

Testing an API

```
module.exports = [
  { method: 'GET', path: '/api/candidates', config: CandidatesApi.find },
  { method: 'GET', path: '/api/candidates/{id}', config: CandidatesApi.findOne },
  { method: 'POST', path: '/api/candidates', config: CandidatesApi.create },
  { method: 'DELETE', path: '/api/candidates/{id}', config: CandidatesApi.deleteOne },
  { method: 'DELETE', path: '/api/candidates', config: CandidatesApi.deleteAll },

  { method: 'GET', path: '/api/users', config: UsersApi.find },
  { method: 'GET', path: '/api/users/{id}', config: UsersApi.findOne },
  { method: 'POST', path: '/api/users', config: UsersApi.create },
  { method: 'DELETE', path: '/api/users/{id}', config: UsersApi.deleteOne },
  { method: 'DELETE', path: '/api/users', config: UsersApi.deleteAll },
];
```

- User & Candidate APIs

Candidate Tests

- The tests we have written so far are somewhat verbose and repetitive.
- For tests to be effective, they must remain concise and easy to maintain and evolve.

```
suite('Candidate API tests', function () {  
  
  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');  
  
  });  
  
  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');  
  
  });  
  
  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');  
  
  });  
});
```

Candidate Tests

- To simplify tests, we attempt to encapsulate both the http requests and the donation service access into a set of classes:
 - ***SyncHttpService***: encapsulate http requests/responses
 - ***DonationService***: deliver a client-side api to the remote service
- Simplify our tests and enable us to more easily devise more tests as the API evolves.

```
var request = require('sync-request');  
  
class SyncHttpService {  
  ...  
}  
  
module.exports = SyncHttpService;
```

```
const SyncHttpService = require('./sync-http-client');  
const baseUrl = 'http://localhost:4000';  
  
class DonationService {  
  ...  
}  
  
module.exports = DonationService;
```

SyncHttpClient - Specification

- This class serves as a simple wrapper around the sync-request module
- Assumes certain defaults, implement get, post and delete operations against a baseUrl (the remote service)

```
var request = require('sync-request');  
  
class SyncHttpClient {  
  constructor(baseUrl) {  
    ...  
  }  
  
  get(url) {  
    ...  
  }  
  
  post(url, obj) {  
    ...  
  }  
  
  delete (url) {  
    ...  
  }  
}  
  
module.exports = SyncHttpClient;
```

SyncHttpService - Implementation

- For get and post, If return code < 300, assume we have JSON payload

HTTP status codes

- 1xx Informational
- 2xx Success
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

```
var request = require('sync-request');

class SyncHttpService {

  constructor(baseUrl) {
    this.baseUrl = baseUrl;
  }

  get(url) {
    var returnedObj = null;
    var res = request('GET', this.baseUrl + url);
    if (res.statusCode < 300) {
      returnedObj = JSON.parse(res.getBody('utf8'));
    }

    return returnedObj;
  }

  post(url, obj) {
    var returnedObj = null;
    var res = request('POST', this.baseUrl + url, { json: obj });
    if (res.statusCode < 300) {
      returnedObj = JSON.parse(res.getBody('utf8'));
    }

    return returnedObj;
  }

  delete (url) {
    var res = request('DELETE', this.baseUrl + url);
    return res.statusCode;
  }
}

module.exports = SyncHttpService;
```

DonationService

- Use the HttpSyncService to deliver higher level API to test client code.
- Test code can now be rewritten to use this class - simplifying the code and eliminating some repetition.

```
const SyncHttpService = require('./sync-http-client');
const baseUrl = 'http://localhost:4000';

class DonationService {

  constructor(baseUrl) {
    this.httpService = new SyncHttpService(baseUrl);
  }

  getCandidates() {
    return this.httpService.get('/api/candidates');
  }

  getCandidate(id) {
    return this.httpService.get('/api/candidates/' + id);
  }

  createCandidate(newCandidate) {
    return this.httpService.post('/api/candidates', newCandidate);
  }

  deleteAllCandidates() {
    return this.httpService.delete('/api/candidates');
  }

  deleteOneCandidate(id) {
    return this.httpService.delete('/api/candidates/' + id);
  }

  getUsers() {
    return this.httpService.get('/api/users');
  }

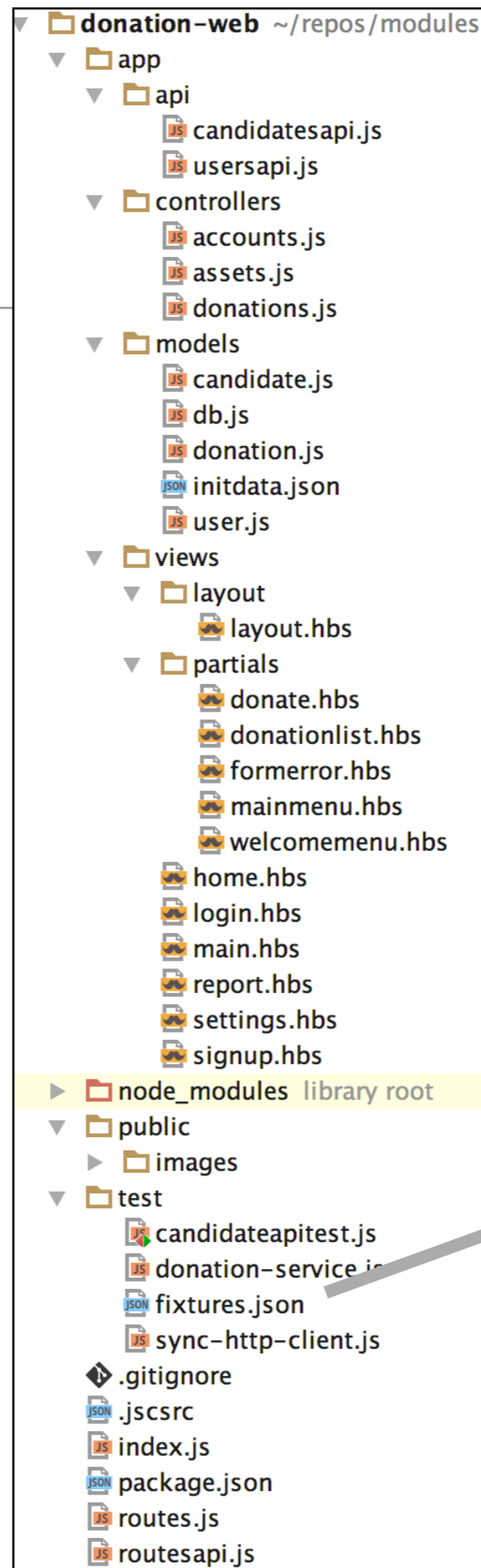
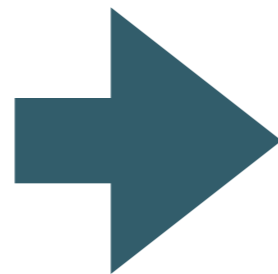
  getUser(id) {
    return this.httpService.get('/api/users/' + id);
  }

  createUser(newUser) {
    return this.httpService.post('/api/users', newUser);
  }
}

module.exports = DonationService;
```

Project Structure

- Test folder contains these wrapper classes + our unit tests



fixtures.json

```
{
  "candidates": [
    {
      "firstName": "Lisa",
      "lastName": "Simpson",
      "office": "President"
    },
    {
      "firstName": "Donald",
      "lastName": "Simpson",
      "office": "President"
    }
  ],
  "newCandidate": {
    "firstName": "Barnie",
    "lastName": "Grumble",
    "office": "President"
  }
}
```

test data

create a candidate test

```
'use strict';

const assert = require('chai').assert;
const DonationService = require('./donation-service');
const fixtures = require('./fixtures.json');

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

  let candidates = fixtures.candidates;
  let newCandidate = fixtures.newCandidate;

  const donationService = new DonationService('http://localhost:4000');

  test('create a candidate', function () {
    const returnedCandidate = donationService.createCandidate(newCandidate);
    assert.equal(returnedCandidate.firstName, newCandidate.firstName);
    assert.equal(returnedCandidate.lastName, newCandidate.lastName);
    assert.equal(returnedCandidate.office, newCandidate.office);
    assert.isDefined(returnedCandidate._id);
  });
});
```

```
{
  "candidates": [
    {
      "firstName": "Lisa",
      "lastName": "Simpson",
      "office": "President"
    },
    {
      "firstName": "Donald",
      "lastName": "Simpson",
      "office": "President"
    }
  ],
  "newCandidate":
  {
    "firstName": "Barnie",
    "lastName": "Grumble",
    "office": "President"
  }
}
```

create a candidate test

```
test('create a candidate', function () {
  const returnedCandidate = donationService.createCandidate(newCandidate);
  assert.equal(returnedCandidate.firstName, newCandidate.firstName);
  assert.equal(returnedCandidate.lastName, newCandidate.lastName);
  assert.equal(returnedCandidate.office, newCandidate.office);
  assert.isDefined(returnedCandidate._id);
});
```

```
{
  "candidates": [
    {
      "firstName": "Lisa",
      "lastName": "Simpson",
      "office": "President"
    },
    {
      "firstName": "Donald",
      "lastName": "Simpson",
      "office": "President"
    }
  ],
  "newCandidate":
  {
    "firstName": "Barnie",
    "lastName": "Grumble",
    "office": "President"
  }
}
```

- Test is now simplified, and easier to understand
- All access to the API is via donationService object

```
test('create a candidate', function () {
  const returnedCandidate = donationService.createCandidate(newCandidate);
  assert.equal(returnedCandidate.firstName, newCandidate.firstName);
  assert.equal(returnedCandidate.lastName, newCandidate.lastName);
  assert.equal(returnedCandidate.office, newCandidate.office);
  assert.isDefined(returnedCandidate._id);
});
```

- Simplified Test?



```
test('create a candidate', function () {
  const returnedCandidate = donationService.createCandidate(newCandidate);
  assert.equal(returnedCandidate, newCandidate);
  assert.isDefined(returnedCandidate._id);
});
```

- Will it pass?

```
assert.equal(returnedCandidate, newCandidate);
```

Comparison Failure

Side-by-side viewer | Do not ignore | Highlight words | 1 difference

Expected (Read-only)		Actual (Read-only)
{	1	1 {
"firstName": "Barnie"	2	"_v": 0
"lastName": "Grumble"	3	"_id": "57b803ffd31db75b5e0069cc"
"office": "President"	4	"firstName": "Barnie"
}	5	"lastName": "Grumble"
	6	"office": "President"
	7	}

- Returned object contains additional fields
- Equals will fail

lodash

- All purpose ‘swiss army knife’ of utilities for Javascript



Lo | Lodash
A modern JavaScript utility library delivering modularity, performance & extras.

[Documentation](#) [FP Guide](#)

```
_.defaults({ 'a': 1 }, { 'a': 3, 'b': 2 });  
// → { 'a': 1, 'b': 2 }  
_.partition([1, 2, 3, 4], n => n % 2);  
// → [[1, 3], [2, 4]]
```

[Star](#) 19,062 [Fork](#) 1,867 [Follow @bestiejs](#) [Tweet](#)

Download

[Core build \(~4kB gzipped\)](#)
[Full build \(~23kB gzipped\)](#)
[CDN copies](#)

Lodash is released under the [MIT license](#) & supports [modern environments](#).
Review the [build differences](#) & pick one that's right for you.

Installation

In a browser:

```
<script src="lodash.js"></script>
```

Using npm:

```
$ npm i -g npm  
$ npm i --save lodash
```

In Node.js:

```
// Load the full build.  
var _ = require('lodash');  
// Load the core build.  
var _ = require('lodash/core');  
// Load the FP build for immutable auto-curried iteratee-first data-last methods.  
var fp = require('lodash/fp');
```

- useful utility methods, particularly for manipulating arrays & collections

Search

⊕ Array

⊖ Collection

- _.countBy
- _.each -> forEach
- _.eachRight -> forEachRight
- _.every
- _.filter
- _.find
- _.findLast
- _.flatMap
- _.flatMapDeep
- _.flatMapDepth
- _.forEach
- _.forEachRight
- _.groupBy
- _.includes
- _.invokeMap
- _.keyBy
- _.map
- _.orderBy
- _.partition
- _.reduce
- _.reduceRight
- _.reject
- _.sample
- _.sampleSize
- _.shuffle
- _.size
- _.some
- _.sortBy

.some(collection, [predicate=.identity])

[source](#) [npm package](#)

Checks if predicate returns truthy for **any** element of collection. Iteration is stopped once predicate returns truthy. The predicate is invoked with three arguments: *(value, index|key, collection)*.

Since

0.1.0

Arguments

collection (Array/Object): The collection to iterate over.

[predicate=_.identity] (Function): The function invoked per iteration.

Returns

(boolean): Returns true if any element passes the predicate check, else false.

Example

```
_.some([null, 0, 'yes', false], Boolean);
// => true

var users = [
  { 'user': 'barney', 'active': true },
  { 'user': 'fred', 'active': false }
];

// The `_.matches` iteratee shorthand.
_.some(users, { 'user': 'barney', 'active': false });
// => false

// The `_.matchesProperty` iteratee shorthand.
_.some(users, ['active', false]);
// => true

// The `_.property` iteratee shorthand.
_.some(users, 'active');
```

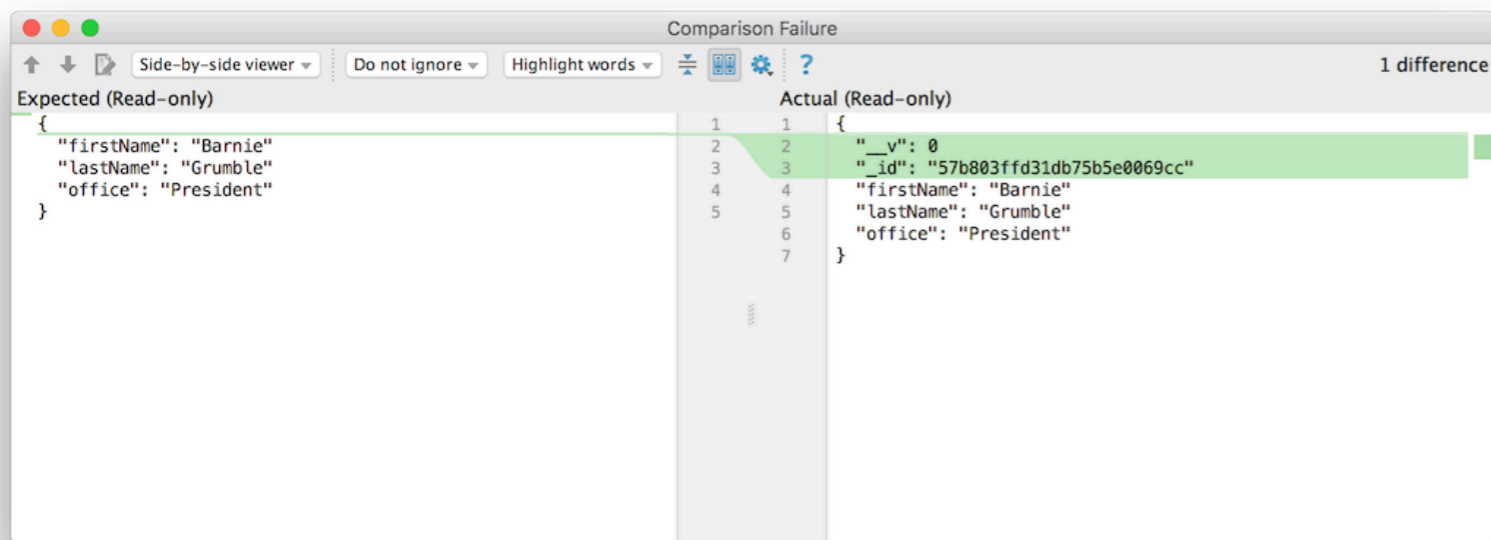
First we install lodash:

```
npm install lodash -save
```

Require is at the top of our test:

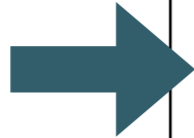
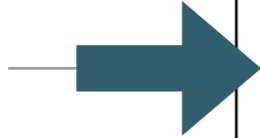
```
const _ = require('lodash');
```

```
test('create a candidate', function () {  
  const returnedCandidate = donationService.createCandidate(newCandidate);  
  assert(_.some([returnedCandidate], newCandidate),  
    'returnedCandidate must be a superset of newCandidate');  
  assert.isDefined(returnedCandidate._id);  
});
```



- assert true if returnedCandidate is a superset of candidate

- Called before and after each test.
- Ensures each test has a 'blank slate' to work with



```
suite('Candidate API tests', function () {  
  
  let candidates = fixtures.candidates;  
  let newCandidate = fixtures.newCandidate;  
  
  const donationService = new DonationService('http://localhost:4000');  
  
  beforeEach(function () {  
    donationService.deleteAllCandidates();  
  });  
  
  afterEach(function () {  
    donationService.deleteAllCandidates();  
  });  
  
  test('create a candidate', function () {  
    const returnedCandidate = donationService.createCandidate(newCandidate);  
    assert(_.some([returnedCandidate], newCandidate),  
           'returnedCandidate must be a superset of newCandidate');  
    assert.isDefined(returnedCandidate._id);  
  });  
  
  test('get candidate', function () {  
    const c1 = donationService.createCandidate(newCandidate);  
    const c2 = donationService.getCandidate(c1._id);  
    assert.deepEqual(c1, c2);  
  });  
  
  test('get invalid candidate', function () {  
    const c1 = donationService.getCandidate('1234');  
    assert.isNull(c1);  
    const c2 = donationService.getCandidate('012345678901234567890123');  
    assert.isNull(c2);  
  });  
  
  test('delete a candidate', function () {  
    const c = donationService.createCandidate(newCandidate);  
    assert(donationService.getCandidate(c._id) != null);  
    donationService.deleteOneCandidate(c._id);  
    assert(donationService.getCandidate(c._id) == null);  
  });  
});
```

Comprehensive Candidate Tests