



UNIVERSIDAD
CATÓLICA
BOLIVIANA

Tercer Parcial: Pruebas de API y Pruebas E2E en Todo.ly

Oscar Campohermoso Berdeja

Universidad Católica Boliviana

SIS-312: Gestión de Calidad de Sistemas

Profesor: Lic. Cecilia Alvarado Monrroy

20 de noviembre de 2024

Índice general

1. API Testing	3
1.1. Resumen de Ejecución de Pruebas	3
1.2. Descripción de las Pruebas	5
1.2.1. GET /Filters/Id	5
1.2.2. GET /Filters/Id/Items	6
1.2.3. GET /Filters/Id/DoneItems	7
1.2.4. GET /Items	8
1.3. Conclusión	9
2. Pruebas E2E	10
2.1. Casos de Prueba E2E	10
2.1.1. Editar el Penúltimo Ítem en <i>Work</i> y Asignarle Prioridad	10
2.1.2. Eliminar el Segundo Ítem	11
2.2. Conclusión	13
3. Anexos	14
3.1. Repositorio de Gestión de Calidad	14
3.2. Repositorio de Pruebas E2E con Playwright	14
3.3. Instrucciones para Clonar y Ejecutar los Repositorios	14
3.3.1. Clonar Repositorios	14
3.3.2. Ejecución de Pruebas en Playwright	15
3.3.3. Ejecución de Pruebas de API con Newman	15

Índice de figuras

1.1. Resultado de la prueba GET /Filters/Id en Postman	6
1.2. Resultado de la prueba GET /Filters/Id/Items en Postman	7
1.3. Resultado de la prueba GET /Filters/Id/DoneItems en Postman	8
1.4. Resultado de la prueba GET /Items en Postman	9
2.1. Reporte de ambas pruebas E2E en Playwright en múltiples navegadores	10
2.2. Resumen de los pasos de la prueba en Playwright, realizados en chromium	11
2.3. Resumen de los pasos de la prueba en Playwright, realizados en firefox	12

Capítulo 1

API Testing

Este capítulo describe el proceso sistemático de pruebas de API realizadas sobre el servicio de Todo.ly. El objetivo principal de estas pruebas fue evaluar la funcionalidad, el rendimiento y la estructura de las respuestas en los endpoints seleccionados. Este ejercicio refleja una metodología estructurada para garantizar la confiabilidad y calidad del sistema, a través de pruebas de API automatizadas.

1.1. Resumen de Ejecución de Pruebas

Las pruebas realizadas en Postman se ejecutaron utilizando la herramienta Newman, que permite correr colecciones de Postman desde la línea de comandos. El siguiente es un resumen de la ejecución:

```
> newman run API-testing-OscarCampohermoso.postman_collection.json -e
QAenv-OscarCampohermoso.postman_environment.json --color off
```

```
API-testing-OscarCampohermoso
```

```
-> Crear Usuario
  | 'Generated Email:', 'taylor_johnson@example.com'
  | 'Generated Full Name:', 'Taylor Johnson'
  POST https://todo.ly/api/user.json [200 OK, 705B, 1037ms]

-> POST /Projects
  POST https://todo.ly/api/projects.json [200 OK, 721B, 229ms]

-> POST /Items 1rs item
  POST https://todo.ly/api/items.json [200 OK, 851B, 232ms]

-> POST /Items 2nd item
  POST https://todo.ly/api/items.json [200 OK, 851B, 247ms]

-> POST /Items 3rd item
  POST https://todo.ly/api/items.json [200 OK, 851B, 249ms]
```

```

-> PUT /Items/Id
  PUT https://todo.ly/api/items/11600340.json [200 OK, 870B, 232ms]

-> GET /Filters
  GET https://todo.ly/api/filters.json [200 OK, 637B, 227ms]

-> * GET /Filters/Id
  GET https://todo.ly/api/filters/-1.json [200 OK, 390B, 228ms]
  - El código de estado es 200
  - El tiempo de respuesta es menor a 2000ms
  - El contenido del filtro es una cadena de texto
  - El tipo de ítem es 4 (Filtro)
  - El ID del filtro es un número negativo
  - El número de ítems es un número entero

-> * GET /Filters/Id/Items
  GET https://todo.ly/api/filters/-2/items.json [200 OK, 1.39kB, 234ms]
  - El código de estado es 200
  - El tiempo de respuesta es menor a 2000ms
  - La respuesta es una lista de ítems
  - Verificar que cada ítem tenga el OwnerId igual a {{user_id}}
  - Verificar que cada ítem tenga un ID válido

-> * GET /Filters/Id/DoneItems
  GET https://todo.ly/api/filters/-2/doneitems.json [200 OK, 872B, 228ms]
  - El código de estado es 200
  - El tiempo de respuesta es menor a 2000ms
  - La respuesta es una lista de ítems completados
  - Verificar que cada ítem tenga el OwnerId igual a {{user_id}}
  - Verificar que cada ítem tenga un ID válido
  - Verificar ítems completados

-> * GET /Items
  GET https://todo.ly/api/items.json [200 OK, 1.95kB, 226ms]
  - El código de estado es 200
  - El tiempo de respuesta es menor a 2000ms
  - La respuesta es una lista de ítems
  - Verificar que cada ítem tenga el OwnerId igual a {{user_id}}
  - Cada ítem tiene un ID y contenido válidos

```

1.2. Descripción de las Pruebas

Las pruebas se diseñaron para los siguientes endpoints, y como se observa en el resumen de la ejecución de pruebas, se llevaron a cabo con éxito. Cada prueba fue planificada para verificar tanto la estructura de las respuestas como la consistencia y validez de los datos retornados por la API.

1.2.1. GET /Filters/Id

Este endpoint devuelve los detalles de un filtro específico y se validaron los siguientes aspectos:

- Código de estado 200 ✓
- Tiempo de respuesta menor a 2000 ms ✓
- El contenido del filtro es una cadena de texto ✓
- El tipo de ítem es 4 (Filtro) ✓
- El ID del filtro es un número negativo ✓
- El número de ítems es un número entero ✓

Este endpoint permite al usuario obtener detalles sobre filtros específicos, como los que muestran las tareas de hoy o próximas. Verificar la integridad de estos datos es crucial para la correcta visualización de la información en la interfaz.

```
1 pm.test("El código de estado es 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("El tiempo de respuesta es menor a 2000ms", function () {
6     pm.expect(pm.response.responseTime).to.be.below(2000);
7 });
8
9 pm.test("El contenido del filtro es una cadena de texto", function () {
10     var jsonData = pm.response.json();
11     pm.expect(jsonData.Content).to.be.a('string');
12 });
13
14 pm.test("El tipo de ítem es 4 (Filtro)", function () {
15     var jsonData = pm.response.json();
16     pm.expect(jsonData.ItemType).to.eqls(4);
17 });
18
19 pm.test("El ID del filtro es un número negativo", function () {
20     var jsonData = pm.response.json();
21     pm.expect(jsonData.Id).to.be.below(0);
22 });
23
24 pm.test("El número de ítems es un número entero", function () {
25     var jsonData = pm.response.json();
26     pm.expect(jsonData.ItemsCount).to.be.a('number');
27 });
```

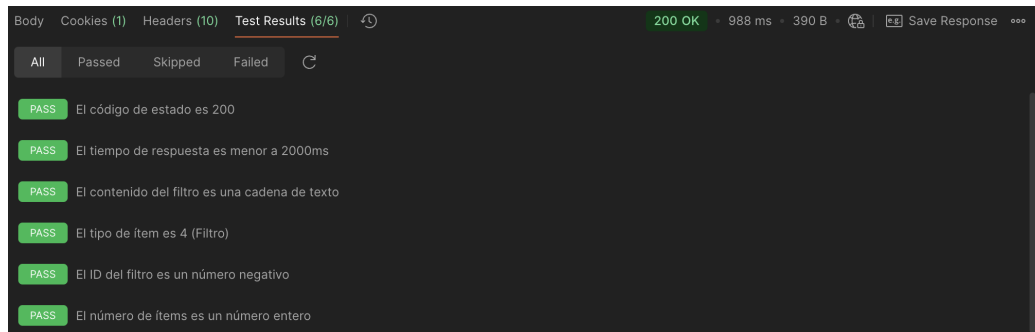


Figura 1.1: Resultado de la prueba GET /Filters/Id en Postman

1.2.2. GET /Filters/Id/Items

Este endpoint devuelve la lista de ítems asociados a un filtro específico. Las validaciones incluyen:

- Código de estado 200 ✓
- Tiempo de respuesta menor a 2000 ms ✓
- La respuesta es una lista de ítems ✓
- Cada ítem tiene un OwnerId igual a {{user_id}} ✓
- Cada ítem tiene un ID válido ✓

Este endpoint es útil para listar las tareas filtradas por un criterio específico, como las tareas que deben realizarse hoy. Las pruebas aseguran que cada ítem en la lista esté correctamente asociado al usuario autenticado.

```

1 pm.test("El código de estado es 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("El tiempo de respuesta es menor a 2000ms", function () {
6     pm.expect(pm.response.responseTime).to.be.below(2000);
7 });
8
9 pm.test("La respuesta es una lista de ítems", function () {
10     var jsonData = pm.response.json();
11     pm.expect(jsonData).to.be.an('array');
12 });
13
14 pm.test("Verificar que cada ítem tenga el OwnerId igual a {{user_id}}", function () {
15     var jsonData = pm.response.json();
16     var userId = pm.variables.get("user_id");
17     jsonData.forEach(function(item) {
18         pm.expect(item.OwnerId).to.eqls(parseInt(userId));
19     });
20 });
21
22 pm.test("Verificar que cada ítem tenga un ID válido", function () {

```

```

23 var jsonData = pm.response.json();
24 jsonData.forEach(function(item) {
25     pm.expect(item.Id).to.be.a('number').and.to.be.above(0);
26 });
27 });

```

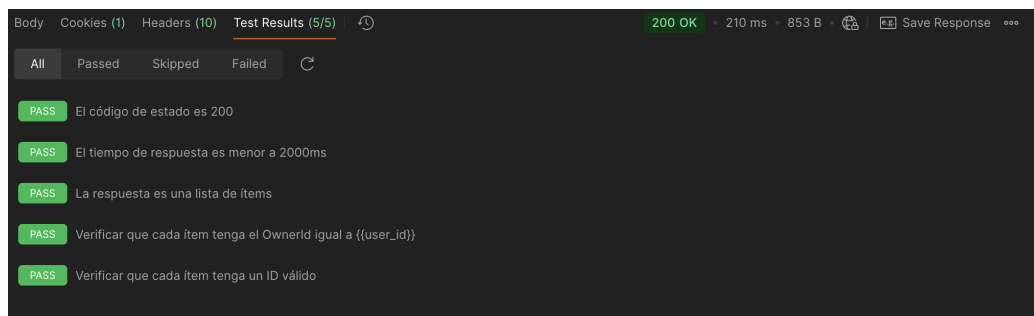


Figura 1.2: Resultado de la prueba GET /Filters/Id/Items en Postman

1.2.3. GET /Filters/Id/DoneItems

Este endpoint devuelve la lista de ítems completados asociados a un filtro específico. Las validaciones realizadas fueron:

- Código de estado 200 ✓
- Tiempo de respuesta menor a 2000 ms ✓
- La respuesta es una lista de ítems completados ✓
- Cada ítem tiene un OwnerId igual a {{user_id}} ✓
- Cada ítem tiene un ID válido ✓
- Verificar que los ítems estén marcados como completados ✓

Este endpoint permite al usuario ver las tareas que ha completado. Las pruebas aseguran que todos los ítems devueltos estén efectivamente marcados como completados y asociados al usuario correcto.

```

1 pm.test("El código de estado es 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("El tiempo de respuesta es menor a 2000ms", function () {
6     pm.expect(pm.response.responseTime).to.be.below(2000);
7 });
8
9 pm.test("La respuesta es una lista de ítems completados", function () {
10     var jsonData = pm.response.json();
11     pm.expect(jsonData).to.be.an('array');

```



```

12 });
13
14 pm.test("Verificar que cada ítem tenga el OwnerId igual a {{user_id}}", function () {
15     var jsonData = pm.response.json();
16     var userId = pm.variables.get("user_id");
17     jsonData.forEach(function(item) {
18         pm.expect(item.OwnerId).to.eqls(parseInt(userId));
19     });
20 });
21
22 pm.test("Verificar que cada ítem tenga un ID válido", function () {
23     var jsonData = pm.response.json();
24     jsonData.forEach(function(item) {
25         pm.expect(item.Id).to.be.a('number').and.to.be.above(0);
26     });
27 });
28
29 pm.test("Verificar ítems completados", function () {
30     var jsonData = pm.response.json();
31     jsonData.forEach(function(item) {
32         pm.expect(item.Checked).to.eqls(true);
33     });
34 });

```

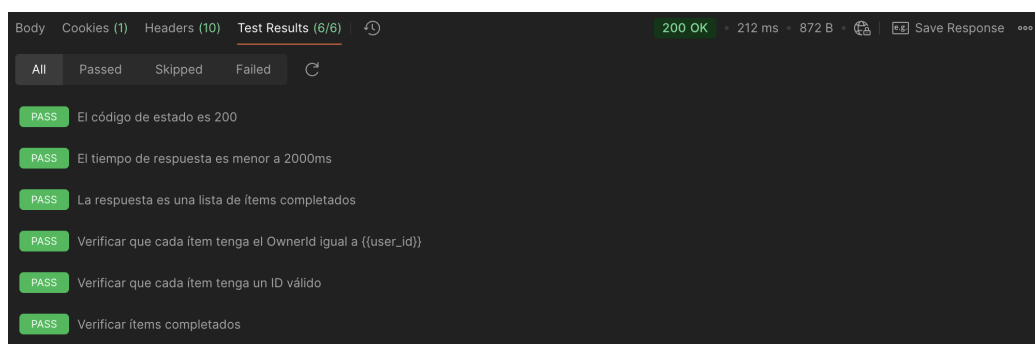


Figura 1.3: Resultado de la prueba GET /Filters/Id/DoneItems en Postman

1.2.4. GET /Items

Este endpoint devuelve la lista de todos los ítems del usuario autenticado. Las pruebas realizadas fueron:

- Código de estado 200 ✓
- Tiempo de respuesta menor a 2000 ms ✓
- La respuesta es una lista de ítems ✓
- Cada ítem tiene un OwnerId igual a {{user_id}} ✓
- Cada ítem tiene un ID y contenido válidos ✓

Este endpoint es fundamental para mostrar todas las tareas de un usuario y comprobar que los datos devueltos sean correctos y completos.

```
1 pm.test("El código de estado es 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("El tiempo de respuesta es menor a 2000ms", function () {
6     pm.expect(pm.response.responseTime).to.be.below(2000);
7 });
8
9 pm.test("La respuesta es una lista de items", function () {
10     var jsonData = pm.response.json();
11     pm.expect(jsonData).to.be.an('array');
12 });
13
14 pm.test("Verificar que cada ítem tenga el OwnerId igual a {{user_id}}", function () {
15     var jsonData = pm.response.json();
16     var userId = pm.variables.get("user_id");
17     jsonData.forEach(function(item) {
18         pm.expect(item.OwnerId).to.eqls(parseInt(userId));
19     });
20 });
21
22 pm.test("Cada ítem tiene un ID y contenido válidos", function () {
23     var jsonData = pm.response.json();
24     jsonData.forEach(function(item) {
25         pm.expect(item.Id).to.be.a('number').and.to.be.above(0);
26         pm.expect(item.Content).to.be.a('string').and.to.not.be.empty;
27     });
28 });
```

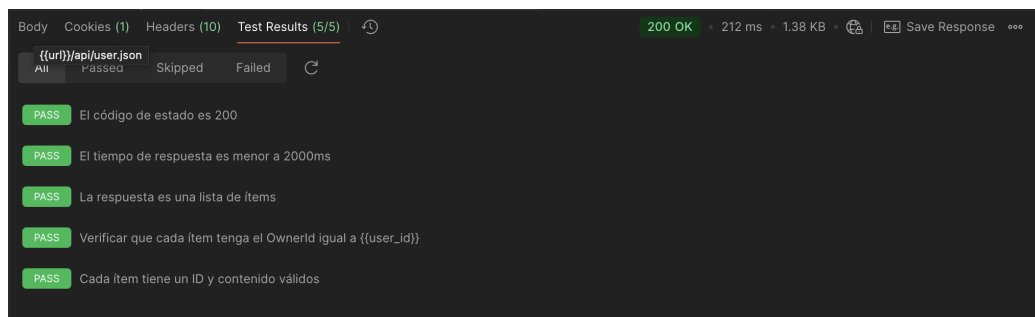


Figura 1.4: Resultado de la prueba GET /Items en Postman

1.3. Conclusión

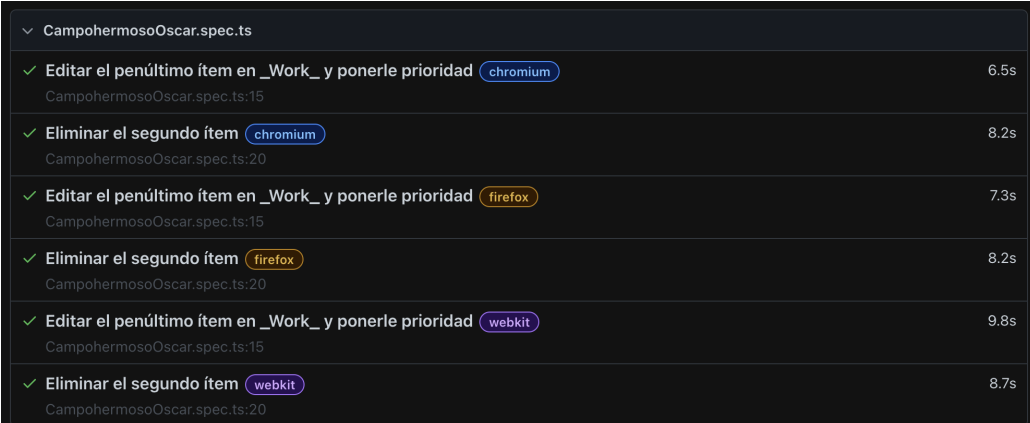
El proceso de pruebas de API permitió evaluar la estabilidad, confiabilidad y calidad de los endpoints de Todo.ly, enfocándose en tiempo de respuesta, validez de datos e integridad estructural. Este ejercicio destaca un enfoque sistemático para validar la funcionalidad de una API.

Capítulo 2

Pruebas E2E

Este capítulo aborda las pruebas end-to-end (E2E) diseñadas para verificar flujos completos de usuario en la aplicación Todo.ly. Estas pruebas aseguran que las diferentes funcionalidades trabajen correctamente en conjunto y que el sistema mantenga su integridad durante la interacción con múltiples componentes.

2.1. Casos de Prueba E2E



✓	CampohermosoOscar.spec.ts		
✓	Editar el penúltimo ítem en _Work_ y ponerle prioridad	chromium	6.5s
	CampohermosoOscar.spec.ts:15		
✓	Eliminar el segundo ítem	chromium	8.2s
	CampohermosoOscar.spec.ts:20		
✓	Editar el penúltimo ítem en _Work_ y ponerle prioridad	firefox	7.3s
	CampohermosoOscar.spec.ts:15		
✓	Eliminar el segundo ítem	firefox	8.2s
	CampohermosoOscar.spec.ts:20		
✓	Editar el penúltimo ítem en _Work_ y ponerle prioridad	webkit	9.8s
	CampohermosoOscar.spec.ts:15		
✓	Eliminar el segundo ítem	webkit	8.7s
	CampohermosoOscar.spec.ts:20		

Figura 2.1: Reporte de ambas pruebas E2E en Playwright en múltiples navegadores

Se llevaron a cabo dos pruebas en el módulo de tareas de la aplicación Todo.ly. Estas pruebas fueron diseñadas para evaluar la interacción entre las diferentes funciones de la aplicación. A continuación, se describen los casos realizados:

2.1.1. Editar el Penúltimo Ítem en Work y Asignarle Prioridad

Este caso de prueba verifica que un usuario pueda editar una tarea existente dentro del proyecto Work, asignándole prioridad correctamente. Los pasos principales incluyen:

- Navegar hasta la lista de tareas en el proyecto Work.
- Seleccionar el penúltimo ítem.
- Abrir el menú de opciones.
- Asignar la prioridad esperada al ítem.

La prueba asegura que el cambio de prioridad se refleje visualmente mediante un cambio de color en la tarea editada. A continuación, el código de la prueba:

```

1  async openPenultimateTodoItemPriorityMenu() {
2    await this.penultimateTodoItemLI.hover();
3    await this.penultimateTodoItemMoreOptionsButton.click();
4    await this.penultimateTodoItemPriorityButton.click();
5    await this.loader.waitFor({ state: 'hidden' });
6
7    // Esperar a que el color cambie a 'rgb(22, 139, 184)'
8    await expect(this.penultimateTodoItemLI.locator('.ItemContentDiv'))
9      .toHaveCSS('color', 'rgb(22, 139, 184)');
10  }

```

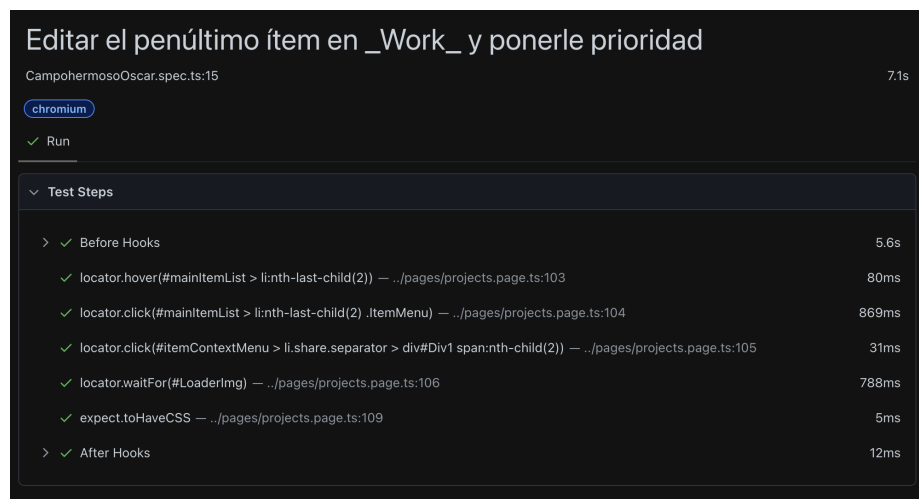


Figura 2.2: Resumen de los pasos de la prueba en Playwright, realizados en chromium

2.1.2. Eliminar el Segundo Ítem

Este caso de prueba valida que un usuario pueda eliminar correctamente una tarea específica, en este caso el segundo ítem de la lista de tareas. Los pasos principales incluyen:

- Navegar hasta la lista de tareas.
- Seleccionar el segundo ítem.
- Abrir el menú de opciones.

- Eliminar la tarea seleccionada.

La prueba asegura que el ítem eliminado desaparezca de la lista y que no mantenga atributos residuales en el sistema. También valida que el mensaje de confirmación se muestre correctamente y que el proceso de eliminación sea manejado sin problemas incluso si involucra múltiples apariciones del cargador (`loader`). A continuación, el código de la prueba:

```

1  async deleteSecondTodoItem() {
2      const itemIdOfSecondTodoItem = await
3          this.secondTodoItemLI.getAttribute('itemid');
4
5      if (itemIdOfSecondTodoItem === null) {
6          throw new Error('Second todo item does not have an itemid attribute');
7      }
8
9      await this.secondTodoItemLI.hover();
10     await this.secondTodoItemMoreOptionsButton.click();
11     await this.secondTodoItemDeleteButton.click();
12
13     // Wait for the info message to appear
14     await this.messageInfo.waitFor({ state: 'visible' });
15
16     // Assert the message content
17     await expect(this.messageInfo).toHaveText('Info. Item has been Deleted');
18
19     // Wait for the loader to disappear (second appearance)
20     await this.loader.waitFor({ state: 'hidden' });
21
22     // Ensure the item is no longer in the list
23     await expect(this.mainContentTasks).not.toContainText(itemIdOfSecondTodoItem);
24 }

```

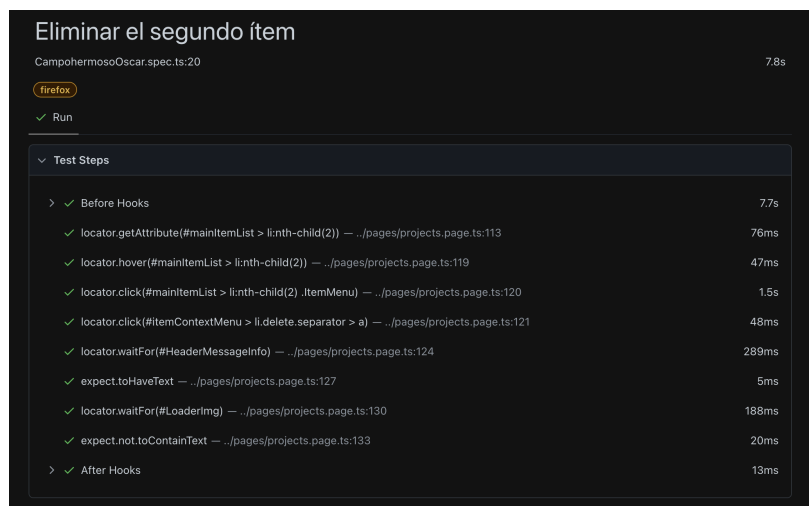


Figura 2.3: Resumen de los pasos de la prueba en Playwright, realizados en firefox

2.2. Conclusión

Las pruebas E2E realizadas demuestran que las funcionalidades clave de Todo.ly operan correctamente en conjunto. Estas pruebas confirman que las tareas pueden ser editadas y eliminadas de manera efectiva, asegurando que los cambios realizados por el usuario se reflejen consistentemente en la interfaz. Además, se validó que el flujo de eliminación maneja correctamente las transiciones entre el mensaje de confirmación y el cargador. La implementación de Playwright permitió automatizar los casos de prueba, optimizando su ejecución y validación.

Capítulo 3

Anexos

En esta sección se incluyen recursos adicionales relevantes para la implementación, ejecución y análisis de las pruebas de API y pruebas E2E en la aplicación Todo.ly.

3.1. Repositorio de Gestión de Calidad

El repositorio asociado a este trabajo contiene la implementación de las pruebas realizadas en Postman en formato json, así como documentación en formato \LaTeX .

- <https://github.com/OscarCampohermoso/gestion-calidad-3er-parcial.git>

3.2. Repositorio de Pruebas E2E con Playwright

Este repositorio incluye el código y configuración necesarios para ejecutar las pruebas E2E en la aplicación Todo.ly utilizando Playwright. Contiene scripts, documentación técnica y configuraciones útiles para replicar el entorno de pruebas. Está disponible en el siguiente enlace:

- https://github.com/CarrascoAlejandro/CALI_TodoLy_Playwright.git

3.3. Instrucciones para Clonar y Ejecutar los Repositorios

A continuación, se describen los pasos básicos para clonar y ejecutar los repositorios mencionados:

3.3.1. Clonar Repositorios

Ejecute los siguientes comandos en su terminal para clonar los repositorios:

```
# Clonar el repositorio de gestión de calidad
git clone https://github.com/OscarCampohermoso/gestion-calidad-3er-parcial.git
```

```
# Clonar el repositorio de pruebas E2E
git clone https://github.com/CarrascoAlejandro/CALI_TodoLy_Playwright.git
```

3.3.2. Ejecución de Pruebas en Playwright

1. Instalar las dependencias necesarias:

```
npm install
```

2. Ejecutar las pruebas:

```
npx playwright test
```

3.3.3. Ejecución de Pruebas de API con Newman

1. Instalar las dependencias necesarias:

```
npm install -g newman
```

2. Ejecutar las colecciones de Postman:

```
newman run API-testing-OscarCampohermoso.postman_collection.json \  
-e QAenv-OscarCampohermoso.postman_environment.json --color off
```