Basically to install zoa, you just need to do these 4 things:

* [download node.js](https://nodejs.org/en/download/)
* open a command shell
* cd zoa/src/serverHeadless
* type: node ./serverMain.js

To see it working, start up a browser (chrome) and go to the IP address that zoa server spits out when it starts up (e.g. 192.168.1.7:43770)

If that works, we are 90% of the way there. I'll need to write an example.html that reads a csv file and writes one.

In the meantime, any files you put in zoa/src/client/  will get served using the happy, headless zoa server.

Express is the communication between the client and server. GET is the method supported. A get method in the server is a function with a “request” and a “response” as its parameters. The field on the request we seem to use is request.query, There are various interesting fields on the response, especially response.send() or response.render(). This page shows all the server side cases. Unfortunately, minimal info on the client side. <http://expressjs.com/en/guide/routing.html>

Use jQuery to make a GET request.

<http://www.w3schools.com/jquery/ajax_get.asp>

To send an object as a string you can use the builtin JSON support:

ws.send(JSON.stringify(object));

To send an object as a typed array you can use a javascript BSON library such as [this one](https://github.com/schteppe/js-bson):

ws.send(BSON.serialize(object));

When you receive a WebSocket message you will need to deserialize it.

To deserialize a JSON string from a WebSocket message:

ws.onmessage = function (e) {

var object = JSON.parse(e.data);

...

};

We have an array of palettes that are available for editing. We can read them in all at once or add to the list. We can read them in one/time (and eventually from a text box, consider the text box i/o for the devmockup version)

A palette is a name, and two arrays of colors: Original and Changed. Palettes are generated from arrays of hex values.

* Both arrays and object vars are pointers, so we can simply move a pointer to edit in place. State has a current palette
* The state.colors becomes a pointer to the edited version
* There should be an easy way to create new palette from the edited version, then add it to the list.
* We should be able to combine two palettes completely, rather than using the cut buffer (good for darks vs lights)
* Display a list of all palettes, eventually with a view that includes a summary iconic view, rather than use the HTML selector.
* Compare/restore for a palette.

Palette operators

* Write XML. Uses the palette name. Needs an editable path, eventually, to support multiple users.
* Save/Restore: Reads and Writes a list of palettes and colors (need standard name, needs to use the path above. Or needs to let people name it…do we want to go there?)
* Reset Palette: Restores from original values (saved in the state, not in the file)
* Clear palette: Deletes all the colors.
* Without versions (versions would be nice someday)
  + Make Copy, duplicates the palette, adds (copy) to the name. Use rename to get a better name
  + Rename Palette: Sets the name, overwrites existing names. Can only edit text box after rename palette.
  + Need a palette selector that can be updated.

Multi-select. Ctrl click seems the most straightforward

* Patch now displays the average color
* Changing the sliders changes each color a percentage (this will take some tweaking)

Delete point, selected plus delete button, or del key (how do I do the key?)

Select multiple points and average them.

To copy colors from Cristy’s palette, need a visible cut buffer. (minimal cut buffer is the current patch, one only)

* Select point(s) then move to cut buffer.
* “Add to palette button” on the cut buffer, applies to current palette.

In the display area, would be nice to get drag to work. (or maybe drag makes a copy, which can then be moved around)

* Need to trap mouse-down, move, then mouse-up (or is there a drag?)
* Simply change the xy coordinates.
* Redraw needs to not wipe out the svgs.