Charles Bochet Follow
@charlesBochet CTO @LuckeyHomes
Apr 25, 2017 · 7 min read

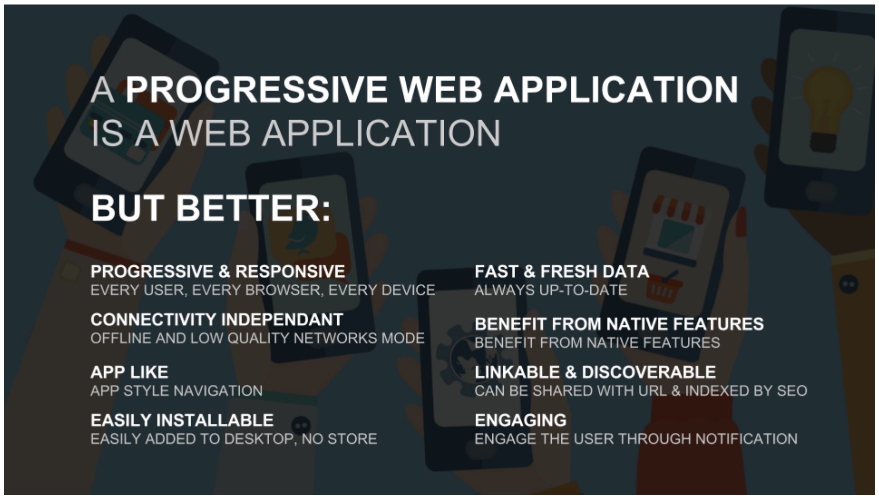
A progressive Web application with Vue JS, Webpack & Material Design [Part 1]

[Updated 11/27/2017]

Progressive web applications are the future. And more and more big companies are starting playing with them (such as Twitter: https://mobile.twitter.com/).

Imagine a Web Application that you can browse in the subway, that keeps engaging its user through notifications, up-to-date data and that offers app-like navigation, and you get an overview of PWAs capabilities.

A Progressive Web Application (PWA) is a web application that offers an app-like user experience. PWAs benefit from the modern Web technological innovations (Service Workers, Native APIs, JS frameworks) and raise web application quality standard.



A **PROGRESSIVE WEB APPLICATION**
IS A WEB APPLICATION

BUT BETTER:

PROGRESSIVE & RESPONSIVE
EVERY USER, EVERY BROWSER, EVERY DEVICE

CONNECTIVITY INDEPENDANT
OFFLINE AND LOW QUALITY NETWORKS MODE

APP LIKE
APP STYLE NAVIGATION

EASILY INSTALLABLE
EASILY ADDED TO DESKTOP, NO STORE

FAST & FRESH DATA
ALWAYS UP-TO-DATE

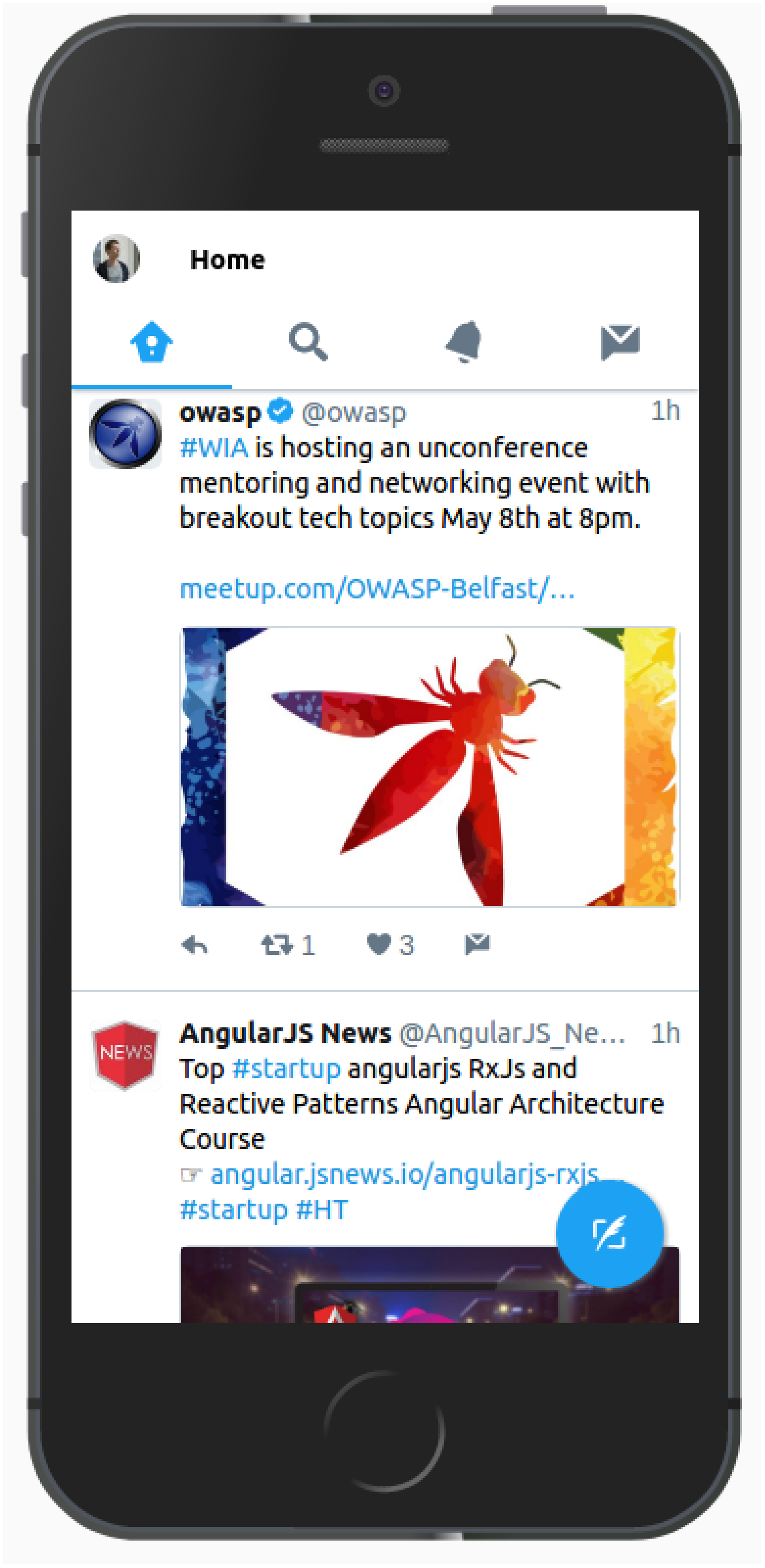
BENEFIT FROM NATIVE FEATURES
BENEFIT FROM NATIVE FEATURES

LINKABLE & DISCOVERABLE
CAN BE SHARED WITH URL & INDEXED BY SEO

ENGAGING
ENGAGE THE USER THROUGH NOTIFICATION

If you want to learn more about PWAs, please visit this great Google developer page.

Look at the following PWA ! It looks like a native app, doesn't it?


Home
owasp @owasp 1h
#WIA is hosting an unconference mentoring and networking event with breakout tech topics May 8th at 8pm.
meetup.com/OWASP-Belfast/...
1
3
AngularJS News @AngularJS_Ne... 1h
Top #startup angularjs RxJs and Reactive Patterns Angular Architecture Course
☞ angular.jsnews.io/angularjs-rxjs...
#startup #HT

Twitter progressive web application

From the developer point of view, PWAs have huge plus on native applications. It's basically a website, so:

- you can write them with any framework you want;
- one code to rule them all: it is cross-platform and cross-devices (the code is executed by user's browser);
- easy to ship: no need to download it through a Store.

However, in early 2017, PWAs still face some restrictions:

- Safari does not support some basic PWAs features, such as Service workers, but Apple seems to be working on it;
- some native functions are still not supported: for more information, see this page What web can do.

Tutorial objective

This tutorial aims to create a basic but complete progressive web application with VueJS and Webpack, from scratch. Our application will meet all the requirements announced in introduction: progressive, responsive, connectivity independant, etc. I want to give you an overview of what can be achieved with PWAs: fluid native-like application, offline behaviors, native features interface, push notifications.

To keep things challenging, we are going to build a cat picture messaging app: CropChat! Cropchat users will be able to read a main flow of cat pictures, open them to view details and post new cat pictures (first from internet, then from device drive or camera).

The tutorial will be split in several parts, that will be published successively:

- [Part 1] Create a Single Page Application with VueJS, Webpack and Material Design Lite
- [Part 2] Connect the App with a distant API with Vue-Resource and VueFire
- [Part 3] Implement offline mode with Service Workers
- [Part 4] Access device camera to take pictures
- [Part 5] Access device drive to upload pictures

- [Part 6] Implement push notifications
- [Part 7] Access device location

Basic components of our PWA

Our Progressive Web Application is based on modern components you are going to like!

- VueJS 2 View Layer: renders our views with a little help of Material Design Lite
- Vue-Router: handles the routing of the SPA
- Vue-Resource & Vuefire: handle communication with an existing Firebase database
- Service Worker: handles offline mode and keeps data up-to-date
- Webpack & Vue-loader: build our application, provides hot reload, ES2016 and pre-processors.

Let's start with part 1!

[PART 1] Create a Single Page Application with VueJS, Webpack and Material Design Lite

If you are not familiar with VueJS 2, I strongly recommend that you take a look at the official tutorial

Build the VueJS App base

We are going to use Vue-cli to scaffold our application:

```
npm install -g vue-cli
```

Vue-cli comes along with a few templates. We will choose pwa template. Vue-cli is going to create a dummy VueJS application with

Webpack, vue-loader (hot reload!), a proper manifest file and basic offline support through service workers.

Vue pwa template is built on top of Vue webpack template. Webpack is a modern and powerful module bundler for Javascript application that will process and build our assets.

```
vue init pwa cropchat
```

You will be asked a few questions. Here is the configuration I used:

```
? Project name cropchat
? Project short name: fewer than 12 characters to not be
truncated on homescreens (default: same as name) cropchat
? Project description A cat pictures messaging application
? Author Charles BOCHET <charlesb@theodo.fr>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Setup unit tests with Karma + Mocha? Yes
? Setup e2e tests with Nightwatch? No

vue-cli · Generated "cropchat".
```

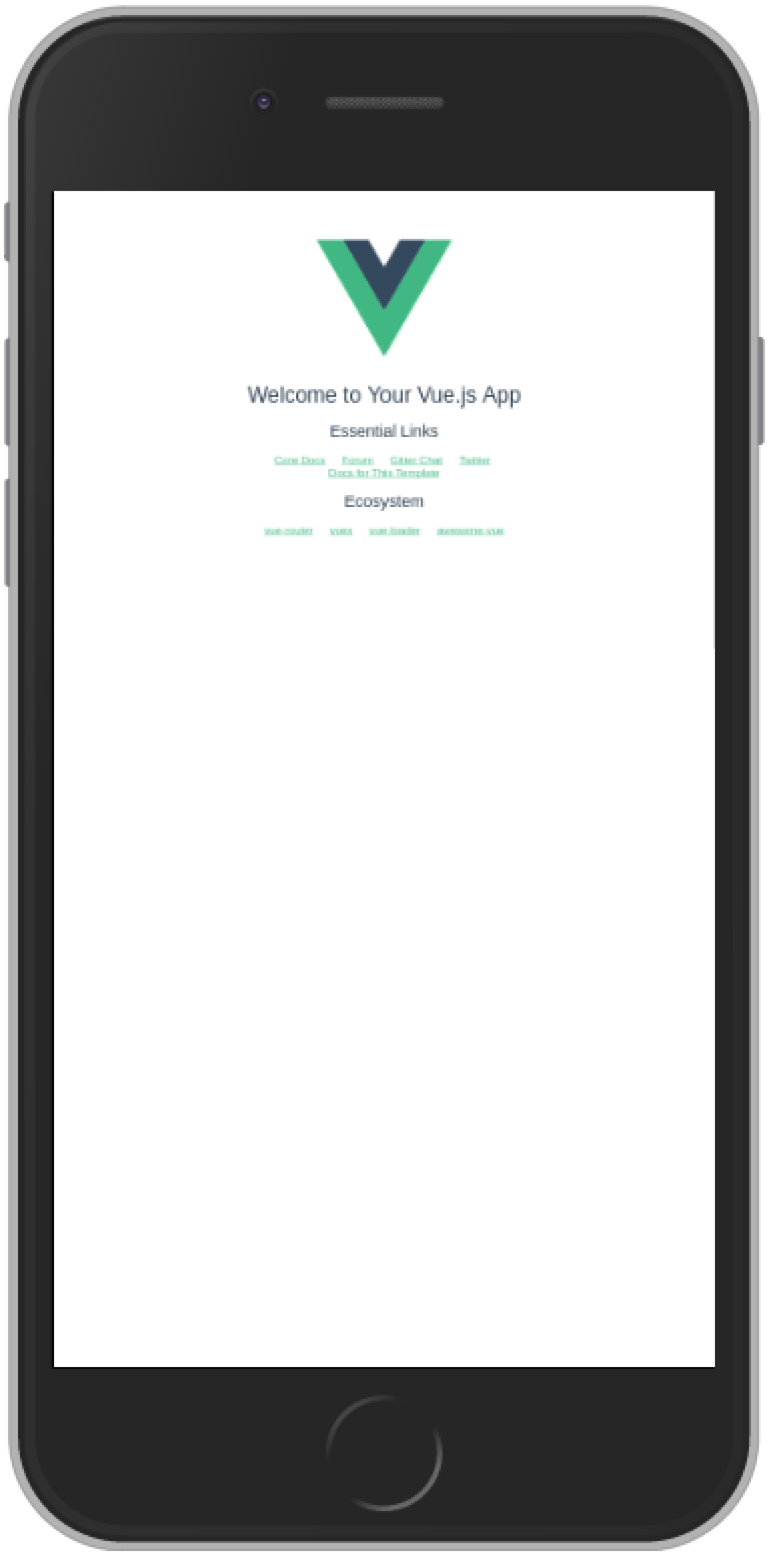
This process creates a project folder with following subfolders:

- build: contains webpack and vue-loader configuration files
- config: contains our app config (environments, parameters…)
- src: source code of our application
- static: images, css and other public assets
- test: unit test files propelled by Karma & Mocha

Then run:

```
cd cropchat
npm install
npm run dev
```

This will open your browser on `localhost:8080` :



Manifest.json: make your application installable

One of the biggest plus of PWA is that applications are easily installable and sharable. All web applications that provides a valid `manifest.json` file in their `index.html` are installable.

Vue pwa template provides a default `manifest.json` file.

Edit default `static/manifest.json` file to customize your project:

```
{
  "name": "cropchat",
  "short_name": "cropchat",
  "icons": [
    {
      "src": "/static/img/icons/cropchat-icon-64x64.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/static/img/icons/cropchat-icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "/static/img/icons/cropchat-icon-256x256.png",
      "sizes": "256x256",
      "type": "image/png"
    },
    {
```

manifest.json

You can customize a few things here:

- app **names** ;
- app **icons** that will be displayed on device home screen and splash screen (you can find cropchat official icons on our GitHub repository) ;
- **start url** ;
- **display** and **orientation** ;

- **background** and **theme colors.**

Here is the complete list of manifest options on Mozilla Developer website.

Take a look at `index.html` and see that manifest is already linked here:

```
<link rel="manifest" href="<%= htmlWebpackPlugin.files.publi
```

index.html

Make sure you also have a viewport declared in the head section of `index.html` :

```
<meta name="viewport" content="width=device-width, initial-s
```

index.html

That's it! Let's try to install CropChat on a mobile device. There are many ways to access our `localhost:8080` from distant mobile device. My favorite one is to use ngrok.

Ngrok is a service that relays your local environment on a distant dns, for free!

To install it:

```
npm install -g ngrok
```

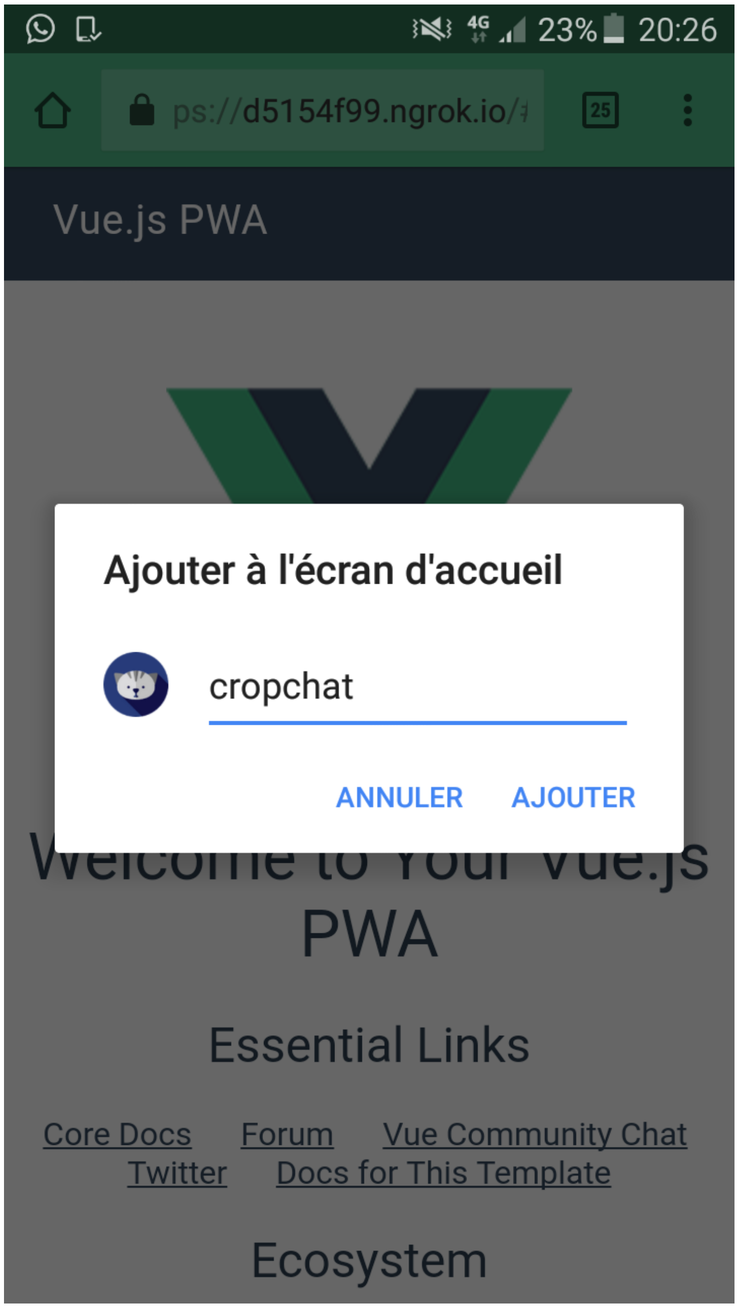
Then, run:

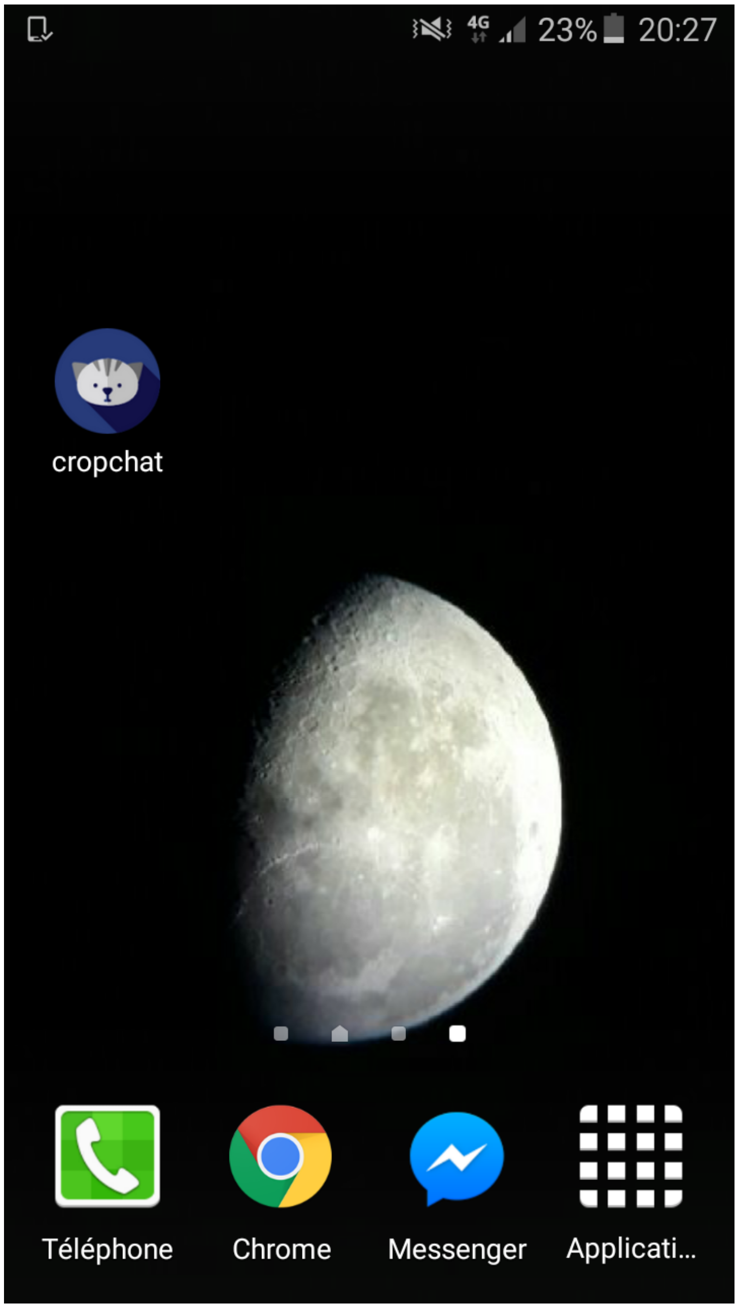
```
ngrok http 8080
```

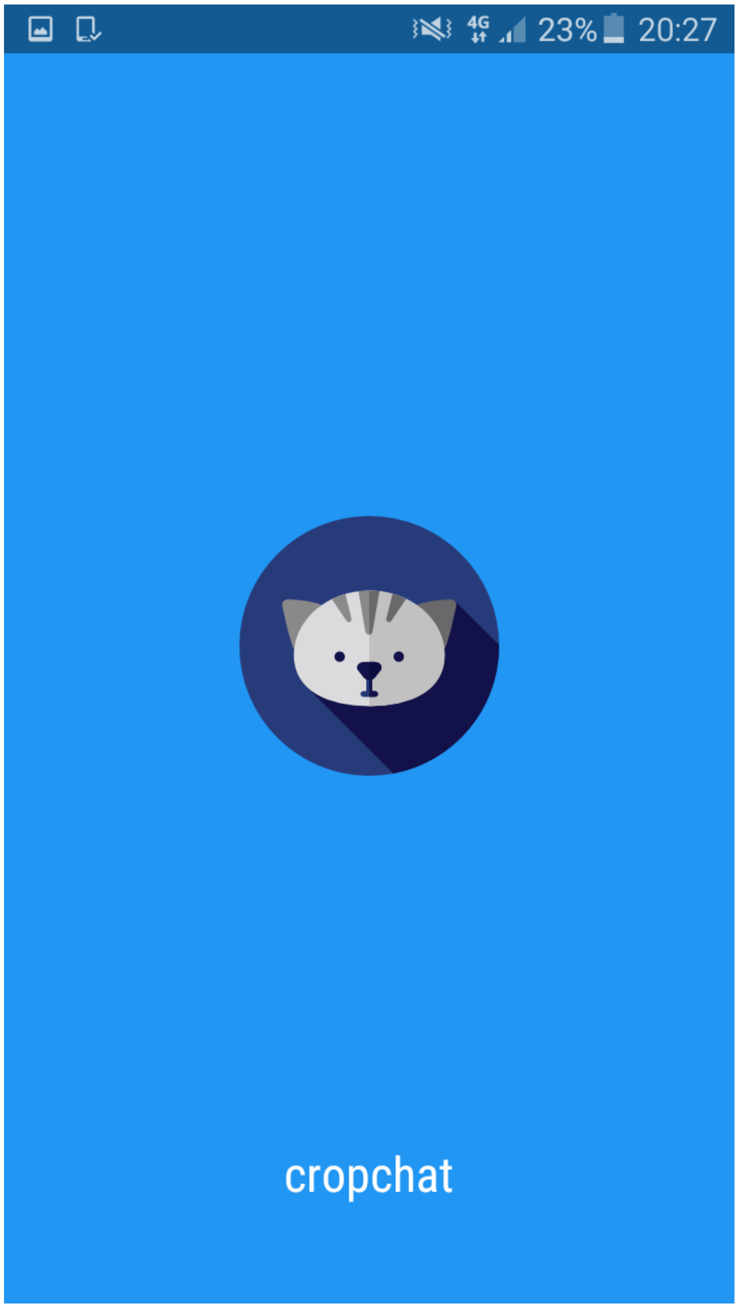
That should give you the follolwing output:

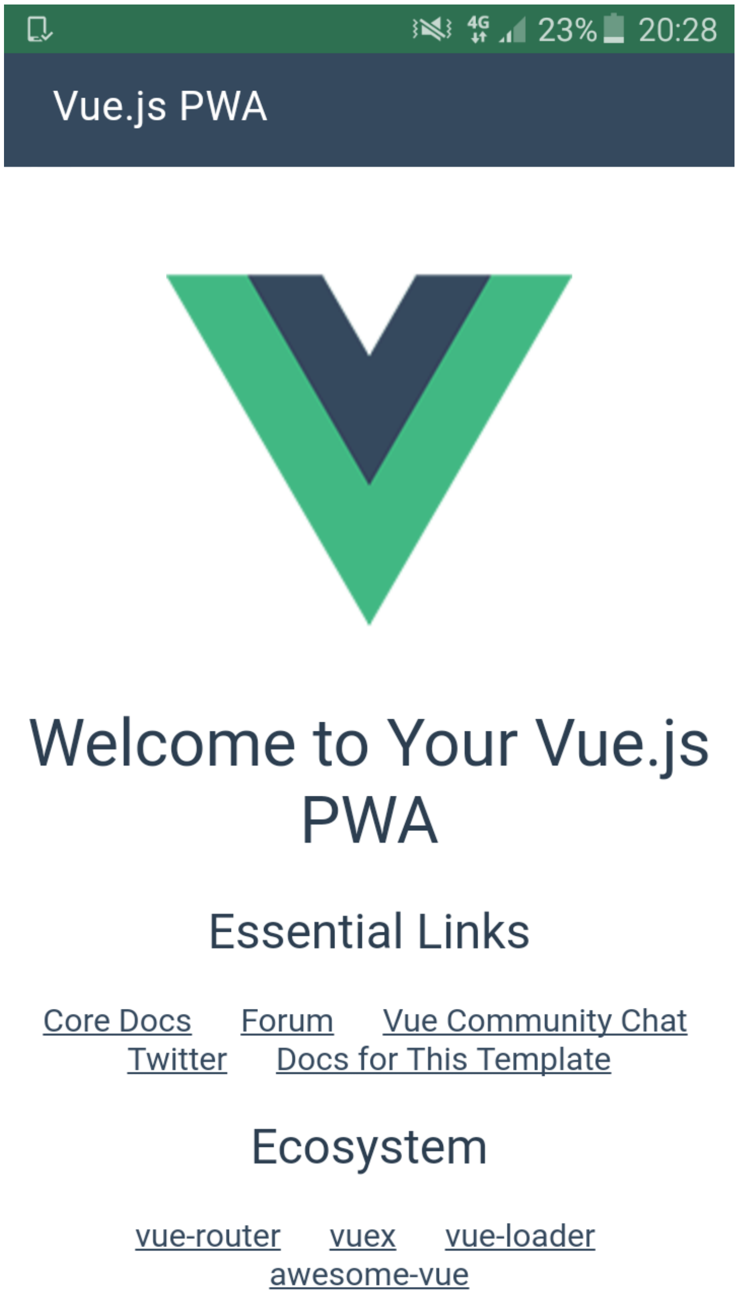
```
ngrok by @inconshreveable
(Ctrl+C to quit)

Session Status                online
Version                       2.1.18
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://5ef29506.ngrok.io ->
localhost:8080
Forwarding                    https://5ef29506.ngrok.io ->
localhost:8080

Connections                   ttl     opn     rt1     rt5
p50     p90
                              39      3       0.01    0.01
120.01  881.89
```

Browse ngrok url `http://5ef29506.ngrok.io` with your smartphone. You can add it to your device desktop!


23% 20:26
ps://d5154f99.ngrok.io/
Vue.js PWA
Ajouter à l'écran d'accueil
cropchat
ANNULER
AJOUTER
Welcome to Your Vue.js PWA
Essential Links
Core Docs
Forum
Vue Community Chat
Twitter
Docs for This Template
Ecosystem


23% 20:27
cropchat
Téléphone
Chrome
Messenger
Applicati...





Vue.js PWA

Welcome to Your Vue.js PWA

Essential Links

Core Docs Forum Vue Community Chat Twitter Docs for This Template

Ecosystem

vue-router vuex vue-loader awesome-vue

Cropchat's sources are available on GitHub here. Git history follows the tutorial steps.

To learn more about ngrok, you can read Matthieu Auger's article: Expose your local environment to the world with ngrok.

Create view skeleton and handle routing

Now that we have a proper base we are going to start building CropChat features. CropChat has three views:

- Home View: displays cat pictures thread as a list of images
- Detail View: displays a specific cat image details (accessible by click from Home View)
- Post View: enables the user to post a new picture.

Create a `src/components/HomeView.vue` view with following skeleton:

```
<template>
  <ul class="list">
  </ul>
</template>
<script>
export default {
}
</script>
<style scoped>
  .list {
```

src/components/HomeView.vue

Same for `src/component/DetailView.vue` view:

```
<template>
  <div class="card-image">
  </div>
</template>
<script>
  export default {
  }
```

src/components/DetailView.vue

Same for `src/components/PostView.vue` view:

```
<template>
  <div class="waiting">
    Not yet available
  </div>
</template>
<script>
export default {
}
</script>
<style scoped>
```

src/components/PostView.vue

Finally, update routing file `src/router/index.js` :

```
import Vue from 'vue'
import Router from 'vue-router'
import HomeView from '@/components/HomeView'
import DetailView from '@/components/DetailView'
import PostView from '@/components/PostView'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView
    },
    {
      path: '/detail/:id',
      name: 'detail',
```

src/router/index.js

Remove unused Hello.vue view too. You should see the changes impacting your mobile app directly (Hot reload is great, isn't it?)

Install Material Design Lite

Don't you know Material Design Lite? It's a great framework that let you implement Material Design easily and lightly on your web application.

You can find an exhaustive documentation here: Get MDL.io

Update CropChat dependencies:

```
npm install material-design-lite --save
```

Update `src/App.vue` component to import MDL style and load MDL module:

```
<script>
  require('material-design-lite')
  ...
</script>
<style>
  @import url('https://fonts.googleapis.com/icon?family=Mate
```

src/App.vue

Provide your Single Page Application with a Navigation Bar:

Update main component `src/App.vue` template section:

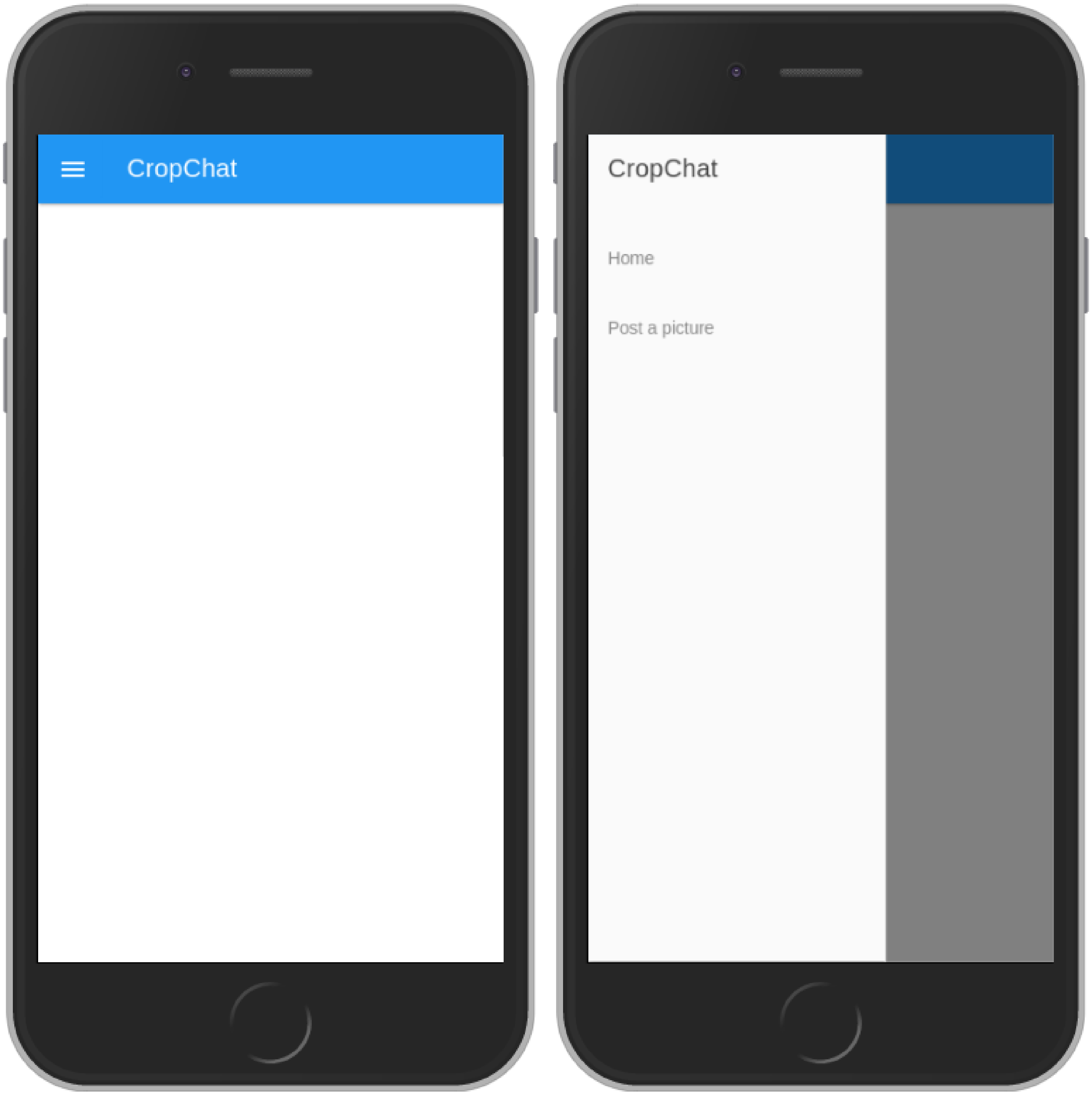
```
<template>
  <div class="mdl-layout mdl-js-layout mdl-layout--fixed-he
    <header class="mdl-layout__header">
      <div class="mdl-layout__header-row">
        <span class="mdl-layout-title">CropChat</span>
      </div>
    </header>
    <div class="mdl-layout__drawer">
      <span class="mdl-layout-title">CropChat</span>
      <nav class="mdl-navigation">
        <router-link class="mdl-navigation__link" to="/" @c
        <router-link class="mdl-navigation__link" to="/post
      </nav>
    </div>
    <main class="mdl-layout__content">
```

src/App.vue

Because Material Design Lite is not designed specifically to build Single Page Application, we need to hide the burger menu when user clicks on a menu link:

```
<script>
...
export default {
  name: 'app',
  methods: {
    hideMenu: function () {
      document.getElementsByClassName('mdl-layout__drawer')
      document.getElementsByClassName('mdl-layout__obfuscat
    }
```

src/App.vue



Populate views and bring application to life

We are not yet connected to a backend server. We are going to use fake data as of now.

Create a `src/data.js` file:

```
export default {
  pictures: [
    {
      'id': 0,
      'url': 'https://25.media.tumblr.com/tumblr_m40h4ksiUa
      'comment': 'A cat game',
      'info': 'Posted by Kevin on Friday'
    },
    {
      'id': 1,
      'url': 'https://25.media.tumblr.com/tumblr_lhd7n9Qec0
      'comment': 'Tatoo & cat',
      'info': 'Posted by Charles on Tuesday'
    },
    {
      'id': 2,
      'url': 'https://24.media.tumblr.com/tumblr_m4j2atctRm
      'comment': 'Santa cat',
      'info': 'Posted by Richard on Monday'
    },
    {
      'id': 3,
```

src/data.js

Import data in `HomeView.vue` script section and link pictures to their corresponding DetailView:

```
<script>
  import data from '../data'
  export default {
    methods: {
      displayDetails (id) {
        this.$router.push({name: 'detail', params: { id: id
      }
    },
    data () {
      return {
```

src/components/HomeView.vue

Update `HomeView.vue` template and style:

```
<template>
  <div>
    <div class="mdl-grid">
      <div class="mdl-cell mdl-cell--3-col mdl-cell mdl-cel
      <div class="mdl-cell mdl-cell--6-col mdl-cell--4-col-
        <div v-for="picture in this.pictures" class="image-
          <div class="image-card__picture">
            <img :src="picture.url" />
          </div>
          <div class="image-card__comment mdl-card__actions
            <span>{{ picture.comment }}</span>
          </div>
        </div>
      </div>
    </div>
    <router-link class="add-picture-button mdl-button mdl-j
      <i class="material-icons">add</i>
    </router-link>
  </div>
</template>
...
<style scoped>
  .add-picture-button {
    position: fixed;
    right: 24px;
    bottom: 24px;
    z-index: 998;
  }
  .image-card {
    position: relative;
    margin-bottom: 8px;
  }
```

src/components/HomeView.vue

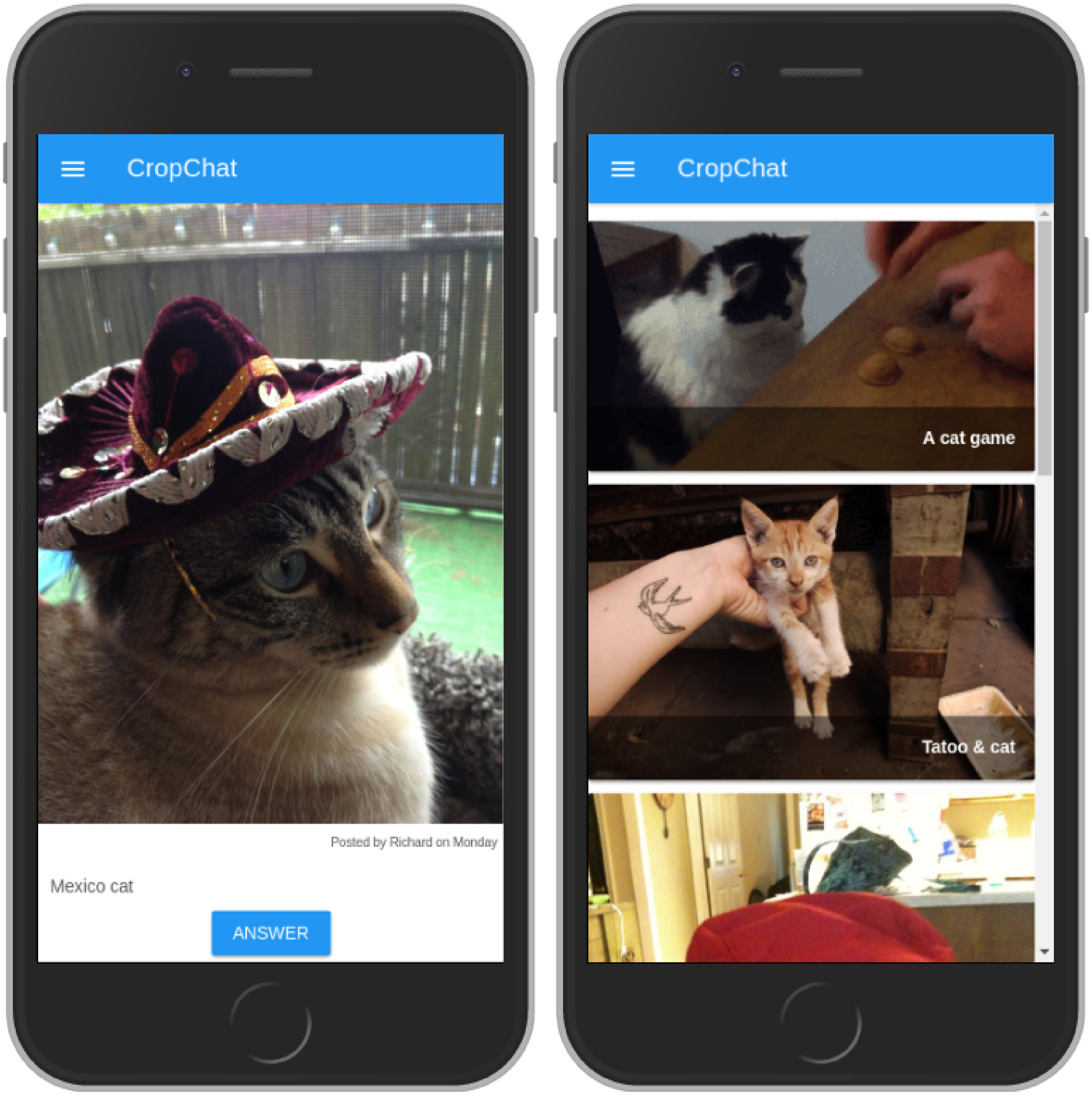
Proceed to the same enhancements on `DetailView.vue` :

```
<template>
  <div class="mdl-grid">
    <div class="mdl-cell mdl-cell--8-col">
      <div class="picture">
        <img :src="this.pictures[$route.params.id].url" />
      </div>
      <div class="info">
        <span>{{ this.pictures[$route.params.id].info }}</s
      </div>
    </div>
    <div class="mdl-cell mdl-cell--4-col mdl-cell--8-col-ta
      <div class="comment">
        <span>{{ this.pictures[$route.params.id].comment }}
      </div>
      <div class="actions">
        <router-link class="mdl-button mdl-js-button mdl-bu
          ANSWER
        </router-link>
      </div>
    </div>
  </div>
</template>
<script>
import data from '../data'
export default {
  data () {
    return {
      'pictures': data.pictures
    }
  }
}
</script>
<style scoped>
```

src/components/DetailView.vue

Final Result

We are done, CropChat is up!



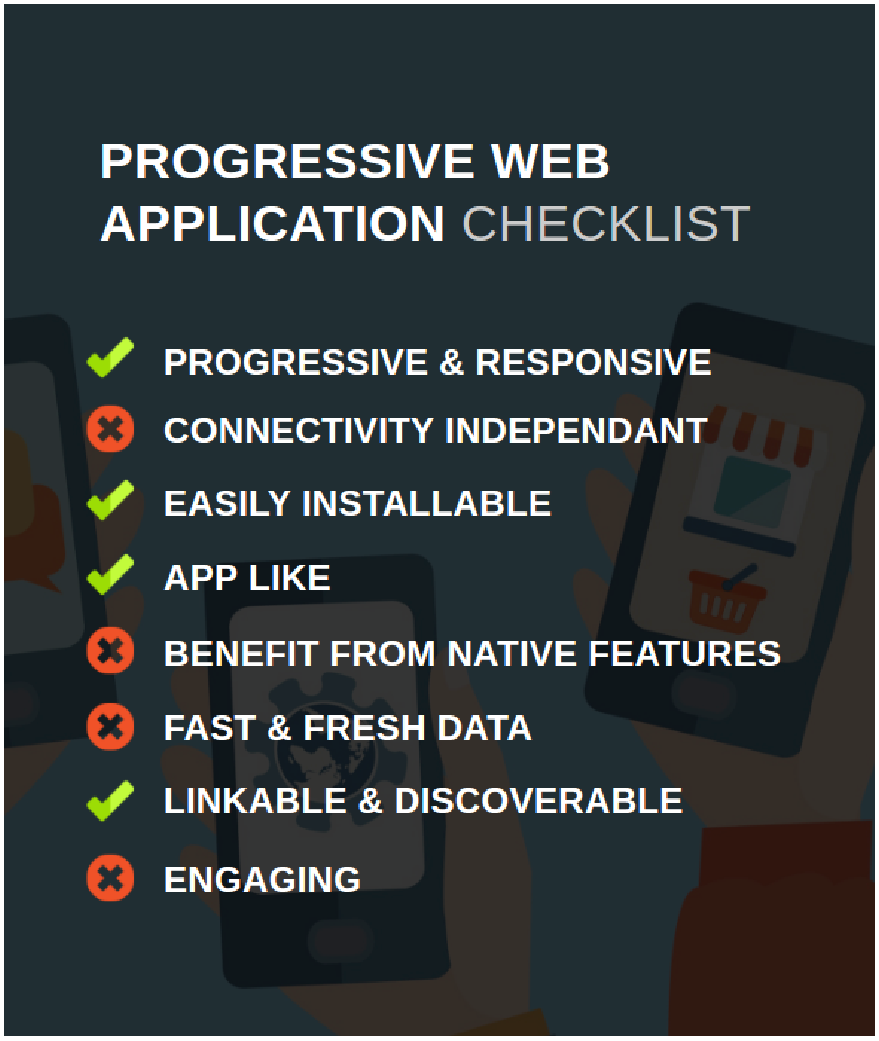
Code source available on this GitHub repository: https://github.com/charlesBochet/vueJSPwa

Conclusions

I hope I have convinced you of the capabilities of VueJS and Webpack to create a web application with very little effort. To sum things up:

- Vue-cli can create a dummy VueJS + Webpack application in one command line
- Make your web application installable by adding a Manifest.json file
- Use Vue-Router and Material Design to create a app-like user experience

However, CropChat is not yet a Progressive Web Application: let's have a look to the PWA requirements checklist:



PROGRESSIVE WEB APPLICATION CHECKLIST

- ✓ PROGRESSIVE & RESPONSIVE
- ✗ CONNECTIVITY INDEPENDANT
- ✓ EASILY INSTALLABLE
- ✓ APP LIKE
- ✗ BENEFIT FROM NATIVE FEATURES
- ✗ FAST & FRESH DATA
- ✓ LINKABLE & DISCOVERABLE
- ✗ ENGAGING

Half of the requirements are not yet met. These are the objectives of the next parts.

You can now switch to Part II that focuses on providing fast and fresh data for our CropChat application.

For those who are in Paris, I am co-organizing a Progressive Web App MeetUp. Do not hesitate to come and say 'Hi!'.

Did you like this article? Feel free to comment or contact me.