**Unit Test in JavaScript**

**چرا برای اپلیکیشن هامون تست مینویسیم؟؟**یک طرز تفکری که وجود داره اینکه اگر ما برای اپلیکیشن هامون بخواهیم تست بنویسم سرعت توسعه پروژه مون کاهش پیدا میکنه البته ممکنه این تفکر تا یک قسمتهایی درست باشه اما زمانی که بخواهیم در دراز مدت یک پروژه رو توسعه بدیم یا مثلا فیچرهای جدید به پروژه مون اضافه کنیم این تفکر اصلا درست نیست و اگر تست مینوشتیم این کار میتونست حتی خیلی بیشتر سرعت مون رو افزایش بده و سریع تر تسک هامونو انجام بدیم.اگر از تست استفاده کنیم مثلا اگر ما چند تا فیچر جدید به پروژه مون اضافه کردیم ..اون تست هایی که نوشتیم میاد خودش تک تک فیچر ها رو تست میکنه که مشکلی نداشته باشه و اگر مشکلی داشته باشه به ما میگه که کجای کد مون مشکل داره و حتی بررسی میکنه که ایا با فیچر های دیگه تداخل داره یا نه...!!! و خیلی راحت تر از مدل قدیمیهه که قبلا دستی خودمون فیچر ها رو چک میکردیم.  
**مزایای تست نویسی**

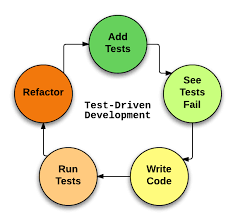
* افزایش سرعت توسعه نرم افزار
* تضمین کارکرد کدها توسط تست
* ورودی و خروجی بخش های مختلف پروژه در ابتدای پروژه مشخص میشود
* تاثیر متقابل کدهای مختلف و شناسایی تداخل در کدها
* کاهش هزینه های مالی

**تست واحد یا (Unit Test)**

منظور از تست واحد اینکه کد های پروژه ت را به بخش های کوچیکی تقسیم میکنیم و اون بخش های کوچیک رو تست میکنیم و بصورت کلی زمانی که این بخش های کوچک را کنار هم بزاریم میشه کل پروژه ای مون. دو روش رایج در تست واحد روش های 1- TDD 2-BDD

**روش TDD (Test-driven development)**

یک چرخه یا سایکل هست که با رعایت کردن این روش میتوانیم اپلیکیشن مان را به وجود بیاوریم.



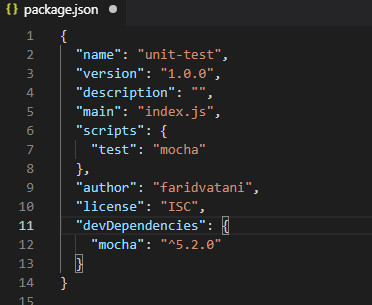
اگر به شکل دقت کنید در TDD میگوید که اولین کاری که باید در پروژه ت انجام بدی اینکه برای اون ویژگی که مورد نظرت هست یک تست بنویس به عنوان مثال ما میخواهیم یکسری اطلاعات رو توی یک مدلی ذخیره کنیم و بعد از این که تست رو نوشتی بعد بیا اون تست خودتو تست کن و اگر اون تست شکست خورد اون مقدار تغییر را به اون کدی که در تست اولیه (بخش Add Tests) نوشتیم اضافه کن که باعث این بشه که تست ما با موفقیت انجام بشه و بعد مجددا کد رو تست کن و اگر با موفقیت انجام شدهش رو بیا ریفکتور کن و بعد برو سراغ تست بعدی.....

**روش BDD (behavior-driven development)**

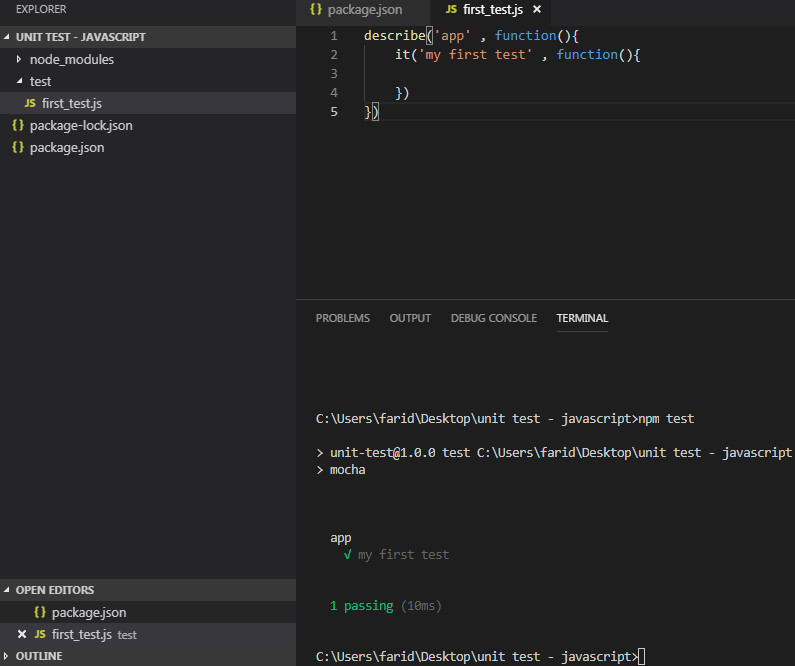
برگرفته شده از TDD ست و شبیه به هم هستند منتها در BDD میگوید که ما رفتار پروژه خودمان یا رفتار این کد یا رفتار این فیچر را بفهمیم در واقع مفهومی ترهه و از نظر پیاده سازی با TDD تفاوتی ندارد.

**آشنایی و شروع تست با Mocha.js**

mocha با یکی از قدرتمندترین ابزار تست در جاوااسکریت هست که در پروژه های زیادی هم مورد استفاده قرار میگیرد و یکی از ویژگی های دوست داشتنی موکا اینکه یک خروجی قابل فهم در کامند لاین ارائه میدهد و از async هم پشتیبانی میکند.  
در قدم اول ما برای اینکه تست هامون را اجرا کنیم نیاز به package.json داریم. ما برای تست این بخش میخواهیم از node.js در بکند استفاده کنیم.با دستور (npm init) package.json پروژه مونو درست میکنیم و بعد شروع به نصب mocha.js میکنیم…



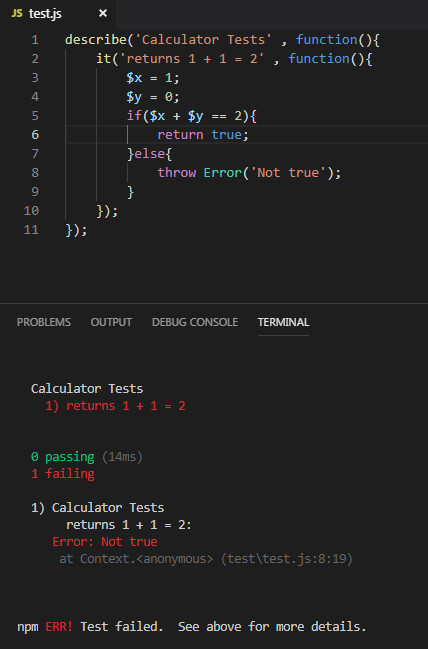
Mocha برای اینکه تست هامونو شناسایی کنه باید یک فولدر به اسم test در دایرکتوری پروژه مون بوجود بیاوریم و بعد شروع کنیم تست نوشتن.در قدم اول باید عناوین تست هامونو مشخص کنیم. هر عنوان میتواند تعداد زیادی تست را شامل شود و برای توصیف عناوین میتوانیم از متد describe(‘’) توصیف شون کنیم.



**نوشتن اولین تست**

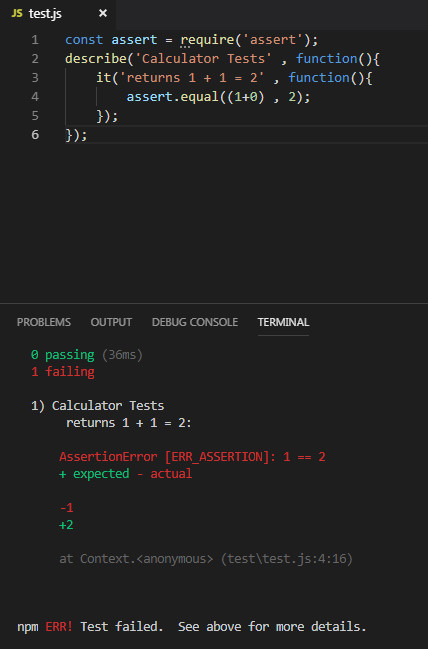
در این بخش میخواهیم چند نمونه از تست های مختلف را بنویسیم. اولین تستی که میخواهیم بنویسیم تست های مربوط به محاسبات ریاضی است.

در تست زیر ما دو مقدار داریم که اگر جمع دو عدد درست باشه مقدار true رو برای ما برگردونه واگر غلط باشه پیام not True رو برگردونه...(ما در این جا مقادیر رو طوری وارد کردیم که شرط غلط باشه)

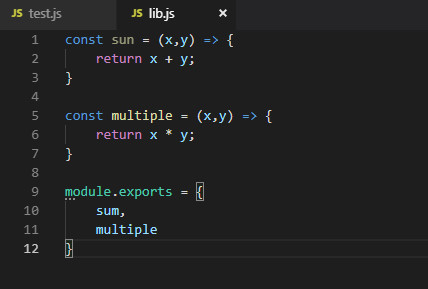


ما یک ماژولی در node.js داریم به اسم Assertion که برای تست استفاده میشود که میتونیم از این متد استفاده کنیم چراکه نوشتن کد مثل مثال بالا خطوط کدهای ما رو زیاد میکنه...

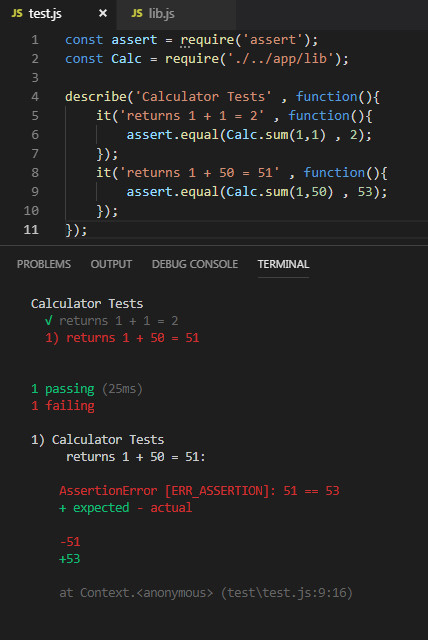
https://nodejs.org/api/assert.html



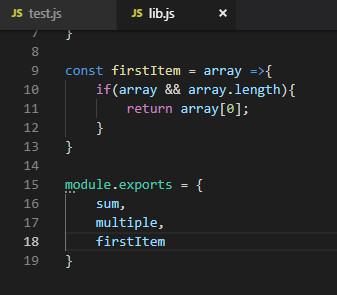
فرض کنید ما یک فایل داریم به عنوان lib.js که میخواهیم از متد های ان را بصورت ماژول در تست هامون استفاده کنیم.

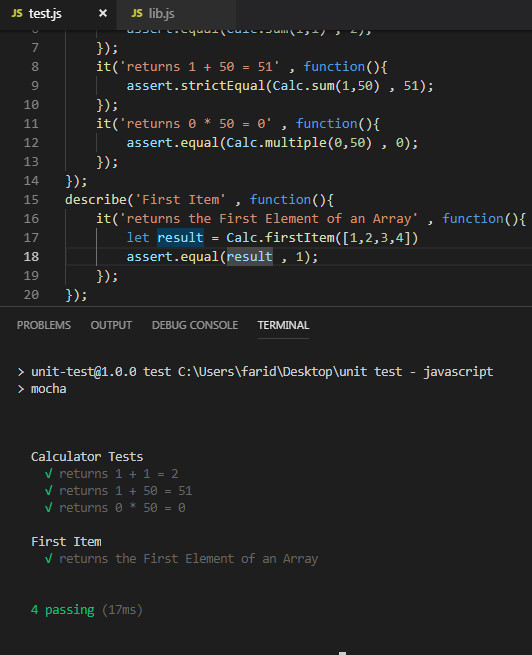


که اگر بخواهیم از این متد در تست هامون استفاده کنیم بصورت زیر میشود.



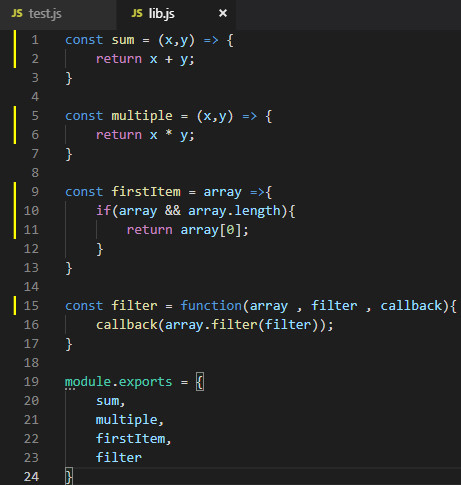
نکته دیگر اینکه متد assert.strictEqual باید هم از نظر مقدار و هم از نظر نوع داده (Data Type) باید باهم برابر باشند.مثال دیگری را که میخواهیم کار کنیم در مورد اینکه میخواهیم اولین ایتم از یک ارایه را برای ما برگردونه که ما تابع اونو در lib.js مینویسیم و سپس در تست مون ازش استفاده میکنیم.

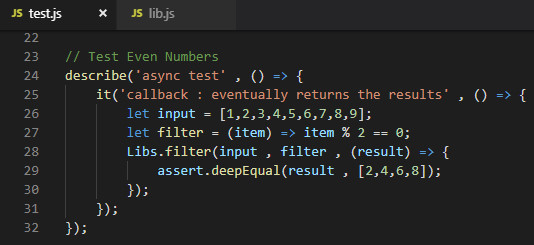


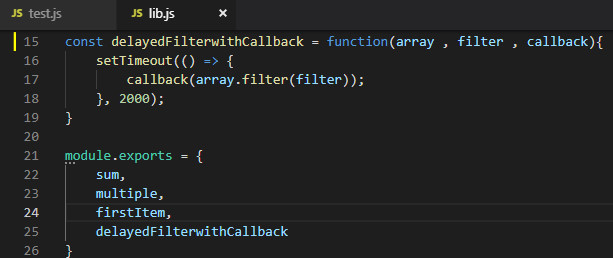


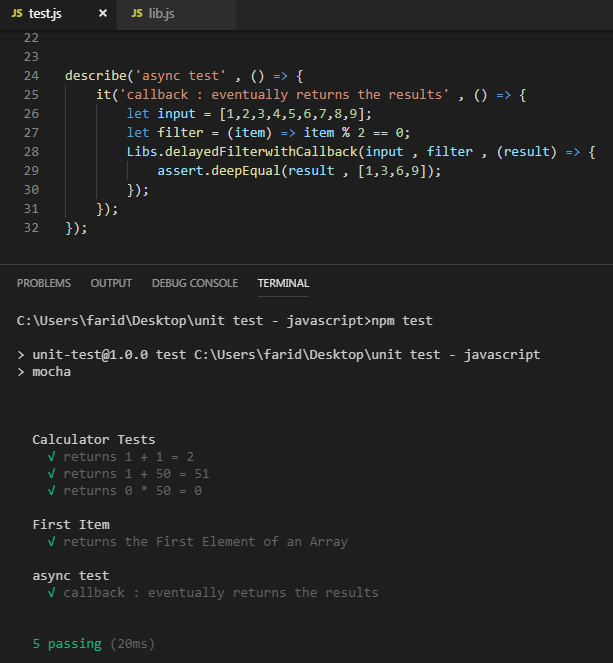
**تست توابع async**

فرض کنید ما یک متدی تحت عنوان filter که در قالب یک تابع یک ارایه را بگیرد و یک تابع تحت فیلتر کردن را بگیرد و در نهایت نتیجه در قالب کال بک برگردوند



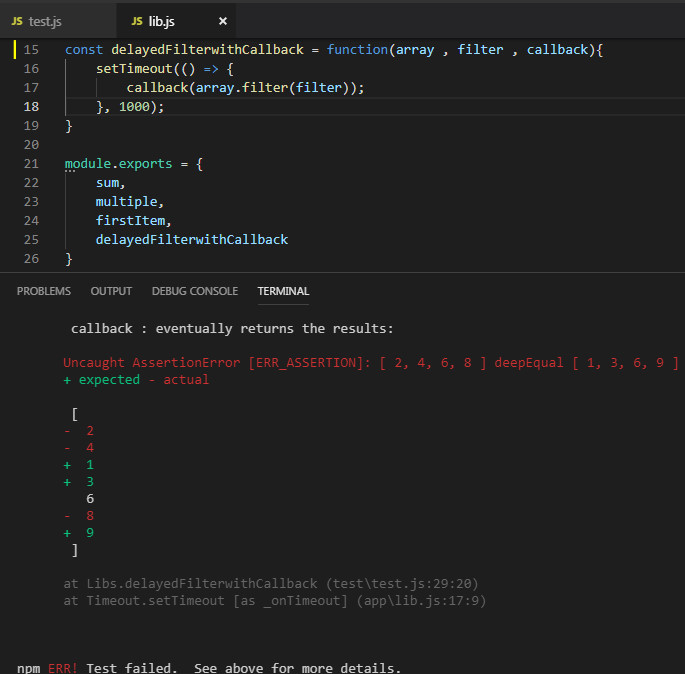
و میخواهیم برای ما اعداد زوج را برگرداند.  


مثالی که زدیم یک کال بک واقعی نیست چون زمانی رو از ما نمیگیرد اما ما میخواهیم یک زمانی را برای ما صبر کند به عنوان مثال 2 ثانیه و بعدش این مقدار رو برای ما برگرداند.به مثال زیر توجه کنید

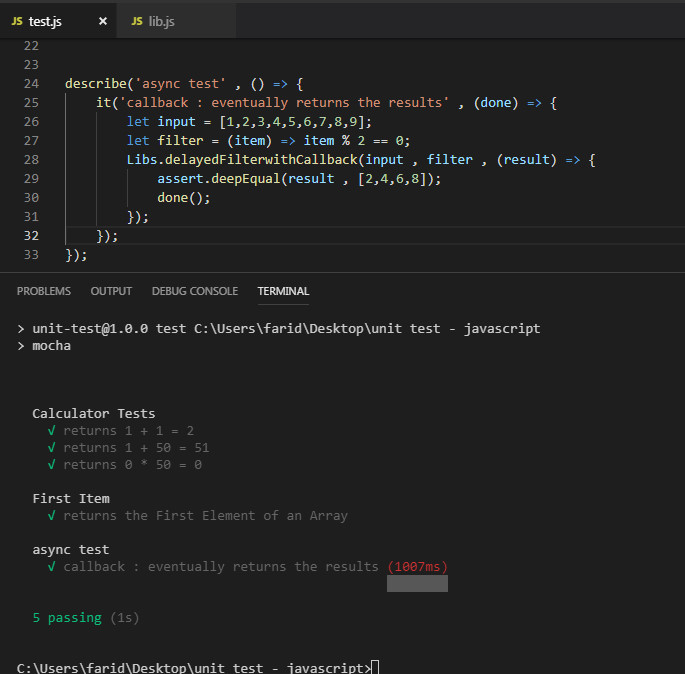


خوب همونطور که مشاهده میکنید نتیجه تست خیلی سریع برای ما برگشت داده می شود و نکته جالب اینجاست که ما تست رو غلط نوشتیم که ارور بده اما تست شکست نخورد و دلیل ش چیه؟

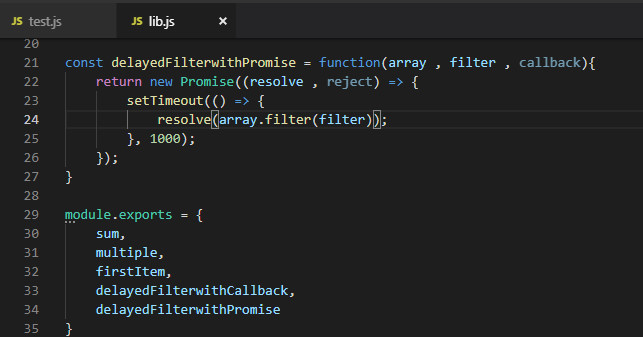
به این دلیل که زمانی که ما از یک کال بک استفاده میکنیم و یک زمانی رو صبر میکنه تا نتیجه رو برگردونه اما منتظر نمیکنه و چون هیچ اروری برگشت پیدا نمیکنه پس میگه که بقیه ش درسته و بقیه ش رو اجرا نمیکنه اما برای این که صبر کنه و وقتی که داریم از کال بک ها استفاده میکنیم میتونیم در در it مون از done استفاده کنیم بصورت زیر

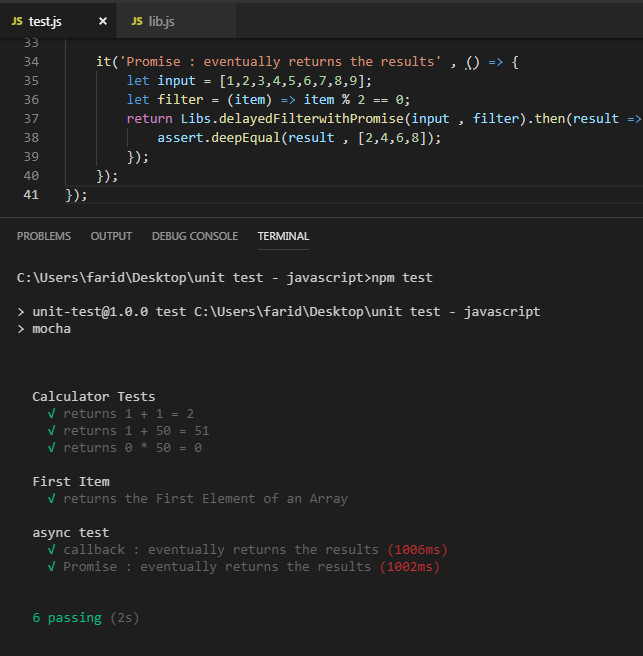


میبینید که وقتی مقدار اشتباه وارد کردیم خطا میدهد اما اگر مقدار درست بدهیم نتیجه چگونه خواهد بود؟! مشاهده میکنید که یک تایمی رو منتظر میماند...

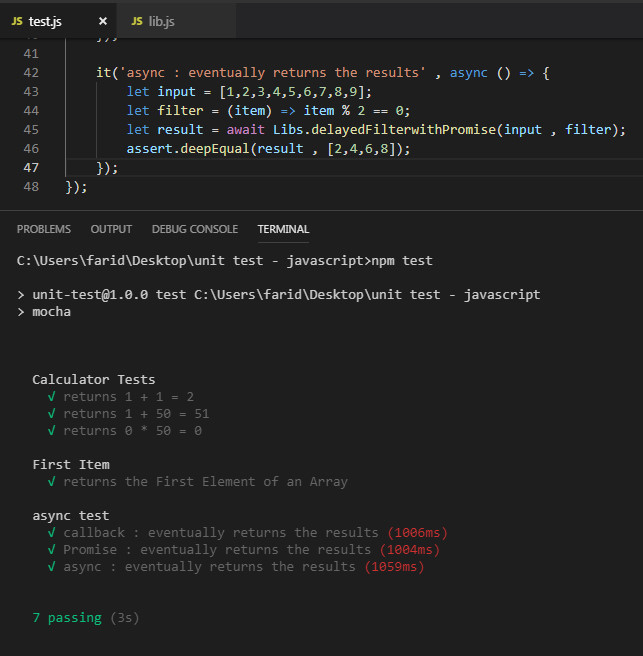


**در این قسمت میخواهیم از Promise ها استفاده کنیم!**



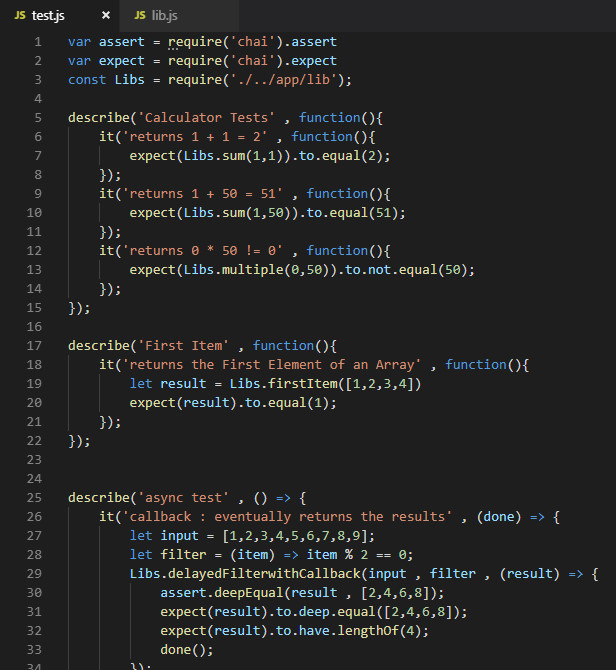
****

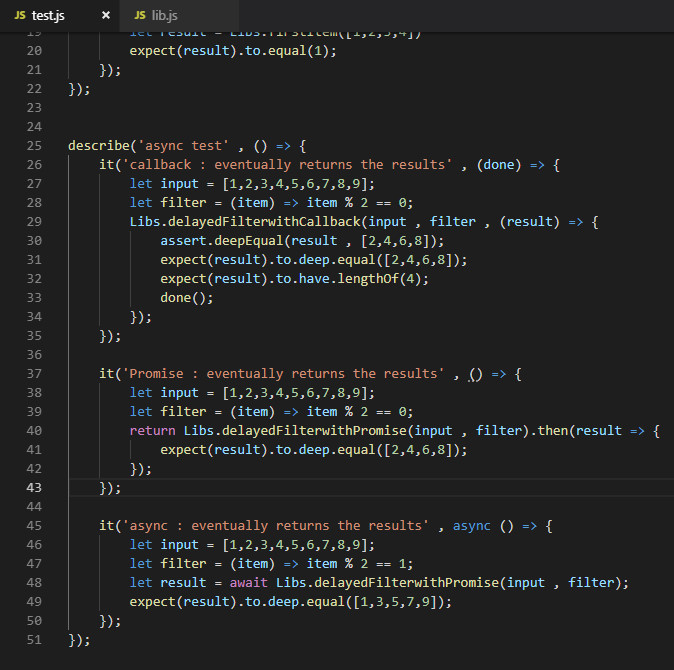
**Async و await در تست نویسی جاوا اسکریپت**

****

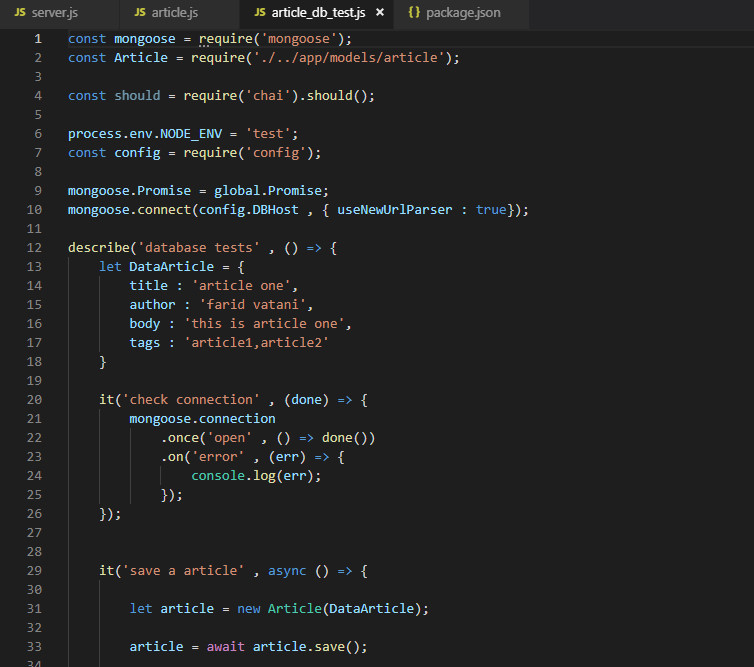
**تست نوشتن روش BDD (Behavior driven development)**

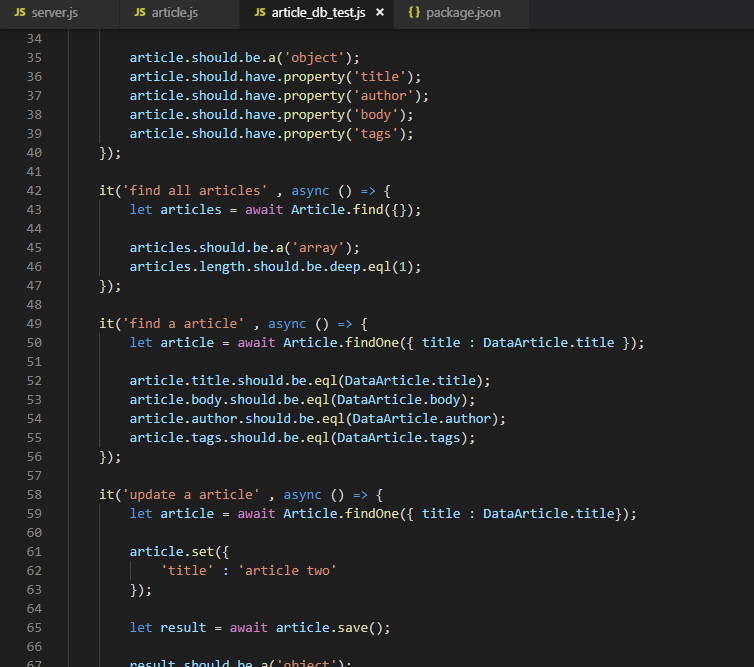
ما ماژول های مختلفی برای BDD داریم مثل : [better-assert](https://github.com/tj/better-assert) ، [Chai Assertion Library](https://www.chaijs.com/) و ....  
ما میخواهیم از chai استفاده میکنیم. برای تست های BDD ما از should و expect استفاده میکنیم.ما مثال هایی که با روش TDD زده بودیم را الان با روش BDD مینویسیم.

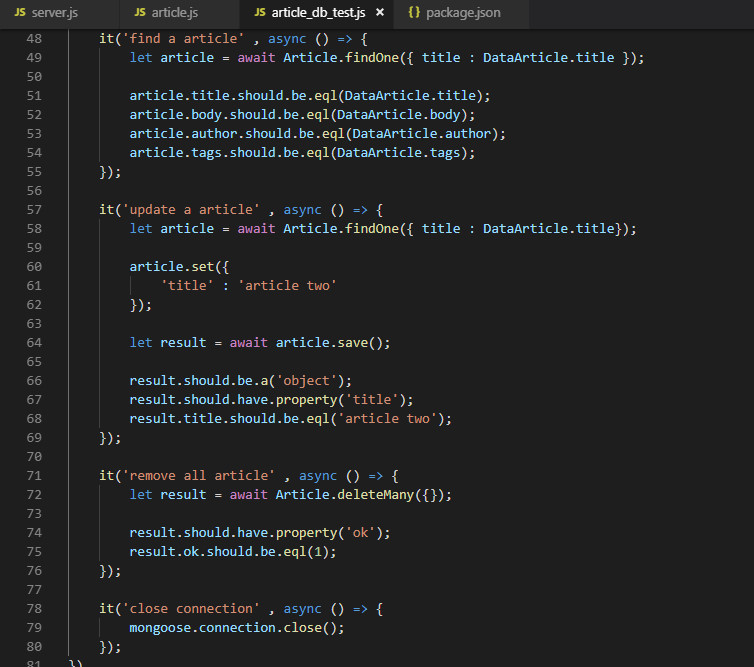


****

**نوشتن تست برای دیتابیس**برای نوشتن تست ما یک پروژه رو ایجاد میکنیم که مثلا یک مقاله رو ذخیره ، نمایش ، اپدیت ، حذف کردن را انجام بدهیم. تست هایی که میخواهیم بنویسم به روش BDD می باشد







آشنایی با قلاب ها یا هوک های تست

قلاب یا hook در تست به ما کمک میکند که قبل یا بعد از انجام یک تست یا همه تست های خودتون، عملیات خاصی رو به انجام برسونید که مطمئنا در نتیجه تست شما تاثیر زیادی دارد .ما 4 صورت قلاب داریم که بصورت زیر هست   
https://mochajs.org/#hooks

describe('hooks', function() {

before(function() {

// runs before all tests in this block

});

after(function() {

// runs after all tests in this block

});

beforeEach(function() {

// runs before each test in this block

});

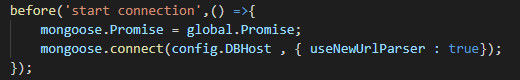
afterEach(function() {

// runs after each test in this block

});

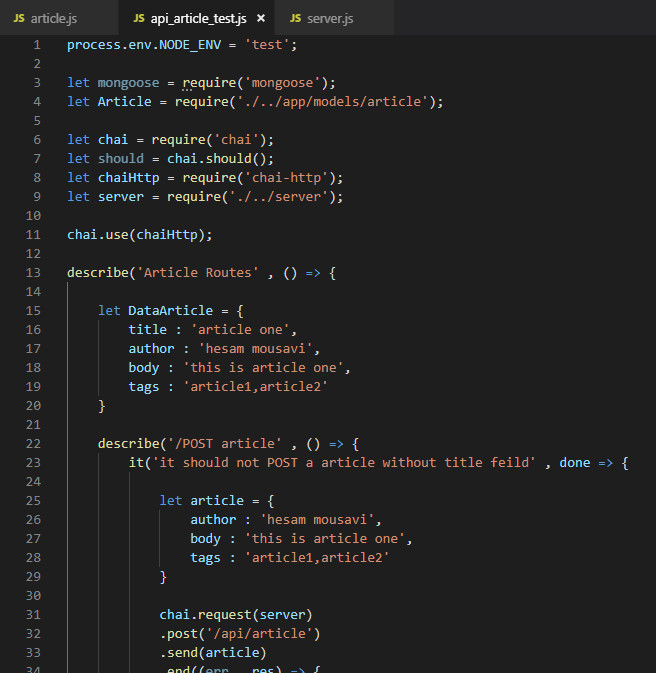
// test cases

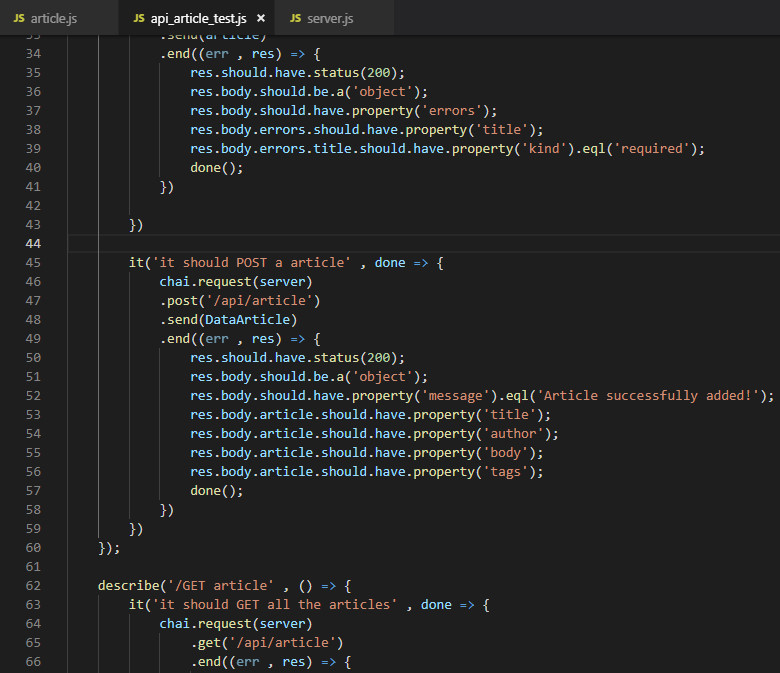
});

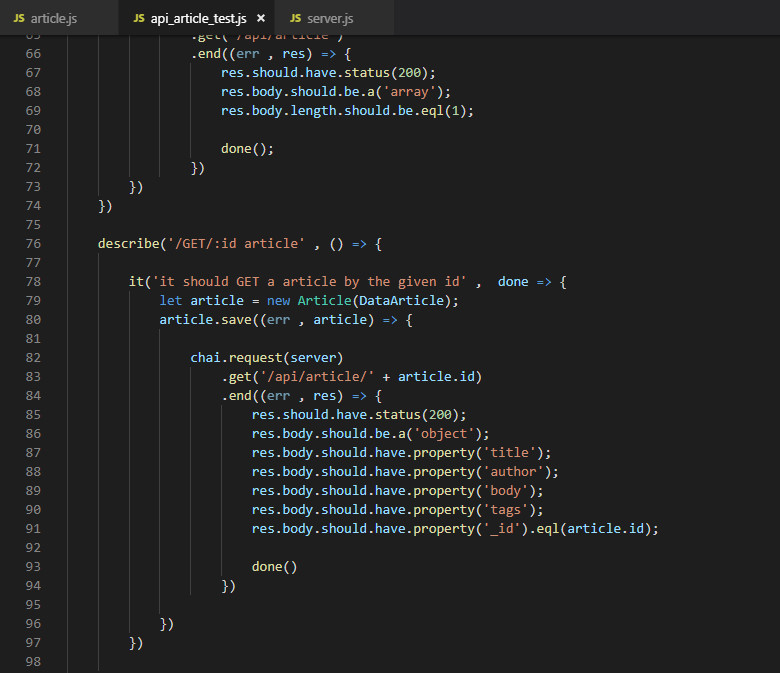


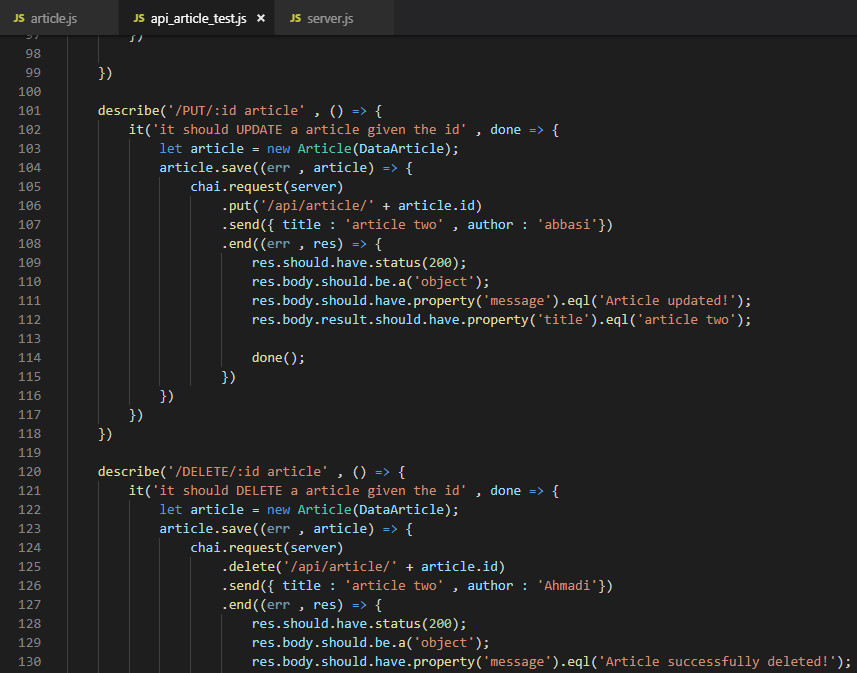
ساخت یک restful api کامل

در این قسمت ما ابتدا باید یک پروژه رو به صورت Restful api بنویسیم و بعد برای ان شروع به نوشتن تست میکنیم.











پایان !  
لینک پروژه :  
https://bitbucket.org/faridvatani/jsunittest/src/master/