lightning / app / controllers / session.js / Jump to

mathisonian cleanup sockets History

2 contributors

692 lines (562 sloc) | 22 KB

```javascript

/*!
 * Module dependencies.
 */
var debug = require('debug')('lightning:server:controllers:session');
var _ = require('lodash');
var multiparty = require('multiparty');
var knox = require('knox');
var randomstring = require('randomstring');
var path = require('path');
var easyimage = require('easyimage');
var async = require('async');
var models = require('../models');
var Q = require('q');
var commandExists = require('command-exists');
var config = require('../../config/config');
var fs = require('fs-extra');



exports.index = function (req, res, next) {

    models.Session.findAll({
        order: '"createdAt" DESC'
    }).then(function(sessions) {
        res.render('session/index', {
            sessions: sessions
        });
    }).catch(next);
};


exports.feed = function (req, res, next) {

    var sessionId = req.params.sid;
    var Session = models.Session;
    var VisualizationType = models.VisualizationType;
    var Visualization = models.Visualization;

    Q.all([
        Session
            .find({
                where: {
                    id: sessionId
                }
            }),
        Visualization
            .findAll({
                where: {
                    SessionId: sessionId
                },
                include: [VisualizationType]
            }),
        VisualizationType.findAll({
            order: '"name" ASC'
        })
    ]).spread(function(session, visualizations, vizTypes) {

        if(!session) {
            return res.status(404).send('Session not found');
        }
        res.render('session/feed', {
            session: session,
            visualizations: visualizations,
            vizTypes: _.object(_.map(vizTypes, function(item) {
                return [item.name, item];
            }))
        });
    }).fail(next);
};


exports.listVisualizations = function (req, res, next) {

    var sessionId = req.params.sid;
    var Visualization = models.Visualization;

    Visualization
```

```
        .findAll({
            where: {
                SessionId: sessionId
            }
        })
        .then(function(visualizations) {
            return res.json(visualizations)
        })
        .catch(next);
};


exports.publicRead = function (req, res, next) {

    var sessionId = req.params.sid;
    var Session = models.Session;
    var VisualizationType = models.VisualizationType;
    var Visualization = models.Visualization;

    Q.all([
        Session
            .find({
                where: {
                    id: sessionId
                }
            }),
        Visualization
            .findAll({
                where: {
                    SessionId: sessionId
                }
            }),
        VisualizationType.findAll({
            order: '"name" ASC'
        })
    ]).spread(function(session, visualizations, vizTypes) {

        if(!session) {
            return res.status(404).send('Session not found');
        }

        res.render('session/feed-public', {
            session: session,
            visualizations: visualizations,
            vizTypes: _.object(_.map(vizTypes, function(item) {
                return [item.name, item];
            }))
        });
    }).fail(next);
};


exports.update = function (req, res, next) {

    var sessionId = req.params.sid;
    var Session = models.Session;


    Session
        .update(req.body, {
            where: {
                id: sessionId
            }
        }).then(function(sessions) {
            return res.json(sessions);
        }).catch(next);

};



exports.create = function(req, res, next) {
    models.Session
        .create(_.pick(req.body, 'name'))
        .then(function(session) {
            req.io.of(session.getSocketNamespace())
                .emit('init');
            return res.json(session);
        }).catch(next);
};

exports.getCreate = function(req, res, next) {
    models.Session
        .create()
        .then(function(session) {
            req.io.of(session.getSocketNamespace())
                .emit('init');
            return res.redirect(config.baseURL + 'sessions/' + session.id + '/feed/');
        }).catch(next);
};

exports.delete = function(req, res, next) {
    var sessionId = req.params.sid;

    models.Session
        .findById(sessionId)
        .then(function(session) {
            session.destroy({where: {}}).then(function() {
                return res.json(session);
            }).catch(next);
        }).catch(next);
};



exports.getDelete = function(req, res, next) {

    var sessionId = req.params.sid;

    models.Session
        .findById(sessionId)
        .then(function(session) {
            session.destroy({where: {}}).then(function() {
                return res.redirect(config.baseURL + 'sessions/');
            }).catch(next);
        }).catch(next);
};



exports.addData = function (req, res, next) {
    var sessionId = req.params.sid;

    var Visualization = models.Visualization;
    var vt;

    if(req.is('json')) {
        models.VisualizationType.find({
            where: {
                name: req.body.type
            }
        }).then(function(vizType) {
            if(!vizType) {
                throw new Error('Unknown Viztype');
            }
            vt = vizType;
            return Visualization.create({
                data: req.body.data,
                opts: req.body.options,
                description: req.body.description,
                SessionId: sessionId,
                VisualizationTypeId: vizType.id
            });
        }).then(function(viz) {
            var jsonViz = viz.toJSON();
            jsonViz.visualizationType = _.pick(vt.toJSON(), 'name', 'moduleName', 'initialDataFields', 'isStreaming', 'id');
            req.io.of(viz.getSessionSocketNamespace())
                .emit('viz', jsonViz);
            return res.json(jsonViz);
        }).catch(function(err) {
            if(err.message && err.message === 'Unknown Viztype') {
                return res.status(404).send('Could not find viz type ' + req.body.type);
            }
            return next(err);
        });
    } else {
        var form = new multiparty.Form();

        form.parse(req, function(err, fields, files) {

            _.each(files, function(f) {
                thumbnailAndUpload(f, sessionId, function(err, data) {

                    if(err) {
                        debug('error in thumbnailAndUpload');
                        return res.status(500).send('error creating image thumbnail');
                    }
                    var imgData = data.imgData;
                    var type = 'image';
                    if(fields.type) {
                        if(_.isArray(fields.type) || _.isObject(fields.type)) {
                            type = fields.type[0];
                        } else {
                            type = fields.type;
                        }
                    }

                    models.VisualizationType.find({
                        where: {
                            name: type
                        }
                    }).then(function(vizType) {
                        if(!vizType) {
                            throw new Error('Unknown Viztype');
                        }
                        vt = vizType;
                        return Visualization.create({
                            images: [imgData],
                            opts: JSON.parse(fields.options || '{}'),
                            SessionId: sessionId,
                            description: req.body.description,
                            VisualizationTypeId: vizType.id
                        });
                    }).then(function(viz) {
                        var jsonViz = viz.toJSON();
                        jsonViz.visualizationType = vt;
                        req.io.of(viz.getSessionSocketNamespace())
                            .emit('viz', jsonViz);
                        return res.json(jsonViz);
                    }).catch(function(err) {
                        if(err.message && err.message === 'Unknown Viztype') {
                            return res.status(404).send('Could not find viz type ' + type);
                        }
                        return next(err);
                    });
                });
            });
        });
    }
};



exports.appendData = function (req, res, next) {
    var sessionId = req.params.sid;
    var vizId = req.params.vid;
    var fieldName = req.params.field;
    var VisualizationType = models.VisualizationType;
    debug(req.params);

    models.Visualization
        .find({
            where: {
                id: vizId
            },
            include: [VisualizationType]
        }).then(function(viz) {
            if(req.is('json')) {
                if(fieldName) {

                    if(_.isArray(viz.data[fieldName])) {
                        if(viz.VisualizationType.isStreaming) {
                            _.each(req.body.data, function(d, i) {
                                if(i < viz.data[fieldName].length) {
                                    viz.data[fieldName][i] = viz.data[fieldName][i].concat(d);
                                }
                            });
                        } else {
                            var vdata = viz.data;
                            vdata[fieldName].push(req.body.data);
                            viz.data = vdata;
                        }


                    } else if(_.isUndefined(viz.data[fieldName])) {
                        viz.data[fieldName] = req.body.data;
                    } else {
                        debug('unknown field');
                    }
                } else {
                    // debug(viz.data);
                    debug(req.body.data);
                    if(_.isArray(viz.data)) {
                        if(_.isArray(req.body.data)) {

                            if(viz.VisualizationType.isStreaming) {
                                _.each(req.body.data, function(d, i) {

                                });
                            } else {
                                viz.data = viz.data.concat(req.body.data);
                            }


                        } else {
                            var vdata = viz.data;
                            vdata.push(req.body.data);
                            viz.data = vdata;
                        }
                    } else if(_.isUndefined(viz.data)) {
                        viz.data = req.body.data;
                    } else {
                        debug('unknown field');
                    }
                }

                viz
                    .save()
                    .then(function() {
                        return res.json(viz);
                    }).catch(next);

                console.log(viz.getSessionSocketNamespace());
                req.io.of(viz.getSessionSocketNamespace())
                    .emit('append', {
                        vizId: viz.id,
                        data: req.body.data
                    });

            } else if(fieldName === 'images') {

                var form = new multiparty.Form();

                form.parse(req, function(err, fields, files) {
                    _.each(files, function(f) {

                        thumbnailAndUpload(f, sessionId, function(err, data) {

                            if(err) {
                                debug('error in thumbnailAndUpload');
                                return res.status(500).send('error creating image thumbnail');
                            }
                            var imgData = data.imgData;
                            var s3Response = data.response;

                            if(viz.images) {
                                var vimages = viz.images;
                                vimages.push(imgData);
                                viz.images = vimages;
                            } else {
                                viz.images = [imgData];
                            }

                            viz
                                .save()
                                .then(function() {
                                    if(typeof s3Response === 'object') {
                                        res.statusCode = s3Response.statusCode;
                                        s3Response.pipe(res);
                                    } else {
                                        return res.status(data.response).send('');
                                    }
                                });

                            req.io.of(viz.getSessionSocketNamespace())
                                .emit('append', {
                                    vizId: viz.id,
                                    data: imgData
                                });

                        });
                    });
                });
            } else {
                return next(500);
            }
        }).catch(next);
};



exports.updateData = function (req, res, next) {

    var sessionId = req.params.sid;
    var vizId = req.params.vid;
    var fieldName = req.params.field;


    models.Visualization
        .findById(vizId)
        .then(function(viz) {
            if(req.is('json')) {

                if(fieldName) {
                    viz.data[fieldName] = req.body.data;
                } else {
                    viz.data = req.body.data;
                }

                viz
                    .save()
                    .then(function() {
                        return res.json(viz);
                    }).catch(next);

                req.io.of(viz.getSessionSocketNamespace())
                    .emit('update', {
                        vizId: viz.id,
                        data: req.body.data
                    });

            } else if(fieldName === 'images') {

                var form = new multiparty.Form();

                form.parse(req, function(err, fields, files) {
                    _.each(files, function(f) {
                        thumbnailAndUpload(f, sessionId, function(err, data) {

                            if(err) {
                                debug('error in thumbnailAndUpload');
                                return res.status(500).send('error creating image thumbnail');
                            }
                            var imgData = data.imgData;
                            var s3Response = data.response;

                            viz.images = [imgData];

                            viz
                                .save()
                                .then(function() {

                                    if(typeof s3Response === 'object') {
                                        res.statusCode = s3Response.statusCode;
                                        s3Response.pipe(res);
                                    } else {
                                        return res.status(data.response).send('');
                                    }
                                });

                            req.io.of(viz.getSessionSocketNamespace())
                                .emit('update', {
                                    vizId: viz.id,
                                    data: imgData
                                });

                        });
                    });
                });
            } else {
                return next(500);
            }


        }).catch(next);

};



var thumbnailAndUpload = function(f, sessionId, callback) {



    var staticUrl = '/';
    if(config.url) {
        staticUrl = 'http://' + config.url + '/';
    }


    // check if thumbnailing exists,
    // and if s3 creds exist
    var s3Exists = !!config.s3.key;
    var s3Client = null;

    if(s3Exists) {

        s3Client = knox.createClient({
            secure: false,
            key: process.env.S3_KEY,
            secret: process.env.S3_SECRET,
            bucket: process.env.S3_BUCKET,
        });
     }

    var maxWidth = 500;
    var maxHeight = 500;

    // Image file info
    var imgPath = f[0].path;
    var extension = path.extname(imgPath).toLowerCase();
    var filenameWithoutExtension = path.basename(imgPath, extension);


    var thumbnailPath;

    if(process.env.NODE_ENV === 'production') {
        thumbnailPath = path.resolve(__dirname + '/../../' + './tmp/' + filenameWithoutExtension + '_thumbnail' + extension);
    } else {
        thumbnailPath = path.dirname(imgPath) + '/' + filenameWithoutExtension + '_thumbnail' + extension;
    }

    // Upload paths for s3
    var uploadName = randomstring.generate();
    var destPath = '/sessions/' + sessionId + '/';
    var originalS3Path = destPath + uploadName;
    var thumbnailS3Path = destPath + uploadName + '_small';

    // s3 headers
    var headers = {
      'x-amz-acl': 'public-read',
      'Access-Control-Allow-Origin': '*',
    };
    if( extension === '.jpg' || extension === '.jpeg' ) {
        headers['Content-Type'] = 'image/jpeg';
    } else if (extension === '.png') {
        headers['Content-Type'] = 'image/png';
    }

    commandExists('identify', function(err, imageMagickExists) {

        if(imageMagickExists) {
            easyimage
                .info(imgPath)
                .then(function(file) {
                    var thumbWidth;
                    var thumbHeight;

                    debug('outputing to: ' + thumbnailPath);

                    if(file.width > file.height) {
                        thumbWidth = Math.min(maxWidth, file.width);
                        thumbHeight = file.height * (thumbWidth / file.width);
                    } else {
                        thumbHeight = Math.min(maxHeight, file.height);
                        thumbWidth = file.width * (thumbHeight / file.height);
                    }

                    return easyimage.resize({
                        src: imgPath,
                        dst: thumbnailPath,
                        width: thumbWidth,
                        height: thumbHeight
                    });
                }).then(function() {

                    if(s3Exists) {
                        async.parallel([
                            function(callback) {
                                s3Client.putFile(imgPath, originalS3Path, headers, callback);
                            },
                            function(callback) {
                                s3Client.putFile(thumbnailPath, thumbnailS3Path, headers, callback);
                            }
                        ], function(err, results) {
                            var s3Response = results[0];

                            var imgURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + originalS3Path;
                            // var thumbURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + thumbnailS3Path;

                            var imgData = imgURL;

                            callback(null, {
                                response: s3Response,
                                imgData: imgData
                            });

                        });
                    } else {

                        debug('S3 Credentials not found. Using local images');

                        async.parallel([
                            function(callback) {
                                var outpath = path.resolve(__dirname + '/../../public/images/uploads' + originalS3Path);
                                fs.copy(imgPath, outpath, callback);
                            },
                            function(callback) {
                                var outpath = path.resolve(__dirname + '/../../public/images/uploads' + thumbnailS3Path);
                                fs.copy(thumbnailPath, outpath, callback);
                            }
                        ], function(err) {
                            if(err) {
                                return callback(err);
                            }

                            return callback(null, {
                                response: 200,
                                imgData: staticUrl + 'images/uploads' + originalS3Path
                            });
                        });
                    }

                }, function(err) {
                    debug(err);
                    callback(err);
                });
        } else {

            if(s3Exists) {
                async.parallel([
                    function(callback) {
                        debug(imgPath + ':' + originalS3Path);
                        s3Client.putFile(imgPath, originalS3Path, headers, callback);
                    },
                    function(callback) {
                        debug(thumbnailPath + ':' + thumbnailS3Path);
                        s3Client.putFile(thumbnailPath, thumbnailS3Path, headers, callback);
                    }
                ], function(err, results) {
                    var s3Response = results[0];

                    var imgURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + originalS3Path;
                    // var thumbURL = 'https://s3.amazonaws.com/' + process.env.S3_BUCKET + thumbnailS3Path;

                    var imgData = imgURL;

                    callback(null, {
                        response: s3Response,
                        imgData: imgData
                    });

                });
            } else {

                console.log('S3 Credentials not found. Using local images');

                async.parallel([
                    function(callback) {
                        var outpath = path.resolve(__dirname + '../../../public/images/uploads' + originalS3Path);
                        debug(outpath);
                        fs.copy(imgPath, outpath, callback);
                    },
                    function(callback) {
                        var outpath = path.resolve(__dirname + '../../../public/images/uploads' + thumbnailS3Path);
                        debug(outpath);
                        fs.copy(imgPath, outpath, callback);
                    }
                ], function(err) {
                    if(err) {
                        return callback(err);
                    }

                    return callback(null, {
                        response: 200,
                        imgData: staticUrl + 'images/uploads' + originalS3Path
                    });
                });
            }
        }
    })


};
```