Ein Einblick in Web-APIs

Simon Skoczylas, Karakun

Web-APIs, als Programmierschnittstellen im Browser, bieten im Bereich der Web-Entwicklung immer mehr Möglichkeiten zur Interaktion mit dem Nutzer und der Welt außerhalb des Browsers. Hier möchte ich ein paar der bekannten und unbekannten Web-APIs vorstellen, um Entwickler auf verschiedene Web-APIs aufmerksam zu machen. Weiterhin möchte ich zeigen, dass die Entscheidung zwischen nativen, hybriden und Web-Anwendungen immer sorgfältiger getroffen werden muss.

Das Web entwickelt sich kontinuierlich weiter. Seit Jahrzehnten existiert der Wunsch, Anwendungen über das Web erreichbar und nutzbar zu machen. Auf diesem Weg verwendete man verschiedene Technologien, dazu gehörten in der Vergangenheit auch Java Applets und Flash, und in Zukunft dann WebAssembly *[1]* – wobei Letzteres als Standard für das Web entwickelt wurde. Unabhängig von diesen Technologien basiert das Web auf Standards wie HTML, CSS und einer Vielzahl durch das World Wide Web Consortium (W3C) standardisierten Web-APIs, die von modernen Browsern zur Verfügung gestellt werden. Diese Web-APIs zu kennen und korrekt einzusetzen, stellt eine neue Herausforderung dar. Kenntnisse über vorhandene Schnittstellen unterstützen und erleichtern Entscheidungen für oder gegen eine Technologie, weshalb das Wissen über die vorhandenen Web-APIs bei Entscheidungen für eine Web-Anwendung hilfreich ist. Hinzu kommt, dass dank Progressive Web Apps der Fokus von Anwendungen immer häufiger ins Web verlagert wird und das Wissen über bestehende Web-APIs zur Verbesserung der Web-Anwendung führt. Wo früher nur unzuverlässig bestimmt werden konnte, welche Funktionalität ein Browser bietet, ist es heutzutage mit Prinzipien wie der sogenannten „Feature Detection" problemlos möglich, die neuesten Web-APIs zu verwenden, ohne bestehen-

de Funktionalitäten einer Web-Anwendung in Gefahr zu bringen. Ganz im Sinne von Progressive Enhancement kann auf diese Weise dem Anwender dort, wo die Möglichkeit besteht, nach und nach ein Mehrwert geboten werden. Dank der Dokumentation seitens der Browser-Hersteller und dank der allgemeinen Dokumentation der Mozilla Foundation *[2]* sind viele Web-APIs verständlich beschrieben und müssen nicht erst mühevoll erarbeitet werden. Falls innerhalb der Dokumentation der Mozilla Foundation noch Lücken vorhanden sind oder Beiträge nicht in die benötigte Sprache übersetzt sind, ist jeder dazu aufgerufen, an der Dokumentation mitzuarbeiten. Des Weiteren können Entwickler über die Seite *http://www.caniuse.com* feststellen, in welchem Browser ein bestimmtes API zur Verfügung steht.

Feature Detection

Um eine bestimmte Funktionalität zu verwenden, muss vorher sichergestellt werden, dass diese im Browser des Anwenders vorhanden und einsatzbereit ist. Ohne Prüfung kann es innerhalb der Anwendung zu Fehlern kommen, wenn ein Nutzer mit einem Browser ohne diese geforderte Funktionalität auf die Web-Anwendung zugreift. In der Vergangenheit wurde bei dieser Prüfung oft auf den User-Agent-String *(siehe Listing 1)* zurückgegriffen und dann im Browser, oder sogar schon auf dem Server, eine Entscheidung über die vorhandenen Funktionalitäten getroffen.

Auf diese Art und Weise wurde versucht, den Browser, die Version und das Betriebssystem zu ermitteln. Verständlicherweise ist eine Prüfung über den User-Agent-String alles andere als zuverlässig, da dieser auf einen beliebigen Wert gesetzt werden kann und es in der Vergangenheit häufiger zu Missverständnissen kam. Hatte zum Beispiel der Internet Explorer bis Version 10 die Zeichenkette „MSIE" innerhalb des User-Agent-String verwendet, änderte sich dies mit Version 11. Dies führte innerhalb verschiedener Web-Anwendungen zu Problemen, da sie auf spezielle Funktionalitäten des Internet Explorers angewiesen waren und diesen nicht mehr erkannten.

In aktuellen Web-Anwendungen verzichtet man deshalb auf die Verwendung des User-Agent-String und prüft mittels Feature Detection, ob eine Funktionalität vorhanden ist *[3]*. Im Grunde basiert die Feature Detection darauf, im Browser zu prüfen, ob ein bestimmtes Objekt oder eine bestimmte Funktion vorhanden sind *(siehe Listing 2)*. Für erweiterte Prüfungen zu Funktionalitäten kann auf die JavaScript-Bibliothek Modernizr *[4]* zurückgegriffen werden.

Page Visibility API

Mit dem Page Visibility API *[5]* kann bestimmt werden, ob die aktuelle Web-Anwendung aktuell für den Nutzer sichtbar ist und der Fokus auf ihr liegt. Dies ist vor allem für das sogenannte „Tabbed Browsing" relevant, da die eigene Web-Anwendung sich in einem Tab im Hintergrund befinden kann. Währenddessen können ressourcenintensive Aufgaben, wie zum Beispiel das Polling nach Daten, gestoppt werden. Um den aktuellen Status zu ermitteln, wird

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/73.0.3683.86 Safari/537.36
```

Listing 1: Beispiel für einen User-Agent-String

```
if (navigator.bluetooth) {
// Funktionalität kann verwendet werden
} else {
// Alternativen verwenden
}
```

Listing 2: Feature Detection für das Web Bluetooth API

das `Document`-Objekt um zwei Attribute und ein Event erweitert. Das Attribut `hidden` repräsentiert einen booleschen Wert, der `true` zurückgibt, wenn die Anwendung nicht sichtbar, und `false`, wenn die Anwendung sichtbar ist. Das Attribut `visibilityState` liefert eine Zeichenkette abhängig vom aktuellen Status der Anwendung. Ist der Rückgabewert `visible`, dann ist die `Web`-Anwendung aktuell sichtbar, ist der Rückgabewert `hidden`, dann ist die Anwendung aktuell für den Nutzer nicht sichtbar. Da `document.hidden` aus historischen Gründen in der Spezifikation beibehalten wurde, sollten Entwickler vor allem auf `document.visibilityState` zurückgreifen, um den Status der Sichtbarkeit zu bestimmen. Das Event `visibilitychange` wird vom Browser bei Änderungen an der Sichtbarkeit gesendet *(siehe Listing 3)*.

NavigatorOnLine (oder Browser state)

Die HTML5-Spezifikation definiert über die `NavigatorOnline`-Schnittstelle *[6]* das boolesche Attribut `onLine` am `Navigator`-Objekt. Dieses Attribut dient zur Bestimmung des aktuellen Verbindungsstatus des Browsers. In anderen Worten, ob der Browser gerade online `(true)` oder offline `(false)` ist. Zusätzlich dazu werden die Events `online` und `offline` bereitgestellt, um über eine Änderung am Verbindungsstatus zu informieren. *Listing 4* zeigt, wie das `onLine`-Attribut ausgelesen und die Events verwendet werden können.

Notification API

Dank des Notification API *[7]* kann aus Web-Anwendungen heraus eine Benachrichtigung über das Benachrichtigungssystem des Betriebssystems an den Nutzer gesendet werden. Eine solche Benachrichtigung kann damit auch angezeigt werden, wenn die Web-Anwendung aktuell nicht im Vordergrund ist und der Nutzer zum Beispiel einen anderen Tab geöffnet oder eine andere Anwendung im Fokus hat. Bevor der Nutzer eine Benachrichtigung erhalten kann, muss er dies zuvor erlauben.

Für alle Aufgaben des Notification API wird das `Notification`-Objekt definiert. Dieses Objekt enthält im Attribut `permission` den aktuellen Status der Erlaubnis. Mögliche Werte hierfür sind:

```
if (document.visibilityState) {
  window.addEventListener('visibilitychange', (event) => console.log(document.visibilityState));
}
```

Listing 3: Beispiel für das Page Visibility API

```
if (window.navigator) {
  console.log('Status', navigator.onLine);
  window.addEventListener('online', (event) => console.log('Wir sind online'));
window.addEventListener('offline', (event) => console.log('Wir sind offline'));
}
```

Listing 4: Beispiel für NavigatorOnLine

```
if (window.Notification) {
  window.Notification.requestPermission((permission) => {
     if (permission === "granted") {
        const notification = new Notification('Hallo', {
           body: 'Welt',
           icon: 'images/world.jpg',
        });
        notification.addEventListener('close', (event) => console.log('Nachricht geschlossen'));
     }
  });
}
```

Listing 5: Beispiel für das Notification API

```
if (navigator.share) {
    navigator.share({
        title: 'Java aktuell',
        text: 'Einblick in Web-APIs',
        url: 'https://www.ijug.eu/de/java-aktuell/',
    })
      .then(() => console.log('Erfolgreich geteilt'))
      .catch((error) => console.log('Fehler', error));
  }
}
```

Listing 6: Beispiel für das Web Share API

- `denied`, wenn der Nutzer keine Erlaubnis erteilt hat
- `granted`, wenn der Nutzer die Erlaubnis erteilt hat
- `default`, wenn der Nutzer noch keine Erlaubnis erteilt hat

Um eine Erlaubnis für Benachrichtigungen beim Nutzer zu erfragen, muss die Funktion `Notification.requestPermission()` aufgerufen werden, die ein `Promise` zurückgibt. Dieses enthält den oben erwähnten Status der Erlaubnis, wenn sie erfüllt wird. Ist die Erlaubnis einmal erteilt, kann über das Erstellen einer neuen Instanz des `Notification`-Objekts eine Benachrichtigung erstellt und gleichzeitig auch angezeigt werden. Des Weiteren kann eine Benachrichtigung über ein `options`-Attribut angepasst werden. Auf diese Weise können zum Beispiel ein Text und ein Bild für die Benachrichtigung gewählt werden, wie im *Listing 5* dargestellt wird.

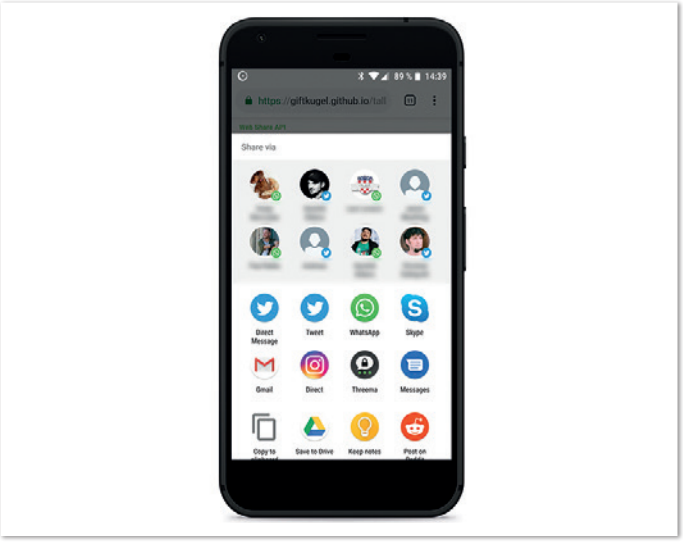

Abbildung 1: Teilen mit dem Web Share API auf Android (Quelle: Simon Skoczylas)

Web Share API

Mit dem Web Share API *[8]* ist es möglich, den nativen Mechanismus zum Teilen von Inhalten zu verwenden *(siehe Abbildung 1)*. Dazu wird das `Navigator`-Objekt im Browser um die `share`-Funktion erweitert. Als Parameter erwartet die `share`-Funktion ein Objekt mit den Attributen `url`, `text` und `title`. Der Rückgabewert ist ein `Promise`, der erfüllt wird, wenn der Nutzer die Operation zum Teilen abschließt und andernfalls zurückgewiesen wird. *Listing 6* zeigt den Aufruf der `share`-Funktion, wobei der Aufruf im Normalfall an eine Nutzeraktion geknüpft werden sollte, zum Beispiel den Klick auf einen Link oder Button.

Web Speech API

Mithilfe des Web Speech API *[9]* kann der Browser Texte in Sprache ausgeben (Text-to-Speech) und sogar Sprache verstehen (Speech Recognition). Zur Ausgabe von Sprache wird eine Instanz von `SpeechSynthesisUtterance` benötigt. Diese kann mit einem Text, der gesprochen werden soll, instanziiert werden. Weitere Attribute können dann optional direkt für die Instanz gesetzt werden. Dazu

```
if (window.speechSynthesis) {
    const utterance = new SpeechSynthesisUtterance('Hallo Welt!');
    utterance.rate = 0.8;
    utterance.lang = 'de';
    utterance.voice = speechSynthesis.getVoices()[0];
    speechSynthesis.speak(utterance);
}
```

Listing 7: Beispiel für das Web Speech API

```
if (navigator.vibrate) {
    navigator.vibrate([200, 50, 200]);
}
```

Listing 8: Beispiel für das Vibration API

Background Sync	Navigation Timing	Vibration
Battery Status	Network Information	Web Audio
Canvas	Notifications	Web Bluetooth
Clipboard	Page Visibility	Web Crypto
Credential Management	Payment Request	WebGL
Fetch	Presentation	WebRTC
Fullscreen	Push	Web Share
Gamepad	Selection	WebSocket
Generic Sensor	Service Worker	Web Speech
Geolocation	Server-sent events	Web Storage
IndexedDB	Shape Detection	WebUSB
Navigator OnLine	Streams	WebVR

Tabelle 1: Übersicht über die Web-APIs

zählen `lang` für die Sprache, `voice` für die Stimme sowie die Attribute `volume`, `rate` und `pitch`, um die Lautstärke, Geschwindigkeit und Tonhöhe zu verändern. Die Attribute `text` und `lang` sind Zeichenketten, `volume`, `rate` und `pitch` Gleitkommazahlen und `voice` eine Instanz von `SpeechSynthesisVoice`, die vom Browser ermittelt werden kann. Um die vorhandenen Stimmen zu erhalten, kann die Methode `getVoices` des `speechSynthesis`-Objekts verwendet werden. Wurden die gewünschten Werte gesetzt, kann die `SpeechSynthesisUtterance` an die `speak`-Funktion des `speechSynthesis`-Objekts übergeben werden, wie in *Listing 7* gezeigt wird.

Vibration API

Um auf die Vibrationshardware des Gerätes zuzugreifen, wurde das Vibration API *[10]* eingeführt. Diese Schnittstelle ermöglicht es dem Nutzer, durch Vibration ein physikalisches Feedback zu geben, zum Beispiel bei einer fehlerhaften Eingabe. Falls das Gerät keine Möglichkeit zur Vibration hat, passiert bei der Verwendung des Vibration API nichts. Für das Vibration API definiert man am `Navigator`-Objekt die Funktion `vibrate`. Diese Funktion erwartet als Parameter einen Wert für die Dauer der Vibration in Millisekunden oder ein Array mit der Dauer für eine Vibration und der Dauer für eine Pause im Wechsel. Zum Beispiel 200, 50, 200 für eine Abfolge von 200ms Vibration, 50ms Pause und 200ms Vibration, wie das *Listing 8* zeigt.

Überblick der vorhandenen Web-APIs

Neben den hier erwähnten Web-APIs bietet das Web noch eine Vielzahl weiterer Schnittstellen, die darauf warten, genutzt zu werden. *Tabelle 1* soll einen Überblick über die Möglichkeiten geben und vielleicht dazu animieren, sich das eine oder andere Web-API noch mal im Detail anzuschauen. In den nächsten Jahren wird die Liste mit Sicherheit noch etwas länger, es bleibt spannend.

Quellen

[1] *https://webassembly.org/*
[2] *https://developer.mozilla.org/en-US/docs/Web/API*
[3] *https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/Feature_detection*
[4] *https://modernizr.com/*
[5] *https://www.w3.org/TR/page-visibility-2/*
[6] *https://www.w3.org/TR/html5/browsers.html#browser-state*
[7] *https://www.w3.org/TR/notifications/*
[8] *https://wicg.github.io/web-share/*
[9] *https://w3c.github.io/speech-api/*
[10] *https://w3c.github.io/vibration/*

Simon Skoczylas
simon@skoczylas.net

Simon Skoczylas ist Senior Software Engineer bei der Karakun AG am Standort Dortmund. Er ist leidenschaftlicher Softwareentwickler und Softwarearchitekt. Seine Erfahrungen sammelte er vor allem in der Entwicklung von Software auf Basis von Java und JavaScript. Sein Schwerpunkt liegt auf der Erstellung von Web-Anwendungen.