

--- AvailabilityEntity.py ---

```
import asyncio
```

```
from utils.exportUtils import ExportUtils
```

```
from entity.BrowserEntity import BrowserEntity
```

```
from utils.css_selectors import Selectors
```

```
from selenium.webdriver.common.by import By
```

```
from selenium.webdriver.support.ui import WebDriverWait
```

```
from selenium.webdriver.support import expected_conditions as EC
```

```
class AvailabilityEntity:
```

```
    def __init__(self):
```

```
        self.browser_entity = BrowserEntity()
```

```
    async def check_availability(self, url: str, date_str=None, timeout=5):
```

```
        try:
```

```
            # Use BrowserEntity to navigate to the URL
```

```
            self.browser_entity.navigate_to_website(url)
```

```
            # Get selectors for the given URL
```

```
            selectors = Selectors.get_selectors_for_url(url)
```

```
            # Perform date selection (optional)
```

```
            if date_str:
```

```
                try:
```

```
                    date_field = self.browser_entity.driver.find_element(By.CSS_SELECTOR,
```

```
selectors['date_field'])
```

```

        date_field.click()

        await asyncio.sleep(1)

        date_button = self.browser_entity.driver.find_element(By.CSS_SELECTOR,
f"{selectors['select_date']} button[aria-label*='{date_str}']")

        date_button.click()

    except Exception as e:

        return f"Failed to select the date: {str(e)}"

    await asyncio.sleep(2) # Wait for updates to load

# Initialize flags for select_time and no_availability elements
select_time_seen = False
no_availability_seen = False
try:

    # Check if 'select_time' is available within the given timeout
    WebDriverWait(self.browser_entity.driver, timeout).until(

        EC.presence_of_element_located((By.CSS_SELECTOR, selectors['select_time']))

    )

    select_time_seen = True # If found, set the flag to True
except:

    select_time_seen = False # If not found within timeout
try:

    # Check if 'no_availability' is available within the given timeout
    WebDriverWait(self.browser_entity.driver, timeout).until(

        lambda driver: len(driver.find_elements(By.CSS_SELECTOR,
selectors['show_next_available_button'])) > 0

    )

```

```

        no_availability_seen = True # If found, set the flag to True

    except:

        no_availability_seen = False # If not found within timeout

    # Logic to determine availability

    if select_time_seen:

        return f"Selected or default date {date_str if date_str else 'current date'} is available for
booking."

    elif no_availability_seen:

        return "No availability for the selected date."

    else:

        return "Unable to determine availability. Please try again."

except Exception as e:

    return f"Failed to check availability: {str(e)}"

def export_data(self, dto):

    """Export price data to both Excel and HTML using ExportUtils.

    dto: This is a Data Transfer Object (DTO) that contains the command, URL, result, date, and
time.

    """

    # Extract the data from the DTO

    command = dto.get('command')

    url = dto.get('url')

    result = dto.get('result')

```

```
entered_date = dto.get('entered_date') # Optional, could be None
```

```
entered_time = dto.get('entered_time') # Optional, could be None
```

```
# Call the Excel export method from ExportUtils
```

```
excelResult = ExportUtils.log_to_excel(
```

```
    command=command,
```

```
    url=url,
```

```
    result=result,
```

```
    entered_date=entered_date, # Pass the optional entered_date
```

```
    entered_time=entered_time # Pass the optional entered_time
```

```
)
```

```
print(excelResult)
```

```
# Call the HTML export method from ExportUtils
```

```
htmlResult = ExportUtils.export_to_html(
```

```
    command=command,
```

```
    url=url,
```

```
    result=result,
```

```
    entered_date=entered_date, # Pass the optional entered_date
```

```
    entered_time=entered_time # Pass the optional entered_time
```

```
)
```

```
print(htmlResult)
```

```
--- BrowserEntity.py ---
```

```
import asyncio
```

```
from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from utils.css_selectors import Selectors
```

```
class BrowserEntity:
```

```
    _instance = None
```

```
    def __new__(cls, *args, **kwargs):
```

```
        if not cls._instance:
```

```
            cls._instance = super(BrowserEntity, cls).__new__(cls, *args, **kwargs)
```

```
        return cls._instance
```

```
    def __init__(self):
```

```
        self.driver = None
```

```
        self.browser_open = False
```

```
    def set_browser_open(self, is_open: bool):
```

```
        self.browser_open = is_open
```

```
    def is_browser_open(self) -> bool:
```

```
return self.browser_open
```

```
def launch_browser(self):
```

```
    if not self.browser_open:
```

```
        options = webdriver.ChromeOptions()
```

```
        options.add_argument("--remote-debugging-port=9222")
```

```
        options.add_experimental_option("excludeSwitches", ["enable-automation"])
```

```
        options.add_experimental_option('useAutomationExtension', False)
```

```
        options.add_argument("--start-maximized")
```

```
        options.add_argument("--disable-notifications")
```

```
        options.add_argument("--disable-popup-blocking")
```

```
        options.add_argument("--disable-infobars")
```

```
        options.add_argument("--disable-extensions")
```

```
        options.add_argument("--disable-webgl")
```

```
        options.add_argument("--disable-webrtc")
```

```
        options.add_argument("--disable-rtc-smoothing")
```

```
        self.driver = webdriver.Chrome(service=Service(), options=options)
```

```
        self.browser_open = True
```

```
        result = "Browser launched."
```

```
        print(result)
```

```
        return result
```

```
    else:
```

```
        result = "Browser is already running."
```

```
        print(result)
```

```
        return result
```

```
def close_browser(self):  
    if self.browser_open and self.driver:  
        self.driver.quit()  
        self.browser_open = False  
        result = "Browser closed."  
        print(result)  
        return result  
    else:  
        result = "No browser is currently open."  
        print(result)  
        return result
```

```
def navigate_to_website(self, url):  
    # Ensure the browser is launched before navigating  
    if not self.is_browser_open():  
        self.launch_browser()  
  
    # Navigate to the URL if browser is open  
    if self.driver:  
        self.driver.get(url)  
        result = f"Navigated to {url}"  
        print(result)  
        return result  
    else:
```

```
result = "Failed to open browser."
```

```
print(result)
```

```
return result
```

```
async def login(self, url, username, password):
```

```
    # Navigate to the website
```

```
    self.navigate_to_website(url)
```

```
    await asyncio.sleep(3)
```

```
    # Enter the username
```

```
        email_field = self.driver.find_element(By.CSS_SELECTOR,
```

```
Selectors.get_selectors_for_url(url)['email_field'])
```

```
        email_field.send_keys(username)
```

```
        await asyncio.sleep(3)
```

```
    # Enter the password
```

```
        password_field = self.driver.find_element(By.CSS_SELECTOR,
```

```
Selectors.get_selectors_for_url(url)['password_field'])
```

```
        password_field.send_keys(password)
```

```
        await asyncio.sleep(3)
```

```
    # Click the login button
```

```
        sign_in_button = self.driver.find_element(By.CSS_SELECTOR,
```

```
Selectors.get_selectors_for_url(url)['SignIn_button'])
```

```
        sign_in_button.click()
```

```
        await asyncio.sleep(5)
```



```

# Wait for the homepage to load

try:

WebDriverWait(self.driver,

30).until(EC.presence_of_element_located((By.CSS_SELECTOR,

Selectors.get_selectors_for_url(url)['homePage'])))

    result = f"Logged in to {url} successfully with username: {username}"

    print(result)

    return result

except Exception as e:

    result = f"Failed to log in: {str(e)}"

    print(result)

    return result

```

--- PriceEntity.py ---

```

from selenium.webdriver.common.by import By

from entity.BrowserEntity import BrowserEntity

from utils.exportUtils import ExportUtils # Import ExportUtils for handling data export

from utils.css_selectors import Selectors # Import selectors to get CSS selectors for the browser

```

```

class PriceEntity:

```

```

    """PriceEntity is responsible for interacting with the system (browser) to fetch prices

    and handle the exporting of data to Excel and HTML."""

```

```

def __init__(self):

    self.browser_entity = BrowserEntity()


def get_price_from_page(self, url: str):

    # Navigate to the URL using BrowserEntity

    self.browser_entity.navigate_to_website(url)

    selectors = Selectors.get_selectors_for_url(url)

    try:

        # Find the price element on the page using the selector

        price_element = self.browser_entity.driver.find_element(By.CSS_SELECTOR,
selectors['price'])

        result = price_element.text

        return result

    except Exception as e:

        return f"Error fetching price: {str(e)}"


def export_data(self, dto):

    """Export price data to both Excel and HTML using ExportUtils.

    dto: This is a Data Transfer Object (DTO) that contains the command, URL, result, date, and
time.

    """

    # Extract the data from the DTO

    command = dto.get('command')

    url = dto.get('url')

    result = dto.get('result')

```

```
entered_date = dto.get('entered_date') # Optional, could be None
```

```
entered_time = dto.get('entered_time') # Optional, could be None
```

```
# Call the Excel export method from ExportUtils
```

```
excelResult = ExportUtils.log_to_excel(
```

```
    command=command,
```

```
    url=url,
```

```
    result=result,
```

```
    entered_date=entered_date, # Pass the optional entered_date
```

```
    entered_time=entered_time # Pass the optional entered_time
```

```
)
```

```
print(excelResult)
```

```
# Call the HTML export method from ExportUtils
```

```
htmlResult = ExportUtils.export_to_html(
```

```
    command=command,
```

```
    url=url,
```

```
    result=result,
```

```
    entered_date=entered_date, # Pass the optional entered_date
```

```
    entered_time=entered_time # Pass the optional entered_time
```

```
)
```

```
print(htmlResult)
```

```
--- __init__.py ---
```

```
#empty init file
```

