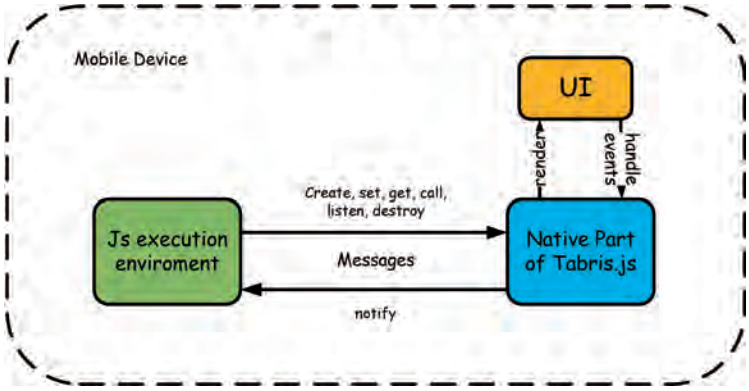
APPS FÜR ANDROID UND IOS MIT TABRIS.JS ENTWICKELN

Schnell und gut

Mit Tabris.js lassen sich App für iOS und Android schnell erstellen.

Sie alle eint das Ziel, es einfacher und damit schneller zu machen als mit Android Studio und Xcode. Das Framework Tabris.js hat Version 3.5 erreicht und bietet einige Vorteile. Wir haben es uns genauer angesehen.

Möchte man Apps für die mobilen Systeme erstellen, dann kommt man unweigerlich an einen Punkt, wo man mehrere Entscheidungen treffen muss. Aus technischer Perspektive gilt es zwischen nativen Apps, Web Apps und hybriden Apps zu unterscheiden. Während man mit nativen Apps alle Features der jeweiligen Plattform erreichen kann, sind die Einschränkungen bei den Web Apps am größten. Native Apps werden üblicherweise über die App Stores vertrieben, während dieser Weg bei Web Apps nicht möglich ist. Performance, UI-Gestaltung und die Verwendung von speziellen Sensoren, das alles geht mit nativen Apps stets besser.



Die Architektur des Renderings bei Tabris.js (Bild 1)

Diesen Vorteilen gegenüber steht der nahezu doppelte Aufwand sowohl für Android als auch für iOS zu entwickeln. Zu unterschiedlich sind die Systeme, als dass man nennenswerte Bestandteile an Code zwischen den Plattformen teilen könnte. Gewissermaßen mittendrin sind die hybriden Apps angesiedelt. Sie verwenden Web Technologien in einem Sandkasten und ermöglichen die App Store Distribution. Hier konkurrieren viele Frameworks und Entwicklungsansätze. Dennoch ist die Entwicklung mit HTML, CSS und JavaScript oft mit viel Aufwand verbunden.

Ein Lösungsansatz kann plattform- oder geräteübergreifende Programmierung sein. Dabei wird aus einer Quellcodebasis die App für beide Zielsysteme generiert. Idealerweise kann man dabei während der Entwicklung von den Plattformbesonderheiten weitgehend abstrahieren. Bekannte Ansätze sind Xamarin (Microsoft), Flutter (Google), React Nativ (Facebook) und Native Script (Telerik by Progress). Etwas weniger bekannt ist das JavaScript basierte Framework Tabris.js. Folgende Vorteile soll dieser Ansatz bieten:

- Plattformübergreifend: App für iOS und Android aus einer gemeinsamen Quellcodebasis.
- Online-Build: Man kann die App-Packages direkt aus dem Quellcode und aus GitHub online generieren.
- Reduzierte Hardwarevoraussetzungen: Im Zusammenhang mit dem möglichen Online-Build, wird während der Entwicklungsphase kein Mac-PC der Entwicklungsumgebung Xcode benötigt.
- Deklarative Erstellung des User Interfaces (UI): Das UI wird mittels eines XML-ähnlichen Dialektes direkt im Quellcode deklariert.
- Cross Plattform API: Alle Widgets und Dialoge sind für eine Verwendung auf Android und iOS ausgelegt. Lediglich wenige Eigenschaften beziehen sich nur eine Plattform.
- Single Language: Die Codierung erfolgt komplett in JavaScript oder einfacher in TypeScript, das heißt, es ist kein HTML und kein CSS erforderlich.
- Erprobte Hardwareschnittstellen: Nutzung der Plug-ins von Cordova für die Ansteuerung der Hardware.
- Schneller Entwicklungszyklus: Mittels einer Developer App auf dem Smartphone ist während der Entwicklung der eigentlichen App eine unmittelbare Vorschau möglich.

Wir wollen uns Tabris.js nun genauer ansehen. Auch der Entwicklungszyklus verdient Aufmerksamkeit, denn wir finden hier durchaus ein etwas anderes Vorgehen.

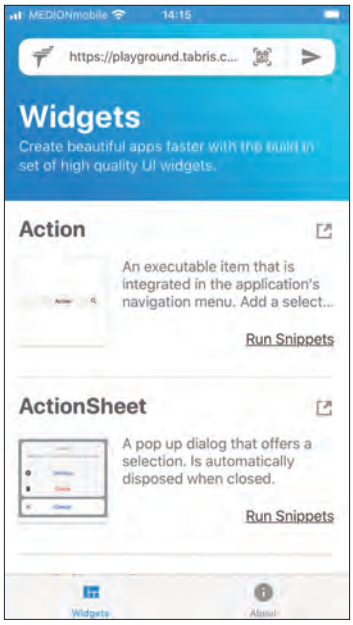
Ein Framework für mobile Apps

Mit dem Begriff Framework wird heute sehr schnell hantiert. Jede größere Bibliothek bezeichnet sich heute gern als Framework, so dass man einen kleinen Augenblick darauf verwenden kann, was ein solches Framework eigentlich ausmacht. Um der Bezeichnung gerecht zu werden, muss es einen Rahmen im Sinne einer umfassenden Architektur bieten, an welcher man die zu erstellende App ausrichtet. Tabris.js bietet einen solchen Entwicklungsrahmen für mobile Apps der Systeme Android und iOS. Dazu gehören: eine Architektur für den Systemaufbau der App, eine komponentenbasierte Möglichkeit das User Interface zu deklarieren, die Anbindung der Hardware der mobilen Devices über Plug-ins und Unterstützung durch Entwicklungswerkzeuge und Tools.

Es lassen sich native Apps für iOS und Android aus einer gemeinsamen Quellcodebasis entwickeln. Programmiert wird vollständig in JavaScript, bevorzugt jedoch in Ty-

peScript. Interessant ist, dass keine weiteren Programmiersprachen und auch kein HTML und CSS benötigt werden. Tabris.js verwendet kein WebView Container zum Rendern der Benutzeroberfläche. Stattdessen werden native Widgets auf der jeweiligen mobilen Plattform aus JavaScript eingesetzt (Bild 1). Das sorgt dafür, dass die Oberflächen den Vorgaben von Android beziehungsweise iOS entsprechen.

Die Tool-Chain bestimmt maßgeblich, wie effizient man mit einem Vorgehensmodell arbeiten kann. Die Entwicklung mit Tabris.js basiert auf einem direkten Deployment der App auf das Endgerät, zum Beispiel ein iPhone, iPad oder Android-Smartphone, bereits während der Entwicklung. Im Zentrum des Entwicklungsvorgangs steht die so genannte Developer App. Man benötigt also direkt ein mobiles Gerät. Die Developer App lädt man kostenfrei aus dem Google Play- bzw. Apple App-Store (Bild 2).



Die Developer App für Android und iOS bildet den Mittelpunkt der Entwicklung (Bild 2)

Neben der Demonstration zur Funktion und Leistungsfähigkeit von Widgets (UI-Elementen), dient die App in erster Linie dazu, dass man den eigenen Quellcode bezüglich der Lauffähigkeit verifizieren kann. Dazu wird der Developer App die URL zum Hosting der eigenen App bekannt gemacht (zum Beispiel über einen Scann des QR-Codes) und die eigene App wird dann innerhalb der Developer App ausgeführt. In Kombination mit dem Online-Build ergibt sich ein innovativer Entwicklungszyklus.

Das geht am einfachsten über den so genannten Playground der Webseite von Tabris.js (Bild 3). Ein Live Code-Editor ermöglicht die Auswahl, Änderung und die Erfassung von JavaScript-Code. Alternativ kann TypeScript eingesetzt werden. Über diesen Online Playground können Experimente bei der Erstellung des User Interface vorgenommen werden. Beispielsweise kann man das Hello World-Beispiel (*hello.jsx*) in TypeScript probieren.

Den Ausgabetext kann man nach seinen Vorstellungen anpassen. Mit dem Smartphone und aktiver Developer App scannt man den Barcode, welcher auf der Webseite angezeigt wird. Danach wird die App, welche im Hintergrund auf dem Server erstellt wird, sofort innerhalb der Developer App gestartet (Bild 4).

Mit dem Quellcode auf der Webseite können Sie nach Belieben experimentieren. Im Hintergrund wird lauffähiger Code generiert, welcher direkt in der Developer App ausgeführt werden kann. Für die produktive App-Entwicklung braucht es ein Setup des Entwicklungsrechners.

Entwicklermaschine einrichten

Für die App-Entwicklung können Sie einen beliebigen Editor beziehungsweise eine Entwicklungsumgebung Ihrer Wahl einsetzen. Sehr gut funktioniert es mit Visual Studio Code. Installieren Sie dazu zusätzlich die folgende Software:

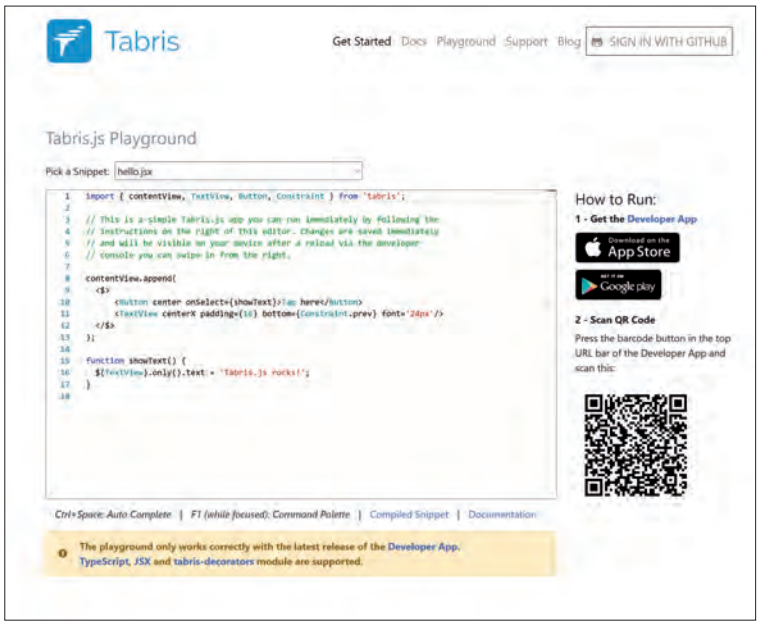
```
Node.js, inklusive NMP von: https://nodejs.org/en/
Tabris CLI mit: npm install -g tabris-cli
```

Das mobile Gerät mit der aktiven Developer App und der Entwicklungsrechner müssen im gleichen lokalen Netzwerk sein. Eine erste App erstellen wir auf der Kommandozeile. Erstellen Sie dazu ein neues Arbeitsverzeichnis. Legen Sie ein neues Projekt mit dem Befehl *tabris init* an. Beantworten Sie Fragen nach dem Projektnamen, Version und wählen Sie als Template zum Beispiel *Hello World (TypeScript/JSX)*. Zur Auswahl stehen die folgenden Templates: *Model-View-ViewModel (TypeScript/JSX)*, *Model-View-Presenter (TypeScript/JSX)*, *Hello World (TypeScript/JSX)*, *Hello World (JavaScript/JSX)* und *Hello World (JavaScript)*.

Danach erzeugt Tabris.js die grundlegende Projektstruktur und lädt alle notwendigen Bibliotheken. Insbesondere der Ordner *node_modules* enthält alle notwendigen Packages. Von besonderem Interesse sind die folgenden Dateien:

- *package.json*: In dieser Datei wird das Projekt (App) deklariert und die Abhängigkeiten von Bibliotheken notiert. Das Format ist JSON.
- *config.xml*: Für den Build-Prozess ist eine Datei config.xml notwendig. Diese liegt im Unterordner */cordova*.

Grundsätzlich müssen Sie an den Dateien *package.json* und *config.xml* zunächst keine Anpassun- ▶



Erste Versuche mit Tabris.js starten im Online Playground von Tabris.js (Bild 3)

gen vornehmen. Öffnen Sie Ihr Projekt und passen Sie nach Bedarf den TypeScript-Code an. Wechseln Sie dann wieder auf die Kommandozeile in das Projektverzeichnis der App. Die App können Sie nun auf dem Entwicklungsrechner starten. Das geschieht mit dem Befehl *tabris serve*. Dieser startet den HTTP-Server von Nodes.js. Auf der Kommandozeile gibt der Server aus, unter welcher Adresse der App-Code zu erreichen ist. Wechseln Sie zur Developer-App auf Ihrem mobilen Gerät und verbinden Sie sich zum Server unter der angegebenen Adresse. Hat alles funktioniert, wird die App direkt innerhalb der Developer-App gestartet und ausgeführt.

Während der Entwicklung wird die App in der Developer Apps ausgeführt (Bild 4)

Die App wird in der Developer-App ausgeführt (Bild 5)

In der aktuellen Version von Tabris.js 3.5 wurden gegenüber den Vorversionen eine Reihe von Verbesserungen vorgenommen. Unter anderem wurde die Leistungsfähigkeit zum Erstellen des UI verbessert. Das UI besteht aus nativen Widgets, die durch JavaScript-Objekte dargestellt werden. Es gibt verschiedene Arten von Widgets wie eine Button-, ein TextView- oder ein ScrollView-Komponente.

Jeder Widget-Typ ist ein Subtyp des Objektes *Widget*. Das Basisobjekt *Widget* stellt allgemeine Methoden zum Abrufen und Festlegen von Eigenschaften, zum Benachrichtigen von Ereignissen und zum Zuordnen an ein übergeordnetes Element bereit. Dazu ein einfaches Beispiel:

```
let button = new Button({
  left: 10,
  top: 10,
  text: 'OK'
});
```

Damit erstellt man einen Button mit der Aufschrift *OK* und den angegebenen Abständen vom Rand, das heißt, von oben und von links jeweils 10 Pixel. Für die Darstellung wird ein globales *Tabris Object* verwendet. Dieses beinhaltet die Elemente: *statusBar*, *navigationBar*, *contentView* und *drawer*. Über die Methode *append* können wir Kind-Elemente an die Elemente *contentView* und *drawer*, hinzufügen:

```
import { contentView, TextView, Button} from 'tabris';
new Button({
  left: 10,
  top:10,
  text: Ein Button})
  .appendTo(contentView);
```

Um Klassen und Objekte nutzen zu können, müssen Sie diese zuvor importieren. Probieren Sie es im Playground. Statt das User Interface über TypeScript zu erstellen, können Sie dieses auch in JSX (Javascript XML) deklarieren. Das bedeutet, statt obigen Listings können wir den Button auch wie folgt deklarieren:

```
import { contentView, TextView, Button, Constraint }
from 'tabris';
contentView.append(
 <$>
 <Button top={10} left={10} text="Button mit JSX def."/>
 </$>
);
```

Diese Schreibweise ist kürzer und daher in der Regel bevorzugt. Tabris.js bietet eine ganze Palette von Widgets, zum Beispiel: *Button*, *Canvas*, *CheckBox*, *Picker*, *StatusBar*, *Tab* etc. In der aktuellen Version sind es etwa 30 unterschiedliche Widgets und zusätzlich 5 Dialog- /Pop-up Typen für die Gestaltung des UI.Tabris.js verwendet geräteunabhängige Pixel. Bei allen Widgets kann festgelegt werden, wie diese im übergeordneten Element angeordnet werden sollen (Layout):

```
contentView.append(
 <$>
 <Button id="Button1" centerX={0} top={10}
 text="Button 1"/>
 <Button id="Button2" centerX={0} top='prev() 20'
 text="Button 2"/>
  </$>
);
```

Wir definieren zwei Buttons. Für den ersten Button erfolgt die Ausrichtung horizontal zentriert (*centerX={0}*) und mit einem Abstand vom oberen Rand in der Größe von 10 Pixel (*top={10}*). Der zweite Button wird erneut horizontal zentriert (*centerX={0}*) und jetzt mit einem Abstand von 20 Pixel vom vorherigen Element (*Button1*) in vertikaler Richtung platziert. Das Ergebnis besteht aus zwei untereinander angeordneten Buttons, welche beide horizontal zentriert sind (Bild 5).

Apache Cordova

Apache Cordova ist ein Open Source-Framework, um unter anderem Apps für mobile Geräte zu entwickeln. Es verwendet die Standard-Technologien HTML5, CSS3 und JavaScript. Über das Framework bekommt die App Zugriff auf die Systemfunktionen, wie die Kamera oder das Adressbuch. Das geschieht mit Hilfe von Plugins. Ausgewählte Plugins verwendet Tabris.js, um App-spezifische Funktionen zu realisieren. Informationen zu Apache Cordova finden Sie unter https://cordova.apache.org/.

Im Programmcode müssen Sie regelmäßig ein Widget zugreifen. Sie könnten jedem Widget eine Variable zuordnen. Der Nachteil: Bei einer großen Anzahl von Widgets würde das sehr unübersichtlich werden. Tabris.js bietet daher die Methoden, ein Widget alternativ nach Typ, ID oder Klassenattributen (im Stil von CSS) auszuwählen. Als Framework für mobile Apps bietet sich eine Reihe von Optionen zur Unterstützung der Touch-Interaktion, in Form von spezialisierten Gesture-Events (Gesten).

Sensoren in den Griff bekommen

Das Tabris.js-API ist primär eine User Interface-Bibliothek, welche jedoch von Version zu Version mit App spezifischen Features, zum Beispiel Permission Management, Worker, Social Share API etc., angereichert wurde. Um weitere App-typische Funktionen bereitzustellen, wie zum Beispiel den Zugriff auf die Hardware der mobilen Geräte kann Tabris.js mit Apache Cordova-Plugins erweitert werden.

Die JavaScript-API-Dokumentation der Cordova-Plugins ist mit einer geringfügigen Ausnahme auch in Tabris.js gültig: Vor dem Zugriff auf das Plugin-API müssen Sie jedoch das *devicready*-Ereignis nicht abhören. Alle Plugins sind nutzbar, wenn das *main*-Modul der Anwendung geladen wird. Folgende Plugins sind direkt in der Tabris-App verfügbar: *Badge, Camera, Device Motion, Network Information, Toast, Barcode Scanner, Google Analytics* und *Google Play Services*. Bei der Nutzung von Plugins müssen wir beachten: Tabris.js generiert eine native Benutzeroberfläche und verwendet kein HTML5. Für viele Cordova-Plugins spielt das keine Rolle. Greift das Cordova-Plugin jedoch auf den DOM einer in HTML5 definierten Oberfläche zu, kann es nicht mit Tabris.js verwendet werden.

Tabris.js bietet weitere Funktionen, die für Entwickler von Interesse sind.

Um die App außerhalb der Developer App zu starten ist das Build durchzuführen. Es werden dabei die App Packages für das Deployment über die Stores erstellt. Tabris.js verwendet für das Build und das Erstellen der App Packages Apache Cordova. Wir unterscheiden ein lokales und ein Online Build. Das Online Build ist ein Service unter https://tabris.com/ und erfolgt direkt aus einem GitHub-Repository. Dabei können die App Packages (auch iOS) ohne eigene Hardwareressourcen, erstellt werden. In der kostenfreien Variante von Tabris.js können Sie beliebig viele Builds aus öffentlichen GitHub-Repositories durchführen beziehungsweise ein privates GitHub-Repository verarbeiten. Nach erfolgreichen Build stehen die App Packages für die Zielplattform zum Download zur Verfügung.

Für die Fehlersuche stehen die üblichen Methoden für die Arbeit mit der Konsole, das heißt *console.log*, *info*, *warn* und *error* zur Auswahl. Bei komplexeren Anwendungen ist es jedoch in der Regel vorzuziehen, einen vollwertigen Debugger zu verwenden, mit dem Sie die Skriptausführung anhalten und Variablen prüfen können. Das Debugging ist an die Zielplattform gebunden. Für Android werden die Chrom Developer Tools verwendet. Für das Mobile Device, das heißt, das Smartphone beziehungsweise das Tablet, ist in den Entwickleroptionen das USB-Debugging zu aktivieren. Das eigentliche Debugging funktioniert mit Hilfe der Entwicklertools von Chrome. Diese erreichen Sie in Ihren Browser unter chrome://inspect/#devices. Die App muss auf dem mobilen Gerät gestartet werden (Debug-Modus) und über die Developer-Konsole von Chrome kann dann der gewünschte Breakpoint gesetzt werden.

Links zum Thema

- Projektseite von Tabris.js
  https://tabris.com/
- Xamarin
  https://visualstudio.microsoft.com/de/xamarin/
- Flutter
  https://flutter.dev/
- React Native
  https://reactnative.dev/
- Native Script
  https://nativescript.org

Einschränkungen

Ein plattformübergreifendes Vorgehen muss immer den gemeinsamen Nenner der unterstützten Systeme suchen. Das betrifft zum Beispiel die Umsetzung der Benutzerschnittstelle. Bei der Hardwareunterstützung setzt man auf die Nutzung der Cordova Plugins. Damit ein Sensor aus einer Tabris.js App verwendet werden kann, ist man darauf angewiesen, dass es ein kompatibles Plugin gibt. Für langfristige Projekte auf der Basis von Tabris.js muss man darauf vertrauen, dass der Hersteller das Framework weiterhin aktiv und in kurzen Abständen pflegt. Eine fortlaufende Anpassung an neue Android- und iOS-Versionen ist notwendig.

Fazit

Tabris.js ist eine interessante Alternative im nunmehr doch recht unübersichtlichen Feld der Ansätze zur Programmierung von mobilen Apps. Das Framework liegt nunmehr auch bereits in der Version 3.5 vor und beweist eine gewisse Kontinuität. Hervorzuheben sind der effiziente Entwicklungszyklus, die Möglichkeit des direkten Online Builds aus einem GitHub-Repository und die Nutzung von Cordova-Plugins für die Hardwareinteraktion. ■

Elena Bochkor
beschäftigt sich mit Fragen rund um die Gestaltung von Webseiten und Benutzeroberflächen für Apps der mobilen Plattformen. Weitere Informationen zu diesen und anderen Themen der IT finden Sie unter
http://larinet.com