ACTIVIDAD

Pruebas en JavaScript y Ajax

Cesar Jesus Lara Avila

Calagua Mallqui Jairo Andre

20210279F

Facultad de Ciencias, Universidad Nacional de Ingeniería

noviembre del 2023

Comenzamos la actividad agregando al GemFile de nuestro proyecto gem Jasmine ejecutamos bundle como siempre, así:

```
74    group :development, :test do
75      gem 'jasmine-rails'
76    end
```

PREGUNTA:

¿Cuáles son los problemas que se tiene cuando se debe probar Ajax?

La principal dificultad al probar Ajax radica en la asincronía de las llamadas. Las pruebas deben esperar a que las respuestas del servidor lleguen antes de poder evaluarlas. Además, se debe simular el comportamiento del servidor para pruebas unitarias, y asegurarse de que las llamadas Ajax se realizan correctamente sin depender del servidor real.

PREGUNTA:

¿Qué son los stubs, espías y fixture en Jasmine para realizar pruebas de Ajax?

Stubs: Son funciones falsas utilizadas para simular el comportamiento de funciones reales durante las pruebas. En Jasmine, se usan para reemplazar funciones existentes por versiones controladas y predefinidas.

Espías: Son funciones que permiten rastrear llamadas a otras funciones y verificar cómo fueron utilizadas. En el contexto de Jasmine, se usan para verificar si una función ha sido llamada, cuántas veces y con qué argumentos.

Fitxture: Para el caso de pruebas, una “fixture” es un conjunto de datos o elementos predefinidos utilizados como base para las pruebas. En Jasmine, las fixtures se usan para simular datos o entornos específicos, como el contenido HTML o datos de respuesta de una llamada Ajax.

Sobre el código brindado:

PREGUNTA:

¿Qué hacen las siguientes líneas del código anterior?,¿Cuál es el papel de spyOn de Jasmine y los stubs en el código dado?

```
it('calls correct URL', function() {
  spyOn($, 'ajax');
  $('#movies a').trigger('click');
  expect($.ajax.calls.mostRecent().args[0]['url']).toEqual('/movies/1');
});
```

Este código espía la función $.ajax. Luego, simula hacer clic en un enlace dentro de #movies y verifica si la URL llamada por $.ajax coincide con '/movies/1'.

```
beforeEach(function() {
  let htmlResponse = readFixtures('movie_info.html');
  spyOn($, 'ajax').and.callFake(function(ajaxArgs) {
    ajaxArgs.success(htmlResponse, '200');
  });
  $('#movies a').trigger('click');
});
it('makes #movieInfo visible', function() {
  expect($('#movieInfo')).toBeVisible();
});
it('places movie title in #movieInfo', function() {
  expect($('#movieInfo').text()).toContain('Casablanca');
});
```

Aquí se carga una fixture HTML, se espía '$.ajax' y se reemplaza con una función falsa que simula una respuesta exitosa. Luego, se simula un clic en un enlace y se comprueba si el elemento '#movieInfo' se vuelve visible y si contiene el título de la película 'Casablanca'

Función setup en movie_popup.js:

La función setup en movie_popup.js agrega un elemento oculto div al final de la página y asigna un evento de clic a los enlaces dentro de #movies. Su objetivo es preparar el escenario para mostrar información detallada de una película cuando se hace clic en un enlace.

Stubs y fixtures en Jasmine y Jasmine-jQuery:

En Jasmine, las fixtures se pueden cargar usando loadFixtures y readFixtures para proporcionar datos simulados para las pruebas. Jasmine-jQuery extiende Jasmine con funciones de ayuda para trabajar con el DOM durante las pruebas.

Identificar filas ocultas de la tabla usando solo JavaScript del lado del cliente:

Puedes identificar las filas ocultas de la tabla verificando si el atributo display de las filas es igual a 'none' usando JavaScript.

Código AJAX para crear menús en cascada:

El código AJAX para menús en cascada depende de cómo esté estructurado el backend y cómo se manejan las solicitudes. Generalmente, se usa un evento de cambio en el primer menú para desencadenar una solicitud AJAX que devuelve datos específicos para rellenar el segundo menú.

Extensión de la función de validación en ActiveModel:

Para generar automáticamente código JavaScript que valide las entradas del formulario antes de enviarlo, se pueden utilizar bibliotecas como client_side_validations en Rails para manejar las validaciones del lado del cliente basadas en las validaciones del modelo del lado del servidor. Esto puede incluir verificaciones de campos obligatorios, longitudes, valores numéricos y expresiones regulares para asegurar datos válidos antes de enviar el formulario.