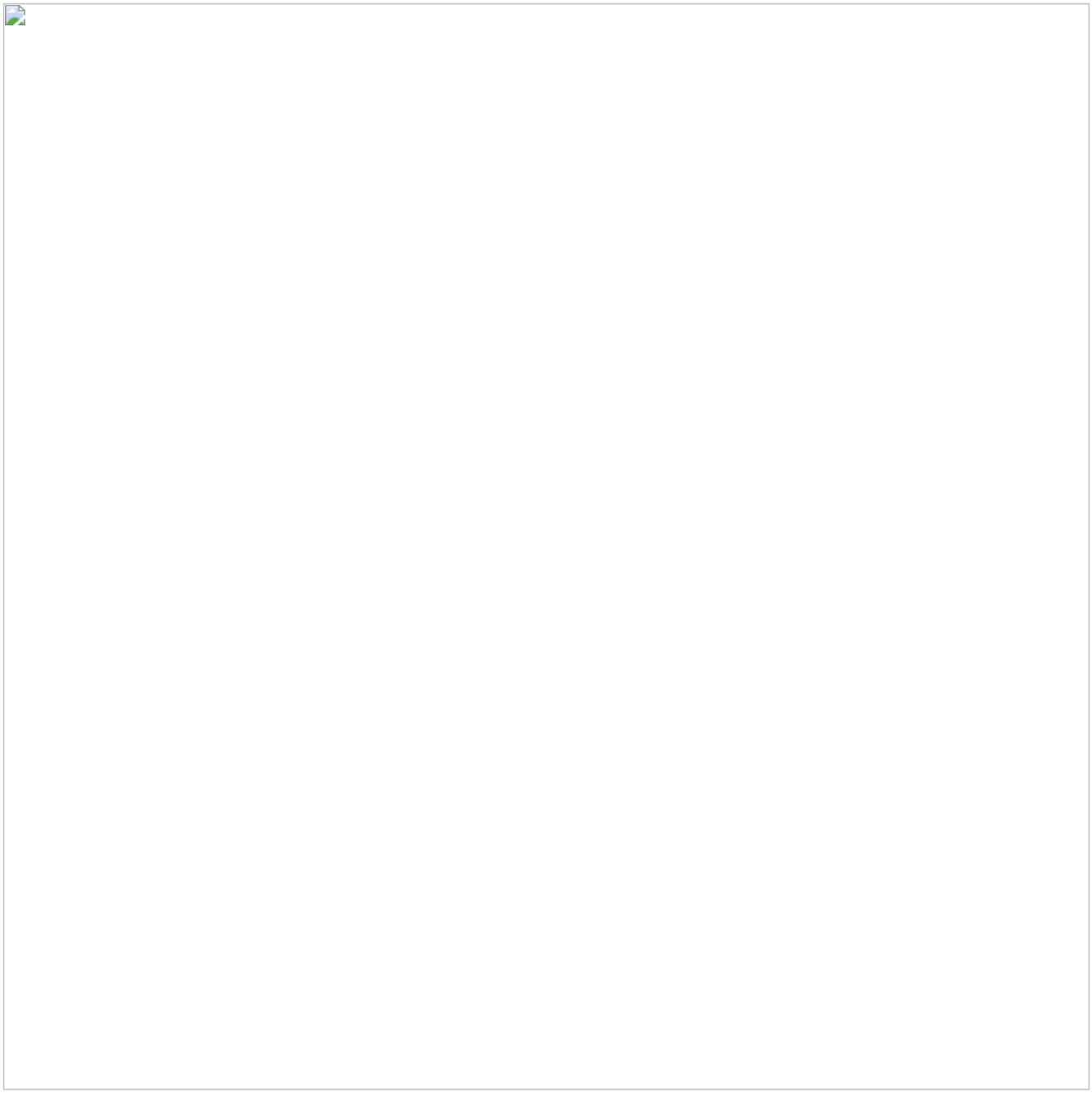
ExpressJS - RESTFul APIs

tutorialspoint.com/expressjs/expressjs_restful_apis.htm



An API is always needed to create mobile applications, single page applications, use AJAX calls and provide data to clients. An popular architectural style of how to structure and name these APIs and the endpoints is called **REST(Representational Transfer State)**. **HTTP 1.1** was designed keeping REST principles in mind. REST was introduced by **Roy Fielding** in 2000 in his Paper Fielding Dissertations.

RESTful URIs and methods provide us with almost all information we need to process a request. The table given below summarizes how the various verbs should be used and how URIs should be named. We will be creating a movies API towards the end; let us now discuss how it will be structured.

Method	URI	Details	Function
GET	/movies	Safe, cachable	Gets the list of all movies and their details

GET	/movies/1234	Safe, cachable	Gets the details of Movie id 1234
POST	/movies	N/A	Creates a new movie with the details provided. Response contains the URI for this newly created resource.
PUT	/movies/1234	Idempotent	Modifies movie id 1234(creates one if it doesn't already exist). Response contains the URI for this newly created resource.
DELETE	/movies/1234	Idempotent	Movie id 1234 should be deleted, if it exists. Response should contain the status of the request.
DELETE or PUT	/movies	Invalid	Should be invalid. **DELETE** and **PUT** should specify which resource they are working on.

Let us now create this API in Express. We will be using JSON as our transport data format as it is easy to work with in JavaScript and has other benefits. Replace your **index.js** file with the **movies.js** file as in the following program.

index.js

```
var express = require('express');
var bodyParser = require('body-parser');
var multer = require('multer');
var upload = multer();

var app = express();

app.use(cookieParser());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(upload.array());

//Require the Router we defined in movies.js
var movies = require('./movies.js');

//Use the Router on the sub route /movies
app.use('/movies', movies);

app.listen(3000);
```

Now that we have our application set up, let us concentrate on creating the API.

Start by setting up the movies.js file. We are not using a database to store the movies but are storing them in memory; so every time the server restarts, the movies added by us will vanish. This can easily be mimicked using a database or a file (using node fs module).

Once you import Express then, create a Router and export it using *module.exports* −

```
var express = require('express');
var router = express.Router();
var movies = [
   {id: 101, name: "Fight Club", year: 1999, rating: 8.1},
   {id: 102, name: "Inception", year: 2010, rating: 8.7},
   {id: 103, name: "The Dark Knight", year: 2008, rating: 9},
   {id: 104, name: "12 Angry Men", year: 1957, rating: 8.9}
];

//Routes will go here
module.exports = router;
```

GET routes

Let us define the GET route for getting all the movies −

```
router.get('/', function(req, res){
   res.json(movies);
});
```

To test out if this is working fine, run your app, then open your terminal and enter −

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
localhost:3000/movies
```

The following response will be displayed −

```
[{"id":101,"name":"Fight Club","year":1999,"rating":8.1},
{"id":102,"name":"Inception","year":2010,"rating":8.7},
{"id":103,"name":"The Dark Knight","year":2008,"rating":9},
{"id":104,"name":"12 Angry Men","year":1957,"rating":8.9}]
```

We have a route to get all the movies. Let us now create a route to get a specific movie by its id.

```
router.get('/:id([0-9]{3,})', function(req, res){
   var currMovie = movies.filter(function(movie){
      if(movie.id == req.params.id){
         return true;
      }
   });
   if(currMovie.length == 1){
      res.json(currMovie[0])
   } else {
      res.status(404);//Set status to 404 as movie was not found
      res.json({message: "Not Found"});
   }
});
```

This will get us the movies according to the id that we provided. To check the output, use the following command in your terminal −

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
localhost:3000/movies/101
```

You'll get the following response −

```
{"id":101,"name":"Fight Club","year":1999,"rating":8.1}
```

If you visit an invalid route, it will produce a **cannot GET error** while if you visit a valid route with an id that doesn't exist, it will produce a 404 error.

We are done with the GET routes, let us now move on to the **POST** route.

POST route

Use the following route to handle the **POSTed** data −

```
router.post('/', function(req, res){
   //Check if all fields are provided and are valid:
   if(!req.body.name ||
      !req.body.year.toString().match(/^[0-9]{4}$/g) ||
      !req.body.rating.toString().match(/^[0-9]\.[0-9]$/g)){

      res.status(400);
      res.json({message: "Bad Request"});
   } else {
      var newId = movies[movies.length-1].id+1;
      movies.push({
         id: newId,
         name: req.body.name,
         year: req.body.year,
         rating: req.body.rating
      });
      res.json({message: "New movie created.", location: "/movies/" + newId});
   }
});
```

This will create a new movie and store it in the movies variable. To check this route, enter the following code in your terminal −

```
curl -X POST --data "name = Toy%20story&year = 1995&rating = 8.5" http://localhost:3000/movies
```

The following response will be displayed −

```
{"message":"New movie created.","location":"/movies/105"}
```

To test if this was added to the movies object, Run the get request for **/movies/105** again. The following response will be displayed −

```
{"id":105,"name":"Toy story","year":"1995","rating":"8.5"}
```

Let us move on to create the PUT and DELETE routes.

PUT route

The PUT route is almost the same as the POST route. We will be specifying the id for the object that'll be updated/created. Create the route in the following way.

```
router.put('/:id', function(req, res){
   //Check if all fields are provided and are valid:
   if(!req.body.name ||
      !req.body.year.toString().match(/^[0-9]{4}$/g) ||
      !req.body.rating.toString().match(/^[0-9]\.[0-9]$/g) ||
      !req.params.id.toString().match(/^[0-9]{3,}$/g)){

      res.status(400);
      res.json({message: "Bad Request"});
   } else {
      //Gets us the index of movie with given id.
      var updateIndex = movies.map(function(movie){
         return movie.id;
      }).indexOf(parseInt(req.params.id));

      if(updateIndex === -1){
         //Movie not found, create new
         movies.push({
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
         });
         res.json({message: "New movie created.", location: "/movies/" + req.params.id});
      } else {
         //Update existing movie
         movies[updateIndex] = {
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
         };
         res.json({message: "Movie id " + req.params.id + " updated.",
            location: "/movies/" + req.params.id});
      }
   }
});
```

This route will perform the function specified in the above table. It will update the object with new details if it exists. If it doesn't exist, it will create a new object. To check the route, use the following curl command. This will update an existing movie. To create a new Movie, just change the id to a non-existing id.

```
curl -X PUT --data "name = Toy%20story&year = 1995&rating = 8.5"
http://localhost:3000/movies/101
```

Response

```
{"message":"Movie id 101 updated.","location":"/movies/101"}
```

DELETE route

Use the following code to create a delete route. −

```
router.delete('/:id', function(req, res){
   var removeIndex = movies.map(function(movie){
      return movie.id;
   }).indexOf(req.params.id); //Gets us the index of movie with given id.

   if(removeIndex === -1){
      res.json({message: "Not found"});
   } else {
      movies.splice(removeIndex, 1);
      res.send({message: "Movie id " + req.params.id + " removed."});
   }
});
```

Check the route in the same way as we checked the other routes. On successful deletion(for example id 105), you will get the following output −

```
{message: "Movie id 105 removed."}
```

Finally, our **movies.js** file will look like the following.

```
var express = require('express');
var router = express.Router();
var movies = [
   {id: 101, name: "Fight Club", year: 1999, rating: 8.1},
   {id: 102, name: "Inception", year: 2010, rating: 8.7},
   {id: 103, name: "The Dark Knight", year: 2008, rating: 9},
   {id: 104, name: "12 Angry Men", year: 1957, rating: 8.9}
];
router.get('/:id([0-9]{3,})', function(req, res){
   var currMovie = movies.filter(function(movie){
      if(movie.id == req.params.id){
         return true;
      }
   });

   if(currMovie.length == 1){
      res.json(currMovie[0])
   } else {
      res.status(404);  //Set status to 404 as movie was not found
      res.json({message: "Not Found"});
   }
});
router.post('/', function(req, res){
   //Check if all fields are provided and are valid:
   if(!req.body.name ||
      !req.body.year.toString().match(/^[0-9]{4}$/g) ||
      !req.body.rating.toString().match(/^[0-9]\.[0-9]$/g)){
      res.status(400);
      res.json({message: "Bad Request"});
   } else {
      var newId = movies[movies.length-1].id+1;
      movies.push({
         id: newId,
         name: req.body.name,
         year: req.body.year,
         rating: req.body.rating
      });
      res.json({message: "New movie created.", location: "/movies/" + newId});
   }
});

router.put('/:id', function(req, res) {
   //Check if all fields are provided and are valid:
   if(!req.body.name ||
      !req.body.year.toString().match(/^[0-9]{4}$/g) ||
      !req.body.rating.toString().match(/^[0-9]\.[0-9]$/g) ||
      !req.params.id.toString().match(/^[0-9]{3,}$/g)){
      res.status(400);
      res.json({message: "Bad Request"});
   } else {
      //Gets us the index of movie with given id.
      var updateIndex = movies.map(function(movie){
         return movie.id;
      }).indexOf(parseInt(req.params.id));

      if(updateIndex === -1){
         //Movie not found, create new
         movies.push({
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
         });
         res.json({
            message: "New movie created.", location: "/movies/" + req.params.id});
      } else {
         //Update existing movie
         movies[updateIndex] = {
            id: req.params.id,
            name: req.body.name,
            year: req.body.year,
            rating: req.body.rating
         };
         res.json({message: "Movie id " + req.params.id + " updated.",
            location: "/movies/" + req.params.id});
      }
```

```
   }
});

router.delete('/:id', function(req, res){
   var removeIndex = movies.map(function(movie){
      return movie.id;
   }).indexOf(req.params.id); //Gets us the index of movie with given id.

   if(removeIndex === -1){
      res.json({message: "Not found"});
   } else {
      movies.splice(removeIndex, 1);
      res.send({message: "Movie id " + req.params.id + " removed."});
   }
});
module.exports = router;
```

This completes our REST API. Now you can create much more complex applications using this simple architectural style and Express.