Web Development

RESTful Routes, Databases, MongoDb and Mongoose

RESTful Routes

- In the last class we created a form to submit a new friend right in the page we were showing the list if friends.
- For basic understanding how Routes work that was fine, but in actual web development you need to have creation, reading, updating and deleting posts or content done in certain manner.
- All dynamic websites have Creation, Read, Updation and Deletion of post actions.
- Referred to as CRUD actions.
- A RESTful route is a route that provides mapping between HTTP verbs (get, post, put, etc) to control CRUD actions.
- https://learn.co/lessons/sinatra-restful-routes-readme

RESTful Routes

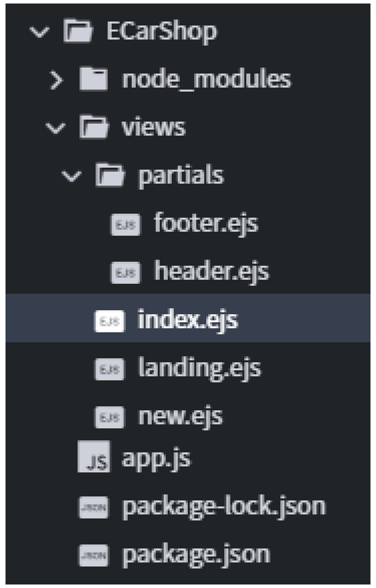
- The internet would be a really confusing place without a convention for how to handle URLS - to delete an Instagram photo might be www.instagram.com/delete-this-photo, but Twitter might be www.twitter.com/remove-this-tweet.
- Without a specific convention to follow, it would be hard to create new content, edit content, and delete it.
- REST – Representational State Transfer.
- RESTful routes provides a design pattern that allows for easy data manipulation.
- It's nicer for users and nicer for developers to have everything consistent.

RESTful Routes – Routes and Actions

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
PATCH	'/articles/:id'	update action	*modifies* an existing article 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

RESTful Routes – E Carshop Example

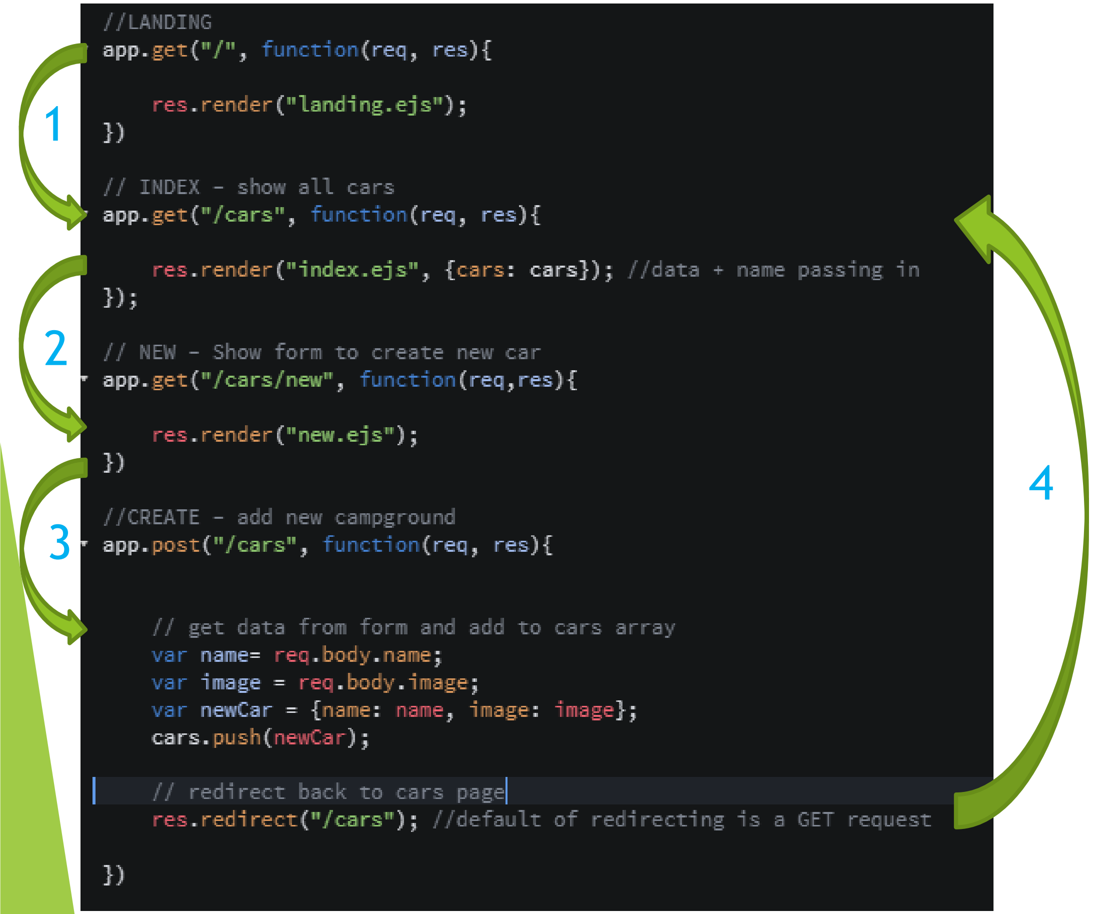
- Create a new project directory that will have a list of some Japanese car names and images.
- Landing page will be the main page.
- From Landing page user will click the button and will be taken to the cars route where the index page will be shown, displaying all the cars.
- In the index page there will be a button, Add A New Car which will go to /cars/new route, which will show the user new.ejs file.
- new.ejs will have a form in it so that the user can add a name and an image url.
- Once the form is submitted, will be taken to the cars post request.
- Cars post request will add the car to the array and redirect to the cars get request.



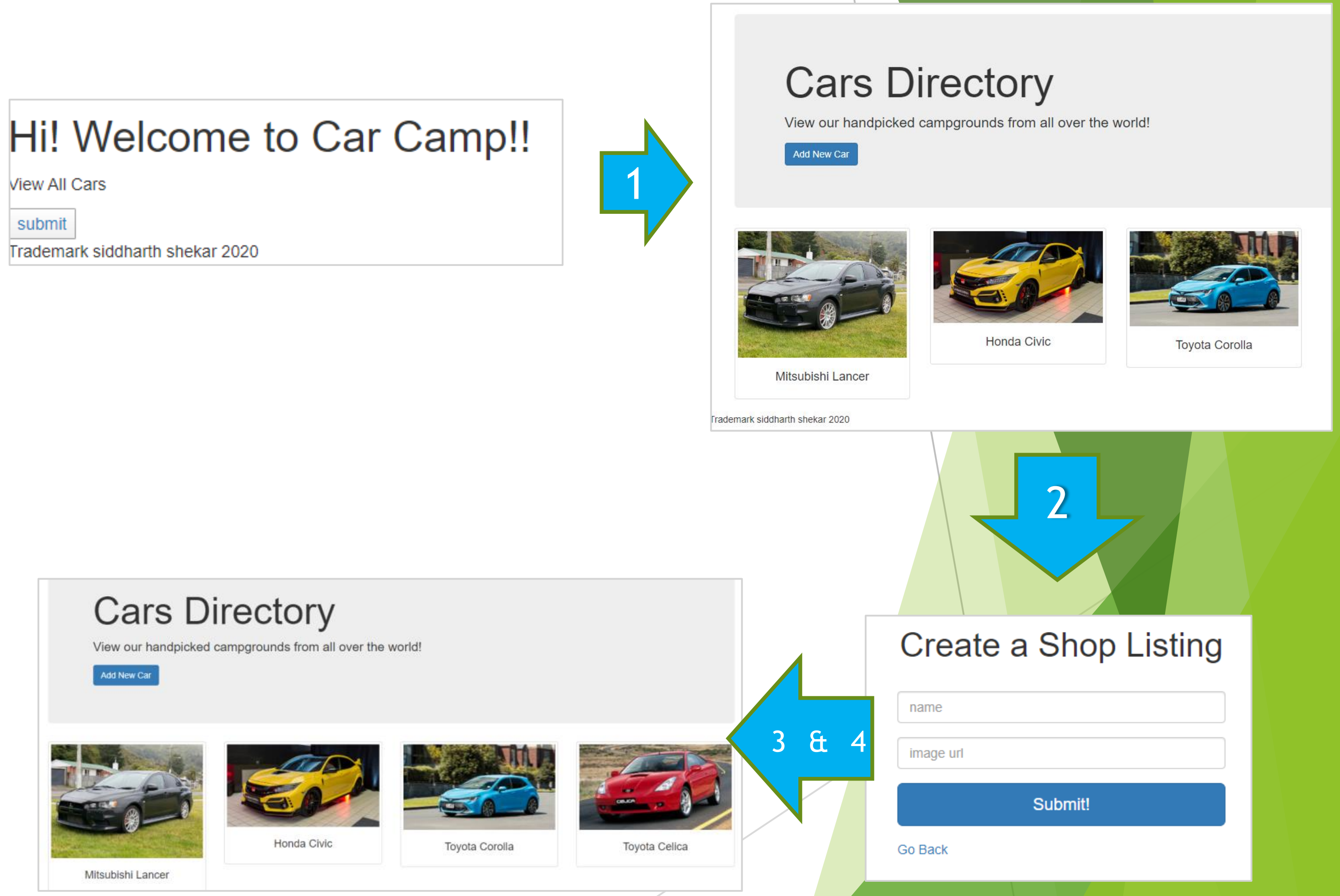
```
ECarShop
  node_modules
  views
    partials
      footer.ejs
      header.ejs
    index.ejs
    landing.ejs
    new.ejs
  app.js
  package-lock.json
  package.json
```

HTTP VERB	ROUTE	Action	Used For
GET	/cars'	index action	index page to display all cars
GET	/cars/new'	new action	displays create car form
POST	/cars'	create action	creates one car and adds to list of cars

E Carshop Webapp - Flow



```
//LANDING
app.get("/", function(req, res){

    res.render("landing.ejs");
})

// INDEX - show all cars
app.get("/cars", function(req, res){

    res.render("index.ejs", {cars: cars}); //data + name passing in
});

// NEW - Show form to create new car
app.get("/cars/new", function(req,res){

    res.render("new.ejs");
})

//CREATE - add new campground
app.post("/cars", function(req, res){


    // get data from form and add to cars array
    var name= req.body.name;
    var image = req.body.image;
    var newCar = {name: name, image: image};
    cars.push(newCar);

    // redirect back to cars page
    res.redirect("/cars"); //default of redirecting is a GET request

})
```



Hi! Welcome to Car Camp!!

View All Cars

submit

Trademark siddharth shekar 2020

Cars Directory

View our handpicked campgrounds from all over the world!

Trademark siddharth shekar 2020

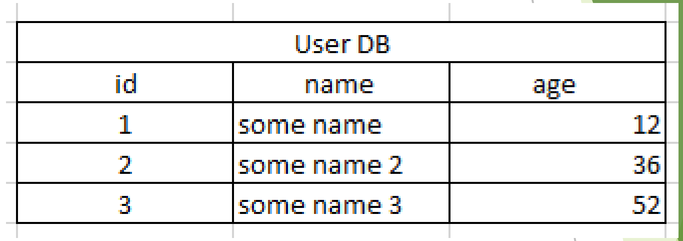
Create a Shop Listing

Databases

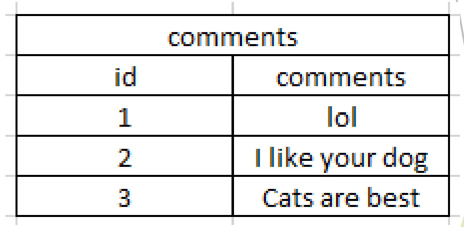
- A collection of information.
- You can create any type of database.
- It could be a collection of data like cars and images
- Or username information
- Databases can be of two types SQL or non SQL

SQL

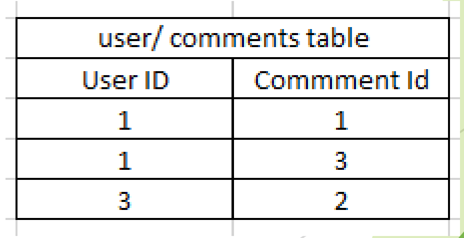
- SQL databases are relational.
- Meaning each DB is connected to other DB using a relation.
- You have one database which stores user data and another database with comments
- And you can create a SQL database connecting the user to the comments made by that user.
- Example: MySQL, Oracle, PostgreSQL



User DB		
id	name	age
1	some name	12
2	some name 2	36
3	some name 3	52



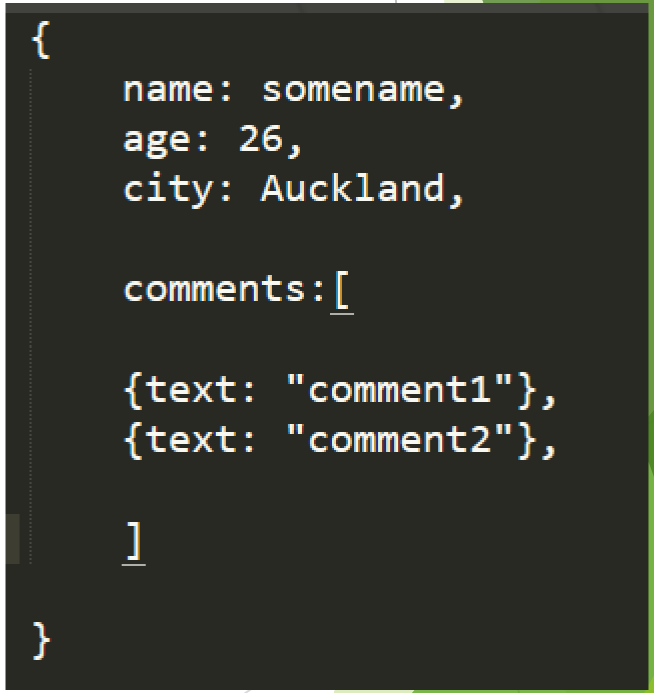
comments	
id	comments
1	lol
2	I like your dog
3	Cats are best



user/ comments table	
User ID	Commment Id
1	1
1	3
3	2

No SQL

- Data is stored with in the user itself.
- There isn’t a separate database connecting one type of database to the other.
- No SQL are document based instead of Table based.
- They are more scalable as you can add more servers to the DB.
- Preferred for larger and changing database
- Example: MongoDB, Bigtable, etc.



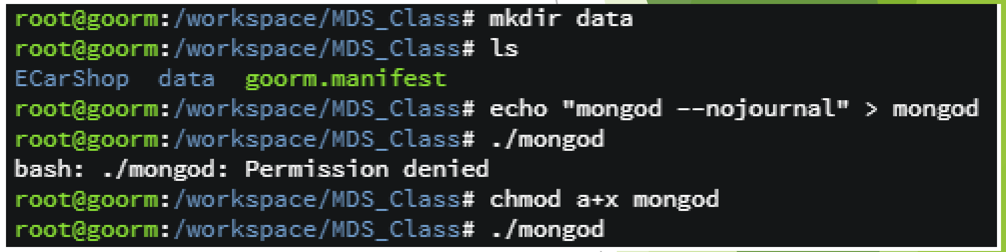
```
{
    name: somename,
    age: 26,
    city: Auckland,

    comments:[

    {text: "comment1"},
    {text: "comment2"},

    ]

}
```

Mongo DB

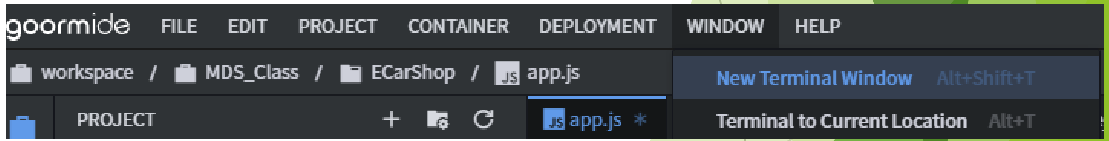
- While creating the new container we already installed MongoDB on Goorm.
- Using mongoDB we will be creating a local database and add data into it.
- For storing and retrieving data we have to start the mongo deamon before we add/ delete or retrieve the data locally.
- Also since we will be using the local database for testing it is better to have the data in a separate directory.

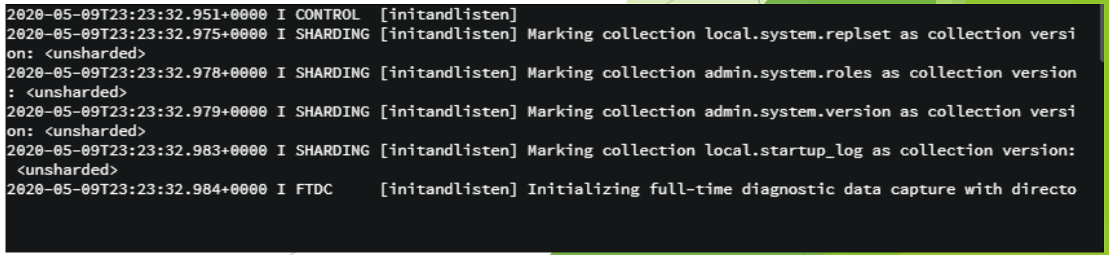
Mongo DB

- Create a new directory **outside** our current project directory.
- Then type **echo “mongod --nojournal” > mongod**
- To run the mongo deamon type ./mongod.
- This might give a permission denied error
- To remove it, type chmod a+x mongod
- Now when you type ./mongod the deamon should work now
- Ctrl + c to stop the deamon.
- It is better to create a new terminal window and run the deamon in it.
- Goto Window->New Terminal Window to create a new terminal window.
- Run the deamon from there by typing ./mongod
- Keep deamon running while working with databases



```
root@goorm:/workspace/MDS_Class# mkdir data
root@goorm:/workspace/MDS_Class# ls
ECarShop  data  goorm.manifest
root@goorm:/workspace/MDS_Class# echo "mongod --nojournal" > mongod
root@goorm:/workspace/MDS_Class# ./mongod
bash: ./mongod: Permission denied
root@goorm:/workspace/MDS_Class# chmod a+x mongod
root@goorm:/workspace/MDS_Class# ./mongod
```

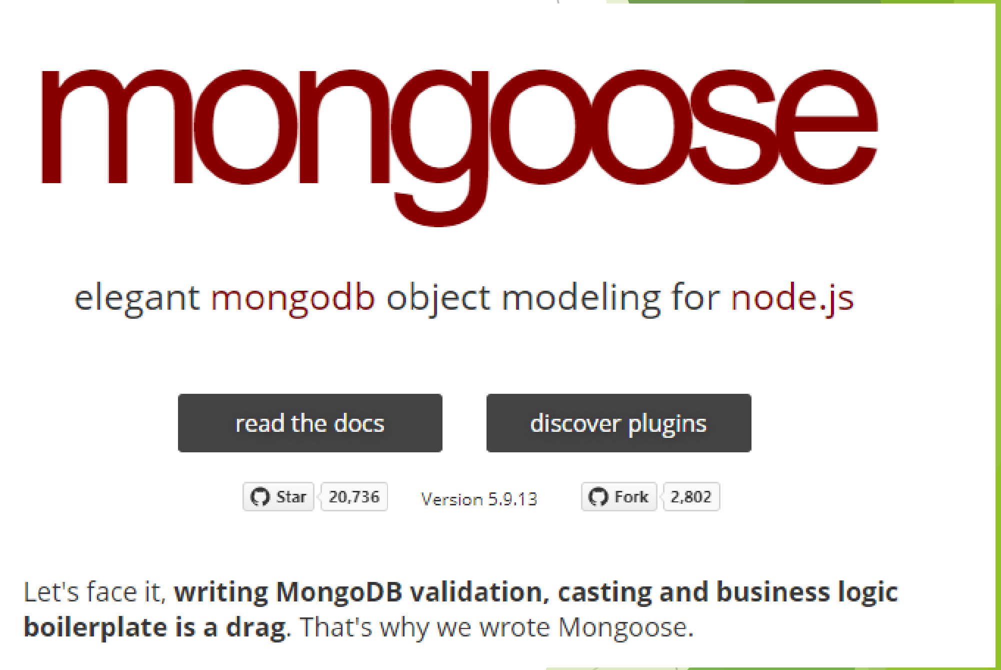

goormide
FILE
EDIT
PROJECT
CONTAINER
DEPLOYMENT
WINDOW
HELP
workspace / MDS_Class / ECarShop / app.js
New Terminal Window Alt+Shift+T
PROJECT
app.js *
Terminal to Current Location Alt+T



```
2020-05-09T23:23:32.951+0000 I CONTROL  [initandlisten]
2020-05-09T23:23:32.975+0000 I SHARDING [initandlisten] Marking collection local.system.replset as collection versi
on: <unsharded>
2020-05-09T23:23:32.978+0000 I SHARDING [initandlisten] Marking collection admin.system.roles as collection version
: <unsharded>
2020-05-09T23:23:32.979+0000 I SHARDING [initandlisten] Marking collection admin.system.version as collection versi
on: <unsharded>
2020-05-09T23:23:32.983+0000 I SHARDING [initandlisten] Marking collection local.startup_log as collection version:
 <unsharded>
2020-05-09T23:23:32.984+0000 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with directo
```

Mongoose

- For creating and organizing our data we will use the mongoose package.
- It is used for object modelling for node.js
- It uses schema based solution to model the application data.
- It includes built-in type casting, validation, query building out of the box.
- https://mongoosejs.com/

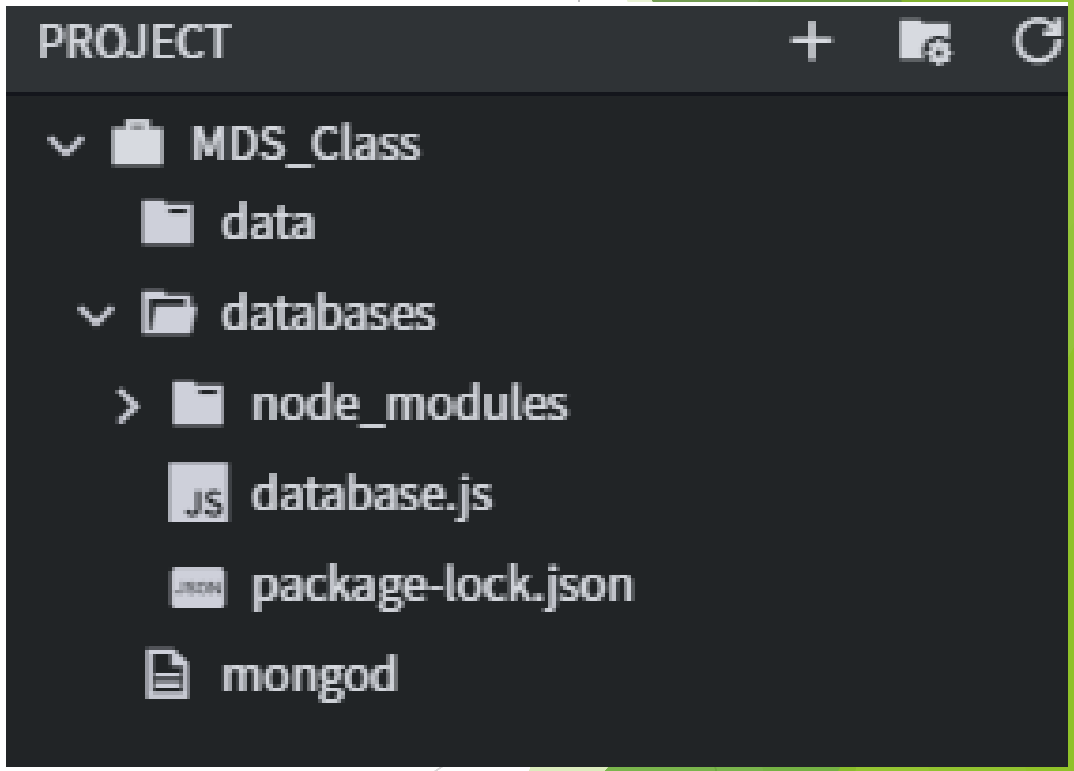


elegant mongodb object modeling for node.js

Let's face it, **writing MongoDB validation, casting and business logic boilerplate is a drag**. That's why we wrote Mongoose.

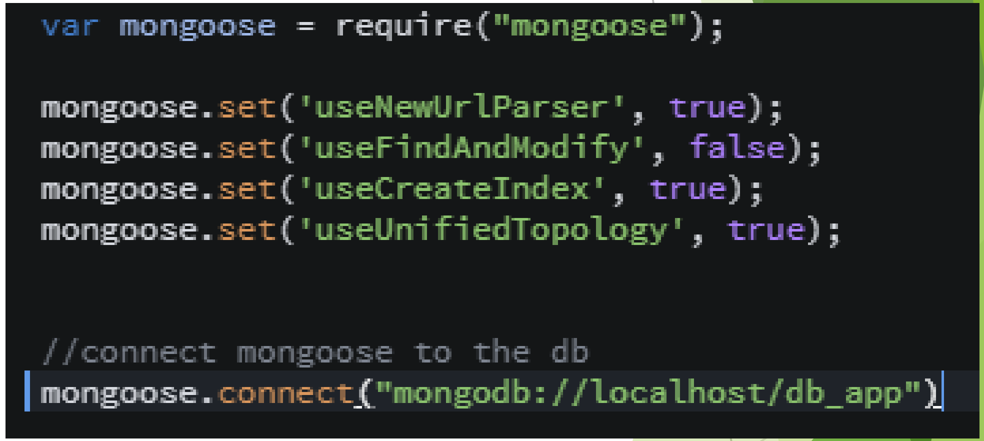
Installing Mongoose

- For using mongoose we need to install mongoose in our project.
- Create a new project directory called databases.
- We will use this project to understand how to create a DB, store and retrieve data.
- Install mongoose by calling npm install mongoose
- Create a new database.js file in it
- Run ./mongod in a separate terminal, if not running it already.



Connecting to local Database

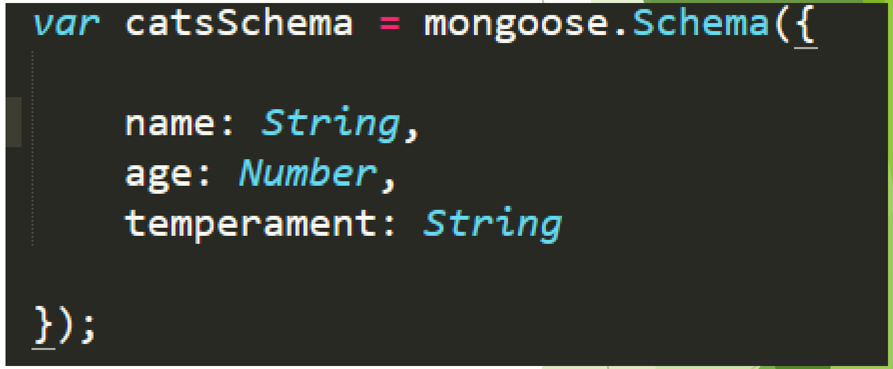
- In the database.js file require mongoose so that we can use mongoose and save it in a var called mongoose.
- We create a local database by calling mongoose.connect("mongodb://localhost/db_app")
- Here db_app is the name of our database for this application.
- You will create a new db for each application.
- We also need to set some parameters to remove the warning messages.
- So add that as well as shown.



```
var mongoose = require("mongoose");

mongoose.set('useNewUrlParser', true);
mongoose.set('useFindAndModify', false);
mongoose.set('useCreateIndex', true);
mongoose.set('useUnifiedTopology', true);


//connect mongoose to the db
mongoose.connect("mongodb://localhost/db_app")
```

Mongoose – Schema

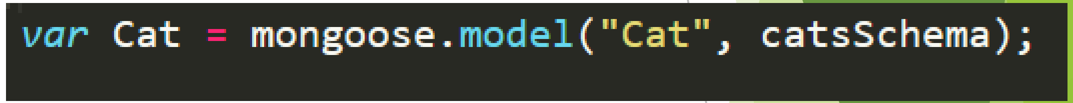
- A schema is used to structure our data in the database.
- Create a new variable called catsSchema and assign a schema to it by calling mongoose.schema which takes an object.
- Here we pass-in an object which will store a cats name, age and temperament.



```
var catsSchema = mongoose.Schema({

    name: String,
    age: Number,
    temperament: String

});
```

Mongoose - Models

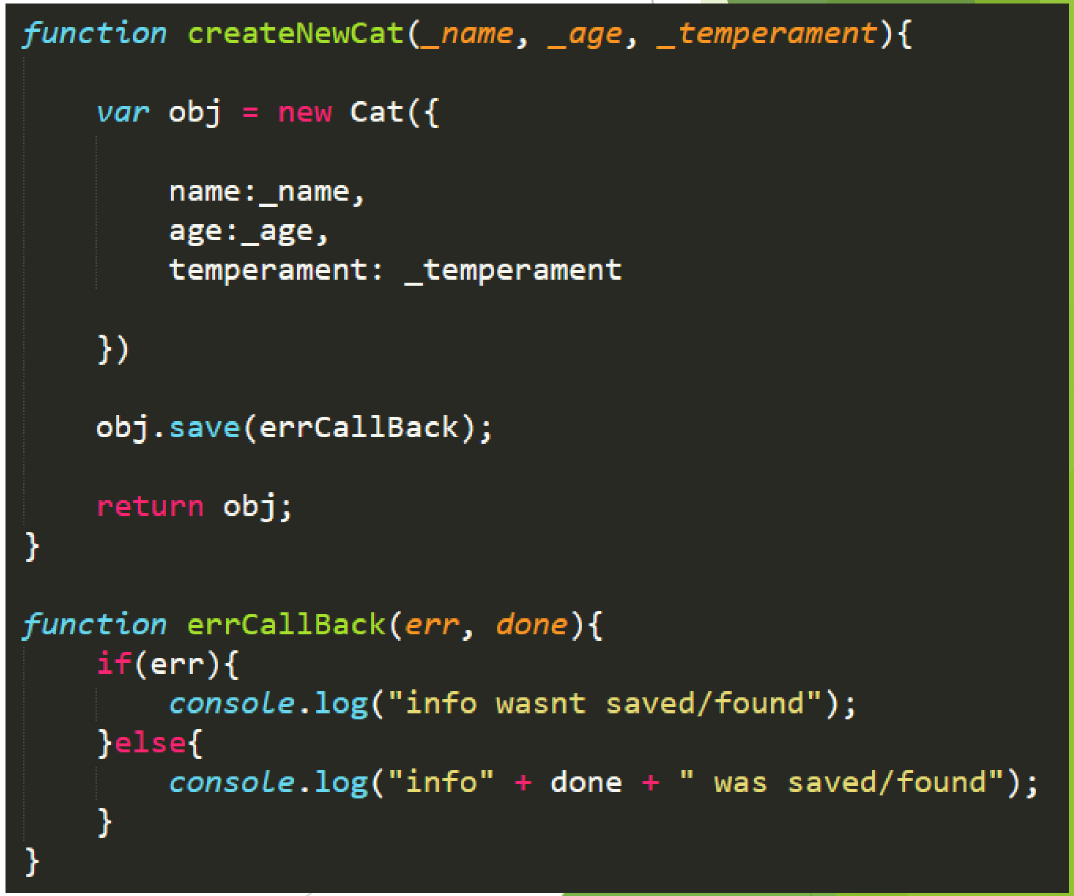
- Models are constructors compiled from Schema.
- Models are responsible for creating and reading data from the mongo database.
- Create a new model called Cat and pass in the schema to it and store it in a variable called Cat.



```
var Cat = mongoose.model("Cat", catsSchema);
```

Mongoose – save()

- We can create new variable called obj.
- We create a new Cat model by passing in the schema properties like name, age and temperament.
- And assign the new Cat model to variable obj.
- We can now save the new data to the database by calling save() on the obj variable.
- The save function can take a callback function in which can check if the data was stored in our database.
- The callback function takes 2 parameters, Error and the data saved.
- We can print out the errors or the data stored.

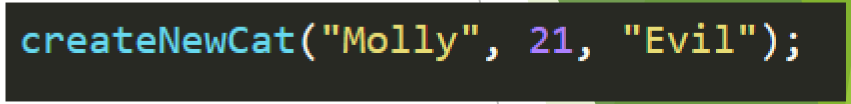


```
function createNewCat(_name, _age, _temperament){

    var obj = new Cat({

        name:_name,
        age:_age,
        temperament: _temperament

    })

    obj.save(errCallBack);

    return obj;
}

function errCallBack(err, done){
    if(err){
        console.log("info wasnt saved/found");
    }else{
        console.log("info" + done + " was saved/found");
    }
}
```

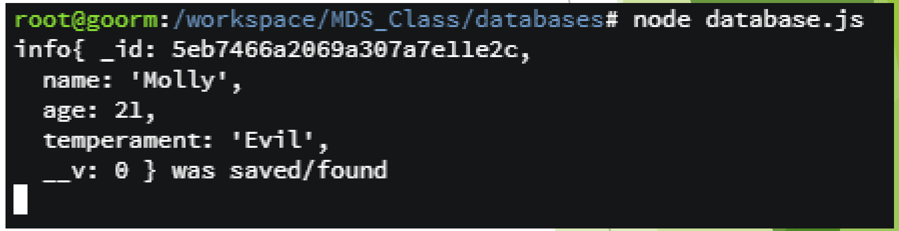
Mongoose – Data id

- At the end of the database.js
- Call the createNewCat function and pass a name, age and temperament.
- With the mongo deamon running on the other terminal.
- Call **node databases.js** to see the new data getting added to the database.

- Note that for each data added, the data is assigned a an **id** as well, which can be used to retrieve the data later on.



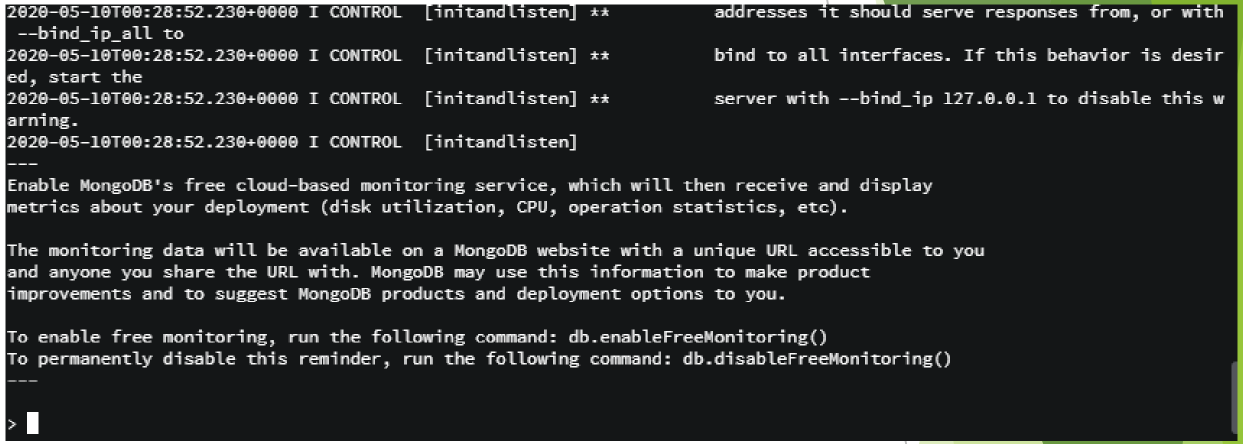
```
createNewCat("Molly", 21, "Evil");
```



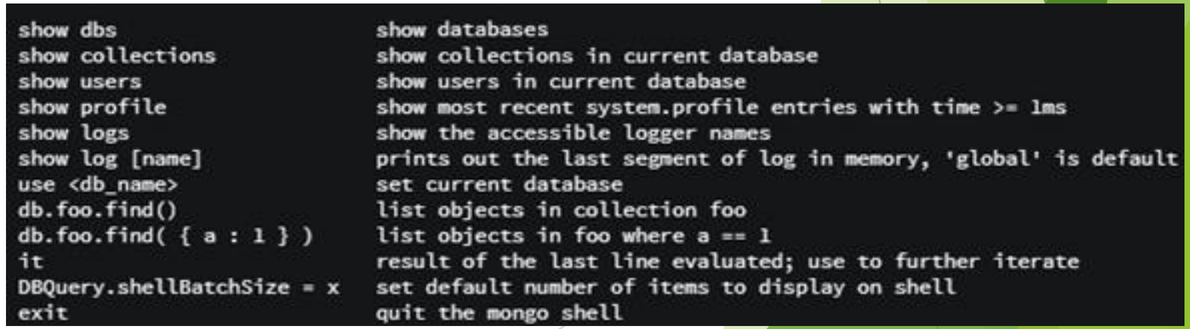
```
root@goorm:/workspace/MDS_Class/databases# node database.js
info{ _id: 5eb7466a2069a307a7e11e2c,
  name: 'Molly',
  age: 21,
  temperament: 'Evil',
  __v: 0 } was saved/found
```

Mongo- Shell

- Now we can look if the data was in fact added into the **db_app** database.
- This can done using the **mongo** shell.
- If you are running databases.js, ctrl+c to close it.
- With the mongo deamon running on the other terminal.
- In the primary terminal type ***mongo***.
- This will open the mongo shell
- In the shell you can type ***help*** to see all the commands you can type in the mongo shell.



```
2020-05-10T00:28:52.230+0000 I CONTROL  [initandlisten] **          addresses it should serve responses from, or with
 --bind_ip_all to
2020-05-10T00:28:52.230+0000 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desir
ed, start the
2020-05-10T00:28:52.230+0000 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this w
arning.
2020-05-10T00:28:52.230+0000 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

> 
```

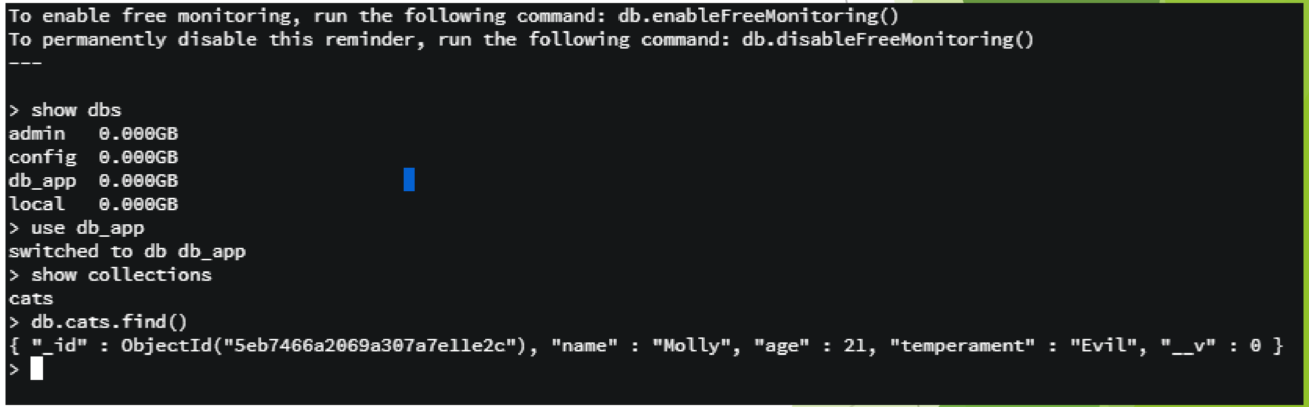


```
show dbs                     show databases
show collections             show collections in current database
show users                   show users in current database
show profile                 show most recent system.profile entries with time >= 1ms
show logs                    show the accessible logger names
show log [name]              prints out the last segment of log in memory, 'global' is default
use <db_name>                set current database
db.foo.find()                list objects in collection foo
db.foo.find( { a : 1 } )     list objects in foo where a == 1
it                           result of the last line evaluated; use to further iterate
DBQuery.shellBatchSize = x   set default number of items to display on shell
exit                         quit the mongo shell
```

Mongo – Shell commands

- To view our database
- Type **show dbs** to show all databases present
- This will show db_app database
- To access this database we need to type

  use <database name>
- So type: **use db_app**
- Now we can type show collections to show all the models stores in this data collection.
- This will show only the **cats** collection
- To see all entries stored in the cats collections
- Type: **db.cats.find()**

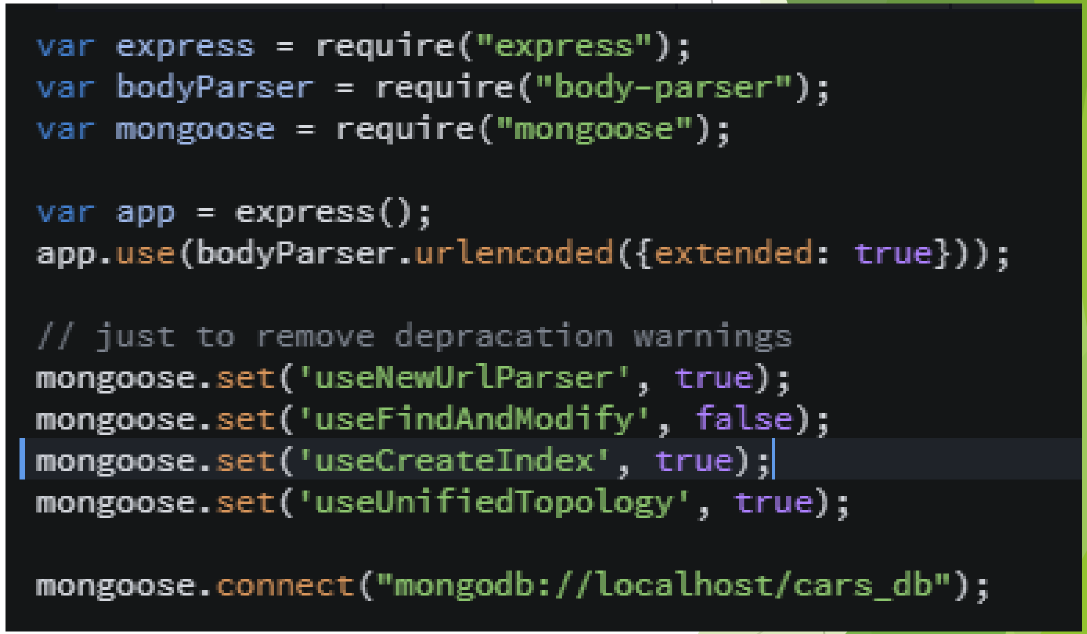


```
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

> show dbs
admin   0.000GB
config  0.000GB
db_app  0.000GB
local   0.000GB
> use db_app
switched to db db_app
> show collections
cats
> db.cats.find()
{ "_id" : ObjectId("5eb7466a2069a307a7e11e2c"), "name" : "Molly", "age" : 21, "temperament" : "Evil", "__v" : 0 }
>
```

Adding database to Car Shop project

- Go back to ECarShop project
- Install mongoose with –-save to add it to package.json file

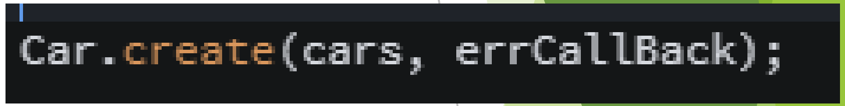
  npm install mongoose --save
- In app.js require mongoose, and save it to mongoose variable.
- Set variables to remove warnings.
- Call connect on mongoose and set the local database and call it **cars_db**



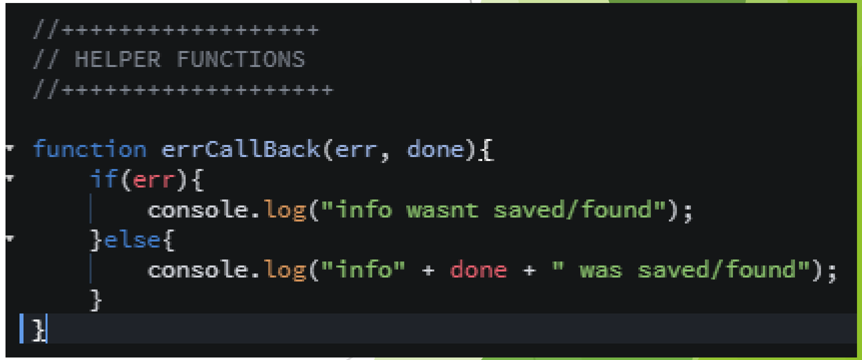
```
var express = require("express");
var bodyParser = require("body-parser");
var mongoose = require("mongoose");

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");
```

Adding Car Schema and Model

- Create carSchema using mongoose.schema to store name and image of each car.
- Create **Car** mongoose model by giving a name and schema.
- This is similar to how we created cats schema and model.
- Then call Car.create() and pass in the cars array and a callback function to check if the database was created properly.
- Now run the app.js to populate the database.
- Type ctrl+c to exit
- **Comment out Car.create()** as db is loaded and this is not required anymore
- Go to mongo shell and check if cars_db shows as collection and all the loaded data is present.
- If there is an error storing data properly. Call db.cars.remove({}) to remove all data from mongoose shell.



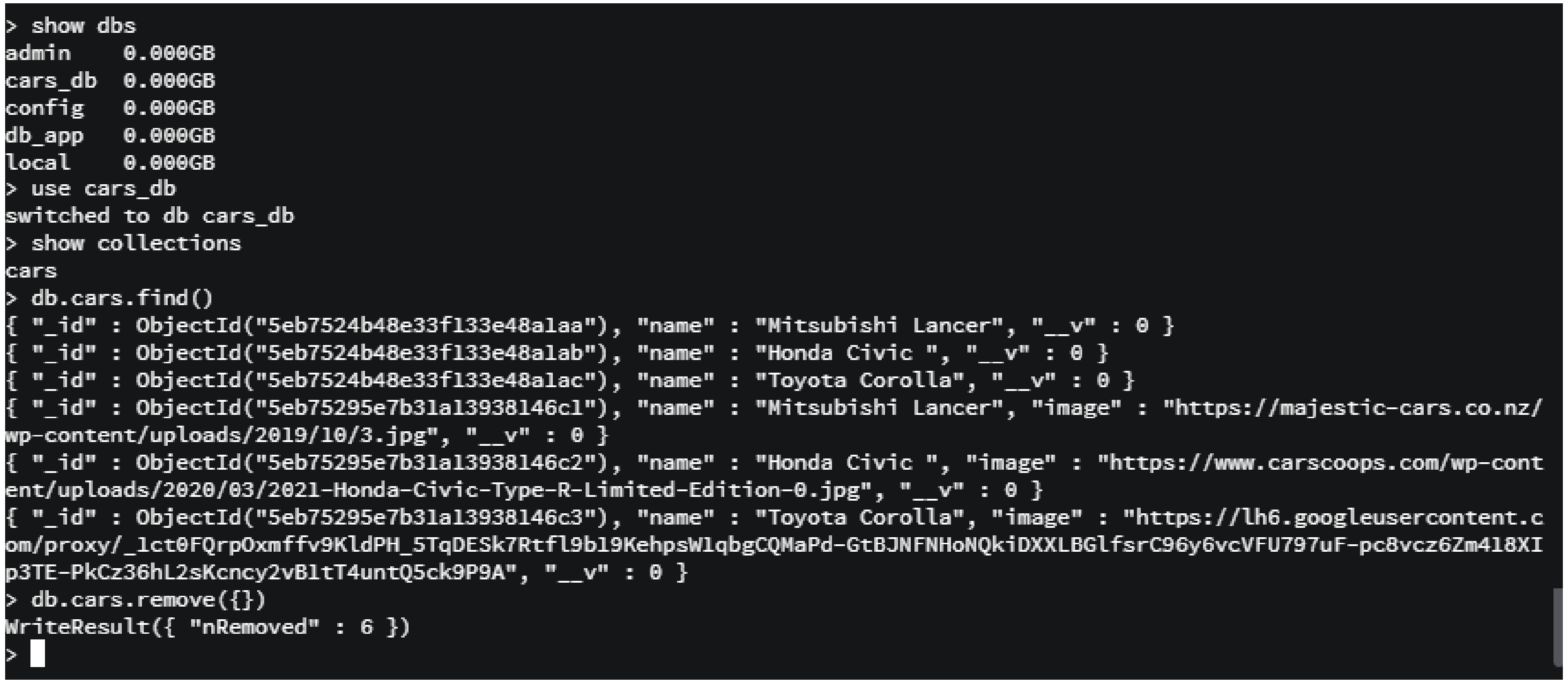
```
Car.create(cars, errCallBack);
```



```
//++++++++++++++++++
// HELPER FUNCTIONS
//+++++++++++++++++++

function errCallBack(err, done){
    if(err){
        console.log("info wasnt saved/found");
    }else{
        console.log("info" + done + " was saved/found");
    }
}
```

Removing data from collection

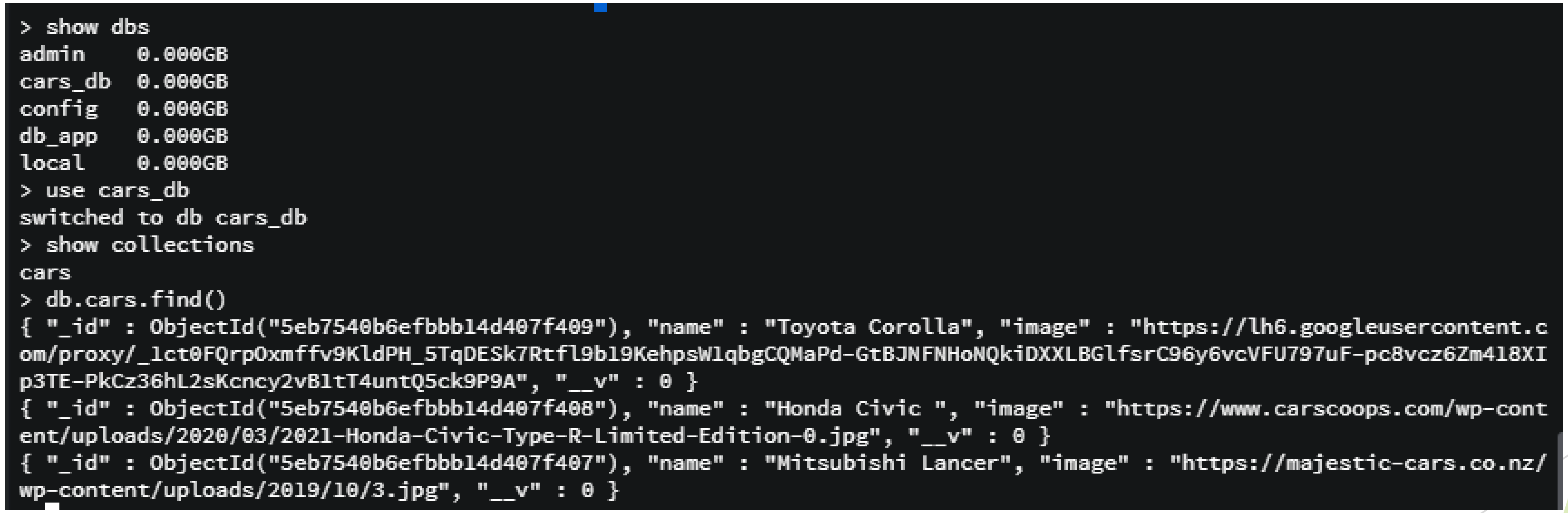
- If there is an error storing data properly. Call db.cars.remove({}) to remove all data.



```
> show dbs
admin     0.000GB
cars_db   0.000GB
config    0.000GB
db_app    0.000GB
local     0.000GB
> use cars_db
switched to db cars_db
> show collections
cars
> db.cars.find()
{ "_id" : ObjectId("5eb7524b48e33f133e48a1aa"), "name" : "Mitsubishi Lancer", "__v" : 0 }
{ "_id" : ObjectId("5eb7524b48e33f133e48a1ab"), "name" : "Honda Civic ", "__v" : 0 }
{ "_id" : ObjectId("5eb7524b48e33f133e48a1ac"), "name" : "Toyota Corolla", "__v" : 0 }
{ "_id" : ObjectId("5eb75295e7b31a13938146c1"), "name" : "Mitsubishi Lancer", "image" : "https://majestic-cars.co.nz/
wp-content/uploads/2019/10/3.jpg", "__v" : 0 }
{ "_id" : ObjectId("5eb75295e7b31a13938146c2"), "name" : "Honda Civic ", "image" : "https://www.carscoops.com/wp-cont
ent/uploads/2020/03/2021-Honda-Civic-Type-R-Limited-Edition-0.jpg", "__v" : 0 }
{ "_id" : ObjectId("5eb75295e7b31a13938146c3"), "name" : "Toyota Corolla", "image" : "https://lh6.googleusercontent.c
om/proxy/_1ct0FQrpOxmffv9KldPH_5TqDESk7Rtfl9bl9KehpsW1qbgCQMaPd-GtBJNFNHoNQkiDXXLBGlfsrC96y6vcVFU797uF-pc8vcz6Zm418XI
p3TE-PkCz36hL2sKcncy2vBltT4untQ5ck9P9A", "__v" : 0 }
> db.cars.remove({})
WriteResult({ "nRemoved" : 6 })
> 
```

Adding data from collection

- Make corrections and run app.js
- Check if Data is proper with name and image in mongo shell.

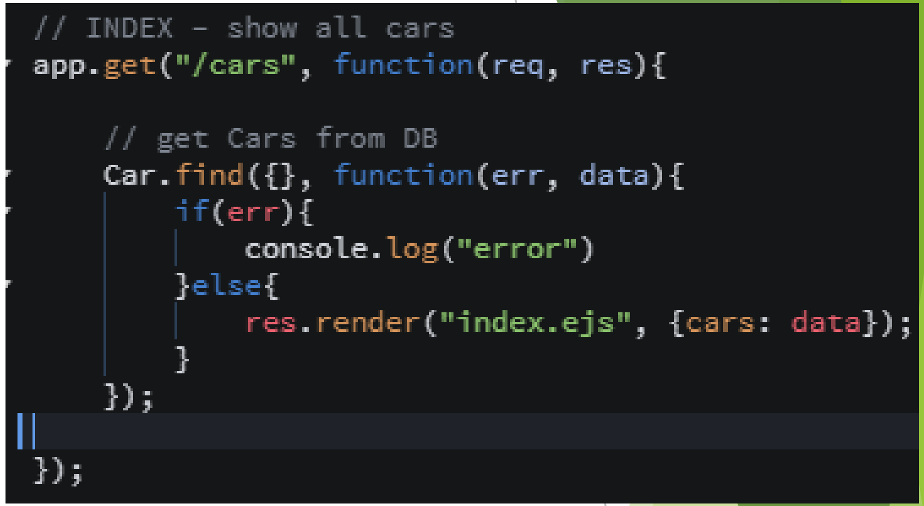


```
> show dbs
admin    0.000GB
cars_db  0.000GB
config   0.000GB
db_app   0.000GB
local    0.000GB
> use cars_db
switched to db cars_db
> show collections
cars
> db.cars.find()
{ "_id" : ObjectId("5eb7540b6efbbb14d407f409"), "name" : "Toyota Corolla", "image" : "https://lh6.googleusercontent.c
om/proxy/_1ct0FQrpOxmffv9KldPH_5TqDESk7Rtfl9bl9KehpsW1qbgCQMaPd-GtBJNFNHoNQkiDXXLBGlfsrC96y6vcVFU797uF-pc8vcz6Zm418XI
p3TE-PkCz36hL2sKcncy2vBltT4untQ5ck9P9A", "__v" : 0 }
{ "_id" : ObjectId("5eb7540b6efbbb14d407f408"), "name" : "Honda Civic ", "image" : "https://www.carscoops.com/wp-cont
ent/uploads/2020/03/2021-Honda-Civic-Type-R-Limited-Edition-0.jpg", "__v" : 0 }
{ "_id" : ObjectId("5eb7540b6efbbb14d407f407"), "name" : "Mitsubishi Lancer", "image" : "https://majestic-cars.co.nz/
wp-content/uploads/2019/10/3.jpg", "__v" : 0 }
```

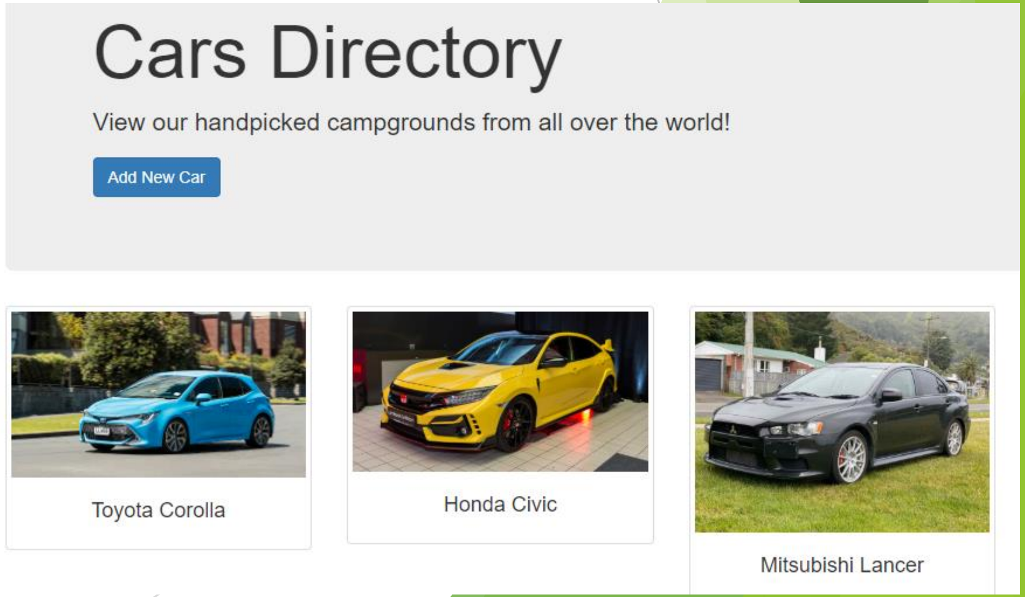
- Comment Car.create(cars, errCallBack); as data is loaded in DB.

Loading Cars from Database

- In the Cars get route, instead of loading data from the Cars array we can now load the Cars info from the database.
- Run node app.js
- Check if the index page is loading correctly.
- Now data is loaded from the database and not from the array.
- But even if you add a new car by pressing **Add New Car** and filling out the form and pressing submit.
- It will still show the 3 cars in the database as we haven't saved the new car data into the database.



```
// INDEX - show all cars
app.get("/cars", function(req, res){

    // get Cars from DB
    Car.find({}, function(err, data){
        if(err){
            console.log("error")
        }else{
            res.render("index.ejs", {cars: data});
        }
    });

});
```

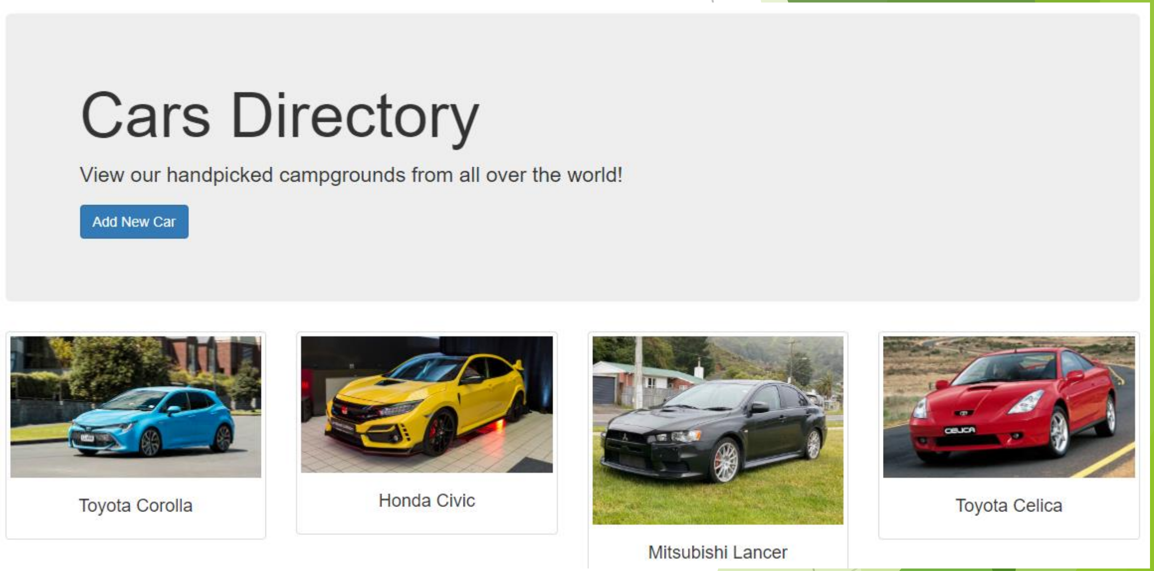


Cars Directory

View our handpicked campgrounds from all over the world!

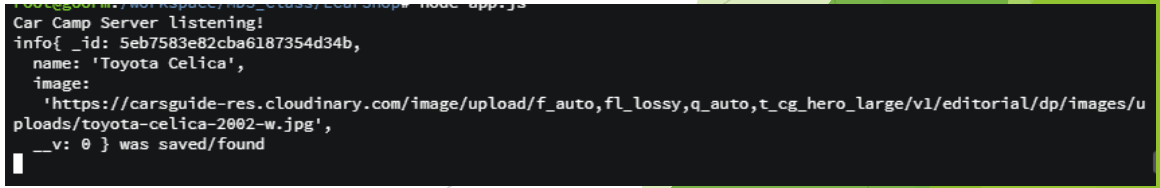
Adding Car into Database

- To save the information from the new car form,
- In the POST request, Instead of pushing the new car into the array,
- We create a new **Car** model and pass in the **newCar** variable into it.
- And Store it in a new variable called **car**.
- Then call car.save() and pass in the error call back function to store the new information into the database.
- Similar to how we saved the new cat into the database.
- Now when you add a new car, the new car data is stored in the database.
- Even after closing and restarting the app.js the data is not lost.
- The page will still show the new car added last time.



Cars Directory

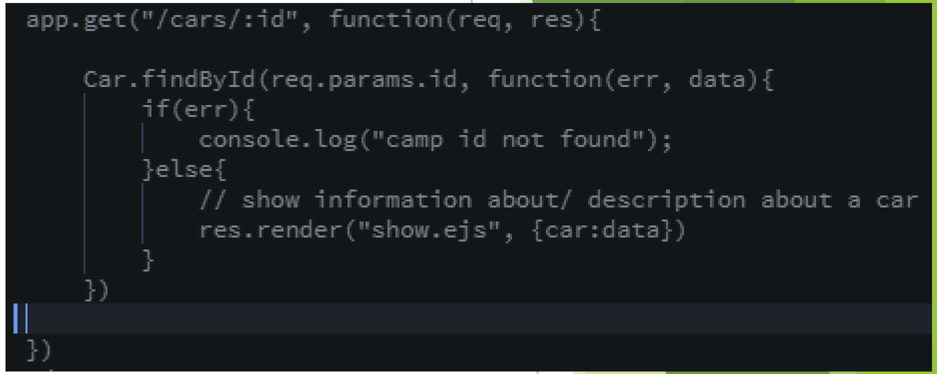
View our handpicked campgrounds from all over the world!



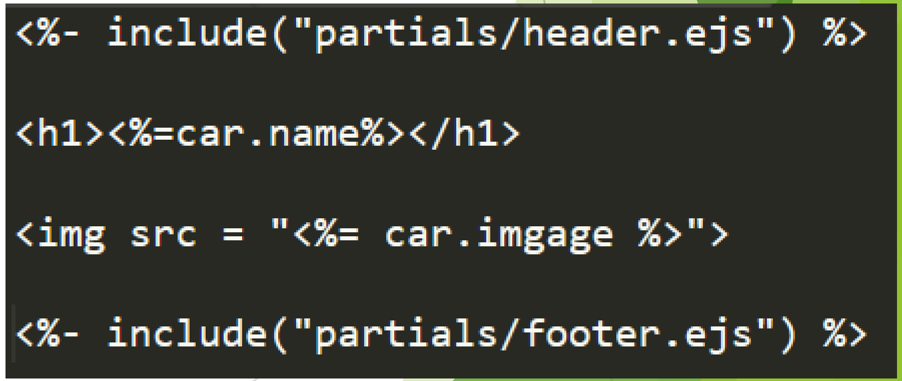
```
Car Camp Server listening!
info{ _id: 5eb7583e82cba6187354d34b,
  name: 'Toyota Celica',
  image:
   'https://carsguide-res.cloudinary.com/image/upload/f_auto,fl_lossy,q_auto,t_cg_hero_large/v1/editorial/dp/images/u
ploads/toyota-celica-2002-w.jpg',
  __v: 0 } was saved/found
```

Exercise

- After adding the cars to Database
- In the index.ejs file add a button called show Info under each car.
- Once the button is clicked, the href should go to "/cars/<%=cars._id%>"
- Create a new get request to path cars/:id
- In it use Car.findById() and get the data for the car
- Create a new Show.ejs file and pass the data to it which will show the Name and Image when the ShowInfo button is clicked on the index page.



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

    Car.findById(req.params.id, function(err, data){
        if(err){
            console.log("camp id not found");
        }else{
            // show information about/ description about a car
            res.render("show.ejs", {car:data})
        }
    })

})
```



```
<%- include("partials/header.ejs") %>

<h1><%=car.name%></h1>

<img src = "<%= car.imgage %>">

<%- include("partials/footer.ejs") %>
```