Testing Endpoints

Automated Testing

- Using Postman or Insomnia are useful for exploring APIs
- However, they are 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.

http://chaijs.com/

Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.

Download Chai for Node

3.5.0 / 2016-01-28

Another platform? Browser Rails

The **chai** package is available on npm.

```
$ npm install chai
```

View Node Guide

Issues | Fork on GitHub | Releases | Google Group | Build Status

Getting Started

Learn how install and use Chai through a series of guided walkthroughs.

API Documentation

Explore the BDD & TDD language specifications for all available assertions.

Plugin Directory

Extend Chai's with additional assertions and vendor integration.

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

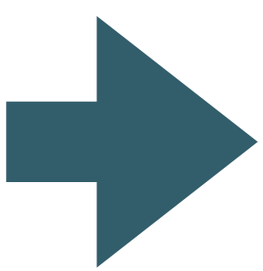
- The assert style is exposed through assert interface. This provides the classic assert-dot notation, similar to that packaged with node.js.

```
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
```

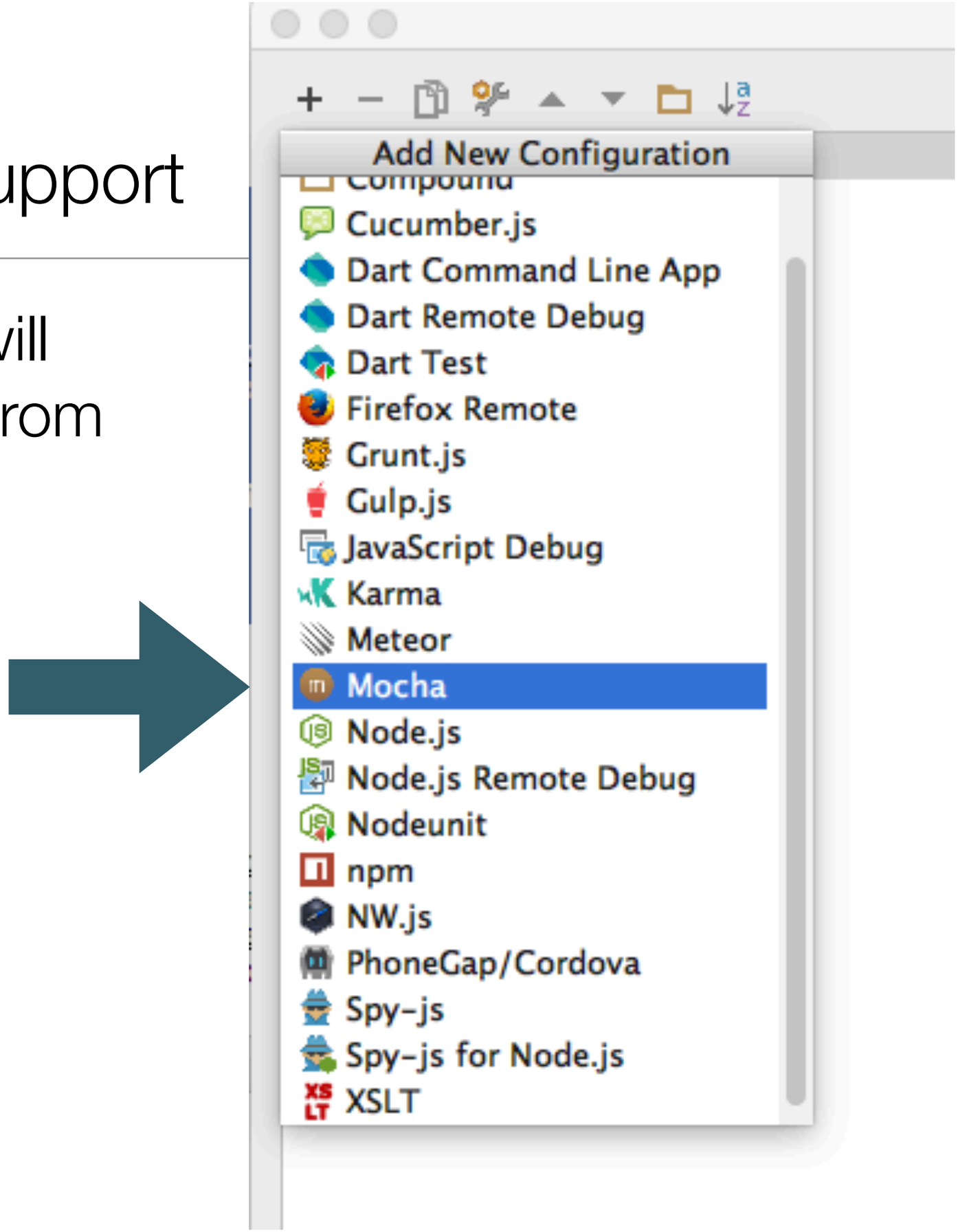
- Install mocha + chai as ‘development’ dependencies



```
{
  "name": "donation-web",
  "version": "1.0.0",
  "description": "an application to host donations for candidates",
  "main": "index.js",
  "scripts": {
    "start": "node index",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "boom": "^3.2.2",
    "handlebars": "^4.0.5",
    "hapi": "^14.1.0",
    "hapi-auth-cookie": "^6.1.1",
    "inert": "^4.0.1",
    "joi": "^9.0.4",
    "mongoose": "^4.5.8",
    "mongoose-seeder": "^1.2.1",
    "vision": "^4.1.0"
  },
  "devDependencies": {
    "chai": "^3.5.0",
    "mocha": "^3.0.2"
  }
}
```

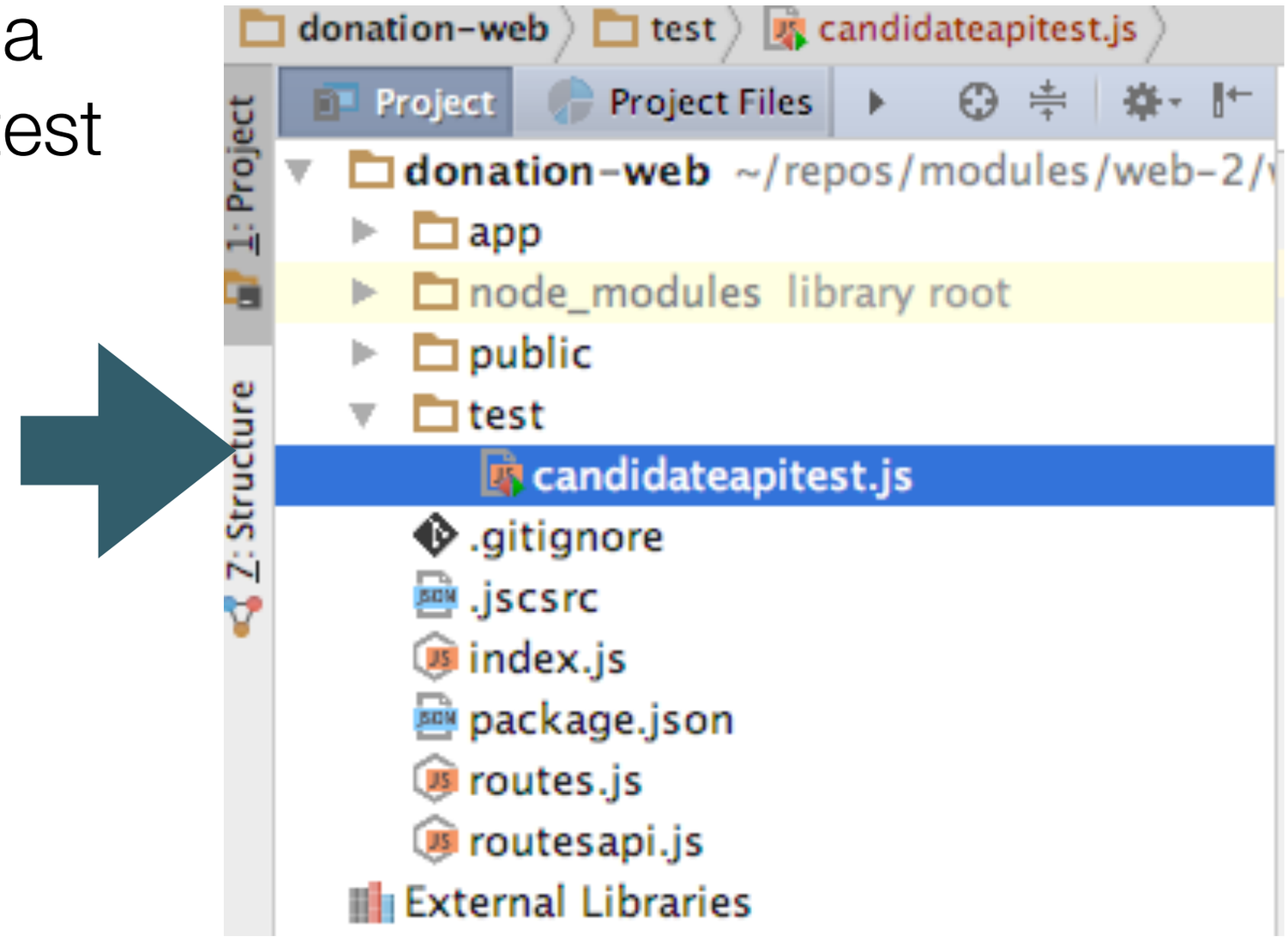
WebStorm Mocha Support

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



Test Folder

- Package all tests in a separate high level test folder

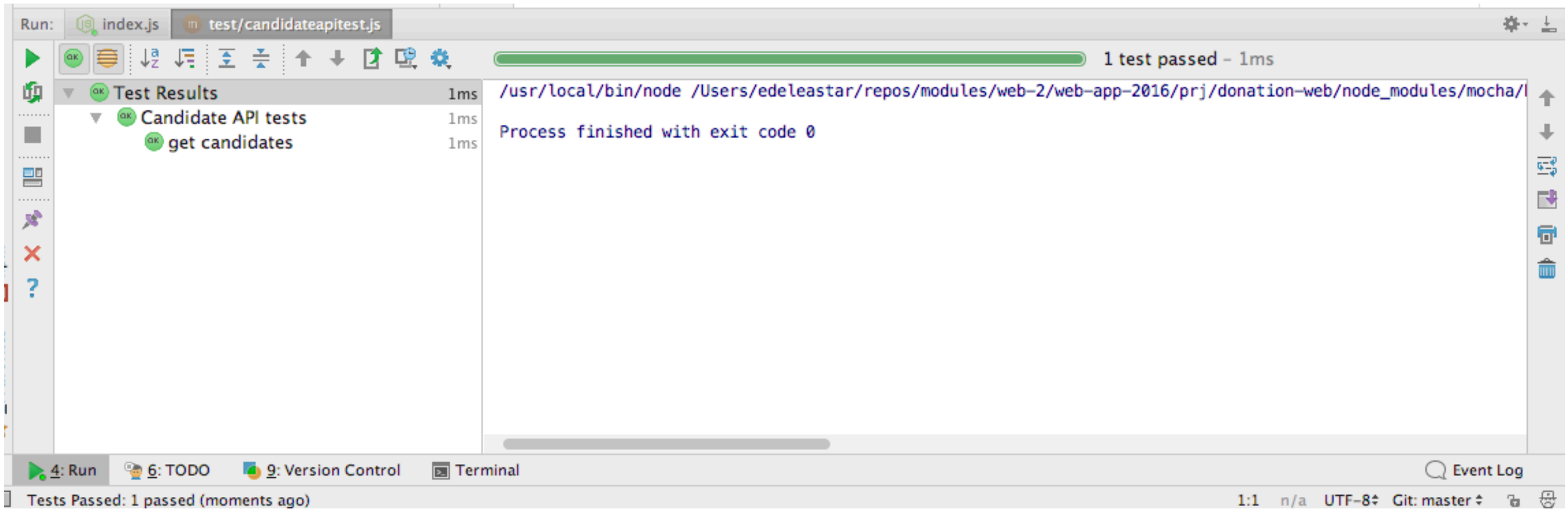


First Unit Test

- 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

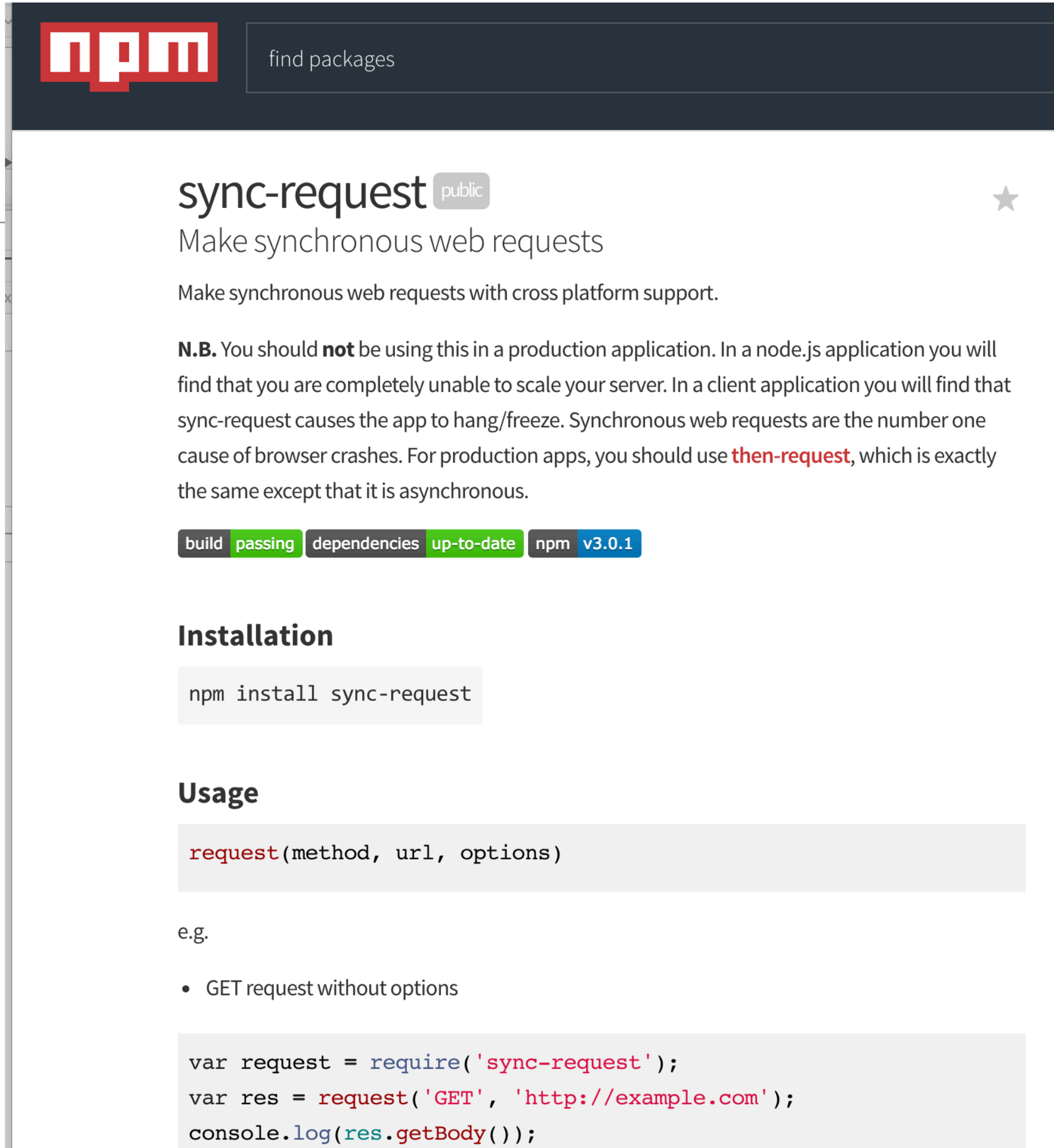
- Red for fail…

sync-request

- To keep unit tests simple, we can switch to synchronous mode.

- Although inefficient, this is not a concern for tests

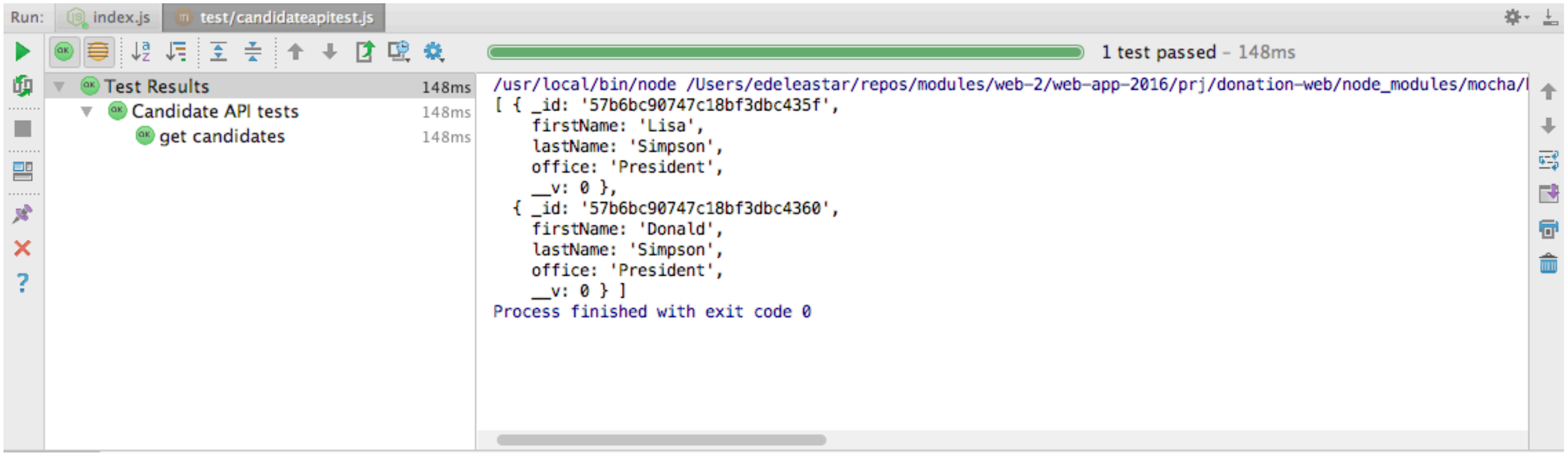
- Simplified tests (no callbacks, promises etc..) more clearly communicates developer intent



First API Test

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

```
'use strict';

const assert = require('chai').assert;
var request = require('sync-request');

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

  test('get candidates', function () {

    const url = 'http://localhost:4000/api/candidates';
    var res = request('GET', url);
    console.log(JSON.parse(res.getBody('utf8')));

  });
```



```
test('get candidates', function () {

  const url = 'http://localhost:4000/api/candidates';
  var res = request('GET', url);
  const candidates = JSON.parse(res.getBody('utf8'));

  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');

});
```

- Simple test to verify candidates preloaded by database seeding

- Not a sustainable approach to test data, but sufficient to get started


```
{
  "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"
    }
  }
}
```

Get Single Candidate Endpoint Test

```
test('get one candidate', function () {

  const allCandidatesUrl = 'http://localhost:4000/api/candidates';
  var res = request('GET', allCandidatesUrl);
  const candidates = JSON.parse(res.getBody('utf8'));

  const oneCandidateUrl = allCandidatesUrl + '/' + candidates[0]._id;
  res = request('GET', oneCandidateUrl);
  const oneCandidate = JSON.parse(res.getBody('utf8'));

  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: CandidatesApi.create },
```

```
exports.create = {

  auth: false,

  handler: function (request, reply) {
    const candidate = new Candidate(request.payload);
    candidate.save().then(newCandidate => {
      reply(newCandidate).code(201);
    }).catch(err => {
      reply(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', function () {

  const candidatesUrl = 'http://localhost:4000/api/candidates';
  const newCandidate = {
    firstName: 'Barnie',
    lastName: 'Grumble',
    office: 'President',
  };


  const res = request('POST', candidatesUrl, { json: newCandidate });
  const returnedCandidate = JSON.parse(res.getBody('utf8'));

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

});
```

Delete a Candidate Endpoint

```
{ method: 'DELETE', path: '/api/candidates/{id}', config: CandidatesApi.deleteOne },
```

```
exports.deleteOne = {

  auth: false,

  handler: function (request, reply) {
    Candidate.remove({ _id: request.params.id }).then(candidate => {
      reply(candidate).code(204);
    }).catch(err => {
      reply(Boom.notFound('id not found'));
    });
  },

};
```

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