SZERVER OLDALI JAVASCRIPT

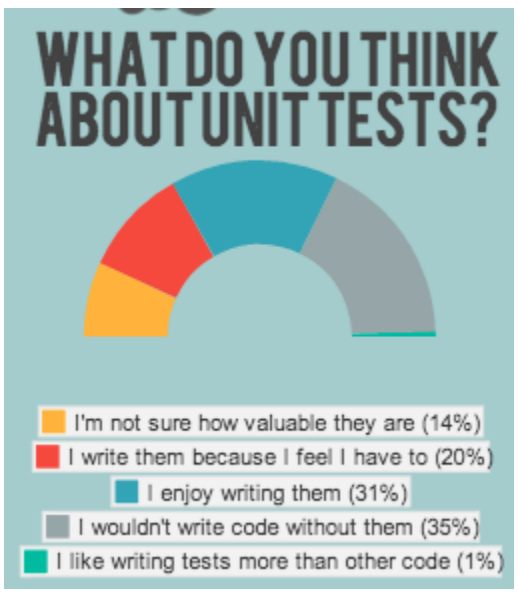
12. hét

Unit és integrációs tesztek, mocha, assert struktúrák, TDD / DBB

TESZTELÉS

Miért tesztelünk?

TESZTELÉS


WHAT DO YOU THINK ABOUT UNIT TESTS?
I'm not sure how valuable they are (14%)
I write them because I feel I have to (20%)
I enjoy writing them (31%)
I wouldn't write code without them (35%)
I like writing tests more than other code (1%)

UNIT TEST

Elkészítünk egy modult, egy funkciót, egy nagyon pici elemi egységet, és kíváncsiak vagyunk, fut-e: manuálisan teszteljük.

Ha a kis egység működése már definiálva van, akkor az egyes szabályokra lehet teszteket írni.

UNIT TEST

```
function osszead(a,b){
   return a + b;
}
```

Mit teszteljek?

- osszead(1,2) ?= 3
- osszead(2,1) ?= 3
- osszead(-1,2) ?= 1

UNIT TEST

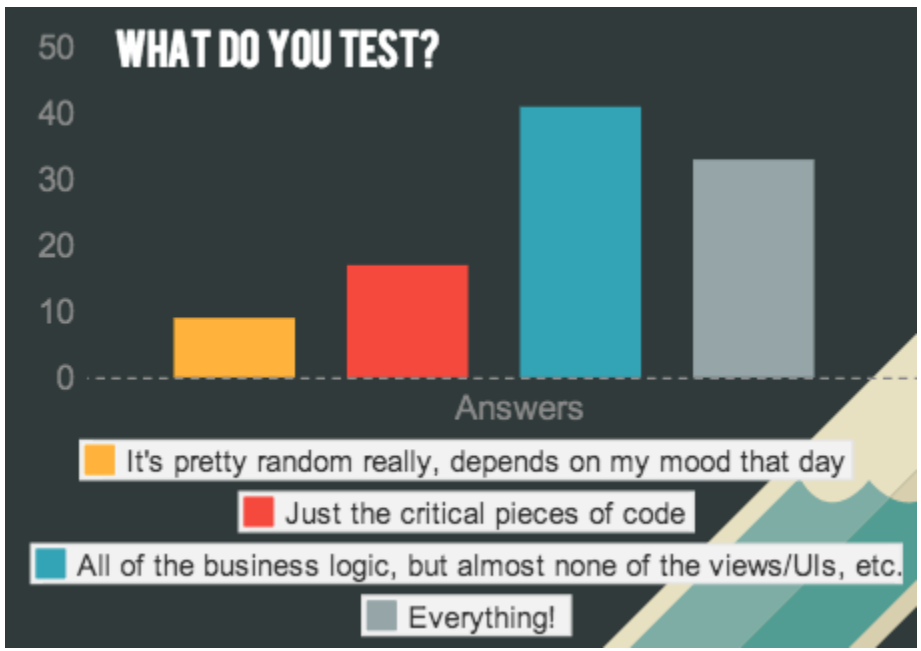
Mi van, ha változik a kód?

```
function osszead(a,b){
    if (a>20)
        return 20;
   return a + b;
}
```

- osszead(1,2) ?= 3
- osszead(2,1) ?= 3
- osszead(-1,2) ?= 1

Kell-e működnie 20-nál nagyobb számokra az osszeadas-nak?

UNIT TEST


WHAT DO YOU TEST?
50
40
30
20
10
0
Answers
It's pretty random really, depends on my mood that day
Just the critical pieces of code
All of the business logic, but almost none of the views/UIs, etc.
Everything!

INTEGRÁCIÓS TESZT

Ha már az építőkövek tesztelve vannak, nagyobb folyamatrészleteket / folyamatokat is lehet tesztelni. Ezt még mindig kód szintű segítséggel.

END-TO-END TESZTEK

Felhasználó irányából megfogott, teljes folyamatot lefedő tesztek.

MIVEL TESZTELÜNK

Több javascripttesting framework is van, a 3 talán legnépszerűbb:

- **QUnit** - http://qunitjs.com/
- **Jasmine** - http://jasmine.github.io/
- **MochaJs** - https://mochajs.org/

Mi a Mocha-val tesztelünk, ennek viszont nincs beépített assertation libraryje.

MOCHA + CHAI

A **Mocha** mellé én a **Chai**-t fogom venni mint assert, abból is az **expect** formát. Talán ennek a legegyszerűbb a szintaxisa.

```
npm i mocha -g
npm i chai
```

Írjunk tesztet!

MOCHA + CHAI

És egy teszteset:

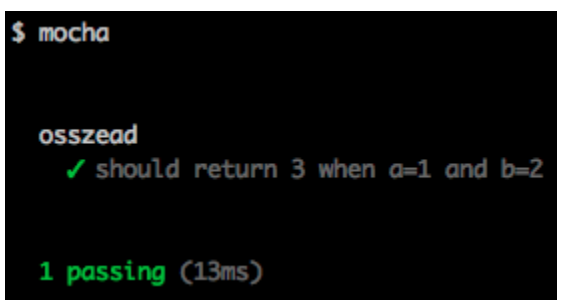
```
var assert = require('assert');
var expect = require('chai').expect;

describe('osszead', function () {
  it('should return 3 when a=1 and b=2', function () {
    var c = osszead(1, 2);
    expect(c).to.be.equal(3);
  });
});
```

MOCHA + CHAI

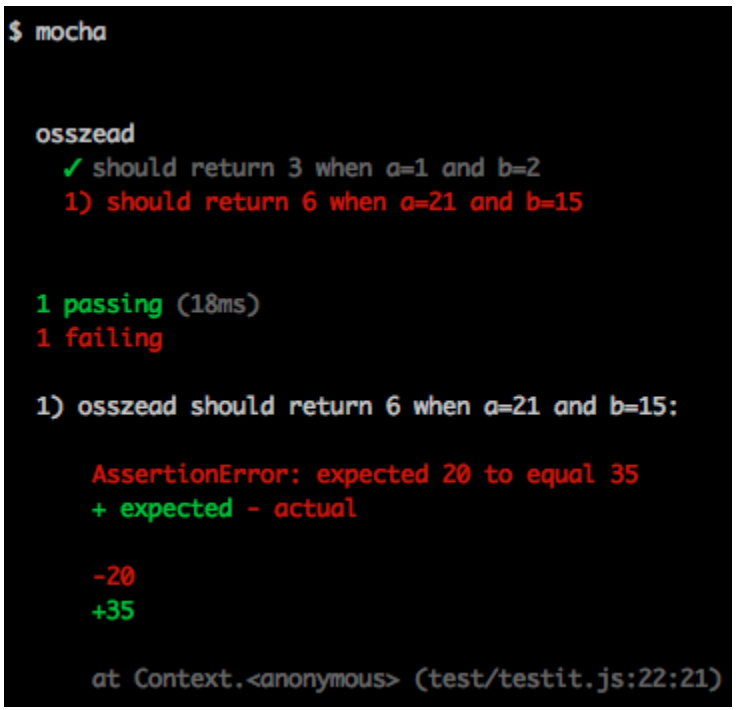
Lefuttatni parancssorban lehet a teszteket:

```
mocha --recursive
```



MOCHA + CHAI

Ha esetleg hiba lenne:



A fájl és sorszám a tesztesetben lévő expect helyét mondja meg, NEM a hiba helyét!

MOCHA + CHAI

Mi van, ha async amit tesztelni kell?

```
function osszead(a, b, cb){
    setTimeout(function(){
        cb(a+b);
    }
}
```

Ez ugye nem ad vissza semmit sem...

MOCHA + CHAI

```
describe('osszead', function (done) {
  it('should return 3 when a=1 and b=2', function () {
    osszead(1, 2, function(c){
        expect(c).to.be.equal(3);
        done();
    });
  });
});
```

Van egy default timeout, 2 sec, ha ez alatt nem hívunk rá a done-ra, akkor jelzi, hogy nem történt meg a callback.

MOCHA

Mocha esetében a szinteket **describe**-al, a tényleges teszteket **it**-el definiáljuk.

A root, suite szinten is lehet extra elő/utó műveleteket definiálni. **NE** tegyük, ha van rá mód, nehezíti az átláthatóságot.

```
beforeEach(function() {
  console.log('before every test in every file');
});
describe('osszead', function (done) {
    before(function() {
      console.log('before all test in this suite, runs once');
    });

    it('should return 3 when a=1 and b=2', function () {
        //...
    });

    after(function() {
      console.log('after all test in this suite, runs once');
    });
});
```

CHAI ASSERT

Az Chai expect része Behavior-driven development (BDD) alapján áll össze: http://chaijs.com/api/bdd/

Az olvashatóság miatt támogatja a "töltelékszavakat": to, be, is, and, has

CHAI ASSERT

.A('NULL') - TÍPUS ELLENŐRZÉS

```
expect({ foo: 'bar' }).to.be.an('object');
expect(null).to.be.a('null');
expect(undefined).to.be.an('undefined');
```

.INCLUDE(...) - STRING RÉSZLET, OBJEKT/LISTA KULCS ELLENŐRZÉS

```
expect([1,2,3]).to.include(2);
expect('foobar').to.contain('foo');
expect({ foo: 'bar', hello: 'universe' }).to.include.keys('foo');
```

CHAI ASSERT

.EXIST - NEM NULL VAGY UNDEFINED

```
expect(foo).to.exist;
expect(bar).to.not.exist;
```

.EQUAL - EGYENLŐ

```
expect({ foo: 'bar' }).to.eql({ foo: 'bar' });
expect([ 1, 2, 3 ]).to.eql([ 1, 2, 3 ]);
```

CHAI ASSERT

.INSTANCEOF - EGY ADOTT OSZTÁLY TÍPUS ELLENŐRZÉSE

```
var Tea = function (name) { this.name = name; }
  , Chai = new Tea('chai');

expect(Chai).to.be.an.instanceof(Tea);
expect([ 1, 2, 3 ]).to.be.instanceof(Array);
```

.PROPERTY - OBJEKTUM KULCS/ÉRTÉK ELLENŐRZÉS

```
var obj = { foo: 'bar' };
expect(obj).to.have.property('foo');
expect(obj).to.have.property('foo', 'bar');
```

CHAI ASSERT

.THROW - HIBA DOBÁSÁNAK ELLENŐRZÉSE

```
var err = new ReferenceError('This is a bad function.');
var fn = function () { throw err; }
expect(fn).to.throw(ReferenceError);
expect(fn).to.throw(Error);
```

.SATISFY(METHOD) - FÜGGVÉNNYEL KIÉRTÉKELÉS

```
expect(1).to.satisfy(function(num) { return num > 0; });
```

HOGYAN IS TESZTELÜNK?

Egyszerű függvényeket könnyű (require + tesztek), de nézzünk meg egy middleware-t:

```
var requireOption = require('../common').requireOption;

module.exports = function (objectrepository) {
  var userModel = requireOption(objectrepository, 'userModel');
  return function (req, res, next) {
    userModel.find({}, function (err, results) {
      if (err) {
        return next(err);
      }
      res.tpl.users = results;
      return next();
    });
  };
};
```

MOCKOLJUNK!

Helyettesítsünk minden külső objektumot és dependenciát egy végletekig leegyszerűsített változattal!

Mit kell mockolni?

objectrepository, req, res.tpl.users, next, userModel, userModel.find

De legalább tudjuk majd tesztelni adatbázis nélkül!!!

MIDDLEWARE TESZT

```
var expect = require('chai').expect;
var getUserListMW = require('../../../middleware/user/getUserList');
describe('getUserList middleware ', function () {
  it('should return users', function (done) {
    var req = {}; var res = { tpl: {} };
    var fakeUserModel = { find: function (some, cb) {
        cb(undefined, ['user1', 'user2'])
      } };
    getUserListMW({
      userModel: fakeUserModel
    })(req, res, function (err) {
      expect(res.tpl.users).to.eql(['user1', 'user2']);
      expect(err).to.eql(undefined);
      done();
    });
  });});
```

Nem is annyira bonyolult ahhoz képest, hogy egy async, modelt használó express alatt lévő middlewaret tesztelünk, mindezen kompenensek nélkül!!!

MOCKOLNI BÁRMIT ÉR

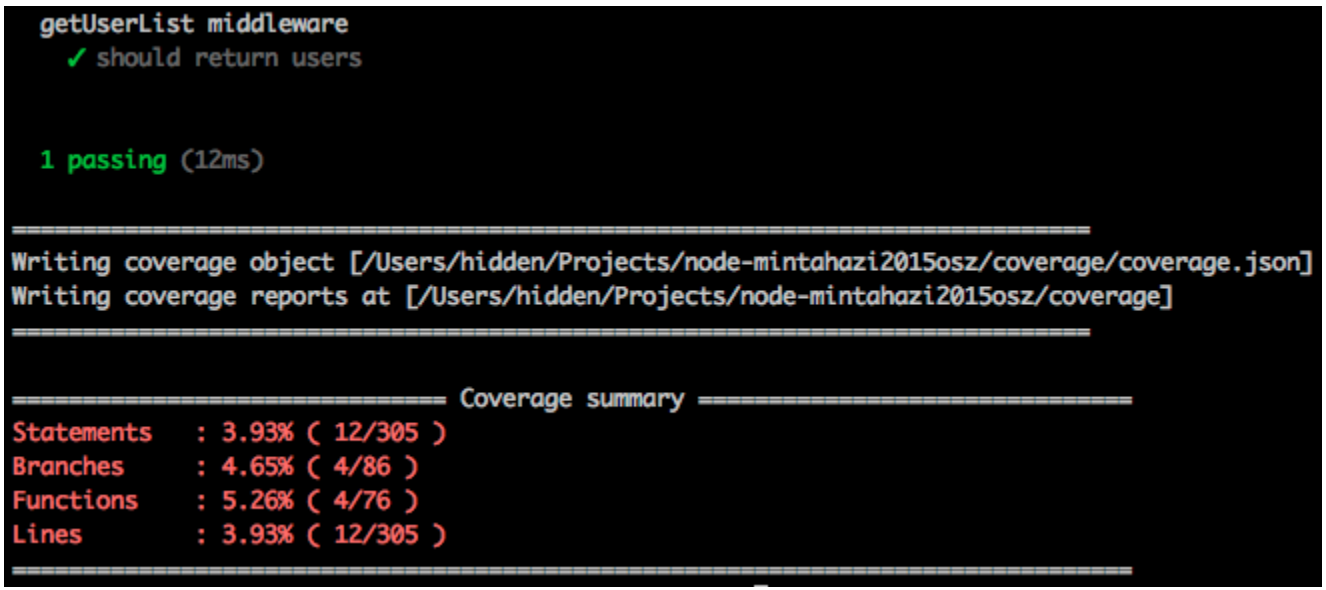
```
var req = {
        body: {
          email: 'user@server.com',
          password: 'asdasd'
        }
};
```

```
var res = {
        send: function(){},
        tpl: {
          valamikomplex: {
              alma: "korte",
              korte: function(cb){
                  return this.alma;
              }
          }
        }
};
```

CODE COVERAGE

Code coverage: mennyi sort, ágat, lehetőséget fed le az össze unit test.

Istanbult fogok használni, gyakorlaton megmutatom, nem szükséges a házik esetében code coveraget vizsgálni.



CODE COVERAGE TELJES KÓDRA

3.93% Statements 12/305 **4.65%** Branches 4/86 **5.26%** Functions 4/76 **3.93%** Lines 12/305

File ▲		Statements		Branches		Functions		Lines	
node-mintahazi2015osz/		0%	0/21	100%	0/0	0%	0/3	0%	0/21
node-mintahazi2015osz/config/		0%	0/3	100%	0/0	100%	0/0	0%	0/3
node-mintahazi2015osz/middleware/		25%	4/16	75%	3/4	16.67%	1/6	25%	4/16
node-mintahazi2015osz/middleware/comment/		0%	0/55	0%	0/16	0%	0/16	0%	0/55
node-mintahazi2015osz/middleware/generic/		0%	0/22	0%	0/6	0%	0/11	0%	0/22
node-mintahazi2015osz/middleware/task/		0%	0/68	0%	0/32	0%	0/18	0%	0/68
node-mintahazi2015osz/middleware/user/		15.69%	8/51	3.57%	1/28	23.08%	3/13	15.69%	8/51
node-mintahazi2015osz/models/		0%	0/12	100%	0/0	100%	0/0	0%	0/12
node-mintahazi2015osz/routes/		0%	0/57	100%	0/0	0%	0/9	0%	0/57

CODE COVERAGE TELJES KÓDRA

Innen látszik, ha már "kilóra" kevés a teszt. Hiányzik egy

teszt arra, ha az err igaz.

A TESZTELÉS MŰVÉSZET

Ezt tényleg csinálni kell, hogy meglegyen a rutin, hogy mit érdemes tesztelni, mi törhet / törik össze, mit hogyan tesztelek (hiba meglétét vs hiba szövegét).

Gyakorlaton mindent tesztelni fogunk, nem meglepő módon.