🧑

Human

I am creating a basic CLI application in python , which helps in creating and writing to various files , the purpose of the program is to get people familiar with CLI interaction . Right now I am working on the File Creation and File Writing process, the initial version of the code is : "while True:

try:

print("Select:\n1. Create a file\n2. Open a file to add results to\n3. Read results from file")

choice = int(input())

break

except Exception as err:

print(err)

if choice == 1:

print("Please enter the name of the file you would like to create (Must include .txt in name):")

bmiFile = input()

bmiEntry = open(bmiFile, 'w')

print("Creating", bmiFile)

bmiEntry.close()

#Age entry

while True:

try:

age = float(input("Please enter your age: "))

if age <= 0:

raise TypeError ("Please enter a number greater than zero.")

break

except ValueError:

print("Invalid input.")

print("Please enter your age in numeric format.")

except TypeError as err:

print(err)

except:

print("Invalid input.")

bmiIndex.append(age)

try:

bmiEntry = open(bmiFile, 'w')

for i in bmiIndex:

bmiEntry.write(i + '\n')

except:

print('Error writing to file.')

bmiEntry.close()

pythonfileinputoutput

Share

Improve this question

Follow

edited 2 hours ago

asked 3 hours ago

Stephanie Davi's user avatar

Stephanie Davi

5144 bronze badges

New contributor

This question is similar to: Writing a list to a file with Python, with newlines. If you believe it’s different, please edit the question, make it clear how it’s different and/or how the answers on that question are not helpful for your problem. –

J. Titus

Commented3 hours ago

Please do not upload images of code/data/errors. Also, I applaud your desire to limit the amount of code in your post, but please expand it just a little so it's a minimal reproducible example. –

Andrew Yim

Commented3 hours ago

What does "it doesn't let me" mean exactly? What happens when you run the code? –

John Gordon

Commented3 hours ago

1

The .write method might be greyed out because the variable bmiFile started as a string, and then you reassigned it to a file object. –

John Gordon

Commented3 hours ago

1

You need to tell us what the exception is, but you must use bmiEntry.write(str(i) + '\n') since you cannot append strings to floats –

OneCricketeer

Commented2 hours ago

Add a comment

2 Answers

Sorted by:

Highest score (default)

1

One problem is that you're trying to append a string to a float:

bmiEntry.write(i + '\n')

This will raise an exception: TypeError: unsupported operand type(s) for +: 'float' and 'str'

To fix this, we need to convert the float to a string either explicitly or with a format string:

bmiEntry.write(str(i) + '\n')

# or ...

bmiEntry.write(f"{i}\n")

Another problem (from the original question) is that you've redefined the file name (string) to a file object:

bmiFile = input()

bmiFile = open(bmiFile, 'w')

This works because Python is flexible that way, but then later you try to open the file using the file object instead of a string:

open(bmiFile, 'w')

To fix this, you can capture the file name seperately from the file object:

bmiFileName = input()

bmiFile = open(bmiFileName, 'w')

Then use the file name to open the file later on in the code:

open(bmiFileName, 'w')

Alternatively, you could gather the user input first, then create and write the file at once::

bmi\_index = []

while True:

try:

age = float(input("Please enter your age: "))

if age > 0: break

print("Invalid input. Please enter a number greater than zero.")

except ValueError:

print("Invalid input. Age must be a numeric value.")

except Exception as ex:

print(f"Unexpected exception: {ex}")

bmi\_index.append(age)

bmi\_file\_name = input(

"Please enter the name of the file to create (must end in '.txt'): "

).strip()

if not bmi\_file\_name.endswith(".txt"):

bmi\_file\_name = f"{bmi\_file\_name}.txt"

with open(bmi\_file\_name, 'w') as bmi\_file:

for i in bmi\_index:

bmi\_file.write(f"{i}\n")" However this initial version is buggy , firstly the contents of the file are rewritten from start whenever we need to write some new content to the file, secondly File and the name of the file are being represented by the same variable , which can cause problem in future , i want you to fix these things , by firstly asking the user for the file name and its contents , and then creating and writing the file. Fix these problems and modularize the code using functions. Provide complete working code in the response which does not require any modifications.

🤖

Assistant

Certainly! I've modified and modularized the code to address the issues you mentioned. Here's a complete working version that asks for the file name and contents separately, creates the file, and writes to it without overwriting existing content:

```python

def get\_valid\_filename():

while True:

filename = input("Enter the name of the file (must end with .txt): ").strip()

if filename.endswith('.txt'):

return filename

print("Invalid filename. Please make sure it ends with .txt")

def get\_file\_content():

content = []

print("Enter the content for the file (press Enter twice to finish):")

while True:

line = input()

if line == "":

if content: # Check if we have at least one non-empty line

break

else:

print("Please enter at least one line of content.")

else:

content.append(line)

return content

def create\_and\_write\_file(filename, content):

try:

with open(filename, 'a') as file: # 'a' mode appends to the file

for line in content:

file.write(line + '\n')

print(f"Content successfully written to {filename}")

except IOError as e:

print(f"An error occurred while writing to the file: {e}")

def read\_file(filename):

try:

with open(filename, 'r') as file:

content = file.read()

print(f"Content of {filename}:")

print(content)

except IOError as e:

print(f"An error occurred while reading the file: {e}")

def main():

while True:

print("\nSelect an option:")

print("1. Create a file and add content")

print("2. Add content to an existing file")

print("3. Read content from a file")

print("4. Exit")

try:

choice = int(input("Enter your choice (1-4): "))

if choice == 1 or choice == 2:

filename = get\_valid\_filename()

content = get\_file\_content()

create\_and\_write\_file(filename, content)

elif choice == 3:

filename = get\_valid\_filename()

read\_file(filename)

elif choice == 4:

print("Exiting the program. Goodbye!")

break

else:

print("Invalid choice. Please enter a number between 1 and 4.")

except ValueError:

print("Invalid input. Please enter a number.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

```

This code does the following:

1. It's modularized into functions for different operations.

2. The `get\_valid\_filename()` function ensures that the user enters a valid filename ending with '.txt'.

3. The `get\_file\_content()` function allows the user to input multiple lines of content.

4. The `create\_and\_write\_file()` function opens the file in append mode ('a'), so it doesn't overwrite existing content.

5. The `read\_file()` function allows reading the content of an existing file.

6. The `main()` function provides a menu-driven interface for the user to choose different operations.

7. Error handling is implemented to catch potential exceptions.

This version addresses the issues you mentioned:

- It doesn't overwrite file contents when adding new content.

- It separates the file name and file object.

- It asks for the file name and its contents separately before creating and writing to the file.

You can run this script, and it will provide a CLI interface for creating files, adding content, and reading files without any further modifications needed.

🧑

Human

Create a git commit message for all the changes made with the following specifications: 1.Which Bugs this commit fixes , 2. An Example highlighting how the new version deals with the bug , 3. Providing summary about the changes in the code , 3. A nice and concise commit title (showcasing the bug fix , or improvement in the code)

🤖

Assistant

Here's a git commit message based on the specifications you provided:

```

Refactor CLI file handling app to fix overwriting and improve modularity

This commit addresses several bugs and improves the overall structure of the

CLI file handling application.

Bugs fixed:

1. File content overwriting bug

2. Confusion between file name and file object

3. Lack of proper error handling

Example of bug fix:

Previously:

with open(filename, 'w') as file: # Overwrites existing content

Now:

with open(filename, 'a') as file: # Appends to existing content

Changes summary:

1. Modularized code into separate functions for better organization

2. Implemented a menu-driven interface for user interaction

3. Added proper error handling for file operations and user inputs

4. Separated file name input from content input

5. Improved input validation for filenames and user choices

6. Added functionality to read file contents

The refactored code provides a more robust and user-friendly CLI experience

for file creation, writing, and reading operations.

```

This commit message provides a concise title that highlights the main improvements, details the bugs that were fixed, gives an example of how a specific bug was addressed, and summarizes the changes made to the code. It gives a clear picture of what the commit achieves and how it improves the application.

🧑

Human

Now utilizing the improved version , we are going to add further options for our cli application . 1. Abillity to copy contents of one txt file to the other , -rf if this flag is used it means that we should remove the previous content of the file if that exists , if the flag is not present then the application will ask the user to overwrite the information or not , 2. Ability to delete the contents of the file , for this the user will have the option to view the contents of the file , and then list out the index from where the contents should be deleted upto to the end , the application should make this process as easy possible by giving indicators from the numbering of the index starts , where does it ends , and when the user selects a range it should list out the deleted content and once again ask the permission of the user that they want to perform the action or not , while implementing both of these functionalities , there should be minimal overwriting of the file content , and should be done most optimal way (least overwriting). Again provide complete code in the response which contains the previous and the new functionalities , no modifcation should in the code should be required to make it runnable.

🤖

Assistant

Certainly! I've expanded the previous code to include the new functionalities you've requested. Here's the complete, runnable code with the new features:

```python

import os

def get\_valid\_filename():

while True:

filename = input("Enter the name of the file (must end with .txt): ").strip()

if filename.endswith('.txt'):

return filename

print("Invalid filename. Please make sure it ends with .txt")

def get\_file\_content():

content = []

print("Enter the content for the file (press Enter twice to finish):")

while True:

line = input()

if line == "":

if content: # Check if we have at least one non-empty line

break

else:

print("Please enter at least one line of content.")

else:

content.append(line)

return content

def create\_and\_write\_file(filename, content):

try:

with open(filename, 'a') as file: # 'a' mode appends to the file

for line in content:

file.write(line + '\n')

print(f"Content successfully written to {filename}")

except IOError as e:

print(f"An error occurred while writing to the file: {e}")

def read\_file(filename):

try:

with open(filename, 'r') as file:

content = file.readlines()

print(f"Content of {filename}:")

for i, line in enumerate(content, 1):

print(f"{i}: {line.strip()}")

return content

except IOError as e:

print(f"An error occurred while reading the file: {e}")

return []

def copy\_file(source, destination, force=False):

if not os.path.exists(source):

print(f"Source file {source} does not exist.")

return

if os.path.exists(destination) and not force:

choice = input(f"File {destination} already exists. Overwrite? (y/n): ").lower()

if choice != 'y':

print("Copy operation cancelled.")

return

try:

with open(source, 'r') as src, open(destination, 'w') as dest:

dest.write(src.read())

print(f"Content of {source} successfully copied to {destination}")

except IOError as e:

print(f"An error occurred while copying the file: {e}")

def delete\_content(filename):

content = read\_file(filename)

if not content:

return

print(f"Enter the range to delete (1-{len(content)}):")

start = int(input("Start index: "))

end = int(input("End index: "))

if start < 1 or end > len(content) or start > end:

print("Invalid range.")

return

deleted\_content = content[start-1:end]

print("Content to be deleted:")

for i, line in enumerate(deleted\_content, start):

print(f"{i}: {line.strip()}")

confirm = input("Confirm deletion? (y/n): ").lower()

if confirm == 'y':

new\_content = content[:start-1] + content[end:]

try:

with open(filename, 'w') as file:

file.writelines(new\_content)

print("Content deleted successfully.")

except IOError as e:

print(f"An error occurred while updating the file: {e}")

else:

print("Deletion cancelled.")

def main():

while True:

print("\nSelect an option:")

print("1. Create a file and add content")

print("2. Add content to an existing file")

print("3. Read content from a file")

print("4. Copy content from one file to another")

print("5. Delete content from a file")

print("6. Exit")

try:

choice = int(input("Enter your choice (1-6): "))

if choice == 1 or choice == 2:

filename = get\_valid\_filename()

content = get\_file\_content()

create\_and\_write\_file(filename, content)

elif choice == 3:

filename = get\_valid\_filename()

read\_file(filename)

elif choice == 4:

source = get\_valid\_filename()

destination = get\_valid\_filename()

force = input("Use -rf flag to force overwrite? (y/n): ").lower() == 'y'

copy\_file(source, destination, force)

elif choice == 5:

filename = get\_valid\_filename()

delete\_content(filename)

elif choice == 6:

print("Exiting the program. Goodbye!")

break

else:

print("Invalid choice. Please enter a number between 1 and 6.")

except ValueError:

print("Invalid input. Please enter a number.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

```

This updated version includes the following new functionalities:

1. Copy contents of one txt file to another:

- The `copy\_file()` function handles this operation.

- It includes the `-rf` flag functionality: if used, it overwrites the destination file without asking.

- If the flag is not used, it asks the user for permission to overwrite.

2. Delete contents of a file:

- The `delete\_content()` function handles this operation.

- It first displays the file contents with line numbers.

- The user can specify a range of lines to delete.

- It shows the content to be deleted and asks for confirmation before proceeding.

- The deletion process minimizes file overwriting by reading the entire file once, modifying the content in memory, and writing back only once.

The main menu has been updated to include these new options. The code maintains the modular structure and error handling from the previous version while incorporating the new features.

This code is complete and runnable as-is, requiring no further modifications to function as described.

🧑

Human

There is a major bug in the deletion functionality , the index should represent a character however in the current implementation they are representing an entire line , fix this bug , by providing only the changes necessary to remedy it.

🤖

Assistant

I apologize for the oversight. You're right, the deletion functionality should work on character indices rather than line indices. Here are the necessary changes to fix this bug:

Replace the `delete\_content` function with the following updated version:

```python

def delete\_content(filename):

try:

with open(filename, 'r') as file:

content = file.read()

except IOError as e:

print(f"An error occurred while reading the file: {e}")

return

if not content:

print("The file is empty.")

return

print(f"File content:\n{content}")

print(f"Total characters: {len(content)}")

print(f"Enter the range to delete (0-{len(content)-1}):")

start = int(input("Start index: "))

end = int(input("End index: "))

if start < 0 or end >= len(content) or start > end:

print("Invalid range.")

return

deleted\_content = content[start:end+1]

print("Content to be deleted:")

print(deleted\_content)

confirm = input("Confirm deletion? (y/n): ").lower()

if confirm == 'y':

new\_content = content[:start] + content[end+1:]

try:

with open(filename, 'w') as file:

file.write(new\_content)

print("Content deleted successfully.")

except IOError as e:

print(f"An error occurred while updating the file: {e}")

else:

print("Deletion cancelled.")

```

This updated `delete\_content` function:

1. Reads the entire file content as a single string.

2. Displays the entire file content and its total character count.

3. Asks for character indices instead of line numbers.

4. Properly slices the string based on character indices.

5. Shows the exact characters to be deleted for confirmation.

6. Updates the file with the characters removed if confirmed.

These changes will fix the bug and allow deletion based on character indices rather than line indices. The rest of the code remains the same.

🧑

Human

Again write a commit message , highlighting the new functionality added , provide breif summary of the functionality in the commit message , and summary of the new additions in the commit message title

🤖

Assistant

Here's a commit message for the new functionality and bug fix:

```

Add file copy and character-based deletion features

This commit introduces two major functionalities to the CLI file handling application:

1. File Copy Feature:

- Ability to copy contents from one txt file to another

- Implements -rf flag for forced overwriting

- User confirmation for overwriting when -rf flag is not used

2. Character-Based Content Deletion:

- Allows users to delete specific portions of a file by character indices

- Displays file content with character count for easy reference

- Provides confirmation step before actual deletion

Bug Fix:

- Fixed deletion functionality to work on character level instead of line level

Summary of changes:

- Added copy\_file() function for file copying operations

- Updated delete\_content() function to handle character-based deletion

- Modified main menu to include new options

- Improved user interaction for precise content manipulation

- Enhanced error handling for file operations

These additions provide more granular control over file contents and expand

the application's utility for file management tasks.

```

This commit message:

1. Has a concise title that highlights the main additions

2. Provides a brief summary of each new functionality

3. Mentions the bug fix for the deletion feature

4. Gives an overview of the changes made to the codebase

5. Explains the benefits of these new features

This message effectively communicates the scope and impact of the new changes to the project.

# Task # 2:

🧑

Human

I am working on CLI application for my company , the cli is based on typescript utilizing the open source "oclif" library for CLI development , the present version of the CLI is able to transform specifications , validate specifications , and generate a docs-as-code portal , a novel solution of my company , which takes spec files , some markdown files and a build file to generate a portal , my task is to create a new function in the cli ,through which we can visualize the portal made , similar to how we can view html files using live server, the code structure of the repository is as follows:

🤖

Assistant

Thank you for providing the context of your CLI application. To create a new function in your CLI that allows you to visualize the generated portal, similar to how you can view HTML files using a live server, you can follow these steps:

1. Create a new command file:

First, create a new command file in the `src/commands` directory. Let's call it `serve.ts`.

2. Implement the command:

In `serve.ts`, implement the command to start a local server that serves the generated portal. Here's a basic implementation:

```typescript

import { Command } from '@oclif/core';

import \* as express from 'express';

import \* as path from 'path';

export default class Serve extends Command {

static description = 'Serve the generated portal locally';

static examples = [

'<%= config.bin %> serve',

];

static flags = {};

static args = [];

async run(): Promise<void> {

const app = express();

const portNumber = 3000; // You can make this configurable

// Assuming the generated portal is in a 'build' directory

const buildPath = path.join(process.cwd(), 'build');

app.use(express.static(buildPath));

app.listen(portNumber, () => {

this.log(`Server started at http://localhost:${portNumber}`);

});

}

}

```

3. Install dependencies:

You'll need to install the `express` package:

```

npm install express

npm install --save-dev @types/express

```

4. Update `package.json`:

Add the new command to your `package.json` file in the `oclif` section:

```json

"oclif": {

"commands": "./lib/commands",

"bin": "your-cli-name",

"plugins": [

"@oclif/plugin-help"

]

}

```

5. Build and run:

Build your CLI application and then you can run the new command:

```

npm run build

./bin/run serve

```

This will start a local server and serve your generated portal.

6. (Optional) Add open browser functionality:

If you want to automatically open the browser when the server starts, you can use the `open` package:

```

npm install open

```

Then modify your `serve.ts`:

```typescript

import { Command } from '@oclif/core';

import \* as express from 'express';

import \* as path from 'path';

import \* as open from 'open';

export default class Serve extends Command {

// ... (previous code)

async run(): Promise<void> {

// ... (previous code)

app.listen(portNumber, async () => {

this.log(`Server started at http://localhost:${portNumber}`);

await open(`http://localhost:${portNumber}`);

});

}

}

```

This implementation will serve your generated portal locally and optionally open it in the default browser. You can further customize this command by adding flags for port number, build directory path, etc.

🧑

Human

continuing with the source code structure , within src there is command folder , which contains particular files for each command, here the generate.ts file within portal folder: import \* as path from "path";

import \* as fs from "fs-extra";

import { Command, flags } from "@oclif/command";

import { Client, DocsPortalManagementController } from "@apimatic/sdk";

import { AxiosError } from "axios";

import { SDKClient } from "../../client-utils/sdk-client";

import { GeneratePortalParams } from "../../types/portal/generate";

import { downloadDocsPortal } from "../../controllers/portal/generate";

import { zipDirectory, replaceHTML, isJSONParsable } from "../../utils/utils";

import { AuthenticationError } from "../../types/utils";

export default class PortalGenerate extends Command {

static description =

"Generate and download a static API Documentation portal. Requires an input directory containing API specifications, a config file and optionally, markdown guides. For details, refer to the [documentation](https://portal-api-docs.apimatic.io/#/http/generating-api-portal/build-file)";

static flags = {

folder: flags.string({

parse: (input) => path.resolve(input),

default: "./",

description: "path to the input directory containing API specifications and config files"

}),

destination: flags.string({

parse: (input) => path.resolve(input),

default: path.resolve("./"),

description: "path to the downloaded portal"

}),

force: flags.boolean({ char: "f", default: false, description: "overwrite if a portal exists in the destination" }),

zip: flags.boolean({ default: false, description: "download the generated portal as a .zip archive" }),

"auth-key": flags.string({

default: "",

description: "override current authentication state with an authentication key"

})

};

static examples = [

`$ apimatic portal:generate --folder="./portal/" --destination="D:/"

Your portal has been generated at D:/

`

];

async run() {

const { flags } = this.parse(PortalGenerate);

const zip = flags.zip;

const sourceFolderPath: string = flags.folder;

const portalFolderPath: string = path.join(flags.destination, "generated\_portal");

const zippedPortalPath: string = path.join(flags.destination, "generated\_portal.zip");

const overrideAuthKey: string | null = flags["auth-key"] ? flags["auth-key"] : null;

// Check if at destination, portal already exists and throw error if force flag is not set for both zip and extracted

if (fs.existsSync(portalFolderPath) && !flags.force && !zip) {

throw new Error(`Can't download portal to path ${portalFolderPath}, because it already exists`);

} else if (fs.existsSync(zippedPortalPath) && !flags.force && zip) {

throw new Error(`Can't download portal to path ${zippedPortalPath}, because it already exists`);

}

try {

if (!(await fs.pathExists(flags.destination))) {

throw new Error(`Destination path ${flags.destination} does not exist`);

} else if (!(await fs.pathExists(flags.folder))) {

throw new Error(`Portal build folder ${flags.folder} does not exist`);

}

const client: Client = await SDKClient.getInstance().getClient(overrideAuthKey, this.config.configDir);

const docsPortalController: DocsPortalManagementController = new DocsPortalManagementController(client);

const zippedBuildFilePath = await zipDirectory(sourceFolderPath, flags.destination);

const generatePortalParams: GeneratePortalParams = {

zippedBuildFilePath,

portalFolderPath,

zippedPortalPath,

docsPortalController,

overrideAuthKey,

zip

};

const generatedPortalPath: string = await downloadDocsPortal(generatePortalParams, this.config.configDir);

this.log(`Your portal has been generated at ${generatedPortalPath}`);

} catch (error) {

if (error && (error as AxiosError).response) {

const apiError = error as AxiosError;

const apiResponse = apiError.response;

if (apiResponse) {

const responseData = apiResponse.data.toString();

if (apiResponse.status === 422 && responseData.length > 0 && isJSONParsable(responseData)) {

const nestedErrors = JSON.parse(responseData);

if (nestedErrors.error) {

return this.error(replaceHTML(nestedErrors.error));

} else if (nestedErrors.message) {

return this.error(replaceHTML(nestedErrors.message));

}

} else if (apiResponse.status === 401 && responseData.length > 0) {

this.error("You are not authorized to perform this action");

} else if (apiResponse.status === 403 && apiResponse.statusText) {

return this.error("Your subscription does not allow on premise portal generation");

} else {

return this.error(apiError.message);

}

}

} else if ((error as AuthenticationError).statusCode === 401) {

this.error("You are not authorized to perform this action");

} else if (

(error as AuthenticationError).statusCode === 402 &&

(error as AuthenticationError).body &&

typeof (error as AuthenticationError).body === "string"

) {

this.error((error as AuthenticationError).body);

} else {

this.error(`${(error as Error).message}`);

}

}

}

}

and here is the the generate.ts file within the controller: import cli from "cli-ux";

import \* as fs from "fs-extra";

import \* as FormData from "form-data";

import { baseURL } from "../../config/env";

import { deleteFile, unzipFile } from "../../utils/utils";

import { GeneratePortalParams } from "../../types/portal/generate";

import { AuthInfo, getAuthInfo } from "../../client-utils/auth-manager";

import axios, { AxiosRequestConfig, AxiosResponse } from "axios";

// TODO: Remove after SDK is patched

const downloadPortalAxios = async (zippedBuildFilePath: string, overrideAuthKey: string | null, configDir: string) => {

const formData = new FormData();

const authInfo: AuthInfo | null = await getAuthInfo(configDir);

formData.append("file", fs.createReadStream(zippedBuildFilePath));

const config: AxiosRequestConfig = {

headers: {

Authorization: authInfo || overrideAuthKey ? `X-Auth-Key ${authInfo?.authKey.trim() || overrideAuthKey}` : "",

"User-Agent": "APIMatic CLI",

...formData.getHeaders()

},

responseType: "arraybuffer"

};

const { data }: AxiosResponse = await axios.post(`${baseURL}/portal`, formData, config);

return data;

};

// Download Docs Portal

export const downloadDocsPortal = async (

{ zippedBuildFilePath, portalFolderPath, zippedPortalPath, overrideAuthKey, zip }: GeneratePortalParams,

configDir: string

) => {

cli.action.start("Downloading portal");

// Check if the build file exists for the user or not

if (!(await fs.pathExists(zippedBuildFilePath))) {

throw new Error("Build file doesn't exist");

}

// TODO: \*\*\*CRITICAL\*\*\* Remove this call once the SDK is patched

const data: ArrayBuffer = await downloadPortalAxios(zippedBuildFilePath, overrideAuthKey, configDir);

await deleteFile(zippedBuildFilePath);

await fs.writeFile(zippedPortalPath, data);

// TODO: Uncomment this code block when the SDK is patched

// const file: FileWrapper = new FileWrapper(fs.createReadStream(zippedBuildFilePath));

// const { result }: ApiResponse<NodeJS.ReadableStream | Blob> =

// await docsPortalController.generateOnPremPortalViaBuildInput(file);

// if ((data as NodeJS.ReadableStream).readable) {

// await writeFileUsingReadableStream(data as NodeJS.ReadableStream, zippedPortalPath);

if (!zip) {

await unzipFile(fs.createReadStream(zippedPortalPath), portalFolderPath);

await deleteFile(zippedPortalPath);

}

cli.action.stop();

return zip ? zippedPortalPath : portalFolderPath;

// } else {

// throw new Error("Couldn't download the portal");

// }

};

utilizing this code , i want to create a new command within the the cli to visualize the docs as code portal using an http server , ability to perform hot reload and so forth , also currently there is a restriction in the code that prevents sending huge file 30mb or so to the the server , I also need to get rid of this restriction , this can easily be done by modifying axios config file. In the response I want you to provide complete code piece-by-piece for each new file to be added and in which directory it should be added , along with the commands to build the library and test it again ,

🤖

Assistant

Certainly! I'll provide you with the necessary code and instructions to create a new command for visualizing the docs-as-code portal using an HTTP server with hot reload functionality, and also remove the file size restriction. Let's go through this step by step:

1. Create a new command file:

Create a new file named `serve.ts` in the `src/commands/portal` directory:

```typescript

// src/commands/portal/serve.ts

import { Command, flags } from "@oclif/command";

import \* as path from "path";

import \* as fs from "fs-extra";

import \* as express from "express";

import \* as livereload from "livereload";

import \* as connectLivereload from "connect-livereload";

import \* as open from "open";

export default class PortalServe extends Command {

static description = "Serve and visualize the generated docs-as-code portal with hot reload";

static flags = {

port: flags.integer({

char: "p",

description: "Port to serve the portal",

default: 3000,

}),

directory: flags.string({

char: "d",

description: "Directory containing the generated portal",

default: "./generated\_portal",

parse: (input) => path.resolve(input),

}),

open: flags.boolean({

char: "o",

description: "Open the portal in the default browser",

default: false,

}),

};

async run() {

const { flags } = this.parse(PortalServe);

const portalDir = flags.directory;

const port = flags.port;

if (!(await fs.pathExists(portalDir))) {

this.error(`The specified directory does not exist: ${portalDir}`);

}

const app = express();

// Set up live reload

const liveReloadServer = livereload.createServer();

liveReloadServer.watch(portalDir);

app.use(connectLivereload());

// Serve static files

app.use(express.static(portalDir));

// Start the server

app.listen(port, () => {

this.log(`Server started at http://localhost:${port}`);

if (flags.open) {

open(`http://localhost:${port}`);

}

});

// Keep the process running

return new Promise(() => {});

}

}

```

2. Update the `package.json` file:

Add the following dependencies:

```json

{

"dependencies": {

"express": "^4.17.1",

"livereload": "^0.9.3",

"connect-livereload": "^0.6.1",

"open": "^8.4.0"

},

"devDependencies": {

"@types/express": "^4.17.13",

"@types/livereload": "^0.9.2",

"@types/connect-livereload": "^0.5.1"

}

}

```

3. Install the new dependencies:

Run the following command:

```

npm install

```

4. Modify the Axios configuration to remove file size restrictions:

Create a new file named `axios-config.ts` in the `src/utils` directory:

```typescript

// src/utils/axios-config.ts

import axios from "axios";

const axiosInstance = axios.create({

maxContentLength: Infinity,

maxBodyLength: Infinity,

});

export default axiosInstance;

```

5. Update the `generate.ts` file in the `src/controllers/portal` directory:

Replace the existing axios import with the custom axiosInstance:

```typescript

// src/controllers/portal/generate.ts

import axiosInstance from "../../utils/axios-config";

// ... (rest of the file remains the same)

// Replace the existing axios.post call with:

const { data }: AxiosResponse = await axiosInstance.post(`${baseURL}/portal`, formData, config);

```

6. Build the CLI:

Run the following command to build the CLI:

```

npm run build

```

7. Test the new command:

After building, you can test the new command using:

```

./bin/run portal:serve

```

This will start the server and serve your generated portal. You can use the following options:

- `--port` or `-p`: Specify a custom port (default: 3000)

- `--directory` or `-d`: Specify the directory containing the generated portal (default: ./generated\_portal)

- `--open` or `-o`: Open the portal in the default browser

For example:

```

./bin/run portal:serve --port 8080 --directory ./my\_portal --open

```

This implementation will serve your generated portal locally with hot reload functionality. When you make changes to the files in the portal directory, the browser will automatically refresh to show the updates.

The file size restriction has been removed by creating a custom Axios instance with `maxContentLength` and `maxBodyLength` set to `Infinity`.

Remember to update your CLI's documentation to include information about this new command and its usage.

🧑

Human

The functionality n the serve.ts file is incomplete , for the portal to generate we also need the directory which contains the specs, content and build file , by zipping this folder and sending it to the API we get the generated portal as zip which we would use to further , so in this code : import { Command, flags } from "@oclif/command";

import \* as path from "path";

import \* as fs from "fs-extra";

import \* as express from "express";

import \* as livereload from "livereload";

import \* as connectLivereload from "connect-livereload";

import \* as open from "open";

export default class PortalServe extends Command {

static description = "Serve and visualize the generated docs-as-code portal with hot reload";

static flags = {

port: flags.integer({

char: "p",

description: "Port to serve the portal",

default: 3000,

}),

directory: flags.string({

char: "d",

description: "Directory containing the generated portal",

default: "./generated\_portal",

parse: (input) => path.resolve(input),

}),

open: flags.boolean({

char: "o",

description: "Open the portal in the default browser",

default: false,

}),

};

async run() {

const { flags } = this.parse(PortalServe);

const portalDir = flags.directory;

const port = flags.port;

if (!(await fs.pathExists(portalDir))) {

this.error(`The specified directory does not exist: ${portalDir}`);

}

const app = express();

// Set up live reload

const liveReloadServer = livereload.createServer();

liveReloadServer.watch(portalDir);

app.use(connectLivereload());

// Serve static files

app.use(express.static(portalDir));

// Start the server

app.listen(port, () => {

this.log(`Server started at http://localhost:${port}`);

if (flags.open) {

open(`http://localhost:${port}`);

}

});

// Keep the process running

return new Promise(() => {});

}

}

, Make necessary amends to fully complete the functionality of this command , we can utilize another flag e-g source which tells us about where all the necessary files necessary to build the portal are present.

🤖

Assistant

You're absolutely right. Let's modify the `serve.ts` file to include the functionality for generating the portal before serving it. We'll add a `source` flag and incorporate the portal generation process. Here's the updated `serve.ts` file:

```typescript

// src/commands/portal/serve.ts

import { Command, flags } from "@oclif/command";

import \* as path from "path";

import \* as fs from "fs-extra";

import \* as express from "express";

import \* as livereload from "livereload";

import \* as connectLivereload from "connect-livereload";

import \* as open from "open";

import { Client, DocsPortalManagementController } from "@apimatic/sdk";

import { SDKClient } from "../../client-utils/sdk-client";

import { GeneratePortalParams } from "../../types/portal/generate";

import { downloadDocsPortal } from "../../controllers/portal/generate";

import { zipDirectory } from "../../utils/utils";

export default class PortalServe extends Command {

static description = "Generate, serve and visualize the docs-as-code portal with hot reload";

static flags = {

port: flags.integer({

char: "p",

description: "Port to serve the portal",

default: 3000,

}),

directory: flags.string({

char: "d",

description: "Directory to store and serve the generated portal",

default: "./generated\_portal",

parse: (input) => path.resolve(input),

}),

source: flags.string({

char: "s",

description: "Source directory containing specs, content, and build file",

required: true,

parse: (input) => path.resolve(input),

}),

open: flags.boolean({

char: "o",

description: "Open the portal in the default browser",

default: false,

}),

"auth-key": flags.string({

description: "Override current authentication state with an authentication key",

}),

};

async run() {

const { flags } = this.parse(PortalServe);

const portalDir = flags.directory;

const sourceDir = flags.source;

const port = flags.port;

const overrideAuthKey = flags["auth-key"] || null;

if (!(await fs.pathExists(sourceDir))) {

this.error(`The specified source directory does not exist: ${sourceDir}`);

}

// Generate the portal

try {

const client: Client = await SDKClient.getInstance().getClient(overrideAuthKey, this.config.configDir);

const docsPortalController: DocsPortalManagementController = new DocsPortalManagementController(client);

const zippedBuildFilePath = await zipDirectory(sourceDir, path.dirname(portalDir));

const generatePortalParams: GeneratePortalParams = {

zippedBuildFilePath,

portalFolderPath: portalDir,

zippedPortalPath: path.join(path.dirname(portalDir), "generated\_portal.zip"),

docsPortalController,

overrideAuthKey,

zip: false

};

await downloadDocsPortal(generatePortalParams, this.config.configDir);

this.log(`Portal generated successfully at ${portalDir}`);

} catch (error) {

this.error(`Failed to generate portal: ${(error as Error).message}`);

}

// Set up and start the server

const app = express();

// Set up live reload

const liveReloadServer = livereload.createServer();

liveReloadServer.watch(portalDir);

app.use(connectLivereload());

// Serve static files

app.use(express.static(portalDir));

// Start the server

app.listen(port, () => {

this.log(`Server started at http://localhost:${port}`);

if (flags.open) {

open(`http://localhost:${port}`);

}

});

// Watch for changes in the source directory

fs.watch(sourceDir, { recursive: true }, async (eventType, filename) => {

if (eventType === 'change') {

this.log(`Change detected in ${filename}. Regenerating portal...`);

try {

await this.generatePortal(sourceDir, portalDir, overrideAuthKey);

this.log('Portal regenerated successfully');

} catch (error) {

this.error(`Failed to regenerate portal: ${(error as Error).message}`);

}

}

});

// Keep the process running

return new Promise(() => {});

}

private async generatePortal(sourceDir: string, portalDir: string, overrideAuthKey: string | null) {

const client: Client = await SDKClient.getInstance().getClient(overrideAuthKey, this.config.configDir);

const docsPortalController: DocsPortalManagementController = new DocsPortalManagementController(client);

const zippedBuildFilePath = await zipDirectory(sourceDir, path.dirname(portalDir));

const generatePortalParams: GeneratePortalParams = {

zippedBuildFilePath,

portalFolderPath: portalDir,

zippedPortalPath: path.join(path.dirname(portalDir), "generated\_portal.zip"),

docsPortalController,

overrideAuthKey,

zip: false

};

await downloadDocsPortal(generatePortalParams, this.config.configDir);

}

}

```

This updated version of `serve.ts` does the following:

1. Adds a new required `source` flag to specify the directory containing the specs, content, and build file.

2. Generates the portal before starting the server using the `downloadDocsPortal` function.

3. Watches for changes in the source directory and regenerates the portal when changes are detected.

4. Serves the generated portal and provides live reload functionality.

To use this command, you would run:

```

./bin/run portal:serve --source ./path/to/source/files --directory ./path/to/generated/portal

```

This will generate the portal, serve it, and watch for changes in the source directory. When changes are detected, it will regenerate the portal, and the browser will automatically refresh to show the updates.

Remember to update your CLI's documentation to include information about this new command and its usage, especially the new `source` flag.

🧑

Human

Upon running the command , i get this log : "./bin/run portal:serve --source C:\Users\PMLS\Documents\GitHub\VerizenSpec --directory Desktop

Downloading portal... !

» Error: Failed to generate or serve the portal: Request failed with status code 500" , to remedy this we need to implement some further checks and better error logs , firstly while zipping the contents of the source folder , the program should check that there must be a content folder , a spec folder and a static folder and APIMATIC-BUILD.json file , these directories and file must be prsent and the if there are any other files or directories they shoulde be disgraded when they are zipped , also improve error handling showing also the response body in the error logs.

🤖

Assistant

You're absolutely right. We need to implement better checks and error handling. Let's update the `serve.ts` file and create a new utility function to handle the directory checks and zipping process. Here's the improved version:

First, let's create a new utility function in `src/utils/portal-utils.ts`:

```typescript

// src/utils/portal-utils.ts

import \* as fs from 'fs-extra';

import \* as path from 'path';

import \* as archiver from 'archiver';

export async function validateAndZipPortalSource(sourceDir: string, outputPath: string): Promise<string> {

const requiredItems = ['content', 'spec', 'static', 'APIMATIC-BUILD.json'];

for (const item of requiredItems) {

const itemPath = path.join(sourceDir, item);

if (!await fs.pathExists(itemPath)) {

throw new Error(`Required item '${item}' not found in the source directory.`);

}

}

const output = fs.createWriteStream(outputPath);

const archive = archiver('zip', {

zlib: { level: 9 }

});

return new Promise((resolve, reject) => {

output.on('close', () => resolve(outputPath));

archive.on('error', err => reject(err));

archive.pipe(output);

for (const item of requiredItems) {

const itemPath = path.join(sourceDir, item);

if (fs.lstatSync(itemPath).isDirectory()) {

archive.directory(itemPath, item);

} else {

archive.file(itemPath, { name: item });

}

}

archive.finalize();

});

}

```

Now, let's update the `serve.ts` file:

```typescript

// src/commands/portal/serve.ts

import { Command, flags } from "@oclif/command";

import \* as path from "path";

import \* as fs from "fs-extra";

import \* as express from "express";

import \* as livereload from "livereload";

import \* as connectLivereload from "connect-livereload";

import \* as open from "open";

import { Client, DocsPortalManagementController } from "@apimatic/sdk";

import { SDKClient } from "../../client-utils/sdk-client";

import { GeneratePortalParams } from "../../types/portal/generate";

import { downloadDocsPortal } from "../../controllers/portal/generate";

import { validateAndZipPortalSource } from "../../utils/portal-utils";

import axios from 'axios';

export default class PortalServe extends Command {

static description = "Generate, serve and visualize the docs-as-code portal with hot reload";

static flags = {

port: flags.integer({

char: "p",

description: "Port to serve the portal",

default: 3000,

}),

directory: flags.string({

char: "d",

description: "Directory to store and serve the generated portal",

default: "./generated\_portal",

parse: (input) => path.resolve(input),

}),

source: flags.string({

char: "s",

description: "Source directory containing specs, content, and build file",

required: true,

parse: (input) => path.resolve(input),

}),

open: flags.boolean({

char: "o",

description: "Open the portal in the default browser",

default: false,

}),

"auth-key": flags.string({

description: "Override current authentication state with an authentication key",

}),

};

async run() {

const { flags } = this.parse(PortalServe);

const portalDir = flags.directory;

const sourceDir = flags.source;

const port = flags.port;

const overrideAuthKey = flags["auth-key"] || null;

if (!(await fs.pathExists(sourceDir))) {

this.error(`The specified source directory does not exist: ${sourceDir}`);

}

// Generate the portal

try {

await this.generatePortal(sourceDir, portalDir, overrideAuthKey);

this.log(`Portal generated successfully at ${portalDir}`);

} catch (error) {

this.handleError(error);

}

// Set up and start the server

const app = express();

// Set up live reload

const liveReloadServer = livereload.createServer();

liveReloadServer.watch(portalDir);

app.use(connectLivereload());

// Serve static files

app.use(express.static(portalDir));

// Start the server

app.listen(port, () => {

this.log(`Server started at http://localhost:${port}`);

if (flags.open) {

open(`http://localhost:${port}`);

}

});

// Watch for changes in the source directory

fs.watch(sourceDir, { recursive: true }, async (eventType, filename) => {

if (eventType === 'change') {

this.log(`Change detected in ${filename}. Regenerating portal...`);

try {

await this.generatePortal(sourceDir, portalDir, overrideAuthKey);

this.log('Portal regenerated successfully');

} catch (error) {

this.handleError(error);

}

}

});

// Keep the process running

return new Promise(() => {});

}

private async generatePortal(sourceDir: string, portalDir: string, overrideAuthKey: string | null) {

const client: Client = await SDKClient.getInstance().getClient(overrideAuthKey, this.config.configDir);

const docsPortalController: DocsPortalManagementController = new DocsPortalManagementController(client);

const zippedBuildFilePath = await validateAndZipPortalSource(sourceDir, path.join(path.dirname(portalDir), "portal\_source.zip"));

const generatePortalParams: GeneratePortalParams = {

zippedBuildFilePath,

portalFolderPath: portalDir,

zippedPortalPath: path.join(path.dirname(portalDir), "generated\_portal.zip"),

docsPortalController,

overrideAuthKey,

zip: false

};

await downloadDocsPortal(generatePortalParams, this.config.configDir);

}

private handleError(error: any) {

if (axios.isAxiosError(error)) {

const axiosError = error;

if (axiosError.response) {

this.error(`Failed to generate or serve the portal: ${axiosError.message}\nResponse: ${JSON.stringify(axiosError.response.data)}`);

} else {

this.error(`Failed to generate or serve the portal: ${axiosError.message}`);

}

} else {

this.error(`Failed to generate or serve the portal: ${(error as Error).message}`);

}

}

}

```

These changes accomplish the following:

1. We've created a new `validateAndZipPortalSource` function that checks for the required directories and files before zipping. It only includes the required items in the zip file.

2. We've improved error handling by creating a `handleError` method that provides more detailed error information, including the response body for Axios errors.

3. We're now using the `validateAndZipPortalSource` function instead of the previous `zipDirectory` function.

4. We've added more descriptive error messages throughout the process.

To use this updated command, you'll need to install the `archiver` package:

```

npm install archiver

npm install --save-dev @types/archiver

```

This implementation should provide better error handling and ensure that only the required files and directories are included in the zipped portal source. It will also provide more detailed error messages, including the response body when available, which should help in diagnosing issues like the 500 status code you encountered.

🧑

Human

Now create a comprehensive git commit message , containing the feat , which was achieved in this commit , here we have completed an entire new command serve ,highlighting what the serve command does and how it can be utilizied also mention the minor quality of life feature implemented , removing the file size restriction when generating the portal , use this as a template and add information related to the functionality , also provide a neat and concise title like this: feat(compliation pipeline): changes to upload spec file & enqueue messages, Why

This PR contains changes to upload spec file to Blob storage and add message to azure queue for compilation pipeline

Type of change

Select multiple if applicable.

New feature (non-breaking change which adds functionality)

Impact Area

Core.API

🤖

Assistant

Here's a comprehensive git commit message for the new `serve` command and the file size restriction removal:

```

feat(portal): add 'serve' command for live portal visualization & remove file size limit

This commit introduces a new 'serve' command to the CLI, enhancing the portal development experience, and removes the file size restriction when generating portals.

The 'serve' command allows developers to:

- Generate a docs-as-code portal from source files

- Serve the generated portal locally

- Automatically regenerate the portal on source file changes

- Provide live reload functionality for immediate visual feedback

Usage:

./bin/run portal:serve --source <path-to-source> --directory <output-directory>

Additional features:

- Validates required source directory structure (content, spec, static folders, and APIMATIC-BUILD.json)

- Improved error handling with detailed error messages

- Option to open the served portal in the default browser

Quality of life improvement:

- Removed file size restriction when generating portals, allowing for larger spec files

Why:

This feature streamlines the portal development process, providing developers with a more efficient workflow for creating and visualizing API documentation portals. The removal of the file size restriction accommodates larger projects and more complex API specifications.

Type of change:

- New feature (non-breaking change which adds functionality)

- Performance improvement (file size limit removal)

Impact Area:

CLI, Portal Generation

```

This commit message provides a clear and comprehensive overview of the new `serve` command, its functionality, and the additional improvements made. It follows the structure you provided, including a concise title, detailed description of the changes, rationale, type of change, and impact area.