User-moderated Commenting System REST API

Introduction

This is a possible implementation of the REST API for A1 based on observing Reddit and Hacker News, however, the implementation of this web app isn't exactly the same. This web application is a single site only.

Frontend

Topics and Comments

Every topic and every comment is assigned a unique ID upon its creation. Topic and comment IDs do not overlap.

1. *'/'* or *'/topic'* gives a list of all the 140 char topic summaries with their associated links. Each topic has buttons that perform requests to the server
    a. *'/topic?id=n'* which shows the comments for the topic with id *n.*
        i. Each comment will have a reply button that will perform *'/reply?id=n'*
    b. *'/reply?id=n'* which allows the user to comment on a topic or comment with the given ID.
        i. Clicking reply to a topic, *n* will equal the topic ID.
        ii. Clicking reply to a comment, *n* will equal the ID of the parent comment.
    c. *'/comments/upvote?id=n'*, upvotes the comment indicated by the given ID. Since topics cannot be upvoted, this should throw some kind of error if the ID belongs to a topic as opposed to a comment.

The Frontpage

1. The frontpage displays all topics as a vertical list at address: *'/'.*
    a. Each summarized topic includes: title, interest link, comment count link, view topic button.
        i. The comment count link *'/comments?id=n'* tells the DOM to reveal all comments for the topic associated with the given ID. This will reveal any

reply buttons on each comment.

ii. By clicking on the topic title the user will be sent to the link of interest.

Singular Topics

1. Singular topics will have all the elements already present in the summarized topics. In addition to the following:
    a. Refreshing the browser will request '/'. The server will respond with an up-to-date list of topic objects that the DOM has to display.

Creating Topics

1. The option to create a new topic will be available on the frontpage. The DOM will reveal a form to create a new topic.
2. There will be cancel option present that will hide the form. The submit button will allow the user to post using the action *'/topic/submit'* by clicking the submit button.
    a. The action *'/topic/submit'* will send to the server all the necessary data to make up a new topic.
    b. The client will parse the text fields in order to prevent sending junk data to the server. There are potential security risks, but for the scope of this assignment it is a reasonable mechanism.

Creating Comments

1. *'/reply/?id=n'* submits the comment on the indicated thread or comment in the URL. The comment field will reveal itself on the client side by clicking on the comment count.

General

1. The server will react to *'/'* or *'/topic'* by sending the entire up-to-date collection of topic objects to the client.

Handling Topics

1. Upon receiving *'/topic/submit'* the server will assume that the browser checked the preconditions for the title. It will create a new object containing the submission data and send the user to the frontpage.

Backend

Storing Topics/Comments

1. Each topic and comment is considered a “node” in the backend, a JSON object stored in an array indexed by the node ID, making topic retrieval from the browser URL easy. Each node has the following key-value pairings:
    - “type” : “topic” or “comment”. Whether the given object is a topic or comment.
    - “content” : The description of the topic/comment. If this object is a topic, should be limited to 140 characters.
    - “vote_count” : How many votes the given topic/comment has. If this object is a topic, should be the sum of all children comment vote_counts.
    - “root_id” : The ID of the topic/comment this node belongs to. For topics, this should be equivalent to its own ID, and for comments this should be the parent topic of this comment.
    - “id”: The id of the node
    - “children_ids” : A list of IDs of the children comments on this topic/comment. Does not include children of children, since implementing that would be more of a pain (would have to store parent comment ID with each comment and go up the tree updating each parent thread/comment when a new comment is created, adding significant complexity and overhead), and we can just compute that recursively anyways.
    - “link” : The link associated with this topic. Should only be present if this node is a topic.
    - “comment_count”: How many comments the given topic/comment has.
2. To determine the index to assign to a new node, assuming indices start at 0, just assign the length of the list of nodes as the index(equivalent to appending it to the end of the list). This way we don’t have to worry about remembering to increment some other variable or something every time a new node is added.

Submitting Topics

1. When a request to *‘/topic/submit’* is made, the user has presumably just filled out the topic form and sent it in. We receive this data and process it into a node object, adding it to the list of nodes.
2. Return to the front page. The page’s DOM should be updated to show the new topic.

Submitting Comments

1. Similar to submitting topics, since they’re basically the same, just without a link field.

User sends *'/reply?id=n'*, parse data from form and add new node. However, not only do we add the new node to the list, we also need to take the index and add it to the children_ids list of the node this comment is acting as a reply to. We can figure out the *topic_id* of the new node from its parent node's *topic_id*.

Upvoting Comments

1. When the user sends *'/comments/upvote?id=n'*, check to make sure that the ID is valid and does not correspond to a topic. If it's valid, increment the vote count of the node specified by the given ID, as well as the vote count of the topic associated with the node.