

[◀ Back to Blog](#)

May 19, 2016

Testing APIs with Mocha

by Caio Ribeiro PereiraTags: *JavaScript, Tutorials, Webdev*

Creating automated tests is something highly recommended. There are several types of tests: **unitary**, **functional**, **integration** and others. In this chapter, we will focus only on the **integration tests**. In our case we aim to test the outputs and behaviors of the API routes.

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)

First of all, we need to build a small and simple API which will be called `task-api` and we will add some routes that will be enough to manage a task list. After that we will create some tests. For this purpose, let's create a Node.js project by running these commands:

```
mkdir task - api
cd task - api
npm init
```


Let's install some useful modules to build our API. We'll use: `express` as web framework, `body-parser` as json serializer, `lowdb` as json database and `uuid` as unique id generator. To install them, just run this command:

```
npm install express body - parser lowdb uuid--save
```

And now with all dependencies installed, let's build our API by creating the `index.js` file in the project's root:

```
// Loading modules
var express = require('express');
var lowdb = require('lowdb');
var storage = require('lowdb/file-sync');
var uuid = require('uuid');
var bodyParser = require('body-parser');
// Instantiating express module
var app = express();
// Instantiating database module
// This will create db.json storage in the root folder
app.db = lowdb('db.json', {
  storage: storage
});
// Adding body-parser middleware to parser JSON data
app.use(bodyParser.json());
```

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)

 [Code Integrity](#) [Enterprise Webpage](#) [Pricing & Plans](#) [Docs](#) [Blog](#)

```
app.get('/tasks', function(req, res) {
  return res.json(app.db('tasks').find({
    id: req.params.id
  }));
});

// Finding a task
app.get('/tasks/:id', function(req, res) {
  var id = req.params.id;
  var task = app.db('tasks').find({
    id: id
  });
  if (task) {
    return res.json(task);
  }
  return res.status(404).end();
});

// Adding new task
app.post('/tasks', function(req, res) {
  var task = req.body;
  task.id = uuid();
  app.db('tasks').push(task);
  return res.status(201).end();
});

// Updating a task
app.put('/tasks/:id', function(req, res) {
  var id = req.params.id;
  var task = req.body;
  app.db('tasks')
    .chain()
    .find({
      id: id
    })
    .assign(task)
    .value()
  return res.status(201).end();
});

// Delete a task
app.delete('/tasks/:id', function(req, res) {
  var id = req.params.id;
```

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)



```
    });  
    return res.integrity(201).end();  
  });  
  // API server listing port 3000  
  app.listen(3000, function() {  
    console.log('API up and running');  
  });  
  // Exporting the app module  
  module.exports = app;
```

Code Integrity Enterprise Webpage Integrity Pricing & Plans Docs Blog

Introduction to Mocha

To create and execute these tests, we will have to use a test runner. We'll use Mocha that is very popular in Node.js community.

Mocha has the following features:

- TDD style;
- BDD style;
- Code coverage HTML report;
- Customized test reports;
- Asynchronous test support;
- Easily integrated with the modules: should, assert and chai.

And it is a complete environment to create tests. You can learn more about it by accessing: mochajs.org.

Setting up the test environment

In our project, we are going to explore **integration tests** to create some tests for our API routes. To create them, we are going to use these modules:

- **mocha**: the main test runner;
- **chai** to write BDD tests via expect function;

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)

So let's install them:

jscrambler

Code
Integrity

Enterprise

Webpage
Integrity

Pricing
&
Plans

Docs

Blog

```
npm install mocha chai supertest--save - dev
```

Now, let's encapsulate the `mocha` test runner into the `npm test` alias command to internally run the command: `NODE_ENV=test mocha test/**/*.js` which is responsible for running all tests. And to start the API server, we are going to use the `npm start` alias command to run the `node index.js` command. To implement these new commands, edit the `package.json` and include the `scripts.start` and `scripts.test` attributes:

```
{
  "name": "task-api",
  "version": "1.0.0",
  "description": "Task list API",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "NODE_ENV=test mocha test/**/*.js"
  },
  "dependencies": {
    "body-parser": "^1.15.0",
    "express": "^4.13.4",
    "lowdb": "^0.12.5",
    "uuid": "^2.0.2"
  },
  "devDependencies": {
    "chai": "^3.5.0",
    "mocha": "^2.4.5",
    "supertest": "^1.2.0"
  }
}
```

To finish our test's environment setup, let's prepare some Mocha settings. This helper

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)

variables. To implement this helper, create the file `test/helpers.js` file:

Jscrambler

Code
Integrity

Enterprise

Webpage
Integrity

Plans

Helping
&

Docs

Blog

```
var supertest = require('supertest');
var chai = require('chai');
var uuid = require('uuid');
var app = require('../index.js');

global.app = app;
global.uuid = uuid;
global.expect = chai.expect;
global.request = supertest(app);
```

Then, let's create a simple file which allows to include some settings as parameters to the `mocha` module. This will be responsible to load first the `test/helpers.js` and also use the `--reporter spec` flag to customize the test's report. So, create the `test/mocha.opts` file using the following parameters:

```
--require test / helpers
--reporter spec
```

Writing tests!!!

We finished the setup related to the test environment. What about actually testing something? What about writing some tests for the API?

To create tests, first you need to use the `describe` function to describe what this test is for, and inside it you create tests by using the `it` function. You can also use nested `describe`'s. In all of the tests we are going to use the `request` module to do some requests in the routes of our API. To validate the tests' result we use the `expect` module, which has a lot of useful functions to check if some variable is responding according to an expected behavior. Finally, to finish a test you must run in the the `done` function. To learn all assertion's functions from `expect` module, you can take a look at [chai BDD style documentation](#) here and you can learn about everything from `supertest`

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)

module too. To see in practice how to write tests, let's create the main [Jscrambler](#) [Code](#) [Enterprise](#) [Webpage](#) [Privacy](#) [&](#) [Docs](#) [Blog](#) [Integrity](#) [Plans](#) file and write these tests below:

```
describe('Task API Routes', function() {
  // This function will run before every test to clear database
  beforeEach(function(done) {
    app.db.object = {};
    app.db.object.tasks = [{
      id: uuid(),
      title: 'study',
      done: false
    }, {
      id: uuid(),
      title: 'work',
      done: true
    }];
    app.db.write();
    done();
  });

  // In this test it's expected a task list of two tasks
  describe('GET /tasks', function() {
    it('returns a list of tasks', function(done) {
      request.get('/tasks')
        .expect(200)
        .end(function(err, res) {
          expect(res.body).toHaveLength(2);
          done(err);
        });
    });
  });

  // Testing the save task expecting status 201 of success
  describe('POST /tasks', function() {
    it('saves a new task', function(done) {
      request.post('/tasks')
        .expect(201)
        .end(function(err, res) {
          expect(res.status).toBe(201);
          done(err);
        });
    });
  });
});
```

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)

[Code
Integrity](#)[done: false
Enterprise](#)[Webpage
Integrity](#)[Pricing
&
Plans](#)[Docs](#)[Blog](#)

```
        .expect(201)
        .end(function(err, res) {
            done(err);
        });
    });
});
```

// Here it'll be tested two behaviors when try to find a task by id

```
describe('GET /tasks/:id', function() {
    // Testing how to find a task by id
    it('returns a task by id', function(done) {
        var task = app.db('tasks').first();
        request.get('/tasks/' + task.id)
            .expect(200)
            .end(function(err, res) {
                expect(res.body).toEqual(task);
                done(err);
            });
    });
});
```

// Testing the status 404 for task not found

```
it('returns status 404 when id is not found', function(done) {
    var task = {
        id: 'fakeId'
    }
    request.get('/tasks/' + task.id)
        .expect(404)
        .end(function(err, res) {
            done(err);
        });
    });
});
```

// Testing how to update a task expecting status 201 of success

```
describe('PUT /tasks/:id', function() {
```



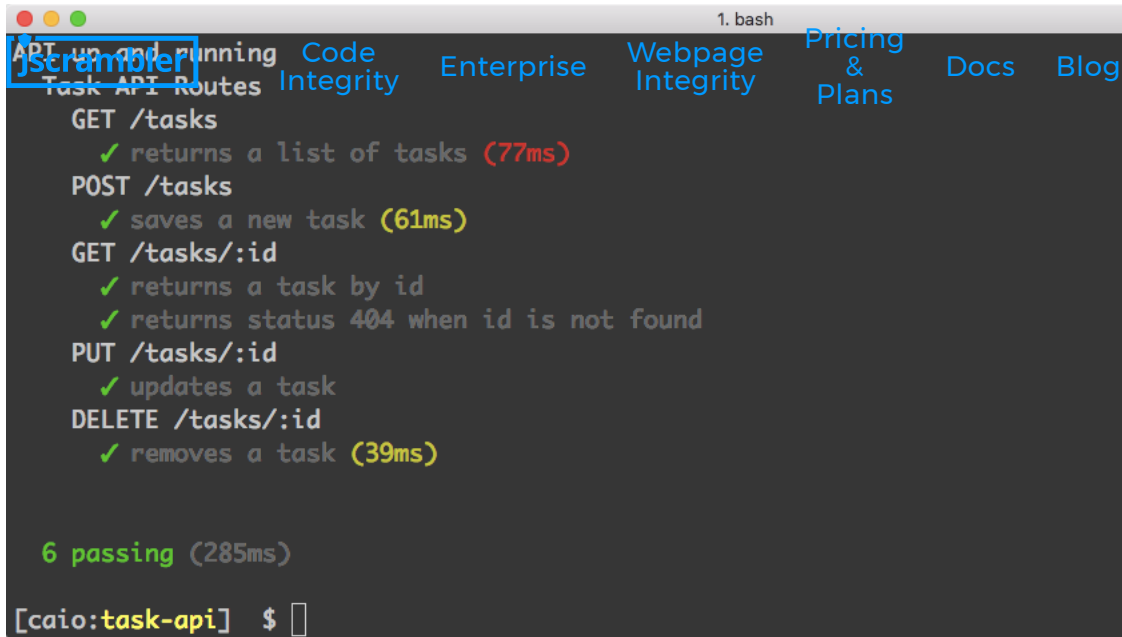
```
request.put('/tasks/' + task.id)
    .end({
      title: 'travel',
      done: false
    })
    .expect(201)
    .end(function(err, res) {
      done(err);
    });
});

// Testing how to delete a task expecting status 201 of success
describe('DELETE /tasks/:id', function() {
  it('removes a task', function(done) {
    var task = app.db('tasks').first();
    request.put('/tasks/' + task.id)
      .expect(201)
      .end(function(err, res) {
        done(err);
      });
  });
});
});
```

Now you just need to run all the tests. To do so, just run the command:

```
npm test
```

And see with your own eyes the successful result of these tests:



```
1. bash
Jscrambler
Task API Routes
GET /tasks
  ✓ returns a list of tasks (77ms)
POST /tasks
  ✓ saves a new task (61ms)
GET /tasks/:id
  ✓ returns a task by id
  ✓ returns status 404 when id is not found
PUT /tasks/:id
  ✓ updates a task
DELETE /tasks/:id
  ✓ removes a task (39ms)

6 passing (285ms)

[caio:task-api] $
```

Conclusion

If you reached this point, then you have created a small API using Node.js with some integration tests to ensure the code quality of your project. This was all attained using some popular frameworks like: `mocha`, `chai` and `supertest`.

Share this Blog Post



Recommended Posts



We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)



[Code Integrity](#) [Enterprise](#) [Webpage Integrity](#) [May 05, 2016](#) [Pricing & Plans](#) [Docs](#) [Blog](#)

Girls Develop and They're Damn Good at It!

We would love to share with you our sponsorship experience with Girl Develop It - non-profit organization supporting women inclusion in the world of IT.

Getting started with



GraphQL

May 11, 2016

Getting Started with GraphQL

GraphQL is a declarative, compositional and strong-typed query language. Useful for querying dynamic data with a strong-typed schema.

[Enterprise](#)

[About Us](#)

[Contact Us](#)

[Help Center](#)

[Privacy & Security](#)

[Careers](#)

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. [I understand](#)



[Code
Integrity](#)

[Enterprise](#)

[Webpage
Integrity](#)

[Pricing
&
Plans](#)

[Docs](#)

[Blog](#)

We use cookies to give you the best and most relevant experience. By using Jscrambler you are complying with this. **I understand**