Technical assessment.

API docs.

- **Swagger**: https://deep-index.moralis.io/api-docs-2.2/
- **Endpoint**: Wallets → GET */wallets/{address}/history*

Task 1: Test plan.

Assumptions.

- There's a proper and isolated testing environment to try out this endpoint.
- **X-API-Key** has the rights to create, read, update and delete information.
- **X-API-Key** has enough rate limit for testing purposes.
- The address given for testing purposes has diverse types of transactions.

Test cases.

1. **Validate that endpoint returns data with default parameters**.
   a. **Preconditions**.
      i. **chain**: *eth*.
      ii. **order**: *DESC*.
      iii. **address**: *0xcB1C1FdE09f811B294172696404e88E658659905*
   b. **Results for the response**.
      i. **HTTP status**: *200*.
      ii. **result** property should be an array of transaction objects for the given address.
      iii. **page_size** and **limit** properties should have the same value: *100*.
2. **Validate that endpoint returns data in ascending order, with eth chain**.
   a. **Preconditions**.
      i. **chain**: *eth*.
      ii. **from_block**: *14843501*
      iii. **order**: *ASC*.
      iv. **address**: *0xcB1C1FdE09f811B294172696404e88E658659905*
   b. **Results for the response**.
      i. **HTTP status**: *200*.
      ii. The data should be in ascending order, based on the **block_timestamp** and **block_number** properties.
3. **Validate that endpoint returns data with internal transactions, with eth chain**.

   a. **Preconditions**.
      i. **chain**: *eth*.
      ii. **include_internal_transactions**: *true*.
      iii. **order**: *DESC*.
      iv. **address**: *0xcB1C1FdE09f811B294172696404e88E658659905*
   b. **Results from the response**.
      i. **HTTP status**: *200*.
      ii. If the transactions retrieved have internal transactions, the **internal_transactions** property should include them inside an array of objects.
      iii. If not, the array should be empty.

4. **Validate that endpoint returns data given a range of dates, with eth chain**.
   a. **Preconditions**.
      i. **chain**: *eth*.
      ii. **order**: *DESC*.
      iii. **from_date**: *2025-05-01T00:00:00.000Z*
      iv. **to_date**: *2025-05-31T23:59:59.000Z*
   b. **Results from the response**.
      i. **HTTP status**: *200*.
      ii. If there are transactions in the given range of dates, these ones should be displayed in descending order.
5. **Validate that endpoint returns data when nft_metadata is set as true, with eth chain**.
   a. **Preconditions**.
      i. **chain**: *eth*.
      ii. **order**: *DESC*.
      iii. **nft_metadata**: *true*.
   b. **Results from the response**.
      i. **HTTP status**: *200*.
      ii. The **normalized_metadata** property should be present, if info, in the transactions, showing the correct data.
         1. If there's data in the **attributes** property, this should be displayed in an array of objects.
         2. If not, the array should be empty.
      iii. If there's no metadata in the transactions, the **normalized_metadata** property should not be present.
6. **Validate that the cursor retrieves new data, with eth chain**.
   a. **Preconditions**.
      i. **chain**: *eth*.
      ii. **order**: *DESC*.
      iii. **cursor**: *eyJhbGciOiJIUzI1NiJ9.eyJsYXN0S2V5Ijp7InRpbWVzdGFtcCI6MTcxMDYyMTQ3OTAwMCwiYmxvY2tOdW1iZXIiOjE5NDQ5ODcxLCJ0cmFuc2FjdGlvbkluZGV4IjoxMjB9LCJwYWdlIjowLCJyZXF1ZXN0Ijp7ImxpbWl0IjoxMDAsImNoYWluSWQiOiIweDEiLCJ3YWxsZXRBZGRyZXNzIjoiMHhjYjFjMWZkZTA5ZjgxMWIyOTQxNzI2OTY0MDRlODhlNjU4NjU5OTA1IiwiaXNBc2NlbmRpbmdPcmRlciI6ZmFsc2UsInRyYW5zYW*

        N0aW9uSW5kZXgiOjEyLCJmcm9tVGltZXN0YW1wIjoxNDM4MjY5OTczMDAwLCJ0b1RpbWVzdGFtcCI6MTcyNTc5OTYzMTAwMCwidG9CbG9jayI6MjA3MDU5ODN9fQ.tJa8ATltadSC9Yu74BCQvJMZG-AgfJfZF8S57frDO4M

   b. **Results from the response**.
      i. **HTTP status**: *200*.
      ii. New transactions should be shown by the server, and these ones shouldn't be repeated in other pages.
      iii. **page** property in the JSON response should be updated with the right value of the page.
         1. **There's an error here**, because by briefly checking the transactions I can see they're different, but the **page** property is always shown as *0*.

A summary of edge cases.

a) **Pagination boundary**.
   a. Validate that an error message appears when the limit is set to more than *100*, and server doesn't crash.
   b. The *400* HTTP status code should be thrown.

b) **Invalid range dates**.
   a. If **from_date** is set, for example, as *2025-05-31T23:59:59.000Z*, and **to_date** is set as *2025-05-01T00:00:00.000Z*, the server should throw an invalid error message. **from_date** value should not be bigger than **to_date** value.
      i. Right now, the server responds with a *200* HTTP status, and no transactions. Fix this with the proper HTTP status code (*400*) and show an error message.

c) **Case-insensitive address**.
   a. Validate that the server can interpret the **address** regardless of if everything is uppercase or lowercase or mixed-case, and returns the right data for that address.

d) **Invalid cursor**.
   a. If I set a **cursor** value that doesn't exist, the server should return an error message and a *400* HTTP status code.

Questions.

- Why some properties returned in the responses are printed out as ***strings*** instead of their proper type?
  - For example, **nonce** and **transaction_index** are ***strings***. I consider they should be ***integers***. Or is the string type defined on purpose? If yes, why?
- Are pending transactions included in the payload, or just the finished ones?

Task 2: Hands-On API Testing.

- **Wallet address**: *0xcB1C1FdE09f811B294172696404e88E658659905*
    - **Transaction**: *0xf53ef12d0b1604acee0a30b27e025980a3370e51cf087581ae8adc3800f3d0ae*

I'll first pay attention to the addresses used in the transaction (**from_address** and **to_address**), and the summary of the transaction to validate that everything was send correctly.

I'll also check the **gas** properties to then validate the **transaction_fee** property.

Talking about exploratory (manual) and automation, I'll use first **Postman** (exploratory testing) and any documentation available to try out the endpoints and validate my testing assumptions (correctness of data, calculations, etcetera). Then I'll create a script in **Playwright** to automate this process for then validating data I want. In the basic script I created, I'm validating that the default properties (**page_size**, **page**, **limit**, **result** and **cursor**) are present, but also, I'm checking the page size, limit, hashes (transactions), addresses, gas prices, receipts of gas used and transaction fee values. Also, I'm validating the schema.

The script can be extended by adding another case in which it validates a specific transaction by calling the endpoint */transaction/{transaction_hash}*, and I can verify that the information displayed in the response is the same as the other endpoint (*/wallets/{address}/history*).

Task 3: Reliability and monitoring.

I'll talk with the developers to verify if there's a monitoring tool (like **Datadog** or **Elasticsearch**) set to log and trace all the actions performed by the features developed by them. If that's the case, I'll ask what kind of alerts are set for detecting issues in the processes. One alert could be to detect anomalous values in the transactions, by sending a notification to a dedicated **Slack** channel for incident management or through email to the ones interested on these issues.

About the ***schemas***, it would be possible to run a nightly suite of automation tests (an automation smoke testing) just to ensure that the schemas are correct. The results can be sent through the right **Slack** channel and notify the persons interested in them and shown in a report. As a team, we can decide which tests can be candidates for this smoke nightly execution.

Bonus points.

- The assessment didn’t include in the beginning a way to generate the token. Thanks for updating the docs.
- I will consider putting more explicit examples of values to set in the parameters and specify the boundaries for them. Talking about the **from_date** and **to_date** parameters, **Moment.js** can generate multiple ***datestring*** values, in diverse formats, but there should be one accepted by the endpoints. For the **limit** parameter, I would consider including the label ***1 to 100***.
- If *1* means ***true*** and *0* ***false***, I will consider in changing these values for the **receipt_status** property. Or what does it mean?