**Aim:** Generate stress test. Test report using JMeter for the software.

**Description:** **1. Setting up JMeter for Stress Testing**

1. **Download and Install JMeter**:
   * Download Apache JMeter from [here](https://jmeter.apache.org/download_jmeter.cgi).
   * Extract the files and navigate to the bin folder to run jmeter.bat (Windows) or jmeter.sh (Linux).
2. **Launch JMeter**:
   * Open the jmeter.bat file in the bin folder (for Windows users) to launch the JMeter UI.

**2. Creating a Stress Test in JMeter**

**Step 1: Create a New Test Plan**

1. **Test Plan Setup**:
   * In the JMeter UI, right-click on the **Test Plan** and choose **Add > Threads (Users) > Thread Group**. This will be the foundation for your test.
   * A **Thread Group** represents a group of virtual users performing requests.
2. **Configure Thread Group**:
   * **Number of Threads (Users)**: Set the number of virtual users. For stress testing, this number should be high, such as 500 or 1000, depending on the system you want to test.
   * **Ramp-Up Period**: Time (in seconds) to ramp up the number of users to the target value. For example, a ramp-up period of 100 seconds means that JMeter will add 1000 threads gradually over 100 seconds.
   * **Loop Count**: The number of times to execute the requests per user. You can set it to forever for continuous testing.

Example configuration:

* + **Number of Threads**: 500
  + **Ramp-Up Period**: 100 seconds
  + **Loop Count**: Forever

**Step 2: Add HTTP Request Sampler**

1. **Add an HTTP Request**:
   * Right-click on the **Thread Group** and select **Add > Sampler > HTTP Request**.
   * This sampler will be used to simulate HTTP requests (such as GET or POST) to your web application.
2. **Configure the HTTP Request**:
   * **Server Name or IP**: Enter the domain name or IP address of the server you're testing.
   * **Port**: Set the port number (usually 80 for HTTP or 443 for HTTPS).
   * **Method**: Select the HTTP method, such as GET, POST, etc.
   * **Path**: Enter the path to the resource you're testing (e.g., /api/endpoint).

**Step 3: Add Listeners for Test Reporting**

1. **Add View Results Tree**:
   * Right-click on the **Thread Group** and select **Add > Listener > View Results Tree**.
   * This listener will show the result of each request, including response data and status.
2. **Add Aggregate Report**:
   * Right-click on the **Thread Group** and select **Add > Listener > Aggregate Report**.
   * This listener will display a summary of test results, including throughput, response time, error rates, etc.
3. **Add Summary Report**:
   * Right-click on the **Thread Group** and select **Add > Listener > Summary Report**.
   * This listener provides a high-level overview of the performance of your application.

**Step 4: Configure Stress Test Parameters**

1. **Stress Test Duration**:
   * Set the test to run for a sufficient duration, such as 30 minutes or more, to simulate continuous heavy traffic.
   * Under the **Thread Group** settings, ensure the **Loop Count** is set to a high number or **Forever** if you want the test to run indefinitely.
2. **Simulate User Load**:
   * Ensure that the **Ramp-Up Period** is long enough to avoid overwhelming your system instantly.
   * Adjust the **Number of Threads** to simulate a load that your system would normally not handle comfortably (e.g., 500, 1000, or higher).

**3. Running the Stress Test**

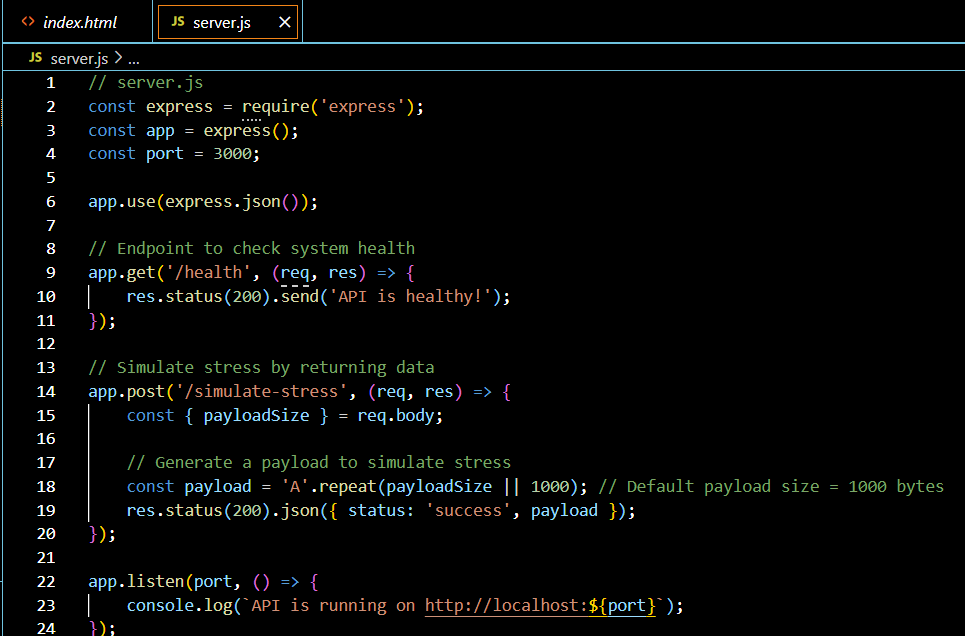
1. **Start the Test**:
   * Click on the **Start** button (green play icon) in the JMeter toolbar.
   * JMeter will begin running the test, sending requests to the server, and collecting results in real-time.
2. **Monitor Results**:
   * Monitor the **View Results Tree**, **Aggregate Report**, and **Summary Report** listeners to track the test's progress and performance.
   * You can track parameters like response times, request success/failure rates, and throughput during the test.

**4. Analyzing the Test Results**

Once the test is completed, you can analyze the **test report** to evaluate the system's performance under stress:

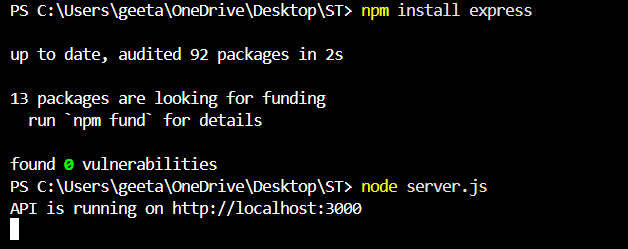
1. **Throughput**: Measures how many requests were handled per second. High throughput indicates that the system is able to handle the load effectively.
2. **Response Time**: Shows the time it takes for the system to respond to requests. Longer response times may indicate the system is under stress.
3. **Error Rate**: Indicates the percentage of failed requests. A high error rate suggests that the system failed to handle the load.
4. **CPU/Memory Usage**: Optionally, monitor the server's CPU and memory usage during the stress test to identify potential bottlenecks.

**Implementation:**

****

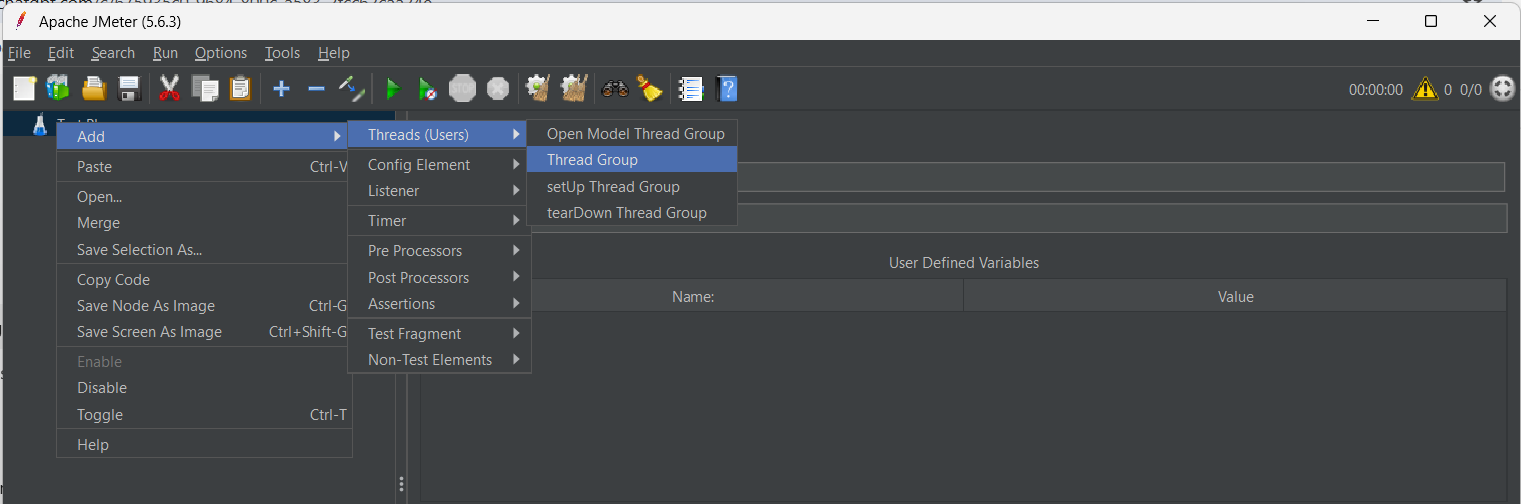
Run the API:

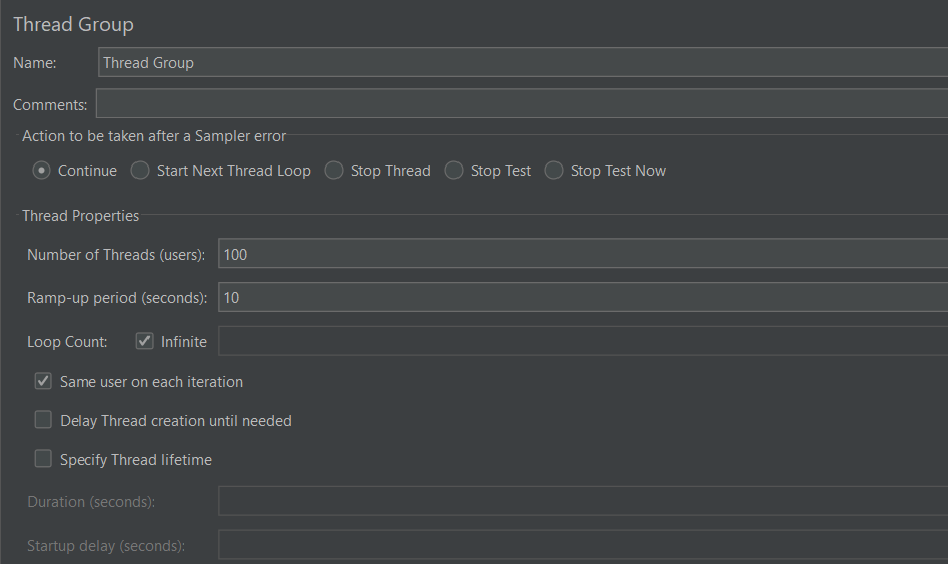
1. Install Node.js dependencies: npm install express.
2. Run the script: node server.js.

****

Step 2: Set up JMeter for Stress Testing

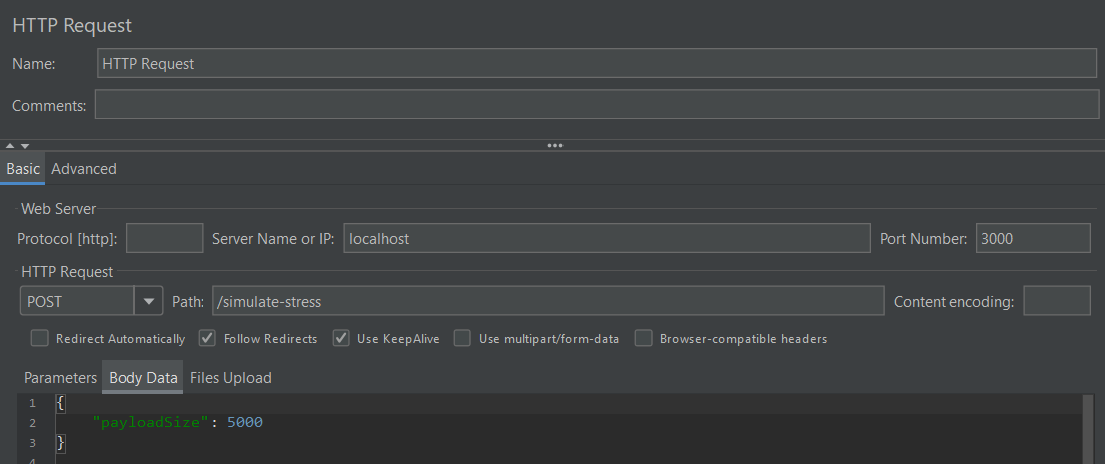
1. Install JMeter  
   Download and install JMeter from the [official site](https://jmeter.apache.org/).
2. Create a Test Plan
   * Open JMeter.
   * Add a Thread Group to simulate multiple users. Set:
     + Number of Threads (users): e.g., 100
     + Ramp-Up Period: e.g., 10 seconds
     + Loop Count: e.g., Infinite or a specific number

****

****

Add an HTTP Request Sampler under the Thread Group:

* Set the server name or IP: localhost
* Port: 3000
* Method: POST
* Path: /simulate-stress
* Add a JSON body (e.g., { "payloadSize": 5000 }).

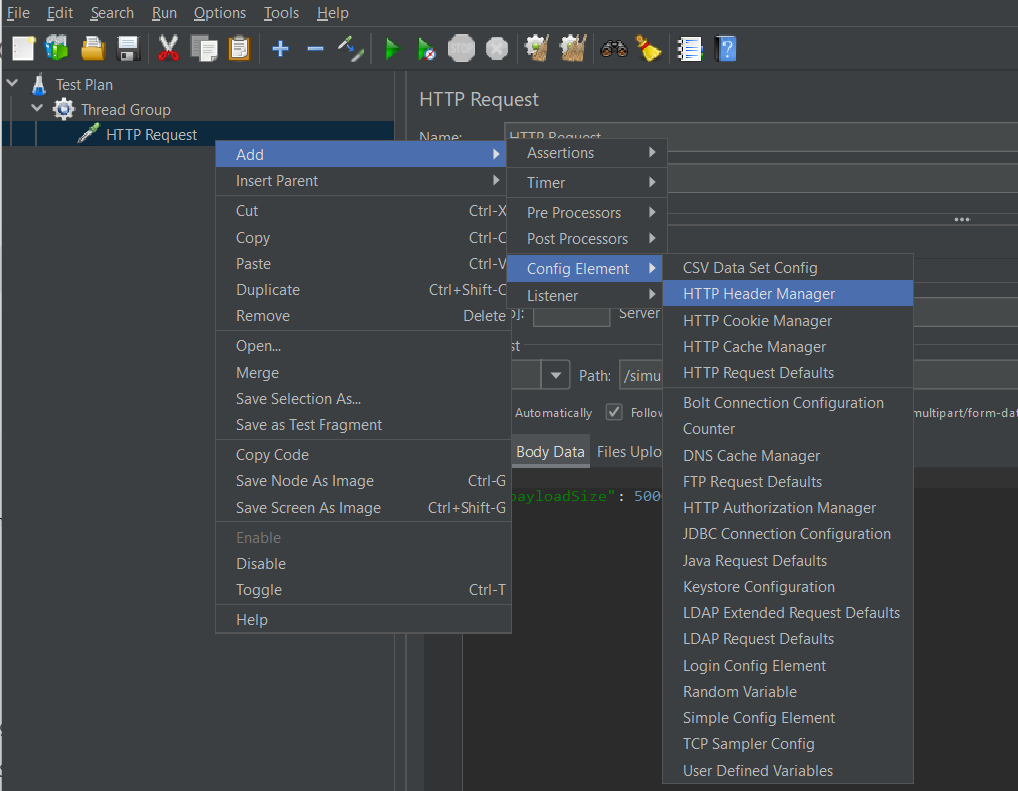
****

Set the Content-Type Header:

* JMeter needs to know you're sending JSON. To do this:
  + Add an HTTP Header Manager to the same HTTP Request Sampler.
  + In the HTTP Header Manager, add the following header:
    - Name: Content-Type
    - Value: application/json

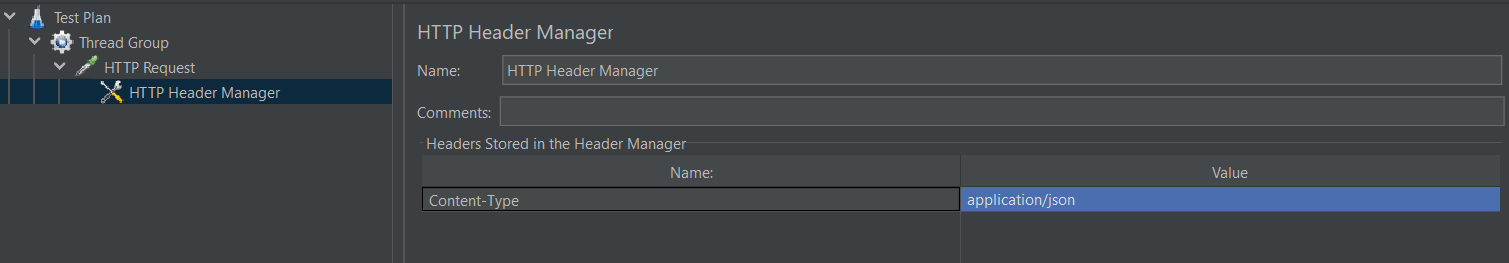
Steps to Add an HTTP Header Manager:

1. Select the HTTP Request Sampler:
   * In your JMeter test plan, select the HTTP Request Sampler where you want to add the header (e.g., the POST request to /simulate-stress).
2. Add the HTTP Header Manager:
   * Right-click on the HTTP Request Sampler in the Test Plan tree.
   * Select Add > Config Element > HTTP Header Manager.
3. Configure the HTTP Header Manager:
   * After adding the HTTP Header Manager, you'll see it listed under the HTTP Request Sampler.
   * Click on the HTTP Header Manager to configure it.
   * In the HTTP Header Manager, click on the Add button (located on the right side).

****

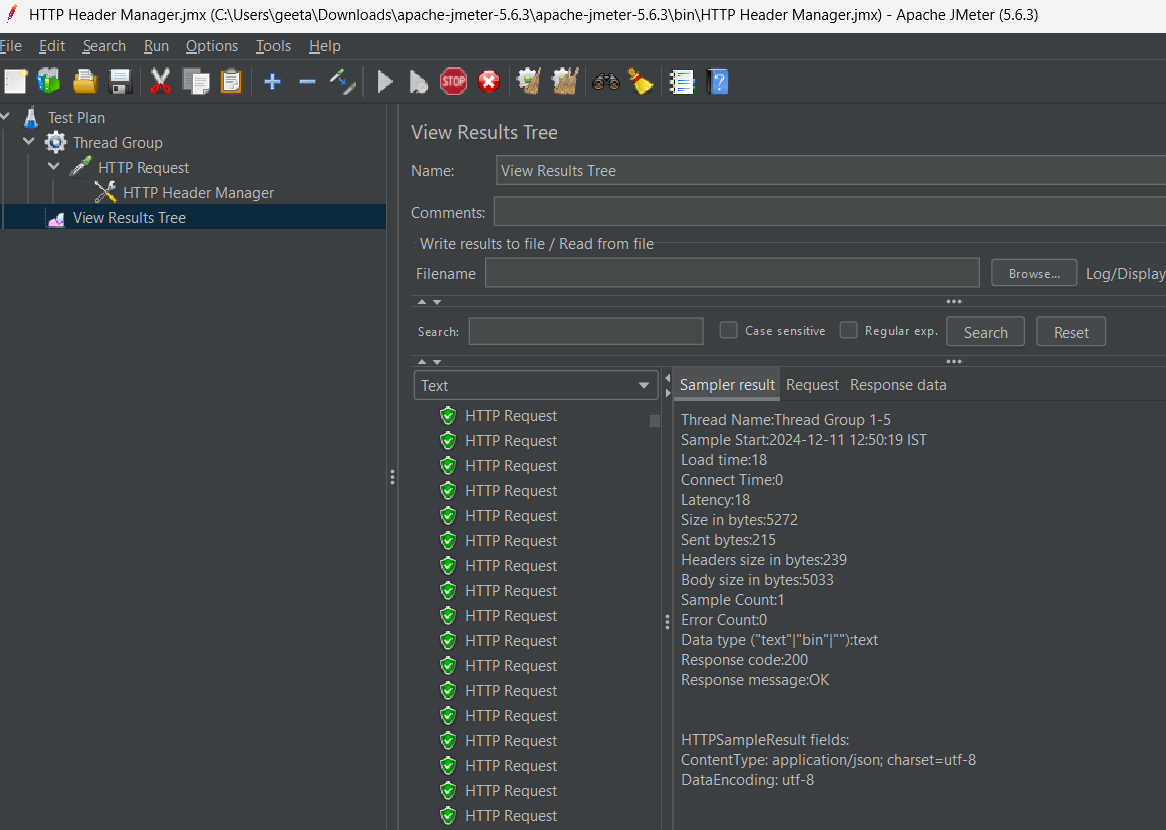
Add the Content-Type Header:

* In the new row that appears:
  + Name: Content-Type
  + Value: application/json

****

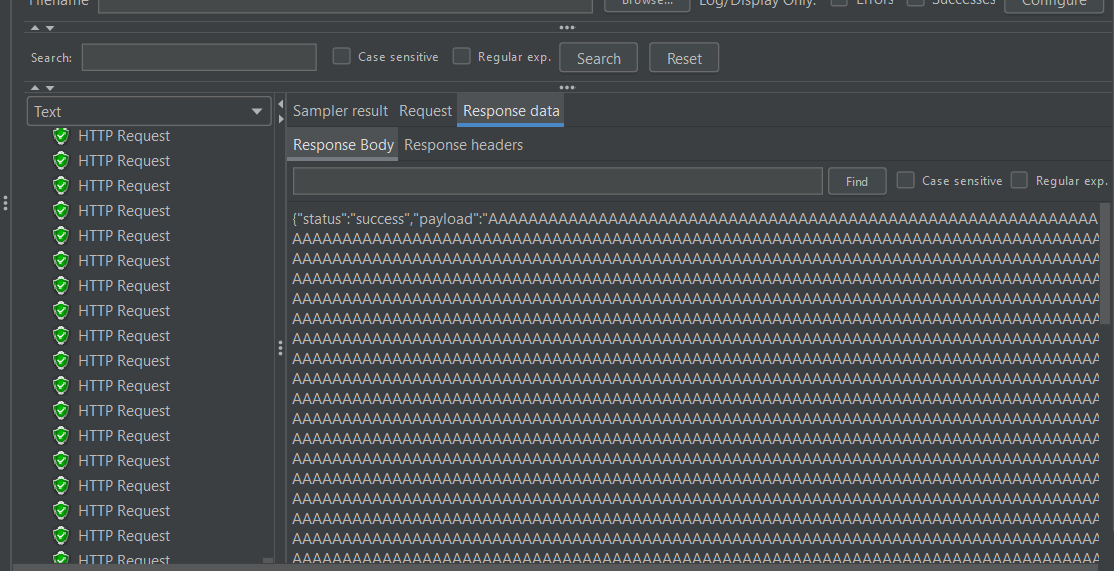
Step 2: Add the View Results Tree Listener

1. Add View Results Tree Listener:
   * Right-click on the Test Plan or Thread Group where you want to see the results.
   * Choose Add > Listener > View Results Tree.
   * The View Results Tree will now appear in the tree on the left side, and it will show detailed results as the test runs.

****

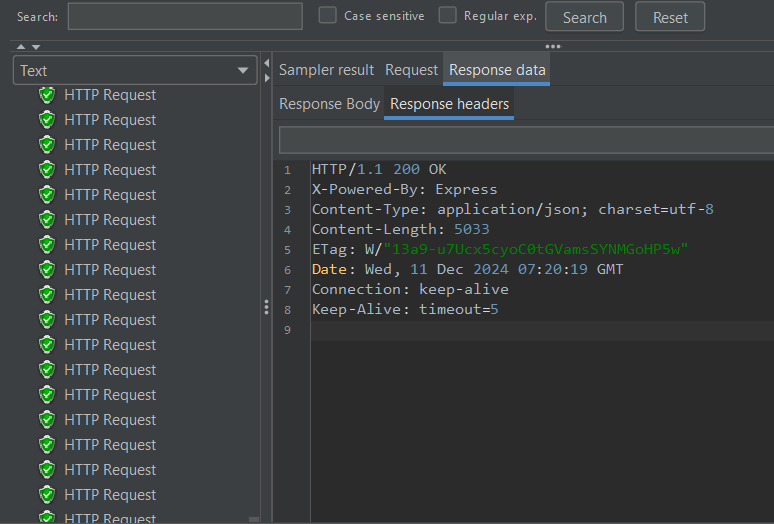
Step 4: Monitor the Results in View Results Tree

1. View Results Tree:
   * As the test runs, the View Results Tree listener will populate with each request's response.
   * You can click on the individual results to see detailed information about:
     + Request: What was sent to the server (including headers, body, and parameters).
     + Response: What the server sent back (including status code, response body, etc.).

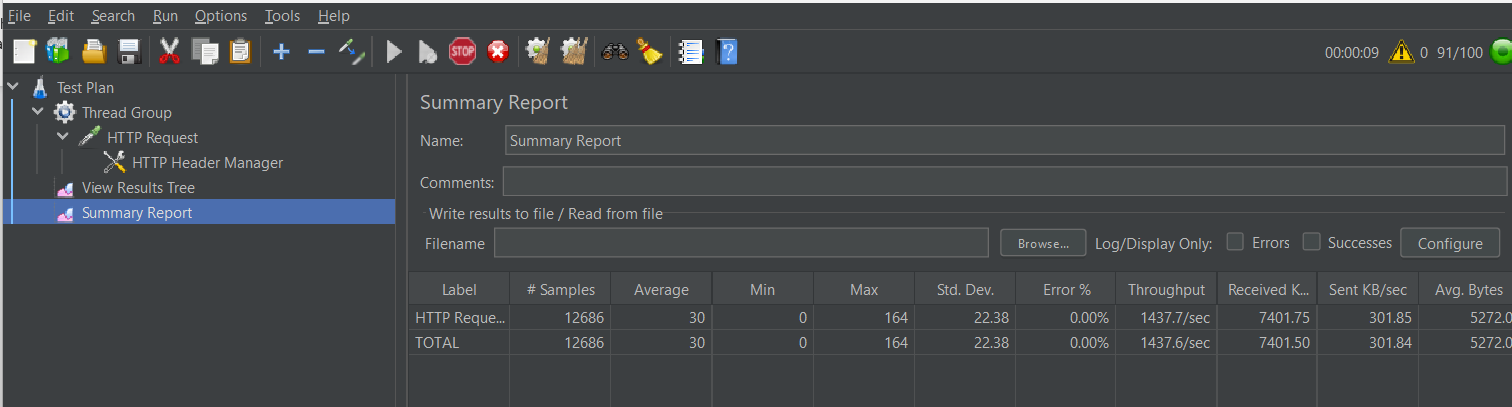
****

Check for Errors:

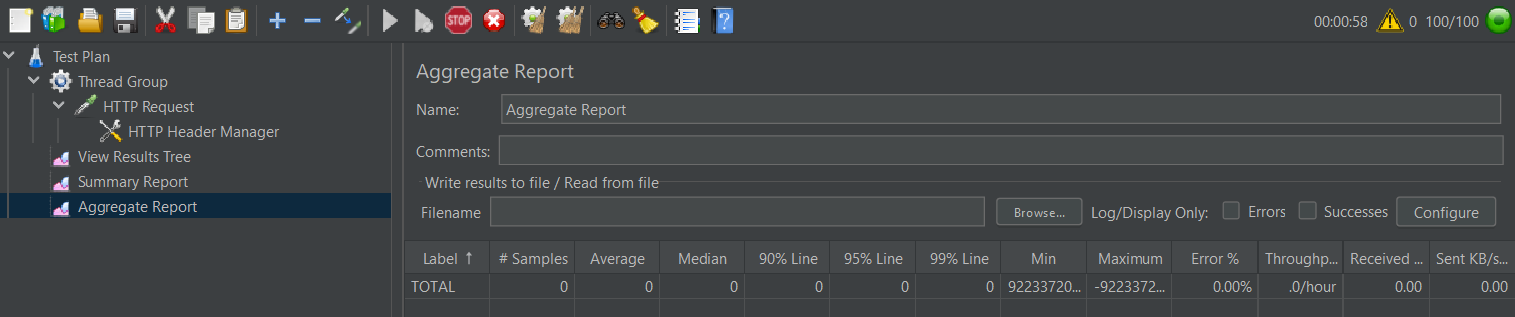
* In the View Results Tree, if the request was successful, you’ll see a 200 OK status in the Response tab.
* If there were errors, they would be displayed, and you can analyze them to troubleshoot.

****

**Summary Report:**

****

**Aggregate Report:**

****

**Conclusion:**   
**Stress testing** with **JMeter** provided valuable insights into how the REST API behaves under load. It revealed the **limits of the API's performance** and highlighted areas where improvements could be made.

The experiment showed that the API can handle a limited amount of traffic but needs **optimization** for handling higher loads efficiently.

The use of **JMeter** as a performance testing tool proved effective in simulating various load conditions and assessing how well the API can scale.