عنوان: **ساخت یک بلاگ ساده با Ember.js، قسمت چهارم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۹ ۸:۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, jQuery, SPA, EmberJS

در قسمت قبل ، اطلاعات نمایش داده شده، از یک سری آرایه ثابت جاوا اسکریپتی تامین شدند. در یک برنامه‌ی واقعی نیاز است داده‌ها را یا از HTML 5 local storage تامین کرد و یا از سرور به کمک Ajax. برای اینگونه اعمال، ember.js به همراه افزونه‌ای است به نام Ember Data که جزئیات کار با آن‌را در این قسمت بررسی خواهیم کرد.

استفاده از Ember Data با Local Storage

برای کار با HTML 5 local storage نیاز به Ember Data Local Storage Adapter نیز هست که در قسمت اول این سری، آدرس دریافت آن معرفی شد. این فایل‌ها نیز در پوشه‌ی Scripts\Libs برنامه کپی خواهند شد.
در ادامه به فایل Scripts\App\store.js که در قسمت قبل جهت تعریف دو آرایه ثابت مطالب و نظرات اضافه شد، مراجعه کرده و محتوای فعلی آن‌را با کدهای زیر جایگزین کنید:

```
Blogger.ApplicationSerializer = DS.LSSerializer.extend();
Blogger.ApplicationAdapter = DS.LSAdapter.extend();
```

این تعاریف سبب خواهند شد تا Ember Data از Local Storage Adapter استفاده کند.
در ادامه با توجه به حذف دو آرایه‌ی posts و comments که پیشتر در فایل store.js تعریف شده بودند، نیاز است مدل‌های متناظری را جهت تعریف خواص آن‌ها، به برنامه اضافه کنیم. این‌کار را با افزودن دو فایل جدید comment.js و post.js به پوشه‌ی Scripts\Models انجام خواهیم داد.
محتوای فایل Scripts\Models\post.js :

```
Blogger.Post = DS.Model.extend({
  title: DS.attr(),
  body: DS.attr()
});
```

محتوای فایل Scripts\Models\comment.js :

```
Blogger.Comment = DS.Model.extend({
  text: DS.attr()
});
```

سپس مداخل تعریف آن‌ها را به فایل index.html نیز اضافه خواهیم کرد:

```
<script src="Scripts/Models/post.js" type="text/javascript"></script>
<script src="Scripts/Models/comment.js" type="text/javascript"></script>
```

برای تعاریف مدل‌ها در Ember data مرسوم است که نام مدل‌ها، اسامی جمع نباشند. سپس با ایجاد وهله‌ای از DS.Model.extend یک مدل ember data را تعریف خواهیم کرد. در این مدل، خواص هر شیء را مشخص کرده و مقدار آن‌ها همیشه DS.attr() خواهد بود. این نکته را در دو مدل Post و Comment مشاهده می‌کنید. اگر دقت کنید به هر دو مدل، خاصیت id اضافه نشده‌است. این خاصیت به صورت خودکار توسط Ember data تنظیم می‌شود.

اکنون نیاز است برنامه را جهت استفاده از این مدل‌های جدید به روز کرد. برای این منظور فایل Scripts\Routes\posts.js را گشوده و مدل آن‌را به نحو ذیل ویرایش کنید:

```
Blogger.PostsRoute = Ember.Route.extend({
    //controllerName: 'posts', // مقدار پیش فرض است و نیازی به ذکر آن نیست
```

```
    //renderTemplare: function () {
    //    this.render('posts'); // مقدار پیش فرض است و نیازی به ذکر آن نیست
    //},
    model: function () {
        return this.store.find('post');
    }
});
```

در اینجا this.store معادل data store برنامه است که مطابق تنظیمات برنامه، همان ember data می‌باشد. سپس متد find را به همراه نام مدل، به صورت رشته‌ای در اینجا مشخص می‌کنیم.
به همین ترتیب فایل Scripts\Routes\recent-comments.js را نیز جهت استفاده از data store ویرایش خواهیم کرد:

```
Blogger.RecentCommentsRoute = Ember.Route.extend({
    model: function () {
        return this.store.find('comment');
    }
});
```

و فایل Scripts\Routes\post.js که در آن منطق یافتن یک مطلب بر اساس آدرس مختص به آن قرار دارد، به صورت ذیل بازنویسی می‌شود:

```
Blogger.PostRoute = Ember.Route.extend({
    model: function (params) {
        return this.store.find('post', params.post_id);
    }
});
```

اگر متد find بدون پارامتر ذکر شود، به معنای بازگشت تمامی عناصر موجود در آن مدل خواهد بود و اگر پارامتر دوم آن مانند این مثال تنظیم شود، تنها همان وهله‌ی درخواستی را بازگشت می‌دهد.

افزودن امکان ثبت یک مطلب جدید

تا اینجا اگر برنامه را اجرا کنید، برنامه بدون خطا بارگذاری خواهد شد اما فعلا رکوردی را برای نمایش ندارد. در ادامه، برنامه را جهت افزودن مطالب جدید توسعه خواهیم داد. برای اینکار ابتدا به فایل Scripts\App\router.js مراجعه کرده و سپس مسیریابی جدید new-post را تعریف خواهیم کرد:

```
Blogger.Router.map(function () {
    this.resource('posts', { path: '/' });
    this.resource('about');
    this.resource('contact', function () {
        this.resource('email');
        this.resource('phone');
    });
    this.resource('recent-comments');
    this.resource('post', { path: 'posts/:post_id' });
    this.resource('new-post');
});
```

اکنون در صفحه‌ی اول سایت، توسط قالب Scripts\Templates\posts.hbs، دکمه‌ای را جهت ایجاد یک مطلب جدید اضافه خواهیم کرد:

```
<h2>Ember.js blog</h2>
<ul>
    {{#each post in arrangedContent}}
    <li>{{#link-to 'post' post.id}}{{post.title}}{{/link-to}}</li>
    {{/each}}
</ul>

<a href="#" class="btn btn-primary" {{action 'sortByTitle' }}>Sort by title</a>
{{#link-to 'new-post' classNames="btn btn-success"}}New Post{{/link-to}}
```

در اینجا دکمه‌ی New Post به مسیریابی جدید new-post اشاره می‌کند.
برای تعریف عناصر نمایشی این مسیریابی، فایل جدید قالب Scripts\Templates\new-post.hbs را با محتوای زیر اضافه کنید:

```
<h1>New post</h1>
<form>
  <div class="form-group">
    <label for="title">Title</label>
    {{input value=title id="title" class="form-control"}}
  </div>

  <div class="form-group">
    <label for="body">Body</label>
    {{textarea value=body id="body" class="form-control" rows="5"}}
  </div>


  <button class="btn btn-primary" {{action 'save'}}>Save</button>
</form>
```

با نمونه‌ی این فرم در قسمت قبل در حین ویرایش یک مطلب، آشنا شدیم. دو المان دریافت اطلاعات در آن قرار دارند که هر کدام به خواص مدل برنامه bind شده‌اند. همچنین یک دکمه‌ی save، با اکشنی به همین نام در اینجا تعریف شده‌است.
پس از آن نیاز است نام فایل قالب new-post را به template loader برنامه در فایل index.html اضافه کرد:

```
<script type="text/javascript">
    EmberHandlebarsLoader.loadTemplates([
       'posts', 'about', 'application', 'contact', 'email', 'phone',
       'recent-comments', 'post', 'new-post'
    ]);
</script>
```

برای مدیریت دکمه‌ی save این قالب جدید نیاز است کنترلر جدیدی را در فایل جدید Scripts\Controllers\new-post.js تعریف کنیم؛ با این محتوا:

```
Blogger.NewPostController = Ember.Controller.extend({
    actions: {
        save: function () {
            var newPost = this.store.createRecord('post', {
                title: this.get('title'),
                body: this.get('body')
            });
            newPost.save();
            this.transitionToRoute('posts');
        }
    }
});
```

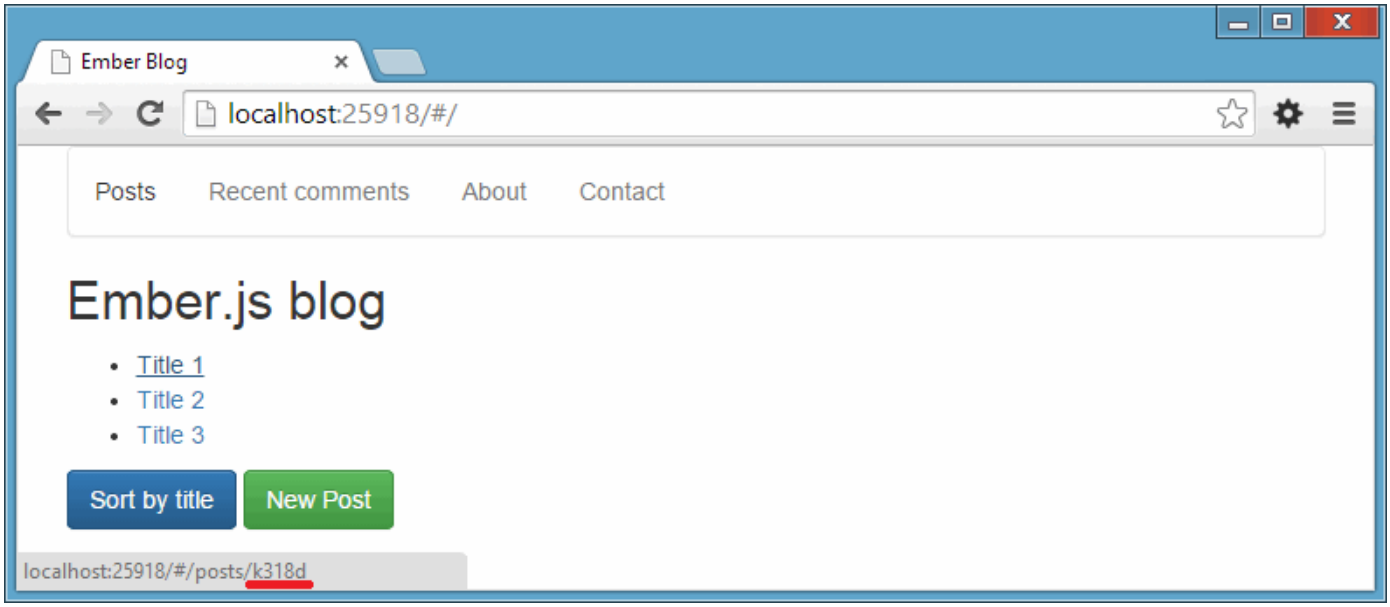
به همراه افزودن مدخلی از آن به فایل index.html برنامه:

```
<script src="Scripts/Controllers/new-post.js" type="text/javascript"></script>
```

در اینجا کنترلر جدید NewPostController را مشاهده می‌کنید. از این جهت که برای دسترسی به خواص مدل تغییر کرده، از متد this.get استفاده شده‌است، نیازی نیست حتما از یک ObjectController مانند قسمت قبل استفاده کرد و Controller معمولی نیز برای اینکار کافی است.
آرگومان اول this.store.createRecord نام مدل است و آرگومان دوم آن، وهله‌ای که قرار است به آن اضافه شود. همچنین باید دقت داشت که برای تنظیم یک خاصیت، از متد this.set و برای دریافت مقدار یک خاصیت تغییر کرده از this.get به همراه نام خاصیت مورد نظر استفاده می‌شود و نباید مستقیما برای مثال از this.title استفاده کرد.
this.store.createRecord صرفا یک شیء جدید (ember data object) را ایجاد می‌کند. برای ذخیره سازی نهایی آن باید متد save آن‌را فراخوانی کرد (پیاده سازی الگوی active record است). به این ترتیب این شیء در local storage ذخیره خواهد شد.
پس از ذخیره‌ی مطلب جدید، از متد this.transitionToRoute استفاده شده‌است. این متد، برنامه را به صورت خودکار به

صفحه‌ی متناظر با مسیریابی posts هدایت می‌کند.

اکنون برنامه را اجرا کنید. بر روی دکمه‌ی سبز رنگ new post در صفحه‌ی اول کلیک کرده و یک مطلب جدید را تعریف کنید. بلافاصله عنوان و لینک متناظر با این مطلب را در صفحه‌ی اول سایت مشاهده خواهید کرد.
همچنین اگر برنامه را مجددا بارگذاری کنید، این مطالب هنوز قابل مشاهده هستند؛ زیرا در local storage مرورگر ذخیره شده‌اند.

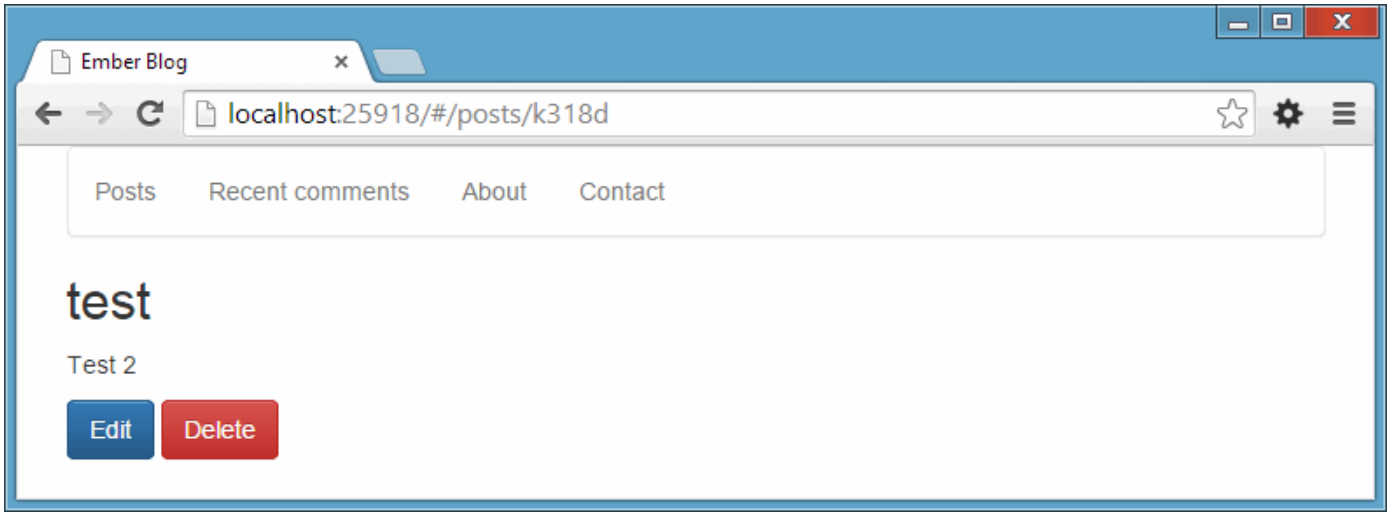


در اینجا اگر به لینک‌های تولید شده دقت کنید، id آن‌ها عددی نیست. این روشی است که local storage با آن کار می‌کند.

افزودن امکان حذف یک مطلب به سایت

برای حذف یک مطلب، دکمه‌ی حذف را به انتهای قالب Scripts\Templates\post.hbs اضافه خواهیم کرد:

```
<h2>{{title}}</h2>
{{#if isEditing}}
<form>
    <div class="form-group">
        <label for="title">Title</label>
        {{input value=title id="title" class="form-control"}}
    </div>
    <div class="form-group">
        <label for="body">Body</label>
        {{textarea value=body id="body" class="form-control" rows="5"}}
    </div>
    <button class="btn btn-primary" {{action 'save' }}>Save</button>
</form>
{{else}}
<p>{{body}}</p>
<button class="btn btn-primary" {{action 'edit' }}>Edit</button>
<button class="btn btn-danger" {{action 'delete' }}>Delete</button>
{{/if}}
```



سپس کنترلر Scripts\Controllers\post.js را جهت مدیریت اکشن جدید delete به نحو ذیل تکمیل می‌کنیم:

```
Blogger.PostController = Ember.ObjectController.extend({
    isEditing: false,
    actions: {
        edit: function () {
            this.set('isEditing', true);
        },
        save: function () {
            var post = this.get('model');
            post.save();

            this.set('isEditing', false);
        },
        delete: function () {
            if (confirm('Do you want to delete this post?')) {
                this.get('model').destroyRecord();
                this.transitionToRoute('posts');
            }
        }
    }
});
```

متد destroyRecord، مدل انتخابی را هم از حافظه و هم از data store حذف می‌کند. سپس کاربر را به صفحه‌ی اصلی سایت هدایت خواهیم کرد.
متد save نیز در اینجا بهبود یافته‌است. ابتدا مدل جاری دریافت شده و سپس متد save بر روی آن فراخوانی می‌شود. به این ترتیب اطلاعات از حافظه به local storage نیز منتقل خواهند شد.

ثبت و نمایش نظرات به همراه تنظیمات روابط اشیاء در Ember Data

در ادامه قصد داریم امکان افزودن نظرات را به مطالب، به همراه نمایش آن‌ها در ذیل هر مطلب، پیاده سازی کنیم. برای اینکار نیاز است رابطه‌ی بین یک مطلب و نظرات مرتبط با آن‌را در مدل ember data مشخص کنیم. به همین جهت فایل Scripts\Models\post.js را گشوده و تغییرات ذیل را به آن اعمال کنید:

```
Blogger.Post = DS.Model.extend({
  title: DS.attr(),
  body: DS.attr(),
  comments: DS.hasMany('comment', { async: true })
});
```

در اینجا خاصیت جدیدی به نام comments به مدل مطلب اضافه شده‌است و توسط آن می‌توان به تمامی نظرات یک مطلب دسترسی یافت؛ تعریف رابطه‌ی یک به چند، به کمک متد DS.hasMany که پارامتر اول آن نام مدل مرتبط است. تعریف async: true

برای کار با local storage اجباری است و در نگارش‌های آتی ember data حالت پیش فرض خواهد بود.
همچنین نیاز است یک سر دیگر رابطه را نیز مشخص کرد. برای این منظور فایل Scripts\Models\comment.js را گشوده و به نحو ذیل تکمیل کنید:

```
Blogger.Comment = DS.Model.extend({
    text: DS.attr(),
    post: DS.belongsTo('post', { async: true })
});
```

در اینجا خاصیت جدید post به مدل نظر اضافه شده‌است و مقدار آن از طریق متد DS.belongsTo که مدل post را به یک نظر، مرتبط می‌کند، تامین خواهد شد. بنابراین در این حالت اگر به شیء comment مراجعه کنیم، خاصیت جدید post.id آن، به id مطلب متناظر اشاره می‌کند.

در ادامه نیاز است بتوان تعدادی نظر را ثبت کرد. به همین جهت با تعریف مسیریابی آن شروع می‌کنیم. این مسیریابی تعریف شده در فایل Scripts\App\router.js نیز باید تو در تو باشد؛ زیرا قسمت ثبت نظر (new-comment) دقیقا داخل همان صفحه‌ی نمایش یک مطلب ظاهر می‌شود:

```
Blogger.Router.map(function () {
    this.resource('posts', { path: '/' });
    this.resource('about');
    this.resource('contact', function () {
        this.resource('email');
        this.resource('phone');
    });
    this.resource('recent-comments');
    this.resource('post', { path: 'posts/:post_id' }, function () {
        this.resource('new-comment');
    });
    this.resource('new-post');
});
```

لینک آن‌را نیز به انتهای فایل Scripts\Templates\post.hbs اضافه می‌کنیم. از این جهت که این لینک به مدل جاری اشاره می‌کند، با استفاده از متغیر this، مدل جاری را به عنوان مدل مورد استفاده مشخص خواهیم کرد:

```
<h2>{{title}}</h2>
{{#if isEditing}}
<form>
    <div class="form-group">
        <label for="title">Title</label>
        {{input value=title id="title" class="form-control"}}
    </div>
    <div class="form-group">
        <label for="body">Body</label>
        {{textarea value=body id="body" class="form-control" rows="5"}}
    </div>
    <button class="btn btn-primary" {{action 'save' }}>Save</button>
</form>
{{else}}
<p>{{body}}</p>
<button class="btn btn-primary" {{action 'edit' }}>Edit</button>
<button class="btn btn-danger" {{action 'delete' }}>Delete</button>
{{/if}}

<h2>Comments</h2>
{{#each comment in comments}}
<p>
    {{comment.text}}
</p>
{{/each}}

<p>{{#link-to 'new-comment' this class="btn btn-success"}}New comment{{/link-to}}</p>
{{outlet}}
```

پس از تکمیل روابط مدل‌ها، قالب Scripts\Templates\post.hbs را جهت استفاده از این خواص به روز خواهیم کرد. در تغییرات جدید، قسمت <h2>Comments</h2> به انتهای صفحه اضافه شده‌است. سپس حلقه‌ای بر روی خاصیت جدید comments تشکیل

شده و مقدار خاصیت text هر آیتم نمایش داده می‌شود.
در انتهای قالب نیز یک {{outlet}} اضافه شده‌است. کار آن نمایش قالب ارسال یک نظر جدید، پس از کلیک بر روی لینک New Comment می‌باشد. این قالب را با افزودن فایل Scripts\Templates\new-comment.hbs با محتوای ذیل ایجاد خواهیم کرد:

```
<h2>New comment</h2>

<form>
  <div class="form-group">
    <label for="text">Your thoughts:</label>
    {{textarea value=text id="text" class="form-control" rows="5"}}
  </div>

  <button class="btn btn-primary" {{action "save"}}>Add your comment</button>
</form>
```

سپس نام این قالب را به template loader فایل index.html نیز اضافه می‌کنیم؛ تا در ابتدای بارگذاری برنامه شناسایی شده و استفاده شود:

```
<script type="text/javascript">
    EmberHandlebarsLoader.loadTemplates([
       'posts', 'about', 'application', 'contact', 'email', 'phone',
       'recent-comments', 'post', 'new-post', 'new-comment'
    ]);
</script>
```

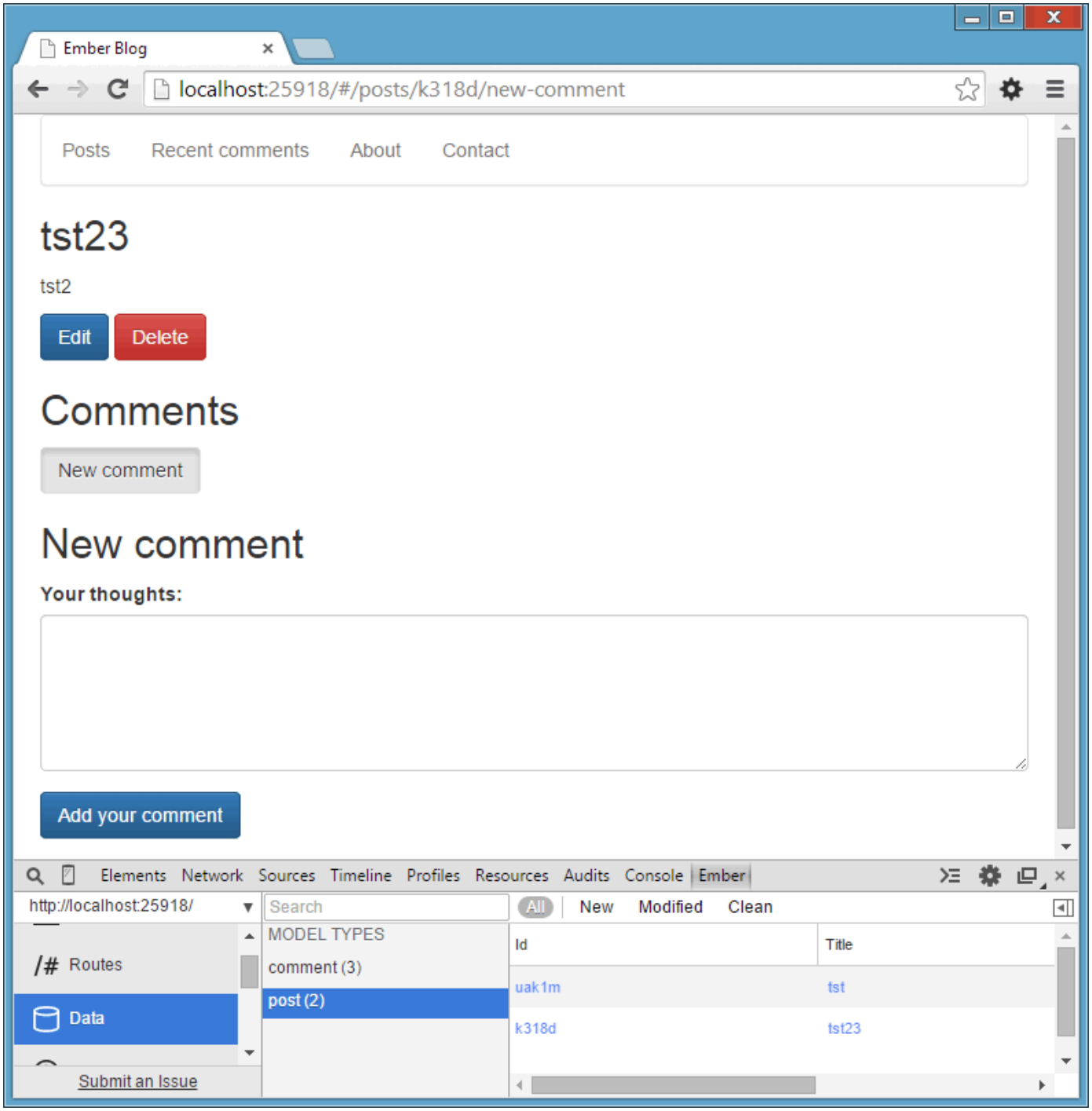
این قالب به خاصیت text یک comment متصل بوده و همچنین اکشن جدیدی به نام save دارد. بنابراین برای مدیریت اکشن save، نیاز به کنترلری متناظر خواهد بود. به همین جهت فایل جدید Scripts\Controllers\new-comment.js را با محتوای ذیل ایجاد کنید:

```
Blogger.NewCommentController = Ember.ObjectController.extend({
    needs: ['post'],
    actions: {
        save: function () {
            var comment = this.store.createRecord('comment', {
                text: this.get('text')
            });
            comment.save();

            var post = this.get('controllers.post.model');
            post.get('comments').pushObject(comment);
            post.save();

            this.transitionToRoute('post', post.id);
        }
    }
});
```

و مدخل تعریف آن‌را نیز به صفحه‌ی index.html اضافه می‌کنیم:

```
<script src="Scripts/Controllers/new-comment.js" type="text/javascript"></script>
```

قسمت ذخیره سازی comment جدید با ذخیره سازی یک post جدید که پیشتر بررسی کردیم، تفاوتی ندارد. از متد this.store.createRecord جهت معرفی وهله‌ای جدید از comment استفاده و سپس متد save آن، برای ثبت نهایی فراخوانی شده‌است.
در ادامه باید این نظر جدید را به post متناظر با آن مرتبط کنیم. برای اینکار نیاز است تا به مدل کنترلر post دسترسی داشته باشیم. به همین جهت خاصیت needs را به تعاریف کنترلر جاری به همراه نام کنترلر مورد نیاز، اضافه کرده‌ایم. به این ترتیب می‌توان توسط متد this.get و پارامتر controllers.post.model در کنترلر NewComment به اطلاعات کنترلر post دسترسی یافت. سپس خاصیت comments شیء post جاری را یافته و مقدار آن‌را به comment جدیدی که ثبت کردیم، تنظیم می‌کنیم. در ادامه با فراخوانی متد save، کار تنظیم ارتباطات یک مطلب و نظرهای جدید آن به پایان می‌رسد.
در آخر با فراخوانی متد transitionToRoute به مطلبی که نظر جدیدی برای آن ارسال شده‌است باز می‌گردیم.



همانطور که در این تصویر نیز مشاهده می‌کنید، اطلاعات ذخیره شده در local storage را توسط افزونه‌ی Ember Inspector نیز می‌توان مشاهده کرد.

افزودن دکمه‌ی حذف به لیست نظرات ارسالی

برای افزودن دکمه‌ی حذف، به قالب Scripts\Templates\post.hbs مراجعه کرده و قسمتی را که لیست نظرات را نمایش می‌دهد، به نحو ذیل تکمیل می‌کنیم:

```
{{#each comment in comments}}
```

```
<p>
    {{comment.text}}
    <button class="btn btn-xs btn-danger" {{action 'delete' }}>delete</button>
</p>
{{/each}}
```

همچنین برای مدیریت اکشن جدید delete، کنترلر جدید comment را در فایل Scripts\Controllers\comment.js اضافه خواهیم کرد.

```
Blogger.CommentController = Ember.ObjectController.extend({
    needs: ['post'],
    actions: {
        delete: function () {
            if (confirm('Do you want to delete this comment?')) {
                var comment = this.get('model');
                comment.deleteRecord();
                comment.save();

                var post = this.get('controllers.post.model');
                post.get('comments').removeObject(comment);
                post.save();
            }
        }
    }
});
```

به همراه تعریف مدخل آن در فایل index.html :

```
<script src="Scripts/Controllers/comment.js" type="text/javascript"></script>
```

در این حالت اگر برنامه را اجرا کنید، پیام «Do you want to delete this **post**» را مشاهده خواهید کرد بجای پیام «Do you want to delete this **comment**». علت اینجا است که قالب post به صورت پیش فرض به کنترلر post متصل است و نه کنترلر comment. برای رفع این مشکل تنها کافی است از itemController به نحو ذیل استفاده کنیم:

```
{{#each comment in comments  itemController="comment"}}
<p>
    {{comment.text}}
    <button class="btn btn-xs btn-danger" {{action 'delete' }}>delete</button>
</p>
{{/each}}
```

به این ترتیب اکشن delete به کنترلر comment ارسال خواهد شد و نه کنترلر پیش فرض post جاری.
در کنترلر Comment روش دیگری را برای حذف یک رکورد مشاهده می‌کنید. می‌توان ابتدا متد deleteRecord را بر روی مدل فراخوانی کرد و سپس آن‌را save نمود تا نهایی شود. همچنین در اینجا نیاز است نظر حذف شده را از سر دیگر رابطه نیز حذف کرد. روش دسترسی به post جاری در این حالت، همانند توضیحات NewCommentController است که پیشتر بحث شد.

کدهای کامل این قسمت را از اینجا می‌توانید دریافت کنید:

EmberJS03_04.zip