Recipe Backend App

We will build Rest Apis that can create, retrieve, update, delete and find Recipes by title.

You can install MongoDB Compass in your machine. The installation instructions can be

found at

Official MongoDB Compass installation manual

.

1.

Inside our Project folder, initialize the Node.js App with a

package.json

file:

npm init

name: (recipe-backend)

version: (1.0.0)

description: API with Node.js, Express and MongoDB

entry point: (index.js) server.js

test command:

git repository:

keywords:

author: Your\_name

license: (ISC)

Is this ok? (yes) yes

2.

We need to install necessary modules:

express

,

mongoose

,

body-parser

and

cors

.

Run the command:

npm install express mongoose body-parser cors --save

3.

In the root folder, let’s create a new

server.js

file:

const

express

=

require

(

"express"

);

const

bodyParser

=

require

(

"body-parser"

);

const

cors

=

require

(

"cors"

);

const

app

=

express

();

let

corsOptions

=

{

origin

:

"http://127.0.0.1:5500"

};

app

.

use

(

cors

(

corsOptions

));

// parse requests of content-type - application/json

app

.

use

(

bodyParser

.

json

());

// parse requests of content-type - application/x-www-form-urlencoded

app

.

use

(

bodyParser

.

urlencoded

({

extended

:

true

}));

// simple route

app

.

get

(

"/"

,

(

req

,

res

)

=>

{

res

.

json

({

message

:

"Welcome to backend application."

});

});

// set port, listen for requests

const

PORT

=

process

.

env

.

PORT

||

8080

;

app

.

listen

(

PORT

,

()

=>

{

console

.

log

(

`Server is running on port

${

PORT

}

.`

);

});

4.

Now let’s run the app with command:

node server.js

.

Open your browser with url

http://localhost:8080/

.

You will se:

"Welcome to backend application."

First step is done.

MongoDB database & Mongoose

We’re gonna work with MongoDB and Mongoose in this section.

5.

Create an

app

folder, then create a separate

config

folder for configuration

with

db.config.js

file like this:

module

.

exports

=

{

url:

'mongodb+srv://

your\_user:your\_psswd

@cluster0.ygnjz.azure.mongodb.net/

your-database

?

retryWrites=true&w=majority',

};

6.

Create

app

/

models

/

index.js

with the following code:

const

dbConfig

=

require

(

"../config/db.config.js"

);

const

mongoose

=

require

(

"mongoose"

);

mongoose

.

Promise

=

global

.

Promise

;

const

db

=

{};

db

.

mongoose

=

mongoose

;

db

.

url

=

dbConfig

.

url

;

db

.

recipes

=

require

(

"./recipe.model.js"

)(

mongoose

);

module

.

exports

=

db

;

7.

Now add

connect()

method in

server.js (after first app.use())

:

const

db

=

require

(

"./app/models"

);

db

.

mongoose

.

connect

(

db

.

url

,

{

useNewUrlParser

:

true

,

useUnifiedTopology

:

true

}).

then

(()

=>

{

console

.

log

(

"Connected to the database!"

);

}).

catch

(

err

=>

{

console

.

log

(

"Cannot connect to the database!"

,

err

);

process

.

exit

();

});

8.

In

models

folder, create

recipe.model.js

file like this:

module

.

exports

=

mongoose

=>

{

let

schema

=

mongoose

.

Schema

(

{

title

:

String

,

publisher

:

String

,

ingredients

:

Array,

source\_url: String,

image\_url: String,

servings: Number,

cooking\_time: Number

},

{

timestamps

:

true

}

);

schema

.

method

(

"toJSON"

,

function

()

{

const

{

\_\_v

,

\_id

,

...

object

}

=

this

.

toObject

();

object

.

id

=

\_id

;

return

object

;

});

const

Recipe

=

mongoose

.

model

(

"recipe"

,

schema

);

return

Recipe

;

};

After finishing the steps above, Model supports:

•

create a new Recipe: object.

save()

•

find a Recipe by id:

findById

(id)

•

retrieve all Recipes:

find()

•

update a Recipe by id:

findByIdAndUpdate

(id, data)

•

remove a Recipe:

findByIdAndRemove

(id)

•

remove all Recipes:

deleteMany()

•

find all Recipes by title: find({ title: { $regex: new RegExp(title), $options: “i” } })

These functions will be used in our Controller.

Create the Controller

9.

Inside

app

/

controllers

folder, let’s create

recipe.controller.js

with these CRUD functions:

•

create

•

findAll

•

findOne

•

update

•

delete

•

deleteAll

•

findAllPublished

const

db

=

require

(

"../models"

);

const

Recipe

=

db

.

recipes

;

Let’s implement these functions.

Create a new object

10.Create and Save a new Recipe:

exports

.

create

=

(

req

,

res

)

=>

{

// Validate request

if

(

!

req

.

body

.

title

)

{

res

.

status

(

400

).

send

({

message

:

"Content can not be empty!"

});

return

;

}

// Create a Recipe

const

recipe

=

new

Recipe

({

title

:

req

.

body

.

title

,

description

:

req

.

body

.

description

,

published

:

req

.

body

.

published

?

req

.

body

.

published

:

false

});

// Save Recipe in the database

recipe

.

save

(

recipe

)

.

then

(

data

=>

{

res

.

send

(

data

);

})

.

catch

(

err

=>

{

res

.

status

(

500

).

send

({

message

:

err

.

message

||

"Some error occurred while creating the Recipe."

});

});

};

Retrieve objects (with condition)

11.Retrieve all Recipes/ find by title from the database:

exports

.

findAll

=

(

req

,

res

)

=>

{

const

title

=

req

.

query

.

title

;

let

condition

=

title

?

{

title

:

{

$regex

:

new

RegExp

(

title

),

$options

:

"i"

}

}

:

{};

Recipe

.

find

(

condition

)

.

then

(

data

=>

{

res

.

send

(

data

);

})

.

catch

(

err

=>

{

res

.

status

(

500

).

send

({

message

:

err

.

message

||

"Some error occurred while retrieving recipes."

});

});

};

Retrieve a single object

12.Find a single Recipe with an

id

:

exports

.

findOne

=

(

req

,

res

)

=>

{

const

id

=

req

.

params

.

id

;

Recipe

.

findById

(

id

)

.

then

(

data

=>

{

if

(

!

data

)

res

.

status

(

404

).

send

({

message

:

"Not found Recipe with id "

+

id

});

else

res

.

send

(

data

);

})

.

catch

(

err

=>

{

res

.

status

(

500

)

.

send

({

message

:

"Error retrieving Recipe with id="

+

id

});

});

};

Update an object

13.Update a Recipe identified by the

id

in the request:

exports

.

update

=

(

req

,

res

)

=>

{

if

(

!

req

.

body

)

{

return

res

.

status

(

400

).

send

({

message

:

"Data to update can not be empty!"

});

}

const

id

=

req

.

params

.

id

;

Recipe

.

findByIdAndUpdate

(

id

,

req

.

body

,

{

useFindAndModify

:

false

})

.

then

(

data

=>

{

if

(

!

data

)

{

res

.

status

(

404

).

send

({

message

:

`Cannot update Recipe with id=

${

id

}

. Maybe Recipe was not

found!`

});

}

else

res

.

send

({

message

:

"Recipe was updated successfully."

});

})

.

catch

(

err

=>

{

res

.

status

(

500

).

send

({

message

:

"Error updating Recipe with id="

+

id

});

});

};

Delete an object

14.Delete a Recipe with the specified

id

:

exports

.

delete

=

(

req

,

res

)

=>

{

const

id

=

req

.

params

.

id

;

Recipe

.

findByIdAndRemove

(

id

)

.

then

(

data

=>

{

if

(

!

data

)

{

res

.

status

(

404

).

send

({

message

:

`Cannot delete Recipe with id=

${

id

}

. Maybe Recipe was not

found!`

});

}

else

{

res

.

send

({

message

:

"Recipe was deleted successfully!"

});

}

})

.

catch

(

err

=>

{

res

.

status

(

500

).

send

({

message

:

"Could not delete Recipe with id="

+

id

});

});

};

Delete all objects

15.Delete all Recipes from the database:

exports

.

deleteAll

=

(

req

,

res

)

=>

{

Recipe

.

deleteMany

({})

.

then

(

data

=>

{

res

.

send

({

message

:

`

${

data

.

deletedCount

}

Recipes were deleted successfully!`

});

})

.

catch

(

err

=>

{

res

.

status

(

500

).

send

({

message

:

err

.

message

||

"Some error occurred while removing all recipes."

});

});

};

Define Routes

When a client sends request for an endpoint using HTTP request (GET, POST, PUT, DELETE),

we need to determine how the server will reponse by setting up the routes.

These are our routes:

•

/api/recipes

: GET, POST, DELETE

•

/api/recipes/:id

: GET, PUT, DELETE

•

/api/recipes/published

: GET

17.Create a

recipe.routes.js

inside

app/routes

folder with content like this:

module

.

exports

=

app

=>

{

const

recipes

=

require

(

"../controllers/recipe.controller.js"

);

let

router

=

require

(

"express"

).

Router

();

router

.

post

(

"/"

,

recipes

.

create

);

// Create a new Recipe

router

.

get

(

"/"

,

recipes

.

findAll

);

// Retrieve all Recipes

router

.

get

(

"/:id"

,

recipes

.

findOne

);

// Retrieve a single Recipe with id

router

.

put

(

"/:id"

,

recipes

.

update

);

// Update a Recipe with id

router

.

delete

(

"/:id"

,

recipes

.

delete

);

// Delete a Recipe with id

router

.

delete

(

"/"

,

recipes

.

deleteAll

);

// Delete all Recipes

app

.

use

(

'/api/recipes'

,

router

);

};

You can see that we use a controller from

/controllers/recipe.controller.js

.

We also need to include routes in

server.js

(right before

app.listen()

):

require

(

"./app/routes/recipe.routes"

)(

app

);

Test the APIs

Run our Node.js application with command:

node server.js

.

Using Postman, we’re gonna test all the Apis above.

Install Postman from website

.

•

Create a new Recipe using

POST /recipes

Api

You can get sample data from

the page

https://forkify-api.herokuapp.com/v2

.

After creating some new Recipes, you can check MongoDb collection.

•

Retrieve a single Recipe by id using

GET /recipes/:id

Api

•

Update a Recipe using

PUT /recipes/:id

Api

Check

recipes

collection after some documents were updated:

•

Find all Recipes which title contains ‘chicken’:

GET /recipes?title=chicken

•

Delete a Recipe using

DELETE /recipes/:id

Api

•

Delete all Recipes using

DELETE /recipes

Api

Connecting Frontend

18.Copy your forkify Project to a new folder.

19. Change your url variable according to your new backend.

20. Get data from server.

21. Create and send a new recipe from DOM to the serve