Project Overview

For the **Readable** project, you will build a content and comment web app. Users will be able to post content to predefined categories, comment on their posts and other users' posts, and vote on posts and comments. Users will also be able to edit and delete posts and comments.

Why this project?

This content and comment structure is common across a large number of websites and applications, from news sites to blogs to aggregators like Hacker News and Reddit. By building this project, you will gain an understanding of how Redux can function in a standard type of application.

Specification

You will start with **local backend development server**. The server is built in Node, but it is very simple. You won't need to edit the server code; instead, your code will talk to the server using documented API endpoints. You can use the server's endpoints to manage storing, reading, updating, and deleting data for your application.

Using this server, you will build a React/Redux front end for the application. The specification provided below is the minimum required for this project. You may extend your project however you like, however, as long as the minimum specification is met.

Data

There are three types of objects stored on the server:

- Categories
- Posts
- Comments

Categories

The server supports a small, fixed number of categories that users can put posts into. Categories are simple objects containing a name and a URL path (usually the same string). The server does not have methods for creating/modifying/deleting these categories. If you wish to add to the categories for your app, simply add your desired object to the Array in `categories.js` in the provided server.

Posts

Posts are the building blocks of your application. Posts include:

Attribute	Type	Description
id	String	Unique identifier
timestamp	Integer	Time created - default data tracks this in **Unix time**. You can use `Date.now()` to get this number

Attribute	Type	Description
title	String	Post title
body	String	Post body
author	String	Post author
category	String	Should be one of the categories provided by the server
voteScore	Integer	Net votes the post has received (default: 1)
deleted	Boolean	Flag if post has been 'deleted' (inaccessible by the front end), (default: false)

Comments

Comments are attached to parent posts. They include:

Attribute	Type	Description
id	String	Unique identifier
parentId	String	id of the parent post
timestamp	Integer	Time created - default data tracks this in **Unix time**. You can use `Date.now()` to get this number
body	String	Comment body
author	String	Comment author
voteScore	Integer	Net votes the post has received (default: 1)
deleted	Boolean	Flag if comment has been 'deleted' (inaccessible by the front end), (default: false)
parentDeleted	Boolean	Flag for when the the parent post was deleted, but the comment itself was not.

> This application is anonymous, with *no* authentication or authorization. There are no user objects, and comments and posts accept any username/name for creation and editing.
>
> The server is *very* light weight. It performs *zero* data validation to enforce the above data types. Make sure you are using the correct types when sending requests to the server.

Views

Your application should have, at a minimum, four views:

- Default (Root)

    - should list all available categories, which should link to a category view for that category
    - should list all of the posts ordered by voteScore (highest score first)
    - should have a control for changing the sort method for the list, including at minimum, order by voteScore and order by timestamp
    - should have a control for adding a new post

- Category View
    - identical to the default view, but filtered to only include posts with the selected category

- Post Detail View
    - should show the details of a post, including: Title, Body, Author, timestamp (in user readable format), and vote score
    - should list all of the comments for that post, ordered by voteScore (highest first)
    - should have a control for reordering comments by score or timestamp
    - should have controls to edit or delete the post
    - should have a control to add a new comment.
    - implement comment form however you want (inline, modal, etc.)
    - comments should also have controls for editing or deleting

- Create/Edit View
    - should have a form to create new post or edit existing posts
    - when editing, existing data should be populated in the form

> **Post/Comment UI**
>
> Posts and comments, in all views where they are displayed, should display their current score and should have controls to increment or decrement the `voteScore` for the object. Posts should display the number of comments associated with the post.

Specific Requirements

Use React to build your application UI. Remember that composition is key. It's rarely a mistake to break a component into smaller pieces. Look for opportunities to reuse your components. We recommend using `create-react-app` to bootstrap your project, but it is not required for this project.

While the focus (and specification) of this project is based on functionality, rather than styling, please ensure that your app is presentable and easy to navigate.

Use Redux to manage your application state. This includes all user actions and responses from the API server. You should have *no* state handled by your components. All of the state for your application should be controlled by your reducers.

> There are times in production environments when it is appropriate to have components own some part of their own state. Since the purpose of this project is to assess your ability to implement Redux in code, that is not an option available for this project.