**Node using Express.js**

**Express:**

* Sinatra inspired web development framework for Node.js.
* Sinatra- web development framework for ruby.

**Usage:**

* Easy route urls to callbacks
* Middleware(from connect)
* Environment based configuration
* Redirection helpers
* File uploads

npm install --save express

**simple program**

var express = require('express');

    //install the modules and add it to our dependency files (ie. Package.json).

var app=express();

    //create instance of express

app.get('/', function(request, response){

    response.sendFile(\_\_dirname+ '/index.html');

});

    //define endpoints at the root route

    //something that is going to call this get request is going to call the callback that we are sending in

    //sendFile will read in the file from current directory and will send it back with response

app.listen(8080);

* end point is server.
* \_\_dirname gives the current directory.
* sendFile is used to send the parameter as response.
* Listen is used to mention the port where you would like to listen the app.

**program to get first 10 tweets from an account:**

app.js

var express = require('express');

var app = express();

var request = require('request');

var url = require('url');

app.get('/tweets/:username',function(request,response){

    var username =  request.param.username;

    options = {

        protocol:'http:',

        host:'api.twitter.com',

        pathname:'/1/statuses/user\_timeline.json',

        query: {screen\_name:username, count :10}

    }

    //get the 10 tweets for screen\_name

    var twitterUrl = url.format(options);

    request(twitterUrl).pipe(response);

    //pipe the request to response

    //pipe fn is used to read data from readable stream and writes it to the destination writable stream

});

**Commands:**

* node app.js
* curl –s <http://localhost:8080/tweets/ragavi>

* npm install prettyjson –g
* curl –s <http://localhost:8080/tweets/ragavi> | prettyjson

//piping the output to prettyjson to make the json prettier

Putting the json into web browser using ejs:

Ejs: embedded java script

npm install –save ejs

//by default it looks for the template under the views directory (default directory : views)

Instead of pipe if we use a callback function then we could access error , response and body.

**REST:**

Representational State Transfer

Series of rules that are in place for the server so that everyone that uses the service understands what it is and how it works.

Constraint between two systems that talk to each other:

1)**The client Server constraint:**

request

server

client

response

2)**Stateless server constraint:**

server

request

client

server

response

server

As the load from the client increases on server , then we start to add more servers. There can be a situation server may contain information about the client that doesn’t transfer from one server to the other .

Instead the client sends the request to the server and it doesn’t matter what server it goes to. So everything that the server is going to need to process the request should be included in that request itself. Based on that request and based on all the information that’s in the request, it’ll send you an response.

3)**caching constraint:**

As the server sends more information back to the client , actually sometimes the data that is sent doesn’t change very often. In case of book authors , author is not going to change . so the next time the client pulls the data , we want to know whether or not we should pull it. It lets the client know how long is the data good for so that the client doesn’t have to come back to the server for that data over and over again,

Only if needed

server

request

client

server

response

server

**HATEOAS:** Hypermedia As The Engine of Application State

**Revealing module pattern:**

Controller follows revealing module pattern wherein the get and post functions are private and exposed outside with the help of return statement.

**Test driven development:**

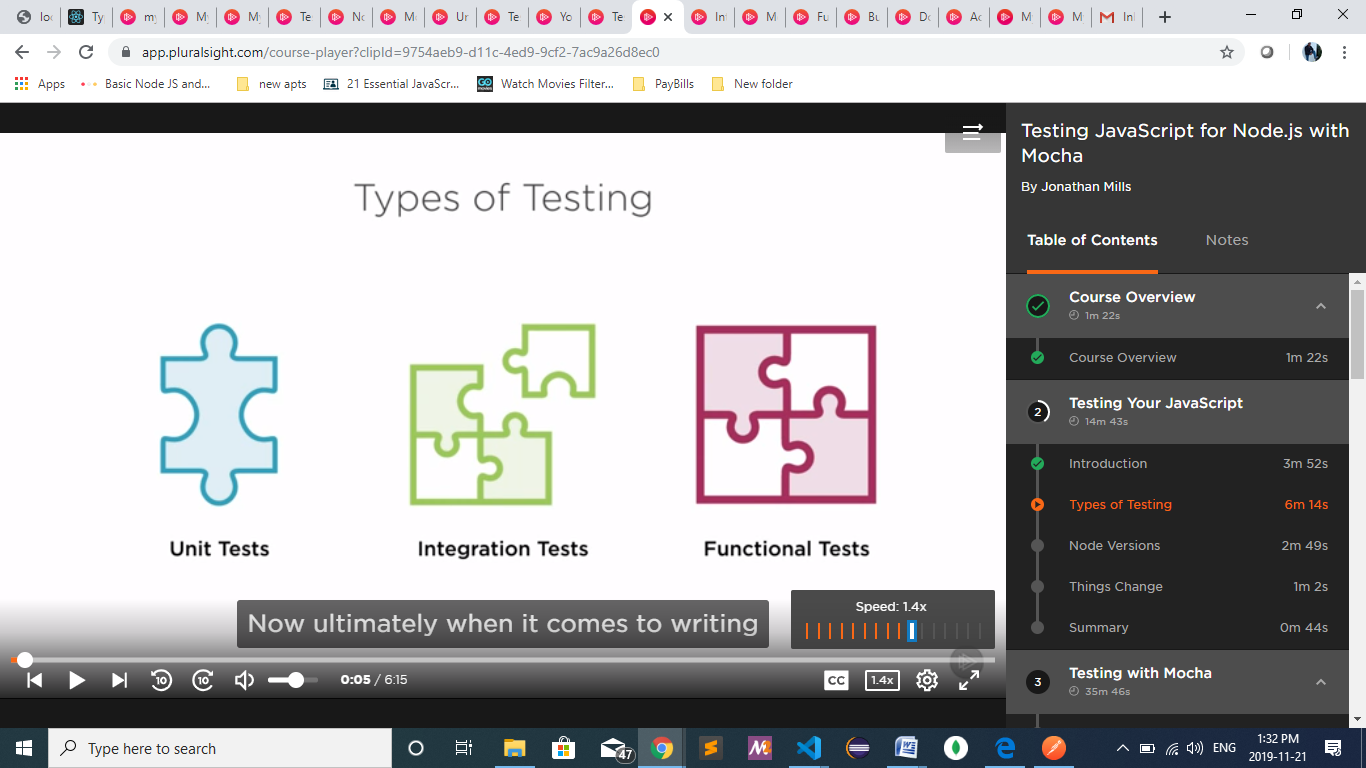
Create a test that replicates the issue so that it will break and code will be written to solve the issue

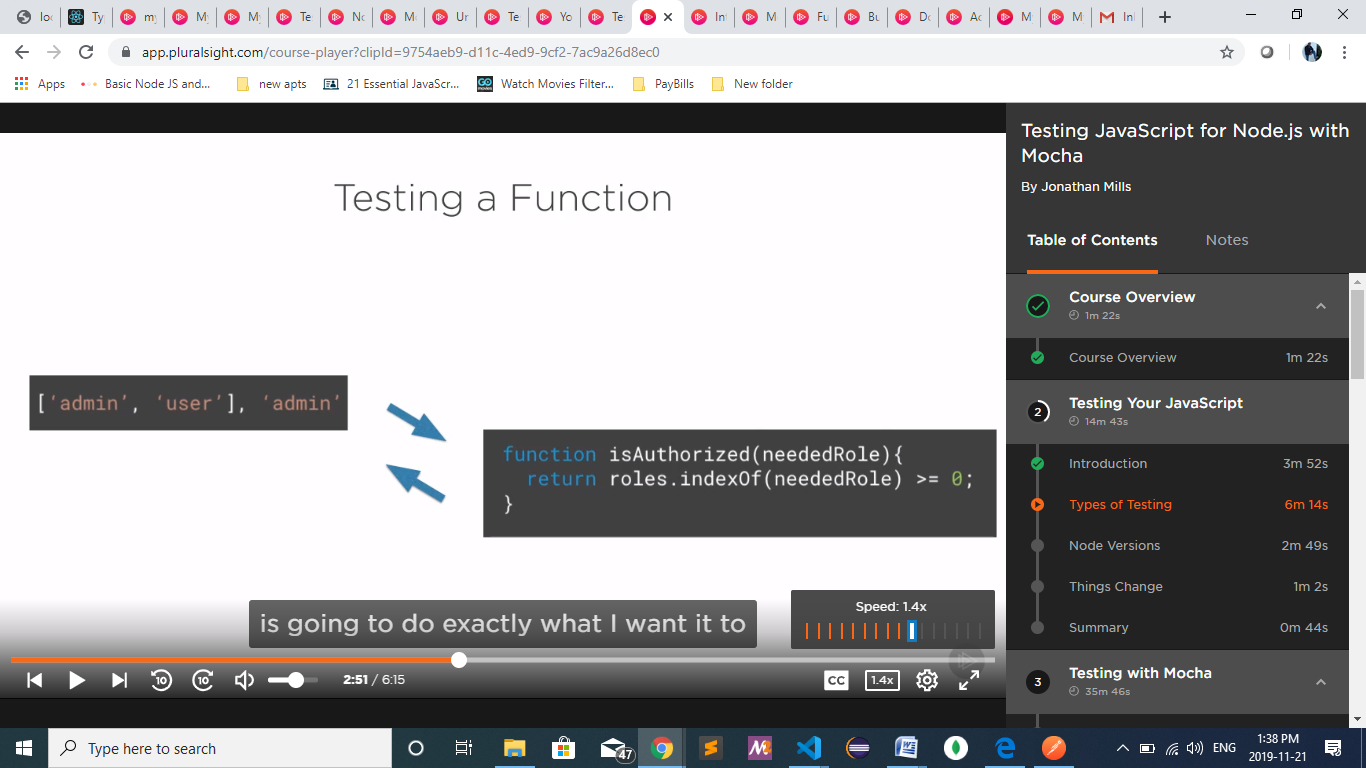
Collection of unit testing: integration testing

Collection of integration testing : functional testing(end to end testing)

**Unit testing :**

* Test the single piece like single function or few lines of code or one small unit. Ex:mocha
* It testes just that piece of code and not consider the surrounding things .





**Integration testing:**

We are integrating our code with other pieces of code .

Ex:

The function has to work in the larger scope and not by itself (tying things together) and has to interact with other things and test their interactions.

**Functional testing**:

We test the entire application end to end.ex: black-box testing

**Black box testing:**

We will not know anything about what’s going on inside of the application or how things work inside , but i handed this, make an http call and it returned the response.

**Assertion framework:**

Ex:should, chai

**Type of architecture:**

* Behaviour driven development: it is a type of test driven framework wherein we test the behaviour of the component or method or element.

Ex: open popup are the behaviour of popups, post call behaviours like validation.

* Test driven framework:

**Sinon.js**

It is used to create a mockup for the request and response

**Get Call:**

**Commands:**

node –versio

npm init

npm install express

npm install [express@4.16.4](mailto:express@4.16.4) (for specific version download)

app.js

    const express = require('express');

    const app = express();

    const port = process.env.PORT || 3000;

    app.get('/', (req, res) => {

        res.send('welcome to my API');

    });

    app.listen(port, () => {

        console.log(`Running on port ${port}` );

    });

To fire this up (command):

node app.js

**Setting up tools:**

**Eslint :**

npm i eslint –D

npm run lint -- --init (everything after lint is passed on to init) 🡪 use a popular style guide 🡪 airbnb

do you use react? No 🡪 config file :javascript 🡪would you like to install them now with npm? Yes

under scripts tag in package.json:

 "scripts": {

    "lint": "eslint .",

    "test": "echo \"Error: no test specified\" && exit 1"

  }

**Nodemon:**

Handle the environment stuff, going to watch our files and restart automatically if something changes

npm install nodemon

package.json

 "scripts": {

    "lint": "eslint .",

    "start": "nodemon app.js",

    "test": "echo \"Error: no test specified\" && exit 1"

  }

 "nodemonConfig": {

    "restartable": "rs",

    "ignore": [

      "node\_modules/\*\*/node\_modules"

    ],

    "delay": "2500",

    "env": {

      "NODE\_ENV": "development",

      "PORT": 4000

    }

  }

npm start

**implementing http get ( without mongodb):**

const express = require('express');

const app = express();

const port = process.env.PORT || 3000;

const bookRouter = express.Router();

bookRouter.route('/books')

  .get((req, res) => {

    const response = ({ hello: 'This is my API!' });

    res.json(response);

  });

app.use('/api', bookRouter);

app.get('/', (req, res) => {

  res.send('welcome to my API');

});

app.listen(port, () => {

  console.log(`Running on port${port}`);

});

**Wiring up with my mongodb:**

* New terminal:

mongod

* to import put the json into the mongodb

run ::cmd /c 'mongo bookAPI < booksJson.js' ::

(also make sure mongodb is running then)

* to pull data from mongodb , we need mongoose (ie setting up a connection like jdbc)

app.js:

npm install mongoose;

const mongoose = require('mongoose');

const db = mongoose.connect('mongodb://localhost/bookAPI');

// the bookAPI in the url comes from the import command that we write run ::cmd /c 'mongo bookAPI < booksJson.js' ::

const Book = require('/models/bookModel');

    bookRouter.route('/books')

    .get((req, res) => {

      Book.find((err, books) => {

        if (err) {

          return res.send(err);

        }

        return res.json(books);

      });

    });

npm start

models/bookModel.js

const mongoose = require('mongoose');

const { Schema } = mongoose;

const bookModel = new Schema(

  {

    title: { type: String },

    author: { type: String },

    genre: { type: String },

    read: { type: Boolean, default: false },

  },

);

module.exports = mongoose.model('Book', bookModel);

**Get Call with filter or query string:**

app.js

bookRouter.route('/books')

  .get((req, res) => {

    const query = {};

    if (req.query.genre) {

      query.genre = req.query.genre;

    }

    Book.find(query, (err, books) => {

      if (err) {

        return res.send(err);

      }

      return res.json(books);

    });

  });

**Getting a single item:**

**app.js**

bookRouter.route('/books/:bookId')

  .get((req, res) => {

    Book.findById(req.params.bookId, (err, book) => {

      if (err) {

        return res.send(err);

      }

      return res.json(book);

    });

  });

**Post call**:

npm install body-parser

const bodyParser = require('body-parser');

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json);

bookRouter.route('/books')

 .post((req, res) => {

    const book = new Book(req.body);

    book.save();

    return res.status(201).json(book);

  });

**Code cleanup:**

routes/bookRouter.js

const express = require('express');

function routes(Book) {

  const bookRouter = express.Router();

  bookRouter.route('/books')

    .get((req, res) => {

      const { query } = req;

      Book.find(query, (err, books) => {

        if (err) {

          return res.send(err);

        }

        return res.json(books);

      });

    })

    .post((req, res) => {

      const book = new Book(req.body);

      book.save();

      return res.status(201).json(book);

    });

  bookRouter.route('/books/:bookId')

    .get((req, res) => {

      Book.findById(req.params.bookId, (err, books) => {

        if (err) {

          return res.send(err);

        }

        return res.json(books);

      });

    });

  return bookRouter;

}

module.exports = routes;

app.js

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const app = express();

const port = process.env.PORT || 3000;

// model

const Book = require('./models/bookModel');

const bookRouter = require('./routes/bookRouter')(Book);

mongoose.connect('mongodb://localhost/bookAPI');

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.use('/api', bookRouter);

app.get('/', (req, res) => {

  res.send('Welcome to my Nodemon API!');

});

app.listen(port, () => {

  console.log(`Running on port ${port}`);

});

**Put call:**

routes/bookRouter.js

bookRouter.route('/books/:bookId')

 .put((req, res) => {

      Book.findById(req.params.bookId, (err, book) => {

        if (err) {

          return res.send(err);

        }

        book.title = req.body.title;

        book.author = req.body.author;

        book.read = req.body.read;

        book.genre = req.body.genre;

        book.save();

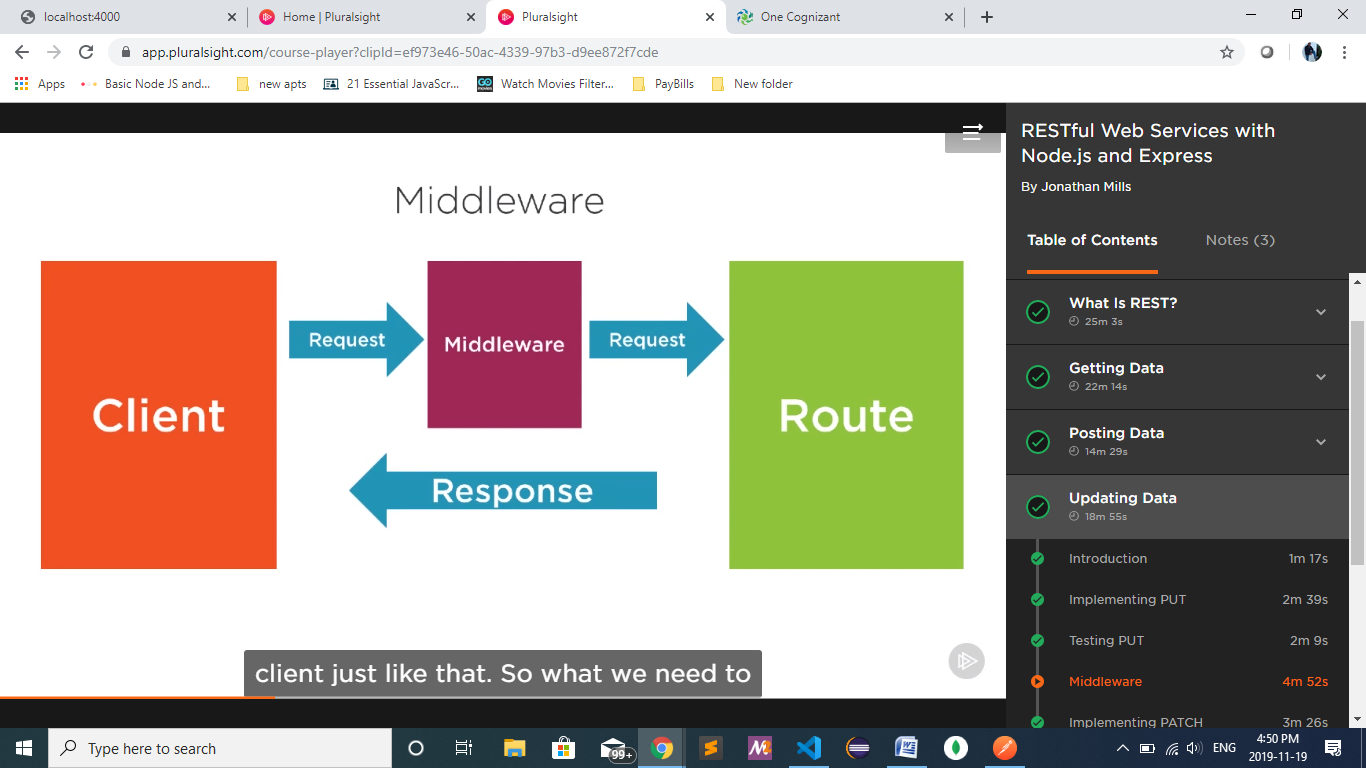
        return res.json(book);

      });

    });

**Middleware**:

The middleware function is going to interrupt the request



routes/bookRouter.js

 bookRouter.use('/books/:bookId', (req, res, next) => {

    Book.findById(req.params.bookId, (err, book) => {

      if (err) {

        return res.send(err);

      }

      if (book) {

        req.book = book;

        return next();

      }

      return res.sendStatus(404);

    });

  });

 bookRouter.route('/books/:bookId')

    .get((req, res) => res.send(req.book))

    .put((req, res) => {

      const { book } = req;

      book.title = req.body.title;

      book.author = req.body.author;

      book.read = req.body.read;

      book.genre = req.body.genre;

      req.book.save((err) => {

        if (err) {

          return res.send(err);

        }

        return res.json(book);

      });

    })

**Patch call** for updating only one detail of an object:

routes/bookRouter.js

bookRouter.route('/books/:bookId')

.patch((req, res) => {

      const { book } = req;

      // eslint-disable-next-line no-underscore-dangle

      if (req.body.\_id) {

        // eslint-disable-next-line no-underscore-dangle

        delete req.body.\_id;

      }

      Object.entries(req.body).forEach((item) => {

        const key = item[0];

        const value = item[1];

        book[key] = value;

      });

      req.book.save((err) => {

        if (err) {

          return res.send(err);

        }

        return res.json(book);

      });

    })

    .delete((req, res) => {

      req.book.remove((err) => {

        if (err) {

          return res.send(err);

        }

        return res.sendStatus(204);

      });

    });

**Delete call:**

routes/bookRouter.js

bookRouter.route('/books/:bookId')

.delete((req, res) => {

      req.book.remove((err) => {

        if (err) {

          return res.send(err);

        }

        return res.sendStatus(204);

      });

    });

**Entire App:**

routes/bookRouter.js

/\* eslint-disable no-param-reassign \*/

const express = require('express');

function routes(Book) {

  const bookRouter = express.Router();

  bookRouter.route('/books')

    .get((req, res) => {

      const { query } = req;

      Book.find(query, (err, books) => {

        if (err) {

          return res.send(err);

        }

        return res.json(books);

      });

    })

    .post((req, res) => {

      const book = new Book(req.body);

      book.save();

      return res.status(201).json(book);

    });

  bookRouter.use('/books/:bookId', (req, res, next) => {

    Book.findById(req.params.bookId, (err, book) => {

      if (err) {

        return res.send(err);

      }

      if (book) {

        req.book = book;

        return next();

      }

      return res.sendStatus(404);

    });

  });

  bookRouter.route('/books/:bookId')

    .get((req, res) => res.send(req.book))

    .put((req, res) => {

      const { book } = req;

      book.title = req.body.title;

      book.author = req.body.author;

      book.read = req.body.read;

      book.genre = req.body.genre;

      req.book.save((err) => {

        if (err) {

          return res.send(err);

        }

        return res.json(book);

      });

    })

    .patch((req, res) => {

      const { book } = req;

      // eslint-disable-next-line no-underscore-dangle

      if (req.body.\_id) {

        // eslint-disable-next-line no-underscore-dangle

        delete req.body.\_id;

      }

      Object.entries(req.body).forEach((item) => {

        const key = item[0];

        const value = item[1];

        book[key] = value;

      });

      req.book.save((err) => {

        if (err) {

          return res.send(err);

        }

        return res.json(book);

      });

    })

    .delete((req, res) => {

      req.book.remove((err) => {

        if (err) {

          return res.send(err);

        }

        return res.sendStatus(204);

      });

    });

  return bookRouter;

}

module.exports = routes;

app.js

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const app = express();

const port = process.env.PORT || 3000;

// model

const Book = require('./models/bookModel');

const bookRouter = require('./routes/bookRouter')(Book);

mongoose.connect('mongodb://localhost/bookAPI');

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.use('/api', bookRouter);

app.get('/', (req, res) => {

  res.send('Welcome to my Nodemon API!');

});

app.listen(port, () => {

  console.log(`Running on port ${port}`);

});

DataimportInstruction:

To import Book data into your mongoDB database.

Make sure MongoDB is running then run ::cmd /c 'mongo bookAPI < booksJson.js' ::

from the command line.

.eslintrc.js

module.exports = {

    "extends": "airbnb-base",

    "rules": {

        "comma-dangle": 0

    }

};

Models/bookModel.js

const mongoose = require('mongoose');

const { Schema } = mongoose;

const bookModel = new Schema(

  {

    title: { type: String },

    author: { type: String },

    genre: { type: String },

    read: { type: Boolean, default: false },

  },

);

module.exports = mongoose.model('Book', bookModel);

**Testing:**

controllers/booksController.js

function booksController(Book) {

  function post(req, res) {

    const book = new Book(req.body);

    book.save();

    return res.status(201).json(book);

  }

  function get(req, res) {

    const { query } = req;

    Book.find(query, (err, books) => {

      if (err) {

        return res.send(err);

      }

      return res.json(books);

    });

  }

  return { post, get };

}

module.exports = booksController;

routes/bookRouter.js

/\* eslint-disable no-param-reassign \*/

const express = require('express');

const booksController = require('../controllers/booksController');

function routes(Book) {

  const bookRouter = express.Router();

  const controller = booksController(Book);

  bookRouter.route('/books')

    .get(controller.get)

    .post(controller.post);

  bookRouter.use('/books/:bookId', (req, res, next) => {

    Book.findById(req.params.bookId, (err, book) => {

      if (err) {

        return res.send(err);

      }

      if (book) {

        req.book = book;

        return next();

      }

      return res.sendStatus(404);

    });

  });

  bookRouter.route('/books/:bookId')

    .get((req, res) => res.send(req.book))

    .put((req, res) => {

      const { book } = req;

      book.title = req.body.title;

      book.author = req.body.author;

      book.read = req.body.read;

      book.genre = req.body.genre;

      req.book.save((err) => {

        if (err) {

          return res.send(err);

        }

        return res.json(book);

      });

    })

    .patch((req, res) => {

      const { book } = req;

      // eslint-disable-next-line no-underscore-dangle

      if (req.body.\_id) {

        // eslint-disable-next-line no-underscore-dangle

        delete req.body.\_id;

      }

      Object.entries(req.body).forEach((item) => {

        const key = item[0];

        const value = item[1];

        book[key] = value;

      });

      req.book.save((err) => {

        if (err) {

          return res.send(err);

        }

        return res.json(book);

      });

    })

    .delete((req, res) => {

      req.book.remove((err) => {

        if (err) {

          return res.send(err);

        }

        return res.sendStatus(204);

      });

    });

  return bookRouter;

}

module.exports = routes;

npm install –D mocha should sinon

Mocha :testing framework

Should:assertion framework

Sinon:mock