Technologies for Web and Social Media - Hamsterchan

Daniel Kartin

Introduction

This website was built using EmberJS, and the accompanying server was built using node.js the database is a noSQL mongoDB cluster hosted on atlas. There is also a dash of ajax, for making the *POST* request.

Emberjs

Emberjs is a javascript framework much akin to angular and react, but with a much better looking mascot.

Figure 1: The Emberjs mascot, Tomster

Mostly I went with Emberjs instead of Angular due to the documentation, syntax, and ease of use.

Purpose

The purpose of this webpage is for connecting people who share a passion for hamsters, who with a title, image, and description can take part in the hamster culture. I initially begun work on implementing comments, but after much fiddling couldn't get it to work with the database schema.

The Technologies

Starting off with the frontend, Emberjs makes it quick to create something tangible, as it creates routing automatically, and has many addons that enables functionality like sass or scss. Emberjs pages are consisting of templates that are writing in the hbs (handlebars) format.

For creating the main page, where the newest 50 posts are being displayed, a sort of for loop was created, where we iterate over each post in the data, from ember.data. And creates the post taking in the `post.id`, `post.image`, `post.title`, and `post.text` to fill in each post with content.

```
{{#each this.model as |post|}}
     <div class="post_container">
         <p class="post_id">Id: {{post.id}}</p>
         <img src="{{post.image}}" class="post_image"
         style='max-width: 200px; max-height: 200px;'>
         h1 class="post_title">{{post.title}}</h1>
         <h3 class="post_text">{{post.text}}</h3>
     </div>
```

And ember even allows for else statements, where if no posts were found, it shows an empty post, urging the user to create the first post.

```
{{else}}
    <div class="post_container">
        <img src="image url" class="post_image"
        style='max-width: 200px;max-height: 200px;'>
        <h1 class="post_title">No posts have been made yet</h1>
        <h3 class="post_text">Hurry up and create the first post</h3>
    </div>
{{/each}}
```

So far it has only been about showing the content of the database on the frontend, but how does one make a new post, you might ask. Why, with the new post button of course. A simple form, saving the values in their desired variables, to be used in the ajax request.

```
Title: {{input value=title}} <br>
Image url: {{input value=image}} <br>
Text: {{textarea value=text}} <br>
<input type="submit" {{action "sendRequest" title image text}}>
```

The actual ajax request here, with the title, image, and text from the new post button template.

```
if(title.length > 5 && text.length > 20) {
    this.get('ajax').request('https://web-miniproject-
    server.herokuapp.com/api/posts', {
        method: 'POST',
        data: {
        post: {
            title: title,
            image: image,
            text: text
        }
    }
});
```

On the server side, the nodejs server is equipped with express, mongoose, and body parser, with which I made the post schema, and allowed it to save the incoming data as posts, and send back the newest 50 posts.

The function called on the *Post* schema, gets every post in the database, limits it to 50 and sorts it in reverse, so that the newest post would be on top of the page.

```
Post.find(function(err, posts) {
    if (err) {
        res.send(err);
    }
    res.json({posts: posts});
}).limit(50).sort({_id:-1});
```