Wetter-Schätzen will gelernt sein – die ChillSkill-App vergleicht mithilfe von Echtzeitdaten des Webservice forecast.io die geschätzte Temperatur mit der offiziellen Fühl-Temperatur

Wetterfühlig

Ionic ist ein noch junges Open-Source-Framework auf Basis der JavaScript-Bibliothek AngularJS, das die Entwicklung nativer mobiler Apps vereinfacht. Webdesigner Benjamin Rabe und Entwickler Benjamin Behr bauten damit die spielerische Wetter-App ChillSkill

● Ionic bietet Designern den idealen Einstieg in die Entwicklung mobiler Apps, denn es vereinfacht den Umgang mit Webtechnologien wie HTML, CSS und JavaScript ungemein. Es basiert auf dem JavaScript-Framework AngularJS und bietet zahlreiche Module, mit denen sich das Design und die Funktionalität von Apps recht unkompliziert gestalten lassen. Ideal für die Umsetzung unserer Wetter-App. Für ChillSkill wollten wir uns aufs Wesentliche konzentrieren: Die App liefert lediglich die aktuelle Temperatur am eigenen Ort. Die Wetterdaten dafür kann man über Services wie Forecast.io einfach und recht günstig beziehen.

Um das Konzept noch etwas aufzupeppen, führten wir außerdem eine spielerische Interaktion ein: Wir lassen den User seine gefühlte Temperatur, den sogenannten Windchill, eingeben und vergleichen diese mit der offiziellen gefühlten Temperatur. Hat der User gut geschätzt, erhält er eine positive Wertung und ein paar lobende Worte. Das ausführliche Tutorial inklusive des gesamten Codes gibt's auch auf GitHub (https://github.com/benjundbenj/chillskill). Los geht's!

Prototyp erstellen

1 Zunächst gilt es, unsere Idee zu validieren. Mit der Zeichen-App SketchTime (http://sketchtimeapp.com) halten wir erste Scribbles auf dem iPhone fest und definieren darin die wichtigsten Ansichten: Temperaturanzeige, Schätzinteraktion und einen Settings-Screen für Lautstärke und andere Einstellungen. Die Interaktionen planen wir mit der Paper-Prototyping-App POP (http://popapp.in). ❶ Man lädt sich die App aufs Smartphone, knipst damit die Papierskizzen oder importiert andere Scribbles aus der Fotobibliothek und ordnet sie wie gewünscht an. Außerdem kann man mit POP auf den importierten Screens Hotspots anlegen und über Device-typische Gesten zum nächsten Screen überblenden ❷. Wir waren mit der User Experience ziemlich schnell zufrieden und sparten uns die genauere Definition von Schrift, Farbe oder anderer Designparameter.

Ionic installieren, Projekt starten

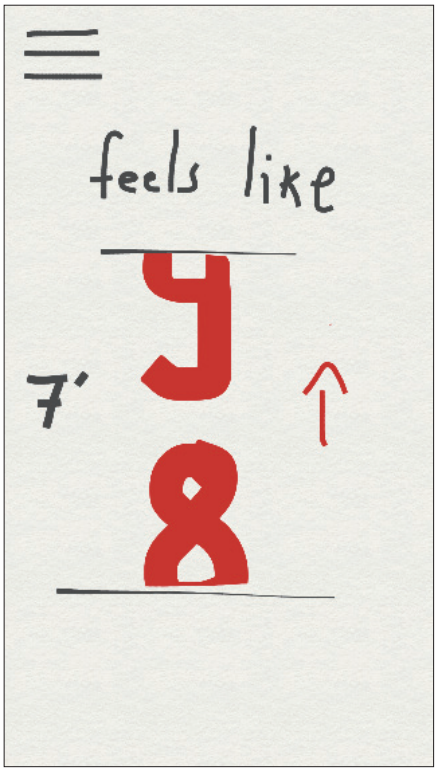
2 Bevor Sie mit der eigentlichen Entwicklung loslegen, müssen Sie sicherstellen, dass die aktuellen Android- beziehungsweise iOS-SDKs auf Ihrem System laufen, denn ohne sie kommt leider auch Ionic nicht aus. Zudem benötigen Sie node.js (http://nodejs.org), um Ionics Kommandozeilen-Tools nutzen zu können. Nach der Installation geben Sie dafür im Terminal `*$ npm install ionic*` ein. Dann können Sie endlich Ionic installieren, indem Sie `*$ npm install -g cordova ionic*` eingeben. In Ionic finden Sie vordefinierte Projekt-Templates, etwa für Tab-Navigation oder für ein Side-Menü. Wir lassen das alles beiseite und starten mit einem leeren Projekt:

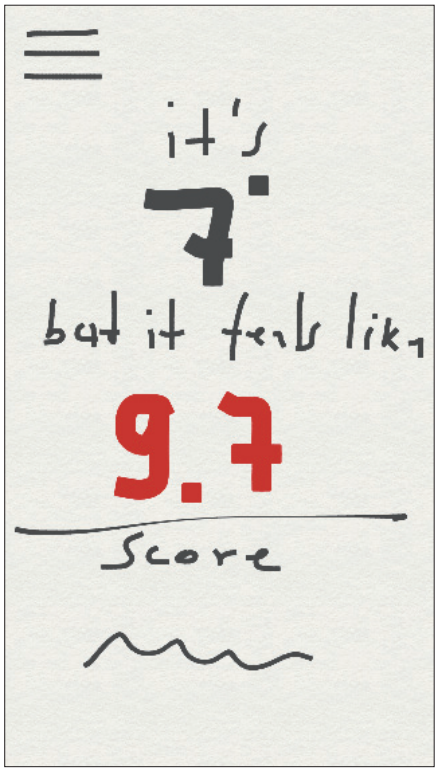
```
$ ionic start ChillSkill blank
```

Auch Ionic müssen Sie mitteilen, in welchem SDK Sie arbeiten. Wir entwickelten ChillSkill in iOS und schrieben daher:

```
$ ionic platform add ios
```

Einer der großen Vorzüge von Ionic ist, dass man im Prinzip eine Web-App entwickelt. Sie können daher die gewohnten Tools nutzen: den Text-Editor zum Entwickeln sowie den Browser zum Testen und Debuggen. →





❶ ❷

→

Grundstruktur der App anlegen

3 Wir erstellen zunächst ein Template, dann einen Controller und zu guter Letzt eine Route. Erst wenn man diese drei einfachen Schritte hinter sich gebracht hat, kann man die App zum ersten Mal richtig anschauen. Das Template ist eine einfache HTML-Datei, in der wir wie im Webdesign das Layout definieren. Um möglichst lange offline und unabhängig von sich ändernden Temperaturwerten arbeiten zu können, verwenden wir zunächst eine vorläufige, fixe Gradzahl – ohne Wetterdienst-Echtzeitanbindung.

Ein Template benötigt immer einen Controller, der die Variablen und Aktionen bereitstellt, damit es die richtigen Daten anzeigt und korrekt funktionieren kann. Die Verbindung zwischen Template und Controller wird über den sogenannten `$scope` hergestellt. Zu diesem Zeitpunkt ist unser Controller noch leer.

Danach legen Sie noch die Route an, damit AngularJS weiß, welches Template und welchen Controller es nutzen muss, um die richtige Bildschirmseite anzuzeigen: Die Route benötigen wir, um auf andere Teile unserer Anwendung zu verweisen und zu verlinken.

```
.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider
  .state('weather', {
    url: "/weather",
    templateUrl: "templates/weather.html",
    controller: "WeatherController"
  })
  $urlRouterProvider.otherwise('/weather');
});
```

Slider einbauen

4 Für den Slider nutzen wir kein Plug-in, sondern bauen uns den Schieberegler mit CSS und ein wenig JavaScript selbst. Das Prinzip dahinter ist einfach: Unseren Slider positionieren wir absolut, und wir aktualisieren den Topwert per JavaScript mit dem Wert, den wir von Ionic erhalten.

Wir bauen also in unserem Template um unseren Temperaturwert ein div mit einer Klasse und positionieren es per CSS absolut. Im Controller registrieren wir jetzt einen Event-Handler auf das `drag`-Event `$ionicGesture.on("drag", drag, $element);` und implementieren die `drag`-Funktion, die die aktuelle Position des Fingers auf dem Display weiterverarbeitet. Die `drag`-Funktion setzt den `y`-Wert auf dem `$scope`, um ihn im Template nutzen zu können. Als letzten Schritt setzen wir den `y`-Wert in ein `style`-Attribut und definieren hier den `top`-Wert. Nun ist unser Slider fertig, und wir sollten ihn im Browser mit der Maus bewegen können.

Ionic bringt, genau wie AngularJS, schon eine ganze Menge Direktiven mit, die es uns ermöglichen, einfach HTML-Elemente zu definieren, sie mit Interaktion auszustatten und an verschiedenen Stellen in der Anwendung einzusetzen. Eine Direktive besteht im Wesentlichen aus zwei Teilen: dem direktiveneigenen Template und der sogenannten `link`-Funktion. Das Template enthält das HTML, dass später anstelle des Direktiven-Codes im Template auftaucht, und die `link`-Funktion definiert die Logik und das Verhalten der Direktive. Wir haben bis zu diesem Zeitpunkt übrigens schon Direktiven eingesetzt: `ion-content` (http://is.gd/ionicframework).

Jetzt verschieben wir das div {{in Script-Schrift??}} im Template in den Template-Teil unserer neuen Direktive. Dann verschieben wir $ionicGesture.on("drag", drag, $element); und die drag-Funktion aus dem Controller in die link-Funktion unserer Direktive. Zum Schluss fügen wir unsere Direktive in das Template ein, wo vorher unser div stand.

```
<ion-content scroll="false">
  <weather-slider></weather-slider>
</ion-content>
```

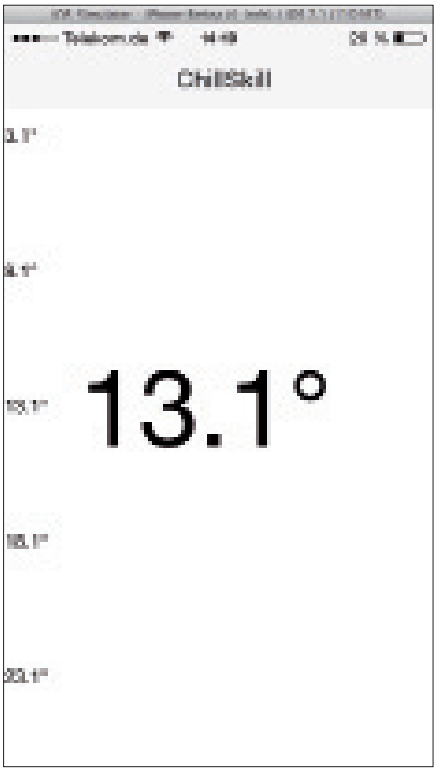
Es ist jetzt an der Zeit, unseren Temperaturwert, den wir vorher statisch im Template eingebaut haben, durch den anhand der Position des Sliders berechneten Wert zu ersetzen. Den Code zum Berechnen der Temperatur mithilfe der aktuellen y-Koordinate finden Sie im Beispiel-Source-Code. Zum Implementieren arbeiten wir in diesem Teil nur noch in der Direktive.

Da wir später die gemessene Temperatur als Ausgangspunkt nutzen wollen, zu diesem Zeitpunkt aber noch keine reale Temperatur zur Verfügung haben, behelfen wir uns mit einen festen Wert: $scope.startTemperature = 13.1;. Wir definieren die startTemperature auf dem $scope, da wir diesen Wert auch zur Erstellung unserer Skala an der linken Seite benötigen. In der drag-Funktion berechnen wir die aktuell gewählte Temperatur und befüllen $scope.guessedTemperature mit dem berechneten Wert. Im Template ersetzen wir die statisch eingetragene Temperatur durch {{guessedTemperature}}.

Wie oben schon erwähnt, werden die Werte, die wir auf $scope definieren in dem jeweiligen Template zur Verfügung gestellt. Wir setzen in der drag-Funktion $scope.temperature immer auf die neu berechnete Temperatur und geben diesen Wert in den doppelt geschweiften Klammern im Template wieder aus. In diesem Fall müssen Sie $scope.temperature und $scope.y innerhalb der $scope.$apply-Funktion setzen, da AngularJS die Veränderungen im Template nicht aktualisiert. Im Normalfall ist dies nicht erforderlich – in unserem Fall aber müssen wir die $apply-Funktion nutzen, da AngularJS Events außerhalb von AngularJS nicht bemerkt. Zu diesen Events gehört unter anderem auch das drag-Event von Ionic. So, im Browser sollten wir jetzt beim Sliden eine sich verändernde Temperatur sehen.

Die App emulieren

5 An dieser Stelle schauen wir einmal kurz zurück, was wir bisher geschafft haben. Per Paper Prototyping haben wir uns Gedanken darüber gemacht, wie unsere App später funktionieren soll. Wir haben Ionic installiert und das Projekt erstellt. Dann haben wir eine App gebaut, die einen Slider zeigt, der die Temperatur anhand der Position des Fingers auf dem Display berechnet und anzeigt.



❸

Bislang haben wir alles im Browser getestet, jetzt wollen wir uns die App im Emulator anschauen. Wir entwickeln unter Ionic in einem Arbeitsverzeichnis. Damit daraus eine App wird, brauchen wir am Ende ein vollständiges Xcode-Projekt. Keine Sorge, auch das übernimmt wieder Ionic für uns. Im Terminal stellen wir sicher, dass wir in unserem Verzeichnis sind, und geben ein:

```
$ ionic build ios
```

Ionic baut für uns jetzt das Xcode-Projekt zusammen. Wir können nun die App im Emulator testen und geben dazu folgende Zeile ein:

```
$ ionic emulate ios
```

Nach kurzer Zeit öffnet sich der iOS-Simulator, und wir haben einen ersten Blick auf die native App ❸.

Temperatur schätzen

6 Der Slider ist fertig. Nun wollen wir den Nutzer nach dem Sliden fragen, ob er mit dem Schätzen fertig ist, um ihm im nächsten Schritt das Ergebnis seiner Schätzung mitteilen zu können. Die Abfrage und die weitere Interaktion ist kein Teil der Direktive mehr, sondern findet wieder im Controller statt. Da der Controller nichts vom Sliden mitbekommt, muss der Slider dem Controller mitteilen, dass er mit dem Sliden fertig ist. Hierzu schicken wir ein Event Richtung Controller, der dann darauf reagieren kann.

Als Erstes arbeiten wir im Template und bauen uns ein div, das den Satz »So, you feel like it's …°?!« und einen Knopf anzeigt, der bei Klick das Ergebnis der Schätzung zeigen soll. Dieses div bekommt jetzt die AngularJS-Direktive ng-show="slideFinished" →

→ als zusätzliches Attribut, damit wir es verstecken oder zeigen können, je nachdem, ob der Nutzer mit dem Sliden fertig ist oder nicht. Nun registrieren wir in unserer Direktive einen Event-Handler auf das dragend-Event von Ionic und schicken dann ein eigenes Event Richtung Controller: `$scope.$emit(„temperature-slider.dragend", event);`. Im Controller registrieren wir einen Event-Handler, der auf temperature-slider.dragend reagiert und eine eigene Funktion ausführt, die den Knopf anzeigt. Dieser Event-Handler setzt `$scope.slideFinished = true`, und das Template zeigt daraufhin unseren Satz, weil ng-show den Wert true bekommen hat.

Wie Sie sicher schon bemerkt haben, ist der Satz »So, you feel like it's …°?!« noch unvollständig, da er die geschätzte Temperatur noch nicht darstellt. Bisher weiß nur der Slider über die geschätzte Temperatur Bescheid. Da der Controller, der ja irgendwie auch das Ergebnis präsentieren muss, ebenfalls die geschätzte Temperatur kennen muss, nutzen wir eine der zentralsten Grundeigenschaften von AngularJS aus: das sogenannte 2-Way-Data-Binding. Im Endeffekt bedeutet das nichts anderes, als dass die Direktive und der Controller beide Zugriff auf dasselbe Temperatur-Objekt haben. In unserer App verändert die Direktive den Wert des Temperatur-Objekts, und da der Controller mit demselben Objekt arbeitet, bemerkt er die Änderungen. Für unsere App bedeutet das folgende Änderungen:

Im Controller setzen wir `$scope.guessedTemperature = 13.1;`. Im Template fügen wir dem Slider-Element das Attribut `guessed-temperature="guessedTemperature"` hinzu. Attribute auf Direktiven-Elementen können von der Direktive als Variablen übernommen werden. Man muss sich das so vorstellen, als ob man im JavaScript einer Funktion Parameter übergibt. Dieses Attribut baut die Verbindung zwischen dem Controller und der Direktive auf und ermöglicht beiden, mit demselben Temperatur-Objekt guessedTemperature zu arbeiten. In der Direktive müssen wir jetzt im scope: `{ guessedTemperature: "=" }` gesetzt werden. Nun steht guessedTemperature in der Direktive zur Verfügung. Im Template können wir unseren Satz dann wie folgt verändern: ¿¿»So, you feel like it's {{guessedTemperature}}°?!«.

Damit haben wir den komplexesten Teil unserer App umgesetzt! Der Controller liefert nun den initialen Wert für die geschätzte Temperatur, das Template stellt der Direktive das Objekt zur Verfügung und die Direktive verändert beim Sliden den Wert des Objekts. Da wir das 2-Way-Data-Binding nutzen, sind die Werteveränderungen sofort im Template und im Controller verfügbar.

Schätzwert und offiziellen Temperaturwert abgleichen

7 Als Nächstes wollen wir dem Nutzer das Ergebnis seiner Schätzung anzeigen, indem wir anhand der Differenz zwischen geschätzter Temperatur und realem Temperaturwert einen Feedback-Satz auswählen. Je genauer die Schätzung, desto positiver das Feedback. Zudem soll der echte Wert von oben in den Slider bouncen.

Im Template fügen wir unter unserem Satz ¿¿einen Button `let's check your chill skill!` {{meint ihr den Button »let's check your chill skill!« ?}} mit dem Attribut {{mithilfe des Attributs?}} `ng-click="showResults()"` ein. Ng-click ist auch wieder eine Direktive, die von AngularJS mitliefert wird. Sie führt die definierte Funktion aus, wenn der Nutzer auf das Element klickt. Im Controller definieren wir jetzt auf $scope die Funktion showResults. Wie oben erwähnt, können Funktionen, die auf $scope im Controller definiert wurden, im Template verwendet werden. Die showResults-Funktion stellt dem Template die echte Temperatur zur Verfügung und wählt den der Genauigkeit der Schätzung entsprechenden Satz aus und stellt auch diesen dem Template zur Verfügung. Des Weiteren verändert er noch die Werte zweier Variablen, die wir benötigen, um den Bestätigungsknopf auszublenden und den Ergebnissatz anzuzeigen.

Im Template fügen wir jetzt ein weiteres div mit dem Inhalt {{rating}} unter das div mit dem Bestätigenknopf ein. Dieses div bekommt das Attribut `ng-show="rateSkill"`, was die Inhalte des divs anzeigt, wenn $scope.rateSkill endlich true wird. Im Controller legen wir zunächst die »echte Temperatur« durch einen eigenen Wert fest: `$scope.currentTemperature = 14.2;`. Auch an dieser Stelle setzen wir currentTemperature wieder auf den $scope, da wir diesen Wert auch für den bouncenden Temperaturwert benötigen. Würden wir ihn nur im Controller benötigen, würden wir ihn auf eine Variable ohne $scope setzen. Jetzt definieren wir die chooseRating-Funktion im Controller und rufen diese in unserer showResults-Funktion auf: `$scope.rating = chooseRating($scope.guessedTemperature, $scope.currentTemperature);`. $scope.rating enthält jetzt den Satz und wird in unserem neuen div dargestellt.

Erneut überprüfen wir im Browser, ob alles funktioniert. Nachdem Beenden des Slidens sollte unten der Bestätigenknopf auftauchen, der bei Klick wiederum den Ergebnissatz anzeigt.

Anzeige der realen Temperatur

8 Der von forecast.io gelieferte Wert für die gefühlte Temperatur soll von oben an die korrekte Stelle im Slider fallen. Wir fügen daher im Template ein weiteres Attribut zu unserer Direktive hinzu: `current-temperature="currentTemperature"`. Zudem müssen wir in dieser der scope-Definition currentTemperature hinzufügen, um currentTemperature in ihr nutzen zu können. Im Direktiven-Template definieren wir ein div, das die currentTemperature anzeigen soll. Es bekommt das Attribut ng-show="currentTemperature", das dafür sorgt, dass {{?wer??}} das div nur dann anzeigt, wenn currentTem-

perature definiert ist.

Jetzt stellt der Browser den echten Temperaturwert im Slider dar, sobald er definiert wurde. Wir wollen ihn an der richtigen Stelle im Slider anzeigen. Dafür definieren wir in der link-Funktion einen sogenannten watch. Watches sind dazu da, Werte zu überwachen und bei Veränderung Funktionen auszuführen. Auch die Veränderung von undefiniert zu definiert wird von den Watches wahrgenommen, was uns an dieser Stelle hilft. Der $scope.$watch berechnet die Position der echten Temperatur im Slider und stellt sie dem Direktive-Template auf $scope.currentY zur Verfügung. Im Direktiven-Template setzen wir in unserem div mit der echten Temperatur das style-Attribut: style="top: {{currentY}}px;". Im CSS haben wir dem div schon ein position: absolute; gegeben, sodass eine Veränderung der top-Eigenschaft zu Positionsveränderungen führt.

Der Browser zeigt jetzt die echte Temperatur an der richtigen Position im Slider an. Zum Schluss bauen wir noch die Animation ein. Hierzu nutzen wir die Bibliothek animate.css, die uns verschiedene Animationsmöglichkeiten bereitstellt. Wir fügen in unserem Direktiven-Template unserem div mit der echten Temperatur die animated-Klasse hinzu, um animate.css zu sagen, dass dieses Element animiert werden soll. Des Weiteren bekommt das div {{animation}} als weitere Klasse gesetzt ¿¿und erhält dadurch die Klasse, die gleich auf $scope.animation setzen werden {{ok?}}. Zuletzt setzt unser watch zusätzlich $scope.animation = "bounceInDown";. Als Ergebnis hat das div die Klassen animated bounceInDown und animiert das Element korrekt, sodass wir im Browser einen an die richtige Position fallenden Temperatur-Wert sehen.

Reset-Button einbauen

9 Vielleicht will der User sein Glück noch einmal versuchen, so das wir jetzt noch einen Reset-Knopf einbauen. Im Template fügen wir dem div, das den Ergebnissatz anzeigt noch einen Button hinzu. Der Button erhält das Attribut ng-click="reset()". Im Controller definieren wir die $scope.reset-Funktion, die alle Variablen zurück auf ihre Startwerte stellt. Da AngularJS nach einer Nutzeraktion immer die Templates neu berechnet, brauchen wir an dieser Stelle nichts weiter zu unternehmen. AngularJS hat die Benutzeroberfläche für uns schon aktualisiert.

Reale Temperaturdaten einbinden

10 Fast fertig! Wir müssen nur noch die von uns festgelegten Werte durch dynamische, echte Werte ersetzen. Wir haben die Funktionen für die Positionsbestimmung und das Abrufen der aktuellen Wetterdaten in sogenannten Factories definiert. Factories bieten uns die Möglichkeit komplexere Logik aus Controllern und Direktiven zu extrahieren und sie gegebenenfalls auch an anderen Stellen unserer Anwendung wieder zu verwenden. Wir überspringen hier das Abrufen der aktuellen Wetterdaten aus dem Internet, da dies Standard-AJAX-Aufrufe sind. Das Feststellen der aktuellen Position ist interessanter, da wir mit Ionic beziehungsweise mit PhoneGap direkten Zugriff auf die Hardware des Devices bekommen.

Mit $window.navigtor.geolocation.getCurrentPosition() können wir die aktuellen Koordinaten des Devices abfragen. In unserem Wetterdaten-Abruf verwenden wir diese Koordinaten, um vom Wetterdienst die Wetterdaten unserer Postion abfragen zu können. Die Abfrage der Positionsdaten und der Wetterdaten führen wir beim Start der App durch, damit der Nutzer mit seiner Schätzung starten kann, sobald die Zahlen vorliegen. Dazu fügen wir in der »controllers.js«-Datei den Aufruf Weather.getForecastForMyPosition() ein. Direkt nach dem App-Start stehen uns die Geolokalisierungsfunktionen noch nicht zur Verfügung. Hier hilft uns Ionic mit der $ionicPlatform.ready-Funktion, die unseren Geolocation-Code ausführt, sobald das Device bereit dafür ist.

```
function gotForecast(forecast) {
    $scope.forecast = forecast;
    $scope.guessedTemperature = angular.copy(forecast.
    temperature);
}
$ionicPlatform.ready(function() {
    Weather.getForecastForMyPosition().then(gotForecast);
});
```

Design und Veröffentlichung

11 Fertig! Nun fehlt noch das Design, dann können wir unsere App veröffentlichen. Wir wollen das Interface von ChillSkill möglichst simpel halten und entscheiden uns daher für inverses Farbschema, das auch in heller Umgebung gut lesbar ist. Ionic hat ein eigenes dunkles Thema im Gepäck – perfekt. Flat Design ist ein starker Trend, aber deswegen muss man nicht auf Textur verzichten – mithilfe des CSS-Pattern-Generators Patternify (www.patternify.com) erstellen wir unsere eigene, vom Retro-Stil inspirierte Textur. Ionics Standardschriftarten sind Helvetica und Arial. Wir geben nur dem Temperaturwert eine eigene, etwas feiner geschnittene Typo, die Open Sans Condensed, und binden sie im Header der »in-

Benjamin Behr und Benjamin Rabe entwickelten ihre App im Café des Hamburger Maker Hubs und testeten Sie gleich an den Cafégästen. Die meisten lagen beim Schätzen der Temperatur weit daneben, d das Feedback war durchgehend positiv. ↗http://digitalbehr.de, ↗www.nonuts.de

dex.html«-Datei ein.

Ionic generiert automatisch ein Xcode-Projekt, in das sich Icons und Splash-Screens integrieren lassen. In Xcode kann man dann unter »Product > Archive« das App-Package generieren und anschließend an Apple zur Review schicken. Unsere App steht inzwischen als MVP (Minimum Viable Product) bereit – wir haben uns eine solide Basis geschaffen, auf der sich nach Feedback erster Nutzer aufbauen lässt. Ab sofort heißt es rinse and repeat, und dabei hilft uns Ionic auch, wenn das Projekt komplexer werden sollte. *Benjamin Behr & Benjamin Rabe (fb)*