Explanation of the Website Request Flow

1. OAuth Authorization Flow

The website follows an OAuth 2.0 authorization flow to ensure secure access to protected resources. Here's a step-by-step explanation of how this flow works on the website:

1. **Click on "Account"**:
   - When the user clicks on the "Account" button, the website redirects to the login page with a request to `https://login.olx.ro/`.
   - This request includes several parameters: `cc`, `client_id`, `code_challenge`, `code_challenge_method`, `redirect_uri`, `st`, and `state`.
2. **Login Page Redirection**:
   - The initial request to the login page looks like this:

     ```
     https://login.olx.ro/?cc=<cc>&client_id=<client_id>&code_challe
     nge=<code_challenge>&code_challenge_method=<code_challenge_meth
     od>&lang=ro&redirect_uri=https%3A%2F%2Fwww.olx.ro%2Fd%2Fcallbac
     k%2F&st=<st>&state=<state>
     ```

   - These parameters are critical as they are used in the subsequent login request.
3. **Login Request**:
   - To authenticate the user, the login request is sent to:

     ```
     authenticate_url = (f"https://login.olx.ro/api/initiate-auth?"
                         f"client_id={client_id}&"
                         f"redirect_uri={redirect_uri}&"
                         f"code_challenge={code_challenge}&"

        f"code_challenge_method={code_challenge_method}&"
                         f"state={state}%3D%3D&"
                         f"st={st}")
     ```

   - This request includes all the parameters obtained from the initial redirection.
4. **Blocked by CloudFront**:
   - When trying to perform the login request programmatically, you may encounter a CloudFront block. This is likely due to security measures that detect and block automated requests.
5. **Successful Login and Redirection**:
   - Upon successful authentication, the user is redirected to:

     ```
     https://www.olx.ro/d/callback/?code=<code>&state=<state>
     ```

   - The `code` parameter from this URL is then used to request access and refresh tokens.
6. **Token Request**:
   - The `code` obtained from the redirection is used to get the access and refresh tokens. These tokens are essential for making authorized requests to the GraphQL endpoint.

2. Explanation of the Selenium Approach

To handle the OAuth authorization and token retrieval programmatically, Selenium is used to automate the browser interactions:

1. **Automate Browser Interaction**:
    - Selenium automates the steps of clicking the "Account" button, filling in the login form, and submitting it.
    - This allows you to bypass the CloudFront block as the requests are made from a real browser session.
2. **Capture Tokens**:
    - After successful login, Selenium captures the redirected URL containing the `code` and `state` parameters.
    - These parameters are then used to request the access and refresh tokens.
3. **Make GraphQL Requests**:
    - Once the access token is obtained, it is included in the headers of subsequent GraphQL requests to authenticate them.

I have moved to using Selenium to obtain the access and refresh tokens. After obtaining these tokens, we can execute our Selenium requests similarly to how you would with the requests library in JavaScript (an example is commented in the code at the end). Since you want to access GraphQL requests and the returned data is in JSON format, we don't need to worry about time delays. Overall, using Selenium results in an initial delay of about 30-40 seconds, but after this, you can execute and access the GraphQL requests without delay. If you agree to using Selenium, I will complete the code to demonstrate how you can access the GraphQL requests, as I have successfully obtained the access and refresh tokens. This will allow you to make GraphQL requests successfully.

If you agree to proceed with the Selenium approach, the implementation can be completed to ensure reliable access to the required GraphQL endpoints.