I'm reposting here an email I wrote since it was well-received on the CouchDB marketing list, but its formatting did not display well there.

One of CouchDB's developers asked,

How can we make it that CouchApps strengthen CouchDB and not weaken it by adding confusion?

How do CouchApps fit into the CouchDB story?

I've been thinking about this discussion a little, and wanted to offer a few ideas from the perspective of a user. In essence, I think the CouchDB community should focus on marketing two concepts named "CouchDB," and "Couch App Server." This is a middle ground between the two extremes of "R.I.P. CouchApps" and "CouchApps are CouchDB's greatest feature!" The middle ground is achieved by **distinguishing** CouchDB and CouchApps, and marketing them as two distinct ideas. You don't have to
stop
marketing the idea of CouchApps; just market them as something
distinct
from CouchDB. Both can be positively marketed, and succeed or fail on their own merits. But by distinguishing the two, if CouchApps confuse people, they need not turn people away from CouchDB. My thoughts and reasons for this are below.

1. Evaluate the current marketing. CouchApps are mentioned front and center in the first two paragraphs about CouchDB at http://couchdb.apache.org/ . They are mentioned in concept, though not by name. They are implied in the slogan, "A Database for the Web" which is explained by those two paragraphs.
If CouchDB's marketing should take a different direction in the future, that marketing ought to deal with the question of whether the promises made in those two paragraphs are true
. Specifically, "CouchDB comes with a suite of features, such as on-the-fly document transformation and real-time change notifications, that makes web app development a breeze." Does CouchDB "make web app development a breeze," or not? These two paragraphs are the first ones to change.

2. Make CouchApps secure. If multiuser data within CouchApps cannot be properly secured presently because CouchDB does not require clients to send the Host header (I tried to test whether this is the case per this email), then **make a config option to make CouchDB require the Host header** . It sounds easy to do, and the Host header is required in HTTP 1.1. Or create a "default _rewrite path" configuration parameter as Giovanni described. I expect this would make SmileUpps' CouchApp architecture secure for anyone who wants to use that architecture.

3. Don't promise CouchApps are easy. SmileUpps' CouchApp architecture is the only CouchApp architecture I'm aware of which has (almost) implemented document-level ACLs without some proxy server between the browser and CouchDB. Others may exist; I just don't know about or remember them. I have 15 years of part-time experience in full-stack web development, and have written two small CouchApps. It doesn't appear to me that SmileUpps' CouchApp architecture is particularly easy for novices to learn and implement, at least at present. Perhaps with a Yeoman generator or the like it could become easy. But **its current complexity does not seem to me to "make web app development a breeze,"** that is, if you want to prevent some users from reading all the data in one database, which is a normal or at least common requirement in web development. The end result is not good--CouchDB's promise of easy CouchApps will lead novices to build insecure apps. I wish CouchApps did make web app development a breeze, and would like to see CouchDB still be able to use that promise in its marketing. But for now, it seems to me that CouchApps shouldn't be marketed to novice developers as an easy point of entry into web development. CouchApps are rather a way for developers who already know (one of the many ways) how to structure a single-page app to serve that app out of CouchDB, and access CouchDB as the app's database. It seems to me a developer should learn Backbone or Angular before CouchApps (like the Chatty tutorial assumes: https://www.smileupps.com/couchapp-tutorial-chatty-read-api). So, because they generally require 1) knowledge of a client-side framework, 2) knowledge of CouchApps' file structure and functionality, and 3) implementing a very specific CouchApp configuration to be properly secured, CouchApps aren't really an entry point into web development. Instead,
CouchApps are
a way for non-novice developers to use CouchDB as both a database and an app server.

4. CouchApps could rise again! CouchApps' prospect of master-master replication of not only data, but apps, remains attractive to me. Give users their data, and give them their code! It's a powerful thing. What if we all owned Facebook running in PouchDB in our browsers, without a persistent central server? Diaspora, CouchAppSpora/Monocles, and a bunch of other software have aimed in this direction. So **I don't think it's wise to pull back completely from marketing CouchDB's CouchApp features.** Perhaps they could still be the future.

5. Reprioritize marketing to serve today's web. "CouchDB is [still] a database that completely embraces the web." But **the web has changed since CouchDB first embraced it.** Web apps have moved toward offline-first and syncing data to the server via REST. Web apps don't live on the server anymore. They live in your phone. So, as an app developer, I don't need routing or SEF URLs (vhosts/rewrites) on the server; I need routing in the client. While I still want to be able to query a CouchDB view via HTTP, or maybe even a show or list on occasion to provide an RSS feed, more often I want to query my data locally in PouchDB or something like it. So it doesn't make sense to me to prominently market CouchDB's "on-the-fly document transformation" anymore. I still want the feature, but it's not in the core of what I need.
What I need is a database that syncs.
CouchDB's other application server features are nice when I want them.

6. Market an accurate Venn diagram. The relationship between CouchDB's CouchApp features and CouchDB's non-CouchApp features is not one where you can divide the features neatly into two mutually-exclusive sets. Rather, **CouchApps use nearly all of CouchDB's features** (except maybe replication), and non-CouchApps use a subset of CouchDB's features. CouchApps may or may not use replication depending on whether they use something like PouchDB, and whether they use replication for deployment. To put it in terms of my main proposal, future "CouchDB" would be a subset of CouchDB's current features, and "Couch App Server" would be a superset of that future "CouchDB" set. This makes me think that it is hard for people to grasp what it means to disable the features which support CouchApps, because they too easily believe this means disabling the superset; disabling the whole of CouchDB 1.x's features. So I think the best way to explain this to new users is to say that
CouchDB is a database, and it comes with some extra features which are useful for an application server.
The first two paragraphs at
http://couchdb.apache.org/
say as much, but they do not distinguish these two concepts in a way that is obvious and memorable to a newcomer, or that would convince the newcomer that they might want to use CouchDB as a database alone apart from its additional application server features. The newcomer is left with the impression that the features unique to "Couch App Server" are actually part of the database, but while they are part of CouchDB, yet they are not technically database features; instead, they are web file server and document transformation features.

The way I perceive the natural groupings of CouchDB's features is as follows:

i) **Database:** The database that syncs (views (=indexes), HTTP interface, replication, filters,

MVCC, changes feed, authentication)
ii) [App Server?]: Web file server (vhosts, rewrites, attachments) & design docs/stored procedures for further document transformation (views, shows, lists, update handlers)

i) is the core set of features, and i) + ii) = CouchApps. I think this distinction should be displayed textually and graphically on CouchDB's front page.

Interestingly, ii) contains features of the past (server-side apps) and the (maybe) future (client-side apps, deployed through replication). It would be worth noting this distinction on CouchDB's front page to help newcomers decide to what extent they need Couch's app server features. It remains possible that someday the present swing toward client-side apps will reverse, and even these server-side features could become more of the future. So I don't think the marketing should characterize CouchApps as a dying remnant from the past which will not be relevant in the future.

Earlier in CouchDB's lifetime we thought CouchApps using all of ii) were the future. Now because there is less need for server-side routing and document transformation, the main parts of ii) which might be the future are views (for occasional direct queries over HTTP - yes, I included views in both i & ii), attachments (for serving/replicating your app's files from the database), and lists (for RSS feeds or data syndication/federation through filtered replication). But it would be good to say on CouchDB's front page that a developer might not need any of the features in ii) today.

So Couch's app server features should not be marketed as extra features for advanced users. They don't add functionality which every developer will want after they master the basics. Rather, **they are extra features which permit you to write limited server-side logic for your application, if you find you need it.** This is actually a useful point for a newcomer, because while single page applications running in the browser can perform most of the logic we need today, yet (aside from issues of data security and proprietary code, which are the realm of server-side node changes listeners) due to reasons of architecture and efficiency, we are not able to run all our logic in the browser; it's still wise to keep some logic in the database on the server side. Often we're still dependent on server-side logic, and Couch's app server features can meet that need to a limited extent.

One result of the features in ii) is that your app's server-side and client-side logic can be synced from one server (CouchDB instance) to another. For example, from your local

development machine to the deployment server, or from one deployed application instance to several other nodes all running the same application, but with different filtered sets of data. This warrants the slogan, **"Apps that sync!"** However, that slogan might make people think the syncing in view is between CouchDB and the web browser, which is not what I mean by the slogan. The apps' syncing is actually done by the features under i), but they are apps because of the features under ii). So **the "Apps that sync" do so because they are in "The database that syncs."**

7. My proposal. So **I propose splitting the features** described in the first two paragraphs at http://couchdb.apache.org/ , features which are mixed together there under the one name of "CouchDB" without any clear distinction,
into two sets of features under two separate names and slogans:

1. **"CouchDB" - the database that syncs!**
2. **"Couch App Server" - apps that sync!**

Since I (as an app developer) only need CouchDB, market CouchDB most prominently. But since Couch App Server is nice when I want it, and might be a good method for deploying app updates, market it as a nice but limited set of server-side features your app can use (even to serve your app), which can be secured, and which can be used to deploy app updates, but which you probably don't need if you are using a modern client-side application architecture. If people want server-side features, this will be a selling point. If people don't want server-side features, they will appreciate being told that they don't need to research CouchApps.

Because I want Couch App Server's features sometimes (RSS feeds), I think they should be enabled by default. I don't see why it would be necessary to provide a configuration option to turn them off, but I wouldn't mind if they were turned off.

Though a graphic designer could figure out a better way than this, I envision presenting this distinction between "CouchDB" and "Couch App Server" in a **graphic** which shows CouchDB as the **fundamental layer** of Couch's architecture, and Couch App Server as an **optional layer** on top of CouchDB which includes additional features. Below the graphic, I envision **two columns of text**

, with CouchDB's description on the left, and Couch App Server's description on the right, and somehow CouchDB being portrayed as the most important of the two. If you want to hide Couch App Server on the front page, then present only CouchDB on the front page, and provide a link to a page describing "Additional features" which presents the two-layer graphic and two-column descriptions I mentioned above.

If you want to remove the word "Couch" from "Couch App Server," just call it the **"App Server."** That was easy. :)