Servicio Nacional de Aprendizaje(SENA)

Análisis y Desarrollo de Software. Ficha (2977355)

Evidencia de conocimiento: GA9-220501096-AA1-EV01 taller sobre codificación de módulos del software.

Jesus Daniel Plata Castro

Cartagena - Colombia

19/12/2025

Introducción

La actividad consiste en implementar y evaluar una aplicación web de lista de tareas (To-Do List) mediante diferentes tipos de pruebas de software, dentro del marco de un taller de codificación de módulos. A partir de los requisitos del sistema, se pide investigar tipos de pruebas, seleccionar las más adecuadas para el proyecto, instalar herramientas (Jest y Cypress) y documentar los resultados obtenidos durante la ejecución de los casos de prueba.

Desarrollo

Tipos de pruebas de software

- Pruebas unitarias: verifican unidades pequeñas de código (funciones, clases) para detectar errores de forma temprana y reducir el costo de corrección.
- Pruebas de integración: validan la interacción entre módulos o servicios para asegurar que funcionen correctamente en conjunto.
- Pruebas de sistema: evalúan el sistema completo contra los requisitos funcionales y no funcionales (rendimiento, seguridad, usabilidad).
- Pruebas de aceptación: confirman con el cliente o usuario que el software satisface las necesidades del negocio antes de pasar a producción.
- Pruebas de rendimiento: miden velocidad, estabilidad y capacidad de respuesta bajo distintas cargas de usuarios.
- Pruebas de regresión: se ejecutan tras cambios o correcciones para comprobar que no aparezcan defectos en funcionalidades ya probadas.

Características y beneficios

- Las pruebas unitarias son rápidas, automatizables y favorecen el código mantenible, permitiendo detectar defectos justo después de programar.
- Las pruebas de integración y sistema aumentan la calidad general al encontrar fallos en la comunicación entre componentes y en escenarios de uso reales.
- Las pruebas de aceptación incrementan la satisfacción del usuario y reducen retrabajos porque validan los requisitos de negocio.
- Las pruebas de rendimiento ayudan a identificar cuellos de botella y a

garantizar que la aplicación soporte el volumen de usuarios esperado.
- Las pruebas de regresión aseguran la estabilidad del producto en ciclos ágiles y continuos, manteniendo la confiabilidad versión tras versión.

Pruebas que mejor se adaptan al proyecto

- La mayor parte del esfuerzo debe estar en pruebas unitarias porque el proyecto tiene lógica clara y modular (clases Task, TaskList, TaskRepository, hooks y componentes React), lo que facilita automatizar muchas validaciones con bajo costo.
- Se recomienda complementar con algunas pruebas de integración para asegurar la correcta comunicación entre LocalStorage, el estado React y el renderizado de la lista de tareas, especialmente por el requisito de persistencia offline.
- Al menos 2 o 3 casos de prueba end-to-end en el navegador validarán los flujos clave: crear tarea válida, bloqueo de tarea vacía, marcar/desmarcar completada, eliminar tarea y mantener datos tras recargar la página.

Tabla: relación entre requisitos y tipos de pruebas

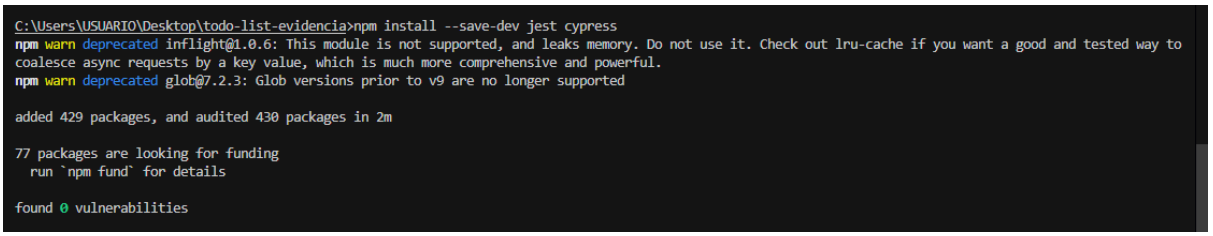
Requisito / Historia	Tipo de prueba principal	Justificación
RF Crear tarea, HU-01	Unitaria + E2E	Se prueba la función addTask y la validación de longitud, y luego el flujo completo de creación desde la UI.
RF Marcar completada, HU-02	Unitaria + Integración	Se valida toggleStatus en Task y su reflejo visual en TodoItem/Statistics.
RF Eliminar tarea, HU-03	Unitaria + E2E	Se prueba removeTask y se confirma en navegador que la tarea desaparece y no regresa al recargar.

RF Visualizar lista, HU-04	Integración + E2E	Se verifica carga desde LocalStorage y renderizado correcto de tareas y estados.
RNF Persistencia offline	Integración	Se comprueba que TaskRepository use bien LocalStorage para guardar y recuperar datos.
RNF Rendimiento < 100 ms	Rendimiento básico	Se pueden hacer mediciones simples o scripts de carga ligeros para asegurar tiempos de respuesta aceptables.

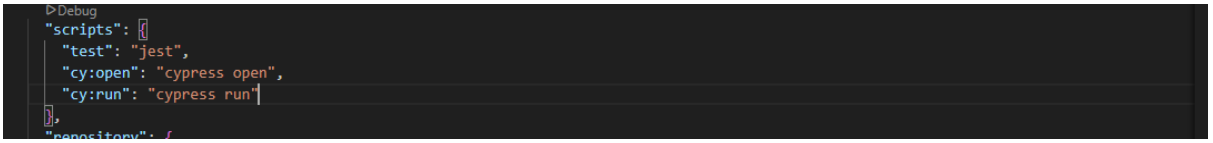
Pruebas

1. Preparar el proyecto para pruebas

- Instala dependencias: npm init -y, luego npm install --save-dev jest cypress para habilitar ambos frameworks en tu app HTML/CSS/JS.

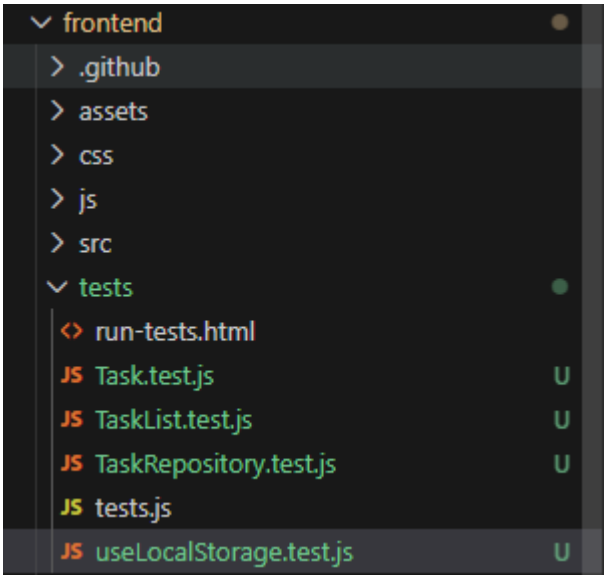


- En package.json agrega scripts: "test": "jest" para unitarias y "cy:open": "cypress open", "cy:run": "cypress run" para abrir o ejecutar pruebas E2E.



2. Pruebas unitarias con Jest (Task, TaskList, TaskRepository, useLocalStorage)

- Crea una carpeta __tests__/ y archivos como Task.test.js, TaskList.test.js, TaskRepository.test.js, useLocalStorage.test.js.



- En Task.test.js instancia la clase y valida métodos:
  - markAsCompleted() cambia isCompleted a true.
  - updateDescription() rechaza textos vacíos o > 200 caracteres (esperas error) y acepta válidos (esperas cambio).

```
// frontend/tests/Task.test.js
import { Task } from '../src/js/Task.js';

describe('Task', () => {
  test('markAsCompleted cambia isCompleted a true', () => {
    const task = new Task(1, 'Tarea de prueba');

    task.markAsCompleted();

    expect(task.isCompleted).toBe(true);
  });

  test('markAsPending cambia isCompleted a false', () => {
    const task = new Task(1, 'Tarea de prueba');
    task.markAsCompleted();

    task.markAsPending();

    expect(task.isCompleted).toBe(false);
  });

  test('updateDescription actualiza con texto válido', () => {
    const task = new Task(1, 'Tarea inicial');

    task.updateDescription('Nueva descripción válida');

    expect(task.description).toBe('Nueva descripción válida');
  });

  test('lanza error si descripción está vacía', () => {
    const task = new Task(1, 'Tarea inicial');

    expect(() => {
      task.updateDescription('');
    }).toThrow();
  });

  test('lanza error si descripción > 200 caracteres', () => {
    const task = new Task(1, 'Tarea inicial');
```

- En TaskList.test.js prueba:
  - addTask() aumenta el total y contiene la tarea.
  - removeTask(id) la elimina.
  - getStatistics() devuelve totales coherentes (pendientes/completadas).

```
// frontend/tests/TaskList.test.js
import { Task } from '../src/js/Task.js';
import { TaskList } from '../src/js/TaskList.js';

describe('TaskList', () => {
  test('addTask agrega una tarea a la lista', () => {
    const list = new TaskList();
    const task = new Task(1, 'Tarea de prueba');

    list.addTask(task);

    expect(list.tasks.length).toBe(1);
    expect(list.tasks[0]).toBe(task);
  });

  test('removeTask elimina una tarea por id', () => {
    const list = new TaskList();
    const task1 = new Task(1, 'Tarea 1');
    const task2 = new Task(2, 'Tarea 2');
    list.addTask(task1);
    list.addTask(task2);

    list.removeTask(1);

    expect(list.tasks.length).toBe(1);
    expect(list.tasks[0].id).toBe(2);
  });

  test('getPendingTasks devuelve solo pendientes', () => {
    const list = new TaskList();
    const t1 = new Task(1, 'Pendiente 1');
    const t2 = new Task(2, 'Pendiente 2');
    const t3 = new Task(3, 'Completada');
    t3.markAsCompleted();
    list.addTask(t1);
    list.addTask(t2);
    list.addTask(t3);

    const pending = list.getPendingTasks();
```

- En TaskRepository.test.js usa localStorage simulado (Jest lo provee o se crea un mock) para comprobar que save() guarda JSON y load() reconstruye correctamente la lista.

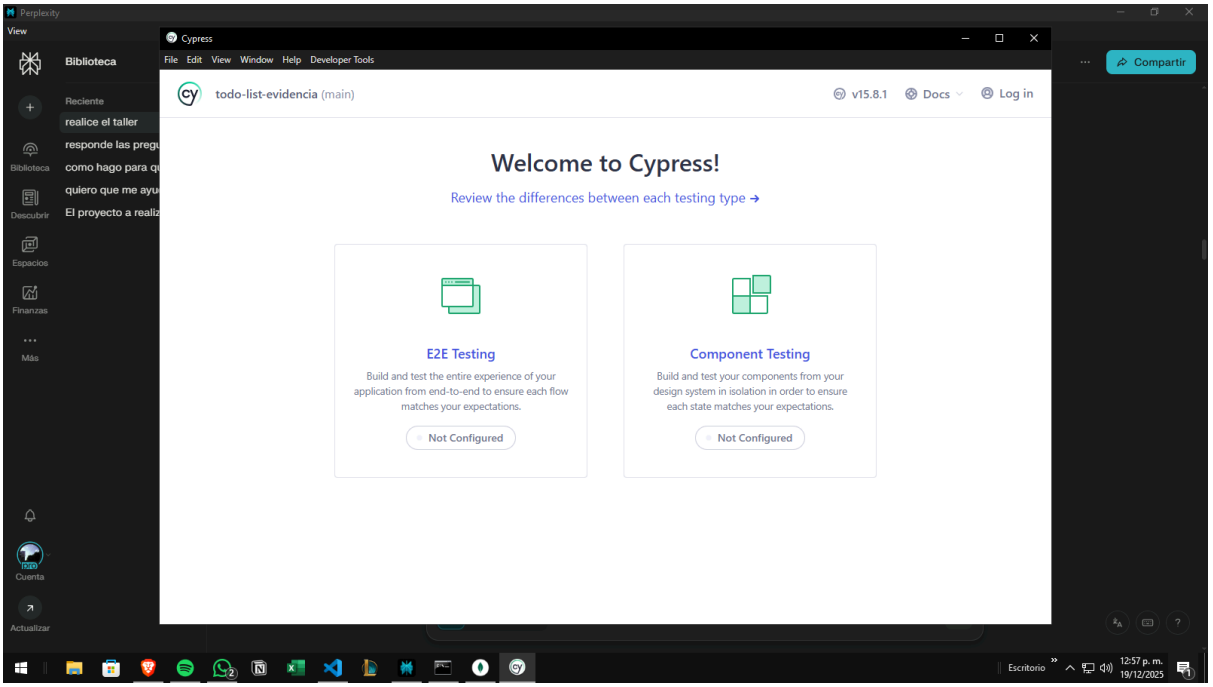
```
// frontend/tests/TaskRepository.test.js
import { Task } from '../src/js/Task.js';
import { TaskList } from '../src/js/TaskList.js';
import { TaskRepository } from '../src/js/TaskRepository.js';

beforeEach(() => {
  // Mock simple de localStorage
  const store = {};
  global.localStorage = {
    getItem: key => store[key] || null,
    setItem: (key, value) => { store[key] = String(value); },
    removeItem: key => { delete store[key]; },
    clear: () => { Object.keys(store).forEach(k => delete store[k]); }
  };
});

describe('TaskRepository', () => {
  test('save guarda la lista en localStorage', () => {
    const repo = new TaskRepository('tasks');
    const list = new TaskList();
    list.addTask(new Task(1, 'Tarea 1'));

    repo.save(list);

    const raw = localStorage.getItem('tasks');
    expect(raw).not.toBeNull();
  });

  test('load reconstruye la lista desde localStorage', () => {
    const repo = new TaskRepository('tasks');
    const list = new TaskList();
    list.addTask(new Task(1, 'Tarea 1'));
    repo.save(list);

    const loadedTasks = repo.load();

    expect(Array.isArray(loadedTasks)).toBe(true);
    expect(loadedTasks.length).toBe(1);
    expect(loadedTasks[0].description).toBe('Tarea 1');
```

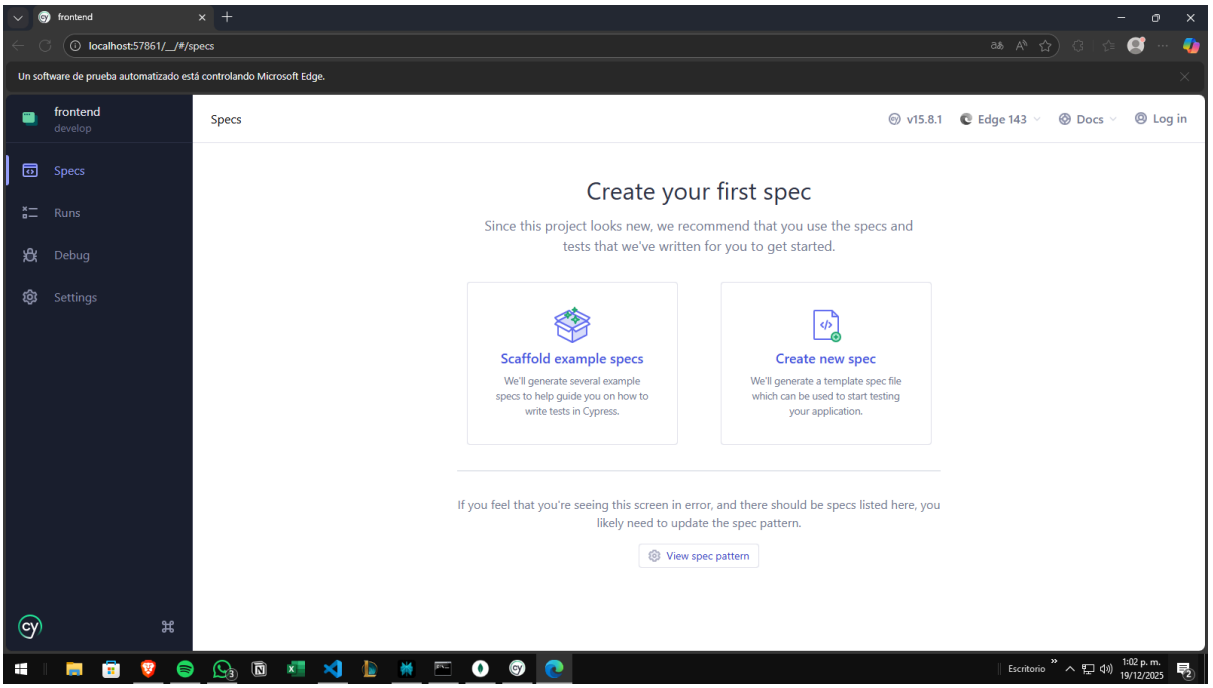
- En useLocalStorage.test.js prueba que:
    - Devuelva el valor inicial si no hay nada en LocalStorage.
    - Actualice LocalStorage cuando llamas al setter del hook.

```jsx
// frontend/src/js/useLocalStorage.js (ejemplo)
import { useState, useEffect } from 'react';

export function useLocalStorage(key, initialValue) {
  const [value, setValue] = useState(() => {
    const stored = window.localStorage.getItem(key);
    return stored ? JSON.parse(stored) : initialValue;
  });

  useEffect(() => {
    window.localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}
```

3. Pruebas end-to-end con Cypress (CP-01 a CP-04)

- Ejecuta npx cypress open una vez; Cypress creará la carpeta cypress/e2e donde escribirás archivos como todo.cy.js.





CP-01 – Crear tarea

- cy.visit('http://127.0.0.1:5500/frontend/auth.html'): Abre la pantalla de autenticación.
- cy.get('[data-cy=login-email]').type('usuario@prueba.com'): Escribe el correo del usuario de prueba.
- cy.get('[data-cy=login-password]').type('123456'): Escribe la contraseña.
- cy.get('[data-cy=login-submit]').click(): Envía el formulario y navega a

index.html.

- cy.get('[data-cy=add-button]').click(): Abre el formulario modal para crear una tarea.
- cy.get('[data-cy=task-input]').type('Nueva tarea de prueba') y cy.get('[data-cy=submit]').click(): Ingresa el título y guarda la tarea.
- cy.contains('Nueva tarea de prueba').should('exist'): Verifica que la nueva tarea aparezca en la lista de pendientes.

CP-02 – Marcar tarea como completada

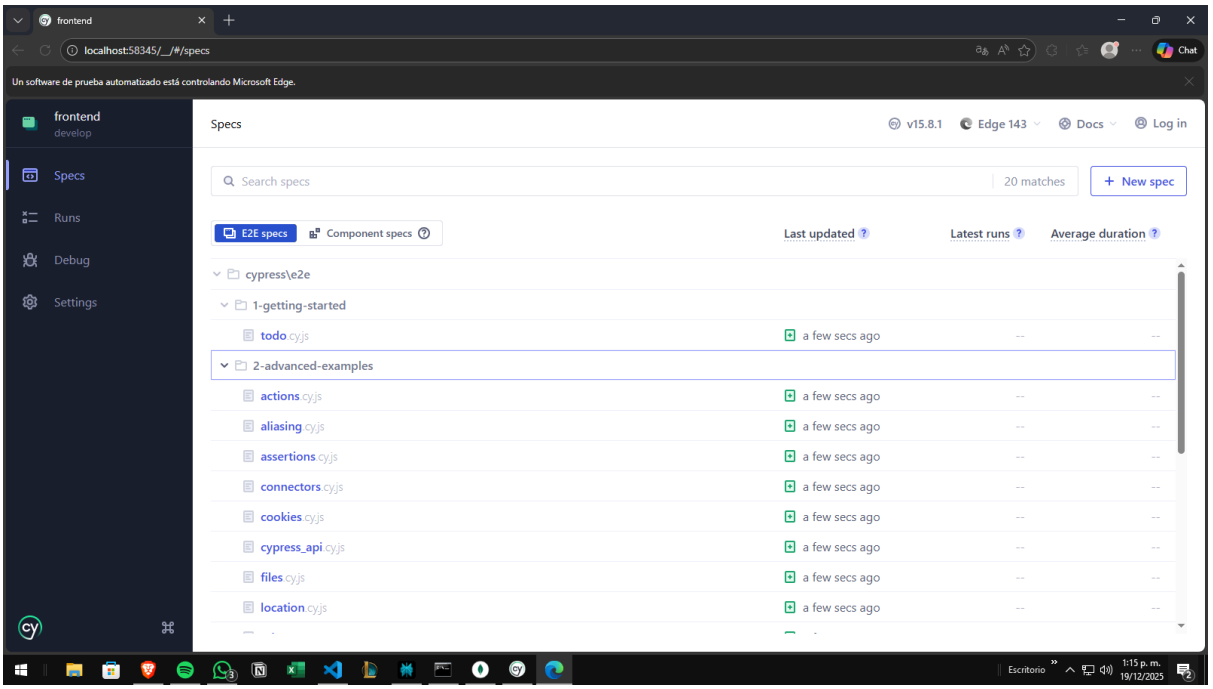
- (Con sesión ya iniciada) cy.get('[data-cy=add-button]').click(): Abre el formulario de nueva tarea.
- cy.get('[data-cy=task-input]').type('Terminar evidencias') y cy.get('[data-cy=submit]').click(): Crea la tarea “Terminar evidencias”.
- cy.contains('Terminar evidencias').parent().find('[data-cy=task-checkbox]').check(): Marca la tarea como completada usando el checkbox.
- cy.contains('Terminar evidencias').parent().find('h3').should('have.class', 'completada'): Comprueba que la tarea se muestre visualmente como completada.

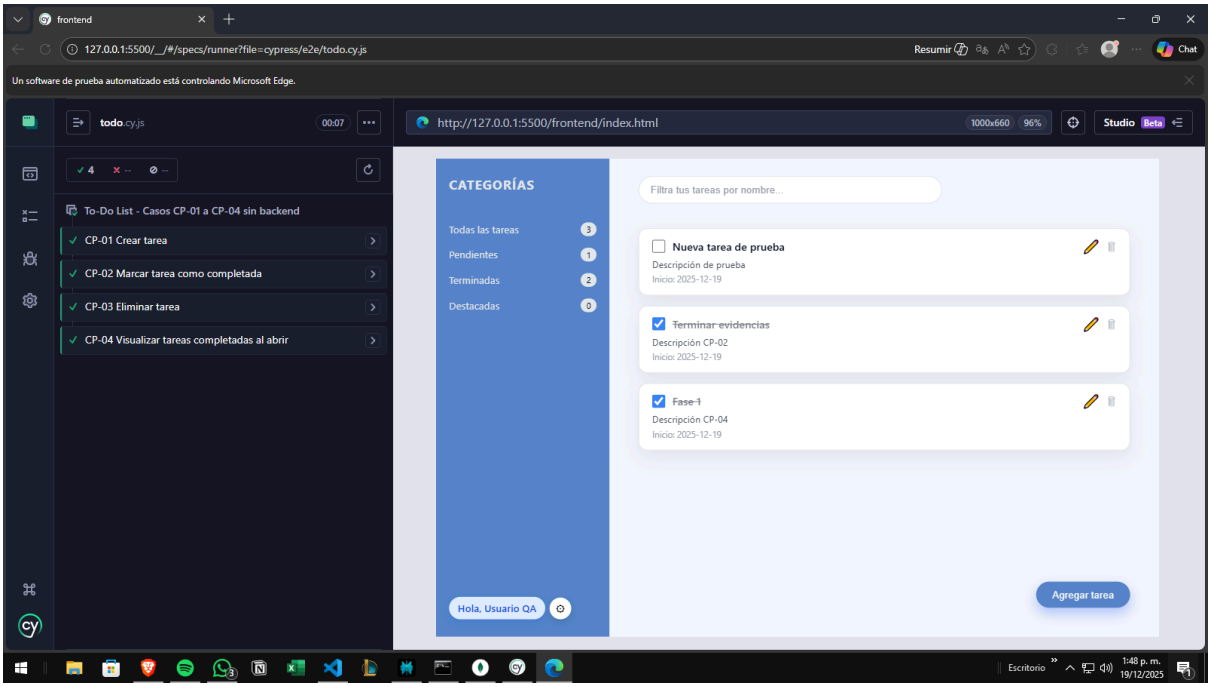
CP-03 – Eliminar tarea

- cy.get('[data-cy=add-button]').click() y cy.get('[data-cy=task-input]').type('Terminar evidencias'): Abre el formulario y escribe el título de la tarea a probar.
- cy.get('[data-cy=submit]').click(): Guarda la tarea en la lista.
- cy.contains('Terminar evidencias').parent().find('[data-cy=delete-button]').click(): Pulsa el icono de eliminar asociado a esa tarea.
- cy.get('[data-cy=delete-confirm]').click(): Confirma la eliminación en el modal.
- cy.contains('Terminar evidencias').should('not.exist'): Verifica que la tarea ya no aparece en la lista ni después de recargar.

CP-04 – Visualizar tareas completadas al abrir (persistencia)

- cy.get('[data-cy=add-button]').click() y cy.get('[data-cy=task-input]').type('Fase 1'): Abre el formulario y escribe la tarea “Fase 1”.
- cy.get('[data-cy=submit]').click(): Guarda la tarea.
- cy.contains('Fase 1').parent().find('[data-cy=task-checkbox]').check(): Marca la tarea como completada.
- cy.reload(): Recarga la página para simular cerrar y volver a abrir la aplicación.
- cy.contains('Fase 1').parent().find('[data-cy=task-checkbox]').should('be.checked'): Verifica que la tarea siga marcada como completada, demostrando que la información se conservó en LocalStorage.
-





Se ejecutó una batería de pruebas unitarias y de extremo a extremo para validar los requisitos funcionales y no funcionales de la aplicación To-Do List.

Pruebas unitarias (Jest)

- Se probaron las clases de dominio (tarea, lista de tareas y repositorio) verificando creación de tareas válidas, cambio de estado (pendiente/completada) y cálculo de estadísticas (total, completadas, pendientes).
- También se validó la lógica de persistencia en el repositorio, comprobando que las tareas se guardan y recuperan correctamente desde el almacenamiento local simulado, evitando pérdida de información ante cierres de la aplicación.

Pruebas E2E automatizadas (Cypress)

- Se diseñaron y automatizaron cuatro casos de prueba: crear tarea (CP-01), marcar tarea como completada (CP-02), eliminar tarea (CP-03) y verificar la persistencia de tareas completadas tras recargar la aplicación (CP-04).
- Para cada caso, se simuló un usuario autenticado, se abrió el formulario de nueva tarea, se interactuó con la interfaz (checkbox, botón eliminar, recarga de página) y se comprobó en la UI que el comportamiento coincide con los

requisitos definidos (RF e historias de usuario HU-01 a HU-04).

Resultados y conclusiones

- Todas las pruebas unitarias ejecutadas pasaron correctamente, lo que indica que la lógica de negocio (validaciones, cambio de estado, estadísticas y persistencia) es consistente y robusta para el alcance actual del proyecto.
- Los cuatro casos E2E en Cypress finalizaron en estado exitoso, evidenciando que, desde la perspectiva del usuario, la aplicación permite crear, completar, eliminar tareas y conservar el estado de las tareas completadas al cerrar y volver a abrir la app, cumpliendo los objetivos de organización y productividad planteados para la solución.

Conclusiones

Las pruebas realizadas permitieron confirmar que la aplicación To-Do List cumple de forma consistente con los requisitos funcionales de crear, completar, eliminar y mantener tareas al recargar la interfaz. Las pruebas unitarias con Jest evidenciaron que la lógica de negocio y la persistencia en almacenamiento local son correctas, mientras que las pruebas E2E con Cypress demostraron que los flujos clave funcionan desde la perspectiva del usuario final. En conjunto, la actividad mostró que incorporar pruebas automatizadas en el ciclo de desarrollo mejora la confiabilidad del software y facilita detectar y corregir errores antes de avanzar a fases posteriores del proyecto.