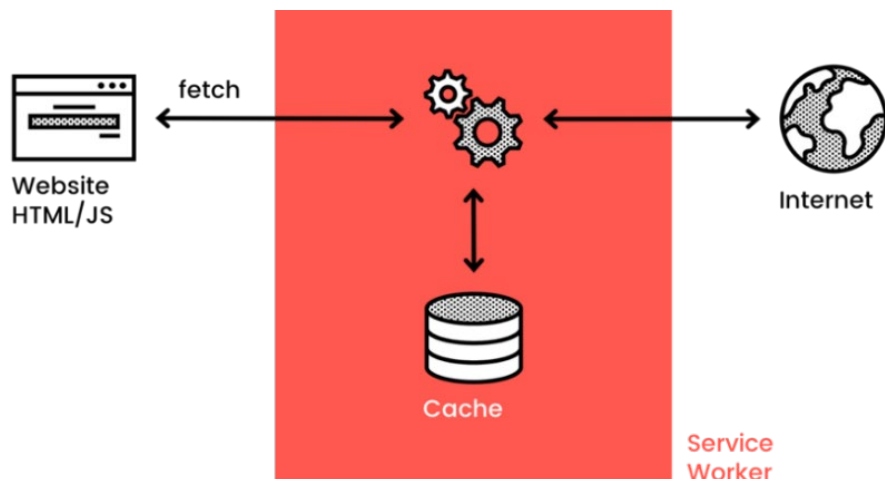


Bestandteile einer PWA

Damit aus einer Web App eine PWA wird, benötigst du neben einer HTTPS Verbindung zwei Dinge: eine Datei namens **manifest.json** (wird in diesem Arbeitsblatt behandelt) und einen **Service Worker** (wird im nächsten Arbeitsblatt genauer behandelt).

Die Datei **manifest.json** liefert Informationen über eine Anwendung (wie Name, Autor, Icon und Beschreibung) in einer JSON-Textdatei. Der Zweck des Manifests ist es, Web-Anwendungen auf dem Homescreen eines Geräts zu installieren. Das liefert Benutzern einen schnelleren Zugriff und lässt eine Web App wie eine native App erscheinen¹.

Ein **Service Worker** ermöglicht die Offline-Verwendung der PWA. Er läuft in einem Thread im Hintergrund und verfügt über einen eigenen Cache. Er kann sich zwischen jeden ausgehenden Netzwerk-Request schalten. Der Service Worker entscheidet, ob er eine Anfrage aus seinem Cache beantworten kann oder die Anfrage an das Netzwerk weiterleitet. Was in den Cache kommt, ist abhängig von der Art der Applikation. Daher ist der Service Worker für jede Applikation anders. Ein Service Worker ermöglicht auch Push-Nachrichten im Client anzuzeigen.



Vom **Vue Plugin PWA** wird für die Manifest Datei und dem Service Worker bereits die nötige Infrastruktur zur Verfügung gestellt.

Progressive Web App (PWA) – durchgehendes Projekt

Um das Basisprojekt nicht zu kompliziert zu machen, beginnen wir mit einer simple Vue.js Applikation namens **employees-simple-pwa** mit Vue CLI 3. Die Applikation soll später eine Liste von Employees mit Bildern anzeigen und das Löschen von Daten erlauben. In der Startdatei ist jedoch nur der Client enthalten, der mit Vue CLI generiert wurde.

Aufgabe 1: Entpacke die Datei **employees-simple-pwa START 1.zip**. Support für die PWA wurde beim Erstellen schon konfiguriert.

Zu deiner Information, so wurde der Client erzeugt (nächste Seite):

¹ Siehe auch: <https://developer.mozilla.org/de/docs/Web/Manifest>

Robert Baumgartner

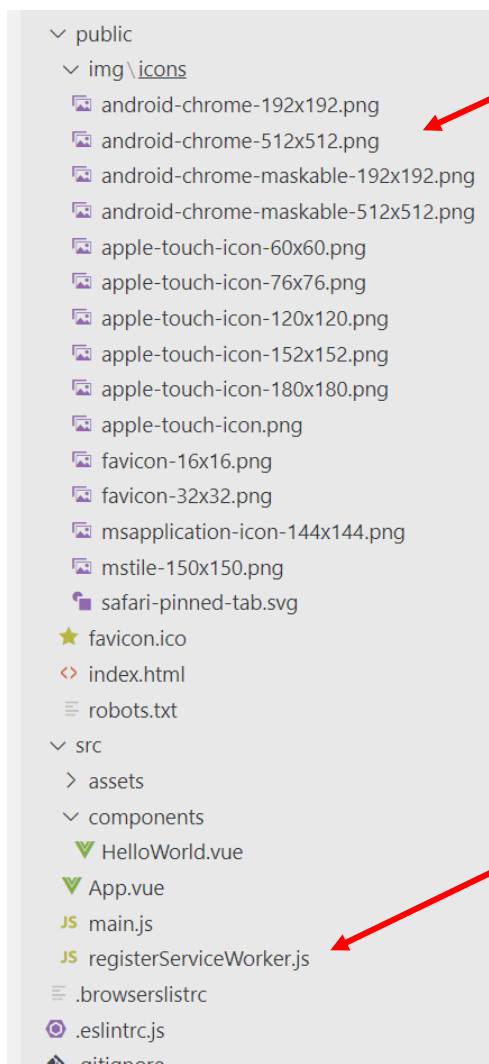
```
Vue CLI v4.5.14
? Please pick a preset: Manually select features
? Check the features needed for your project:
(*) Choose Vue version
(*) Babel
( ) TypeScript
>(*) Progressive Web App (PWA) Support
( ) Router
( ) Vuex
( ) CSS Pre-processors
(*) Linter / Formatter
( ) Unit Testing
( ) E2E Testing
```

Solltest du mal vergessen das PWA Plugin zu installieren, kannst du das mit

```
vue add @vue/pwa
```

jederzeit nachholen!

So sieht ein Teil des Verzeichnisses des Clients aus.



Im Verzeichnis **/public/img/icons** findest du zahlreiche kleine Icons für unterschiedliche Auflösungen. Diese werden später für das Smartphone verwendet. Dabei benutzen Android und iOS Smartphones unterschiedliche Auflösungen.

Du siehst, dass eine Datei **registerServiceWorker.js** erzeugt wurde. Das ist allerdings noch nicht unser Service Worker! Diese Datei registriert ihn bloß!

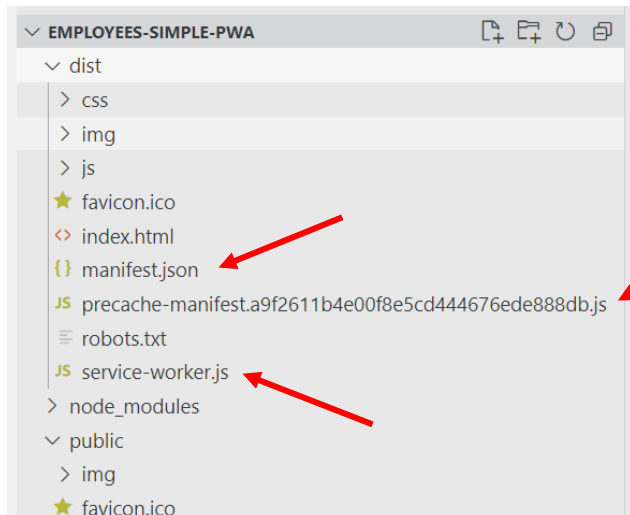
Doch wo ist die **manifest.json** Datei und der eigentliche Service Worker?

Robert Baumgartner

Diese werden im Zuge des Buildprozesses generiert.

Aufgabe 2: Starte den Buildprozess.

Nach der Fertigstellung siehst du, dass **service-worker.js**, **manifest.json** und ein **precache-manifest.hashcode.js** automatisch im **\dist** Verzeichnis mit Defaultwerten erzeugt wurden (Info zum Service Worker und dem Precache Manifest im nächsten Arbeitsblatt).



So sieht die generierte **manifest.json** Datei aus, welche unter anderem die Icons aus **/img/icons** registriert:

```
{
  "name": "employees-simple-pwa",
  "short_name": "employees-simple-pwa",
  "theme_color": "#4DBA87",
  "icons": [
    {
      "src": "./img/icons/android-chrome-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "./img/icons/android-chrome-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    },
    {
      "src": "./img/icons/android-chrome-maskable-192x192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "maskable"
    },
    {
      "src": "./img/icons/android-chrome-maskable-512x512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "maskable"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "background_color": "#000000"
}
```

Robert Baumgartner

Wenn du dich jetzt wunderst, warum in **manifest.json** keine iOS Icons stehen, dann wäre das toll.

Die Erklärung: Safari lädt die Icons über den HTML link Tag². Das Vue Plugin generiert jedoch alle nötigen Tags automatisch!

```
<link rel="apple-touch-icon" href="/img/icons/apple-touch-icon-152x152.png" />
```

Wenn du mehr Auflösungen für iOS möchtest, musst du mehr Links hinzufügen.

Die **manifest.json** Datei enthält aber auch noch weitere Informationen, um das Erscheinungsbild der PWA möglichst an das Smartphone anzupassen. Die wichtigsten Angaben³ sind:

- **name** und **short_name**: Du musst mindestens eines der Properties **short_name** oder **name** angeben. Wenn du beides angibst, wird **short_name** auf dem Startbildschirm des Nutzers, im Launcher oder an anderen Stellen, an denen der Platz begrenzt ist, verwendet. **name** wird verwendet, wenn die App installiert wird.
- **icons**: Wenn ein Nutzer deine PWA installiert, kannst du eine Reihe von Icons für den Browser definieren, die auf dem Startbildschirm, dem App-Launcher, dem Task-Switcher, dem Splash-Screen usw. verwendet werden.

Das Property **icons** ist ein Array von Image Objekten. Jedes Objekt muss eine Quellangabe (**src**), eine Größenangabe (**size**) und den Typ des Bildes (**type**) enthalten.

Chrome benötigt ein 192x192px und ein 512x512px großes Icon. Weitere Größen, wie 270x270 und 180x180, werden empfohlen.

iOS ist ein wenig anders. Wie oben erwähnt, wird von Safari ein Link Tag in HTML dafür verwendet. Die Icons heißen dort Apple Touch Icons⁴.

- **theme_color**: Legt die Standardthemenfarbe für die Anwendung fest. Dies wirkt sich manchmal darauf aus, wie das Betriebssystem die Seite anzeigt (z. B. umrahmt die Farbe der **theme_color** die Seite im Task Switcher von Android).
- **background_color**: Definiert eine Hintergrundfarbe für die Anwendungsseite, die angezeigt wird, bevor das Stylesheet geladen wird. Quasi eine Platzhalterfarbe. Wird auch für den Splash-Screen verwendet (siehe Beispiel weiter unten). Daher sollte **background_color** mit der CSS-Eigenschaft **background-color** im Stylesheet der Website übereinstimmen, um einen reibungslosen Übergang zwischen dem Start der Webanwendung und dem Laden des Website-Inhalts zu gewährleisten.
- **start_url**: Das Property teilt dem Browser mit, wo deine Anwendung starten soll, wenn sie gestartet wird. Sie verhindert, dass die Anwendung auf der Seite startet, auf der sich der Nutzer befand, als er deine Anwendung zu seinem Startbildschirm hinzugefügt hat.

Deine **start_url** sollte den Nutzer direkt in deine App leiten und nicht auf eine Produkt-Landingpage. Überlege dir, was der Nutzer tun möchte, wenn er deine App öffnet, und leite ihn dorthin.

² Manifestdateien werden in Chrome, Edge, Firefox, UC Browser, Opera und dem Samsung-Browser unterstützt. Safari hat nur teilweise Unterstützung. Das kann sich jedoch mit jeder iOS Release ändern.

³ Mehr dazu bei Google Developers: <https://developers.google.com/web/fundamentals/web-app-manifest/>.

⁴ Mehr auf: <https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/app-icon/>

Robert Baumgartner

- **display**: Gibt an, wie die Applikation gestartet werden soll. Es existieren verschiedene Möglichkeiten. Unter anderem: **fullscreen** (Vollbildmodus ohne Browserelemente), **standalone** (ohne Browserelemente, daher Look-and-Feel einer nativen Anwendung), **browser** (Anwendung startet im Browser).

Zum Einbinden der Manifestdatei in die Anwendung wird sie normalerweise im `<head>`-Bereich von **index.html** verlinkt:

```
<link rel="manifest" href="/manifest.json" />
```

Bei Vue.js mit dem PWA Plugin ist sie automatisch eingebunden!

Wie ist das nun mit dem Development Server, den wir normalerweise für das Entwickeln eines Clients verwenden (**npm run dev**)? Es wird zwar dort auch ein **manifest.json** File im Memory generiert, aber kein Service Worker, da es mit dem Caching Probleme gab.

Verwenden wir also den Build vom **/dist** Ordner. Für den Webserver können wir das **npm** Modul **http-server** nutzen. Du kannst den Server global installieren oder per **npm** aufrufen.

Im **/dist** Ordner gib in eine Command Shell ein:

```
Phrozen@DESKTOP-HL8SN06 MINGW64 /f/Bac
$ npx http-server
Starting up http-server, serving ./
Available on:
  http://169.254.123.235:8080
  http://10.0.0.8:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

Aufgabe 3: Starte Chrome und überprüfe im Application Tab der Dev Tools, ob die Manifest Datei und ihre Properties erkannt werden!

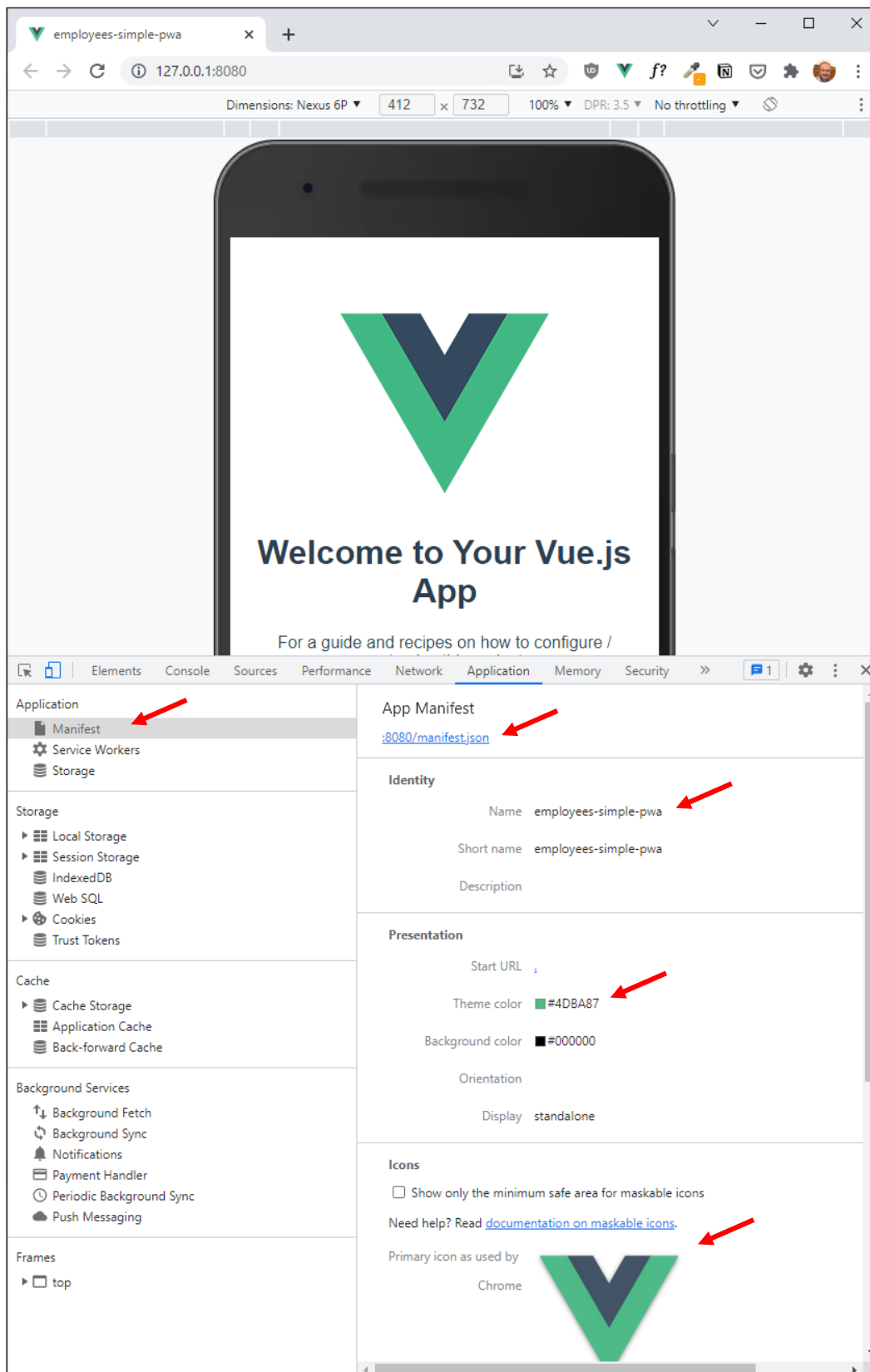
Auf der nächste Seite siehst du, wie es aussehen sollte!

Du kannst die Werte in **manifest.json** auch selbst festlegen. Dazu musst du eine Datei **vue.config.js** (im Rootordner) erzeugen (falls sie noch nicht existiert). In dieser kannst du verschiedene Eigenschaften definieren. Unter anderem auch für die PWA.

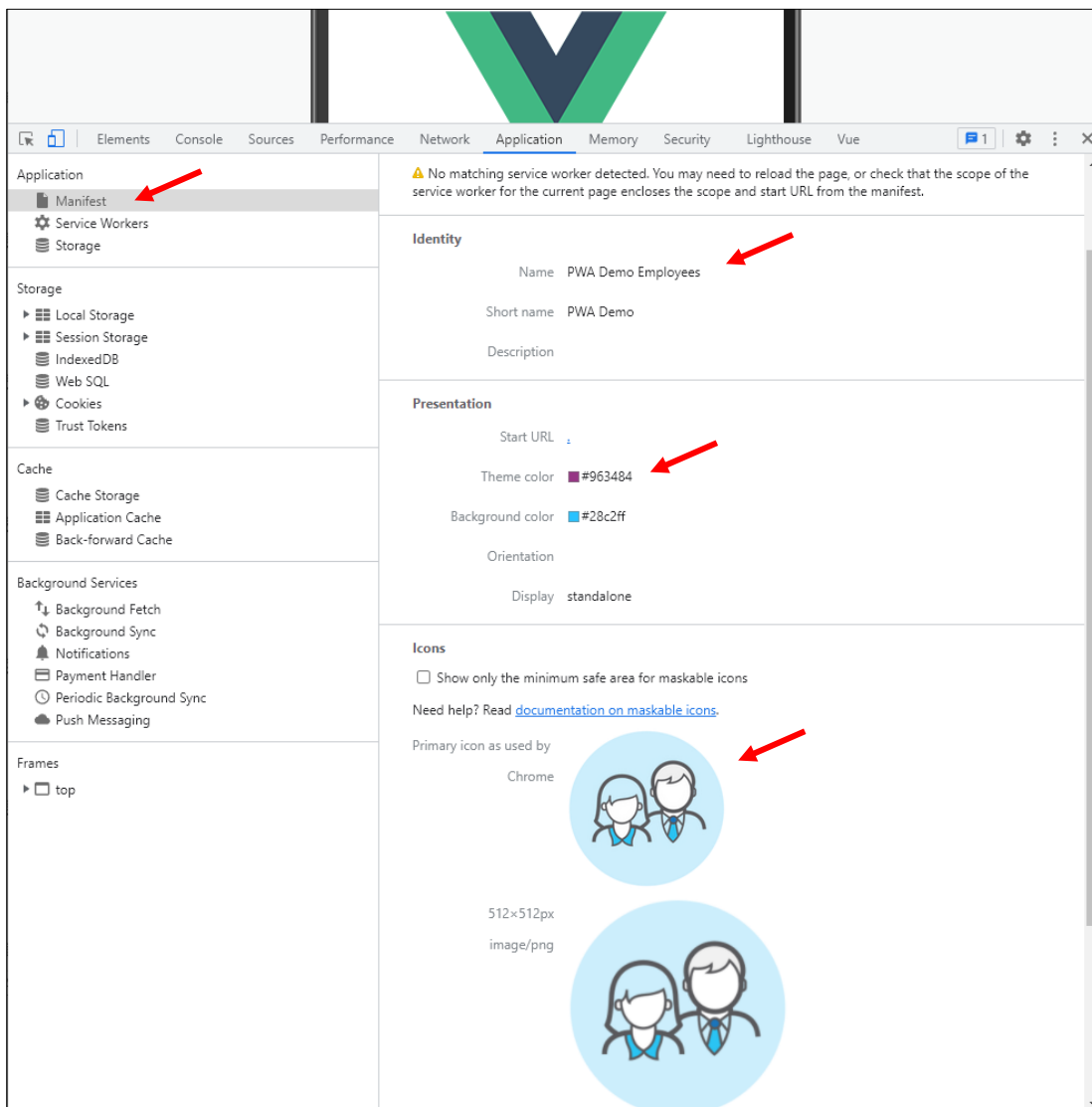
```
module.exports = {
  pwa: {
    manifestOptions: {
      name: 'PWA Demo Employees',
      short_name: 'PWA Demo',
      theme_color: '#963484',
      background_color: '#28c2ff',
      icons: [
        { src: './img/icons/employees_512x512.png', sizes: '512x512', type: 'image/png' },
        { src: './img/icons/employees_512x512.png', sizes: '192x192', type: 'image/png' },
      ],
    },
  },
};
```

Das Ergebnis siehst du auf der übernächsten Seite!

Robert Baumgartner

Ergebnis mit generierter `manifest.json` Datei.

Robert Baumgartner

Ergebnis mit selbst erstellter `manifest.json` Datei.

Wie kommst du zu den nötigen Icons?

Dazu gibt es Tools. Eines ist unten beschrieben:

Nimm ein 512x512 Pixel Bild oder ein SVG Bild und generiere all die anderen Icons! Zum Beispiel mit dem `vue-pwa-asset-generator` (<https://www.npmjs.com/package/vue-pwa-asset-generator>).

```
npx vue-pwa-asset-generator -a {512x512_png_source | svg_source} -o {output_folder}
```



employees.svg

Robert Baumgartner

```

C:\Windows\System32\cmd.exe
F:\Backup E\HTL 2021 - 2022\icons>npx vue-pwa-asset-generator -a employees.svg
Generating images for employees.svg... created!
android-chrome-512x512.png created!
android-chrome-192x192.png created!
android-chrome-maskable-512x512.png created!
android-chrome-maskable-192x192.png created!
apple-touch-icon-60x60.png created!
apple-touch-icon-76x76.png created!
apple-touch-icon-120x120.png created!
apple-touch-icon-152x152.png created!
apple-touch-icon-180x180.png created!
favicon-32x32.png created!
apple-touch-icon.png created!
favicon-16x16.png created!
msapplication-icon-144x144.png created!
mstile-150x150.png created!
favicon.ico created!
manifest.json created!
F:\Backup E\HTL 2021 - 2022\icons>

```

Das Tool generiert die Icons und das **manifest.json** Snippet für die Icons in dem entsprechenden Ordner.



Aufgabe 4: Finde ein geeignetes Bild und generiere die Icons. Erstelle einen passenden **manifest.json** File und starte deinen Client. Überprüfe das Ergebnis im Browser. Mache einen Screenshot.

Deployen des Clients auf das Smartphone

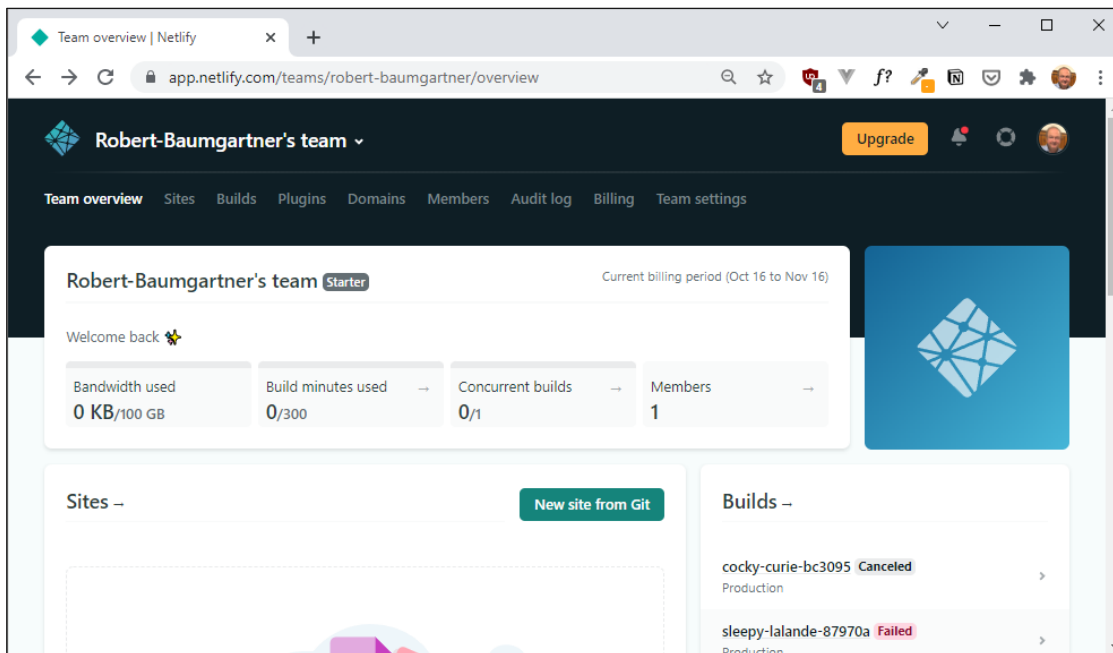
Für das Testen jenseits des Browsers hast du mehrere Möglichkeiten:

- Hosting via HTTPS Server (z.B. **Netlify**, **Heroku**) abseits von selbst gebastelten Zertifikaten
- HTTPS Tunnel zum Beispiel via **ngrok** (Problematisch in Schule) - keine Option!
- **Android Emulator** (optional, > 1 GB Download!)
- **Android Debugging** via USB

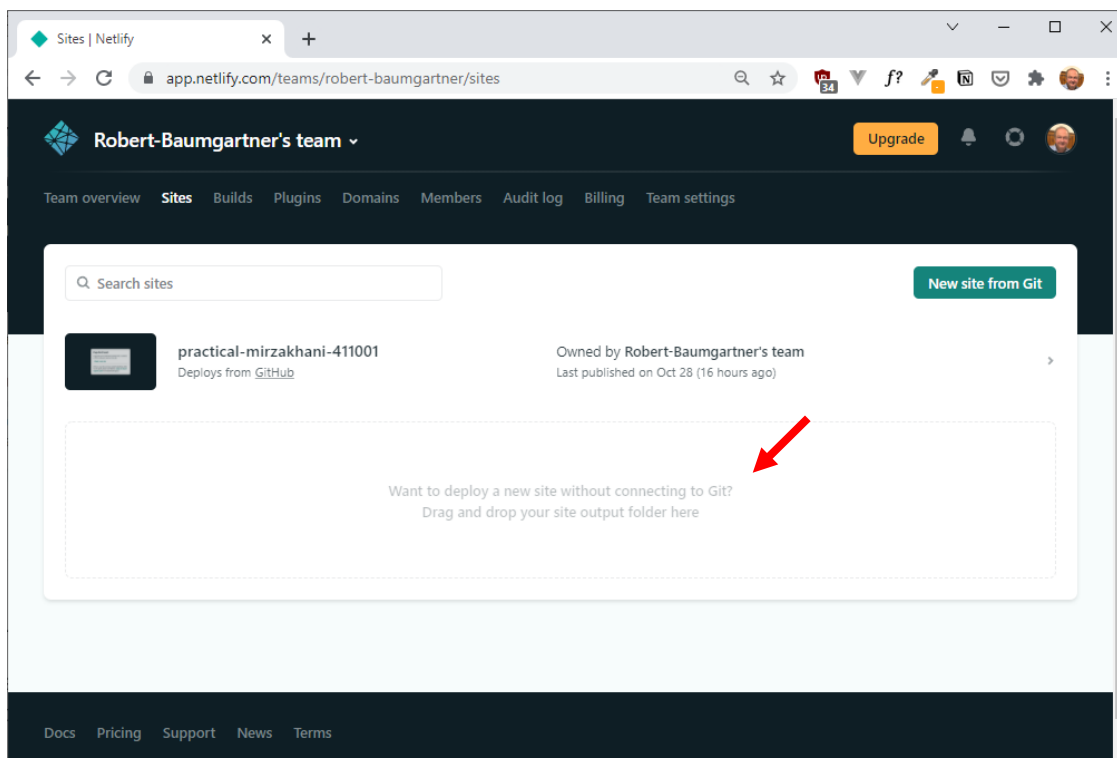
Robert Baumgartner

Aufgabe 5: Die letzten beiden Punkte sind optional. Hoste deinen Client am besten auf **Netlify**.

Lege einen Account auf **Netlify** an (<https://www.netlify.com/>) indem du dich mit deinem Github Account einloggst.



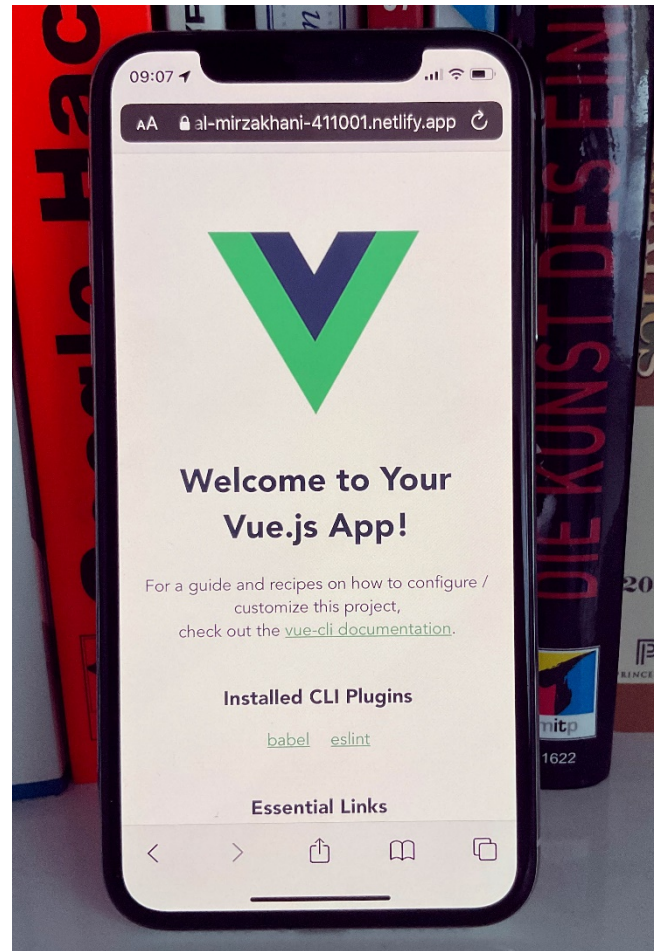
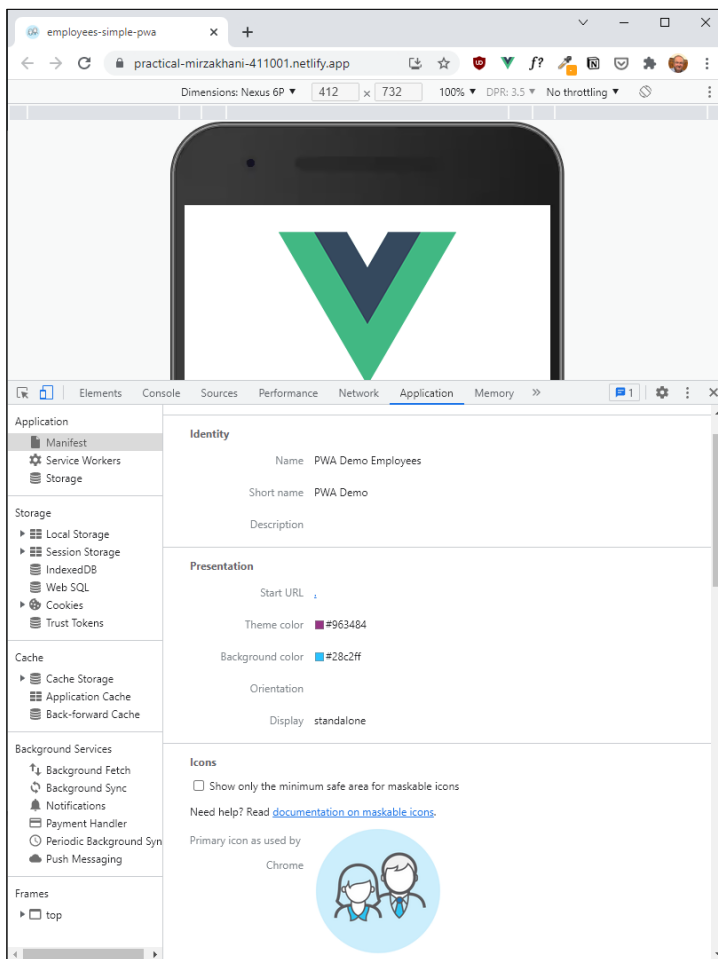
Ziehe deinen **/dist** Ordner in den Drag&Drop Bereich oder verbinde dich via Github. Fertig!



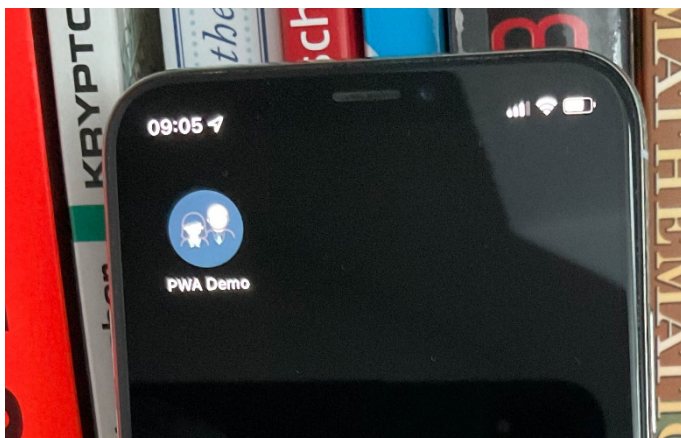
Und schon hast du eine PWA zur Verfügung!

Robert Baumgartner

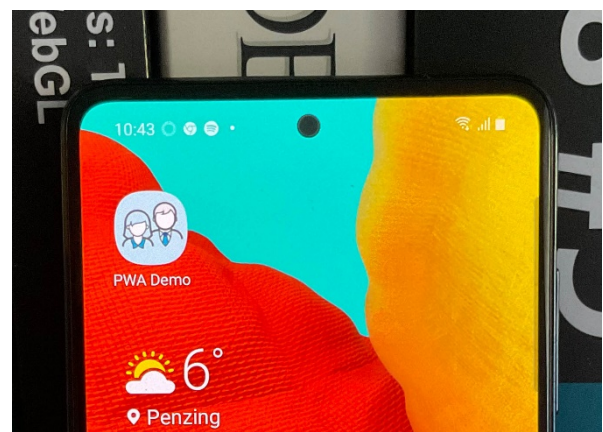
Schau dir deine App im Browser und am Smartphone an:



Speichere die Seite am Handy ab (Überprüfe auch Icon und Titel):

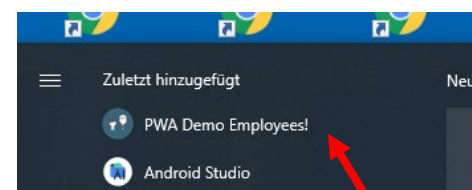
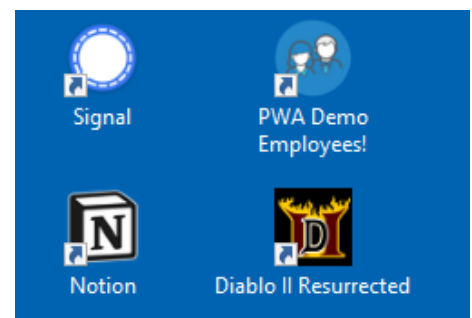
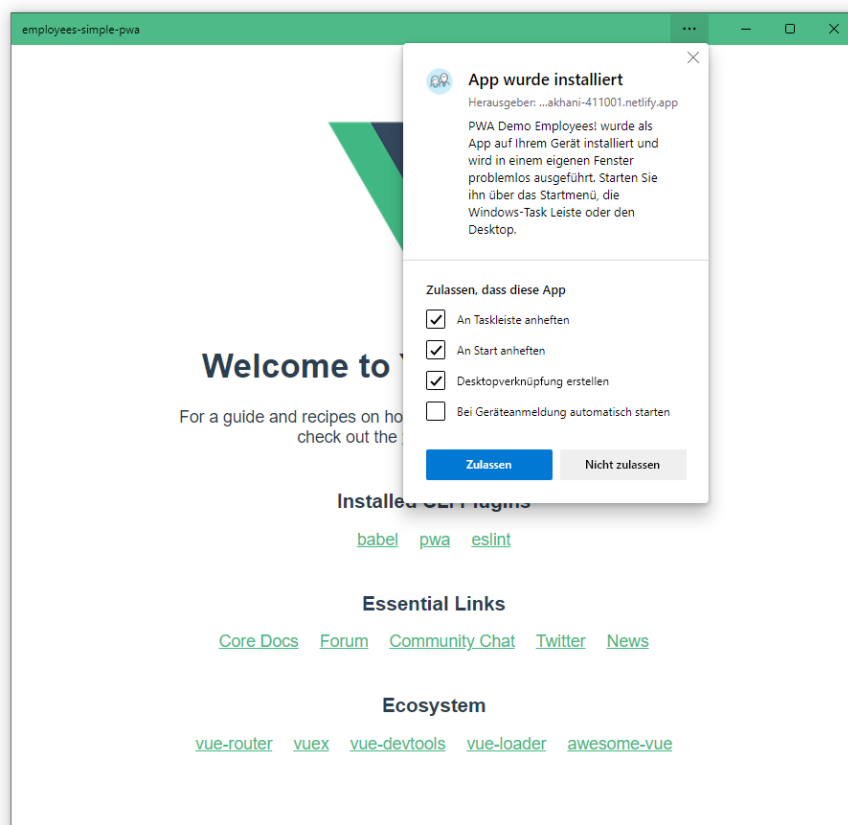
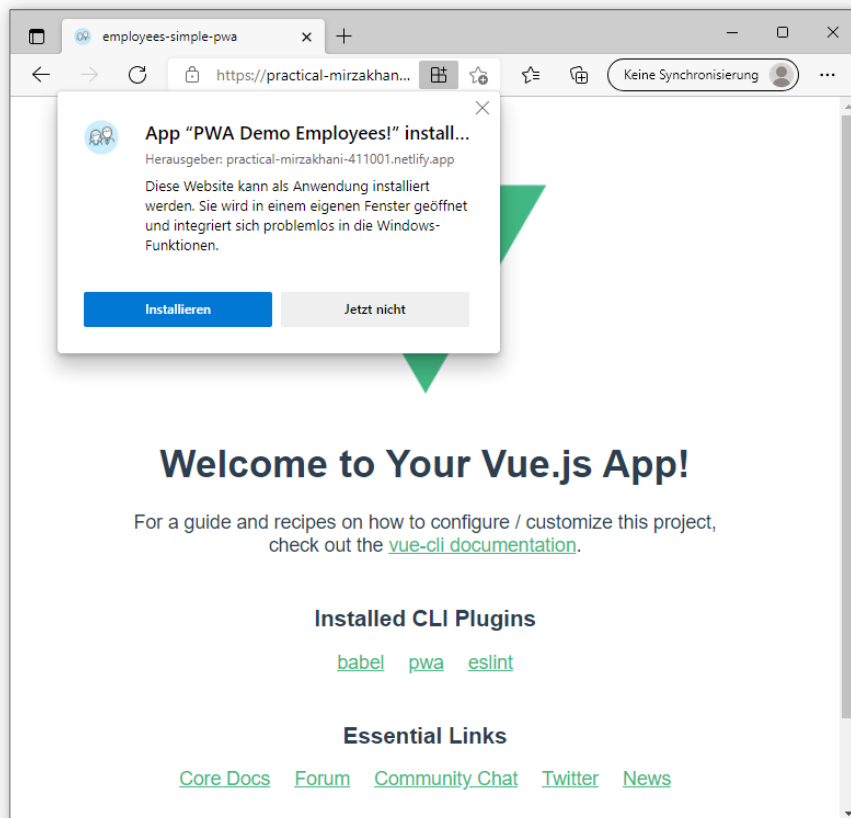


iOS



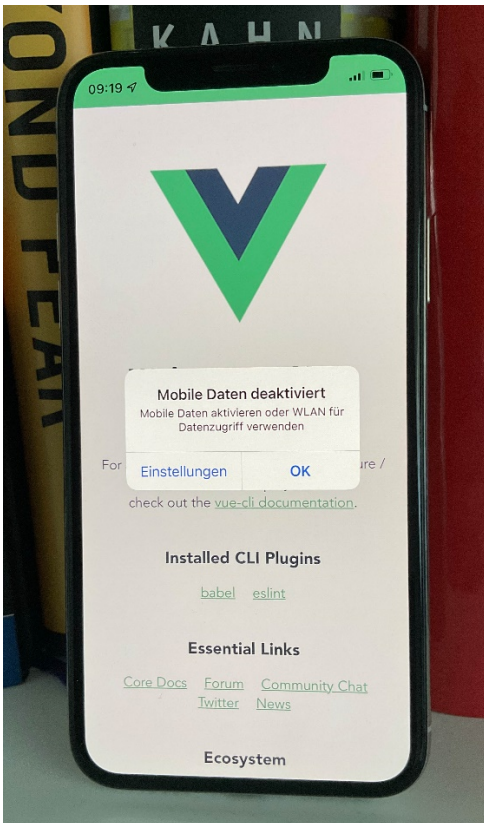
Android

Beachte, dass du deine PWA auch unter Windows installieren kannst. Getestet sowohl mit Chrome als auch mit Edge. Siehe nächste Seite!



Robert Baumgartner

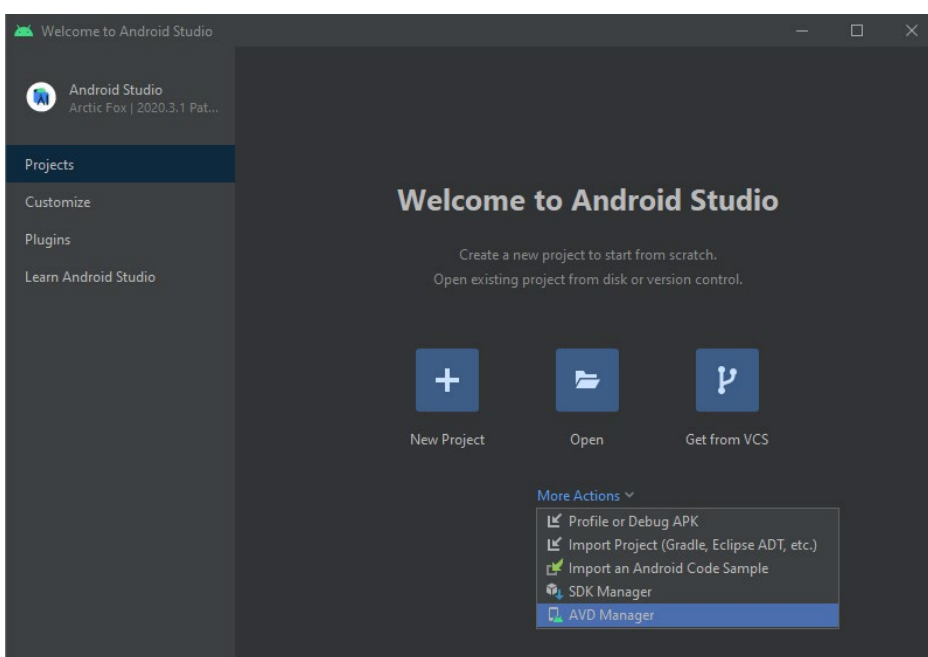
Die App sollte sich auch bei abgeschaltetem Netzwerk starten lassen. Der Rahmen des Browsers sollte nicht sichtbar sein



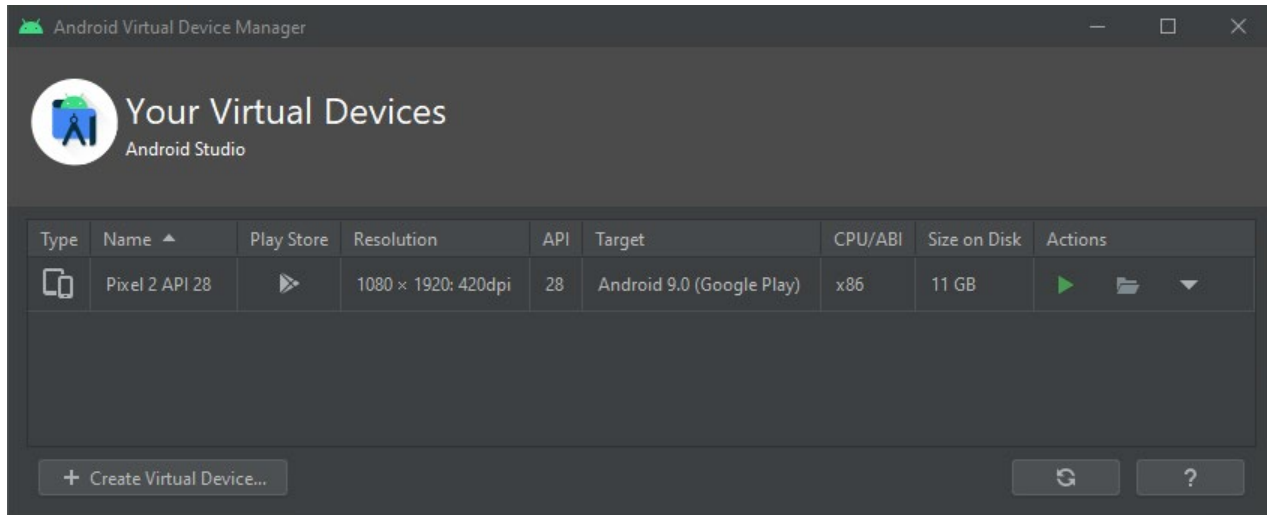
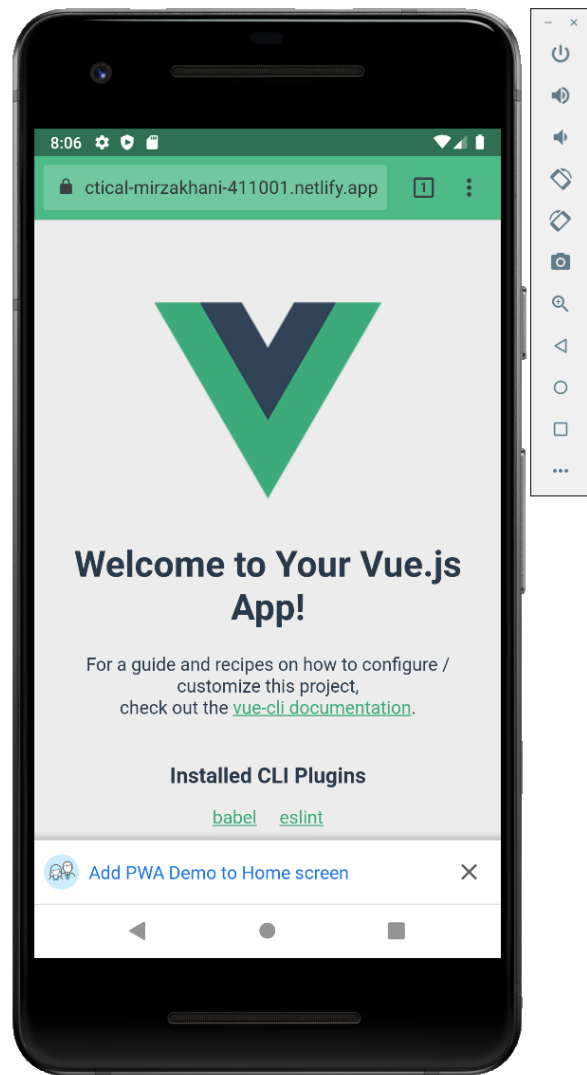
Android Emulator via Android Studio (optional)

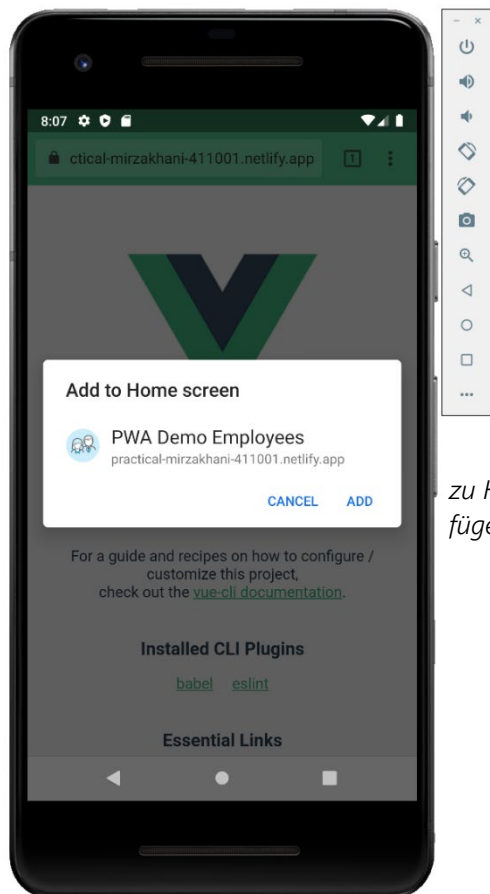
Dazu musst du Android Studio installieren (> 1 GB, siehe: <https://developer.android.com/studio>). Das machst du am besten zu Hause!

Definiere ein virtuelles Gerät (Pixel 2).



Robert Baumgartner

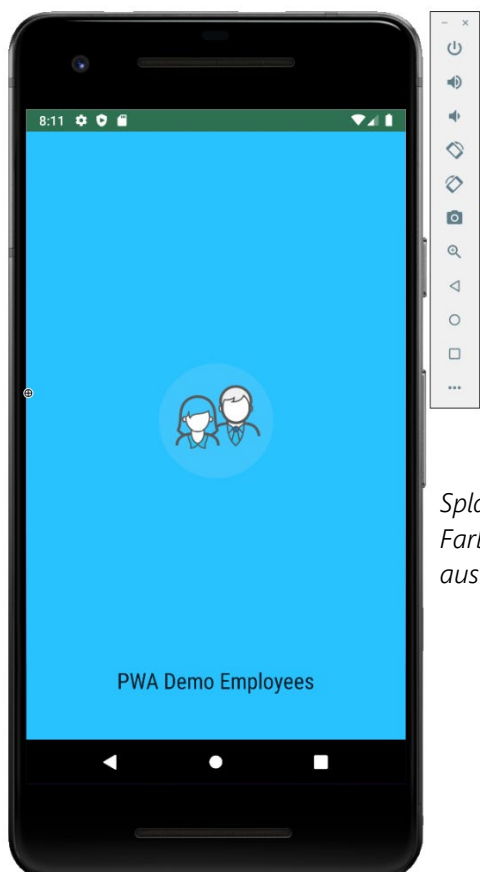
*Starte Emulator**Starte Browser und gib deine
Netlify Adresse an*



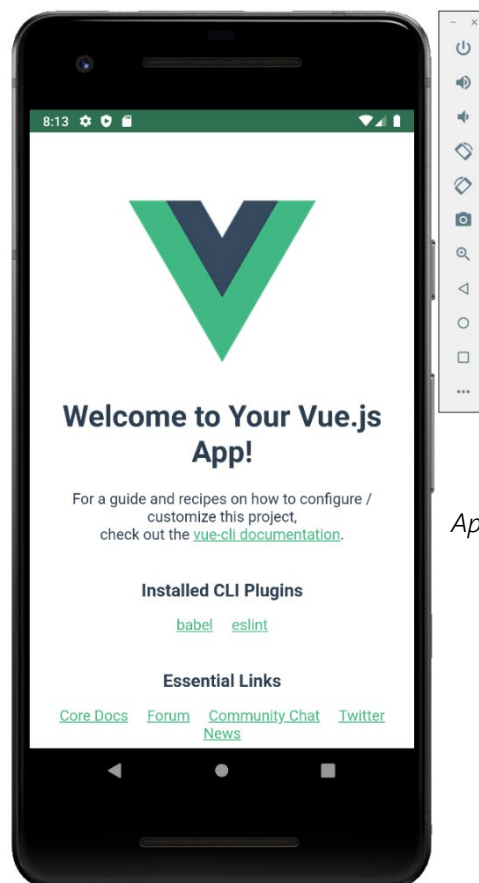
zu Home Screen
fügen



Icon abgelegt, sieht
nur am Emulator so
komisch aus!



Splash Screen mit
Farbe und Icon
aus Manifest



App ohne Rahmen

Robert Baumgartner

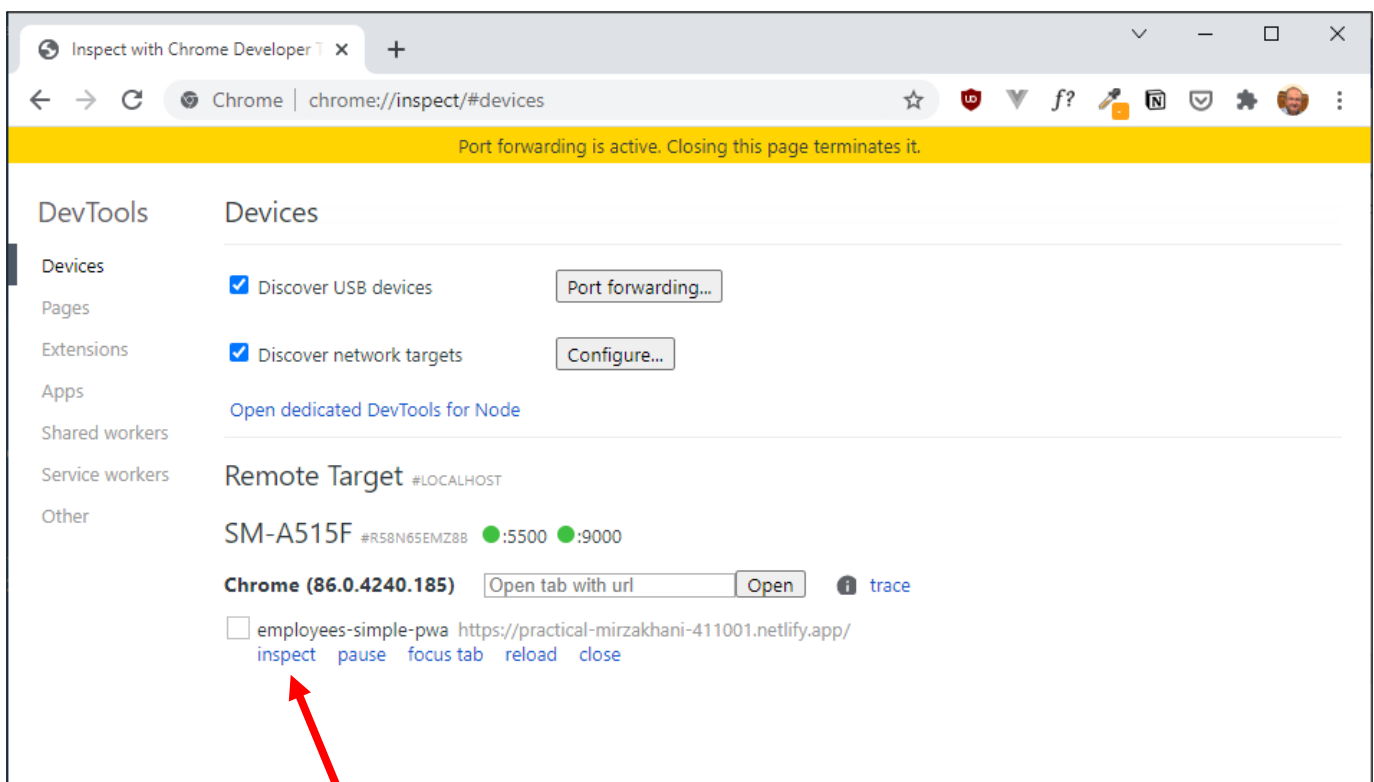
USB Debugging für Android Handy

Voraussetzungen:

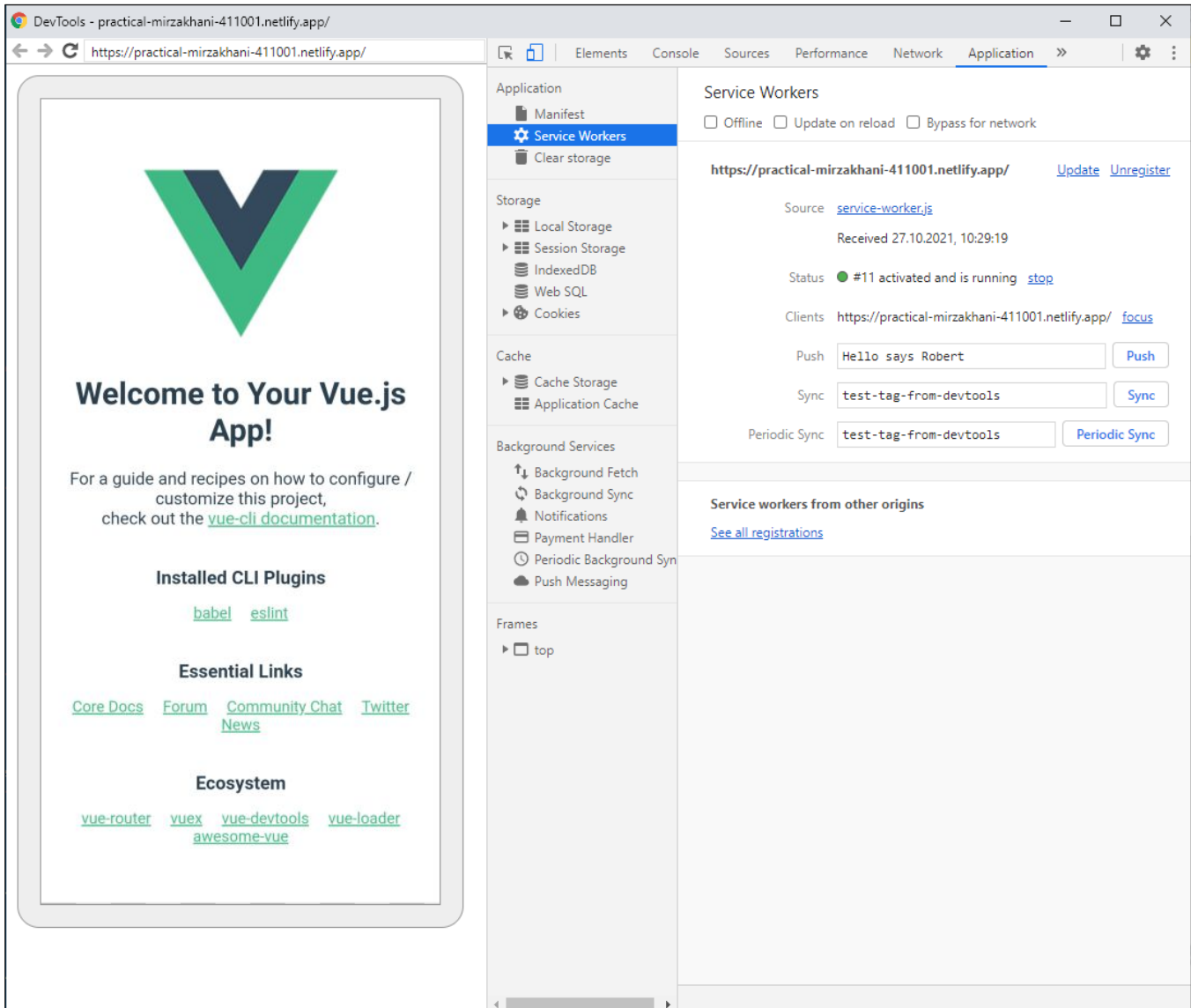
- Geeignetes USB Kabel.
- Android Debugging Bridge via SDK (oder Classroom 😊).
- Entwickler Optionen am Smartphone freischalten.
- USB Debugging einschalten.
- Deine App starten.

USB Debugging einschalten: Siehe Video: <https://www.youtube.com/watch?v=Ucs34BkfPB0>

SDK Tools für ADB : <https://developer.android.com/studio/releases/platform-tools>



Auf der nächsten Seite geht es weiter!



The screenshot shows the DevTools Application tab for a Vue.js Progressive Web App. The main content area displays a welcome message and links to documentation and ecosystem resources. The left sidebar shows the Application panel with sections for Service Workers, Storage, Cache, Background Services, and Frames. The right sidebar shows the Service Workers panel with details for the active service worker.

Application

- Manifest
- Service Workers**
- Clear storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Cache

- Cache Storage
- Application Cache

Background Services

- Background Fetch
- Background Sync
- Notifications
- Payment Handler
- Periodic Background Sync
- Push Messaging

Frames

- top

Service Workers

☐ Offline ☐ Update on reload ☐ Bypass for network

https://practical-mirzakhani-411001.netlify.app/ [Update](#) [Unregister](#)

Source [service-worker.js](#)

Received 27.10.2021, 10:29:19

Status ● #11 activated and is running [stop](#)

Clients [https://practical-mirzakhani-411001.netlify.app/](#) [focus](#)

Push [Push](#)

Sync [Sync](#)

Periodic Sync [Periodic Sync](#)

Service workers from other origins

[See all registrations](#)

Welcome to Your Vue.js App!

For a guide and recipes on how to configure / customize this project, check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

Ecosystem

[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)