```
--- AccountBoundary.py ---
from discord.ext import commands
from control.AccountControl import AccountControl
from DataObjects.global_vars import GlobalState

class AccountBoundary(commands.Cog):
    def __init__(self):
        self.control = AccountControl()  # Initialize control object

    @commands.command(name="fetch_all_accounts")
    async def fetch_all_accounts(self, ctx):
        await ctx.send("Command recognized, passing data to control.")

        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command

        result = self.control.receive_command(command)

        # Send the result (prepared by control) back to the user
        await ctx.send(result)

    @commands.command(name="fetch_account_by_website")
    async def fetch_account_by_website(self, ctx):
        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
```

```
        command = list[0]  # First element is the command
        website = list[1]  # Second element is the URL

        await ctx.send(f"Command recognized, passing data to control for website {website}.")

        result = self.control.receive_command(command, website)

        # Send the result (prepared by control) back to the user
        await ctx.send(result)


    @commands.command(name="add_account")
    async def add_account(self, ctx):
        await ctx.send("Command recognized, passing data to control.")

         list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command
        username = list[1]  # Second element is the username
        password = list[2]  # Third element is the passwrod
        website = list[3]  # Third element is the website

        result = self.control.receive_command(command, username, password, website)

        # Send the result (prepared by control) back to the user
        await ctx.send(result)
```

```
    @commands.command(name="delete_account")
    async def delete_account(self, ctx):

        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command
        account_id = list[1]  # Second element is the account_id

            await ctx.send(f"Command recognized, passing data to control to delete account with ID {account_id}.")

        result = self.control.receive_command(command, account_id)

        # Send the result (prepared by control) back to the user
        await ctx.send(result)



--- AvailabilityBoundary.py ---
from discord.ext import commands
from control.AvailabilityControl import AvailabilityControl
from DataObjects.global_vars import GlobalState

class AvailabilityBoundary(commands.Cog):

    def __init__(self):
        # Initialize control objects directly
```

```
        self.availability_control = AvailabilityControl()



    @commands.command(name="check_availability")
    async def check_availability(self, ctx):
        await ctx.send("Command recognized, passing data to control.")

        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables

        command = list[0]  # First element is the command
        url = list[1]  # Second element is the URL
        date_str = list[2]  # Third element is the date

        # Pass the command and data to the control layer using receive_command
        result = await self.availability_control.receive_command(command, url, date_str)

        # Send the result back to the user
        await ctx.send(result)



    @commands.command(name="start_monitoring_availability")
    async def start_monitoring_availability(self, ctx):
        await ctx.send("Command recognized, passing data to control.")

        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
```

```
        command = list[0]  # First element is the command
        url = list[1]  # Second element is the URL
        date_str = list[2]  # Third element is the date
        frequency = list[3] # Fourth element is the frequency

        response = await self.availability_control.receive_command(command, url, date_str, frequency)

        # Send the result back to the user
        await ctx.send(response)


    @commands.command(name='stop_monitoring_availability')
    async def stop_monitoring_availability(self, ctx):
        """Command to stop monitoring the price."""
        await ctx.send("Command recognized, passing data to control.")

        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables

        command = list[0]  # First element is the command

            response = await self.availability_control.receive_command(command)            # Pass the command to the control layer
        await ctx.send(response)
```

```
--- BotBoundary.py ---
from discord.ext import commands
from control.BotControl import BotControl
from DataObjects.global_vars import GlobalState

class BotBoundary(commands.Cog):
    def __init__(self):
        self.control = BotControl()  # Initialize control object

    @commands.command(name="project_help")
    async def project_help(self, ctx):
        """Handle help command by sending available commands to the user."""
        await ctx.send("Command recognized, passing data to control.")
        try:
            list = GlobalState.parse_user_message(GlobalState.user_message)  # Parse the message
into command and up to 6 variables
            command = list[0]  # First element is the command

            response = await self.control.receive_command(command)  # Call control layer
            await ctx.send(response)  # Send the response back to the user
        except Exception as e:
            error_msg = f"Error in HelpBoundary: {str(e)}"
            print(error_msg)
            await ctx.send(error_msg)

    @commands.command(name="stop_bot")
    async def stop_bot(self, ctx):
```

```
        """Handle stop bot command by shutting down the bot."""
        await ctx.send("Command recognized, passing data to control.")
        try:
            list = GlobalState.parse_user_message(GlobalState.user_message)  # Parse the message into command and up to 6 variables
            command = list[0]  # First element is the command

            result = await self.control.receive_command(command, ctx)  # Call control layer to stop the bot
            print(result)  # Send the result to the terminal since the bot will shut down
        except Exception as e:
            error_msg = f"Error in StopBoundary: {str(e)}"
            print(error_msg)
            await ctx.send(error_msg)



--- BrowserBoundary.py ---
from discord.ext import commands
from control.BrowserControl import BrowserControl
from DataObjects.global_vars import GlobalState

class BrowserBoundary(commands.Cog):
    def __init__(self):
        self.browser_control = BrowserControl()  # Initialize Browser control object

    # Browser-related commands
    @commands.command(name='launch_browser')
```

```
    async def launch_browser(self, ctx):
        await ctx.send(f"Command recognized, passing to control object.")

        list = GlobalState.parse_user_message(GlobalState.user_message)  # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command

            result = await self.browser_control.receive_command(command)  # Pass the updated user_message to the control object
        await ctx.send(result)  # Send the result back to the user

    @commands.command(name="close_browser")
    async def close_browser(self, ctx):
        await ctx.send(f"Command recognized, passing to control object.")

        list = GlobalState.parse_user_message(GlobalState.user_message)  # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command

        result = await self.browser_control.receive_command(command)
        await ctx.send(result)

    # Login-related commands
    @commands.command(name='login')
    async def login(self, ctx):
        await ctx.send("Command recognized, passing data to control.")
```

```
        list = GlobalState.parse_user_message(GlobalState.user_message)  # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command
        website = list[1]

            result = await self.browser_control.receive_command(command, website)  # Pass the command and website to control object

        # Send the result back to the user
        await ctx.send(result)

    # Navigation-related commands
    @commands.command(name='navigate_to_website')
    async def navigate_to_website(self, ctx):
         await ctx.send("Command recognized, passing the data to control object.")  # Inform the user that the command is recognized

        list = GlobalState.parse_user_message(GlobalState.user_message)  # Parse the message into command and up to 6 variables

        command = list[0]  # First element is the command
        website = list[1]  # Second element is the URL

         result = await self.browser_control.receive_command(command, website)  # Pass the parsed variables to the control object
        await ctx.send(result)  # Send the result back to the user
```

```
--- PriceBoundary.py ---
from discord.ext import commands
from control.PriceControl import PriceControl
from DataObjects.global_vars import GlobalState

class PriceBoundary(commands.Cog):
    def __init__(self):
        # Initialize control objects directly
        self.price_control = PriceControl()

    @commands.command(name='get_price')
    async def get_price(self, ctx):
        """Command to get the price from the given URL."""
        await ctx.send("Command recognized, passing data to control.")

        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command
        website = list[1]  # Second element is the URL

        result = await self.price_control.receive_command(command, website) # Pass the command to the control layer
        await ctx.send(f"Price found: {result}")

    @commands.command(name='start_monitoring_price')
```

```
    async def start_monitoring_price(self, ctx):
        """Command to monitor price at given frequency."""
        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command
        website = list[1]  # Second element is the URL
        frequency = list[2]

        await ctx.send(f"Command recognized, starting price monitoring at {website} every {frequency} second(s).")

        response = await self.price_control.receive_command(command, website, frequency)
        await ctx.send(response)


    @commands.command(name='stop_monitoring_price')
    async def stop_monitoring_price(self, ctx):
        """Command to stop monitoring the price."""
        await ctx.send("Command recognized, passing data to control.")

        list = GlobalState.parse_user_message(GlobalState.user_message) # Parse the message into command and up to 6 variables
        command = list[0]  # First element is the command

        response = await self.price_control.receive_command(command)            # Pass the command to the control layer
```

```
        await ctx.send(response)
```

--- __init__.py ---

```
#empty init file
```