Protogrid

Protogrid QuickStart Tutorial (Advanced)

The Protogrid team is pleased that you would like to get to know Protogrid better. This guide shows you how to implement custom Logic using the programming language JavaScript.

Content

0. Prerequisites

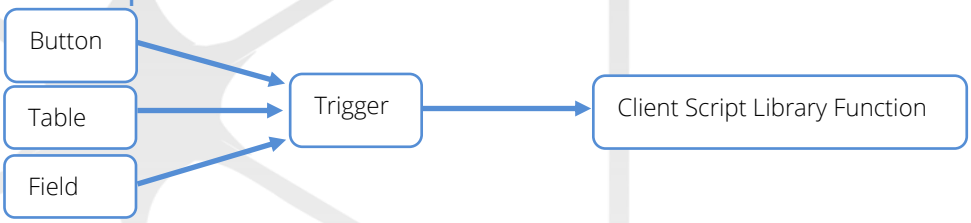
This guide assumes, that you already have completed the Primer and hence have your own Protogrid Environment with the “Document Repository” application.

Additionally, some initial experience with JavaScript, HTML, CSS and web technologies is required, as we won’t be covering the basics here.

Useful links: https://www.javascripttutorial.net/, https://www.geeksforgeeks.org/html/html-css/

Protogrid

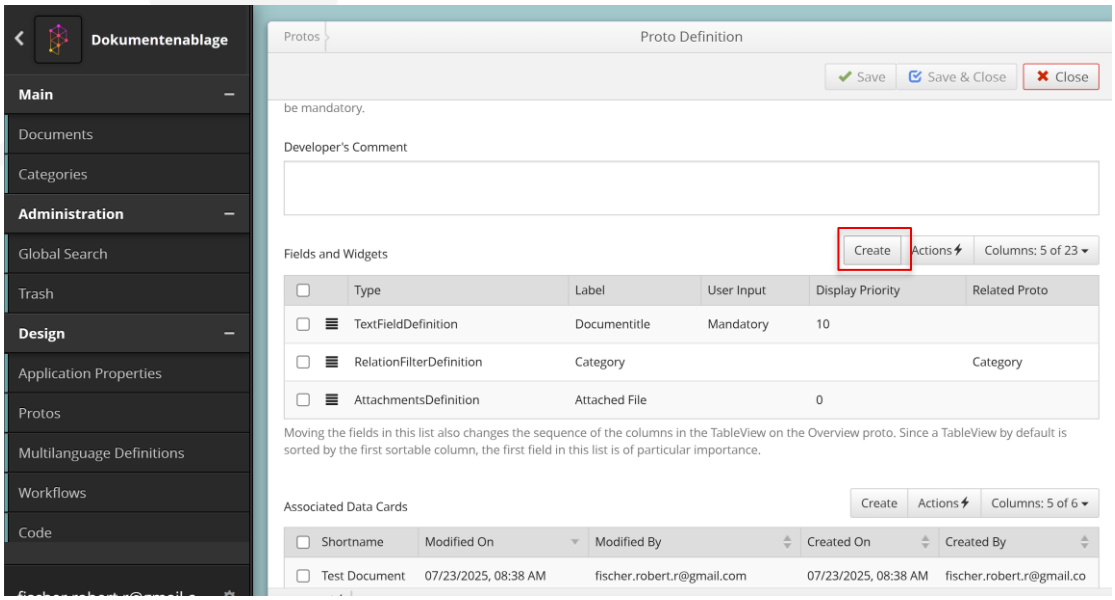
1. Client ScriptLibraries



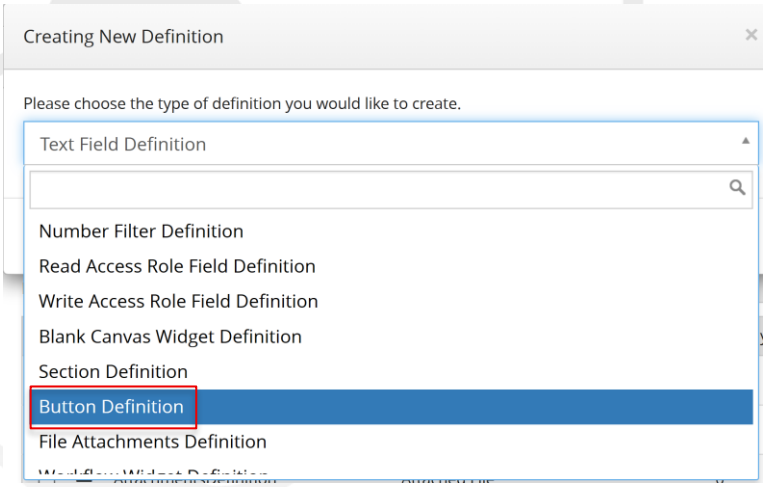
The Client ScriptLibrary (CSL) is the main way of implementing custom Functions in Protogrid. In order to execute your function, you need to have a button, Table or Field which calls a trigger, which then can execute a function in your CSL. It is important to know that CSL's can only be run when the user clicks a button and they will run locally on the device of the user, hence the user will also be able to see the code. Later chapters, other possibilities will be introduced. The CSL's are intended to manipulate the User Interface for calculations, comfort functions or other things.

2. Creating the Necessary Components

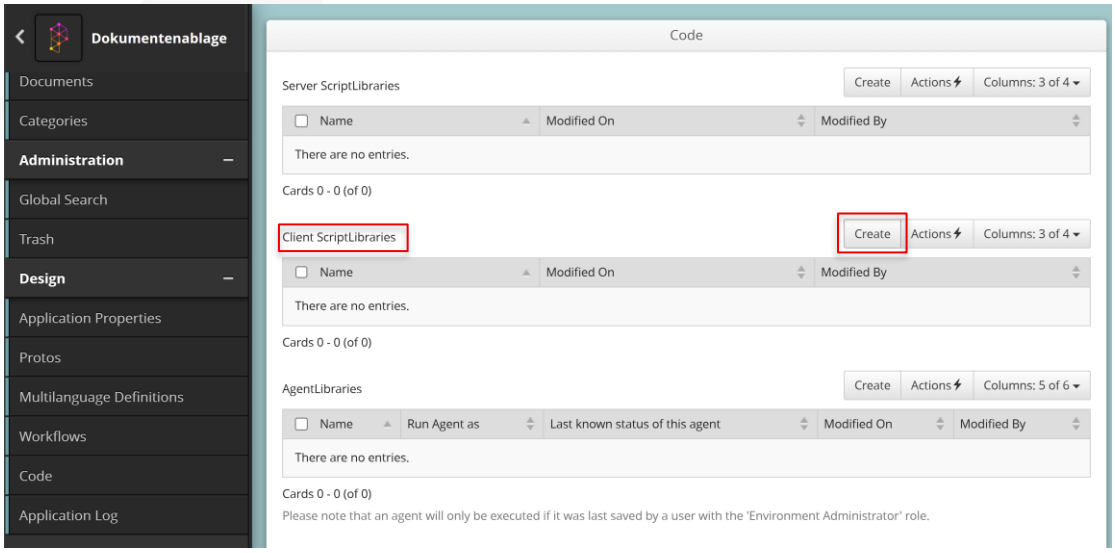
Let's take a look at creating a button. Go to the "Protos" and select the Data Proto of "Document", which we defined in the last Tutorial. Then scroll down to "Fields and Widgets" and click Create.



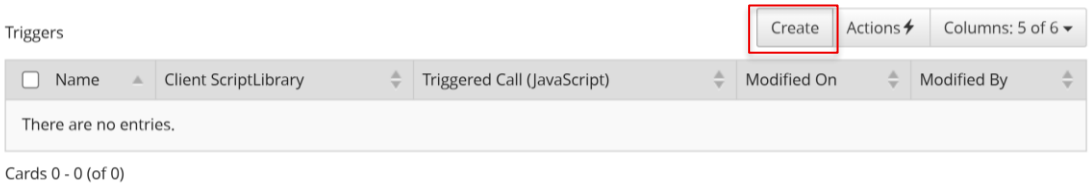
Then select “Button Definition” and give the Button a name which will be displayed when you open the Proto. We choose “Check Title” and for the Help text “Checks if the Document title is long enough” for this example.



To create a CSL go to “Code” and scroll down to “Client ScriptLibraries” and create a new CSL. Give the CSL a meaningful name, for this example we choose “Check Functions”. Then save the created CSL.



To create a Trigger, scroll down to “Triggers” and create a new Trigger. Give the Trigger a meaningful name, for this example we choose “Title Check Trigger”. You can leave the other fields empty for now.



When you created the CSL, you might have noticed the big field below called “Code”. This is where we store our custom Logic, which is written in JavaScript. It already contains a lot of useful functions, which

can be reused. Further down you will find a list with all available functions described. One important function to keep in mind is “after_load()”, which executes every time the Proto is opened or reloaded.

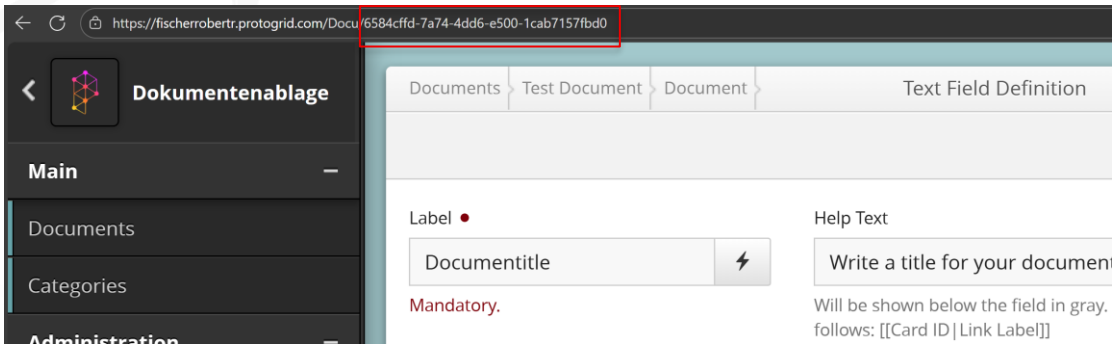
Now in the Code field we create a new function at the top called “check_documentTitle()”.

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

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

    }else{

    }
}
```

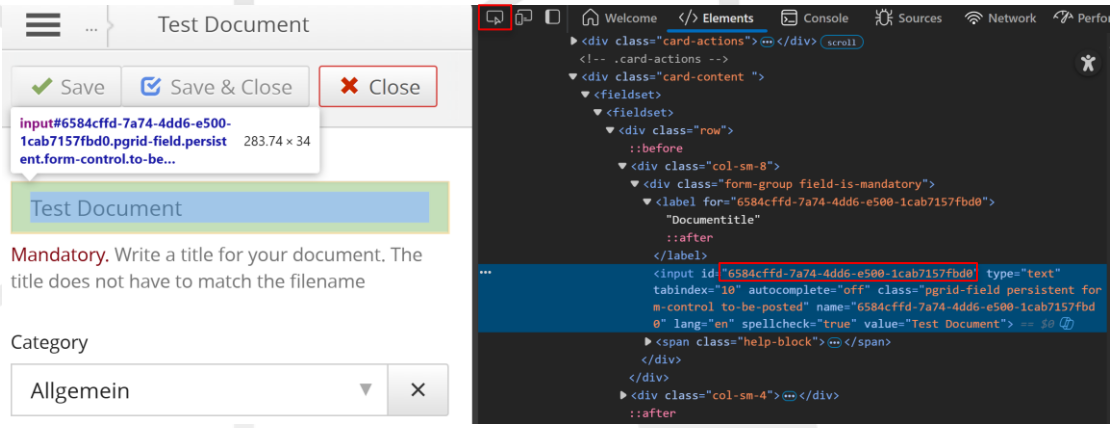
Now one might wonder how we access the Document Title since the function does not take any input variables. We will answer this in the next chapter.

3. Getting Field Identifiers (Keys)

Every field we define for a Proto is identified by a key. The key is a unique word/string which is generated for every field. To find the key for any field it is the most convenient, if we go to the associated data Proto and open the corresponding field from the “Fields and Widgets” section. If we then look in the URL bar of the Browser, we can see the Key for that Field. In our case that would be “6584cffd-7a74-4dd6-e500-1cab7157fbd0”.



An alternative method would be to open a document Card (not the Proto) and use the developer tools from the browser to find the Key. Usually, the tools are opened by pressing F12. In the HTML code of the website use the selector tool to select your desired Field and read the Key und the “id” label.



4. Read Field Values

Now that we know how to identify a field, we can start reading values. If you have read through the already generated code of the CSL you might have noticed the "csl_helpers.get_value()" function. We just pass the key, and it will return the value of the field.

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

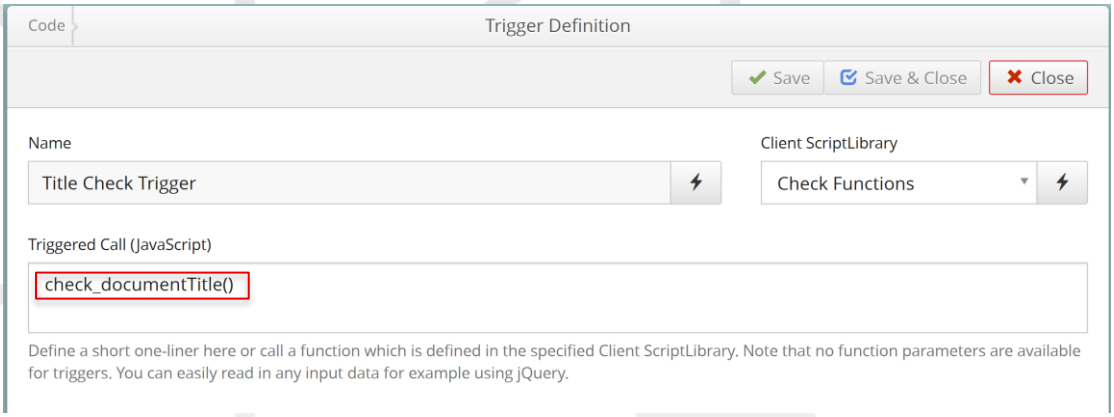
5. Write Field Values

To write a value to a field, we also have a predefined helper function called "csl_helpers.set_value()", which works analogously. But first we need to create a new Text field Definition for the Data Proto of Document, where we can display the output of our function. For this example, we call it "Validation" and leave all other Fields empty. After identifying the key for this new box, we can complete our code to:

```
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.");
    }
}
```

Note that "csl_helpers.set_value()" takes two input arguments, the Key, and the value to set and also provides no return value.

Now if you press the created button, you might notice that nothing will happen. This is because we first need to link all the created elements. First go to the Code tab again and open the created trigger. Under "Client ScriptLibrary" we select "Check Functions" or whatever you called your CSL. In the text field below called "Triggered Call (JavaScript)" we enter "check_documentTitle()". This will execute the function we previously defined and will NOT execute anything else from our CSL.



Then go to the definition of our button in the data Proto and set the Trigger to "Title Check Trigger". It will also conveniently display the associated CSL and the called function in square brackets.

Now if everything has been done correctly you can press the button, and the validation field should display an appropriate output depending on the length of the Document Title. If not, the next chapter will help you. Nonetheless it should be read regardless.

6. Finding Mistakes

If nothing happens the first thing you should check is:

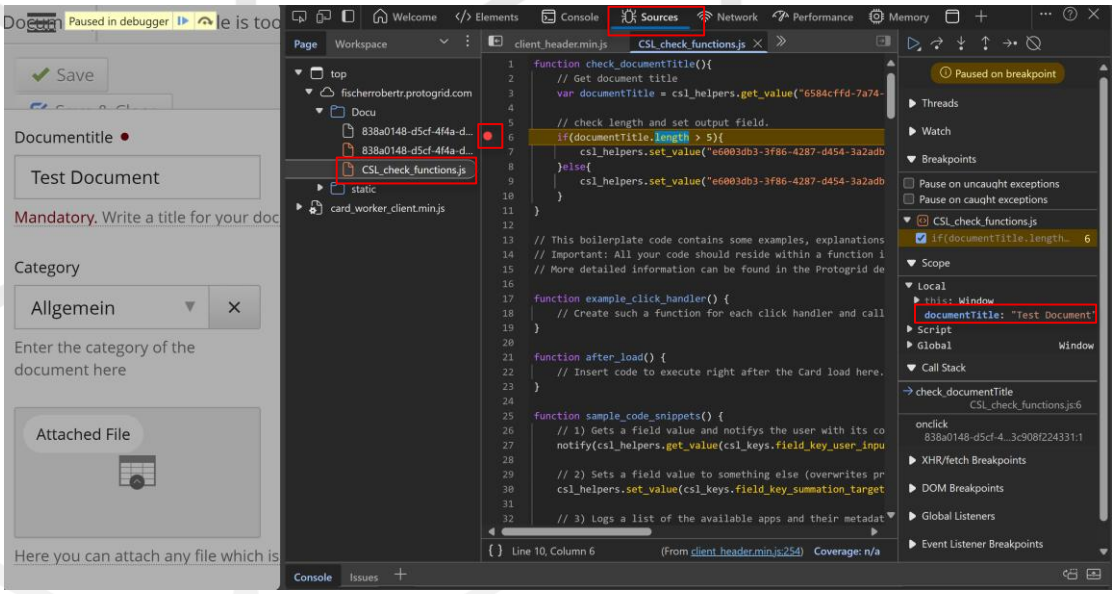
- Is the button linked to the correct trigger?
- Is the trigger linked to the correct CSL?
- Is the trigger calling the correct function?
- Is the function name written correctly with no typos?
- Is the function name followed by brackets? It should be "check_documentTitle()" and NOT "check_documentTitle".

Protogrid will not display any error messages if these are not set.

If you have an incorrect output or think the problem stems from your JavaScript code, we can use the developer tools. Here we explain the process for any browser which stems from chromium (Chrome, Edge, etc.), but the process is likely similar for other browsers.

Open the developer tools with F12 again and navigate to the sources tab and then on the left go to >top >yourenvironment.protogrid.com >Docu. Here you will see a few JavaScript files. Those that start with "CSL_" are your linked CSL's. If they don't show up here, they are not linked correctly. Then click on the CSL you want to investigate, in our case that would be "CSL_check_functions.js".

Then you will once again see the code that you have previously written. If you hover just to the left of the line numbers, you will see red dots appearing. You can click them to set one of them at a certain line. These are called "Breakpoints", and they will halt the execution of your code at this specific line. I have set mine here at the if-statement. This means that if I press the Button in my Proto, the code will execute and stop just before the if-statement. As soon as the execution pauses, we can see to the right under >Scope >Local all variables we have defined and their values at this point in time. The current values are also displayed next to the JavaScript code.



This is everything you need to find any errors in your code. You can ignore the other tabs in the developer tools, as they are not needed to develop custom Protogrid functions. Note that any changes to the code that you make here will not be saved and instead should be made in the code field of the corresponding CSL.

7. Inserting custom Elements

It is possible in Protogrid to also insert your own HTML/CSS code on any Card where a CSL is enabled. We can do this by using jQuery. We use "$(document).ready()" before running inserting to ensure the site has fully loaded and then use the "append()" function to insert our HTML.

In this example I created a new blank widget and via its key we can then insert a colored element. The element is inserted as soon as we press the validate button, as I put the new code in the same function.

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

8. Interacting directly with the Back-End

We can also use jQuery to interact directly with the backend of Protogrid via the JSON-API. In this example we will send an Email to ourselves when we click the validate button. We put the following code in the same function:

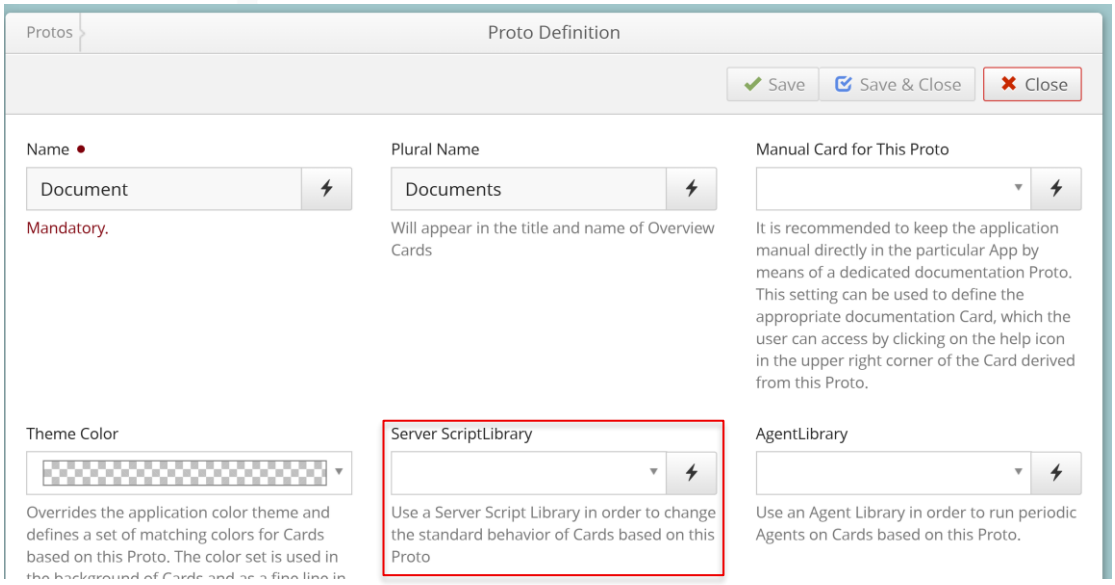
```
$.post({
    url: "https://example.protogrid.com/api/v2/apps/appc55b3773-5503-4c41-a9aa-
3a561c40fa04/mailsend",
    data: JSON.stringify({
        "to": ["User 1 <user1@example.com>", "User 2 <user2@example.ch>"],
        "subject": "Reminder",
        "body_text_plain": "Reminder: Document has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

It must be noted that in order for this to work the mail relay for your Environment has to be set for your organization's mail server by Protogrid support. You can more detailed information here.

9. Server Script Libraries

Server Script Libraries run on the Server instead of the Client as the name implies. They are intended to be used for functions which require an elevated level of security, since the code cannot be seen by the user.

To create a Server Script Library go to the "Code" tab and click create and give it a name. In here you will once again have pre-generated code. The two main functions to look out for are "on_render()" and "on_save()", which both run when the Proto is rendered or saved on the server. All your logic will have to be written in the body of these methods. The newly defined Library will only run on Protos, where it has been linked inside the data Proto, where you have already defined all your fields and widgets.



The "get_shortname()" function returns the shortname of a linked Card. Shortnames are used as a representation of the Card in Relation and Tag Fields, and they are by default auto generated by Protogrid if not overridden in the body of this function.

This becomes useful with the "get_sorting_string()" function, which returns a string which is used to sort and order Cards in a relation field. Think of the dropdown menu we created in the Primer.

The function “get_title()” is used by Protogrid to set the Title that is shown at the top of the Card. You can also implement your own rules here.

10. Agent Libraries

Agent Libraries are the only way of running code, without prior user interaction. They will run every 60 minutes and are not allowed to run for more than 1 minute. The Protogrid Team can increase this on request.

To do this, you will have to create an Agent Library, name it, link an account without administrative rights and link the library to the Proto which it should execute . It is important to link an account with minimal access rights, since the library will be executed from this account.

The Agent Library will run on any linked Proto with a DateTime Field, which lies in the past and has not already been executed on. It will run everything which is defined in “on_schedule()”.

A common use case for Agent Libraries is to check any Protos with time sensitive information and take action if they have expired.

11. Accessing from outside systems via JSON-APII

The JSON-API allows us to interact directly with Protogrid data via http requests in a secure way. Here we will demonstrate how to do this in python, but it is also possible with any other programming language which supports http requests. To get started, install the requests library by running this in the command line:

```
pip install requests
```

Then in a new python file we write the following code.

```
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)
```

When you read the URL, we can see, that we are making our request to a subpage. In this case we call the authentication service. In the header we also include the username and password. The server will then respond if the authentication has been successful. It will also set a session cookie, which allows us to make requests to other subpages, without re-authenticating, if we provide the session cookie. Hence, we save it as a variable.

If the login was successful, you should see something like this in the output:

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

Now that we have the session cookie we can make a call to a specific Endpoint, in our example we use “api/v2/apps” which will then return all apps and their information you can see on the landing page of your Protogrid Environment.

```
# 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)
```

Not only can you read information, but you can also create new Cards. For that you need to get the Application ID. You can find it in the bottom left under >This Card >Show Properties. Then you can use the Key of the Proto of the Card you want to create and fill out any fields. You do not have to fill out every field except for the mandatory ones.

```
# 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)
```

To get a full list of all available Endpoints and more detailed information about the JSON-API check out the wiki.

12. Conclusion

Now you know all possibilities of interacting with Protogrid in an automated fashion. During this process you executed code on the Client and Server side, interacted with the JSON-API directly and executed timed tasks in the background. You can find more detailed information about the full functionality of Protogrid on Protogrid.wiki.

Would you like to implement something more complex, but don't know exactly whether and how this is possible? The Protogrid team is always at your disposal. Send an e-mail to Protogrid-customer-support@ategra.ch or simply call us on +41 44 392 21 20.