Lesson Tuesday

JavaScript (/javascript) / Ember Extended (/javascript/ember-extended) / Optional: Google Maps as an Ember Service

Text

In this lesson, we'll work with an Ember service (http://guides.emberjs.com/v2.0.0/services-and-initializers/services/). Ember services are objects that can be used throughout the application for functionality such as logging, authentication, APIs, etc. The Ember service that we will create in this lesson will be responsible for requesting and receiving a Google Map from the Google Maps API.

Model and Template Updates

To get started, we know from the Google Maps documentation (https://developers.google.com/maps/documentation/javascript/tutorial) that Google Maps need a longitude and latitude to create a map object to display. Let's add those new properties to our `rental` model:

app/models/rental.js

```
import DS from 'ember-data';

export default DS.Model.extend({
  owner: DS.attr(),
  city: DS.attr(),
  type: DS.attr(),
  image: DS.attr(),
  bedrooms: DS.attr(),
  reviews: DS.hasMany('review', { async: true }),
  cost: DS.attr(),
  latitude: DS.attr('number'),
  longitude: DS.attr('number')
});
```

We'll add these fields to our new rental form so that latitude and longitude can be collected for new rentals:

app/components/new-rental.hbs

```
{{#if addNewRental}}
  <h1>New Rental</h1>
  <div>
    <form>
      <div class="form-group">
        <label for="owner">Owner</label>
        {{input value=owner id="owner"}}
      </div>

      <div class="form-group">
        <label for="type">Type</label>
        {{input value=type id="type"}}
      </div>

      <div class="form-group">
        <label for="city">City</label>
        {{input value=city id="city"}}
      </div>

      <div class="form-group">
        <label for="bedrooms">Number of Bedrooms</label>
        {{input value=bedrooms id="bedrooms"}}
      </div>

      <div class="form-group">
        <label for="image">Image URL</label>
        {{input value=image id="image"}}
      </div>

      <div class="form-group">
        <label for="cost">Cost per night</label>
        {{input value=cost id="cost"}}
      </div>

      <div class="form-group">
        <label for="latitude">Latitude</label>
        {{input value=latitude id="latitude"}}
      </div>

      <div class="form-group">
        <label for="longitude">Longitude</label>
        {{input value=longitude id="longitude"}}
      </div>

      <button {{action 'saveRental1'}}>Save</button>
    </form>
  </div>
{{else}}
  <button {{action 'rentalFormShow'}}>New Rental</button>
{{/if}}
```

And in the javascript part of the new-rental component, we'll add the latitude and longitude to our `params` :

app/components/new-rental.js

```
...
    saveRental1() {
      var params = {
        owner: this.get('owner'),
        city: this.get('city'),
        type: this.get('type'),
        image: this.get('image'),
        bedrooms: this.get('bedrooms'),
        cost: parseInt(this.get('cost')),
        latitude: this.get('latitude'),
        longitude: this.get('longitude')
      };
      this.set('addNewRental', false);
      this.sendAction('saveRental2', params);
    }
...
```

Let's add a new area and component to our `rental-detail` template that will display and manage our Google Map for each rental.

app/templates/components/rental-detail.hbs

```
Located in {{rental.city}}, this {{rental.bedrooms}} bedroom {{rental.type}} is
available by arrangement through {{rental.owner}}.
<br>
<p><img src={{rental.image}} alt={{rental.type}}></p>

<h2> Reviews </h2>
<ul>
  {{#each sortedReviews as |review|}}
    {{review-tile review=review destroyReview="destroyReview"}}
  {{/each}}
</ul>

<div>
  <h3>Map of {{rental.owner}}'s {{rental.type}}</h3>
  {{google-map rental=rental}}
</div>

<button {{action 'delete' rental}}>Delete</button>
```

Adding the Component and Service

In the next step, we generate the component *google-map* and the service *google-map.js*. We want the component to handle the behavior of the display. We want the service to handle the API to Google maps.

Following the Google Maps documentation under Map Options (https://developers.google.com/maps/documentation/javascript/tutorial), we know that there are a few required options to make a map:

1. a container to put the map into
2. a center for the map defined by the latitude and longitude
3. and the level of zoom for the map

Let's first create the container in our generated component template:

app/templates/components/google-map.hbs

```
<button {{action 'showMap' rental}}>Show Map</button>

<div class="map-display"></div>
```

Our template will show a button to trigger the component's action and provide a `div` with the class `map-display` where the map will display.

We can make sure our `map-display` area is appropriate to the size of the page by adding some styling:

app/styles/app.css

```
.map-display {
    height: 600px;
    width: 600px;
  }
```

Now let's generate our service:

```
$ ember g service google-map
```

A *google-map.js* file is generated in our app's *services* folder. We'll add the following code:

app/services/google-map.js

```
import Ember from 'ember';

export default Ember.Service.extend({
  googleMaps: window.google.maps,
  findMap(container, options) {
    return new this.googleMaps.Map(container, options);
  },
  center(latitude, longitude) {
    return new this.googleMaps.LatLng(latitude, longitude);
  }
});
```

Let's look at this service's code. We set a variable `googleMaps` to `window.google.maps` so that we have a shortcut for code we will be repeating. Then we set two properties (documented in the Google Maps API) for finding the map and the center.

Our final step is to use the service in our component to return the map to the display:

app/components/google-map.js

```
export default Ember.Component.extend({
  map: Ember.inject.service('google-map'),

  actions: {
    showMap(rental) {
      var container = this.$('.map-display')[0];
      var options = {
        center: this.get('map').center(rental.get('latitude'), rental.get('longitude')),
        zoom: 15
      };
      this.get('map').findMap(container, options);
    }
  }
});
```

To use the service, we inject it into the component by setting the `map` variable to our service named `google-map`. Now, we can use `map` in our component to reference the service as it is needed.

When the `Show Map` button is clicked, the action sets the values needed by Google Maps to generate a map: a container (our display `div`) and options: center and zoom. To set the center, the service is called with the values from the rental: latitude and longitude. With the container and options set, the final job of the component is to show the map by getting it from the service's `findMap` property.

Note that the component never communicates with Google Maps directly but only uses our `google-map` service.