

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 نسخهES5 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 ای که پیاده سازی شده را شرح خواهد داد و بخش دوم، «چگونه از برنامه استفاده کنیم» [[1]](#footnote-1) شرح استفاده از این برنامه را بیان می کند.

**لازم به ذکر می باشد، سورس کد پیاده سازی شده برنامه بعلت حجم بالا و عدم پشتیبانی سایت LMS، در وب سایت گیت هاب به آدرس زیر قرار داده شده است و همچنین می توانید با توجه به بخش «چگونه از برنامه استفاده کنیم» آن را دریافت، نصب و راه اندازی کنید.**

[**https://github.com/bezzad/UnitJest**](https://github.com/bezzad/UnitJest)

**روند انجام پروژه**

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

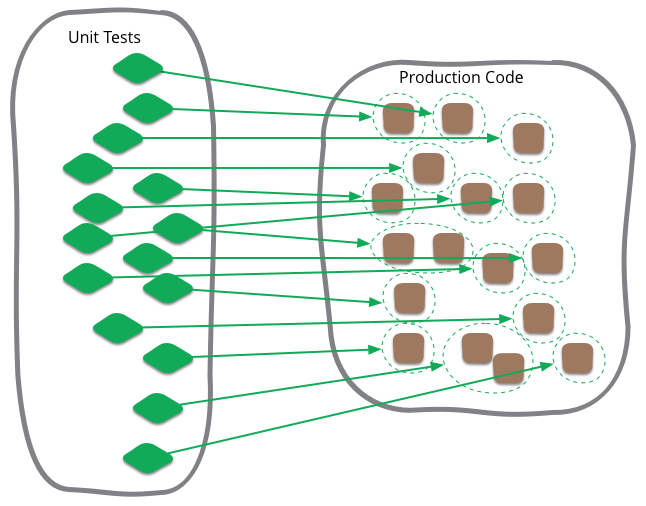
1. معرفی Unit Test
2. انتخاب زبان برنامه نویسی جاوااسکریپت
3. انتخاب چارچوب[[2]](#footnote-2) طراحی تست در زبان انتخابی
4. پیاده سازی نمونه تست در چارچوب انتخابی
5. طراحی پیش پردازشگر زبان انتخابی
6. انتخاب کتابخانه رسم گراف در زبان انتخابی
7. رسم گراف از خروجی مفسر
8. ثبت مراحل انجام کار در سایت GitHub: جهت انجام راحت تر کار گروهی پروژه
9. ایجاد نمونه کدهای آزمایشی جهت تولید تست برای آنها

**شرح** **روند انجام پروژه**

1. **معرفی 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[[3]](#footnote-3)» 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[[4]](#footnote-4) 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.

1. **انتخاب زبان برنامه نویسی:**

از بین زبان های زیر، 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، انتخاب کردیم. همچنین جاوا اسکریپت زبانی است که برای آن Automation Unit Test وجود ندارد.

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

1. **انتخاب چارچوب طراحی تست در زبان انتخابی[[5]](#footnote-5):**

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

* Mocha.js[[6]](#footnote-6)
* Jest[[7]](#footnote-7) ✓
* chai.js
* Jasmine
* YUI Test
* QUnit
* Unit.js
* Jenkins
* Puppeteer

1. **پیاده سازی نمونه تست در چارچوب انتخابی:**
2. Install node.js(minimum supported version is v6 .0 .0)
3. $ npm install jest--global
4. $ jest--init
5. $ mkdir project
6. Create test project by name project / test.js like attached file
7. $ jest test

بعنوان مثال، کد زیر که در فایل test.js نوشته شده است مربوط به تست یک تابع فاکتوریل می باشد:

function fact(num) {

var fact = 1;

do {

fact \*= num;

num -= 1;

} while (num >= 1);

return fact;

}

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);

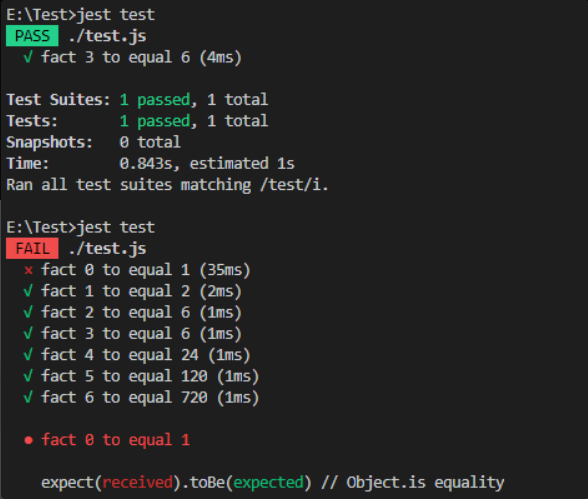
});

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

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

});

که پس از اجرا آن، خروجی آن بصورت زیر می باشد:



شکل 1 خروجی تست پس اجرای دستور jest test

1. **طراحی پیش پردازشگر زبان انتخابی:**

ما از مفسر [[8]](#footnote-8)Esprima برای تفسیر کدهای تست استفاده می کنیم که خروجی آن یک درخت AST می باشد. بدلیل اینکه ممکن است کد مورد آزمایش ما دارای ساختار نامنظم و غیر استاندارد برای تحلیل نمودار و مانیتورینگ Coverage کدها دشوار باشد. بنابراین ساختار AST بدست آمده را به کتابخانه متن باز escodegen[[9]](#footnote-9) می دهیم تا کد ساختار یافته ای از درخت برنامه تست ما بوجود آورد که این کد با فرمت استاندارد می باشد، بطوریکه در آن هر سطر کد مربوط به یک دستور می باشد.

حال نوبت به پرچم گذاری روی تک تک کدهای برنامه است. بعبارت دیگر، برای اینکه بدانیم برنامه در هنگام اجرا از چه مسیری عبور می کند باید بتوانیم برنامه را مانیتور کنیم. برای این کار نیاز به اضافه کردن تکه کدی به بعد هر سطر از برنامه را داریم تا به ما بگوید حالا کدام سطر اجرا شده است که به آن عمل instrument code[[10]](#footnote-10) گفته می شود. عملیات فوق در فایل instrument.js پیاده سازی شده است.

1. **انتخاب کتابخانه رسم گراف در زبان انتخابی:**

همانطور که گفتیم، ما از مفسر Esprima با خروجی درخت AST استفاده می کنیم و سپس از روی آن گراف جریان برنامه با فرمت Graphviz[[11]](#footnote-11) تولید می شود. از بین کتابخانه های رسم گراف که در زیر مشاهده می کنید، تنها کتابخانه [viz.js](http://viz-js.com/) از این فرمت پشتیبانی می کند، بنابراین این کتابخانه را انتخاب کردیم.

<https://www.chartjs.org/>

* <http://sigmajs.org/>
* <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 گراف از خروجی مفسر:**

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

کد فرمت دات برای رسم گراف:

// fact

digraph control\_flow\_graph {

node [shape = doublecircle] 5

node [shape = circle]

// Unconditional edges

4 -> 7 [label = " num = $$params[0]"]

7 -> 8 [label = " fact = 1"]

8 -> 11 [label = " fact \*= num"]

10 -> 5 [label = " return fact"]

11 -> 9 [label = " num -= 1"]

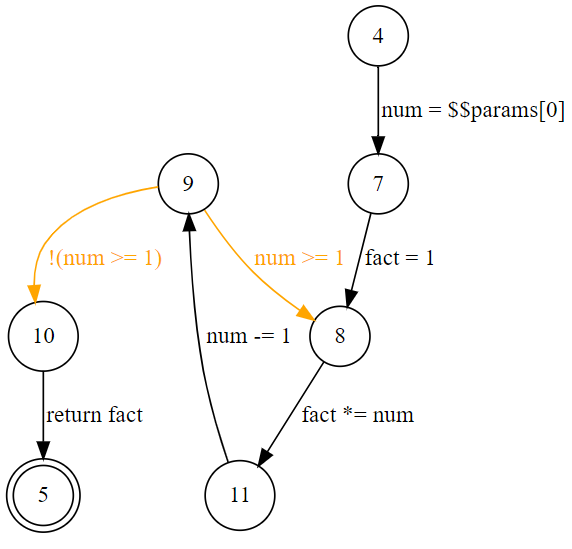
// Conditional edges

edge [color = orange, fontcolor = orange]

9 -> 8 [label = " num >= 1"]

9 -> 10 [label = " !(num >= 1)"]

}



شکل 2رسم گراف کنترل جریان تابع فاکتوریل با vis.js

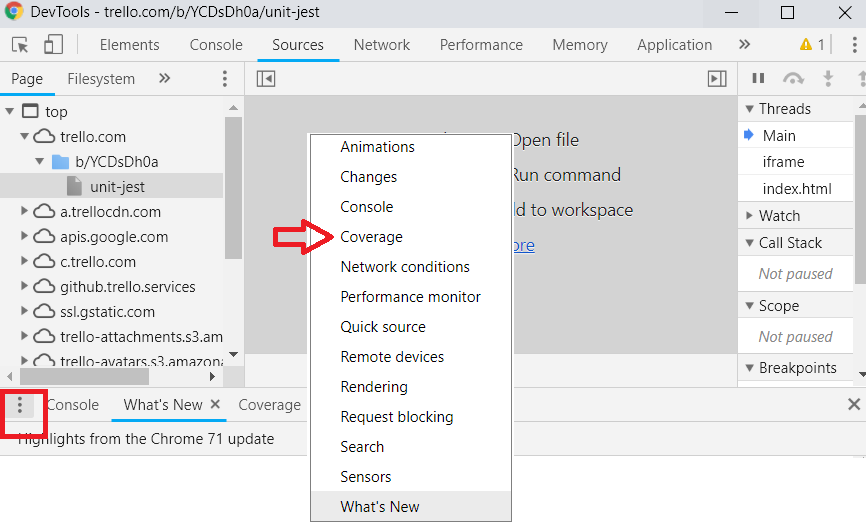
1. **ثبت مراحل انجام کار در سایت Github.com جهت انجام راحت تر کار گروهی پروژه**
2. **ایجاد نمونه کدهای آزمایشی جهت تولید تست برای آنها:**

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:

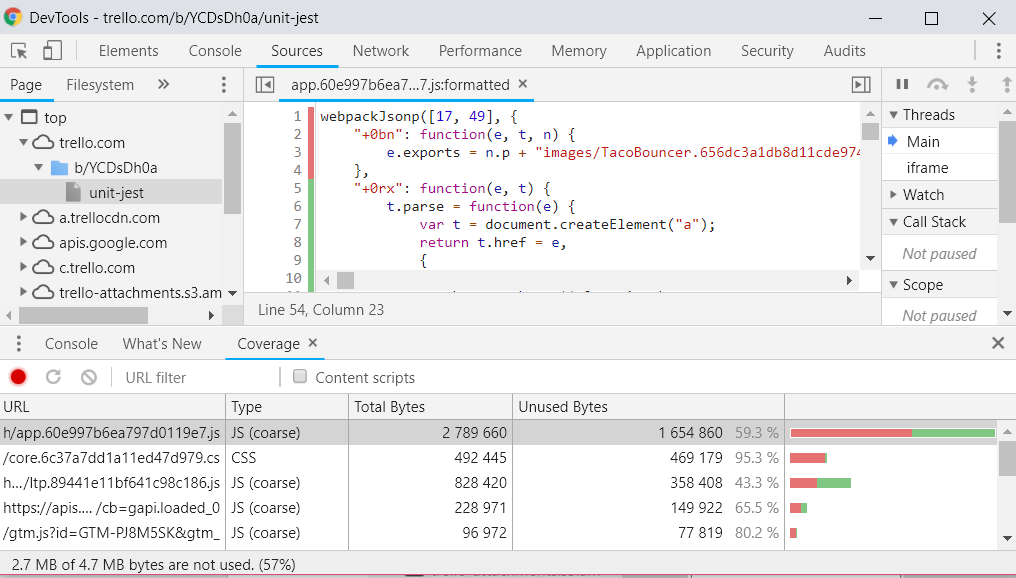
* تابع فاکتوریل[[12]](#footnote-12)
* triangle Classification (دسته بندی مثلث)
* تابع Max
* تابع Min
* تابع فیبوناچی

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.

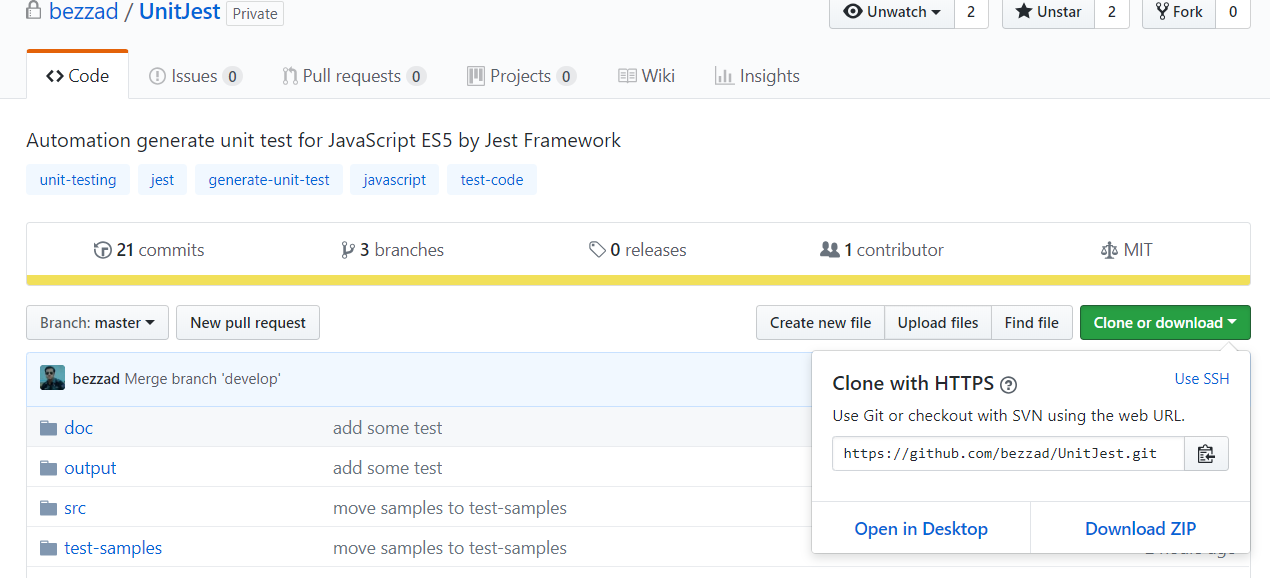
1. **تحلیل برنامه اولیه برای تست کردن کدهای نمونه**
2. **درآوردن فلوچارت تحلیل کد**
3. **ساده سازی مراحل طرح تا حد ممکن و اولویت دهی آن**
4. **شروع پیاده سازی با زبان انتخابی**
5. **پیاده سازی CI ساده**
6. **مستندسازی و تست نرم افزار**
7. **تصویر سازی مستند و تحویل نهایی**

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

1. **index.js:** فایل اصلی[[13]](#footnote-13) 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 از برنامه استفاده کنیم**

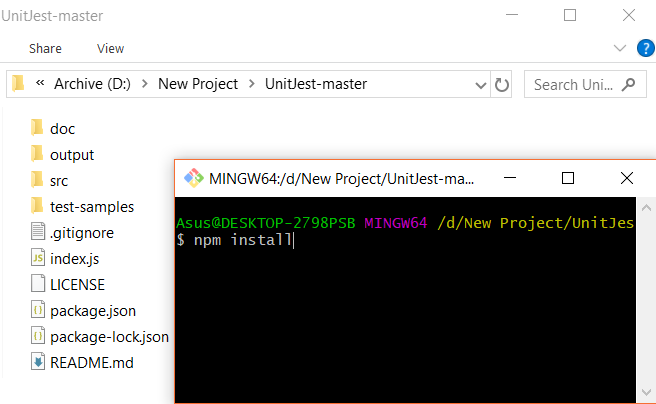
1. node js را بر روی سیستم خود نصب کنید.
2. به آدرس https://github.com/bezzad/UnitJest در وب سایت گیت هاب رفته و از قسمت clone or download طبق شکل زیر پروژه را دانلود کنید.



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

1. وارد پوشه پروژه شوید و محیط Cmd ویندوز[[14]](#footnote-14) 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. https://github.com/jquery/esprima
5. <http://viz-js.com/>
6. <https://en.wikipedia.org/wiki/Graphviz>
7. http://web.cs.iastate.edu/~weile/cs513x/4.ControlFlowAnalysis.pdf
8. <http://martintrojer.github.io/software/2011/11/02/symbolic-execution>
9. <https://github.com/leebyron/testcheck-js>
10. <https://www.cs.ccu.edu.tw/~naiwei/cs5812/st4.pdf>

1. How to use [↑](#footnote-ref-1)
2. Framework [↑](#footnote-ref-2)
3. Unit [↑](#footnote-ref-3)
4. Dependency [↑](#footnote-ref-4)
5. Java Script Test FrameWork [↑](#footnote-ref-5)
6. <https://mochajs.org> [↑](#footnote-ref-6)
7. <https://github.com/facebook/jest> , <https://jestjs.io/docs/en/getting-started> [↑](#footnote-ref-7)
8. <https://github.com/jquery/esprima> [↑](#footnote-ref-8)
9. <https://github.com/estools/escodegen> [↑](#footnote-ref-9)
10. <https://en.wikipedia.org/wiki/Instrumentation_(computer_programming)> [↑](#footnote-ref-10)
11. Graphviz ها اطلاعاتی از نوع dot format برای نمایش گراف جریان برنامه هستند. [↑](#footnote-ref-11)
12. Factorial [↑](#footnote-ref-12)
13. main [↑](#footnote-ref-13)
14. Powershell window [↑](#footnote-ref-14)