The Chatbot Project

Personal Financial Advisor

Purpose:

The purpose of the chatbot is to act as a personal financial advisor that helps users manage their budget, track expenses, and make informed purchasing decisions. It offers features to log spending, analyse financial habits, and provide visualizations of spending patterns. Additionally, the chatbot helps users maintain a Wishlist, manage subscriptions, and receive saving advice based on their spending history, the chatbot assist users to stay organized, prioritize personal expenses, and achieve their financial goals efficiently.

Implementation and Description:

To implement this chatbot, I used Flask, a Python framework that provides a powerful backend environment. The libraries Spacy and TextBlob were integrated to improve the chatbot's natural language processing capabilities. TextBlob was specifically used to correct the grammar of user input before tokenizing and passing it to specific functions. Spacy was used for tokenization, analysing token parts of speech, dependencies, and identifying the root of the token and the root head to ensure that user inputs align with the chatbot's predefined rules. Basic user greeting such as "Good morning," "Hi," or "Good afternoon" are processed on the frontend using JavaScript functions, so it doesn't get send to backend.

After the frontend checks the user input and sends it to the backend through the chat function, the input message is checked using the TextBlob library to ensure there are no trivial grammar mistakes. After that, it is processed through the SpaCy library to identify the sequence of tokens as a document (doc) based on token part of speech, dependency, and the head part of speech. then it will be sent to another function for further processing.

In Table 1 in the appendices, each function is listed along with its trigger conditions. The chatbot has two main categories: spending and subscriptions. It decides based on the user's input. For example, if the user types "[I bought a new bag for $12]," the parse_message function, which falls under the spending category, is called. Based on the action token returned from parse_message, the handle_budget_modification function is invoked. Since the action lemma is "buy," the amount token (12) and the currency symbol ($) are sent to the currency conversion function. This function converts the amount to BHD, which is then received by handle_budget_modification. The converted amount is subtracted from the budget, logged through the log_spending function (including the item, amount, and date), and the updated budget is displayed to the user.

Spendings:

In the parse_message function, I categorized the user input into four components: amount, currency, description, and action:

1. **Action**: The token must be a verb, and its dependency should be the root (the main verb in the sentence).
2. **Description (item)**: The token's part of speech should be NOUN, and its dependency should not be a determiner (det) or a Noun Phrase Adverbial Modifier (npadvmod).
3. **Currency**: The function checks a currency directory. If the user writes the currency as "dollars," it is replaced with the symbol "$." If the user writes the currency symbol, the token's part of speech must be SYM, and its text must be "$" for the currency to be identified as "$."
4. **Price**: The function checks tokens with part of speech [NUM]. If the token dependency is [numod or pobj] and its head part of speech is [SYM, ADP, or VERB], it is processed. The token is checked for digit format, or it is converted to a digit using the word2number library.
5. **Quantity**: If there are two tokens with a part of speech (POS) tag of NUM in the document, the system will check, based on token dependency and head part of speech, whether they represent a quantity. If no token matches the condition, the default quantity is set to 1.
   The **final amount** is the product of quantity and price, Finally All categories are returned to app.py.

Subscriptions**:**

The subscription module works similarly to the parse_message function in the utils module for the spending but includes additional component: renewal_date, duration, and remaining_days.

1. **renewal date**: This is calculated based on the start date provided by the user. For example, if the user writes, "I subscribed to Netflix for 7 days," the start date is logged using the datetime library. For monthly subscriptions, 30 days are added to the start date to determine the renewal date.
2. **duration**: The default duration for subscriptions is 30 days (monthly subscriptions).
3. **remaining_days**: This is calculated by subtracting the current date from the renewal date. If the renewal date is 4 days or fewer away, the system highlights the remaining days in red to notify the user.

The diagram (on the left) illustrates how each module interacts with others and the libraries it uses. The dependency and entity visualization (on the right) demonstrates the structure of user input. Based on this structure, I developed rules for each if statement, including examples of possible user inputs that lead to the same meaning.

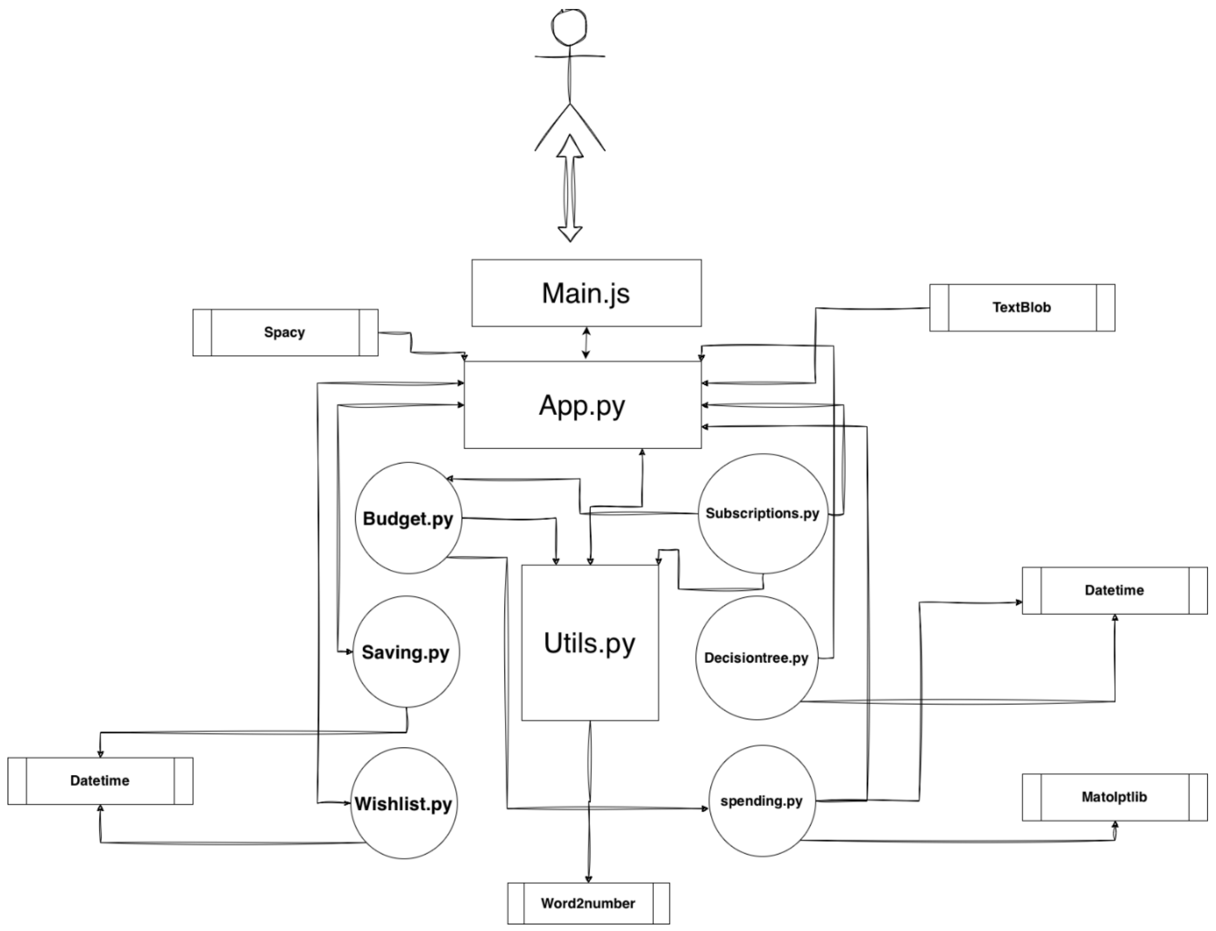


Figure 1 shows the interaction between modules, as well as their interaction with the user and the library used in each module.

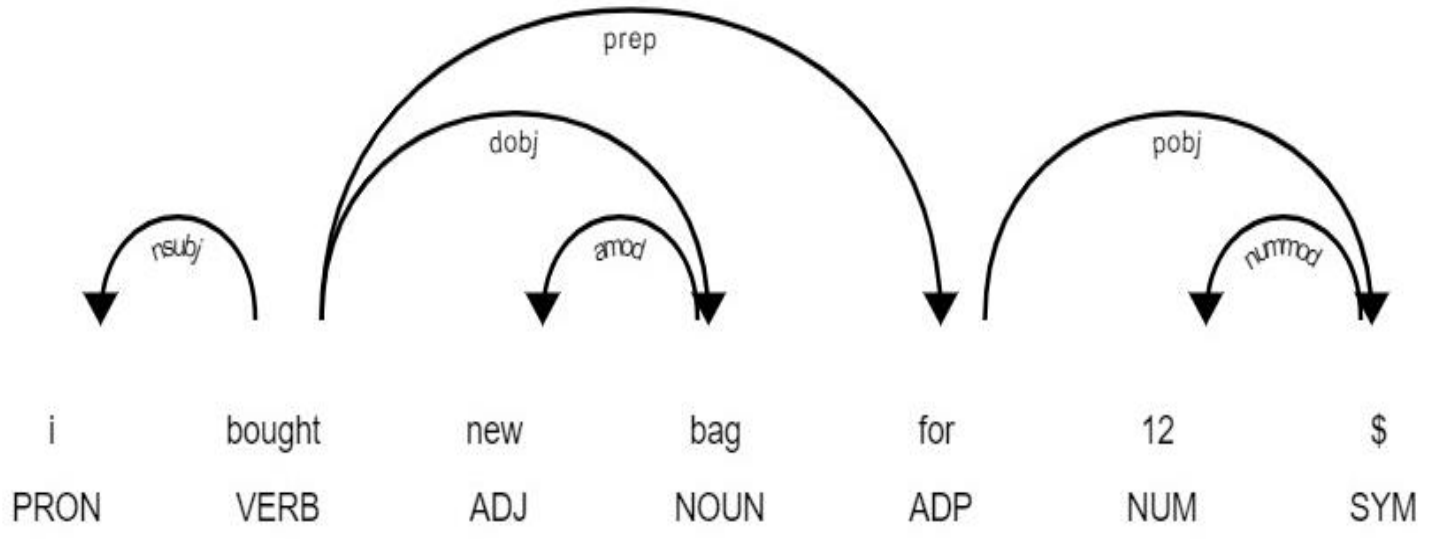
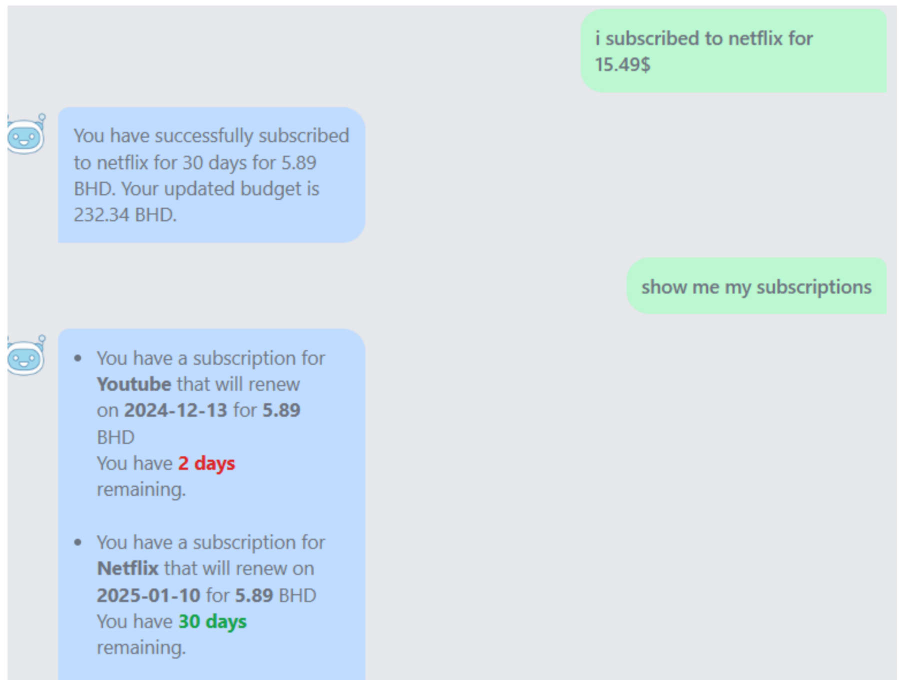


Figure 2 shows the dependency and entity visualization that was used to build the expert system set of rules based on it

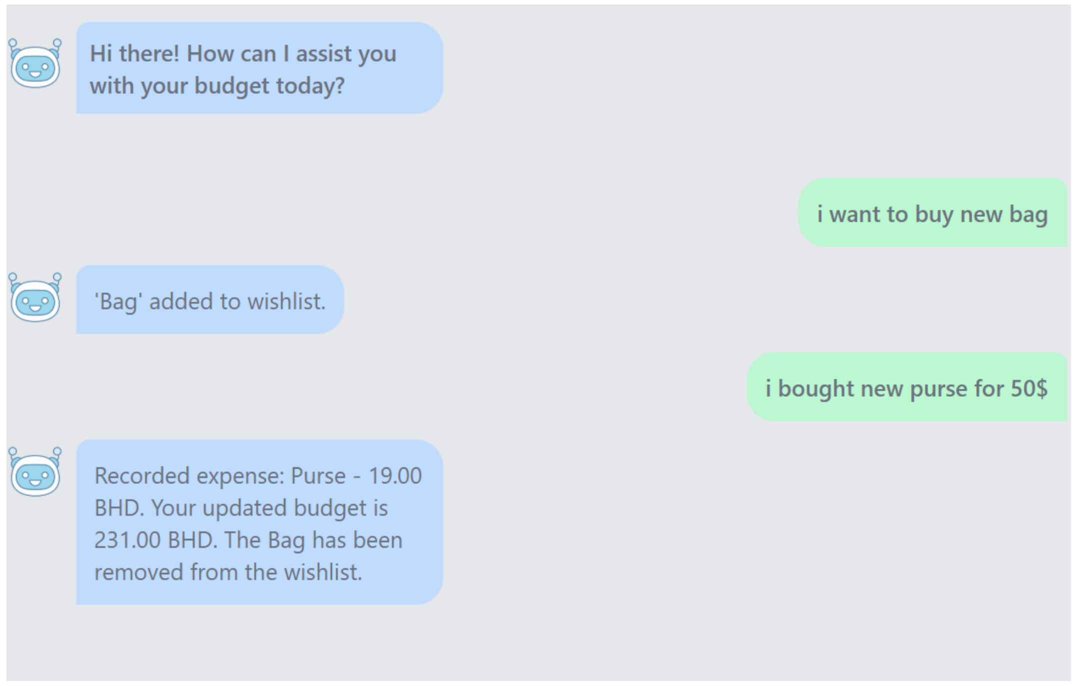

i subscribed to netflix for 15.49$

You have successfully subscribed to netflix for 30 days for 5.89 BHD. Your updated budget is 232.34 BHD.

show me my subscriptions

- You have a subscription for **Youtube** that will renew on **2024-12-13** for **5.89** BHD
  You have **2 days** remaining.
- You have a subscription for **Netflix** that will renew on **2025-01-10** for **5.89** BHD
  You have **30 days** remaining.

Figure 3 shows the subscription functionality


Hi there! How can I assist you with your budget today?

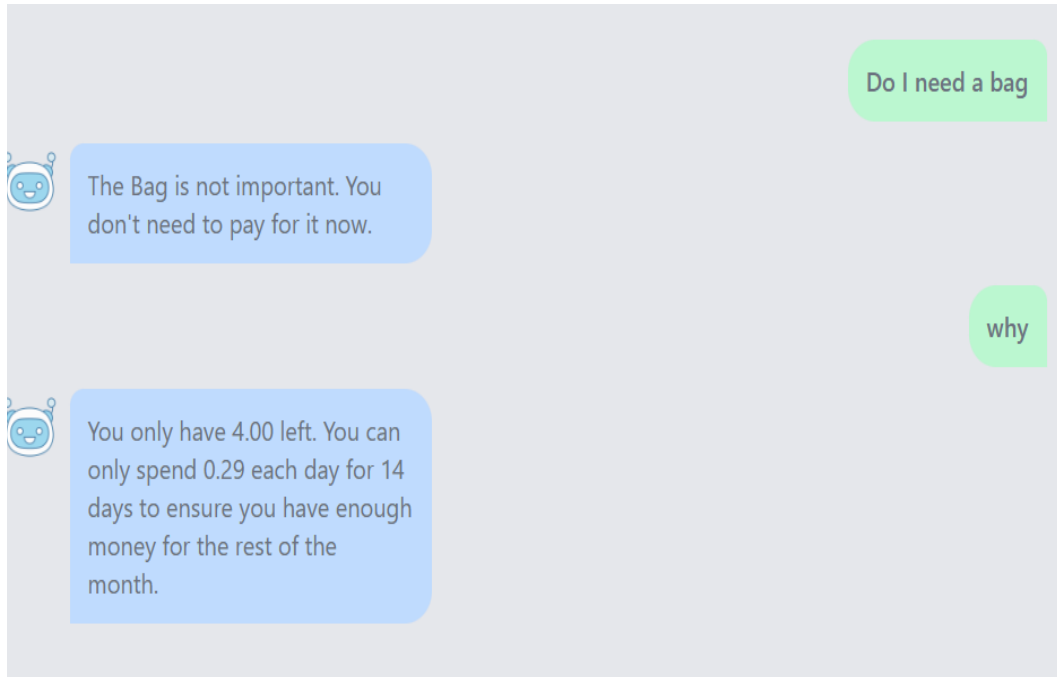
i want to buy new bag

'Bag' added to wishlist.

i bought new purse for 50$

Recorded expense: Purse - 19.00 BHD. Your updated budget is 231.00 BHD. The Bag has been removed from the wishlist.

Figure 4 shows the spending checking similarity in the Wishlist

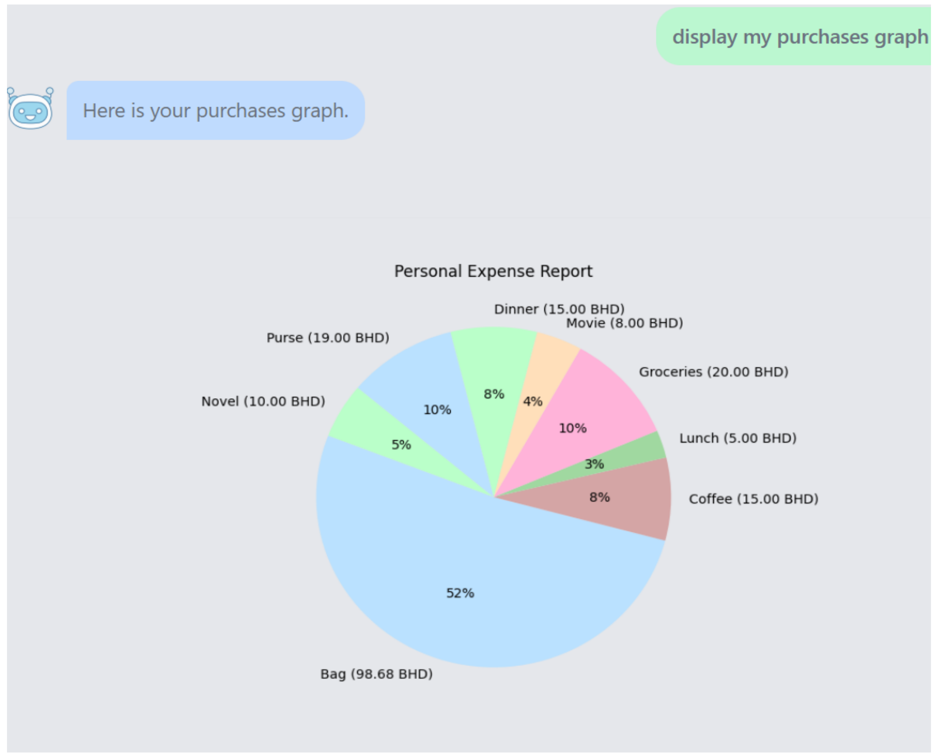

Do I need a bag

The Bag is not important. You don't need to pay for it now.

why

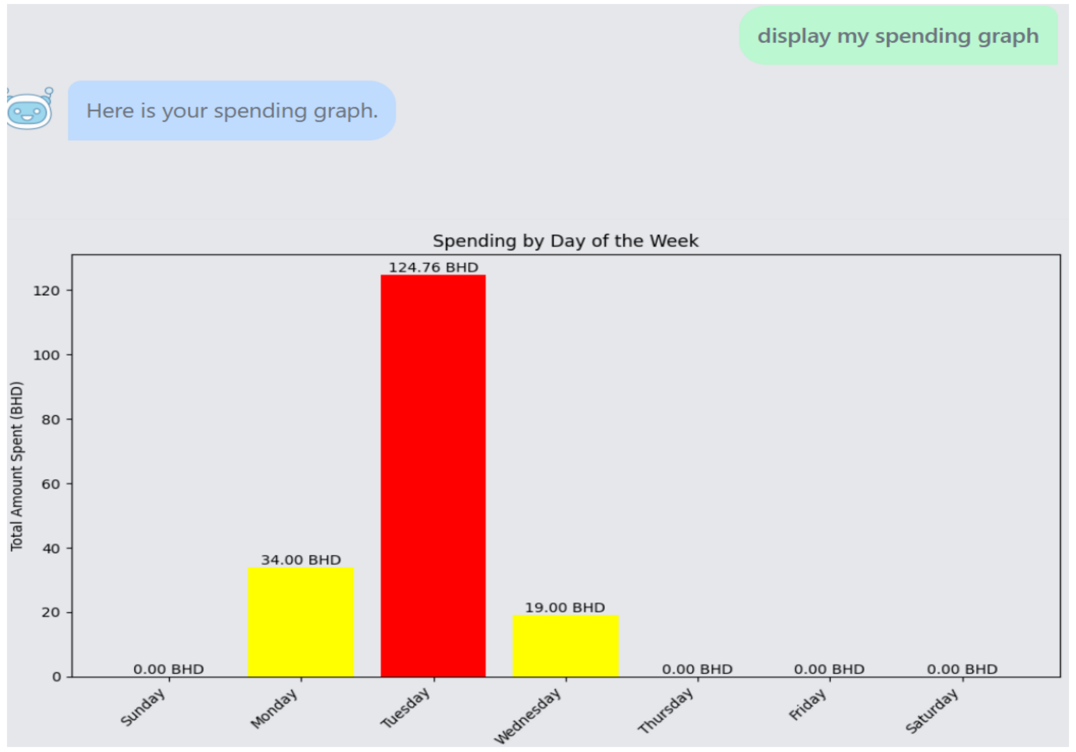
You only have 4.00 left. You can only spend 0.29 each day for 14 days to ensure you have enough money for the rest of the month.

Figure 5 display the chatbot ability to explaining and calculating daily spending


display my purchases graph

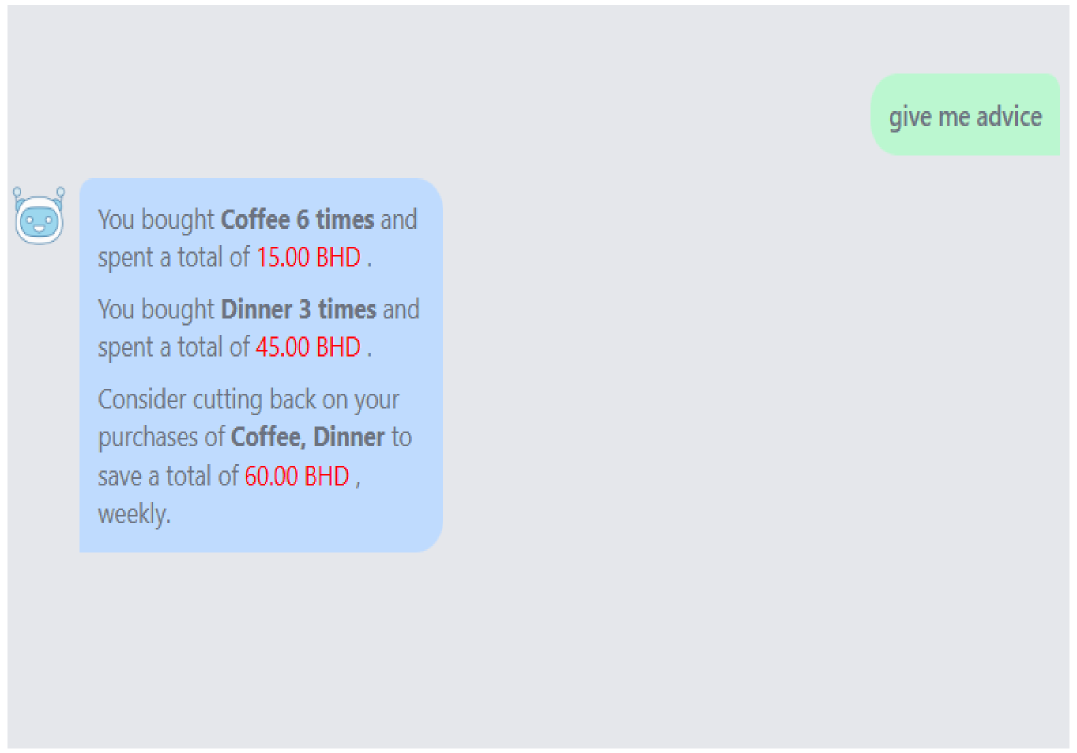
Here is your purchases graph.

Figure 6 shows how the chatbot generate pie chart for purchased item


display my spending graph

Here is your spending graph.

Figure 7 shows how the chatbot generate bar chart for spending habits

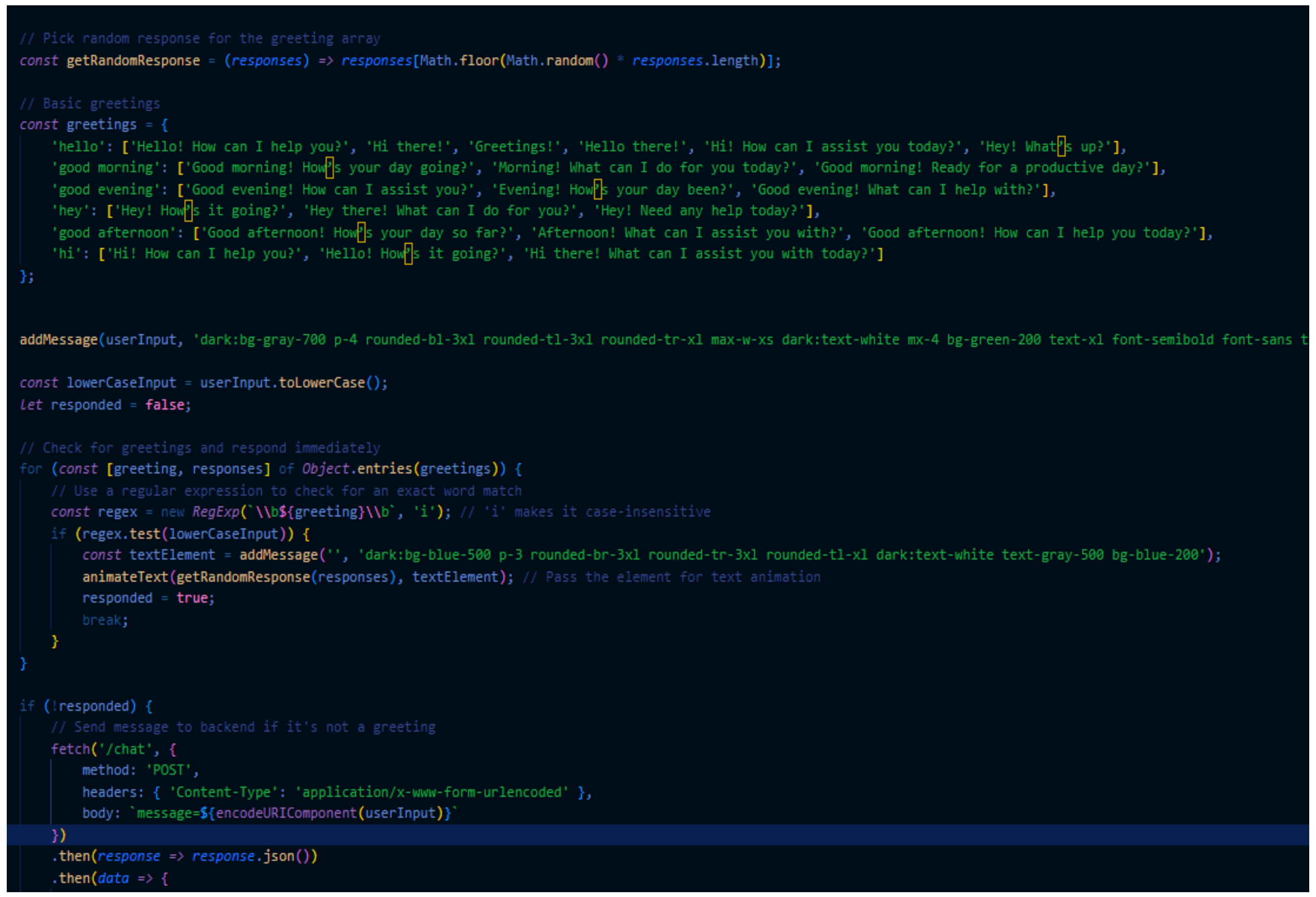

give me advice

You bought **Coffee 6 times** and spent a total of 15.00 BHD .

You bought **Dinner 3 times** and spent a total of 45.00 BHD .

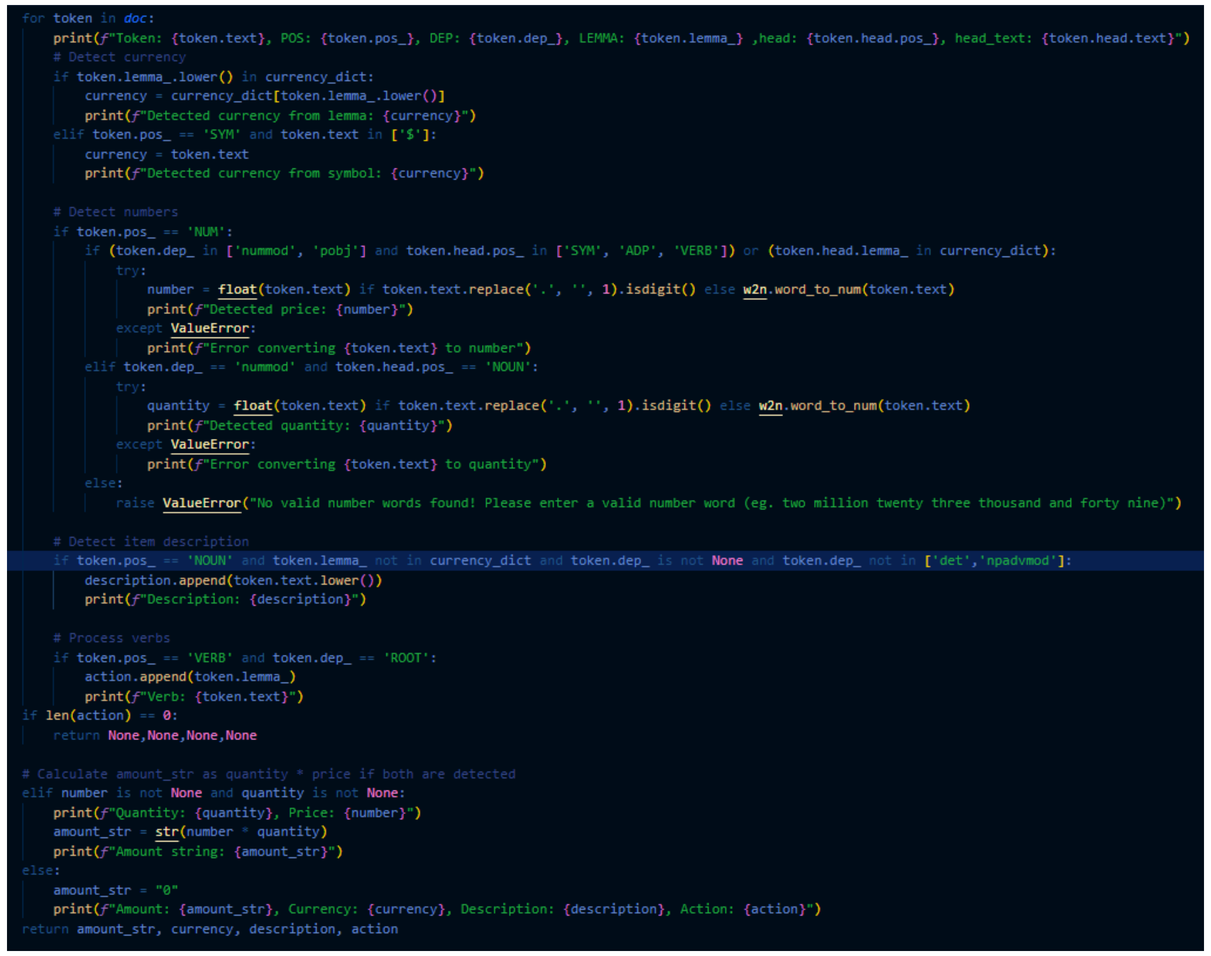
Consider cutting back on your purchases of **Coffee, Dinner** to save a total of 60.00 BHD , weekly.

Figure 8 display chatbot ability to give advice based on user spending habits in same week



```javascript
// Pick random response for the greeting array
const getRandomResponse = (responses) => responses[Math.floor(Math.random() * responses.length)];

// Basic greetings
const greetings = {
    'hello': ['Hello! How can I help you?', 'Hi there!', 'Greetings!', 'Hello there!', 'Hi! How can I assist you today?', 'Hey! What’s up?'],
    'good morning': ['Good morning! How’s your day going?', 'Morning! What can I do for you today?', 'Good morning! Ready for a productive day?'],
    'good evening': ['Good evening! How can I assist you?', 'Evening! How’s your day been?', 'Good evening! What can I help with?'],
    'hey': ['Hey! How’s it going?', 'Hey there! What can I do for you?', 'Hey! Need any help today?'],
    'good afternoon': ['Good afternoon! How’s your day so far?', 'Afternoon! What can I assist you with?', 'Good afternoon! How can I help you today?'],
    'hi': ['Hi! How can I help you?', 'Hello! How’s it going?', 'Hi there! What can I assist you with today?']
};


addMessage(userInput, 'dark:bg-gray-700 p-4 rounded-bl-3xl rounded-tl-3xl rounded-tr-xl max-w-xs dark:text-white mx-4 bg-green-200 text-xl font-semibold font-sans t

const lowerCaseInput = userInput.toLowerCase();
let responded = false;

// Check for greetings and respond immediately
for (const [greeting, responses] of Object.entries(greetings)) {
    // Use a regular expression to check for an exact word match
    const regex = new RegExp(`\\b${greeting}\\b`, 'i'); // 'i' makes it case-insensitive
    if (regex.test(lowerCaseInput)) {
        const textElement = addMessage('', 'dark:bg-blue-500 p-3 rounded-br-3xl rounded-tr-3xl rounded-tl-xl dark:text-white text-gray-500 bg-blue-200');
        animateText(getRandomResponse(responses), textElement); // Pass the element for text animation
        responded = true;
        break;
    }
}

if (!responded) {
    // Send message to backend if it's not a greeting
    fetch('/chat', {
        method: 'POST',
        headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
        body: `message=${encodeURIComponent(userInput)}`
    })
    .then(response => response.json())
    .then(data => {
```

Figure 9 display the JS code for simple greeting function



```python
for token in doc:
    print(f"Token: {token.text}, POS: {token.pos_}, DEP: {token.dep_}, LEMMA: {token.lemma_} ,head: {token.head.pos_}, head_text: {token.head.text}")
    # Detect currency
    if token.lemma_.lower() in currency_dict:
        currency = currency_dict[token.lemma_.lower()]
        print(f"Detected currency from lemma: {currency}")
    elif token.pos_ == 'SYM' and token.text in ['$']:
        currency = token.text
        print(f"Detected currency from symbol: {currency}")

    # Detect numbers
    if token.pos_ == 'NUM':
        if (token.dep_ in ['nummod', 'pobj'] and token.head.pos_ in ['SYM', 'ADP', 'VERB']) or (token.head.lemma_ in currency_dict):
            try:
                number = float(token.text) if token.text.replace('.', '', 1).isdigit() else w2n.word_to_num(token.text)
                print(f"Detected price: {number}")
            except ValueError:
                print(f"Error converting {token.text} to number")
        elif token.dep_ == 'nummod' and token.head.pos_ == 'NOUN':
            try:
                quantity = float(token.text) if token.text.replace('.', '', 1).isdigit() else w2n.word_to_num(token.text)
                print(f"Detected quantity: {quantity}")
            except ValueError:
                print(f"Error converting {token.text} to quantity")
        else:
            raise ValueError("No valid number words found! Please enter a valid number word (eg. two million twenty three thousand and forty nine)")

    # Detect item description
    if token.pos_ == 'NOUN' and token.lemma_ not in currency_dict and token.dep_ is not None and token.dep_ not in ['det','npadvmod']:
        description.append(token.text.lower())
        print(f"Description: {description}")

    # Process verbs
    if token.pos_ == 'VERB' and token.dep_ == 'ROOT':
        action.append(token.lemma_)
        print(f"Verb: {token.text}")
if len(action) == 0:
    return None,None,None,None

# Calculate amount_str as quantity * price if both are detected
elif number is not None and quantity is not None:
    print(f"Quantity: {quantity}, Price: {number}")
    amount_str = str(number * quantity)
    print(f"Amount string: {amount_str}")
else:
    amount_str = "0"
    print(f"Amount: {amount_str}, Currency: {currency}, Description: {description}, Action: {action}")
return amount_str, currency, description, action
```

Figure 10 display python code for parsing message function

Conclusion & Future work

The chatbot assists as a real-world tool for managing personal finances, offering features like expense tracking and subscription management. The NLP library implementation ensures accurate processing of user inputs. Future improvements will further develop usability and personalization.

In future work the chatbot could include implementing image processing for receipts to automatically detect amounts and items from the image and log them accordingly. Additionally, providing yearly and monthly expense summaries would enhance usability. Also Integrating a text generation model from Hugging Face could help user interactions by generating more human-friendly responses. Finally building model for predicting user next month expenses based on the previous months data.

Resourses

- Raj, S., 2019. Building chatbots with Python: Using natural language processing and machine learning. Apress.
- freeCodeCamp. (n.d.). Natural Language Processing with SpaCy (Python Full Course). Retrieved from https://www.freecodecamp.org/news/natural-language-processing-with-spacy-python-full-course
- Hugging Face. (n.d.). Hugging Face Hub: spaCy integration [Documentation]. Hugging Face. https://huggingface.co/docs/hub/spacy

Appendix

Function	Explanation:	When it gets called:	Libraries used
Decisiontree ()	It checks And calculate remaining days of the month and budget and decides wither the user should buy the specific item or not and decides wither it is important or not based on the facts given.	do I need to buy {item}	Datetime
Handle_budget_modifcation ()	Manage the budget, checks wither new item bought is in the Wishlist and similar to it get removed from the Wishlist base on threshold value	I bought {item} for {price}	-
Saving_advice ()	Calculate the total prices and the how many times it got purchases in the same week, and give advice if the item is bought more than 3 times at the same week	give me advice	Datetime, collections
Log_spending ()	Log the item and price and date	I bought {item} for {price}	Datetime
Show_spending_logs()	Display the spending logs in more user-friendly way	Show me my spending log	-
Clear_spending_logs ()	Clear the spending log	Clear my spending logs	-
Spending_graph ()	Calculate spending for each day and display it as bar chart	Display me my spending graph	Datetime, matplotlib
Purchase_graph ()	Calculate spending for each item and display it as pie chart	Display me my purchases graph	Datetime, matplotlib
display_subscription()	Display subscription list and calculate the remaining days for the renewal day based on the current day	Show me my subscription list	
Log_subscription ()	Log the service name start day renewal day, duration(30 day) , amount	Called by Handle_subscription()	-
Handle_subscription ()	Subtract from the budget based on the amount	Called after Subscription_message ()	-
Subscription_message ()	Check user input and parse it to extract the information for the subscription functions	I subscribed to {service_name } for {amount}	-
Convert_currency ()	Receive the amount and currency from the functions and convert it to BHD then return it	only if there is $ or dollars after the amount	-
Parse_message ()	Receive tokenised input and categorized the input based on the token {root , pos , dep }	After the user send their text	word2number
Wishlist ()	Log the item and date	I want to buy {item}	Datetime
Display_wishlist ()	Display Wishlist in more user friendly way	Display my Wishlist	-