École Nationale d'Ingénieurs de Carthage

LIVRABLE TECHNIQUE

Scripts d'Automatisation des Tests

Projet de Validation Logicielle (ISTQB)

Todo App (Spring Boot + React)

Équipe QA

Mohamed Dhia Eddine Thabet - Test Manager
Mohamed Aziz Dridi - Tester

Décembre 2025
Version 1.0

Table des matières

1 Introduction

Ce document rassemble l'intégralité du code source utilisé pour l'automatisation des tests du projet Todo App. Il couvre trois niveaux de la pyramide des tests :

- **Niveau 1 : Tests End-to-End (E2E)** réalisés avec Cypress (Frontend).
- **Niveau 2 : Tests d'Intégration** réalisés avec JUnit 5 et MockMvc (Backend).
- **Niveau 3 : Tests de Performance** réalisés avec JUnit (Simulation de charge).

2 Tests End-to-End (Frontend)

Outil : Cypress
Fichier : `react/cypress/e2e/todo_app_istqb.cy.js`

Ce script simule un parcours utilisateur complet dans le navigateur, incluant la validation des correctifs de bugs (comme le Bug "Zombie").

```
describe('ISTQB E2E Test Suite - Todo App React', () => {

  // Configuration avant chaque test
  beforeEach(() => {
    // 1. On visite l'application
    cy.visit('http://localhost:3000')

    // 2. Attente implicite que les donnees chargent
    cy.contains('h2', 'Todo List', { matchCase: false }).should('be.visible')

    // Pause visuelle initiale
    cy.wait(1000)
  })

  it('TC-01 : Smoke Test & Verification du "Zombie Item" (Bug #002)', () => {
    // ISTQB : Verification de l'etat initial
    // Si la liste est vide, le backend cree "Click to edit task name"
    cy.get('.todoitems')
      .find('input[type="text"]')
      .should('have.value', 'Click to edit task name')

    cy.wait(1000)
  })

  it('TC-02 : Creation d\'une Tache (Functional Testing)', () => {
    cy.get('.todoitems input[type="text"]').then(($items) => {
      const initialCount = $items.length

      // Clic sur le bouton "Add task"
      cy.contains('button', 'Add task').click()
      cy.wait(500)

      // Verifier qu'un nouvel input est apparu
      cy.get('.todoitems input[type="text"]').should('have.length', initialCount + 1)

      // Verifier le texte par defaut
      cy.get('.todoitems input[type="text"]').last()
        .should('have.value', 'Click to edit task name')
        .click()
```

```
    cy.wait(1000)
  })
})

it('TC-03 : Modification de Tache (Input & State Update)', () => {
  const newTaskName = 'Test Cypress Automatise ' + Date.now()

  cy.get('.todoitems input[type="text"]').first()
    .clear()
    .should('have.value', '')
    .wait(200)
    .type(newTaskName, { delay: 100 })
    .blur()

  // Verifier la persistance locale
  cy.get('.todoitems input[type="text"]').first()
    .should('have.value', newTaskName)

  cy.wait(1000)
})

it('TC-04 : Transition d\'Etat (Done/Not Done)', () => {
  cy.get('.todoitems input[type="checkbox"]').first().check()

  // Verifier le style barre
  cy.get('.todoitems input[type="text"]').first()
    .should('have.class', 'done')
    .and('have.css', 'text-decoration', 'line-through solid rgba(0, 0, 0, 0.3)')

  cy.wait(1000)

  cy.get('.todoitems input[type="checkbox"]').first().uncheck()

  cy.get('.todoitems input[type="text"]').first()
    .should('not.have.class', 'done')

  cy.wait(1000)
})

it('TC-05 : Suppression (Delete) & Confirmation Bug Zombie', () => {
  cy.get('button[aria-label="delete"]').first().click()

  // Verification ISTQB du Bug "Zombie"
  cy.wait(1000)

  // Assertion : Si l'element est toujours la, le bug est confirme
  cy.get('.todoitems input[type="text"]').should('exist')

  cy.wait(1000)
})

it('TC-06 : Test de Charge UI (Stress Test)', () => {
   for(let i = 0; i < 5; i++) {
       cy.contains('button', 'Add task').click()
       cy.wait(200)
   }
```

```
97        cy.get('.todoitems input[type="text"]').should('have.length.gte', 5)
98        cy.wait(2000)
99      })
100
101 })
```

Listing 1 – Script Cypress E2E

3 Tests d'Intégration (Backend)

Outil : Spring Boot Test / MockMvc
Fichier : `TodoFullStackIntegrationTest.java`

Ce test valide le cycle de vie complet d'une tâche (CRUD) au niveau de l'API, sans passer par l'interface graphique.

```java
package com.dhia.todoapp.integration;

import static org.hamcrest.Matchers.hasSize;
import static org.hamcrest.Matchers.is;
import static org.springframework.test.web.servlet.request.
    MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.
    MockMvcResultMatchers.*;

import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.
    AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.web.servlet.MockMvc;

import com.dhia.todoapp.domain.TodoItem;
import com.fasterxml.jackson.databind.ObjectMapper;

@SpringBootTest
@AutoConfigureMockMvc
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_CLASS)
class TodoFullStackIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Test
    @Order(1)
    void step1_shouldStartWithDefaultItem() throws Exception {
        mockMvc.perform(get("/api/todoItems"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", hasSize(1)))
                .andExpect(jsonPath("$[0].task", is("Click to edit task
    name")));
    }

    @Test
    @Order(2)
    void step2_shouldCreateNewItem() throws Exception {
        mockMvc.perform(post("/api/todoItems"))
                .andExpect(status().isOk())
```

```
                .andExpect(jsonPath("$.task", is("Click to edit task name")
    ))
                .andExpect(jsonPath("$.isDone", is(false)));

        mockMvc.perform(get("/api/todoItems"))
                .andExpect(jsonPath("$", hasSize(2)));
    }

    @Test
    @Order(3)
    void step3_shouldUpdateItem() throws Exception {
        TodoItem updateData = new TodoItem();
        updateData.setTask("Acheter du pain");
        updateData.setIsDone(true);

        mockMvc.perform(put("/api/todoItems/{id}", 0)
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(objectMapper.writeValueAsString(updateData
    )))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.task", is("Acheter du pain")))
                .andExpect(jsonPath("$.isDone", is(true)));
    }

    @Test
    @Order(4)
    void step4_shouldDeleteItem() throws Exception {
        mockMvc.perform(delete("/api/todoItems/{id}", 0))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", is("ok")));

        mockMvc.perform(get("/api/todoItems"))
                .andExpect(jsonPath("$", hasSize(1)))
                .andExpect(jsonPath("$[0].id", is(1)));
    }

    @Test
    @Order(5)
    void step5_demonstrateZombieBug() throws Exception {
        // Demonstration du Bug "Zombie"
        mockMvc.perform(delete("/api/todoItems/{id}", 1))
                .andExpect(status().isOk());

        // La liste devrait etre vide, mais le backend recree un item
        mockMvc.perform(get("/api/todoItems"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$", hasSize(1)))
                .andExpect(jsonPath("$[0].task", is("Click to edit task
    name")));
    }
}
```

Listing 2 – Test Intégration Spring Boot

4 Tests de Performance (Stress Test)

Outil : JUnit 5 + Java Concurrency
Fichier : `TodoComplexPerformanceTest.java`

Ce test vérifie la robustesse du backend en simulant 50 utilisateurs simultanés (Thread-Safety).

```java
package com.dhia.todoapp.perfermance;

import static org.junit.jupiter.api.Assertions.assertTrue;
import java.util.List;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.dhia.todoapp.domain.TodoItem;
import com.dhia.todoapp.service.TodoService;

@SpringBootTest
@Tag("stress-test")
class TodoComplexPerformanceTest {

    @Autowired
    private TodoService todoService;

    // CONFIGURATION CHARGE
    private static final int VIRTUAL_USERS = 50;
    private static final int TASKS_PER_USER = 100;

    @Test
    @DisplayName("STRESS TEST: Simulation 50 utilisateurs concurrents")
    void shouldHandleHighConcurrency() throws InterruptedException {

        ExecutorService executorService = Executors.newFixedThreadPool(
    VIRTUAL_USERS);
        CountDownLatch latch = new CountDownLatch(VIRTUAL_USERS);
        AtomicInteger errorCount = new AtomicInteger(0);

        System.out.println("DEBUT DU STRESS TEST...");
        long startTime = System.nanoTime();

        for (int i = 0; i < VIRTUAL_USERS; i++) {
            executorService.submit(() -> {
                try {
                    for (int j = 0; j < TASKS_PER_USER; j++) {
                        todoService.createNewTodoItem();
                    }
                } catch (Exception e) {
                    errorCount.incrementAndGet();
                    e.printStackTrace();
```

```
                } finally {
                    latch.countDown();
                }
            });
        }

        boolean finished = latch.await(10, TimeUnit.SECONDS);
        long endTime = System.nanoTime();
        executorService.shutdown();

        // ANALYSE KPI
        double durationSeconds = (endTime - startTime) / 1_000_000_000.0;
        int totalRequests = VIRTUAL_USERS * TASKS_PER_USER;
        double requestsPerSecond = totalRequests / durationSeconds;

        List<TodoItem> finalItems = todoService.fetchAllTodos();

        System.out.println("=== RAPPORT PERFORMANCE ===");
        System.out.println("Temps total      : " + String.format("%.4f",
    durationSeconds) + " s");
        System.out.println("Debit (RPS)      : " + String.format("%.2f",
    requestsPerSecond));
        System.out.println("Items en base    : " + finalItems.size());

        // ASSERTIONS
        assertTrue(requestsPerSecond > 1000, "Performance insuffisante (<
    1000 RPS)");
        assertTrue(finalItems.size() > 0, "La base ne doit pas etre vide");
     }
}
```

Listing 3 – Script de Test de Charge