Project 2- **Pseudo Forum** web Application

Designed and prepared by Nasreddine Hallam

# synopsis:

In this project, you will apply JavaScript coding knowledge and skills acquired thus far to develop a **REACT** web application.

The web application should allow the user to browse a **pseudo-forum**. Forum **data are given** as an endpoint JSON api stored in sonic webserver. Pertinent information about the forum (such as ranking topics, recent posts and stats , etc.) should be filtered and rendered as well. User can also search through the forum using a search query.

# General requirements

* This project **must be done** in groups of **TWO (or THREE) members** only as set by your instructor.
* The duration is **17 days**.

# objectives

* Using Html5, CSS (flex for layout)
* Using JavaScript DOM and events coding
* Using JSON to handle data
* **Mainly coding** using **React** library
* Using Fetch API to **fetch**
* ~~You may use the~~ ~~Browser’s web storage~~?

# Functional Requirements

## 4.1 data description

The data is given in the two json (php) files: **forum-data.php** and **users-data.php,** which can be found in <https://sonic.dawsoncollege.qc.ca/~nasro/js320/project2/>.   
You MUST make yourself familiar with both the data and its underlying structure.

1. **users-data.php** contains a list of authors participating in the forum. Show the most talkative first (i.e. sort by nbepPosts).

**You will create** one stat table for this endpoint (see **ColumnEnd**, last table figure)

1. **Form-data.php** - is the main JSON/php data file that is provided as an api data.   
   The endpoint is the **categories.**

* **categories** - a list of category objects . Each category object has an id, name, author (always set to **admin**), and a **topicList**.
* **topicList** is a list of topic objects structured as (id, topic-title, status (ongoing, closed, resolved, rejected), and **postList**.
* **Poslist** is a list of post objects structured as (id, parentId, author, date, body (2 lines max), rate, likes). Disregard parentId.

**You will create**

**One list for this endpoint** (see **ColumnMain/ListPosts**)

**two stats table for this endpoint** (see **ColumnEnd**, first two table figures)

## 4.2 Main requirements

Develop a React web app that fulfills the following **NINE main requirements**:

1. **Select a forum category** (using the Select drop down menu),
   * then **select a topic** related to that **specific** category,
   * then a list of all posts for that topic **should** **be listed**.
2. **click on like/unlike** - for each post**-** once the like/unlike icon is clicked, a corresponding number should be updated and rendered inside the post itself ONLY**. No data update is needed- this is just visual**
3. **click on delete icon**- if clicked, the corresponding post is removed from the display ONLY- **No remote-sever data update is needed- this is just visual**
4. **Search** through all posts using the search bar.

**Render three panels in the right sidecolumn**:

1. Show **Ranked** topics (based on the number of posts they contain in descending order)
2. Show **recent** posts – posts are displayed based on date
3. Show users’ stats (use-id and the number of posts they posted – sorted by nberPost.  
     
   **Render one panel in the left side column**
4. Show a **dummy ADMIN panel** on the left side-column (no functionality required)
5. Search functionality (Search through posts only) (depicted at the top of the for, in appendix 7)

## 4.3 Structuring you React application

You need to **structure your React-app** in an effective manner.

It is up to you to create folders where you create functional components and their CSS files. You may design as many components as you wish, but nest or structure them properly. For e.g.**, Header, Body, Footer** are the 3 main components under the **App component**. Then, these are further expanded in a sensible way. I am providing here a component design model that you may want to follow or adapt.

## 4.4 Fetching data and rendering components

You *should do* the following:

**To Fetch data**, it is better to design your code in a custom hook that you call **useFetchdata**(url), rather than repeating the same code for the two files. This should be designed as generic as possible and used in different components that need data fetching.

1. **Once date is fetched,** 
   1. Categories – the **Select** **dop down menu** is populated dynamically.   
      User may select one category, then
   2. the **Select drop down** for topics is also populated dynamically based on the selected category.  
      User may select one topic, then
   3. posts related to the selected topic are rendered in the adequate component.
   4. Stat data are produced, sorted and then rendered three tables on the columnEnd (right side)

Data should be fetched as early as possible and passed down to inner components efficiently.

It is up to you to use class or functional components.

# Project Requirements

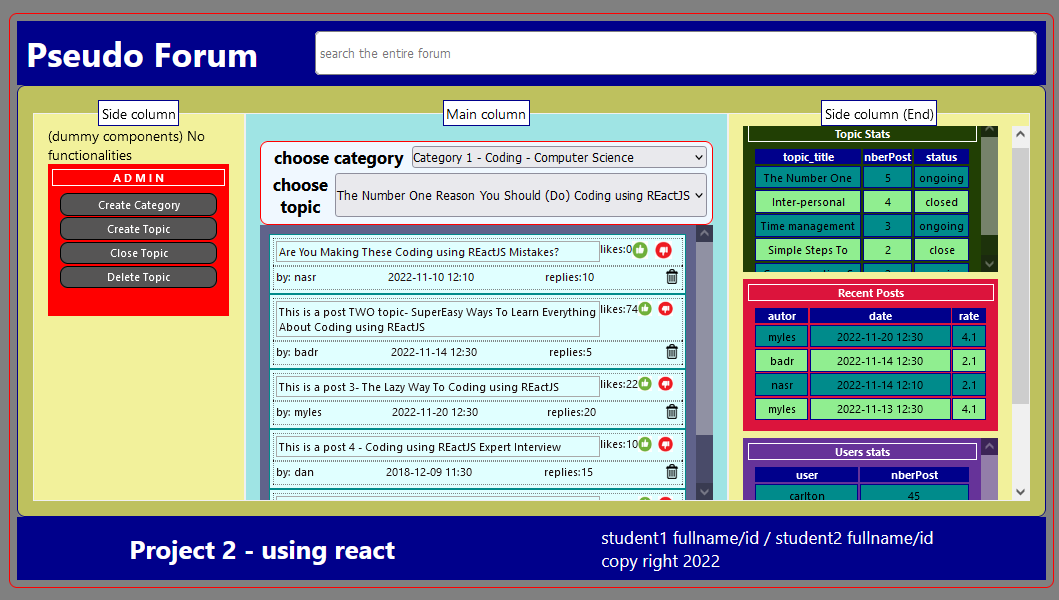
* Teams of **THREE (or two, I decide)** members
* Follow software engineering best practices:
  + **Must** use **flex** layout.
  + Comments are required (your own or JS-DOC).
  + Use proper naming, and comments,
  + Fetch data must **catch errors**
  + **Must use** **map** and **filter array functions**
  + Must be modular

# Submission and deadline

1. Project duration (**17 days**): deadline as set in Lea
2. Name project dw-proj2-grpX-ABCDEFG…
   1. X is a number given to your group
   2. ABCDEFG… is the last name of the group leader
3. Must use git (each member must have the final project in their own git)
4. I must be added as a (git) collaborator to each member of the group
5. Must have **Readme** Text file detailing the content of your project and how to run your program

# Appendix – some GUI

Students can set their own GUI as long as the main requirements are addressed and the GUI is ok/easy to follow/use.



You may follow the component (composition) hierarchy shown below

