# Client-Facing Pages

index.html

* JS: journal.js, timestamps.js, journalsCombo.js
* CSS: screen.css
* This is the landing page for selecting and reading a journal. The main page application.
* It is the frame and controls around the user’s journal content.

journalfiles/\*.html

* Each HTML file in the journal files folder is a body for one journal entry written by the user.
* This content is included by index.html and placetimestamps.html.
* The story is organized into <span class="timestamp" data-time="1258813400"> blocks, which indicate which paragraphs correspond to a particular time.

placetimestamps.html

* JS: timestamps.js, placetimestamps.js, journalsCombo.js
* CSS: “screen two columns 3-16-10.css”
* This is the tool for A) creating a new journal entry which will be put in journalfiles/ and then for B) marking up that entry with image timestamps. The tool can also be used C) to add/remove timestamps to a file already existing in journalfiles/.
* It writes new HTML files to journalfiles/.
* It reads existing HTML files from journalfiles/, updates their HTML and overwrites them.

importAlbums.html

* JS: timestamps.js, importAlbums.js
* CSS: screen.css
* This utility will find all public albums for a particular Google user. You can then select one of these public albums and import the photostream. The imported (and simplified) stream is cached in “picasaxml/\*.xml” files on the server, so that when a query for a range of photos is made the lookup of the photos will be fast (faster than contacting Google that is).
  + Note: it would be better to use a true database rather than create these xml files.

# Server-side PHP

timestampUtils.php

* Utilities for:
  + formatting a timestamp
  + rounding a timestamp to the nearest GMT day

domTreeUtils.php

* Utility functions for:
  + deleting nodes from DOM trees
  + getting a timestamp out of a Google photo feed entry

getPhotosFromCache.php

* Query (#1) takes arguments ‘lowerTimeBound’, ‘upperTimeBound’, and optional ‘keyword’. It finds all photos in the time range by looking in the cached XML files in the picasaxml/ directory. It returns the photos as formatted HTML which is ready for presentation on the page.

makeGPhotoCache.php

* This will load a Google photos feed off of a URL, then parse that feed and store a cached version of that feed as an .xml file in picasaxml/.
* Query (#1) takes arguments ‘picasaUserFeed’, ‘lowerTimeBound’, ‘upperTimeBound’. It will load the Google photos feed off of the URL provided as ‘picasaUserFeed’, filter the feed by the two time bounds, and produce caches (as .xml files in the directory picasaxml/) of those photo-feed items which satisfy the query.
  + Each cached file in the picasaxml/ directory represents one day’s worth of photos. The xml file is named with the UNIX GMT timestamp for that day.
* Query (#2) takes arguments ‘getAlbumStartAndEnd’, ‘picasaUserFeed’ , ‘albumId’. It will load the Google photo feed for a particular user and album, then inspect the photos in the feed to determine the newest and oldest timestamps in the feed.
  + Knowing the range of dates for an album lets us make more targeted queries to (#1) above. For example, we can make one (#1) query for each day in the time span of the album. We do this to reduce the likelihood that our query will timeout on the server because it takes too long to process all of the photos.

saveJournal.php

* Query (#1) takes arguments ‘journalEntryTitle, ‘journalEntryDate’, and ‘journalEntryContent’. It writes the provided text-only journal entry into a HTML file on the server. (If the HTML file already exists, it is overwritten.) This is how a journal entry is save/stored on the server.
  + Additional formatting will be done server-side to the entry content. For example the entry’s Title and Date are added as <h2> and <h3> entries before the content.
* Query(#2) takes arguments ‘journalName’ and ‘applyTimestamps’. It takes a journal entry which has already been marked up with <span data-time=””> tags and writes it to the server.