🏠 Go back to the homepage (https://flaviocopes.com/)

Build a REST API using Node.js, Express and MongoDB

Published Feb 01 2022

Join the 2022 Full-Stack Web Dev Bootcamp! (https://bootcamp.dev)

Flavio Copes

YouTube 13 duizend

In this tutorial we'll create a REST API using Node.js (https://flaviocopes.com/page/node-handbook/) and Express (https://flaviocopes.com/page/express-handbook/) .

This API will expose a set of GET and POST endpoints to allow getting data out, and sending data in.

We'll use a MongoDB (https://flaviocopes.com/mongodb/) database to store this data.

> *Tip: make sure you installed a MongoDB database on your system before going on with the tutorial (or use a cloud MongoDB database if you prefer)*

Our task is to create a trip cost calculator app.

Imagine going on a trip, and you have your app (which can be a progressive web app, or a mobile app for example) where you add any expense you make. Gasoline, hotels, food, tickets and so on.

When the trip ends, you archive it and it becomes part of the history - which you can navigate and see how much you spent in the past trips.

We're not going to create the frontend of the application here, just the API.

Let's now dissect this into details, and translate it into a series of API endpoints.

An endpoint is a unique URL we will call to create an operation.

Like adding a new trip with its name.

At the beginning, there is no trip stored, and we need to add one. I imagine the app will ask the user for the name, and there will be a "Create trip" button. When clicked, the app will send the name to us, to the `/trip` endpoint with the `POST` HTTP method.

We have our first endpoint, which will accept a `name` property.

```
POST /trip { name }
```

Another endpoint will list the trips, and it's

```
GET /trips
```

By default, it will return the trips ordered by creation date.

When the user wants to add a new expense, the app will invoke the `/expense` endpoint with the POST method, with some parameters that describe the expense

```
POST /expense { trip, date, amount, category, description }
```

`trip` is the ID of the currently selected trip.

`category` is the name of the category of the expense. We'll provide a list of categories to choose from, which is static: `travel`, `food`, `accomodation`, `fun`.

When we want to retrieve a trip expenses, we call the `/expenses` endpoint with the GET method:

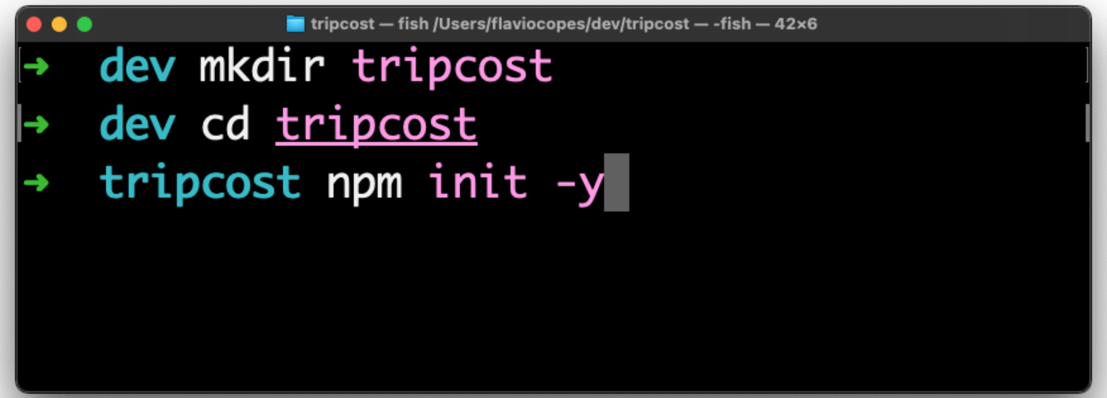
```
GET /expenses { trip }
```

passing the `trip` identifier.

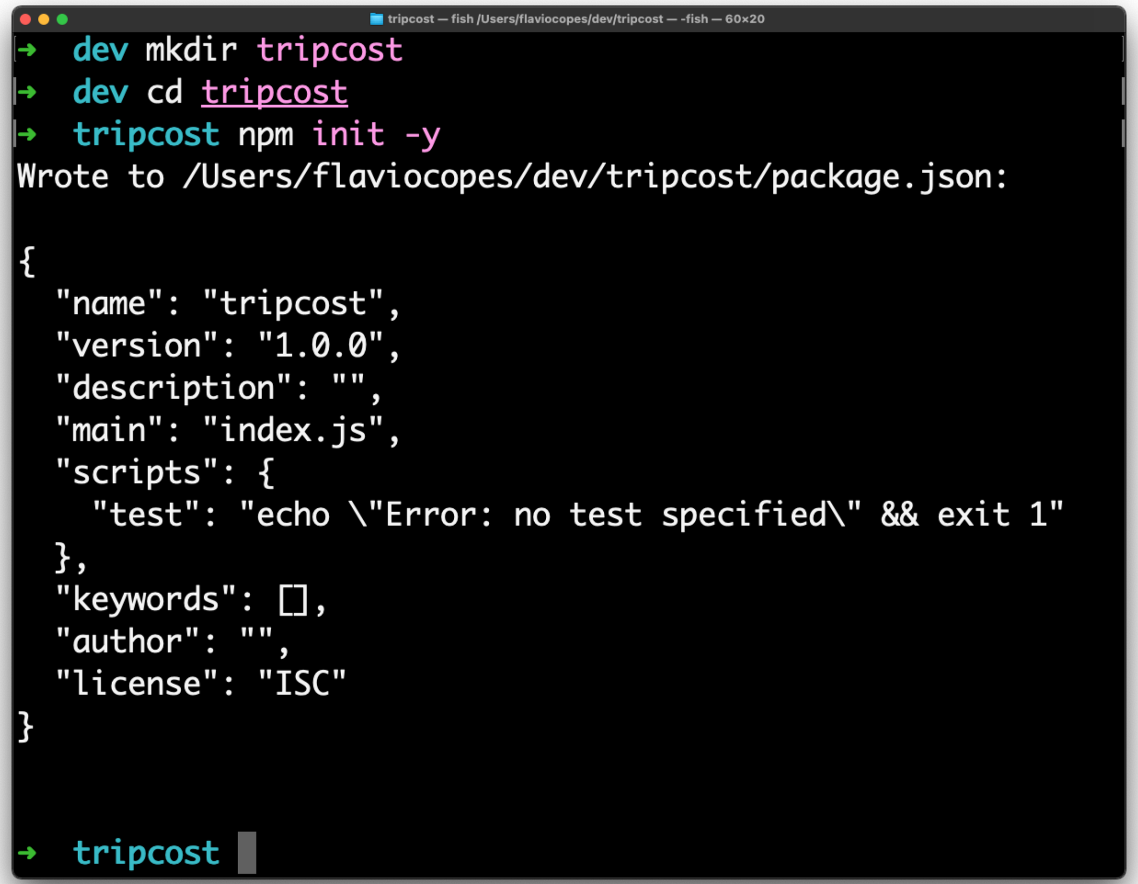
Let's start the project

I am going to use a local installation of Node.js.

We start our Node.js project by going into a new folder (call it `tripcost`) and typing the command `npm init -y`.


tripcost — fish /Users/flaviocopes/dev/tripcost — -fish — 42×6

```
➜  dev mkdir tripcost
➜  dev cd tripcost
➜  tripcost npm init -y
```


tripcost — fish /Users/flaviocopes/dev/tripcost — -fish — 60×20

```
→  dev mkdir tripcost
→  dev cd tripcost
→  tripcost npm init -y
Wrote to /Users/flaviocopes/dev/tripcost/package.json:

{
  "name": "tripcost",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}


→  tripcost
```

We'll use MongoDB as our database.

Install the `mongodb` Node.js package with:

```
npm install mongodb
```

While you're here, also install Express:

```
npm install express
```

Create a `server.js` file now, where we'll store our API code, and start requiring Express and MongoDB:

```
const express = require("express")
const mongo = require("mongodb").MongoClient
```

Initialize the Express app:

```
const app = express()
```

And now we can add the stubs for the API endpoints we support:

```
app.post("/trip", (req, res) => {
  /* */
})
app.get("/trips", (req, res) => {
  /* */
})
app.post("/expense", (req, res) => {
  /* */
})
app.get("/expenses", (req, res) => {
  /* */
})
```

Finally, use the `listen()` method on `app` to start the server:

```
app.listen(3000, () => console.log("Server ready"))
```

You can run the application using `node server.js` in the project folder.

Adding trips

We offer the client a way to add trip using the `POST /trip` endpoint:

```
app.post("/trip", (req, res) => {
  /* */
})
```

Let’s go ahead and implement it.

We already included the MongoDB library, so we can use it in our endpoint implementation:

```
const mongo = require("mongodb").MongoClient
```

Next, we build the MongoDB server URL. If you are running the project locally, and MongoDB locally too, the URL is likely this:

```
const url = "mongodb://localhost:27017"
```

because `27017` is the default port.

Next, let’s connect to the database using `connect()`:

```
let db

mongo.connect(
  url,
  {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  },
  (err, client) => {
    if (err) {
      console.error(err)
      return
    }
    db = client.db("tripcost")
  }
)
```

and while we're here, let's also get a reference to the *trips* and *expenses* collections:

```
let db, trips, expenses

mongo.connect(
  url,
  {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  },
  (err, client) => {
    if (err) {
      console.error(err)
      return
    }
    db = client.db("tripcost")
    trips = db.collection("trips")
    expenses = db.collection("expenses")
  }
)
```

Now we can go back to our endpoint.

This endpoint expects 1 parameter, `name`, which represents how we call our trip. For example *Sweden 2018* or *Yosemite August 2018*.

> *See my tutorial on [how to retrieve the POST query parameters using Express (https://flaviocopes.com/express-post-query-variables/)](https://flaviocopes.com/express-post-query-variables/)*

We expect data coming as JSON, using the `Content-Type: application/json`, so we need to use the `express.json()` middleware:

```
app.use(express.json())
```

We can now access the data by referencing it from `Request.body`:

```
app.post("/trip", (req, res) => {
  const name = req.body.name
})
```

Once we have the name, we can use the `trips.insertOne()` method to add the trip to the database:

```
app.post("/trip", (req, res) => {
  const name = req.body.name
  trips.insertOne({ name: name }, (err, result) =>
{})
})
```

> *If you get an error like "could not read property insertOne of undefined", make sure `trips` is successfully set in `mongo.connect()`. Add a console.log(trips) before calling insertOne() to make sure it contains the collection object.*

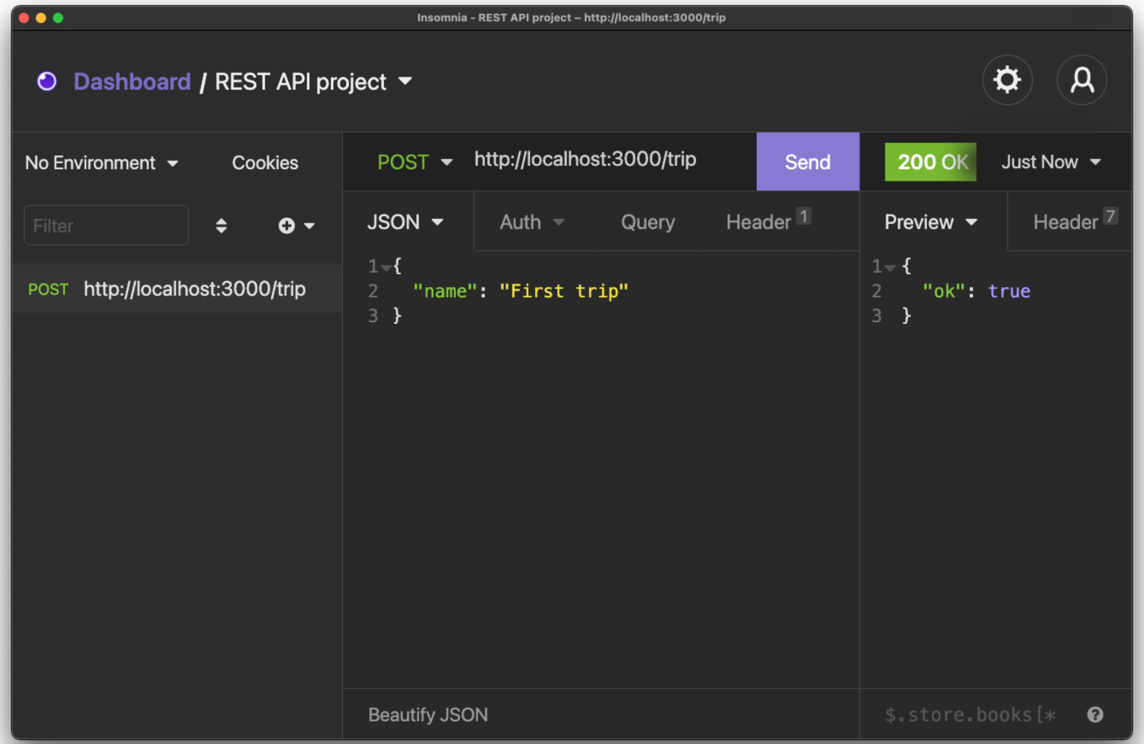
We handle the error, if present in the `err` variable, otherwise we send a 200 response (successful) to the client, adding an `ok: true` message in the JSON response:

```
app.post("/trip", (req, res) => {
  const name = req.body.name
  trips.insertOne({ name: name }, (err, result) =>
{
    if (err) {
      console.error(err)
      res.status(500).json({ err: err })
      return
    }
    console.log(result)
    res.status(200).json({ ok: true })
  })
})
```

That's it!

Now restart the Node application by hitting `ctrl-C` to stop it, and run it again.

You can test this endpoint using the Insomnia application (https://insomnia.rest) , a great way to test and interact with REST endpoints:


Insomnia - REST API project – http://localhost:3000/trip
Dashboard / REST API project
No Environment
Cookies
Filter
POST http://localhost:3000/trip
POST
http://localhost:3000/trip
Send
200 OK
Just Now
JSON
Auth
Query
Header 1
Preview
Header 7
Beautify JSON
$.store.books[*

```
{
  "name": "First trip"
}
```

```
{
  "ok": true
}
```

List the trips

The list of trips is returned by the `GET /trips` endpoint. It accepts no parameters:

```
app.get("/trips", (req, res) => {
  /* */
})
```

We already initialized the `trips` collection, so we can directly access that to get the list.

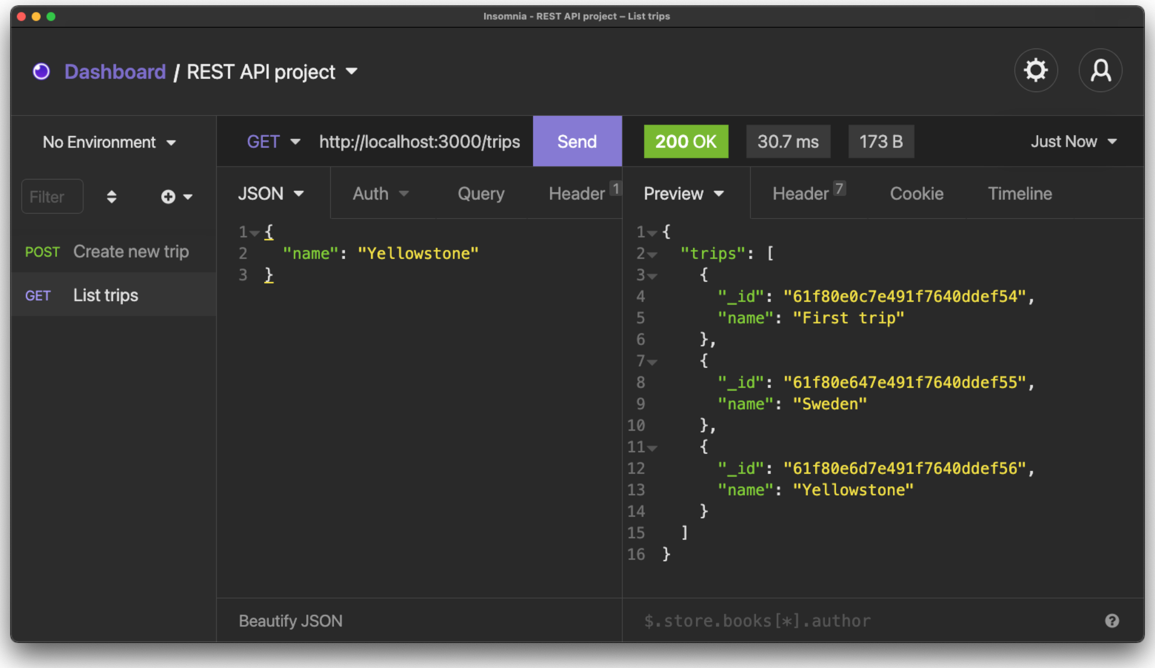
We use the `trips.find()` method, which result we must convert to an array using `toArray()`:

```
app.get("/trips", (req, res) => {
  trips.find().toArray((err, items) => {})
})
```

Then we can handle the `err` and `items` results:

```
app.get("/trips", (req, res) => {
  trips.find().toArray((err, items) => {
    if (err) {
      console.error(err)
      res.status(500).json({ err: err })
      return
    }
    res.status(200).json({ trips: items })
  })
})
```

Here's the result of the API call in Insomnia:



```
{
  "name": "Yellowstone"
}
```

```
{
  "trips": [
    {
      "_id": "61f80e0c7e491f7640ddef54",
      "name": "First trip"
    },
    {
      "_id": "61f80e647e491f7640ddef55",
      "name": "Sweden"
    },
    {
      "_id": "61f80e6d7e491f7640ddef56",
      "name": "Yellowstone"
    }
  ]
}
```

Add an expense

We previously got the list of trips. Every trip has an associated `_id` property which is added by MongoDB directly when it's added:

```
{
  "trips": [
    {
      "_id": "5bdf03aed64fb0cd04e15728",
      "name": "Yellowstone 2018"
    },
    {
      "_id": "5bdf03c212d45cdb5ccec636",
      "name": "Sweden 2017"
    },
    {
      "_id": "5bdf047ccf4f42dc368590f6",
      "name": "First trip"
    }
  ]
}
```

We'll use this `_id` to register a new expense.

If you remember, the endpoint to add a new expense is this:

```
POST /expense { trip, date, amount, category, description }
```

`trip` in this case will be the `_id` of one of the trips we previously registered. Imagine that in the app, the user will add one trip, and that will remain the current trip until a new one is added (or selected).

Let's go ahead and implement our stub:

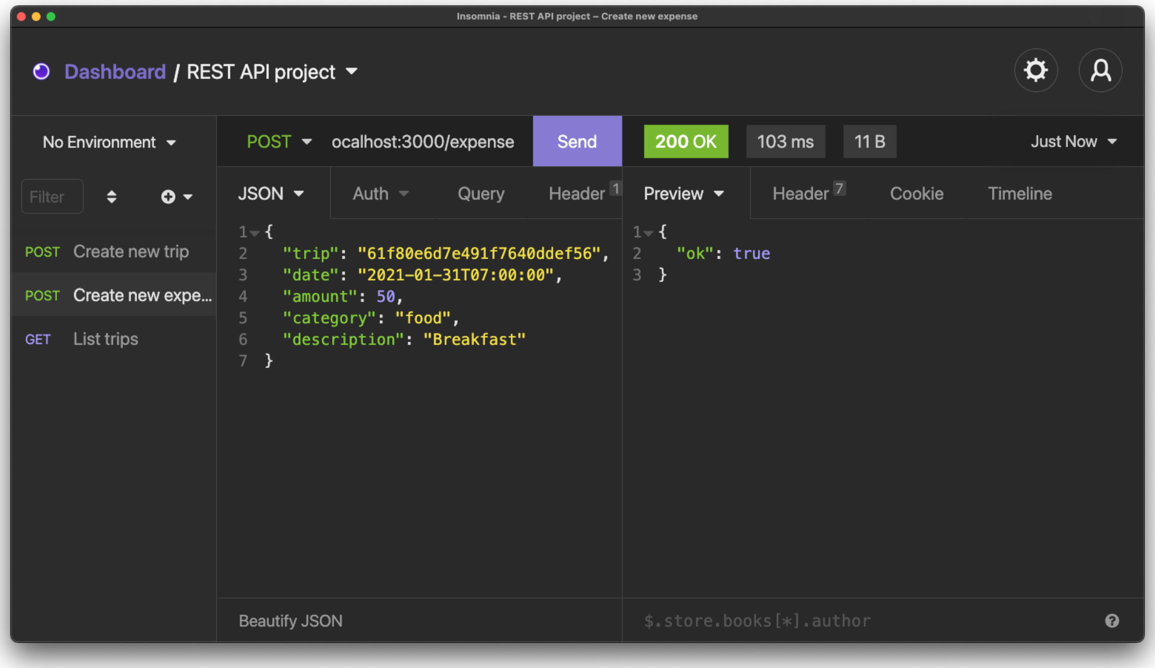
```
app.post("/expense", (req, res) => {
  /* */
})
```

Like when adding a trip, we're going to use the insertOne() method, this time on the `expenses` collection.

We get 5 parameters from the request body:

- `trip`
- `date` the date, in ISO 8601 format (e.g. `2018-07-22T07:22:13`), in the GMT timezone
- `amount` an integer with the amount
- `category` which is one from `travel`, `food`, `accomodation`, `fun`
- `description` a description for the expense, so we'll remember about it later

```
app.post("/expense", (req, res) => {
expenses.insertOne(
{
  trip: req.body.trip,
  date: req.body.date,
  amount: req.body.amount,
  category: req.body.category,
  description: req.body.description,
},
(err, result) => {
  if (err) {
    console.error(err)
    res.status(500).json({ err: err })
    return
  }
  res.status(200).json({ ok: true })
}
)
})
```


Insomnia - REST API project – Create new expense
Dashboard / REST API project
No Environment
POST ocalhost:3000/expense Send
200 OK 103 ms 11 B Just Now
Filter
JSON Auth Query Header 1
Preview Header 7 Cookie Timeline
POST Create new trip
POST Create new expe...
GET List trips
Beautify JSON
$.store.books[*].author

```
{
  "trip": "61f80e6d7e491f7640ddef56",
  "date": "2021-01-31T07:00:00",
  "amount": 50,
  "category": "food",
  "description": "Breakfast"
}
```

```
{
  "ok": true
}
```

List all expenses

The last piece of the puzzle is getting the expenses.

We need to fill the `/expenses` endpoint stub, the last one missing:

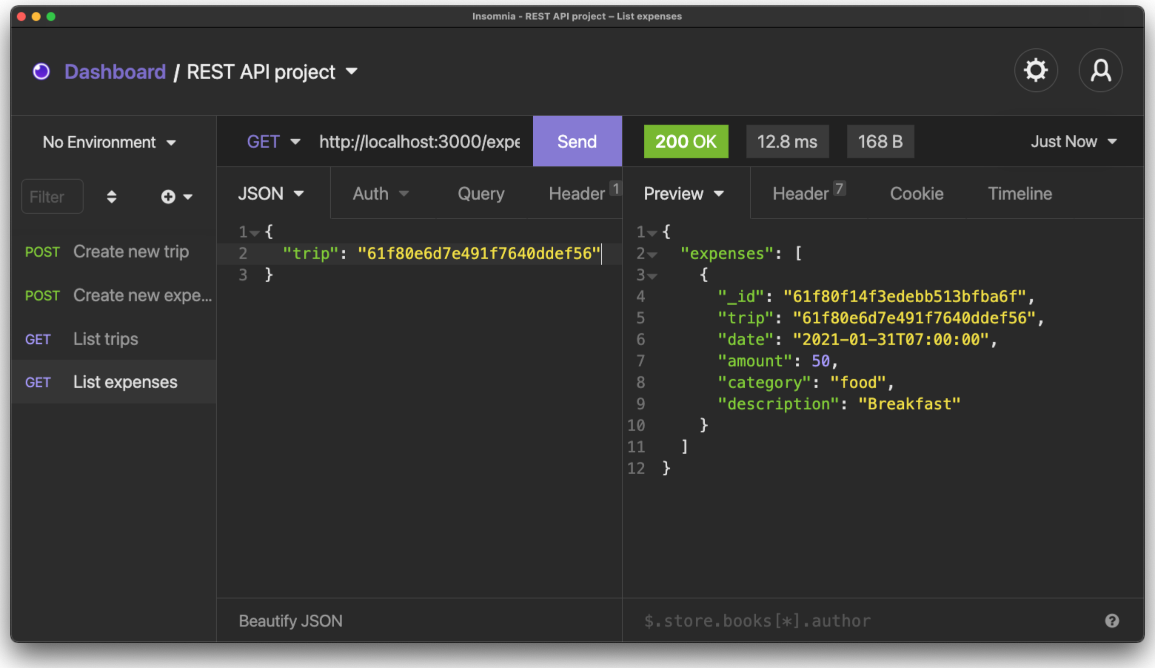
```
app.get("/expenses", (req, res) => {
  /* */
})
```

This endpoint accepts the `trip` parameter, which is the `_id` property of a trip stored in the database.

```
app.get("/expenses", (req, res) => {
  expenses.find({ trip: req.body.trip }).toArray((e
rr, items) => {
    if (err) {
      console.error(err)
      res.status(500).json({ err: err })
```

```
        return
      }
      res.status(200).json({ expenses: items })
    })
  })
```

This Insomnia screenshot shows it at work:



```
{
  "trip": "61f80e6d7e491f7640ddef56"
}
```

```
{
  "expenses": [
    {
      "_id": "61f80f14f3edebb513bfba6f",
      "trip": "61f80e6d7e491f7640ddef56",
      "date": "2021-01-31T07:00:00",
      "amount": 50,
      "category": "food",
      "description": "Breakfast"
    }
  ]
}
```

You might also like