CREATE

The first route we write will handle resource creation a specifically creating a new City in our database. Let's move into our cities.js file inside our routes folder to get started.

Part 1: Writing Create Route

Step 1: At the top of your file import express and create an instance of express.Router set to a constant.

```
const express = require('express')
const router = express.Router()
```

Step 2: Import the City model that we created in the last tutorial.

```
const City = require('../../models/City')
```

Step 3: Begin coding the route method for creating City objects.

```
router.post('/', (req, res) => {
// We make a call to router.post and pass in two arguments. The
first is our API end point as a string. In this case it it
'localhost:3000/cities/'.
// The second argument is a callback function with two
parameters. The request object we will send to our server, and
the response object we will receive.
```

Step 4: Creating an instance of City.

```
  const city = new City(req.body)
// Next we create a new instance of our City model by setting it
to a variable of the same name.
// We pass in the req.body which will have values for all the
properties we defined in our CitySchema.
```

Step 5: Saving our City to the database.

```
  city
    .save()
    .then((city) => {
      res.send(city)
    })
    .catch(err => {
    res.status(500).send('Server Error’)
})
```

```
// Finally we call save(). This mongoose method will return a
promise. If resolved we can send back the object we created in
our response. If rejected we will debug our code.
```

Part 2: Testing on Postman

Step 1: First make sure that our server is running correctly and no errors are in our console.

Step 2: Set the request type to POST and the end point to “localhost:5000/api/cities/”

Step 3: In the “Headers” tab we need to create a key-value pair. Set the key to “Content-Type” and its value to “application/json”

Step 4: Next head to the Body tab and select the “raw” setting.

Step 5: Now create a City object that corresponds with the City Model. Every key must match exactly our mongooseSchema.

Step 6: Find an image online for each city, and paste the link as the value for the image property of our City object. (Hint: if you search for images with the same aspect ratio it can save time later while formatting. I personally like to use panoramic images with an aspect ratio of 1200 x 400 pixels.)

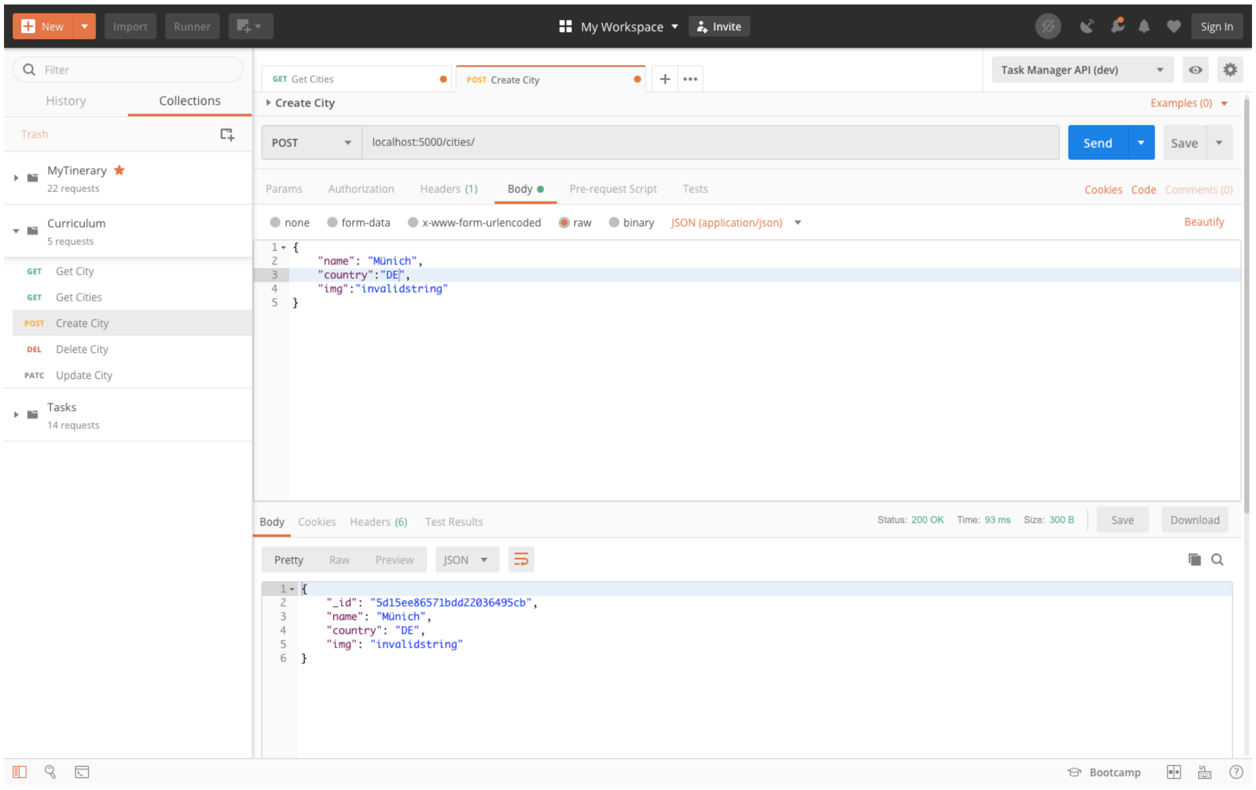
Step 7: Find an image online for each city, and paste the link as the value for the image property of our City object.

Step 8: Send the request. In a few seconds you should either receive a successful response or an error message.

Step 9: If successful a new City object should be returned to us with an _id property that was created by MongoDB. We can then double check our database to make sure the object was mapped to a document and stored in our cities collection.

- If the request fails we will need to debug either the request we sent or the route method itself.
- See Figure 1 for reference.

Figure 1:



POST localhost:5000/cities/

Body: raw, JSON (application/json)

```
{
    "name": "Münich",
    "country":"DE",
    "img":"invalidstring"
}
```

Status: 200 OK   Time: 93 ms   Size: 300 B

```
{
    "_id": "5d15ee86571bdd22036495cb",
    "name": "Münich",
    "country": "DE",
    "img": "invalidstring"
}
```