
CouchApp Documentation

Release 1.0.1

Various CouchApp Contributors

Apr 15, 2017

Contents

| | | |
|----------|--|-----------|
| 1 | Overview | 3 |
| 1.1 | Introduction | 3 |
| 2 | Getting Started | 11 |
| 2.1 | Installation | 11 |
| 2.2 | Getting Started | 13 |
| 3 | User Guide | 17 |
| 3.1 | CouchApp Command Line Tool | 17 |
| 3.2 | JavaScript Application Programming | 26 |
| 3.3 | The Garden | 26 |
| 3.4 | CouchApps and DesktopCouch | 27 |
| 3.5 | List of CouchApps | 27 |
| 4 | Design | 37 |
| 4.1 | The CouchApp Filesystem Mapping | 37 |
| 5 | Contributing to CouchApp Ecosystem | 41 |
| 5.1 | Contributing | 41 |
| 5.2 | Roadmap | 42 |
| 6 | Other Resources | 43 |

Note: This documentation is a work in progress. Contribution welcomed.

CHAPTER 1

Overview

We will introduce the main concepts of CouchApp here.

Introduction

CouchApps are JavaScript and HTML5 applications served directly from CouchDB. If you can fit your application into those constraints, then you get CouchDB's scalability and flexibility **for free** (and deploying your app is as simple as replicating it to the production server).

There are also tools for deploying browser-based apps to JSON web servers and [PouchDB](#) is the future of browser based sync apps. It can sync with the same sync protocols but uses the built-in storage of HTML5.

What is CouchApp?

Note: This article came from a blog post by @jchris. Some contents are out-of-date. We need your contribution!

The Basics

A CouchApp is just a JavaScript and HTML5 app that can be served *directly* to the browser from CouchDB, without any other software in the stack. There are many benefits (and some constraints) to doing it this way. The first section of this article will address these tradeoffs.

In the bad old days (2008 and earlier), if you wanted to write a dynamic database-backed web application, you had to have an architecture that looked like a layer cake:

| |
|---|
| Browser (UI and links between pages) |
| ----- HTTP ----- |
| Application server (business logic, templates, etc) |

```
----- custom binary -----  
Persistence: MySQL, PostgreSQL, Oracle
```

In fact, the bad old days are still with us, as most applications still rely on fragile custom code, running in an application server like Ruby on Rails, Python's Django, or some kinda Java thing. The pain-points of the 3 tier architecture are well known: application developers must understand the concept of shared-nothing state, or else clients can see inconsistent results if they are load balanced across a cluster of app servers. The application server is usually a memory hog. And at the end of the day, when you've finally gotten the app layer to horizontal scalability, it turns out that your database-tier has fatal scalability flaws...

CouchDB is an HTTP server, capable of serving HTML directly to the browser. It is also a database designed from the ground up for horizontal scalability. Did I say silver bullet? ;) (Of course it is not a silver bullet – if you can't fit your app into CouchDB's constraints, you'll still have scaling issues.) If you can build your app *with the grain* of CouchDB's APIs, then you can piggyback on all the work [other](#) people have done to scale.

The fact is, 2-layer applications are simpler:

```
Browser (UI and links between pages)  
----- HTTP -----  
CouchDB (persistence, business logic, and templating)
```

Because CouchDB is a web server, you can serve applications directly the browser without any middle tier. When I'm feeling punchy, I like to call the traditional application server stack “extra code to make CouchDB uglier and slower.”

Aside from simplicity and the scalability that comes with it, there is another major benefit to creating a 100% pure CouchApp: [Replication](#). When your app is hosted by just a CouchDB, that means it can be run from *any* CouchDB, with no need to set up complex server-side dependencies. When your app can run on *any* CouchDB, you are free to take advantage of CouchDB's killer feature: replicating the app and the data anywhere on the network.

Have you ever been frustrated by a slow website? Filling out forms and waiting even a few seconds for the response can be infuriating. Many users will hit the submit button over and over again, compounding whatever performance issues that are effecting them, while introducing data integrity issues as well. Google, Facebook, and other large competitive web properies know that [perceived latency drives user engagement like nothing else, and they invest huge sums to make their sites seem faster](#).

Your site can be faster than theirs, if you serve it from localhost. CouchDB makes this possible. Here are [installers for OSX, Windows, and Linux](#) and you can install [CouchDB on Android here](#).

The take-home message from this section is: CouchDB can scale. If your app is served by raw CouchDB, it can scale just the same. Also, there's no server faster than the server running on your local device. And fast is what matters for users.

In the next section we'll see what it takes to get your app to be served directly from CouchDB, and what you can (and can't) do.

CouchDB's built-in programming model

The CouchDB API is full featured and applicable to a lot of use cases. We can't possible go in-depth here. Instead we'll focus only on the broad outline, and on what is useful and necessary for CouchApps. If you want to learn more, check out [the CouchDB wiki](#) or the [free CouchDB book](#).

The first thing to understand about CouchDB is that the entire API is HTTP. Data is stored and retrieved using the protocol your browser is good at. Even [the CouchDB test suite](#) is written in JavaScript and executed from the browser. **It's all just HTTP.**

HTML Attachments

A common question I get from people starting to write Ajax apps using CouchDB, is “when I try to query the CouchDB with jQuery, it doesn’t work.” Usually it turns out that they have an `index.html` file on their filesystem, which is attempting to do an Ajax call against the CouchDB server. After I explain to them [the same origin security policy](#), they start to understand this this means CouchDB needs to serve their HTML (rather than loading it in the browser direct from the filesystem).

CouchDB documents may have [binary attachments](#). The easiest way to add an attachment to a document is via Futon. We’ll do that later in this blog post.

So, the simplest possible CouchApp is just an HTML file, served directly from CouchDB, that uses Ajax to load and save data from the CouchDB.

Map Reduce queries

What sets CouchDB apart from a simple key value store like Memcached or [Amazon S3](#), is that you can query it by building indexes across the stored objects. You do this by writing JavaScript functions that are passed each of your documents, and can pick from them a set of keys under which you’d like to locate them.

So for a blog post, you might pick out all the tags, and make keys like `[tag, doc.created_at]`. Once you have a view like that, you can easily get a view of all your blog posts with a given tag, in chronological order, no less. By adding the reduce operator `_count` you can also see how many blog posts are tagged `foo` or whatever.

I’m not gonna try to teach you all about views here. Try the [CouchDB book’s guide to views](#), the [wiki](#) and this [chapter on advanced views](#).

Server Side Validations

The second thing people usually ask when they start to grok the CouchApp model, is “How do I keep people from destroying all my data? How do I ensure they only do what they are allowed to do?” The answer to that is [validation functions](#). In a nutshell, each time someone saves or updates a CouchDB document, it is passed to your validation function, which has the option to throw an error. It can either throw `{"forbidden" : "no matter what"}` or `{"unauthorized" : "maybe if you login as someone else"}` where, of course, you are free to craft your own messages. If the function doesn’t have any errors, the save is allowed to proceed.

Rendering Dynamic HTML

After a new user understands validation functions, they have begun to see that perhaps CouchDB / CouchApps is a good candidate for their application. But maybe something is missing... Search engines don’t treat Ajax applications with the same respect they do static HTML applications. Also, a fair proportion of users have JavaScript disabled, or are using a screen-reader type application, which may not understand Ajax.

These are all great reasons your application should ship the basic content of a page as real-deal HTML. Luckily, CouchDB has an answer to that as well.

[Show functions](#) allow you to transform a document GET from JSON into the format of your choice. On [this wiki application](#) the main wiki content is rendered as server-side HTML, using a show function. You can also use a show function to provide an XML, CSV, or even PNG version of your original document. Some folks also use it to filter security-sensitive fields from a JSON document, so that only public data is available to end- users.

[List functions](#) are the analog of show functions, but for view results. A view result is just a long list of JSON rows. A list function transforms those rows to other formats. Here is [the JSON view of recent posts on my blog](#), and here is the [HTML page that results from running that same view through a list function](#).

We added these capabilities to CouchDB because we knew that without the ability to serve plain-old HTML, we wouldn't be completely RESTful.

Rounding out this group, is the ability to accept plain HTML form POSTs. (And other arbitrary input). For that, CouchDB uses update functions, which can take arbitrary input and turn it in to JSON for saving to the database.

Pretty URLs

“All well and good”, you may say, “but I can't really suggest to my clients that their website should have URLs like:

```
http://jchrisa.net/drl/_design/sofa/_list/index/recent-posts?descending=true&limit=5
```

I used to respond with skepticism to such claims, *like a total moaf*. But I've mended my ways, and seen the light. It also didn't hurt that [Benoit](#) committed an [awesome rewriter](#) to CouchDB, so we can provide nice pretty URLs like `/posts/recent` instead of the above mess.

Realtime Updates

Lastly, something folks don't usually ask for, but which is insanely useful: [realtime notification about changes to the database](#). Essentially, CouchDB keeps a record of the order in which operations were applied to a given database. This way, you can always ask it “what's happened since the last time I asked?”

CouchDB implements this with the `_changes` feed, a JSON HTTP response, which sends a single line, whenever something happens to the database. Since CouchDB is implemented in Erlang, it is not expensive for it to hold open tens of thousands of concurrent connections.

The `_changes` feed can be used to power realtime updates to a browser UI. For instance, [this chat room](#) updates in realtime whenever a new message is created.

The `_changes` feed is integral to CouchDB itself (not just a bolted on feature), as it is used to power to replication itself. The replicator listens to the changes feed of the source database, and writes changes to the target database. This is what allows CouchDB to keep 2 database in sync in near realtime.

You can also use `_changes` to drive asynchronous business logic. There will be a webcast in August on this topic, as well as a blog post with more details, from Couchio's [Jason Smith](#).

Filtered replication

One last part of the programming model. You can write a JavaScript function that decides whether to include a given change in the `_changes` feed. The possibilities are endless. See [Jan's blog post on new replication features](#) for some interesting use-cases that might stimulate your imagination.

Hello World

Now that I've described the theory of CouchApps to you, let's dig into the practice. Before we get into the expert toolchain, let's see what we can do with a little bit of HTML. I'll assume you have a CouchDB running at localhost. If you don't, install one now (or signup for hosting at [Iris Couch](#) or [Cloudant](#)).

Quick, create a file called `index.html`, and put this in it:

```
<!DOCTYPE html>
<html>
  <head><title>Tiny CouchApp</title></head>
  <body>
```

```

<h1>Tiny CouchApp</h1>
<ul id="databases"></ul>
</body>
<script src="/_utils/script/jquery.js"></script>
<script src="/_utils/script/jquery.couch.js"></script>
<script>
  $.couch.allDbs({
    success : function(dbs) {
      dbs.forEach(function(db) {
        $("#databases").append('<li><a href="/_utils/database.html?'+db+'">'+db+'</
↪a></li>');
      });
    }
  });
</script>
</html>

```

Now browse to your CouchDB's Futon at http://localhost:5984/_utils and create a database called whatever. Now visit that database, and create a document. You will be creating what is known as a *Design Document*, which is a special kind of document in CouchDB that contains application code. The only thing you need to know now is to set the document id to something that begins with `_design/` and save it. Now click the button labeled *Upload Attachment* and choose the `index.html` file you just created, and upload it. Now click the link in Futon for `index.html`, and you should see a list of the databases on that CouchDB instance.

(rengel, 2012-09-05: Because of the *Same Origin* policy the `index.html` file has to be in the same directory, or a subdirectory thereof, as your whatever database.)

You gotta admit there was nothing to that.

Make it easy it with the CouchApp toolchain

Now that we've seen how you can build a basic CouchApp with the same set of tools you'd use to do plain-old HTML, CSS, and JavaScript development, let's learn how the experts (and the lazy!) do it.

Uploading each changed file to CouchDB via Futon would get tedious quick. Alternatively, you could download the entire design document as JSON, and edit that JSON in your editor... but keeping track of proper JSON escaping and formatting is a task better done by a machine.

Back in the early days of CouchDB, I solved this problem with a Ruby script that would update my map and reduce function from a folder. This way I could open the folder in TextMate, and get all the proper JavaScript syntax highlighting. To deploy the changes I'd run the Ruby script, and CouchDB would have my new Map Reduce views.

That would have been the end of the story, except that for some reason, many people had boatloads of trouble installing the Ruby script. I may have been suffering from a bit of "grass is always greener," because my reaction was to port the Ruby stuff to Python (with a little help from my friends), which I thought would have a cleaner install story. (It almost does!)

Since then the *Python CouchApp script* has grown in capability. It boasts the ability to *push edits in real time*, import vendor modules, and more. Benoit Chesneau keeps it up to date pretty aggressively, it just got some *GeoCouch features today*.

So let's use it!

Installing couchapp.py

There is a lot of documentation already out there about how to install the CouchApp toolchain. I'll just link to it. The basic installation instructions are [in the README](#) and in [the CouchDB Book](#)

Here are some hints about [installing on Windows](#).

Once you have CouchApp installed, the basic usage is simple. From within your application directory, issue the following command.

```
couchapp push . http://myname:mypass@localhost:5984/mydb
```

Replace `myname` and `mypass` with those you set up on your CouchDB using Futon. If you didn't setup an admin password on Futon, you should do that – until you do, your CouchDB can be administered by anyone. Also, if you are running a CouchDB in the cloud, you'll need to replace `localhost:5984` with something like `mycouch.couchone.com`. Also, of course, `mydb` should be changed to the name of the database you want your program to live in.

All this is covered in great detail in the CouchApp README and the book, as linked above.

The Standard Library

We've made it nearly to the end of this post. The last thing to cover are the various JavaScript libraries for making CouchApps. I won't try to document them, just name them, and say a little about their purpose.

I have a mental plan to clean up and consolidate some of these libraries, so they are more modular. This should make it so that CouchApp code loads faster, among other things.

The jQuery CouchDB Client API

We already used [jquery.couch.js](#) in the Tiny CouchApp example HTML above. This is the basic CouchDB library for jQuery. It handles things like `saveDoc` and `openDoc`, view queries, replication requests, etc. Essentially it wraps the CouchDB API in Ajaxy goodness. This library ships as part of CouchDB, as it is used by Futon.

The CouchApp Code Loader

The CouchApp toolchain ships with [jquery.couch.app.js](#), which is tasked with one job – loading your application code into the page. This CouchApp jQuery plugin loads your design document (the JSON saved as a result of a `couchapp push` command), so that the browser has access to your view definitions, show and list functions, etc. It is invoked like so:

```
$.couch.app(function(app){
  // app.db is your jquery.couch.js object
  // app.require("lib/foo") gives you access to libraries
});
```

Essentially, all this function does, is inspect the page you are on, determine how to load the design document, load it, and gives you an object that references it and allows you to require libraries from it. (There is some legacy featuritis in there, but I'm working to remove that.)

Examples

There is a [List of CouchApps](#).

CouchApp Development Tools

To develop a CouchApp, you will need a way to get your javascript, html and other resources onto your CouchDB instance. Typically this is done with a CouchApp command line tool that maps application assets and CouchDB views, lists, shows, etc into one or more Design Documents.

- *The CouchApp Filesystem Mapping* - `couchapp.py` and `erica` (mentioned below) implement a consistent filesystem-to-design-document mapping

Note: The original CouchApp command line tools were created in 2008 / 2009 by @benoitc and @jchris. They still work, and have been feature complete for a long time. `couchapp` has been replaced and is compatible with the old `couchapp` tool.

cURL

The simplest way to develop a couchapp would be to use `curl` from the command line.

CouchApp command line tool

The CouchApp command line tool is used to generate code templates in your application and to push your changes to an instance of CouchDB, among other things. Here is how to get started with the CouchApp command line tool:

- Installing couchapp
- Couchapp configuration
- The couchapp command line tool
- Extending the couchapp command line tool
- Using couchapp with multiple design documents

Note: There can be confusion with the term *CouchApp* because it can refer to this tool, named *CouchApp*, or a general application served from CouchDB. This is probably due to the fact that the CouchApp command line tool, as known as `couchapp.py`, was the first full way of developing a CouchApp.

node.couchapp.js

- <http://japhr.blogspot.com/2010/04/quick-intro-to-nodecouchappjs.html>

This is an alternative tooling to the Python couchapp utility that is instead written in Node.js. It uses a much simpler folder structure than it's Python counterpart and is a generally more minimalist/simplified way of writing couchapps. Note that you cannot use Python couchapp to push couchapps written using `node.couchapp.js` into CouchDB and vice versa.

erica

`erica` is an Erlang-based command line tool that is compatible with the Python and Node.js CouchApp tools.

Kanso

A comprehensive, framework-agnostic build tool for CouchApps.

The [Kanso](#) command line tool can build projects designed for node.couchapp.js, or even the Python couchapp tool, while providing many other options for building your app. These build steps and other code can be shared using the online [package repository](#). Compiling coffee-script, `.less` CSS templates etc. is as easy as including the relevant package.

NPM for CouchApps

[Kanso](#) also lets you merge design docs together, which allows reusable components built with any of the available couchapp tools. The [Kanso](#) tool can help you manage dependencies and share code between projects, as well as providing a library of JavaScript modules for use with CouchDB.

soca

[soca](#) is a command line tool written in ruby for building and pushing couchapps. It is similar to the canonical couchapp python tool, with a number of key differences:

- local directories do not have to map 1-1 to the design docs directory
- lifecycle management & deployment hooks for easily adding or modifying the design document with ruby tools or plugins.
- architected around using Sammy.js, instead of Evently, which is bundled with the python tool. Sammy.js is a Sinatra inspired browser-side RESTframework which is used by default.

Unlike a traditional couchapp, a [soca](#) couchapp is one way - your source directory structure is actually ‘compiled’ into the couchapp `_design` document format.

Compile time plugins:

- Compass
- CoffeeScript
- Mustache
- JavaScript bundling for CouchDB and the browser

Reupholster

[Reupholster](#) is geared for CouchApp beginners and simple CouchApps. What [Reupholster](#) does is allows you to experience writing a CouchApp as fast as possible, with very little learning curve. It just feels like you are editing a normal web project.

CHAPTER 2

Getting Started

Let's get started with the `couchapp.py` command line tools.

Installation

The newest install instructions are always in the [README](#)

In case the below is not updated, check out the [release section](#) in GitHub.

Requirements

- Python 2.x >= 2.6 (Python 3.x will be supported soon)
- the header files of the Python version that is used, which are included e.g. in the according development package `python-dev` (may have a different name depending on your system)

Installing on all UNIXs

To install `couchapp` using `easy_install` you must make sure you have a recent version of `distribute` installed:

```
$ curl -O http://python-distribute.org/distribute_setup.py
$ sudo python distribute_setup.py
$ sudo easy_install pip
```

To install or upgrade to the latest released version of `couchapp`:

```
$ sudo pip install couchapp
$ sudo pip install --upgrade couchapp
```

To install/upgrade development version:

```
$ sudo pip install git+http://github.com/couchapp/couchapp.git#egg=Couchapp
```

Installing in a sandboxed environnement

If you want to work in a sandboxed environnement which is recommended if you don't want to not *pollute* your system, you can use [virtualenv](#) :

```
$ curl -O http://python-distribute.org/distribute_setup.py
$ sudo python distribute_setup.py
$ easy_install pip
$ pip install virtualenv
```

Then to install couchapp :

```
$ pip -E couchapp_env install couchapp
```

This command create a sandboxed environment in `couchapp_env` folder. To activate and work in this environment:

```
$ cd couchapp_env && . ./bin/activate
```

Then you can work on your couchapps. I usually have a `couchapps` folder in `couchapp_env` where I put my couchapps.

Installing on Mac OS X

Warning: This section is out-of-date. We need you help for testing on newer OSX with newer `couchapp.py`

Using CouchApp Standalone executable

Download [couchapp-1.0.0-macosx.zip](#) on [Github](#) then double-click on the installer.

Using Homebrew

To install easily couchapp on Mac OS X, it may be easier to use [Homebrew](#) to install `pip`.

Once you [installed Homebrew](#), do:

```
$ brew install pip
$ env ARCHFLAGS="-arch i386 -arch x86_64" pip install couchapp
```

Installing on Ubuntu

Warning: Our PPA is out-of-date. We need your help for upgrading the packages.

If you use [Ubuntu](#), you can update your system with packages from our PPA by adding `ppa : couchapp / couchapp` to your system's Software Sources.

Follow [instructions here](#).

Installing on Windows

There are currently 2 methods to install on windows:

- [Standalone Executable 1.0.1](#) Does not require Python
- Python installer for Python 2.7 Requires Python

Previous Release

Please check out both [release section](#) and [download section](#) in GitHub.

Note that the download section in GitHub is [deprecated](#).

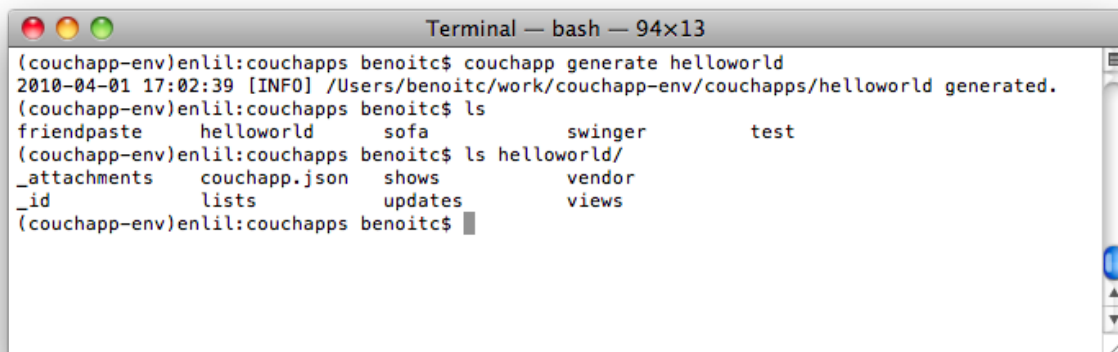
Getting Started

In this tutorial you will learn how to create your first CouchApp (embedded applications in [CouchDB](#)) using the couchapp script.

Generate your application

couchapp provides you the `generate` command to initialize your first CouchApp. It will create an application skeleton by generating needed folders and files to start. Run:

```
$ couchapp generate helloworld
```

A screenshot of a macOS Terminal window titled "Terminal — bash — 94x13". The window shows the execution of the 'couchapp generate helloworld' command. The output indicates that the application was generated successfully at the path '/Users/benoitc/work/couchapp-env/couchapps/helloworld'. Subsequent 'ls' commands show the directory structure of the generated application, including folders like 'friendpaste', 'helloworld', 'sofa', 'swinger', and 'test', and files like '_attachments', 'couchapp.json', 'shows', 'vendor', '_id', 'lists', 'updates', and 'views'.

```
Terminal — bash — 94x13
(couchapp-env)enlil:couchapps benoitc$ couchapp generate helloworld
2010-04-01 17:02:39 [INFO] /Users/benoitc/work/couchapp-env/couchapps/helloworld generated.
(couchapp-env)enlil:couchapps benoitc$ ls
friendpaste  helloworld  sofa        swinger     test
(couchapp-env)enlil:couchapps benoitc$ ls helloworld/
_attachments  couchapp.json  shows        vendor
_id           lists          updates      views
(couchapp-env)enlil:couchapps benoitc$
```

```
$ couchapp generate
```

Create a show function

To display our hello we will create a show function.

```
$ cd helloworld/
$ couchapp generate show hello
```

Here the generate command create a file named `hello.js` in the folder shows. The content of this file is:

```
function(doc, req) {  
  
}
```

which is default template for `show` functions.

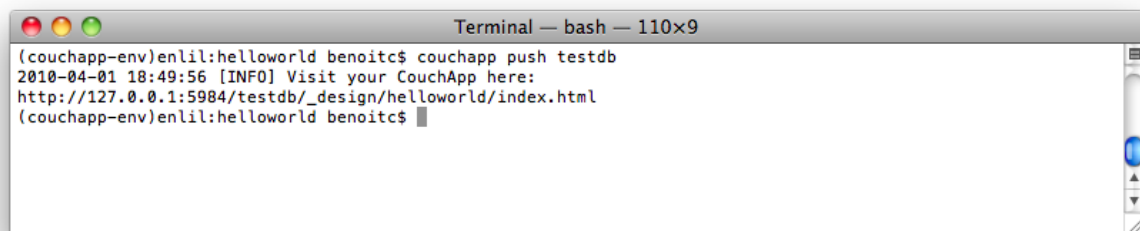
For now we only want to display the string “Hello World”. Edit your `show` function like this:

```
function(doc, req) {  
    return "Hello World";  
}
```

Push your CouchApp

Now that we have created our basic application, it’s time to push it to our CouchDB server. Our CouchDB server is at the url <http://127.0.0.1:5984> and we want to push our app in the database `testdb`:

```
$ couchapp push testdb
```



```
$ couchapp push
```

Go on http://127.0.0.1:5984/testdb/_design/helloworld/index.html, you will see:

```
CouchApp hello world
```

Clone your CouchApp

So your friend just pushed the `helloworld` app from his computer. But you want to edit the CouchApp on your own computer. That’s easy, just `clone` his application:

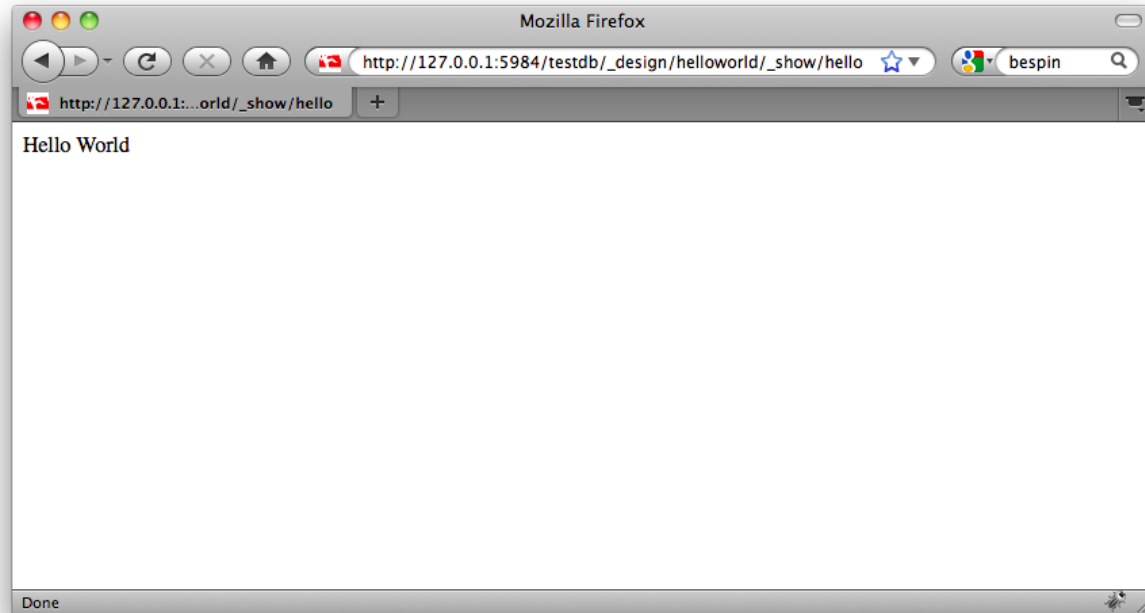
```
$ couchapp clone http://127.0.0.1:5984/testdb/_design/helloworld helloworld
```

This command fetch the CouchApp `helloworld` from the remote database of your friend.

```
$ couchapp clone
```

Now you can edit the couchapp on your own computer.

- External Resources
 - The [Standalone Applications](#) and [Managing Design Documents](#) chapters of the O’Reilly CouchDB book



CouchApp Command Line Tool

CouchApp is designed to structure standalone CouchDB application development for maximum application portability.

CouchApp is a set of scripts and a [jQuery](#) plugin designed to bring clarity and order to the freedom of CouchDB's document-based approach.

Command Line Usage

Warning: There are many undocumented commands. We need your help!

Full command line usage

```
Usage: couchapp [OPTIONS] [CMD] [CMDOPTIONS] [ARGS,...]

Options:
  -d, --debug
  -h, --help
  --version
  -v, --verbose
  -q, --quiet

Commands:
  autopush [OPTION]... [COUCHAPPPDIR] DEST
            --no-atomic           send attachments one by one
            --update-delay [VAL]  time between each update
  browse   [COUCHAPPPDIR] DEST
  clone    [OPTION]...[-r REV] SOURCE [COUCHAPPPDIR]
            -r, --rev [VAL] clone specific revision
```

```
generate [OPTION]... [app|view,list,show,filter,function,vendor] [COUCHAPPPDIR]
↳NAME
    --template [VAL] template name
help
init    [COUCHAPPPDIR]
push    [OPTION]... [COUCHAPPPDIR] DEST
        --no-atomic    send attachments one by one
        --export       don't do push, just export doc to stdout
        --output [VAL] if export is selected, output to the file
        -b, --browse   open the couchapp in the browser
        --force        force attachments sending
        --docid [VAL]  set docid
pushapps [OPTION]... SOURCE DEST
        --no-atomic    send attachments one by one
        --export       don't do push, just export doc to stdout
        --output [VAL] if export is selected, output to the file
        -b, --browse   open the couchapp in the browser
        --force        force attachments sending
pushdocs [OPTION]... SOURCE DEST
        --no-atomic    send attachments one by one
        --export       don't do push, just export doc to stdout
        --output [VAL] if export is selected, output to the file
        -b, --browse   open the couchapp in the browser
        --force        force attachments sending
startapp [COUCHAPPPDIR] NAME
vendor   [OPTION]...[-f] install|update [COUCHAPPPDIR] SOURCE
        -f, --force    force install or update
version
```

Commands

generate

Allows you to generate a basic couchapp. It can also be used to create *template* of functions. e.g.:

```
$ couchapp generate myapp
$ cd myapp
$ couchapp generate view someview
```

init

Initialize a CouchApp. When run in the folder of your application it create a default `.couchapprc` file. This file is needed by couchapp to find your application. Use this command when you clone your application from an external repository (git,hg):

```
$ cd mycouchapp
$ couchapp init
```

push

Push a couchapp to one or more [CouchDB](#) server.

```
$ cd mycouchapp
$ couchapp push http://someserver:port/mydb
```

- `--no-atomic` option allows you to send attachments one by one. By default all attachments are sent inline.
- `--export` options allows you to get the JSON document created. Combined with `--output`, you can save the result in a file.
- `--force`: force attachment sending
- `--docid` option allows you to set a custom docid for this couchapp

pushapps

Like `push` but on a folder containing couchapps. It allows you to send multiple couchapps at once.

```
$ ls somedir/
app1/ app2/ app3/
$ couchapp pushapps somedir/ http://localhost:5984/mydb
```

pushdocs

Like `pushapps` but for docs. It allows you to send a folder containing simple document. With this command you can populate your CouchDB with documents. Another way to do it is to create a `_docs` folder at the top of your couchapp folder.

startapp

It's an alias of `generate app NAME`, e.g.:

```
$ couchapp startapp myapp
```

Configuration

.couchapprc

Every CouchApp **MUST** have a `.couchapprc` file in the application directory. This file is a JSON object which contains configuration parameters that the command-line app uses to build and push your CouchApp. The `couchapp generate` and `couchapp init` commands create a default version of this file for you.

The most common use for the `.couchapprc` file is to specify one or more CouchDB databases to use as the destination for the `couchapp push` command. Destination databases are listed under the `env` key of the `.couchapprc` file as follows:

```
{
  "env" : {
    "default" : {
      "db" : "http://localhost:5984/mydb"
    },
    "prod" : {
      "db" : "http://admin:password@myhost.com/mydb"
    }
  }
}
```

```
}  
}
```

In this example, two environments are specified: `default`, which pushes to a local CouchDB instance without any authentication, and `prod`, which pushes to a remote CouchDB that requires authentication. Once these sections are defined in `.couchapprc`, you can push to your local CouchDB by running `couchapp push` (the environment name `default` is used when no environment is specified) and push to the remote machine using `couchapp push prod`. For a more complete discussion of the `env` section of the `.couchapprc` file, see the [Managing Design Documents](#) chapter of **CouchDB: The Definitive Guide**.

The `.couchapprc` file is also used to configure extensions to the `couchapp` tool. See the [Extend couchapp](#) page for more details.

`~/ .couchapp.conf`

One drawback to declaring environments in the `.couchapprc` file is that any usernames and passwords required to push documents are stored in that file. If you are using source control for your CouchApp, then those authentication credentials are checked in to your (possibly public) source control server. To avoid this problem, the `couchapp` tool can also read environment configurations from a file stored in your home directory named `.couchapp.conf`. This file has the same syntax as `.couchapprc` but has the advantage of being outside of the source tree, so sensitive login information can be protected. If you already have a working `.couchapprc` file, simply move it to `~/ .couchapp.conf` and run `couchapp init` to generate a new, empty `.couchapprc` file inside your CouchApp directory. If you don't have a `.couchapprc` file, `couchapp` will display the dreaded `couchapp error: You aren't in a couchapp message`.

`~/ .couchapp`

Please see [App Template](#)

`.couchappignore`

A `.couchappignore` file specifies intentionally untracked files that `couchapp` should ignore. It's a simple json file containing an array of regexps that will be use to ignore file.

For example:

```
[  
  ".*\\.swp$",  
  ".*~$" ]
```

will ignore all files ending in `.swp` and `~`. Be sure to leave out the final `,` in the list.

You can check if `couchapp` really ignores the files by specifying the `-v` option:

```
couchapp -v push
```

Note: Windows doesn't like files that only have an extension, so creating the `.couchappignore` file will be a challenge in windows. Possible solutions to creating this file are:

Using cygwin, type: `touch .couchappignore cd /to/couchappand then notepad .couchappignore`

TODO: more information about other templates like vendor, view, etc.

App Template

Most of the time, you will use `couchapp generate` to create a new CouchApp with the default directory layout, example functions, and vendor directories. If you find yourself creating multiple CouchApps that always contain the same third-party or in-house files and libraries, you might consider creating a custom app template containing these files and using the `--template` option of the `generate` command to create your customized CouchApps.

After creating a new couchapp, you will have a project structure that looks something like [this template project](#). The following *libraries* are included with your new CouchApp by default.

~/ .couchapp

Custom templates are stored as subdirectories under the `~/ .couchapp/templates` directory. The name of the subdirectory is used in the `--template` option to specify which template files are to be used in the `couchapp generate` command. The default template name is `app`, so by creating `~/ .couchapp/templates/app` and placing files and directories under that path, you can replace almost all of the default files created by `couchapp generate`.

Libraries

CouchDB API `jquery.couch.js`

The JQuery library included with CouchDB itself for use by the Futon admin console is used to interact with couchdb. [Documentation](#)

CouchApp Loader `jquery.couch.app.js`

A utility for loading design document classes into your Javascript application

Mustache

A simple template framework

Extend couchapp

Couchapp can easily be extended using external python modules or scripts. There are 3 kind of extensions:

- extensions: allows you to add custom commands to `couchapp`
- hooks: allows you to add actions on pre-/post (push, clone, pushdocs, pushapps) events.
- vendors handlers: allows you to add support for different sources of vendor

Extensions

Extensions are eggs or python modules registered in the config file in `extensions` member, e.g.:

```
"extensions": [
    "egg:mymodule#name"
]
```

Eggs uri are entry points uri starting with `egg:` prefix. To just load python module use an uri with the form: `python:mymodule.myextension`.

To load eggs add an entry point in `couchapp.extension` sections. More info about entry points [here](#).

An extension is a python module loaded when couchapp start. You can add custom commands to couchapp via this way. To add custom commands simply add a dict named `cmdtable`. This dict is under the format:

```
cmdtable = {
    'cmdname': (function, params, 'help string'),
}
```

`params` is a list of options that can be used for this function (the `-someoption/--someoption= args`):

```
params = [
    ('short', 'long', default, 'help string'),
]
```

`short` is the short option used on command line (ex: `-v`) `long` is the long option (ex: `--verbose`)

`default` could be `True/False/None/String/Integer`

Hooks

Couchapp offers a powerful mechanism to let you perform automated actions in response of different couchapp events (push, clone, generate, vendor).

Hooks are eggs or python modules registered in the config file in `hooks` member, e.g.:

```
"hooks": {
    "pre-push": [
        "egg:couchapp#compress"
    ]
}
```

Like extensions egg uri start with `egg:` prefix and python module with `python:.`. Entry point are added to `couchapp.hook` distribution. Here is the declaration of coupress hook in couchapp *setup.py*:

```
setup(
    name = 'Couchapp',
    ...

    entry_points="""
    ...

    [couchapp.hook]
    compress=couchapp.hooks.compress:hook

    ...
    """,
    ...
)
```

hooks are python functions like this:

```
def hook(path, hooktype, **kwarg):
    ...
```

path is the directory of the CouchApp on the filesystem, hooktype is the name of the event pre-/post- (push|clone|generate|vendor) and kwargs a list of arguments depending on the event:

- push: dbs: list of Database object
- clone: source: the source of the couchapp to clone
- vendor: source, the uri of vendor, action, could be *install* or *update*.
- generate: None

Have a look in [compress hook source](#) for a complete example.

Vendors handlers

```
for vendor_uri in self.conf.get('vendors'):
    obj = util.parse_uri(vendor_uri, "couchapp.vendor")
    vendors_list.append(obj)
```

Vendors handlers are used to manage installation or update of vendors. Like extensions or hooks vendors handlers are eggs or python modules registered in config file:

```
{
  "vendors": [
    "egg:couchapp#git",
    "egg:couchapp#hg",
    "egg:couchapp#couchdb"
  ]
}
```

(above is the default). Entry point are added to couchapp.vendor distribution, e.g.:

```
setup(
    name = 'Couchapp',
    ...

    entry_points="""
    [couchapp.vendor]
    git=couchapp.vendors.backends.git:GitVendor
    hg=couchapp.vendors.backends.hg:HgVendor
    couchdb=couchapp.vendors.backends.couchdb:CouchdbVendor

    ...
    """,
    ...
)
```

A vendor is an object inheriting couchapp.vendor.base.BackendVendor class:

```
class MyVendor(BackendVendor):
    """ vendor backend interface """
    url = ''
    license = ''
    author = ''
    author_email = ''
    description = ''
    long_description = ''
```

```
scheme = None

def fetch(url, path, *args, **opts):
    ...
```

url is the url of the vendor source

license the license of the vendor

author name of author

author_email email of author

description short description of this vendor

long_description long description

scheme list of url prefix on which this handler will be use. (e.g.: ['git', 'git+ssh'] for git://lgit/ssh:// urls)

The fetch function take the url given in console, the path of couchapp.

Here is an example for the default git vendor:

```
class GitVendor(BackendVendor):
    url = 'http://github.com/couchapp/couchapp'
    author = 'Benoit Chesneau'
    author_email = 'benoitc@e-engura.org'
    description = 'Git vendor handler'
    long_description = """couchapp vendor install|update from git::

    git://somerepo.git (use git+ssh:// for ssh repos)
    """

    scheme = ['git', 'git+ssh']

    def fetch(self, url, path, *args, **opts):
        ....
```

Full source is [on the git repo](#).

Using CouchApp with Multiple Design Documents

Here is what I did to use couchapp with multiple design documents. I want to setup a project for a new database test6 with a design doc called design_doc1.

Make sure couchdb and couchapp are installed and that couchdb is started.

First check that test6 doesn't exist:

```
$ curl http://127.0.0.1:5984/test6
{"error": "not_found", "reason": "no_db_file"}
```

OK. That was expected.

Generate a new CouchApp:

```
$ couchapp generate test6
$ cd test6
$ ls
_attachments  eventually  language  shows      vendor
couchapp.json  _id        lists     updates    views
```

Now edit `.couchapprc` as follows so it looks like

```
{ "env":
  { "default":
    { "db": "http://127.0.0.1:5984/test6" }
  }
}
```

Note: It looks like couchapp doesn't pick up the default db in what follows when I do `couchapp push`

Make a directory for design documents:

```
$ mkdir _design
```

Make a directory for `design_doc1`

```
$ mkdir _design/design_doc1
```

move the design doc files created with couchapp generate to the `design_doc1` directory:

```
$ mv _attachments eventually lists shows updates vendor views ./_design/design_doc1
```

review the directory structure

```
$ ls _design/design_doc1
_attachments  eventually  lists  shows  updates  vendor  views
```

Now push `design_doc1`. Note that I have to include the url of the database as a parameter. Couchapp doesn't seem to pick up the default db when I push from the `_design` directory.

```
http://127.0.0.1:5984/test6      is the url of the new db

$ couchapp push _design/design_doc1 http://127.0.0.1:5984/test6
2010-08-23 15:47:45 [INFO] Visit your CouchApp here:
http://127.0.0.1:5984/test6/_design/design_doc1/index.html
```

Now check to see if db `test6` was created:

```
$ curl http://127.0.0.1:5984/test6
{"db_name": "test6", "doc_count": 1, "doc_del_count": 0, "update_seq": 1, "purge_seq": 0,
  ↪ "compact_running": false, "disk_size": 106585, "instance_start_time": "1282603665650439",
  ↪ "disk_format_version": 5}
```

Now go into a browser and take a look at the `test6` db

```
http://127.0.0.1:5984/_utils/database.html?test6
```

You should see `_design/design_doc1` listed on the html page. That's good, it means that `design_doc1` was created.

Take a look at `design_doc1` in the futon web admin. Open this URL in your browser:

```
http://127.0.0.1:5984/_utils/document.html?test6/_design/design_doc1
```

You should see a nice listing of the `design_doc1`. Try opening the index page in your browser:

```
http://127.0.0.1:5984/_utils/document.html?test6/_design/design_doc1/index.html
```

This should serve up `index.html` from the `_attachments` subdirectory `test6/_design/design_doc1/_attachments/index.html`.

Couchapp generate had created a sample view called `recent-items`. Try querying it:

```
$ curl http://127.0.0.1:5984/test6/_design/design_doc1/_view/recent-items
{"total_rows":0,"offset":0,"rows":[]}
```

That's it. Multiple design can be used to create different interfaces for users with different roles. For example, consider some data and the different ways that an admin versus a regular user interacting with it.

JavaScript Application Programming

All application logic in a couchapp is provided by JavaScript. There is a library called `jquery.couch.js` that is distributed with every CouchDB installation. Here is the [documentation for jquery.couch.js](#)

Using backbone.js with CouchApp

`Backbone.js` is minimalist mvc framework for JavaScript, written by Jeremy Ashkenas, the author of coffee script. Backbone is a good choice for creating larger CouchApps, as an alternative to Evently. A robust backbone-couchdb connector that supports realtime updates via the `_changes` feed is supported by Jan Monschke.

See this [introduction](#) to CouchApp with `backbone.js` by Jan.

Extended version of `backbone` couch connector (with fixing some issues, extending functionality) is available here:

<https://github.com/andrzejliwa/backbone-couch>

An example use case

<https://github.com/andrzejliwa/couch-watch> Also check out the [List of CouchApps](#).

The Garden

Warning: The original database of garden was gone. Some legacy resources are [here](#).

The [CouchApp Garden](#) is a CouchApp designed to make sharing other CouchApps easy. Once you have the Garden installed on your CouchDB, you can use it to install other CouchApps.

The basics

Currently, the Garden needs a lot of work, but the basic ideas are there. Essentially, it can copy design documents from your other databases, into the Garden database. As it copies them, it renames them so that they don't have ids that start with `_design`. This means they can be replicated around without the replicator having to run with admin privileges. Also, the apps don't run code when they are just sitting in the Garden.

Once you have a local Garden database, you can install apps from it, into databases on your CouchDB. The garden document will be copied to the target database, as a design document again, and there will be a link to visit that application.

Sharing your app

To add your app to the Garden, install the Garden locally, and use its import link, to add the app to your local garden database. Then replicate that database to <http://couchapp.org/garden>, and check out the updated [Garden](#).

Contributing to the Garden

The [Garden](#) code is on [Github](#), please fork and contribute.

CouchApps and DesktopCouch

In **version 0.7**, `couchapp.py` has a new feature allowing you to push, clone and browse CouchApps in the local CouchDB installed with [desktopcouch](#), so users of linux distributions where desktopcouch has been ported won't have to install another CouchDB to test and will be able to pair it with other desktop.

How it works?

To push to your local couchdb installed with desktopcouch:

```
couchapp push desktopcouch://testdb
```

To clone:

```
couchapp clone desktopcouch://testdb/_design/test test1
```

To browse and use your application:

```
couchapp browse . desktopcouch://mydb
```

and with push option :

```
couchapp push --browse . desktopcouch://mydb
```

List of CouchApps

Please add links to CouchApps (alphabetical order will help avoid duplicates in the long run). You may also be interested in the CouchApp Garden.

Afghan War Diary

A GeoCouch app that provides a browseable map of entries from the Wikileaks Afghan Diaries.

[Code](#)

BlueInk

The beginnings of a conversion of the [BlueInk CMS](#) to a CouchApp. Currently, it can serve as a view-engine for web pages wearing [Mustache](#) templates.

[Code](#)

Bookkeeping

A little CouchApp that helps visualizing expenses for my household. Currently in German only. I am working on a branch that uses views and `_changes` instead of a pure client side implementation with jquery.

[Code](#)

Boom Amazing

Presentation software with a twist. Uses SVG and pan and zoom. Based on [Sammy.js](#).

[Code](#)

Brunch-Colors

Brunch-Colors is a simple, addictive color-matching game that was made with [Brunch](#) that utilizes such tools as [Backbone.js](#), [eco](#) and [stylus](#).

[Play it here](#)

[Code](#)

Costco

A small UI for bulk editing CouchDB documents.

[Code](#)

CouchCrawler

Spiders the web into CouchDB. Uses a Python script for web spidering. [Read the blog post here](#)

[Code](#)

CouchWatch

Simple logs watcher with realtime view and simple searching. For using with Rails and JavaScript logger. Written in [Backbone.js](#)

[Code](#)

CouchDB Contact Form

Simple Contact Form CouchApp for CouchDB. Includes simple mail spooler.

[Code](#)

CouchDB Projector

For doing presentations.

[Code](#)

CouchLog

Application Logging tool. Uses a CouchDB backend with a CouchApp-based interface for sorting through log entries and troubleshooting/debugging applications. Leverages schema-less approach to allow log entries to contain structured meta-information to aid in troubleshooting

[Code](#)

csv2couchdb

small app to populate couchdb using data from CSV files

[Code](#)

Dimensional Drawing

Collaborative 2.5D drawing space.

[Code](#)

[Demo](#)

Focus

A TODO tracker that replicates. Run it on your phone, run it on your server, run it on your laptop. Keep them synchronized. Never forget to do that important thing!

[Code](#)

Deployments:

- [Demo](#)

Food Cart Pages

A catalog of all the food carts in Portland.

Deployments:

- <http://foodcartpages.com>

HejHej

A CouchApp for language learning. Lets you train vocabularies and solve different kinds of games/tests. Has Cucumber tests.

[Code](#)

Hub List

Open source GTD style productivity app. Manage your tasks from bug trackers, pm tools and other online todo lists all in one place. Built with Ext JS 4.

[Code](#)

IrcLog CouchApp

A couchapp to view irc logs stored in CouchDB.

The irclogs can be stored by [gdamjan's ircbot](#) and its [couchdb logging plugin](#).

[Code](#)

Li.Couch

Open source LList notes. Easy track of your items. Built with Knockout.js.

[Code](#)

[Demo](#)

MapChat

A real time chat app on a Google Map. Points on a map as a chat rooms.

[Code](#)

[Demo](#)

Microanalytics

Personal hackable web-analytics.

[Code](#)

[Python command-line client](#)

Modern Forum

A new project aiming to bring real-time, CouchDB-powered forums to the masses.

[Code](#)

Monocles (ex-CouchAppSpora)

diaspora... as a couchapp! in pure javascript and fully OStatus compliant (almost)

[Code](#) and [more info](#) [Demo](#)

MTG Pricing CouchApp

A mobile-centric app to get the pricing information for Magic: The Gathering cards quickly and easily.

[Code](#)

Mytweets

A personal Twitter archive.

Deployments:

- [@yssk22](#)

Nymphormation

A social link sharing tool.

[Code](#)

Deployments:

- [Nymphormation](#)

Pages

A Markdown wiki. This was the wiki used to create this documentation originally.

Install Pages

If anyone else has issues understanding this whole vhosts thing i'll give a *grandma-can-do-it* recount here of my troubles.

If you want to set up *pages* on your machine and you are not very familiar with what's what, here it goes:

Get pages from github: <http://github.com/couchone/pages>

Navigate to your fav directory and do this in the terminal:

```
git clone git://github.com/couchone/pages.git
```

Make sure to install couchapp python helper app, i won't go into the details, [the github instructions are great](#)

Hope you have couchdbx installed on osx or the equivalent on your favourite dev/production platform

Do this from inside your pages directory:

```
couchapp init
couchapp push . http://localhost:5984/pages
```

That will get you the app into your db, nothing new here, the git hub instructions will tell you the same thing, what got me (besides a bad commit ;)) is the instruction on <http://blog.couch.io/post/443028592/whats-new-in-apache-couchdb-0-11-part-one-nice-urls>

The section about vhosts is a bit ambiguous for those that aren't in the know...

The instructions there are as follows:

“Each HTTP 1.1 request includes a mandatory header field Host: hostname.com with the server name it is trying to reach. You can tell CouchDB to look for that Host header and redirect all requests that match to any URL inside CouchDB by adding this to your configuration file local.ini:

```
[vhosts]
couch.io = /couchio/_design/app/_rewrite"
```

Well, what the hell is local.ini?

Who knows, who cares, go to your couchdb app and navigate to the *configuration* section.

Go to the bottom of the page and “Add new section”

Type into the 3 fields that popup:

- vhosts
- your-pages-site-name:5984
- /pages/_design/pages/_rewrite

Now go to the terminal and type:

```
textmate /etc/hosts
```

(Notice i’m making assumptions here, basically, get to the hosts file and open it...)

Add:

```
127.0.0.1 your-pages-site-name
```

Save, go to a browser type:

```
your-pages-site-name:5984
```

Hopefully that worked out OK for you.

See ya!

[Code](#)

Deployments:

- [CouchApp](#)

Processing JS Studio

Web-based application to store Processing JS sketches and renderings. Storage and service provided by CouchDB via CouchApp.

[Code](#)

Proto

A basic CouchApp for inputting info from a form, and listing it in real time. This is the starting point for many other applications, as well as the [Evently Guided Hack Video Tutorial](#).

[Code](#)

Or run `couchapp generate foo` to get your own version, ready for hacking.

Deployments:

- [jChris](#)
- [Jan](#)
- [Goto](#)

Random Lecture!

A simple Sammy-On-CouchApp (soca) app that plays a random technical lecture or tech talk.

- [Demo](#)
- [Code](#)
- [List of all lectures](#)

Sales Stats

A simple CouchApp Demo that displays sales statistics as a bar graph. It uses the `_changes` API together with `Eventually`, so that the sales statistics are updated live (in near realtime).

[Code](#)

[Demo](#)

Scrapboard

A decentralized implementation of the old Orkut scrapbook.

[Code](#)

Skim - Simple knowledgebase for insightful metabolomics

The vision behind Skim is to develop a tool that can help analyze vast quantities of peer reviewed and community-provided information on metabolites, biochemical reactions and pathways. Heavily under development - may be unstable from time to time.

[Code](#)

[Demo](#)

Sleepcam

Whenever a user's computer wakes from sleep, the software takes a picture with their webcam and posts it to their profile on [sleepcam.org](#). Users can like and comment on eachother's pictures.

[Code](#)

[Demo](#)

Sofa

Standalone CouchDB Blog with tagging, Atom feeds, and gravatar comments , used by the O'Reilly CouchDB book.

[Code](#)

Deployments:

- [Daytime Running Lights](#)
- [Chewbranca](#)
- [Plok Light](#)
- [Blog Bleeds](#)

Snippets

A Couchdb snippets app with a Couchfuse backend.

[Code](#)

Swinger

A presentation engine. Like Keynote in the browser, but simpler. Uses [Sammy.js](#).

[Code](#)

Deployments:

- <http://swinger.quirkey.com>

TapirWiki

A wiki couchapp. Uses textile as the markup language and has a few macros, templates and support for attachments.

[Code](#)

Taskr

A task tracker. This one got deprecated by Focus. It's got some cool features so it's worth looking at if you are building something similar.

[Code](#)

The Infinite Maze

A collaborative maze drawing app.

[Code](#)

[Demo](#)

Toast

A real time chat app. One of the first demos of the `_changes` API.

[Code](#)

[Demo](#)

Tweet Eater

A Twitter search archive and real time display. Uses a Ruby backend to import tweets from the streaming API.

[Code](#)

hckr.it

A [Hacker News](#) clone built entirely using CouchDB that can be served as a couchapp.

[Code](#)

[Demo](#)

The following documentation tell you how `couchapp.py` works.

The CouchApp Filesystem Mapping

The `couchapp` script has a cool way of pushing files to CouchDB's design documents. The `filesystem mapping` is done via the `couchdbkit` Python library.

If you have folders like:

```
myapp/  
  views/  
    foobar/  
      map.js  
      reduce.js
```

It will create a design document like this:

```
{  
  "_id" : "_design/myapp",  
  "views" : {  
    "foobar" : {  
      "map" : "contents of map.js",  
      "reduce" : "contents of reduce.js"  
    }  
  }  
}
```

This is designed to make it so you get proper syntax highlighting in your text editor.

Complete Filesystem-to-Design Doc Mapping Example

```
myapp/
  _attachments/
    images/
      logo.png
  _docs/
    sample.json
    doc_needing_encoding/
      _id (the ID for the document as text on the first line of this file)
      title (same as ID, just for the title field. Repeat pattern as needed)
      content.html (HTML content that will be encoded when it's added to the JSON doc)
  lists/
    xml.js
  rewrites.js
  shows/
    preview.js
    xml.js
  updates/
    in-place.js
  views/
    foobar/
      map.js
      reduce.js
  validate_doc_update.js
```

The `_attachments` folder will turn each file into an attachment on the resulting Design Document. The attachments will be named based on their file path (ex: “image/logo.png”).

The contents of the `_docs` folder are turned into actual JSON documents in CouchDB. The contents of the `.json` files will be input exactly as they are in the file. The name of the document will be either the file name or the `_id` field from the JSON object in that file.

Folders under `_docs` will be turned into documents with each file in the folder being a key/value pair in the resulting JSON document. HTML and XML files (and maybe others?) will be JSON encoded before being added to the JSON document. An `_id` file will be used (if present) as the ID of the new document. Otherwise the folder name will become the ID.

The rest of the folder structure above will become this JSON Design Document

```
{
  "_id" : "_design/myapp",
  "_attachments": {
    "images/logo.png": {
      "content_type": "image/png",
      "revpos": 1,
      "digest": "md5-GDPL+eLwE7kzEDWY7X4KdQ==",
      "length": 886,
      "stub": true
    }
  },
  "lists": {
    "xml": "function..."
  },
  "rewrites": "function...",
  "shows": {
    "preview": "function...",
    "xml": "function..."
  }
}
```

```
"updates": {
  "in-place": "function..."
},
"views": {
  "foobar": {
    "map": "function...",
    "reduce": "function..."
  }
},
"validate_doc_update": "function...",
}
```

Contributing to CouchApp Ecosystem

Contributing

This repository holds all of the code in the project. The Python `couchapp` script is the bulk of the repository, but the JavaScript stuff is in there to.

The `jquery.couch.app.js` and `jquery.evently.js` files are both [in the vendor directory](#).

If you have a commit to one of the CouchApp files (JavaScript or otherwise) please let us know on the [mailing list](#) as we don't always get the messages in our Github inbox.

Also, documentation and blog posting is **very much appreciated**. Don't be afraid to tell us how CouchApp sucks. We want it to be very easy to use, so giving us a high bar to reach is important.

If you prefer developing mobile apps with Titanium, @pegli maintains a module which wraps Couchbase Lite for that platform.

If you've built a sync powered app and are starting to hit the point where Apache CouchDB filtered replication doesn't scale for you, you might want to check out the Couchbase Sync Gateway which uses the same sync protocol but is designed to give efficient subsets of a big data corpus to sync clients. So you can sync to CouchDB or Couchbase Lite (nee TouchDB).

Or simply use rcouch a custom distribution of Apache CouchDB with a bunch of new features that offers since a while incremental view changes (indexed on the disk) and replication support using a view and allows you to replicate in an efficient manner subsets of your databases.

Last thing, PouchDB is the future of browser based sync apps. It can sync with the same sync protocols but uses the built-in storage of HTML5.

Roadmap

Warning: The content is out of date.

Developer Toolchain

`couchapp.py`

- upload only changed attachments (md5 on attachment stubs?)
- Coverage rate improvement
- Python3 support

Node.js CouchApp Tools

- try Mikeal's Node.js CouchApp style

JavaScript Libraries

`$.couch.app()`

- Make this responsible only for loading code from the Couch, and bootstrapping the CommonJS runtime.
- make `$.couch.app.utils` a commonjs library.
- move into Apache CouchDB's `share/www/script`

RFC: Please Comment

This please suggest anything you think is missing.

CHAPTER 6

Other Resources

- **IRC**
 - #couchdb
 - #couchapp
- Search The CouchDB Mailing List/IRC Archive
- **Mailing Lists**
 - http://mail-archives.apache.org/mod_mbox/couchdb-couchapp/
 - <http://groups.google.com/group/couchapp>
- eNotes CouchApp Tutorial