jQuery & Ajax applications

Internet Applications, ID1354

Linus Berg Linus@Fenix.me.uk

December 10, 2018

1 Introduction

The seminar consisted of learning JavaScript, jQuery, and implement Ajax request into the recipes website.

2 Literature Study

I was already familiar with jQuery and Ajax requests, all previous seminars already utilised jQuery and Ajax.

3 Method

As stated above, all previous seminars utilised Ajax and jQuery, therefor no changes were made for this seminar. However my initial thought during development was that Ajax is a more modern method for updating a webpage rather than conducting POST/GET requests by redirection, and I chose to implment it as I found it easier.

In the previous seminars, a simple .php file with a script was developed, as can be seen in seminar 2, then each one was easily requested via Ajax and a response was easily managed. This separated the PHP from HTML further, and made the program easier to manage.

During seminar 3, when a MVC framework had to be implemented, the files were switched to the Laravel routing, and utilised the controllers instead to manage each Ajax request.

Tool	Choice
Editor	*Vim*
Version Control	*Git - Github*
Web Server	*nginx*
Database	*PostgreSQL 10.5-1*
PHP	*7.2.10-1*
Framework	*Laravel 5.7*

4 Result

Github
As the javascript code was included since the first seminar the code has gone through several structural changes in iterations, however, login has remained similar throughout the program, with the addition of the CSRF token as can be seen in 1. The login buttons have bound submit events via jQuery to submit an ajax request to the server upon clicking one of the buttons, upon completion it reloads the site (this is obviously not the way ajax was intended to be used, however the seminar did not mention you had to make it rewrite the current page), this is because if the user was sucessfully logged in, the page will reload and the laravel Auth model will display the logged in user instead.

```
<script>
  var CSRF_TOKEN = $('meta[name="csrf-token"]').attr('content');
  $('#auth_login').on('submit', function (e) {
    var data = {
      email: $('#email').val(),
      password: $('#password').val()
    };

    $.ajax({
      headers: {
        'X-CSRF-TOKEN': CSRF_TOKEN
      },
      type : 'POST',
      url : '{{ route("login") }}',
      data: data,
      success : function(d) {
        location.reload();
      },
      error : function(XMLHttpRequest, textStatus, errorThrown) {
        console.log(XMLHttpRequest);
        console.log(textStatus);
        console.log(errorThrown);
      }
    });
    e.preventDefault();
  });

  $('#auth_logout').on('submit', function (e) {
    $.ajax({
      headers: {
        'X-CSRF-TOKEN': CSRF_TOKEN
      },
      type : 'POST',
      url : '{{ route("logout") }}',
      data: {},
      success : function(d) {
        location.reload();
      },
      error : function(XMLHttpRequest, textStatus, errorThrown) {
        console.log(XMLHttpRequest);
        console.log(textStatus);
        console.log(errorThrown);
      }
    });
    e.preventDefault();
  });
</script>
```

Figure 1: Login and out.

```
    /* Current user. */
@if (Auth::check())
    const user = '{{ Auth::user()->email }}';
@endif
    const recipe_id = <?php echo $DISH_ID ?>;
    var CSRF_TOKEN = $('meta[name="csrf-token"]').attr('content');

    /* COMMENT - POST. */
    $('#comment_form').on('submit', function (e) {
      //ajax call here
      $.ajax({
        headers: {
          'X-CSRF-TOKEN': CSRF_TOKEN
        },
        type : 'POST',
        url : '{{ url("comment/post") }}',
        data: {
          dish_id : recipe_id,
          txt: $('#comment_txt').val()
        },
        success : function(d) {
          location.reload();
        },
        error : function(XMLHttpRequest, textStatus, errorThrown) {
          console.log(errorThrown);
          console.log(textStatus);
          console.log(XMLHttpRequest);
        }
      });
      e.preventDefault();
    });
```

Figure 2: Post comment.

Upon posting a comment the same as in the login buttons occur, a submit event is bound via jQuery and upon triggering it submits an ajax request to the server with the relevant data to store the comment in the database, again upon completion it reloads the current page.

```
$.ajax({
  type : 'GET',
  url : '{{ url("comment/get_all") }}',
  dataType : 'json',
  data: {
    dish_id : recipe_id
  },
  success : printComments,
  error : function(XMLHttpRequest, textStatus, errorThrown) {
    console.log(errorThrown);
    console.log(textStatus);
    console.log(XMLHttpRequest);
  }
});

function printComments(data) {
  /* Build comment section. */
  console.log(data)
  var wrapper = d3.select('#comments');
  for (var i in data) {
    var comment_user = data[i]['user'];
    var comment = data[i]['comment'];
    var row = wrapper.append('div').attr("class", "comment");
    var cm = row.append('span')
                .text('\'' + comment + '\' - ' + comment_user);

    /* If the user is logged in... */
    /* DELETE COMMENT BTN */
    if (user == comment_user) {
      cm.append('input')
        .attr('type', 'button')
        .attr('class', 'btn')
        .attr('name', 'del-comment')
        .attr('value', 'Delete')
        .attr("cmt_id", data[i]['id'])
        .on('click', function (el) {
          $.ajax({
            headers: {
              'X-CSRF-TOKEN': CSRF_TOKEN
            },
            type: 'DELETE',
            url: '{{ url("comment/delete") }}',
            data: {
              'cmt_id': d3.select(this).attr("cmt_id")
            },
            success: function(msg){
              location.reload();
            },
            error : function(XMLHttpRequest, textStatus, errorThrown) {
              console.log(errorThrown);
              console.log(textStatus);
              console.log(XMLHttpRequest);
            }
          });
        });
    }
  }
}
```

When the recipe page is loaded the scripts in 3 are executed, creating a GET request to the server to recieve all comments for the relevant recipe, in a json structure, no HTML data is sent, this section could easily be modified to separate the printComments function so the post comment code (2) does not need to reload the page.

```
$.getJSON("{{ asset('json/recipes.json') }}", function(data) {
  var recipe = data[recipe_id];
  $('#dish').text(recipe.dish);
  $('#prep_time').text('Prep time: ' + recipe.prep);
  $('#servings').text('Servings: ' + recipe.servings);
  for (var key in recipe.ingredients) {
    $("#ingredients").append('<li>'+ key + ' - ' + recipe.ingredients[key]  '</li>');
  }
});
```

Figure 4: Build recipe.

Last but not least is the javascript to build the recipe description, prep time, and all the relevant information. 4 utilises the built in getJSON in jQuery, resulting in not needing to set headers and manage data at a more rudimentary level.

5 Discussion

All the mandatory tasks have been met, ajax is used for almost every functionality on the site, however not efficently as it still reloads the pages, this was more out of stress (I have to retake an exam Monday the 17th) and I chose to not dwell too deep into the seminar to focus more on the exam.

The javascript code could have been more structured as well, such as put into it's own file and actually used classes, however it felt uneccessary for such a simple task. Again javascript/jquery/ajax has been used since the first seminar, and some details were hard to recall and I can not actually recall if any issues were encountered, however, I do not think there were any major problems. In general it was hard to recall many details which is why the report is so sparse.