Web Development

PUT and DELETE routes and Deployment

Method-Override Package

- Lets you use HTTP verbs such as PUT or DELETE in places where the client doesn't support it.
- Use a query string value to override the method.
- Specify the query string key as a string argument to the methodOverride() function.
- This method of using a query value would typically be used in conjunction with plain **HTML <form> elements** when trying to support legacy browsers but still use newer methods.
- Type **npm install method-override --save** to install.
- Note the method is still POST instead of DELETE in example.
- [Refer: https://www.npmjs.com/package/method-override](https://www.npmjs.com/package/method-override)

```
$ npm install method-override
```

```
var express = require('express')
var methodOverride = require('method-override')
var app = express()

// override with POST having ?_method=DELETE
app.use(methodOverride('_method'))
```

Example call with query override using HTML `<form>`:

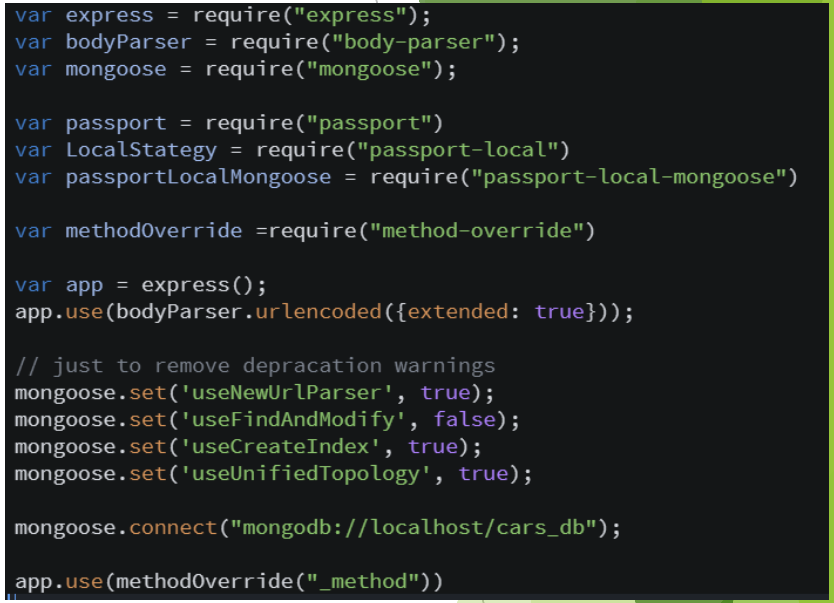
```
<form method="POST" action="/resource?_method=DELETE">
  <button type="submit">Delete resource</button>
</form>
```

RestFUL routes

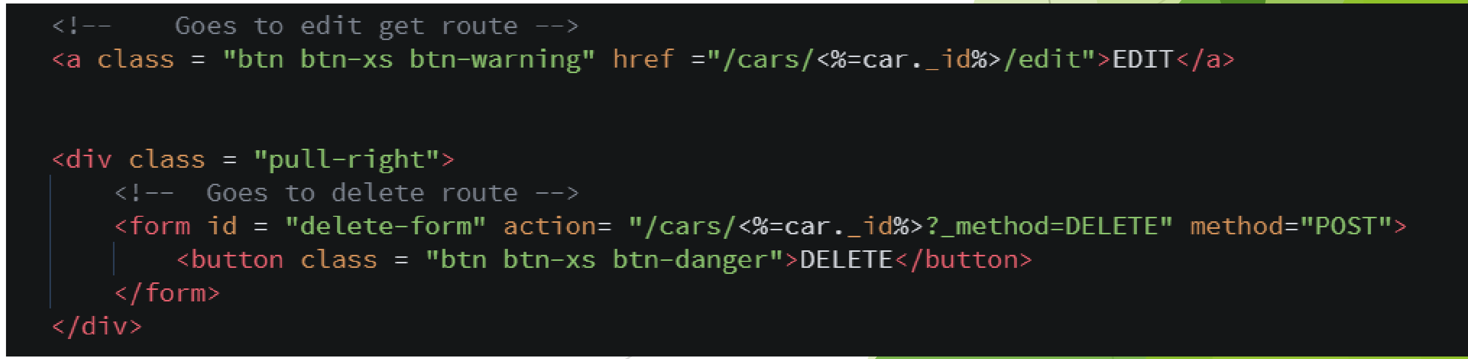
HTTP VERB	ROUTE	Action	Used For
GET	'/articles'	index action	index page to display all articles
GET	'/articles/new'	new action	displays create article form
POST	'/articles'	create action	creates one article
GET	'/articles/:id'	show action	displays one article based on ID in the url
GET	'/articles/:id/edit'	edit action	displays edit form based on ID in the url
PUT	'/articles/:id'	update action	*replaces* an existing article based on ID in the url
DELETE	'/articles/:id'	delete action	deletes one article based on ID in the url

Edit and Delete buttons

- After installing method-override package require it in the app.js file.
- In **index.ejs** add buttons for Editing and Deleting each post under the **Submitted by** element in the page.
- We use the btn class of Bootstrap to add the buttons. In which we also specify the size and type of button.
- The Edit buttons will take the user to /cars/car.id/edit **GET** route, which will show the user the edit.ejs file where the user can edit the details of the post.
- **Note the edit button doesn't take the user to the PUT route yet.**



```
var express = require("express");
var bodyParser = require("body-parser");
var mongoose = require("mongoose");

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

var methodOverride =require("method-override")

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

// just to remove depracation warnings
mongoose.set('useNewUrlParser', true);
mongoose.set('useFindAndModify', false);
mongoose.set('useCreateIndex', true);
mongoose.set('useUnifiedTopology', true);

mongoose.connect("mongodb://localhost/cars_db");

app.use(methodOverride("_method"))
```

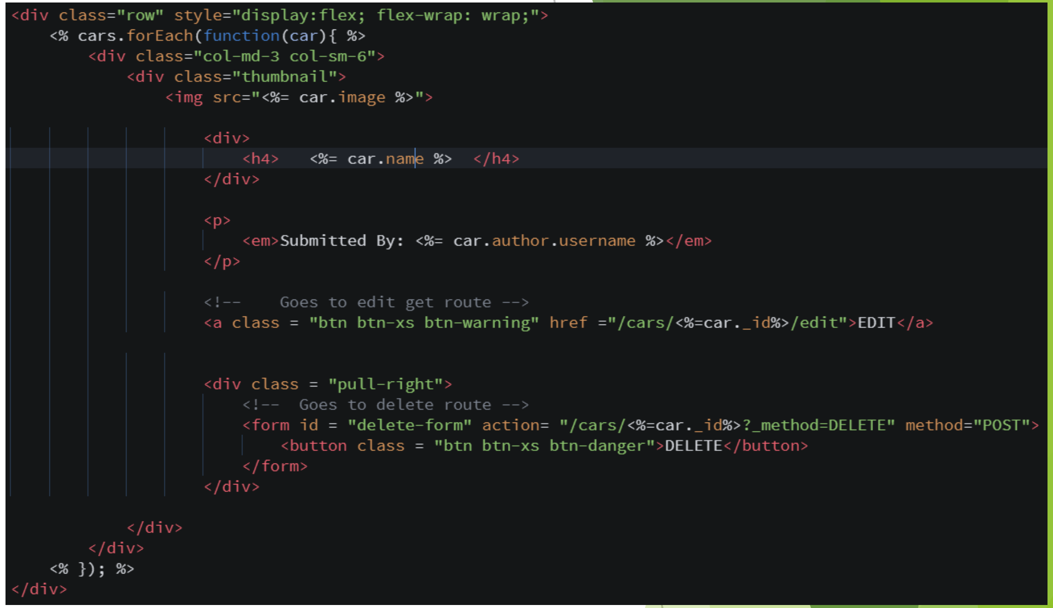


```
<!--    Goes to edit get route -->
<a class = "btn btn-xs btn-warning" href ="/cars/<%=car._id%>/edit">EDIT</a>


<div class = "pull-right">
    <!--  Goes to delete route -->
    <form id = "delete-form" action= "/cars/<%=car._id%>?_method=DELETE" method="POST">
        <button class = "btn btn-xs btn-danger">DELETE</button>
    </form>
</div>
```

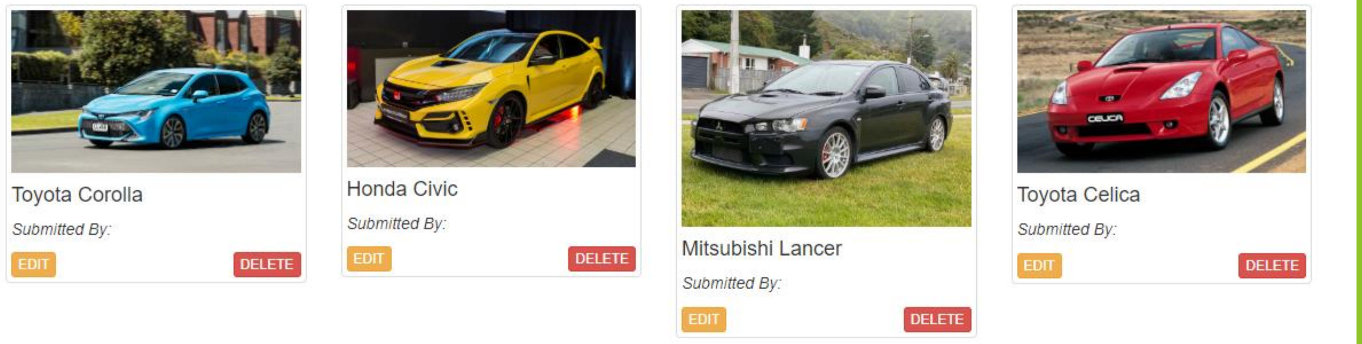
Edit and Delete buttons

- The delete button is enclosed in a form. Once submitted,
- The form will take the user to the DELETE route for that post using the **_method=DELETE** method override.
- Note the method=POST although this is directing to the DELETE route.

- Note also that the information for each post is not centered anymore.
- And the pull-right class is used of Bootstrap to keep the edit and delete buttons at same level.

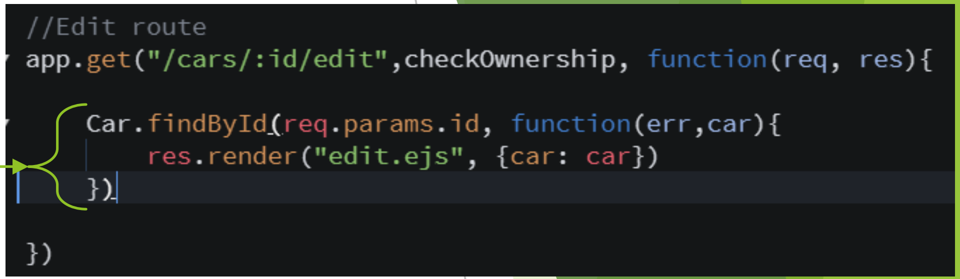


```
<div class="row" style="display:flex; flex-wrap: wrap;">
    <% cars.forEach(function(car){ %>
        <div class="col-md-3 col-sm-6">
            <div class="thumbnail">
                <img src="<%= car.image %>">

                    <div>
                        <h4>    <%= car.name %>   </h4>
                    </div>

                    <p>
                        <em>Submitted By: <%= car.author.username %></em>
                    </p>

                    <!--    Goes to edit get route -->
                    <a class = "btn btn-xs btn-warning" href ="/cars/<%=car._id%>/edit">EDIT</a>


                    <div class = "pull-right">
                        <!--  Goes to delete route -->
                        <form id = "delete-form" action= "/cars/<%=car._id%>?_method=DELETE" method="POST">
                            <button class = "btn btn-xs btn-danger">DELETE</button>
                        </form>
                    </div>

            </div>
        </div>
    <% }); %>
</div>
```

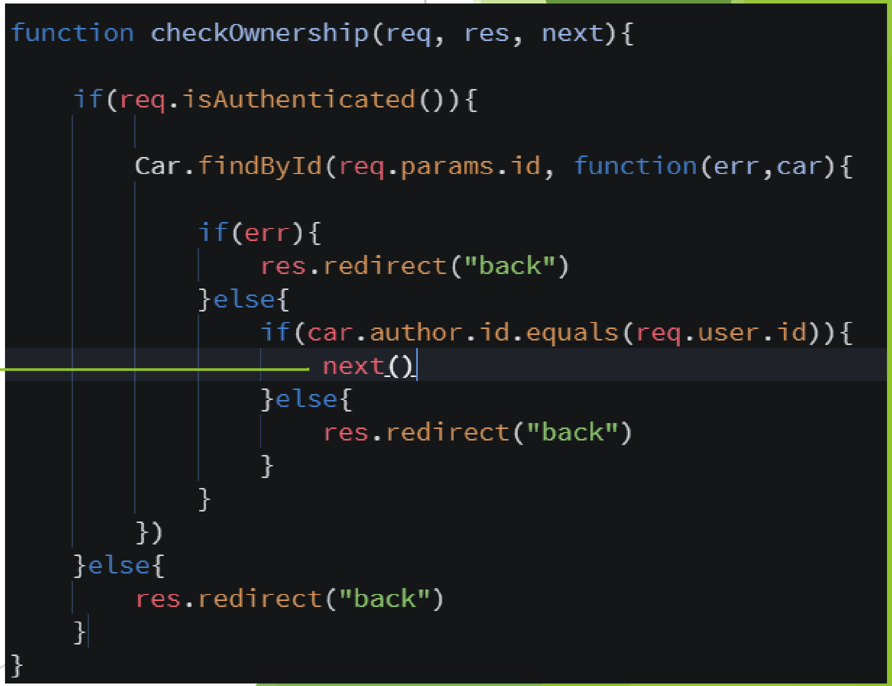


Edit Route

- Once the edit button is clicked, user is taken to the edit GET route.
- In the edit route we use a middleware to check if the user has ownership of the post. This way only the author of the post is allowed to edit the post.
- In the checkOwnership function we check if the user is authenticated first.
- The findById() takes in the **post's id** and gets the details of the post associated with that id. Which is stored in car.
- The car has details of username and userid of the user who made the post. We check this **user id** stored against the details of the user currently logged in.
- If it is the same then next() is called to proceed to the next step.
- Otherwise the user is redirected to back.
- On next(), the user is allowed In the edit route.
- In the route we use the findById method() to get the details of the car using the **id of the post**.
- We then render edit.ejs file and pass in the details of the car data retrieved.



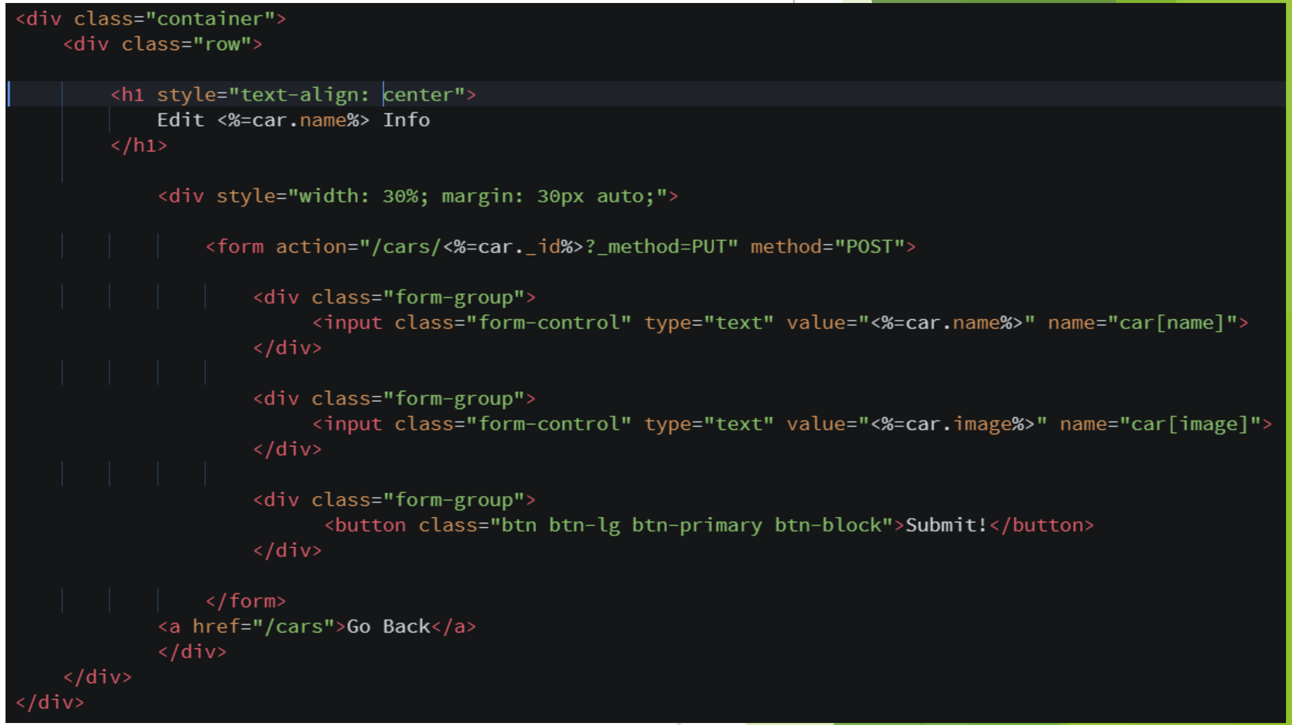
```
//Edit route
app.get("/cars/:id/edit",checkOwnership, function(req, res){

    Car.findById(req.params.id, function(err,car){
        res.render("edit.ejs", {car: car})
    })

})
```



```
function checkOwnership(req, res, next){

    if(req.isAuthenticated()){

        Car.findById(req.params.id, function(err,car){

            if(err){
                res.redirect("back")
            }else{
                if(car.author.id.equals(req.user.id)){
                    next()
                }else{
                    res.redirect("back")
                }
            }
        })
    }else{
        res.redirect("back")
    }
}
```

Edit.ejs

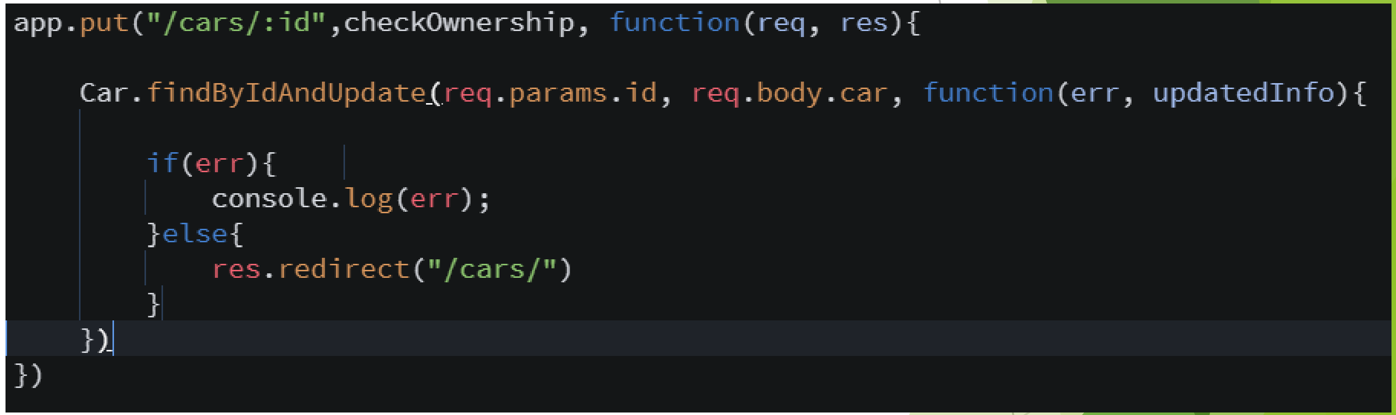
- In edit.ejs the user is allowed to edit the post.
- The edit.ejs file looks similar to new.ejs.
- There are a few differences.
- In the input tag for name and image, we add in the **value** of the name and image instead of **placeholder** text.
- We also add in **name** attribute and store car[name] and car[image]. car[] is just an array used to store the name and image information.
- The form once submitted gets sent to the/cars/car.id PUT route with the edited post.
- Once again note that the method=POST but we are using the method-override to send the form to the **PUT** route.



```
<div class="container">
    <div class="row">

        <h1 style="text-align: center">
            Edit <%=car.name%> Info
        </h1>

            <div style="width: 30%; margin: 30px auto;">

                <form action="/cars/<%=car._id%>?_method=PUT" method="POST">

                    <div class="form-group">
                         <input class="form-control" type="text" value="<%=car.name%>" name="car[name]">
                    </div>

                    <div class="form-group">
                         <input class="form-control" type="text" value="<%=car.image%>" name="car[image]">
                    </div>

                    <div class="form-group">
                          <button class="btn btn-lg btn-primary btn-block">Submit!</button>
                    </div>

                </form>
            <a href="/cars">Go Back</a>
            </div>
    </div>
</div>
```

PUT route

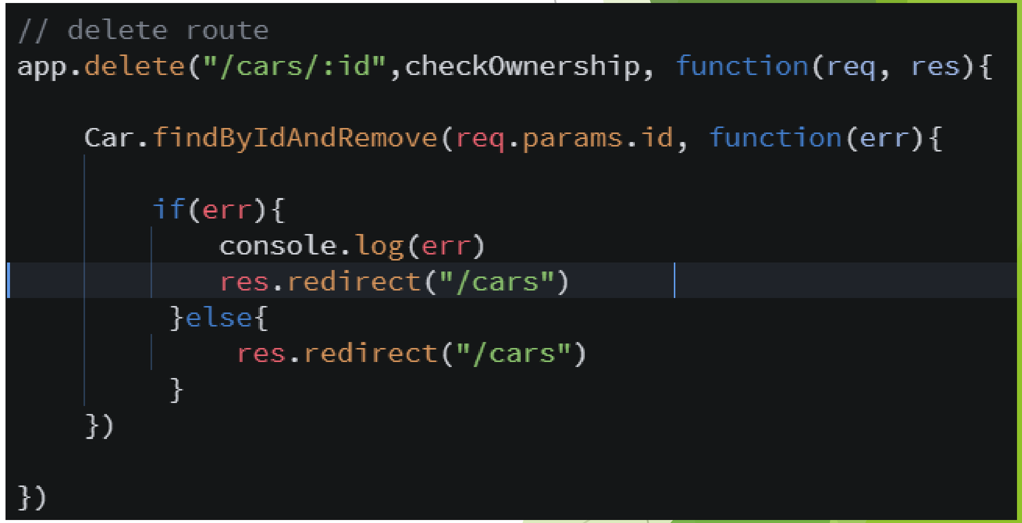
- In the car/:id put route we check for the ownership again just in case.
- Also note that the method name is now PUT.
- We then use the **findByIdAndUpdate()** function to update the information stored in that post id.
- The functions takes in the post id and the information of the name and image we stored in an array.
- It also takes a callback function which gives an error or the updatedInfo
- If there is an error we console log out the error or else we redirect the user back to the cars index page.



```
app.put("/cars/:id",checkOwnership, function(req, res){

    Car.findByIdAndUpdate(req.params.id, req.body.car, function(err, updatedInfo){

        if(err){
            console.log(err);
        }else{
            res.redirect("/cars/")
        }
    })
})
```

DELETE route

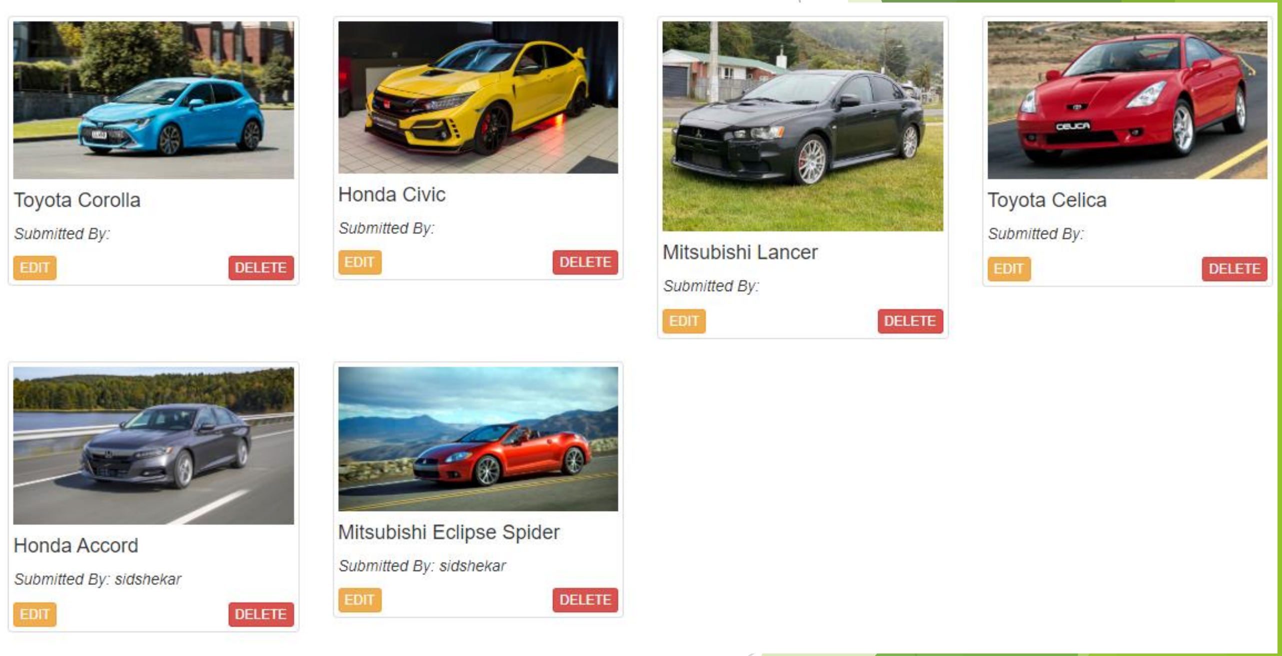
- The delete route is called once the delete button is pressed.
- In the cars/id DELETE route, we check ownership of the post again.
- Then we use the **findByIdAndRemove()** function which just takes in the post id and performs the function on it.
- The user is taken to the cars index page either way.
- If there is an error the message is sent to console log.



```
// delete route
app.delete("/cars/:id",checkOwnership, function(req, res){

    Car.findByIdAndRemove(req.params.id, function(err){

        if(err){
            console.log(err)
            res.redirect("/cars")
         }else{
             res.redirect("/cars")
         }
    })

})
```

Output

- Now start mongod
- And start the app.js file.
- Now you can edit and delete the post if you are logged in.
- Note that for posts that don't have user information, if you try delete or edit the post the server will crash.
- So make sure to **remove** the posts from the database.


Toyota Corolla
Submitted By:
EDIT DELETE

Honda Civic
Submitted By:
EDIT DELETE

Mitsubishi Lancer
Submitted By:
EDIT DELETE

Toyota Celica
Submitted By:
EDIT DELETE

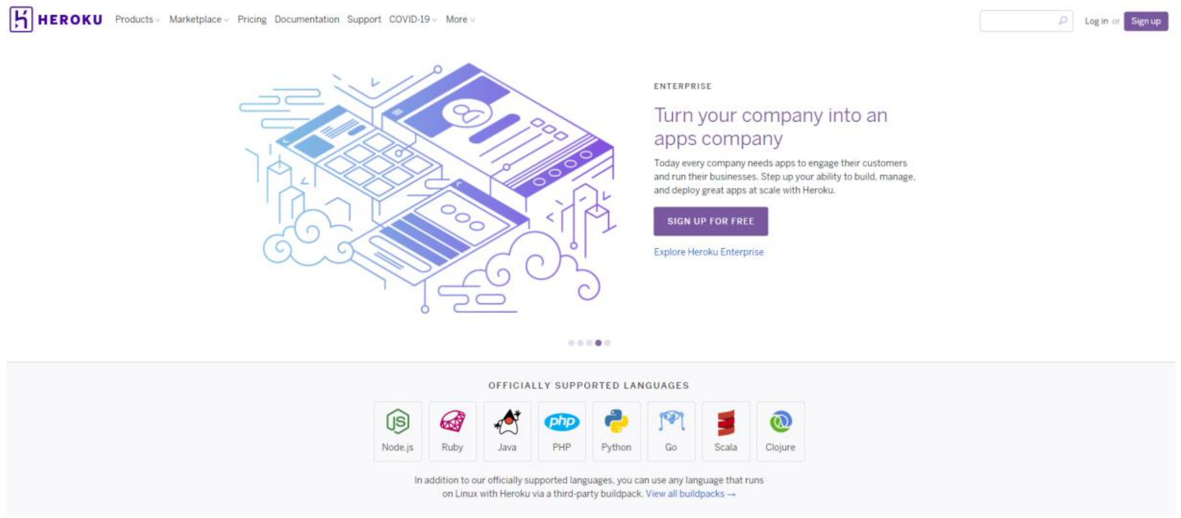
Honda Accord
Submitted By: sidshekar
EDIT DELETE

Mitsubishi Eclipse Spider
Submitted By: sidshekar
EDIT DELETE

Deployment

Heroku

- Create a free user account on Heroku
- Heroku.com
- Create account using email as this email will be used for authentication inside goorm
- Use Hobbyist as role.
- For primary development language you can select node.js.



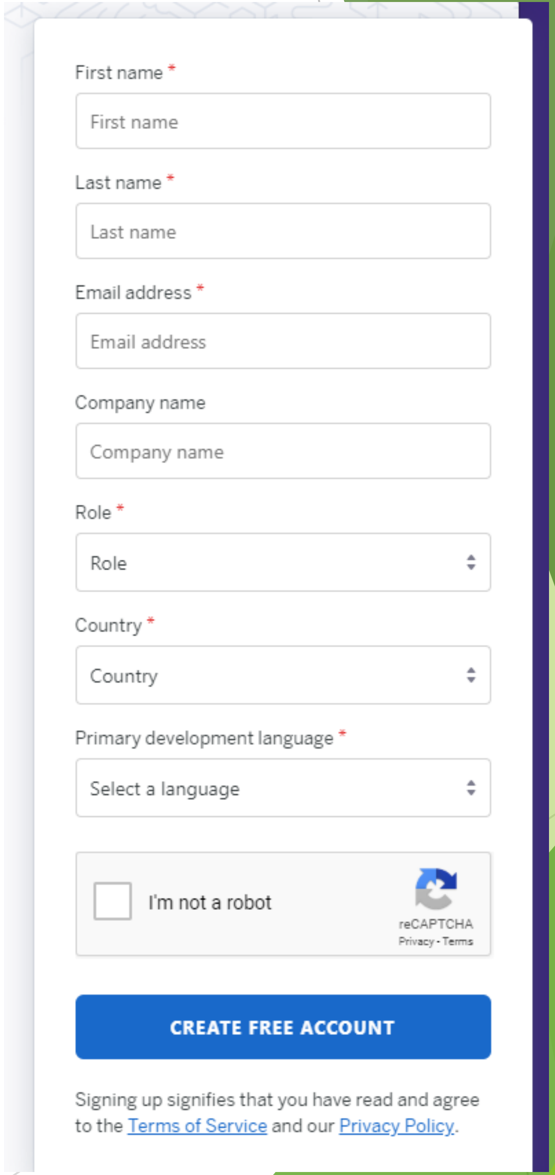
ENTERPRISE

Turn your company into an apps company

Today every company needs apps to engage their customers and run their businesses. Step up your ability to build, manage, and deploy great apps at scale with Heroku.

OFFICIALLY SUPPORTED LANGUAGES

In addition to our officially supported languages, you can use any language that runs on Linux with Heroku via a third-party buildpack. View all buildpacks →

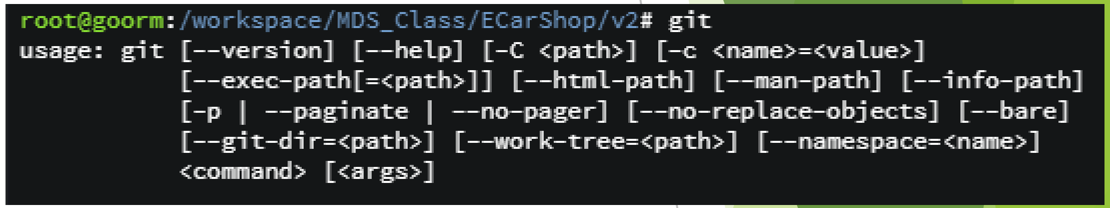


First name *	First name
Last name *	Last name
Email address *	Email address
Company name	Company name
Role *	Role
Country *	Country
Primary development language *	Select a language

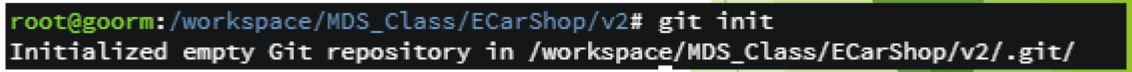
Signing up signifies that you have read and agree to the Terms of Service and our Privacy Policy.

Git bash

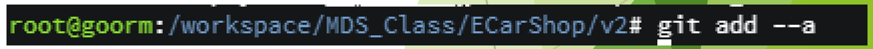
- Now we will ready our web application to be sent to Heroku using git.
- Make sure you are in the correct project folder. Also make sure the server is not running by pressing ctrl+c.
- We will first check **git** is present.
- In the terminal type git.
- Next type **git init** to initialize the repository.
- Then type ***git add --a*** to add all files to the git repo.
- Next set the user.email and user.name for git as otherwise it wont let you commit.
- Then commit to git by typing **git commit -m “first commit”**



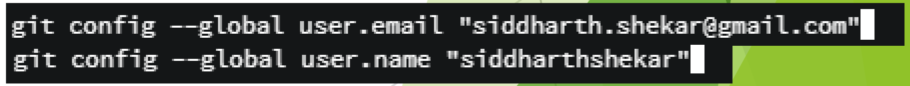
```
root@goorm:/workspace/MDS_Class/ECarShop/v2# git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```



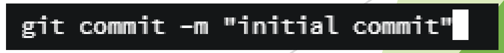
```
root@goorm:/workspace/MDS_Class/ECarShop/v2# git init
Initialized empty Git repository in /workspace/MDS_Class/ECarShop/v2/.git/
```



```
root@goorm:/workspace/MDS_Class/ECarShop/v2# git add --a
```



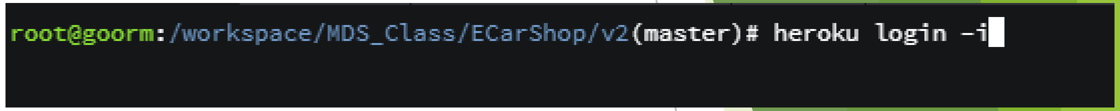
```
git config --global user.email "siddharth.shekar@gmail.com"
git config --global user.name "siddharthshekar"
```



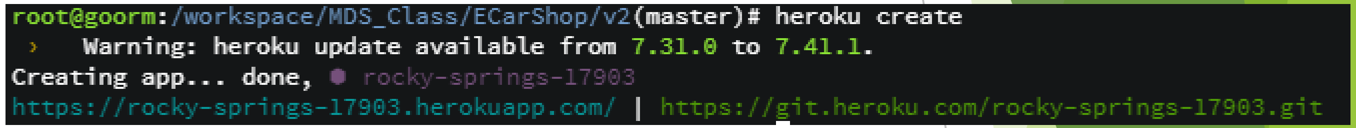
```
git commit -m "initial commit"
```

Heroku login

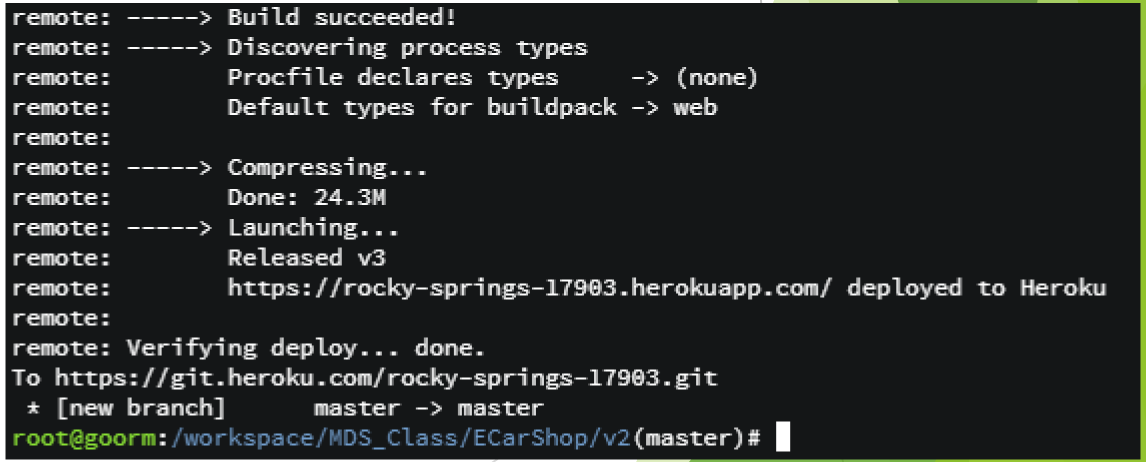
- In the terminal login to Heroku by typing **Heroku login -i**
- Type in Heroku login and password
- Then call **Heroku create**
- This will
  - make space,
  - make new application,
  - generate a url and
  - associate url with the git repo
- Next type **git push Heroku master**
- To push the project to heroku



```
root@goorm:/workspace/MDS_Class/ECarShop/v2(master)# heroku login -i
```



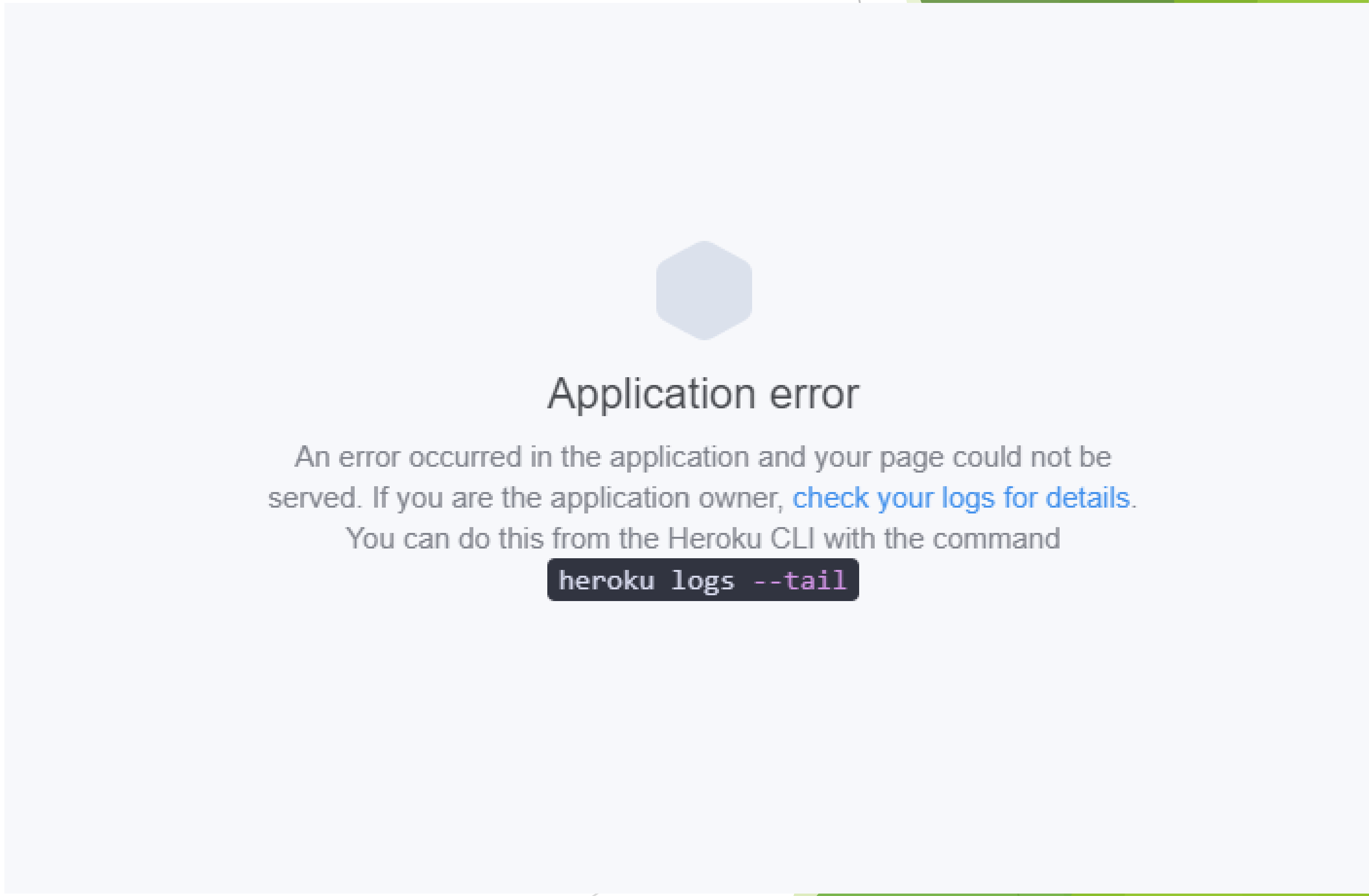
```
root@goorm:/workspace/MDS_Class/ECarShop/v2(master)# heroku create
 ›   Warning: heroku update available from 7.31.0 to 7.41.1.
Creating app... done, ⬢ rocky-springs-17903
https://rocky-springs-17903.herokuapp.com/ | https://git.heroku.com/rocky-springs-17903.git
```



```
remote: -----> Build succeeded!
remote: -----> Discovering process types
remote:        Procfile declares types     -> (none)
remote:        Default types for buildpack -> web
remote:
remote: -----> Compressing...
remote:        Done: 24.3M
remote: -----> Launching...
remote:        Released v3
remote:        https://rocky-springs-17903.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/rocky-springs-17903.git
 * [new branch]      master -> master
root@goorm:/workspace/MDS_Class/ECarShop/v2(master)#
```

Deplyment Error!

- Now login to Heroku.
- In the dashboard you will see the new app deployed.
- Click on the App
- Press Open App at the top right corner.
- This will open the app in a new page.
- Unfortunately there are some errors we have to fix before the app can run properly.



Application error

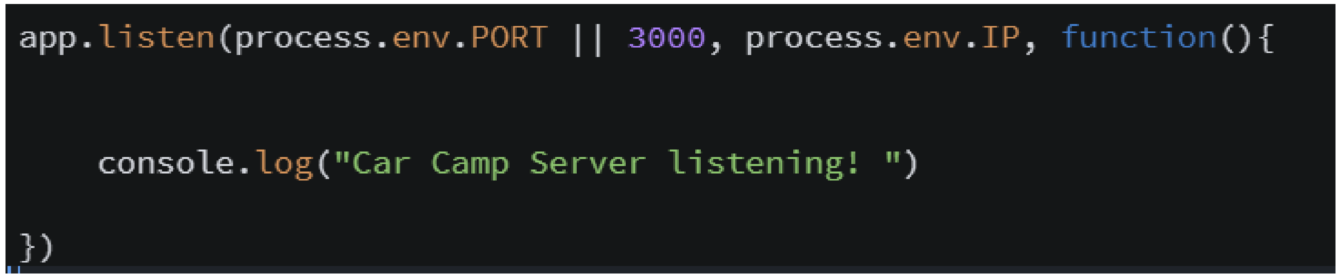
An error occurred in the application and your page could not be served. If you are the application owner, check your logs for details. You can do this from the Heroku CLI with the command

```
heroku logs --tail
```

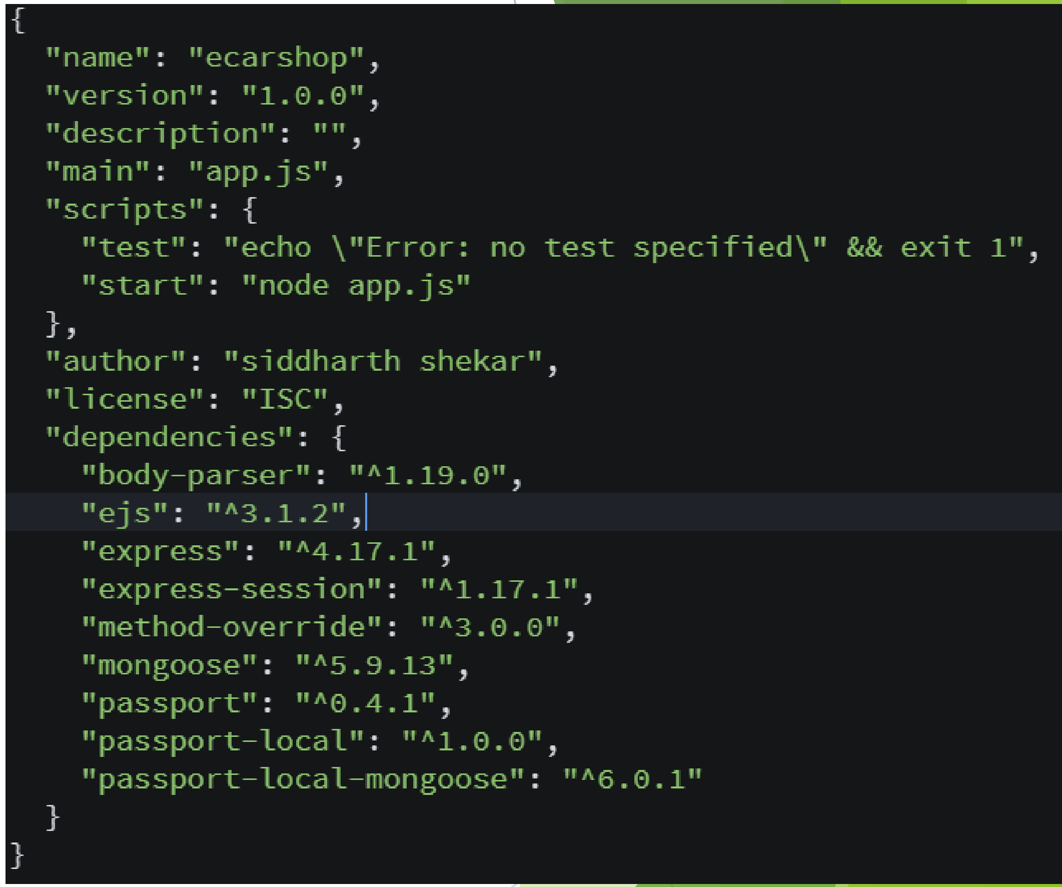
Resolution

- 1. Change package.json to initiate app.js at start.
  - Under scripts
  - **Add comma after previous line and**
  - add **“start” : “node app.js”** under it.

- 2. In app.listen() change port to PORT and ip to IP.
  - Yes otherwise this will cause an error.



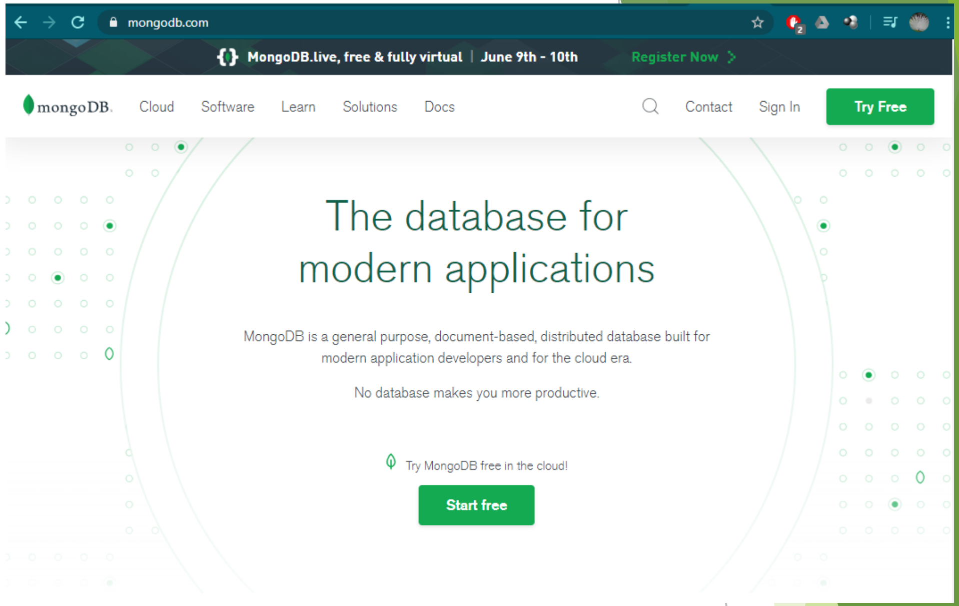
```
app.listen(process.env.PORT || 3000, process.env.IP, function(){

    console.log("Car Camp Server listening! ")

})
```



```
{
  "name": "ecarshop",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node app.js"
  },
  "author": "siddharth shekar",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "ejs": "^3.1.2",
    "express": "^4.17.1",
    "express-session": "^1.17.1",
    "method-override": "^3.0.0",
    "mongoose": "^5.9.13",
    "passport": "^0.4.1",
    "passport-local": "^1.0.0",
    "passport-local-mongoose": "^6.0.1"
  }
}
```

Remote Database

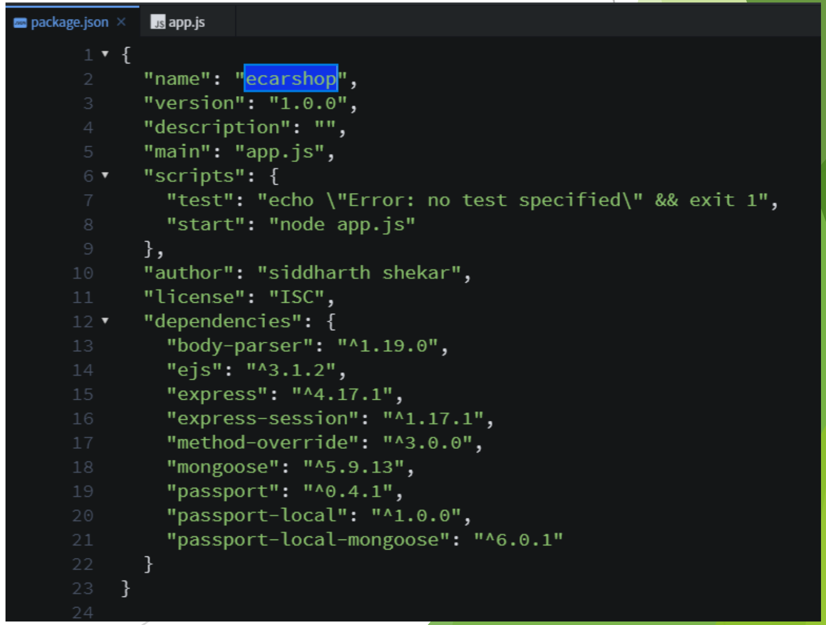
- 3. We are still using local database.
- We have to create a remote database.
- Goto Mongodb.com
- Click TryFree
- Register with email and password.
- Under Projects create a New Project
- Name it same as the current project name – ecarshop
- Click Create Project



The database for modern applications

MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era.

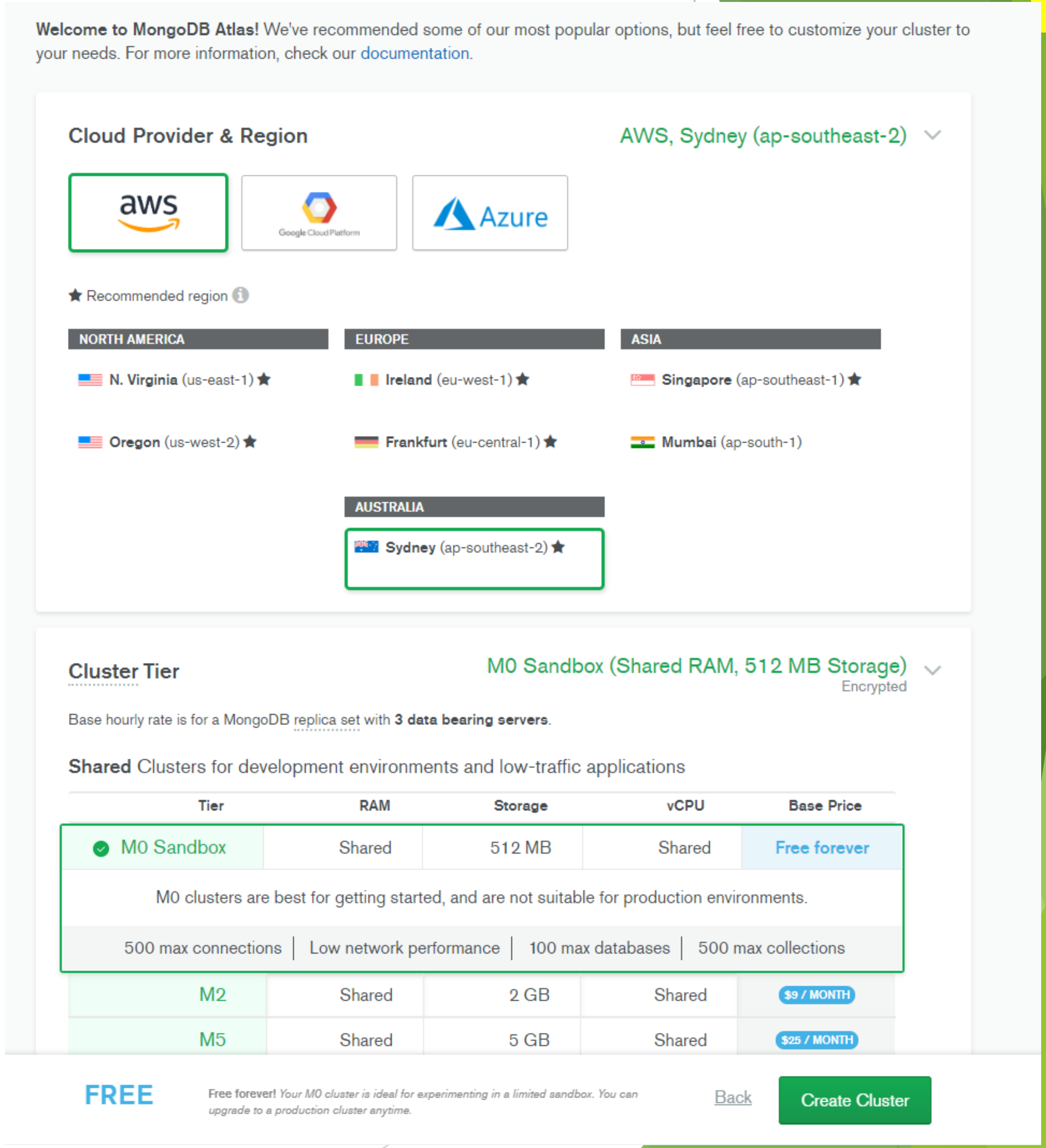
No database makes you more productive.



```
{
  "name": "ecarshop",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node app.js"
  },
  "author": "siddharth shekar",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "ejs": "^3.1.2",
    "express": "^4.17.1",
    "express-session": "^1.17.1",
    "method-override": "^3.0.0",
    "mongoose": "^5.9.13",
    "passport": "^0.4.1",
    "passport-local": "^1.0.0",
    "passport-local-mongoose": "^6.0.1"
  }
}
```

Remote Database

- Under Clusters press Build a Cluster
- Choose Create a cluster – FREE.
- Choose the Cloud Provider and Region.
- Select a closer region if it is FREE.
- Make sure Cluster Tier is Free as well.
- Press Create Cluster.
- You will be taken to the clusters page and your cluster will be created.
- **This might take 5 mins to create!!**



Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our documentation.

Cloud Provider & Region — AWS, Sydney (ap-southeast-2)

★ Recommended region

NORTH AMERICA	EUROPE	ASIA
N. Virginia (us-east-1) ★	Ireland (eu-west-1) ★	Singapore (ap-southeast-1) ★
Oregon (us-west-2) ★	Frankfurt (eu-central-1) ★	Mumbai (ap-south-1)
	AUSTRALIA	
	Sydney (ap-southeast-2) ★	

Cluster Tier — M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted

Base hourly rate is for a MongoDB replica set with **3 data bearing servers**.

Shared Clusters for development environments and low-traffic applications

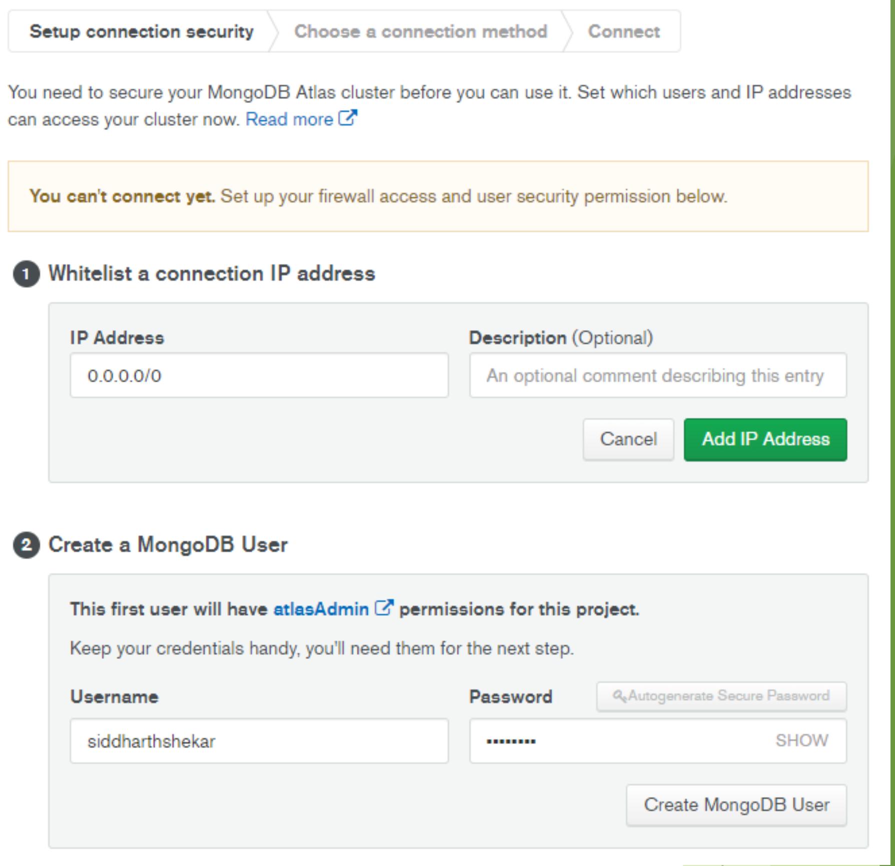
Tier	RAM	Storage	vCPU	Base Price
✓ M0 Sandbox	Shared	512 MB	Shared	Free forever
M0 clusters are best for getting started, and are not suitable for production environments.				
500 max connections | Low network performance | 100 max databases | 500 max collections				
M2	Shared	2 GB	Shared	$9 / MONTH
M5	Shared	5 GB	Shared	$25 / MONTH

FREE

Free forever! *Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.*

Remote Database

- Next under Cluster0 click on the **CONNECT** button.
- Under IP Address add 0.0.0.0/0 as IP address, That will enable access to our database server from any IP.
- Add a username and strong password.
- Click Create MongoDB User.
- Next Press Choose a Connection Method and select Connect your Application.
- Copy the mongodb+srv code.
- We will use this to connect to the database.
- In app.js replace mongoose.connect(“”) with this and replace <password> with the password you created above.



You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more

You can't connect yet. Set up your firewall access and user security permission below.

1 Whitelist a connection IP address

IP Address: 0.0.0.0/0

Description (Optional): An optional comment describing this entry

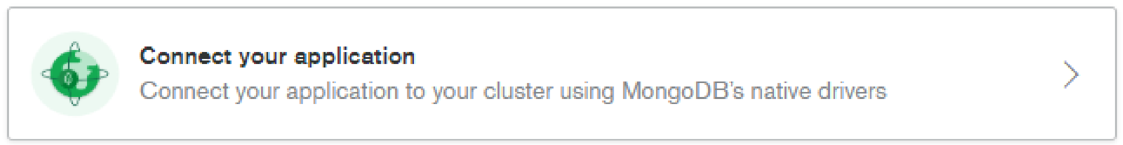
2 Create a MongoDB User

This first user will have atlasAdmin permissions for this project.

Keep your credentials handy, you'll need them for the next step.

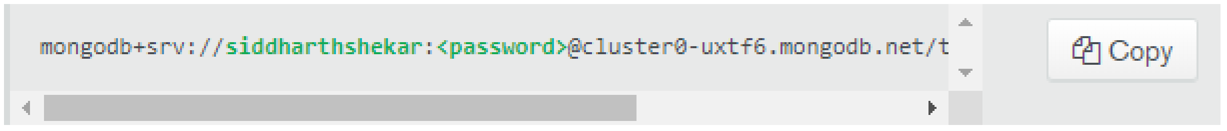
Username: siddharthshekar

Password: ••••••••• SHOW



Connect your application

Connect your application to your cluster using MongoDB's native drivers

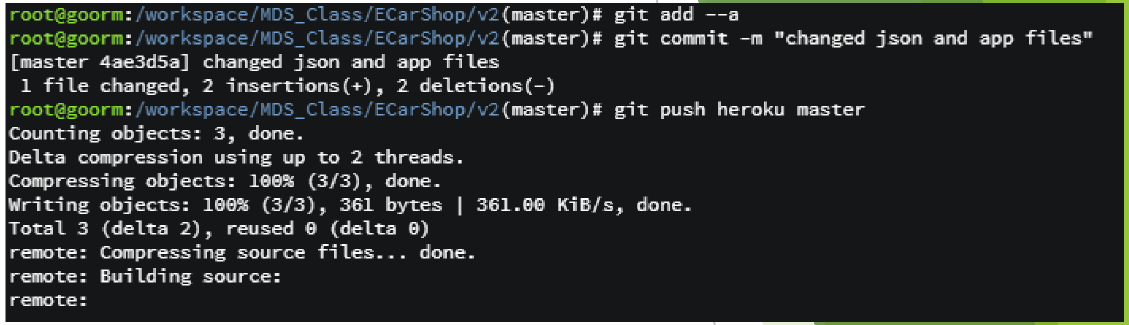


```
mongodb+srv://siddharthshekar:<password>@cluster0-uxtf6.mongodb.net/t
```

```
mongoose.connect("mongodb+srv://siddharthshekar:<password>@cluster0-uxtf6.mongodb.net/test?retryWrites=true&w=majority")
```

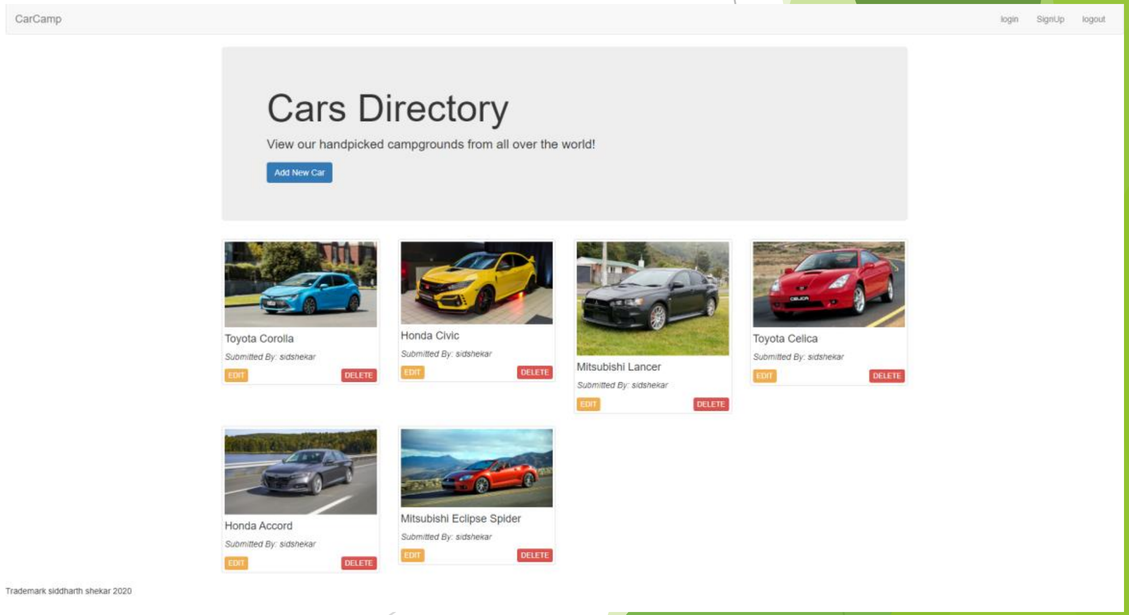
Final Deployment

- Since you made changes to the goorm project.
- Add the changes to git, commit
- Then push the project again to Heroku.



```
root@goorm:/workspace/MDS_Class/ECarShop/v2(master)# git add --a
root@goorm:/workspace/MDS_Class/ECarShop/v2(master)# git commit -m "changed json and app files"
[master 4ae3d5a] changed json and app files
 1 file changed, 2 insertions(+), 2 deletions(-)
root@goorm:/workspace/MDS_Class/ECarShop/v2(master)# git push heroku master
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 361 bytes | 361.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
```

- Refresh the Heroku webpage again. The page should work now.
- But now since you are using remote database there no current data present, so none of the cars info will load.
- Register as a new user and then add the database again manually.
- Or you can goto Cluster->Collections and add the data from there in mongodb.
- The web app is now live and ready to be shared with the world.



Cars Directory

View our handpicked campgrounds from all over the world!

Toyota Corolla
Submitted By: sidshekar

Honda Civic
Submitted By: sidshekar

Mitsubishi Lancer
Submitted By: sidshekar

Toyota Celica
Submitted By: sidshekar

Honda Accord
Submitted By: sidshekar

Mitsubishi Eclipse Spider
Submitted By: sidshekar

Trademark siddharth shekar 2020