Versuch:

Home-Control-App für iOS und Android

HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES
INFORMATIONSTECHNIK & ELEKTROTECHNIK

Studiengänge:

- Informationssystemtechnik/ Multimediatechnik/ Elektrotechnik

Ausbildungsziel:

- Kennen lernen des Titanium Frameworks
- Kennen lernen des prinzipiellen Vorgehens zum Programmieren einer einfachen Smartphone-App

Ausbildungsinhalte:

- Kennen lernen einer Server-Client-Architektur
- Nutzung des Anwendungsprotokolls HTTP(S) zur Kommunikation zwischen Server und Client
- Kennen lernen der Sprachen Javascript, HTML5, CSS
- Kennen lernen der Entwicklungsumgebung “Titanium Mobile”

Gerätetechnik:

- 1 Raspberry Pi
- 1 433MHz-Sender
- 1 Temperatursensor DS18B20
- 1 Rechner mit der Entwicklungsumgebung Titanium-SDK

Vorkenntnisse:

- Kenntnisse im Bereich Grundlagen der KT

Beispiel-App

- Web-App: http://www.staff.hs-mittweida.de/~rthomane/HIT/app/
- Android-App: http://www.staff.hs-mittweida.de/~rthomane/praktikumkt/raspberry/SmartHome.apk

Versuchsumfeld

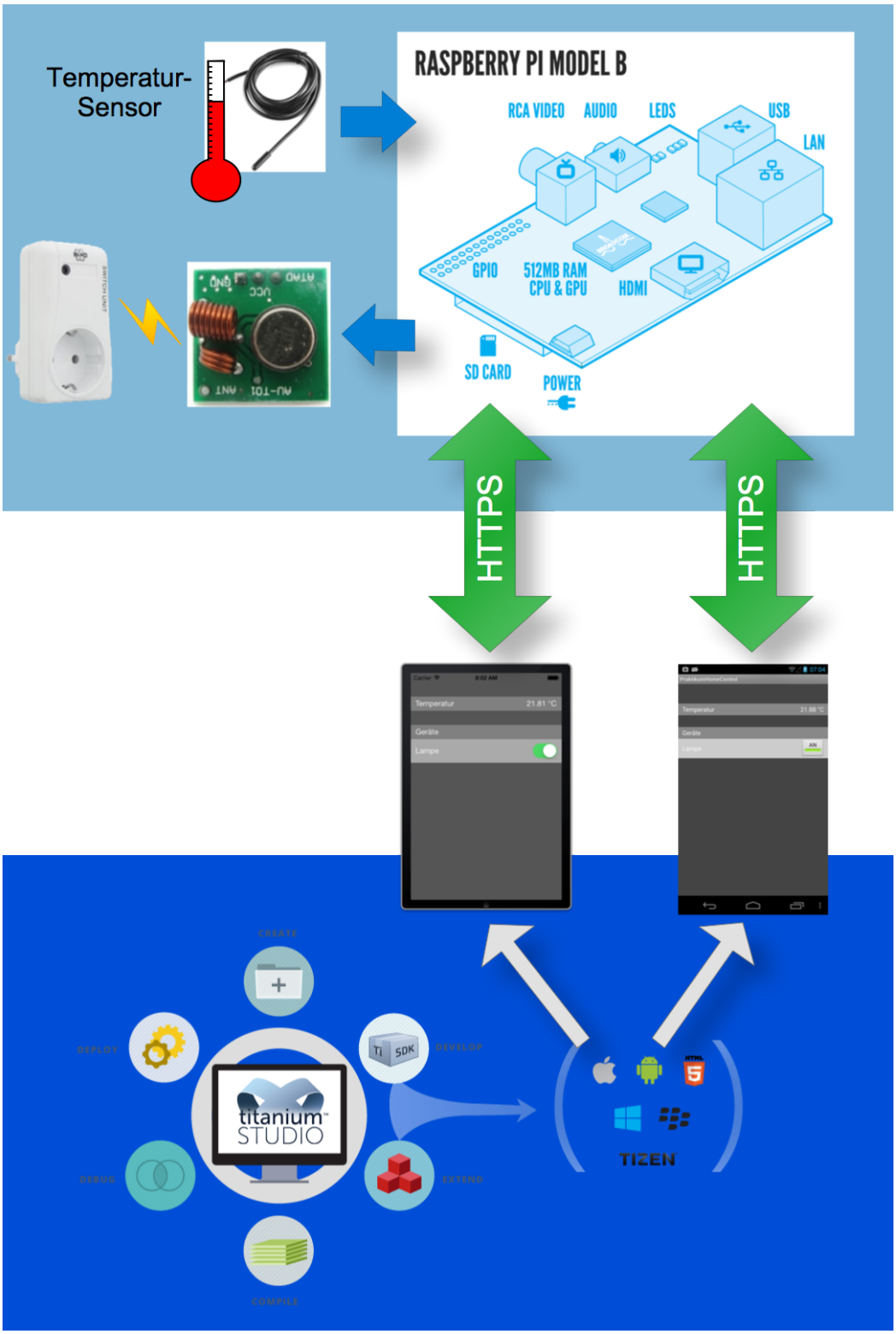


Abbildung 1: Versuchsumfeld

Einführung

Die Verbreitung von Smartphones und Tablets hat in den letzten Jahren stark zugenommen. Aufgrund des hohen Mobilitätsfaktors dieser Geräte, sind sie auch hervorragend für die Steuerung und Überwachung von Heimgeräten geeignet. Um dies zu realisieren, wird jedoch eine App auf dem Endgerät benötigt, welche die Kommunikation mit dem häuslichen Steuergerät (z.B. Raspberry Pi) übernimmt.

Ziel dieses Versuches ist es daher, eine einfache Home-Control-App zu entwerfen, welche die aktuelle Raumtemperatur anzeigt und das Schalten von Endgeräten (z.B. Beleuchtung, Unterhaltungselektronik, usw.) ermöglicht. Um bei Wahl der Zielplattform (iOS, Android,...) unabhängig zu sein, soll die Applikation mit dem quelloffenen Framework „Titanium Mobile“ entwickelt werden.

Bei Titanium Mobile handelt es sich um ein Framework, welches die nativen APIs (Application Programming Interface) der mobilen Betriebssysteme in einer einheitlichen Abstraktionsschicht kapselt. Hierdurch erfolgt die Programmierung der App nicht mit den Programmiersprachen für die mobilen Endgeräte (z.B. Objective-C, Java, .NET,...), sondern mittels JavaScript, HTML5 und CSS. Optional ist auch die Verwendung von PHP, Ruby und Python möglich. Der große Vorteil von Titanium liegt dabei darin, dass eine App nur einmal programmiert werden muss. Anschließend können dann die nativen Apps für die gewünschten mobilen Betriebssysteme kompiliert werden.

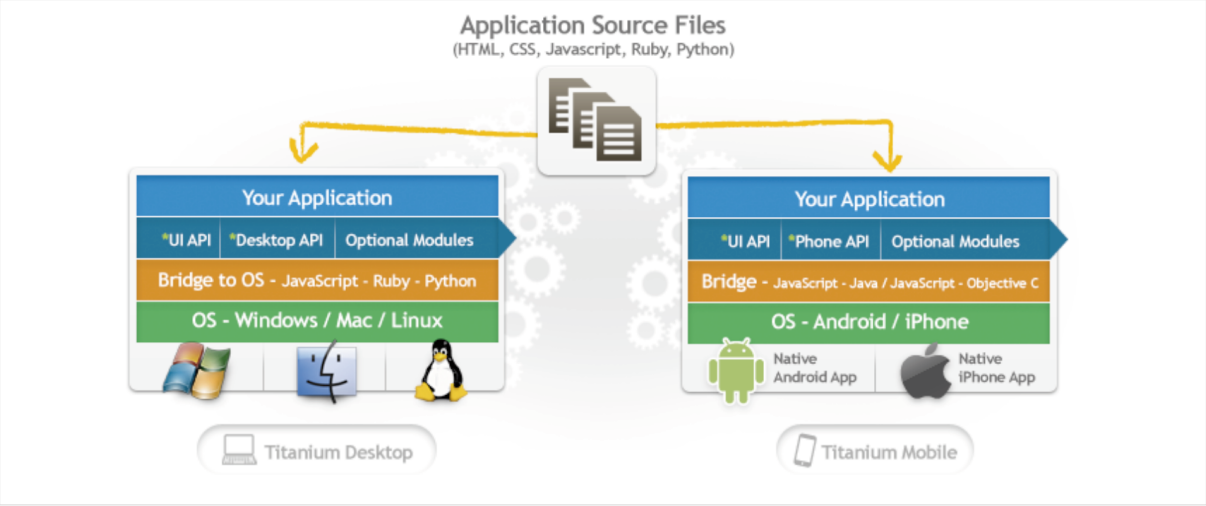


Abbildung 2: Das Titanium-Framework

Einen guten Einstieg in die Programmierung mit Titanium findet man unter http://docs.appcelerator.com/titanium/latest/ wo eine vollständige API-Dokumentation mit vielen Quellcode-Beispielen zu finden ist. Darüber hinaus befindet sich im Primo Bibliotheksportal der Hochschulbibliothek auch eine online Ressource „Titanium Mobile – Der schnelle Einstieg in die App-Programmierung“.

Versuchsdurchführung

Ein neues App-Projekt erzeugen

Die zu entwickelnde App, soll nach dem objektorientierten MVC-Architekturmuster (Model-View-Controller) entwickelt werden. MVC hat zum Ziel die Interaktionen des Nutzers, von den dadurch veränderten Daten und deren Darstellung zu trennen. Der Controller ist für die Verarbeitung der Nutzereingaben und die Kommunikation mit dem Modell zuständig. Das Modell stellt dabei die Datenquelle (z.B. Datenbank) dar. MVC unterteilt somit das Aussehen der Oberflächen-Elemente (View) von ihrem Verhalten (Controller). Ein Button hat somit keine eigene Bedeutung, sondern er ist nur eine Schaltfläche, die der Nutzer betätigen kann. Das Verhalten des Buttons wird stattdessen vom Controller gesteuert.

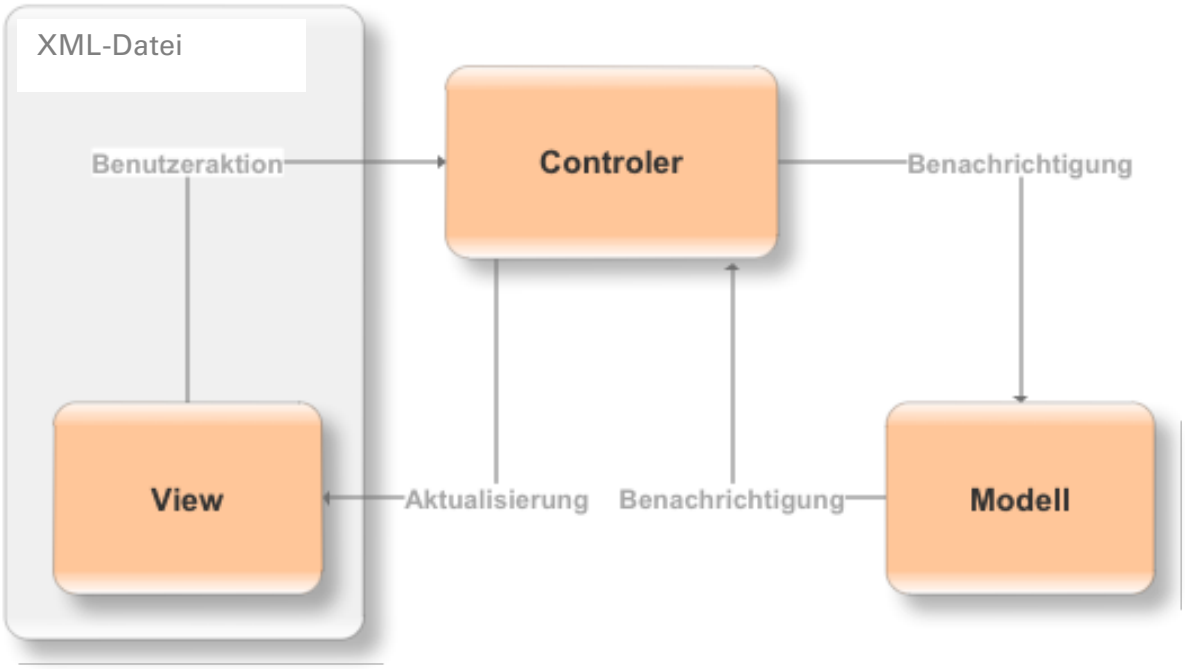
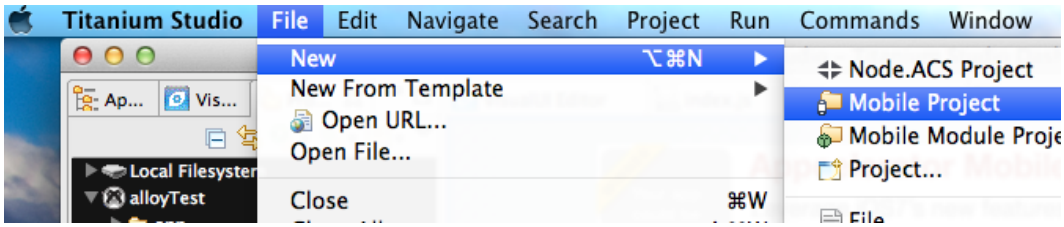
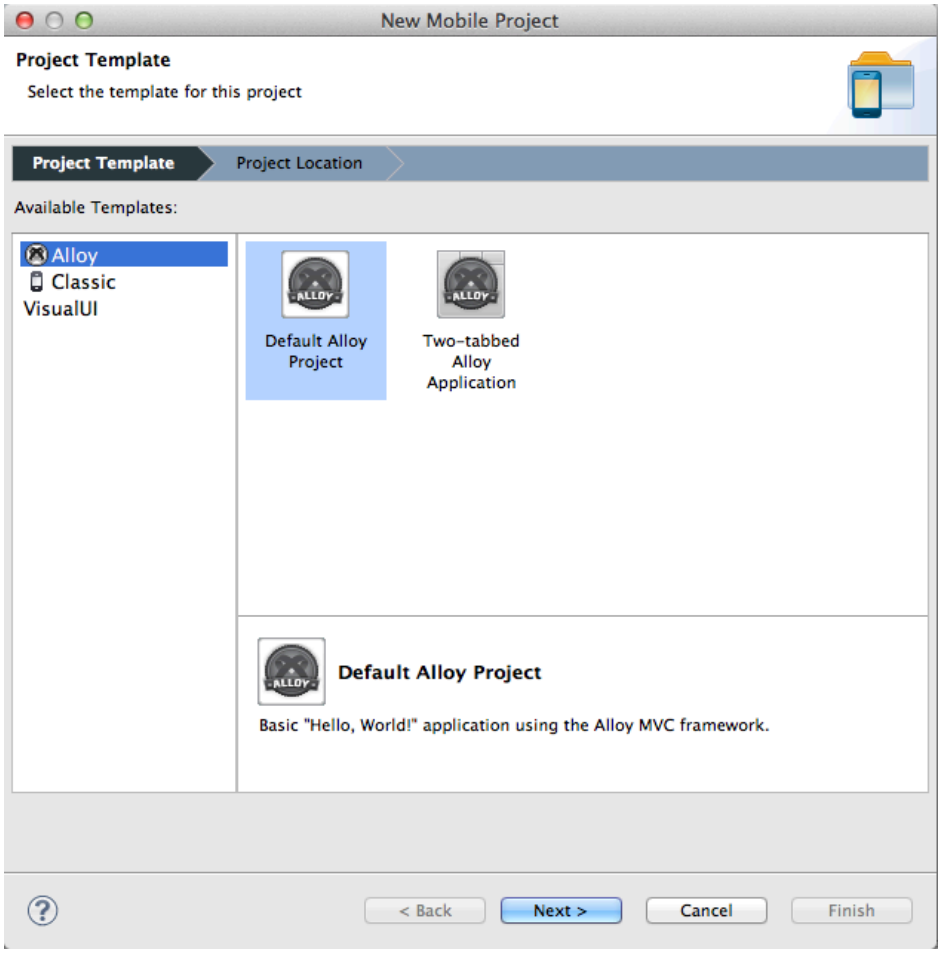


Abbildung 3: MVC-Architekturmuster bei Titanium Mobile

1. Starten Sie die Entwicklungsumgebung „Titanium Studio“!
2. Gehen Sie in das Menü und wählen Sie unter „File→New→Mobile Project“ aus!



3. Wählen Sie im erscheinenden Dialog das Template „Alloy → Default Alloy Project“ aus und klicken Sie anschließend auf „Next“!



4. Geben Sie folgende Projektdaten an und klicken Sie anschließend auf „Finish“!
   Project name: **HomeControl**
   App Id: **de.hsmw.eit.homeControl**
   Company/Personal URL: **http://www.eit.hsmw.de**

Titanium SDK Version: **höchste Versionsnummer angeben**

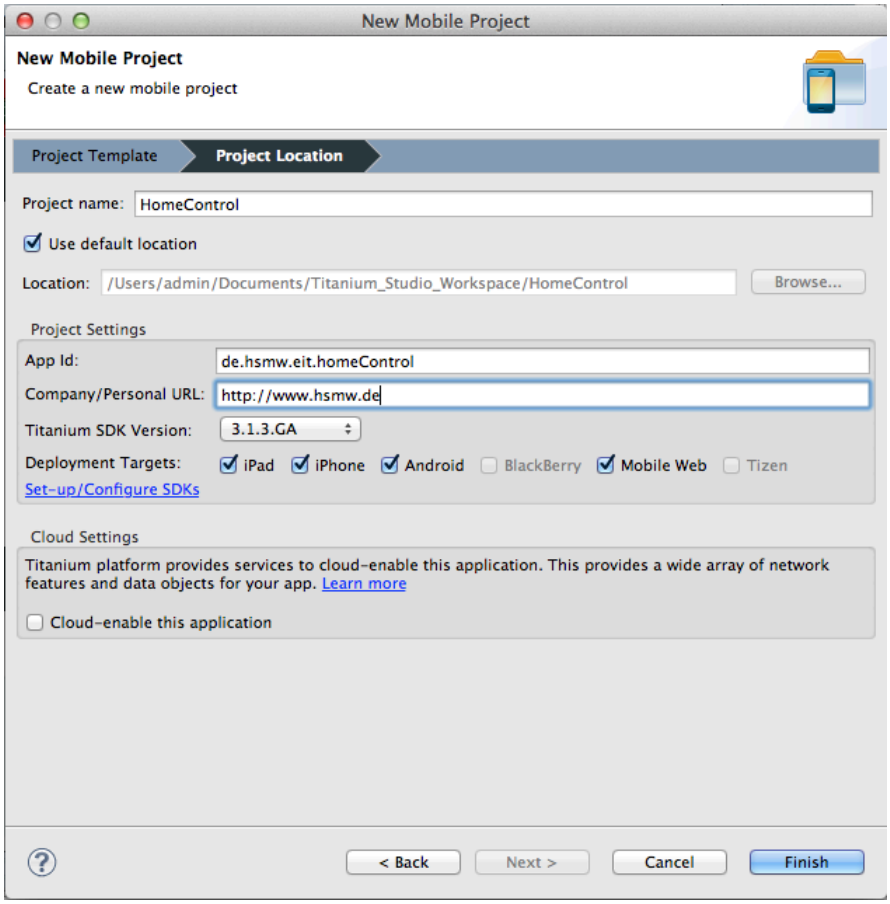


Abbildung 4: Projekteinstellungen

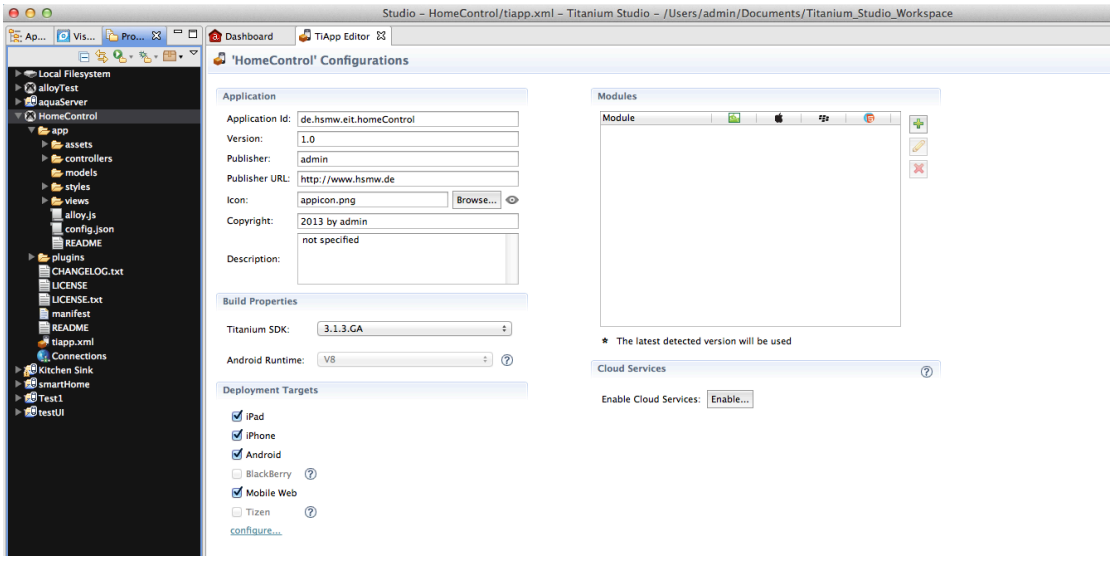


Abbildung 5: Neu erzeugtes Projekt

Erzeugte App ausführen

Nach Erzeugung des Projektes, steht bereits eine ausführbare App zur Verfügung. Führen Sie daher die App für einen ersten Funktionstest im iOS- und Android-Simulator aus!

1. Gehen Sie im Project Explorer auf das Icon: 
2. Wählen Sie „Android Emulator“ und warten Sie, bis die App im Emulator angezeigt wird!
   → Sollten keine Fehler vorhanden sein, sollte „Hello, World“ angezeigt werden. Wenn Sie auf den Text „Hello, World“ klicken, wird eine Nachricht angezeigt.

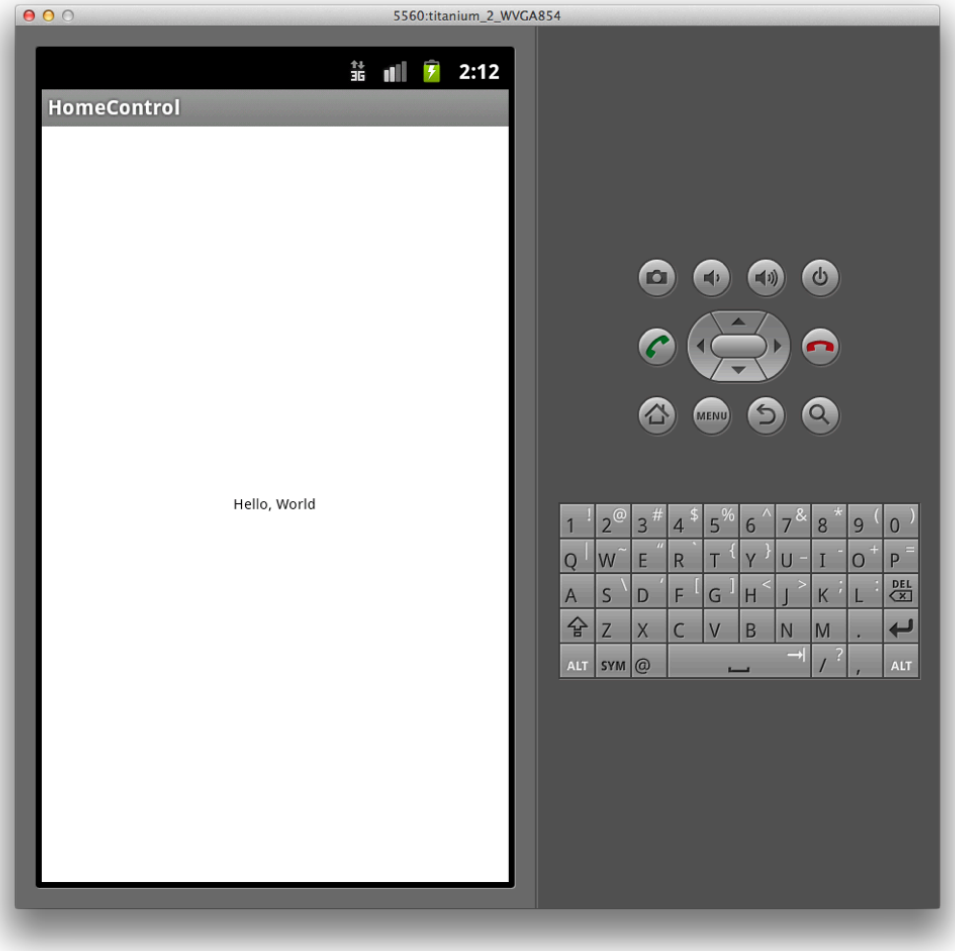
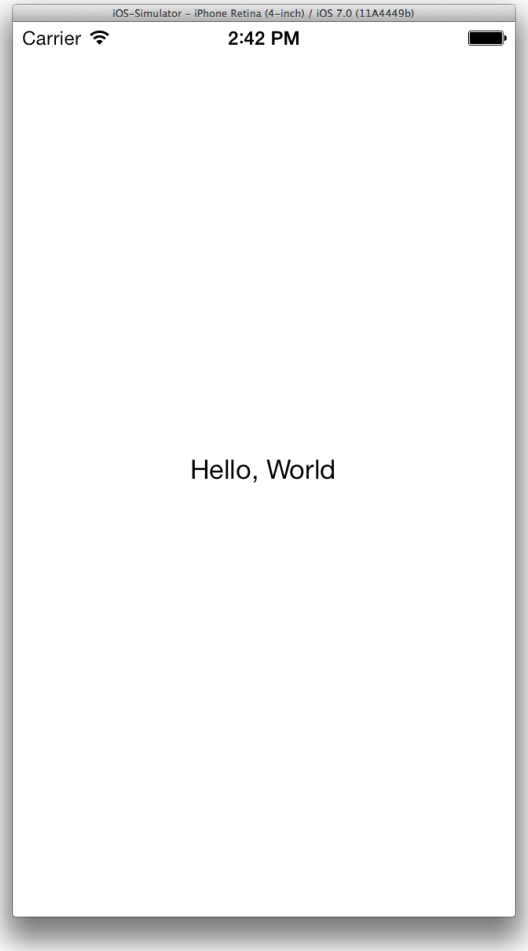


Abbildung 6: Geöffnete App im Android-Emulator

3. Beenden Sie die App durch Betätigung des Buttons „Terminate“  , welchen Sie in Titanium unterhalb des Editors finden!
4. Gehen Sie im Project Explorer auf das Icon: 
5. Wählen Sie „iPhone Simulator“ und warten Sie, bis die App im Simulator angezeigt wird!
   → Sollten keine Fehler vorhanden sein, sollte „Hello, World“ angezeigt werden.

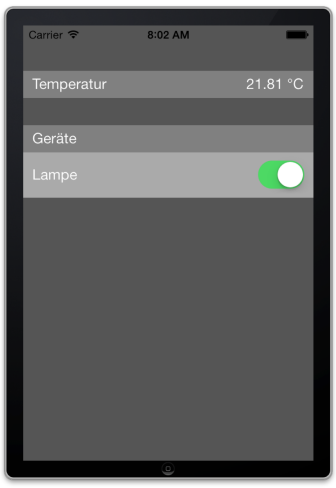
Wenn Sie auf den Text „Hello, World“ klicken, wird eine Nachricht angezeigt.



6. Beenden Sie die App durch Betätigung des Buttons „Terminate“  , welchen Sie in Titanium unterhalb des Editors finden!

Oberfläche der App erzeugen

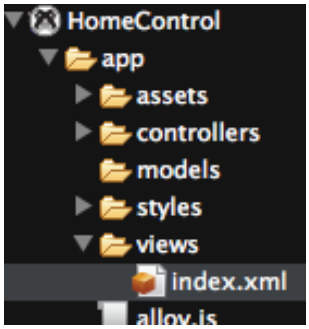
Die zu erzeugenden App soll wie in folgender Abbildung ersichtlich aufgebaut sein.



Zur Darstellung der Texte „Temperatur“, „Geräte“, „Lampe“ und „Temperaturwert“ soll ein Label zum Einsatz kommen. Zum Schalten der Funksteckdose soll ein Switch verwendet werden.

Die Beschreibung der Oberfläche erfolgt in Titanium innerhalb der `index.xml`-Datei, die über die Ordnerstruktur `app` → `views` zu finden ist.

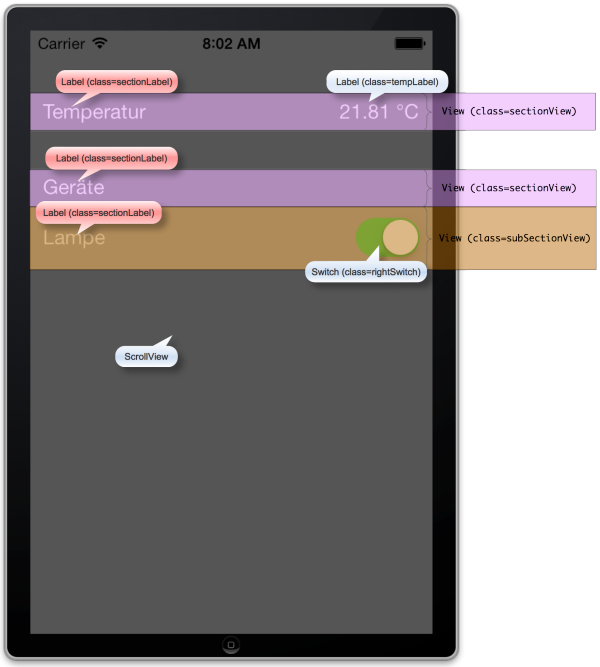
1. Öffnen Sie die Datei `index.xml` durch Doppelklick im Editor!



2. Innerhalb von `index.xml` finden Sie folgenden Inhalt:

```
<Alloy>
  <Window class="container">
    <Label id="label" onClick="doClick">Hello, World</Label>
  </Window>
</Alloy>
```

→ Entfernen Sie die Zeile 3 (`<label id="label" onClick="doClick">...`)!



Um eine einfache Positionierung der einzelnen Elemente zu erzielen, sollen zusammengehörige Oberflächenelemente jeweils durch eine View umschlossen sein. Eine View ist dabei ein Rechteck, das andere Elemente aufnehmen kann. Eine View ist standardmäßig nicht sichtbar, was aber über Eigenschaftszuweisungen geändert werden kann, z.B. Hintergrundfarbe ändern.
Die nebenstehende Abbildung soll die zu implementierenden Oberflächeelemente verdeutlichen.
Dabei sollen die dargestellten View's jeweils die Label-Elemente bzw. den Switch aufnehmen.

3. Beschreiben Sie anhand der dargestellten Abbildung das Aussehen der Oberfläche innerhalb der Datei `index.xml`!
   → Nutzen Sie für die Oberflächenelemente folgende Tags:

<ScrollView></ScrollView>
<View></View>
<Label></Label>
<Switch></Switch>
<ActivityIndicator></ActivityIndicator>

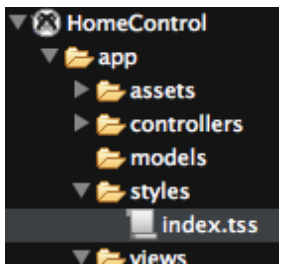
→Somit ergibt sich:

```
<Alloy>
    <Window class="container">
        <ScrollView layout="vertical">
            <View class="sectionView">
                <Label class="sectionLabel">Temperatur</Label>
                <Label id="valueTempLabel" class="tempLabel"></Label>
                <ActivityIndicator id="tempActivity" class="tempActivity"></ActivityIndicator>
            </View>
            <View class="sectionView">
                <Label class="sectionLabel">Geräte</Label>
            </View>
            <View class="subSectionView">
                <Label class="sectionLabel">Lampe</Label>
                <Switch class="rightSwitch" value="false"></Switch>
            </View>
        </ScrollView>
    </Window>
</Alloy>
```

Die in der 3. Zeile festgelegte Eigenschaft `layout="vertical"` bewirkt, dass alle View-Elemente automatisch untereinander platziert werden.

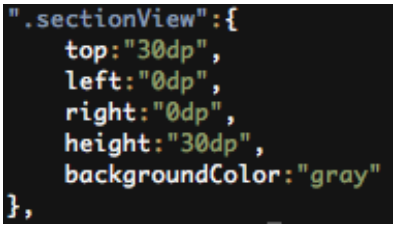
Positionierung und Design der Oberflächenelemente

Wie im vorherigen Abschnitt erläutert, werden alle View-Elemente direkt untereinander platziert. Die weitere Positionierung dieser Elemente erfolgt demzufolge relativ zur aktuellen Position. Die Wertzuweisung `top="30dp"` bewirkt z.B., dass ein Abstand zwischen dem aktuellen und dem darüber liegenden Element von 30 Punkten erzeugt wird. Bei der Einheit „dp“ (Density Pixel) handelt es sich um eine Maßangabe, bei der die Positionierung von Elementen anhand der Displayauflösung berechnet wird. Somit werden Oberflächen auf Geräten mit unterschiedlichen Auflösungen nahezu gleich dargestellt.



Generell können die Werte zur Positionierung und Design der Oberfläche auch in die Datei `index.xml` integriert werden. Zur Besseren Trennung zwischen Oberflächeninhalt und Oberflächendesign erfolgt jedoch die Beschreibung des Oberflächendesigns in der Datei `index.tss`. Die `index.tss`-Datei befindet sich in der Ordnerstruktur `app` → `styles`.

Die in der Datei `index.tss` verwendete Syntax entspricht dabei CSS (Cascading Style Sheets). Üblicherweise erfolgt die Eigenschaftszuweisung mittels Klassen- und ID-Selektoren. Die hierfür notwendigen Attribute wurden bereits in der `index.xml`-Datei definiert (z.B. class=“sectionView“).



Der nebenstehende Quellcode zeigt die CSS-Syntax zur Zuweisung aller notwendigen Eigenschaften für die Klasse „sectionView“. Wie bei CSS typisch, wird ein Klassenselektor mittels des Punktes vor dem Klassenbezeichner gekennzeichnet. Eine Besonderheit ist dabei, dass der Selektor in Anführungszeichen zu setzen ist. Die gesamte Wertzuweisung erfolgt dann zwischen den geschweiften Klammern, wobei der Selektor und die Wertzuweisungen durch einen Doppelpunkt voneinander getrennt werden. Wie im dargestellten Quellcode ersichtlich, werden für die Klasse „sectionView“ folgende Positionierungs- und Designeigenschaften definiert:

- Abstand zum oberen Element: 30 Punkte,
- Abstand zum linken Bildschirmrand: 0 Punkte → Das Element geht bis an den linken Rand,
- Abstand zum rechten Bildschirmrand: 0 Punkte,
- Höhe der View: 30 Punkte,
- Hintergrundfarbe der View: Grau.

1. Fügen Sie die Positions- und Designeigenschaften in die index.tss-Datei ein! Nutzen Sie hierfür folgende Eigenschaften:

   Klasse: „sectionLabel“
   `font:{fontSize:'16sp'},left:"10dp",color:"white"`

   Klasse: „sectionView“
   `top:"30dp",left:"0dp",right:"0dp",height:"30dp",backgroundColor:"gray"`

   Klasse: „tempLabel“
   `font:{fontSize:'16sp'},right:"10dp",color:"white"`

   Klasse: „rightSwitch“
   `right:"10dp"`

   Klasse: „subSectionView“
   `left:"0dp",right:"0dp",height:"50dp",backgroundColor:"LightGray"`

   Klasse: „tempActivity“
   `right:"10dp",backgroundColor:"gray",color:"white",message:"Loading..."`

2. Führen Sie die Applikation im iOS-Simulator und im Android-Emulator aus und prüfen Sie die Oberfläche!

Die gesamte Positionierungs- und Designbeschreibung kann dann wie folgt aussehen:

```
".container": {
    "backgroundColor": "DarkGray"
},
".sectionLabel":{
    font:{fontSize:'16sp'},
    left:"10dp",
    color:"white"
},
".tempLabel":{
    font:{fontSize:'16sp'},
    right:"10dp",
    color:"white"
},
".tempActivity":{
     right:"10dp",
     backgroundColor:"gray",
     color:"white",
     message:"Loading..."
},
".rightSwitch":{
    right:"10dp"
},
".sectionView":{
    top:"30dp",
    left:"0dp",
    right:"0dp",
    height:"30dp",
    backgroundColor:"gray"
},
".subSectionView":{
    left:"0dp",
    right:"0dp",
    height:"50dp",
    backgroundColor:"LightGray"
}
```

Abbildung 7: Inhalt der index.tss-Datei

Implementation: Steckdose schalten

Die Implementation der Funktionalität erfolgt bei Titanium innerhalb der Controller-Datei „`index.js`“. Wie die Dateiendung `.js` schon deutet, erfolgt die Programmierung mittels JavaScript. Die `index.js`-Datei ist dabei eng mit der `index.xml`-Datei verknüpft. Denn den einzelnen Oberflächenelementen kann in der index.xml-Datei ein ID-Attribut zugewiesen werden. Der Wert dieses Attributes kann dann als Variablenname innerhalb der index.js-Datei verwendet werden. Die Zuweisung von Ereignissen zu zugehörigen Methoden erfolgt ebenfalls in der index.xml-Datei. Typische Ereignisse sind beispielsweise onChange, onStart, oder onStop. Der zugewiesene Wert muss dann in der index.js-Datei als Methode deklariert werden.

Um bei Betätigung des Switches die Steckdose zu schalten, muss eine Ereignismethode und eine Variable eingerichtet werden. Die Ereignismethode soll dabei immer aufgerufen werden, wenn der Switch betätigt wird, sich demzufolge der boolesche Wert des Switches ändert. Über die Variable soll dann ermittelt werden, ob der Switch ein- oder ausgeschaltet wurde. Daraufhin müssen dann die im Versuch „Home Control mit Raspberry Pi“ definierten HTTPS-Requests gesendet werden.

1. Definieren Sie in index.xml für das Switch-Element das `id`-Attribut und weisen Sie diesem den Wert `plug_1` zu!
2. Definieren Sie in index.xml für das Switch-Element das Ereignis `onChange` und weisen Sie diesem den Wert `onOff` zu!
3. Öffnen Sie die Datei index.js!
4. Löschen Sie folgenden Inhalt in index.js:

```
function doClick(e) {
    alert($.label.text);
}
```

5. Erzeugen Sie in der ersten Zeile eine Variable host, welche die Grundparameter für die HTTP-Requests beinhalten soll. Achten Sie darauf, den **Hostname** Ihres Raspberry Pi zu verwenden!

```
var host = "https://praktikum:praktikumkt@tc-raspberry-01.eit.hs-mittweida.de";
```

6. Definieren Sie in index.js die Methode onOff, wie folgt:

```
function onOff(e){

}
```

Zum Versenden eines HTTPS-GET-Requests kann die Klasse HTTPClient genutzt werden. Hierfür ist es notwendig ein Objekt mittels der Methode „createHTTPClient()“ aus dem Titanium.Network-Framework zu erzeugen. Das erzeugte Objekt führt einen asynchronen Request aus. Dies hat den Vorteil, dass die Anwendung nicht blockiert, während Daten gesendet oder empfangen werden. Die Methode „createHTTPClient() erwartet folgende Parameter:

- Die Callback-Methoden „onload“ und „onerror“ liefern Informationen über Erfolg oder Misserfolg des Requests und können zur weiteren Verarbeitung dienen.
- Der timout-

```
function onOff(e){
    var url = "";
    var client = Ti.Network.createHTTPClient({
        onload:function(f){
            Ti.API.info("Received text: "+this.responseText);
        },
        onerror:function(f){
            Ti.API.debug(f.error);
            alert('Schalten nicht möglich!');
            if($.plug_1.value==true){
                $.plug_1.value=0;
            }else{
                $.plug_1.value=1;
            }
        },
        timeout:5000
    });
```

Parameter definiert die Zeit in Sekunden, nach der ein erfolgloser Kommunikationsversuch abgebrochen werden soll.

In Abhängigkeit des eingestellten Switch-Wertes (An/Aus) muss ein entsprechender HTTPS-GET-Request an den Raspberry Pi gesendet werden. Diese Requests sind wie folgt definiert:

- *Switch ist aus:*
  https://praktikum:praktikumkt@tc-raspberry-01.eit.hs-mittweida.de/switchOnOff.php?gpio=14&systemcode=11000&unitcode=1&onoff=0
- *Switch ist an:*
  https://praktikum:praktikumkt@tc-raspberry-01.eit.hs-mittweida.de/switchOnOff.php?gpio=14&systemcode=11000&unitcode=1&onoff=1
- Erklärung:
  - Definiert GPIO, wo 433MHz-Sender angeschlossen ist
  - Definiert den in der Steckdose eingestellten System-Code, beachten Sie, hier den System-Code Ihrer Steckdose zu nutzen
  - Definiert den in der Steckdose eingestellten Device-Code
  - Definiert den Zustand, 0=ausschalten, 1=einschalten

Der eingestellte Wert des Switches kann über die Variable „`plug_1`“ überprüft werden. Alternativ kann auch die Variable „`e`“ der onOff-Methode genutzt werden. Über den Eigenschaftswert „`value`“ kann dann geprüft werden, ober der Switch an- oder ausgeschaltet wurde. In Abhängigkeit des eingestellten Wertes soll dann in eine Variable „`url`“ der entsprechenden HTTPS-GET-Request gespeichert werden. Achten Sie darauf, als Systemcode die binäre Nummer **Ihres** Raspberry Pi zu verwenden, z.B. tc-raspberry-12.eit.hs-mittweida.de → Nummer 12= 01100. Als Unitcode wird immer 1 verwendet.

```
if(e.value==true){
    url=host+"/switchOnOff.php?gpio=14&systemcode=01100&unitcode=1&onoff=1";
}else{
    url=host+"/switchOnOff.php?gpio=14&systemcode=01100&unitcode=1&onoff=0";
}
```

Als nächstes muss der HTTPS-Request spezifiziert werden. Dies erfolgt mittels der HTTPClient-Methode „open“ und deren Übergabeparameter. Hierbei kann prinzipiell zwischen GET- und POST-Request unterschieden werden. Weiterhin muss der aufzurufende URL als String-Wert übergeben werden. Optional besteht noch die Möglichkeit, den Request synchron oder asynchron auszuführen. Bei weglassen des letzten Parameters erfolgt der Request asynchron.

- `open( String method, String url, [Boolean async] )`
  - `method: Kennzeichnet den Request-Typ „GET“ oder „POST“`
  - `url: Kennzeichnet den aufzurufenden URL`
  - `async: gibt an, ob Request asynchon erfolgen soll oder nicht`

Um den HTTPS-Request letztendlich zu senden, kann die HTTPClient-Methode „send()“ verwendet werden.

```
client.open("GET",url);
client.send();
```

7. Starten Sie die App und testen Sie die Funktionalität!

Implementation: Temperatur auslesen

Wie im Versuch „Home Control mit Raspberry Pi“ bereits vorbereitet, soll nun noch die Temperatur mittels HTTPS-GET-Request angezeigt werden. Der hierfür eingerichtete URL lautet beispielsweise:
https://tc-raspberry-01.eit.hs-mittweida.de/getTemperature.php

→Achten Sie darauf, den **Hostname** Ihres Raspberry Pi zu verwenden!

Die abgefragte Temperatur soll dann im Label „valueTempLabel“ dargestellt werden.

1. Erzeugen Sie in `index.js` eine neue Methode „`getTemperature`“!
2. Zum Senden des HTTPS-GET-Requests wird wieder ein Objekt von HTTPClient benötigt. Dabei soll bei erfolgreichen Request der Response-Wert an die Eigenschaft „text“ des Labels „valueTempLabel“ übergeben werden. Bei fehlerhaften Request soll die Eigenschaft „text“ des Labels „valueTempLabel“ mit dem Inhalt „- °C“ versehen werden!

```
var getTemperature = function(){
    var url = host+"/getTemperature.php";
    var client = Ti.Network.createHTTPClient({
        onload:function(e){
            $.valueTempLabel.text=this.responseText+" °C";
        },
        onerror:function(e){
            Ti.API.debug(e.error);
            $.valueTempLabel.text="- °C";
        },
        timeout:5000
    });
    $.valueTempLabel.text="";
    client.open("GET",url);
    client.send();
};
```

3. Rufen Sie die Methode „getTemperature“ bei Start der App auf. Fügen Sie hierfür folgenden Quellcode vor die Zeile „`$.index.open();`“!
   `getTemperature();`
4. Starten Sie die App und testen Sie die Funktionalität!

Mit der bisherigen Implementation wird die Temperatur allerdings nur einmalig beim Start der App geladen. Wünschenswert wäre eine zyklische Aktualisierung der Temperaturanzeige. Hierfür wird die Implementation eines Timers benötigt. Timer können in JavaScript mittels folgender Methode implementiert werden:

setInterval(`Function function, Number delay`) : Number

Wie ersichtlich, benötigt die Methode „`setInterval`“ zwei Übergabeparameter. Der Parameter `function` definiert dabei die Methode, die zyklisch aufgerufen werden soll. Der Parameter `delay` gibt dann die Wiederholungsdauer in Millisekunden an.

5. Starten Sie nach der Zeile „getTemperature();" einen Timer, der aller 60 Sekunden die Methode „getTemperature();“ aufruft!

```
getTemperature();
var tempTimer = setInterval(getTemperature, 60000);
```

Wünschenswert wäre weiterhin noch eine optische Hervorhebung, wenn die Temperatur aktualisiert wird. Hierfür kann beispielsweise ein Activity Indicator verwendet werden. Bei einem Activity Indicator handelt es sich um eine kleine Animation, die eine Hintergrundaktivität symbolisieren soll.

In der Datei `index.xml` ist bereits ein Activity Indicator mit der ID „`tempActivity`“ angelegt wurden. Über die ID kann in der Datei `index.js` dann die Sichtbarkeit des Activity Indicators gesteuert werden. Dabei soll mit Begin der Temperaturaktualisierung der Activity Indicator angezeigt werden. Dies erfolgt mittels der Methode „`show()`“. Nach erfolgreicher aber auch erfolgloser Temperaturaktualisierung soll der Activity Indicator wieder ausgeblendet werden. Dies erfolgt mittels der Methode „`hide()`“.

6. Fügen Sie vor der HTTPClient-Methde „send()“ den Quellcode zur Anzeige des Activity Indicators ein!

```
$.tempActivity.show();
```

7. Blenden Sie den Activity Indicator anschließend nach erfolgreicher bzw. erfolgloser Aktualisierung der Temperatur wieder aus! Fügen Sie hierfür in die Methoden für onload und onerror folgenden Quellcode ein!

```
$.tempActivity.hide();
```

8. Testen Sie die Funktionalität der App!