Basic Entity CRUD Template

This template demonstrates how to make the front-end UI communicate with back-end functionality when it comes to the CRUD actions for the basic entities (staff, cameras, etc.)

Read: HTML Table

Example file: ***`static/staff.html`***

Creating the HTML table for the read section of CRUD actions uses the collections facility in the `HtmlRenderer` class. When loading the page, the back-end will automatically provide a collection of the relevant entities. The following code block allows you to iterate over them:

```
<table>
   <thead>
      <tr>
         <td>Field One</td>
         <td>Field Two</td>
         <td>...</td>
         <td>Actions</td>
      </tr>
   </thead>
   <tbody>
      #[collection: collectionname]

      #[pre]#[/pre]

      #[each]
      <tr>
         <td>#field1</td>
         <td>#field2</td>
         <td>#...</td>
         <td>
            <button class="btn edit-button" data-id="#id">Edit</button>
            <button class="btn delete-button" data-id="#id">Delete</button>
         </td>
      </tr>
      #[/each]

      #[post]#[/post]

      #[empty]
      <tr>
         <td colspan="4">No records found.</td>
      </tr>
      #[/empty]

      #[/collection]
   </tbody>
</table>
```

The blue sections indicate the markup for `HtmlRenderer` – they are not printed to the browser and **they don't need to be changed** when they are copied into each new entity CRUD page. Each section has a specific meaning:

- `collection` – this signals the start of a collection block and tells the renderer that you are going to be iterating through the collection identified by the key *`collectionname`* (see below).

- `pre` – the HTML in this section precedes the iterative section, and in this case is left blank (note: it cannot be removed, even if blank).
- `each` – the HTML in this section is printed once for each iteration of the items in the collection. The *`#field1`* tags are used to print attributes of the item – the field names are the same as the column name in the database (see the relevant model file for all of the field names in each item). Note that the names are not "field1", "field2", etc. If you need something more complex than simply printing out a field value, speak to someone from the controller team and they can add it to the controller (or show you how to).
- `post` – the HTML in this section is printed after the iterative section, and again it can be left blank here.
- `empty` – the HTML in this section is printed if and only if the collection is empty.

The red sections indicate the things that should be changed by you:

- *`Field One`* – column headers.
- *`collectionname`* – the name of the collection you are iterating over. It will be the simple plural form of the entity name, in lower case (e.g. "medicines", "staff", "cameras", etc.).
- *`#field1`* – these can access the individual fields via their name, or more complex values (see above).
- *`4`* – should equal the number of columns in use (don't forget the actions column).

The action buttons on each row should not be changed, structurally. Feel free to re-style them (keeping them consistent across pages), but do not remove the **`edit-button`** and **`delete-button`** classes, or the **`data-id`** attributes.

This should be all that is needed to produce the HTML table of the entities.

Create, Update & Delete: The Modal Form

Example files: ***`static/staff.html`*** and ***`static/js/crud-page.js`***

As decided, creating and editing entities will take place through on-page modals. This user action will be repeated on all pages, so it makes sense to make as much of the code as possible fully generic and reusable (in essence, DRY).

So, for each entity page, only the form must be created. Everything else (modals, form-filling, submission, etc.) is handled by ***`crud-page.js`***, which is totally generic and requires no editing for other entity types.

An example of the form is available towards the end of ***`staff.html`***. Much of the structure should remain the same (particularly class names and the buttons) – only the actual form inputs need to be edited. It is absolutely crucial that all names for each input (the three red sections below) match the database column names exactly.

```
<label for="username">Username:</label>
<input type="text" name="username" id="username"/>
```

The form also contains two hidden fields, at the start – only one of these must be changed. The `entity-id` field should stay set to zero; the `entity-type` field should be the simple singular form of the entity being created/edited (e.g. "staff", "camera", etc.).

Throughout the form, there are two control classes that can be used:

- `edit-mode` – these elements will be shown only when *editing* an entity.
- `create-mode` – these elements will be shown only when *creating* an entity.

Below the form, there is more code that must be included in **exactly** this order:

```
#[def: entity-name-title = Staff Member]
#[def: entity-name-body = staff member]
#[def: entity-type = staff]

##_delete-modal.html
##_loading-modal.html

#[collectionmap: objectMap: staff]
#[def: crud = yes]
<script src="/js/staff.js"></script>

##_footer.html
```

Again, red indicates code that should be changed:

- The three `def` fields are used to define details about the delete confirmation modal. The first field fills in the blank in the modal's title (*"Delete ____"*); the second field fills in the blank in the modal's main content (*"Are you sure you want to delete this ____?"*); the third one gives the entity type, which should match the hidden field explained above.
- The two include statements (which start with `##`) load the pre-build modals for the loading message and the delete confirmation – these don't need to be edited.
- The `collectionmap` line tells `HtmlRenderer` to produce a valid JavaScript map of the collection identified by the key in red (see page two).
- The fourth `def` statement enabled CRUD mode.
- The `script` tag includes an (optional) file detailing how the entities should be validated on-page before the forms are submitted (more info below).
- The final line includes the footer of the page, which again should not be changed.

That's it – everything else is handled automatically by *`crud-page.js`*. It deals with hiding/showing modals, responding to button clicks, pre-populating the form, sending data to the back-end via AJAX, etc.

<u>Entity Validation</u>

Example file: `static/js/staff.js`

The entities can be validated before the forms are submitted by specifying functions for the *`validateEditForm`* and *`validateCreateForm`* files in an entity-specific JavaScript file. The functions are passed one argument: an objectivised version of the form. An input field called *`username`* can be accessed as *`formObject.username`*.

These functions should check all attributes: if there are problems they should display errors and return `false`; otherwise, return `true`. If the variables are not specified, no validation takes place.