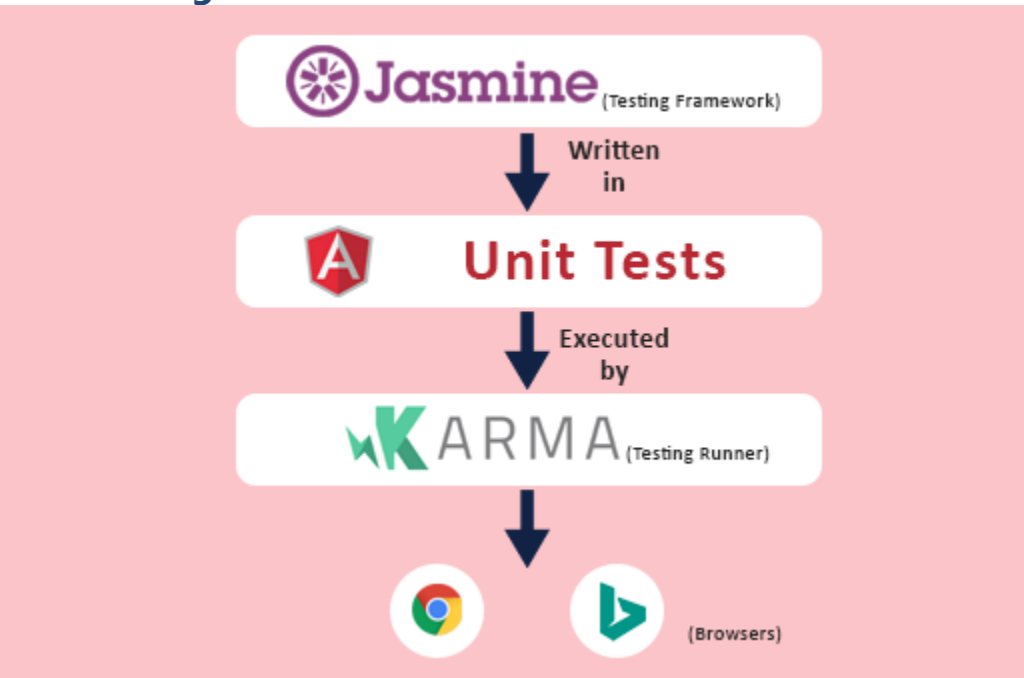
Unit Testing with Jasmine and Karma



Jasmine and Karma

Tool	Purpose
Jasmine	A testing **framework** used to write and structure your unit tests. Provides functions like **describe(), it(), expect().**
Karma	A **test runner** that runs your Jasmine tests in real browsers (like Chrome) and reports results.

Jasmine: Writing Tests

Key Jasmine Functions

- **describe()** → Groups related tests.
- **it()** → Defines a single test case.
- **expect()** → Asserts expected behavior.

Example

```
describe('MathService', () => {
  it('should add numbers', () => {
    const result = 2 + 3;
    expect(result).toBe(5);
  });
});
```

Karma: Running Tests

Karma:

- Loads app in a browser
- Executes Jasmine test cases
- Reports results in CLI and browser
- Watches your files for changes

Karma Config File (karma.conf.js)

Angular CLI generates this automatically.

Important parts:

```
frameworks: ['jasmine'],
browsers: ['Chrome'],
reporters: ['progress'],
```

How to Run Tests

From Angular CLI project:

ng test

This will:

- Use **Karma** to launch a browser

- Run all **Jasmine** tests
- Show test results live

Typical Workflow in Angular

1. Write tests using **Jasmine**
2. Run them with ng test (uses **Karma**)
3. Fix code or tests as needed
4. Repeat until all tests pass

1.Import Dependencies

```
import { TestBed } from '@angular/core/testing';
import { MathService } from './math.service';
```

- **TestBed:** Angular's testing utility that sets up and configures the testing environment.
- **MathService:** The service we're going to test.

---

2. Start the Test Suite

```
describe('MathService', () => {
```

- **describe(...):** A Jasmine function to group related tests.
- "**MathService":** A label for this group of tests.

---

3. Declare Service Variable

```
  let service: MathService;
```

- This declares a variable to hold the instance of **MathService** used in each test.

---

4. Configure Test Environment

```
beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(MathService);
  });
```

- **beforeEach(...):** Runs **before each individual test** in the suite.
- **TestBed.configureTestingModule({}):** Sets up the Angular testing module.
- **TestBed.inject(MathService):** Gets an instance of the MathService to use in tests.

---

5. Individual Unit Tests

Each **it(...)** block is a **unit test case**.

Test: Service Creation

```
  it('should be created', () => {
    expect(service).toBeTruthy();
  });
```

- Checks if the service was created successfully.

Test: add()

```
it('should add two numbers', () => {
    expect(service.add(2, 3)).toBe(5);
  })
```

- **Calls add(2, 3)**
- **Expects the result to be 5.**

Test: subtract()

```
  it('should subtract two numbers', () => {
    expect(service.subtract(5, 2)).toBe(3);
  });
```

- **Calls subtract(5, 2)**
- **Expects the result to be 3.**

Test: multiply()

```
it('should multiply two numbers', () => {
    expect(service.multiply(4, 3)).toBe(12);
  });
```

- **Calls multiply(4, 3)**

- **Expects the result to be 12.**

Test: divide()

```
it('should divide two numbers', () => {
   expect(service.divide(10, 2)).toBe(5);
 });
```

- **Calls divide(10, 2)**
- **Expects the result to be 5.**

Division by Zero

```
 it('should throw error when dividing by zero', () => {
    expect(() => service.divide(10, 0)).toThrowError('Cannot divide by
zero');
  });
```

- **Tests if dividing by zero throws the expected error.**

Summary

This file ensures:

- MathService is properly created
- Each math operation (add, subtract, multiply, divide) works
- Division by zero throws an error
