My Library Online- API Design

The AP layer will act as abstraction between the UI as well database layer. Tshi ensures that there is loose coupling between frontend and database. Most of the business logic and some validations reside in API layer. If in future the database or frontend needs to use a different technology this can be done easily as integration change would only be updating the API URLs from old UI to new. Business logic which resides in API can remain unchanged and hence minimize the overall changes.

I plan to use HTTP protocol for communication between UI and backend. For UI I would use HTML and JavaScript (and is possible then incorporate Angular framework with which I can use existing libraries for the boilerplate code). For APIs I plan to use nodeJS.

Following are the endpoints that I envision for the 'MyLibraryOnline' project.

1) **GET: resources/searchResource/{criterion}/{searchKeyword}/{sortBy} (MVP):**

   Parameters:

   - criterion: The criterion selected by the user on eth dropdown for the same.
   - searchKeyword: The keyword entered by the user in search screen

   Response:

```
{
"resourceList":[
        {"resourceId":0,
        "resourceName": "string",
         "publicationDate" : 0,
         "resourceCategory": "string"},
        {"resourceId":1,
        "resourceName": "string",
         "publicationDate" : 0,
         "resourceCategory": "string"}
        ]
}
```

   This API will search and return a list of resources which match the user provided keyword against the selected criteria (eg title, author, description). Each record in the list will be data

pulled from resource table. It can be joined with category table to pull the category details if required to display on screen.

Goal: To render data on search screen when user enters search keyword and click on Go button.

2) GET: resources/getResourceDetails/{resourceId}(MVP)

Parameters:

- resourceId: The unique id of the resource which was selected by the user to view details.

Response:

{

"resourceId":0,

"resourceName": "string",

"resourceDescription: "string"

"authorName": "string"

"publicationDate" : 0,

"resourceCategory": "string"

}

Error: "System error while retrieving the resource list, please try again"

This API will pull the details for given resource and return it in response. The resourceId will be provided as parameter.

Goal: To display the resource details when use clicks on the resource to view its details or perform any operation on it.

3) GET: resources/browseResources (MVP)

Parameters:

Response:

{
"resourceListByCategory":[

{
"categoryId" : 0,

"categoryName":"string"

"resourceList": [

{“resourceId”:0,

“resourceName”: “string”,

“resourceDescription: “string”

“authorName”: “string”

“publicationDate” : 0,

“resourceCategory”: “string”},

{“resourceId”:0,

“resourceName”: “string”,

“resourceDescription: “string”

“authorName”: “string”

“publicationDate” : 0,

“resourceCategory”: “string”},

]

}

}

Error: “System error while browsing the resource list, please try again”

This API will return a list of all the resource categories along with the resources that belong to each category.

Goal: To render data on screen when user selects the menu option to browse the resources

4) GET: user/getUserDashboard

This API will return all the resources that are held by the given user. It will be a list of records achieved by joining the user table, resource loans table and resource table.

Goal: To display the list of the resources checked out by user as well as on hold items on the user dashboard.

5) POST: user/placeHold/{resourceId}

This API will be used to place hold on a resource by the logged in user. The resource id will be shared as parameter but can also be provided in request body along with other details.

Goal: To reserve a resource when user clicks place hold on any resource. This API will make an entry in resource loans table.

Success message: The resource has been successfully placed on hold.

Error message: We could not place hold on the resource, please try again.

6) **PUT: user/requestExtension/{resourceId}**

This API will be used to request extension on a resource which is already held or checked out by the user. In database it will update the existing resource loan record to update the due date and increment the extension attempt count.

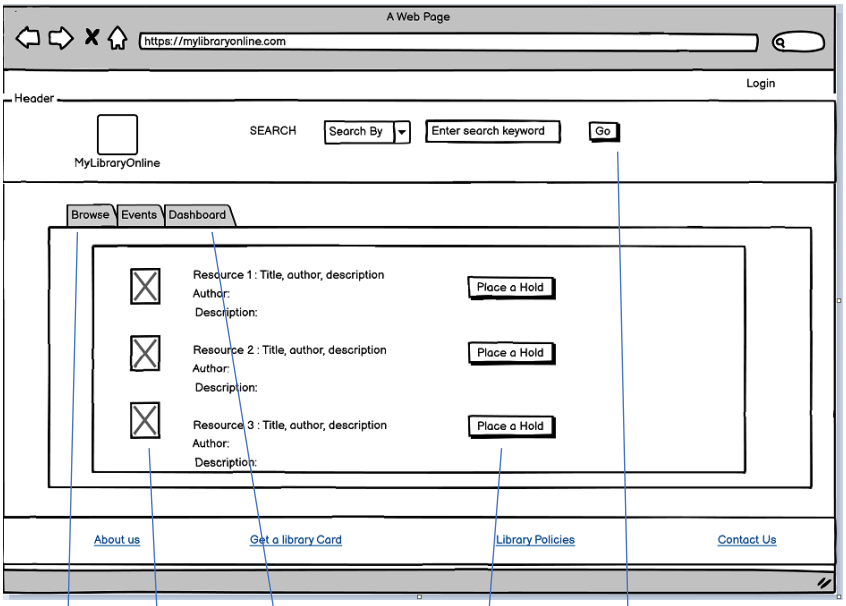
Goal: To extend the hold/checkout on any resource. If the count of extension requested is less than maximum permitted, then the due date of the resource will be extended by the hold period. If the extension attempt exceeds the maximum limit, then the operation will fail and display error to user.

7) **GET: user/login**

This API will receive the user login details (membership id / library card) as well as the last name. These details will be verified and if the data matches then user will be logged in. The response will have user related data (such as first name / last name) to be displayed on the screen if user logs in successfully. If the data does not exist or match, then API returns error message.

Goal: To login the user in application to perform user specific operations such as view dashboard, place hold and request extension.

API calls from application screens



GET: resources/searchResource/{criterion}/{searchKeyword}/{sortBy}

GET: resources/getResourceDetails/{resourceId}

GET: resources/browseResources

GET: user/getUserDashboard

PUT: user/requestExtension/{resourceId}