SMS Journal Whitepaper

Philip Kiely
GrammieGram LLC

ABSTRACT

Presented within is a system for keeping a journal by SMS text message writing to Google Drive. The system is a Web App, which tend to have three components: a frontend, a server, and a database. SMS Journal is frontendless, in that the daily interaction is handled by existing frontends: SMS and Google Drive, and the signup is handled through the existing Google Authentication widget on an otherwise static website. SMS Journal is serverless, in that is will function by a variety of API calls from AWS Lambda to Google Drive API and either AWS SNS or Twilio.

USE CASE

A user activates their preferred method of sending an SMS text message: by phone (almost any kind), computer, smart speaker, or many other products. They then send a text message to a provided phone number. This text message is a journal entry, which is captured by an API, tagged, and recorded to the user's Google Drive file. By taking this quick and familiar action multiple times daily, the user has created a daily journal for whatever purpose they want. To review recent entries, they review their text message history. To review their journal or edit entries, they log into their existing Google Drive account and open the file that the journal is stored in. To create an account with our service, they sign up with their google account, accept some permissions, and send the first text, a process that should take only minutes.

Expanded use cases include sending pictures and other MMS media to the journal. Google currently offers an API for sheets, and the docs API is in early access. While it will be possible to use the sheets API to implement the service, the docs API would give a better user experience. Furthermore, an authorization key activated by one-time or annual payment could be implemented through Stripe without significantly altering the architecture. Some services on the market send daily prompts/reminders, which could be an extension.

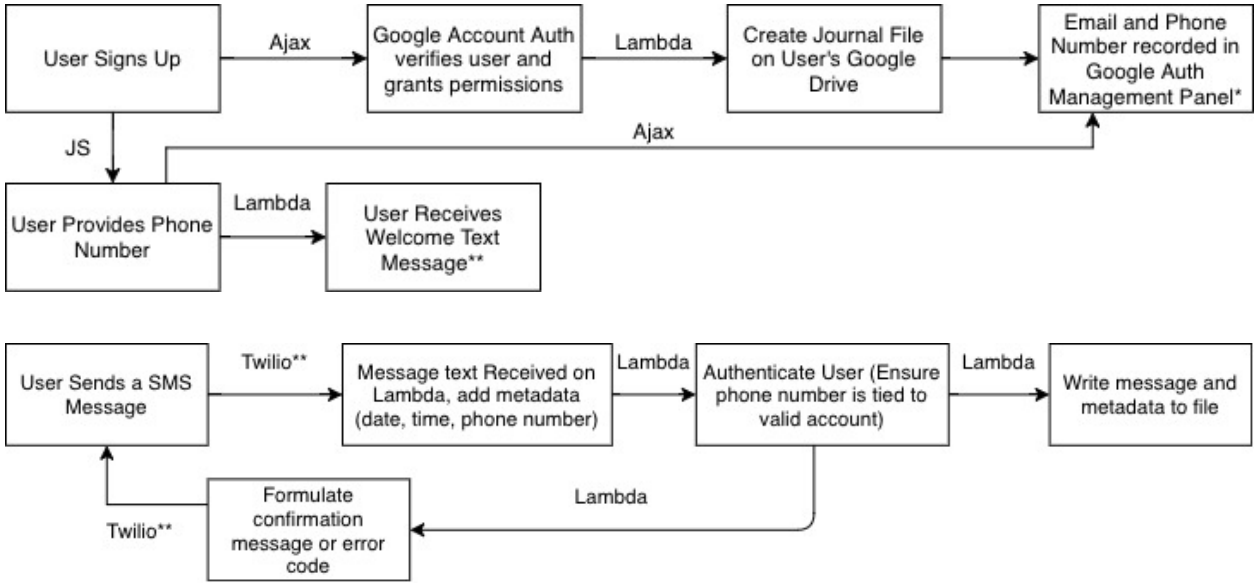
MARKET

Daily journaling is a popular activity in which people record their days and thoughts regularly to achieve a continuous narrative. There are a variety of daily journaling options on the market. Some people prefer to keep paper notes, other prefer digital tools. For digital tools, some people use text processors (Microsoft Word, Notepad), online notebooks (Google Docs,

Evernote), or dedicated journaling software. Day One leads the journaling software industry, and is a paid service at 35 dollars per year. A variety of other services, often open-sourced or self-hosted exist as well. Journaling is a fragmented market where people look for and are willing to pay for products that fit their exact needs, so it is possible to carve a niche.

There are a few products on the market that are similar to SMS Journal. Day One offers an IFTTT integration to let you do basic updates to your Day One journal by text message. This is, of course, only an option for paying customers. Day One pays an undisclosed amount to IFTTT to maintain this integration. Some people build similar custom solutions for themselves using Zapier, another automation as a service provider. TextMyJournal offers a similar user experience, but is currently not accepting new users and hasn't for over a year. Qeepsake is operational and offers all of the proposed front-end features, but with their own back-end, but it is focused on new parents keeping baby journals, leaving most of the market still open.

SYSTEM SCHEMATIC



* For security and/or convenience, we may have to abandon the "databaseless" aspect of the proposal and create a small PostgreSQL authentication database. However, we may be able to use AWS Cognito instead for custom authentication.

** Using either SNS or Twilio, see dependencies for details.

DEPENDENCIES

The system depends on three components: a programmatic SMS messaging service, a variety of AWS serverless infrastructure services, and the Google Sheets or Docs API.

For the programmatic SMS messaging service, Twilio and AWS SNS are the two contenders. The provider must be able to send and receive text messages to a single number (which powers the system for all users) from multiple numbers (users). These messages must be able to trigger AWS Lambda API calls and pass along their contents and metadata. The system should also be able to support MMS if the proposed expansion is implemented. The provider will be selected on the basis of cost and usability.

From AWS, Lambda will provide serverless API calls. Thanks to a generous perpetual free tier, the majority of the logic and processing can be run on AWS Lambda for free at smaller scale. These AWS Lambda calls will be written in Python 3. AWS Cognito may provide authentication, otherwise an AWS RDS PostgreSQL database will be used.

Google provides a Google Sheets API that allows programmatic writes to sheets. There is a similar docs API in limited developer beta. If access is granted to the Docs API and the Docs API is mature and featured enough for the use case, then the system will use AWS Lambda to trigger Google Docs API calls. Otherwise, the system will use the Google Sheets API.

Users will be able to sign up on a static website hosted on GitHub pages with a Hover domain. The website will use AJAX and JavaScript to access the Google Authentication service and make calls to the lambda API.

IMPLEMENTATION DETAILS

Authentication will be handled by Google at sign up, which will generate keys that will allow the system access to specific files. The system assumes that if a SMS number is tied to an account, then an incoming message from that number is from that user. While SMS number spoofing is possible, it is rare and, in many cases, illegal. Furthermore, SMS number spoofing cannot be used to read from other users' data, only write to it. The main website will have a help email for user complaints such as this unlikely instance. The message text will be processed on AWS Lambda, an isolated environment, which protects against injection attacks. Preprocessing and validation on the individual Lambda functions further protects the system. Rate limiting will also be necessary, which may come from the SMS provider, Lambda, or both. Finally, automated testing and error reporting will help secure the system against errors.