Step 1: Choose API EndpointsFor the CoinGecko API, common endpoints used include:

1. /coins/markets – Get current cryptocurrency market data.

2. /coins/{id}/history – Get historical data for a specific cryptocurrency.

3. /simple/price – Get the current price of specific cryptocurrencies.

Step 2: Sample Test Cases

1. Positive Test Case

Test Case ID: TC\_01

Test Description: Verify the retrieval of current market data for Bitcoin.

Pre-conditions: CoinGecko API is accessible.

Test Steps:

1. Send a GET request to /coins/markets with the parameter vs\_currency=usd&ids=bitcoin.

2. Check the response status code.

3. Validate that the response contains data for Bitcoin.

Expected Results: Status code is 200 OK, and the response body contains market data for Bitcoin.

Actual Results: (Leave blank for now)

2. Negative Test Case

Test Case ID: TC\_02

Test Description: Verify the API response for an invalid cryptocurrency ID.

Pre-conditions: CoinGecko API is accessible.

Test Steps:

1. Send a GET request to /coins/markets with vs\_currency=usd&ids=invalid\_coin.

2. Check the response status code.

Expected Results: Status code is 404 Not Found, with an error message indicating an invalid ID.

Actual Results: (Leave blank for now)

3. Boundary Test Case

Test Case ID: TC\_03

Test Description: Verify the API handles a large date range for historical data.

Pre-conditions: CoinGecko API is accessible.

Test Steps:

1. Send a GET request to /coins/{id}/history for bitcoin with a date far in the past (e.g., 01-01-2000).2. Check the response status code.

Expected Results: Status code is 400 Bad Request or a message indicating that the date is out of range.

Actual Results: (Leave blank for now)

Step 3: Execute Tests Using Postman

1. Set Up Postman:

Import the CoinGecko API endpoints.

Create requests based on the test cases.

Run the tests and compare the actual results with the expected results.

2. Document Bugs (if any):

If a test case fails, create a bug report using the format provided:

Bug ID: BUG\_01Summary: API returns incorrect status code for an invalid cryptocurrency ID.

Steps to Reproduce: See TC\_02 steps.

Expected Result: 404 Not Found

Actual Result: 200 OK with an empty response.

Severity: Major

Screenshot/Logs: Attach if available.

Step 4: Review and Improve

After executing all test cases, reflect on the effectiveness of the tests:

Were there any edge cases you missed? For example, testing with special characters in IDs or testing the API response time under high load.

Add any new test cases based on what you observed during execution.

Step 1: Creating a Test Report

A good test report helps document the testing process and results clearly. Here’s a template you can use for your test report:

Status: Mark each test as Passed, Failed, or Blocked.

Comments: Add any additional observations.

Step 2: Bug Documentation

If any test case fails, create a detailed bug report. Here’s a template for bug reporting:

Bug Report Template

Bug ID: BUG\_01

Summary: Incorrect response status for invalid cryptocurrency ID.Steps to Reproduce:

1. Send a GET request to /coins/markets with vs\_currency=usd&ids=invalid\_coin.

2. Observe the response status code.Expected Result: Status code 404 Not Found with an error message.

Actual Result: Status code 200 OK with an empty response.Severity: Major

Priority: High

Environment: CoinGecko API, tested via Postman.

Screenshots/Logs: Attach any relevant logs or screenshots from Postman.

Step 3: Setting Up and Using Postman

1. Create a Collection in Postman:

Open Postman and click New Collection.

Name your collection (e.g., CoinGecko API Tests).

Add a new request for each test case to the collection.

2. Adding Requests

Click Add Request and set the request type to GET

Enter the API endpoint URL (e.g., https://api.coingecko.com/api/v3/coins/markets).

In the Params tab, add necessary parameters (e.g., vs\_currency=usd, ids=bitcoin).

3. Checking Responses:

Click Send to execute the request.

Check the Status Code and Response Body in the response pane.

If the status or data doesn’t match your expected results, document it as a bug.

4. Automate Tests in Postman:

Example for Test Case 1 (TC\_01):

pm.test("Status code is 200", function () {

pm.response.to.have.status(200);

});

pm.test("Response contains Bitcoin data", function () {

const response = pm.response.json();

pm.expect(response[0].id).to.eql("bitcoin");

});

Step 4: Generate a Test Report in Postman

1. Click Runner in Postman.

2. Select your collection and click Run.

3. After execution, review the results and export the report as a JSON or HTML file for documentation.

Step 5: Review and Improve Test Cases

Go through the results and identify any additional scenarios you might have missed.

Consider adding tests for:

Performance: Measure response time.

Security: Test with invalid headers or missing API keys (if applicable).

Error Handling: Test with incorrect data types or invalid parameters.

Test Report

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case id | Test Case Description | Preconditions | Test Steps | Expected Result | Actual result | Status |
| TC\_01 | Retrieve market data for bitcoin | API accessible | SeeTC\_01steps | 200 ok | 200 ok | Pass |
| TC\_02 | Invalid cryptocurrency Id | API accessible | See TC\_02 steps | 404 not found | 200 ok | Fail |
| TC\_03 | Large data range for history | API accessible | See TC\_03  steps | 400 Bad request | 400 Bad request | Pass |
| TC\_04 | Retrieve prices for multiple coins | API accessible | See TC\_04 steps | 200 ok | 200 ok | Pass |