**Introduction:**

This document is a reference guide for the various end points that the front-end team may connect to through http requests. For each endpoint, there will be a section detailing what object will need to be sent by the front-end team (if needed) to the back-end, as well as, a section detailing what the front-end will expect back from the back-end. There will also be specific http request types that will be specified for each endpoint. Currently, there are only GET and POST request options but PUT and DELETE options are coming in the future.

**User related endpoints:**

First, you’ll need to know what exactly the user itself is so you can create an object to represent a user on the front-end. You will need to make an object that represents what needs to be sent to the back-end and one for what will come back from the back-end. Example – when adding a user, the userID and the last 3 fields won’t need to be sent. However, when a user is sent back, an object will be needed to accept all of these fields.

***HubUser:*** Doesn’t need to be sent but will be returned from back end.

* userID – Integer (not int)
* userName – String
* password – String – Needs to be sent but won’t come back.
* email – String
* fName – String
* lName – String
* dptID – Integer (FK)
* frID – Integer (FK)
* prmID – Integer (FK)

**/users** (controller level) – This endpoint by itself does nothing, however, it is used in conjunction with the rest of the user related endpoints and acts as a controller level endpoint. For example, the method level endpoints are **“/add”, “/all”, and “/{id}”**, and if you wanted **“/add”**, the endpoint you would call would be **/users/add**. The /users gets the controller itself, while the /add is the specific method you want.

**/all** (method = GET) – This endpoint will retrieve all the users in the database. It sends back an Iterable list of HubUser objects. It will send back a list of the above object.

**/{id}** (method = GET) – This method sends back a single HubUser object like above. However, as part of this call, you will need to send the id of the user you wish to get back as a path variable. Example - /users/17. This will return a user with the userID of 17.

**/add** (method = POST) – This method adds a user to the database. The front-end will need to send an object that represents the user class above, minus the userID (that is automatically generated), userComments, tags, and contents fields. The user that was created is then sent back, if everything worked (with an id and the other fields).

**/bydepartment/{dptID}** (method = GET) – This method will send back a list of users who have the matching department id that is sent in as a path variable.

**/byfranchise/{frID}** (method = GET) – Same as bydepartment except it searches based on frID instead.

**/bypermission/{prmID}** (method = GET) – Yet again, it’s the same but uses prmID.

**/byusername/{userName}** (method = GET) – This endpoint will return one user with the same username as what came in with the path variable.

**Side Note: Everything said in this first section will be the case for practically every other entity in the following sections. The main ids of the entity as well as the arrayLists and HashSets will not be needed for sending in an object but will be sent back from the back-end with these fields. So, most entities will need 2 different objects, one for sending and one for receiving.**

**Department+Franchise+HubPermission Entities: These classes don’t have controller level endpoints.**

***HubPermissions:***

* Integer - prmID
* boolean - createItem
* String - prmName

***Department:*** Doesn’t need to be sent but will be returned from back end.

* Integer – dptID
* String – dptName

***Franchise:*** Doesn’t need to be sent but will be returned from back end.

* Integer – frID
* String – frName
* String – frLocation

**/permissions** (method = GET) – This will return an Iterable list of HubPermissions objects. Similar to the users /all endpoint above.

**/permissions/{id}** (method = GET) – This will send back a HubPermissions object with the specified prmID. Example /permissions/1.

**Note: There is no option to add a permission because there are only 2 possible options that have already been added in the database.**

**/departments** (method = GET) – This method sends back an Iterable list of Department objects.

**/departments/{id}** (method = GET) – This method sends back a single Department object with the specified dptID as the path variable. Example - **/departments/3**.

**/departments/add** (method = POST) – This method adds a new Department to the database. This is the same as with the HubUser /add endpoint. Once the department has been added, if everything worked, the department that was added is sent back to the front-end as a Department object with all the fields included (dptID, arrayLists, etc.).

**/franchises** (method = GET) – This method is the same as /departments. It sends an Iterable list of Franchise objects.

**/franchises/{id}** (method = GET) – This method gets a Franchise object using the frID as a path variable. Example - /franchises/4.

**/franchises/add** (method = POST) – This method adds a new Franchise to the database. This is similar to the HubUser /add method. Once the franchise is added, a Franchise object is sent back with all the fields included.

**/franchises/byname/{frName}** (method = GET) – This endpoint returns a franchise by the name passed into the path variable.

**/departments/byname/{dptName}** (method = GET) – This is the same as the one above but will return a department with the name passed in through the path variable.

**Note: You’ll notice that the same kind of methods are used over and over. This will be the case for our program and the following sections will be similar. For this reason, I’m going to keep the rest of the method explanations short and to the point. They will all be similar to ones that we have already mentioned, just with different objects.**

***HubEvent:*** Doesn’t need to be sent but will be returned from back end.

* eventID – Integer
* dptID – Integer (FK)
* frID – Integer (FK)
* eventName – String
* eventDetails – String
* startDate – String
* endDate – String

**/events – Controller level endpoint. Used in conjunction with the following endpoints, similar to HubUser.**

**/all** (method = GET) – This method returns an Iterable list of HubEvent objects.

**/{id}** (method = GET) – This method returns a single HubEvent object using the eventID as a path variable. Example - /events/1.

**/add** (method = POST) – This method adds a HubEvent object to the database. Similar to every other /add method. Returns the same HubEvent object with all fields back to the front-end if everything worked.

/department/{dptID} (method = GET) – This endpoint will return a list of events that have the matching dptID as what was sent in through the path variable.

/franchise/{frID} (method = GET) – This endpoint is the same as the one above, the only difference being that it uses the frID sent in the path variable to return a list of events based on franchise.

***Tag:*** Doesn’t need to be sent but will be returned from back end.

* tagID – Integer
* tagName – String

**/tags – Controller level endpoint. Used in conjunction with the following endpoints, similar to HubUser.**

**/all** (method = GET) – This method returns an Iterable list of Tag objects.

**/{id}** (method = GET) – This method returns a single Tag object using the tagID as a path variable. Example - /tags/1.

**/add** (method = POST) – This method adds a Tag object to the database. Similar to every other /add method. Returns the same Tag object with all fields back to the front-end if everything worked.

***Content:***  Doesn’t need to be sent but will be returned from back end.

* contentID – Integer
* filename – String
* contentName – String
* fileDownloadUri – String
* contentType – String
* size – long
* createDate – String
* active – Boolean
* likes – ArrayList of Like objects, a like is modeled as such: likeID -Integer, userID – Integer, contentID – Integer. It will never be sent in, only returned.

**/content** (method = GET) – This method returns an Iterable list of all the Content objects in the database.

**/content/{id}** (method = GET) – This method returns a single Content object using contentID as a path variable. Example - /content/1.

**/content/add** (method = POST) – This method adds a Content object to the database. Requires 3 parameters in form data format. file(the file being sent), contentName(String), and contentType(String). That’s it, everything else is done by the back end. Similar to every other /add method, this endpoint returns the same Content object with all fields back to the front-end if everything worked. The file is not a field in the content object. The file itself will be saved in the back end and the fileDownloadUri will be sent to the database to use for retrieval of the file.

**/content/all/{contentType}** (method = GET) – This method will return all content with the same content type as the path variable in a list.

**/content/like/{userID}/{contentID}** (method = POST) – This method will add a like to the content and make a new like object to be sent into the database. The 2 path variables are used to associate the like to a user and the content they liked. These path variables are all that is needed and a like object is not required to be sent in.

***Comments:*** Doesn’t need to be sent but will be returned from back end.

* commentID – Integer
* comment – String
* createDate – String
* userName – String
* contentID – Integer (FK)
* userID – Integer (FK)

**/comments – This is the controller level endpoint used in conjunction with the following endpoints. Similar to HubUsers.**

**/all** (method = GET) – This method returns an Iterable list of all the Comments objects in the database.

**/{id}** (method = GET) – This method returns a single Comments object using commentID as a path variable. Example - /comments/1.

**/add** (method = POST) – This method adds a Comment object to the database. Similar to every other /add method. Returns the same Comment object with all fields back to the front-end if everything worked.

**/all/content/{contentID}** (method = GET) – This method returns a list of every comment that shares a contentID with the contentID sent into the path variable.

**/all/user/{userID}** (method = GET) – This endpoint is similar to the one above but returns a list of comments based on the userID that gets sent in the path variable.

**/all/username/{userName}** (method = GET) – This endpoint will return a list of every comment that was made by a certain userName that is sent into the path variable.