Express.Router

11 min

An Express router provides a subset of Express

Preview: Docs Methods are object properties that contain functions.

methods

. To create an instance of one, we invoke the .Router() method on the top-level Express import.

To use a router, we *mount* it at a certain path using app.use() and pass in the router as the second argument. This router will now be used for all paths that begin with that path segment. To create a router to handle all

Preview: Docs Loading link description

requests

beginning with /monsters, the code would look like this:

```
const express = require('express');
const app = express();

const monsters = {
 '1': {
   name: 'godzilla',
   age: 250000000
 },
 '2': {
   name: 'manticore',
   age: 21
 }
}

const monstersRouter = express.Router();

app.use('/monsters', monstersRouter);

monstersRouter.get('/:id', (req, res, next) => {
 const monster = monsters[req.params.id];
 if (monster) {
   res.send(monster);
 } else {
   res.status(404).send();
 }
```

});

Inside the monstersRouter, all matching routes are assumed to have /monsters prepended, as it is mounted at that path. monstersRouter.get('/:id') matches the full path /monsters/:id.

When a GET /monsters/1 request arrives, Express matches /monsters in app.use() because the beginning of the path ('/monsters') matches. Express' route-matching algorithm enters the monstersRouter's routes to search for full path matches. Since monstersRouter.get('/:id) is mounted at /monsters, the two paths together match the entire request path (/monsters/1), so the route matches and the callback is invoked. The 'godzilla' monster is fetched from the monsters object and sent back.

Instructions

1. Checkpoint 1 Passed

1.

Create an expressionsRouter instance of Express.Router. Mount it at /expressions at your base app level with app.use.

After doing so, create a route for your expressionsRouter that will send all expressions for a GET request.

Hint

Remember, if the expressionsRouter is mounted properly at /expressions, the GET request for all expressions will use the '/' path inside the expressionsRouter.

app.js

```
const express = require('express');
const app = express();
const expressionsRouter = express.Router();

const { getElementById, getIndexById, updateElement,
  seedElements, createElement } = require('./utils');

const PORT = process.env.PORT || 4001;
// Use static server to serve the Express Yourself Website
app.use(express.static('public'));
app.use('/expressions', expressionsRouter)
```

```
let expressions = [];
seedElements(expressions, 'expressions');
let animals = [];
seedElements(animals, 'animals');

// Get all expressions
expressionsRouter.get('/', (req, res, next) => {
  res.send(expressions);
});

// Get a single expression
app.get('/expressions/:id', (req, res, next) => {
  const foundExpression = getElementById(req.params.id, expressions);
  if (foundExpression) {
    res.send(foundExpression);
  } else {
    res.status(404).send();
  }
});

// Update an expression
app.put('/expressions/:id', (req, res, next) => {
  const expressionIndex = getIndexById(req.params.id, expressions);
  if (expressionIndex !== -1) {
    updateElement(req.params.id, req.query, expressions);
    res.send(expressions[expressionIndex]);
  } else {
    res.status(404).send();
  }
```

```
});

// Create an expression
app.post('/expressions', (req, res, next) => {
  const receivedExpression = createElement('expressions', req.query);
  if (receivedExpression) {
    expressions.push(receivedExpression);
    res.status(201).send(receivedExpression);
  } else {
    res.status(400).send();
  }
});

// Delete an expression
app.delete('/expressions/:id', (req, res, next) => {
  const expressionIndex = getIndexById(req.params.id, expressions);
  if (expressionIndex !== -1) {
    expressions.splice(expressionIndex, 1);
    res.status(204).send();
  } else {
    res.status(404).send();
  }
});

// Get all animals
app.get('/animals', (req, res, next) => {
  res.send(animals);
});

// Get a single animal
```

```
app.get('/animals/:id', (req, res, next) => {
  const animal = getElementById(req.params.id, animals);
  if (animal) {
    res.send(animal);
  } else {
    res.status(404).send();
  }
});

// Create an animal
app.post('/animals', (req, res, next) => {
  const receivedAnimal = createElement('animals', req.query);
  if (receivedAnimal) {
    animals.push(receivedAnimal);
    res.send(receivedAnimal);
  } else {
    res.status(400).send();
  }
});

// Update an animal
app.put('/animals/:id', (req, res, next) => {
  const animalIndex = getIndexById(req.params.id, animals);
  if (animalIndex !== -1) {
    updateElement(req.params.id, req.query, animals);
    res.send(animals[animalIndex]);
  } else {
    res.status(404).send();
  }
});
```

```
// Delete a single animal
app.delete('/animals/:id', (req, res, next) => {
  const animalIndex = getIndexById(req.params.id, animals);
  if (animalIndex !== -1) {
    animals.splice(animalIndex, 1);
    res.status(204).send();
  } else {
    res.status(404).send();
  }
});

app.listen(PORT, () => {
  console.log(`Server is listening on ${PORT}`);
});
```