**Project Structure**

* App.js
* Schemas.js
* Middlewares.js
* Models
  + Campground.js
* Views
  + Home.ejs
  + Campgrounds
    - Index.ejs
    - Show.ejs
    - New.ejs
    - Edit.ejs
* Utils
  + ExpressError.js
  + catchAsync.js
* routes
  + campgrounds
  + reviews
* public
  + javascripts
    - formValidator.js
  + stylesheets

**Steps**

**CRUD**

1. Install the required npm libraries (method-override, express, ejs, and mongoose), create the express and mongoose boilerplate, and the campground mongoose model {title, price, description, location}.
2. Use the seed files to fill the database with the needed data for the campground model.
3. Show the campgrounds in the campground ejs file.
4. Create the campground show ejs file.
5. Create the delete function.
6. Create the new campground ejs file and its route.
7. Create the edit campground ejs file and its route.

**Adding Basic Styling**

1. Install ejs-mate library and require it then create the layout ejs file.
2. Add the Bootstrap links.
3. Create the navbar and footer partials.
4. Update the model by adding image, then seed the database again with the new data including price and description then update the show ejs file to show the image and description.
5. Edit the campground ejs file by making the points of the list as cards and add the images as well as the links to the show page.
6. Edit the new ejs file with the new styling and parameters.
7. Edit the edit ejs styling.
8. Style the show ejs file.

**Errors & Data Validation**

1. Add the client side validation using Bootstrap.
2. Edit the project structure for the new errors and data validation files.
3. Create ExpressError class and errorsWrapper function for Mongoose errors handling and a generic error handler middleware.
4. Create a middleware for not found routes and throw a customized error and make sure to handle this customization in the error handler middleware.
5. Create the errors handler template and pass the error data to it.
6. Add JOI library validation on the new campground route and get the resulting error and pass it to the errors handler route.
7. Create a new validation function for JOI validation and put its schema in different file then add the validation to the required routes.

**Adding the Review Model**

1. Create the review model and relate it with the campground one.
2. Create the review form in the same page of the campground show page.
3. Create the post review route.
4. Add the review validation by adding a new schema for the JOI library and creating the validation function then inserting it as a middleware to the required routes.
5. Display the reviews under the review form in the campground show ejs file.
6. Edit the style of the reviews.
7. Delete a review.
8. Delete a campground.

**Routes Restructuring & Flash**

1. Restructure the campground and review routes.
2. Create a static path for CSS and JS files.
3. Use express-session and configure it.
4. Create Flash success message for the needed routes and use a partial and bootstrap dismissible alert.
5. Create Flash error message.

**Adding in Authentication**

1. Install passport, passport-local, and passport-local-mongoose packages.
2. Create the user model and include the passport library in the schema plugin.
3. Configure passport in app.js file.
4. Create the user routes file and sign up form.
5. Create the sign up post route and add another try and catch to make the error appear in a flash message at the same page instead of redirecting to another.
6. Create the sign in routes.
7. Create the sign in authentication middleware function and add it to the necessary routes.
8. Create the sign out, in, and up buttons as well as the sign out route.
9. Get the current signed in user and add it to the locals variables then customize the navbar in a way that whether shows sign in and up or sign out.
10. Fix the sign up post route so that the user stays signed in after signing up by the help of passport library.
11. Add the return into feature in which the user gets returned to the page he wanted to visit when he wasn’t signed in and got redirected to the sign in page.

**Basic Authorization**

1. Seed the campgrounds with users, and add the user to the campground show pages, then edit the new campground post route so the user is added.
2. Show or hide the campground edit and delete buttons depending on the campground author and the signed in user.
3. Prevent the user from editing or deleting the campgrounds that don’t belong to him if he tried to reach that using Postman or AJAX or something.
4. Add the made authorization check into a separate function and use it as a middleware, then move all middlewares into a separate file.
5. Add users to the review model and edit the review post route so the user is added and create an authorization that only signed in users can see the review form and send a post request using methods like Postman and AJAX.
6. Populate the authors of the reviews then add the users to the reviews campground show page then authorize who can delete the reviews or not from the template then add the is signed in middleware to the review delete route and add the authorization to the route as a middleware.

**Controllers & Star Ratings**

1. Create the controllers folder and add all routes to their files.
2. Restructure the routes by chaining.
3. Add the star rating css file and edit the ratings for the reviews.
4. Edit the rating form.

**Image Upload**

1. Identify the needed environment variables regarding Cloudinary and require them in the main app file.
2. Install Cloudinary and Cloudinary for multer packages then configure them and add the new storage to multer.
3. Use multer middleware in the new campground post route and edit the campground schema in which the images become array, then edit the post route in which the images are added, then show the images in the campground show file.
4. Display the images in a carousel in the campground show page.
5. Update the seeds for new images and edit the campgrounds index page to show the first image.
6. Add multer to the edit route and fix the image input in the edit show page.
7. Edit the file input style and use the recommended javascript link and don’t forget to add the old classes that are expected from this javascript library (because Bootstrap updated the classes while the javascript library didn’t).
8. Add the images in the campground edit page with checkboxes and style them as thumbnails then associate each image with its value by joining them into an array so when submitting the form, the images would be in the same array.
9. Delete the images from the database and from Cloudinary.
10. Use Cloudinary transformation API and unify the size of images in the thumbnails.
11. Delete the images from Cloudinary when deleting the campground. [Me]

**Adding Maps**

1. Signing up with Mapbox and get the API token then add it to the .env file.
2. Install Mapbox package for the geocoding service (I still don’t know how Colt knew that we should install it because I couldn’t find this in the documentation) and configure it in the code then get the location when creating a new campground.
3. Edit the campground schema in which the geometry (he stored the whole geometry and not the coordinates only because this is gonna be useful later on) is stored and edit the new campground post route.
4. Show the map on the campground show page.
5. Add a marker to the map and apply the campgrounds locations on the maps.
6. Fix the index campground page problem of the campground that don’t have images while it’s required in the template, then fix the seeds in which all campgrounds have geometries.
7. Customize a map pop up that shows the location and title for example.

**Fancy Cluster Map**

1. Add the default cluster map (earthquake) that is provided by Mapbox to the campground index page.
2. Make sure that the campgrounds locations are seeded properly with their actual longitude and latitude.
3. Add the campgrounds to the cluster and check before how the cluster code should see the data sent to it.
4. Change the earthquake label in the cluster to campgrounds.
5. Change the size and color of the cluster. [Optional]
6. Add a customized pop up and before add a virtual with the properties and pop up key that has the pop up value.
7. Add map controls to the cluster and the show page map.

**Styles Clean Up**

1. Create a home page and style it with little Bootstrap.
2. Style the home page without Bootstrap.
3. Style the login page using Bootstrap.
4. Style the sign up page using Bootstrap and make the new and edit campground pages responsive.
5. Remove the lineup styles added to the maps and add them to a separate file in the public folder.

**Common Security Issues**

1. Install express-mongo-sanitize package and use it as a middleware for protection against Mongo injection.
2. Create JOI HTML sanitizing function by making use of sanitize-html package for protection against XSS attacks.
3. Change the name of the session so it cannot be known by its default value.
4. Hide the errors stack in the production mode.
5. Use helmet package as a middleware with putting contentSecurityPolicy to false for setting the http headers needed for protection against common security bugs.
6. Configure the content security policy header.

**Deploying**

1. Create an account on Mongo Atlas and configure it by adding a user then white listing my IP address, then connect the app to the database by the provided URL. Knowing that the URL should be stored in an environment variable so the password cannot be shown to others.
2. Install connect-mongo package to store the session in the mongo database, then configure the storage in the main app.
3. Create an account on Heroku then install its CLI and connect with it by typing heroku login.
4. Deploy the app by following the steps below, but before, create .gitignore file and add the env and node modules files to it:

1- type "heroku login" in the terminal.  
2- make sure that you are in the main app folder and type "heroku create". Knowing that this is done once because it works as a url to the host provided by heroku.  
3- edit the env variables in which they have another value in case the env variables weren't available.  
4- type "git remote -v"  
5- type "git add ."  
6- type "git commit -m "ready to deploying""  
7- type "git push heroku master"

1. Figure out the heroku errors by typing “heroku logs --tail”, then set up the script by making the app start when typing start inside the json file, and change the port to the default env variable that heroku uses.
2. Configure the environment variables in heroku and white list all ip addresses in mongo atlas to let heroku access it.