Native Apps versus Web-Apps und mobilen Web-Applikationen – Das mobile Umfeld von Cordova

2

Inhaltsverzeichnis

2.1 Was behandeln wir im einleitenden Kapitel?

In diesem Kapitel stellen wir Web-Apps und native Apps gegenüber und beleuchten die Vor- und Nachteile beider Lösungen. Das gehen wir auch praktisch an und erstellen beispielhaft native Apps, die ja über die Cordova-Wrapper die Basis von Cordova-Apps bilden. Damit soll Ihnen eine unbedingt notwendige Grundlage für die spätere Programmierung mit Cordova gegeben werden. Ebenso vergleichen wir in dem Kapitel Web-Apps und mobile Web-Applikationen bzw. Webseiten. Darüber hinaus werden wir die Bedeutung von einigen Begriffen erläutern, die für die relevanten Themen in dem Buch von Belang sind und in der Folge vorausgesetzt werden.

R. Steyer, *Cordova*, DOI 10.1007/978-3-658-16724-0_2

2.2 Mobile Web-Applikationen und mobile Webseiten versus Web-Apps

Wir wollen uns zuerst um die Frage kümmern, was nun mobile Web-Applikationen bzw. Webseiten von Web-Apps unterscheidet? Denn es gibt ja auch die Möglichkeit für mobile Endgeräte einfach optimierten Inhalt zur Verfügung zu stellen, der nicht als App auf das Endgerät gelangt.

▶ Unter dem Begriff einer **Web-App** möchte ich im Folgenden einfach eine App verstehen, die im Wesentlichen mittels Web-Technologien erstellt wird. Der Begriff ist so aber nicht standardisiert.

Tatsächlich kann man in der Tat mobile Inhalte in vielen Situationen dem Anwender so präsentieren, dass er kaum einen Unterschied zwischen einer mobilen Web-Applikationen bzw. mobilen Webseiten auf der einen Seite und solchen Web-Apps auf der anderen Seite erkennt. Es gibt viele Gemeinsamkeiten, aber auch Unterschiede, die sich zum Teil auf Grund der Programmierung, der Speicherung und Bereitstellung der Daten und insbesondere der Verwendung auf dem mobilen Endgerät resultieren.

2.2.1 Webseiten

Einmal sollte man reine Webseiten als eher passive Angebote verstehen, die dem Besucher klassische Web-Funktionalitäten bereitstellen. Das gilt auch für spezielle mobile Webseiten, wie sie etwa mit jQuery Mobile erzeugt werden können. Diese sind zwar auf die mobilen Gegebenheiten durch geeignete Benutzerschnittstellen hin optimiert. Sie sind dennoch gewöhnliche Webseiten, die nur den Besonderheiten mobiler Umgebungen orientiert sind. Das sind etwa die spezifischen Eingabemöglichkeiten oder die oft geringen Bildschirmgrößen und -auflösungen.

2.2.2 Web-Applikationen – RIA

Web-Applikationen hingegen sind zuerst einmal aus Sicht des Betrachters viel interaktiver als konventionelle Webseiten. Es sind vom Verhalten her eben echte Applikationen, mit denen man richtig interagieren kann und die Leistungen bereitstellen, wie man sie von Desktop-Applikationen kennt. Auch wenn diese wie normale Webseiten in einem Browser geladen werden. Etwa Routenplaner wie Google Maps, Terminverwaltung wie Google Calendar oder die zahlreichen Web-Spiele. Unter dem Begriff RIA (Rich Internet Application) fasst man diese Angebote meist zusammen. Diese Web-Applikationen kann man natürlich auch gut an die Gegebenheiten auf den mobilen Endgeräten anpassen.

Für unsere Situation wollen wir diese Unterschiede zwischen mobilen Webseiten und Web-Applikationen aber nicht weiter ausarbeiten, denn wir wollen ja Apps in den Fokus stellen.

2.2.3 Single-page-Webanwendung

Als Single-page-Webanwendung (SPA) wird eine Webanwendung bzw. RIA bezeichnet, die aus einem einzigen HTML-Dokument besteht und deren Inhalte dynamisch nachgeladen werden. Diese Art von Web-Architektur steht im Gegensatz zu klassischen Webanwendungen, welche aus mehreren, untereinander verlinkten HTML-Dokumenten bestehen. Durch solch eine Architektur wird die Grundlage für eine Webanwendung in Form einer Rich-Client- bzw. Fat-Client-Verteilung geschaffen. Funktionalität wird verstärkt auf die Clientseite verlagert (inklusive dem Speichern der Sitzungszustände im Client), was zu der Reduzierung der Serverlast führt und auch Offline-Anwendungen ermöglicht.

Das SPA-Paradigma eignet sich auch ausdrücklich für die Einbettung in native mobile Anwendungen in Form von hybriden Anwendungen, die über ein Framework HTML-Seiten verwenden – eben Apache-Apps.

2.2.4 Besonderheiten von Web-Apps

Web-Apps lassen sich recht deutlich von den reinen Webseiten/Web-Applikationen differenzieren. Allgemein stellen mobile Web-Applikation oder mobile Webseite Inhalte bereit, die von einem Webserver in einem – bereits auf dem mobilen Endgerät vorhandenen – Browser geladen werden und auch nur dort „leben“. Sie werden innerhalb dieses Browsers interpretiert und es wird nichts auf dem Gerät des Anwenders installiert. Dadurch, dass mobile Webseiten und Web-Applikationen jedoch immer im Rahmen eines Standardbrowsers ausgeführt werden, werden sie auch durch den Browser beschränkt. Das ist ein sinnvolles Sicherheitsfeature, aber eben auch eine Einschränkung gegenüber den Möglichkeiten, die native Programme auf einer Plattform bieten. Eine Web-App arbeitet hingegen zwar auch mit gewöhnlichen Web-Technologien wie HTML oder JavaScript, wird aber über einen geeigneten Mechanismus als **eigenständige App** auf dem mobilen Endgerät installiert und kann – gegebenenfalls über Schnittstellen wie Cordova – Ressourcen des mobilen Endgeräts mehr und besser nutzen als es bei Web-Applikationen möglich ist. Oder anders und ein bisschen leger ausgedrückt – eine Web-App bringt ihren eigenen Browser mit, der auf eine geeignete Weise auf dem mobilen Gerät installiert wird (bzw. den vorhandenen Browser so klont und anpasst, dass das möglich ist). Über diese Browserkomponente wird die Web-App letztendlich nativ ausgeführt. Man sollte aber dennoch noch einmal festhalten, dass der Kerncode von mobilen Web-Applikationen/Webseiten und Web-Apps hinsichtlich der „Geschäftslogik“ weitgehend identisch ist bzw. sein kann.

▶ **Tipp** Im Rahmen einer Cordova-Web-App können Sie dort eine „normale" Webseite oder Web-Applikation integrieren. Dazu wird in der App ein eigenständiges Browserfenster samt eigenständiger Engine (ein sogenannter **Inappbrowser**) integriert. Technisch wird das in Cordova über ein spezielles Plugin[1] (*cordova-plugin-inappbrowser*) realisiert. Ein spezielles Objekt stellt dann eine *open()*-Methode bereit, die Sie wahrscheinlich aus dem DOM-Konzept von dem *window*-Objekt kennen. Genau wie dort wird damit einfach eine neue Internet-Adresse (URL – Uniform Ressource Locator) geöffnet und angezeigt. Etwa so:

```
var ref = cordova.InAppBrowser.open('http://rjsde', '_blank', 'location=yes');
```

2.2.5 Native Apps

Kommen wir noch einmal auf die genauen Spezifika der echten nativen Apps und deren Erstellung zurück. Der Begriff einer App ist traditionellerweise (sofern man bei so wenigen Jahren schon von einer Tradition sprechen kann) mit einer nativen Applikation verbunden, die auf einem speziellen mobilen Gerät installiert und dann dort ausgeführt wird. Schon Mitte der 90iger-Jahre hatte zum Beispiel Sun dafür spezielle Java-APIs zur Verfügung gestellt, über die sogenannte Midlets erstellt werden konnten. Das waren Java-Programme, die sich an die speziellen Gegebenheiten auf mobilen Geräten anpassen konnten.

Im Grunde ist eine native App einer nativen Desktop-Applikation ähnlich. Wie auch diese ist eine native App bereits in plattformspezifische Anweisungen übersetzt und wird beim Anwender installiert. Sie ist damit für eine bestimmte Plattform optimiert. Mit allen Vorteilen und Nachteilen, die im Buch schon angedeutet wurden. Die Vorteile einer nativen App sind eine gute Performance und vor allen Dingen eine optimale Anpassung an eine spezielle Plattform sowie die Möglichkeit zum Zugriff auf besondere Hardwarebestandteil der mobilen Endgeräte. Eine solche App kann sich auch harmonisch in übergeordnete Bedienkonzepte einfügen, wie es Apple mit seinem proprietären Habitat oder auch Microsoft mit seinem Design ab Windows Phone 7 fordern. Ebenso können viele vorgefertigte Features aus nativen Schnittstellen und APIs benutzt werden. Besonders wichtig ist, dass die verschiedenen App-Stores echte Apps einfordern. Wenn Sie also einen kommerziellen Vertrieb oder auch nur eine etwas weitergehende Verbreitung anstreben, kommen Sie um eine echte App nicht herum.

Die Nachteile echter nativer Apps liegen aber darin, dass ein Programmierer sich mit anspruchsvollen Programmiertechniken auskennen (etwa Java unter Android, Object C für Apple oder C# und .NET unter Windows Phone) sowie die dafür notwendigen

[1] Was Plugins genau sind, werden wir noch ausführlich behandeln. Sie können es sich im Moment als Erweiterung der normalen Funktionalität vorstellen.

Entwicklungswerkzeuge beherrschen muss. Und ein Anwender muss eben die App auch installieren. Der entscheidende Punkt ist jedoch, dass Sie bei einer App – wie schon mehrfach erwähnt – immer für eine spezielle Zielplattform entwickeln.

▶ Wie angedeutet werden wir mit Cordova und seinen Wrapper über diese Web-Apps einen **hybriden** Weg zwischen Web-Applikationen und nativen Apps gehen und somit das Beste aus beiden Welten nutzen können.

2.2.6 Die Arbeitsweise nativer Apps

Nun ist es meines Erachtens für das Verständnis der Arbeitsweise von Apps mit Cordova elementar, dass Sie einen kurzen Einblick in die Erstellung echter nativer Apps bekommen. Details brauchen und werden wir nicht verfolgen, aber zumindest für Android und Windows Phone sowie iOS mit etwas einfacher Praxis andeuten. Der Abschnitt wird aber vor allen Dingen das Verständnis fördern, was Sie bei nativen Apps machen müssten, wie diese grundsätzlich arbeiten und wo Ihnen Web-Apps den Weg erleichtern. Zudem werden hier die Grundlagen zum Erstellen eines Cordova-Projekts mit einer typischen IDE gelegt.

Hintergrundinformation

Die Beispielcodes in diesem Kapitel und dem folgenden Kapitel werden bewusst nicht zu Verfügung gestellt, damit Sie das Anlegen eines Projekts wirklich selbst durchführen müssen. Zum einen sind das ja nur native Dummyprojekte, aber zum anderen ist es aus verschiedenen Gründen später unabdingbar, dass Sie das Anlegen eines Projekts beherrschen und es hier üben. Sie müssen später in der Praxis insbesondere die Projekte immer so anlegen, dass sie für Ihre Entwickler- als auch Zielplattform(en) passen. Ein reines Kopieren vorhandener Projekte kann schon deswegen oft schiefgehen, da etwa unter Android bei Ihnen möglicher Weise ganz andere Targets installiert sind. Und dann sind da noch die ganzen Metainformationen und Einstellungen, die nur beim direkten Anlegen eines Projekts richtig passen. Das ist auch ein Grund, warum die späteren konkreten Cordova-Listings, die auf der Webseite zum Buch bereitgestellt werden, auf die reinen Ressourcen beschränkt sind, die wir wirklich unter Cordova selbst programmieren bzw. im Web-Umfeld nutzen.

Ein weiterer Grund ist, dass nur diese Parts neutral von der Zielplattform und deren Version sind. Es ist kaum sinnvoll, wenn Sie da umfangreiche Projekte laden, die dann doch nicht zu Ihren gewünschten Entwicklungs- und Zielplattformen passen, und Sie erst in den Projektstrukturen den Teil suchen müssen, den Sie wirklich benötigen.

2.2.6.1 Eine beispielhafte Vorgehensweise anhand Android

Ich möchte Ihnen zunächst den Weg zu einer nativen App anhand von Android beschreiben, ohne wie gesagt in die Details (insbesondere zu Java) tief einzusteigen. Ebenso werden Sie erfahren, was Sie als Entwickler zum Schreiben und Testen von Apps benötigen – speziell für native Android-Apps, aber auch schon allgemein für Web-Apps. Denn wenn wir später im Buch Web-Apps mit Hilfe von Cordova entwickeln wollen, brauchen wir diese Grundlagen als Basis.

Hintergrundinformation
Ich persönlich entwickle gerne unter Android und die konkrete Kodierung von Apps ist unter Android nach meiner Meinung auch einfach und logisch. Aber das Einrichten und Vorbereiten der Entwicklungsumgebung sowie der sogenannten Targets (Emulationen von realen Geräten) ist meines Erachtens unter Android leider oft recht umständlich und aufwändig. Dies zieht sich durch bis zur Arbeit mit Cordova, was wir noch sehen werden. Wobei die Cordova-Tools das Verfahren sehr vereinfachen.

Unter dem Android-Betriebssystem von Google kann man in verschiedenen Sprachen native Applikationen erstellen. Aber native Apps für Android sind in der Regel in Java geschrieben, zumal das Google selbst mit diversen Tools und APIs unterstützt. Java gilt nun als interpretiert und kompiliert zur gleichen Zeit, was im Grunde einen Widerspruch darstellt. Beide Vorgänge (Interpretation und Kompilierung) beschreiben den Vorgang der Übersetzung von Quelltext in lauffähigen Binärcode, der von einem Computer oder auch einem Smartphone oder Tablet ausgeführt werden kann. Dies kann man auf zwei Arten machen.

Entweder wird der Quelltext auf einen Schlag mit einem geeigneten Programm übersetzt und dann dieser daraus resultierende Binärcode auf dem Zielgerät zum Laufen gebracht. Das bedeutet dann, dass der Quelltext kompiliert wurde.

Man kann aber auch bei einem Anwender den Quelltext laden, Zeile für Zeile lesen und direkt zur Laufzeit des Programms übersetzen lassen. Das ist dann der Vorgang der Interpretation.

Java nutzt beide Vorgänge zur Übersetzung und geht damit einen dritten Weg. Der eigentliche Quellcode wird vor der Auslieferung an den Anwender in einen binären Zwischencode (so genannten Bytecode) kompiliert, der ein architekturneutrales und noch nicht vollständiges Objekt-Code-Format ist. Er ist jedoch noch nicht lauffähig und muss von einer Laufzeitumgebung interpretiert werden. Dies ist die sogenannte JRE (Java Runtime Environment), deren wesentlichen Bestandteil die JVM (Java Virtual Machine – ein virtueller Prozessor) ist.

Da jede Java-Laufzeitumgebung plattformspezifisch ist, arbeitet das endgültige Programm (und damit auch eine entsprechende App) auf dieser virtuellen Plattform. Dort werden alle Elemente hinzugebunden, die für eine spezielle physikalische Plattform notwendig sind. Damit also eine Java-Applikation auf einem Gerät ausgeführt werden kann, muss dort eine passende JRE zur Verfügung stehen. Und das gilt natürlich auch für ein Android-Gerät, wobei Android von Google standardmäßig mit einer JRE ausgeliefert wird.

Hintergrundinformation
Microsoft geht mit seinem.NET-Konzept, das die Basis der nativen Apps unter Windows Mobile bildet, ebenfalls den Weg über eine virtuelle Maschine und einen neutralen Zwischencode, der dort interpretiert wird.

2.2.6.1.1 Die konkrete Erstellung

Java-Apps für Android erstellen Sie am besten mit dem JDK und dem Android Studio, was Ihnen ja im vorherigen Kapitel schon vorgestellt wurde. Sofern dieses bereitsteht, können Sie eine Android-App damit erzeugen, testen und ausliefern. Dies läuft in mehreren Schritten ab, die immer ähnlich sind:

- Zuerst werden Sie ein neues Android-Projekt anlegen. Dabei können Sie verschiedene Angaben zu der Zielplattform etc. machen.
- Dann schreiben Sie den notwendigen Java-Code samt den ergänzenden Codestrukturen, die etwa auf XML beruhen. Das setzt natürlich entsprechende Kenntnisse voraus. Für unseren Fall werden wir uns weitgehend auf die Vorgabecodes beschränken, die von Vorlagen im Android Studio (Templates) bereitgestellt werden.
- Eine Android-App können Sie wie gesagt aus der IDE heraus in einem Emulator ausführen und testen. Das werden Sie vor einer Fertigstellung immer wieder machen. Dabei ist dieser Emulator allgemeiner zu sehen – Sie können darüber auch hervorragend mobile Web-Apps testen, denn der Emulator enthält – wie jedes reale Android-System – einen Web-Browser, über den Sie Web-Apps bzw. mobile Webseiten laden und testen können.
- Wenn die App fertig ist, werden Sie diese weitergeben wollen. Um eine App weiterzugeben, erzeugen Sie ein spezielles Package. Dieses kann man mit der IDE erzeugen. Ebenso finden Sie hier Features, um Ihre App im Marktplatz von Google zu veröffentlichen, was aber im Moment unseren Rahmen sprengt.

Gehen wir es nun konkret an:

Legen Sie zuerst im Android Studio ein neues Project an. Das finden Sie im Android Studio unter dem Menüpunkt FILE/NEW/NEW PROJET. Über das Dateimenü können Sie auch ein bereits bestehenden Projekt wieder öffnen, ein Projekt direkt von Git klonen, ein Projekt aus einer anderen Programmierumgebung zu wie z. B. Eclipse importieren und dieses in ein Android Studio Projekt umwandeln, diverse Einstellungen vornehmen und ggf. Tutorials und Dokumente einsehen.

▶ **Tipp** Je nach Version der IDE kann sich die genaue Vorgehensweise zum Anlegen von einem Projekt im Detail etwas unterscheiden, aber die grundsätzlichen Schritte sind immer identisch. Wir wollen hier die Version 2.1.2 des Android Studios als Basis nehmen. Beachten Sie, dass sich auch die einzelnen Dialoge je nach konkreter Wahl von Optionen unterscheiden können.

Im ersten Schritt vergeben Sie einen Namen für Ihr Projekt sowie den Firmennamen, der die Package-Struktur unter Java gleich mit festlegt. Gegebenenfalls können Sie den Speicherort ändern (Abb. 2.1).

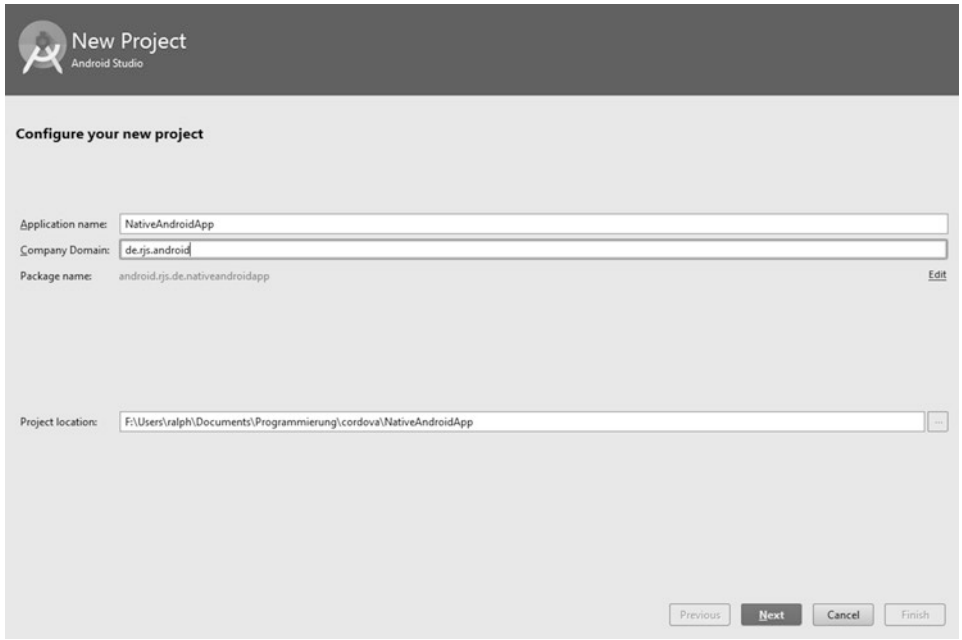


Abb. 2.1 Name, Package-Struktur und Speicherort des Android-Projekts festlegen

Der nächste Schritt wird interessant. Hier kann man auswählen, für welche Plattform man entwickeln möchte. Unter Minimum SDK können Sie die Android Version auswählen, die minimal unterstützt wird (Abb. 2.2). Sie können auch angeben, bei Sie Wear unterstützen wollen. Das ist ein Betriebssystem auf Basis von Android und speziell für Smartwatches und andere „Wearables".

Android Studio bringt eine Reihe von bereits vorgenerierten Templates (Activities), die Sie in dem folgenden Schritt auswählen können (Abb. 2.3). Probieren Sie einfach ein interessant erscheinendes Template hier aus. Für unsere Zwecke genügt aber die leere Activity.

▶ Eine Activity ist in Java erst einmal eine Klasse. Unter einer nativen Android-App repräsentiert sie genau ein sichtbares Benutzerinterface. Beispielsweise kann eine Activity den gesamten Bildschirm des mobilen Endgeräts einnehmen – mit oder ohne ein Auswahlmenü. Ebenso verwaltet die Activity eventuell mit einem Menü oder Schaltflächen verbundenen Aktionen der App. Eine App kann mehrere Activities besitzen, zwischen denen gewechselt werden kann. Jedoch kann bei einer App immer nur eine Activity aktiv sein. Eine Activity ist also in gewisser Weise modal.

Im nächsten Schritt können eventuell notwendige Anpassungen von Namen für Layout, Java-Klasse etc. vorgenommen werden (Abb. 2.4).

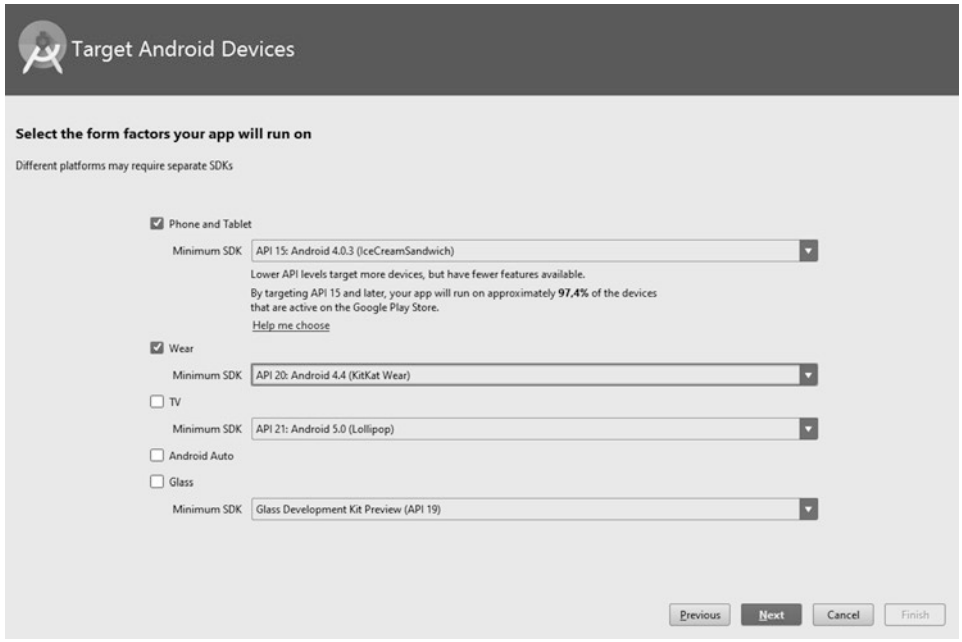


Abb. 2.2 Auswahl der Zielplattformen mit Minimalforderungen

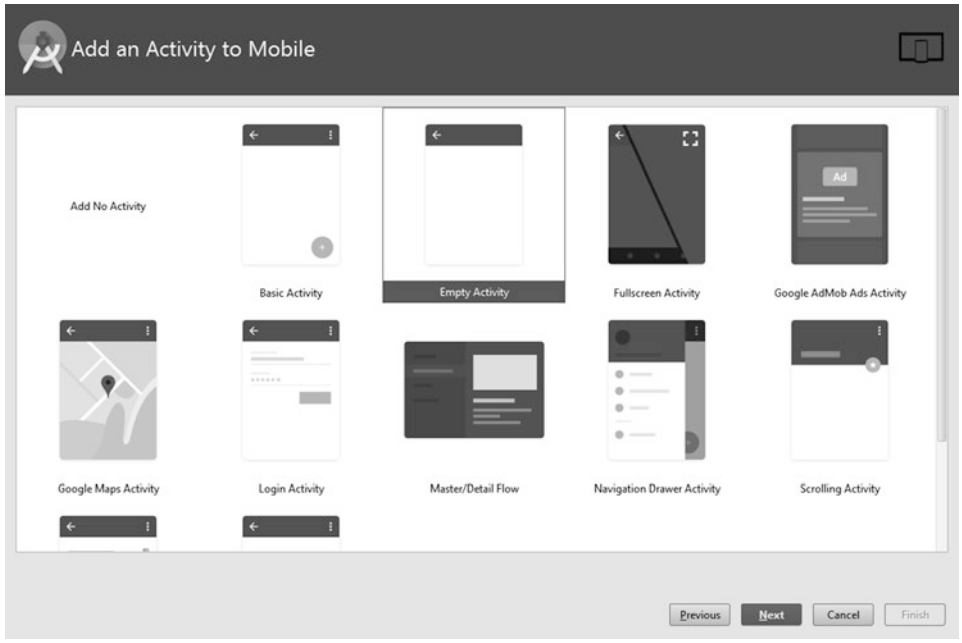


Abb. 2.3 Eine Activity auswählen

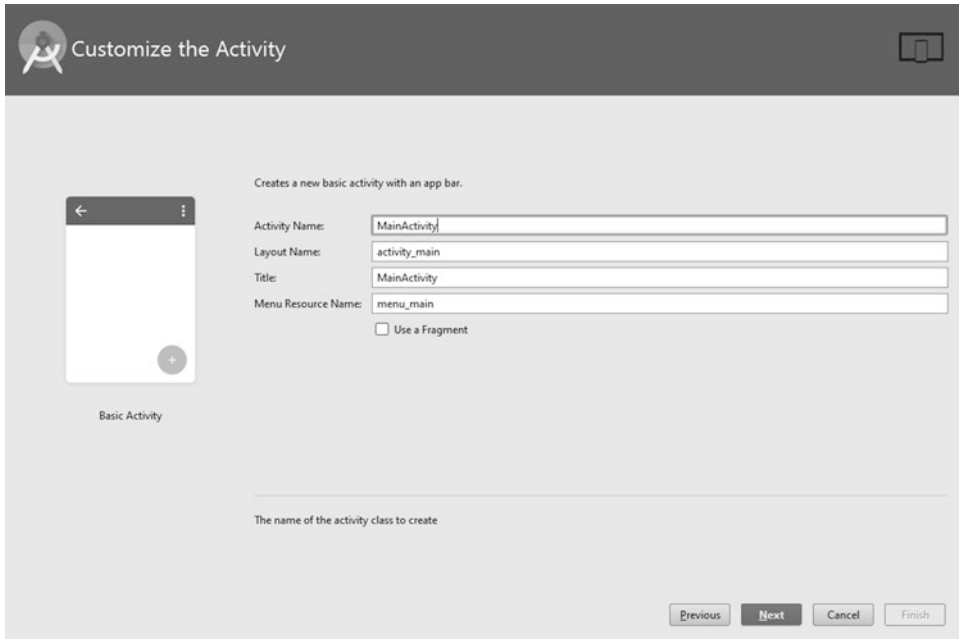


Abb. 2.4 Falls notwendig, können hier noch Namen angepasst werden

Sofern Sie etwa Unterstützung für Wear oder Smart-TVs voreingestellt haben, tauchen eventuell weitere Schritte auf, die noch erledigt werden müssen. So kann etwa ein Schritt noch notwendig werden, in dem Sie noch eine Activity für Android Wear auswählen können, wenn Sie dafür Unterstützung gewählt haben (Abb. 2.5).

Der Klick auf den FINISH-Button – egal in welchen Dialog er zu sehen ist – beendet den Assistenten. Nach einer kurzen Ladezeit müsste das Projekt erstellt und in der IDE geladen werden. Dann sehen Sie auf der linken Seite den sogenannten Project Explorer des Android Studios und auf der rechten Seite eine oder mehrere Dateien des Projekts.

2.2.6.1.2 Eine Android-App im Emulator ausführen

Nun können Sie die App im Explorer der IDE mit der rechten Maustaste auswählen und aus dem folgenden Kontextmenü oder mit dem RUN-Menü im Emulator ausführen.

Der Start der App im Emulator kann eine Weile[2] dauern – also nicht ungeduldig werden. Sie werden auch keine sonderlich spannende App sehen, denn es ist nur etwas Default-Code generiert worden. Aber rein funktional ist die App vollständig und so unter Android lauffähig (Abb. 2.6).

[2]Selbst auf meiner Workstation mit Xeon-Prozessor, 6 echten Kernen, SSD und 32 GByte RAM kann ich mir zwischenzeitlich einen Kaffee kochen.

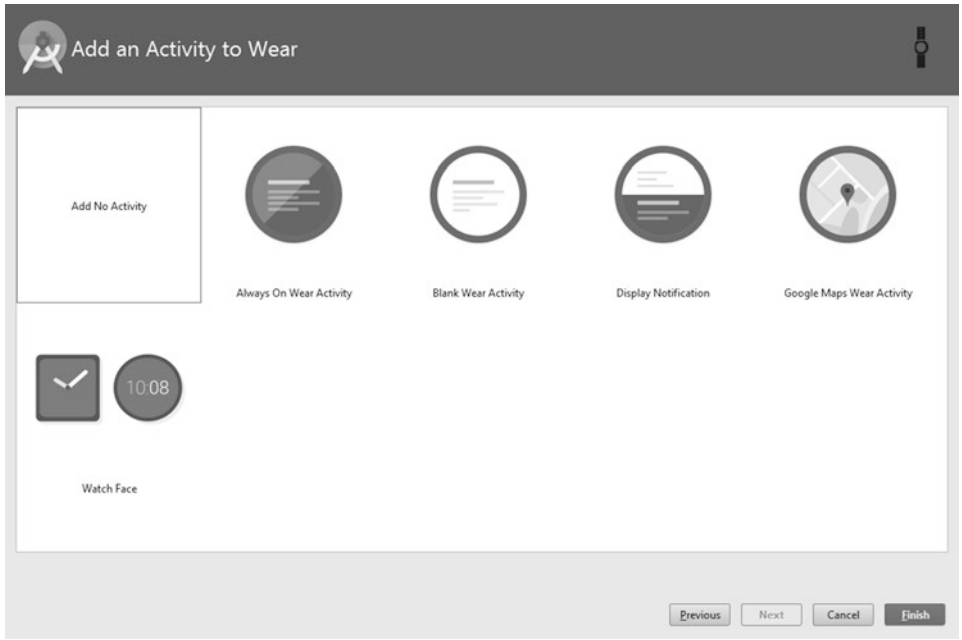


Abb. 2.5 Bei Bedarf kann Wear-Unterstützung hinzugefügt werden

Es kann auch sein, dass Sie vor einer Ausführung der App in einer Emulation erst einmal ein Target anlegen oder aktualisieren müssen. Auch dazu gibt es einen Assistenten, den Sie einfach mit den Vorgabeeinstellungen durchlaufen sollten. Dabei kann das Laden und Einrichten aber ebenfalls ziemlich lange dauern.

2.2.6.1.3 Eine Android-App im Ripple-Emulator ausführen

Der Ripple-Emulator ist ein Tool, welches im Rahmen des Chrome-Browsers als RIA ausgeführt wird und die Produktivität bei der Entwicklung hybrider Web-Apps erheblich verbessert. Denn insbesondere das Testen mit dem Android-SDK-Emulator kostet viel Zeit für das Erstellen sowie das Starten der Anwendung im Emulator. Selbst bei guter Hardware kann man wie gesagt oft zwischenzeitlich Kaffeekochen gehen – von der Wartezeit auf schwacher Hardware ganz zu schweigen. Der Ripple-Emulator hingegen startet sehr schnell und vor allem kann auch die Sensorik von mobilen Endgeräten damit sehr gut getestet werden. Aber wie gesagt – als Web-Applikation zeigt auch der Ripple-Emulator einige Verhaltensweise, die nur näherungsweise das echte Verhalten der App abbilden (etwa funktioniert kein wirklicher Zugriff auf das Dateisystem oder XML- und HTML-Tags im Content werden als Teil der Webseite verarbeitet).

Der Ripple-Emulator wird Ihnen – entsprechende Erweiterungen und Chrome vorausgesetzt – in Visual Studio zur Verfügung stehen und es gibt für Chrome auch eine

Abb. 2.6 Die gestartete App im Emulator

Standard-Ripple-Emulator-Erweiterung, aber die ist meist veraltet. Die neuste Version von Ripple installiert man am besten mit npm:

```
npm install -g ripple-emulator
```

Hintergrundinformation
Auf npm gehen wir noch genauer ein (Kap. 3). Das ist ein Paketmanager, den wir für die Installation von Cordova verwenden werden.

Anschließend kann man den Emulator über die Konsole aufrufen

```
ripple emulate [Pfad zur Indexseite der App]
```

Der Chrome-Browser wird daraufhin mit der App, die innerhalb von Ripple läuft, gestartet.

2.2.6.1.4 Eine Android-App exportieren und auf einem mobilen Gerät ausführen

Um eine Android-App auf einem realen Smartphone oder Tablet auszuführen, muss sie da installiert werden. Dazu kopieren Sie einfach die.apk-Datei mit der App auf das Gerät und öffnen diese dort. Sofern keine Rechteprobleme bestehen, wird die App installiert und Sie können sie ausführen. Ebenso benötigen so eine.apk-Datei, wenn Sie die App im Marktplatz von Google veröffentlichen bzw. vermarkten wollen.

Aber wie wird aus Ihrer App eine solche.apk-Datei? Mit Hilfe des BUILD-Menüs im Android Studio können Sie die App in der Form exportieren. Dabei können Sie sowohl eine signierte (der Befehl GENERATE SIGNED APK) als auch unsignierte App (mit dem Befehl BUILD APK) exportieren, was letztendlich sowohl die Akzeptanz der App als auch die Rechte für die Installation berührt (Abb. 2.7).

Wollen Sie die App später vertreiben, müssen Sie auf jeden Fall den Assistenten zum Signieren durchlaufen. Aber auch zum Testen ist das besser, da Sie sich sonst möglicherweise unnötige Probleme bei der späteren Installation in der Praxis einhandeln. Der Assistent zum Exportieren und Signieren Ihrer App fordert von Ihnen nach der Angabe eines Namens die Spezifikation eines passwortgeschützten Schlüsselspeichers (Keystore – Abb. 2.8).

Darin werden die Schlüssel für Ihre App verwaltet, über die Sie die App mit einem Zertifikat signieren können (Abb. 2.8). In dem gewählten Schlüsselspeicher können Sie

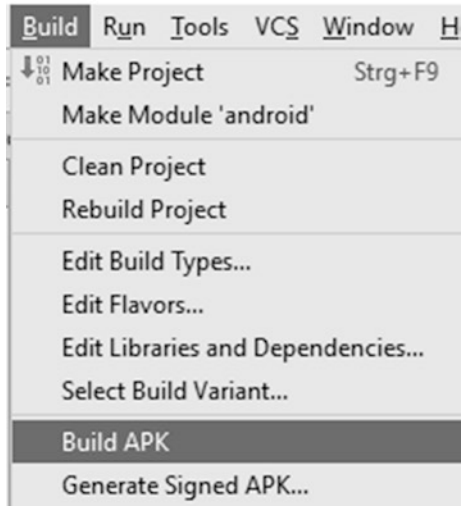


Abb. 2.7 Eine App zum Export erstellen

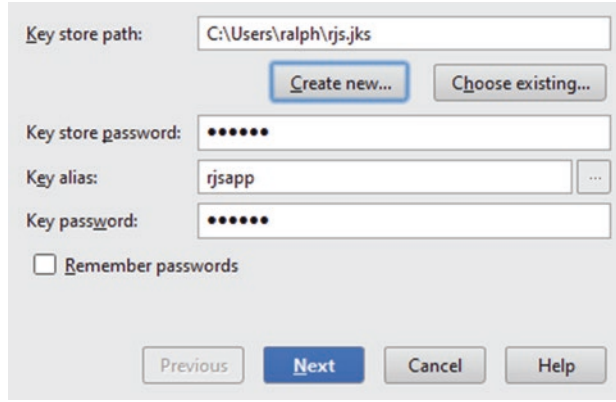


Abb. 2.8 Auswahl des Schlüsselspeichers

dann einen konkreten Schlüssel auswählen, über den Sie die App signieren wollen. Sie können sowohl einen vorhandenen Schlüsselspeicher verwenden als auch selbst einen neuen anlegen (Abb. 2.9).

Entweder verwenden Sie einen bereits vorhandenen Schlüssel oder aber Sie legen einen neuen Schlüssel an. Bei einem neuen Schlüssel müssen Sie in einem weiteren Schritt neben einem Alias für den Schlüssel und einem Passwort einige Eckdaten wie Ihren Namen und Wohnort angeben, woraus dann ein eindeutiges Zertifikat generiert wird.

Im letzten Schritt speichern Sie dann die zertifizierte App. Anschließend können Sie diese auf Ihr Smartphone kopieren oder im Google Markt veröffentlichen.

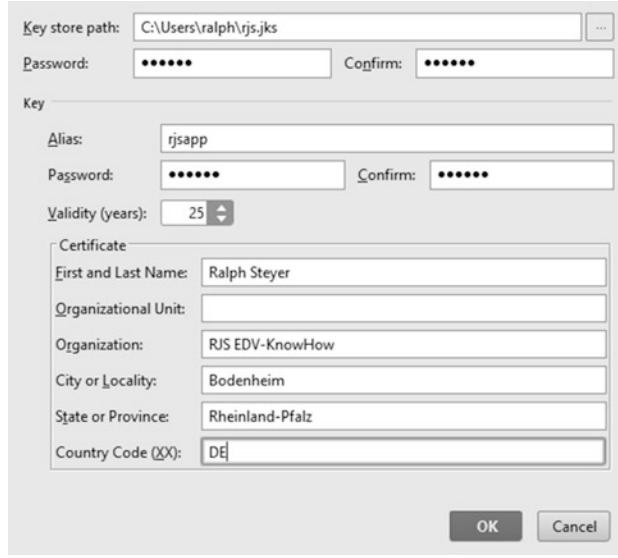


Abb. 2.9 Einen neuen Keystore anlegen

2.2.6.2 Native Apps mit Visual Studio

Ich möchte Ihnen nun einen zweiten Weg zu einer nativen App in einer anderen Welt vorstellen. Dabei verwenden wir als IDE Visual Studio, womit man ursprünglich gezielt für Windows Phone entwickelt hatte.[3] Microsoft unterstützt mittlerweile aber ausdrücklich auch andere Plattformen, so dass dieser Abschnitt zwar im Fokus auf mobile Windows-Systeme abzielt, aber auch Android und iOS umfasst. Deshalb nennt sich dieser Abschnitt auch nicht „Native Windows Apps mit Visual Studio", sondern ist universeller gehalten.

Sie werden sehen, dass sich trotz diverser Unterschiede in den Details die Grundschritte ähneln werden, da man im Grunde für alle nativen Apps ähnlich vorgehen muss – gleich welche Zielplattform man im Auge hat und welche IDE oder Werkzeuge man verwendet. Und auch hier gilt, dass wir weitere notwendige Grundlagen für Web-Apps mit unter Cordova legen.

Zuerst aber möchte ich ein paar Begriffe klären, die im Umfeld mobiler Windows-Apps auftauchen.

2.2.6.2.1 Windows Phone und Windows Mobile versus UWP-Apps

Windows Phone bezeichnet den direkten Nachfolger von **Windows Mobile**, mit dem Microsoft im Zusammenhang mit Windows 8 eine weitere Fusion von mobilen und stationären Anwendungen und Geräten vorantreiben wollte. Insbesondere wurde ein vereinheitlichtes Designkonzept in den Fokus gestellt. Zentraler Aspekt war der Zwang für Hardware-Hersteller, dass ihre Geräte einen Mindeststandard einhalten und eine einheitliche Benutzeroberfläche bereitstellen mussten, damit Windows Phone damit ausgeliefert werden durfte. Bei der Benutzerführung ging Microsoft neue Wege und hatte sich damit im Look & Feel als auch der Philosophie deutlich von Android-Geräten abgehoben.

Durch die mittlerweile weitgehend vollzogene Ablösung von Windows 7 und 8.x durch Windows 10 verliert aber auch Windows Phone deutlich an Bedeutung und Windows Mobile kann man vollständig ignorieren. Die Zukunft gehört den **Universal Windows Platform App**s – auch universelle Windows Apps oder einfach nur Windows Apps. Dabei handelt es sich um spezielle Anwendungen, die auf verschiedenen Geräten unter Windows 10 eingesetzt werden können.

Universal Windows Platform Apps (kurz UWP Apps) sind Anwendungen, die sich ausdrücklich auf verschiedenen physikalischen Plattformen einsetzen lassen – zum Beispiel einem Tablet Computer, einem Smartphone, einer Spielekonsole, aber auch auf einem normalen Personal Computer. UWP Apps sind also nicht beschränkt auf den Einsatz auf einem bestimmten Gerätetyp, sondern können für eine ganze Gerätefamilie entwickelt werden.

▶ Beachten Sie, dass aber für UWP-Apps auf allen diesen unterschiedlichen Plattformen Windows 10 laufen muss. Auch ist als Entwicklerplattform Windows 10 notwendig – Vorläuferversionen von Windows eignen sich nicht.

[3] Und natürlich allgemein für Windows.

Da mobile Geräte in der Regel über Touchscreens bedient werden, arbeiten Universal Windows Platform Apps mit anderen Bedienkonzepten als klassische Anwendungen. So gibt es zum Beispiel Bedienelemente wie eine Symbolleiste oder ein Menüband in dieser Form nicht.

Grundsätzlich lassen sich Universal Windows Platform Apps aber auch mit der Maus und der Tastatur bedienen. Das ist zum Beispiel dann wichtig, wenn Sie sie auf einem klassischen Computer ohne Touchscreen einsetzen.

Der Vertrieb der Universal Windows Platform Apps erfolgt so gut wie ausschließlich über den **Windows Store**, der „traditionell" zum Verteilen von Windows-Apps genutzt wird. Dabei handelt es sich um einen virtuellen Marktplatz im Internet, auf dem unterschiedlichste Apps angeboten werden. Das Konzept ist im Grunde vollkommen analog wie bei Google oder Apple. Der Windows Store kann direkt über Windows aufgerufen werden und steht damit grundsätzlich allen Windows-Anwendern zur Verfügung.

Universal Windows Platform Apps weisen im Vergleich mit Windows Forms-Anwendungen und auch WPF-Anwendungen (was für die älteren Formen der Apps im Windows-Umfeld die Basis war) einige Besonderheiten auf. Insbesondere bilden sogenannte **Seiten** (pages) die Basis einer App. Die Darstellung der Inhalte erfolgt immer auf einer oder mehreren Seiten. Jede Seite übernimmt dabei eine eindeutige Aufgabe – zum Beispiel die Anzeige von vorhandenen Dateien und das Laden einer Datei. Zwischen den Seiten einer App kann der Anwender hin- und herwechseln.

Universal Windows Platform Apps weisen vor allen Dingen hinter den Kulissen erhebliche Unterschiede zu Windows Forms- und WPF-Anwendungen auf, die für Programmierer wichtig sind:

- Eine UWP-App muss verschiedene Darstellungen unterstützen und auf verschiedenen Geräten einsetzbar sein.
- Die App muss daher mit unterschiedlichen Auflösungen arbeiten können. Zusätzlich muss die App für mobile Geräte auch noch das Hoch- und das Querformat unterstützen.
- Eine App hat nur eingeschränkten Zugriff auf das System. Dazu kommt eine Sandbox zum Einsatz. Auf andere Bereiche des Systems hat die App nur dann Zugriff, wenn der Benutzer das ausdrücklich genehmigt. Die Zugriffsregeln werden durch Capabilities beschrieben. So etwas kennt man ja bei Android schon lange. Als Programmierer hat man damit aber nicht mehr kompletten Zugriff auf die Systemressourcen. So kann man zum Beispiel nicht mehr ohne weiteres Daten aus dem Internet herunterladen oder die Kamera eines Smartphones nutzen.
- Mehrere Apps können über aber über Contracts (Verträge) zusammenarbeiten. Damit kann man als Programmierer auf die Dienste bereits vorhandener Apps zurückgreifen oder aber die Dienste einer App selber anderen Apps zur Verfügung stellen.
- Jede UWP App hat einen eigenen Lebenszyklus. Sobald die App in den Hintergrund gestellt wird, wird sie durch das Betriebssystem in eine Art Schlafzustand (Suspend-Modus) versetzt, aber nicht beendet. Nur im Ausnahmefall wird das Betriebssystem eine App komplett beenden.

Auch für die Gestaltung der konkreten App fordert Microsoft die Einhaltung von ein paar Regeln:

- Die App soll möglichst einfach und sachlich erscheinen. Das folgt dem Konzept, dass eine App so selbsterklärend wie eben möglich sein sollte. Im Mittelpunkt steht der Inhalt – und nicht die Gestaltung. Dazu wird nicht nur die Oberfläche sehr sachlich gehalten, sondern auch Bedienelemente wie Symbolleisten oder Menüleisten sollen nicht verwendet werden. Auf besondere grafische Effekte soll komplett verzichtet werden. Der Einsatz von Animationen – zum Beispiel für Seitenübergänge – ist aber durchaus erlaubt und von Microsoft sogar gewünscht.
- Die App soll einfach über Gesten bedient werden können. Die Bedienelemente sollten daher nicht zu klein sein oder zu eng nebeneinander liegen. Denn hier kann bei der Bedienung mit den Fingern unter Umständen nicht mehr exakt genug gezielt werden. Die Bedienung soll nach Möglichkeit direkt über Elemente auf einer Seite erfolgen. Daher fallen auch viele klassische Menüleisten, Menübänder und Symbolleisten aus. Sie eignen sich für die Bedienung über Gesten nicht, weil die Elemente zu nah beieinander liegen oder sind in der Anwendung zu kompliziert. Als Ersatz kommen aber einige neue Elemente wie die Befehlsleiste zum Einsatz.
- Die App soll schnell und flüssig arbeiten. Der Benutzer soll nie das Gefühl haben, auf die App warten zu müssen. Daher sollte der Anwender bei jeder Aktion, die er über die Oberfläche startet, nach Möglichkeit Feedback erhalten – zum Beispiel in Form einer Animation oder eines Seitenwechsels. Dadurch weiß der Anwender auch, dass die App seine Bedienkommandos angenommen hat.

2.2.6.2.2 Die Entwicklung von Universal Windows Platform Apps

Bei der Entwicklung von Universal Windows Platform Apps wird konsequent zwischen der Präsentation und der Logik getrennt. Für die Beschreibung der Oberfläche können Sie neben XAML zum Beispiel auch HTML verwenden. Die Logik – also das eigentliche Programmverhalten – lässt sich mit Programmiersprachen wie C++, JavaScript, C# oder Visual Basic abbilden.

Für das Erstellen von Apps verwendet man Visual Studio in einer geeigneten Version. Grundsätzlich genügt die Community Edition 2015 (oder Folgeversionen), die aber unter Umständen erweitert werden muss. Nach dem Start legen Sie dort ein neues Projekt an. Dazu gehen Sie unter Datei -> Neu -> Projekt.

Die Vorlagen für Universal Windows Platform Apps finden Sie im Zweig für Ihre bevorzugte Sprache. Etwa unter Visual Basic/Windows. Dort finden Sie auch Vorlagen für Windows 8, aber in dem Buch konzentrieren wir uns auf UWP Apps (Abb. 2.10).

Wir verwenden für unser Beispiel eine leere App aus der Kategorie Visual Basic.

▶ **Tipp** Je nachdem wie Sie die Visual Studio Installation durchgeführt hatten – als Standardinstallation, als vollständige Installation beziehungsweise als benutzerdefinierte Installation – existiert die Vorlage für Universelle Windows

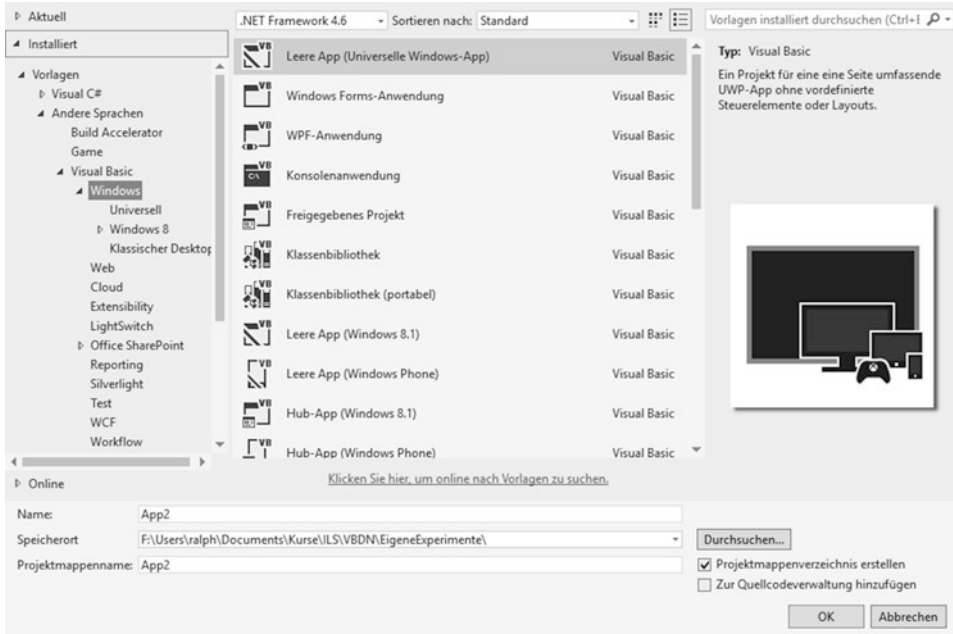


Abb. 2.10 Eine Vorlage für eine UWP App

Apps auf Ihrem System möglicherweise noch nicht. Sie haben dann aber die Möglichkeit, die Installation direkt aus dem Fenster zum Anlegen eines neuen Projekts über einen Doppelklick auf den Eintrag UNIVERSELLE WINDOWS-TOOLS INSTALLIEREN zu starten. Sie müssen dazu einfach die fehlenden Features installieren und anschließend den Anweisungen der Installationsroutine folgen. Aber auch über das EXTRAS-Menü und dort die Anweisung EXTENSIONS UND UPDATES können Sie fehlende Features nachinstallieren.

Der ganze Vorgang kann erheblich Zeit in Anspruch nehmen, da gegebenenfalls auch weitere Visual Studio Updates mitinstalliert werden. Unter Umständen müssen Sie während des Installationsvorgangs Ihren Computer auch mehrfach neu starten.

Wenn die Vorlage bereitsteht, markieren Sie den Eintrag für eine leere universelle Windows App. Geben Sie anschließend einen Namen für das Projekt ein. Wir verwenden in unserem Beispiel den Namen *HalloWeltApp*. Klicken Sie auf OK, um das Projekt anzulegen. Sie legen dann noch die Ziel- und Mindestplattformversionen fest, die Ihre universelle Windows-Anwendung unterstützt bzw. vorhanden sein muss (Abb. 2.11).

▶ **Tipp** Es kann sein, dass Sie im nächsten Schritt den Entwicklermodus für Windows 10 aktivieren müssen, sofern das noch nicht vorgenommen wurde (einmal eingestellt, brauchen Sie das nicht mehr machen). Sie finden diese

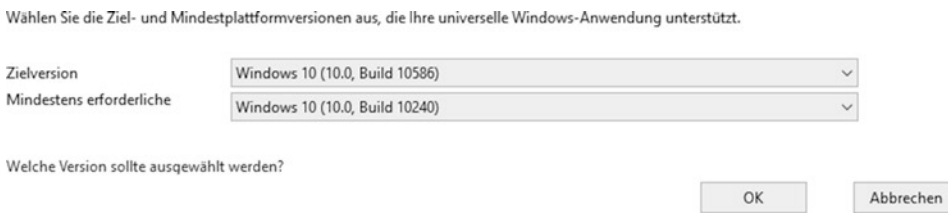


Abb. 2.11 Ziel- und Mindestplattformversionen

> Einstellungsmöglichkeit in Windows 10 unter EINSTELLUNGEN und dort im Bereich UPDATE UND SICHERHEIT. Hier gibt es den Link „FÜR ENTWICKLER". Klicken Sie diesen an. Im nächsten Schritt müssen Sie den Entwicklermodus für Windows 10 aktivieren. Bestätigen Sie ggfls. die Sicherheitsabfrage.

Das Projekt wird nun angelegt und erscheint in der Entwicklungsumgebung (Abb. 2.12). Dabei wird zunächst die Datei *App.xaml.vb* im Editor angezeigt. Diese Datei ist die Code-behind-Datei der eigentlichen App.

Im Projektmappen-Explorer rechts im Fenster von Visual Studio sehen Sie auch noch weitere Dateien und Ordner. Im Ordner *Assets* werden unter anderem Logos für die App abgelegt.

Wechseln Sie bitte in die Startseite der App. Doppelklicken Sie dazu auf die Datei *MainPage.xaml* im Projektmappen-Explorer. Damit wir der visuelle Bearbeitungsmodus geöffnet (Abb. 2.13).

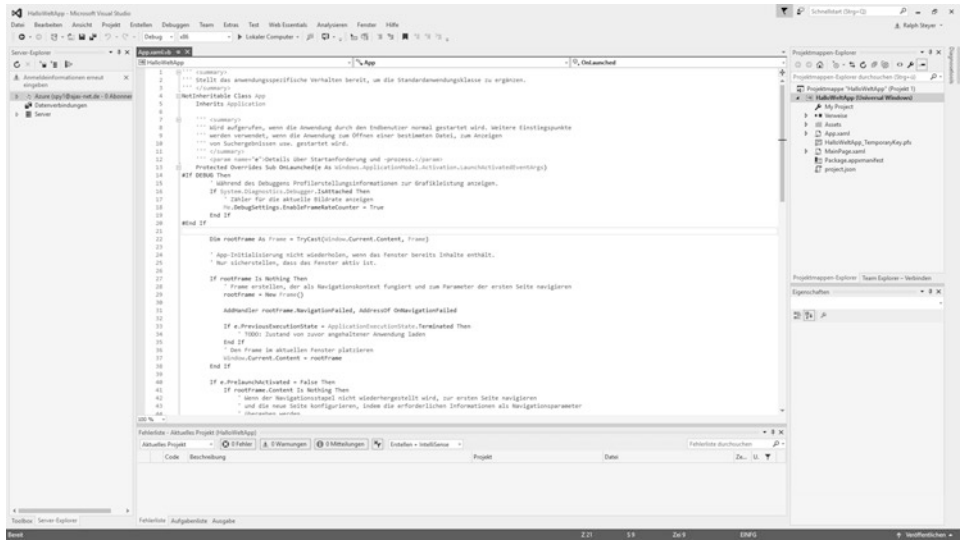


Abb. 2.12 Die App in Visual Studio

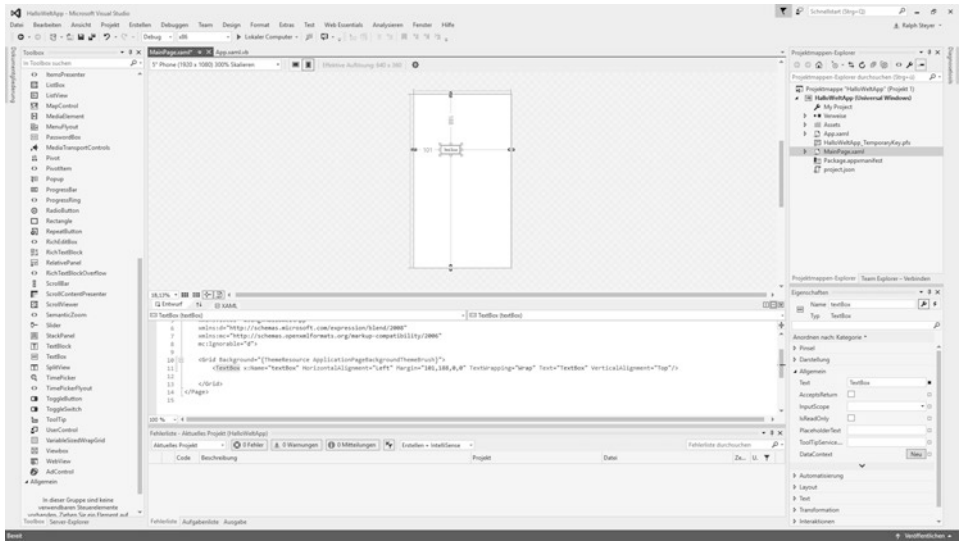


Abb. 2.13 Die XAML-Datei in Visual Studio visuell bearbeiten

Über die Toolbox oder auch direkt im XAML-Code können Sie jetzt Steuerelemente einfügen. Probieren Sie das bitte aus.

Setzen Sie einen Textblock für die Anzeige von Text in die App. Das entsprechende Steuerelement finden Sie in der Toolbox unter dem Namen TextBlock.

Ändern Sie den Text für den Textblock dann über die Eigenschaften – zum Beispiel in „Hallo Welt“. Damit der Text besser zu sehen ist, können Sie auch die Schriftgröße anpassen – zum Beispiel auf 24. Die entsprechenden Einstellungen finden Sie bei den Eigenschaften im Bereich Text.

Positionieren Sie danach das Steuerelement neu, und lassen Sie die App ausführen. Dazu können Sie über das Listenfeld zum Ausführen der App diverse Zielplattformen wählen. Etwa den lokalen Computer, ein angeschlossenes Device (also etwa ein per USB verbundenes Smartphone), einen Simulator oder einen Emulator (Abb. 2.14).

Nach kurzer Zeit sollte die App in einem Emulator zu sehen sein (Abb. 2.15).

Sie können bei Bedarf auch weitere Emulatoren nachinstallieren.

Hintergrundinformation

Ausgeführt wird eine App von Visual Studio in der Standardeinstellung auf Ihrem lokalen Rechner. Damit wird sie auch so angezeigt wie sie auf einem Desktop aussehen würde. Mit einem Simulator oder verschiedenen Emulatoren lässt sich das Verhalten auf einem Tablet oder Smartphone simulieren. Ein Emulator stellt dabei ein anderes Gerät möglichst exakt nachstellt. Hier wird die App in einem emulierten Gerät ausgeführt und verhält sich dabei annähernd so wie auf dem echten Gerät. Die Emulatoren für Windows-Systeme in Visual Studio arbeiten allerdings mit der Virtualisierungssoftware **Microsoft Hyper-V**, die Ihnen nicht in allen Windows-Versionen zur Verfügung steht. Gravierender ist das Problem, dass populäre Virtualisierungs-Lösungen wie VMWare als auch VirtualBox mit Hyper-V kollidieren. Ist Hyper-V aktiviert, lassen sich keine virtuellen Maschinen

Abb. 2.14 Die App aus Visual Studio heraus ausführen

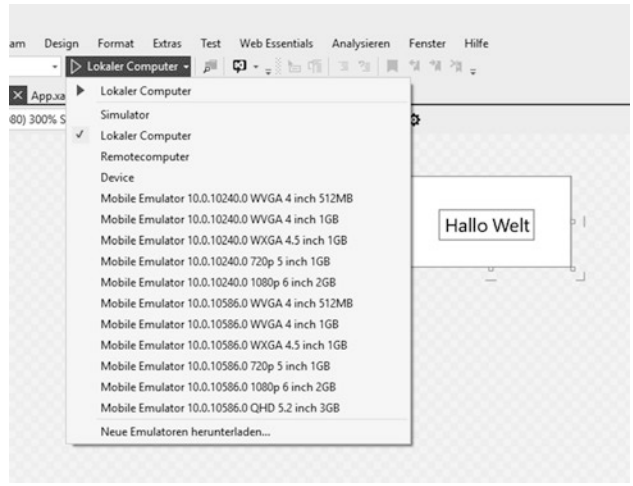
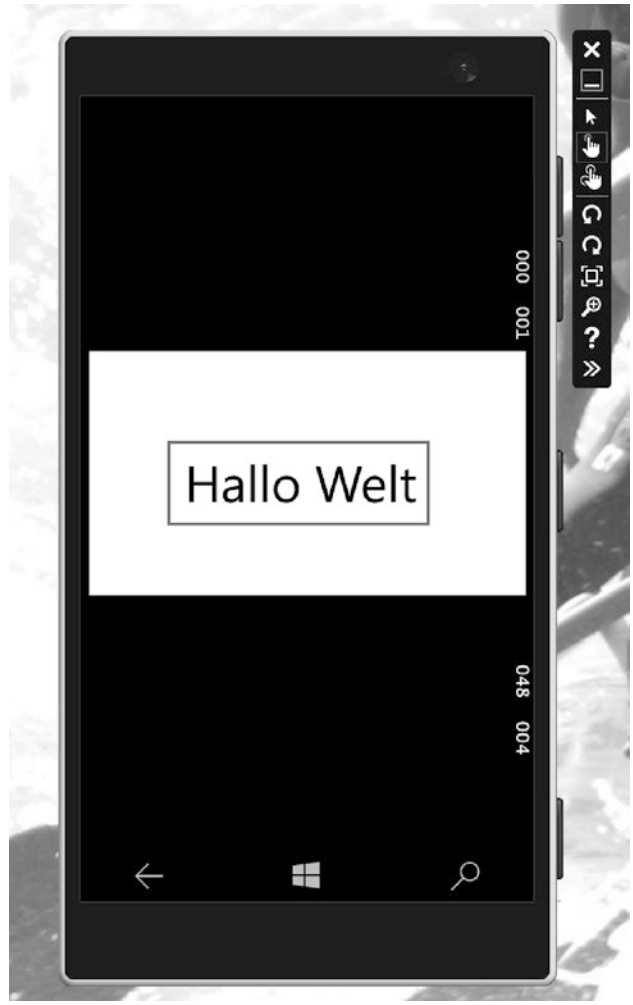
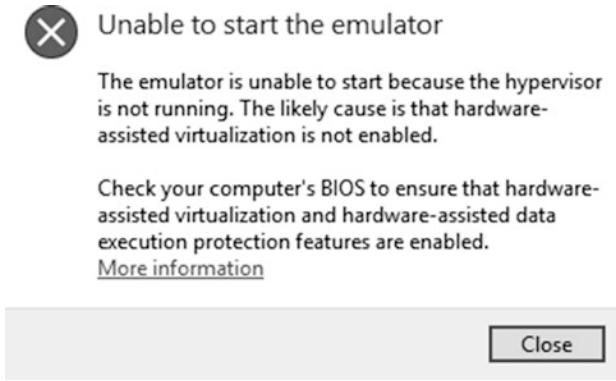


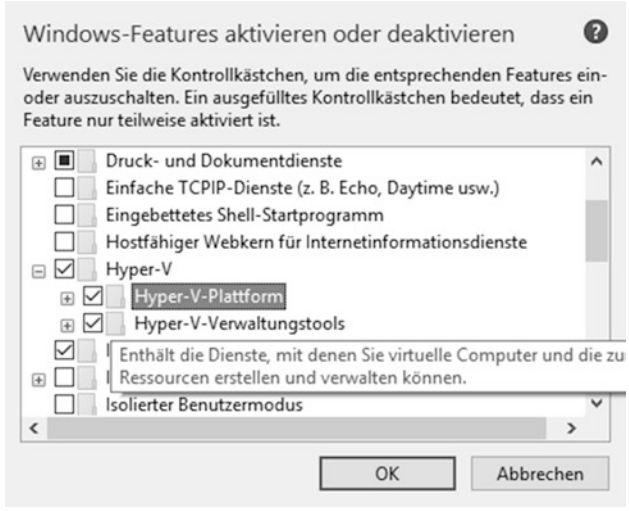
Abb. 2.15 Die App wurde in einem Emulator aus dem Visual Studio ausgeführt



mit diesen Programmen starten und deshalb schaltet man Hyper-V oft ab. Was aber zu einem Fehler beim Start des Emulators für eine Windows-App führt.



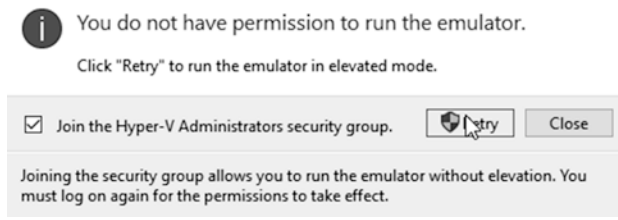
Wenn man also sowohl den Emulator als auch Virtualisierung über VMWare oder VirtualBox benötigt, muss man leider Gottes ständig Hyper-V an- und ausstellen, was jedes Mal einen Neustart des Rechners zur Folge hat. Das bereitet leider erhebliche Umstände. Wenn man in Windows 10 die Rolle „Hyper-V“ aktivieren möchte, kann dies über die Systemsteuerung in Windows 10, PowerShell oder das DISM-Tool (Deployment Imaging Servicing and Management) erfolgen. Wir behandeln nur den Vorgang über die Systemsteuerung. Dazu wählen Sie dort Programme und Features und dann Windows-Features aktivieren oder deaktivieren. Im Folgedialog wählen Sie Hyper-V aus und legen fest, ob Hyper-V aktiviert oder deaktiviert sein soll. Nach dem Neustart des Rechners ist dann Hyper-V entweder neu aktiviert oder deaktiviert.



Nun kann beim Einsatz eines Emulators aus Visual Studio aber auch eine Meldung der Art auftauchen:

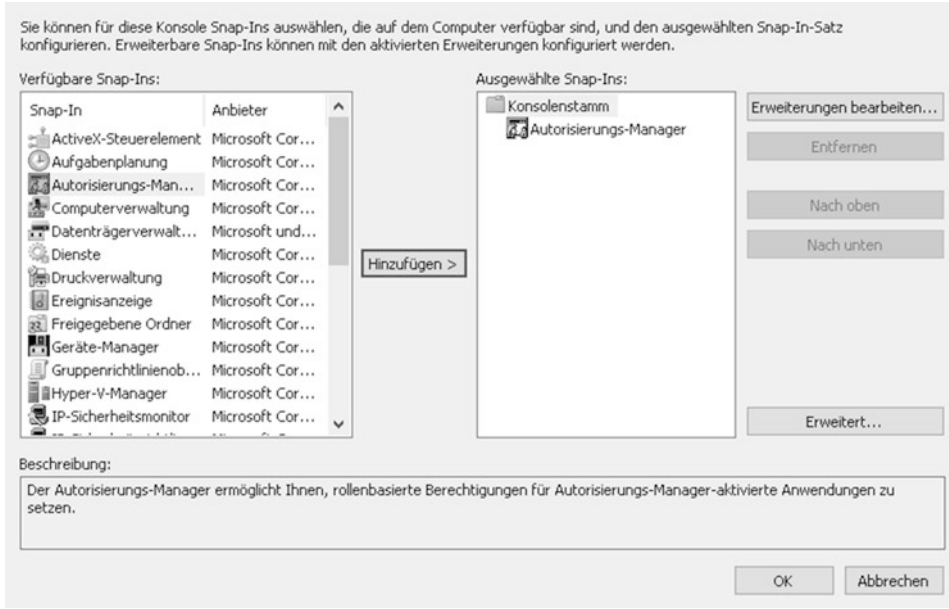
„*Join the Hyper-V Administration security group*“.

Das ist dann der Fall, wenn der angemeldete Benutzer in Windows zu geringe Rechte hat bzw. eben dieser Gruppe nicht angehört und dennoch Manipulationen an Hyper-V notwendig sind (was beim Start des Emulators leider sein kann). Wenn Sie diese Meldung bekommen, können Sie wie über das Service-Tool *mmc* dieser Gruppe beitreten – etwa indem Sie dort das Dateimenü öffnen und SNAP-IN HINZUFÜGEN/ENTFERNEN … wählen oder auch auf andere Weise dieses Hinzufügen bewerkstelligen. Aber dies geht viel zu tief in die Windows-Administration und führt hier definitiv zu weit in eine Richtung, die nicht in unserem Interesse sein kann. Glücklicherweise genügt in den meisten Fällen ein Klick auf die Schaltfläche RETRY und der Emulator startet dann in dem sogenannten elevated-Modus. Das ist für so gut wie alle Aktionen aus der App heraus ausreichend.



Doch die potentiellen Probleme mit Hyper-V sind immer noch nicht ganz abgehandelt. Leider können bestimmte Konfigurationen in Hyper-V auch die Einstellungen der Netzwerkschnittstellen so verändern, dass Sie keinen Zugang zum Internet mehr haben. Der Hyper-V Extensible Virtual Switch wird etwa bei bestimmten Android-Emulationen benötigt, blockiert aber den Internet-Zugang der normalen Netzwerkschnittstelle.

Aus dem Grund ist der Einsatz eines Emulators nicht immer die beste Wahl zum Testen einer Windows- App. Glücklicher Weise gibt es aber in Visual Studio noch den Simulator, der Hyper-V nicht benötigt.



Ein **Simulator** versucht allgemein, wesentliche Aspekte einer ganzen Plattform nachzubilden. Über die Symbolleiste am rechten Rand des Simulators in Visual Studio können Sie zum Beispiel die Bedienung mit dem Finger mit der Maus simulieren. Dazu verwenden Sie die Symbole im oberen Bereich der Symbolleiste. Sie können aber auch die Darstellung im Simulator drehen beziehungsweise ein Drehen des Tablets simulieren.

Die entsprechenden Symbole finden Sie im mittleren Bereich der Symbolleiste. Der Simulator stellt Ihren Rechner als Tablet nach. So finden Sie zum Beispiel auf dem Desktop des Simulators genau die Anwendungen, die Sie auch über den Desktop Ihres Rechners starten können. Allerdings werden die Anwendungen auch als Desktop-Anwendungen ausgeführt. Sie sehen also nicht anders aus als auf Ihrem Desktop.

Wie auch immer Sie die App aus Visual Studio starten – nach einiger Zeit erscheint zuerst der Splash Screen – der Startbildschirm – der Anwendung. Da wir in dieser einfachen App noch kein eigenes Logo angelegt haben, wird hier lediglich ein Viereck mit einem X angezeigt. Kurze Zeit später wird aber auch die Oberfläche der App angezeigt. Entweder in dem Simulator, als Desktop-App oder in einem der verschiedenen Emulatoren, die Visual Studio mitbringt.

2.2.6.3 Eine Android-App unter Visual Studio

Wie erwähnt bemüht sich Microsoft in der letzten Zeit immer mehr plattformübergreifend zu arbeiten. So kann man in Visual Studio mittlerweile ganz hervorragend Android-Apps entwickeln und auch aus der IDE heraus ausführen. Sie benötigen dazu aber dann den **Visual Studio Emulator for Android**. Diese installieren Sie bei Bedarf – wie alle Erweiterungen und insbesondere sämtliche Features für Android als auch Cordova – über Extensions und Updates (Abb. 2.16).

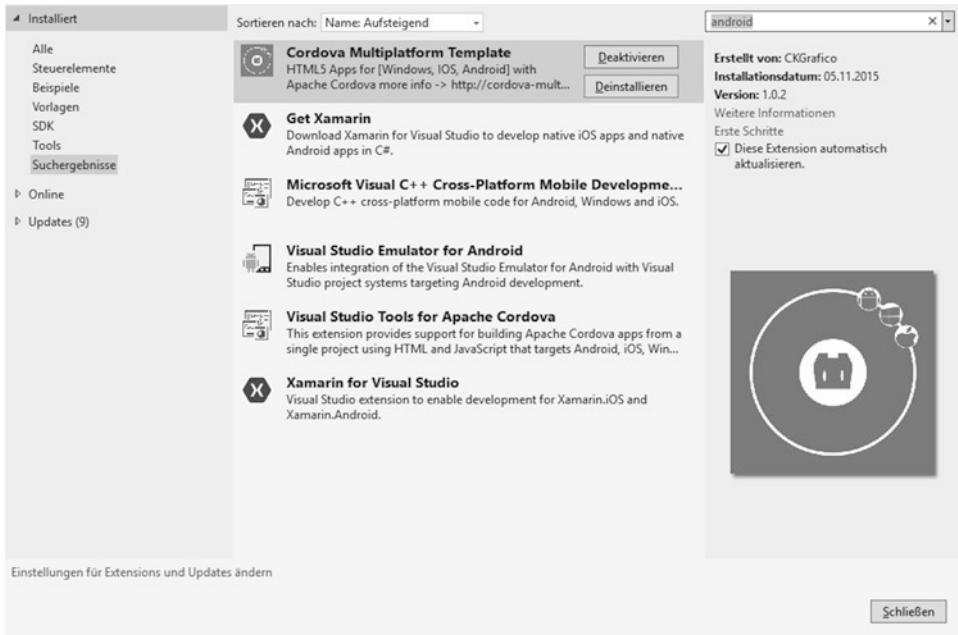


Abb. 2.16 Visual Studio lässt sich vielfältig erweitern

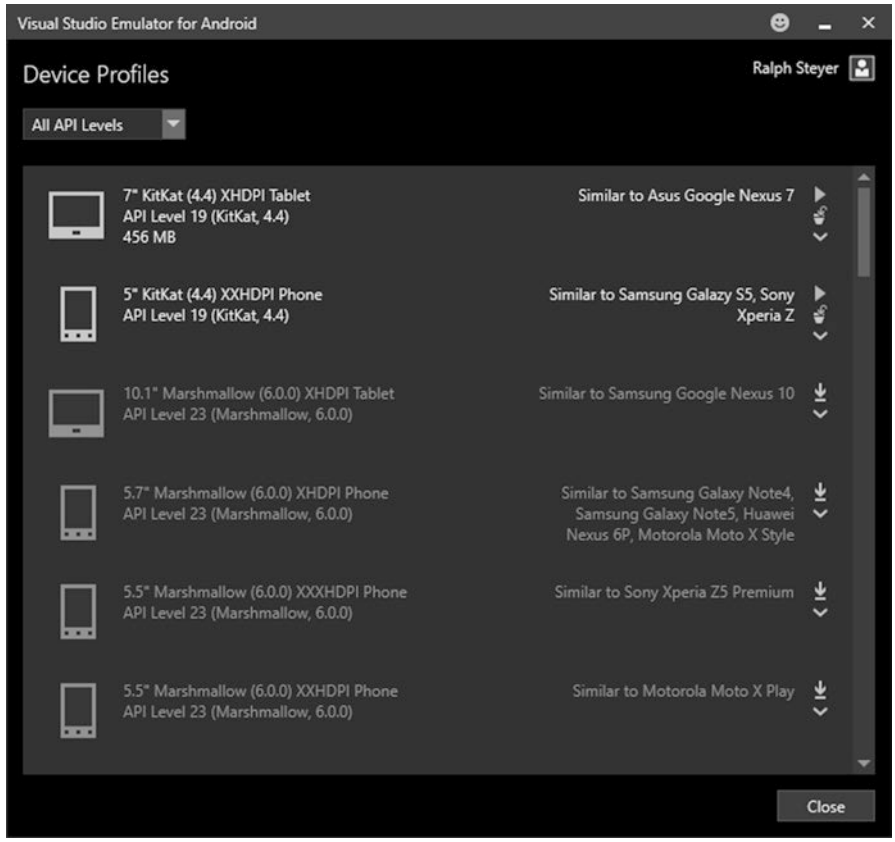


Abb. 2.17 Devices im Android-Emulator unter Visual Studio

Wenn die Unterstützung für Android samt dem zum Test notwendigen Emulator installiert ist, stehen Ihnen Android-Targets bzw. -Devices aus der IDE (Abb. 2.17) als auch Templates zum Erstellen von Android-Apps (Abb. 2.18) zur Verfügung.

Wenn Sie dann in Visual Studio ein neues Projekt anlegen, wählen Sie in den Vorlagen bei der gewünschten Sprache (etwa Visual C# oder Visual Basic) in der Kategorie ANDROID eine App (etwa eine Blank App) aus.

Im Folgenden können Sie wie bei einer Windows-App weiter kodieren (Abb. 2.19). Das wird für uns vor allen Dingen dann interessant, wenn wir Cordova-Apps mit Visual Studio erstellen. Denn da braucht man nur die Web-Ressourcen weiter zu beachten.

Die App kann aber auch aus der IDE heraus direkt in einem Android-Emulator ausgeführt werden (Abb. 2.20).

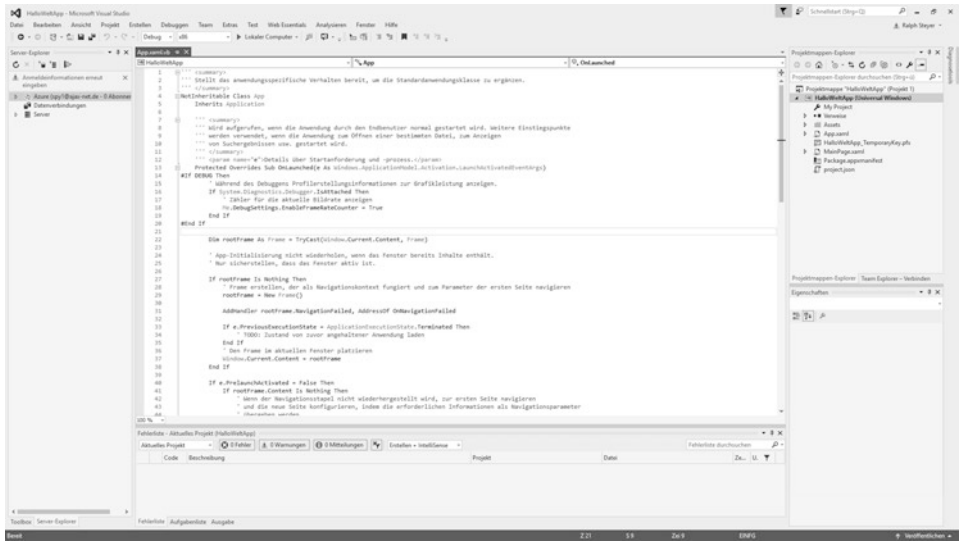


Abb. 2.18 Ein Android-Projekt unter Visual Studio anlegen

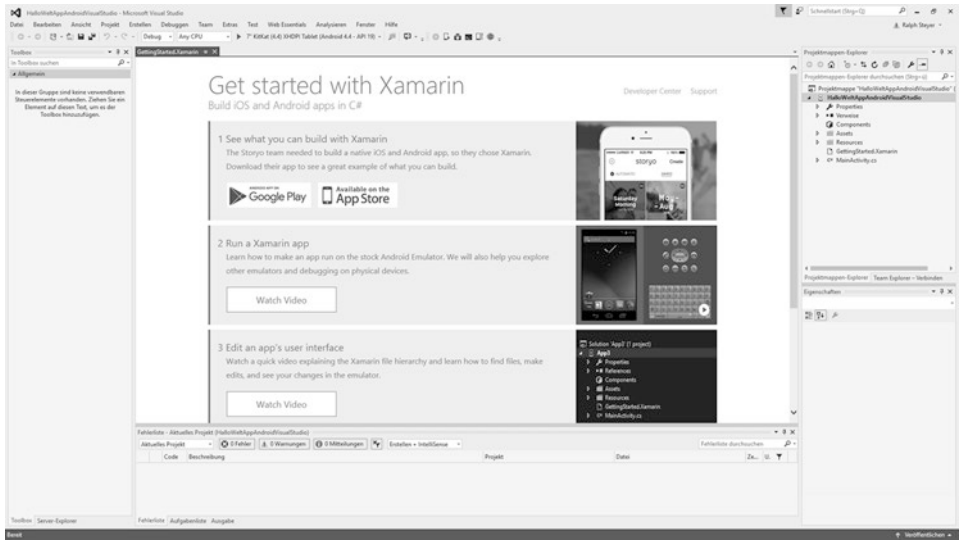


Abb. 2.19 Ein Android-Projekt unter Visual Studio

Abb. 2.20 Die Android-App wurde aus Visual Studio im Emulator gestartet

Und auch der Ripple-Emulator steht in Visual Studio zur Verfügung, wenn Sie die entsprechenden Android/Cordova-Erweiterungen installiert haben.

2.2.6.4 Eine native-App für iOS

Mit dem Visual Studio können Sie mittlerweile sogar Apps für iOS entwickeln, wenn die entsprechenden Erweiterungen installiert sind. Wobei Sie für eine ernsthafte Arbeit auch die entsprechenden Lizenzen benötigen. Aber die Entwicklung einer nativen als auch später einer Cordova-App unter iOS wird in der Regel mit Xcode durchgeführt und ist

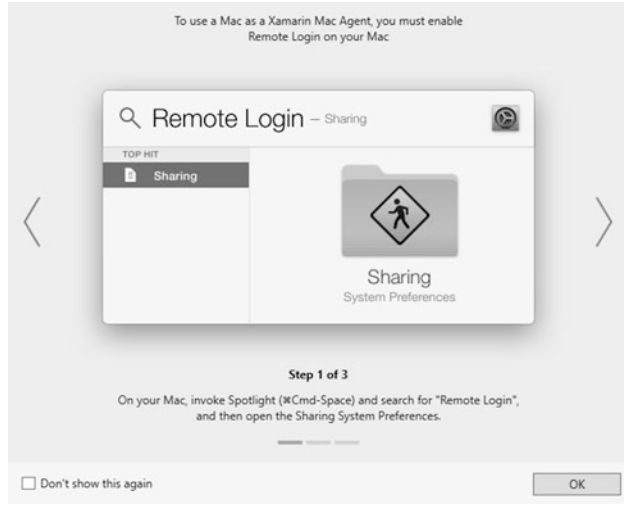


Abb. 2.21 Entwicklung unter iOS

von den Grundschritten ähnlich einfach wie der Vorgang für Windows Phone. Gegebenenfalls müssen Sie einen Mac oder Xamain Mac Agent so konfigurieren, dass einen Remote Login auf dem Mac gestatten (Abb. 2.21).

Unter Xcode wird mit CREATE A NEW XCODE PROJECT ein neues Projekt angelegt. Aus den folgenden Templates können Sie sich eine geeignete vorgegebene Projektstruktur auswählen und dann den folgenden Assistenten durchgehen.

Im so aufgerufenen Assistenten vergeben Sie die wichtigsten Daten zu dem Projekt (analog der Vorgehensweise für Android und Windows – in diesem Fall Name, Organisation, Zieldevice, Speicherort etc.). Bleiben Sie ansonsten bei den Vorgaben. Wenn Sie den Assistenten beenden, haben Sie bereits eine vollständig lauffähige App, die mit dem Klick auf den grünen Pfeil links oben im iOS-Simulator gestartet werden kann.

Zusammenfassung

Sie haben in einer Übersicht das weite Gebiet der mobilen Apps kennengelernt. Sie kennen diverse Begriffe, die wir im Umfeld von Cordova benötigen werden. Insbesondere haben wir einen Blick auf native Apps geworfen und wie Sie diese erstellen können. Sie haben in diesem kurzen Einblick gesehen, dass die Erstellung nativer Apps in verschiedener Hinsicht aufwändig und keineswegs trivial sein kann. Und wir haben ja nur an der Oberfläche gekratzt und für voll funktionale native Apps wäre noch sehr viel zu tun. Zudem ist es notwendig, dass Sie für jede Zielplattform unterschiedlich vorgehen.

Sie wissen, wo die Vor- und Nachteile von nativen Apps und Web-Apps liegen. Es hat eine Menge Charme klassische RIAs auf mobile Endgeräte zu übertragen bzw. dahingehend anzupassen. Denn die meisten modernen Handys, Tablets und Smartphones

sind Internet-fähig und verfügen über einen Standardbrowser. Und die klassischen Webtechnologien sind in der Regel einfacher zu beherrschen als die mächtigen nativen Techniken wie C# oder Java. Insbesondere ist das Wissen verbreitet, da viele Leute Webseiten erstellen und diese Kenntnisse und Erfahrungen übertragen können. Nicht zuletzt durch HTML5, aber auch CSS3 werden Webtechnologien auf mobilen Endgeräten konkurrenzfähig. Gerade im mobilen Umfeld wird HTML5 schon sehr weitreichend unterstützt. Und in Verbindung mit JavaScript und einem leistungsfähigen Framework hat man Funktionalitäten, die sich hinter nativen Möglichkeiten oft nicht verstecken brauchen.

http://www.springer.com/978-3-658-16723-3

Cordova
Entwicklung plattformneutraler Apps
Steyer, R.
2017, XIV, 395 S. 180 Abb., Softcover
ISBN: 978-3-658-16723-3