Custom Chai Matchers

RoleModel

Setup—Point Class

```
1 class Point {
2   constructor(x, y) {}
3
4   x() {}
5
6   y() {}
7
8   distanceTo(other) {}
9
10  toString() {}
11 }
```

RoleModel

Goal

```
1 expect(point).to.be.locatedAt(x, y)
2 expect(point).to.be.locatedAt.origin
3
4 // Should give a decent failure message for non-points
5 expect(notAPoint).to.be.locatedAt.origin
6
7 expect(p1).to.be.atDistanceOf(distance).inches.from(p2)
8 expect(p1).to.be.atDistanceOf(distance).feet.from(p2)
```

Setup—Test Boilerplate

```
1 // point-matchers.js
2 function pointMatchers(chai, u  ls) {
3   // Define matchers here
4 }
```

Where we'll focus

```
1 // test.js
2 chai.use(pointMatchers);
3 // The matchers are now available in your tests
```

.to.be.locatedAt(x, y)

```
1 chai.Assertion.addMethod('locatedAt', function(x, y) {
2   const object = utils.flag(this, 'object');
3   this.assert(
4     object.x() === x && object.y() === y,
5     `${object} should be located at (${x}, ${y})`,
6     `${object} should not be located at (${x}, ${y})`,
7   );
8 }
```

RoleModel

utils.flag

- Get "actual" value—utils.flag(this, 'object')
- Pass data between matchers

RoleModel

.to.be.locatedAt.origin

```
1 chai.Assertion.addProperty('origin', function() {
2   const object = utils.flag(this, 'object');
3   this.assert(
4     object.x() === 0 && object.y() === 0,
5     `expected ${object} to be located at origin`,
6     `expected ${object} to not be located at origin`
7   );
8 });
```

RoleModel

Non-point Actuals

```
1 chai.Assertion.addChainableMethod(
2   'locatedAt',
3
4     // Runs when matcher is called with parentheses
5     function methodBehavior(x, y) {}, // Same as before
6
7     // Runs first, with or without parentheses
8     function propertyBehavior() {
9       const object = utils.flag(this, 'object');
10      new chai.Assertion(object).instanceOf(Point);
11    }
12 );
```

RoleModel

Results

```
1 expect(point).to.be.locatedAt(x, y);
2 expect(point).to.not.be.locatedAt(x, y);
3
4 expect(point).to.be.locatedAt.origin;
5 expect(point).to.not.be.locatedAt.origin;
6
7 // Gives a reasonable failure message
8 expect(notAPoint).to.be.locatedAt(x, y);
```

But Wait, There's More!

RoleModel

What's Left

```
1 expect(p1).to.be.atDistanceOf(distance).inches.from(p2)
2
3 expect(p1).to.be.atDistanceOf(distance).feet.from(p2)
```

atDistanceOf(d).<unit>.from(p2)

```
1 chai.Assertion.addMethod(
2   'atDistanceOf',
3   function(distance) {
4     utils.flag(this, 'distance', distance);
5   }
6 );
```

Flags

object distance

RoleModel

atDistanceOf(d).<unit>.from(p2)

```
1 chai.Assertion.addProperty('inches', function() {
2   utils.flag(this, 'distanceMultiplier', 1);
3   utils.flag(this, 'humanDistanceUnit', 'inches');
4 });
5 // Similarly, "inch"
6
7 chai.Assertion.addProperty('feet', function() {
8   utils.flag(this, 'distanceMultiplier', 12);
9   utils.flag(this, 'humanDistanceUnit', 'feet');
10 });
11 // Similarly, "foot"
```

Flags

object distance distanceMultiplier humanDistanceUnit

atDistanceOf(d).<unit>.from(p2)

```
1 chai.Assertion.addMethod('from', function(target) {
2   const object = utils.flag(this, 'object'),
3     multiplier = utils.flag(this, 'distanceMultiplier'),
4     humanUnit = utils.flag(this, 'humanDistanceUnit'),
5     expectedDistance = utils.flag(this, 'distance'),
6     actualDistance = object.distanceTo(target);
7
8   this.assert(
9     actualDistance === expectedDistance * multiplier,
10    // failure messages
11  );
12 });
```

Flags

object · distance · distanceMultiplier · humanDistanceUnit

RoleModel

Summary

```
1 expect(point).to.be.locatedAt(x, y)
2 expect(point).to.be.locatedAt.origin
3
4 // Gives a reasonable failure message for non-points
5 expect(notAPoint).to.be.locatedAt.origin
6
7 expect(p1).to.be.atDistanceOf(distance).inches.from(p2)
8
9 expect(p1).to.be.atDistanceOf(distance).feet.from(p2)
```

Questions/Comments

RoleModel

Resources

- Chai documentation
  - Plugins
    - http://chaijs.com/api/plugins/
  - Built-in matchers
    - http://chaijs.com/api/bdd/

RoleModel