Protogrid

Tutorial de inicio rápido de Protogrid (avanzado)

El equipo de Protogrid se complace en que desee conocer mejor Protogrid. Esta guía le muestra cómo implementar lógica personalizada utilizando el lenguaje de programación JavaScript.

Contenido

0. Requisitos previos

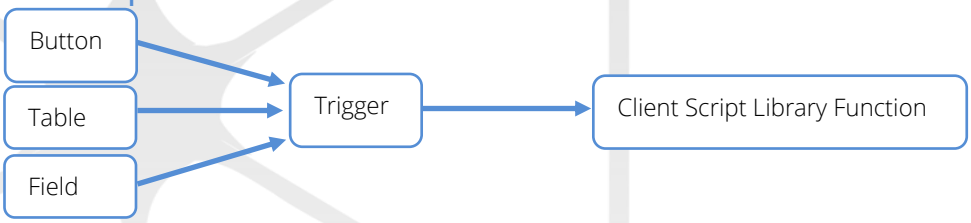
Esta guía da por sentado que ya ha completado el curso introductorio y, por lo tanto, dispone de su propio entorno Protogrid con la aplicación «Document Repository».

Además, se requiere cierta experiencia previa con JavaScript, HTML, CSS y tecnologías web, ya que aquí no se tratarán los conceptos básicos.

Enlaces útiles: https://www.javascripttutorial.net/, https://www.geeksforgeeks.org/html/html-css/

Protogrid

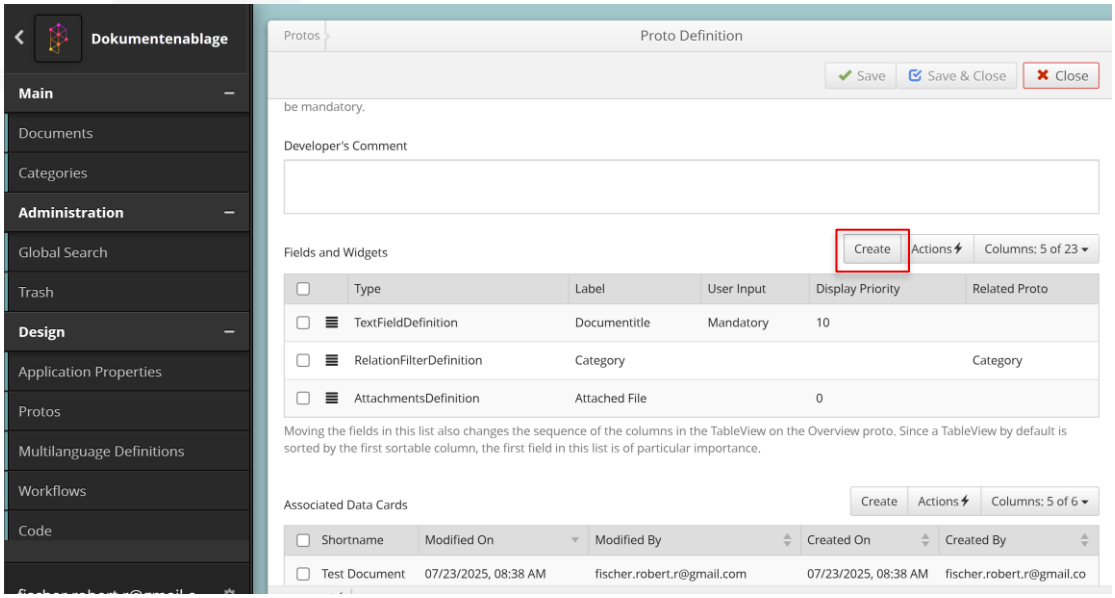
1. Client ScriptLibraries



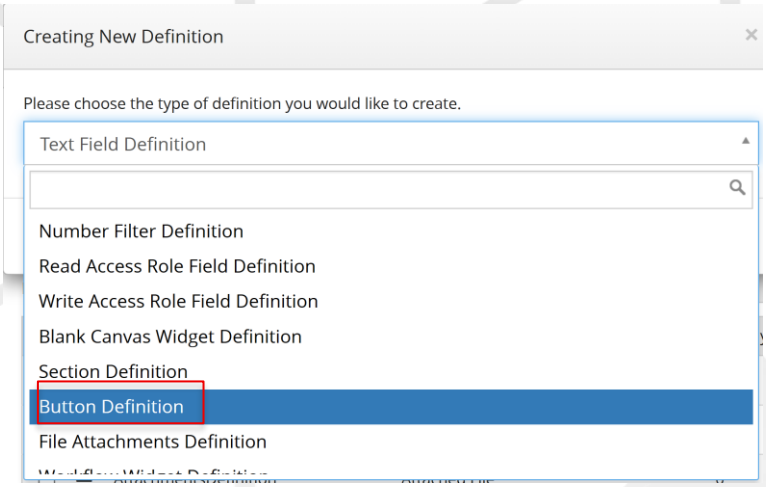
La biblioteca de scripts del cliente (CSL) es la principal forma de implementar funciones personalizadas en Protogrid. Para ejecutar su función, necesita tener un botón, una tabla o un campo que active un disparador, que a su vez puede ejecutar una función en su CSL. Es importante saber que las CSL solo se pueden ejecutar cuando el usuario hace clic en un botón y se ejecutan localmente en el dispositivo del usuario, por lo que este también podrá ver el código. En capítulos posteriores se presentarán otras posibilidades. Las CSL están pensadas para manipular la interfaz de usuario para cálculos, funciones de comodidad u otras cosas.

2. Creación de los componentes necesarios

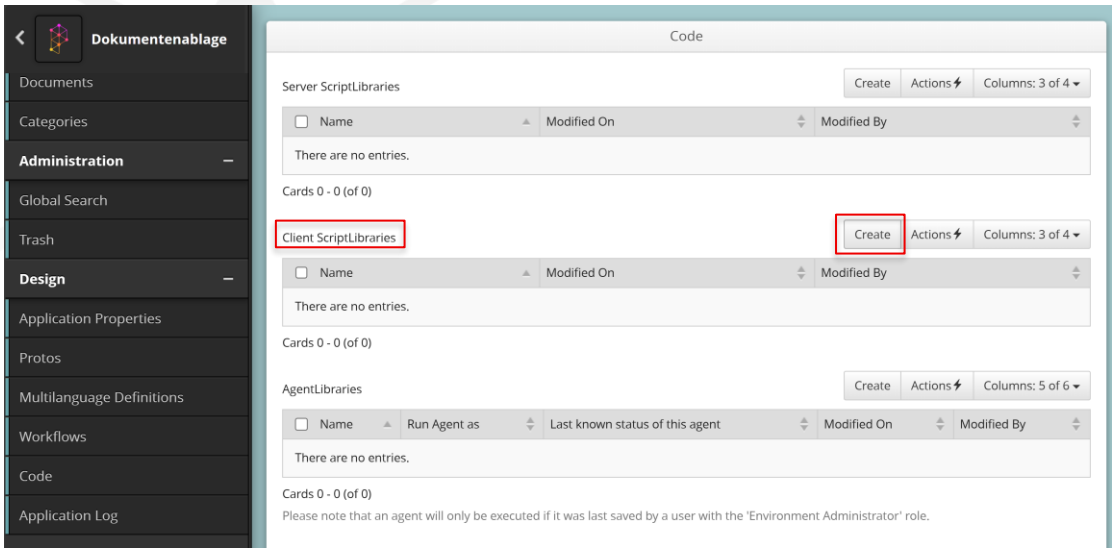
Veamos cómo crear un botón. Ve a «Protos» y selecciona el Data Proto de «Document», que definimos en el último tutorial. A continuación, desplázate hacia abajo hasta «Fields and Widgets» y haz clic en Create.



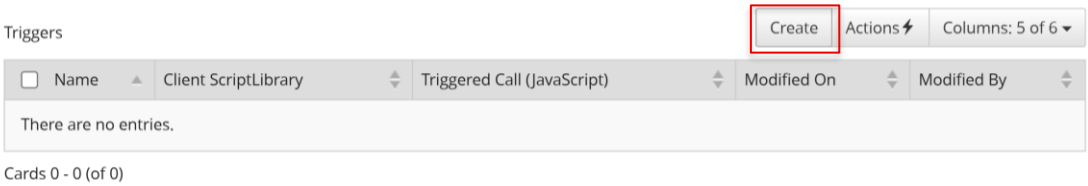
A continuación, seleccione «Definición del botón» y asigne al botón un nombre que se mostrará al abrir el Proto. En este ejemplo, elegimos «Comprobar título» y, como texto de ayuda, «Comprueba si el título del documento es lo suficientemente largo».



Para crear una CSL, vaya a «Código», desplácese hacia abajo hasta «Bibliotecas de scripts del cliente» y cree una nueva CSL. Asigne a la CSL un nombre significativo; en este ejemplo, elegimos «Funciones de verificación». A continuación, guarde la CSL creada.



Para crear un activador, desplácese hacia abajo hasta «Activadores» y cree uno nuevo. Asigne al activador un nombre significativo; en este ejemplo, elegimos «Activador de verificación de título». Por ahora, puede dejar los demás campos vacíos.



Cuando creaste el CSL, es posible que hayas notado el gran campo debajo llamado «Código». Aquí es donde almacenamos nuestra lógica personalizada, que está escrita en JavaScript. Ya contiene muchas funciones útiles que se pueden reutilizar. Más abajo encontrarás una lista con todas las funciones

disponibles descritas. Una función importante a tener en cuenta es «after_load()», que se ejecuta cada vez que se abre o se recarga el Proto.

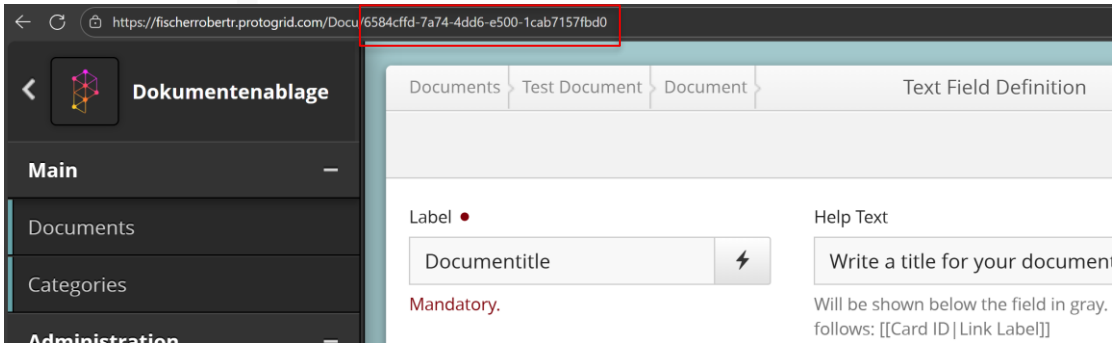
Ahora, en el campo Código, creamos una nueva función en la parte superior llamada «check_documentTitle()».

```
function check_documentTitle(){
    // Get document title
    var documentTitle;

    // check length and set output field.
    if(documentTitle.length > 5){

    }else{

    }
}
```

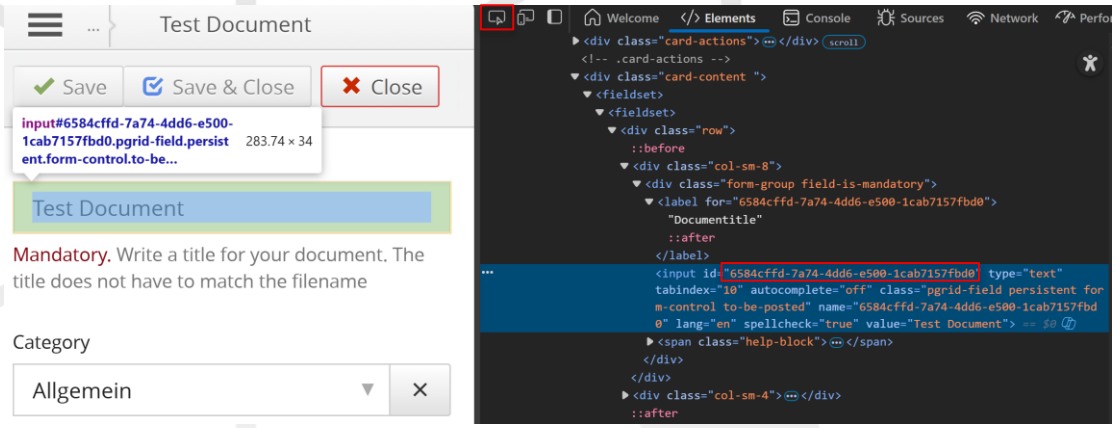
Ahora uno podría preguntarse cómo accedemos al título del documento, ya que la función no toma ninguna variable de entrada. Responderemos a esto en el próximo capítulo.

3. Obtención de identificadores de campo (claves)

Cada campo que definimos para un Proto se identifica mediante una clave. La clave es una palabra/cadena única que se genera para cada campo. Para encontrar la clave de cualquier campo, lo más conveniente es ir al Proto de datos asociado y abrir el campo correspondiente desde la sección «Campos y widgets». Si luego miramos en la barra de direcciones del navegador, podemos ver la clave de ese campo. En nuestro caso, sería «6584cffd-7a74-4dd6-e500-1cab7157fbd0».



Un método alternativo sería abrir una tarjeta de documento (no el Proto) y utilizar las herramientas de desarrollo del navegador para encontrar la clave. Por lo general, las herramientas se abren pulsando F12. En el código HTML del sitio web, utilice la herramienta de selección para seleccionar el campo deseado y lea la clave y la etiqueta «id».



4. Leer valores de campo

Ahora que sabemos cómo identificar un campo, podemos empezar a leer valores. Si has leído el código ya generado del CSL, habrás notado la función «csl_helpers.get_value()». Solo tenemos que pasar la clave y nos devolverá el valor del campo.

```
    // Get document title
    var documentTitle = csl_helpers.get_value("6584cffd-7a74-4dd6-e500-1cab7157fbd0");
```

5. Escribir valores de campo

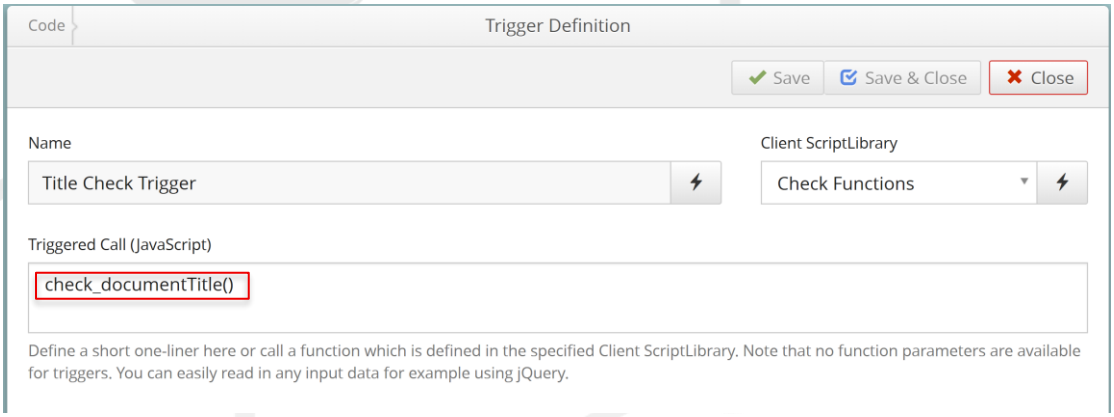
Para escribir un valor en un campo, también disponemos de una función auxiliar predefinida llamada «csl_helpers.set_value()», que funciona de manera análoga. Pero primero tenemos que crear una nueva definición de campo de texto para el prototipo de datos del documento, donde podamos mostrar el resultado de nuestra función. Para este ejemplo, lo llamaremos «Validación» y dejaremos todos los demás campos vacíos. Después de identificar la clave para este nuevo cuadro, podemos completar nuestro código con:

```
function check_documentTitle(){
    // Get document title
    var documentTitle = csl_helpers.get_value("6584cffd-7a74-4dd6-e500-1cab7157fbd0");

    // check length and set output field.
    if(documentTitle.length > 5){
        csl_helpers.set_value("e6003db3-3f86-4287-d454-3a2adbb02873", "Document Title is
correct.");
    }else{
        csl_helpers.set_value("e6003db3-3f86-4287-d454-3a2adbb02873", "Document Title is too
short. Should contain more than 5 characters.");
    }
}
```

Tenga en cuenta que «csl_helpers.set_value()» toma dos argumentos de entrada, la clave y el valor a establecer, y tampoco proporciona ningún valor de retorno.

Ahora, si pulsa el botón creado, observará que no ocurre nada. Esto se debe a que primero debemos vincular todos los elementos creados. Vuelva a la pestaña Código y abra el disparador creado. En «Biblioteca de scripts del cliente», seleccione «Funciones de comprobación» o el nombre que haya dado a su CSL. En el campo de texto situado debajo, denominado «Llamada activada (JavaScript)»,

introduzca «check_documentTitle()». Esto ejecutará la función que hemos definido anteriormente y NO ejecutará nada más de nuestro CSL.



A continuación, vaya a la definición de nuestro botón en el Proto de datos y establezca el desencadenador en «Title Check Trigger». También mostrará convenientemente el CSL asociado y la función llamada entre corchetes.

Ahora, si todo se ha hecho correctamente, puede pulsar el botón y el campo de validación debería mostrar un resultado adecuado en función de la longitud del título del documento. Si no es así, el siguiente capítulo le ayudará. No obstante, debe leerse independientemente.

6. Encontrar errores

Si no ocurre nada, lo primero que debes comprobar es:

- ¿El botón está vinculado al activador correcto?
- ¿El activador está vinculado al CSL correcto?
- ¿El activador está llamando a la función correcta?
- ¿El nombre de la función está escrito correctamente, sin errores tipográficos?
- ¿El nombre de la función va seguido de corchetes? Debe ser «check_documentTitle()» y NO «check_documentTitle».

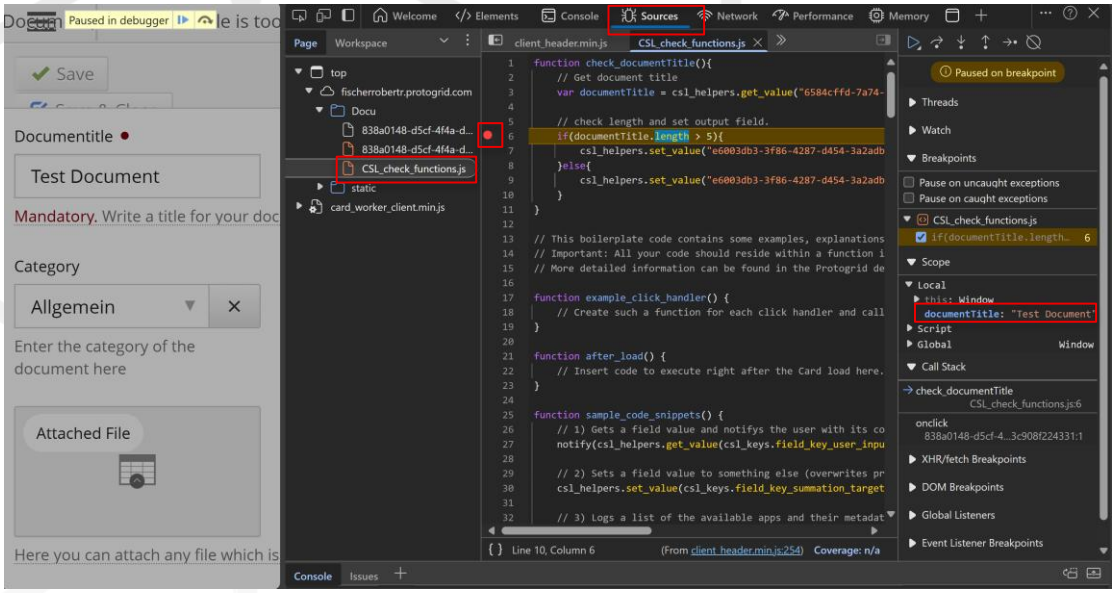
Protogrid no mostrará ningún mensaje de error si no se configuran.

Si obtiene un resultado incorrecto o cree que el problema proviene de su código JavaScript, podemos utilizar las herramientas de desarrollo. A continuación, explicamos el proceso para cualquier navegador basado en Chromium (Chrome, Edge, etc.), pero es probable que el proceso sea similar para otros navegadores.

Vuelva a abrir las herramientas de desarrollo con F12 y navegue hasta la pestaña «Fuentes» y, a continuación, en la parte izquierda, vaya a >top >yourenvironment.protogrid.com >Docu. Aquí verá algunos archivos JavaScript. Los que comienzan por «CSL_» son sus CSL vinculados. Si no aparecen aquí, es que no están vinculados correctamente. A continuación, haga clic en el CSL que desea investigar, en nuestro caso sería «CSL_check_functions.js».

Volverá a ver el código que ha escrito anteriormente. Si pasa el cursor justo a la izquierda de los números de línea, verá que aparecen puntos rojos. Puede hacer clic en ellos para establecer uno de

ellos en una línea determinada. Se denominan «puntos de interrupción» y detendrán la ejecución de su código en esa línea específica. Yo he establecido el mío aquí, en la instrucción if. Esto significa que si pulso el botón de mi Proto, el código se ejecutará y se detendrá justo antes de la instrucción if. Tan pronto como se detiene la ejecución, podemos ver a la derecha, en >Ámbito >Local, todas las variables que hemos definido y sus valores en ese momento. Los valores actuales también se muestran junto al código JavaScript.



Esto es todo lo que necesitas para encontrar cualquier error en tu código. Puedes ignorar las otras pestañas de las herramientas de desarrollo, ya que no son necesarias para desarrollar funciones personalizadas de Protogrid. Ten en cuenta que cualquier cambio que realices aquí en el código no se guardará, sino que deberá realizarse en el campo de código del CSL correspondiente.

7. Inserción de elementos personalizados

En Protogrid también es posible insertar tu propio código HTML/CSS en cualquier tarjeta en la que esté habilitado CSL. Podemos hacerlo utilizando jQuery. Utilizamos «$(document).ready()» antes de ejecutar la inserción para asegurarnos de que el sitio se ha cargado completamente y, a continuación, utilizamos la función «append()» para insertar nuestro HTML.

En este ejemplo, he creado un nuevo widget en blanco y, a través de su clave, podemos insertar un elemento de color. El elemento se inserta tan pronto como pulsamos el botón de validación, ya que he puesto el nuevo código en la misma función.

```
$(document).ready(function() {
    $('#8c93347e-2d6a-42d1-cb17-9a883711df9e').append('<button style="color: red">Click
    Me</button>');
});
```

8. Interactuar directamente con el back-end

También podemos utilizar jQuery para interactuar directamente con el backend de Protogrid a través de la API JSON. En este ejemplo, nos enviaremos un correo electrónico a nosotros mismos cuando hagamos clic en el botón «Validar». Introducimos el siguiente código en la misma función:

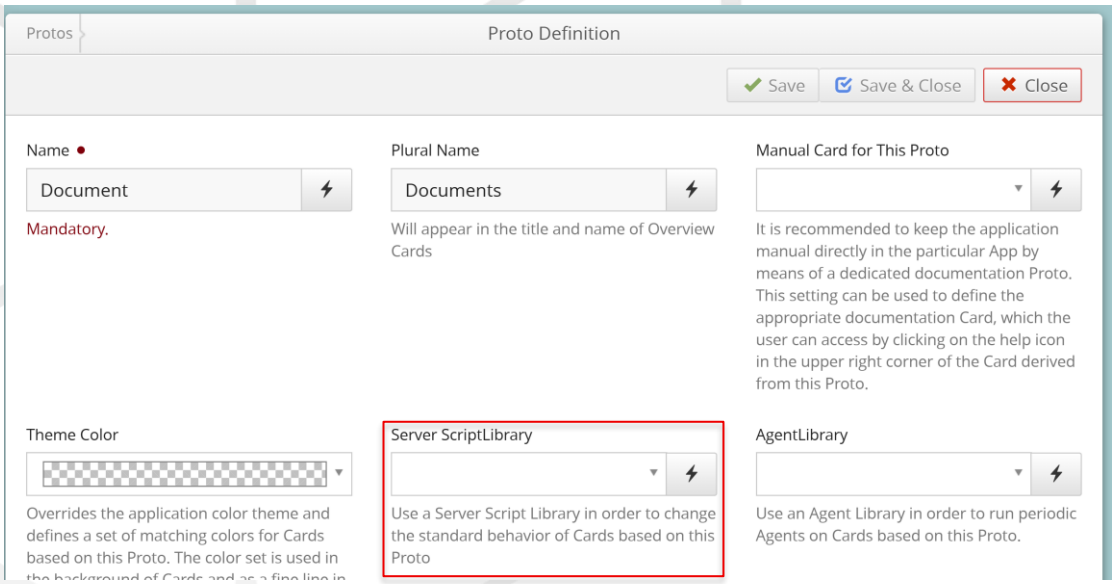
```
$.post({
    url: "https://example.protogrid.com/api/v2/apps/appc55b3773-5503-4c41-a9aa-
3a561c40fa04/mailsend",
    data: JSON.stringify({
        "to": ["User 1 <user1@example.com>", "User 2 <user2@example.ch>"],
        "subject": "Reminder",
        "body_text_plain": "Reminder: Document has been validated",
    }),
    contentType: 'application/json; charset=utf-8'
}).done(function (response) {
    alert("Data: " + JSON.stringify(response));
});
```

Cabe señalar que, para que esto funcione, el soporte técnico de Protogrid debe configurar el relé de correo de su entorno para el servidor de correo de su organización. Puede obtener información más detallada aquí.

9. Server Script Libraries

Las bibliotecas de scripts del servidor se ejecutan en el servidor en lugar de en el cliente, como su nombre indica. Están pensadas para utilizarse en funciones que requieren un nivel elevado de seguridad, ya que el usuario no puede ver el código.

Para crear una biblioteca de scripts de servidor, vaya a la pestaña «Código», haga clic en «Crear» y asígnele un nombre. Aquí volverá a tener código pregenerado. Las dos funciones principales a tener en cuenta son «on_render()» y «on_save()», que se ejecutan cuando el Proto se renderiza o se guarda en el servidor. Toda su lógica deberá escribirse en el cuerpo de estos métodos. La biblioteca recién definida solo se ejecutará en Protos, donde se ha vinculado dentro del Proto de datos, donde ya ha definido todos sus campos y widgets.



La función «get_shortname()» devuelve el nombre corto de una tarjeta vinculada. Los nombres cortos se utilizan como representación de la tarjeta en los campos de relación y etiqueta, y Protogrid los genera automáticamente de forma predeterminada si no se sobrescriben en el cuerpo de esta función.

Esto resulta útil con la función «get_sorting_string()», que devuelve una cadena que se utiliza para clasificar y ordenar las tarjetas en un campo de relación. Piensa en el menú desplegable que creamos en el manual básico.

Protogrid utiliza la función «get_title()» para establecer el título que se muestra en la parte superior de la tarjeta. También puedes implementar tus propias reglas aquí.

10. Agent Libraries

Las bibliotecas de agentes son la única forma de ejecutar código sin interacción previa del usuario. Se ejecutan cada 60 minutos y no pueden ejecutarse durante más de 1 minuto. El equipo de Protogrid puede aumentar este tiempo si se solicita.

Para ello, deberá crear una biblioteca de agentes, asignarle un nombre, vincular una cuenta sin derechos administrativos y vincular la biblioteca al Proto que debe ejecutar . Es importante vincular una cuenta con derechos de acceso mínimos, ya que la biblioteca se ejecutará desde esta cuenta.

La biblioteca de agentes se ejecutará en cualquier Proto vinculado con un campo DateTime, que se encuentre en el pasado y aún no se haya ejecutado. Ejecutará todo lo que se defina en «on_schedule()».

Un caso de uso común de las bibliotecas de agentes es comprobar cualquier Proto con información sensible al tiempo y tomar medidas si ha caducado.

11. Acceso desde sistemas externos a través de JSON-API

La API JSON nos permite interactuar directamente con los datos de Protogrid a través de solicitudes http de forma segura. Aquí mostraremos cómo hacerlo en Python, pero también es posible con cualquier otro lenguaje de programación que admita solicitudes http. Para empezar, instala la biblioteca de solicitudes ejecutando esto en la línea de comandos:

```
pip install requests
```

A continuación, en un nuevo archivo Python, escribimos el siguiente código.

```
import requests
url = "https://yourenvironmentname.protogrid.com/api/v2/authenticate"
headers = dict(username="testuser@example.com", password="test_password")
req = requests.post(url, headers=headers)
response = req.text
cookie = req.cookies['session']

print(response)
```

Al leer la URL, podemos ver que estamos realizando nuestra solicitud a una subpágina. En este caso, llamamos al servicio de autenticación. En el encabezado también incluimos el nombre de usuario y la contraseña. A continuación, el servidor responderá si la autenticación se ha realizado correctamente. También establecerá una cookie de sesión, lo que nos permite realizar solicitudes a otras subpáginas sin volver a autenticarnos, siempre que proporcionemos la cookie de sesión. Por lo tanto, la guardamos como una variable.

Si el inicio de sesión se ha realizado correctamente, debería ver algo como esto en la salida:

```
{"errors":[],"Protogrid_environment_version":"2.16.7e","result":"Login successful!"}
```

Ahora que tenemos la cookie de sesión, podemos realizar una llamada a un punto final específico. En nuestro ejemplo, utilizamos «api/v2/apps», que devolverá todas las aplicaciones y su información, que podrá ver en la página de inicio de su entorno Protogrid.

```
# The cookie variable was set above in the authentication example.
url = "https:// yourenvironmentname.protogrid.com/api/v2/apps"
req = requests.get(url, cookies={"session":cookie})
response = req.text

print(response)
```

No solo puede leer información, sino que también puede crear nuevas tarjetas. Para ello, necesita obtener el ID de la aplicación. Lo encontrará en la parte inferior izquierda, en >Esta tarjeta >Mostrar propiedades. A continuación, puede utilizar la clave del prototipo de la tarjeta que desea crear y rellenar los campos que desee. No es necesario rellenar todos los campos, salvo los obligatorios.

```
# The cookie variable was set above in the authentication example.
url = "https:// yourenvironmentname.protogrid.com/api/v2/apps/app77344ff9-0633-4c6d-b2e4-96c3f8968448/Cards"
payload = {
  "Proto_key": "afa1c744-0289-4666-bb02-9a03fd25fc01",
  "design_elements": [
    {
      "definition_key": "6584cffd-7a74-4dd6-e500-1cab7157fbd0",
      "value": "JSON_API_DOCUMENT"
    },
    {
      "definition_key": "e6003db3-3f86-4287-d454-3a2adbb02873",
      "value": "JSON_TEST"
    },
  ]
}
response = requests.post(url=url, cookies={"session":cookie}, json=payload)
```

Para obtener una lista completa de todos los puntos finales disponibles e información más detallada sobre la API JSON, consulta la wiki.

12. Conclusión

Ahora ya conoce todas las posibilidades de interactuar con Protogrid de forma automatizada. Durante este proceso, ha ejecutado código en el lado del cliente y del servidor, ha interactuado directamente con la API JSON y ha ejecutado tareas programadas en segundo plano. Puede encontrar información más detallada sobre todas las funciones de Protogrid en protogrid.wiki.

¿Le gustaría implementar algo más complejo, pero no sabe exactamente si es posible ni cómo hacerlo? El equipo de Protogrid está siempre a su disposición. Envíe un correo electrónico a Protogrid-customer-support@ategra.ch o simplemente llámenos al +41 44 392 21 20.