# README.md

## ✅ Todo App QA Project (React + Node.js)

This project is a full-stack Todo application with automated tests for both UI and API layers. It demonstrates best practices in software QA engineering.

## 📦 Project Structure

todo-app-qa-project/  
├── client/ # React frontend  
│ ├── src/  
│ └── tests/ # Playwright UI tests  
├── server/ # Node.js backend  
│ ├── routes/  
│ └── tests/ # Supertest API tests  
├── .github/workflows/ # GitHub Actions CI  
├── README.md  
├── TEST\_PLAN.md

## 🛠️ Setup Instructions

### 1. Clone the repo

git clone https://github.com/resmi-tech/applogin.git  
cd applogin

### 2. Start Backend API

cd server  
npm install  
npm start

Runs at http://localhost:5000

### 3. Start Frontend App

cd ../client  
npm install  
npm start

Runs at http://localhost:3000

## ✅ Running Tests

### 🔬 API Tests (Supertest)

cd server  
npm test

### 🎯 UI Tests (Playwright)

cd client  
npx playwright install  
npx playwright test

## 🚀 GitHub Actions CI

* Tests run automatically on every push via .github/workflows/ci.yml

## 📄 License

MIT License

# TEST CASES

## ✅ Playwright UI Test Scenarios (client/tests/todo.spec.ts)

import { test, expect } from '@playwright/test';  
  
test.describe('Todo App UI Tests', () => {  
 test('Login with valid credentials', async ({ page }) => {  
 await page.goto('http://localhost:3000/login');  
 await page.fill('#username', 'admin');  
 await page.fill('#password', 'password');  
 await page.click('button[type="submit"]');  
 await expect(page).toHaveURL('http://localhost:3000/dashboard');  
 });  
  
 test('Create a new todo item', async ({ page }) => {  
 await page.goto('http://localhost:3000/dashboard');  
 await page.fill('#new-todo', 'Buy milk');  
 await page.click('button#create-todo');  
 await expect(page.locator('li.todo-item')).toContainText('Buy milk');  
 });  
  
 test('Edit an existing todo', async ({ page }) => {  
 await page.goto('http://localhost:3000/dashboard');  
 await page.click('li.todo-item >> text=Edit');  
 await page.fill('input.edit-todo', 'Buy almond milk');  
 await page.click('button#save-edit');  
 await expect(page.locator('li.todo-item')).toContainText('Buy almond milk');  
 });  
  
 test('Delete a todo item', async ({ page }) => {  
 await page.goto('http://localhost:3000/dashboard');  
 await page.click('li.todo-item >> text=Delete');  
 await expect(page.locator('li.todo-item')).not.toContainText('Buy almond milk');  
 });  
});

## ✅ API Test Scenarios (server/tests/api.test.js)

const request = require('supertest');  
const app = require('../server');  
  
describe('Todo API Tests', () => {  
 let todoId;  
  
 it('POST /login - valid credentials', async () => {  
 const res = await request(app)  
 .post('/login')  
 .send({ username: 'admin', password: 'password' });  
 expect(res.statusCode).toEqual(200);  
 expect(res.body.token).toBeDefined();  
 });  
  
 it('POST /todos - create todo', async () => {  
 const res = await request(app)  
 .post('/todos')  
 .send({ title: 'Learn testing' });  
 expect(res.statusCode).toEqual(201);  
 todoId = res.body.id;  
 });  
  
 it('GET /todos - fetch todos', async () => {  
 const res = await request(app).get('/todos');  
 expect(res.statusCode).toEqual(200);  
 expect(res.body.length).toBeGreaterThan(0);  
 });  
  
 it('PUT /todos/:id - update todo', async () => {  
 const res = await request(app)  
 .put(`/todos/${todoId}`)  
 .send({ title: 'Learn Supertest' });  
 expect(res.statusCode).toEqual(200);  
 });  
  
 it('DELETE /todos/:id - delete todo', async () => {  
 const res = await request(app).delete(`/todos/${todoId}`);  
 expect(res.statusCode).toEqual(204);  
 });  
});

Let me know if you want me to push these files to your GitHub repo directly or if you’d like them zipped for download.

i need zipped file to download

i need zipped file to download