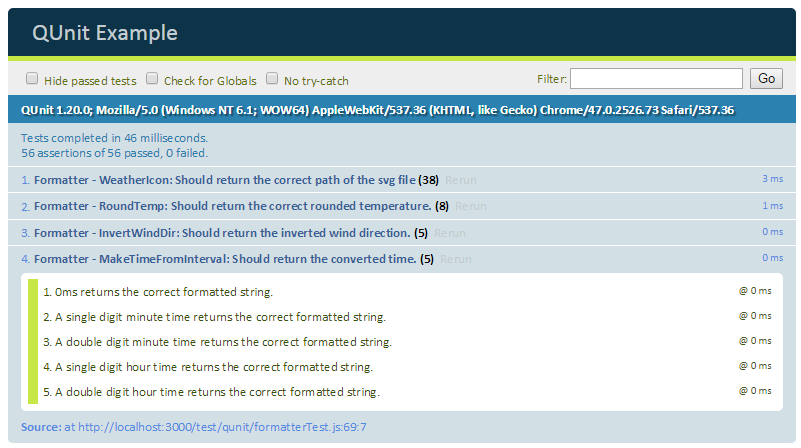
Abbildung 1 - OPA5 Tests im Browser

QUnit

Um fehlerhafte Änderungen des Codes zu erkennen wurden mehrere Unit Tests entwickelt. Diese wurden mit QUnit, einem Framework zum Testen von JS Code, umgesetzt. Es wurden die Formatter Funktionen getestet. Als Formatter Funktionen bezeichnet man Funktionen, die Daten aus dem Model für die Anzeige auf dem UI aufbereiten. Zu diesen zählen folgende Funktionen:

|  |  |
| --- | --- |
| **Funktion** | **Beschreibung** |
| WeatherIcon | Diese Funktion nimmt den Namen des Wetters entgegen z.B. *fog* und gibt den Pfad des svg Icons zurück z.B. *./icons/SVG/sw-09.svg* . Bei dieser Funktion wurden alle möglichen Wettersituationen durchgetestet |
| RoundTemp | Diese Funktion nimmt einen String, der eine Dezimalzahl enthält, entgegen und gibt eine gerundete Zahl zurück. Bei diesem Test wurden die Testdaten so gewählt, dass pro mögliche Länge (zweistellig, einstellig, einstellig negativ und zweistellig negativ) eine Test Zahl aufgerundet und eine abgerundet wird |
| InvertWindDir | Die Funktion nimmt eine Windrichtung in ° entgegen und gibt die gegenüberliegende Position zurück. Dies wird vom „Windanzeigekompass“ benötigt um aus der Windrichtung die Richtung in die der Wind weht zu ermitteln. Bei diesem Test wurde als Testdaten…   * eine Zahl die < 180° ist gewählt, da das Ergebnis normal berechnet werden kann * eine Zahl die > 180° ist gewählt, da das Ergebnis wieder bei 0° anfangen muss und nicht > 359°. * die Zahl 0°, da sie die kleinste mögliche Zahl ist. * die Zahl 180°, da das Ergebnis 0° und nicht 360° sein soll. * Die Zahl 360°, da sie die größte mögliche Zahl ist. |
| MakeTimeFromInterval | Die Funktion nimmt eine ms Zahl entgegen und rechnet diese in das Format Stunden:Minuten um. Sie wird benötigt für die Eingabe des Refresh-Intervalls im Konfigurationsbereich. Bei diesem Test wurden als Testdaten…   * die Zahl 0, da sie die kleinstmöglichste Zahl ist * eine ms Zeit, die 0 Stunden und eine einstellige Minutenzeit zurückgibt. * eine ms Zeit, die 0 Stunden und eine zweistellige Minutenzeit zurückgibt. * eine ms Zeit, die eine einstellige Stundenzeit zurückgibt.   eine ms Zeit, die eine zweistellige Stundenzeit zurückgibt. |

Die Tests können über den Browser aufgerufen werden und geben Details zu den einzelnen Tests (und eventuellen Fehlern).



Struktogramm der Methode „**checkDataChanged**“ der **Main.controller.js**



# 3. Lessons Learned

* Ausarbeitung der Projektstruktur und Konzepte hat zu lange gedauert
* Aufgabenverteilung (?)
* Anders als geplant verlaufen, bzgl. Änderungen während der Entwicklung
* Keine Designvorgabe, daher lange Vorbereitung nötig

# 4. Schritte zum Starten des Servers

1. Öffnen der Kommandozeile .
2. Mit der Kommandozeile in den Ordner der App navigieren.
3. Vor dem ersten Starten des Servers muss der Befehl „npm install“ ausgeführt werden.
4. Der Server kann jetzt mit dem Befehl „node server.js“ gestartet werden.