**6.CODING**

**1. ALEXA.JSON**

1. {
2. "interactionModel": {
3. "languageModel": {
4. "invocationName": "the automation",
5. "intents": [
6. {
7. "name": "AMAZON.FallbackIntent",
8. "samples": []
9. },
10. {
11. "name": "AMAZON.CancelIntent",
12. "samples": []
13. },
14. {
15. "name": "AMAZON.HelpIntent",
16. "samples": []
17. },
18. {
19. "name": "AMAZON.StopIntent",
20. "samples": []
21. },
22. {
23. "name": "AMAZON.NavigateHomeIntent",
24. "samples": []
25. },
26. {
27. "name": "WhatsMyCommandIntent",
28. "slots": [],
29. "samples": [
30. "good bye",
31. "last commands"
32. ]
33. },
34. {
35. "name": "ThingsYouCanDo",
36. "slots": [],
37. "samples": [
38. "what can u do for me",
39. "List the commands",
40. "Tell me what u can do",
41. "What you can do"
42. ]
43. },
44. {
45. "name": "ExecutingCommands",
46. "slots": [
47. {
48. "name": "SingleCommands",
49. "type": "singlelinecommands"
50. }
51. ],
52. "samples": [
53. "can u show my {SingleCommands} ",
54. "open {SingleCommands}",
55. "can u {SingleCommands}",
56. "arrange the {SingleCommands}",
57. "run the {SingleCommands}",
58. "clean the {SingleCommands}",
59. "start {SingleCommands}",
60. "can u start {SingleCommands}",
61. "{SingleCommands}"
62. ]
63. },
64. {
65. "name": "OpeingNormalPrograms",
66. "slots": [
67. {
68. "name": "NormalProgram",
69. "type": "ProgramNames"
70. }
71. ],
72. "samples": [
73. "run {NormalProgram}",
74. "start {NormalProgram}",
75. "open {NormalProgram}"
76. ]
77. },
78. {
79. "name": "DialogCommands",
80. "slots": [
81. {
82. "name": "commands",
83. "type": "commands",
84. "samples": [
85. "{projectname}",
86. "name is {projectname}"
87. ]
88. },
89. {
90. "name": "projectname",
91. "type": "AMAZON.SearchQuery",
92. "samples": [
93. "name of the project is {projectname}",
94. "name is {projectname}",
95. "project name is {projectname}",
96. "{projectname}"
97. ]
98. }
99. ],
100. "samples": [
101. "{commands}",
102. "create {commands}",
103. "can create {commands} "
104. ]
105. }
106. ],
107. "types": [
108. {
109. "name": "singlelinecommands",
110. "values": [
111. {
112. "name": {
113. "value": "screenmirror",
114. "synonyms": [
115. "screen mirroring",
116. "screenmirroring",
117. "screen",
118. "screen share"
119. ]
120. }
121. },
122. {
123. "name": {
124. "value": "desktopCleaner",
125. "synonyms": [
126. "desktop",
127. "cleaner",
128. "desktop cleaner"
129. ]
130. }
131. }
132. ]
133. },
134. {
135. "name": "ProgramNames",
136. "values": [
137. {
138. "name": {
139. "value": "start chrome",
140. "synonyms": [
141. "chrome",
142. "browser"
143. ]
144. }
145. },
146. {
147. "name": {
148. "value": "notepad"
149. }
150. },
151. {
152. "name": {
153. "value": "code",
154. "synonyms": [
155. "vscode"
156. ]
157. }
158. },
159. {
160. "name": {
161. "value": "mspaint",
162. "synonyms": [
163. "paint"
164. ]
165. }
166. }
167. ]
168. },
169. {
170. "name": "commands",
171. "values": [
172. {
173. "name": {
174. "value": "flutterproject",
175. "synonyms": [
176. "flutter project",
177. "project",
178. "flutter"
179. ]
180. }
181. }
182. ]
183. }
184. ]
185. },
186. "dialog": {
187. "intents": [
188. {
189. "name": "ExecutingCommands",
190. "delegationStrategy": "ALWAYS",
191. "confirmationRequired": false,
192. "prompts": {},
193. "slots": [
194. {
195. "name": "SingleCommands",
196. "type": "singlelinecommands",
197. "confirmationRequired": false,
198. "elicitationRequired": false,
199. "prompts": {}
200. }
201. ]
202. },
203. {
204. "name": "DialogCommands",
205. "confirmationRequired": false,
206. "prompts": {},
207. "slots": [
208. {
209. "name": "commands",
210. "type": "commands",
211. "confirmationRequired": false,
212. "elicitationRequired": true,
213. "prompts": {
214. "elicitation": "Elicit.Slot.763323224860.1442318176051"
215. }
216. },
217. {
218. "name": "projectname",
219. "type": "AMAZON.SearchQuery",
220. "confirmationRequired": false,
221. "elicitationRequired": true,
222. "prompts": {
223. "elicitation": "Elicit.Slot.763323224860.769562388878"
224. }
225. }
226. ]
227. }
228. ],
229. "delegationStrategy": "ALWAYS"
230. },
231. "prompts": [
232. {
233. "id": "Confirm.Intent.898526131785",
234. "variations": [
235. {
236. "type": "PlainText",
237. "value": "can tell project name"
238. }
239. ]
240. },
241. {
242. "id": "Elicit.Slot.763323224860.769562388878",
243. "variations": [
244. {
245. "type": "PlainText",
246. "value": "can tell the name of project"
247. },
248. {
249. "type": "PlainText",
250. "value": "project name ?"
251. },
252. {
253. "type": "PlainText",
254. "value": "can you say project name"
255. },
256. {
257. "type": "PlainText",
258. "value": "what is project name"
259. }
260. ]
261. },
262. {
263. "id": "Elicit.Slot.763323224860.1442318176051",
264. "variations": [
265. {
266. "type": "PlainText",
267. "value": "project name please?"
268. },
269. {
270. "type": "PlainText",
271. "value": "can you tell project name"
272. }
273. ]
274. }
275. ]
276. }
277. }

**AWS LAMBDA(lambda\_function.py)**

1. from \_\_future\_\_ import print\_function
2. import requests

5. def lambda\_handler(event, context):
6. """ Route the incoming request based on type (LaunchRequest, IntentRequest,
7. etc.) The JSON body of the request is provided in the event parameter.
8. """
9. """
10. Uncomment this if statement and populate with your skill's application ID to
11. prevent someone else from configuring a skill that sends requests to this
12. function.
13. """
14. # if (event['session']['application']['applicationId'] !=
15. # "amzn1.echo-sdk-ams.app.[unique-value-here]"):
16. # raise ValueError("Invalid Application ID")
18. if event['session']['new']:
19. on\_session\_started({'requestId': event['request']['requestId']},
20. event['session'])
22. if event['request']['type'] == "LaunchRequest":
23. return on\_launch(event['request'], event['session'])
24. elif event['request']['type'] == "IntentRequest":
25. return on\_intent(event['request'], event['session'])
26. elif event['request']['type'] == "SessionEndedRequest":
27. return on\_session\_ended(event['request'], event['session'])

30. def on\_session\_started(session\_started\_request, session):
31. """ Called when the session starts """
33. print("on\_session\_started requestId=" + session\_started\_request['requestId']
34. + ", sessionId=" + session['sessionId'])

37. def on\_launch(launch\_request, session):
38. """ Called when the user launches the skill without specifying what they
39. want
40. """
42. print("on\_launch requestId=" + launch\_request['requestId'] +
43. ", sessionId=" + session['sessionId'])
44. # Dispatch to your skill's launch
45. return get\_welcome\_response()

48. def on\_intent(intent\_request, session):
49. """ Called when the user specifies an intent for this skill """
51. print("on\_intent requestId=" + intent\_request['requestId'] +
52. ", sessionId=" + session['sessionId'])
54. intent = intent\_request['intent']
55. intent\_name = intent\_request['intent']['name']
56. print(intent\_name)
57. # Dispatch to your skill's intent handlers
58. if intent\_name == "ThingsYouCanDo":
59. return list\_commands()
60. elif intent\_name =="ExecutingCommands":
61. return sending\_command\_to\_device(intent,session)
62. elif intent\_name == "DialogCommands":
63. return dialogCommand(intent,session)
64. elif intent\_name == "AMAZON.HelpIntent":
65. return get\_welcome\_response()
66. elif intent\_name=="WhatsMyCommandIntent":
67. return get\_command\_from\_session(intent,session)
68. elif intent\_name=="OpeingNormalPrograms":
69. return sending\_command\_to\_device(intent,session)
70. elif intent\_name=="AMAZON.FallbackIntent":
71. return help(intent,session)
72. else:
73. print(intent\_name)
74. raise ValueError("Invalid intent")

77. def on\_session\_ended(session\_ended\_request, session):
78. """ Called when the user ends the session.
79. Is not called when the skill returns should\_end\_session=true
80. """
81. print("on\_session\_ended requestId=" + session\_ended\_request['requestId'] +
82. ", sessionId=" + session['sessionId'])
83. session\_attributes = {}
84. card\_title = "Error"
85. speech\_output = "Thanking for using our skill"
87. # If the user either does not reply to the welcome message or says something
88. # that is not understood, they will be prompted again with this text.
89. reprompt\_text = "come back gain"
90. should\_end\_session = False
91. return build\_response(session\_attributes, build\_speechlet\_response(
92. card\_title, speech\_output, reprompt\_text, should\_end\_session))
93. # add cleanup logic here
95. # --------------- Functions that control the skill's behavior ------------------

98. def get\_welcome\_response():
99. """ If we wanted to initialize the session to have some attributes we could
100. add those here
101. """
102. url=getting\_device\_location()
103. session\_attributes = {}
104. card\_title = "Welcome"
105. html = requests.get(url)
106. speech\_output = "Welcome to We R noobs automation feel good to intract with us. "+html.text
108. # If the user either does not reply to the welcome message or says something
109. # that is not understood, they will be prompted again with this text.
110. reprompt\_text = "say , create project or Get results or clean my Desktop"
111. should\_end\_session = False
112. return build\_response(session\_attributes, build\_speechlet\_response(
113. card\_title, speech\_output, reprompt\_text, should\_end\_session))

116. def set\_command\_in\_session(intent, session):
117. """ Sets the command in the session and prepares the speech to reply to the
118. user.
119. """
121. card\_title = intent['name']
122. session\_attributes = {}
123. should\_end\_session = False
125. if 'Command' in intent['slots']:
126. favorite\_command = intent['slots']['Command']['value']
127. session\_attributes = create\_favorite\_command\_attributes(favorite\_command)
128. html = requests.get('http://example.ngrok.io/command?command='+favorite\_command)
129. speech\_output = "I sent the command to your system.Let me know if you want me to send another command."
130. reprompt\_text = "Please tell me the command I should send to your system by saying, " \
131. "Send the shutdown command"
132. else:
133. speech\_output = "I did not understand that. Please try again."
134. reprompt\_text = "Please tell me the command I should send to your system by saying, " \
135. "Send the shutdown command"

138. return build\_response(session\_attributes, build\_speechlet\_response(
139. card\_title, speech\_output, reprompt\_text, should\_end\_session))


143. def create\_favorite\_command\_attributes(favorite\_command):
144. return {"favoriteCommand": favorite\_command}

147. def help(intent,session):
148. session\_attributes = {}
149. card\_title = "Error"
150. speech\_output = "unable to understand you request"
152. # If the user either does not reply to the welcome message or says something
153. # that is not understood, they will be prompted again with this text.
154. reprompt\_text = "say , create flutter project or Get results or clean my Desktop \n To know more command say , what can you do for me"
155. should\_end\_session = False
156. return build\_response(session\_attributes, build\_speechlet\_response(
157. card\_title, speech\_output, reprompt\_text, should\_end\_session))

160. def dialogCommand(intent,session):
162. url=getting\_device\_location()
164. passingcommand=gettingvalueFormintent(intent['slots']['commands'])
165. arguments=intent['slots']['projectname']['value']
166. favorite\_command =intent['slots']['commands']['value']
167. card\_title =intent['name']
168. # print(url+"/dialogcommand?command="+passingcommand+"&args="+arguments)
169. html = requests.get(url+"/dialogcommand?command="+passingcommand+"&args="+arguments)
170. should\_end\_session=False
171. speech\_output='Executing '+html.text
172. reprompt\_text="i think it is Executed"
174. session\_attributes=create\_favorite\_command\_attributes(favorite\_command)
175. return build\_response(session\_attributes, build\_speechlet\_response(
176. card\_title, speech\_output, reprompt\_text, should\_end\_session))

179. def get\_command\_from\_session(intent, session):
180. session\_attributes = {}
181. reprompt\_text = None
183. if "favoriteCommand" in session.get('attributes', {}):
184. favorite\_command = session['attributes']['favoriteCommand']
185. speech\_output = "Your last command was " + favorite\_command + \
186. ". Goodbye."
187. should\_end\_session = True
188. else:
189. speech\_output = "I'm not sure what your last command was. " \
190. "Please tell me the command I should send to your system by saying, " \
191. "Send the shutdown command"
192. should\_end\_session = False
194. # Setting reprompt\_text to None signifies that we do not want to reprompt
195. # the user. If the user does not respond or says something that is not
196. # understood, the session will end.
197. return build\_response(session\_attributes, build\_speechlet\_response(
198. intent['name'], speech\_output, reprompt\_text, should\_end\_session))

201. def list\_commands():
202. session\_attributes = {}
203. url=getting\_device\_location()
204. card\_title = "Commands"
205. html = requests.get(url+"/command?command=getcommands")
206. speech\_output = f"{html.text}"
207. # If the user either does not reply to the welcome message or says something
208. # that is not understood, they will be prompted again with this text.
209. reprompt\_text = "say , Create project \nDesktop Cleaner \nget Results"
210. should\_end\_session = False
211. return build\_response(session\_attributes, build\_speechlet\_response(
212. card\_title, speech\_output, reprompt\_text, should\_end\_session))

215. def sending\_command\_to\_device(intent, session):
216. url=getting\_device\_location()
217. session\_attributes = {}
219. if 'NormalProgram' in intent['slots'] :
220. favorite\_command =intent['slots']['NormalProgram']['value']
221. card\_title =intent['name']
222. value=gettingvalueFormintent(intent['slots']['NormalProgram'])
223. should\_end\_session=False
224. session\_attributes=create\_favorite\_command\_attributes(favorite\_command)
225. url = getting\_device\_location\_form\_tmp()
226. html = requests.get(url+"/command?command="+value)
227. speech\_output='opening ' +value
228. reprompt\_text="i think it is openend please check it"
230. elif 'SingleCommands' in intent['slots']:
231. favorite\_command =intent['slots']['SingleCommands']['value']
232. value=gettingvalueFormintent(intent['slots']['SingleCommands'])
233. card\_title =intent['name']
234. should\_end\_session=False
235. session\_attributes=create\_favorite\_command\_attributes(favorite\_command)
236. print(url)
237. html = requests.get(url+"/command?command="+value)
238. speech\_output='Executing ' +favorite\_command
239. reprompt\_text="i think it is Executed"
240. else :
241. card\_title="error"
242. speech\_output="unable to process ur request"
243. reprompt\_text="say, list commands"
244. should\_end\_session=True
245. return build\_response(session\_attributes, build\_speechlet\_response(
246. card\_title, speech\_output, reprompt\_text, should\_end\_session))
248. # --------------- Function for getting location of devcies ----------------------#
249. def getting\_device\_location():
250. url=requests.get("https://alexautomation.herokuapp.com/read?id=url")
251. with open('/tmp/url.txt', 'w') as file:
252. file.write(url.text)
253. return url.text

256. def getting\_device\_location\_form\_tmp():
257. try:
258. with open('/tmp/url.txt','r') as file:
259. gobal=file.readline()
260. if gobal!="":
261. # print(gobal)
262. return gobal
263. else :
264. getting\_Device\_location()
265. except:
266. print("identifier")
267. # --------------- Helpers that build all of the responses ----------------------#

270. def build\_speechlet\_response(title, output, reprompt\_text, should\_end\_session):
271. return {
272. 'outputSpeech': {
273. 'type': 'PlainText',
274. 'text': output
275. },
276. 'card': {
277. 'type': 'Simple',
278. 'title': 'WeRnoobs - ' + title,
279. 'content': output
280. },
281. 'reprompt': {
282. 'outputSpeech': {
283. 'type': 'PlainText',
284. 'text': reprompt\_text
285. }
286. },
287. 'shouldEndSession': should\_end\_session
288. }

291. def build\_response(session\_attributes, speechlet\_response):
292. return {
293. 'version': '1.0',
294. 'sessionAttributes': session\_attributes,
295. 'response': speechlet\_response
296. }

299. #--- utils--#
300. def gettingvalueFormintent(intent):
301. value=""
302. for i in intent['resolutions']['resolutionsPerAuthority']:
303. a=i['values']
304. for i in a:
305. value=i['value']['name']
306. return value

**HEROKU**

**App.py**

1. from flask import Flask, escape, request, jsonify,render\_template
2. from firebase\_admin import credentials, firestore, initialize\_app
3. import json
4. cred = credentials.Certificate("key.json")
5. default\_app = initialize\_app(cred)
6. db = firestore.client()
7. todo\_ref = db.collection('urls')
9. app = Flask(\_\_name\_\_)

12. @app.route('/')
13. def index():
14. return render\_template("index.html")

17. @app.route('/command', methods=['GET'])
18. def handle\_command():
19. command = request.args.get('command', '')
20. return command

23. @app.route('/add', methods=['POST'])
24. def create():
25. # http://300bf87b.ngrok.io/add?url=https://google.in
26. try:
27. \_urls = str(request.args.get('url', ''))
28. \_data = {u"url": \_urls}
29. todo\_ref.document("url").set(\_data)
30. t = todo\_ref.document("url").get()
31. print("checking")
32. # read()
33. if t.to\_dict() == \_data:
34. return jsonify({"success": True}), 200
35. else:
36. return jsonify({"data not entered correctly": False}), 400
37. except Exception as e:
38. return f"An Error Occured: {e}"

41. @app.route('/read', methods=['GET'])
42. def read():
43. # http://300bf87b.ngrok.io/read?id=url
45. try:
46. todo\_id = request.args.get('id')
47. print(todo\_id)
48. if todo\_id:
49. todo = todo\_ref.document(todo\_id).get()
50. # y=todo.to\_dict()
51. print(todo.to\_dict()["url"])
52. return todo.to\_dict()["url"], 200
53. else:
54. all\_todos = [doc.to\_dict() for doc in todo\_ref.stream()]
55. return jsonify(all\_todos), 200
56. except Exception as e:
57. return f"An Error Occured: {e}"

60. @app.route('/delete', methods=['GET', 'DELETE'])
61. def delete():
62. # http://300bf87b.ngrok.io/delete?id=url
63. try:
64. todo\_id = request.args.get('id')
65. todo\_ref.document(todo\_id).delete()
66. return jsonify({"success": True}), 200
67. except Exception as e:
68. return f"An Error Occured: {e}"

71. if \_\_name\_\_ == "\_\_main\_\_":
72. app.jinja\_env.auto\_reload = True
73. app.config['TEMPLATES\_AUTO\_RELOAD'] = True
74. app.run(debug=True)

**CLIENT API**

**APP.PY**

1. from Executing import ExecutingCommands as EC
2. from flask import Flask, escape, request
3. import os
4. app = Flask(\_\_name\_\_)

7. @app.route('/')
8. def hello():
9. name = os.environ['COMPUTERNAME']
10. return f'Hello, {escape(name)}!'

13. @app.route('/command', methods=['GET'])
14. def handle\_command():
15. command = request.args.get('command', '').replace(" ", "")
16. print(command)
17. return EC.handle\_Commands(command)

20. @app.route('/dialogcommand', methods=['GET'])
21. def handle\_dialogcommand():
22. command = request.args.get('command', '').replace(" ", "")
23. args = request.args.get('args', '').replace(" ", "")
24. print(command)
25. return EC.DialogCommand(command, args)

28. if \_\_name\_\_ == "\_\_main\_\_":
29. app.jinja\_env.auto\_reload = True
30. app.config['TEMPLATES\_AUTO\_RELOAD'] = True
31. app.run(debug=True)

 Executing\_command.py

1. import os
2. import os.path
3. import shutil

6. def handle\_Commands(command):
7. #-----opeing file to check the where it suppoerts are not-----#
8. with open("Executing//WindowsApps.txt", "r") as \_file:
9. \_WindowsApps = \_file.read()
10. with open("Executing//automatoins.txt") as \_file:
11. \_automation = \_file.read()
13. #-----Main function-----#
14. if command in \_WindowsApps:
15. os.popen(command)
16. elif command in \_automation:
17. print(command)
18. return automation(command)
19. elif command == "getcommands":
20. return listingCommands(command)
21. else:
22. return "no commands found"
23. return "opened"

26. def automation(command):
27. pwd = os.getcwd()
28. os.popen(pwd+"\\Executing\\"+command+".lnk")
29. return f"{command} started"

32. def DialogCommand(command, args):
33. pwd = os.getcwd()
34. os.popen(pwd+"\\Executing\\"+command+".lnk " + args)
35. return f"{command} started"

38. def listingCommands(command):
39. with open("Executing/AlexaCando.txt") as \_file:
40. data = \_file.read()
41. return str(data)
42. import os
43. import os.path
44. import shutil

47. def handle\_Commands(command):
48. #-----opeing file to check the where it suppoerts are not-----#
49. with open("Executing//WindowsApps.txt", "r") as \_file:
50. \_WindowsApps = \_file.read()
51. with open("Executing//automatoins.txt") as \_file:
52. \_automation = \_file.read()
54. #-----Main function-----#
55. if command in \_WindowsApps:
56. os.popen(command)
57. elif command in \_automation:
58. print(command)
59. return automation(command)
60. elif command == "getcommands":
61. return listingCommands(command)
62. else:
63. return "no commands found"
64. return "opened"

67. def automation(command):
68. pwd = os.getcwd()
69. os.popen(pwd+"\\Executing\\"+command+".lnk")
70. return f"{command} started"

73. def DialogCommand(command, args):
74. pwd = os.getcwd()
75. os.popen(pwd+"\\Executing\\"+command+".lnk " + args)
76. return f"{command} started"

79. def listingCommands(command):
80. with open("Executing/AlexaCando.txt") as \_file:
81. data = \_file.read()
82. return str(data)

**AUTOMATIONS**

**DesktopCleaner.py**

1. import os
2. import shutil
4. UserProfile = os.environ.get('USERPROFILE')
6. folder\_to\_track = f'{UserProfile}\\Desktop\\'
8. os.chdir(folder\_to\_track)

11. extensions\_folders = {
12. # No name
13. 'noname': f"{UserProfile}\\Desktop\\Other\\Uncategorized",
14. # Audio
15. '.aif': f"{UserProfile}\\Desktop\\Media\\Audio",
16. '.cda': f"{UserProfile}\\Desktop\\Media\\Audio",
17. '.mid': f"{UserProfile}\\Desktop\\Media\\Audio",
18. '.midi': f"{UserProfile}\\Desktop\\Media\\Audio",
19. '.mp3': f"{UserProfile}\\Desktop\\Media\\Audio",
20. '.mpa': f"{UserProfile}\\Desktop\\Media\\Audio",
21. '.ogg': f"{UserProfile}\\Desktop\\Media\\Audio",
22. '.wav': f"{UserProfile}\\Desktop\\Media\\Audio",
23. '.wma': f"{UserProfile}\\Desktop\\Media\\Audio",
24. '.wpl': f"{UserProfile}\\Desktop\\Media\\Audio",
25. '.m3u': f"{UserProfile}\\Desktop\\Media\\Audio",
26. # Docs
27. '.txt': f"{UserProfile}\\Desktop\\Docs\\TextFiles",
28. '.doc': f"{UserProfile}\\Desktop\\Docs\\Microsoft\\Word",
29. '.docx': f"{UserProfile}\\Desktop\\Docs\\Microsoft\\Word",
30. '.odt ': f"{UserProfile}\\Desktop\\Docs\\TextFiles",
31. '.pdf': f"{UserProfile}\\Desktop\\Docs\\PDF",
32. '.rtf': f"{UserProfile}\\Desktop\\Docs\\TextFiles",
33. '.tex': f"{UserProfile}\\Desktop\\Docs\\TextFiles",
34. '.wks ': f"{UserProfile}\\Desktop\\Docs\\TextFiles",
35. '.wps': f"{UserProfile}\\Desktop\\Docs\\TextFiles",
36. '.wpd': f"{UserProfile}\\Desktop\\Docs\\TextFiles",
37. # Video
38. '.3g2': f"{UserProfile}\\Desktop\\Media\\Video",
39. '.3gp': f"{UserProfile}\\Desktop\\Media\\Video",
40. '.avi': f"{UserProfile}\\Desktop\\Media\\Video",
41. '.flv': f"{UserProfile}\\Desktop\\Media\\Video",
42. '.h264': f"{UserProfile}\\Desktop\\Media\\Video",
43. '.m4v': f"{UserProfile}\\Desktop\\Media\\Video",
44. '.mkv': f"{UserProfile}\\Desktop\\Media\\Video",
45. '.mov': f"{UserProfile}\\Desktop\\Media\\Video",
46. '.mp4': f"{UserProfile}\\Desktop\\Media\\Video",
47. '.mpg': f"{UserProfile}\\Desktop\\Media\\Video",
48. '.mpeg': f"{UserProfile}\\Desktop\\Media\\Video",
49. '.rm': f"{UserProfile}\\Desktop\\Media\\Video",
50. '.swf': f"{UserProfile}\\Desktop\\Media\\Video",
51. '.vob': f"{UserProfile}\\Desktop\\Media\\Video",
52. '.wmv': f"{UserProfile}\\Desktop\\Media\\Video",
53. # Images
54. '.ai': f"{UserProfile}\\Desktop\\Media\\Images",
55. '.bmp': f"{UserProfile}\\Desktop\\Media\\Images",
56. '.gif': f"{UserProfile}\\Desktop\\Media\\Images",
57. '.ico': f"{UserProfile}\\Desktop\\Media\\Images",
58. '.jpg': f"{UserProfile}\\Desktop\\Media\\Images",
59. '.jpeg': f"{UserProfile}\\Desktop\\Media\\Images",
60. '.png': f"{UserProfile}\\Desktop\\Media\\Images",
61. '.ps': f"{UserProfile}\\Desktop\\Media\\Images",
62. '.psd': f"{UserProfile}\\Desktop\\Media\\Images",
63. '.svg': f"{UserProfile}\\Desktop\\Media\\Images",
64. '.tif': f"{UserProfile}\\Desktop\\Media\\Images",
65. '.tiff': f"{UserProfile}\\Desktop\\Media\\Images",
66. '.CR2': f"{UserProfile}\\Desktop\\Media\\Images",
67. # Internet
68. '.asp': f"{UserProfile}\\Desktop\\Other\\Internet",
69. '.aspx': f"{UserProfile}\\Desktop\\Other\\Internet",
70. '.cer': f"{UserProfile}\\Desktop\\Other\\Internet",
71. '.cfm': f"{UserProfile}\\Desktop\\Other\\Internet",
72. '.cgi': f"{UserProfile}\\Desktop\\Other\\Internet",
73. '.pl': f"{UserProfile}\\Desktop\\Other\\Internet",
74. '.css': f"{UserProfile}\\Desktop\\Other\\Internet",
75. '.htm': f"{UserProfile}\\Desktop\\Other\\Internet",
76. '.js': f"{UserProfile}\\Desktop\\Other\\Internet",
77. '.jsp': f"{UserProfile}\\Desktop\\Other\\Internet",
78. '.part': f"{UserProfile}\\Desktop\\Other\\Internet",
79. '.php': f"{UserProfile}\\Desktop\\Other\\Internet",
80. '.rss': f"{UserProfile}\\Desktop\\Other\\Internet",
81. '.xhtml': f"{UserProfile}\\Desktop\\Other\\Internet",
82. # Compressed
83. '.7z': f"{UserProfile}\\Desktop\\Other\\Compressed",
84. '.arj': f"{UserProfile}\\Desktop\\Other\\Compressed",
85. '.deb': f"{UserProfile}\\Desktop\\Other\\Compressed",
86. '.pkg': f"{UserProfile}\\Desktop\\Other\\Compressed",
87. '.rar': f"{UserProfile}\\Desktop\\Other\\Compressed",
88. '.rpm': f"{UserProfile}\\Desktop\\Other\\Compressed",
89. '.tar.gz': f"{UserProfile}\\Desktop\\Other\\Compressed",
90. '.z': f"{UserProfile}\\Desktop\\Other\\Compressed",
91. '.zip': f"{UserProfile}\\Desktop\\Other\\Compressed",
92. # Disc
93. '.bin': f"{UserProfile}\\Desktop\\Other\\Disc",
94. '.dmg': f"{UserProfile}\\Desktop\\Other\\Disc",
95. '.iso': f"{UserProfile}\\Desktop\\Other\\Disc",
96. '.toast': f"{UserProfile}\\Desktop\\Other\\Disc",
97. '.vcd': f"{UserProfile}\\Desktop\\Other\\Disc",
98. # Data
99. '.csv': f"{UserProfile}\\Desktop\\Programming\\Database",
100. '.dat': f"{UserProfile}\\Desktop\\Programming\\Database",
101. '.db': f"{UserProfile}\\Desktop\\Programming\\Database",
102. '.dbf': f"{UserProfile}\\Desktop\\Programming\\Database",
103. '.log': f"{UserProfile}\\Desktop\\Programming\\Database",
104. '.mdb': f"{UserProfile}\\Desktop\\Programming\\Database",
105. '.sav': f"{UserProfile}\\Desktop\\Programming\\Database",
106. '.sql': f"{UserProfile}\\Desktop\\Programming\\Database",
107. '.tar': f"{UserProfile}\\Desktop\\Programming\\Database",
108. '.xml': f"{UserProfile}\\Desktop\\Programming\\Database",
109. '.json': f"{UserProfile}\\Desktop\\Programming\\Database",
110. # Executables
111. '.apk': f"{UserProfile}\\Desktop\\Other\\Executables",
112. '.bat': f"{UserProfile}\\Desktop\\Other\\Executables",
113. '.com': f"{UserProfile}\\Desktop\\Other\\Executables",
114. '.exe': f"{UserProfile}\\Desktop\\Other\\Executables",
115. '.gadget': f"{UserProfile}\\Desktop\\Other\\Executables",
116. '.jar': f"{UserProfile}\\Desktop\\Other\\Executables",
117. '.wsf': f"{UserProfile}\\Desktop\\Other\\Executables",
118. # Fonts
119. '.fnt': f"{UserProfile}\\Desktop\\Other\\Fonts",
120. '.fon': f"{UserProfile}\\Desktop\\Other\\Fonts",
121. '.otf': f"{UserProfile}\\Desktop\\Other\\Fonts",
122. '.ttf': f"{UserProfile}\\Desktop\\Other\\Fonts",
123. # Presentations
124. '.key': f"{UserProfile}\\Desktop\\Docs\\Presentations",
125. '.odp': f"{UserProfile}\\Desktop\\Docs\\Presentations",
126. '.pps': f"{UserProfile}\\Desktop\\Docs\\Presentations",
127. '.ppt': f"{UserProfile}\\Desktop\\Docs\\Presentations",
128. '.pptx': f"{UserProfile}\\Desktop\\Docs\\Presentations",
129. # Programming
130. '.c': f"{UserProfile}\\Desktop\\Programming\\C&C++",
131. '.class': f"{UserProfile}\\Desktop\\Programming\\Java",
132. '.dart': f"{UserProfile}\\Desktop\\Programming\\Dart",
133. '.py': f"{UserProfile}\\Desktop\\Programming\\Python",
134. '.sh': f"{UserProfile}\\Desktop\\Programming\\Shell",
135. '.swift': f"{UserProfile}\\Desktop\\Programming\\Swift",
136. '.html': f"{UserProfile}\\Desktop\\Programming\\C&C++",
137. '.h': f"{UserProfile}\\Desktop\\Programming\\C&C++",
138. # Spreadsheets
139. '.ods': f"{UserProfile}\\Desktop\\Docs\\Microsoft\\Excel",
140. '.xlr': f"{UserProfile}\\Desktop\\Docs\\Microsoft\\Excel",
141. '.xls': f"{UserProfile}\\Desktop\\Docs\\Microsoft\\Excel",
142. '.xlsx': f"{UserProfile}\\Desktop\\Docs\\Microsoft\\Excel",
143. # System
144. '.bak': f"{UserProfile}\\Desktop\\Other\\System",
145. '.cab': f"{UserProfile}\\Desktop\\Other\\System",
146. '.cfg': f"{UserProfile}\\Desktop\\Other\\System",
147. '.cpl': f"{UserProfile}\\Desktop\\Other\\System",
148. '.cur': f"{UserProfile}\\Desktop\\Other\\System",
149. '.dll': f"{UserProfile}\\Desktop\\Other\\System",
150. '.dmp': f"{UserProfile}\\Desktop\\Other\\System",
151. '.drv': f"{UserProfile}\\Desktop\\Other\\System",
152. '.icns': f"{UserProfile}\\Desktop\\Other\\System",
153. '.ini': f"{UserProfile}\\Desktop\\Other\\System",
154. '.lnk': f"{UserProfile}\\Desktop\\Other\\System",
155. '.msi': f"{UserProfile}\\Desktop\\Other\\System",
156. '.sys': f"{UserProfile}\\Desktop\\Other\\System",
157. '.tmp': f"{UserProfile}\\Desktop\\Other\\System",
159. '': f"{UserProfile}\\Desktop\\Folders"
160. }

163. def Cleaner():
164. for filename in os.listdir(folder\_to\_track):
165. i = 1
166. if filename not in ['desktop.ini', 'automations']:
167. try:
168. new\_name = filename
169. extension = 'noname'
170. try:
171. extension = str(os.path.splitext(
172. folder\_to\_track + '/' + filename)[1])
173. except Exception:
174. extension = 'noname'
176. folder\_destination\_path = extensions\_folders[extension]
177. if not os.path.exists(folder\_destination\_path):
178. os.makedirs(folder\_destination\_path)
180. file\_exists = os.path.isfile(
181. folder\_destination\_path + "/" + new\_name)
182. while file\_exists:
183. i += 1
184. new\_name = os.path.splitext(folder\_to\_track + '/' + filename)[0] + str(
185. i) + os.path.splitext(folder\_to\_track + '/' + filename)[1]
186. new\_name = new\_name.split("/")[4]
187. file\_exists = os.path.isfile(
188. folder\_destination\_path + "/" + new\_name)
189. src = folder\_to\_track + "/" + filename
191. new\_name = folder\_destination\_path + "/" + new\_name
192. os.rename(src, new\_name)
193. except Exception as e:
194. print(e)

197. Cleaner()

**Create project(Flutter)**

1. import argparse
2. import os
3. from github import Github

6. def local():
7. try:
8. if os.path.exists(path):
9. os.chdir(path)
10. else:
11. print('Cannot create a file when that file already exists')
12. exit()
13. os.system(f'flutter create {foldername}')
14. os.chdir(\_dir)
15. os.system('git init')
16. os.system(f'echo "# {foldername}" > README.md')
17. os.system('git add README.md')
18. os.system('git commit -m "first commit"')
20. print(f'{foldername} created locally')
21. os.system('code .')
23. except Exception as e:
24. print(e)

27. def remote():
28. try:
29. g = Github(token)
30. user = g.get\_user()
31. login = user.login
32. user.create\_repo(foldername)
33. except Exception as e:
34. print(e)
36. commands = ['git init',
37. f'git remote add origin https://github.com/{login}/{foldername}.git',
38. 'git add .',
39. 'git commit -m "Initial commit"',
40. 'git push -u origin master']
42. try:
43. os.chdir(path)
44. os.system(f'flutter create {foldername}')
45. os.chdir(\_dir)
46. for c in commands:
47. os.system(c)
49. print(f'{foldername} initialized')
50. os.system('code .')
52. except Exception as e:
53. print(e)

56. if \_\_name\_\_ == "\_\_main\_\_":
58. UserProfile = os.environ.get('USERPROFILE')
60. # add projects dirctory to the env vars
61. path = f'{UserProfile}\\Desktop'
62. token = os.environ.get('gt') # add github token to the env vars
64. p = argparse.ArgumentParser()
65. p.add\_argument('name', help='name of the project')
66. p.add\_argument('-f', default='l', help='name of the project')
67. args = p.parse\_args()
69. foldername = args.name
70. \_dir = path + '\\' + foldername
72. if args.f == 'l':
73. local()
74. elif args.f == 'r':
75. remote()
76. else:
77. print('fpia --help for help')