

--- Config.py ---

#ignored not pushed to git!

class Config:

DISCORD_TOKEN =

'MTI2OTM4MTE4OTA1NjMzNTk3Mw.GJdUct.-2RsoynZh78VFGdoXdrXWFhFQPbUCHM7V2w-u8'

CHANNEL_ID = 1269383349278081054

DATABASE_PASSWORD = 'postgres'

--- css_selectors.py ---

class Selectors:

SELECTORS = {

"google": {

"url": "https://www.google.com/"

},

"ebay": {

"url": "https://signin.ebay.com/signin/",

"email_field": "#userid",

"continue_button": "[data-testid*='signin-continue-btn']",

"password_field": "#pass",

"login_button": "#sgnBt",

"price": ".x-price-primary span" # CSS selector for Ebay price

},

"bestbuy": {

"priceUrl":

"https://www.bestbuy.com/site/microsoft-xbox-wireless-controller-for-xbox-series-x-xbox-series-s-xb

ox-one-windows-devices-sky-cipher-special-edition/6584960.p?skuId=6584960",

"url": "https://www.bestbuy.com/signin/",

```

"email_field": "#fld-e",

#"continue_button": ".cia-form__controls button",

"password_field": "#fld-p1",

"SignIn_button": ".cia-form__controls button",

"price": "[data-testid='customer-price'] span", # CSS selector for BestBuy price

"homePage": ".v-p-right-xxs.line-clamp"

},

"opentable": {

"url": "https://www.opentable.com/",

"unavailableUrl": "https://www.opentable.com/r/bar-spero-washington/",

"availableUrl": "https://www.opentable.com/r/the-rux-nashville",

"availableUrl2": "https://www.opentable.com/r/hals-the-steakhouse-nashville",

"date_field": "#restProfileSideBarDtpDayPicker-label",

"time_field": "#restProfileSideBarDtpTimePickerDtpPicker",

"select_date": "#restProfileSideBarDtpDayPicker-wrapper", # button[aria-label*="{ }"]

"select_time": "h3[data-test='select-time-header']",

"no_availability": "div._8ye6OVzeOuU- span",

"find_table_button": ".find-table-button", # Example selector for the Find Table button

"availability_result": ".availability-result", # Example selector for availability results

    "show_next_available_button": "button[data-test='multi-day-availability-button']", # Show
next available button

    "available_dates": "ul[data-test='time-slots'] > li", # Available dates and times

}

}

```

@staticmethod

```
def get_selectors_for_url(url):  
    for keyword, selectors in Selectors.SELECTORS.items():  
        if keyword in url.lower():  
            return selectors  
  
    return None # Return None if no matching selectors are found
```

--- exportUtils.py ---

```
import os
```

```
import pandas as pd
```

```
from datetime import datetime
```

```
class ExportUtils:
```

```
    @staticmethod
```

```
    def log_to_excel(command, url, result, entered_date=None, entered_time=None):
```

```
        # Determine the file path for the Excel file
```

```
        file_name = f"{command}.xlsx"
```

```
        file_path = os.path.join("ExportedFiles", "excelFiles", file_name)
```

```
        # Ensure directory exists
```

```
        os.makedirs(os.path.dirname(file_path), exist_ok=True)
```

```
        # Timestamp for current run
```

```
        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

```
        # If date/time not entered, use current timestamp
```

```
        entered_date = entered_date or datetime.now().strftime('%Y-%m-%d')
```

```
entered_time = entered_time or datetime.now().strftime('%H:%M:%S')
```

```
# Check if the file exists and create the structure if it doesn't
```

```
if not os.path.exists(file_path):
```

```
    df = pd.DataFrame(columns=["Timestamp", "Command", "URL", "Result", "Entered Date",  
"Entered Time"])
```

```
    df.to_excel(file_path, index=False)
```

```
# Load existing data from the Excel file
```

```
df = pd.read_excel(file_path)
```

```
# Append the new row
```

```
new_row = {  
    "Timestamp": timestamp,  
    "Command": command,  
    "URL": url,  
    "Result": result,  
    "Entered Date": entered_date,  
    "Entered Time": entered_time  
}
```

```
# Add the new row to the existing data and save it back to Excel
```

```
df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True)
```

```
df.to_excel(file_path, index=False)
```

```
return f"Data saved to Excel file at {file_path}."
```

@staticmethod

```
def export_to_html(command, url, result, entered_date=None, entered_time=None):
```

```
    """Export data to HTML format with the same structure as Excel."""
```

```
    # Define file path for HTML
```

```
    file_name = f"{command}.html"
```

```
    file_path = os.path.join("ExportedFiles", "htmlFiles", file_name)
```

```
    # Ensure directory exists
```

```
    os.makedirs(os.path.dirname(file_path), exist_ok=True)
```

```
    # Timestamp for current run
```

```
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

```
    # If date/time not entered, use current timestamp
```

```
    entered_date = entered_date or datetime.now().strftime('%Y-%m-%d')
```

```
    entered_time = entered_time or datetime.now().strftime('%H:%M:%S')
```

```
    # Data row to insert
```

```
    new_row = {
```

```
        "Timestamp": timestamp,
```

```
        "Command": command,
```

```
        "URL": url,
```

```
        "Result": result,
```

```
        "Entered Date": entered_date,
```

```
        "Entered Time": entered_time
```

```
    }
```

```

# Check if the HTML file exists and append rows

if os.path.exists(file_path):

    # Open the file and append rows

    with open(file_path, "r+", encoding="utf-8") as file:

        content = file.read()

        # Look for the closing </table> tag and append new rows before it

        if "</table>" in content:

                                                    new_row_html    =

f"<tr><td>{new_row['Timestamp']}</td><td>{new_row['Command']}</td><td>{new_row['URL']}</td><

td>{new_row['Result']}</td><td>{new_row['Entered          Date']}</td><td>{new_row['Entered

Time']}</td></tr>\n"

        content = content.replace("</table>", new_row_html + "</table>")

        file.seek(0) # Move pointer to the start

        file.write(content)

        file.truncate() # Truncate any remaining content

        file.flush() # Flush the buffer to ensure it's written

else:

    # If the file doesn't exist, create a new one with table headers

    with open(file_path, "w", encoding="utf-8") as file:

        html_content = "<html><head><title>Command Data</title></head><body>"

        html_content += f"<h1>Results for {command}</h1><table border='1'>"

                                                    html_content    +=

f"<tr><th>Timestamp</th><th>Command</th><th>URL</th><th>Result</th><th>Entered

Date</th><th>Entered Time</th></tr>"

                                                    html_content    +=

f"<tr><td>{new_row['Timestamp']}</td><td>{new_row['Command']}</td><td>{new_row['URL']}</td><

```

```

td>{new_row['Result']}</td><td>{new_row['Entered
Date']}</td><td>{new_row['Entered
Time']}</td></tr>\n"

    html_content += "</table></body></html>"

    file.write(html_content)

    file.flush() # Ensure content is written to disk

    return f"HTML file saved and updated at {file_path}."

```

--- MyBot.py ---

```

import discord

from discord.ext import commands

from boundary.BrowserBoundary import BrowserBoundary
from boundary.AccountBoundary import AccountBoundary
from boundary.AvailabilityBoundary import AvailabilityBoundary
from boundary.PriceBoundary import PriceBoundary
from boundary.BotBoundary import BotBoundary

from DataObjects.global_vars import GlobalState # Import the global variable

# Bot initialization

intents = discord.Intents.default()

intents.message_content = True # Enable reading message content

class MyBot(commands.Bot):

    def __init__(self, *args, **kwargs):

        super().__init__(*args, **kwargs)

```

```

async def on_message(self, message):

    if message.author == self.user: # Prevent the bot from replying to its own messages

        return

    print(f"Message received: {message.content}")

    GlobalState.user_message = message.content

    if GlobalState.user_message.lower() in ["hi", "hey", "hello"]:

        await message.channel.send("Hi, how can I help you?")

    elif GlobalState.user_message.startswith("!"):

        print("User message: ", GlobalState.user_message)

    else:

        await message.channel.send("I'm sorry, I didn't understand that. Type !project_help to see
the list of commands.")

    await self.process_commands(message)

    GlobalState.reset_user_message() # Reset the global user_message variable

    #print("User_message reset to empty string")

async def setup_hook(self):

    await self.add_cog(BrowserBoundary()) # Add your boundary objects

    await self.add_cog(AccountBoundary())

    await self.add_cog(AvailabilityBoundary())

    await self.add_cog(PriceBoundary())

```



```

await self.add_cog(BotBoundary())

async def on_ready(self):

    print(f"Logged in as {self.user}")

    channel = discord.utils.get(self.get_all_channels(), name="general") # Adjust the channel
name if needed

    if channel:

        await channel.send("Hi, I'm online! Type '!project_help' to see what I can do.")

async def on_command_error(self, ctx, error):

    if isinstance(error, commands.CommandNotFound):

        print("Command not recognized:")

        print(error)

        await ctx.channel.send("I'm sorry, I didn't understand that. Type !project_help to see the list
of commands.")

# Initialize the bot instance

bot = MyBot(command_prefix="!", intents=intents, case_insensitive=True)

def start_bot(token):

    """Run the bot with the provided token."""

    bot.run(token)

```