

master

...

lightning / app / controllers / session.js / &lt;&gt; Jump to

mathisonian cleanup sockets

History

2 contributors

692 lines (562 sloc) 22 KB

```
1
2  /*!
3   * Module dependencies.
4   */
5   var debug = require('debug')('lightning:server:controllers:session');
6   var _ = require('lodash');
7   var multipart = require('multipart');
8   var knex = require('knex');
9   var randomstring = require('randomstring');
10  var path = require('path');
11  var easyimage = require('easyimage');
12  var async = require('async');
13  var models = require('../models');
14  var Q = require('q');
15  var commandExists = require('command-exists');
16  var config = require('../../config/config');
17  var fs = require('fs-extra');
18
19
20
21  exports.index = function (req, res, next) {
22
23    models.Session.findAll({
24      order: '"createdAt" DESC'
25    }).then(function(sessions) {
26      res.render('session/index', {
27        sessions: sessions
28      });
29    }).catch(next);
30  };
31
32
33  exports.feed = function (req, res, next) {
34
35    var sessionId = req.params.sid;
36    var Session = models.Session;
37    var VisualizationType = models.VisualizationType;
38    var Visualization = models.Visualization;
39
40    Q.all([
41      Session
42        .find({
43          where: {
44            id: sessionId
45          }
46        }),
47      Visualization
48        .findAll({
49          where: {
50            SessionId: sessionId
51          },
52          include: [VisualizationType]
53        }),
54      VisualizationType.findAll({
55        order: '"name" ASC'
56      })
57    ]).spread(function(session, visualizations, vizTypes) {
58
59      if(!session) {
60        return res.status(404).send('Session not found');
61      }
62      res.render('session/feed', {
63        session: session,
64        visualizations: visualizations,
65        vizTypes: _.object(_.map(vizTypes, function(item) {
66          return [item.name, item];
67        })))
68    });
69    }).fail(next);
70  };
71
72
73  exports.listVisualizations = function (req, res, next) {
74
75    var sessionId = req.params.sid;
76    var Visualization = models.Visualization;
77
78    Visualization
```

```

79     .findAll({
80         where: {
81             SessionId: sessionId
82         }
83     })
84     .then(function(visualizations) {
85         return res.json(visualizations)
86     })
87     .catch(next);
88 };
89
90
91 exports.publicRead = function (req, res, next) {
92
93     var sessionId = req.params.sid;
94     var Session = models.Session;
95     var VisualizationType = models.VisualizationType;
96     var Visualization = models.Visualization;
97
98     Q.all([
99         Session
100         .find({
101             where: {
102                 id: sessionId
103             }
104         }),
105         Visualization
106         .findAll({
107             where: {
108                 SessionId: sessionId
109             }
110         }),
111         VisualizationType.findAll({
112             order: '"name" ASC'
113         })
114     ]).spread(function(session, visualizations, vizTypes) {
115
116         if(!session) {
117             return res.status(404).send('Session not found');
118         }
119
120         res.render('session/feed-public', {
121             session: session,
122             visualizations: visualizations,
123             vizTypes: _.object(_.map(vizTypes, function(item) {
124                 return [item.name, item];
125             })))
126     });
127     }).fail(next);
128 };
129
130
131 exports.update = function (req, res, next) {
132
133     var sessionId = req.params.sid;
134     var Session = models.Session;
135
136
137     Session
138     .update(req.body, {
139         where: {
140             id: sessionId
141         }
142     }).then(function(sessions) {
143         return res.json(sessions);
144     }).catch(next);
145
146 };
147
148
149
150 exports.create = function(req, res, next) {
151     models.Session
152     .create(_.pick(req.body, 'name'))
153     .then(function(session) {
154         req.io.of(session.getSocketNamespace())
155         .emit('init');
156         return res.json(session);
157     }).catch(next);
158 };
159
160 exports.getCreate = function(req, res, next) {
161     models.Session
162     .create()
163     .then(function(session) {
164         req.io.of(session.getSocketNamespace())
165         .emit('init');
166         return res.redirect(config.baseUrl + 'sessions/' + session.id + '/feed/');
167     }).catch(next);
168 };
169
170 exports.delete = function(req, res, next) {
171     var sessionId = req.params.sid;
172
173     models.Session
174     .findById(sessionId)
175     .then(function(session) {
176         session.destroy({where: {}}).then(function() {

```



```

275         req.io.of(viz.getSessionSocketNamespace())
276         .emit('viz', jsonViz);
277         return res.json(jsonViz);
278     }).catch(function(err) {
279         if(err.message && err.message === 'Unknown Viztype') {
280             return res.status(404).send('Could not find viz type ' + type);
281         }
282         return next(err);
283     });
284 });
285 });
286 });
287 }
288 };
289
290
291
292 exports.appendData = function (req, res, next) {
293     var sessionId = req.params.sid;
294     var vizId = req.params.viz;
295     var fieldName = req.params.field;
296     var VisualizationType = models.VisualizationType;
297     debug(req.params);
298
299     models.Visualization
300     .find({
301         where: {
302             id: vizId
303         },
304         include: [VisualizationType]
305     }).then(function(viz) {
306         if(req.is('json')) {
307             if(fieldName) {
308
309                 if(_.isArray(viz.data[fieldName])) {
310                     if(viz.VisualizationType.isStreaming) {
311                         _.each(req.body.data, function(d, i) {
312                             if(i < viz.data[fieldName].length) {
313                                 viz.data[fieldName][i] = viz.data[fieldName][i].concat(d);
314                             }
315                         });
316                     } else {
317                         var vdata = viz.data;
318                         vdata[fieldName].push(req.body.data);
319                         viz.data = vdata;
320                     }
321
322                 } else if(_.isUndefined(viz.data[fieldName])) {
323                     viz.data[fieldName] = req.body.data;
324                 } else {
325                     debug('unknown field');
326                 }
327             }
328         } else {
329             // debug(viz.data);
330             debug(req.body.data);
331             if(_.isArray(viz.data)) {
332                 if(_.isArray(req.body.data)) {
333
334                     if(viz.VisualizationType.isStreaming) {
335                         _.each(req.body.data, function(d, i) {
336
337                             });
338                     } else {
339                         viz.data = viz.data.concat(req.body.data);
340                     }
341
342                 } else {
343                     var vdata = viz.data;
344                     vdata.push(req.body.data);
345                     viz.data = vdata;
346                 }
347             }
348         } else if(_.isUndefined(viz.data)) {
349             viz.data = req.body.data;
350         } else {
351             debug('unknown field');
352         }
353     })
354
355     viz
356     .save()
357     .then(function() {
358         return res.json(viz);
359     }).catch(next);
360
361     console.log(viz.getSessionSocketNamespace());
362     req.io.of(viz.getSessionSocketNamespace())
363     .emit('append', {
364         vizId: viz.id,
365         data: req.body.data
366     });
367
368     } else if(fieldName === 'images') {
369
370         var form = new multiparty.Form();
371
372         form.parse(req, function(err, fields, files) {

```

```

373     _._each(files, function(f) {
374
375         thumbnailAndUpload(f, sessionId, function(err, data) {
376
377             if(err) {
378                 debug('error in thumbnailAndUpload');
379                 return res.status(500).send('error creating image thumbnail');
380             }
381             var imgData = data.imgData;
382             var s3Response = data.response;
383
384             if(viz.images) {
385                 var vimages = viz.images;
386                 vimages.push(imgData);
387                 viz.images = vimages;
388             } else {
389                 viz.images = [imgData];
390             }
391
392             viz
393                 .save()
394                 .then(function() {
395                     if(typeof s3Response === 'object') {
396                         res.statusCode = s3Response.statusCode;
397                         s3Response.pipe(res);
398                     } else {
399                         return res.status(data.response).send('');
400                     }
401                 });
402
403             req.io.of(viz.getSessionSocketNamespace())
404                 .emit('append', {
405                     vizId: viz.id,
406                     data: imgData
407                 });
408
409             });
410         });
411     });
412     } else {
413         return next(500);
414     }
415     }).catch(next);
416 };
417
418
419
420 exports.updateData = function (req, res, next) {
421
422     var sessionId = req.params.sid;
423     var vizId = req.params.viz;
424     var fieldName = req.params.field;
425
426
427     models.Visualization
428         .findById(vizId)
429         .then(function(viz) {
430             if(req.is('json')) {
431
432                 if(fieldName) {
433                     viz.data[fieldName] = req.body.data;
434                 } else {
435                     viz.data = req.body.data;
436                 }
437
438                 viz
439                     .save()
440                     .then(function() {
441                         return res.json(viz);
442                     }).catch(next);
443
444                 req.io.of(viz.getSessionSocketNamespace())
445                     .emit('update', {
446                         vizId: viz.id,
447                         data: req.body.data
448                     });
449
450             } else if(fieldName === 'images') {
451
452                 var form = new multiparty.Form();
453
454                 form.parse(req, function(err, fields, files) {
455                     _._each(files, function(f) {
456                         thumbnailAndUpload(f, sessionId, function(err, data) {
457
458                             if(err) {
459                                 debug('error in thumbnailAndUpload');
460                                 return res.status(500).send('error creating image thumbnail');
461                             }
462                             var imgData = data.imgData;
463                             var s3Response = data.response;
464
465                             viz.images = [imgData];
466
467                             viz
468                                 .save()
469                                 .then(function() {

```

```

471         if(typeof s3Response === 'object') {
472             res.statusCode = s3Response.statusCode;
473             s3Response.pipe(res);
474         } else {
475             return res.status(data.response).send('');
476         }
477     });
478
479     req.io.of(viz.getSessionSocketNamespace())
480         .emit('update', {
481             vizId: viz.id,
482             data: imgData
483         });
484
485     });
486     });
487     });
488     } else {
489         return next(500);
490     }
491
492     }).catch(next);
493
494
495 };
496
497
498 var thumbnailAndUpload = function(f, sessionId, callback) {
499
500
501
502
503     var staticUrl = '/';
504     if(config.url) {
505         staticUrl = 'http://' + config.url + '/';
506     }
507
508
509     // check if thumbnailing exists,
510     // and if s3 creds exist
511     var s3Exists = !!config.s3.key;
512     var s3Client = null;
513
514     if(s3Exists) {
515
516         s3Client = knox.createClient({
517             secure: false,
518             key: process.env.S3_KEY,
519             secret: process.env.S3_SECRET,
520             bucket: process.env.S3_BUCKET,
521         });
522     }
523
524     var maxWidth = 500;
525     var maxHeight = 500;
526
527     // Image file info
528     var imgPath = f[0].path;
529     var extension = path.extname(imgPath).toLowerCase();
530     var filenameWithoutExtension = path.basename(imgPath, extension);
531
532
533     var thumbnailPath;
534
535     if(process.env.NODE_ENV === 'production') {
536         thumbnailPath = path.resolve(__dirname + '/../..' + './tmp/' + filenameWithoutExtension + '_thumbnail' + extension);
537     } else {
538         thumbnailPath = path.dirname(imgPath) + '/' + filenameWithoutExtension + '_thumbnail' + extension;
539     }
540
541     // Upload paths for s3
542     var uploadName = randomstring.generate();
543     var destPath = '/sessions/' + sessionId + '/';
544     var originalS3Path = destPath + uploadName;
545     var thumbnailS3Path = destPath + uploadName + '_small';
546
547     // s3 headers
548     var headers = {
549         'x-amz-acl': 'public-read',
550         'Access-Control-Allow-Origin': '*',
551     };
552     if( extension === '.jpg' || extension === '.jpeg' ) {
553         headers['Content-Type'] = 'image/jpeg';
554     } else if (extension === '.png') {
555         headers['Content-Type'] = 'image/png';
556     }
557
558     commandExists('identify', function(err, imageMagickExists) {
559
560         if(imageMagickExists) {
561             easyimage
562                 .info(imgPath)
563                 .then(function(file) {
564                     var thumbWidth;
565                     var thumbHeight;
566
567                     debug('outputing to: ' + thumbnailPath);
568

```

```

569         if(file.width > file.height) {
570             thumbWidth = Math.min(maxWidth, file.width);
571             thumbHeight = file.height * (thumbWidth / file.width);
572         } else {
573             thumbHeight = Math.min(maxHeight, file.height);
574             thumbWidth = file.width * (thumbHeight / file.height);
575         }
576
577         return easyimage.resize({
578             src: imagePath,
579             dst: thumbnailPath,
580             width: thumbWidth,
581             height: thumbHeight
582         });
583     }).then(function() {
584
585         if(s3Exists) {
586             async.parallel([
587                 function(callback) {
588                     s3Client.putFile(imgPath, originalS3Path, headers, callback);
589                 },
590                 function(callback) {
591                     s3Client.putFile(thumbnailPath, thumbnailS3Path, headers, callback);
592                 }
593             ], function(err, results) {
594                 var s3Response = results[0];
595
596                 var imgURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + originalS3Path;
597                 // var thumbURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + thumbnailS3Path;
598
599                 var imgData = imgURL;
600
601                 callback(null, {
602                     response: s3Response,
603                     imgData: imgData
604                 });
605             });
606         } else {
607
608             debug('S3 Credentials not found. Using local images');
609
610             async.parallel([
611                 function(callback) {
612                     var outputPath = path.resolve(__dirname + '/../../public/images/uploads' + originalS3Path);
613                     fs.copy(imgPath, outputPath, callback);
614                 },
615                 function(callback) {
616                     var outputPath = path.resolve(__dirname + '/../../public/images/uploads' + thumbnailS3Path);
617                     fs.copy(thumbnailPath, outputPath, callback);
618                 }
619             ], function(err) {
620                 if(err) {
621                     return callback(err);
622                 }
623
624                 return callback(null, {
625                     response: 200,
626                     imgData: staticUrl + 'images/uploads' + originalS3Path
627                 });
628             });
629         }
630     }
631
632     }, function(err) {
633         debug(err);
634         callback(err);
635     });
636 } else {
637
638     if(s3Exists) {
639         async.parallel([
640             function(callback) {
641                 debug(imgPath + ':' + originalS3Path);
642                 s3Client.putFile(imgPath, originalS3Path, headers, callback);
643             },
644             function(callback) {
645                 debug(thumbnailPath + ':' + thumbnailS3Path);
646                 s3Client.putFile(thumbnailPath, thumbnailS3Path, headers, callback);
647             }
648         ], function(err, results) {
649             var s3Response = results[0];
650
651             var imgURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + originalS3Path;
652             // var thumbURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + thumbnailS3Path;
653
654             var imgData = imgURL;
655
656             callback(null, {
657                 response: s3Response,
658                 imgData: imgData
659             });
660         });
661     } else {
662
663         console.log('S3 Credentials not found. Using local images');
664
665         async.parallel([

```

```
667     function(callback) {
668         var outpath = path.resolve(__dirname + '../../public/images/uploads' + originalS3Path);
669         debug(outpath);
670         fs.copy(imgPath, outpath, callback);
671     },
672     function(callback) {
673         var outpath = path.resolve(__dirname + '../../public/images/uploads' + thumbnailS3Path);
674         debug(outpath);
675         fs.copy(imgPath, outpath, callback);
676     }
677 ], function(err) {
678     if(err) {
679         return callback(err);
680     }
681
682     return callback(null, {
683         response: 200,
684         imgData: staticUrl + 'images/uploads' + originalS3Path
685     });
686 });
687 }
688 }
689 })
690
691
692 };
```