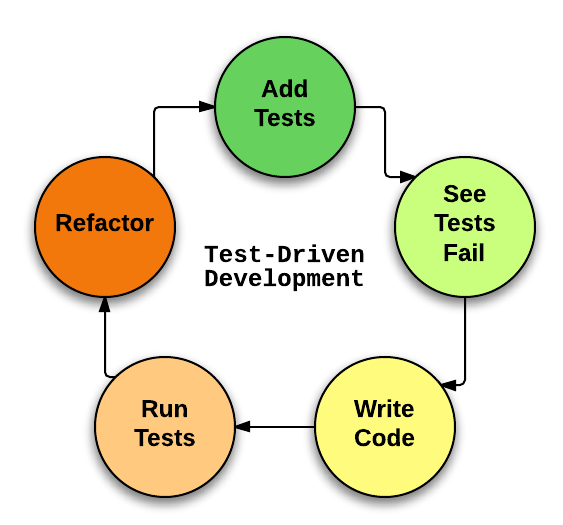
Unit Test

* With testing we understand our project better and improve the development.
* TDD (test-driven development): write the code as a test. Change the code if it fails. Refactor to standardize it and add another test, until the application is tested completely.
* For example: if a URL is called, the webpage returns 200 – successful.

In backend testing, we are not working with the web browser, CSS, or HTML ( front-end testing with the help of the browser)

* BDD (Behavior-Driven Development): Similar to TDD but about the behavior, the implementation is the same, but we want to document the behavior what is the input what is the process and what should be the output.

## Tools

**Mocha** ( a package )

* Js: easy to create folders and understandable output and supports async
* Need a package.json file to run the test
  + Install node.js and run the npm init command line in the project folder
* Install:
  + npm i --save-dev mocha // install in development mode
* Use:
  + in the package.json file in the script: test section write

"test": "mocha"

* + in the project, folder create a folder named test, because mocha needs that
  + for each part of the project we write different tests, for example for the home page, and database, we can put them in one js file and use describe and its methods to write the test.

describe('app', function(){

it('my first test(desc)', function(){

throw Error('my first test')

})

})

* Run: npm test
* Instead of using so many if and else in the body of a test, we can use assertion ( or assert)
  + For example assertion testing in node js:
  + In NodeJS, require() is a built-in function to include external modules that exist in separate files. require() statement basically reads a JavaScript file, executes it, and then proceeds to return the export object. To use different modules in our project.

Const assert = require('assert');

describe('app', function(){

it('my first test(desc)', function(){

assert.equal((1+2), 2)

})

})

* By node.js => libs.js

const sum1 = (x,y)=>{

return x+y

}

const multiply = (x,y)=>x\*y

module.exports = {

sum1,

multiply

}

const assert = require('assert')

const calc = require('./../app/libs')

describe('app', function(){

it('my first test(desc)', function(){

// assert.equal(calc.sum1(1,2), 2)

assert.strictEqual(calc.sum1(1,1), '2')

})

})

**Mocha sidebar**

* VS code extension
* Need a package.json file to run the test

**Asynch test**

* Asynch and await in echma8
* To run a test when there is a function which takes time
* When we want to compare arrays or objects we can use deepEqual function of node js
* In libs.js:

const delayedFilterWithCallback = function(array, filter, callback){

setTimeout(() => {

callback(array.filter(filter))

}, 1000);

}

* We can use done in it, and run it as a callback function when we want. In test.js:

describe('asynch', function(){

it('asynch', function(done){

array = [1,2,10,22,55,70]

filter = (item)=> item % 2 == 0

libs.delayedFilterWithCallback(array, filter, (result)=>{

assert.deepEqual(result, [2,10,22,70])

done()

})

})

})

Promis and await:

const delayedFilterwithPromise = function(array , filter , callback) {

**return** new Promise((resolve , reject) => {

setTimeout(() => {

**resolve**(array.filter(filter));

}, 1000);

})

}

it('promise : eventually returns the results' , () => {

let input = [1,2,3,4,5,6,7,8,9];

let filter = (item) => item % 2 == 0;

return libs.delayedFilterwithPromise(input , filter).**then**(result => {

assert.deepEqual(result , [2,4,6,8])

})

})

it('async : eventually returns the results' , **async** () => {

let input = [1,2,3,4,5,6,7,8,9];

let filter = (item) => item % 2 == 1;

let result = **await** libs.delayedFilterwithPromise(input , filter);

assert.deepEqual(result , [1,3,5,7,9]);

})

**TDD:**

* It is done by [assert](https://nodejs.org/api/assert.html)
* Other Assert Libraries:
  + Better-assert => github/tj/better-assert
  + Chai [assertion](https://www.chaijs.com/api/assert/) library => more liked for tdd and bdd
    - Npm i --save-dev chai

const assert = require('chai').assert

**BDD:**

* To write it like a sentence
* It is done by expect/should
* Except/should libraries:
  + [Should](https://github.com/shouldjs/should.js)
  + [expect](https://github.com/automattic/expect.js)
  + Chai
  + <https://unexpected.js.org/>
* Chai
  + Using methods like which, be, been, have. They don't have much affect on writing the tests, but are used for understanding the test.
  + Not, deep, … these methods have affects.

const expect = require('chai').expect

describe('cal test', function(){

it('50\*0 != 50', function(){

expect(libs.mul(50,0)).to.not.equal(50)

})

})

## Restful API:

To understand unit test better we write an api for database, with a couple of routes:

Nodejs api with express js and mongoDB:

* npm init
* npm i -D mocha chai // d dev dependency
* npm i --save express mongoose body-parser config mongoose-paginate// bodyparser for sending json info to some urls
* create a file in root folder => server.js

const express = require('express')

const app = express(); // construct method

const mongoose = require('mongoose')

const bodyParser = require('body-parser') //control browser body

const port = 3000// port for db

let config = require('config') // search the package dependency

* Create a folder by the name config and a file in it with default.json, dev.json, test.json. With these we can have some option in every mode.

mongoose.connect('')//url to db

* In test.json. 27017 mongoDB a very good db. For test we use a another db, so we don't touch the original db.

{

"DBHost" : "mogodb://localhost:27017/dra-test"

}

* And in package.json: SET NODE\_ENV = dev to change the config
* Use the && after dev without space because sometimes it gives the error that thinks the space if part of dev

"scripts": {

"start": "SET NODE\_ENV = dev&& node server.js",

"test": "mocha"

},

* In server.js

//set some middlewares for the app

app.use(bodyParser.json()) // returen info as json, default setting in bodyParser

app.use(bodyParser.urlencoded({extended: true}))

app.use(bodyParser.text())

app.use(bodyParser.json({type: 'application/json'}))

app.listen(port, ()=>console.log(`Listening on port ${port}`))

module.exports ={

app

}

* In root folder create a folder named app and three folder inside it: controllers, models, routes
* In model => a file article.js

const mongoose = require('mongoose')

const schema = mongoose.Schema

const mongoosePaginate = require('mongoose-paginate')

const articleSchema = schema({

title: { type: String, required: true },

author: { type: String, required: true },

body: { type: String, required: true },

tags: { type: String, required: true },

veiewCount: { type: String, required: true }

}, {timestamps: true })

articleSchema.plugin(mongoPaginate)

module.exports = mongoose.model('article', articleSchema)

* We want to write a restful api in some urls that takes all the articles, delete some, saves some, and updates ( all four operations show create delete update).
* So first we write a test for it, to have a better understanding of what we have to do. And even if we change some part we can test multiple parts with one test.
* In test folder create article\_db\_test.js file:

const mongoose = require('mongoose')

const Article = require('./../app/models/article')

**const should = require('chai').should()**

//to change the package.json node\_env

process.env.NODE\_ENV = 'test'

const config = require('config')

//where to put these?

mongoose.Promise = global.Promise // use nodejs promise

mongoose.connect(config.DBHost, { usenewURLParser: true})

describe('database tests', ()=>{

// done using call back functions

it('check connection' , (done) => {

mongoose.connection

.once('open' , () => done())

.on('error' , (err) => {

console.log(err);

});

});

})

* To save
* Another method for BDD
  + const should = require('chai').should, this way since testing is reading the files, we can directly write should for article. Nodejs can use it in its programs.

it('save article', async ()=>{

let body = {

title : 'article one',

author : 'alma zia',

body : 'this is article one',

tags : 'article1, article2'

}

let article = new Article(body);

article = await article.save()

//bdd test

should.exist(article);

// what type inside a()

article.should.be.a('object')

article.should.have.property('title')

article.should.have.property('author')

article.should.have.property('body')

article.should.have.property('tags')

})

* To get all the articles

it('find all articles', async ()=>{

//returns an array

let articles = await Article.find({});

//bdd test

should.exist(articles);

articles.should.be.a('array')

articles.length.should.be.eq(1)

})

* To get one article

it('find an article', async ()=>{

let article = await Article.findOne({title: 'article one'});

article.length.should.be.deep.eq(1)

article.body.should.be.eq(articleData.body)

})

* To update

it('update an articles', async ()=>{

let article = await Article.findOne({title: articleData.title});

article.set({

body: 'this is the body of article one'

})

let result = await article.save()

result.should.be.a('object')

result.body.should.be.eq('this is the body of article one')

})

* To delete

it('remove all articles', async ()=>{

let result = await Article.deleteMany({});

result.should.have.property('acknowledged')

result.acknowledged.should.be.eq(true)

})

* To close the connection:

it('close the db connection', ()=>{

mongoose.connection.close()

})

## Mocha Hooks:

* To do something in some parts the test. Usually are in describes, and are related to a specific describe.
  + Before // before all the tests of the describe do something
  + After
  + beforeEach // before each test is executed do something
  + afterEach

before(()=>{

mongoose.Promise = global.Promise // use nodejs promise

mongoose.connect(config.DBHost, { usenewURLParser: true})

})

after('close db connection', ()=>{

mongoose.connection.close()

})

## Completing the API:

* In the controller folder create articles.js
* In the routes create index.js
* In server.js, add

const routes = require('./app/routes/index.js')

app.use(routes)