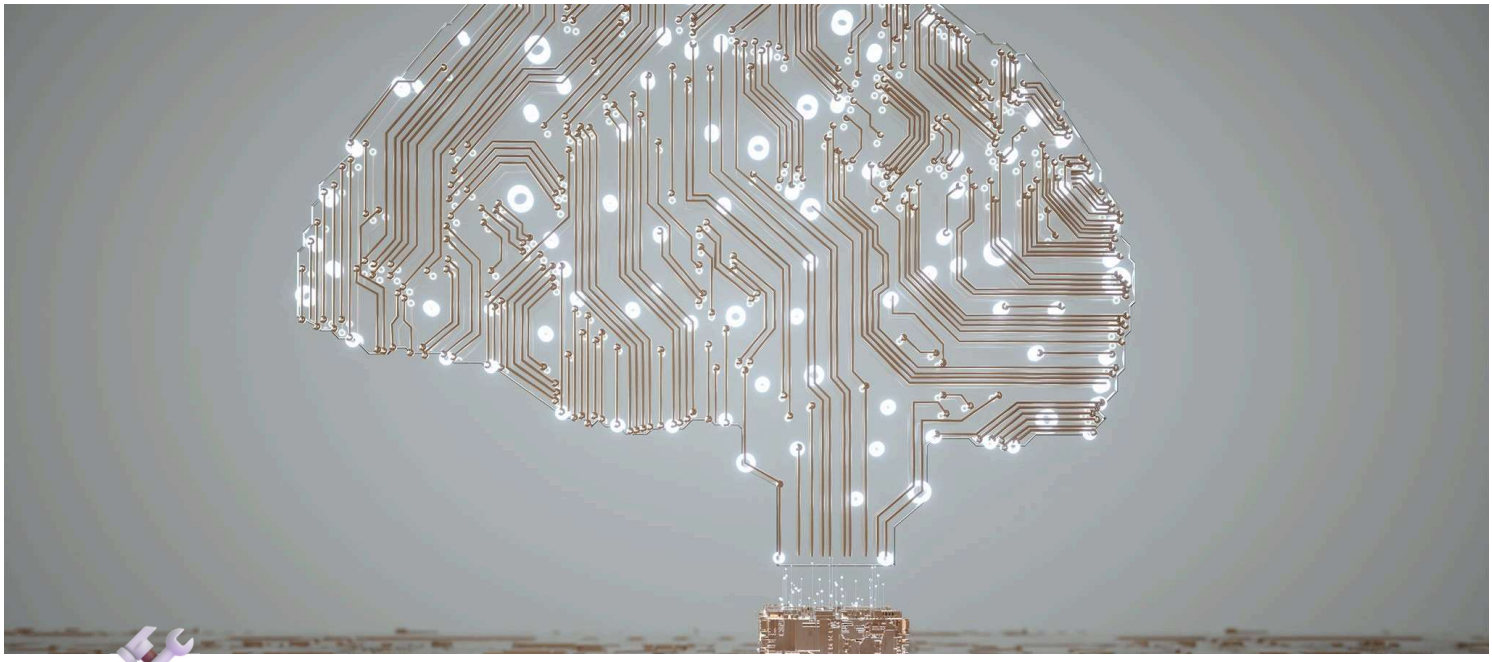


Hands On Lab #2: Metadata, Search, and Copilot

Overview

Part 1

In Part 1 of this lab, we will use the Postman client to call SharePoint Embedded Graph APIs to set metadata on containers and files. Then we'll use the search query APIs to find content. Then we'll explore how SharePoint Embedded content shows up in other M365 experiences like Office and OneDrive.

Part 2

Part 2 is all about Copilot. We'll explore how we can access SharePoint Embedded content from M365 Copilot. Then, we will go back to our sample app in VS Code and edit the code to drop-in the private preview Copilot-powered chat control into our app. Then we'll explore how you can deliver a custom, Copilot-powered chat experience inside your app.

Pre-Reqs

1. Lab #1 completed
2. Copilot requires at least one user assigned a Copilot license

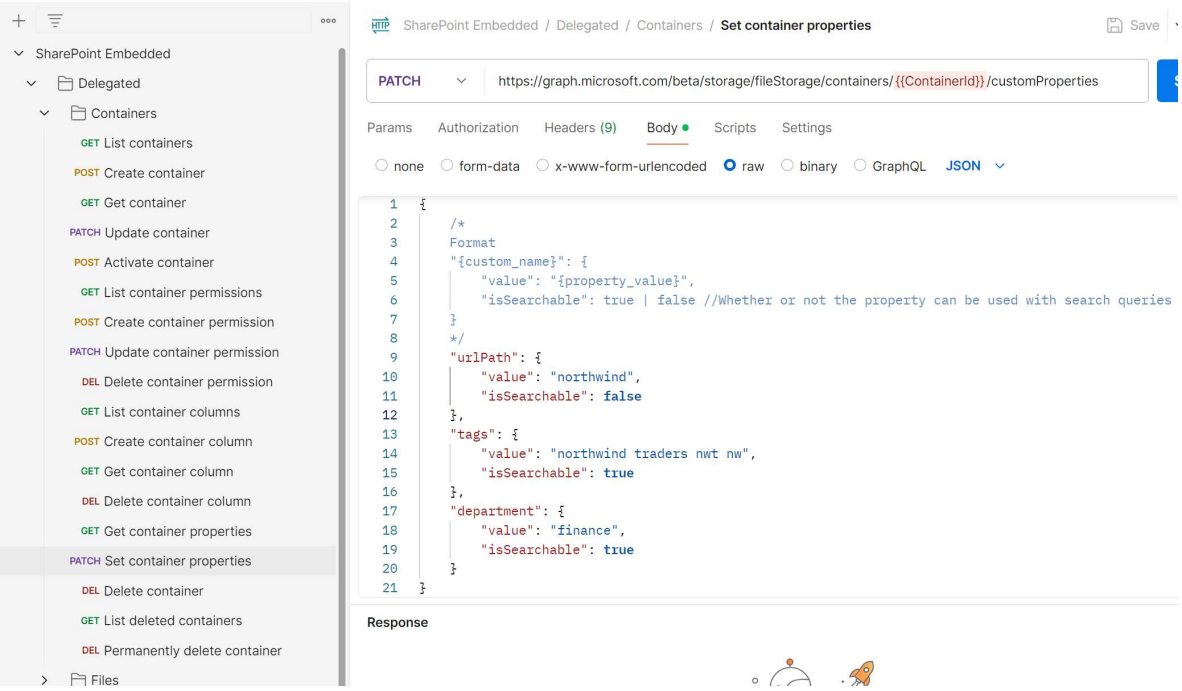
---

Steps

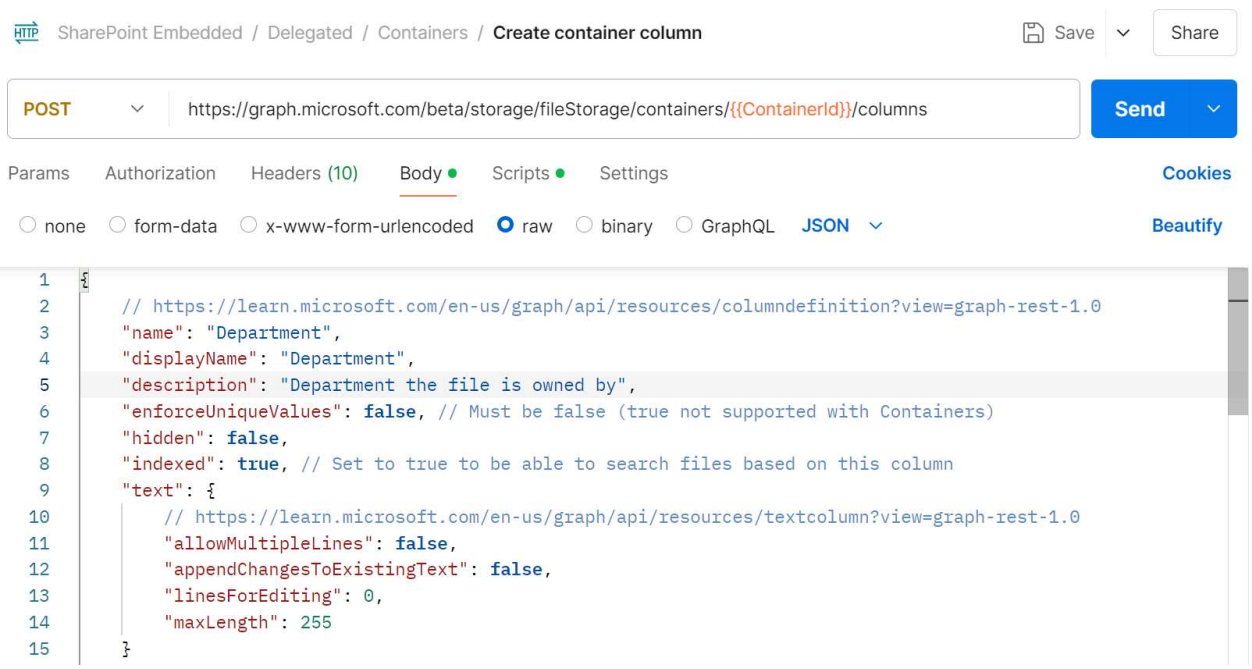
📌 Steps in VS Code are blue
Steps in Postman client are orange
Steps in a Web browser are purple

Part 1

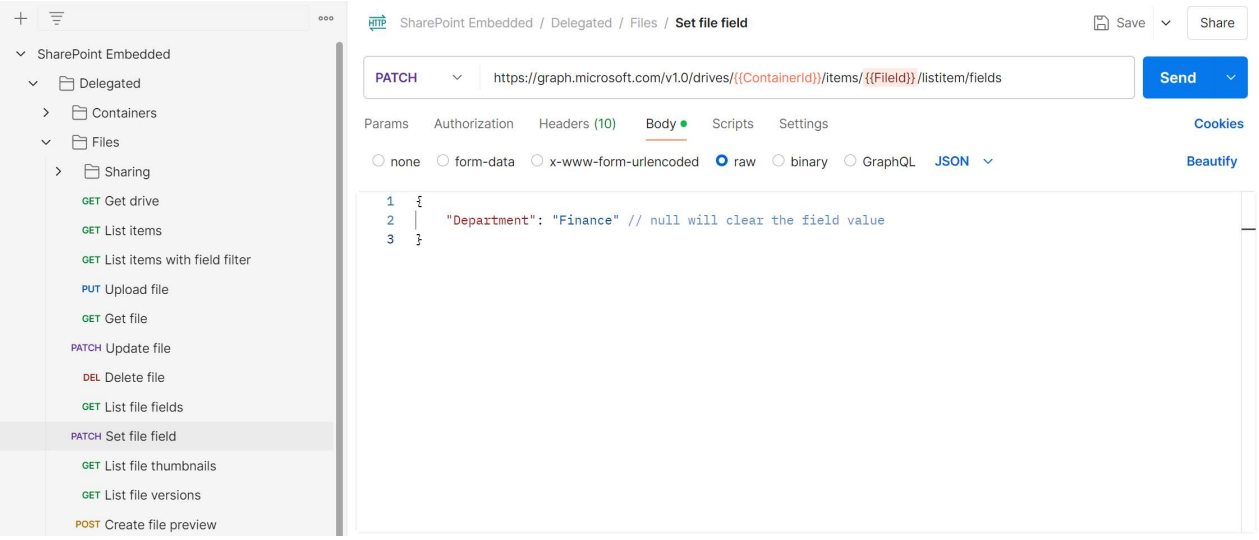
1. Ensure you still have the *ContainerId* property set in your environment variables to the container you were working with in Lab 1.
2. Use the 'Set container properties' request to set custom metadata on the container.



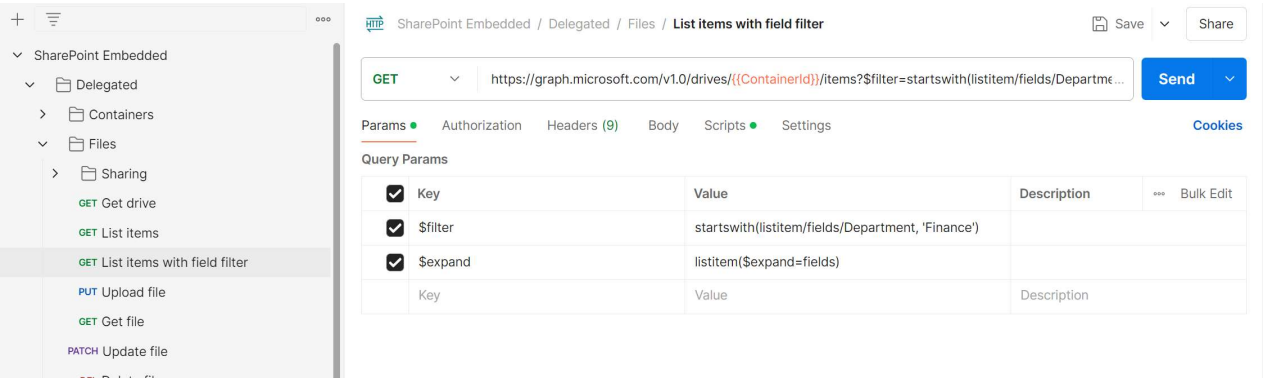
3. Use the 'Create container column' request to add the following column on your container. Notice the other column types that are supported.



4. Make sure you still have the *FileId* parameter set in your environment for the file you uploaded earlier. Use the 'Set file field' request to set values for your new columns on the file.



5. Use the 'List items with field filter' request to list files that match the metadata you just applied. On the 'Params' tab, set the $filter parameter to startswith(listitem/fields/Department, 'Finance') to locate the file you just updated.

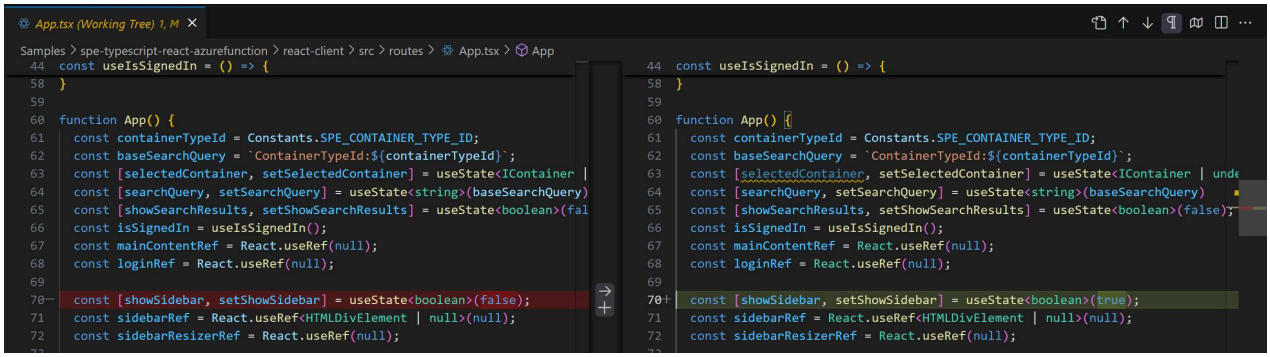


6. Use the Search requests to try the following queries. Note that search indexing is asynchronous on the SharePoint Embedded platform, just like it is for M365 content in general. Latencies are typically very fast (seconds), but sometimes things can take longer. If you aren't getting the results that you expect, try again later.
   a. Find all of your containers by {{ContainerTypeId}}
      i. "queryString": "ContainerTypeId:*{{ContainerTypeId}}*"
   b. Find a specific container by name (Title matches Container.displayName)
      i. "queryString": "Title:'My Container' and ContainerTypeId:*{{ContainerTypeId}}*"
   c. Find all files in a specific container
      i. "queryString": "ContainerId:*{{ContainerId}}*"
   d. Find a specific file by name in a specific container
      i. "queryString": "ContainerId:*{{ContainerId}}* AND Title:'*{{filename}}*'"
   e. Find files by full-text content search within a specific container
      i. "queryString": "'project' AND ContainerId:*{{ContainerId}}*"
   f. Find files by searching metadata (field values based on container columns). Note that you have to include the 'OWSTEXT' string at the end of the column name as shown below.
      i. "queryString": "ContainerId:*{{ContainerId}}* and Department:'Finance'"
7. In a Web browser, go to office.com as your admin user. Because we enabled Discoverability on the container type, you can see this content in other M365 experiences like Office. If we hadn't changed that setting, this content would not be showing up. Go ahead and check out other experiences like OneDrive, and Office clients.

Part 2

> 📌 The following steps will only work if your tenant has been added to our Copilot Private Preview. If you haven't requested that yet and want to, no problem! Fill out the SPE and Copilot Preview Signup Form or send us an email at SharePointEmbedded@microsoft.com and we can discuss getting you added. In addition to getting your tenant added to our preview, you will also need to have at least one user that is licensed for Copilot. At the moment, any user that interacts with the Copilot chat control in your application will need to have a Copilot licensed assigned to them for it to work.
>
> Form: https://forms.microsoft.com/r/rdJCrZ9DuZ

1. Back in VS Code, open **App.tsx** from the react-client/src/routes folder and update the default value of the *showSidebar* state variable to **true**.



2. Open the react-client/src/components/ChatSidebar.tsx file and update the ChatSidebar component with the following code.

```
export const ChatSidebar: React.FunctionComponent = () => {
    const [chatAuthProvider, setChatAuthProvider] = React.useState<ChatAuthProvider | undefined>();

    const onApiReady = async (api: ChatEmbeddedAPI) => {
        await api.openChat();
        ChatController.instance.addDataSourceSubscriber(dataSources => {
            api.setDataSources(dataSources);
        });
    }

    ChatAuthProvider.getInstance().then(setChatAuthProvider).catch(console.error);
    return (<>
        {chatAuthProvider && (
            <ChatEmbedded
                authProvider={chatAuthProvider}
                onApiReady={onApiReady}
            />
        )}
    </>);
}
```

3. The previous steps should now give you a working Copilot-powered chat control in your app. Go back to https://localhost:8080/containers to see it in action. Try out a few prompts in there such as "Show me my recent files" or "What do you know about oranges". You can also use the "/" character to bring up the working set and reference specific people or files.

4. The new chat control works okay, but we want to customize it a bit more so it matches the look and feel of our app and is more tailored to our needs. Update the ChatSidebar component code as follows -- this will:

   a. Set the header title to our own custom string

   b. Update the theme to match our app's colors

   c. Set some zero query prompts that show at the top of the control

   d. Set some initial suggested prompts that show near the input box on the control

   e. Set the initial instruction prompt which will tell the chat control how to behave. In this case, we are asking it to talk like a pirate

   f. Set a subscriber for when containers are selected on the /containers page that will scope the data sources to only the selected containers for subsequent prompts. This example only scopes it to containers, but you can look at the underlying API to understand how you can scope it to other things such as files or folders.

```
export const ChatSidebar: React.FunctionComponent = () => {
    const [chatAuthProvider, setChatAuthProvider] = React.useState<ChatAuthProvider | undefined>();

    const [chatConfig] = React.useState<ChatLaunchConfig>({
        header: ChatController.instance.header,
        theme: ChatController.instance.theme,
        zeroQueryPrompts: ChatController.instance.zeroQueryPrompts,
        suggestedPrompts: ChatController.instance.suggestedPrompts,
        instruction: ChatController.instance.pirateMetaPrompt,
    });


    const onApiReady = async (api: ChatEmbeddedAPI) => {
        await api.openChat(chatConfig);
        ChatController.instance.addDataSourceSubscriber(dataSources => {
            api.setDataSources(dataSources);
        });
    }

    ChatAuthProvider.getInstance().then(setChatAuthProvider).catch(console.error);

    return (<>
    {chatAuthProvider && (
        <ChatEmbedded
            authProvider={chatAuthProvider}
            onApiReady={onApiReady}
        />
    )}
    </>);
}
```

5. Try out the updated chat control in the browser. Use the zero-query and suggested prompts. On the /containers page, try selecting a specific container and see how it scopes subsequent queries to that data source. Try different prompts on the content we loaded earlier such as "Make a table summarizing recent files about fruit" or "What is the most popular vegetable". The answers you get back should be referencing your app's content in SharePoint Embedded.