BAX 422 - FINAL PROJECT

Group 7

Mia Lai

Sadhanha Anand

Cindy Jeon

EXECUTIVE SUMMARY

Lux introduces "The Luximus (2024)" lineup, targeting a niche market with luxury home appliances. To overcome delivery challenges posed by extreme weather, Lux integrates real-time weather data into its delivery process. Utilizing OpenWeatherMap and sophisticated web-scraping techniques, data is analyzed and stored in MongoDB, enabling Lux to adapt delivery routes and schedules proactively.

This strategic integration significantly enhances customer satisfaction by ensuring timely and damage-free deliveries, irrespective of adverse weather conditions. Lux's innovative approach not only elevates the delivery experience for luxury appliances but also reinforces the brand's reputation for excellence and customer-centric operations in the luxury market.

PROJECT BACKGROUND

Business scenario

Lux - Delivering a Premium Customer Experience for Luxury Home Appliances New Product Lineup, "The Luximus (2024) "

Background

The Luximus line-up includes a luxury 98-inch wide flat TV, refrigerators, washing machines, steam closet, and oven. Lux's products target a niche market of design-conscious individuals who appreciate luxury and enjoy showcasing their lifestyle through online social networks (SNS). Due to their high cost and large size, these products require extremely special care during transportation and installation. Traditional delivery methods often lack the finesse and attention to detail needed to ensure a seamless and damage-free experience for Lux's premium clientele.

Challenges

1) Extreme Temperatures: During very hot or cold weather, delicate electronic components within the appliances could be damaged during transportation. Additionally, extreme heat can affect the performance of certain appliances (e.g., refrigerators) during delivery.

2) Precipitation: Rain, snow, or hail can pose a risk of water damage to the appliances during transport. It could also create hazardous driving conditions, increasing the risk of accidents and delays.

3) High Winds: Strong winds can make it difficult to safely transport and maneuver large appliances into and out of delivery trucks and homes.

4) Humidity: High humidity can lead to condensation on the appliances during delivery, potentially causing rust or other damage.

These weather-related challenges can:

1) Increase Damage Risk: Extreme temperatures, precipitation, and high winds can all increase the chance of damage to the appliances during transport and installation.

2) Lead to Delays: Adverse weather conditions like heavy rain, snow, or strong winds can slow down deliveries and cause scheduling disruptions.

3) Compromise Customer Satisfaction: Delays and damaged appliances can lead to frustration and dissatisfaction for Lux's premium clientele.

By integrating the weather data analysis system with the delivery process, Lux can:

1) Plan for Weather Events: Delivery routes can be planned to avoid areas experiencing extreme weather conditions. Deliveries can be rescheduled for more favorable weather windows when necessary.

2) Equip Delivery Teams: Teams can be equipped with weather-appropriate gear and packing materials to protect the appliances during transport.

3) Communicate with Customers: Lux can proactively communicate with customers about potential weather delays and take steps to minimize inconvenience.

DATA ARCHITECTURE

Data Sources

For Lux's project, our primary data sources include real-time and historical weather data APIs, such as those offered by national meteorological services, and global weather tracking services, here we used the web-scraped data from OpenWeatherMap ("https://openweathermap.org/") . These sources provide detailed information on temperatures, precipitation, wind speeds, and humidity levels—crucial for planning safe and timely deliveries.

Web-Scraping Routine(s)

To complement API-driven data collection, our web-scraping routines target online platforms that publish weather forecasts and warnings, traffic updates, and consumer reviews. These routines are crafted using Python, with BeautifulSoup for parsing HTML content and Selenium for automating interactions with web pages that require dynamic content loading. The scraping is scheduled to run at regular intervals, ensuring timely updates on potential weather hazards. Ethical scraping practices are strictly followed, with adherence to websites' terms of service and rate limiting to prevent server overload.

The specific steps as follow:

Step 1: Autonomous login and Configure Selenium WebDriver for dynamic content loading to save HTML pages

- **Creating and Verifying the OpenWeatherMap account**

1) Account Creation: Create an account on https://openweathermap.org to an external site.

2) Manual Login Verification: Before automating the login process, ensure we can manually log in to openweathermap.org with our new credentials. This confirms that our accounts are active and our credentials are correct.

- **Exploring the Login Mechanism**

1) Navigate to the login page of https://openweathermap.org to an external site.

2) Use the browser's developer tools to inspect the page, focusing on the `<form>` tag involved in the login process.

3) Document all `<input>` fields within the login form, paying special attention to their name attributes. These fields are crucial for submitting the login request programmatically.

- **Analyzing Network Traffic for Login Request**

1) With the network tab of the browser's developer tools open, log in to the site again.

2) Identify the network request made when we submit the login form (GET or POST).

3) Carefully examine the payload that was submitted to the server during login. Compare this payload to the `<form>` / `<input>` fields we previously analyzed.

- **Automating the Login Process**

1) Using Python and appropriate libraries like requests, simulate the login process.

2) Create a session object to maintain the login state across multiple requests.

3) Prepare a payload with the login credentials and other necessary form data identified from the login page and the network analysis.

4) Send a POST request to the login form's action URL to log in, using the session object.

- **Verifying Successful Login**

1) After attempting to log in, inspect the cookies saved in the session object to understand the information WhoScored.com stores on the computer.

2) Use the session object to access https://home.openweathermap.org/home to an external site.

3) Verify successful login by checking for the presence of your user information that is only available when logged in.

The results are as follows:

```python
# Access cookies from the session
cookies = session.cookies.get_dict()
print(cookies)
```

```
u50ua/OZNQfDqaWwoOUd9qss9s/Wke3KInmyVpZCkMJMae6J9KXx0AouIp1A1LW9VpvDRpbdPRlnPEyi9tvJow==
{'_members_session_1473164855': 'RFRzcXpqNGRvZktEOEgweVZoVGNFRTFvWVY0S1JZSUR1RHBwdUYwaXNFd2ZpVWlsSUlNdG9KY3NEQlZieH
o2aVdrTmdIWmcrWlcxeVp0bFhka2Z3RFBIUHB1YVlLR3FMbXVFTHB3eDJZcS9kZE1MSVdQbnI5dUp6SWV5azJDdFRocG5DQS9JQ1ZEOW0wYTl2Snk2Z
kEvcnZDVy9OcjNnQmN3VGR0cENURFBIY1BBRlQxUUV2Z203TmtIT3l2RU4rQ3lmTVRCcHJaaGxBR2VrQ1p0WmFMTXNaUXVPb09QcUEwUGYyNzF3ZkpR
L3NlYzNHU0Q1azd2cWhyR2tkR29vRi0tRnZDSWlOZmQ2WFlUYkwrY2ozNjQ0QT09--32cfc1f195a604a3ff53c6db9a1d6632a4ead2f8', 'signe
d_in': 'sadhanha'}
```

```python
elements = soup_2.find_all(string = 'sadhanha') # your username here
for element in elements:
    print(str(element.parent))
```

```
<a href="/">sadhanha</a>
```

- **Automate Browser to Access OpenWeatherMap**

1) Use Selenium in Python or Java to launch a browser and navigate to https://home.openweathermap.org/home.

- **Auto Search each city information and Save pages into HTML files**

1) Create a list (search_cities) that contains the names of the cities you want to search for on the website. we use the 10 cities as follows: New York City, New York, Los Angeles, California, Chicago, Illinois, Houston, Texas, Phoenix, Arizona, Philadelphia, Pennsylvania, San Antonio, Texas, San Diego, California Dallas, Texas, San Jose, California

2) Find the search input field on the webpage by using its CSS selector that identifies the placeholder text "Weather in your city". And Send the city name as keystrokes to the search input field followed by an Enter key press to submit the search.

3) Once the link is found and clickable, perform a click action to navigate to the corresponding city's weather page.

4) Construct a valid file path for each city, sanitizing the city name to be used as the file name. Open a new file at the defined path and write the current page's HTML content to it.

5) *To make it more precise, put the city's name, comma, 2-letter country code (ISO3166).*

   You will get all proper cities in your chosen country. The order is important - the first is

   the city name then comma then country. Example - London, GB or New York, US.

The results are as follows:

📄 Chicago_US.html	Yesterday, 11:03 PM	178 KB	HTML document
📄 Dallas_US.html	Yesterday, 11:06 PM	178 KB	HTML document
📄 Houston_US.html	Yesterday, 11:03 PM	177 KB	HTML document
📄 Los_Angeles_US.html	Yesterday, 11:02 PM	173 KB	HTML document
📄 New_York_US.html	Yesterday, 11:02 PM	182 KB	HTML document
📄 Philadelphia_US.html	Yesterday, 11:04 PM	176 KB	HTML document
📄 Phoenix_US.html	Yesterday, 11:04 PM	174 KB	HTML document
📄 San_Antonio_US.html	Yesterday, 11:05 PM	178 KB	HTML document
📄 San_Diego_US.html	Yesterday, 11:05 PM	174 KB	HTML document
📄 San_Francisco.html	Yesterday, 10:52 PM	178 KB	HTML document
📄 San_Jose_US.html	Yesterday, 11:06 PM	174 KB	HTML document

Step 2: Set up BeautifulSoup for HTML content parsing. Useful for extracting specific information from static web pages.

- **Extract Information**

1) For each HTML file, use BeautifulSoup to parse the file content. Extract and print the following details: Name of the City, Weather Forcast Time, Description, Curret Temp, Humidity, Visibility

The part of results are as follows:

```
Name of the City: San Francisco, US
Weather Forcast Time: Mar 17, 10:52pm
Description: Feels like 11°C. Few clouds. Gentle Breeze
Curret Temp: 12°C
Humidity: 84%
Visibility: 10.0km
----------------------------------------------------------
Name of the City: Los Angeles, US
Weather Forcast Time: Mar 17, 11:02pm
Description: Feels like 13°C. Clear sky. Light breeze
Curret Temp: 14°C
Humidity: 79%
Visibility: 10.0km
----------------------------------------------------------
Name of the City: Chicago, US
Weather Forcast Time: Mar 18, 01:03am
Description: Feels like −8°C. Broken clouds. Moderate breeze
Curret Temp: −2°C
Humidity: 53%
Visibility: 10.0km
----------------------------------------------------------
```

Step 3: Identify and register for APIs providing real-time and historical weather data, and store scraped data in a preliminary format for further processing

- **Connect to MongoDB**

1) Connects to your local MongoDB instance.

2) If 'localhost' can't work. We can use connecting to MongoDB cloud Atlas

- **Database Programming**

1) Creates a database named "open_weather"

2) Creates a collection called "cities" within the "open_weather" database.

- **Extract Data From JSON and Store into MongoDB**

1) Lists latitude and longitude contain the latitude and longitude values of different locations for the listed cities.

2) For each pair of coordinates, an HTTP GET request is sent to the OpenWeatherMap API's "weather" endpoint using the requests.get() method. The URL is constructed with latitude, longitude, and the API key to retrieve weather data for each location.

3) The response.json() method is called to parse the JSON data returned by the API into a Python dictionary (data).

4) Print a line to visually separate each city's data. The weather data for the current city is extracted from the data dictionary and printed, including details like the city name, temperature, "feels like" temperature, minimum and maximum temperatures, pressure, and humidity.

5) The db_data is then inserted into a MongoDB collection with collection.insert_one(db_data). This implies that there's a collection object defined elsewhere in the script, which represents a MongoDB collection where the data is being stored.

The results are as follows:
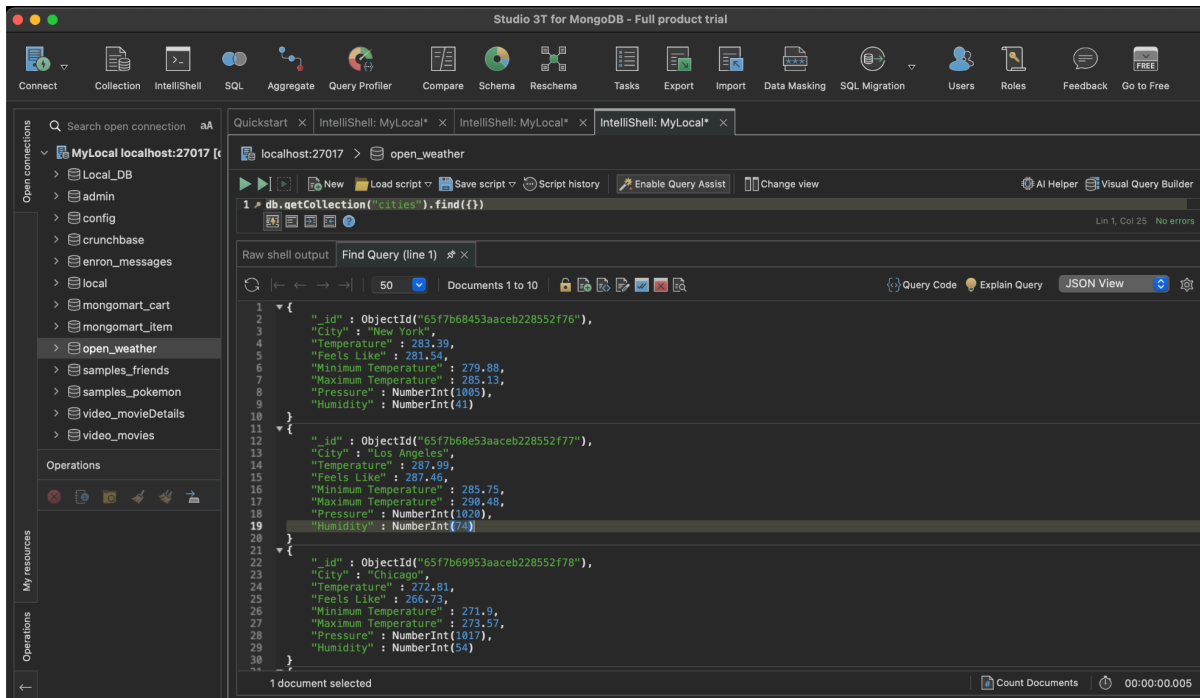
Database details:
```
{'_id': ObjectId('65f7dea5552e0ed0cd16d484'), 'City': 'New York', 'Temperature': 280.99, 'Feels Like': 278.44, 'Min
imum Temperature': 279.09, 'Maximum Temperature': 282.24, 'Pressure': 1007, 'Humidity': 50}
{'_id': ObjectId('65f7deb0552e0ed0cd16d485'), 'City': 'Los Angeles', 'Temperature': 286.87, 'Feels Like': 286.31, '
Minimum Temperature': 284.98, 'Maximum Temperature': 288.97, 'Pressure': 1018, 'Humidity': 77}
{'_id': ObjectId('65f7deba552e0ed0cd16d486'), 'City': 'Chicago', 'Temperature': 271.47, 'Feels Like': 265.48, 'Mini
mum Temperature': 270.24, 'Maximum Temperature': 272.75, 'Pressure': 1017, 'Humidity': 54}
{'_id': ObjectId('65f7dec5552e0ed0cd16d487'), 'City': 'Houston', 'Temperature': 290.54, 'Feels Like': 290.63, 'Mini
mum Temperature': 289.31, 'Maximum Temperature': 291.46, 'Pressure': 1015, 'Humidity': 88}
{'_id': ObjectId('65f7decf552e0ed0cd16d488'), 'City': 'Phoenix', 'Temperature': 286.57, 'Feels Like': 285.53, 'Mini
mum Temperature': 284.89, 'Maximum Temperature': 287.89, 'Pressure': 1015, 'Humidity': 60}
{'_id': ObjectId('65f7deda552e0ed0cd16d489'), 'City': 'Philadelphia', 'Temperature': 281.2, 'Feels Like': 277.68, '
Minimum Temperature': 279.15, 'Maximum Temperature': 282.29, 'Pressure': 1006, 'Humidity': 54}
{'_id': ObjectId('65f7dee4552e0ed0cd16d48a'), 'City': 'San Antonio', 'Temperature': 287.86, 'Feels Like': 287.58, '
Minimum Temperature': 286.54, 'Maximum Temperature': 289.01, 'Pressure': 1016, 'Humidity': 84}
{'_id': ObjectId('65f7deef552e0ed0cd16d48b'), 'City': 'San Diego', 'Temperature': 286.03, 'Feels Like': 285.65, 'Mi
nimum Temperature': 284.02, 'Maximum Temperature': 287.28, 'Pressure': 1018, 'Humidity': 87}
{'_id': ObjectId('65f7def9552e0ed0cd16d48c'), 'City': 'Dallas', 'Temperature': 286.14, 'Feels Like': 284.75, 'Minim
um Temperature': 284.51, 'Maximum Temperature': 287.29, 'Pressure': 1021, 'Humidity': 48}
{'_id': ObjectId('65f7df04552e0ed0cd16d48d'), 'City': 'San Jose', 'Temperature': 284.12, 'Feels Like': 283.2, 'Mini
mum Temperature': 281.35, 'Maximum Temperature': 286.05, 'Pressure': 1019, 'Humidity': 74}
```

DATA MODELLING AND DESIGN

Purpose of Dataset: Stores real-time and forecasted weather data by region, including parameters like temperature, precipitation, wind speed, and humidity. It underpins decision-making for delivery scheduling and routing.

Design Choice:

➢ Facilitates a granular analysis of weather conditions to assess risks to appliance deliveries. By associating weather data with specific delivery schedules, Lux can proactively manage or mitigate adverse weather impacts.

➢ By linking weather data with specific deliveries, Lux can proactively assess risks and take preventive measures. This might involve providing weatherproof packaging, rescheduling deliveries, or communicating potential delays to customers.

BUSINESS-RELEVANT QUESTIONS

Risk assessment:

● Will extreme temperatures, precipitation, or high winds pose a threat to deliveries in specific regions?

● How can delivery routes be adjusted to avoid areas with hazardous weather conditions?

Delivery planning:

● What is the optimal time window for scheduling deliveries based on weather forecasts?

● Should deliveries be rescheduled due to unexpected weather events to minimize delays and damage risks?

Resource allocation:

● What type of protective gear and packing materials are needed for deliveries in different weather conditions?

DATABASE IMPLEMENTATION

The choice to use MongoDB, a NoSQL database, for storing and processing this weather data offers several advantages over traditional relational databases. MongoDB's schema-less nature allows for flexibility in the types of data stored, which is particularly useful for the varied and dynamic nature of weather data. It supports rapid development and iteration, as the data model can evolve with changing business needs without requiring costly database refactoring.

Additionally, MongoDB's ability to handle large volumes of data and its powerful querying capabilities make it well-suited for real-time analytics, which is crucial for Lux's operational model.

BUSINESS VALUE

It enhances the customer experience by ensuring timely, safe, and damage-free deliveries, even under adverse weather conditions. This, in turn, improves customer loyalty and brand reputation, which are critical for success in the luxury appliance market. The efficiency gains from optimized delivery routes and schedules also reduce operational costs and improve profitability.

CONCLUSION

Lux's initiative addresses the significant challenges of delivering luxury appliances amid extreme weather by integrating real-time weather data, traffic reports, and customer feedback into a proactive delivery system. This data-driven strategy, supported by advanced analytics and MongoDB's flexible data storage, enables Lux to enhance customer satisfaction through timely and safe deliveries, reducing operational costs and boosting brand reputation. Ultimately, this approach sets a new standard in luxury appliance delivery, showcasing the importance of innovation and strategic planning in improving business operations and customer experiences amidst weather-related challenges.

APPENDIX:

Data Source:

https://openweathermap.org/

Code & Scraped Files:

https://drive.google.com/drive/folders/1YHOe_1W84UUqxgXbT8z5jOkmeI8VZou1?usp=sharing