```
--- AccountControl.py ---
from DataObjects.AccountDAO import AccountDAO


class AccountControl:
    def __init__(self):
        self.account_dao = AccountDAO()  # DAO for database operations

    def add_account(self, username: str, password: str, website: str):
        """Add a new account to the database."""
        self.account_dao.connect()  # Establish database connection
            result = self.account_dao.add_account(username, password, website)  # Call DAO to add
account
        self.account_dao.close()  # Close the connection
        return result

    def delete_account(self, account_id: int):
        """Delete an account by ID."""
        self.account_dao.connect()  # Establish database connection
        result = self.account_dao.delete_account(account_id)
        self.account_dao.reset_id_sequence()  # Reset the ID sequence
        self.account_dao.close()  # Close the connection
        return result

    def fetch_all_accounts(self):
        """Fetch all accounts using the DAO."""
        self.account_dao.connect()  # Establish database connection
        accounts = self.account_dao.fetch_all_accounts()  # Fetch accounts from DAO
```

```
        self.account_dao.close()  # Close the connection
        return accounts if accounts else None

    def fetch_account_by_website(self, website: str):
        """Fetch an account by website."""
        self.account_dao.connect()  # Establish database connection
        account = self.account_dao.fetch_account_by_website(website)
        self.account_dao.close()  # Close the connection
        return account if account else None



--- CheckAvailabilityControl.py ---
from entity.AvailabilityEntity import AvailabilityEntity
from datetime import datetime

class CheckAvailabilityControl:
    def __init__(self, browser_entity):
        self.availability_entity = AvailabilityEntity(browser_entity)  # Initialize entity

    async def check_availability(self, url: str, date_str=None):
        """Handle the availability check and pass results for export."""
        # Get availability info from the entity layer
        availability_info = await self.availability_entity.check_availability(url, date_str)
        # Prepare the result message
        result = f"Checked availability: {availability_info}"

        # Create a DTO (Data Transfer Object) to organize the data for export
```

```
        data_dto = {
            "command": "start_monitoring_availability",  # Command executed
            "url": url,  # URL of the availability being monitored
            "result": result,  # Result of the availability check
            "entered_date": datetime.now().strftime('%Y-%m-%d'),  # Current date
            "entered_time": datetime.now().strftime('%H:%M:%S')  # Current time
        }

        # Pass the DTO to AvailabilityEntity to handle export to Excel and HTML
        self.availability_entity.export_data(data_dto)
        return result

--- CloseBrowserControl.py ---
class CloseBrowserControl:
    def __init__(self, browser_entity):
        self.browser_entity = browser_entity

    def close_browser(self):
        return self.browser_entity.close_browser()


--- GetPriceControl.py ---
from entity.PriceEntity import PriceEntity
from utils.css_selectors import Selectors

class GetPriceControl:
    def __init__(self, browser_entity):
```

```
        self.price_entity = PriceEntity(browser_entity)


    async def get_price(self, url: str):
        # Fetch the url using the correct CSS selector
        if not url:
            selectors = Selectors.get_selectors_for_url("bestbuy")
            url = selectors.get('priceUrl')  # Get the price URL
            if not url:
                return "No URL provided, and default URL for BestBuy could not be found."
            print("URL not provided, default URL for BestBuy is: " + url)

        # Step 3: Call the entity to get the price
        price = self.price_entity.get_price_from_page(url)
        return price




--- HelpControl.py ---
class HelpControl:
    def get_help_message(self):
        """Returns a list of available bot commands."""
        help_message = (
            "Here are the available commands:\n"
            "!project_help - Get help on available commands.\n"
            "!login 'website' - Log in to a website.\n"
            "!launch_browser - Launch the browser.\n"
            "!close_browser - Close the browser.\n"
```

```
            "!navigate_to_website - Navigate to a website.\n"
            "!get_price - Check the price of a product.\n"
            "!monitor_price - Monitor a product price.\n"
            "!stop_monitoring - Stop monitoring a product.\n"
            "!check_availability - Check the availability in a restaurant.\n"
            "!monitor_availability - Monitor the availability in a restaurant.\n"
            "!stop_monitoring_availability - Stop monitoring availability.\n"
            "!stop_bot - Stop the bot.\n"
        )
        return help_message


--- LaunchBrowserControl.py ---
class LaunchBrowserControl:
    def __init__(self, browser_entity):
        self.browser_entity = browser_entity

    def launch_browser(self):
        return self.browser_entity.launch_browser()


--- LoginControl.py ---
from entity.BrowserEntity import BrowserEntity
from control.AccountControl import AccountControl
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```
from utils.css_selectors import Selectors
import asyncio


class LoginControl:
    def __init__(self, browser_entity):
        self.browser_entity = browser_entity  # Manages browser state
        self.account_control  = AccountControl()  # Manages account data



    async def login(self, site: str):
        # Step 1: Fetch account credentials from the entity object
        account_info = self.account_control.fetch_account_by_website(site)
        if not account_info:
            return f"No account found for {site}"

        # account_info is a tuple (username, password), so access it by index
        username, password = account_info[0], account_info[1]
        print(f"Username: {username}, Password: {password}")

        # Step 3: Get the URL from the CSS selectors
        url = Selectors.get_selectors_for_url(site).get('url')
        print(url)
        if not url:
            return f"URL for {site} not found."

        # Step 4: Navigate to the URL and perform login (handled by the entity object)
        result = await self.browser_entity.perform_login(url, username, password)
```

```
        return result


--- MonitorAvailabilityControl.py ---
import asyncio
from entity.AvailabilityEntity import AvailabilityEntity
from datetime import datetime


class MonitorAvailabilityControl:
    def __init__(self, browser_entity):
        self.availability_entity = AvailabilityEntity(browser_entity)  # Reuse check control logic
        self.is_monitoring = False  # Store the running task
        self.results = []


    async def start_monitoring_availability(self, ctx, url: str, date_str=None, frequency=15):
        """Start monitoring availability at the given frequency."""
        if self.is_monitoring:
            return "Already monitoring prices."
        self.is_monitoring = True  # Set monitoring state to true
        try:
            while self.is_monitoring:
                availability_info  = await self.availability_entity.check_availability(ctx, url, date_str)


                # Prepare the result message
                result = f"Checked availability: {availability_info}"


                # Append the result to the results list
                self.results.append(result)
```

```
            # Create a DTO (Data Transfer Object) to organize the data for export
            data_dto = {
                "command": "start_monitoring_availability",  # Command executed
                "url": url,  # URL of the availability being monitored
                "result": result,  # Result of the availability check
                "entered_date": datetime.now().strftime('%Y-%m-%d'),  # Current date
                "entered_time": datetime.now().strftime('%H:%M:%S')  # Current time
            }

            # Pass the DTO to AvailabilityEntity to handle export to Excel and HTML
            self.availability_entity.export_data(data_dto)

            # Sleep for the specified frequency before the next check
            await asyncio.sleep(frequency)

        except Exception as e:
            self.results.append(f"Failed to monitor availability: {str(e)}")
            return f"Error: {str(e)}"
        return self.results

    def stop_monitoring(self):
        """Stop the availability monitoring loop."""
        self.is_monitoring = False  # Set monitoring state to false
        # Return all the results collected during the monitoring period
        return self.results if self.results else ["No data collected."]
```

```
--- MonitorPriceControl.py ---
import asyncio
from datetime import datetime
from entity.PriceEntity import PriceEntity
from utils.css_selectors import Selectors

class MonitorPriceControl:
    """MonitorPriceControl handles the business logic of monitoring the price over time
    and instructs PriceEntity to fetch prices and export data."""

    def __init__(self, browser_entity):
        self.price_entity = PriceEntity(browser_entity)  # Initialize PriceEntity for data fetching and
export
        self.is_monitoring = False  # Control flag for monitoring state
        self.results = []  # List to store results during monitoring

    async def start_monitoring_price(self, ctx, url: str = None, frequency=20):
        """Start monitoring the price at a given interval and provide updates to the user via Discord.

        ctx: Context from Discord command.
        url: URL of the product page to monitor.
        frequency: Time interval (in seconds) between price checks.
        """
        if self.is_monitoring:
            return "Already monitoring prices."
        self.is_monitoring = True  # Set monitoring state to true
        previous_price = None  # Track the last price fetched
```

```
        try:
            while self.is_monitoring:
                # Fetch the current price from PriceEntity
                if not url:
                    selectors = Selectors.get_selectors_for_url("bestbuy")
                    url = selectors.get('priceUrl')  # Get the price URL
                    if not url:
                        return "No URL provided, and default URL for BestBuy could not be found."
                    print("URL not provided, default URL for BestBuy is: " + url)
                current_price = self.price_entity.get_price_from_page(url)

                # Determine price changes and prepare the result
                result = ""
                if current_price:
                    if previous_price is None:
                        result = f"Starting price monitoring. Current price: {current_price}"
                    elif current_price > previous_price:
                        result = f"Price went up! Current price: {current_price} (Previous: {previous_price})"
                    elif current_price < previous_price:
                                result = f"Price went down! Current price: {current_price} (Previous:
{previous_price})"
                    else:
                        result = f"Price remains the same: {current_price}"
                    previous_price = current_price
                else:
                    result = "Failed to retrieve the price."
```

```
            # Add the result to the results list
            self.results.append(result)

            # Create a DTO (Data Transfer Object) to organize the data for export
            data_dto = {
                "command": "start_monitoring_price",  # Command executed
                "url": url,  # URL of the product being monitored
                "result": result,  # Result of the price check
                "entered_date": datetime.now().strftime('%Y-%m-%d'),  # Current date
                "entered_time": datetime.now().strftime('%H:%M:%S')  # Current time
            }

            # Pass the DTO to PriceEntity to handle export to Excel and HTML
            self.price_entity.export_data(data_dto)

            await asyncio.sleep(frequency)  # Wait for the next check based on frequency

        except Exception as e:
            self.results.append(f"Failed to monitor price: {str(e)}")

    def stop_monitoring(self):
        """Stop the price monitoring loop."""
        self.is_monitoring = False  # Set monitoring state to false
        # Return the full list of results gathered during monitoring
        return self.results if self.results else ["No data collected."]
```

--- NavigationControl.py ---

```
from entity.BrowserEntity import BrowserEntity
from utils.css_selectors import Selectors

class NavigationControl:
    def __init__(self, browser_entity):
        self.browser_entity = browser_entity

    def navigate_to_website(self, url: str = None):
        if not url:
            selectors = Selectors.get_selectors_for_url("google")
            url = selectors.get('url')
            if not url:
                return "No URL provided, and default URL for google could not be found."
            print("URL not provided, default URL for Google is: " + url)
        return self.browser_entity.navigate_to_url(url)
```

--- StopControl.py ---

```
import discord

class StopControl:
    async def stop_bot(self, ctx, bot):
        """Stop the bot gracefully."""
        await ctx.send("The bot is shutting down...")
        await bot.close()  # Close the bot
```

--- __init__.py ---

#empty init file