**Automated Testing**Automated Testing is the practice of writing code to test the code then run those tests in automated test. In contrast to manual testing where we manually click and test through the app

**Code = Production Code + Test Code**

**Types of Automated Tests  
Unit Tests  
Integration Tests  
End To End Tests**

**Unit Tests: -** Test a unit of application without its external dependencies such as files, databases etc. very fast to execute

**Integration Tests: -** Test the application with its external dependencies. It tests the integration of your code with external dependencies. Such as integration with database. Takes Longer to execute

**End To End Tests: -** Drives an application through its UI. Check the User Interface by mimicking User interaction. Very slow

**Unit Testing With Jest**

To Write Test with Jest You need to create a file with name of your file.test.js for example if your code file is code.js then the test file should be code.test.js

Import the code file in test file

To run Jest Test Go to Package.json file and add a test script

  "scripts": {

    "test": "jest"

  },

Example of testing a sum function

sum.js file

function sum(a, b) {

  return a + b

}

module.exports = sum

sum.test.js file

const sum = require('./sum')

test('add 1+2 equals to 3', () => {

  expect(sum(2, 2)).toBeGreaterThan(3)

  expect(sum(2, 2)).toBeGreaterThanOrEqual(3.5)

  expect(sum(2, 2)).toBeLessThan(5)

  expect(sum(2, 2)).toBeLessThanOrEqual(4.5)

})

To write define a test function that takes a description of the test and and a callback function. In which you call except function and pass the function with input you want to call. This test used expect and different matchers to test the conditions. These chain function are called matchers and they match the output of the function called to your expected output.

For example toBeGreaterThan matches the output 4 should be greater than expected output 3

**Testing Numbers With Jest**

Test cases for one function can be grouped together using describe statement and test statement can also be replace by it statement. For Example

module.exports.absolute = function (number) {

  return number >= 0 ? number : -number

}

describe('absolute', () => {

  it('should return a positive number if input is positive', () => {

    const result = lib.absolute(1)

    expect(result).toBe(1)

  })

  it('should return a positive number if input is negative', () => {

    const result = lib.absolute(-1)

    expect(result).toBe(1)

  })

  it('should return a 0 if input is 0', () => {

    const result = lib.absolute(0)

    expect(result).toBe(0)

  })

})

These following matchers can be used with Numbers

  toBeGreaterThan()

  toBeGreaterThanOrEqual()

  toBeLessThan()

  toBeLessThanOrEqual()  
 ToBe()

**Testing String With Jest**

ToBe() can also be used to match string but it will match exact but if you want to match just a specific substring should be there in string then toMatch(//) can be used that use regex or toContain() can be used

module.exports.greet = function (name) {

  return `Welcome ${name}`

}

describe('greet', () => {

  it('should return a greeting message', () => {

    expect(lib.greet('rishabh')).toMatch(/rishabh/)

  })

})

test checks for if greet message have rishabh in it

**Testing Arrays With Jest**

module.exports.getCurrencies = function () {

  return ['USD', 'AUD', 'EUR']

}

describe('getCurrencies', () => {

  it('should return supported currencies', () => {

    const result = lib.getCurrencies()

    expect(result).toEqual(expect.arrayContaining(['EUR', 'USD', 'AUD']))

  }

Matches returning array should contain these currencies if there are other currencies added the test won’t break

**Testing Objects With Jest**

*// Testing objects*

module.exports.getProduct = function (productId) {

  return { id: productId, price: 10, name: 'Soap' }

}

describe('getProduct', () => {

  it('should return product with given id', () => {

    const result = lib.getProduct(1)

    expect(result).toMatchObject({ id: 1, price: 10 })

  })

})

toBe should not be used when testing objects because it compare memory location of object which is going to going to be different for each object created. toMatchObject or toEqual should be used.

However to equal will throw an error if there are extra properties to the object. So that’s why in above example toMatchObject is used.

**Testing Exceptions With Jest**

*// Testing exceptions*

module.exports.registerUser = function (username) {

  if (!username) throw new Error('Username is required.')

  return { id: new Date().getTime(), username: username }

}

describe('register User', () => {

  it('should throw if username is falsy', () => {

    const args = [null, undefined, NaN, 0, '', false]

    args.forEach((a) => {

      expect(() => {

        lib.registerUser(a)

      }).toThrow()

    })

  })

  it('should return user object if valid user name is passed', () => {

    const result = lib.registerUser('rishabh')

    expect(result).toMatchObject({ username: 'rishabh' })

    expect(result.id).toBeGreaterThan(0)

  })

})

Note: the function that is going to throw exception should not be passed directly to expect like this expect(lib.registerUser(a)).toThrow() it will throw an error it should be passed in a callback like above example.

Testing for two path first that for non valid values. Register function should throw an exception and for valid input passed object should be returned with username and id should be a positive number

**Mocking functions**

When we are testing are code w we might find a code where function calls another function or external dependency. For unit testing we should not worry about these things we just want to test our individual functions not their interaction with others components. So we mock these functions and dependencies to continue testing.

We can manually mock them by replacing their implementation but jest comes with helper functions for mocking node module and functions

**Manually Mocking function example**

module.exports.applyDiscount = function (order) {

  const customer = db.getCustomerSync(order.customerId)

  if (customer.points > 10) order.totalPrice \*= 0.9

}

applyDiscount depends on external db dependency to get customer

describe('applyDiscount', () => {

  it('should apply 10% discount if customer has more than 10 points', () => {

    db.getCustomerSync = function (customerId) {

*// console.log('fake reading customer...')*

      return { id: customerId, points: 20 }

    }

    const order = { customerId: 1, totalPrice: 10 }

    lib.applyDiscount(order)

    expect(order.totalPrice).toBe(9)

  })

})

Here we are manually replacing db.getCustomerSync implementation

**Mocking functions using Jest example**

*// Mock functions*

module.exports.notifyCustomer = function (order) {

  const customer = db.getCustomerSync(order.customerId)

  mail.send(customer.email, 'Your order was placed successfully.')

}

describe('notifyCustomer', () => {

  it('should send and email to the customer', () => {

    db.getCustomerSync = jest.fn().mockReturnValue({ email: 'a' })

    mail.send = jest.fn()

    lib.notifyCustomer({ customerId: 1 })

    expect(mail.send).toHaveBeenCalled()

    expect(mail.send.mock.calls[0][0]).toBe('a')

    expect(mail.send.mock.calls[0][1]).toMatch(/order/)

  })

})

Jest.fn mocks a function and we can chain mockReturnValue to return value

If we want to check if function has been called we can use toHaveBeenCalled method

We can also check if method is called with proper argument using mockFuntionName.mock.calls where first index specify call and second argument positon

And our we are testing our mail mock function should be called with a as first arg and second arg should contain order word in it

**Mocking Node Modules using Jest example**

const axios = require('axios')

const getUsersList = *async* () => {

  const response = await axios.get('https://jsonplaceholder.typicode.com/users')

  return response.data

}

module.exports = getUsersList

const axios = require('axios')

const getUsersList = require('./users')

jest.mock('axios')

*//mocking resolved values*

describe('getUsersList', () => {

  it('should return array of users objects', () => {

    const users = [

      {

        id: 10,

        name: 'Clementina DuBuque',

        username: 'Moriah.Stanton',

        email: 'Rey.Padberg@karina.biz',

      },

    ]

    const response = { data: users }

    axios.get.mockResolvedValue(response)

    getUsersList().then((data) => expect(data).toEqual(users))

  })

We use jest.mock function to mock modules here we are mocking axios.get method to return promised mockResolvedValue returns promise then we are using .then to process promise

**Jest Resolve Methods**

**mockReturnValue:- returns simple value**

**mockResolvedValue:- resolves a promise with value then you attach then or await to get value**

**mockRejectedValue:– reject a promise with value use exception handing to catch the exception thrown by mockRejectedValue**

**Link to read more:** [**https://jestjs.io/docs/en/mock-function-api**](https://jestjs.io/docs/en/mock-function-api%20)

**Mocking ES6 Classes**

const axios = require('axios')

class PostClient {

*async* getById(id) {

    const url = `https://jsonplaceholder.typicode.com/posts/${id}`

    const response = await axios.get(url)

    return response.data

  }

}

module.exports = PostClient

const PostClient = require('./PostClient')

class PostManager {

*async* getPostToManage(id) {

    const postClient = new PostClient()

    const postToManage = await postClient.getById(id).catch((err) => alert(err))

    return postToManage

  }

}

module.exports = PostManager

const PostManager = require('./PostManager')

const PostClient = require('./PostClient')

jest.mock('./PostClient')

describe('PostManager', () => {

  it('should return the post to the given id', *async* () => {

    const expectedResult = {

      userId: 4,

      id: 35,

      title: 'id nihil consequatur molestias animi provident',

      body:

        'nisi error delectus possimus ut eligendi vitae\nplaceat eos harum cupiditate facilis reprehenderit voluptatem beatae\nmodi ducimus quo illum voluptas eligendi\net nobis quia fugit',

    }

    const mockGetById = jest.fn()

    PostClient.prototype.getById = mockGetById

    mockGetById.mockReturnValue(Promise.resolve(expectedResult))

    const postManager = new PostManager()

    const result = await postManager.getPostToManage(35)

    expect(result.id).toBe(35)

  })

})

**Link:** [**https://dev.to/jackcaldwell/mocking-es6-class-methods-with-jest-bd7**](https://dev.to/jackcaldwell/mocking-es6-class-methods-with-jest-bd7)