CouchApp Tutorial Documentation

Release 0.2

Markus Mayr

January 20, 2012

CONTENTS

You will learn how to build a simple note taking application which is stored in a CouchDB. The complete logic, design and data of the application will be contained within a single database. This gives the benefit of CouchDBs replication feature and other database specific properties to your application for free. The application will run in any recent webbrowser that is able to interpret JavaScript - this means that it will work with mobile devices as well as with desktop computers.

Basic knowledge of CouchDB, JavaScript and HTML is assumed. We will combine these and other techniques to produce a simple example.

In this tutorial, system specific commands are based on *Linux* / *Bash*.

Please be advised that you should follow this tutorial step by step if you want to have your CouchApp working right. There are a lot of cross references inside the code and it is easy to lose track. However, if you are confident enough to do otherwise, please feel free to do so. ;-)

If you are interested in expanding this tutorial or correct an error, take a look at it's Github page: https://github.com/scubbx/CouchApp-eNotes-Tutorial

The code on github is open source.

CHAPTER

ONE

NEEDED SOFTWARE

Here will be explained what software to use and why. This page gives only a small overview of the capabilities of software used for CouchApps. For a more detailed explanation please refer to tutorials written specifically for each individual software.

For our application, we need three types of information:

- Layout
- Data
- Program Logic

For this we will only use software and software-principles that are also used in current state of the art web development projects:

- HTML & CSS
- JavaScript (including JSON, jQuery and evently)
- moustache
- CouchDB

The responsibilities of different pieces of software can be seen like this:

Part of Application	Techniques used
Layout	HTML, CSS, jQuery, mustache
Data	JSON, evently
Program Logic	JavaScript, jQuery, evently

1.1 HTML & CSS

Everyone involved in web-programming knows HTML at least to some extent. In the CouchApp context, this *Hyper-TextMarkupLanguage* is used to describe the user interface and design of the application. It can easily be stored inside the CouchDB as an **attachment** and accessed by a single URL.

Example:

```
<html>
        <head>

        </head>
        <body>
                <a>Lore ipsum</a>
        </body>
</html>
```

HTML holds the **layout** of the CouchApp.

1.2 JavaScript

Since HTML does not allow any program logic or multidirectional interaction to be implemented by itself, JavaScript is used to accieve this goal. This is only logical because it is natively supported by various browsers on different systems. JavaScript code can be stored inside the CouchDB as **part of CouchDB - documents** and accessed via the REST interface.

Example:

```
function testFunction() {
        alert("This is just a test.");
}
```

JavaScript is responsible for **storing the logic** of the program.

1.2.1 JSON

There exists a standard notation for describing data in JavaScript. It is called *JavaScriptObjectNotation*. Actually, this is just a data container (variable) in pure JavaScript.

Example:

```
{
        "Entry1" : "Value1",
        "Entry2" : "Value2",
        "Entry3" : "2649",
        "Entry4" : true,
        "Entry5" : [ "blue", "green", "red" ],
        "Entry6" : null
}
```

JSON entries can also be nested:

```
{
        "Entry1" : "Value1"
        "Entry2" : "Value2",
        "Entry3" : {
                "SubEntry1" : "Value3",
                "SubEntry2" : 342
        },
        "Entry4" : "Value4"
}
```

JSON is used whenever information is processed. This covers **data** or **program logic**.

1.2.2 jQuery (mobile)

jQuery is a JavaScript library that simplifies access to the DOM structure of HTML documents. It also offeres standardised ways to access functions that differ with different browsers and offeres additional effects for the design of HTML pages.

jQuery syntax example:

```
$("a").click(
        function() {
                alert("This is just a test.");
        }
);
```

This code means that whenever you click on an element of the HTML page that is formatted with an `<a>` tag (basically every normal text or hyperlink) the function `function(){ alert("This is just a test."); }` is executed. The useage of jQuery with CouchApp is mainly supported by its standardising and graphical powers. It is mainly responsible for the **layout**. We will use a specialized mobile - version of jQuery called *jQuery mobile*.

1.2.3 jQuery UI CSS

The jQuery UI CSS is a simple css template which facilitates the creation of custom control elements in a html page. There is also a mobile - version available which will be used in our context.

http://jqueryui.com/

1.2.4 evently

Evently is a wrapper for jQuery functions. It facilitates the manipulation and addition of event handlers to DOM elements. It also has a mustache parser and allows to apply them via ajax requests.

Basic Example:

```
$("exampleDOMobject").evently( {
        click : function() {
                alert("This is a Test.");
                },
        mouseenter : function() {
                alert("This is ANOTHER Test.");
                }
        }
);
```

Here we can see a structure similar to a JSON object that defines different functions for different events. The purpose of evently with CouchApps lies within the **program logic** and **data** handling.

1.3 mustache

Mustache is another JavaScript library that lets us insert placeholders in HTML (or any other text). These are specified by `{{` at the beginning of the placeholder and `}}` at the end. Take this example for a template text with placeholders like `{{city}}` or `{{temperature_c}}`:

```
You are at {{city}}.
The temperature is {{temperature_c}} degrees C!
{{#in_f}}
        In Fahrenheit this would be {{temperature_f}}
{{/in_f}}
```

The reason this tool is used is that it takes JSON objects as input and inserts its values into placeholders of corresponding name.

The lines between `{{#in_f}}` and `{{/in_f}}` are only rendered when the variable `"in_f"` is set to `true`.

This is the JSON object that is fed into the mustache parser:

```
{
        "city": "Vienna",
        "temperature": 22,
        "temperature_f": ( (22 * 9) / 5 ) + 32,
        "in_f": true
}
```

Every `{{city}}` in our template HTML is replaced by the value of the `city` tag in the JSON object. In this case, the substitute would be `Vienna`. As you can see by `"temperature_f"`, you may also put simple formulas as values (like `( (22 * 9) / 5 ) + 32` in this case) . The final output of the template and the JSON object would look like this:

```
You are at Vienna.
The temperature is 22 degrees C!
In Fahrenheit this would be 71.6
```

Mustache is perfectly suited for our needs because we will receive data our CouchApp requests from the CouchDB as JSON objects.

1.4 CouchDB

CouchDB is the heart of every CouchApp. It is the storage of and interface to every bit of information. One does not need to gain knowledge of or implement a specific API since CouchDB uses simple HTTP requests to deliver data in JSON format. Attached files (like HTML or images) can be accessed by a specifically formatted URL.

```
http://127.0.0.1:5984/
```

This will give you a welcome message, the version of the database and possibly various other meta information in JSON format:

```
{
        "couchdb" : "Welcome",
        "version" : "1.1.0",
}
```

To get a list of all available databases, one would request:

```
http://127.0.0.1:5984/_all_dbs
```

To access previously attached files, only a specific URL is necessary. It is composed of the base URL (`http://127.0.0.1:5984`), the database name (`/albums`), the ID of the database document the file is attached to (`/6e1295ed6c29495e54cc05947f18c8af`) and, at last, the filename itself (`/artwork.jpg`):

```
http://127.0.0.1:5984/albums/6e1295ed6c29495e54cc05947f18c8af/artwork.jpg
```

CouchDB will not be explained in detail. For more information please refer to the excellent free online book "The Definitive Guide to CouchDB".

1.5 CouchApp

CouchApp itself is not only the name for the concept of storing an application inside an CouchApp, but also an application that helps us create CouchApps.

http://couchapp.org/page/what-is-couchapp

CHAPTER

TWO

FIRST STEPS WITH COUCHAPP

Here you will learn some details of the folder structure and the meaning of specific files inside your CouchApp directory. If you feel familiar enough with this, you may already proceed to the next topic, but bear in mind to at least generate an empty CouchApp before proceeding any further.

2.1 Start with CouchApp

2.1.1 Create working directories and files

To set up a CouchApp, we start the actual CouchApp program to generate a directory structure where all of our code will be saved to at corresponding places. After the upload of our CouchApp into a CouchDB, most of these files and folders will form the design document of our application.

Move to any directory you like. Don't worry, a sub directory for our project will be created in our next step automatically. For this example I will go to my *./Documents* folder and start the `couchapp` program with it's appropriate arguments. You may use whatever directory you like.:

```
$ cd Documents

/Documents$ couchapp generate enotes
```

The `couchapp` command is launched with the `generate` option. This will command couchapp to generate a new CouchApp project and trigger the creation of all necessary directories. The `enotes` part describes the name of our application. Acutally, you may change it to your liking, but for reasons of compatibility to this tutorial I recommend you to stick with this proposition.

If we take a look at the newly generated file tree we see the following directories and files (there are also hidden files!):

```
enotes$ ls

./_attachments
./lists
./shows
./updates
./vendor
./views
.chouchappignore
.couchapprc
couchapp.json
_id
language
README.md
```

We will go through each of this entries and explain what they are. If necessary we will also fill in certain values.

2.1.2 .couchappignore

This file defines files or folders that may be located inside our tree but should not be uploaded to the CouchDB. Regular Expressions (RegEx) are used to define these. The `.couchappignore` default looks like this:

```
[
// filenames matching these regexps will not be pushed to the database
// uncomment to activate; separate entries with ","
// ".*~$"
// ".*\\.swp$"
// ".*\\.bak$"
]
```

As we can see, all possible pre-defined values are switched off by the leading `//`. If you ever produce objects that are not necessary for your CouchApp to work (like a SVN control directory), you should include a line for them here. We do not need to do this for our CouchApp tutorial.

2.1.3 .couchapprc

The hidden `.couchapprc` file contains information of the CouchDB (or CouchDBs) our application will be integrated into in *JSON* format. The file looks like this when it is initially generated:

```
{}
```

We don't need to fill anything in here, but it could be very helpful when finally uploading the CouchApp to a CouchDB. It may look like this:

```
{
    "env" : {
        "testDB" : {
            "db" : "http://localhost:5984/test-database"
        },
        "finalDB" : {
            "db" : "http://username:password@anyserver.com/final-database"
        },
        "default" : {
            "db" : "http://localhost:5984/default-database"
        }
    }
}
```

In the example above (which we *don't* enter in our file right now), there are three different databases specified by JSON notation in the `env` section. When uploading the CouchApp later on we don't have to specify the database connections but can simply use the terms `testDB` or `finalDB`. If no database is named when uplading the CouchApp, the one defined by the value `default` is used.

So let's enter the following code to our `.couchapprc` file:

```
{
    "env" : {
        "default" : {
            "db" : "http://localhost:5984/enotes"
        }
    }
}
```

By this, our default database named `enotes` is defined to be on our local machine (`localhost:5894`). There are more options that can be included in this file which we will ignore for the time being.

2.1.4 couchapp.json

This file contains basic information about our project.

```
{
    "name": "Basic CouchApp",
    "description": "CouchApp with changes feed and form support."
}
```

Change it to your liking.

2.1.5 _id

Most of the files we edit will form the design document of our CouchApp. Every document in CouchApp needs an `_id` field. Design documents are identified by an id with the value `_design/` followed by it's name. Hereby, this is also the default value for this file:

```
_design/enotes
```

2.1.6 language

This defines the pragramming language which is used for this project. Since we will only resort to `javascript`, we leave the default entry as it is:

```
javascript
```

2.1.7 README.md

This is some information about what CouchApp is and how to access it.

2.1.8 ./_attachments

Here, files that define the layout, interface and basic functionality of our application are stored. Files ald folders located within its directory are included as attachments to the design document. I want to draw your attention to the file `index.html`. It will contain our interface and is the file that is called when starting our CouchApp.

2.1.9 ./lists ./shows ./updates

These folders are not used by our CouchApp.

2.1.10 ./vendor

Files stored in this directory are also attached to the CouchDB design document. Usually, libraries from sources other than our manually created ones are put here, but also functions connecting the CouchApp programming interface to other libraries are located here.

2.1.11 ./views

All data views that are necessary are stored in this directory. Each sub folder describes one view and contains at least a `map.js` file, but most of the time also a `reduce.js`. (the latter one is not yet present in our example) As you might have guessed, these two files correspond to the *map* and *reduce* functions used by CouchDb.

2.2 Upload the CouchApp

Without any further canges, we could upload this example CouchApp to our CouchDB. For this, we would use the couchapp program again.

```
enotes$ couchapp push enotes
```

This command uses the option `push` to initiate the upload to the database named `enotes`. Note that this database will be created automatically.

If you remember, we have defined a database connection in the `.couchapprc` file. So we could also just enter

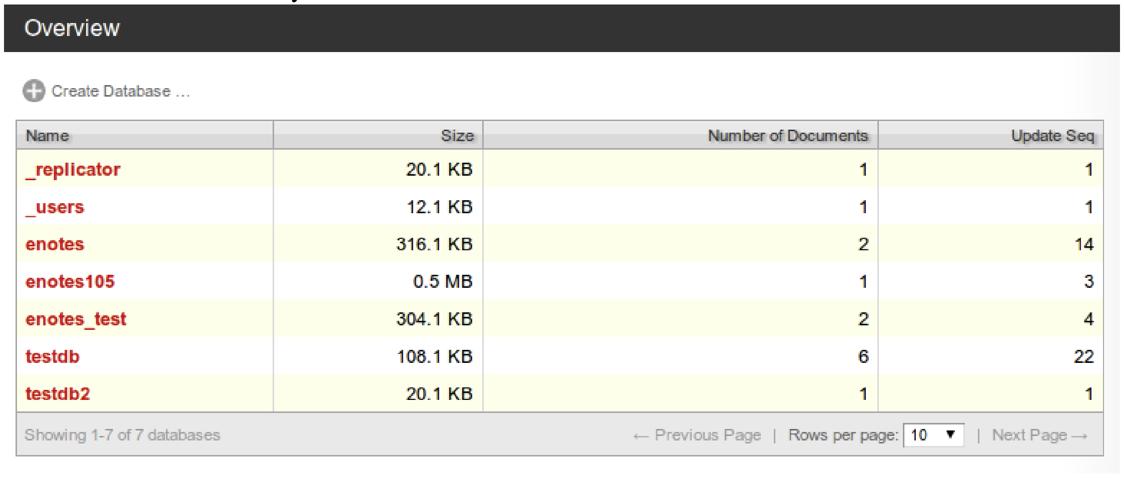
```
enotes$ couchapp push
```

2.3 Review the _design document

To gain a better understanding of the principles behind CouchApp, let's take a look at the database created by our upload. To do so, we enter the CouchDB gui via a web browser:

```
http://localhost:5984/_utils
```

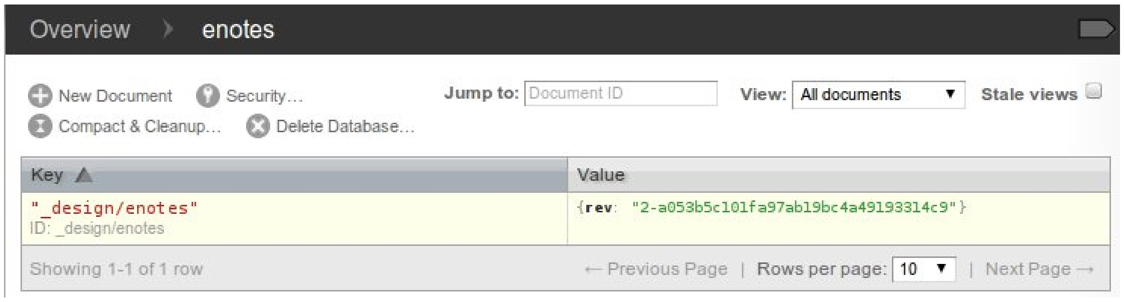
You should see a list of all your CouchDB databases of which one should be named *"enotes"*.



Name	Size	Number of Documents	Update Seq
_replicator	20.1 KB	1	1
_users	12.1 KB	1	1
enotes	316.1 KB	2	14
enotes105	0.5 MB	1	3
enotes_test	304.1 KB	2	4
testdb	108.1 KB	6	22
testdb2	20.1 KB	1	1

Showing 1-7 of 7 databases

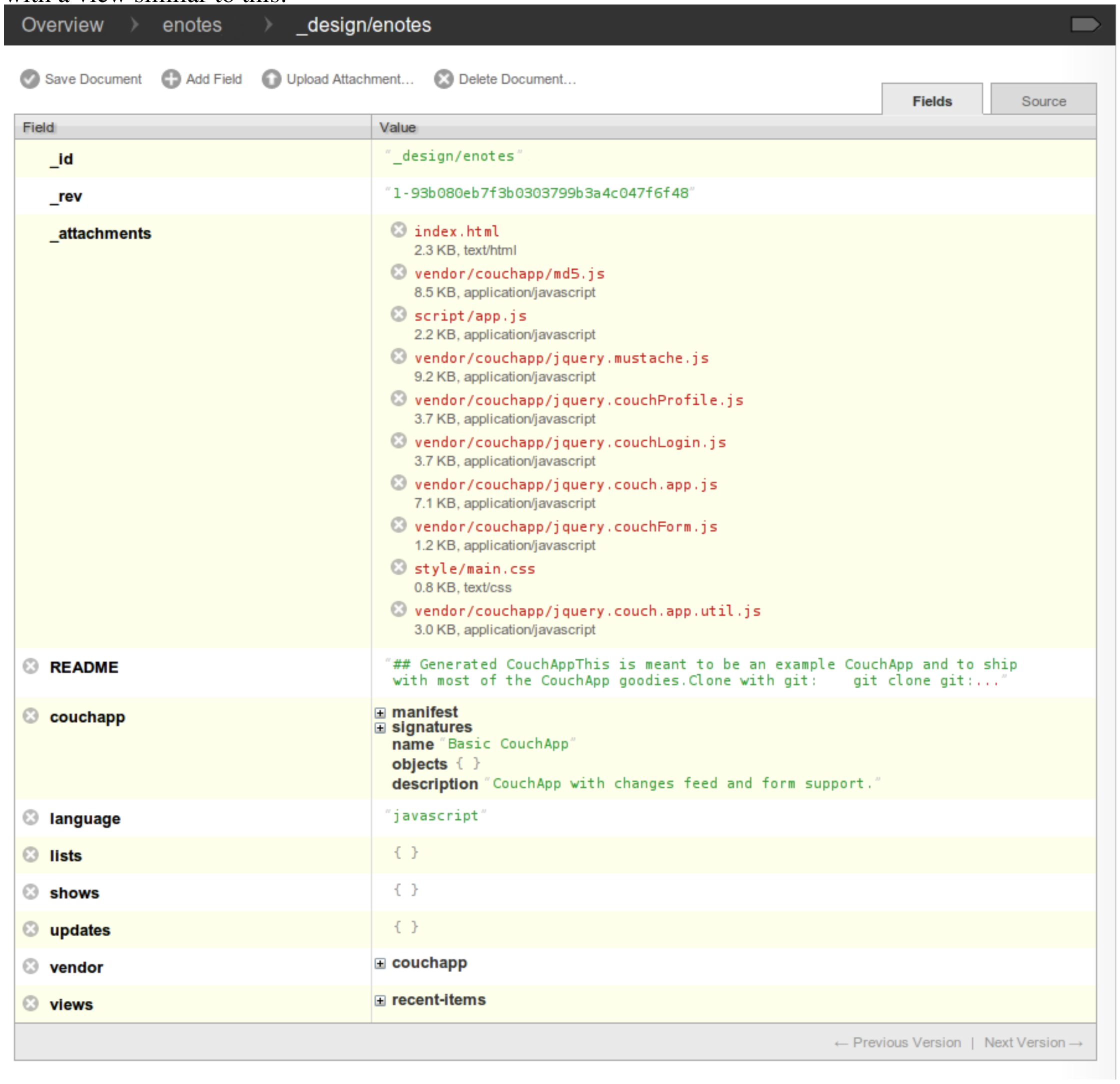
Klick on it and you will see all documents that are currently stored inside this database (which contains our enotes application).



Key	Value
"_design/enotes" ID: _design/enotes	{rev: "2-a053b5c101fa97ab19bc4a49193314c9"}

Showing 1-1 of 1 row ← Previous Page | Rows per page: 10 | Next Page →

Don't be surprised, there is only one document present. This is the design document. When adding content later on, it will also be stored as additional documents at this very location. Open the design document. You should be presented with a view similar to this:



Field	Value
_id	"_design/enotes"
_rev	"1-93b080eb7f3b0303799b3a4c047f6f48"
_attachments	index.html 2.3 KB, text/html vendor/couchapp/md5.js 8.5 KB, application/javascript script/app.js 2.2 KB, application/javascript vendor/couchapp/jquery.mustache.js 9.2 KB, application/javascript vendor/couchapp/jquery.couchProfile.js 3.7 KB, application/javascript vendor/couchapp/jquery.couchLogin.js 3.7 KB, application/javascript vendor/couchapp/jquery.couch.app.js 7.1 KB, application/javascript vendor/couchapp/jquery.couchForm.js 1.2 KB, application/javascript style/main.css 0.8 KB, text/css vendor/couchapp/jquery.couch.app.util.js 3.0 KB, application/javascript
README	"## Generated CouchAppThis is meant to be an example CouchApp and to ship with most of the CouchApp goodies.Clone with git: git clone git:..."
couchapp	manifest signatures name "Basic CouchApp" objects { } description "CouchApp with changes feed and form support."
language	"javascript"
lists	{ }
shows	{ }
updates	{ }
vendor	couchapp
views	recent-items

← Previous Version | Next Version →

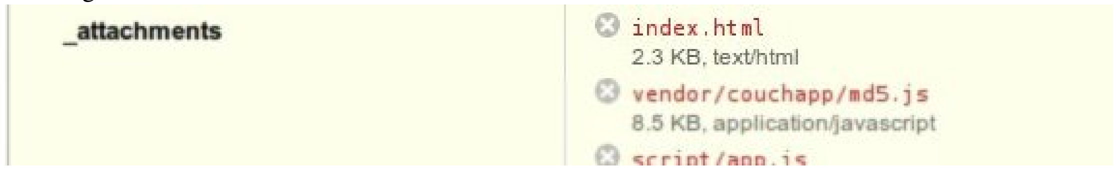
Take a look at the `Field` column. Most of these entries represent files or folders from our CouchApp directory tree. The files have been converted to entries in the design document of our CouchApp. Especially interesting is the `_attachments` entry. These are attached files that can be accessed directly by their URL (but are stored inside the CouchDB). The other fields are using JSON for formatting and storing information (functions, tags, ...)

2.4 Test the CouchApp

To start our CouchApp, we simply have to access the CouchDB. As we know, this is done via a simple HTTP command.:

```
http://localhost:5984/enotes/_design/enotes/index.html
```

This is exaclty the same link as the one accessed when clicking on `index.html` in the `_attachments` entry in our design document.



_attachments

index.html
2.3 KB, text/html
vendor/couchapp/md5.js
8.5 KB, application/javascript
script/app.js

Let's take a closer look at this url. It is quite clear that `http://localhost:5984` points to the CouchDB server on our local machine. The following `/enotes` describes the database where our CouchApp is stored. Appending `/_design/enotes` tells the server to open the design document with the Id *_design/enotes*. The final string `/index.html` accesses the appended *index.html* and therefore starts our CouchApp.

CHAPTER

THREE

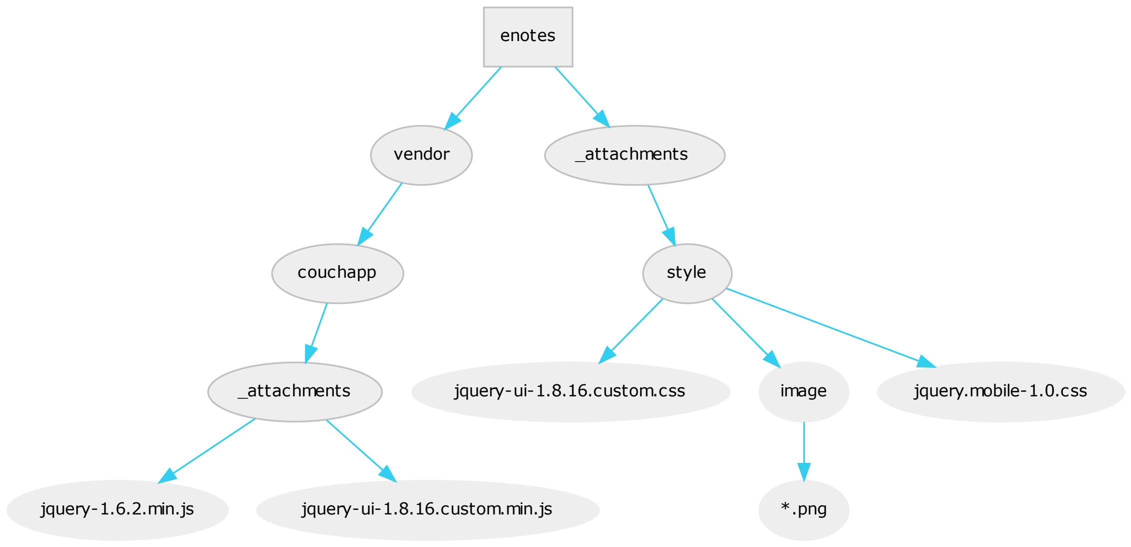
WRITE THE INTERFACE

As said previously, the interface layout is defined by HTML and CSS. This HTML and CSS are stored as attachments in the design document of our application. When we want to execute our CouchApp, we simply call the HTML. We can access attachments just like normal web-pages via its URL.

So, let's just open the file `index.html` in the folder `./_attachments` for editing.

It is already filled with code which we will remove completely. So we will be left with an empty file. Now it's time to start filling in our own code!

The following graphic gives an overview of the files we have to download and add to our CouchApp:



Elements not encircled by a gray line are files and folders we will create in the next sections.

3.1 HTML Structure & <head>

First, we enter a basic HTML structure. In the `<head>` section, we will fill in **meta-information** and the page title. Additionally we enter references to **two css stylesheets** we will download afterwards:

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width,minimum-scale=1,maximum-scale=1">
        <title>eNotes</title>

        <link rel="stylesheet" href="style/jquery.mobile-1.0.css" type="text/css"/>
        <link rel="stylesheet" href="style/jquery-ui-1.8.16.custom.css" type="text/css"/>
    </head>
    <body>

    </body>
</html>
```

The "meta-information" consists of the `charset` used to encode the html and the definition of the `viewport`. The viewport variable is important for mobile browsers so that they can resize our layout accordingly.

3.2 CSS

We have to download the two css stylesheet files we just referenced to and move them to their appropriate location in our CouchApp directory tree.

3.2.1 jQuery UI CSS

- You can get the **jQuery UI CSS** at http://jqueryui.com/download/jquery-ui-1.8.16.custom.zip .
- Open the downloaded archive and go to `/css/smoothness/`. Here you will see the file `jquery-ui-1.8.16.custom.css` and the directory `/image`. Extract both of them to the `/enotes/_attachments/style/` folder.
- In the archive at `/js/` you will find two *.js* files. These need to be extracted to `/enotes/vendor/couchapp/_attachments/`.

3.2.2 jQuery Mobile CSS

Download the file http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.css to `/enotes/_attachments/style/`.

3.3 JavaScript Library

While we are at it, we may already download **jQuery mobile**. You can find it at http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js . Place it at `/enotes/vendor/couchapp/_attachments/`.

3.4 HTML <body>

Now, we will add some actually visible content to our `index.html`. We will use jQuery Mobile and its specialized functions a lot because this CouchApp should be usable with mobile devices.

3.4.1 Pages

To speed up loading times when switching between different "windows" of our CouchApp, we will define *subpages* within our HTML document. So, the complete application layout is already loaded when `index.html` is called. *Subpages* are a speciality of jQuery Mobile and are defined by adding the attribute `data-role="page"` to a div-element. These pages can be linked to by a `href = #idOfThePage`. For more information on pages, take a look at http://jquerymobile.com/test/docs/pages/page-anatomy.html.

3.4.2 "Tags" Window

Let's add our first application window - a view that will list all tags applied to any posts. This is done by defining a *page* inside the `<body>` of our html file (we will use comments to make the start and end of a page more visual):

```
...
...
<body>
    <!-- ====== tagListPage =====  -->
    <div data-role="page" data-theme="b" id="tagListPage">

    </div>
    <!-- tagListPage -->
</body>
...
...
```

Let's take a look at the entry that defines our page `<div data-role="page" data-theme="b" id="tagListPage">`. As previously said, `data-role="page"` defines a new page. By specifying `data-theme="b"` we select the jQuery theme named *"b"* for our page. This mainly defines the colour scheme (for more information on this tag, take a look at http://jquerymobile.com/demos/1.0/docs/pages/pages-themes.html. At last, `id="tagListPage"` gives our page a name.

Now, we need to add some content to the "tagListPage". Actually, we could just enter some html, but jQuery mobile gives us the possibility to define a *header*, the *content* and a *footer*. This is done again by a `<div>` element with e.g. the attribute `data-role="header"`. We should add these three sections to our newly created page:

```
...
...
<!-- ====== tagListPage =====  -->
<div data-role="page" data-theme="b"id="tagListPage">

    <div data-role="header" data-position="fixed">

    </div>
    <div data-role="content" id="tagListContent" >

    </div>
    <div data-role="footer" id="tagListFooter" data-position="fixed">

    </div>

</div>
<!-- tagListPage -->
...
...
```

For the time being, we just need to enter actual content to the `header` and `footer`. Data displayed in the `content` section will be generated later programmatically.

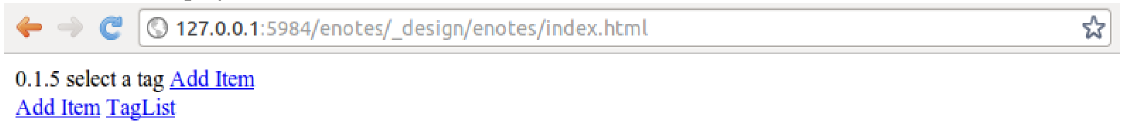
For the data-role **header**, fill in:

```
<div data-role="controlgroup" data-type="horizontal">
    <a>0.1.5 select a tag</a>
    <a href="#addPage" data-transition="slideup" data-role="button"  >Add Item</a>
</div>
```

The data-role **footer** has to be filled with:

```
<div data-role="controlgroup" data-type="horizontal">
    <a href="#addPage" data-transition="slideup" data-role="button" >Add Item</a>
    <a href="#tagListPage" data-transition="slideup" data-role="button" >TagList</a>
</div>
```

If we would push our CouchApp directory tree to our CouchDB right now with the command `enotes$ couchapp push enotes`, we wold generate a CouchApp with the changes we have made so far. Then, when we would open the link `http://localhost:5984/enotes/_design/enotes/index.html` in our browser, an image like this would be displayed:



0.1.5 select a tag Add Item
Add Item TagList

Here we can see exactly the content of our `index.html`. Since we have not added any styling information or program logic, all these are displayed as plain text or hyperlinks. Nothing would happen yet, if we were to click on those links.

3.4.3 "List" Window

When we click on any tag listed, we want to be presented with all notes that are tagged with this specific tag on a new page. This "list" window also has to be described in our `index.html`. Add the following lines:

```
...
...
<!-- ====== titleListPage =====  -->
<div data-role="page" data-theme="b"id="titleListPage">

    <div data-role="header" data-position="fixed">
        <h1>select a note</h1>
        <a href="#addPage" data-transition="slideup" >Add Item</a>
        <a href="#tagListPage" data-icon="grid" >TagList</a>
    </div>
    <div data-role="content" id="titleListContent" >

    </div>
    <div data-role="footer" data-position="fixed">
        <a href="#addPage" data-transition="slideup" >  Add Item  </a>
        <a href="#tagListPage" data-icon="grid" >  TagList   </a>
    </div>

</div>
<!-- titleListPage -->
...
...
```

This defines the layout of the page displaying a list of notes.

3.4.4 "Add" Window

Now, we want to add an additional page to our `index.html`. Add the following code inside the `<body>`:

```
...
...
<!-- ====== add ===== -->
<div data-role="page" data-theme="b" id="addPage">

    <div data-role="header"  >
        <p>add a single item - addPage</p>
        <a href="#tagListPage" data-icon="grid" class="ui-btn-right">tag list</a>
    </div>

    <div data-role="content" id="addContent" >
        <a>add content in index.html </a>
    </div>

    <div data-role="footer" id="addFooter" >
        <div data-role="controlgroup" data-type="horizontal">
        <a href="#tagListPage" id="addCancelButton" data-role="button" data-theme="d">Cancel and go t
        <input type="submit" value="add new - Submit" data-role="button" data-theme="a">
        </div>
    </div>

</div>
<!-- add -->
...
...
```

This page is shown when we want to add a new note to our application. When you take a look at the code for the other pages, you will find a link to `#addPage`. This link refers to this very `id="addPage`.

3.4.5 "Edit" Window

To view or edit a note, we also need a special window. Add this code to `index.html`:

```
...
...
<!-- ====== show and edit ===== -->
<div data-role="page" data-theme="b" id="editPage">
    <!--    <form id="editNote"> -->
    <div data-role="header"  >
        <p>edit a single item - editpage</p>
        <a href="#tagListPage" data-icon="grid" class="ui-btn-right">tag list</a>
    </div>
    <div data-role="content" id="editContent" >
        <a>edit content in index.html </a>
    </div>
    <div data-role="footer" id="editFooter"  >
        <div data-role="controlgroup" data-type="horizontal">
            <a op="delete" href="#titleListPage" data-role="button" data-theme="b" >DELETE Note</a>
            <a href="#titleListPage" data-role="button" data-theme="d" >Select other Note</a>
            <a href="#addPage" data-role="button" data-theme="d" >Add Item</a>
            <a href="#tagListPage" data-role="button" data-theme="d" >TagList</a>
            <a op="save" href="#titleListPage"  data-role="button" data-theme="a" >SAVE Note</a>
        </div>
    </div>
</div>
```

```
<!-- edit -->
...
...
```

This page is not very different from the others because the actual content within the `id=editContent` will be added programmatically.

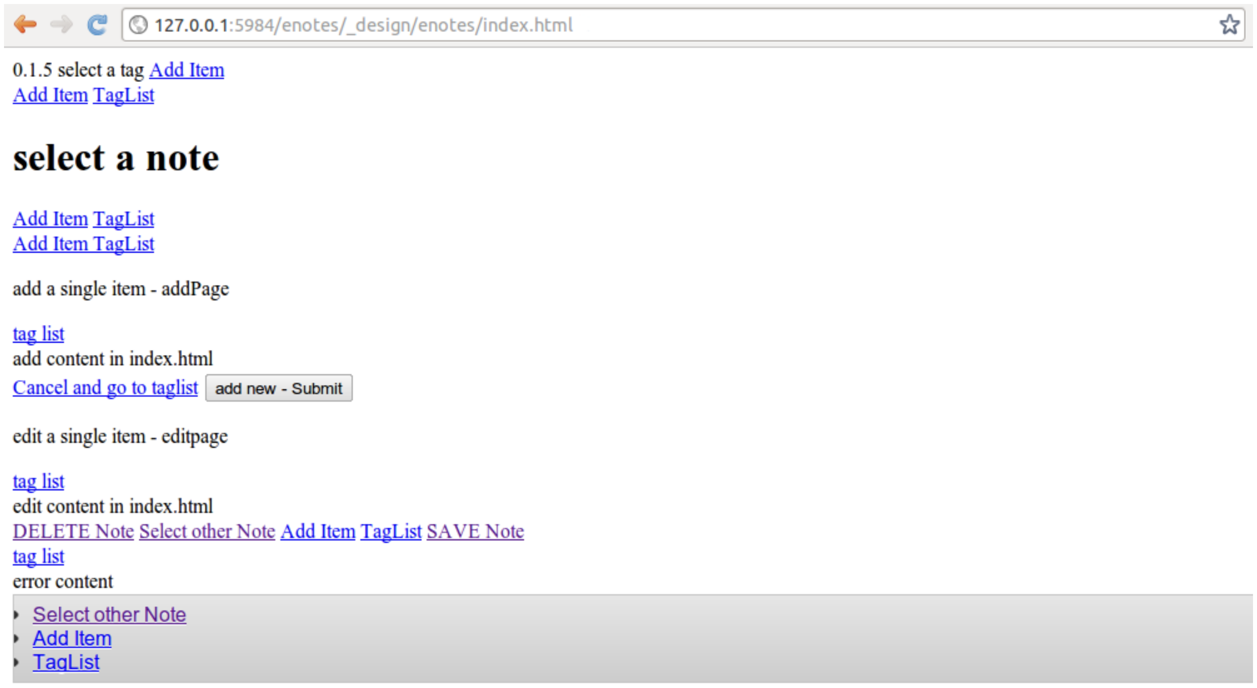
3.4.6 "Error" Window

The last page we will add is a special window for error messages:

```
...
...
<!-- ====== error =====  -->
<div data-role="page" data-theme="b" id="errorPage">
    <div data-role="header" id="errorHeader" data-nobackbtn="true">
        <a href="#tagListPage" data-icon="grid" class="ui-btn-right">tag list</a>
    </div>
    <div data-role="content" id="errorContent" >
        error content
    </div>
    <div data-role="footer" id="errorFooter"  >
        <div class="ui-body ui-body-b">
            <ul class="ui-block-b">
                <li> <a href="#titleListPage"  data-role="button" data-theme="d">  Select other Note
                <li> <a href="#addPage"  data-role="button" data-theme="d">   Add Item   </a> </li>
                <li> <a href="#tagListPage"  data-role="button" data-theme="d">   TagList    </a> </li
            </ul>
        </div>
    </div>
</div>
<!-- error -->
...
...
```

3.5 A Look at our CouchApp

If you want to take a look at our CouchApp so far, you have to export the dictionary tree of the CouchApp to CouchDB with: `$enotes couchapp push enotes`. After that, open our application in any web browser with the url `http://127.0.0.1:5984/enotes/_design/enotes/index.html`. You should see the following page:


127.0.0.1:5984/enotes/_design/enotes/index.html

0.1.5 select a tag Add Item
Add Item TagList

select a note

Add Item TagList
Add Item TagList

add a single item - addPage

tag list
add content in index.html
Cancel and go to taglist add new - Submit

edit a single item - editpage

tag list
edit content in index.html
DELETE Note Select other Note Add Item TagList SAVE Note
tag list
error content

- Select other Note
- Add Item
- TagList

Since we have not implemented any jQuery or CSS, the content of `index.html` is displayed as it is. Each sub page we have entered is rendered at once. This will change with the next chapter.

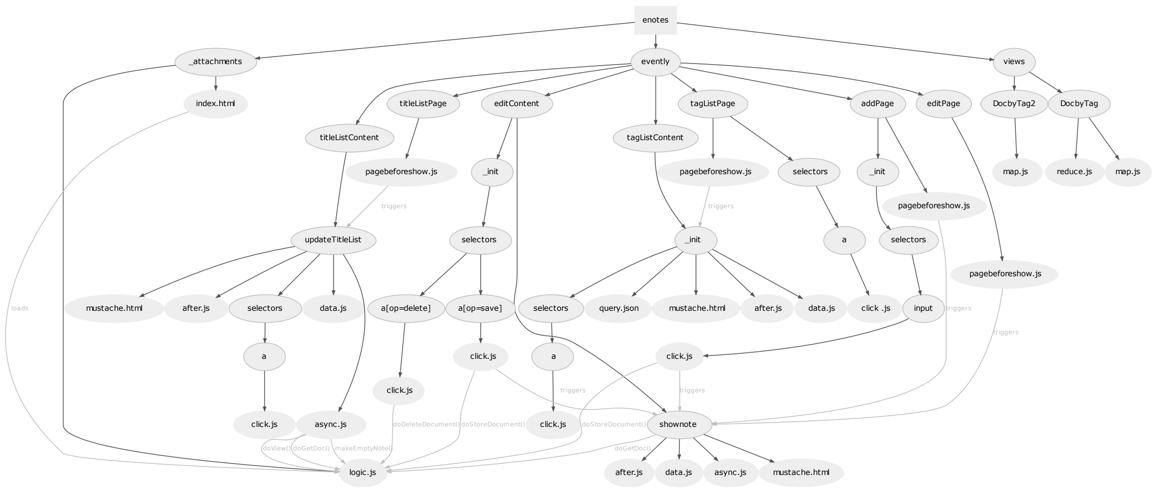
CHAPTER
FOUR

PROGRAM LOGIC

We just have provided enought code to define a simple interface which does not do anything but display static words right now. To change this, we need to enter some JavaScript code.

4.1 Overview

To give you an overview of the inner workings of our CouchApp and as a reference, you may refer to the following diagram. Dotted lines represent function calls and normal ones folder relations. Only folders relevant for us are represented, so if you miss e.g. the folders `lists`, `shows`, `updates` or `vendor` don't wonder.



4.2 StartUp Code

First, we need some code that initializes our application. We stay at our HTML file and add the following lines to the end of the file (but still within the `<html>`)

```
...
...
</body>
<script src="vendor/couchapp/loader.js"></script>
```

```
    <script src="logic.js"></script>
</html>
```

This code will execute two java scripts after loading the HTML page. One stored at vendor/couchapp/loader.js and the other one just called logic.js. Since we refer to these files, we will create them now.

4.2.1 loader.js

This file is located inside the vendor directory of our CouchApp tree. So, let's navigate to this folder and enter the subfolder called couchapp. Here you see an additional directory called _attachments. This one holds all files which are attached to the design document of our CouchApp under the context of "vendor". Its exactly the same as the default "_attachments" directory at our base with the difference that these files are uploaded within the "vendors" field. We create the empty file loader.js at this location and open it with a text editor.

Enter the following function to the file:

```
function couchapp_load(scripts) {
    for (var i=0; i < scripts.length; i++) {
        document.write('<script src="'+scripts[i]+'"><\/script>')
    };
};
```

This function takes an array as argument (scripts) which holds a list of libraries. These libraries then are formatted to represent HTML calls and are injected into the HTML from where this function was called (in your case, into index.html). Thereby, these JavaScript libraries are loaded themselves.

But for the function to do anything it must not only be defined, but also be called. So, let's call it. After the function definition, insert:

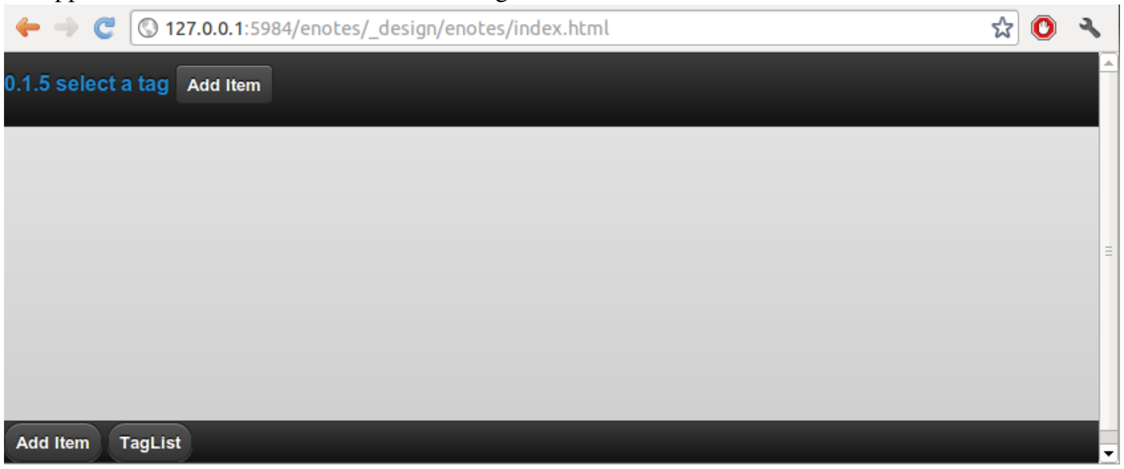
```
...
...
couchapp_load([
    "/_utils/script/sha1.js",
    "/_utils/script/json2.js",
    "/_utils/script/jquery.js",

    "vendor/couchapp/jquery-1.6.2.min.js",

    "vendor/couchapp/jquery.couch.js",
    "vendor/couchapp/jquery.couch.app.js",
    "vendor/couchapp/jquery.couch.app.util.js",

    "vendor/couchapp/jquery.pathbinder.js",
    "vendor/couchapp/jquery.mustache.js",
    "vendor/couchapp/jquery.evently.js",

    "vendor/couchapp/jquery-ui-1.8.11.custom.min.js",
    "vendor/couchapp/jquery.mobile-1.0.min.js"
]);
```

This last code consists of the function call couchapp_load and an array of libraries to be loaded. So, when the file loader.js is opened from our HTML file, exactly this happens. The order in which these libraries are loaded is important since some of them depend on each other and cannot be loaded properly if the files they depend on have not already been loaded.

Let's review these libraries and copy the ones still missing to vendor/couchapp/_attachments/:

file	explanation	where to get
`/_utils/*.*`	They refer to files already available within CouchDB. Remember, the path `/_utils/` points to the CouchDB user interface when used within a web browser.	already present
`jquery.couch.js`	Comes with every CouchDB installation and offers a JavaScript interface for the CouchApp.	find it with a file search tool on your system
`jquery-ui*.min.js`	The actual name of this file depends on the version of CouchDB you have installed. You have to change the entry in `loader.js` accordingly!	find it with a file search tool on your system
`jquery-1.6.2.min.js`	This is the actual jQuery library.	`Download a working copy here` or go to http://docs.jquery.com/Downloading_jQuery
`jquery.mobile-1.0b1.min.js`	The mobile version of jQuery. A newer version than the one proposed here may also work for you.	`Download here` or find it at http://jquerymobile.com/download/
`jquery.pathbinder.js`	We need this file to use evently functions stored in the CouchApp directory tree.	`Download here`
`jquery.evently.js`	Evently bindings	`Download here`
other files	All other files are already present.	

If we export our CouchApp to our CouchDB again (`enotes$ couchapp push enotes`) and open the exported CouchApp (`http://localhost:5984/enotes/_design/enotes/index.html`), we will see that the appearance of the `index.html` has changed.



This change has happened solely because of the reference to the additional JavaScript libraries. The reason the graphics have been added is due to jQuery. In our `index.html` we have already set certain attributes that tell jQuery how to design the elements (like for example hyperlinks as buttons).

If you click on the button in the lower right labeled "Add Item", our "Add Item" subpage is shown. This happens, because the hyperlink behind this button links to `#addPage`. jQuery interpretes this link and loads the subpage with the id "addPage".

4.2.2 logic.js

Finally, the file `logic.js` handles the program logic unique to our application. Here the functions are defined or linked to, that are executed when a certain event is triggered in our HTML file.

Since it does not exist yet, we have to create it. It should be located at `/_attachments/logic.js`. Now, lets fill in some code!

```
$dbname = "enotes";
$appname= "enotes";
$db = $.couch.db($dbname);

$("body").data =    { "tagSelected" : "NOTAG"
                , "idSelected" : 0
                , "docEdited" : ""
                , "tagsUsed"  : []
                , "textSearchString" : false
                , "titleSearchString" : false
                , "tagSelected" : false
                };
```

This first part creates the variables `$dbname`, `$appname` and `$db`. As you may have guessed, the first two entries denote the name of the database we are using and the second one states the name of our CouchApp. The third variable defines our database connection. `$.couch.db($dbname)` is a method that is located within the file `/vendor/couchapp/_attachments/jquery.couch.js`. The command `$("body").data` is a jQuery specific method. It attaches any information in JSON format to a given DOM element. In our case it is the `body` element this information is attached to. This defines the initial state our CouchApp is in when it is started.

We will go on by adding the following lines to the same file:

```
$.couch.app(function (app) {
        $("#addContent").evently("editContent", app);
        $("#addPage").evently("addPage", app);
        $("#tagListPage").evently("tagListPage", app);
        $("#tagListContent").evently("tagListContent", app);
        $("#titleListPage").evently("titleListPage", app);
        $("#titleListContent").evently("titleListContent", app);
        $("#editPage").evently("editPage", app);
        $("#editContent").evently("editContent", app);
});
```

The functions we just called links certain links inside the HTML file (like `#addPage`) to evently events. At this point we only add links to eight functions. For our CouchApp to work properly, we will come back here to add some more links later on. Notice that both `#addContent` and `#editContent` link to the same element (`editContent`) since they will use the same method and layout to display information.

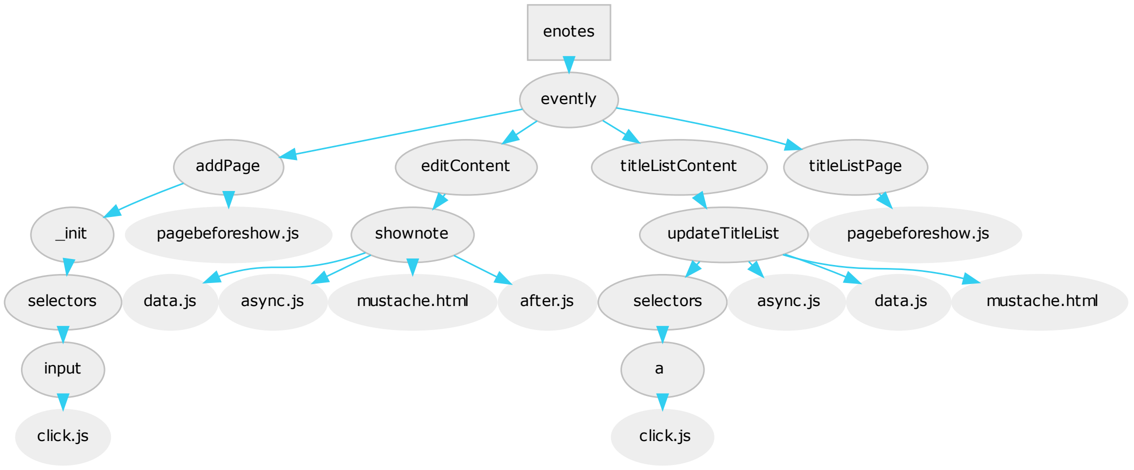
4.3 Action Code / Evently

Now we will add some code that is only executed when certain events are triggered (like clicking on a hyperlink). We will use evently to help us organise these event-based functions, but also have to add certain pieces of code to other places.

As we have encountered before, CouchApps are organised by maintaining a directory tree. This directory tree represents hierarchies. We can define functions for evently by storing these in the `evently` sub folder, we will create right now. Each function is located at a folder of its own inside the `evently` directory and replaces any already available function with the same name.

At the beginning of every chapter you will be presented with a diagram of folders and files you have to create. If you get confused during the passages of text you can refer to this graphics to reorientate.

4.3.1 addPage - Event



There has to be added a folder for our `addPage` link inside the `evently` folder. Inside this `addPage` folder we will create a file named `pagebeforeshow.js`. As the name tells us, the content within this file is executed before a specific page is shown. Fill it with the following code:

```
function (data) {
        $.log("evently/addPage/pagebeforeshow.js" + data)
        // indicates 'new note'
        $("body").data.idSelected = 0;
        $("#addContent").trigger("shownote");
        return data;
}
```

This mainly calles the function `shownote` from the element `addContent` (which is linked to an evently event named `editContent`. This does not yet exist, but we will create it soon).

Now, we have to add some more folders. Inside the `addPage` directory, we have to create a folder with the name `_init`. This stands for the init-function that is executed when the function is loaded. Here, add another folder named `selectors`. The content of this new folder describes DOM elements to which content added further on applies to. We will add a directory named `input` inside the `selectors` folder. Now, inside `input`, create a file with the name `click.js`. This directory structure will be interpreted as a jQuery function by the couchapp application. It would look like this: `$("input").click`.

At last, we can define the function that is executed when a click happens to the "input" element our directory structure referres to. We do this by adding the following code to `click.js`:

```
function() {
        $.log("addPage - shownote - click.js");

        var docvalue= $("body").data.docEdited; //edata.docEdited;
        $.log(docvalue);
        var
                title = $('input[name=title]').val(),
                text = $('textarea#editTextField').val(),
                tags = $('input[name=tags]').val().split(" ");

        $("body").data.tagSelected=tags[0].toUpperCase();
        var document = {
                "type" : "note", "TextNote" : {
                        "note" : {
                                "title": title, "text" : text, "tags" : tags
                        }
                }
        };
        //no fields for docid and docrev in 'addpage' which is new doc
        // could check docvalue._id = 0

        $.log(document);
        doStoreDocument(document); //braucht das alte doc fuer die revision?

        $.log("addPage after store");
        $.log(document);
        $("#editContent").trigger("shownote");  //possible an argument, how to use?
        //      the saving of the id comes too late, async issue
        $.mobile.changePage("#titleListPage", "slidedown", true, true);
        // id is found too late
}
```

Take a look at the line `$("#editContent").trigger("shownote");`. Here, the function `shownote` which is linked to `#editContent` is called. Since this function does not yet exist, we have to add a second event to the `evently` folder called `editContent` (remember our `logic.js` - there we assign the function `editContent` to the link `#addContent`). The actual link `#addContent` that calls the function `editContent` is located in `index.html` inside the `addPage` page. To create this function, again make a folder called `editContent` inside our `evently` folder. Within this new folder we create another directory called `shownote` - this is the name of our function. Within this directory create the following files with the following content:

- **data.js**: We can see that the content of a stored note is loaded into the variable `noteContent`:

  ```
  function (data) {
      var note;
      $.log("evently/editContent/shownote/data.js");
      $.log(data);
      $("bdoy").data.docEdited=data;
      var noteContent=data.TextNote.note;
      noteContent.taglist=data.TextNote.note.tags.join(" ");
      $.log(noteContent);
      $.log("put doc in docEdited");
      return noteContent;
  }
  ```

- **async.js**: Within this code, remember the call for the function `makeEmptyNote()` - we will add it later on.

  ```
  function(callback) {
      var idvalue=$("body").data.idSelected;
      var emptydata = makeEmptyNote();
      $.log("evently/editContent/shownote/async.js: edit content - id is found " + idvalue);
      $.log($("body").data.tagSelected);
      $.log(emptydata);
      if (idvalue != 0) {
                          doGetDoc (idvalue, callback);
                          }
      else {callback (emptydata)};
  }
  ```

- **mustache.html**:

```
<div>
    <!-- <p> mustache of editContent/_init id= {{_id}} rev= {{_rev}} </p> -->
    <a class=docidrev "docid"={{_id}}  "docrev"={{_rev}} />
    <div data-role="fieldcontain">
        <label for="title">Title:</label>
        <input id="editTitleField" name="title" type="text" data-role="none"
            value="{{title}}" >
    </div>
    <div data-role="fieldcontain">
        <label for="text">Text:</label>
        <textarea id="editTextField" name="text" cols="40" rows="8" >{{text}}
        </textarea>
    </div>
    <div data-role="fieldcontain" class="ui-widget>
        <label for="tags">Tags:</label>
        <input id="editTagsField" name="tags" type="text" data-role="none"
            value="{{taglist}}" />
    </div>
</div>
```

- **after.js**:

```
function (data) {
    function split( val ) {
        return val.split( /,\s*/ );
    }
    $.log("evently/shownote/after.js" + data)
    $.log(data);
    var thetags=$("body").data.tagsUsed;
    $.log(thetags);
    $("#editTagsField").autocomplete({source:thetags,
        max: 6,
        highlightItem: true,
        multiple: true,
        multipleSeparator: " "
        });
    return data;
}
```

Since the titleListPage is referred to in the `click.js` function of the `addPage` event, we need to add the folder `titleListPage` to the `evently` folder. There create the file `pagebeforeshow.js` and fill it with the following code:

```
    function (data) {
            $.log("evently/titleListPage/pagebeforeshow.js" + data);
            $.log(data);
            $("#titleListContent").trigger("updateTitleList");
            return data;
}
```

As we can see, the funcion `updateTitleList` from the object `titleListContent` ist called. We have to create this folder. So, inside `evently` create a directory named `titleListContent`. Within that make another one called `updateTitleList`. Inside this directory, create one named `selectors` and then one named `a`. Finally, within that one make a file called `click.js` and fill it with the following code:

```
function(event, name, pass) {
    var target = $(event.target);
    $.log("selectNote in evently/titleListContent/updateTitleList/selectors/a");
```

```
        if (target.is('a')) {
            var idval = target.attr("id");
            $("body").data.idSelected=idval;
        }
    }
```

Back in the `updateTitleList` folder (take a look at the graphic at the beginning of this section if you got lost), we have to add some files quite different from the last time:

- **data.js**: This file is the main component of events. Here, data specific action is taking place. Usually, one would refer to a CochDB view or gather data otherwise here. In our case, the autocompletion field (which itslef is defined in the file `mustache.html`) is set up.

  ```
      function (data) {
  $.log("evently/titleListContent/updateTitleList/data.js");
  var tag =  $("body").data.tagSelected;
  data.key = tag;
  return  data;

  }
  ```

- **async.js**:

  ```
  function(callback) {
      $.log("evently/titleListContent/updateTitleList/async.js" );
      var tag = $("body").data.tagSelected;
      var titleWord = $("body").data.titleSearchString;
      var textWord = $("body").data.textSearchString;
      if (tag) {
          doView ("DocbyTag2", { "key":tag}, callback);
          } else if (titleWord) {doView ("byTitleWord",
              {"reduce" :false, "key":titleWord}, callback);
          } else if (textWord) {doView ("byTextWord",
              {"reduce" : false, "key" : textWord}, callback);
      };
  }
  ```

- **mustache.html**: Here the layout of the editPage/addPage is defined. Fill it with the following code.

  ```
  <ul id="titleslist" data-role="listview">
      {{#rows}}
          <li>
          {{#value}}
          <a class="title" href="#editPage" id={{_id}} >
          {{#TextNote}}
          {{#note}}
                {{title}}
          {{/note}}
          {{/TextNote}}
           </a>
          {{/value}}
          </li>
      {{/rows}}
  </ul>
  ```

Remember, anything between `{{```and ``}}` will be replaced by the corresponding JSON value when this file is used later on.

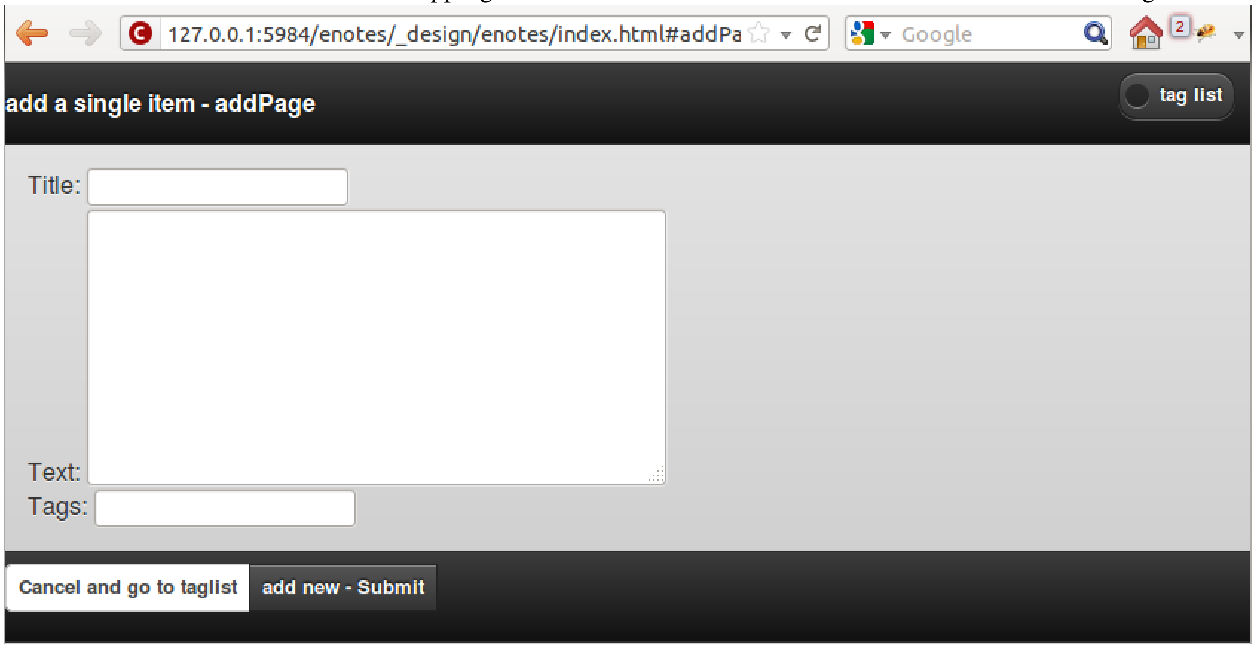
- **after.js**:

```
        function (data) {
            $.log("evently/titleListContent/updateTitleList/after.js" )
            $("#titleslist").listview();
            $.log("after listview  ");
            return data;
        }
```

We are calling a JavaScript function called `makeEmptyNote` in our code here. This does not yet exist, so we have to create it. It should be located in `logic.js` inside our `/_attachments` folder. Just open this file and add the following lines to it at the bottom:

```
function makeEmptyNote () {
        $.log("logic.js - makeEmptyNote");
        var emptynote = {"_id" :   ""
                , "_rev" : ""
                , "type" : "note"
                , "TextNote" : {"note" : {"title": ""
                        , "text" : ""
                        , "tags" : []
                }
                }
        };
        return emptynote;
};
```

If we were to take a look at our CouchApp right now and would add a new note, we would see the following:


127.0.0.1:5984/enotes/_design/enotes/index.html#addPa
Google
add a single item - addPage
tag list
Title:
Text:
Tags:
Cancel and go to taglist
add new - Submit

4.3.2 Save Notes

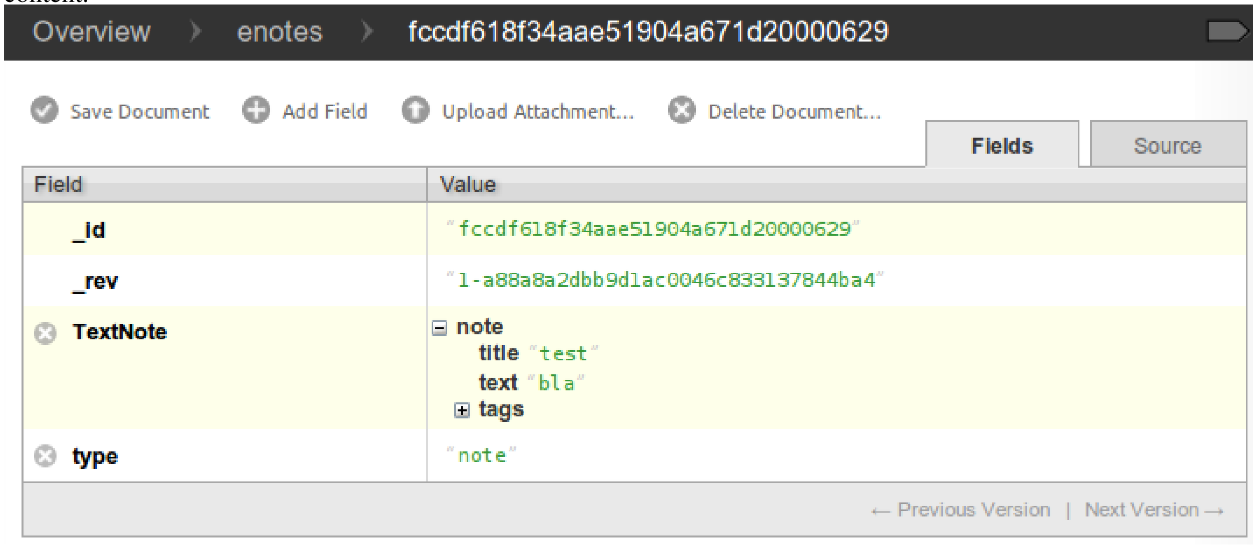
To be able to save a new note, we have to click on the "add new - Submit" button. But this does not work yet, because we are still missing code to save our entry. When the page is loaded, the code in `evently/addPage/_init/selectors/input` is attached to each input-type element. Therefore, if we click on the button, the code stored within the `click.js` file we edited earlier is fired. There is exactly one very important line in this code, namely `doStoreDocument(document);`. This is a function call of another function that still has to be included in our `logic.js`. So lets add the following code to `logic.js` inside our `/_attachments` directory:

```
function doStoreDocument(document) {
        $db.saveDoc(document, {
                async : false,
                success: function (data) {
                        $("body").data.docEdited = data.id;
        $.log("store - success" + data.id + " " +  data.rev);
                        //  $.mobile.changePage("#editPage", "slidedown", true, true);
                },
    error: function () {
        alert("Cannot save new document.");
    }
});
}
```

Here a CouchDB specific function (`$db.saveDoc(document, {`) is called on the `$db` object AJAX style to actually do the saving.

4.3.3 Review the Document

I encourage you to export our CouchApp (`¼enotes couchapp push enotes`) and open it in your web-browser. Now, create a new note inside our running CouchApp and save it. Don't be shocked if you won't see it afterwards in the list of available notes or get any error messages in a debug window - if you use any (this code is still to be implemented). What I want to show you now is the actual document that was saved into our CouchDB by clicking on "add new - Submit". In the web browser, open the page http://127.0.0.1:5984/_utils/database.html?enotes . There you will not only find our `_design/enotes` document, but also another one with random ID. Click on it to see its content.



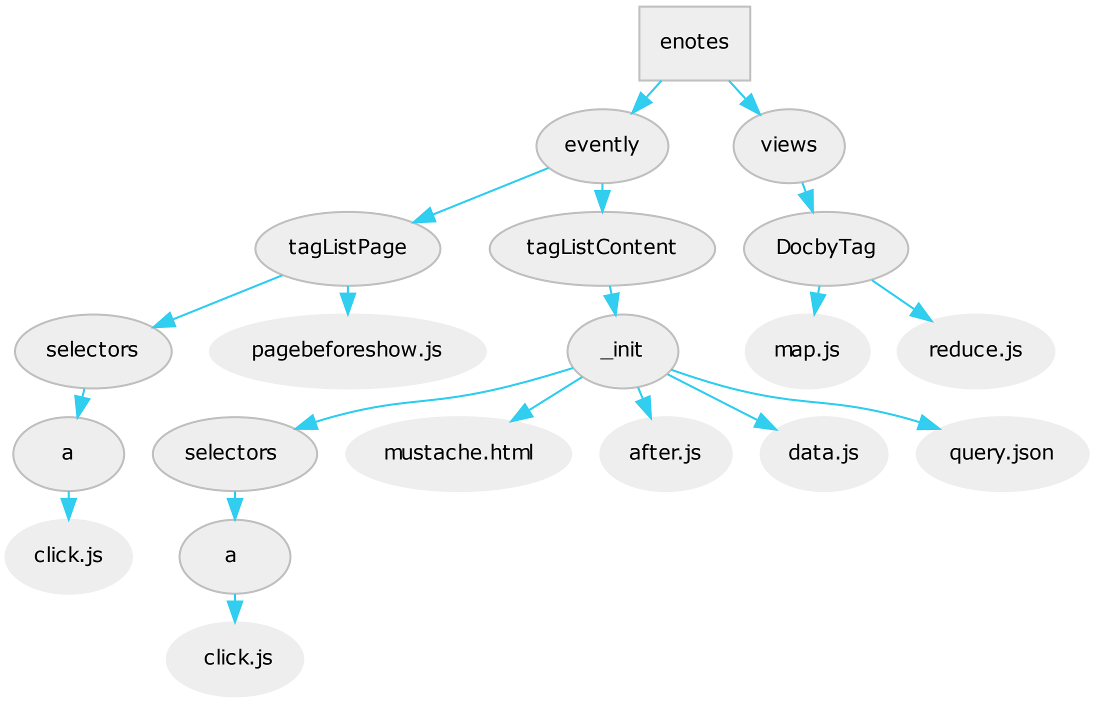
fccdf618f34aae51904a671d20000629

Field	Value
_id	"fccdf618f34aae51904a671d20000629"
_rev	"1-a88a8a2dbb9d1ac0046c833137844ba4"
TextNote	note title "test" text "bla" tags
type	"note"

As we can see, this is the document that holds our new note.

Now, we have to make it visible in our CouchApp. So far, we can create new notes, but cannot look at them.

4.3.4 List used Tags



We want the list of available notes to be refreshed every time the main page is shown. The main page is the one with `id=tagListContent` which shows all notes available grouped by tags. To accieve this, we need again some evently code and a specific CouchDB view that groups our notes according to their tags.

First, we create a new folder in our `evently` folder named `tagListPage`. Within this new folder, create a file called `pagebeforeshow.js`. This code is executed right before our "tagListPage" is shown. Insert:

```
    function (data) {
            $.log("evently/tagListPage/pagebeforeshow.js" + data)
            $.log(data);
            $("#tagListContent").trigger("_init"); // ("updateTagList" );
            return data;
}
```

As we can see, the event `_init` of the object `tagListContent` is called. We will add this function shortly. But first, create an additional directory named `selectors` and then, inside the one before, one called `a`. Within `a` create a file called `click.js`. Fill it with the following code:

```
function(event) {
        var target = $(event.currentTarget);
    //probably currentTarget needed because this is a button
        $.log("tagListPage - selectors - a -click.js ");
        $.log(event);
        $.log(target);
        var aid = target.attr("op");
        $.log("clicked - the operation is " + aid );
}
```

Now we will create the function which is called from the previous `beforepageshow.js` by, creating an additional folder inside `evently` named `tagListContent`. Now, the function `_init` is defined by creating a folder named `_init` inside the `tagListContent` folder. Here we will add the files we know from before:

- **mustache.html**: This is the actual design of one entry in the tag list (the list that groups notes by their tags)

```
<ul id="taglist" data-role="listview">
{{#rows}}
    <li>
        <a href="#titleListPage" id={{key}}>{{key}} occurs {{value}} times</a>
    </li>
{{/rows}}
</ul>
```

Take a look at `{{key}}` and `{{value}}` - later, these occurences will be replaced by the corresponding values from a CouchDB query which itself returns key - value pairs.

- **data.js**:

```
function (data) {
    $.log("evently/tagListContent/_init/data.js" )
    var thetags = data.rows.map(function(x)
        {var t=x.key;
        return t;});
    $("body").data.tagsUsed = thetags;
    // store them for later use in input gui
    return data;
}
```

- **query.json**: Here the view that will be used to query data from our CouchDB is defined.

```
{
    "view" : "DocbyTag",
    "group"  : true
}
```

We will have to create the view named `DocbyTag` later to be able to use it.

- **after.js**:

```
function (data) {
    $.log("evently/tagListContent/after.js" )
    $("#taglist").listview();
    return data;
}
```

A `selectors` folder should be created: Inside the directory `_init` create a folder called `selectors`, then, inside it, another one called `a`. Now, as before, create a file named `click.js`.

```
function(event, name, pass) {
        var target = $(event.target);
        $.log("evently/tagListContent/_init/selectors/a/click.js");
var tag = target.attr("id");
$("body").data.tagSelected = tag;
}
```

Now, we have to define the view for the query to the database. In our example, the view is called `DocbyTag`. So, we have to move to the `./views` folder and add one with the name `DocbyTag`. Inside this folder, we can create two files representing a CouchDB view: one called `map.js` and the other one called `reduce.js`. The code we have to enter to each of them is as follows:

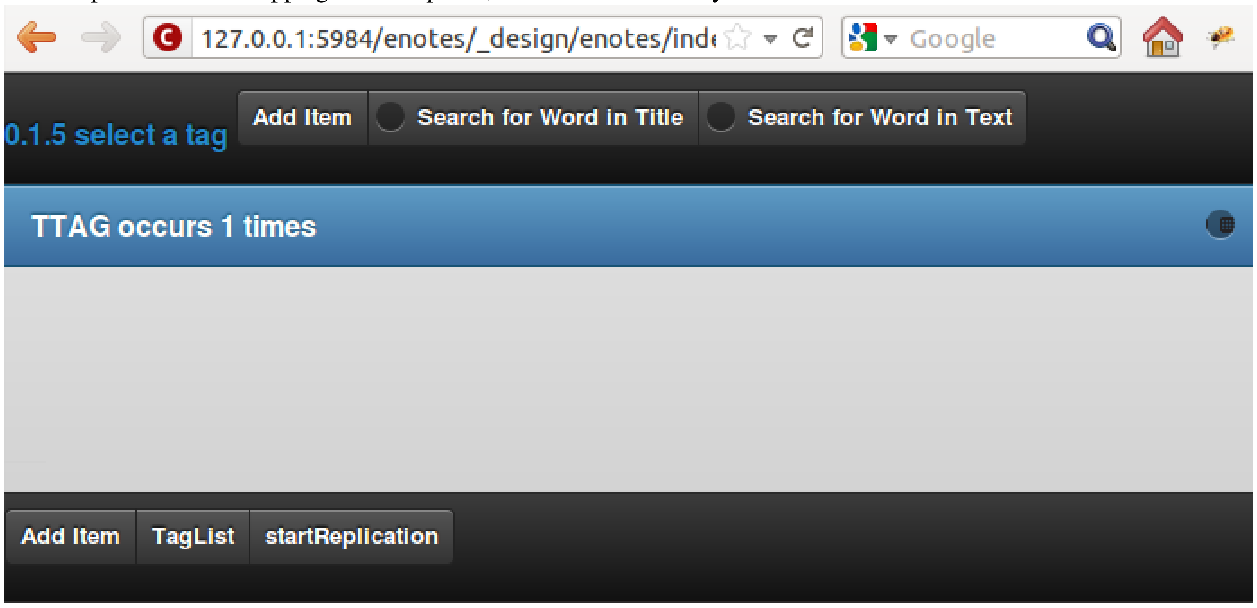
- **map.js**

```
function(doc) {
        if (doc.TextNote.note.tags) {
                var words = doc.TextNote.note.tags;
                for (var i = 0; i < (words.length); i++) {
                        var value = words[i].toUpperCase();
                        emit(value, 1);
                }
        }
}
```

- **reduce.js**

```
function(key, values) {
        return sum(values);
}
```

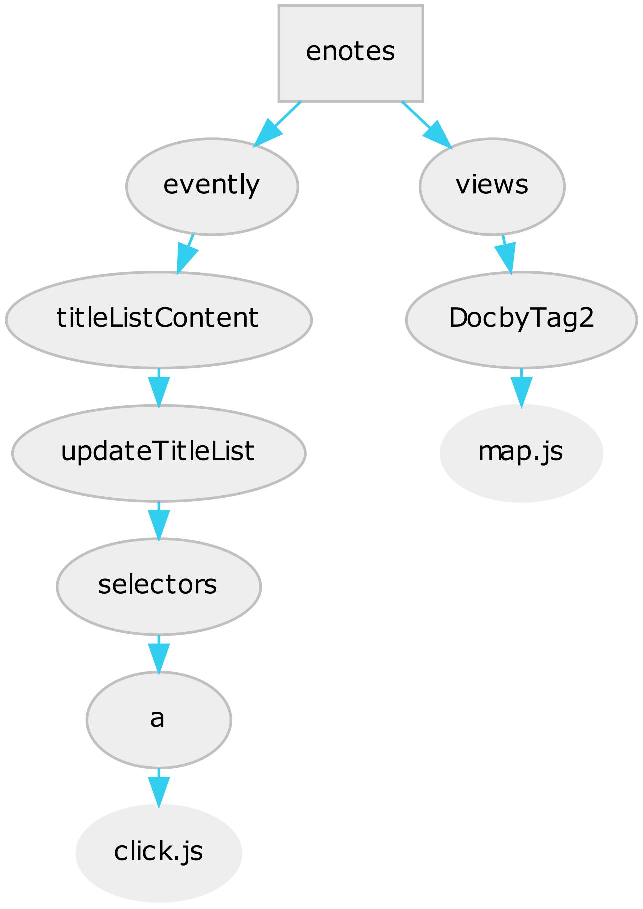
If we export our CouchApp again and open it, we will see one entry:



TTAG occurs 1 times

This is the summary of all notes in our CouchApp so far. As we have just one note inside our CouchApp, the list shows only one tag with only one occurence. But if we click on it, nothing happens yet, because we have not coded the part yet to list all notes from a specific tag.

4.3.5 List Notes from Tag



We want the list of notes to be refreshed every time the page is shown. The list itself is located inside `index.html` at the DIV with `id=titleListContent`. So, if we move to this evently event and open the file `/evently/titleListContent/updateTitleList/async.js` we can see a function call for `doView`. This function does not exist yet. We will have to add it to our already infamous‘‘logic.js‘‘ inside the `/_attachments` folder:

```
function doView (view, json, callback) {
        $.log("dbViewWithKey ");
        $.log(json);
        $db.view(($appname + "/" + view),
                XXmerge (json, {
                        async : false,
                        success: function (data) {
                                callback(data);
                        },
                        error: function () {
                                alert("Cannot find the document with id " + keyvalue);
                        }
    })
        );
}
```

This function needs another one, we will define right now, called `XXmerge`:

```
function XXmerge(o, ob) {
        for (var z in ob) {
                if (ob.hasOwnProperty(z)) {
                        o[z] = ob[z];
                }
        }
        return o;
}
```

Now we create the view `DocbyTag2` which is used to request notes of a specific tag from the CouchDB. Go to the `./views` foder and add one with the name `DocbyTag2`. Inside this new folder, create a files representing our view:

- **map.js**

  ```
  function(doc) {
      if (doc.TextNote.note.tags) {
          var words = doc.TextNote.note.tags;
          for (var i = 0; i < (words.length); i++) {
              var value = words[i].toUpperCase();
              emit(value, doc);
          }
      }
  }
  ```

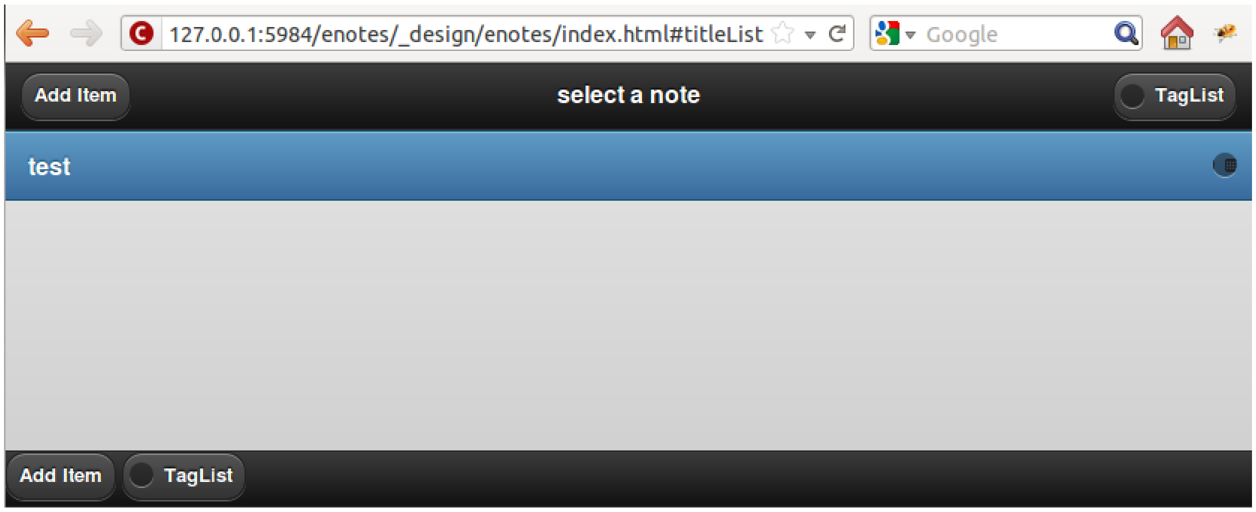
We won't need a reduce function here.

To signal the selection of an entry, we will have to add a selector for "a" to the `updateTitleList` event inside the `evently` folder: Create a new folder named `selectors` inside the `updateTitleList` folder which itself is located inside the `titleListContent` directory. In there, go on by creating another one named `a`. Inside this directory we will create a file named `click.js`. The content of this file should be:

```
function(event, name, pass) {
        var target = $(event.target);
        $.log("selectNote in evently/titleListContent/updateTitleList/selectors/a");
        if (target.is('a')) {
                var idval = target.attr("id");
                $("body").data.idSelected=idval;
        }
}
```

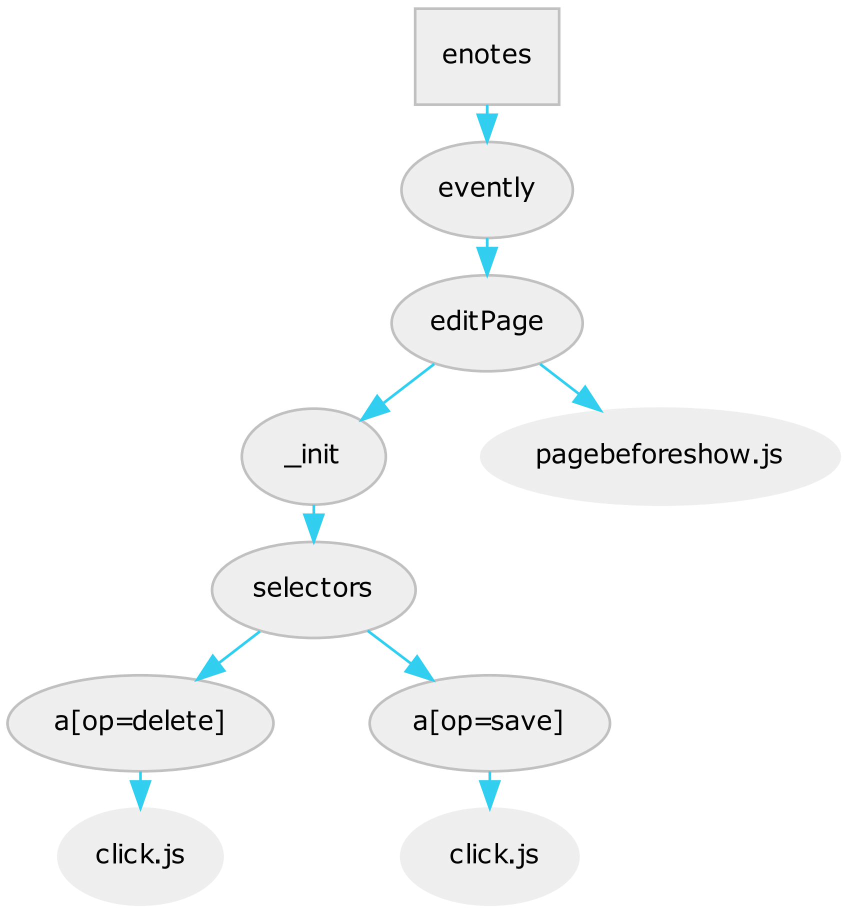
This will set the `idSelected` value stored in the `data` element of the `body` to the ID of the selected note. This is important later on when we will actually display the content of a note.

This is what you will see, if you click on any tagList item now:



Now, we will implement the parts that allow us to open and edit an existing note.

4.3.6 Open Notes



To open a note, we need to implement another evently event. Go to your `evently` folder and create a directory named `editPage`.

There you have to create the file `pagebeforeshow.js`. Its content should read as follows:

```
function (data) {
        $.log("evently/editPage/pagebeforeshow.js" + data)
        $.log(data);
        $("#editContent").trigger("shownote");
        return data;
}
```

This seems familiar. The function `shownote` in the element `editContent` is triggered. But within this `shownote` function, another one is called that does not yet exist. It has the name `doGetDoc`. We will add it to our `./_attachments/logic.js`:

```
function doGetDoc (docid, callback) {
        $.log("doGetDoc " + docid);
        $db.openDoc (docid, {
                success: function (data) {
                        callback(data);
                },
                error: function () {
                        alert("Cannot find the document with id " + keyvalue);
    }
        });
}
```

Next, we will add a "_init" method by creating a directory called `_init` within the `editPage` folder. This function is executed when *editPage* is initialized. Inside `_init` create a `selectors` directory. Here we have to add two elements to be selected, depending on whether we want to delete or save a note. Let's take a look at an excerp of the editPage inside our `index.html`:

```
...
<a op="delete" href="#titleListPage"  data-role="button" data-theme="b">DELETE Note</a>
...
<a op="save" href="#titleListPage"  data-role="button" data-theme="a">SAVE Note</a>
...
```

As we can see, there are two `a` elements, one with the attribute `op="delete"` and the other one with `op="save"`. With CouchApp and Evently it is not only possible to attach a function to the `a` element, but also distinguish them by their attributes. So, let's add a function for the delete option. Still within `selectors` create a directory named `a[op=delete]`. I guess you have noticed how to include attributes in this notation. Enter our new directory and create a file called `click.js`, because we want do define an action to be executed when we click the delete button. The content of this file should look like this:

```
function(event) {
        var target = $(event.currentTarget);
        //probably currentTarget needed because this is a button
        $.log("editPage - shownote - a - delete -click.js ")
        $.log(event);
        $.log(target);
        var aid = target.attr("op");
        $.log("clicked - the tag is " + aid );
        $.log("editContent/_init/selectors/a[op=delete]/click.js");
        $.log($("body").data);
        var docvalue= $("body").data.docEdited;
        $.log(docvalue);
        var docid = docvalue._id,
                docrev =docvalue._rev

        $.log("docid in delete " + docid + " rev " + docrev);
        var document = {"_id": docid, "_rev": docrev};
        $.log(document);
        doDeleteDocument (document);

        $.log("editPage after delete");
        $.log(document);
}
```

The most important part of this code is the line `doDeleteDocument (document);`. This represents a function that should be located inside `/_attachments/logic.js`. So, we will enter it there:

```
function doDeleteDocument(document) {
        $db.removeDoc(document, {
                async : false,
    success: function (data) {
        $("body").data.docEdited = data.id;
        $.log("document deleted - success" + data.id + " " +  data.rev);
    },
    error: function () {
        alert("Cannot delete document.");
    }
});
}
```

Next, we will include functionality to edit an existing note. As with the "delete" option, we need to create a folder inside `./evently/editPage/_init/selectors`. Only this time, we will name it `a[op=save]`. Inside this folder, also add a `click.js` file and fill it with this code:

```
function(event) {
        var target = $(event.currentTarget);
        //probably currentTarget needed because this is a button
        $.log("editPage - shownote - a save -click.js ")
        $.log(event);
        $.log(target);
        var aid = target.attr("op");
        $.log("clicked - the tag is " + aid );

        $.log("editContent/_init/selectors/a[op=save]/click.js");
        $.log($("body").data);  // why is this not availalbe here?
        var docvalue= $("body").data.docEdited; //edata.docEdited;
        $.log(docvalue);
        var docid = docvalue._id,
                docrev =docvalue._rev,
                title = $('input[name=title]').val(),
                text = $('textarea#editTextField').val(),
                tags = $('input[name=tags]').val().split(" ");

        $("body").data.tagSelected=tags[0].toUpperCase();

        $.log("docid in save doc " + docid + " rev " + docrev + " title " + title);
        var document = {"_id": docid,
                "_rev": docrev,
                "type" : "note",
                "TextNote" : {"note" : {"title": title, "text" : text, "tags" : tags}}
        };
        $.log(document);
        doStoreDocument(document); //braucht alte id und rev

        $.log("editPage after store");
        $.log(document);
        $("#editContent").trigger("shownote");
        $.mobile.changePage("#editPage", "slidedown", true, true);
}
```

With this we have completed the core of our CouchApp. We can create new notes which will be displayed, grouped by their assigned tags and edit them. All of this happens inside the CouchDB and your browser.

As usual, you should export the CouchApp directory tree to your CouchDB with the command `enotes$ couchapp push enotes`. After that, access your CouchApp in your web browser at http://127.0.0.1:5984/enotes/_design/enotes/index.html.