Testing Endpoints

Automated Testing

- Using Postman is a useful for exploring APIs
- However, it is very limited tools for developing APIs
- Automated testing, where we manipulate the API as a Javascript client are considerably more useful
- For this, we need xUnit test frameworks and associated test runner tools.

https://mochajs.org/

simple, flexible, fun

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing *simple* and *fun*. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.

gitter join chat | backers 20 | sponsors 1

http://chaijs.com/

Assertion Styles

Chai has several interfaces that allow the developer to choose the most comfortable. The chain-capable BDD styles provide an expressive language & readable style, while the TDD assert style provides a more classical feel.

Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.length(3);
tea.should.have.property('flavors')
  .with.length(3);
```

Visit Should Guide

Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.length(3);
expect(tea).to.have.property('flavors')
  .with.length(3);
```

Visit Expect Guide

Assert

```
var assert = chai.assert;

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

Visit Assert Guide

Assert Style

- The assert style is exposed through assert interface. This provides the classic assert-dot notation, similar to that packaged with node.js.

```
var assert = require('chai').assert
  , foo = 'bar'
  , beverages = { tea: [ 'chai', 'matcha', 'oolong' ] };

assert.typeOf(foo, 'string'); // without optional message
assert.typeOf(foo, 'string', 'foo is a string'); // with optional message
assert.equal(foo, 'bar', 'foo equal `bar`');
assert.lengthOf(foo, 3, 'foo`s value has a length of 3');
assert.lengthOf(beverages.tea, 3, 'beverages has 3 types of tea');
```

Installation

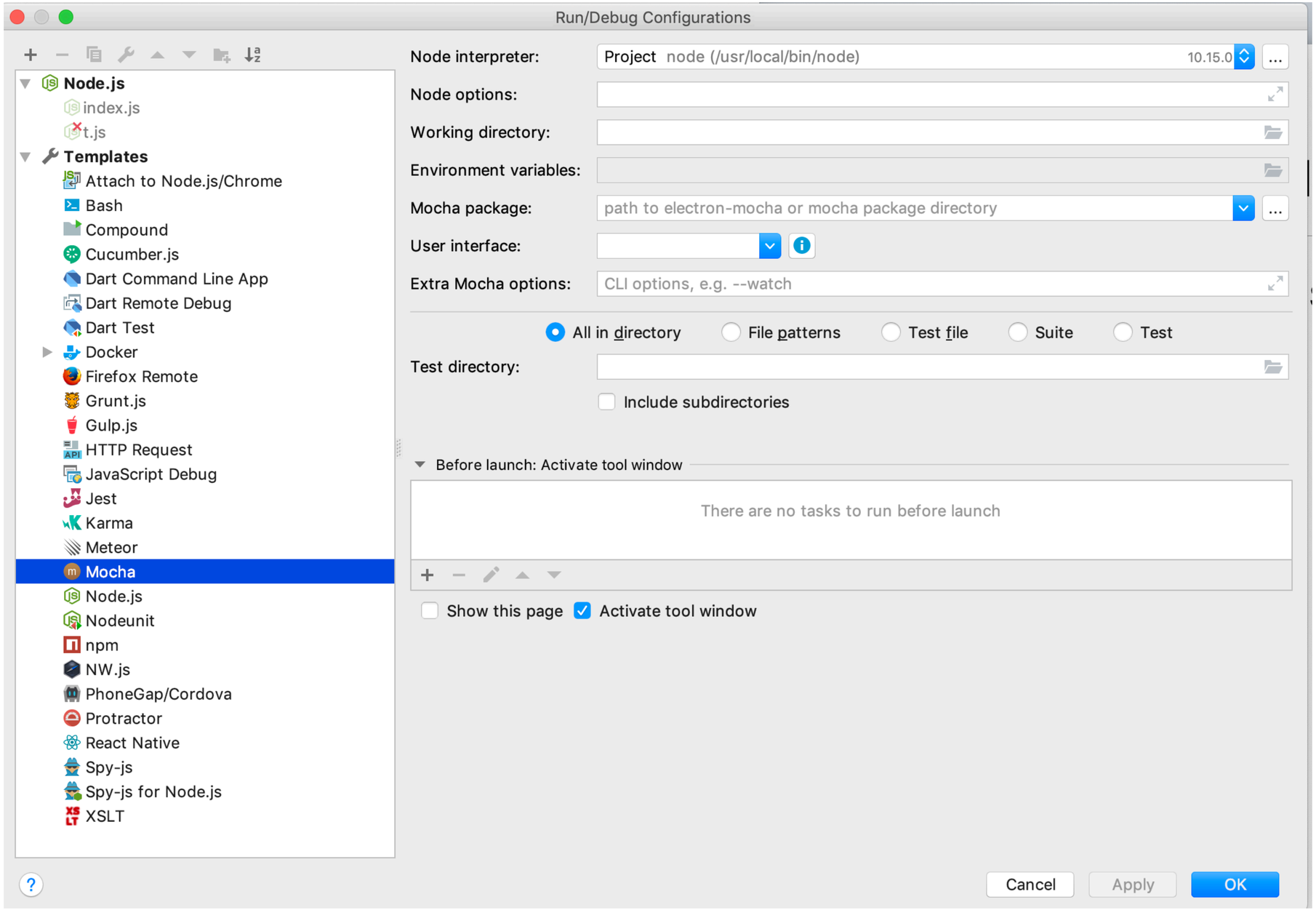
```
npm install mocha -save-dev
npm install chai -save-dev
```

- Install mocha + chai as ‘development’ dependencies

```
{
  "name": "donation-web",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "boom": "^7.3.0",
    "dotenv": "^6.2.0",
    "handlebars": "^4.0.12",
    "hapi": "^18.0.0",
    "hapi-auth-cookie": "^9.1.0",
    "inert": "^5.1.2",
    "joi": "^14.3.1",
    "mais-mongoose-seeder": "^1.0.7",
    "mongoose": "^5.4.7",
    "vision": "^5.4.4"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "mocha": "^6.0.1",
    "prettier": "^1.16.0"
  },
  "prettier": {
    "singleQuote": true,
    "printWidth": 120
  }
}
```

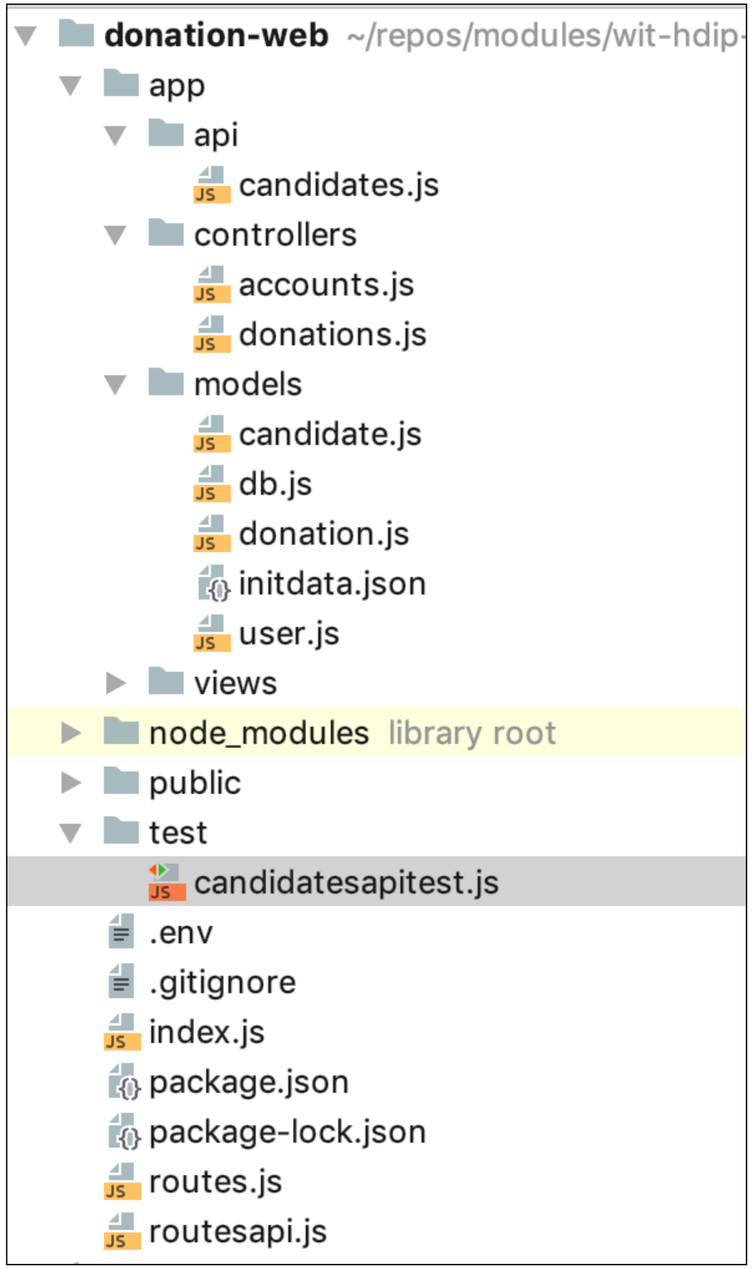
WebStorm Mocha Support

- A Mocha test runner will simplify running tests from within the IDE



Test Folder

- Package all tests in a separate high level test folder

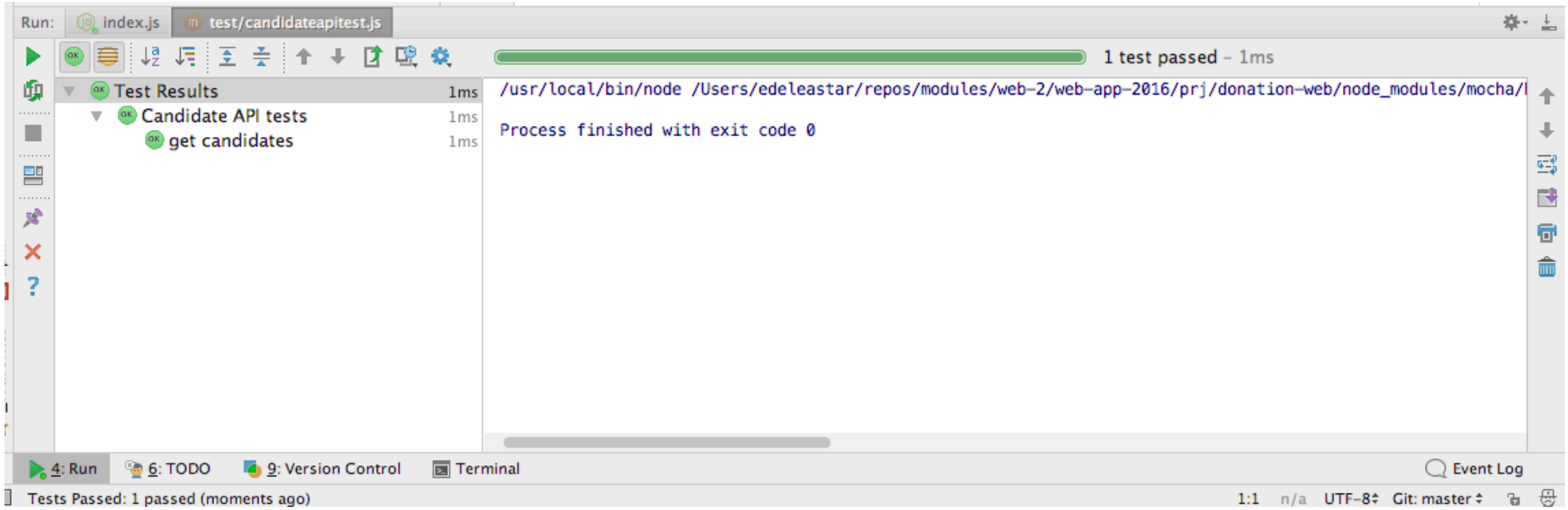


First Unit Test

- A complete unit test suite
- Always passes (1 == 1)
- Uses the ‘chai’ assertion library

```
'use strict';

const assert = require('chai').assert;

suite('Candidate API tests', function () {

  test('get candidates', function () {

    assert.equal(1, 1);

  });
});
```

Mocha Test Runner in Webstorm



- Convenient test runner, green for pass…

Mocha Test Runner in Webstorm

- Red for fail…

axios

0.18.0 • Public • Published a year ago

Axios : https://github.com/axios/axios

Readme | 2 Dependencies | 14,437 Dependents | 38 Versions

axios

npm v0.18.0 | build failing | coverage 94% | downloads 12M/m | chat on gitter | code helpers 64

Promise based HTTP client for the browser and node.js

Features

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF

Browser Support

install

```
> npm i axios
```

weekly downloads

3,687,144

version	license
0.18.0	MIT
open issues	**pull requests**
413	96
homepage	**repository**
github.com	github

last publish

5 months ago

```javascript
const axios = require('axios');

// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .then(function () {
    // always executed
  });

// Optionally the request above could also be done as
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  })
  .then(function () {
    // always executed
  });
```

```javascript
// Want to use async/await? Add the `async` keyword to your outer function/method.
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

Axios GET Requests

First API Test

- Not really a test, as we have no ‘assert’ yet.

```
'use strict';

const assert = require('chai').assert;
const axios = require('axios');

suite('Candidate API tests', function () {

  test('get candidates', async function () {
    const response = await axios.get('http://localhost:3000/api/candidates');
    console.log(response.data);
  });
});
```

```javascript
test('get candidates', async function () {
  const response = await axios.get('http://localhost:3000/api/candidates');
  const candidates = response.data;
  assert.equal(2, candidates.length);

  assert.equal(candidates[0].firstName, 'Lisa');
  assert.equal(candidates[0].lastName, 'Simpson');
  assert.equal(candidates[0].office, 'President');

  assert.equal(candidates[1].firstName, 'Donald');
  assert.equal(candidates[1].lastName, 'Simpson');
  assert.equal(candidates[1].office, 'President');
});
```

```json
{
  "users": {
    "_model": "User",
    "homer": {
      "firstName": "Homer",
      "lastName": "Simpson",
      "email": "homer@simpson.com",
      "password": "secret"
    },
    "marge": {
      "firstName": "Marge",
      "lastName": "Simpson",
      "email": "marge@simpson.com",
      "password": "secret"
    },
    "bart": {
      "firstName": "Bart",
      "lastName": "Simpson",
      "email": "bart@simpson.com",
      "password": "secret"
    }
  },
  "candidates": {
    "_model": "Candidate",
    "lisa": {
      "firstName": "Lisa",
      "lastName": "Simpson",
      "office": "President"
    },
    "donald": {
      "firstName": "Donald",
      "lastName": "Simpson",
      "office": "President"
    }
  },
  "donations": {
    "_model": "Donation",
    "one": {
      "amount": 40,
      "method": "paypal",
      "donor": "->users.bart",
      "candidate": "->candidates.lisa"
    },
    "two": {
      "amount": 90,
      "method": "direct",
      "donor": "->users.marge",
      "candidate": "->candidates.lisa"
    },
    "three": {
      "amount": 430,
      "method": "paypal",
      "donor": "->users.homer",
      "candidate": "->candidates.donald"
    }
  }
}
```

- Simple test to verify candidates preloaded by database seeding
- Not a sustainable approach to test data, but sufficient to get started

Get Single Candidate Endpoint Test

```
 test('get one candidate', async function () {
  let response = await axios.get('http://localhost:3000/api/candidates');
  const candidates = response.data;
  assert.equal(2, candidates.length);

  const oneCandidateUrl = 'http://localhost:3000/api/candidates/' + candidates[0]._id;
  response = await axios.get(oneCandidateUrl);
  const oneCandidate = response.data;

  assert.equal(oneCandidate.firstName, 'Lisa');
  assert.equal(oneCandidate.lastName, 'Simpson');
  assert.equal(oneCandidate.office, 'President');
});
```

- Get all Candidates first.
- Then use ID of first candidate to test get Single Candidate

Create Candidate Endpoint

```
{ method: 'POST', path: '/api/candidates', config: Candidates.create },
```

```
create: {
  auth: false,
  handler: async function(request, h) {
    const newCandidate = new Candidate(request.payload);
    const candidate = await newCandidate.save();
    if (candidate) {
      return h.response(candidate).code(201);
    }
    return Boom.badImplementation('error creating candidate');
  }
},
```

- Retrieve the candidate JSON from the payload
- Create and Save Mongo Object
- Return new candidate + http code ‘201 - Created’ - the valid response when a resource successfully added

Create Candidate Test

```
test('create a candidate', async function () {
  const candidatesUrl = 'http://localhost:3000/api/candidates';
  const newCandidate = {
    firstName: 'Barnie',
    lastName: 'Grumble',
    office: 'President',
  };

  const response = await axios.post(candidatesUrl, newCandidate);
  const returnedCandidate = response.data;
  assert.equal(201, response.status);

  assert.equal(returnedCandidate.firstName, 'Barnie');
  assert.equal(returnedCandidate.lastName, 'Grumble');
  assert.equal(returnedCandidate.office, 'President');
});
```

Rest Endpoints Verbs

- Comparing database (sql) and HTTP Verbs

SQL	REST
SELECT	GET
INSERT	POST
UPDATE	PUT
DELETE	DELETE

Action varies with HTTP Method

URI	HTTP METHOD	ACTION PERFORMED
/status/	GET	Get all status
/status/3	GET	Get status with id 3
/status/	POST	Add a new status
/status/4	PUT	Edit status with id 4
/status/4	DELETE	Delete status with id 4

HTTP Response Codes

HTTP Status Codes	Informational
200	OK
201	Resource created
204	No content
400	Bad Request
401	Unauthorised
404	Not found
405	Method Not allowed
500	Internal Server Error