
Abstract

The purpose of this project was to create an early literacy messaging service called StoryTime to send illustrated children’s books by text message and messaging apps to families without books at home. It uses a curriculum intervention developed to improve learning outcomes for children ages 2-6 on the program. Parents may be sent digital books through popular messaging platforms. This project is limited to Facebook Messenger and SMS messaging—however, the goal was to create a generalizable system that can integrate new messaging platforms into a single, simple interface to send digital stories. By creating an easy interface to integrate new messaging APIs, it is easier to reach many more parents using the apps they already have.

The basic functionality is simple: sending ordered sequences of text and picture messages to users, scheduling reading reminders and notifications, and handling user preferences like home language and send time. Each messaging platform has unique interfaces and requirements; but fundamentally, they share the most basic function: messaging. I have created a single system that provides a generalized interface for all available messaging platforms. This system unites the basic functionality in a clean, modular design. It provides an easy interface for developers to write platform-agnostic code to send message sequences to users. And it makes adding new platforms to the system much easier.

Completed deliverables

1. A web backend to send SMS and Facebook Messenger stories on schedule
   • A single interface for sending text and picture messages (platform-agnostic)
   • implementations for SMS and Facebook messaging functions
   • modules for message scheduling, user preferences, and notifications
2. A domain-specific language that uses meta-programming techniques to write the StoryTime curriculum scripts.
3. The algorithm and implementation to enforce order and timeliness in SMS sequences.
Technology Stack

1. Ruby + Sinatra for the web backend
2. Twilio for sending SMS and MMS
3. Facebook Messenger platform (and associated Ruby API’s)
4. Sidekiq (a Ruby gem) for background worker processes (concurrent messaging)
5. I18n Ruby gem for translations
6. PostgreSQL database using the Ruby Sequel ORM

The DSL for StoryTime scripts

The following images are samples of message sequence scripts written in the StoryTime domain-specific language. The first is code that describes a sequence of SMS/MMS; the second describes a sequence of Facebook messages. Actions are organized into ‘sequences’, which contain instructions for which messages to send. Between SMS and Facebook Messenger scripts, they have in common the following:

1. All scripts are instantiated with the new_script method, given the name of the script (e.g. ’day4’) and the specific platform (‘sms’ or ‘fb’, for Facebook).

2. Scripts are composed of sequences, which are methods defined by the DSL module. The sequence method returns a Ruby Proc, which is similar to an anonymous function that can be executed later. They are stored within a hash map of all StoryTime sequences, indexed by the sequence name and platform.

3. Primarily, sequences run the “send” method, which takes arguments that specify the type of message to be sent, the content of that message, and the recipient thereof. The “send” method is universal across platforms; however, across platforms, there are a few distinctions in the required parameters. For example, the SMS platform requires two extra arguments that specify the sequences’ relative order. The Facebook platform takes the content argument as a Ruby hash instead of a string.

4. Sequences also run the “delay” method, which enforces timing in between messages.

This design for message sequences gives the developer an intuitive approach to writing StoryTime scripts. When each sequence may be a unique combination of images, text, and other widgets, it is very helpful to be able to script out message sequences chronologically.

It is possible to accomplish this using Ruby’s meta programming capabilities, specifically the “instance_exec” method as used on Procs. This allows for each
sequence block to be executed in the context of the calling object, thereby giving access to that object’s instance variables. For this implementation, the calling function is a “StoryTimeScript” object, which contains metadata about the script day, including language information, text, and image data for that day.

```
sequence 'firstmessage' do [recipient]
  txt = 'scripts.intro_sms__poc__[3]'
  puts "sending intro txt..."
  send recipient, txt, 'firstmessage', 'imagel', 'MSG'
  delay recipient, 'imagel', SMS_WAIT
end

sequence 'imagel' do [recipient]
  img = 'mms.stories.chomp[0]'
  puts "sending first image..."
  send recipient, img, 'imagel', 'image2', 'IMG'
end

sequence 'image2' do [recipient]
  puts "sending second image..."
  img = 'mms.stories.chomp[1]'
  send recipient, img, 'image2', 'IMG'
end
```

StoryTime scripts are composed of sequences. Each sequence contains “send” and “delay” actions, which control the content, timing, and recipients of messages. These scripts are for SMS and Facebook users, respectively.
SMS Books

Many parents do not have smartphones, but do have feature phones that can text. StoryTime accommodates these parents with SMS/MMS stories.¹

There are challenges to using SMS/MMS technology to deliver ordered sequences of text and images. This antiquated tech was not designed for this use case. To create the best reading experience for parents on SMS, text messages must arrive in a reasonable amount of time (“reasonable” means to minimize the amount of idle time between consecutive messages). Text messages must also arrive in a specific order, as narrative stories go. However, cell phone carriers promise neither the time nor the arrival order of messages (see Twilio documentation)— texts are completely independent of one another once sent to the carrier. Thus, StoryTime’s application of SMS and MMS technology pushes beyond their standard use case.

Before StoryTime, the standard way to send a sequence of text messages was to send them one at a time in a specific order and essentially pause for five to fifteen seconds before sending the next one. The wait-time is arbitrary, but if you pause for too little time, your recipient might receive his messages out-of-order; if you pause for too long, the recipient must wait full minutes to receive longer text sequences. There is no gold standard wait-time because timing is inconsistent for both the carrier and the user’s handset phone.

The system is a smarter way to time message sending. Basically, it orders message sequences into a queue. The up-next sequence in the queue is only called once it receives a “success” callback from the previous message. If no callback is received, the system waits for an allotted period of time (around 45 seconds). If the callback is still not received (signifying that the recipient has still not received the last message), then the previous message is re-sent.

The implementation of this system relies on:

1. Twilio API’s SMS callback feature,
2. A custom-built API to receive the Twilio callback requests,
3. Sidekiq workers to execute text messaging jobs as asynchronous background processes, and
4. An asynchronous timer.

¹ SMS sends texts; MMS sends pictures.
Sequence order is built into the SMS/MMS sequences themselves. Each sequence contains two parameters—“current_sequence” and “next_sequence”—which define the names of both the current sequence and the next sequence in the queue. This way, once the “success” callback is received, the system can call the next sequence simply by referencing it with the “next_sequence” parameter.

The “success” callback is a feature of Twilio, and is mostly used for analytics and error handling for mass text messaging (like in political campaigns). Once the cell phone carrier is notified that the message was delivered to the recipient’s handset, it will issue this callback in the form of an HTTP request to an HTTP endpoint that I specify. That endpoint was the custom-built API that I created to parse the callback response and dispatch text messaging worker processes.

The callback method significantly reduced the time to receive a sequence of text messages by a factor of 4. The reduction varied between handsets and carriers, but always performed significantly better than the standard method of sending long sequences of text and picture messages.

**Language**

Instead of sending text messages directly in sequences, text codes are sent that translate into the user’s preferred language (e.g. ‘scripts.intro_sms.__poc__.\[3\]’). The i18n translation Ruby gem was used to accomplish this. The actual text of each language is stored in .yml key-value files. The keys are referenced from the sequences.

**Scheduling, Reminders**

Stories are typically sent at around 7pm in the time zone of the user (defaulting to EST). This is accomplished through a clock on the web server that runs continuously in the background. At periodic intervals, the clock checks to see if any users in the database are due to receive their daily story. If a user’s send-time is within 5 minutes of the current time, then the clock process issues a Sidekiq worker process that executes the story sequence.

With the Facebook Messenger platform, developers can know which users have read their messages through “read-receipts” and other interactions. StoryTime sends “story buttons” for users to press. Once the user presses the button, they will be sent the full sequence of story messages. The system records in the database when users were last sent a story button, along with when they last pressed it. If the user does not
press the story button within 3 days, a “reading reminder” is sent out to the user at their usual story time.

**Code**

See the code in my GitHub repository here:

https://github.com/jdmcpeek/storytime

**Demo**

Visit

https://www.evernote.com/l/AuYjRdDE4OtAoqQhV3SdDTWWhcX3D0SQL0s

**Future Work**

StoryTime has accrued over 250 active parent users so far.

I would like to procure more content and more languages for non-English-speaking parents, and share StoryTime with more teachers so that more parents can have easy access to books.

**References and Resources**

Technology used:

- Ruby gems
  1. Sinatra: https://rubygems.org/gems/sinatra/versions/1.4.7
  2. Sidekiq: https://rubygems.org/gems/sidekiq
  3. Sequel: https://rubygems.org/gems/sequel
  4. i18n: https://rubygems.org/gems/i18n
  5. Twilio-Ruby: https://rubygems.org/gems/twilio-ruby
  6. Facebook-Messenger: https://rubygems.org/gems/facebook-messenger/versions/0.1.0
- PostgreSQL: https://www.postgresql.org/
• Facebook Messenger platform: https://developers.facebook.com/docs/messenger-platform

• Twilio REST API: https://www.twilio.com/docs/api/rest