

دانشگاه شهید بهشتی

دانشکده مهندسی و علوم کامپیوتر

**Automatic Unit Test Generation**

**in JavaScript**

گزارش پروژه آزمون نرم افزار پیشرفته

کارشناسی ارشد نرم افزار

نگارش

بهـــزاد خسروی فر

الهام استدلالی

استاد

**دکتر حقــیقی**

زمستان 1397

**هدف پروژه**

هدف این پروژه تولید خودکار تست واحد نرم افزار با استفاده از زبان جاوا اسکریپت نسخه ES5 در چهارچوب Jest می باشد.بنابراین نیاز به نصب node js در سیستم مورد استفاده می باشد. این گزارش کار شامل چند بخش می باشد، بخش اول، روند انجام پروژه و شرح آن، بخش دوم، چگونه کار می کنیم[[1]](#footnote-1) که الگوریتم کار را شرح خواهیم داد و بخش دوم، چگونه استفاده می کنیم[[2]](#footnote-2) که شرح استفاده از این پروژه را خواهیم گفت

**روند انجام پروژه**

روند انجام پروژه بصورت زیر می باشد که در ادامه با جزییات شرح داده شده است:

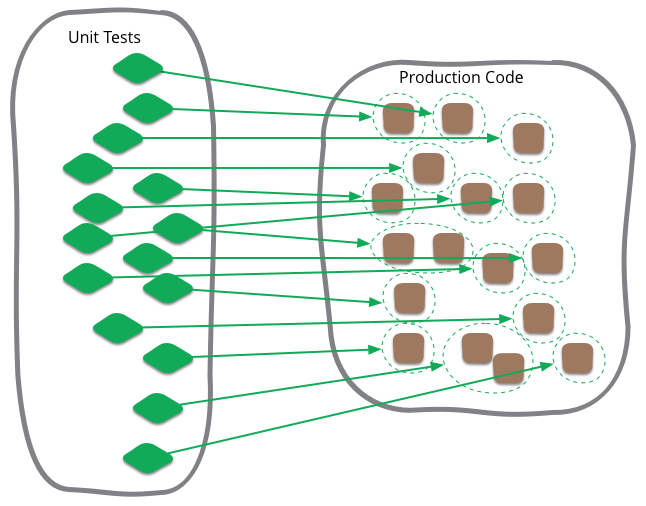
1. بررسی مفاهیم Unit Test.
2. انتخاب زبان برنامه نویسی مناسب.
3. انتخاب چارچوب[[3]](#footnote-3) طراحی تست در آن زبان.
4. پیاده سازی نمونه تست زنی در چارچوب انتخابی بر اساس مستندات کتابخانه آن.
5. انتخاب کتابخانه رسم گراف در آن زبان.
6. رسم چند گراف با داده های تستی.
7. ثبت مراحل انجام کار در سایت GitHub: جهت انجام راحت تر کار گروه پروژه در این لینک نیز بارگزاری شده است. <https://github.com/bezzad/UnitJest>
8. ایجاد نمونه کدهای اولیه برای ورودی برنامه نهایی.
9. طراحی پیش پردازشگر زبان انتخابی.
10. تحلیل برنامه اولیه برای تست کردن کدهای نمونه.
11. درآوردن فلوچارت تحلیل کد.
12. ساده سازی مراحل طرح تا حد ممکن و اولویت دهی آن.
13. شروع پیاده سازی با زبان انتخابی.
14. پیاده سازی CI ساده.
15. مستندسازی و تست نرم افزار تست.
16. تصویر سازی مستند و تحویل نهایی.

**شرح** **روند انجام پروژه**

1. **بررسی مفاهیم Unit Test:**

در [برنامه‌نویسی](https://fa.wikipedia.org/wiki/%D8%A8%D8%B1%D9%86%D8%A7%D9%85%D9%87%E2%80%8C%D9%86%D9%88%DB%8C%D8%B3%DB%8C)، Unit Test روشی است برای آزمودن واحدهای کوچکی از [کد منبع](https://fa.wikipedia.org/wiki/%DA%A9%D8%AF_%D9%85%D9%86%D8%A8%D8%B9) برنامه و اطمینان از درست کار کردن آن‌ها؛ در این روش، درستی هر قسمت از کد، که به آن «واحد[[4]](#footnote-4)» گفته می‌شود، با استفاده از کدهای دیگری که توسط برنامه‌نویس نوشته شده ارزیابی می‌گردد.

در حالت ایده‌آل هر کدام از تست ها از بقیه مستقل است. معمولاً تست های واحد توسط [توسعه‌دهندگان](https://fa.wikipedia.org/wiki/%D8%AA%D9%88%D8%B3%D8%B9%D9%87%E2%80%8C%D8%AF%D9%87%D9%86%D8%AF%D9%87) نرم‌افزار به کار گرفته می‌شوند. نحوه تست واحد می‌تواند از ارزیابی نتیجه روی کاغذ، تا اجرای خودکار چندین آزمایش توسط برنامه و تحلیل نتیجهٔ آن‌ها توسط خود برنامه، متغیر باشد.



یک واحد برنامه دارای خصوصیات زیر می باشد:

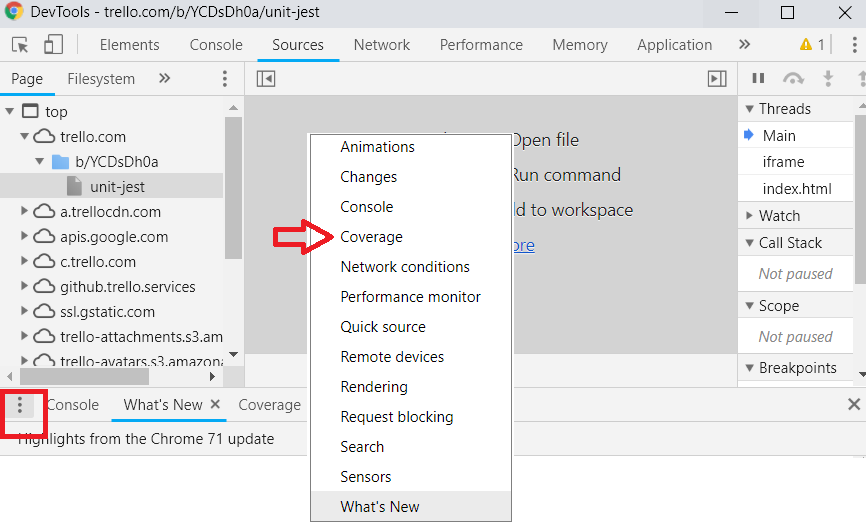
* Solid است و اصولا هیچ وابستگی[[5]](#footnote-5) به بخش های دیگر برنامه ندارد.
* اگر ورودی آن تا ابد برابر با ورودی قبلی بود، خروجی نیز برابر با خروجی که قبلا گرفته ایم باشد. یعنی خروجی نباید رندوم باشد یا برحسب GUID یا زمان تغییر کند.

در این پروژه، ما بر روی چند کد واحد شناخته شده تست را انجام می دهیم که در پوشه test-samples قرار گرفته اند. عبارتند از:

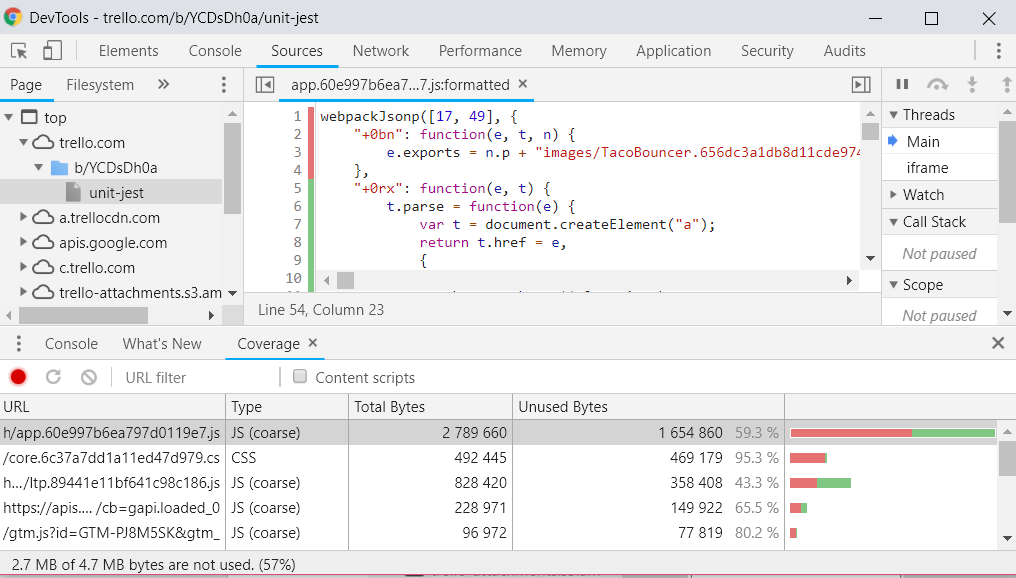
* تابع فاکتوریل[[6]](#footnote-6)
* triangle Classification (دسته بندی مثلث)
* تابع Max
* تابع Min
* تابع فیبوناچی

انواع Coverage Test عبارتند از:

* Statement: تکه هایی از کد بعنوان block Statement شناخته شده و تمام block ها باید اجرا شود.
* Function: اجرای توابع برنامه را نشان می دهد.
* Line/Code: تک تک کدها یکبار اجرا شوند. حدود 99 درصد فضای تست در دنیا روی Code Coverage می باشد. بعنوان مثال در مرورگر کروم از ورژن 15 به بعد امکانی بنام Chrome DevTools اضافه شده است که کد سمت font-end سایت مورد نظر را تحلیل کرده و میزان Coverage را در آن اجرا نمایش می دهد.



شکل زیر جزئیات را نمایش می دهد:



* Branch: برنامه ای که بر حسب شرط ها به شاخه هایی تقسیم می شود در کدام شاخه ها اجرا شده است.

1. **انتخاب زبان برنامه نویسی مناسب:**

از بین زبان های زیر، ابتدا زبان های جاوا اسکریپت و C# منتخب شدند، در نهایت زبان جاوا اسکریپت را بدلیل آشنایی بیشتر و اینکه درگیر مباحثی مانند Object Oriented و syntax گسترده آن زبان ها نشویم و کار سریعتر پیش برود را انتخاب کردیم. همچنین جاوا اسکریپت زبانی است که تابحال برای آن Test Automation نوشته نشده است.

* c
* c++
* c#
* java
* javascript ✓
* python
* ruby
* [vb.net](http://vb.net/)

1. **انتخاب چارچوب طراحی تست در آن زبان[[7]](#footnote-7):**

زبان جاوا اسکریپت بصورت built-in چارچوبی برای «تست واحد» ندارد و از بین پکیج هایی که این امکان را در زبان جاوا اسکریپت فراهم می کنند، Mocha.js (کاملترین چارجوب که امکانات کامل Coverage را دارد)، Jest و chai.js منتخب شدند. اما jest را بدلیل سادگی بیان، Coverage ای که در اختیار قرار می دهد، سرعت و تجربه کاری مان انتخاب کردیم.

* Mocha.js[[8]](#footnote-8)
* Jest[[9]](#footnote-9) ✓
* chai.js
* Jasmine
* YUI Test
* QUnit
* Unit.js
* Jenkins
* Puppeteer

1. **پیاده سازی نمونه تست زنی در چارچوب انتخابی بر اساس مستندات کتابخانه آن:**

Install node.js(minimum supported version is v6 .0 .0)

$ npm install jest--global

$ jest--init / y / y / y

$ mkdir project

Create test project by name project / test.js like attached file

$ jest test

Run tests just changed files based on git(uncommitted files): $ jest - o

Run tests that match this spec name(match against the name in describe or test, basically).jest - t name - of - spec

از مزایای آن این است که اگر در git، commit زده ایم، می توان گفت فقط Un Commit File ها تست شوند. در غیر اینصورت کل پوشه پروژه را تست می کند.

بعنوان مثال، کد زیر در فایل test.js نوشته شده است:

function fact(num) {

var fact = 1;

do {

fact \*= num;

num -= 1;

} while (num >= 1);

return fact;

}

test('null', () => {

const n = null;

expect(n).toBeNull();

expect(n).toBeDefined();

expect(n).not.toBeUndefined();

expect(n).not.toBeTruthy();

expect(n).toBeFalsy();

});

test('zero', () => {

const z = 0;

expect(z).not.toBeNull();

expect(z).toBeDefined();

expect(z).not.toBeUndefined();

expect(z).not.toBeTruthy();

expect(z).toBeFalsy();

});

test('two plus two', () => {

const value = 2 + 2;

expect(value).toBeGreaterThan(3);

expect(value).toBeGreaterThanOrEqual(3.5);

expect(value).toBeLessThan(5);

expect(value).toBeLessThanOrEqual(4.5);

// toBe and toEqual are equivalent for numbers

expect(value).toBe(4);

expect(value).toEqual(4);

});

test('adding floating point numbers', () => {

const value = 0.1 + 0.2;

//expect(value).toBe(0.3); This won't work because of rounding error

expect(value).toBeCloseTo(0.3); // This works.

});

test('there is no I in team', () => {

expect('team').not.toMatch(/I/);

});

test('but there is a "stop" in Christoph', () => {

expect('Christoph').toMatch(/stop/);

});

const shoppingList = [

'diapers',

'kleenex',

'trash bags',

'paper towels',

'beer',

];

test('the shopping list has beer on it', () => {

expect(shoppingList).toContain('beer');

});

test('fact 0 to equal 1', () => {

expect(fact(0)).toBe(1);

});

test('fact 1 to equal 2', () => {

expect(fact(1)).toBe(1);

});

test('fact 2 to equal 6', () => {

expect(fact(2)).toBe(2);

});

test('fact 3 to equal 6', () => {

expect(fact(3)).toBe(6);

});

test('fact 4 to equal 24', () => {

expect(fact(4)).toBe(24);

});

test('fact 5 to equal 120', () => {

expect(fact(5)).toBe(120);

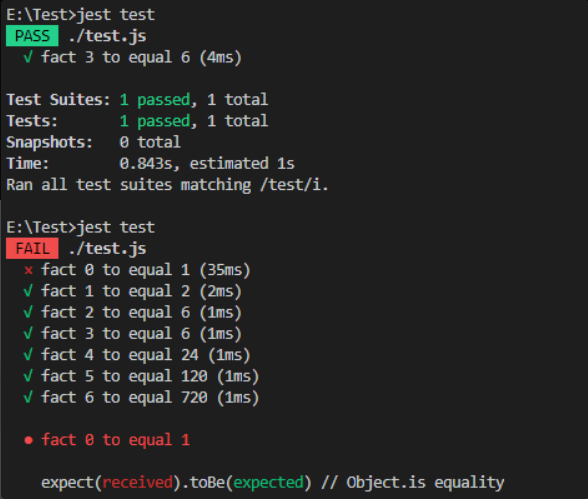
});

test('fact 6 to equal 720', () => {

expect(fact(6)).toBe(720);

});

اجرای آن بصورت زیر می باشد:



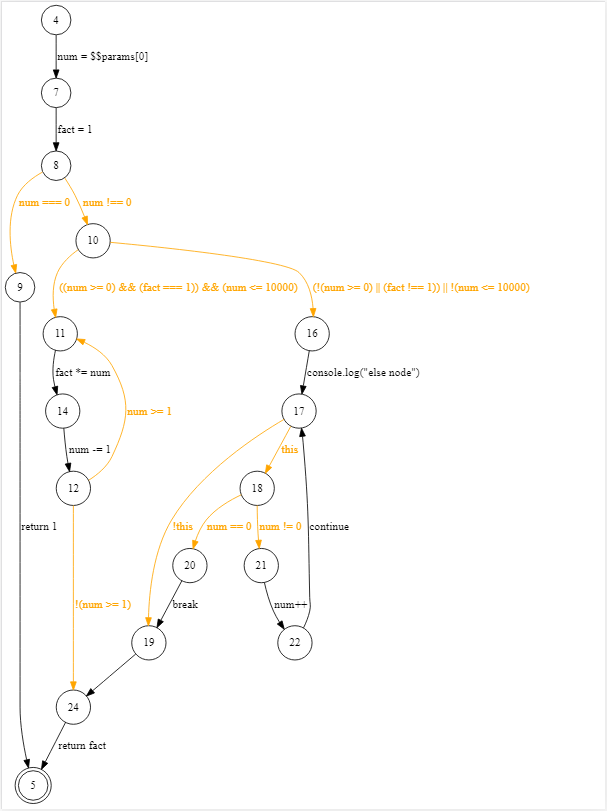
1. **انتخاب کتابخانه رسم گراف در آن زبان.**

از بین کتابخانه های رسم گراف که در زیر قابل مشاهده هستند، بدلیل اینکه بدنبال کتابخانه ای از رسم گراف بودیم که بتواند خروجی کامپایلر ما را رسم کند، اجبارا از کتابخانه viz.js استفاده کردیم. [[10]](#footnote-10)Graphviz ها اطلاعاتی از نوع dot format که استاندارد جهانی است برای نمایش data flow ها هستند. خروجی کامپایلر ما در مرحله آخر به ما Graphviz می دهد و ما با استفاده از کتابخانه viz.js، dot format بدست آمده را بصورت گراف بدست می آوریم.

* <https://www.chartjs.org/>
* <http://visjs.org/>
* <http://sigmajs.org/> (<https://www.bsimard.com/2018/04/25/graph-viz-with-sigmajs.html>)
* <https://d3js.org/>
* [https://canvasjs.com](https://canvasjs.com/)
* <https://developers.google.com/chart/>
* <http://gionkunz.github.io/chartist-js/>
* <http://n3-charts.github.io/line-chart>
* <http://www.amcharts.com/>
* <https://www.highcharts.com/>
* <http://js.cytoscape.org/>
* <https://github.com/anvaka/VivaGraphJS>
* <http://viz-js.com/> ✓

1. **رسم چند گراف با داده های تستی.**

بعنوان نمونه گراف مربوط به تابع فاکتوریل اینگونه بدست آمده است:

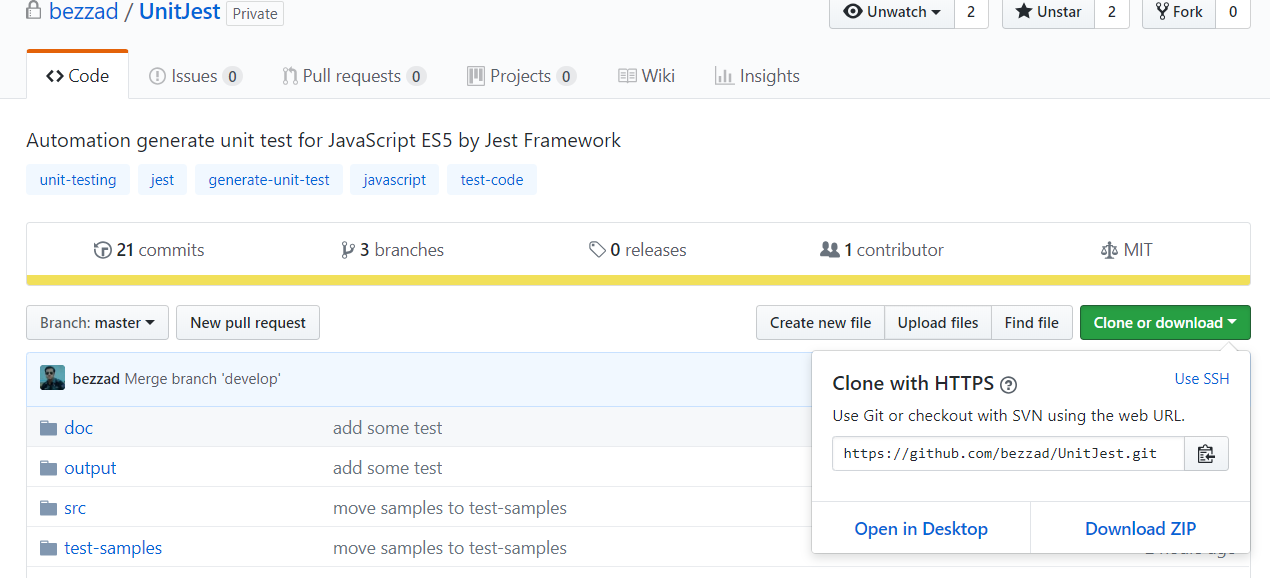


**ساختار کلی پروژه بصورت زیر می باشد:**

1. **index.js:** فایل اصلی[[11]](#footnote-11) پروژه می باشد.
2. **utilities.js:** شامل کلاس Rand است که عدد رندومی تولید می کند که هیچ گاه تکراری نیست. بدلیل اینکه تولید اعداد رندوم برحسب زمان می باشد، در سیکل های زمانی نزدیک بهم تکراری تولید می شوند. در الگوریتم ژنتیک نیز انتخاب پدر و مادر برای تولید نسل بعدی بسیار اهمیت دارد که شبیه به هم نباشند. بنابراین ساخت اعداد رندوم اینگونه بهینه شده اند. کلاس StopWatch، جهت محاسبه مدت زمان اجرای کد بکار می رود تا بدانیم الگوریتم ژنتیک چقدر بهینه شده است و در کل مدت زمان اجرای برنامه را بدست خواهیم آورد.

**چگونه استفاده کنیم**

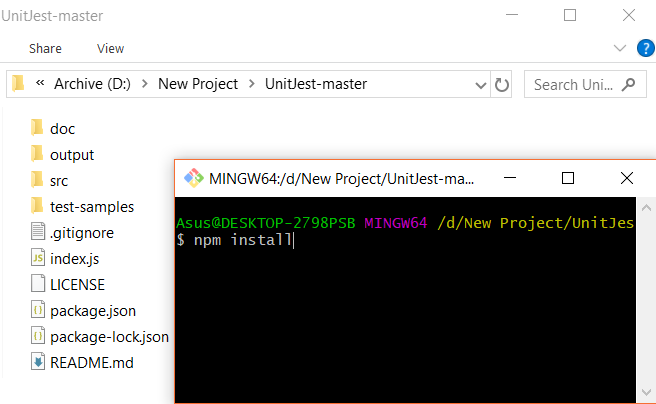
1. ابتدا node js را بر روی سیستم خود نصب کنید.
2. به آدرس https://github.com/bezzad/UnitJest در سایت گیت هاب رفته و از قسمت clone or download طبق شکل زیر پروژه را دانلود کنید.



در محل دلخواهی از کامپیوتر فایل دریافتی را از حالت فشرده خارج کنید.

1. وارد پوشه پروژه شوید و محیط Cmd ویندوز[[12]](#footnote-12) را در آن پوشه باز کنید. دستور زیر را وارد کنید تا پکیج های استفاده شده در پروژه از سایت گیت هاب بر روی کامپیوتر شما دانلود شود.

npm install



با اجرای دستور فوق، پوشه node\_modules ساخته شده و ماژول ها در آن قرار می گیرند.

1. بعد از اجرای برنامه پوشه output ساخته شده و داده ها در آن قرار می گیرند.(کد کامپایل شده)
2. با دستور زیر برنامه را تست می کنیم (با این دستور فایل test.js توسط فریم ورک jest اجرا می شود):

$ jest test.js

**مراجع:**

1. <https://en.wikipedia.org/wiki/Unit_testing>
2. <https://github.com/facebook/jest>
3. <https://jestjs.io/docs/en/getting-started>
4. <http://viz-js.com/>
5. https://en.wikipedia.org/wiki/Graphviz
6. <http://martintrojer.github.io/software/2011/11/02/symbolic-execution>
7. <https://github.com/leebyron/testcheck-js>
8. <https://www.cs.ccu.edu.tw/~naiwei/cs5812/st4.pdf>

1. How to work [↑](#footnote-ref-1)
2. How to use [↑](#footnote-ref-2)
3. Framework [↑](#footnote-ref-3)
4. Unit [↑](#footnote-ref-4)
5. Dependency [↑](#footnote-ref-5)
6. Factorial [↑](#footnote-ref-6)
7. Java Script Test FrameWork [↑](#footnote-ref-7)
8. https://mochajs.org [↑](#footnote-ref-8)
9. <https://github.com/facebook/jest>, https://jestjs.io/docs/en/getting-started [↑](#footnote-ref-9)
10. https://en.wikipedia.org/wiki/Graphviz [↑](#footnote-ref-10)
11. main [↑](#footnote-ref-11)
12. Powershell window [↑](#footnote-ref-12)