Worksheet

Routing

In this worksheet you will be learning how to build a front-end for CouchDB using AngularJS, and routing in particular.

There are three parts in this lab. 1) You continue with what you did in the previous lab, i.e. working on CouchDB. The purpose of this section is that you build a database that is ready for later sections. 2) You develop a simple SPA using AngularJS routing. 3) You combine the first two parts and build a front-end for CouchDB.

# CouchDB on Codio

In the previous lab you used CouchDB on IrisCouch website. Now in this lab I would like you to use CouchDB on Codio.

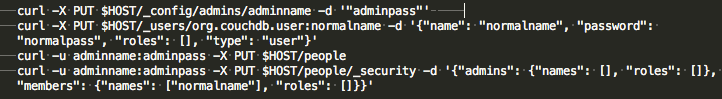
Start a new project onCodio, open terminal, and run the following commands:



you should get something similar to below, which means CouchDB is up and running.



Next, run the following commands



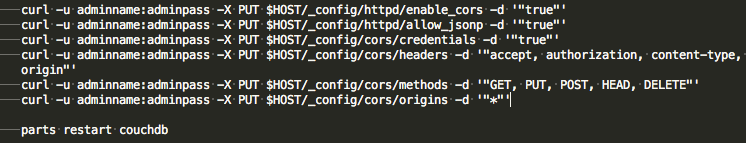
Basically the above commands create a system-wide admin account called ‘adminname’ and an ordinary user called ‘normalname’, and we also created a database called ‘people’, which only ‘normalname’ can access it.

Next, upload the file people2.json onto your project, and run the following:



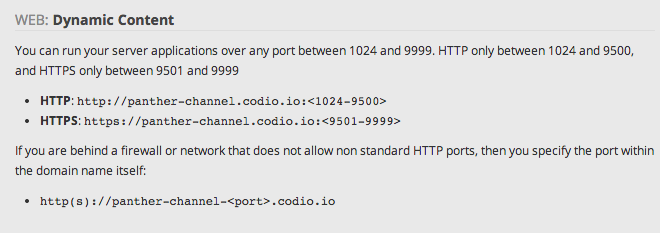
this will upload the contents of people2.json into the people database. This file and other source codes can be found on Colin’s repository.

Finally, we do some housekeeping stuff:



Most of the commands should be familiar to you already. If not, go back to the previous lab sheet and have a look again.

On Codi🡪Project, open Project Box Info, you should see something similar to below:



take a not of this address. Now open a new browser tab, put in <http://panther-channel.codio.io:5984/_utils/>, so basically your project address plus :5984/\_utils/, don’t forget replace the begging part of the url with your own specific name. You now will see something similar to below. If not, you’ll need to go back and find out why.

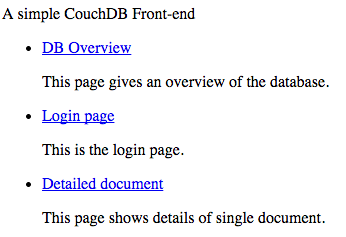


## A simple routing example

The above image gives us a rough idea of what our front-end GUI for CouchDB will be like. Now, start a new html file, and key in the contents as follows:

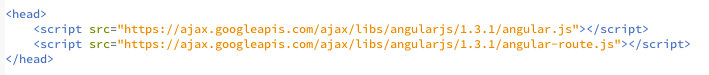


If you open it in a browser view, you’ll see something similar to this:

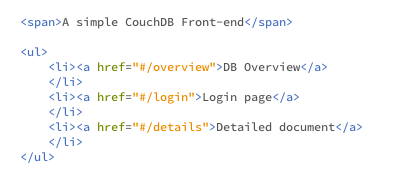


So basically our front-end will have three sections. But in an ideal case, we’ll want these to be on separate pages, or at least ‘appear’ to be on separate pages. This can be done easily using AngularJS routing.

What you need to do, first of all, is to include relevant JS files, as follows:



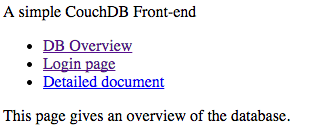
Next, re-format the main body, make it look like below:



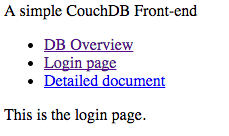
The above is easy to understand. Note the # sign in the url, this stops browser from reloading the whole page. The import part is the following:



Before we go into details, make sure you type in everything properly. If everything goes well, you will be able to click the links to navigate through the contents. A snapshot of the finished page is as follows:



and



Note here we used ‘config’ to configure a system service called $routeProvider. For the when clause we give a url and template. We also give an ‘otherwise’ clause to redirect page requests that don’t match with any of the given ones. $routeProvider is more on the controller side of MVC. On the view side, we need to have ng-view to present different templates.

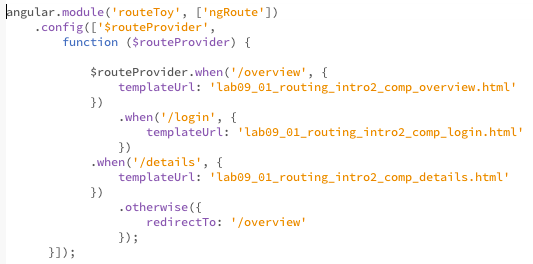
What we did here is to include all ‘templates’ and scripts within a single file. However, for larger application, we’ll need to separate those into different files and link them together. For example, if we put all ‘template’ into separate template files, and all scripts into a separate js file, our main page will look at below:



and one of the template will look like this:



and the js file will look like this:

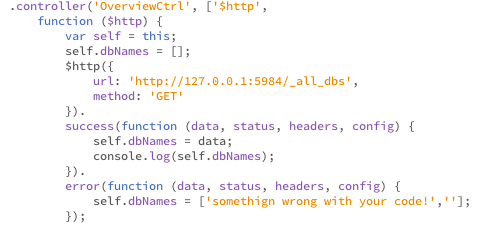


Now you should have 5 files: one for index, one for js, and three for those three templates.

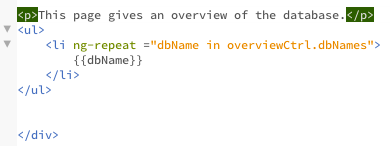
## A CouchDB front-end

Now we’re going to build a CouchDB front-end. What I did for this tutorial is that I used locally installed CouchDB. So when you see 127.0.0.1 in this tutorial you need to replace it with your own server address.

First we’ll work on the ‘overview’ part, as it doesn’t require and login credentials. Type in the following code into your js file:

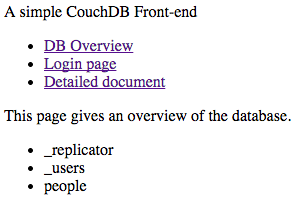


and make sure you ‘overview’ template looks like this:

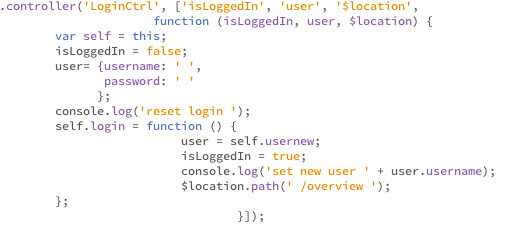


what has happened here is that we connect to the database and get all the names back, and display them in the template. If you follow along the previous labsheet, there aren’t anything new here. However, you should notice that we create a separate controller for this template.

What your application looks should be like the following:



Next, we’ll work on the login section. Make sure you add another controller to your js file, and it looks like the following:



What we have here as ‘isLoggedIn’ and ‘user’ are pre-defined values, added using ‘.value’. Note that these are ‘constant’ variables. As we go along, we’ll use services instead. But for the moment, let’s leave them as constants.



and your template is similar to this:

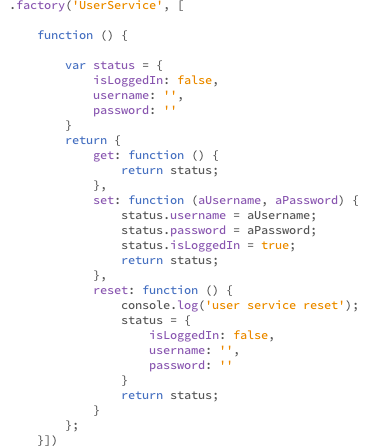


Now that once your click the submit button, you’ll be redirect to the ‘overview’ page, also your user name will be printed in the console. Note here we used $location service to redirect.

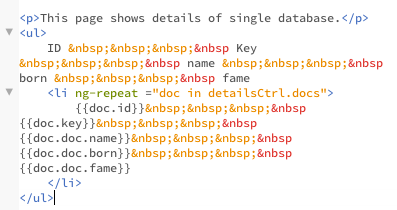
Now, we need to do the complex task, i.e. getting the detailed people database. First, you need to make your controllers look like the following:



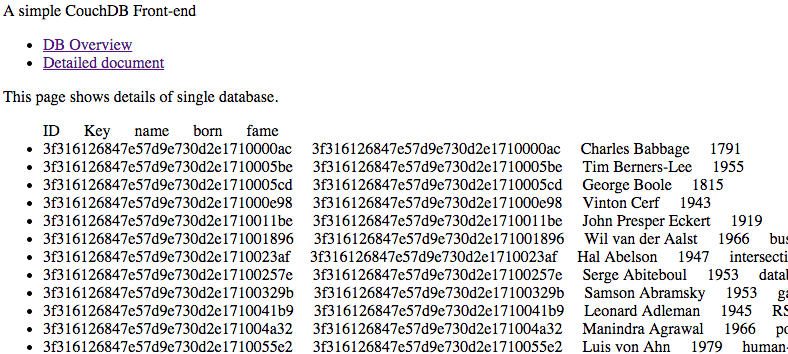
The ‘UserService’ is defined as the following. Don’t worry for now if you don’t completely understand – we’ll go through them later in the labs. The most important thing is that we download people data from the database, using authorization.



Next, we update our template to match this:



If you follow along, after logging in, you’ll see ‘detailed document’ as follows:



In the above screenshot, I hided ‘login’ link once I have logged in. Detailed implementation can be found in Colin’s repository. But I would still recommend you to type along so that you know that they are.