# Testing an API

## Candidate Endpoints

```
const Candidates = require('./app/api/candidates');

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

## 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.

```javascript
'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');
    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');
  });

  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');
  });

  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');
  });
});
```

## Candidate Tests

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

```
class DonationService {
  constructor(baseUrl) {
    this.baseUrl = baseUrl;
  }

  async getCandidates() {
    try {
    const response = await axios.get(this.baseUrl + '/api/candidates');
    return response.data;
    } catch (e) {
      return null;
    }
  }

  async getCandidate(id) {
    try {
      const response = await axios.get(this.baseUrl + '/api/candidates/' + id);
      return response.data;
    } catch (e) {
      return null;
    }
  }

  async createCandidate(newCandidate) {
    try {
      const response = await axios.post(this.baseUrl + '/api/candidates', newCandidate);
      return response.data;
    } catch (e) {
      return null;
    }
  }

  async deleteAllCandidates() {
    try {
      const response = await axios.delete(this.baseUrl + '/api/candidates');
      return response.data;
    } catch (e) {
      return null;
    }
  }

  async deleteOneCandidate(id) {
    try {
      const response = await axios.delete(this.baseUrl + '/api/candidates/' + id);
      return response.data;
    } catch (e) {
      return null;
    }
  }
}
```

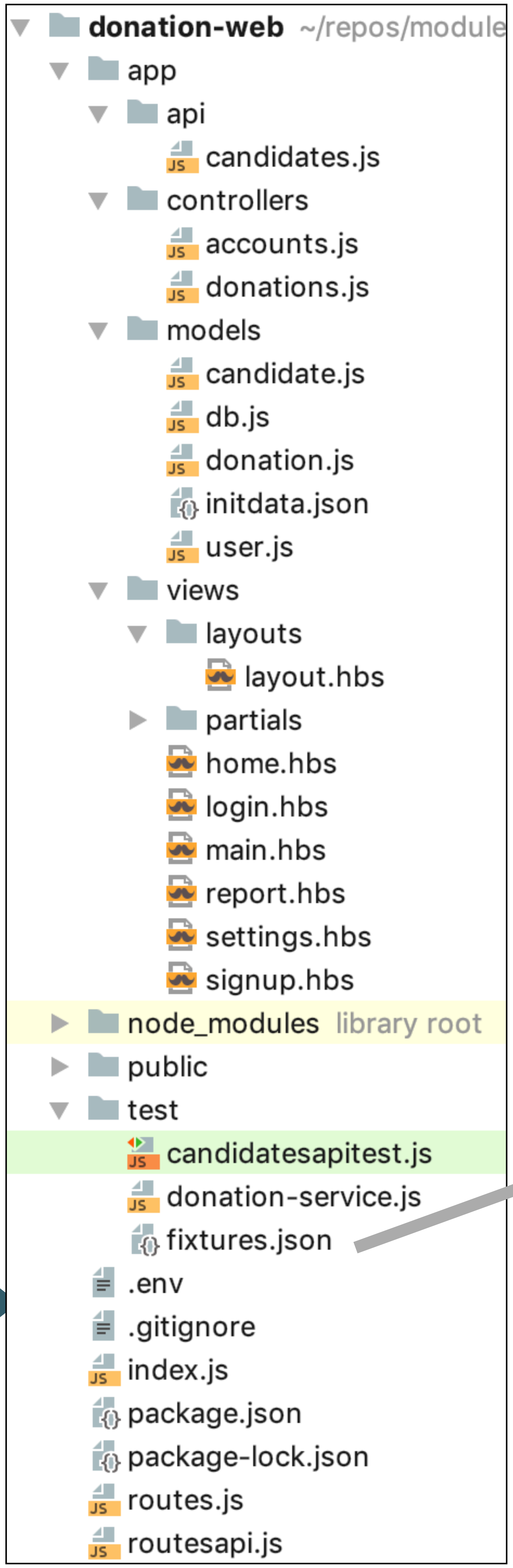
```
class DonationService {
  constructor(baseUrl) {
    this.baseUrl = baseUrl;
  }

  async getCandidates() {
    try {
    const response = await axios.get(this.baseUrl + '/api/candidates');
    return response.data;
    } catch (e) {
      return null;
    }
  }

  async getCandidate(id) {
    try {
      const response = await axios.get(this.baseUrl + '/api/candidates/' + id);
      return response.data;
    } catch (e) {
      return null;
    }
  }

  async createCandidate(newCandidate) {
    try {
      const response = await axios.post(this.baseUrl + '/api/candidates', newCandidate);
      return response.data;
    } catch (e) {
      return null;
    }
  }

  async deleteAllCandidates() {
    try {
      const response = await axios.delete(this.baseUrl + '/api/candidates');
      return response.data;
    } catch (e) {
      return null;
    }
  }

  async deleteOneCandidate(id) {
    try {
      const response = await axios.delete(this.baseUrl + '/api/candidates/' + id);
      return response.data;
    } catch (e) {
      return null;
    }
  }
}
```

## DonationService

- Use the axios 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.
- Swallow exceptions - and transform errors into null return values (for the moment).

## Project Structure

- Test folder contains these wrapper classes + our unit tests



```
donation-web ~/repos/module
  app
    api
      candidates.js
    controllers
      accounts.js
      donations.js
    models
      candidate.js
      db.js
      donation.js
      initdata.json
      user.js
    views
      layouts
        layout.hbs
      partials
      home.hbs
      login.hbs
      main.hbs
      report.hbs
      settings.hbs
      signup.hbs
  node_modules library root
  public
  test
    candidatesapitest.js
    donation-service.js
    fixtures.json
  .env
  .gitignore
  index.js
  package.json
  package-lock.json
  routes.js
  routesapi.js
```

fixtures.json

```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

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

```javascript
'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', async function () {
    const returnedCandidate = await 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);
  })

});
```

## create a candidate test

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

```javascript
test('create a candidate', async function () {
  const returnedCandidate = await 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);
})
```

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

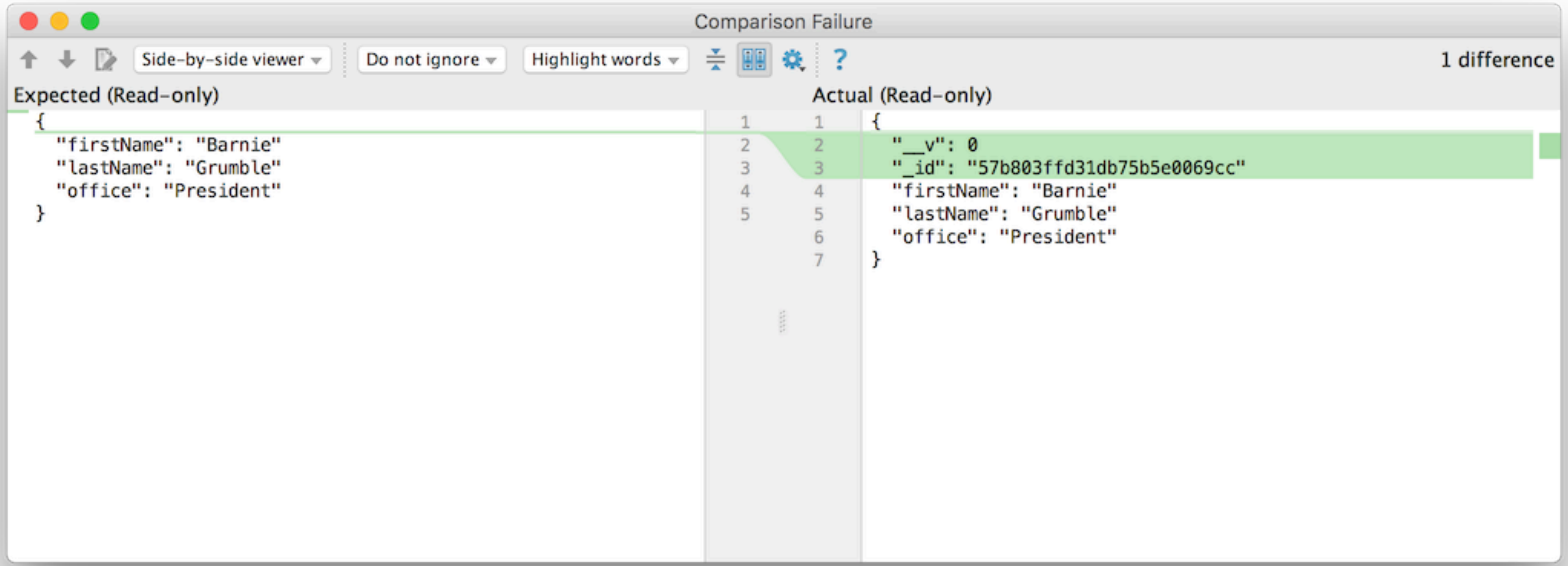
```
  test('create a candidate', async function () {
    const returnedCandidate = await donationService.createCandidate(newCandidate);
    assert.equal(returnedCandidate.firstName, newCandidate.firstName);
    assert.equal(returnedCandidate.lastName, newCandidate.lastName);
    assert.equal(returnedCandidate.office,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

- Simplified Test?

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

- Will is pass?

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



Expected (Read-only)

```
{
  "firstName": "Barnie"
  "lastName": "Grumble"
  "office": "President"
}
```

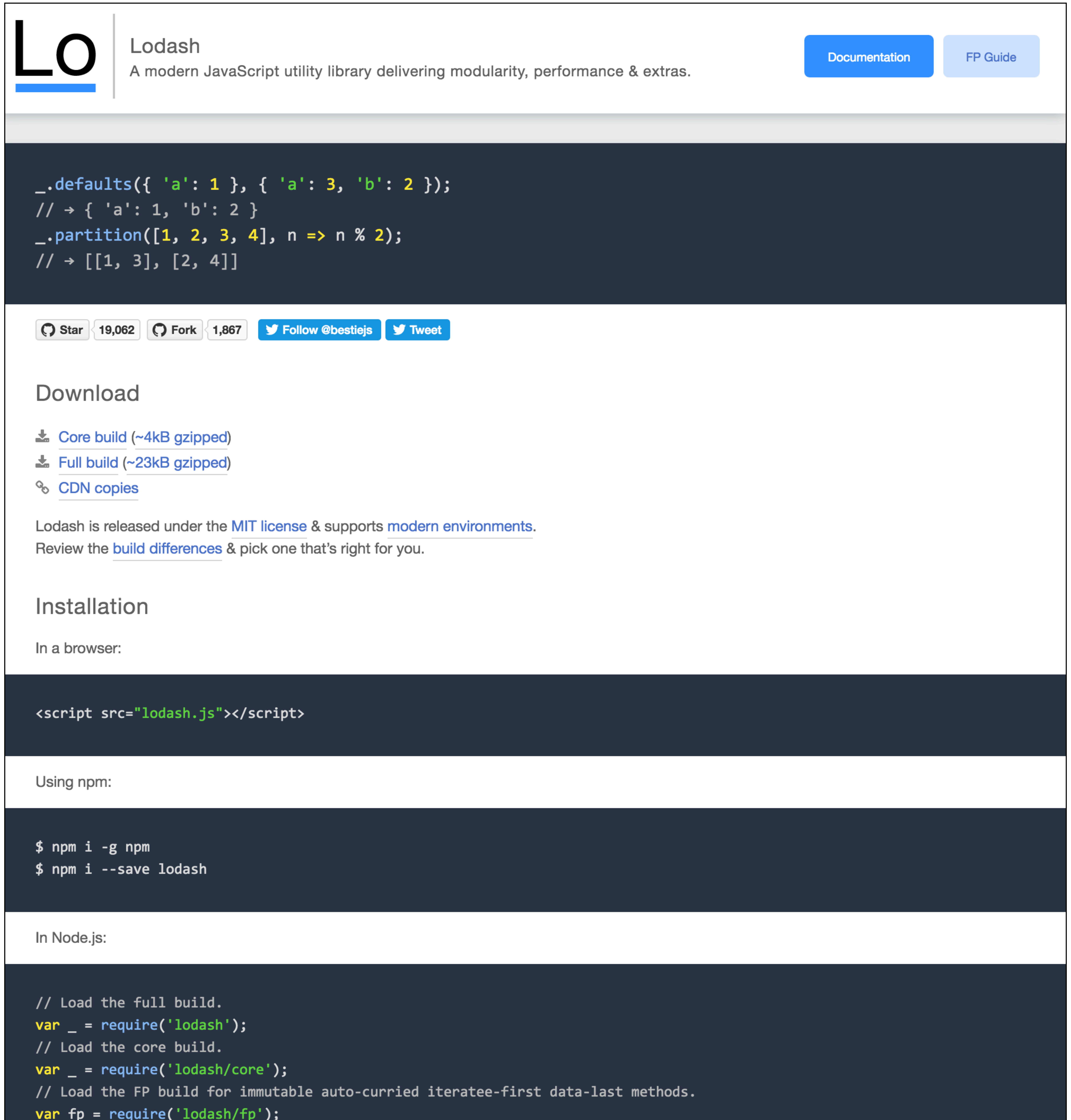
Actual (Read-only)

```
{
  "__v": 0
  "_id": "57b803ffd31db75b5e0069cc"
  "firstName": "Barnie"
  "lastName": "Grumble"
  "office": "President"
}
```

- Returned object contains additional fields
- Equals will fail

## loadash

- All purpose 'swiss army knife' of utilities for Javascript



Lo

Lodash

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

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

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>
```

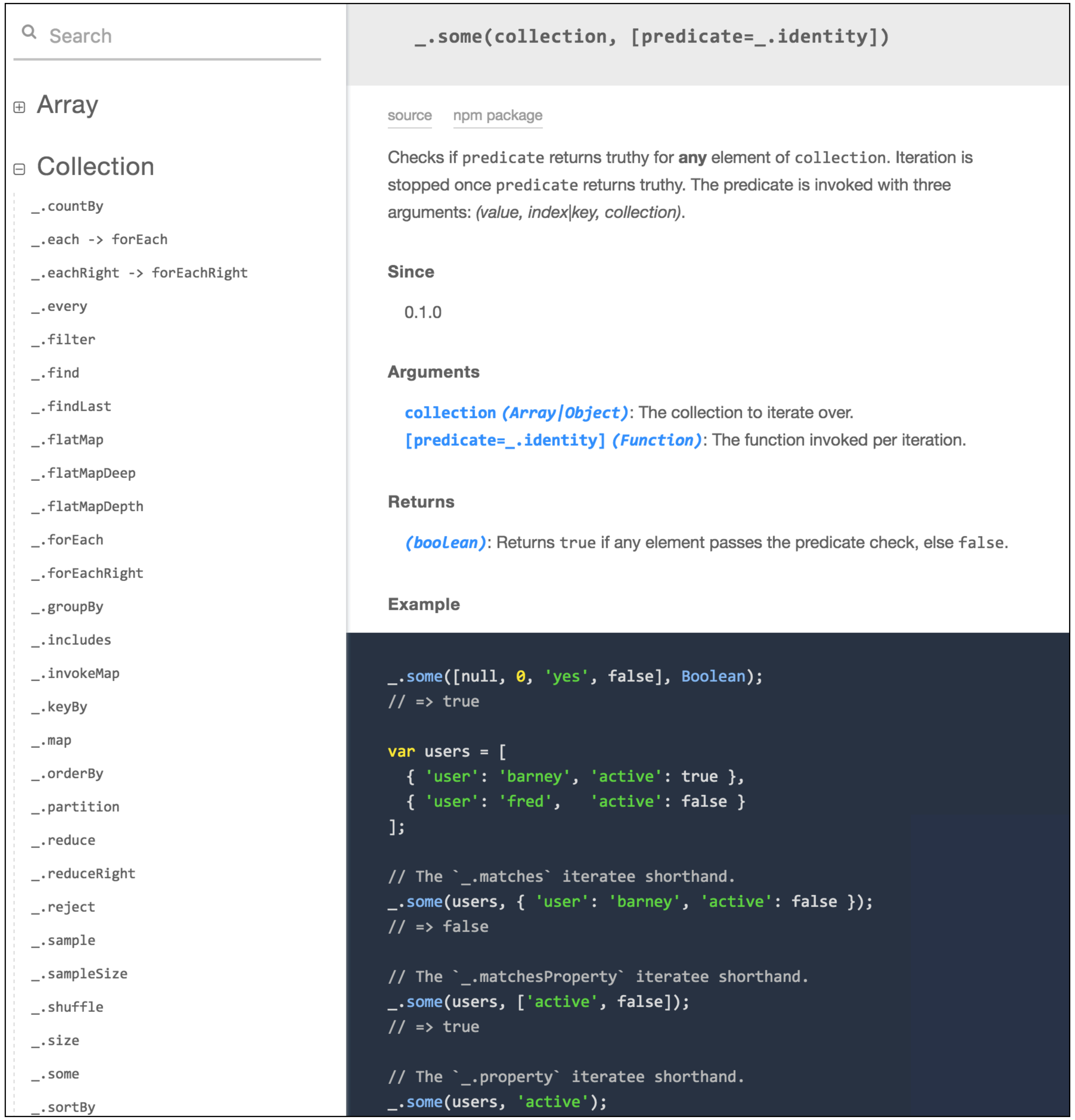
Using npm:

```
$ 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



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

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**

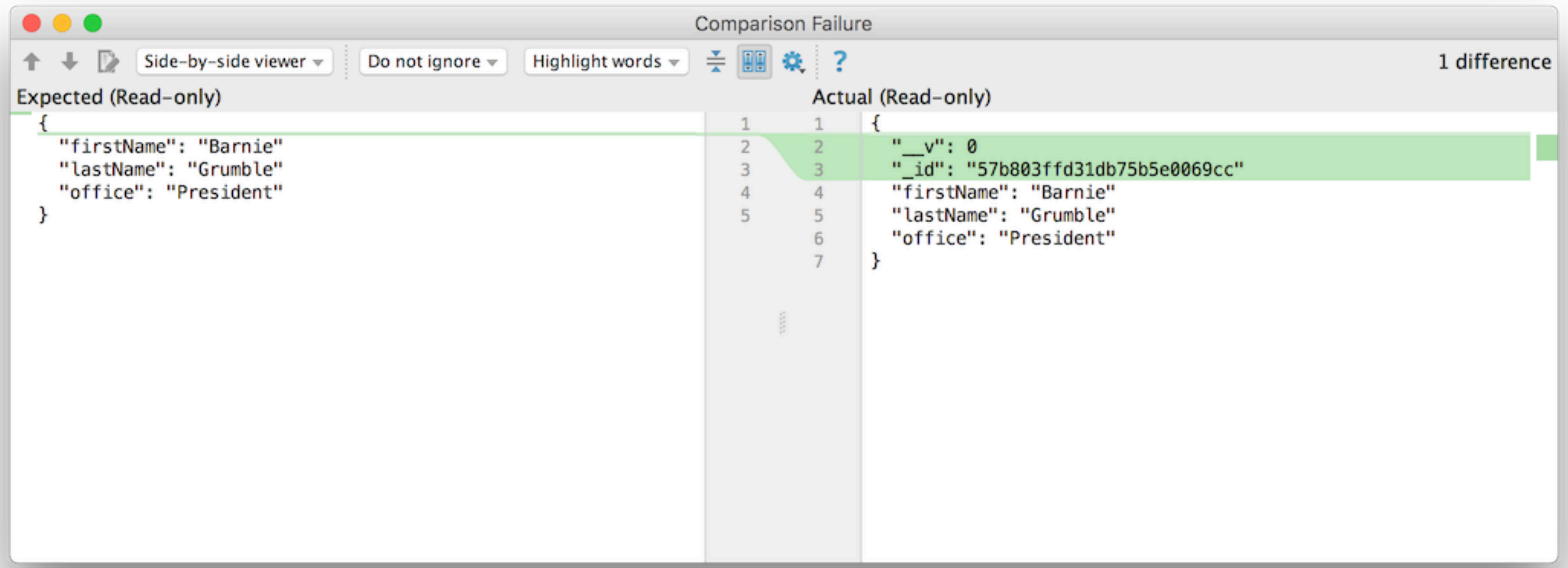
*(boolean)*: Returns `true` if any element passes the predicate check, else `false`.

**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');
```

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

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



Expected (Read-only)

```
{
  "firstName": "Barnie"
  "lastName": "Grumble"
  "office": "President"
}
```

Actual (Read-only)

```
{
  "__v": 0
  "_id": "57b803ffd31db75b5e0069cc"
  "firstName": "Barnie"
  "lastName": "Grumble"
  "office": "President"
}
```

- 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:3000');

  setup(async function () {
    await donationService.deleteAllCandidates();
  });

  teardown(async function () {
    await donationService.deleteAllCandidates();
  });

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

  test('get candidate', async function () {
    const c1 = await donationService.createCandidate(newCandidate);
    const c2 = await donationService.getCandidate(c1._id);
    assert.deepEqual(c1, c2);
  });

});
```

## Comprehensive Candidate Tests

- More extensive tests

```
test('get invalid candidate', async function () {
  const c1 = await donationService.getCandidate('1234');
  assert.isNull(c1);
  const c2 = await donationService.getCandidate('012345678901234567890123');
  assert.isNull(c2);
});


test('delete a candidate', async function () {
  let c = await donationService.createCandidate(newCandidate);
  assert(c._id != null);
  await donationService.deleteOneCandidate(c._id);
  c = await donationService.getCandidate(c._id);
  assert(c == null);
});

test('get all candidates', async function () {
  for (let c of candidates) {
    await donationService.createCandidate(c);
  }

  const allCandidates = await donationService.getCandidates();
  assert.equal(allCandidates.length, candidates.length);
});

test('get candidates detail', async function () {
  for (let c of candidates) {
    await donationService.createCandidate(c);
  }

  const allCandidates = await donationService.getCandidates();
  for (var i = 0; i < candidates.length; i++) {
    assert(_.some([allCandidates[i]], candidates[i]), 'returnedCandidate must be a superset of newCandidate');
  }
});

test('get all candidates empty', async function () {
  const allCandidates = await donationService.getCandidates();
  assert.equal(allCandidates.length, 0);
});
```

## SUT

- This is the System Under Test
- We now have a comprehensive test of this feature
- We have confidence now to:
  - Upgrade dependent APIs (e.g. mongoose)
  - Introduce Authentication
  - Change the Schema
  - Change the Mongo Provider
- All of the above in the knowledge that our tests will serve as a solid regression test to verify the stability of the feature.

```
const Candidates = {

  find: {
    auth: false,
    handler: async function(request, h) {
      const candidates = await Candidate.find();
      return candidates;
    }
  },

  findOne: {
    auth: false,
    handler: async function(request, h) {
      try {
        const candidate = await Candidate.findOne({ _id: request.params.id });
        if (!candidate) {
          return Boom.notFound('No Candidate with this id');
        }
        return candidate;
      } catch (err) {
        return Boom.notFound('No Candidate with this id');
      }
    }
  },

  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');
    }
  },

  deleteAll: {
    auth: false,
    handler: async function(request, h) {
      await Candidate.deleteMany({});
      return { success: true };
    }
  },

  deleteOne: {
    auth: false,
    handler: async function(request, h) {
      const response = await Candidate.deleteOne({ _id: request.params.id });
      if (response.deletedCount == 1) {
        return { success: true };
      }
      return Boom.notFound('id not found');
    }
  }
};
```