**📚 Resources**

👉 Learn more from Cohere with the [official Docs](https://docs.cohere.com/)

👉 Master the Building Collaboration Format component with our [Documentation](https://docs.thatopen.com/api/@thatopen/components/classes/BCFTopics).

👉 Prompt like a pro by learning [prompt tuning](https://www.datacamp.com/tutorial/understanding-prompt-tuning).

**🗝 Key Concepts**

**⚙️ CLI Command Creation**

Defining a custom command within a project allows for external execution, enabling flexible and accessible scripting capabilities. Utilizing the bin property in a package.json file facilitates the creation of such commands.

**📁 File System Traversal**

Implementing functions to retrieve files from a given directory is essential for processing external data. This involves parsing command-line arguments and using filesystem access to read file paths.

**🤖 AI Prompt Integration**

The ability to prompt an AI with context from loaded files enables powerful analysis and processing capabilities. This involves collecting user input, formatting prompts, and handling AI responses.

**🤓 Exercise Development**

The following is the step-by-step guide on how to accomplish the exercise. Use this as complementary information from the video. 📽

**🏗️ Scaffold the Project**

To initialize the project, use the command:

npm init -y

**📦 Adjust the Package.json file**

// Add the bin command to enable script execution

// And the type.

/\*{

"name": "bcf-ai-chart",\*/

"type": "module",

"bin": {

"bcfai-chat": "./index.js"

},

/\*"dependencies": {

"@inquirer/prompts": "7.2.1",

"@thatopen/components": "2.4.3",

"cohere-ai": "^7.15.2",

"dotenv": "^16.4.7"

}

}\*/

Create the index.js file

Add to the top of the file:

#!/usr/bin/env node

Execute this command:

npm link

We do this so then in the terminal we can type our bin name from the package.json file and execute the code.

**📂 Set File Reading Logic**

import \* as fs from "fs"

import \* as path from "path"

function readFile(joinedPath) {

return fs.readFileSync(joinedPath)

}

function getFiles() {

// Get the last argument from the vector (the path to the files)

const input = process.argv[2]

// Indicate usage

if (!input) {

throw new Error('Usage: bcfai-chat <path/to/bcf/files>')

}

let files

try {

// Read directory and filter needed files

files = fs.readdirSync(

input

).filter(file => path.extname(file).toLowerCase() === '.bcf')

} catch {

throw new Error('Insert a valid directory.')

}

if (files.length === 0) {

throw new Error(`No BCF files found in ${input}`)

}

// Return the full path of the file

return files.map(file => path.join(input, file))

}

**🔃 Load the BCF Components**

npm I @thatopen/components -E

import \* as OBC from "@thatopen/components"

// Instantiate the engine and get the topics component

function getComponents() {

const components = new OBC.Components()

return components.get(OBC.BCFTopics)

}

// De data comes from the readFile function

// Gets turned into a int array and loaded to the engine.

async function loadTopics(data) {

const bcf = getComponents()

const buffer = new Uint8Array(data)

const bcfData = await bcf.load(buffer)

return JSON.stringify(bcfData.topics)

}

**🤖 Initialize Cohere and env.**

// Install doten and cohere

npm i dotenv

npm i cohere-ai -E

npm I @inquirer/prompts -E

Then head to [cohere.com](http://cohere.com/) and register if you haven’t. If you are already signed in, head to the dashboard.

Then API Keys on the left panel and you’ll find your key in the Trial keys section:

Copy the value and create a ‘.env’ file in the project’s folder and do something like this:

APIKEY = <your api key>

Import the packages and cohere.

import \* as env from "dotenv"

import { CohereClient } from 'cohere-ai';

function getKey() {

env.config()

return process.env.APIKEY

}

function initAI(apiKey) {

return new CohereClient({

token: apiKey

})

}

**💬 Create Request and Chat functions:**

import \* as prompt from '@inquirer/prompts';

async function makeRequest(stringData) {

const key = getKey()

const cohere = initAI(key)

// This is the structure needed for input prompt

const userPrompt = {

name: 'prompt', // A name

type: 'string', // The type of the input

message: 'Write your prompt:' // Message to be displayed

}

while (true) {

// There are other options to use with prompt but this one works just fine.

const input = await prompt.input(userPrompt)

if (!input) {

console.log('No input given, try writing something.')

continue

}

// This prompt can be tuned, but it works good to

// get the conversation with cohere going

// stringData is the information from the topics.

const coherePrompt = {

message: `Based on the following data ${stringData}

You should only create the response based on the information given.

The data contains this structure, that does not mean that you are going to give me this

{

title: string

description: string

type: string

priority: string

labels: Array

stage: string

assignee: string

}

Information that is not found on ${stringData} should not be presented on the

result

your job is to answer the following question: ${input}.

If the question is empty, say that you can't process empty questions and to try again.`

}

const response = await cohere.chat(coherePrompt)

console.log(response.text)

}

}

async function chat() {

const filesPath = getFiles()

let stringData

for (const file of filesPath) {

const fileData = readFile(file)

stringData = await loadTopics(fileData)

}

await makeRequest(stringData)

}

await chat()

**💡 Bonus Tip - Export Chat**

//async function makeRequest(stringData) {

// Include an array to store the messages.

const messages = []

/\*const key = getKey()

const cohere = initAI(key)\*/

...

const question = await prompt.input(userPrompt)

/\*const coherePrompt = {

message: `You are now a BIM expert manager and based on the following data ${stringData}

your job is to answer the following inquire: ${question}

You should only create the response based on the information given.

Information that is not found on ${stringData} should not be presented on the result.`

}

const response = await cohere.chat(coherePrompt)

const responseText = response.text

console.log(responseText)\*/

// Push an object in this format to the messages array

messages.push(

{

user: question,

cohere: responseText

}

)

const fileName = 'CohereChat.json'

const stringifiedMessages = JSON.stringify(messages, null, 2)

fs.writeFileSync(fileName, stringifiedMessages)

console.log(`Chat exported to ${process.cwd()}\\\\${fileName}`)

}

Execute the command in terminal:

bcfai-chat <folder-path>

**⚔ Quest (code: d5t7W)**

The quest is your chance to grow. It will let you increase your level to earn different badges and get access to benefits that only people with certain levels can achieve, like getting Bounties 💰 (cash rewards) for contributing to That Open Company's open-source libraries, or special gifts. To complete this lesson’s quest, do the following: 👇

Enable the chat to create a JSON-formatted version of the last response given by the AI.

**🦶 Development Steps**

If you have implemented the bonus tip from the guide, you should already have the messages array and the push of the conversation object. If not, create it in the makeRequest function.

1. Create an object called questions.
   1. type is list
   2. name is whatever you feel it should be, I used actionToTake
   3. message can be whatever you want as well, i used Select one:
   4. Choices will be ['Ask Something', 'Export last message to JSON', 'Export Chat', 'Exit']
2. Inside the while, create a [prompt.select](http://prompt.select/) and pass the questions as argument.
3. Implement a switch:

switch (variableToVerify) {

case caseName1:

logicHere

break

case caseName2:

logicHere

break

...

1. First case must match the Ask Something. Move the userPrompt object, prompt input, cohere prompt, response and the log. Here the messages.push should be set.
2. Next case is for Export last message to JSON.
3. Verify that there are message or log that there is nothing to export.
4. Create a prompt that includes the last message from the messages array and that indicates the AI to give back a JSON structure based on that.
5. Give the prompt to cohere, get the text and log it. Remember the break clause.