Tweeter documentation

Basic character counter

```
$(document).ready(function() {

      $("#tweet-text").on('keyup', function(event) {

        var charCount = $(this).val().length;

        // Update the character count display

        $('#char-counter').text(charCount);

      });

    });
```

Character Limit counter

```
$(document).ready(function() {

    // --- our code goes here ---

      $("#tweet-text").on('keyup', function(event) {

        var charCount = $(this).val().length;
        var setLimit = 140;
        var charCounter = setLimit - charCount;
        $('#char-counter').text(charCounter);
        if (charCounter < 0){

        $('#char-counter').css( "color", "red");

        }

      });

    });
```

Rendering tweets

JSON data

```
const data = [
  {
    "user": {
      "name": "Newton",
      "avatars": "https://i.imgur.com/73hZDYK.png" ,
      "handle": "@SirIsaac"
    },
    "content": {
      "text": "If I have seen further it is by standing on the shoulders of giants"
    },
    "created_at": 1461116232227
  },
  {
    "user": {
      "name": "Descartes",
      "avatars": "https://i.imgur.com/nlhLi3I.png",
      "handle": "@rd" },
    "content": {
      "text": "Je pense , donc je suis"
    },
    "created_at": 1461113959088
  }
];
```

Render Tweets

This function can be responsible for *taking in an array of tweet objects* and then appending each one to the `#tweets-container`. In order to do this, the `renderTweets` will need to leverage the `createTweetElement` function you wrote earlier by passing the tweet object to it, then using the returned jQuery object by appending it to the `#tweets-container` section.

Calls `createTweetElement(tweet)` every loop

```
const renderTweets = function(tweets) {

  // loops through tweets
```

```
    // calls createTweetElement for each tweet

    // takes return value and appends it to the tweets container

    // data [{},{},{}]

    console.log(tweets.length);

    for (let i = 0; i < tweets.length; i++) {// {}, {}, {}

      let tweet = tweets[i];

      let newTweet = createTweetElement(tweet);

      $('#tweets-container').prepend(newTweet);

    }

  };
```

Create tweet element

This is the template we will call every loop from `renderTweets()`

```
const createTweetElement = function(tweet) {

    const $tweet = $(`

      <article class="tweet">

        <header>

          <img src="${tweet.user.avatars}" alt="${tweet.user.name}"
width="64">

          <h3>${tweet.user.name} <span class="username">${tweet.user.handle}
</span></h3>

        </header>

        <div class="content">

          ${tweet.content.text}
        </div>

        <footer>

          <div class="date">${new Date(tweet.created_at).toLocaleString()}
</div>

          <div class="actions">

            <span><i class="fa-solid fa-flag"></i></span>

            <span><i class="fa-solid fa-retweet"></i></span>

            <span><i class="fa-solid fa-heart"></i></span>

          </div>

        </footer>

      </article>

      </br>

    `);

    return $tweet;

  };
```

Run function

```
  renderTweets(data);
```

Submitting a tweet

Submitting a form - Add an Event Listener and Prevent the Default Behaviour

`.preventDefault()` prevent the default form submission behaviour of sending the post request and reloading the page

example 1

```
$( "#target" ).on( "submit", function( event ) {
  alert( "Handler for `submit` called." );
  event.preventDefault();
});
```

example 2

```
$( "form" ).on( "submit", function( event ) {
  if ( $( "input" ).first().val() === "correct" ) {
    $( "span" ).text( "Validated... " ).show();
    return;
  }

  $( "span" ).text( "Not valid!" ).show().fadeOut( 1000 );
  event.preventDefault();
} );
```

Serializing

.serialize()` function turns a set of form data into a query string. This _serialized_ data should be sent to the server in the data field of the AJAX POST` request.

example 1

```
$( "form" ).on( "submit", function( event ) {
event.preventDefault();
console.log( $( this ).serialize() );
});
```

example 2

```
  function showValues() {
    var str = $( "form" ).serialize();
    $( "#results" ).text( str );
  }
  $( "input[type='checkbox'], input[type='radio']" ).on( "click", showValues
);
  $( "select" ).on( "change", showValues );
  showValues();
```

Ajax POST method

example 1

```
 $('#myForm').on('submit', function(event) {
              event.preventDefault(); // Prevent the default form submission

              $.ajax({
                  url: 'https://example.com/submit', // Replace with your
server URL
                  type: 'POST',
                  data: {
                      name: $('#name').val(),
                      email: $('#email').val()
                  },
                  success: function(response) {
                      console.log('Form submitted successfully!');
                      console.log(response);
                  },
                  error: function(error) {
                      console.log('Error submitting form');
                      console.log(error);
                  }
              });
          });
```

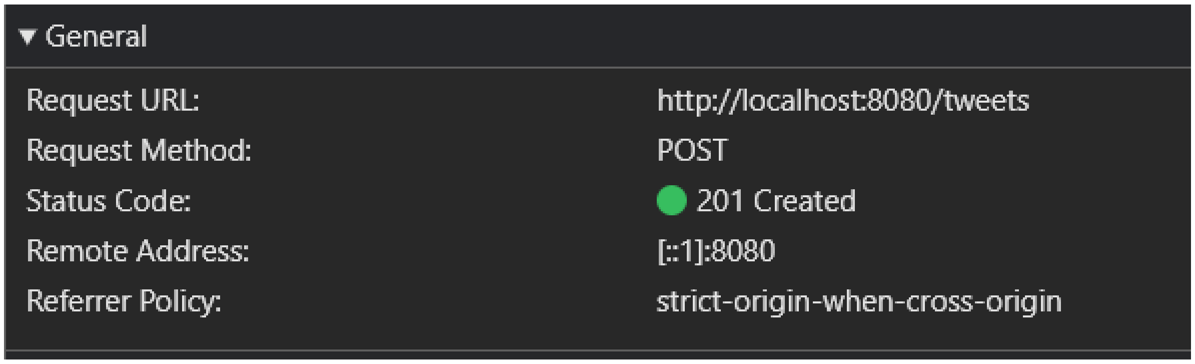
Ajax Post implementation

This creates the rendered tweets (called from `loadTweets()`) asynchronously once the user presses the submit button.

```
  const $tweetForm = $('#tweet-form'); // the id for the form
```

```
 $tweetForm.submit(function(event) {

   event.preventDefault();

   $.ajax({
     url: '/tweets',
      method: 'POST',
      data: $(this).serialize()
    })

      .then(() => {
        $('#tweet-text').val(''); // clears text area after a sucessful post
        $('#char-counter').text(140); // Resets counter when post
        loadTweets(); // Load tweets again to get the new one
        $('.error').slideUp();
        $("#tweet-text").val().length = 0;
      })
      .catch((error) => {
        console.log("error: ", error);

      });
});
```

In the networks tab you should see this POST request made once the user presses the button.



```
▼ General
Request URL:            http://localhost:8080/tweets
Request Method:         POST
Status Code:            ● 201 Created
Remote Address:         [::1]:8080
Referrer Policy:        strict-origin-when-cross-origin
```

Ajax GET implementation

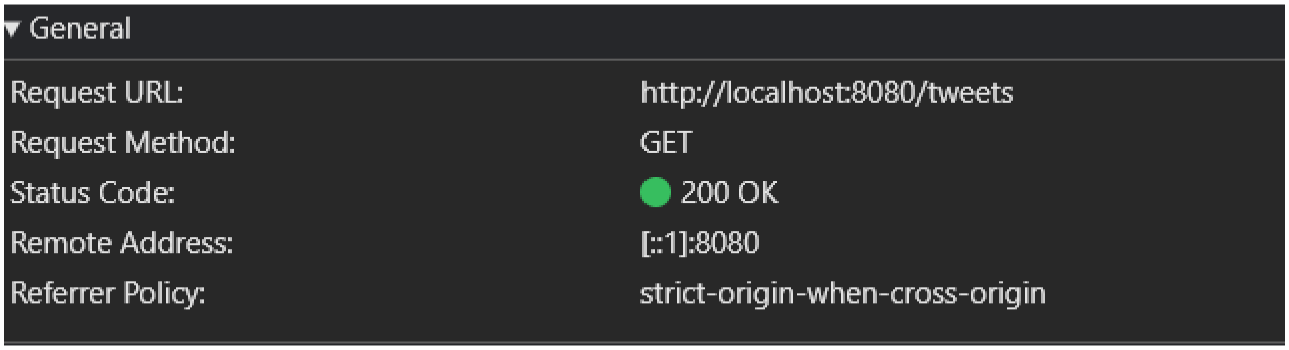
Placed before the POST, `loadTweets()` makes a GET HTTP request to fetch the tweets by calling `renderTweets()`. if successful and catches error if failed. Once the document is fully loaded, `loadTweets()` is called and is run.

```
const loadTweets = function() {
    $.ajax({
      url: '/tweets',
      method: 'GET',
    })
      .then((data) => {
        renderTweets(data.reverse()); // To show the recent tweets on top
      })
      .catch((error) => {
        console.log("error: ", error);
      });
  };
  loadTweets(); // This is called once the document is ready
```

We should also adjust the options on `<form>` in the `index.html`

```
<form action="/tweets" method="POST" id="tweet-form">
<!-- Your form content -->
</form>
```

We should see this GET request upon first opening the document, as well as when a user submits the form



```
▾ General
Request URL:                    http://localhost:8080/tweets
Request Method:                 GET
Status Code:                    ● 200 OK
Remote Address:                 [::1]:8080
Referrer Policy:                strict-origin-when-cross-origin
```

Error handling

I handled errors using the following conditional statements that slides down styled messages with the jQuery `.slideDown()` built in function.

```


 let setLimit = 140;
 if ($("#tweet-text").val().length === 0){ // user inputs 0 chars
$('.error').slideUp(); // slide up in case you have an error message already
 return $("#empty").slideDown(); // the new error message
   }
    if ([ ... $("#tweet-text").val()].every(char => char === ' ')){ //converts
the string to indiv chars and checks if every char is a space
    $('.error').slideUp(); // slide up in case you have an error message
already
    return $("#empty").slideDown();

      }
if ($("#tweet-text").val().length > setLimit){ // user inputs over set char
limit
```

```
    $('.error').slideUp(); // slide up in case you have an error message
already
    return $("#overLim").slideDown();

    }
```

We would place the below conditionals around the `POST` request, just to prevent the user from going to tweet route if the conditions are not met

```
if ($("#tweet-text").val().length > 0 && $("#tweet-text").val().length ≤
setLimit ){

// AJAX POST request here

}
```