Wrap Up

<1 min

In this exercise, you were able to create a full server allowing users to implement all CRUD operations for two kinds of resources: Expressions and Animals! With these skills and knowledge of the HTTP request-response cycle, you could implement an API for any project needing CRUD functionality. You could build a trip planner, an address book, a grocery list, an image-sharing application, an anonymous message board, the sky's the limit!

Continue on to the next lesson to learn more about how to keep your code clean and modular with Express Routers!

Instructions

Great work! Move on to the next lesson when you're ready.

app.js

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

const app = express();


// Use static server to serve the Express Yourself Website

app.use(express.static('public'));


const { getElementById, getIndexById, updateElement,

  seedElements, createElement } = require('./utils');


const PORT = process.env.PORT || 4001;


const expressions = [];

seedElements(expressions, 'expressions');

const animals = [];

seedElements(animals, 'animals');


// Get all expressions

app.get('/expressions', (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.status(201).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}`);
});
```