Develop an Android app

Can you create an app with just a Raspberry Pi and install it on your Android phone? Of course you can, and here's how...

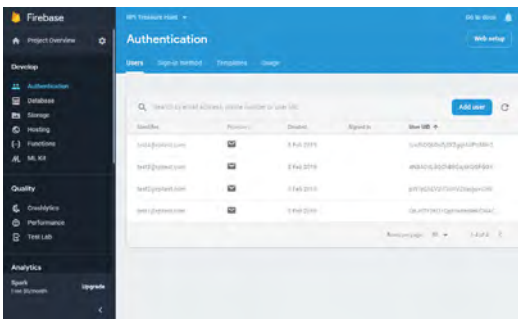
MAKER

Mark Vanstone

Educational software author from the nineties, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by the Raspberry Pi!

magpi.cc/YiZnxL

The Raspberry Pi is generally considered to be a great platform for learning how to program and control electronics projects, but can you create programs to be used on other platforms like PCs or even mobile phones? Well, considering the cost of buying a Raspberry Pi, you may think it could be a bit limited for real-world development, but this article will explain how you can use your Pi to create a real-world, data-driven, useful mobile app that you can install on a phone and use to communicate in real-time with your Raspberry Pi.



▶ In the Firebase Console, under Authentication, you can set up users for your app

There are several methods of making mobile apps. Generally, the way to produce an app is to write it using a development system which compiles your code into a package for a specific platform. These are called 'native' apps and you would need to compile different versions for Android phones and iPhones or Windows phones. There are other apps which have a native app wrapper but then display HTML5 pages inside the wrapper or maybe even just work inside a web browser. These are called 'web' apps.

The problem with web apps is that they rely on having a connection to the internet to view the HTML pages, and any data handling will generally require calls to server-side scripts and databases, as with normal webpages. Now, there is a solution to this problem and it is called 'PWA'. That stands for 'progressive web app' and has elements of the way a native app works, but can be programmed using webpage coding.

A PWA can keep a copy of the app screens and even a local copy of database data so that when there is no internet connection it still works. It can also be installed onto a mobile phone, with its

own icon on the home screen and, when launched, looks and works like a native app.

To write a PWA on the Raspberry Pi, and a system to capture data from the app and then make external electronics do things in response, we are going to get the help of a framework by Google called Firebase. .

> We will make a treasure hunt app for mobile phones

So here's the plan: using Raspbian, we will install the necessary modules on the Raspberry Pi and set up a new Firebase project. We will make a treasure hunt app for mobile phones which will talk to a Python program on the Raspberry Pi. That program will use the GPIO pins to light up coloured lights for each team as they find hidden treasure or complete the treasure hunt. All of this can be developed and deployed from the Raspberry Pi!

WHO IS SUPPORTING **PWA?**

This is a very simple example and the scope of what can be done with progressive web apps is much greater. For a lot of mobile app projects, this framework will be quite sufficient and perhaps it needs to be asked if it is necessary to submit apps to the App Store or Play Store (and the cost incurred in that) when a simple link can be sent to a user who can install a functional app that can look and feel like a native app. There is extensive documentation on the Firebase site covering all the elements of the framework. Google is committed to supporting PWA technology, and Microsoft has said that it will support it too. Although Apple is less vocal about its support, a PWA shortcut does install an app on an iPhone and the PWA acts to the user like a native app. To see a list of what is available through a PWA, view the chart at **whatwebcando.today**. Excepting a few hardware-specific functions of a mobile device, the list provides a great range of functions which will no doubt expand as more PWAs are made and used.

Making the app

A step-by-step guide to building your mobile app

You'll Need

- Raspbian (latest version)
- Node.js **magpi.cc/ogOWTs**
- firebase-tools **firebase.google.com**
- Pyrebase Python module **magpi.cc/OPQUtg**
- Android phone
- LEDs, resistors, jumper wires, and breadboard

01 Getting the right version of Node.js

The first thing to do is get Node.js installed. Unfortunately, currently the version that is available via the apt-get command is not up-to-date enough, so we need to do a manual install. Head over to **magpi.cc/ogOWTs** and download the ARM package. To find out which version you need, type `uname -m` into a Terminal window. Most Pi models will need the v7 package. You will need to unpack the file to a suitable directory (home directory is fine) and then, in the Terminal window, go to the unpacked directory with, for example, `cd node-v10.15.1-linux-armv71`. Now type `sudo cp -R * /usr/local/` to copy the files over.

02 Get Firebase tools

Firebase needs to use Node.js for its toolset and once you have the Node files in place, you can check that they are ready to use by typing `node -v`, which should reply with the version you have just installed (10.15.1 in our case). You can also check the Node Package Manager by typing `npm -v`. We will need npm to install the Firebase tools. We do that by typing `sudo npm install -g firebase-tools` into the Terminal window. The install will take a little time; when it's finished, run the same command again, to update one last package.

03 Get a Firebase account

Google very kindly provides basic access to its Firebase service for free. All you need to do is register for an account at **firebase.google.com** and you'll be able to do all the things in this tutorial. You can do this by signing in with your existing Google account, if you have one. Google also provides a sliding scale payment system if you need to expand your requirements. When you have your account, you'll be able to access the Firebase console at **console.firebase.google.com**. On the console front page, you'll see an option to add a new project.

04 The new project

Select the 'Add Project' button and you can set up your Firebase project. You'll need to type in a project name and accept the terms and conditions, then select 'Create project'. When the project has been created, you'll be taken to the Firebase console where you'll see a range of tools down the left-hand side. We will be using the 'Develop' tools which should be shown here. First, let's look at Authentication. You'll need to set a sign-in method, for which 'Email/Password' can be enabled. Then, in the 'Users' tab, you can Add a user. Also at this stage, go into the Database section and create a Firestore database.

05 Storage

Firebase offers three categories of data storage. The first is database storage from which we can use either Firestore or the Realtime Database (more on this later). The second is labelled as 'Storage' in the menu, which provides

manifest.json

Language: **JSON**

```
{
    "short_name": "RPiTreasure",
    "name": "RPi Treasure Hunt",
    "icons": [
      {
        "src": "/images/rpit192.png",
        "type": "image/png",
        "sizes": "192x192"
      },
      {
        "src": "/images/rpit512.png",
        "type": "image/png",
        "sizes": "512x512"
      }
    ],
    "start_url": "/",
    "background_color": "#fff",
    "display": "standalone",
    "scope": "/",
    "theme_color": "#fff"
  }
```

treasure.py

DOWNLOAD THE FULL CODE: magpi.cc/OdnyDP

Language: **Python**

```
001. import pyrebase
002. import time
003. from gpiozero import LED
004.
005. config = {
006.     "apiKey": "Your apiKey goes here",
007.     "authDomain": "Your hosting domain goes here",
008.     "databaseURL": "Your hosting URL goes here",
009.     "projectId": "Your project id",
010.     "storageBucket": "Your storage domain",
011.     "messagingSenderId": "Your sender id"
012. }
013.
014. firebase = pyrebase.initialize_app(config)
015. numberOfTreasure = 3
016. led = {}
017. teams = {}
018. teams[0] = {"email":"test1@rpitest.com",
     "led":17,"treasure":[]}
019. teams[1] = {"email":"test2@rpitest.com",
     "led":18,"treasure":[]}
020. teams[2] = {"email":"test3@rpitest.com",
     "led":22,"treasure":[]}
021. teams[3] = {"email":"test4@rpitest.com",
     "led":23,"treasure":[]}
022.
023. for f in range(len(teams)):
024.     led[f] = LED(teams[f]["led"])
025.     led[f].off()
026.
027. db = firebase.database()
028.
029. def processMessage(d):
030.     if(d != None):
031.         for v in d.values():
032.             updateTeam(v["email"], v["item"])
033.
034. def ledFlash(t):
035.     for f in range(5):
036.         led[t].on()
037.         time.sleep(.2)
038.         led[t].off()
039.         time.sleep(.2)
040.
041. def ledOn(t):
042.     led[t].on()
043.
044. def updateTeam(t,i):
045.     for td in teams:
046.         if teams[td]["email"] == t:
047.             if i not in teams[td]["treasure"]:
048.                 teams[td]["treasure"].append(i)
049.                 if len(teams[td]["treasure"]) >=
     numberOfTreasure:
050.                     print(t+" complete!")
051.                     ledOn(td)
052.                 else:
053.                     ledFlash(td)
054.
055. def streamHandler(message):
056.     if message["event"] == "put" or
     message["event"] == "patch":
057.         processMessage(message["data"])
058.
059. myStream = db.child("msg").stream(streamHandler)
060.
061. while 1:
062.     time.sleep(.1)
```

an area to store files generated by an app. The third, which we will look at now, is 'Hosting'. If we go into the Hosting section, we'll see instructions about installing firebase-tools, which we did previously. Then there are instructions about how to set up the project on your Raspberry Pi – so let's follow along with them.

06 Local projects

First, in our Terminal window, make a directory for our app with `mkdir appname`. Then `cd appname` to go into that directory. From the Firebase instructions, type `firebase login`. A browser window will appear and you'll be asked to log in to your Google/Firebase account. With that done, go back to the Terminal window and type `firebase init`. You'll be asked which features you'd like to use, which could be all of them for the moment. It will ask you to select your project, then it'll ask you a string of questions about your project. Just select the default option (press **ENTER**) for all of the questions.

07 Setup done

Now Firebase has set up a default project for us, which has everything we need to build our app. Go back to the Firebase Console and select 'Finish' (we'll deploy a bit later). Inside our app folder we will find another folder called **public**. Inside this we have our **index.html** file, which is where our app

sw.js

Language: **JavaScript**

```
var cacheName = 'rpitreasure';
var filesToCache = [
  '/',
  '/index.html',
  '/images/logo.png',
];

self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
  e.waitUntil(
    caches.open(cacheName).then(function(cache) {
      console.log(
'[ServiceWorker] Caching app shell');
      return cache.addAll(filesToCache);
    })
  );
});

self.addEventListener('activate',  event => {
  event.waitUntil(self.clients.claim());
});

self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request,
{ignoreSearch:true}).then(response => {
      return response || fetch(event.request);
    })
  );
});
```

will be built. Let's have a look at that file: open your favourite programming editor (you could try Geany if you are undecided) and load the **index.html** file. Now go to the Firebase console, select the cog next to 'Project Overview' and select 'Project Settings'.

08 Add some Firebase

In the Project Settings page, near the bottom you'll see a panel saying that there are no apps in your project. Select the round web icon (</>) and you'll get a pop-up with some JavaScript code. Copy that code and go back to editing your **index.html** file. Now replace the code that starts `<!-- update the version number` and ends `/init.js></script>` with the code you have just copied from the Firebase console. This bit of code will connect our web app with the Firebase services. We can now deploy our test app by going back to our Terminal window and typing `firebase deploy`. After uploading the necessary files, the app will be ready for testing in a browser.

IS THIS ANY GOOD FOR REAL APPS?

Of course, you will get nay-sayers proclaiming that if it is not a native app then it's not a real app, and the restrictions of the App Store and Play Store currently mean that it is difficult to adapt a PWA for distribution that way. The fact of the matter is that in industry, the cost of any project is key to it happening and while native app development is a very costly activity, PWAs can be coded in a fraction of the time and have no restrictions of the App Store or Play Store. That's not to say that PWAs cannot be published through those portals, it's just that they have to be wrapped in a framework like Cordova to make them into native apps.

For the maker/coder community, the ability to publish apps without these restrictions is no doubt a benefit and possibly a technology that will supersede the platform-dependent stores. If nothing else, PWA technology gives us a great opportunity to create useful apps with just a Raspberry Pi.

09 Testing testing

We can test on the Raspberry Pi Chromium browser or on a mobile device, but make sure that you are using either Chrome or Chromium. Other browsers do support PWAs, but let's keep it simple for now. After the project has been deployed, we'll be given a web address of the hosting URL in our Terminal window. Go to this address in a browser and you should see confirmation that the app is working and has connected. You can also test locally if you set up the Chrome Web Server extension. If you don't see a message saying 'Firebase SDK loaded with auth, database, messaging, storage', then go back over the previous steps

10 Let's make an app

The first thing to code will be the app that goes on the phone. Have a look at the listing of **index.html**. What this script does is to present a

index.html

Language: **HTML**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="manifest" href="/manifest.json">
    <title>Welcome to The RPi Treasure Hunt</title>
<script src="https://www.gstatic.com/firebasejs/5.8.1/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "Your apiKey goes here",
    authDomain: "Your hosting domain goes here",
    databaseURL: "Your database URL goes here",
    projectId: "Your project id",
    storageBucket: "Your storage domain",
    messagingSenderId: "Your sender id"
  };
  firebase.initializeApp(config);
</script>
<script>
  if('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js')
      .then(function() {
            console.log('Service Worker Registered');
      });
  }
</script>
    <style media="screen">
      body { background: #fff; color: #000; font-family: Helvetica, Arial, sans-serif; margin: 0; padding: 0; text-align: center; background: url(images/logo.png) no-repeat 50% 300px fixed; background-size: 50% }
      #title{ background: #000;color:#fff}
      #signin{ margin:10px; padding:10px;}
      #signout{ position: fixed; bottom: 0px;text-align: center}
      .clueholder{ background:#ca0d4c; color:#fff; border:10px solid #fff; margin:0px; padding:10px;-webkit-border-radius: 15px; -moz-border-radius: 15px;border-radius: 15px;}
      .theClue{display:none}
      #loginform{background: #ca0d4c;color:#fff;padding:10px;text-align: right;width:300px;margin: auto;-webkit-border-radius: 3px; -moz-border-radius: 3px;border-radius: 3px;}
      #load {
        display: block; text-align: center; background: #000; text-decoration: none; color: white;
        position: fixed; bottom: 0;padding-top: 5px;padding-bottom: 5px;
        width: 100%; height:18px;
      }
      input{ -webkit-border-radius: 3px; -moz-border-radius: 3px;border-radius: 3px;margin:3px;padding:2px}
      button {
        border:1px solid #000; -webkit-border-radius: 3px; -moz-border-radius: 3px;border-radius: 3px;font-size:12px;font-family:arial, helvetica, sans-serif; padding: 10px 10px 10px 10px; text-decoration:none; display:inline-block;font-weight:bold; color: #fff;
        background-image: -webkit-gradient(linear, left top, left bottom, from(rgb(77, 77, 77)), to(rgb(29, 29, 27)));;
      }
    </style>
  </head>
  <body>
    <div id="title">
      <h1>RPi Treasure Hunt</h1>
    </div>
    <div id="content">
      Loading RPi Treasure Hunt App.
    </div>
    <p id="load">Connecting ...</p>
    <script>
      clues = null;
      email = "";
      document.addEventListener('DOMContentLoaded', function() {
        try {
          let app = firebase.app();
          let features = ['auth', 'database', 'messaging', 'storage'].filter(feature => typeof app[feature] === 'function');
          document.getElementById('load').innerHTML = `Connected to Treasure Hunt.`;
          firebase.firestore().enablePersistence();
        } catch (e) {
          console.error(e);
          document.getElementById('load').innerHTML = 'Error connecting to the Treasure Hunt.';
        }
      });
      firebase.auth().onAuthStateChanged(function(user) {
        window.user = user; // user is undefined if no user signed in
        console.info("user changed - is now "+user);
```

```
074.          if (user == null){
075.            setLoginPage();
076.          }else{
077.            document.getElementById('load').style =
      "display:none"
078.            getClueData();
079.          }
080.        });
081.        function signinUser(){
082.          email = document.getElementById("email").
      value
083.          password = document.getElementById("pass").
      value
084.          firebase.auth().
      signInWithEmailAndPassword(email, password)
085.          .catch(function(err) {
086.            console.error(err);
087.          });
088.        }
089.
090.        function getClueData(){
091.          var db = firebase.firestore();
092.          db.collection("Clues").get().
      then(function(querySnapshot) {
093.            setCluesPage(querySnapshot);
094.          });
095.        }
096.
097.        function signoutUser(){
098.          firebase.auth().signOut()
099.          .catch(function (err) {
100.            console.error(err);
101.          });
102.        }
103.
104.        function openThisClue(o){
105.          cluelist = document.
      getElementsByClassName('theClue')
106.          for(c=0;c<cluelist.length;c++){
107.              cluelist[c].style = "display:none";
108.          }
109.          theclue = document.getElementById(o);
110.          theclue.style = "display:block";
111.        }
112.
113.        function foundItem(i){
114.          code = document.getElementById("clueCode_"+i).
      value;
115.          clues.forEach(function(doc){
116.            if(doc.id == i && code == doc.data().code){
117.              console.log("we have a winner");
118.              var newMsgKey = firebase.database().
      ref().child('msg').push().key;
119.              var postData = {
120.                email: email,
121.                item: i
122.              };
123.              var updates = {};
124.              updates['/msg/' + newMsgKey] = postData;
125.              firebase.database().ref().
      update(updates);
126.              document.getElementById("clue_"+i).style
      = "display:none"
127.            }
128.          });
129.        }
130.
131.        function setLoginPage(){
132.          document.getElementById('content').
      innerHTML = "<div id='loginform'>Email:
      <input id='email' type='text'><br>Password:
      <input id='pass' type='password'><br><button
      onclick='signinUser();'>Sign In</button></div>";
133.        }
134.
135.        function getClueDisplay(cluelist){
136.          out = "";
137.          clues = cluelist;
138.          clues.forEach(function(doc){
139.            out += `<div class="clueholder"
      id="clue_`+doc.id+`"><button
      onclick="openThisClue('clueDetail_`+doc.
      id+`')">`+doc.data().name+`</button><div
      id="clueDetail_`+doc.id+`" class="theClue"><p>`+doc.
      data().clue+`</p><p>Treasure Code: <input
      id="clueCode_`+doc.id+`" type="text"><button
      onclick="foundItem(`+doc.id+`)">Found It!</
      button></p></div></div>`;
140.          });
141.          return out;
142.        }
143.
144.        function setCluesPage(clues){
145.          out = getClueDisplay(clues);
146.          out += "<div id='signout'><button
      onclick='signoutUser()'>Sign Out</button></div>";
147.          document.getElementById('content').innerHTML
148.  = out;
149.        }
      </script>
150.
151.    </body>
152.  </html>
```

sign-in page to allow users (defined in our Firebase Authentication) to sign in and then it gets a list of clues from the Firebase Firestore database and displays them. When a user finds an item of treasure, there will be a code number (we will set this in our Firestore) which, if they enter it correctly, will remove that clue from the list and send a message to the Firebase Realtime Database. There are two types of database that Firestore supports: the Firestore that our treasure hunt data is in, and the Realtime Database.

11 Realtime Database

The Realtime Database will trigger an event if another program is listening when data changes. That's how we'll transfer data back to our Raspberry Pi. We need to make an adjustment to the security rules to make this easy. Note that this is not a good idea for production purposes, but we can change the security so that we can read and write to the database without authentication. We do this by going to the Realtime Database in the Firebase console, selecting the 'Rules' tab, and changing the '.read' value to `true` – and the same with the '.write' value. Normally we would want to have a bit more security around a database, but for this example we'll keep it simple.

12 Make the app act like an app

There are a couple of other things we need to add to make our app installable on a mobile phone. The first is the **sw.js** file. This is a service worker file and enables us to cache files so that they don't need an internet connection to load. You'll see the script to register this file in **index.html**. The other is a manifest file that allows the app to be installed and will produce a message asking the user if they want to install it. The manifest file is linked near the top of **index.html**.

13 The Python connection

We need to gather the data that is being created by our app – this could be done with a simple webpage on our Raspberry Pi, but that would be boring. Why not have a score indicator with flashing lights and things like that? To make this, we'll need to write a Python program. Have a look at **treasure.py** and see how we do this… but wait: we need to have access to a module called pyrebase, which is a wrapper for the Firebase functions and makes it really easy for us. To install it, just type `pip3 install pyrebase` into a Terminal window.

14 Wiring up

To be honest, you could have any sort of grand scheme for a scoreboard, but we can wire up an LED for each team or player in our treasure hunt. See the wiring diagram (**Figure 1**) we are using, which will flash the team's LED when they discover treasure and keep the LED lit when they have finished. If you have lots of players, you may want to use LED strips or even a much larger build.

15 What does this mean?

What we have done is to create a real-world app for mobile devices with just a £35 Raspberry Pi and a free Google account. This is quite a new development and, even as this article was being written, the Firestore service went from beta to live. This means that things may change quite quickly, so you may need to refer back to the Firebase website for updates.

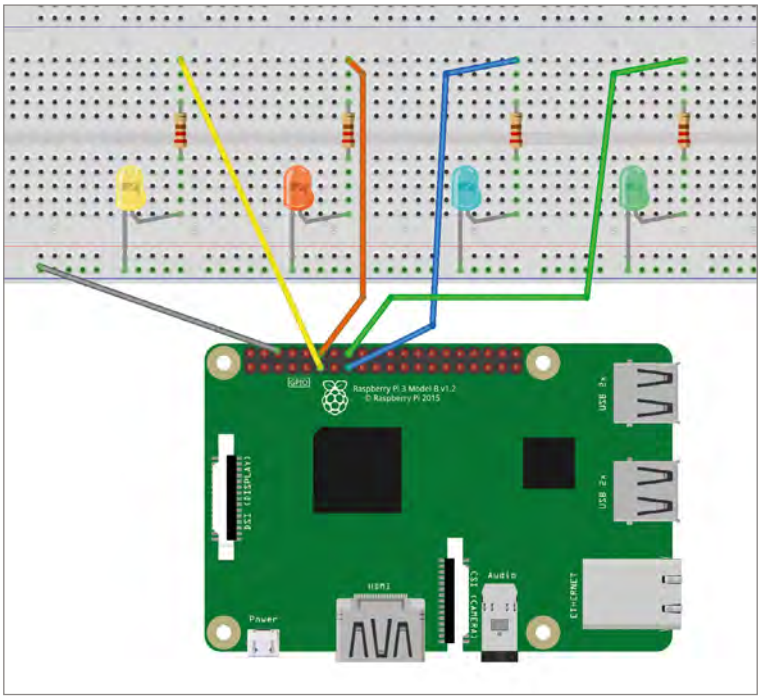


Figure 1 How to connect four LEDs. You can flash the LEDs when data is received using GPIO connections