**Introduction**

This document includes details and plans for implementing our application Dropp from the back-end perspective. This perspective includes everything that the user doesn’t care about. This document outlines the technology necessary to provide our service to our users.

**Back-End Technology**

The core of our back-end technology will be NodeJS, which will implement the logic to send data and responses to the front-end client (user devices). Firebase will be our database to store the data and Node.js will perform operations on the Firebase database.

We currently have the option to host our own Firebase server or use the Firebase service. We will host our Node.js server on the cloud platform provider DigitalOcean. The Node.js server will respond to GET and POST requests from the client.

This is our current design plan; however, should the client-server model need constant communication, we will explore the option to use Socket.IO. Additionally, we will use Amazon Web Services to host our server if DigitalOcean proves to be unsuccessful.

|  |  |
| --- | --- |
| Hosting | AWS or Digital Ocean |
| Database | Firebase |
| Server | NodeJS server or Third Party Service |

* The server will have some form of communication with the client device (either through NodeJS, third party service or other).
* The server respond to the request for content according to the client location.
* The server identity client’s content before being post and stored in database to avoid spam.
* The server clear content with certain age from the database.
* The server validate user’s location before posting the client’s content.

**Front-End Technology**

The front-end aspect of Dropp will Objective-C code for iOS devices, and Java code for Android devices. We will let our back-end service handle data on Firebase; however, we will reserve an alternative option that allows client-side code to access the Firebase database using Firebase libraries.

We also considered using Xamarin to develop front-end code that would run on user devices; however, we decided to develop in native iOS and Android code. Xamarin allows for easier cross-development for the different mobile operating systems, but developing native code allows for greater access to APIs and increased functionality. The only drawback is maintaining separate codebases, but we believe this is worth it given the advantages of developing in the native languages for iOS and Android.

* The receiving device will be a native application capable of communicate with the host server.
* The device’s location service must be on in order for the application to work properly as the application require the user location.
* The device will fetch data from the server according to their location,
* The device will post their content according to their location.

|  |  |
| --- | --- |
| End-User | Mobile Application (Apple or Android) |
| UI | Native iOS/Android or Xamarin |
| Mobile Location Server | Device will be able to tell the server their current location in order for the server to responds with appropriate contents |
| Unique Identification | Device will receive unique id to use to identify their post from other. |
| Communication with Server | GET/POST request to the NodeJS server. Alternatively, client might be able to directly communicate with database if need to. |