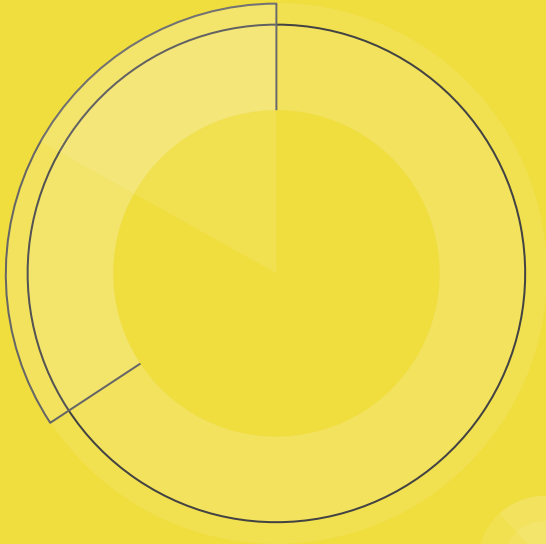
Tests



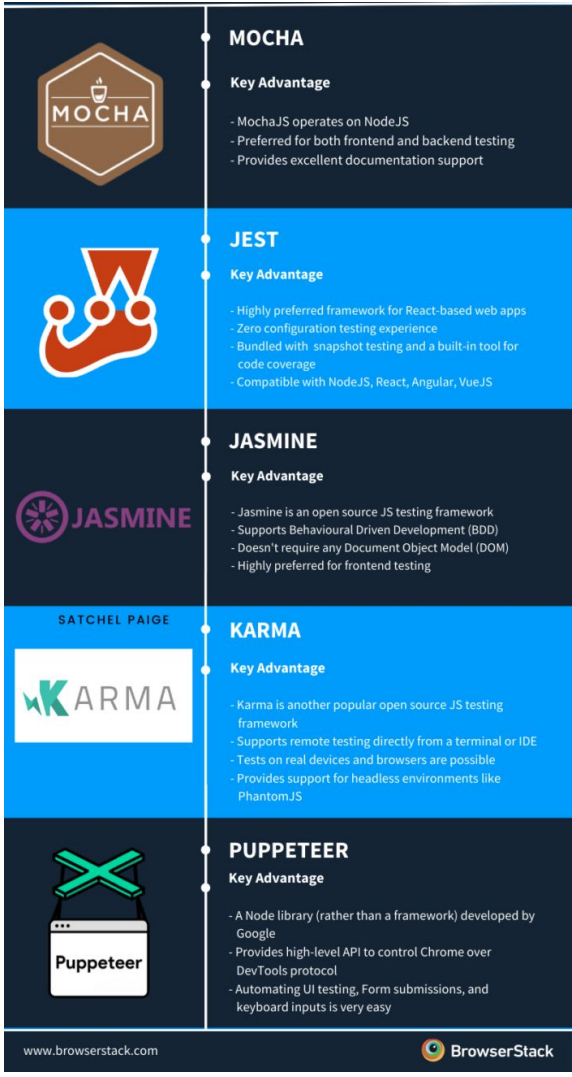
https://es.javascript.info/testing-mocha
https://www.cloudbees.com/blog/mocha-js-chai-sinon-frontend-javascript-code-testing-tutorial
https://www.testim.io/blog/getesting-with-mocha-and-chai/

Motivacions

- Fins ara, per a provar la web, la posem en un servidor, entrem i provem les coses.
- És molt complicat provar-ho tot totes les vegades.
- Si modifiquem alguna cosa, pot tenir efectes col·laterals que són difícils de predir.
- Necessitem automatitzar els tests.
- Hi ha moltes estratègies:
  - Fer tests de coses concretes sense un pla.
  - Test Driven Development (TDD)
  - Behaviour Driven Development (BDD)
  - Acceptance Test Driven Development (ATDD)

Frameworks de tests

- Jasmine
- Mocha + Chai
- Jest
- Selenium



Instal·lar Jasmine

- En aplicacions JS “vanilla” sense mòduls: `jasmine-browser-runner` Amb npm
- En aplicacions JS “Vanilla” amb mòduls: **Jasmine standalone.**
- En aplicacions JS “Vanilla” amb Webpack: Extensió `webpack-karma-jasmine`
- Angular ja el té incorporat per defecte.
- En la versió standalone hi ha un .html d’exemple que podem mantindre.

Describe .. it

- Cada describe és per a comprovar una especificació (spec) i engloba test relacionats.
- It cadascun dels tests

```
describe('Array', function() {
 describe('#indexOf()', function() {
   it('should return -1 when the value is not present', function() {
     expect([1, 2, 3].indexOf(4)).toBe(-1);
   });
 });
});
```

Exemple:


```
describe("pow", function() {
 it("Quadrat de 2 és 4", function() {
   expect(pow(2, 2)).toBe(4);
 });
 it("eleva a la n-ésima potencia", function() {
   expect(pow(2, 4)).toBe(16);
 });
});
```

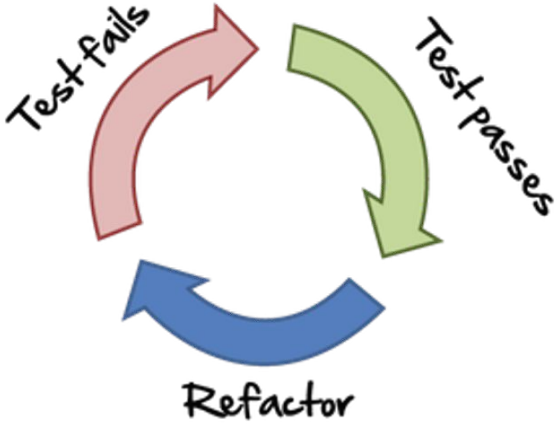
```
function pow(a,b){
   return a * a;
}
```

Test Driven Development

- Part de la metodologia Extreme Programming
- Consisteix bàsicament en fer proves unitàries.
- Tres lleis:
  - No escriuràs codi de producció sense abans escriure un test que falle.
  - No escriuràs més d'un test unitari suficient per a fallar (i no compilar és fallar).
  - No escriuràs més codi de l'necessari per fer passar el test.

Red - Green - Refactor

- Red: Escriure un test que falle, és a dir, hem de fer el test abans d'escriure la implementació. (Se solen fer test unitaris o d'integració).
- Green: S'implementa el mínim codi necessari perquè el test passe.
- Refactor: Una vegada passa, cal veure si es pot millorar
- Una vegada que hem tancat el cicle, comencem de nou amb el següent requisit.
- Tal vegada en cada cicle es poden redefinir o afegir requisits



Exemple de Red -Green

Anem a fer una funció que retorne en n-esim element de la successió de Fibonacci.

n	0	1	2	3	4	5	6	7	8	9	10	11	12
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144

npm, Mocha i Webpack

- No té molt de sentit deixar els tests en una web en producció.
- Els fitxers que no necessiten DOM es poden probar en el CLI.
- Hi ha un pluguin https://github.com/privatenumber/instant-mocha
  - Primer empaqueta el JS i després llança els tests.
  - El resultat ix per la terminal
- En cas de voler veure els tests al navegador i provar les funcions del DOM, necessitem adaptar el html per afegir els tests.
- Finalment, en la posada en producció, es pot llevar.

```
import  'mocha/mocha.css';
import mocha from "mocha/mocha-es2018";
import chai from  'chai';
```

Jasmine

Instal·lació

- Per a Javascript “Vanilla” es reconama descarregar la versió **Standalone.**
- Al descomprimir tenim uns tests d’exemple. Podem aprofitar el SpecRunner.html.
- Es recomana separar els fitxers de text del codi del programa.

Mocha + Chai

Preparar l'entorn

- En aquest cas anem a descarregar en npm, podria ser un CDN o descarregar el .js i .css
- Instal·lem Mocha (pot ser en global o com a dependència de desenvolupament d'un projecte)
- Podem utilitzar el assert de Nodejs o Chai, que és més avançat. (En cas de frontend necessitem precís chai)

```
npm install --save-dev mocha

npm install --save-dev chai
```

- **La majoria dels tutorials que trobareu es refereixen a executar mocha per provar aplicacions de nodejs, no de frontend.**
- **https://github.com/xxjcaxx/dwec-2022/tree/master/tests**

Fer una web de tests


```
<head>
  ...
   <link rel="stylesheet" href="./node_modules/mocha/mocha.css">
   <script src="./node_modules/mocha/mocha.js"></script>
   <script src="./node_modules/chai/chai.js"></script>
</head>
<body>
   <div id="mocha"></div>
   <script class="mocha-init">mocha.setup('bdd'); mocha.checkLeaks();</script>
   <script src="scripts.js"></script>
   <script src="test.js"></script>
   <script class="mocha-exec">mocha.run(); </script>
</body>
```

Chai

- Chai permet fer asercions en l'estil TDD (Test Driven Development) i BDD (Behavior Driven Development)
- Les de BDD són Expect/Should
- Les de TDD són els Asserts

```
describe("pow", function() {
 it("Quadrat de 2 és 4 en expect", function() {
   expect(pow(2, 2)).to.deep.equal(4);
 });
 it("eleva a la n-ésima potencia en expect", function() {
   expect(pow(2, 4)).to.deep.equal(16);
 });
 it("Quadrat de 2 és 4 en assert", function() {
   assert.equal(pow(2, 2), 4);
 });
 it("eleva a la n-ésima potencia en assert", function() {
   assert.equal(pow(2, 4), 16);
 });
});
```

Hooks

- Permeten executar comandament before(), after(), beforeEach(), i afterEach() dels tests.
- Si volem inicialitzar una variable, buidar alguna base de dades... abans o després de fer els tests

```
describe('hooks', function() {
  before(function() {
    // runs once before the first test in this block
  });
  after(function() {
    // runs once after the last test in this block
  });
  beforeEach(function() {
    // runs before each test in this block
  });
  afterEach(function() {
    // runs after each test in this block
  });
  // test cases
});
```