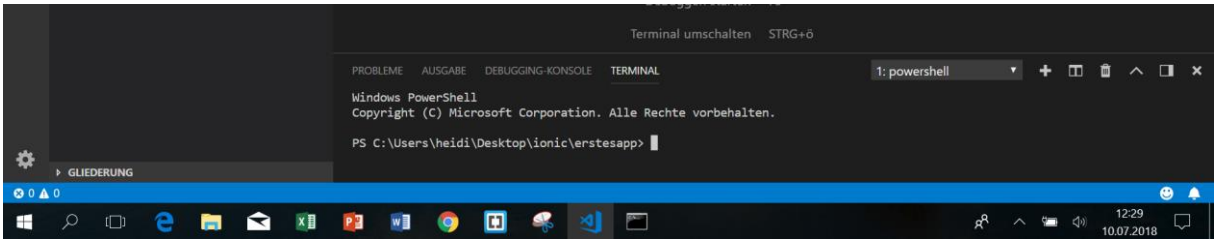
# Ionic 2-Teil – bereits mit der IDE Visual Studio Code

**Tipp:**

Wenn man Microsoft Visual Studio verwendet, kann man direkt darin auf die Eingabeaufforderung (CMD) zugreifen. Das erspart das Umschalten.

1.Variante: Ganz unten platziere den Cursor so, dass der Doppelpfeil erscheint. Dann ziehe ihn nach oben und es wird ein neuer Bereich mithinauf gezogen.

Klicke auf „Terminal“




2.Variante: Im Menü „Anzeigen“ wähle „integriertes Terminal“.

## 1)neue Seiten anlegen

Öffne die „index.html“ in „www-Ordner“.

Nur zur Info:

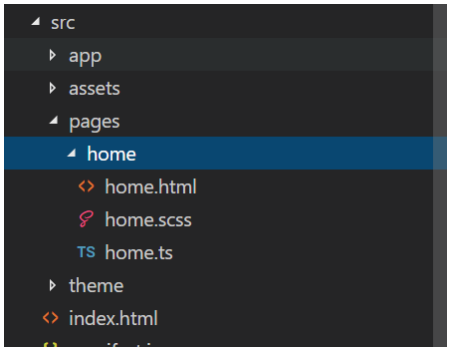
```
36  <body>
37
38    <!-- Ionic's root component and where the app will load -->
39    <ion-app></ion-app>
40
41    <!-- The polyfills js is generated during the build process -->
42    <script src="build/polyfills.js"></script>
```

In Zeile 39 sind zwischen den <ion-app> noch keine Seiten eingetragen. Hier folgen später die einzelnen Seiten der App, von wo sie dann geladen werden.

### 1a)Löschen der „home.html“,

damit man sie danach zwei neue Sites erstellt, die aber bereits die passenden „module-Erweiterungen“ aufweisen.

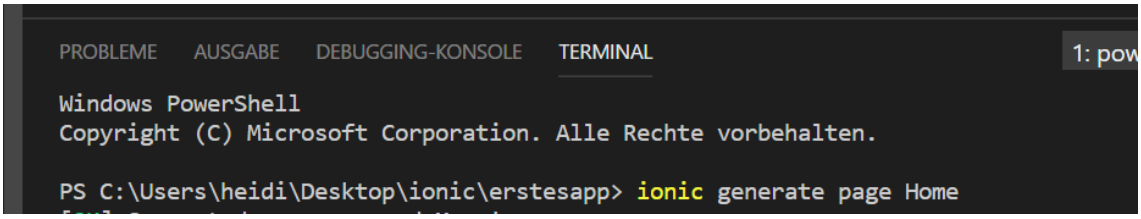
Im Ordner „pages“ lösche den „home-Bereich“ inkl. Allen drei enthaltenen Dateien.



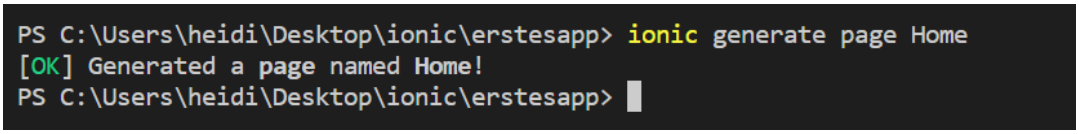
**1b)Neue „page“ erstellen.**

Dazu nutze das integrierte Terminal und gib ein zum Generieren:

- ionic generate page Home



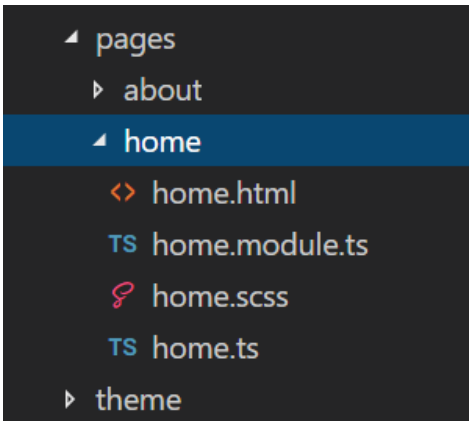

Ergebnis:




Gleich eine zweite Site auch

- ionic generate page About

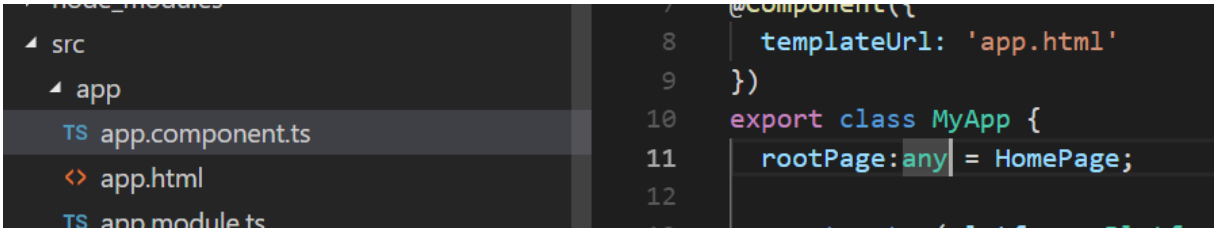
Ergebnis:




Man sieht den Vorteil, da nun auch die neue Datei home.module.ts dazugekommen ist. Man hat sich das selber Anlegen dieser „module“ erspart.

**1c)Laden der einzelnen Seiten über „lacy-loading“**

Dazu muss in der Datei „app.components.ts“ im Ordner „app“ in der Zeile 11 das „any“ durch „string“ ersetzt werden. Und „Homepage“ kommt in Anführungszeichen.




Ergebnis:

```
10  export class MyApp {
11    rootPage:string = "HomePage";
```

Damit werden die Seiten als Zeichenkette „string“ übergeben.

Daher kann auch die Zeile 6 komplett gelöscht werden:

```
4   import { SplashScreen } from '@ionic-native/splash-scre
5
6   import { HomePage } from '../pages/home/home';
7   @Component({
```

Speichern und Schließen.

Zusätzlich öffne das „app.module.ts“

Entferne hier komplett die Zeile 8 (import….) und das Wort „Homepage“ in den Zeien 13 (bei den declarations) und 23 (bei entryComponents).

Ergebnis:

```
5   import { StatusBar } from '@ionic-native/status-bar
6
7   import { MyApp } from './app.component';
8
9
10  @NgModule({
11    declarations: [
12      MyApp,
13
14    ],
15    imports: [
16      BrowserModule,
17      IonicModule.forRoot(MyApp)
18    ],
19    bootstrap: [IonicApp],
20    entryComponents: [
21      MyApp,
22
23    ],
```

Speichern und Schließen.

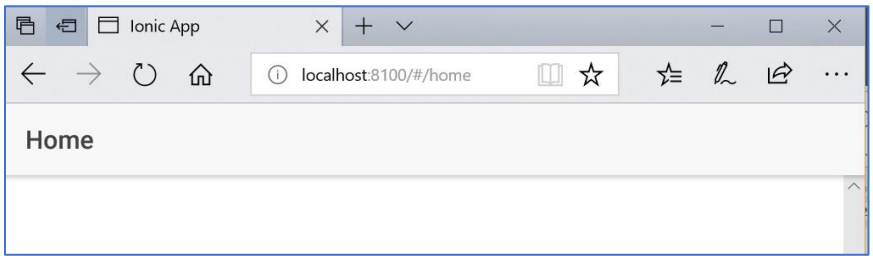
## 2)Seite betrachten

Gib im Terminal ein

- ionic serve

```
PS C:\Users\heidi\Desktop\ionic\erstesapp> ionic serve
```

Ergebnis:




Die Seite im Hintergrund ist die „home.html“.

Öffne diese.

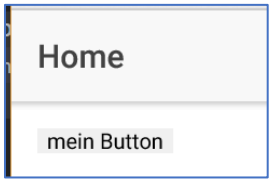
home.html

```
<!--
  Generated template for the HomePage page.

  See http://ionicframework.com/docs/components/#navigation for more info on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Home</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>

</ion-content>
```

### 2a)Button im <ion-content> erstellen

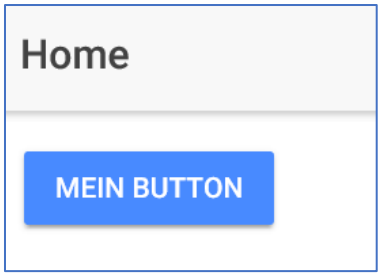
```
<ion-content padding>

  <button>mein Button</button>

</ion-content>
```

Ergebnis:



Das geht schöner:

Mit dem Zusatz „ion-button“

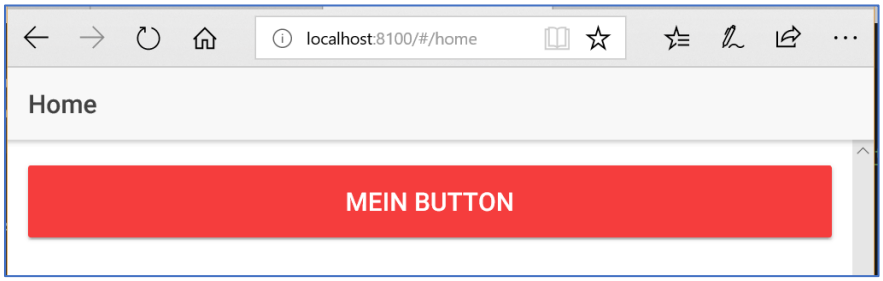



Weitere Änderungen:

Button über de ganze Breite, so groß wie möglich und in roter Farbe:

```
18    <button ion-button block large color="danger">mein Button</button>
```

Ergebnis:




**2b)Eine Ausgabe erstellen, wenn der Button gedrückt wurde**

Öffne die „home.ts“.

Dafür soll unter de „constructor“ eine neue Funktion erstellt werden.

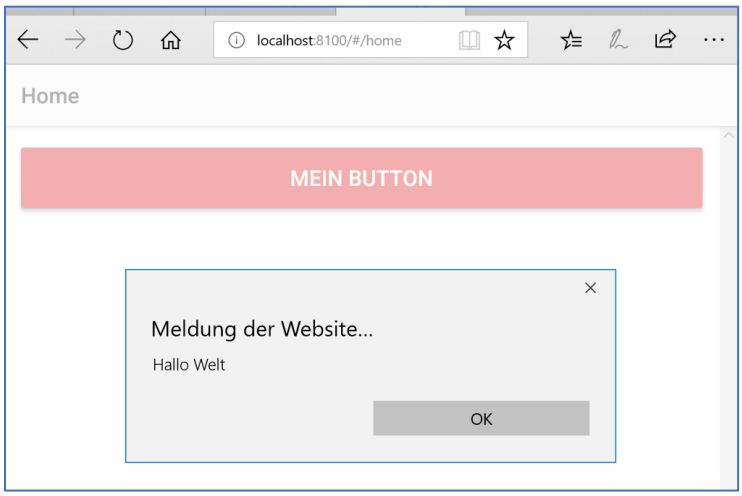
```
21    hello(){
22      alert("Hallo Welt");
23    }
```

Der Zugriff darauf geht über de „home.html“:

Im Button muss die Klick-Möglichkeit auf die Funktion eingefügt werden:

```
16  <ion-content padding>
17
18    <button ion-button block large color="danger" (click)="hello()">mein Button</button>
19
20  </ion-content>
```

Ergebnis:


localhost:8100/#/home
Home
MEIN BUTTON
Meldung der Website...
Hallo Welt
OK

## 3)Eingaben verarbeiten

**Ziel: Text in Textfelder eingeben und dann ausgeben lassen.**

**3a)Zuerst soll statt dem „alert“ von oben ein richtiger „AlertController“ entwickelt werden.**

Arbeite im „home.ts“:

Der „constructor“ in der Zeile 18 muss einen neuen Konstruktor erhalten. Dabei soll der bereits vorhandene „AlertController“ geholt werden. Bei der Schreibweise für den Namen kann man sich an den bereits vorhandenen orientieren, z.B. „navCtrl: NavCotnroller“ , daher

```
18    constructor(public navCtrl: NavController,
19      public navParams: NavParams, private alertCtrl: AlertController) {
20    }
```

In der Zeile 2 wurde der AlertController automatisch dazugefügt, da im Hintergrund der „autoimport“ funktioniert.

Nun muss die „hello-Funktion“ geändert werde:. Das „create“ zum Erstellen und das „present“ für die Anzeige. Im „create“ und den geschweiften Klammern kommt das hinein, was übergeben werden soll, hier die „message“.

```
21
22    hello(){
23      this.alertCtrl.create({
24        "message": "Hallo Welt"
25      }).present();
26    }
```

Info: Damit man sieht, was überhaupt alles übergeben werden kann, drücke die Taste STRG und klicke auf „create“. Da es mit dem „alertCztl“ verbunden ist, zeigt es die richtigen Elemente:
Es wird, wie hier in Zeile 225, angezeigt:

```
223         * @param {AlertOptions} opts Alert. S
224         */
225        create(opts?: AlertOptions): Alert;
226    }
```

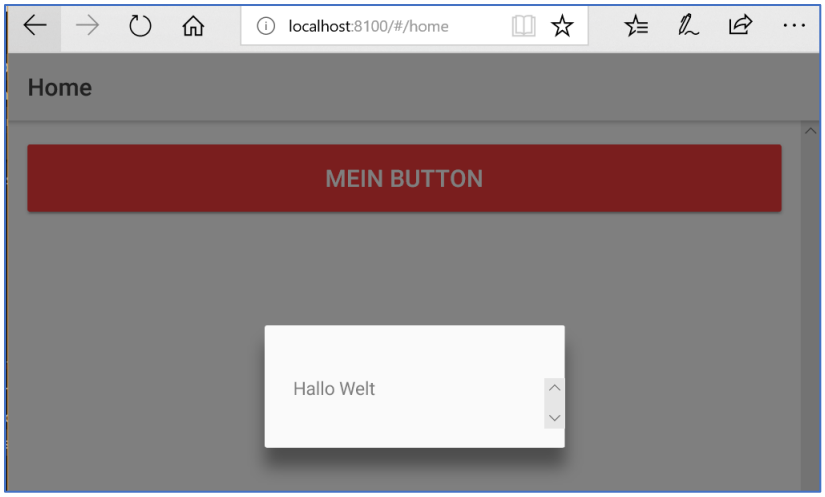
Hier wieder mit gedrückter STRG-Taste auf „AlertOptions“ klicken um die Optionen zu sehen:

In diesem Fall wären dies: z.B. title, subTitle, message usw.

home.ts | alert-options.d.ts | home.html

```
1   export interface AlertOptions {
2       title?: string;
3       subTitle?: string;
4       message?: string;
5       cssClass?: string;
6       mode?: string;
7       inputs?: AlertInputOptions[];
8       buttons?: (AlertButton | string)[];
9       enableBackdropDismiss?: boolean;
```

Ergebnis:




**3b)Variablen erstellen in „home.ts“**

Der String der Variablen wird auf einen leeren String gesetzt, d.h. es ist nichts drinnen.

```
18    firstName: string = "";
19    lastName: string = "";
20
21    constructor(public navCtrl: NavCo
```

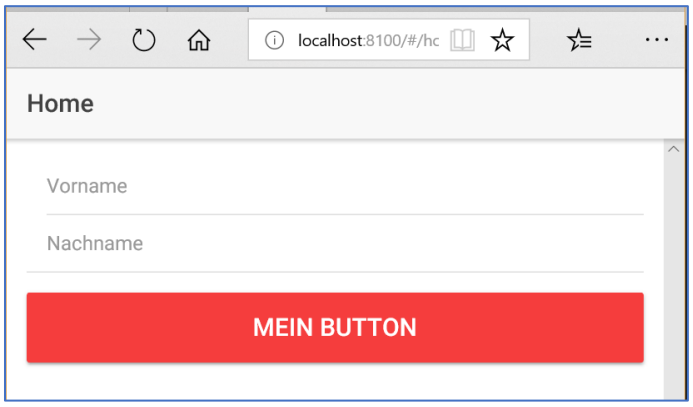
Die Felder werden nun in „home.html“ erstellt. Dies in einer „ion-Liste“ mit sogenannten <ion-item> darinnen. Diese beinhalten ein entsprechendes „label“ und „input“:

```
16  <ion-content padding>
17
18    <ion-list>
19      <ion-item>
20        <ion-label></ion-label>
21        <ion-input></ion-input>
22      </ion-item>
23
24    </ion-list>
```

Kopiere dies einmal und füge es darunter ein. Mit den Daten wird es befüllt:

```
<ion-list>
  <ion-item>
    <ion-label>Vorname</ion-label>
    <ion-input type="text"></ion-input>
  </ion-item>
  <ion-item>
    <ion-label>Nachname</ion-label>
    <ion-input type="text"></ion-input>
  </ion-item>
```

Ergebnis:




Verschönern mit „floating“ beim Label. Das erzeugt ein dynamisches Hochklappen des Labels bei der Eingabe.

```
    <ion-label floating>Vorname</ion-label>
    <ion-input type="text"></ion-input>
  </ion-item>
  <ion-item>
    <ion-label floating>Nachname</ion-label>
```

**3c)Übergabe der Inhalte des Textfeldes:**

Dazu muss beim input die Variable übergeben werden: In eckigen und runden Klammern „ngModel“ und das mit der Variable gleichsetzten.

```
<ion-input type="text" [(ngModel)]="firstName"></ion-input>
```

Das gleiche für den „lastName“.

```
19      <ion-item>
20        <ion-label floating>Vorname</ion-label>
21        <ion-input type="text" [(ngModel)]="firstName"></ion-input>
22      </ion-item>
23      <ion-item>
24          <ion-label floating>Nachname</ion-label>
25          <ion-input type="text" [(ngModel)]="lastName"></ion-input>
26        </ion-item>
```

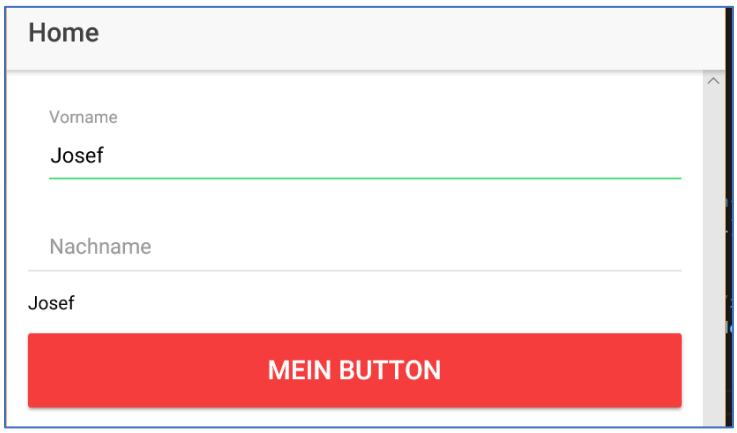
**Ziel**: schon bei der Eingabe soll der Text sofort ausgegeben werden (auto-update)

Dazu gib folgende Zeile unterhalb der Liste ein und zwar den Variablennamen in doppelten geschweiften Klammern:

```
• <p>{{firstName}} {{lastName}}</p>
```

```
26          </ion-item>
27      </ion-list>
28
29      <p>{{firstName}} {{lastName}}</p>
30
31      <button ion-button block large color
```

Ergebnis:




**3d)Aber die Ausgabe erfolgt noch nicht beim Klick auf den Button.**

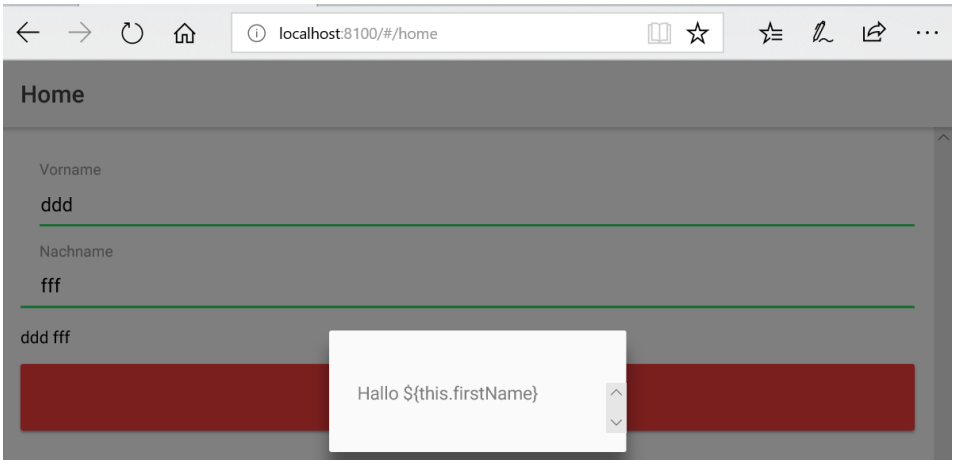
Dazu ändere die Ausgabe in der Variablen „hello“ in der „home.ts“:

Die fixe Message „Hallo Welt“ wird ersetzt:

In einfachen Anführungszeichen, damit man die Variablen dazu verketten kann. Mit Dollarzeichen und geschwungener Klammer:

```
24
25    hello(){
26      this.alertCtrl.create({
27        "message": 'Hallo ${this.firstName} ${this.lastName}'
28      }).present();
```

Ergebnis (leider nicht korrekt):



## 4)Auf andere Pages navigieren

Dafür erstelle eine neue Funktion in der „home.ts“.

Zeile 31-33 kann ohne Bedenken gelöscht werden.

```
30
31  ionViewDidLoad() {
32    console.log('ionViewDidLoad HomePage');
33  }
```

**4a)Neue Funktion mit dem erfundenen Namen „navToAboutPage“:**

```
35  navToAboutPage(){
36    this.navCtrl.push("");
37
38  }
```

- Dazu verwendet man den Nav-Controller, der die Möglichkeit bietet, zwischen den Pages zu wechseln. Zu finden ist er z.B. bereits in Zeile 21.
- Die Umleitung erfolgt mit „push“ und dem Namen, wohin gepuscht werden soll. Der Name für die „about-Site“ ist zu finden in der „about.ts“ Zeile 16 bei „class“:

```
16  export class AboutPage {
```

Daher:

```
35  navToAboutPage(){
36    this.navCtrl.push("AboutPage");
37  }
```

**4b)Auslösung des „push“:**

Dafür wird in der „home.html“ ein Button oben rechts in der Navbar erstellt.

Mit dem „end“ wird er ans Ende gestellt:

```
 9  <ion-navbar>
10    <ion-title>Home</ion-title>
11    <ion-buttons end>
12
13    </ion-buttons>
14  </ion-navbar>
```

Nun kommt der Buttons hinein:

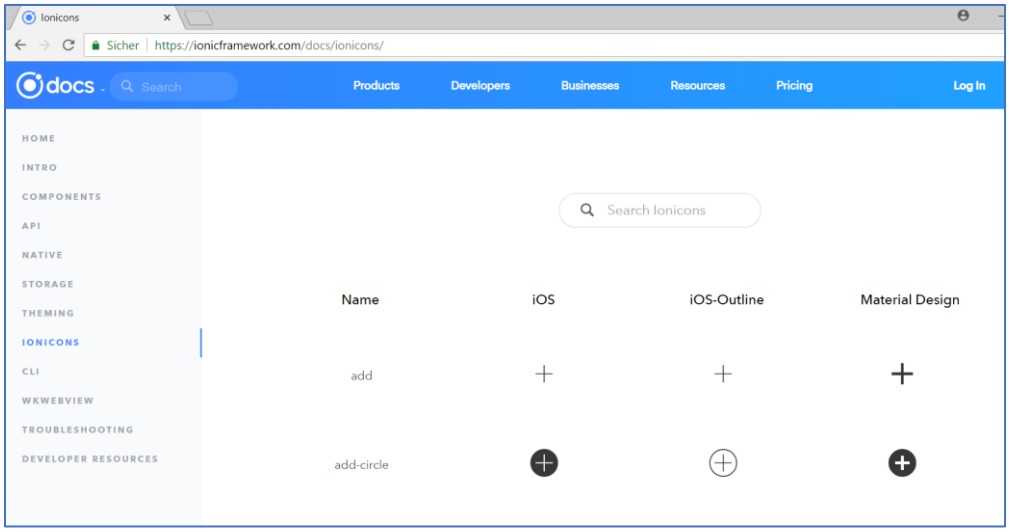
Dieser soll nur das Icon enthalten, ohne Text, daher „icon-only“.

Füge nun den Button ein, mit „icon-only>

```
11      <ion-buttons end>
12        <button ion-button icon-only>
13          |
14        </button>
15      </ion-buttons>
```

Eine Auswahl an Icons findet man auf einer Seite des Frameworks, nämlich

https://ionicframework.com/docs/ionicons/




Ganz rechts findet man de Icons für Android unter „Material Design“.

Wähle z.B.


In den Button kommt „ion-icon“ mit dem speziellen Icon-Symbol: name=“md-skip-forward“.

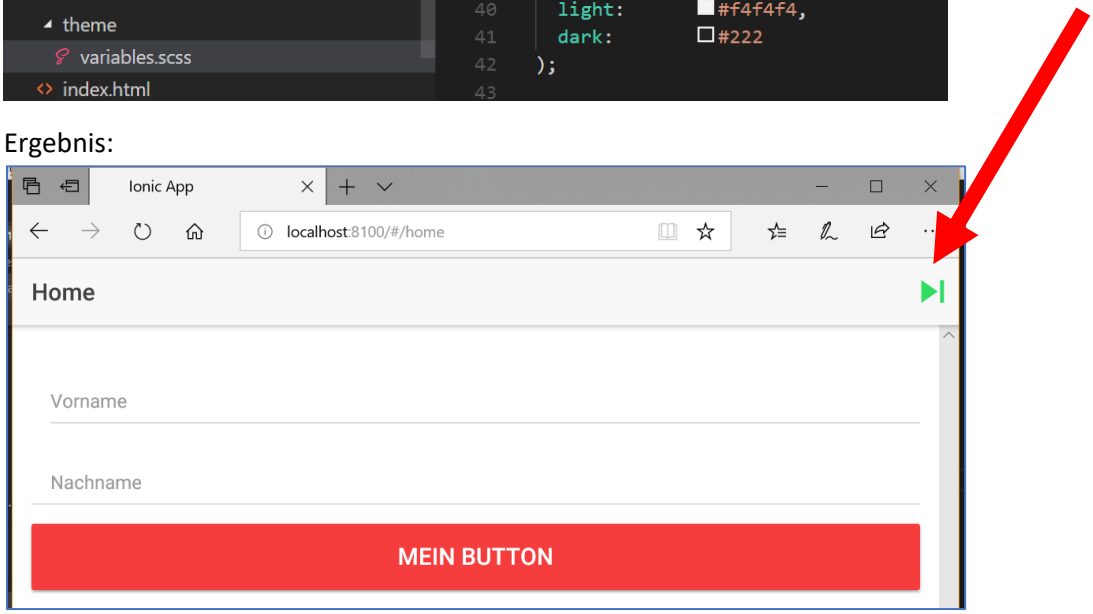
Dann noch eine grüne Farbe für das Icon, mit „color“.

```
11      <ion-buttons end>
12        <button ion-button icon-only color="secondary">
13          <ion-icon name="md-skip-forward"></ion-icon>
14        </button>
15      </ion-buttons>
```

Tipp: Die „color“ kann aus bestehenden vordefinierten 5 Möglichkeiten ausgewählt werden, durch einfaches Verwenden des betreffenden Namens, z.B. „secondary“ für grün.
Natürlich kann der Hexadezimalcode auch individuell abgeändert werde.

Zu finden in der Datei „variables.scss“ im Ordner „theme“:


Ergebnis:




**<u>Event hinterlegen, damit etwas passieren kann durch Anklick:</u>**

Dafür füge im Button ein „click“ in Klammern ein und nach dem Istgleichzeichen folgt der Funktionsnamen aus der „home.ts“

```
<button ion-button icon-only color="secondary" (click)="navToAboutPage()">
  <ion-icon name="md-skip-forward"></ion-icon>
```

```
navToAboutPage(){
  this.navCtrl.push("AboutPage");
}
```

Ergebnis:

Die Verlinkung funktioniert. Die besuchte Seite erhält einen Pfeil für ein Zurück