



ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Node.js - Unit Testing - Docker

Μ. Καραμπάτσης



Unit Testing

Unit testing:

- είναι ένας από τους πιο σημαντικούς τύπους αυτοματοποιημένων δοκιμών.
- επικεντρώνεται στην εξέταση των μικρότερων τμημάτων του κώδικα - των λεγόμενων μονάδων - απομονώνοντας αυτόν τον κώδικα
- εφόσον οι δοκιμές μονάδων χρησιμοποιούν μεμονωμένες μονάδες, αυτό σημαίνει ότι το υπό δοκιμή σύστημα δεν μπορεί να αλληλεπιδράσει με εξωτερικές εξαρτήσεις, όπως βάσεις δεδομένων, σύστημα αρχείων ή υπηρεσίες HTTP.
- Για εξέταση μονάδων που αλληλεπιδρούν εξωτερικές εξαρτήσεις, όπως βάσεις δεδομένων, σύστημα αρχείων ή υπηρεσίες HTTP χρησιμοποιούνται και βοηθητικές βιβλιοθήκες.



Unit Testing

Unit testing σε μονάδες κώδικα:

- οι δοκιμές μονάδων είναι συνήθως εύκολο να γραφτούν
- μπορούν να εκτελεστούν χωρίς σχεδόν καμία διαμόρφωση, καθώς συχνά γίνονται μόνο από μία κλήση συνάρτησης.
- είναι εύκολο να συλλεχθούν και να εμφανιστούν αποτελέσματα από δοκιμές μονάδων.
- είναι σημαντικό να γράφουμε δοκιμές μονάδας με τρόπο που να δοκιμάζει όλα τα πιθανά αποτελέσματα ενός αποσπάσματος κώδικα.
- σε κάθε δοκιμή μονάδας, η επιστρεφόμενη τιμή της μονάδας πρέπει να ισούται με την αναμενόμενη τιμή. Εάν δεν συμβαίνει αυτό, τότε η δοκιμή μας θα αποτύχει.



Unit Testing

Unit testing σε api:

- οι δοκιμές σε api συνήθως απαιτούν κάποια αρχική παραμετροποίηση του εργαλείου ελέγχου.
- είναι πιο δύσκολο να διαμορφωθούν, συχνά γίνονται μόνο από μία κλήση συνάρτησης.
- είναι εύκολο να συλλεχθούν και να εμφανιστούν αποτελέσματα.
- είναι σημαντικό να γράφουμε δοκιμές για api με τρόπο που να δοκιμάζει όλα τα πιθανά αποτελέσματα ενός αποσπάσματος κώδικα.
- σε κάθε δοκιμή, η επιστρεφόμενη τιμή πρέπει να ισούται με την αναμενόμενη τιμή. Εάν δεν συμβαίνει αυτό, τότε η δοκιμή μας θα αποτύχει.



Ενδεικτικά εργαλεία Unit testing

1. Jest (<https://jestjs.io/>)
2. Mocha
3. Storybook
4. Jasmine
5. Cypress
6. Puppeteer
7. Testing Library
8. WebdriverIO
9. AVA
10. Playwright

Για τους δικού μας ελέγχους θα χρησιμοποιήσουμε το **Jest**
Το Jest αρχικά υλοποιήθηκε από το Facebook για να ελέγχει εφαρμογές που έχουν υλοποιηθεί με το React. Τα τελευταία χρόνια το Jest υιοθετείται και για εφαρμογές που δεν είναι React. Σήμερα, το Jest θεωρείται από πολλούς ως το καλύτερο framework για JavaScript ελέγχους.

Ένα από τα πλεονεκτήματα του Jest είναι ότι μπορεί να χρησιμοποιηθεί χωρίς να απαιτούνται κάποιες ρυθμίσεις.



Γιατί χρειαζόμαστε το Unit Testing

- Μας βοηθά να εγγυηθούμε την ποιότητα του κώδικα μας και της εφαρμογής γενικότερα.
- Το unit testing ελέγχει τις μονάδες του κώδικα μας και το business logic της εφαρμογής μας.
- Τέλος, τα unit testing μας επιτρέπουν να κάνουμε ελέγχους σε αυτό που ονομάζεται "catching regression", δηλαδή στον έλεγχο των αλλαγών που κάνουμε στο κώδικα μιας εφαρμογής. Πολλές φορές οι αλλαγές σε ένα σημείο του κώδικα δημιουργούν προβλήματα σε άλλα μέρη της εφαρμογής. Χωρίς unit testing και αυτοματοποιημένους ελέγχους, θα έπρεπε να ελέγχουμε διεξοδικά την πλήρη εφαρμογή μας με το χέρι μετά από κάθε αλλαγή.



Πλεονεκτήματα του Node.js Unit Testing

Improved Code Quality: τα unit testing αυξάνουν την ποιότητα του κώδικα μας. Μας εγγυώνται το business logic αυτής και ότι δεν πρόκειται να προκύψει απροσδόκητη συμπεριφορά. Τέλος, το unit βοηθά να βρούμε σφάλματα και ελαττώματα στον κώδικα μας.

Early Discovery of Code Bugs: βοηθά στην αναζήτηση σφαλμάτων νωρίτερα στον κύκλο ζωής ανάπτυξης του λογισμικού και κατά συνέπεια στην αντιμετώπιση αυτών.

Validation of Your Design and Code Structure:

Τα unit testing αποτελούν μια δεύτερη ευκαιρία για επιβεβαιώσουμε την αρχιτεκτονική και τη δομή του κώδικα μας. Κατά συνέπεια μας βοηθά να αποφεύγουμε το σχεδιασμό πολύπλοκων μονάδων που είναι δύσκολο να δοκιμαστούν. Τέλος, το unit testing μας επιτρέπει να βρούμε παράλογες ροές στο κώδικα που πρέπει να επιλυθούν.



Πλεονεκτήματα του Node.js Unit Testing

Improved Customer Satisfaction

Το unit testing συμβάλλει στη συνολική ποιότητα του κώδικά μας κατά συνέπεια και στην ικανοποίηση των πελατών.

Generate Code Coverage Reports

Τα code Coverage Reports εξετάζουν κατά πόσο τα unit testing που ελέγχουν/εξετάζουν όλο το κώδικα μας. Αν για παράδειγμα σε μια διαδικασία έχουμε δύο συνθήκες ελέγχου θα πρέπει να είμαστε σίγουροι ότι τα unit testing περιέχουν ελέγχους και για τις δύο συνθήκες.



Node.js - Jest και Unit Test

1. Με command line δημιουργήστε φάκελο με όνομα jestApp: `mkdir JestApp`
2. Πηγαίνουμε στο νέο φάκελο και αρχικοποιούμε τη node εφαρμογή

```
cd JestApp  
npm init --y
```

3. Στο φάκελο jestApp υπάρχει πλέον το αρχείο **package.json**.
4. Δημιουργήστε ένα αρχείο **index.js** το οποίο θα περιέχει το κώδικα του προγράμματος μας. Επίσης, δημιουργήστε ένα φάκελο **test** στον οποίο θα αποθηκεύονται όλα τα unit tests
5. Στο φάκελο test δημιουργούμε αρχείο **calculator.test.js**



Node.js - Jest και Unit Test

6. Εγκαθιστούμε το JEST: `npm install --save-dev jest`

7. Στο **package.json** κάντε τη παρακάτω αλλαγή στο πεδίο **scripts**:

```
"scripts": {  
  "test": "jest"  
},
```

8. Για να μας εμφανίσει το **coverage report** προσθέτουμε στο **package.json** την παρακάτω παράμετρο **collectCoverage**. Για να αγνοήσει κάποιους φακέλους την **coveragePathIgnorePatterns**

```
{  
  ...  
  "scripts": {  
    "test": "jest"  
  },  
  "jest": {  
    "collectCoverage": true,  
    "coveragePathIgnorePatterns": [  
      "/node_modules/"  
    ]  
  },  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "jest": "^29.7.0"  
  }  
}
```



Node.js - Jest και Unit Test

9. Στο αρχείο **index.js** προσθέστε το παρακάτω κώδικα:

```
const mathOperations = {  
  sum: function(a,b) {  
    return a + b;  
  },  
  
  diff: function(a,b) {  
    return a - b;  
  },  
  product: function(a,b) {  
    return a * b;  
  },  
  divide: function(a,b) {  
    return a / b;  
  }  
}  
module.exports = mathOperations
```



Node.js - Jest και Unit Test

10. Στο αρχείο **calculator.test.js** προσθέστε τα test cases:

```
const mathOperations = require('../index');

describe("Calculator Tests", () => {
  test("Addition of 2 numbers", () => {
    // arrange and act
    let result = mathOperations.sum(1,2)
    // assert
    expect(result).toBe(3);
  });

  test("Subtraction of 2 numbers", () => {
    // arrange and act
    let result = mathOperations.diff(10,2)
    // assert
    expect(result).toBe(8);
  });
});
```

```
test("Multiplication of 2 numbers", () => {
  // arrange and act
  let result = mathOperations.product(2,8)

  // assert
  expect(result).toBe(16);
});

test("Division of 2 numbers", () => {
  // arrange and act
  let result = mathOperations.divide(24,8)

  // assert
  expect(result).toBe(3);
});
});
```



Δομή του Jest unit testing

1. **describe(name, fn):** δημιουργεί ένα μπλοκ που ομαδοποιεί πολλά σχετικά test.

(<https://jestjs.io/docs/api#describename-fn>)

2. **test(name, fn, timeout):** δημιουργεί ένα test

<https://jestjs.io/docs/api#testname-fn-timeout>

```
1. describe("Calculator Tests", () => {
2.   test("Addition of 2 numbers", () => {
3.     // arrange and act
4.     let result = mathOperations.sum(1,2)
5.     // assert
6.     expect(result).toBe(3);
7.   });
8. })
```

3. **expect(value):** επιτρέπει να ελέγχονται τα αποτελέσματα των τιμών από τα tests. Τα αποτελέσματα θα πρέπει ικανοποιούν διάφορους κανόνες τα matchers.

<https://jestjs.io/docs/expect>



Jest Run

Για να τρέξουμε τους σχετικούς ελέγχους με τα test, πληκτρολογούμε:

```
> npm run test
```

```
npm run test
> jestapp@1.0.0 test
> jest

PASS test/calculator.test.js
  Calculator Tests
    ✓ Addition of 2 numbers (6 ms)
    ✓ Subtraction of 2 numbers (1 ms)
    ✓ Multiplication of 2 numbers (1 ms)
    ✓ Division of 2 numbers (1 ms)

-----|-----|-----|-----|-----|-----
File    | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files|    100   |    100   |    100   |    100   |
index.js|    100   |    100   |    100   |    100   |
-----|-----|-----|-----|-----|-----
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.064 s, estimated 6 s
Ran all test suites.
```

Το Jest αναγνωρίζει τα test αρχεία με τρεις τρόπους:

1. Ψάχνει αρχεία που έχουν κατάληξη `.test.js`
2. Ψάχνει αρχεία που έχουν κατάληξη `.spec.js`
3. Όλα τα αρχεία που βρίσκονται σε φάκελο με όνομα **tests**.



Jest Run - Περίπτωση λάθους

Αλλάξτε το αρχείο **calculator.test.js** και συγκεκριμένα για τους παρακάτω ελέγχους:

```
test("Subtraction of 2 numbers", () => {  
  // arrange and act  
  var result = mathOperations.diff(10,2)  
  // assert  
  expect(result).toBe(21); //updated to fail  
});  
  
test("Multiplication of 2 numbers", () => {  
  // arrange and act  
  var result = mathOperations.product(2,8)  
  // assert  
  expect(result).toBe(10); //updated to fail  
});
```



Jest Run - Περίπτωση λάθους

Για να τρέξουμε τους σχετικούς ελέγχους με τα test, πληκτρολογούμε:

```
> npm run test
```

```
FAIL test/calculator.test.js
Calculator Tests
  ✓ Addition of 2 numbers (15 ms)
  ✗ Subtraction of 2 numbers (5 ms)
  ✗ Multiplication of 2 numbers
  ✓ Division of 2 numbers

• Calculator Tests > Subtraction of 2 numbers

expect(received).toBe(expected) // Object.is equality

Expected: 21
Received: 8

   16 |     // assert
   17 |     // expect(result).toBe(8);
>  18 |     expect(result).toBe(21); //update for fail
      |                        ^
   19 |   });
   20 |
   21 |   test("Multiplication of 2 numbers", () => {
      |
      at Object.toBe (test/calculator.test.js:18:20)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
index.js	100	100	100	100	

Test Suites: 1 failed, 1 total
Tests: 2 failed, 2 passed, 4 total
Snapshots: 0 total
Time: 1.313 s
Ran all test suites.



Node.js, Express.js, Mongoose, Jest, SuperTest

Για τον έλεγχο της εφαρμογής productsApp απαιτείται η εγκατάσταση τριών νέων πακέτων: jest, supertest και του cross-env

```
> npm i jest supertest cross-env
```

- **jest** : framework για το έλεγχο κώδικα που έχει γραφτεί σε JavaScript.
- **supertest**: με το πακέτο Supertest μπορούμε να ελέγξουμε τα endpoints και τα routes σε έναν HTTP server.
- **cross-env**: Χρησιμοποιείται όταν θέλουμε να ορίσουμε environmental variables μέσα σε μια εντολή.



Jest, SuperTest για API

Αρχικά κάνουμε τις παρακάτω αλλαγές στο package.json

```
"scripts": {  
  "test": "cross-env NODE_ENV=test jest --testTimeout=5000",  
  "start": "node app.js",  
  "dev": "nodemon app.js"  
},
```

Σε αυτήν την περίπτωση, χρησιμοποιούμε cross-env για να ορίσουμε ως environment variables, αρχικά το jest για την εκτέλεση των test και τέλος το testTimeout. Το οποίο έχει οριστεί σε 5000 επειδή ορισμένα αιτήματα ενδέχεται να χρειαστούν λίγο χρόνο για να ολοκληρωθούν.



Jest, SuperTest για API

Στη συνέχεια δημιουργήστε ένα φάκελο **tests** και μέσα στο φάκελο το αρχείο **user.test.js**.

Το Jest αναζητά τα αρχεία των test στο φάκελο tests που βρίσκεται στο root της εφαρμογής μας. Η αναζήτηση αυτή πραγματοποιείτε όταν τρέχει η εντολή:

```
> npm run test
```

Στη συνέχεια, στο αρχείο **user.test.js** εισάγουμε τα πακέτα supertest και mongoose.

```
const mongoose = require("mongoose");  
const request = require("supertest");
```



Jest, SuperTest για API

Επίσης, εισάγουμε το πακέτο **dotenv** προκειμένου να διαβάσουμε τις `environment variables` και το **app.js** ή **index.js**, το αρχείο δηλαδή που ξεκινά την εφαρμογή μας.

Τέλος, θα χρειαστεί να συνδέσετε και να αποσυνδέσετε τη βάση δεδομένων πριν και μετά από κάθε `test` (επειδή δεν χρειαζόμαστε τη βάση δεδομένων μόλις ολοκληρωθεί το `test`).

beforeEach(fn, timeout): Εκτελεί μια συνάρτηση πριν εκτελεστεί ένα `test`.

```
const mongoose = require("mongoose");
const request = require("supertest");
const app = require("../app");

require("dotenv").config();

/* Connecting to the database before each test. */
beforeEach(async () => {
  await mongoose.connect(process.env.MONGODB_URI);
});

/* Closing database connection after each test. */
afterEach(async () => {
  await mongoose.connection.close();
});
```

afterEach(fn, timeout): Εκτελεί μια συνάρτηση αφού εκτελεστεί ένα `test`.



Jest, SuperTest για API

1. **describe(name, fn)**: δημιουργεί ένα μπλοκ που ομαδοποιεί πολλά σχετικά test.

(<https://jestjs.io/docs/api#describename-fn>)

2. Το **it** είναι ένα alias του **test**.

test(name, fn, timeout): δημιουργεί ένα test

<https://jestjs.io/docs/api#testname-fn-timeout>

```
1. describe("GET /api/users", () => {
2.   it("should return all users", async () => {
3.     const res = await request(app).get("/api/users");
4.     expect(res.statusCode).toBe(200);
5.     expect(res.body.length).toBeGreaterThan(0);
6.   });
7. });
```

3. **expect(value)**: επιτρέπει να ελέγχονται τα αποτελέσματα των τιμών από τα tests. Τα αποτελέσματα θα πρέπει ικανοποιούν διάφορους κανόνες τα matchers.

<https://jestjs.io/docs/expect>



Jest, SuperTest για API

Για να τρέξουν τα test πρέπει να γίνει μια βασική αλλαγή:

1. Δημιουργήστε αρχείο **server.js** και προσθέστε το παρακάτω κώδικα

```
const app = require("./app");
const port = 3000

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

2. Αφαιρέστε από το app.js ή index.js το:

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

Το Jest τρέχει ξεκινά έναν δικό του server όταν κάνει τους σχετικούς ελέγχους. Κατά συνέπεια δεν θέλουμε να ξεκινά τον server της εφαρμογής μας όταν κάνει τους ελέγχους. Διαφορετικά θα προσπθήσει να τρέξει δύο Server στην ίδια πόρτα

Τροποποιήστε ανάλογα και το package.json

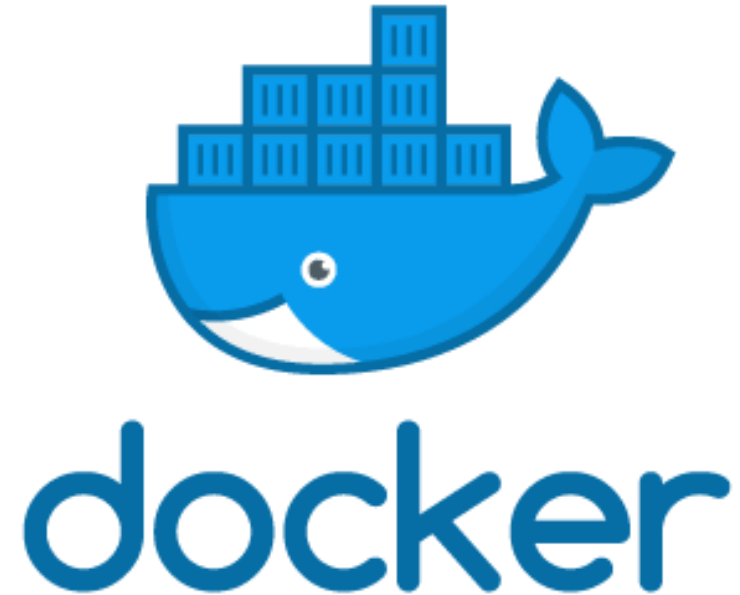
```
"scripts": {
  "start": "node server.js",
  "test": "cross-env NODE_ENV=test jest --testTimeout=10000",
  "dev": "nodemon server.js"
},
```



Node.js - Docker

Το **Docker** είναι μια πλατφόρμα container ανοιχτού κώδικα που δίνει τη δυνατότητα στους προγραμματιστές να δημιουργούν, να αναπτύσσουν, να εκτελούν και να διαχειρίζονται εφαρμογές σε διαφορετικά περιβάλλοντα πολύ αποτελεσματικά.

Χρησιμοποιώντας το Docker μπορούμε να καθορίσουμε το περιβάλλον στο οποίο εκτελείται η εκάστοτε εφαρμογή, συμπεριλαμβανομένων των απαιτούμενων εκδόσεων των βιβλιοθηκών και των εξαρτήσεων.





Node.js - Docker

Docker image: αποτελεί ένα αρχείο μόνο για ανάγνωση (read-only) που περιλαμβάνει τις απαραίτητες οδηγίες για τη δημιουργία ενός container

Ένα container που εκτελείται χρησιμοποιεί ένα απομονωμένο σύστημα αρχείων. Αυτό το απομονωμένο σύστημα αρχείων παρέχεται από ένα image και αυτό το image πρέπει να περιέχει όλα όσα χρειάζονται για την εκτέλεση μιας εφαρμογής - όλες τις εξαρτήσεις, configurations, scripts, binaries αρχεία κ.λπ. Το image περιέχει επίσης άλλες διαμορφώσεις για το container, όπως μεταβλητές περιβάλλοντος, μια προεπιλεγμένη εντολή για εκτέλεση και άλλα μεταδεδομένα.



Node.js - Docker

Docker container: είναι ένα sandbox που εκτελείται σε έναν υπολογιστή και το sandbox αυτό είναι απομονωμένο από όλες τις άλλες διεργασίες που εκτελούνται σε αυτόν τον υπολογιστή, με λίγα λόγια:

- επιτρέπει το "τρέξιμο" ενός image, εκτελώντας εντολές start, stop, move, ή delete μέσω του Docker API ή CLI.
- Μπορεί να τρέξει σε τοποικά μηχανή, στο cloud ή σε virtual machines.
- Είναι μεταφέρσιμο και μπορεί να τρέξει σε οποιοδήποτε OS.
- Είναι απομονωμένο από άλλα containers και τρέχει το δικό του λογισμικό, configurations κλπ.



Docker εγκατάσταση

Στην ιστοσελίδα <https://docs.docker.com/get-docker/> επιλέγουμε το Docker Desktop για το λειτουργικό που έχουμε:



Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.



Docker Desktop for Windows

A native Windows application which delivers all Docker tools to your Windows computer.



Docker Desktop for Linux

A native Linux application which delivers all Docker tools to your Linux computer.



Docker εγκατάσταση


Για την περίπτωση μας επιλέγουμε Windows και κατεβάζουμε το σχετικό αρχείο:

Install Docker Desktop on Windows

This page contains the download URL, information about system requirements, and instructions on how to install Docker Desktop for Windows.

[Docker Desktop for Windows](#)

Στη συνέχεια τρέχουμε το αρχείο

 Docker Desktop Installer.exe

Όταν σας ζητηθεί, βεβαιωθείτε ότι η επιλογή **Use WSL 2 instead of Hyper-V** είναι επιλεγμένη.



Dockerizing Node.js εφαρμογή

1. Στο φάκελο productsApp της εφαρμογής μας δημιουργούμε ένα αρχείο με όνομα Dockerfile:

2. Στο αρχείο πληκτρολογούμε τα παρακάτω:

```
# version of node to use
FROM node:18
# Directory to save image
WORKDIR /usr/src/app
# Install app dependencies
COPY package*.json ./
RUN npm install
# Bundle app source
COPY . .
EXPOSE 3000
CMD [ "npm", "run", "start" ]
```

1. **FROM:** δηλώνουμε για το image την έκδοση του node που επιθυμούμε.
2. **WORKDIR:** στη συνέχεια δηλώνουμε το φάκελλο στον οποίο θα υπάρχει η εφαρμογή στο image που θα δημιουργηθεί.
3. **COPY, RUN:** αντιγράφουμε το package.json στο workdir και εγκαθιστούμε τα πακέτα που υπάρχουν δηλωμένα.



Dockerizing Node.js εφαρμογή

```
# version of node to use
FROM node:18
# Directory to save image
WORKDIR /usr/src/app
# Install app dependencies
COPY package*.json ./
RUN npm install
# Bundle app source
COPY . .
EXPOSE 3000
CMD [ "npm", "run", "start" ]
```

1. **COPY** . .: Αντιγράφουμε τα αρχεία της εφαρμογής μας στο workdir.
2. **EXPOSE**: δηλώνουμε τη πόρτα στην οποία τρέχει η εφαρμογή μας.
3. **CMD**: δηλώνουμε την εντολή, σύμφωνα με το package.json, που ξεκινά η εφαρμογή μας.

Τέλος, δημιουργούμε το αρχείο **.dockerignore** προκειμένου το Docker να αγνοήσει κάποιους φακέλους ή αρχεία.

```
node_modules
npm-debug.log
```



Δημιουργία και τρέξιμο του image

Μεταβείτε στον κατάλογο που έχει το Dockerfile και εκτελέστε την ακόλουθη εντολή για να δημιουργήσετε το docker image.

```
> docker build . -t <your username>/node-products-app
```

Η σημαία -t επιτρέπει την προσθήκη μιας ετικέτας στο image, ώστε να είναι πιο εύκολη η αναζήτηση του

Τα images που έχουμε μπορούμε να τα δούμε με δύο τρόπους:

1. Με την εντολή

```
> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
markos/node-products-app	latest	1dc5d51d2d82	2 minutes ago	1.22GB

2. Με το docker desktop που έχετε εγκαταστήσει στον υπολογιστή σας.



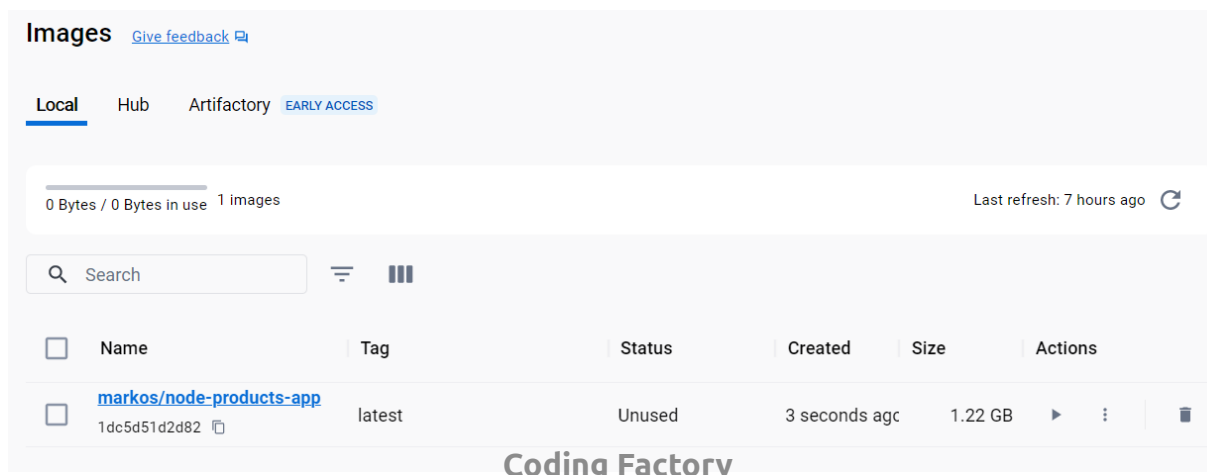
Δημιουργία και τρέξιμο του image

Για να τρέξετε το image που επιθυμείτε έχετε δύο τρόπους:

1. Με χρήση της εντολής: `docker run -p 49160:8080 -d <your username>/node-web-app`

Η επιλογή -p κάνει redirect την public πόρτα σε μια local στο container. Η επιλογή -d τρέχει το image σε detached mode

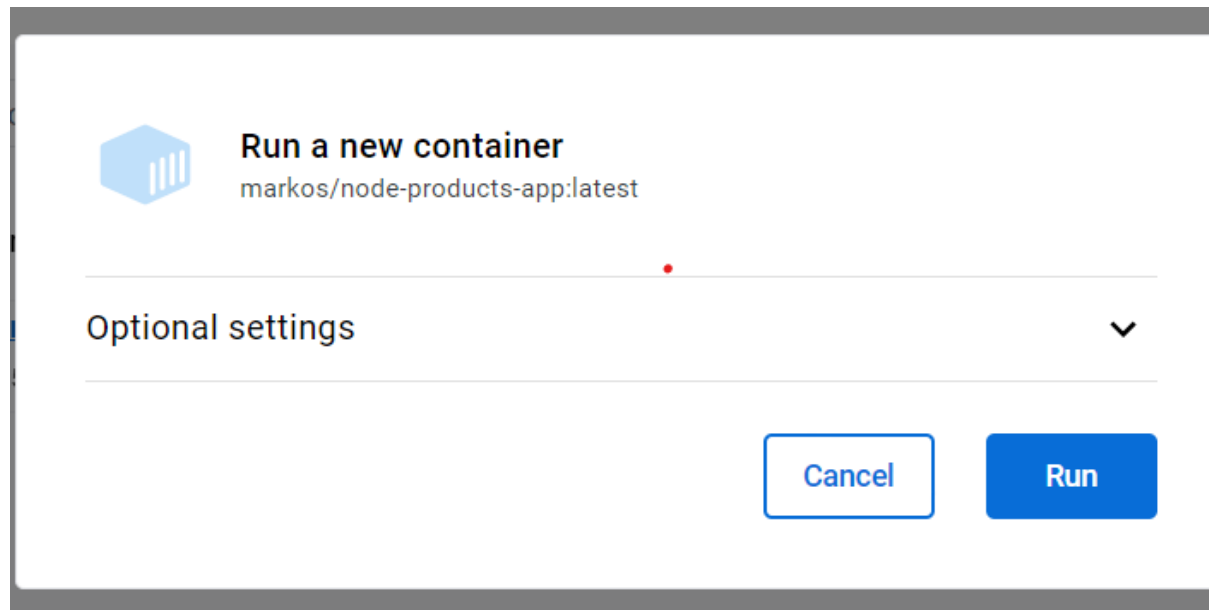
2. Με το docker desktop που έχετε εγκαταστήσει στον υπολογιστή σας και τις επιλογές Actions.





Δημιουργία και τρέξιμο του image


1. Στο docker desktop από την επιλογή **Actions** επιλέξτε **Run**.
2. Στο παράθυρο που εμφανίζεται πατήστε στο Dropdown από το optional settings






Δημιουργία και τρέξιμο του image

Στην επιλογή Ports πληκτρολογήστε 3000 και στη συνέχεια **Run**.


 **Run a new container**
markos/node-products-app:latest

Optional settings 


Container name

A random name is generated if you do not provide one.


Ports
Enter "0" to assign randomly generated host ports.


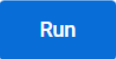
Host port  :3000/tcp

Volumes

Host path Container path 

Environment variables

Variable Value 


 



Δημιουργία και τρέξιμο του image


Η επιλογή Logs του Docker desktop


<



adoring_brattain

[markos/node-products-app:latest](#)

4e52f41e5ca2 

[3000:3000](#) 

Logs

Inspect

Bind mounts

Exec

Files

Stats

```
2023-09-28 12:42:33
2023-09-28 12:42:33 > productsapp@1.0.0 start
2023-09-28 12:42:33 > node server.js
2023-09-28 12:42:33
2023-09-28 12:42:33 Example app listening on port 3000
2023-09-28 12:42:35 Connection to MongoDB established
```

Έλεγχος από postman ότι η εφαρμογή τρέχει.

GET localhost:3000/api/users

Send

Params

Auth

Headers (7)

Body

Pre-req.

Tests


Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

200 OK 289 ms 3.5 KB  Save as Example ...



Pretty

Raw

Preview

Visualize

JSON

```
1 {}
2 "status": true,
3 "data": [
4   {
5     "_id": "64c2aa29cfc8e23f3adb433e",
6     "username": "user4",
```

Coding Factory

34



Άσκηση

1. Δημιουργήστε τους Jest ελέγχους για τον controller products της εφαρμογής Node.js με MongoDB.
2. Ενημερώστε το github με τις νέες αλλαγές.