**Custom connectors**

1. **Agenda**

*Today I’m gonna to tell you about custom connectors and its possibilities. We will create our first custom connector together, create our first flow to see, what we’ve just created and test it.*

*By the end of this session, hopefully you will have learned about different types of authorization, we will find out what a swagger file is and why it’s needed, understand what Postman is, the difference between webhook and polling trigger and other useful stuff.*

*Let’s get started!*

1. **Overview of a custom connectors**

*First of all, let’s recap what we already know. A connector is a wrapper around an API that allows the underlying service to talk to Microsoft Power Automate, Microsoft Power Apps, and Azure Logic Apps. It provides a way for users to connect their accounts and leverage a set of prebuilt actions and triggers to build their apps and workflows.*

*Basically, a custom connector is the same.*

**Actions**

***Actions****are the things you want the workflow to do once the trigger takes place.* *Actions are changes directed by a user. All actions directly map to operations defined in the swagger. We will talk a little bit later, what swagger file is.*

**Triggers**

*A****trigger****is the event that happens and starts the process. That’s what you’re asking the workflow to look out for.*

1. **Authentication types**

*Every time you create a connection to a connector in an app or in a flow, you use the configured authorization type in that connector, you establish a connection to the API in order to communicate with it later.*

*This is an important part of creating custom connectors, without which you won’t be able to use the API to exchange data.*

*There are 4 different types of authentication that are currently supported:*

* *No authentication*
* *Basic authentication*
* *Api Key based authentication*
* *Oauth 2.0*

*Let’s talk a little bit deeper, what each type means.*

1. **No authentication**

*The user will not need any authentication to create a connection to the connector. Any anonymous user can use your connector in this case.*

1. **Basic Authenctication**

*This is the simplest type of authentication, where the user just has to provide the username and password to create the connection.*

*The values you enter under*Parameter label*will be the names for the "username" and "password" fields that the user will see while creating the connection.*

1. **Api Key based authentication**

*The user will need to provide the API key while creating the connection. The*Parameter location*field gives you the option to send the API key to your service in headers or a query string when the request is made.*

*The value you enter under*Parameter label*will be the name of the field the user will see. When a request is made to your service, a header with name 'ApiKey' and value as entered by the user will be added to the request.*

1. **Oauth 2.0**

*This is the most frequently used type, which uses the Oauth 2 authentication framework to authenticate with the service. Before using this authentication type, you'll need to register your application with the service so that it can receive access tokens for the users.*

*I won’t spend much time here. If your API requires the use of Oauth 2 authentication, I’ve placed a few articles under this presentation, so feel free to use it and read more.*

* **Identity Provider**: The user interface currently supports multiple identity providers. The general ones are Generic Oauth 2, which can be used for any service, and Azure Active Directory, which can be used for all Azure services. A few specific identity providers like Facebook, GitHub, Google, and so on, are also supported.
* **Client id**: The client ID of the application you have registered with the service.
* **Client secret**: The client secret of the application you have registered with the service.
* **Authorization Url**: The API authorization endpoint to authenticate with the service.
* **Token Url**: The API endpoint to get the access token after the authorization has been done.
* **Refresh Url**: The API endpoint to refresh the access token once it has expired.

1. **Possible ways to create a custom connector**

Graphical user interface, text, application

Description automatically generated

*Today I’m gonna to tell you about 2 of them:*

*1) Import an OpenAPI file*

*2) Import a Postman collection*

*First, we will create a custom connector from an Open API file, then will move on to a PostMan.*

1. **Create a Custom connector from an OpenAPI Definition (Swagger file)**

*OpenAPI Specification is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including:*

(показать на примере, открыть скачанный open API file)

* ***Available endpoints*** *(/users) and operations on each endpoint (GET /users, POST /users)*
* *Operation parameters Input and output for each operation*
* *Authentication methods*
* *Contact information, license, terms of use and other information.*

**Useful note!**  
*First of all, if you are tasked with creating a custom connector, I recommend that you ask a client to provide one of these:  
1) Swagger File*

*2) API documentation*

*This will give you some great, useful information to get started on creating and configuring a custom connector.*

1. **Import an OpenAPI file**

*(Demonstration)*

*The goal was to show, what a swagger file is and how you can use it to start building your own custom connector. And now let’s leave it as it is for now and move on to the most fascinating part of our session – Postman, Postman collection, testing REST APIs and creating a flow.6. Import a Postman collection*

1. **Import a Postman collection**

*(Demonstration)*

*For our purposes we will use Online REST API, which is directly accessible to everyone.*

(open Online REST API)

*We will use the first one for “users” and recreate these four basic actions (POST, GET, PATCH, DELETE).*

*The last thing you have to know, this API requires you to use Bearer Token for authorization.*

(open Postman)

*Postman is an API platform for building and using APIs. Essentially, it is a developer tool that makes it easy for devs to create, share and test APIs.  
Once we are done with a Postman platform, we will export a postman collection. And you will soon see, that a Postman collection is the same as a swagger file.*

(Create GET, POST, PATCH, DELETE, GET by ID actions)

(Export Postman collection)

(Open Downloaded file)

(Import a Postman collection. Create, save, close and open once again)

*Well folks, this is our workspace for creating and configuring custom connector.*

*First of all, let's start by setting up authorization and establishing our connection.*

*There is no option for us to choose “Bearer token”, as you remember API requires us to use exactly this one.*

*In this case and this is a very opportune moment to tell you about Policies.*

*Policies can be used to modify the behavior of connectors at runtime. For instance, policies are used to enforce throttling limits on API calls, to route calls to different endpoints, and in our case we will configure policy to pass authorization and establish a connection.*

(Authorization was established)

*Now let’s have a look at our actions, how they were imported and whether we need to change anything or if it’s already perfectly created.*

(Have a look at GET)

*Looks fine.*

(Have a look at PATCH)

*Here we don’t see a body for our action. Add body “{}”*

(Have a look at DELETE)

*We will wrap our ID in curly braces.*

*Path parameters are parameters built into the path of the URL. Unlike Query Parameters, Path Parameters are NOT OPTIONAL. This is because they’re required to complete the path of the URL.*

*Query parameters are a series of parameters fixed to the end of a URL. These are URL extensions which are used to help identify different actions or content based on the passed data. To use query parameters we then add to the end of a URL, a ‘?’ (question mark) and then the parameter is used immediately afterwards. These are OPTIONAL.*

(Create new request in a Postman “GET by query”)

(Create new request in a custom connector)

*One of the last steps here, we will need* ***to configure the responses*** *for our actions. To do this, we need to go back to Postman and just copy-paste a response for each action.*

*Why do we need to customize responses?*

*Without customization you will retrieve the whole response, you will get your values. But in order to consume/to use them in a flow, you will need to write expressions. There will be no dynamic content in the flow and as you all know it's indeed inconvenient.*

(Customizing Responses)

(Responses customized)

*The last 2 points I want to draw your attention to are a swagger editor and customizing fields.*

(Swagger editor)

*For instance, you want to delete one of the parameters, or add a new one. As you can see, there is no option for us to click “Add new” or “Delete”. So, to achieve it you have 2 possible options:*

1. *You can delete action and recreate it with your changes.*
2. *Go to a swagger editor and directly make your changes there.*

(Customizing fields)

Add static value to “POST” action

(Create a flow and simply add a few actions)

1. **Path, Query, Headers, Body**

***HTTP headers****let the client and the server pass additional information with an HTTP request or response.*

1. **Triggers. Polling trigger**

*There are two types of triggers:*

* ***Polling Triggers****—These triggers call your service at a specified frequency to check for new data. When new data is available, it causes a new run of your workflow instance with the data as input.*
* ***Webhook Triggers****—These triggers listen for data on an endpoint, that is, they wait for an event to occur. The occurrence of this event causes a new run of your workflow instance.*

# ***Polling trigger*** *is easier to implement, because you don’t need any special requirements as for webhook trigger. Essentially, polling trigger is a the same GET action with memory of the current state.*

***As a workaround*** *to implement polling trigger you can add a GET action, create scheduled flow to run each N hours and in a query, based on created column (if such column exists, write an expression to get values greater than (current time – 2h)*

1. **Webhook triggers**

*To define a webhook trigger in a custom connector, you need to include three essential parts in the OpenAPI definition:*

* *A POST message that describes webhook registration*
* *Content definition for the webhook responses*
* *A DELETE message that describes the webhook tear-down process*

1. **Share a custom connector**

*People within the organization can use the custom connector just like they use other Microsoft-managed connectors.*

*Once a custom connector is created and shared with users in your organization it’s available for use in Power Platform tools (Power Automate, PowerApps)*