Tabla de casos de prueba para TodoService

ID	Caso de prueba	Acción (Act)	Resultado esperado (Assert)
TC-01	Crear tarea	Llamar a add("Aprender Jest")	Devuelve un Todo con text="Aprender Jest", done=false, y se guarda en localStorage.
TC-02	Alternar tarea	Crear una tarea y llamar a toggle(id)	La propiedad done cambia de false → true y se persiste en localStorage.
TC-03	Eliminar tarea existente	Crear una tarea y llamar a remove(id)	Devuelve true y la lista queda vacía.
TC-04	Eliminar tarea inexistente	Llamar a remove(9999) sin que exista	Devuelve false y la lista no cambia.
TC-05	Limpiar completadas	Crear 2 tareas, marcar una como done, llamar clearCompleted()	Devuelve 1 (eliminadas), la lista contiene solo la tarea pendiente.
TC-06	Cargar desde localStorage	Guardar manualmente un array en localStorage y crear un nuevo TodoService	getAll() devuelve exactamente ese array.
TC-07	Persistencia en localStorage	Crear tarea y verificar localStorage.getItem("todos")	Contenido JSON válido con la tarea recién creada.
TC-08	Llamada a localStorage.setItem	Espiar Storage.prototype.setItem y llamar a add()	El spy debe haberse ejecutado al menos una vez.

1) Idea rápida del ejemplo

Creamos TodoService (TypeScript) con métodos: getAll, add, toggle, remove, clearCompleted, y persistencia en localStorage.

Tests que cubren: creación, persistencia, toggle, eliminación, carga desde localStorage y uso de spies.

> Un "spy" (o espía) **en un test de TS (TypeScript)** es una herramienta de **doble de prueba (test double)** que te permite rastrear el comportamiento de una función o método real, registrando información sobre su uso (como cuántas veces se llamó o con qué argumentos) mientras permite que la implementación original de la función se ejecute normalmente. La diferencia principal con un mock es que el spy envuelve la función real en lugar de reemplazarla por completo, ofreciendo más flexibilidad para verificar interacciones sin modificar el comportamiento de la función

2) Preparar el proyecto (comandos)

```
# carpeta del proyecto
mkdir jest-ts-todos && cd jest-ts-todos
npm init -y

# instalar dependencias de desarrollo
npm install -D jest ts-jest @types/jest typescript

# inicializar configuración básica de ts-jest (opcional, crea jest.config.js)
npx ts-jest config:init
```

Añade en package.json scripts útiles:

```
{
  "scripts": {
    "test": "jest --coverage",
    "test:watch": "jest --watch"
  }
```

```
}
```

3) Configuración TypeScript y Jest

tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2019",
    "module": "CommonJS",
    "lib": ["ES2019", "DOM"],
    "strict": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "skipLibCheck": true,
    "outDir": "dist",
    "sourceMap": true
  },
  "include": ["src"]
}
```

Si npx ts-jest config:init no lo hizo por ti, crea jest.config.ts:

```
import type { Config } from '@jest/types';

const config: Config.InitialOptions = {
  preset: 'ts-jest',
  testEnvironment: 'jsdom',    // usamos jsdom porque estamos en entorno cliente
  roots: ['<rootDir>/src'],
  verbose: true,
  collectCoverage: true,
  collectCoverageFrom: ['src/**/*.ts', '!src/**/*.d.ts'],
  coverageDirectory: 'coverage'
};
```

```
export default config;
```

4) Código:

TodoService: (src/TodoService.ts)

```
// src/TodoService.ts
export type Todo = {
  id: number;
  text: string;
  done: boolean;
  createdAt: number;
};

/**
 * Servicio mínimo para manejar una lista de tareas (solo lógica).
 * - Persiste en localStorage con la clave 'todos' por defecto.
 */
export default class TodoService {
  private todos: Todo[] = [];
  private storageKey = 'todos';

  constructor() {
    this.load();
  }

  getAll(): Todo[] {
    return [...this.todos];
  }

  add(text: string): Todo {
    const now = Date.now();
    const todo: Todo = {
```

```
    id: now + Math.floor(Math.random() * 1000),
    text: text.trim(),
    done: false,
    createdAt: now
  };
  this.todos.push(todo);
  this.save();
  return todo;
}

toggle(id: number): Todo | undefined {
  const t = this.todos.find((x) => x.id === id);
  if (!t) return undefined;
  t.done = !t.done;
  this.save();
  return t;
}

remove(id: number): boolean {
  const idx = this.todos.findIndex((x) => x.id === id);
  if (idx === -1) return false;
  this.todos.splice(idx, 1);
  this.save();
  return true;
}

clearCompleted(): number {
  const before = this.todos.length;
  this.todos = this.todos.filter((t) => !t.done);
  const removed = before - this.todos.length;
  if (removed > 0) this.save();
  return removed;
}
```

```
  private save(): void {
    try {
      window.localStorage.setItem(this.storageKey, JSON.stringify(this.todos));
    } catch {
      // en un entorno restringido localStorage puede fallar; lo ignoramos aquí
    }
  }

  private load(): void {
    try {
      const raw = window.localStorage.getItem(this.storageKey);
      if (!raw) return;
      this.todos = JSON.parse(raw) as Todo[];
    } catch {
      this.todos = [];
    }
  }

  // helper para tests/uso: cambiar clave de almacenamiento
  setStorageKey(k: string) {
    this.storageKey = k;
    this.load();
  }
}
```

5) Tests (src/

tests

/todoService.test.ts)

```
// src/__tests__/todoService.test.ts
import TodoService, { Todo } from '../TodoService';

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

  beforeEach(() => {
    localStorage.clear();        // limpiamos entre tests
    service = new TodoService();
    service.setStorageKey('todos'); // seguridad
  });

  test('add() crea una tarea y la persiste en localStorage', () => {
    const todo = service.add('Aprender Jest con TypeScript');
    expect(todo.id).toBeDefined();
    expect(todo.text).toBe('Aprender Jest con TypeScript');
    expect(todo.done).toBe(false);

    const all = service.getAll();
    expect(all).toHaveLength(1);

    // comprobamos que se haya guardado correctamente
    const raw = localStorage.getItem('todos');
    expect(raw).not.toBeNull();
    const parsed = JSON.parse(raw as string);
    expect(parsed).toHaveLength(1);
    expect(parsed[0].text).toBe('Aprender Jest con TypeScript');
  });

  test('toggle() invierte la propiedad done y persiste', () => {
```

```
  const t = service.add('Tarea toggle');
  expect(t.done).toBe(false);

  const toggled = service.toggle(t.id);
  expect(toggled).toBeDefined();
  expect(toggled?.done).toBe(true);

  const parsed = JSON.parse(localStorage.getItem('todos') as string);
  expect(parsed[0].done).toBe(true);
});

test('remove() elimina y devuelve true', () => {
  const t = service.add('Tarea a eliminar');
  expect(service.getAll()).toHaveLength(1);

  const removed = service.remove(t.id);
  expect(removed).toBe(true);
  expect(service.getAll()).toHaveLength(0);
});

test('clearCompleted() elimina solo las completadas', () => {
  const a = service.add('t1');
  const b = service.add('t2');
  service.toggle(b.id); // marcar b como completada

  const removedCount = service.clearCompleted();
  expect(removedCount).toBe(1);
  expect(service.getAll().length).toBe(1);
  expect(service.getAll()[0].id).toBe(a.id);
});

test('constructor/load() lee desde localStorage', () => {
  const fake: Todo[] = [{ id: 999, text: 'desde storage', done: false, createdAt: 1 }];
  localStorage.setItem('todos', JSON.stringify(fake));
```

```
    const s2 = new TodoService();
    expect(s2.getAll()).toEqual(fake);
  });

  test('save() llama a localStorage.setItem (spy)', () => {
    const spy = jest.spyOn(Storage.prototype, 'setItem');
    service.add('invoke save');
    expect(spy).toHaveBeenCalled();
    spy.mockRestore();
  });
});
```

6) Ejecutar los tests

En la terminal:

```
npm install --save-dev jest-environment-jsdom
npm test
# o para desarrollo:
npm run test:watch
```

Con --coverage se generará carpeta coverage/ con informe.

7) Qué enseñar en clase (guion sugerido)

1. Contexto 5′ — por qué testear: confianza, refactor, regresiones.
2. Explicación 10′ — estructura AAA (Arrange-Act-Assert), describe/test/beforeEach.
3. Live coding 20′ — implementar add() y su test; ejecutar para ver test pasar/fallar.
4. TDD mini — pedir a estudiantes que primero escriban test para toggle.
5. Mocks y spies 10′ — mostrar jest.spyOn(Storage.prototype, 'setItem').
6. Reflexión 10′ — cuándo testear lógica vs DOM; buenas prácticas.

8) Casos de prueba recomendados (tabla corta para alumnos)

- ☐ Añadir tarea -> se guarda y persiste.
- ☐ Alternar done -> cambia estado y persiste.
- ☐ Borrar tarea -> se elimina de la lista.
- ☐ Borrar completadas -> elimina solo completadas.
- ☐ Cargar desde localStorage al inicializar.
- ☐ Comprobar que save() llama localStorage.setItem (spy).

9) Extensiones / ejercicios para alumnos

- Ejercicio 1: Añadir validación para que no se creen tareas vacías; escribir tests.
- Ejercicio 2: Añadir método edit(id, newText) con tests.
- Ejercicio 3: Integrar con un componente DOM simple y probar interacción con @testing-library/dom (clase extra).
- Ejercicio 4: Convertir a React + testing-library y enseñar tests de integración.

10) Consejos y pitfalls para la pizarra

- Usa jsdom como testEnvironment para simular window/localStorage.
- Resetea localStorage en beforeEach para evitar dependencia entre tests.
- Prefiere comprobar estado del servicio (retornos/arrays) en vez de replicar internals.
- Usa spies con mockRestore() para no romper otros tests.
- Evitar usar tiempos reales (Date.now()) en comparaciones exactas; si necesitas igualdad exacta, usa jest.useFakeTimers() o sacarlo del test.