DESIGN DOCUMENT for ECHO Service

# LANGUAGE

Echo is primarily a php powered application. Php was chosen because it is one of the core languages used at DRI and is the one I’m most familiar with next to html and css, even more so than javascript.

I am using PHP 7.4.2 as a future upgrade to the internal systems is planned (they currently run 5.x), but despite that I plan to keep track what 5+ features I am using, and if possible avoid it.

# DESIGN UI & UX

After investigating and researching proper ways to structure and display lists and tables, I’ve created a few rough wireframes on paper, which I then refined as a mockup in Adobe Xd.

Due to familiarity I’ve decided to use Bootstrap for the UI component styling and Font Awesome for icons. It doesn’t hurt that due to widespread use of both they will feel familiar to the users.

I’ve consciously separated filtering into its own hide-able section. This allows me to hide it, as well as make it a pop-up when the user is in the middle of a list. 🡪 Illustration & Sources required.

The interaction is as such – you filter by what you need and search for the template you want to use. Clicking on the row will bring you to the create-a-request page. From there you can choose the edit-template button.  
Or, when you hover a menu will appear on the right with an “edit” and “trash” icon. You can go into edit-template menu directly from that.   
  
On the request page, there is a list of all saved request, which you can load in. They show which version of the template they were made with.

The request page is styled after other JSON Ides, as that seemed to make the most sense. It is easily readable and the structure is easy to understand.

If possible I would like try still try and implemented the linked dbdiagram.io layout, but that’s for the last week.

# FILE STRUCTURE

I originally planned to have a tree-like file-structure of nested folders and files to keep track of and save the various templates. I would iterate over that structure and create a json file all the files which could be displayed in a table and searched through.

Though it’s still possible, having received a suggestion from David I’ve decided to look into using SQLite which turned out to be the correct decision. The [general consensus](https://www.microsoft.com/en-us/research/publication/to-blob-or-not-to-blob-large-object-storage-in-a-database-or-a-filesystem/) seems that files smaller than 256kb should be stored within a database. In addition, because the template files would likely never exceed 10kb it might be up to 35% more [efficient and faster](https://www.sqlite.org/fasterthanfs.html) to store and access them directly from an SQLite database.

I have worked out an initial database structure using dxdiagram.io 🡪 screenshot this is the initial thing.

Explain why this structure was made. There won’t be user-authentication, so no user or extra table is really necessary.

Explain the INET\_ATON trick to store and sort by version number. The sorting is less efficient than having 4 separate columns for major.minor.version.revision, but due to other filtering options, it should rarely if ever be used, and due to the relatively small amount of templates (there won’t be 10s of thousands) efficiency isn’t very important, and I think it’s neater this way.

Top of Form

# IMPLEMENTATION

## Browse-page

Describe how you started working.

So at first I made a static page with the filter section and an empty table

Then I created an sqlite database, filled it out with some fake data and implemented it into the page. ~~For now I used varchars, but it can easily be changed for nvarchars in case we need Unicode support for special characters such as ç, é, ë, è etc for names and values of keys~~

Used nvar as that is the one the json type is based on in the full SQL server which will make conversion more easy

Then I implemented the filters

* For now, the filter-by-name uses (javascript) and is client-side, as the whole list is loaded. In case I implement pagination or lazy-loading, it can/should be turned into a query, because the client wont have access to the full list.
* The tag-based filters shouldn’t have a function per button, but the button should trigger the same function that should look at the button contents (what’s the tag itself) and generate a query based on that.   
  So “dipo” will be-> SELECT \* FROM templates WHERE department = “dipo”  
  Should be a catch for the “all” tag where it’s just an \* query.   
  Maybe a case-switch, but seems unneccesary.   
  toggling All should uncheck all the other tags, and triggering a tag should uncheck all.

Then I implemented the sorting

* Again, initially it is implemented client-side using javascript, but it can be done using queries if pagination and/or endless-loading are implemented.

~~Then I tested using weird symbol in the database~~ Unicode with nvarchar

Bottom of Form

## Edit page

Even after spending quite a few hours googling and trying to find good web-based implementations, I wasn’t really able to find anything related/like it.

I’ve been thinking of which I should use for the nested layout – lists or tables.

Because I always have a key-type-value column, table initially seems like the better choise, but as it doesn’t support nesting, list suddenly starts looking like a viable options.

I suppose I should really be able to use either and make it work in different ways:

LIST: I can make nested list, and just style the elements like a table, and give them a set width so the columns inside the <li>s align properly. The actual html structure

TABLE: I can write some javascript that when it sees a nested element,

JSON TEMPLATE STRUCTURE

{

"type":{

"key1": {

“required”:true, //key always present in request.

//need to discuss it. Maybe “locked”:true instead, so you can only change the unlocked values?

"type”:”string”,

“format”:”date”,

}

},

"structure":{

"key1":"2020-03-1",

"key6": {

"key7":"changed"

},

"key8": "not\_changed",

"key10":"changed"

},

}

A request should only store the key-value pairs without a structure