Pinch Pad:
Design and Development of an iPad Application for Sharing Sketches

Ryan Laughlin
April 2012

Advised by Michael Fischer
Submitted to Yale University
Computer Science Senior Project
**Abstract**

Pinch Pad is an application that allows users to make sketches on the Apple iPad and to share them on the web – like Twitter, but with drawings instead of words. Although there are many easy ways to share text and photographs via the internet, there are few ways to send something drawn by hand, unless you own a fax machine or a scanner. Pinch Pad allows users to keep a journal of drawings, drawn directly on the iPad and then posted instantly to the web. The public gallery of sketches from all users is displayed in reverse chronological order as a live feed, and can be filtered by various criteria: creator, date, time of day, and location. Visitors can even subscribe to an RSS feed for a given filter, and receive updates as new sketches are posted.

Pinch Pad was written in Objective-C (for the native iPad application) and in Ruby on Rails (for the web server), and uses jQuery (a Javascript library) for some minor visual effects in the web browser. It authenticates users based on a unique device ID associated with each iPad, derived from the MAC address. User accounts and sketch metadata are stored in a PostgreSQL database, while actual image files are uploaded to Amazon S3 cloud storage.

As of April 2012, the Pinch Pad server is live at [http://pinchpad.com](http://pinchpad.com), and includes over 500 drawings made via the iPad app. (As an initial test run for the software, I kept an hourly sketch journal for the entire month of March 2012.)
Background

When thinking about ideas for my senior project, I wanted to do some kind of development work that I had never done before – perhaps write code in a new language, or use a development pattern that I had not encountered previously. And of course, I wanted the project to be something that I would find interesting and engaging.

Some of my favorite projects in the past had been general-purpose, consumer-oriented applications: that is, applications that any average person could pick up, play with, and enjoy. A few years ago, for instance, I wrote a browser-based music synthesizer and sequencer, which let visitors click on a grid of squares to create basic melodies and loops. The underlying code was complex, but the presentation was very simple and fun for just about anybody.

I was also interested in trying my hand at iOS development and Objective-C. The iPhone and iPad have both been tremendously successful products, and together they provide an opportunity to create a whole new class of applications, driven by touch rather than by a keyboard and mouse. These “post-PC devices” come with you everywhere, are always connected to the internet, and include new capabilities like geolocation that are unavailable in traditional programming environments.

Thus I decided that I wanted to try writing a consumer-oriented iOS application for my senior project. After some more brainstorming, I decided to develop an app that allowed a user to draw sketches and post them to the internet in real-time – sort of like Twitter, but for artists. It would be a good opportunity not only to learn more about iOS and Objective-C, but also to understand the design of a client-server program and the difficulties involved in such an application.

Pinch Pad

Overview

Pinch Pad is an application that allows users to draw simple sketches and share them publicly in real-time. It consists of a client app for the iPad, where the sketches are actually drawn, and a web server, where the sketches are saved and publicly displayed.
Sharing digital sketches is an interesting problem to tackle because at present, there is no easy way to do so. There are a few existing solutions for sending hand-drawn materials over the air:

- Fax machines have been around for decades, but are not widely available to consumers. Also, though some newer fax machines can turn faxes into digital documents, most still are still analog-only and print copies on paper.
- Scanners create digital copies of physical materials, but they are not widely available to consumers and add unnecessary steps to the sketch-sharing process (i.e. draw a picture, scan it in, save as an image file, attach to an email, etc.)
- Desktop drawing tablets, of the sort manufactured by Wacom and other companies, produce native digital drawings, but are expensive and even less available to consumers. Also, most cheap drawing tablets do not include a screen on the tablet itself, meaning that the user is required to look up at the monitor while manipulating a digital pen. This interaction is difficult and takes practice to get used to.

In short, there is no good method at present for a person who wants to quickly send a sketch to a friend or colleague.

Use cases

This is an important problem because there are many scenarios in which handwritten instructions are much clearer than text. Consider how useful a blackboard is when diagramming a complex idea, or how a scratch piece of paper can help you to find the answers on a test. Handwriting allows us to think organically, jumping from one idea to another – associating thoughts not in a strict linear fashion, as word processing forces us to do, but in a dynamic way that better reflects our actual thinking. Handwriting and sketching are crucial in a variety of fields, including the following examples:

- Artists often make use of sketches when preparing a work, whether it be a painting, a sculpture, or a piece of architecture. The ability to share sketches instantly would enable artists to more efficiently collaborate on ideas and to easily share their work with a larger audience.
- Project teams within a company often have meetings that involve lengthy whiteboard sessions for planning and strategic thinking. Better software to share sketches would enable employees working remotely to better follow the discussion and make their own valuable contributions, and cut down on problems of miscommunication and inefficiency.
• My personal interest in this project stems from a pet project of mine that I have worked on for many years. For one month a year, I document my life in a series of hourly sketches, each depicting some event that has happened during that hour. When I was in high school, I could share these “hourly comics” with my family at the end of the day, but after I left for college, it became much more difficult to share my project with them. This year, I did hourly comics during the month of March, and Pinch Pad enabled me to share them in real-time with anyone in the world, by posting live to the internet.

There is clearly some demand for this kind of software, but until recently there was no good piece of hardware for recording precise touch-based information on a screen large enough for sketching. The introduction of the iPad in 2010 changed this, by providing a large, high-fidelity touch screen that is portable, internet-ready, and affordable to consumers. The iPad presents a great opportunity to create a new piece of software for sharing sketches – an application that doesn’t make nearly as much sense on a PC.

Design

Conceptual model

The underlying data structures for Pinch Pad are mostly straightforward. There are two application-specific models: Devices and Sketches.

Devices consist of a unique device ID (UDID) and username, and are used to distinguish one user from another. Rather than asking the user to make up a password when first launching the app, Pinch Pad generates a unique ID for every iPad and sends this parameter along with all requests to the server. The UDID is used for simple authentication, allowing users to create and delete their own sketches. This implicit login system has a few significant issues – when a user replaces a device, for instance, he or she cannot access Pinch Pad with the same credentials – but I chose it in order to experiment with the user experience, as an alternative to traditional username/password login systems. I’ve built other web-based login systems before, and wanted to try something new for this project.

The Sketch model includes metadata for each image file, which allows the full database of images to be filtered by user, date, time of day, and location (latitude and longitude). I decided early on that my sketches would be stored as static image files, rather than as

---

1 This idea was pioneered by John Campbell; see [http://hourlycomic.com](http://hourlycomic.com) for more information.
vector data, and that the drawing portion of my iOS app would only support simple black-and-white images, not intricate many-colored paintings. These choices were made in order to keep the project within a reasonable scope. Given more time, it would be an excellent addition if Pinch Pad supported vector graphics and more complex images. This would allow Pinch Pad to save sketches with infinite fidelity, so that drawings could be reproduced at different sizes and on different devices without loss of clarity.

**iPad app**

The iPad app consists of two screens: a home screen, which lists all of a user’s past pictures, and a sketching screen, where a user can create a new sketch.

The home screen displays a user’s sketches in reverse chronological order, including a link to delete each individual sketch. It also includes a “Settings” button in the upper left, where a user may set his or her username, and a “Draw” button in the upper right, which opens a modal window to create a new sketch.

The sketching screen includes a cancel button and a “Post” button, a button for undo, and a toggle between pencil mode and eraser mode. Dragging one’s finger across the canvas produces a solid back line or erases back to white, depending on the selected tool.

The iPad application home screen
The iPad application sketching screen

The iPad app also includes a few invisible features:

- Offline mode caches new sketches that are created while the iPad cannot connect to the internet, then automatically submits them when a connection is restored. A visual indicator is displayed on the upper-right of the home screen when there are new sketches waiting in the offline queue.
- Geolocation determines the device’s location while the user is sketching, and passes along the current latitude and longitude as sketch metadata.
- Tapping an individual sketch on the home screen opens that sketch in Safari, the iPad’s primary web browser, so that the URL can be easily shared with others.

**Web app**

The web app stores the image data permanently, and makes it publicly available for browsing online. It consists of four main pages:

- The home page shows a listing of all sketches posted to Pinch Pad, in reverse chronological order. It is a live stream of activity on the app.
The web app home page

- Clicking on any name, date, time, or location filters the database of sketches by that criterion, and displays a list of all sketches in that category.
- Clicking on an individual sketch brings the user to a permanent URL for that specific image, which may be shared. It also provides the user with links to share the sketch on several popular social media sites (Facebook, Twitter, Tumblr, and StumbleUpon).
- Clicking the “Subscribe” link at the top of the home page brings up an RSS feed of all sketches added to the system, so that visitors may subscribe to new activity. Clicking the “Subscribe” link at the top of any filter page does the same thing, while also filtering on the given criterion.

Sketches cannot be created directly from the web, but the web site provides a friendly interface for viewing an individual’s journal of sketches or for exploring the body of content contributed by all users (e.g. “What are people drawing around 10am each day?”, “What do people draw near Cross Campus?”, etc.)
Development

Technical setup

Setting up an iOS development environment was straightforward: I purchased a license to the iOS Developer Program for $99, then installed XCode and the iOS 5 SDK. The Apple developer site includes instructions to generate a provisioning profile, which allows you to deploy development applications to an actual device.

My web server setup was slightly more involved. I wrote all the server code in Ruby on Rails (which I had previously set up a local development environment for) and hosted it on Heroku, a web host that provides free Ruby on Rails hosting for apps with low traffic. Image files were uploaded from the server to Amazon S3, a cloud storage service that is free below certain usage limits. Image metadata and device info were stored in a shared PostgreSQL database provided by Heroku.

Web app

As previously mentioned, all sketch data was permanently stored on the web server. This required the creation of two database tables, Devices and Sketches. I wrote models in Ruby on Rails to wrap all interactions with the database and to perform various validations, such as ensuring that all Sketches included image files and that duplicate Sketches were rejected by the system.

Like most Ruby on Rails apps, the web app follows the Model-View-Controller (MVC) pattern. When a URL is accessed, the app performs a lookup and calls the corresponding Controller; that controller fetches Models and passes them to the View, which renders the appropriate HTML, CSS, and Javascript.

Third-party Ruby gems (i.e. plugins) were used to assist with certain data-processing tasks. Image files were uploaded to Amazon S3 via Fog and CarrierWave, while latitude-longitude pairs were translated to street addresses with Geocoder.

A few specific notes:

- The security for UDID-based authentication is minimal – in essence, it is just “security through obscurity”. Given a device’s UDID, an attacker could compromise a user’s account and add or delete pictures at will. A more rigorous authentication system of this type might use the UDID as a private key and pass
the server a public key, which would allow for proper authentication and encryption.

- Filtering is done by specifying many different routes for the same “index” action. In other words, different URLs call the same controller function, but with different parameters, and small adjustments to the controller and view are made accordingly, thus allowing visitors to view organized subsets of the entire sketch archive.
- RSS feeds are generated in the same manner as HTML pages: I wrote an RSS template view that is populated with data from the Sketches table and returned to the visitor.
- Deleted Sketches are not actually deleted; they are merely updated with a “deleted_at” timestamp and excluded from future queries. This is not the Ruby on Rails default, but my own custom code; I added it so that I would not accidentally delete any of my own hourly comics while testing and debugging the code.

iPad app

Learning to code in Objective-C was occasionally frustrating – particularly when I would forget to include “@” in front of a string literal – but as I gained experience I found it very enjoyable to work with. Unsurprisingly, it shares a lot with C and with C++ (it derives directly from C), so I was able to read code examples right away and very quickly start writing code of my own. XCode also generates a rather thorough application template for you when you start a project, so from the very beginning I had a stable base to build upon.

The iOS SDK makes heavy use of delegation and subscription to notifications in order to pass data and events among controllers, views, and models. In iOS 5, it also combines this with a visual process called “storyboarding,” where you can visually arrange user interface elements, then attach them to variables or draw relationships between them (delegations, transitions, etc.) The system is a bit complex, but once I understood the general flow of how to transition from one view controller to another, the storyboard helped me organize my app in a very logical manner.

Here are some of the particular issues I tackled:

- To display the listing of sketches on the home screen, I use a simple UIWebView: essentially a browser window showing a specific page on the website. I added custom handlers to the UIWebView so that some links open within the app (e.g. links to switch between multiple pages of sketches), but others open in the iPad’s native browser (e.g. a link to view all sketches made on a specific date).
• Network status is monitored via a third-party library, Reachability, which starts a separate thread that sends notifications whenever the network status changes. I subscribed to these events so that the app tries to submit the cached offline sketch queue as soon as internet access is restored.

• The iPad’s unique device ID (UDID) is generated by a third-party library, using a combination of the iPad’s MAC address, the application name, and an MD5 hash. See the NSString-MD5Addition and UIDevice-IdentifierAddition classes.

• In order to actually sketch, I tracked continuous touches on the screen and drew lines between subsequent points with Quartz, iOS’s standard 2D drawing library. The performance is acceptable, but it sometimes results in jagged edges, especially when moving rapidly and jerkily (e.g. when quickly writing text). A more efficient implementation would use raw OpenGL calls, but I considered that outside the scope of the project.

  o My sketching code currently produces raster images, primarily because raster images are better supported by the iOS SDK’s built-in image export functions. However, it would be relatively straightforward to add vector graphics support in a future Pinch Pad update. All touch data and line segments are passed as (x,y) points, which could saved in a traditional vector file format like SVG or in a custom file format for Pinch Pad.

• This version of the app supports infinite undo while sketching. Whenever the user draws a line, the sketch’s previous state is pushed onto a stack of undo points; when the user hits undo, the latest undo point is popped off of the stack and painted back on to the canvas. Unfortunately, storing all those image snapshots is rather memory-intensive. A more efficient implementation of undo might store the touch data locally as a series of points, then use that information to reconstruct the sketch at any given point during drawing. (It would also be nice to add “redo” support, so that a user doesn’t accidentally delete a desired stroke.)

• The iOS SDK includes a native library, CoreLocation, for determining the iPad’s location. When the user starts drawing a sketch, the app initializes CoreLocation on a separate thread and subscribes to its events; each event returns the latest, most accurate reading from the GPS or Wi-Fi triangulation. That latitude and longitude data is sent to the server, which then translates it into a street address.

• Sketches made offline are stored using a native iOS library, CoreData, which encodes simple variables and objects in a local SQLite3 database. The Pinch Pad iPad uses a Sketches table to store image data and metadata for later submission.

• Settings are implemented in a simple popover on the home screen; tapping “Save” sends a POST request to the server to update the device’s associated name, then refreshes the list of sketches to reflect the updated name.

• A number of other hooks and callbacks were added to handle additional events. For instance, when the application is suspended and then resumed, it submits any
cached offline sketches, then refreshes the home screen to reflect any changes made via the web site since last launch.

Results

The final app is documented in a few screenshots above; as of April 2012, you can also see the website for yourself at http://pinchpad.com. The app is fully functional and works on any iPad running iOS 5 or later. In fact, I used it myself for the entire month of March, drawing a comic for every hour that I was awake (over 500 comics in total).

Sample sketches created with Pinch Pad
Future areas for improvement

There are a number of features that didn’t make this version of Pinch Pad, primarily because I wanted to keep the scope of the project manageable. Some obvious improvements include:

- A better sketching component for the iPad app, including different colors and pen weights, better undo/redo functionality, and vector image support.
- More ways to share sketches straight from the iPad app, such as sending a sketch via email or syndicating a user’s entire feed to Twitter or Tumblr.
- More collaboration features: allowing visitors to add comments and to expand upon existing sketches. This would be extremely useful for teams that work together remotely, who need a way to pass ideas back and forth in a dynamic way.
  - To take this to its logical extreme, the app could even support real-time collaboration, so that multiple users could sketch on the same canvas at the same time.
- Improved offline support, so that users could view their history of sketches and new pending sketches without an active internet connection.
Lessons learned

Besides general experience with Objective-C and the iOS SDK, this project strengthened my knowledge of a number of things:

- Top-to-bottom app design. Pinch Pad is probably the largest non-website coding project that I have worked on. It was tempting to dive right into the Objective-C code, but I made sure to clearly lay out the different parts of the project first, so that I could figure out how I wanted them all to interact. Planning the overall structure at the beginning helped me to get a handle on the scope of the project, to build each piece in an efficient and modular fashion, and to make sure that I was on track to finish.

- Client/server app design. Although Pinch Pad doesn’t tackle some of the more intricate problems of “cloud” apps, like multi-client syncing, it did give me a feel for the basics: where to store data, when to fetch it, and when to post it.

- The delegation pattern. iOS relies on this pattern heavily, particularly to communicate between view controllers. I had very little previous experience with this pattern, so it took me a few weeks to really wrap my head around it and start using it in an effective manner. It provides an easy way for classes to hand off tasks to each other without needing to know specific implementation details.

- The publish/subscribe pattern. iOS uses this pattern for multithreading and for communication with the OS (when resuming from suspended state, for instance), neither of which I have much experience with. Asynchronous communication between threads is powerful, but can be conceptually complicated and difficult to follow. Interestingly, I realized about halfway through the project that the strengths and issues with publish/subscribe are very similar to those with client/server app design – in many important ways, they are the same problem.

- General approaches to sketching apps/2D drawing. I had never worked on graphical editing software before, so during the development of Pinch Pad I experimented with various things: offscreen image buffers, vector graphics, etc. The final sketching component in Pinch Pad is rather basic, but it was a great introduction for the next time I work on a very graphical app.

Conclusion

Working on Pinch Pad was a great introduction to Objective-C and the iOS SDK, programming patterns like delegation and publish/subscribe, and graphics programming in general. It strengthened my basic knowledge of Object-Oriented Programming and gave me experience with designing and developing a simple app from start to finish. I
wish I’d had a bit more time to polish a few rough edges, but I expect that I will continue maintaining and updating Pinch Pad in the future, so that I can do hourly comics in the same manner in future years.
Bibliography

Resources used

• Tutorials and references:
  o Apple’s iOS Developer Library (http://developer.apple.com/library/ios/navigation/), particularly the Getting Started tutorials, the Quartz 2D Programming Guide for approaches to sketching, and the Introduction to Core Data for building the offline portion of the app
  o The iOS SDK documentation included with XCode
  o RailsGuides (http://guides.rubyonrails.org/index.html)
  o Github documentation for Ruby on Rails gems:
    ▪ Carrierwave (https://github.com/jnicklas/carrierwave)
    ▪ Geocoder (https://github.com/alexreisner/geocoder)
  o Github documentation for third-party iOS libraries:
    ▪ Reachability (https://github.com/tonymillion/Reachability)
    ▪ UIDevice-with-UniqueIdentifier-for-iOS-5 (https://github.com/gekitz/UIDevice-with-UniqueIdentifier-for-iOS-5)
  o Stack Overflow, for miscellaneous minor questions on Ruby on Rails and Objective-C (http://stackoverflow.com)
  o Web template based off of the HTML5 Boilerplate (http://html5boilerplate.com/)

• Development environment:
  o XCode 4.3.1 (https://developer.apple.com/xcode/)
  o Sublime Text 2 (http://www.sublimetext.com/2)
  o The OS X console

• Web hosting:
  o Heroku (http://heroku.com) for free Ruby on Rails hosting
  o Amazon S3 (http://aws.amazon.com) for free image storage

Related works

• Other iOS sketching apps:
  o Penultimate (http://itunes.apple.com/us/app/penultimate/id354098826)

• Twitter (http://twitter.com)
• Hourly Comics by John Campbell (http://hourlycomic.com)