1) Introducción

Este proyecto automatiza pruebas UI y API empleando Cypress junto con el preprocesador Cucumber (Gherkin) y el patrón Page Object Model (POM). Además, integra generación de datos con Faker, soporte multi-entorno vía cypress.env.json y ejecución CI en GitHub Actions.

2) Estructura del proyecto

Carpetas principales y propósito:

• cypress/journeys/features — Escenarios Gherkin (.feature) separados por dominio (api/e2e).

• cypress/journeys/step_definitions — Steps Cucumber en JavaScript. Subcarpeta 'ui' para pasos de interfaz propios.

• cypress/pages — Page Objects (POM). Subcarpetas demoqa, lambdatest, etc.

• cypress/support — Utilidades como DataGenerator (Faker) y configuración global.

• jsonlogs / reports / screenshots / videos — Evidencias y reportes.

• cypress.env.json — URLs y credenciales por entorno (DEV/TST).

3) Archivos clave añadidos

UI / E2E:

• cypress/journeys/features/e2e/bookstore_login.feature

• cypress/journeys/features/e2e/lambdatest_input_form.feature

• cypress/journeys/step_definitions/ui/bookstore_login.steps.js

• cypress/journeys/step_definitions/ui/lambdatest_input_form.steps.js

• cypress/pages/demoqa/LoginPage.js

• cypress/pages/demoqa/ProfilePage.js

• cypress/pages/demoqa/RegisterPage.js

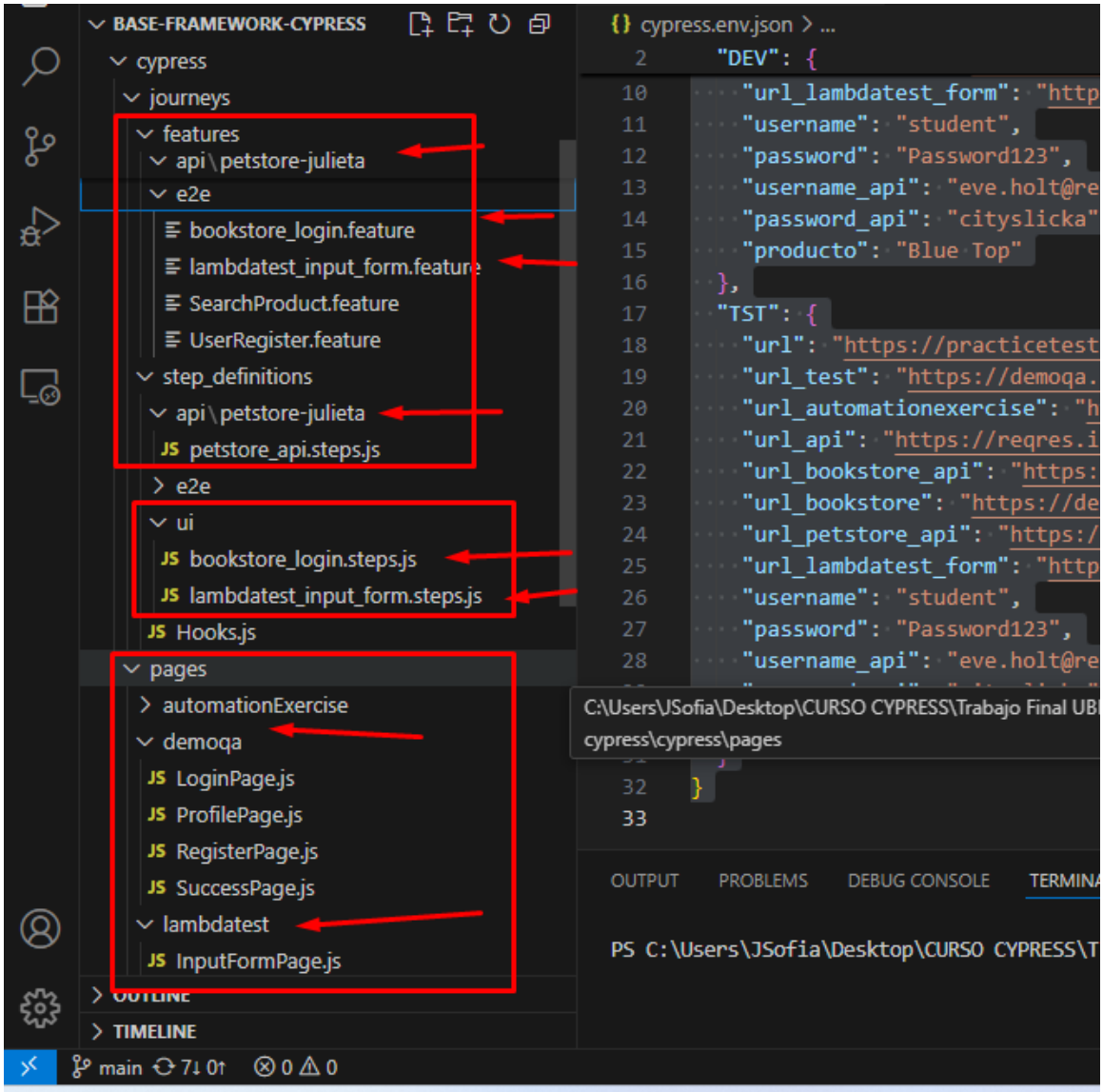
• cypress/pages/demoqa/SuccessPage.js

• cypress/pages/lambdatest/InputFormPage.js

API:

• cypress/journeys/features/api/petstore-julieta/petstore_api.feature

• cypress/journeys/step_definitions/api/petstore.steps.js (con polling y headers comunes)



4) Escenarios automatizados

a) Book Store (DemoQA) – Login con usuario creado por API

- Background: crea usuario real vía POST /Account/v1/User con credenciales Faker.
- Escenario 1: Login válido → redirección a /profile y verificación de #userName-value.
- Escenario 2: Login inválido con usuario no registrado → mensaje “Invalid username or password!”.

b) Petstore – CRUD básico de mascotas

- Crear mascota con id único y nombre Faker, GET por id con reintentos (polling).
- Actualizar nombre, verificar cambio.
- Eliminar y comprobar 404 con espera resiliente.(código en la imagen). Espera que el servicio tenga disponibilidad para consultar el dato que acabo de crear

```
  }

// ---- helpers de polling ----
const waitForPetToExist = (id, retries = 8, delay = 800) => {
  const tryOnce = (n) =>
    cy
      .request({
        method: 'GET',
        url: `${baseUrl}/pet/${id}`,
        headers: getHeaders(),
        failOnStatusCode: false,
      })
      .then((r) => {
        if (r.status === 200) return r
        if (n <= 0) throw new Error(`Pet ${id} not found after retries`)
        return cy.wait(delay).then(() => tryOnce(n - 1))
      })
  return tryOnce(retries)
}

const waitForPetToBeGone = (id, retries = 10, delay = 800) => {
  const tryOnce = (n) =>
```

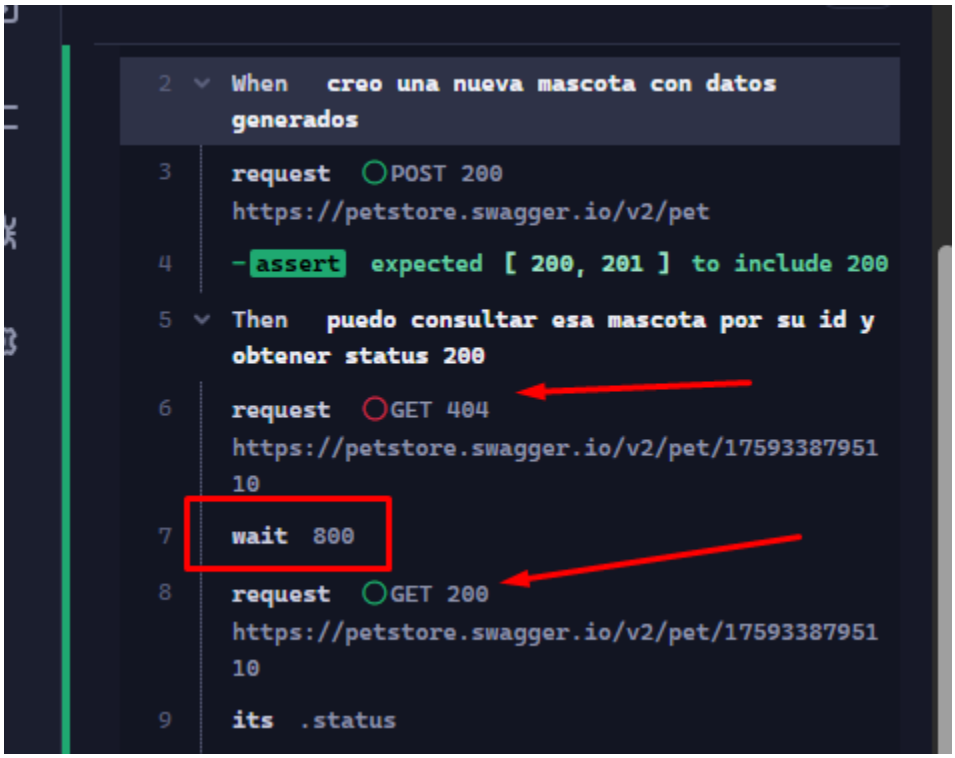
Aplique helpers de polling para que los tests reintenten consultar la API varias veces con pausas.
Sirven para esperar a que el servicio termine de actualizar:

- **waitForPetToExist espera hasta que la mascota aparezca (200).**

- **waitForPetToBeGone espera hasta que desaparezca (404).**

Asi me aseguro que los tests no fallen por demoras de la API**.**

En esta ejecución, se ve como en la primer consulta responde con 404, entonces espera y reintenta



c) LambdaTest – Input Form

- Completa todos los campos (#name, #inputEmail4, #company, etc.) con datos Faker.
- Selección de país robusta y manejo de ruidos (analytics/ads) y popups (cookies/chat).
- Envío y verificación de mensaje ‘Thanks for contacting us…’.

5) POM y Data Generator

Se aplicó POM para encapsular selectores/acciones (ej.: LoginPage, ProfilePage, InputFormPage). Faker se usa desde support/DataGenerator y en steps para generar nombres, emails, contraseñas, direcciones, etc.

Ejemplo de POM (InputFormPage.visit):

```
visit() {
  cy.visit(this.url, { failOnStatusCode: false });
  cy.window({ timeout: 20000 }).its('document.readyState').should('eq',
'complete');
  cy.get('body').then(($b) => {
    const cookieBtn =
'#CybotCookiebotDialogBodyLevelButtonLevelOptinAllowAll';
    if ($b.find(cookieBtn).length) cy.get(cookieBtn).click({ force: true });
    const chatBtn = '#embeddedMessagingConversationButton';
    if ($b.find(chatBtn).length) cy.get(chatBtn).invoke('attr', 'style', 'display:none
!important');
  });
  cy.get('form#seleniumform', { timeout: 20000 }).should('be.visible');
}
```

6) Entornos y navegadores

URLs por entorno en cypress.env.json (DEV/TST) para Book Store, Petstore y LambdaTest. Ejecución en Chrome/Chromium (local), y definida para Linux en CI.

```
# Interactivo
npx cypress open --e2e --browser chrome -e ENV=DEV

# Headless (un feature)
npx cypress run --spec cypress/journeys/features/e2e/bookstore_login.feature -e
ENV=DEV --browser chrome
```

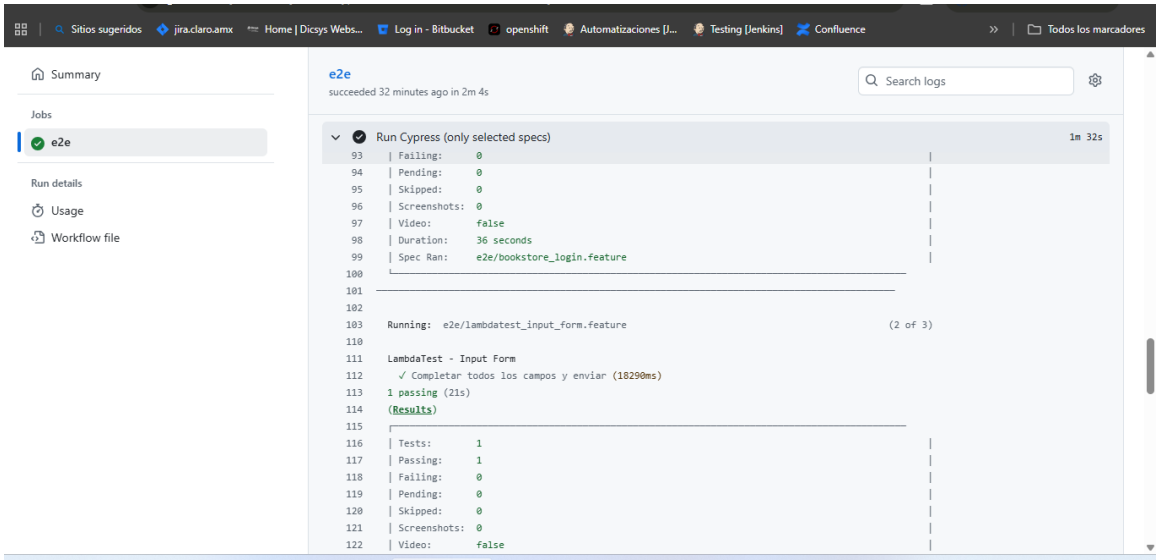
7) CI/CD con GitHub Actions

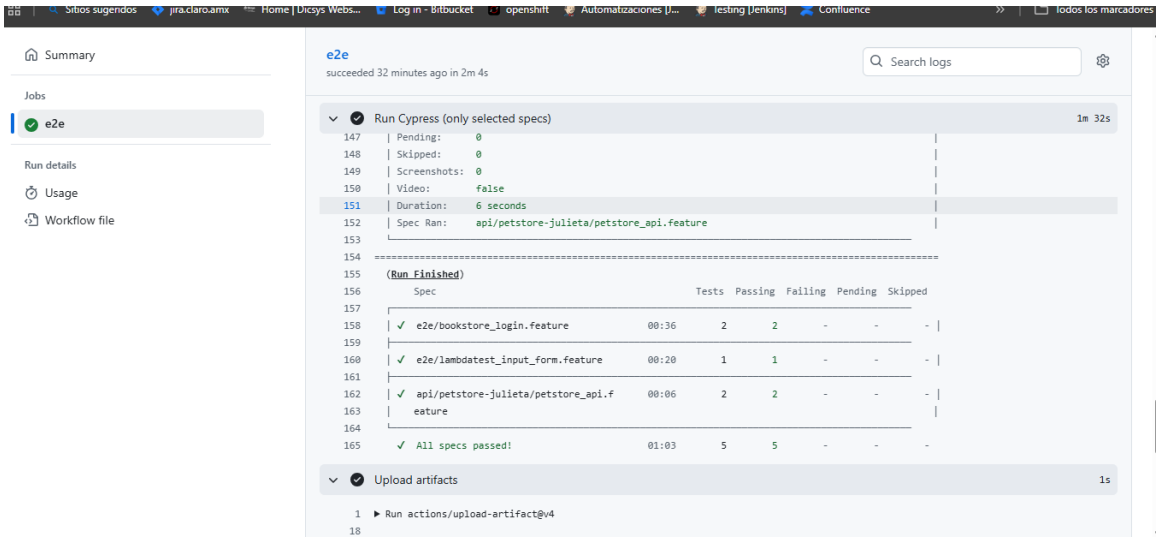
Workflow e2e en .github/workflows: instala dependencias, corrige case-sensitive en Linux, valida existencia de 3 specs propios y ejecuta sólo esos. Sube artefactos (screenshots, videos, jsonlogs, reports).

"Se ejecutaron 3 specs (archivos .feature) con un total de 5 escenarios."

Link : https://github.com/jiulev/Trabajo-Final-Cypress-UBP/actions/runs/18168882555/job/51718309772

```
uses: cypress-io/github-action@v6
with:
  command: >
    npx cypress run -e ENV=DEV
    --spec cypress/journeys/features/e2e/bookstore_login.feature,
          cypress/journeys/features/e2e/lambdatest_input_form.feature,
          cypress/journeys/features/api/petstore-julieta/petstore_api.feature
```





8) Comandos útiles

npm install
npm run test:with-report (genera jsonlogs)
npx cypress open --e2e -e ENV=DEV