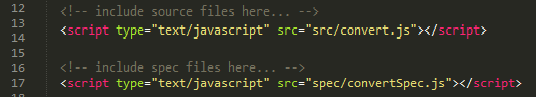
**Introduktion**

I fördjupningen som vi har fått i uppgift att göra i kursen Riautveckling med JavaScript så har jag bestämt mig för att testa Jasmine som är ett ramverk för att testa JavaScript. Jag har under dessa två veckor i samband med kursen Mjukvarutestning gjort en liten *gissa talet APP* och använt mig av Jasmine och Karma för att köra mina tester.

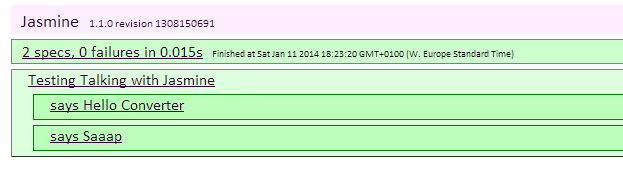
**Komma igång med Jasmine**

Jag har testat två alternativ för att komma igång och skriva tester med Jasmine.

Det första jag testade var att köra Jasmine som ”Stand Alone”. För att göra detta så laddar man hem Jasmine biblioteket och sedan förser man det med sina Jasmine-tester och sourcefilerna som man behöver för att göra sina tester i Specrunner.html (specfiler är testfilerna).



Jag valde att istället använda mig av RequireJS för att ladda in alla mina filer. När Specrunner.html är försedd med de filer som du behöver för att genomföra testerna så öppnar du upp Specrunner.html i din browser och Jasmine presenterar då snyggt och prydligt vilka testar som körts och om de är godkända eller felaktiga (godkända tester är gröna).

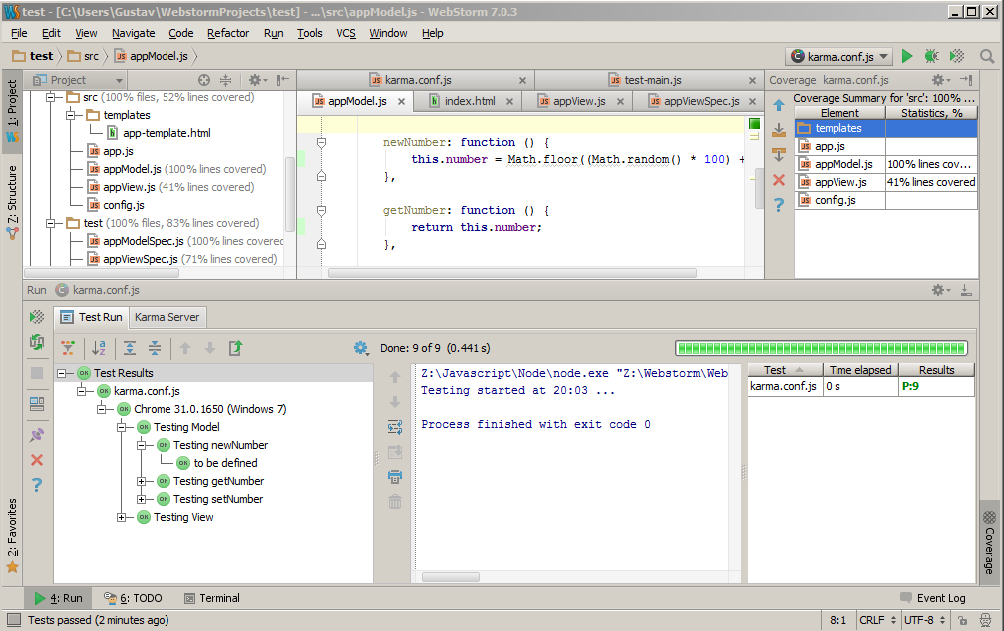


När jag sedan ville se code coverage bestämde jag mig för att även använda Karma och Karma-coverage vilket är mitt alternativ två för att köra Jasmine-tester plus code coverage. Jag bytte även utvecklingsmiljö från Visual Studio till Webstorm, detta för att jag hade problem med min version av Visual Studio när jag skulle installera Karma.

För att komma igång i Webstorm med Karma och Karma-coverage så öppnade jag terminalen och skrev in installationskommandon. När det var färdigt skapade jag en Karma config fil, detta gjorde jag också i terminalen med kommandot ”karma init karma.conf.js”. Här kunde man nu välja inställningar i config filen, t.ex. om man ville använda sig av requireJS eller om Karma automatiskt skulle köra alla testerna igen när man sparar, om man du testa i flera browsers.

Efter att jag hade gjort klart ändringarna i config filen för Karma skapade jag en test-main.js för att läsa in mina testfiler samt köra min config för requireJS som jag även använde i detta fall. Test-main.js är även ansvarig för att Karma kör Jasmine-testerna när requireJS laddat klart.

Här är en bild på hur det ser ut i webstorm när man kör Jasmine-testerna med code coverage.



**Skriva Tester**

Att komma igång och skriva tester i Jasmine är väldigt simpelt, testerna har en väldigt enkelt syntax och väl beskrivande dokumentation. Här följer ett exempel på en kodsnutt:

describe(”beskrivande namn av suiten”), function () {

it(”beskrivande namn på testet”, function () {

expect(true).toBe(true); //Vad som ska testas. I detta fall att True verkligen är True.

});

});

För mer information kring hur man kan skriva tester besök: <http://pivotal.github.io/jasmine/>

**Sammanfattning av arbetet med Jasmine**

Jag använde mig av två olika alternativ för att köra mina tester i Jasmine. Det första alternativet där Jasmine användes som ”Stand Alone upplevde jag som enklare att konfigurera och därav smidigare att komma igång med. I det andra alternativet där jag körde Jasmine med Karma och Karma-coverage stötte jag på några problem med konfigurationen, exempelvis med att läsa in mina templatefiler. Givetvis är det en fördel med code-coverage och för någon som inte har arbetat med Jasmine som verktyg tidigare upplevdes det som betydligt enklare att lägga till code-coverage i Karma än när Jasmine stod som ”Stand Alone”.

Upplevelsen med att utveckla testdrivet var helt ny för mig, Därför var det väldigt behagligt att få använda ett verktyg som Jasmine då jag snabbt fick förståelse för hur testerna skulle göras. Jasmine är enkelt att använda tack vare att dokumentationen för hur man ska gå till väga är väldigt välskriven och att syntax är så enkel. Detta gjorde det lätt att komma igång och börja laborera med Jasmine vilket blev en bra första upplevelse med att arbeta testdrivet.