Digitr

Technical Whitepaper

Omar Elamri

1 Introduction

1.1 What is Digitr?

Digitr is a digital student hallway pass app that provides a smart, simple, scalable and convenient way of managing student passes. Used by over 100 schools across the nation, Digitr keeps students safe and accountable by having them electronically check in with their teacher before leaving the classroom. Once they come back to the classroom, they electronically check in again—ensuring that teachers and administrators know where students are at all times. This is especially tantamount to contact tracing during the COVID-19 pandemic.

1.2 History of Digitr

6 years ago, my STEM teacher came up to me and asked if I could make an app to replace their system of paper hall passes. Of course, I accepted and was very excited to start working. The "zeroth" version of Digitr (then called mPass) was an extremely rudimentary app created in Android App Studio. My teacher was excited, but clearly we'd need something more robust. I decided to take online lessons in iOS App Development and Python in hopes my new skills would be able to create the app.

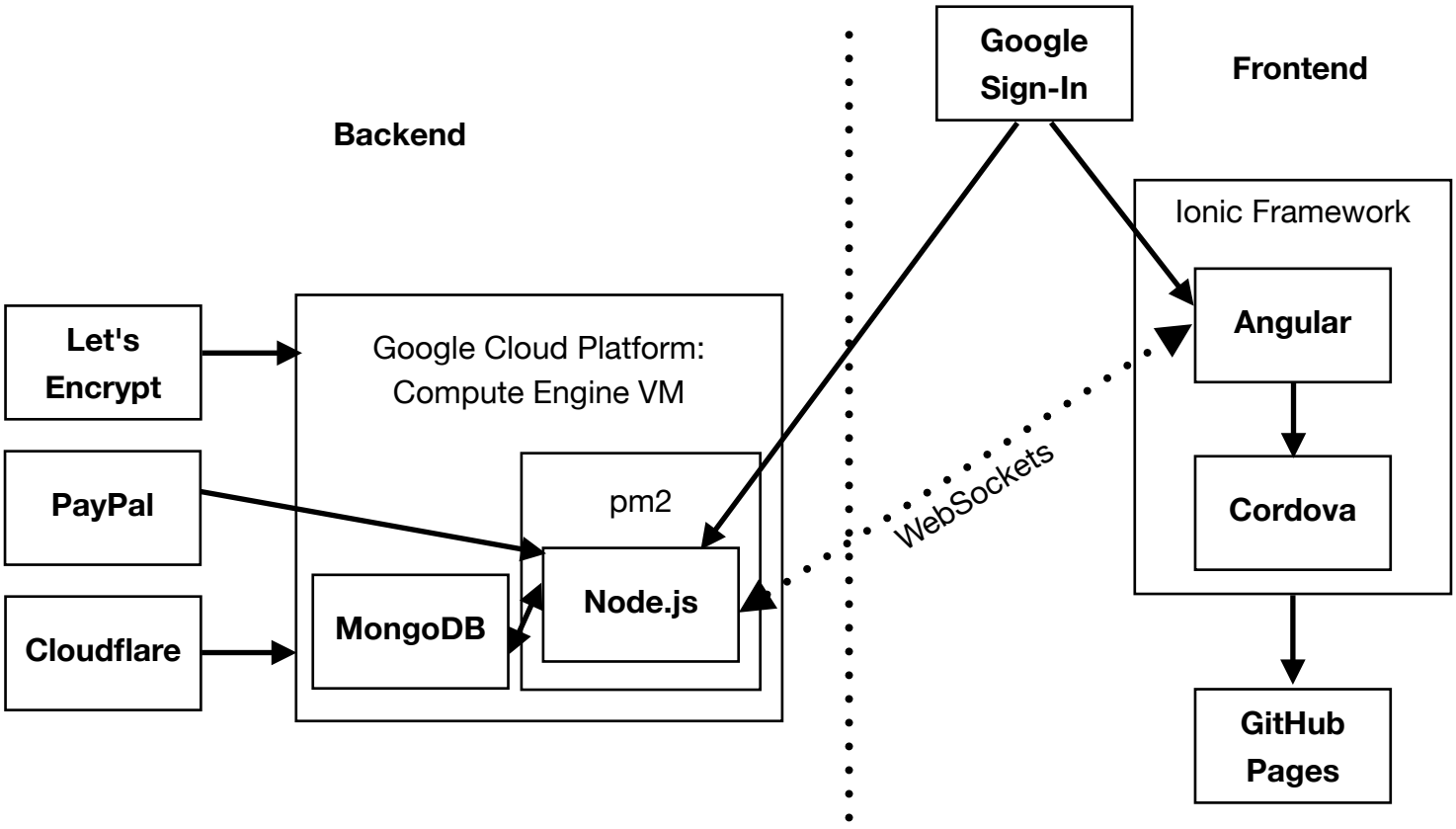
After completing those lessons, I created the first version of Digitr with the help of one of my classmates. It was based on iOS's UIKit and used the Python library Flask as the backend. There were a lot of problems with this first design. Firstly, it didn't integrate with our school's system, so students would have to create their own accounts. Unfortunately, students never remember their passwords. Secondly, our Flask implementation was incredibly amateurish. Every request was an HTTP GET request with passwords being passed around in plaintext.

With these obvious drawbacks, I created the second version of Digitr. For the backend, I used Google's Firebase platform as a backend. This version is the first to not use Apple's UIKit, but instead the Ionic Framework (Angular based). There were some clear advantages to this new version—the main one being that it was available on every device since Angular uses web technologies. In addition, students could log in via Google. The main drawback, however, was the fact you could easily manipulate the app. The frontend took care of all the database logic, so a simple inspect element could easily change any value. Clearly, a new solution was needed.

I then created the third and current version of Digitr which addressed all these issues. A discussion about this version's use of technology can be found below.

2 Technology Overview

2.1 Diagram



2.2 Backend

- Google Cloud Platform

The backend server is hosted using Google Cloud Platform's (GCP) Compute Engine. GCP creates a Debian 9 virtual machine under a scalable x86 Intel server. GCP also provides a static IP address.

- pm2

pm2 is a process manager for Node.js applications. It creates a Linux daemon that the VM runs on start and automatically restarts in the case of a crash. In addition, it provides the ability to split the TCP connections between multiple threads, and gives telemetry options for system monitoring.

- Node.js

Node.js is a framework for developing server side applications using JavaScript. Under the hood, it uses Google Chrome's V8 engine for its interpreter. This implementation of Node.js has a TypeScript transpiler to convert ES6 TypeScript to ES3 JavaScript.
This application's source code is comprised of two main files: `main.ts` and `responder.ts`. The former file is the entry point for the application. It instantiates connection with the MongoDB database, creates the HTTPS server with the certificate files from Let's Encrypt, and finally listens for WebSocket connections and passes it on to a `responder` object. `responder.ts` houses the `responder` object. Based on the body of the WebSocket frame, it looks for a designated function to craft a response to the client. Some responses are a "subscription" to a document, so any changes will be sent for the duration of the connection.

- Let's Encrypt

Let's Encrypt manages the retrieval and renewal of SSL certificates for the application.

- PayPal

PayPal is the application's payment vendor. There is an upgraded version of Digitr called Digitr Analytics. That is not free, and thus needs a payment vendor to handle payments. PayPal is integrated with the app via its REST API. An invoice is created programmatically, and PayPal sends a redirect URL, which is relayed to the frontend to complete the payment.

- Cloudflare

Cloudflare provides the DNS nameservers for the domain digitrapp.com. It also prevents DDOS attacks by proxying all requests to the backend in addition to enforcing HTTPS and HSTS.

- Google Sign-In

After the frontend has initiated a sign-in request, it will relay a sign-in token to the backend. Using Google's JSON Web Token verifier, the backend can verify the user is indeed the person they claim to be and can extract information such as the name, email, and profile picture from the token.

- WebSockets

WebSockets is a protocol that runs on top of HTTPS. To initiate a WebSocket connection, the client sends an HTTP request to upgrade the connection. A handshake occurs, and the TCP connection stays alive for the entirety of the client's session.

Frames can be sent back and forth without the need of creating a new TCP session. This makes WebSockets particularly useful for realtime communication. All the communication between the frontend and the backend happens over WebSockets.

2.3 Frontend

- Ionic Framework

This framework provides prebuilt UI elements to look like an iOS app while using web technologies like Angular. It also provides a command-line interface to build the app for production, export it to Cordova, or update packages.

- Angular

Angular is a JavaScript framework that provides structure and organization to a large application. Built-in elements such as Routers map URLs to Angular components, and services that provided packaged functionality. Digitr utilizes Angular by separating each page of the app into components and putting backend server connection into its own service.

The backend server connection code Digitr uses is fairly complicated to ensure network interruptions don't affect requests made to the server. Any request that goes out to the server is cached in an array. Once a response is received, that request is popped off the array. If the app detects a change in network connection, it will reconnect to the backend server and send the request. This ensures that every request is eventually sent out—and only one time.

- Cordova

Cordova packages an Angular website into an iOS application. Most times, it's just as simple as putting the website on a WebKit WebView, but sometimes, it's necessary to communicate with native binaries. For example, the Google Sign-In prompt is a native binary. Cordova bridges the Angular code with iOS Swift code to ensure a seamless experience.

- GitHub Pages

GitHub Pages acts as a CDN that hosts the web version of Digitr at app.digitrapp.com.

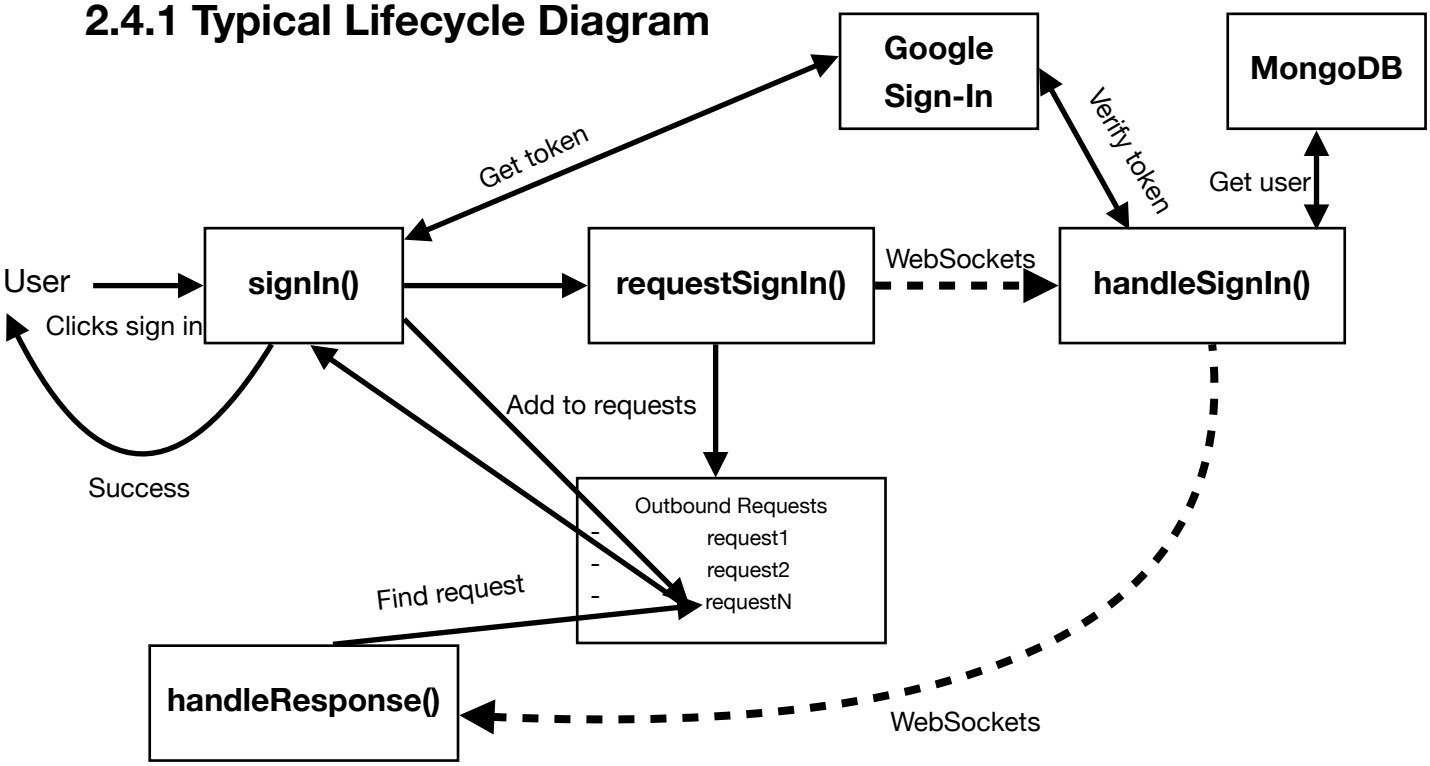
- Google Sign-In

This library serves as Digitr's primary authentication method. GSI provides a simple button that users can click, which redirects them to Google's sign in page. After they sign in, GSI gives the app an authentication token, which the backend can verify.

2.4 Backend-Frontend Communication

WebSockets affords Digitr a seamless way to provide real-time communication to the end user. A typical WebSockets request lifecycle may look like this:

2.4.1 Typical Lifecycle Diagram



2.4.2 Typical Lifecycle Step by Step Process

1. A user clicks sign in.
2. The `signIn()` function will:
    1. Get an authentication token from Google Sign-In after redirecting the user to a Google sign-in page.
    2. Call `requestSignIn()`.
3. The `requestSignIn()` will add a request to an Outbound Requests cache array, which includes a pointer to the originating `signIn()` function. The reasoning for this array is highlighted in section 2.3

4. A WebSockets frame is sent to the backend `handleSignIn()` function, which will
    1. Verify the token sent from the frontend with Google Sign-In
    2. Get the user data from the MongoDB database.
    3. Step 2 will be repeated once it detects changes to the user object. Each repetition implies repetition of the subsequent steps.
    4. Send the user object back to the `handleResponse()` function of the frontend via a WebSockets frame.
5. `handleResponse()` will find a request in the Outbound Requests cache array, and call the pointer to the `signIn()` function with the user data.
6. The `signIn()` function will move on to the main application on successful response.