Node.js Meetup Berlin
17 October 2017
@robinpokorny

Async testing | KOA *with* JEST

Node.js Meetup Berlin
17 October 2017
@robinpokorny

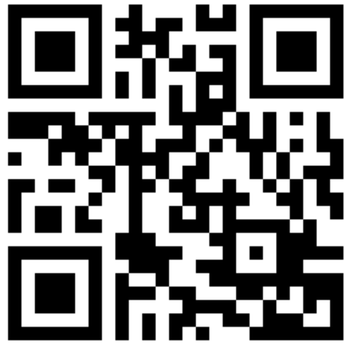
INFO

Slides accompany a talk.

Here, the talk is missing.

I wrote a transcript which can substitute the talk.

Find it on this link:



bit.ly/jest-koa

A

testing | **JEST**

WHAT IS KOA

next generation web framework

Express' spiritual successor

using ES2017 async/await (no callback hell, yay!)

http://koajs.com/

WHAT IS JEST

delightful, zero configuration testing platform

Jasmine's and Expect's (spiritual) successor

first-class mocking, snapshots, async testing

http://facebook.github.io/jest/

1 | *Testing*
MIDDLEWARE

2 | *Testing*
API

1

Testing

MIDDLEWARE

2

Testing

API

```
const greetings = async (ctx, next) ⇒ {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

const app = new Koa()
app.use(greetings)
app.listen(3000)
```

```
const greetings = async (ctx, next) ⇒ {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

const app = new Koa()
app.use(greetings)
app.listen(3000)
```

```
const greetings = async (ctx, next) ⇒ {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

const app = new Koa()
app.use(greetings)
app.listen(3000)
```

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

const app = new Koa()
app.use(greetings)
app.listen(3000)
```

```
const greetings = async (ctx, next) ⇒ {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

const app = new Koa()
app.use(greetings)
app.listen(3000)
```

```
const Koa = require('koa');
const app = new Koa();

// x-response-time

app.use(async (ctx, next) => {
  const start = Date.now();
  await next();
  const ms = Date.now() - start;
  ctx.set('X-Response-Time', `${ms}ms`);
});

// logger

app.use(async (ctx, next) => {
  const start = Date.now();
  await next();
  const ms = Date.now() - start;
  console.log(`${ctx.method} ${ctx.url} - ${ms}`);
});

// response

app.use(async ctx => {
  ctx.body = 'Hello World';
});

app.listen(3000);
```

```
const Koa = require('koa');
const app = new Koa();

// x-response-time

app.use(async (ctx, next) => {
  const start = Date.now();
  await next();
  const ms = Date.now() - start;
  ctx.set('X-Response-Time', `${ms}ms`);
});

// logger

app.use(async (ctx, next) => {
  const start = Date.now();
  await next();
  const ms = Date.now() - start;
  console.log(`${ctx.method} ${ctx.url} - ${ms}`);
});

// response

app.use(async ctx => {
  ctx.body = 'Hello World';
});

app.listen(3000);
```

SIMPLE TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}
```

```
test('greetings works', async () => {
  const ctx = {}

  await greetings(ctx, () => {})

  expect(ctx.body).toBe(
    'Hello. Remember to subscribe.'
  )
})
```

SIMPLE TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}
```

```
test('greetings works', async () => {
  const ctx = {}

  await greetings(ctx, () => {})

  expect(ctx.body).toBe(
    'Hello. Remember to subscribe.'
  )
})
```

SIMPLE TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}
```

```
test('greetings works', async () => {
  const ctx = {}

  await greetings(ctx, () => {})

  expect(ctx.body).toBe(
    'Hello. Remember to subscribe.'
  )
})
```

SIMPLE TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}
```

```
test('greetings works', async () => {
  const ctx = {}

  await greetings(ctx, () => {})

  expect(ctx.body).toBe(
    'Hello. Remember to subscribe.'
  )
})
```

SIMPLE TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}
```

```
test('greetings works', async () => {
  const ctx = {}

  await greetings(ctx, () => {})

  expect(ctx.body).toBe(
    'Hello. Remember to subscribe.'
  )
})
```

SIMPLE TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}
```

```
test('greetings works', async () => {
  const ctx = {}

  await greetings(ctx, () => {})

  expect(ctx.body).toBe(
    'Hello. Remember to subscribe.'
  )
})
```

SIMPLE TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}
```

```
test('greetings works', async () => {
  const ctx = {}

  await greetings(ctx, () => {})

  expect(ctx.body).toBe(
    'Hello. Remember to subscribe.'
  )
})
```

BEFORE-AND-AFTER TEST

```
const greetings = async (ctx, next) ⇒ {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

test('greetings works in order', async () ⇒ {
  const ctx = {}
  const next = jest.fn(() ⇒ {
    expect(ctx.body).toBe('Hello.')
    ctx.body += ' I am content.'
  })
  await greetings(ctx, next)
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx.body).toBe(
    'Hello. I am content. Remember to subscribe.'
  )
})
```

BEFORE-AND-AFTER TEST

```
const greetings = async (ctx, next) ⇒ {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

test('greetings works in order', async () ⇒ {
  const ctx = {}
  const next = jest.fn(() ⇒ {
    expect(ctx.body).toBe('Hello.')
    ctx.body += ' I am content.'
  })
  await greetings(ctx, next)
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx.body).toBe(
    'Hello. I am content. Remember to subscribe.'
  )
})
```

← *before*

BEFORE-AND-AFTER TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

test('greetings works in order', async () => {
  const ctx = {}
  const next = jest.fn(() => {
    expect(ctx.body).toBe('Hello.')
    ctx.body += ' I am content.'
  })
  await greetings(ctx, next)
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx.body).toBe(
    'Hello. I am content. Remember to subscribe.'
  )
})
```

← *before*

← *after*

BEFORE-AND-AFTER TEST

```
const greetings = async (ctx, next) => {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

test('greetings works in order', async () => {
  const ctx = {}
  const next = jest.fn(() => {
    expect(ctx.body).toBe('Hello.')
    ctx.body += ' I am content.'
  })
  await greetings(ctx, next)
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx.body).toBe(
    'Hello. I am content. Remember to subscribe.'
  )
})
```

← *before*

← *after*

BEFORE-AND-AFTER TEST

```
const greetings = async (ctx, next) ⇒ {
  ctx.body = 'Hello.'
  await next()
  ctx.body += ' Remember to subscribe.'
}

test('greetings works in order', async () ⇒ {
  const ctx = {}
  const next = jest.fn(() ⇒ {
    expect(ctx.body).toBe('Hello.')          ← before
    ctx.body += ' I am content.'
  })
  await greetings(ctx, next)
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx.body).toBe(                     ← after
    'Hello. I am content. Remember to subscribe.'
  )
})
```

COMPLETE TEST

```
test('greetings works complete', async () ⇒ {
  const ctx = {
    response: { set: jest.fn() }
    /* ADD OTHER MOCKS */
  }
  const next = jest.fn(() ⇒ {
    expect(ctx).toMatchSnapshot()
  })
  await expect(greetings(ctx, next))
    .resolves.toBeUndefined()
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx).toMatchSnapshot()
  expect(ctx.response.set.mock.calls).toMatchSnapshot()
})
```

COMPLETE TEST

```
test('greetings works complete', async () => {
  const ctx = {
    response: { set: jest.fn() }
    /* ADD OTHER MOCKS */
  }
  const next = jest.fn(() => {
    expect(ctx).toMatchSnapshot()
  })
  await expect(greetings(ctx, next))
    .resolves.toBeUndefined()
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx).toMatchSnapshot()
  expect(ctx.response.set.mock.calls).toMatchSnapshot()
})
```

SNAPSHOT

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`greetings works complete 1`] = `
Object {
  "body": “Hello.",
  "response": Object {
    "set": [Function],
  },
}
`;

exports[`greetings works complete 2`] = `
…
`;
```

COMPLETE TEST

```
test('greetings works complete', async () => {
  const ctx = {
    response: { set: jest.fn() }
    /* ADD OTHER MOCKS */
  }
  const next = jest.fn(() => {
    expect(ctx).toMatchSnapshot()
  })
  await expect(greetings(ctx, next))
    .resolves.toBeUndefined()
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx).toMatchSnapshot()
  expect(ctx.response.set.mock.calls).toMatchSnapshot()
})
```

SNAPSHOT

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`greetings works complete 3`] = `
Array [
  Array [
    "Etag",
    1234,
  ],
]
`;
```

COMPLETE TEST

```
test('greetings works complete', async () => {
  const ctx = {
    response: { set: jest.fn() }
    /* ADD OTHER MOCKS */
  }
  const next = jest.fn(() => {
    expect(ctx).toMatchSnapshot()
  })
  await expect(greetings(ctx, next))
    .resolves.toBeUndefined()
  expect(next).toHaveBeenCalledTimes(1)
  expect(ctx).toMatchSnapshot()
  expect(ctx.response.set.mock.calls).toMatchSnapshot()
})
```

.RESOLVES & .REJECTS

Better error messages

More errors

Readable and short

Read error

✖

Expected received Promise to resolve, instead it rejected to value
 [Error: Read error]

by @kentcdodds

by @kentcdodds

1 | *Testing* **MIDDLEWARE**

2 | *Testing* **API**

MORE THAN SUM

App ≠ compose(app.middleware)

Koa wraps the native response and request

API testing, HTTP assertions

SUPERTEST

HTTP assertions library

wrapper over SuperAgent

support for Promises

https://github.com/visionmedia/supertest

A CLEAR AND CONCISE INTRODUCTION TO TESTING KOA WITH JEST AND SUPERTEST

Valentino Gagliardi

https://www.valentinog.com/blog/testing-api-koa-jest/

SAMPLE APP

```
// server/index.js
const app = new Koa()
const router = new Router()

router.get('/', async ctx ⇒ {
  ctx.body = {
    data: 'Sending some JSON',
    person: {
      name: ‘Ferdinand',        lastname: 'Vaněk',
      role: 'Brewery worker’,   age: 42
    }
  }
})
app.use(router.routes())
module.exports = app
```

```
app.listen(3000)
```

✖

```
app.callback()
```

creates server

need to close after each test

Supertest will open and close

the server for us.

TEST BOILERPLATE

```
// test/root.spec.js

const request = require('supertest')
const app = require(‘../server')

test('root route', async () ⇒ {
  const response = await request(app.callback()).get(‘/');

  expect(response).toBeDefined() // @TODO
})
```

TEST BOILERPLATE

```
// test/root.spec.js

const request = require('supertest')
const app = require(‘../server')

test('root route', async () ⇒ {
  const response = await request(app.callback()).get(‘/');

  expect(response).toBeDefined() // @TODO
})
```

TEST BOILERPLATE

```
// test/root.spec.js

const request = require('supertest')
const app = require(‘../server')

test('root route', async () ⇒ {
  const response = await request(app.callback()).get(‘/');

  expect(response).toBeDefined() // @TODO
})
```

ITEM-LEVEL ASSERTIONS

```
expect(response.status).toEqual(200)

expect(response.type).toEqual(‘application/json')

expect(response.body.data).toEqual('Sending some JSON’)

expect(Object.keys(response.body.person)).toEqual(
  expect.arrayContaining(['name', 'lastname', 'role', 'age'])
)
```

ITEM-LEVEL ASSERTIONS

```
expect(response.status).toEqual(200)

expect(response.type).toEqual(‘application/json')

expect(response.body.data).toEqual('Sending some JSON’)

expect(Object.keys(response.body.person)).toEqual(
  expect.arrayContaining(['name', 'lastname', 'role', 'age'])
)
```

OBJECT EQUALITY

```
expect(response.body).toEqual(
  expect.objectContaining({
    person: {
      name: expect.anything(),
      lastname: expect.any(String),
      role: expect.stringMatching(/^Brewery/),
      age: expect.any(Number)
    }
  })
)
```

OBJECT EQUALITY

```
expect(response.body).toEqual(
  expect.objectContaining({
    person: {
      name: expect.anything(),
      lastname: expect.any(String),
      role: expect.stringMatching(/^Brewery/),
      age: expect.any(Number)
    }
  })
)
```

`expect.objectContaining({ x: 1 })` ✖ `{ x: 1 }`

SNAPSHOTS

```
expect(response.body).toMatchSnapshot()
```

```
// test/__snapshots__/root.spec.js.snap
exports[`root route with object equality 1`] = `
Object {
  "data": "Sending some JSON",
  "person": Object {
    "age": 42,
    "lastname": "VanÄ›k",
    "name": "Ferdinand",
    "role": "Brewery worker",
  },
}
`;
```

TDD × SNAPSHOTS

algorithms

write before

part

structures

concurrent or after

whole

NOT ONLY KOA

applies to other frameworks

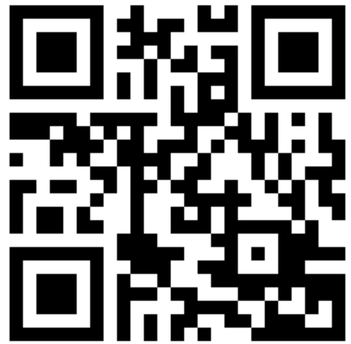
API testing - no change

convenient for refactoring

RELATED

- A clear and concise introduction to testing Koa with Jest and Supertest
- An Introduction to Building TDD RESTful APIs with Koa 2, Mocha and Chai
  - both by Valentino Gagliardi
- API testing with Jest by Koen van Gilst
- Testing async/await middleware? (GitHub Issue)
- Async testing in Jest (recording of presentation)
- Snapshot Testing APIs with Jest by Dave Ceddia
- Snapshot testing in Jest (recording of presentation)

@robinpokorny



bit.ly/jest-koa

ARTICLE