Documentation for the BOT at qxbroker.com

This documentation provides an overview of the key Python and JavaScript scripts used for the BOT at qxbroker.com:

1. **`qxbroker.com.py`**: Manages all system processes related to the BOT, handling core functionalities and system interactions.
2. **`wsHook.js`**: Manages all websocket processes related to the BOT, ensuring proper functionality and system interactions.
3. **`bypass.js`**: Handles bypass mechanisms related to the BOT's operations.
4. **`strategies.py`**: Functions as the decision interceptor, determining actions based on the input provided by the system.

Modifications and Focus:

- **`qxbroker.com.py`**: No changes are necessary unless specifically required. It is responsible for managing system processes, so modifications should be approached with caution.
- **`wsHook.js`**: Similarly, this file should remain unchanged unless there is a specific need. It handles websocket-related functionalities and system interactions.
- **`bypass.js`**: No changes are needed unless absolutely necessary. It deals with bypass mechanisms and should be modified carefully.
- **`strategies.py`**: This is the primary focus. It serves as the decision-making component of the BOT.

Operation:

- **Transaction Handling**: When a transaction is initiated, `qxbroker.com.py` calls the `strategies.py` decision interceptor with the function `strategy(user_input, instruments_list, trade_data)`.
- **File Structure and Comments**: The `strategies.py` file is thoroughly commented, offering detailed explanations of each component of the input system. It includes a basic strategy using random actions to demonstrate fundamental decision-making logic.

Return Values:

- The decision interceptor (`strategies.py`) must return one of the following values:
  - `"call"`
  - `"put"`

This ensures that the BOT receives clear and actionable instructions for each transaction.

Feel free to explore and modify `strategies.py` to enhance the decision-making process, while remembering that `qxbroker.com.py` handles the overall system operations.

Example: Basic Strategy

```
import random

def strategy(user_input, instruments_list, trade_data):

    #user_input['trade_option'] = "put"#If your logic is specified as "put"
    #user_input['trade_option'] = "call"#If your logic is specified as "call"
    user_input['trade_option'] = "random"#If your logic is specified as "random"

    if user_input['trade_option'] == 'random':
        return random.choice(['call', 'put'])
    # Else return a specified one
    return user_input['trade_option']
```