**Camp Guide Project Source Code**

* **app.js**

require("dotenv").config();

var express = require("express"),

app = express(),

bodyParser = require("body-parser"),

mongoose = require("mongoose"),

Campground = require("./models/campground"),

Comment = require("./models/comment"),

passport = require("passport"),

LocalStrategy = require("passport-local"),

User = require("./models/user"),

methodOverride = require("method-override"),

flash = require("connect-flash");

// Requiring routes

var commentRoutes = require("./routes/comments"),

campgroundRoutes = require("./routes/campgrounds"),

indexRoutes = require("./routes/index");

mongoose.connect(

process.env.DATABASEURL,

{ useNewUrlParser: true }

);

app.use(bodyParser.urlencoded({ extended: true }));

app.set("view engine", "ejs");

app.use(express.static(\_\_dirname + "/public"));

app.use(methodOverride("\_method"));

app.use(flash());

// PASSPORT CONFIG

app.use(

require("express-session")({

secret: "shibas are the best dogs in the world.",

resave: false,

saveUninitialized: false

})

);

app.locals.moment = require("moment");

app.use(passport.initialize());

app.use(passport.session());

passport.use(new LocalStrategy(User.authenticate()));

passport.serializeUser(User.serializeUser());

passport.deserializeUser(User.deserializeUser());

app.use(function(req, res, next) {

res.locals.currentUser = req.user;

res.locals.error = req.flash("error");

res.locals.success = req.flash("success");

next();

});

app.use("/", indexRoutes);

app.use("/campgrounds", campgroundRoutes);

app.use("/campgrounds/:id/comments", commentRoutes);

app.get("\*", function(req, res) {

res.render("error");

});

app.listen(process.env.PORT || 3000, function() {

console.log("listening on http://localhost:3000/");

});

* **models**

1. campground.js

var mongoose = require("mongoose");

// SCHEME SETUP

var campgroundSchema = new mongoose.Schema({

name: String,

price: Number,

image: String,

imageId: String,

description: String,

location: String,

lat: Number,

lng: Number,

phone: String,

booking: {

start: String,

end: String

},

tags: [],

createdAt: { type: Date, default: Date.now },

author: {

id: {

type: mongoose.Schema.Types.ObjectId,

ref: "User"

},

username: String

},

comments: [

{

type: mongoose.Schema.Types.ObjectId,

ref: "Comment"

}

],

rateAvg: Number,

rateCount: Number,

hasRated: [

{

type: mongoose.Schema.Types.ObjectId,

ref: "User"

}

]

});

module.exports = mongoose.model("Campground", campgroundSchema);

1. comment.js

var mongoose = require("mongoose");

var commentSchema = new mongoose.Schema({

text: String,

createdAt: { type: Date, default: Date.now },

author: {

id: {

type: mongoose.Schema.Types.ObjectId,

ref: "User"

},

username: String

},

rating: Number

});

module.exports = mongoose.model("Comment", commentSchema);

1. **user.js**

var mongoose = require("mongoose");

var passportLocalMongoose = require("passport-local-mongoose");

var UserSchema = new mongoose.Schema({

username: String,

password: String,

email: String,

phone: String,

fullName: String,

image: String,

imageId: String,

joined: { type: Date, default: Date.now }

});

UserSchema.plugin(passportLocalMongoose);

module.exports = mongoose.model("User", UserSchema);

* index.js

var Campground = require("../models/campground");

var Comment = require("../models/comment");

var User = require("../models/user");

var middlewareObj = {};

middlewareObj.checkCampgroundOwnership = function(req, res, next) {

Campground.findById(req.params.id, function(err, foundCampground) {

if (err || !foundCampground) {

req.flash("error", "Sorry, that campground does not exist!");

res.redirect("/campgrounds");

} else if (

foundCampground.author.id.equals(req.user.\_id) ||

req.user.isAdmin

) {

req.campground = foundCampground;

next();

} else {

req.flash("error", "You don't have permission to do that!");

res.redirect("/campgrounds/" + req.params.id);

}

});

};

middlewareObj.checkCommentOwnership = function(req, res, next) {

Comment.findById(req.params.comment\_id, function(err, foundComment) {

if (err || !foundComment) {

req.flash("error", "Sorry, that comment does not exist!");

res.redirect("/campgrounds");

} else if (

foundComment.author.id.equals(req.user.\_id) ||

req.user.isAdmin

) {

req.comment = foundComment;

next();

} else {

req.flash("error", "You don't have permission to do that!");

res.redirect("/campgrounds/" + req.params.id);

}

});

};

middlewareObj.checkProfileOwnership = function(req, res, next) {

User.findById(req.params.user\_id, function(err, foundUser) {

if (err || !foundUser) {

req.flash("error", "Sorry, that user doesn't exist");

res.redirect("/campgrounds");

} else if (foundUser.\_id.equals(req.user.\_id) || req.user.isAdmin) {

req.user = foundUser;

next();

} else {

req.flash("error", "You don't have permission to do that!");

res.redirect("/campgrounds/" + req.params.user\_id);

}

});

};

middlewareObj.isLoggedIn = function(req, res, next) {

if (req.isAuthenticated()) {

return next();

}

req.flash("error", "You need to be logged in to do that!");

res.redirect("/login");

};

module.exports = middlewareObj;

* **routes**

1. **campground.js**

var express = require("express");

var router = express.Router();

var Campground = require("../models/campground");

var middleware = require("../middleware");

var NodeGeocoder = require("node-geocoder");

var options = {

provider: "google",

httpAdapter: "https",

apiKey: process.env.GEO\_CODE\_API,

formatter: null

};

var geocoder = NodeGeocoder(options);

var multer = require("multer");

var storage = multer.diskStorage({

filename: function(req, file, callback) {

callback(null, Date.now() + file.originalname);

}

});

var imageFilter = function(req, file, cb) {

// accept image files only

if (!file.originalname.match(/\.(jpg|jpeg|png|gif)$/i)) {

return cb(new Error("Only image files are allowed!"), false);

}

cb(null, true);

};

var upload = multer({

storage: storage,

fileFilter: imageFilter

});

var cloudinary = require("cloudinary");

cloudinary.config({

cloud\_name: "dkxmrlblp",

api\_key: process.env.API\_KEY,

api\_secret: process.env.API\_SECRET

});

var Fuse = require("fuse.js");

// INDEX - show all campgrounds

router.get("/", function(req, res) {

var noMatch = null;

if (req.query.search) {

Campground.find({}, function(err, allCampgrounds) {

if (err) {

console.log(err);

} else {

var options = {

shouldSort: true,

threshold: 0.5,

location: 0,

distance: 100,

maxPatternLength: 32,

minMatchCharLength: 2,

keys: ["name", "location"]

};

var fuse = new Fuse(allCampgrounds, options);

var result = fuse.search(req.query.search);

if (result.length < 1) {

noMatch = req.query.search;

}

res.render("campgrounds/index", {

campgrounds: result,

noMatch: noMatch

});

}

});

} else if (req.query.sortby) {

if (req.query.sortby === "rateAvg") {

Campground.find({})

.sort({

rateCount: -1,

rateAvg: -1

})

.exec(function(err, allCampgrounds) {

if (err) {

console.log(err);

} else {

res.render("campgrounds/index", {

campgrounds: allCampgrounds,

currentUser: req.user,

noMatch: noMatch

});

}

});

} else if (req.query.sortby === "rateCount") {

Campground.find({})

.sort({

rateCount: -1

})

.exec(function(err, allCampgrounds) {

if (err) {

console.log(err);

} else {

res.render("campgrounds/index", {

campgrounds: allCampgrounds,

currentUser: req.user,

noMatch: noMatch

});

}

});

} else if (req.query.sortby === "priceLow") {

Campground.find({})

.sort({

price: 1,

rateAvg: -1

})

.exec(function(err, allCampgrounds) {

if (err) {

console.log(err);

} else {

res.render("campgrounds/index", {

campgrounds: allCampgrounds,

currentUser: req.user,

noMatch: noMatch

});

}

});

} else {

Campground.find({})

.sort({

price: -1,

rateAvg: -1

})

.exec(function(err, allCampgrounds) {

if (err) {

console.log(err);

} else {

res.render("campgrounds/index", {

campgrounds: allCampgrounds,

currentUser: req.user,

noMatch: noMatch

});

}

});

}

} else {

Campground.find({}, function(err, allCampgrounds) {

if (err) {

console.log(err);

} else {

res.render("campgrounds/index", {

campgrounds: allCampgrounds,

currentUser: req.user,

noMatch: noMatch

});

}

});

}

});

// CREATE - add new campground to db

router.post("/", middleware.isLoggedIn, upload.single("image"), function(

req,

res

) {

cloudinary.v2.uploader.upload(

req.file.path,

{

width: 1500,

height: 1000,

crop: "scale"

},

function(err, result) {

if (err) {

req.flash("error", err.message);

return res.render("error");

}

req.body.campground.image = result.secure\_url;

req.body.campground.imageId = result.public\_id;

req.body.campground.booking = {

start: req.body.campground.start,

end: req.body.campground.end

};

req.body.campground.tags = req.body.campground.tags.split(",");

req.body.campground.author = {

id: req.user.\_id,

username: req.user.username

};

geocoder.geocode(req.body.campground.location, function(err, data) {

if (err || !data.length) {

console.log(err);

req.flash("error", "Invalid address");

return res.redirect("back");

}

req.body.campground.lat = data[0].latitude;

req.body.campground.lng = data[0].longitude;

req.body.campground.location = data[0].formattedAddress;

Campground.create(req.body.campground, function(err, campground) {

if (err) {

req.flash("error", err.message);

return res.render("error");

}

res.redirect("/campgrounds");

});

});

},

{

moderation: "webpurify"

}

);

});

// NEW - show form to create new campground

router.get("/new", middleware.isLoggedIn, function(req, res) {

res.render("campgrounds/new");

});

// SHOW - shows more information about one campground

router.get("/:id", function(req, res) {

Campground.findById(req.params.id)

.populate("comments")

.exec(function(err, foundCampground) {

if (err || !foundCampground) {

console.log(err);

req.flash("error", "Sorry, that campground does not exist!");

return res.render("error");

}

var ratingsArray = [];

foundCampground.comments.forEach(function(rating) {

ratingsArray.push(rating.rating);

});

if (ratingsArray.length === 0) {

foundCampground.rateAvg = 0;

} else {

var ratings = ratingsArray.reduce(function(total, rating) {

return total + rating;

});

foundCampground.rateAvg = ratings / foundCampground.comments.length;

foundCampground.rateCount = foundCampground.comments.length;

}

foundCampground.save();

res.render("campgrounds/show", {

campground: foundCampground

});

});

});

// EDIT CAMPGROUND ROUTE

router.get(

"/:id/edit",

middleware.isLoggedIn,

middleware.checkCampgroundOwnership,

function(req, res) {

res.render("campgrounds/edit", {

campground: req.campground

});

}

);

// UPDATE CAMPGROUND ROUTE

router.put(

"/:id",

upload.single("image"),

middleware.checkCampgroundOwnership,

function(req, res) {

geocoder.geocode(req.body.campground.location, function(err, data) {

if (err || !data.length) {

req.flash("error", "Invalid address");

return res.redirect("back");

}

req.body.campground.lat = data[0].latitude;

req.body.campground.lng = data[0].longitude;

req.body.campground.location = data[0].formattedAddress;

req.body.campground.booking = {

start: req.body.campground.start,

end: req.body.campground.end

};

req.body.campground.tags = req.body.campground.tags.split(",");

Campground.findByIdAndUpdate(

req.params.id,

req.body.campground,

async function(err, campground) {

if (err) {

req.flash("error", err.message);

res.redirect("back");

} else {

if (req.file) {

try {

await cloudinary.v2.uploader.destroy(campground.imageId);

var result = await cloudinary.v2.uploader.upload(

req.file.path,

{

width: 1500,

height: 1000,

crop: "scale"

},

{

moderation: "webpurify"

}

);

campground.imageId = result.public\_id;

campground.image = result.secure\_url;

} catch (err) {

req.flash("error", err.message);

return res.render("error");

}

}

campground.save();

req.flash("success", "Successfully updated your campground!");

res.redirect("/campgrounds/" + req.params.id);

}

}

);

});

}

);

// DESTROY CAMPGROUND ROUTE

router.delete("/:id", middleware.checkCampgroundOwnership, function(req, res) {

Campground.findById(req.params.id, async function(err, campground) {

if (err) {

req.flash("error", err.message);

return res.redirect("back");

}

try {

await cloudinary.v2.uploader.destroy(campground.imageId);

campground.remove();

res.redirect("/campgrounds");

} catch (err) {

if (err) {

req.flash("error", err.message);

return res.render("error");

}

}

});

});

module.exports = router;

1. **comment.js**

var express = require("express");

var router = express.Router({ mergeParams: true });

var Campground = require("../models/campground");

var Comment = require("../models/comment");

var middleware = require("../middleware");

// Comments new

router.get("/new", middleware.isLoggedIn, function(req, res) {

Campground.findById(req.params.id, function(err, campground) {

if (err) {

console.log(err);

} else {

res.render("comments/new", { campground: campground });

}

});

});

// Comments create

router.post("/", middleware.isLoggedIn, function(req, res) {

Campground.findById(req.params.id, function(err, found) {

if (err) {

console.log(err);

}

var ratedArray = [];

found.hasRated.forEach(function(rated) {

ratedArray.push(String(rated));

});

if (ratedArray.includes(String(req.user.\_id))) {

req.flash(

"error",

"You've already reviewed this campgroud, please edit your review instead."

);

res.redirect("/campgrounds/" + req.params.id);

} else {

Campground.findById(req.params.id, function(err, campground) {

if (err) {

console.log(err);

res.redirect("/campgrounds");

} else {

var newComment = req.body.comment;

Comment.create(newComment, function(err, comment) {

if (err) {

req.flash("error", "Something went wrong.");

res.render("error");

} else {

// add username and id to comment

comment.author.id = req.user.\_id;

comment.author.username = req.user.username;

campground.hasRated.push(req.user.\_id);

campground.rateCount = campground.comments.length;

// save comment

comment.save();

campground.comments.push(comment);

campground.save();

req.flash("success", "Successfully added review!");

res.redirect("/campgrounds/" + campground.\_id);

}

});

}

});

}

});

});

// COMMENT EDIT ROUTE

router.get(

"/:comment\_id/edit",

middleware.isLoggedIn,

middleware.checkCommentOwnership,

function(req, res) {

res.render("comments/edit", {

campground\_id: req.params.id,

comment: req.comment

});

}

);

// COMMENT UPDATE ROUTE

router.put("/:comment\_id", middleware.checkCommentOwnership, function(

req,

res

) {

Comment.findByIdAndUpdate(req.params.comment\_id, req.body.comment, function(

err,

updatedComment

) {

if (err) {

res.redirect("back");

} else {

req.flash("success", "Review updated!");

res.redirect("/campgrounds/" + req.params.id);

}

});

});

// DESTROY COMMENT ROUTE

router.delete("/:comment\_id", middleware.checkCommentOwnership, function(

req,

res

) {

Comment.findByIdAndRemove(req.params.comment\_id, function(err) {

if (err) {

res.redirect("back");

} else {

Campground.findByIdAndUpdate(

req.params.id,

{ $pull: { comments: { $in: [req.params.comment\_id] } } },

function(err) {

if (err) {

console.log(err);

}

}

);

Campground.findByIdAndUpdate(

req.params.id,

{ $pull: { hasRated: { $in: [req.user.\_id] } } },

function(err) {

if (err) {

console.log(er);

}

}

);

req.flash("success", "Review deleted!");

res.redirect("/campgrounds/" + req.params.id);

}

});

});

module.exports = router;

* **views**

1. **campground**
   1. **new.ejs**

<% include ../partials/header %>

<div class="container mt-4">

<div class="row">

<div class="col">

</div>

<div class="col-12 col-sm-12 col-lg-9">

<div class="card shadow border-0">

<div class="card-body">

<h1 class="text-center">Create Campground</h1>

<p>

<small class="text-muted">Fields marked \* are required.</small>

</p>

<form action="/campgrounds" method="POST" enctype="multipart/form-data" class="needs-validation" novalidate>

<div class="form-group">

<label for="">Name\*</label>

<input type="text" class="form-control" name="campground[name]" placeholder="Name" required>

<div class="invalid-feedback">

Please provide a campground name.

</div>

</div>

<div class="form-group">

<label for="">Price/Night\*</label>

<input type="number" class="form-control" name="campground[price]" placeholder="Price/Night" min="0.01" step="0.01" required>

<div class="invalid-feedback">

Please provide a price.

</div>

</div>

<div class="input-group mb-3">

<div class="custom-file">

<input type="file" name="image" accept="image/\*" class="custom-file-input" id="image" aria-describedby="inputGroupFileAddon01"

required>

<div class="invalid-feedback">

Please upload an image of your campground.

</div>

<label class="custom-file-label" for="inputGroupFile01">Image\*</label>

</div>

</div>

<div class="form-group">

<label for="exampleFormControlTextarea1">Description\*</label>

<textarea class="form-control" id="exampleFormControlTextarea1" name="campground[description]" rows="3" required></textarea>

<div class="invalid-feedback">

Please provide a description of your campground.

</div>

</div>

<div class="form-group">

<label for="location">Location\*</label>

<input type="text" class="form-control" name="campground[location]" placeholder="Yosemite National Park, CA" id="location" required>

<div class="invalid-feedback">

Please provide a valid location

</div>

</div>

<div class="form-group">

<label for="">Phone Number\*</label>

<input type="text" class="form-control" name="campground[phone]" placeholder="123-456-7891" required>

<div class="invalid-feedback">

Please provide a phone number.

</div>

</div>

<label for="">Booking Window\*</label>

<div class="form-row">

<div class="form-group col-md-6">

<input type="text" class="form-control" name="campground[start]" placeholder="Starting Month" required>

<div class="invalid-feedback">

Please provide a booking window.

</div>

</div>

<div class="form-group col-md-6">

<input type="text" class="form-control" name="campground[end]" placeholder="Ending Month" required>

</div>

</div>

<div class="form-group">

<label for="">Amenities separated by commas</label>

<input type="text" class="form-control" name="campground[tags]" placeholder="Example: boating, fishing, wifi">

</div>

<button type="submit" class="btn btn-danger btn-block">Submit</button>

</form>

<a class="btn btn-link" href="/campgrounds">Go Back</a>

</div>

</div>

</div>

<div class="col">

</div>

</div>

</div>

<% include ../partials/footer %>

* 1. **edit.ejs**

<% include ../partials/header %>

<div class="container mt-4">

<div class="row">

<div class="col"></div>

<div class="col-12 col-sm-12 col-lg-9">

<div class="card shadow border-0">

<div class="card-body">

<h1 class="text-center">Edit

<%= campground.name %>

</h1>

<p>

<small class="text-muted">Fields marked \* are required.</small>

</p>

<form action="/campgrounds/<%= campground.\_id %>?\_method=PUT" method="POST" class="needs-validation" enctype="multipart/form-data"

novalidate>

<div class="form-group">

<label>Name\*</label>

<input type="text" class="form-control" name="campground[name]" value="<%= campground.name %>">

</div>

<div class="form-group">

<label for="">Price/Night\*</label>

<input type="number" class="form-control" name="campground[price]" value="<%= campground.price %>" min="0.01" step="0.01">

</div>

<div class="input-group mb-3">

<div class="custom-file">

<input type="file" name="image" accept="image/\*" class="custom-file-input" id="image" aria-describedby="inputGroupFileAddon01">

<label class="custom-file-label" for="inputGroupFile01">Image\*</label>

</div>

</div>

<div class="form-group">

<label for="exampleFormControlTextarea1">Description\*</label>

<textarea class="form-control" id="exampleFormControlTextarea1" name="campground[description]" rows="3" required><%= campground.description %></textarea>

</div>

<div class="form-group">

<label for="location">Location\*</label>

<input type="text" class="form-control" name="campground[location]" value="<%= campground.location %>" id="location" required>

<div class="invalid-feedback">

Please provide a valid location

</div>

</div>

<div class="form-group">

<label for="">Phone Number\*</label>

<input type="text" class="form-control" name="campground[phone]" value="<%= campground.phone %>" required>

<div class="invalid-feedback">

Please provide a phone number.

</div>

</div>

<div class="form-row">

<div class="form-group col-md-6">

<label for="">Booking Window (Start)\*</label>

<input type="text" class="form-control" name="campground[start]" value="<%= campground.booking.start %>" required>

<div class="invalid-feedback">

Please provide a booking window.

</div>

</div>

<div class="form-group col-md-6">

<label for="">Booking Window (End)\*</label>

<input type="text" class="form-control" name="campground[end]" value="<%= campground.booking.end %>" required>

</div>

</div>

<div class="form-group">

<label for="">Amenities separated by commas (boating,fishing,wifi)</label>

<input type="text" class="form-control" name="campground[tags]" value="<%= campground.tags %>">

</div>

<button type="submit" class="btn btn-danger btn-block">Submit</button>

</form>

<a class="btn btn-link btn-block" href="/campgrounds/<%= campground.\_id %>">Go Back</a>

</div>

</div>

</div>

<div class="col"></div>

</div>

</div>

<% include ../partials/footer %>