App Code

Technology Stack:

- Vue.js
- Ionic – Android
- Capacitor.js
- ThingSpeak IoT Server
- Arduino C++
- GoogleMaps API
- openWeatherMap API

The application front-end is written in Vue.js Javascript framework and with the help of Ionic framework components it converted to Web Application. From there Capacitor.js components bind the web application with native Android and IOS components. In Firduino's case it was done only in Android, due to its need for IOS system to develop it, but due to Ionic and Capacitors' cross-platform technology it can be easily implemented.

The app has no back-end server of its own, as we are using Thingspeak. Thingspeak, developed by MathWorks Inc, offers IoT server services, providing REST API channels, with each channel capable of reading and writing up to 8 sensor values. Of course if put in production, a back-end server will be essential for an authentication login system, and other reasons that will be explained later. The app also uses Google Maps API and openWeatherMap API, to take the local temperature.

Vue.js

Vue.js offers the ability to build easily Apps with scalable structure. It uses webpack to bundle the different files and assets. The package manager is npm (Node), and capacitor and ionic are configured on the according config.json files. Vue-router was used for the routing and vuex for the statemanagement.

Below there is a code example from the base-layout component used for the App (Figure 1).

```
<template>
  <ion-page>
    <ion-header>
      <ion-toolbar>
        <ion-buttons slot="start">
          <ion-back-button :default-href="pageDefaultBackLink"></ion-back-button>
        </ion-buttons>
        <ion-title>{{ pageTitle }}</ion-title>
        <ion-buttons slot="help">
          <slot name="actions-help"></slot>
        </ion-buttons>
        <ion-buttons slot="end">
          <slot name="actions-end"></slot>
        </ion-buttons>
      </ion-toolbar>
    </ion-header>
    <ion-content>
      <slot />
    </ion-content>

  </ion-page>
</template>
```

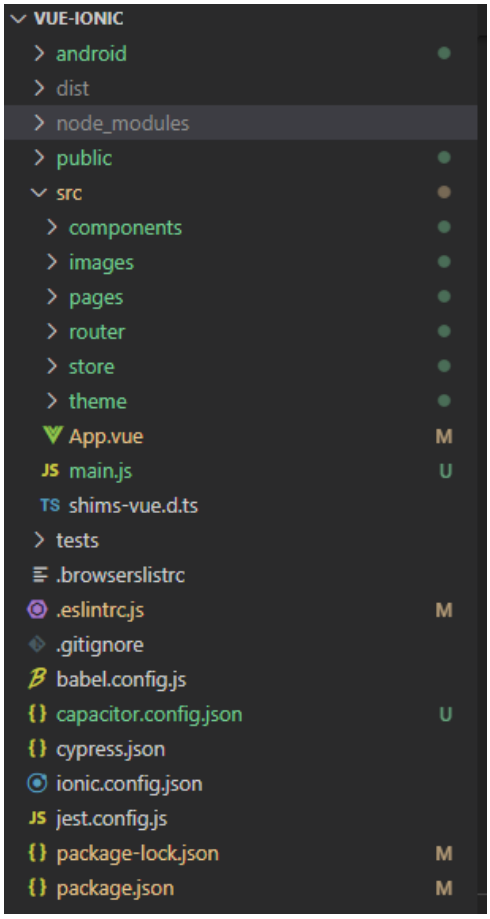

Figure 1: Vue-Ionic, Base-Layout example

Ionic

Ionic is a front-end cross-platfrom alternative to native app languages. Its features are specifically web components that fit best with mobile screens.

It offers a variety of custom components, some of which are essential for the web-app format (IonPage, IonHeader, IonContent, IonTitle), and some just adding utilities and ensuring the best performance on mobile devices (IonButton, IonGrid, IonIcon, IonImg, IonCard).

The Ionic development team, has firm guidelines on their page, while some of them are specifically written for different Javascript frameworks like Vue.js .

Although, the best utility Ionic offers, is its cross-platfrom access device features, which are accessed through Capacitor.js.

Capacitor

Capacitor is an open source native runtime for building Web Native apps, Create cross-platform iOS, Android, and Progressive Web Apps with JavaScript, HTML, and CSS. Just by importing a component and running 2-3 inherited functions, you are able to access and call native mobile servies.

In our App we used 3 of them:

- LocalNotifications
- Geolocation
- Text-to-Speech

LocalNotifications, were used to alert the user of a possible fire detected by one of the sensors, on GridPage.vue component.

Geolocation, is used to obtain the mobiles coordinates in order to place a new sensor on integrated GoogleMaps map.

Text-to-Speech-community, is not an official plugin as the name suggest, but is developed by the capacitor community. It is used to ensure accessibility of the app, and it alerts the user from which direction the possible fire is.

After having the ionic and capacitor code ready, with npx cap copy → npx cap open android, the code is pushed to Android Studio, where the Android app is built and can be tested on its emulators. It needs small adjustments on the android code, mainly on AndroidManifest.xml which handles the user perimissions.

API Calls

The app makes 3 different API Calls.

ThingSpeak server

The ThingSpeak server, offers end points to get raw channel values, as well as dynamic diagrams of every price. The app uses the dynamic diagram call for its diagrams, and the update end point, to get the latest sensors values, in order to calculate if there is a fire.

OpenWeather

With OpenWeather API, the app gets the local temperature , as provided by meteorologists. Then it compares it with the sensor value and if there is a significant difference, it sets the alert for fire.

GoogleMaps

The app uses the GoogleMaps API , and actually does it through the VueGoogleMap component. The map is centered at the home/station location, and it offers the ability to place markers for new sensors placed by obtaining the coordinates from the mobile.

With the use an algorithm, we also calculate the horizon points of the sensor, in order to alert the user of where the fire is, and there are also dynamic colors on the markers, to indicate fire,warning or safe.

Arduino C++

Arduino code is written in C++ in the Arduino IDE. It gets the sensor values as analog inputs, while the ESP8266 wifi-module, establishes a connection to a WiFi. After it sends a GET request to the Thingspeak server with the sensor values, as defined in the ThingSpeak documentation.

Production Changes

If we want to go for productions there are some changes need to be done.
First of all there is the need for a server, as thingspeak can only host a certain amount of channels. We also need a login system.

In addition to that, the app needs a push-notification android component rather than the local-notification we used. Push-notification is triggered by an external API and pops a notification even when the app is closed. We used the local one because we have no server.

Some more changes would be needed as the code isn’t 100% dynamic.