Student CRM — Step-by-Step Testing with Karma & Jasmine

This guide walks students through configuring, writing, and running unit tests for the Student CRM project using Angular's default Karma runner and Jasmine test framework.

> **Prerequisites:** Completed Student CRM project, Node 18+, Angular CLI 20, Google Chrome installed (Karma default launcher).

1. Verify test tooling

1. Open `angular.json` and confirm the `test` target uses `@angular-devkit/build-angular:karma`.
2. Check `karma.conf.js` for the Chrome launcher and Jasmine framework (these are scaffolded by the CLI).
3. Ensure `src/polyfills.ts` is unchanged; tests rely on the same polyfills as the app.

2. Smoke-test the runner

1. Install dependencies if needed: `npm install`.
2. Run Karma once in watch mode: `npm test` or `ng test`.
3. Wait for Chrome to start and the CLI to report *0 failed*. You should see the default specs generated by the CLI (AppComponent, etc.).
4. Stop the watch run with `Ctrl + C`.

3. Organize testing strategy

Create a simple checklist for coverage:

- **Services:** StudentService (adds, toggles, list copy) and LoggerService (logs).
- **Components:** AddStudentComponent (emits form data), StudentListComponent (passes events), StudentCardComponent (emits toggle), HomeComponent (integrates StudentService).
- **Pipes/Utilities:** none in this project.

4. Write StudentService tests

1. Create `src/app/core/student.service.spec.ts` (or edit the generated file if it already exists) with:

```
import { StudentService } from './student.service';

describe('StudentService', () => {
  let service: StudentService;

  beforeEach(() => {
    service = new StudentService();
  });

  it('should list initial students', () => {
    expect(service.list().length).toBe(2);
  });

  it('should add a student with incremental id', () => {
    const result = service.add({ name: 'Charlie', track: 'DevOps', active: true });
    const added = result.find(s => s.name === 'Charlie');
    expect(added?.id).toBeGreaterThan(2);
    expect(added?.active).toBeTrue();
  });

  it('should toggle active status immutably', () => {
    const [first] = service.list();
    const toggled = service.toggleActive(first.id);
    const updated = toggled.find(s => s.id === first.id);
    expect(updated?.active).toBe(!first.active);
    // ensure a new array was returned
    expect(toggled).not.toBe(service['store']);
  });
});
```

5. Test LoggerService

1. In `src/app/core/logger.service.spec.ts` replace contents with a spy-based check:

```
import { LoggerService } from './logger.service';

describe('LoggerService', () => {
  it('should log to console with prefix', () => {
    const svc = new LoggerService();
    spyOn(console, 'log');
```

```
    svc.log('Hello');

    expect(console.log).toHaveBeenCalledWith('[LOG] Hello');
  });
});
```

6. Component tests — AddStudentComponent

1. Edit `src/app/students/add-student/add-student.component.spec.ts`:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormsModule } from '@angular/forms';
import { AddStudentComponent } from './add-student';

describe('AddStudentComponent', () => {
  let component: AddStudentComponent;
  let fixture: ComponentFixture<AddStudentComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [FormsModule, AddStudentComponent]
    }).compileComponents();

    fixture = TestBed.createComponent(AddStudentComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should emit create event on submit', () => {
    spyOn(component.create, 'emit');
    component.name = 'Dana';
    component.track = 'Data';
    component.active = false;

    component.submit();

    expect(component.create.emit).toHaveBeenCalledWith({
      name: 'Dana',
      track: 'Data',
      active: false
    });
    expect(component.name).toBe('');
    expect(component.active).toBeTrue();
  });

  it('should ignore blank names', () => {
    spyOn(component.create, 'emit');
    component.name = '   ';
```

```
    component.submit();
    expect(component.create.emit).not.toHaveBeenCalled();
  });
});
```

7. Component tests — StudentCardComponent

1. Update `src/app/students/student-card/student-card.component.spec.ts`:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { StudentCardComponent } from './student-card';
import { Student } from '../../core/student';

describe('StudentCardComponent', () => {
  let component: StudentCardComponent;
  let fixture: ComponentFixture<StudentCardComponent>;

  const mockStudent: Student = {
    id: 42,
    name: 'Eve',
    track: 'Front-end',
    active: true
  };

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [StudentCardComponent]
    }).compileComponents();

    fixture = TestBed.createComponent(StudentCardComponent);
    component = fixture.componentInstance;
    component.student = mockStudent;
    fixture.detectChanges();
  });

  it('should render the student name', () => {
    const nameEl: HTMLElement = fixture.nativeElement.querySelector('.title');
    expect(nameEl?.textContent).toContain('Eve');
  });

  it('should emit toggle event with id', () => {
    spyOn(component.toggle, 'emit');
    const button: HTMLButtonElement = fixture.nativeElement.querySelector('button');
    button.click();
    expect(component.toggle.emit).toHaveBeenCalledWith(42);
  });
});
```

8. Component tests — StudentListComponent

1. Edit `src/app/students/student-list/student-list.component.spec.ts` to check bindings:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { StudentListComponent } from './student-list';
import { StudentCardComponent } from '../student-card/student-card';
import { Student } from '../../core/student';
import { Component } from '@angular/core';

@Component({
  template: `
    <app-student-list
      [students]="students"
      (toggleActive)="onToggle($event)">
    </app-student-list>
  `
})
class HostComponent {
  students: Student[] = [
    { id: 1, name: 'Alice', track: 'Front-end', active: true },
    { id: 2, name: 'Bob', track: 'Data', active: false }
  ];
  lastToggle?: number;
  onToggle(id: number) { this.lastToggle = id; }
}

describe('StudentListComponent', () => {
  let fixture: ComponentFixture<HostComponent>;
  let host: HostComponent;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [HostComponent],
      imports: [StudentListComponent, StudentCardComponent]
    }).compileComponents();

    fixture = TestBed.createComponent(HostComponent);
    host = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should render a card for each student', () => {
    const cards = fixture.nativeElement.querySelectorAll('app-student-card');
    expect(cards.length).toBe(2);
  });

  it('should bubble toggle events', () => {
```

```
    const cardDebug = fixture.debugElement.queryAllNodes(n => n.name === 'app-student-card'
    cardDebug.triggerEventHandler('toggle', 1);
    expect(host.lastToggle).toBe(1);
  });
});
```

9. Component tests — HomeComponent

1. Replace `src/app/home/home.component.spec.ts` with an integration-style spec:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { HomeComponent } from './home';
import { StudentsModule } from '../students/students-module';
import { StudentService } from '../core/student';

describe('HomeComponent', () => {
  let fixture: ComponentFixture<HomeComponent>;
  let component: HomeComponent;
  let service: StudentService;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [StudentsModule],
      declarations: [HomeComponent]
    }).compileComponents();

    fixture = TestBed.createComponent(HomeComponent);
    component = fixture.componentInstance;
    service = TestBed.inject(StudentService);
    fixture.detectChanges();
  });

  it('should load students on init', () => {
    expect(component.students.length).toBeGreaterThan(0);
  });

  it('should add a new student via child event', () => {
    component.onCreate({ name: 'Frank', track: 'DevOps', active: true });
    expect(component.students.some(s => s.name === 'Frank')).toBeTrue();
  });

  it('should toggle student active state', () => {
    const [first] = service.list();
    component.onToggleActive(first.id);
    const updated = component.students.find(s => s.id === first.id);
    expect(updated?.active).toBe(!first.active);
```

```
  });
});
```

10. Run the suite in watch & CI modes

1. Start watch mode during development: `ng test`.
2. For CI, run headless Chrome: `ng test --watch=false --browsers=ChromeHeadless`.
3. Ensure all specs pass (expect “Executed X of X SUCCESS”).

11. Common troubleshooting tips

- If TestBed cannot resolve a component, confirm the component is imported (standalone) or declared in the testing module.
- For template parse errors, add required Angular modules (`FormsModule`, etc.) to `imports`.
- Use `fixture.detectChanges()` after updating component inputs or calling methods that affect bindings.
- Mock service dependencies with spies or simple classes and provide them via `providers`.

Following these steps gives the class a complete, green unit-test suite for the Student CRM application, demonstrating both service testing and component interaction patterns with Karma and Jasmine.