Module - 4

1. Which components have you used in Load Runner?

In LoadRunner, the main components that are typically used in performance testing are:

1. **Virtual User Generator (VuGen)**:
   * Used to create scripts that simulate user activity. These scripts are recorded based on the protocols the application uses (e.g., HTTP, Web Services, Database).
   * It enables the creation of scripts in various languages (such as C or JavaScript) and allows you to parameterize, correlate, and add custom logic.
2. **Controller**:
   * The Controller is used to manage and orchestrate the load test. It allows you to configure and execute the test by distributing virtual users (VUs) across multiple load generators.
   * It helps in defining the load scenario, setting up test execution, monitoring system performance, and controlling the flow of the test.
3. **Load Generator**:
   * Load generators are machines that simulate virtual users (VUs) and generate the load on the application or system being tested.
   * They are controlled by the Controller and help in scaling the test by distributing VUs across multiple systems to simulate realistic traffic.
4. **Analysis**:
   * After the test is executed, the Analysis component helps in processing and analyzing the results. It provides detailed reports, metrics, and graphs to evaluate performance.
   * Key metrics like response time, throughput, error rates, and system resource utilization are displayed to identify potential bottlenecks or issues.

2. How can you set the number of Vusers in Load Runner?

- In **LoadRunner**, you can set the number of virtual users (VUsers) using the **Controller** component, where you define and control the load testing scenario. Here’s how to set the number of VUsers in LoadRunner:

### **1. Through the Controller:**

The **Controller** is the primary interface where you manage and execute load testing scenarios, including configuring the number of virtual users.

**Steps:**

* **Launch the Controller**.
* **Create or Open a Scenario**: If you already have a scenario, open it; otherwise, create a new scenario.
* **Define the Number of VUsers**:
  1. **Scenario Settings**: In the Controller, go to the **"Scenario"** tab.
  2. **Add/Modify VUsers**: In the **VUser Groups** section, you can either add a new **VUser Group** or modify an existing one.
  3. In the **VUser Group** settings, you will see an option to specify the **number of virtual users (VUsers)** for that group.
  4. You can set a fixed number of VUsers or define a ramp-up (incrementally increasing the number of VUs over time) to simulate realistic user load.
* **Example**: If you want to simulate 1000 virtual users, you would enter 1000 in the number of VUsers field.

### **2. Ramp-Up of Virtual Users:**

* You can configure a **ramp-up** to gradually increase the number of virtual users over time, which helps simulate a more realistic load on the system.
* In the Controller, under **"Run Time Settings"**, you can set parameters like:
  + **Start Time**: When to start the test.
  + **Duration**: How long the test will run.
  + **Ramp-Up Time**: The time period over which VUs are gradually introduced.
  + **Steady-State Time**: The time after which the load stabilizes with a set number of VUs.

3. What is Correlation?

- **Correlation** in LoadRunner (and in performance testing in general) refers to the process of capturing dynamic values (e.g., session IDs, tokens, user-specific data) returned by the server during the execution of a test and then substituting them into subsequent requests. This ensures that the simulated users' actions are realistic and consistent, mimicking the behavior of real users during load testing.

4. What is the process for developing a Vuser Script?

- Developing a **Virtual User (VUser)** script in LoadRunner is a key step in performance testing, as it simulates real user interactions with the application under test. The process involves capturing user interactions, translating them into a script, and making the script dynamic to reflect real-world usage patterns. Below is the general process for developing a VUser script:

### **1. Identify the Business Process**

* Before starting the script development, identify the business processes you want to test. These could be critical user flows, such as logging in, purchasing a product, checking account balances, or submitting a form.
* The business process should reflect the key user journeys that you expect your application to handle during peak traffic.

### **2. Set Up the Environment**

* **Install LoadRunner**: Ensure that LoadRunner is installed on the system where you plan to develop the script.
* **Choose the Appropriate Protocol**: Depending on the application (e.g., web, mobile, or SAP), select the correct protocol in **VuGen (Virtual User Generator)** for the script recording. Common protocols include HTTP/HTTPS, Web Services, Oracle, etc.

### **3. Record the Script (with VuGen)**

* **Launch VuGen**: Open the VuGen (Virtual User Generator) application, which allows you to create, record, and modify scripts.
* **Create a New Script**:
  1. Go to **File > New Script** in VuGen, and choose the appropriate protocol (e.g., Web – HTTP/HTML for web applications).
  2. Click **Record** to start capturing user activity.
* **Perform the User Actions**: During the recording session, perform the steps of the business process that you want to simulate. For example, if you're testing a login flow, you might:
  1. Navigate to the login page.
  2. Enter the credentials.
  3. Click the "Submit" button.
  4. Verify the post-login page loads.
* **Stop the Recording**: After completing the user actions, stop the recording in VuGen. VuGen will generate a script that contains the actions you performed.

### **4. Analyze and Refine the Recorded Script**

* **Understand the Script**: The recorded script typically contains a series of HTTP requests and responses that correspond to the actions you performed. These might be in the form of functions like web\_url (for navigating a page) and web\_submit\_data (for submitting a form).
* **Remove Unnecessary Requests**: Some requests (e.g., images, JavaScript files, etc.) that are not part of the critical user flow may have been recorded. Clean up the script by removing or ignoring these unnecessary requests.

### **5. Parameterize the Script**

* **Parameterization** is the process of replacing hardcoded values (like user credentials, product names, or search terms) with variables, allowing the script to simulate multiple users with different inputs.
* **Steps**:
  + In VuGen, identify any hardcoded values (e.g., login username, password, or item IDs).
  + Use the **"Parameterization"** feature to replace these values with dynamic parameters. You can define different values in a data file or generate random values.
  + For example, replace a hardcoded username like "User1" with a parameter such as {Username}.
  + LoadRunner will use a different value from the parameter list for each iteration of the script.

5. How Load Runner interacts with the application?

- **LoadRunner** interacts with the application under test (AUT) through a series of steps that simulate real user behavior to evaluate the system’s performance under various load conditions. Here’s a detailed overview of how **LoadRunner** interacts with the application:

### **1. Recording User Interactions (VuGen)**

* **Virtual User Generator (VuGen)** records the interactions between a user and the application. When you record a script, VuGen captures the communication between the client (e.g., browser or application) and the server (e.g., web server, database, etc.).
* **Protocols**: LoadRunner supports various protocols (e.g., HTTP/HTTPS, Citrix, Web Services, Database, etc.), and VuGen records the interactions based on the protocol chosen during script creation.
  + For **web-based applications**, VuGen records HTTP requests like **GET** and **POST** requests, the data exchanged (e.g., form submissions, session IDs), and response times.

### **2. Simulating User Behavior (Controller)**

* After recording the script, you can use **Controller** to simulate virtual users (VUsers) interacting with the application. The Controller manages how these VUsers will execute the script and can simulate hundreds or thousands of users.
* **Virtual User Execution**:
  + LoadRunner initiates VUsers as per the defined load scenario (e.g., 1000 users).
  + Each VUser executes the recorded script, which replicates user actions like browsing, logging in, making transactions, or performing queries.
  + The Controller sends requests from VUsers to the application, mimicking real-user behavior.

### **3. User Actions via Requests (Request Generation)**

* LoadRunner generates **HTTP requests** or other protocol-specific requests based on the actions defined in the VUser script.
* For example:
  + **HTTP Requests**: If the script is for a web application, the VUser sends **HTTP requests** to the server, including:
    - **GET** requests (for retrieving web pages or resources like images, CSS, or JavaScript).
    - **POST** requests (for submitting forms or user data).
    - **Headers**: Includes cookies, session IDs, or other user-specific data that may be sent along with the request.
* The server receives these requests and processes them just like it would for real users.

### **4. Processing Dynamic Data (Correlation)**

* Applications often return dynamic data that changes with each session (e.g., session IDs, tokens, and time-sensitive values like order numbers).
* **LoadRunner** captures this dynamic data (through **correlation**) and automatically inserts it into subsequent requests to simulate real user behavior.
* **Example**: A session ID returned in one response is used in the next request to maintain the session, simulating continuity in user activity.

### **5. Parameterization of User Data**

* In the recorded script, values like usernames, passwords, or other inputs may be hardcoded. **Parameterization** is used to replace these static values with dynamic ones, which can vary across each VUser to simulate different users.
* For example, a script might simulate multiple users with different usernames and passwords by parameterizing those values.

### **6. Test Execution and Monitoring (Controller)**

* The **Controller** allows for distributing VUsers across multiple load generators. Each load generator runs a subset of the VUsers, simulating real users interacting with the application.
* **Load Testing Scenarios**: The Controller can manage different types of test scenarios, such as:
  + **Load Testing**: Determining how the system performs under a specific load (e.g., 500 concurrent users).
  + **Stress Testing**: Pushing the application beyond its expected capacity to identify its breaking point.
  + **Spike Testing**: Simulating sudden bursts of traffic to see how the system responds.
* **Monitoring**: During test execution, the Controller continuously monitors system performance metrics (e.g., response times, server resource utilization, error rates) and adjusts load based on real-time feedback.

### **7. Server Response Handling**

* LoadRunner sends requests to the application’s server and processes the responses. Depending on the test script, the response could be an HTML page, a JSON/XML payload, or a simple acknowledgment.
* The **response** is logged by LoadRunner for performance analysis, capturing data such as:
  + **Response Time**: Time taken for the server to respond to a request.
  + **Throughput**: The rate at which data is transferred between the client and server.
  + **Error Codes**: HTTP status codes (e.g., 200 OK, 404 Not Found) or custom error codes indicating failed operations.

6. How many VUsers are required for load testing?

- The number of **Virtual Users (VUsers)** required for load testing depends on several factors, including the objectives of the test, the system’s expected usage, and the specific performance goals you want to achieve. Here’s a detailed breakdown to help determine the appropriate number of VUsers for load testing:

### **Key Factors to Determine the Number of VUsers:**

1. **Test Objectives and Goals**:
   * **Load Testing**: The goal is to understand how the system performs under normal expected conditions. The number of VUsers should represent the **expected peak load** (e.g., the maximum number of concurrent users expected during normal operation).
   * **Stress Testing**: The goal is to push the system beyond its capacity to determine its breaking point. Here, the number of VUsers will be **much higher than the expected load** to test how the system behaves under extreme conditions.
   * **Scalability Testing**: If the goal is to test how well the system scales, you may incrementally increase the number of VUsers to see how performance changes with varying user loads.
   * **Spike Testing**: This tests the system's ability to handle sudden spikes in traffic, so the number of VUsers will simulate an **unexpected traffic surge**.
2. **Expected Traffic and Usage**:
   * **Average Concurrent Users**: Estimate the number of concurrent users that will be using the application at peak times. This could be based on:
     + Historical data (e.g., past usage patterns).
     + Business expectations (e.g., sales or marketing campaigns).
     + Industry benchmarks or similar applications.
   * For example, if your website expects **5000 users** to be active simultaneously during peak traffic, you would likely start by testing with **5000 VUsers**.
3. **Application Type and User Behavior**:
   * Different applications (e.g., web, mobile, enterprise apps) have varying user interaction patterns. Some may have short bursts of high traffic, while others may have steady or periodic usage.
   * **User Actions per Session**: The more complex the user actions, the higher the load on the system. For instance, if users perform multiple heavy operations (e.g., submitting large forms, uploading files), you may need fewer VUsers to simulate the same load.
   * **Session Length**: If a VUser keeps an active session for a long period (e.g., 30 minutes), it simulates more sustained load on the system. Shorter sessions may require more VUsers to simulate the same load in a shorter time.
4. **Application Performance Requirements**:
   * If the application needs to handle a certain number of users without degrading performance, you should calculate the VUsers based on **response time goals**, **throughput**, and **error rates** that meet business expectations.
   * **Performance SLAs** (Service Level Agreements) could specify the number of users the application needs to support while maintaining acceptable response times (e.g., 95% of users must experience a response time under 2 seconds).
5. **Infrastructure and System Constraints**:
   * The underlying infrastructure (servers, network, databases, etc.) can limit how many VUsers can be tested in a single scenario. The number of VUsers you choose for testing should consider the capacity of the system being tested and the test environment’s resources.
   * Load testing is typically done in a controlled environment to avoid affecting live users or production systems. This may limit the number of VUsers you can simulate, depending on available hardware and load generator capacity.

7. What is the relationship between Response Time and Throughput?

- **The Relationship:**

* **Inverse Relationship**: There is often an **inverse relationship** between response time and throughput. That is:
  + As **throughput increases**, response time tends to increase, and vice versa.
  + This happens because as more requests are processed simultaneously or more data is handled, the system may experience more congestion or contention for resources, leading to longer delays for individual requests, thereby increasing response time.
  + On the other hand, if throughput is reduced (fewer requests being handled at once), response times can often decrease as the system is less likely to become overloaded.
* **System Capacity and Load**:
  + At low loads (few requests or tasks), both response time and throughput are often low because the system isn't fully utilized.
  + As load increases, throughput rises because the system is processing more requests in a given time frame.
  + However, beyond a certain point (the system's maximum capacity), increasing load (or throughput) can lead to a significant increase in response time due to bottlenecks, resource exhaustion, or queuing delays.
* **Optimal Balance**:
  + The goal is often to find an optimal balance where throughput is maximized without causing excessive increases in response time. This involves ensuring that the system is capable of handling many requests efficiently while minimizing the time it takes to process each request.

### **Example:**

In a web server scenario:

* If the server receives a high number of requests (high throughput), it may be able to serve more clients per second, but each individual client might experience longer wait times (increased response time) due to server congestion.
* If the server's load is too high, response time may increase significantly as the system struggles to process all the incoming requests, and throughput may actually decrease if requests start queuing up or if the server becomes overloaded.

8. What is Automation Testing?

- **Automation Testing** refers to the use of specialized software tools and scripts to automatically execute tests on software applications, with the goal of verifying that they behave as expected. It involves the automation of repetitive and time-consuming manual testing tasks, enabling quicker and more reliable testing processes.

### **Key Features of Automation Testing:**

1. **Test Execution Automation**:
   * Automation testing allows for the execution of tests without human intervention. Test scripts are written to simulate user interactions and validate whether the system is functioning correctly.
2. **Reusable Test Scripts**:
   * Once written, test scripts can be reused across different test cycles, saving time and effort in repetitive testing scenarios (e.g., regression testing).
3. **Faster Execution**:
   * Automated tests can be run much faster than manual tests, especially in complex scenarios with large amounts of data or multiple iterations.
4. **Consistency and Reliability**:
   * Automated tests can run exactly the same way each time, which reduces the risk of human error and ensures more reliable results.
5. **Continuous Integration (CI)**:
   * Automation testing is an essential component of continuous integration (CI) and continuous delivery (CD) pipelines, where automated tests are executed every time new code is integrated to ensure that new changes do not introduce defects.

9. Which Are The Browsers Supported By Selenium Ide?

- Selenium IDE (Integrated Development Environment) is a popular tool for automating web applications. As of the latest updates, Selenium IDE supports the following browsers:

### **1. Google Chrome**

* Selenium IDE provides an extension for **Google Chrome**, allowing users to record, edit, and run tests directly from the browser.

### **2. Mozilla Firefox**

* Selenium IDE also offers an extension for **Mozilla Firefox**, making it one of the most commonly used browsers for testing in Selenium IDE.

### **3. Microsoft Edge (Limited Support)**

* Starting from recent versions, Selenium IDE supports **Microsoft Edge** as well. However, the level of support might not be as robust as Chrome or Firefox, and it may require certain settings or configurations to work properly.

### **4. Opera (Indirect Support via Chrome Extension)**

* While **Opera** itself is not natively supported, you can use the **Chrome extension** of Selenium IDE with Opera, as Opera is based on Chromium, the same engine used by Chrome.

10. What are the benefits of Automation Testing?

- Automation testing offers numerous benefits, particularly in improving the efficiency, reliability, and scalability of the software testing process. Here are the key benefits of automation testing:

### **1. Faster Execution**

* **Speed**: Automated tests can run significantly faster than manual tests, especially for repetitive tasks or large test suites. This reduces the overall testing cycle time and allows more tests to be executed in a shorter period.
* **Parallel Execution**: Automated tests can be run simultaneously on different machines or environments, speeding up the overall process even further.

### **2. Reusability of Test Scripts**

* Once automated test scripts are written, they can be reused for future testing cycles without additional effort. This is particularly useful for regression testing, where the same tests need to be repeated with every new version of the software.

### **3. Consistency and Reliability**

* Automated tests are executed in the exact same manner each time, eliminating the variability that comes with manual testing (such as human errors, fatigue, or oversight).
* Automated tests ensure consistency in the results, improving the reliability and repeatability of tests across different environments or test cycles.

### **4. Cost-Effectiveness in the Long Run**

* While setting up automation requires initial investment in tools, resources, and test script development, the long-term savings are significant. Automated testing reduces the need for manual labor, cuts down on the time needed for testing, and leads to faster identification of defects.
* Automation can provide substantial cost savings when the tests are repeated frequently or across multiple versions of the software.

### **5. Early Detection of Defects**

* Automated tests can be integrated into continuous integration (CI) and continuous delivery (CD) pipelines, allowing for rapid feedback on new changes. This helps in detecting defects early in the development process, reducing the cost of fixing issues when they are identified later in the lifecycle.

### **6. Increased Test Coverage**

* Automation allows you to execute a larger number of test cases in a shorter time frame, thereby increasing test coverage. This is especially important for testing complex applications or for running tests across different browsers and operating systems.
* It also helps cover edge cases and less frequent scenarios that might otherwise be overlooked in manual testing.

### **7. Better Use of Resources**

* Automation frees up manual testers to focus on more complex and exploratory testing tasks. While automated tests handle repetitive and time-consuming tasks, manual testers can concentrate on tasks like usability testing or testing new features.
* It also enables more tests to be run with fewer testers, optimizing resource allocation.

### **8. Improved Accuracy**

* Automated tests are not subject to human error, which is common in manual testing (e.g., skipping a test case, entering incorrect test data). Automated scripts execute test cases with 100% accuracy every time.
* This leads to more accurate results and fewer false positives or negatives.

11. What are the advantages of Selenium?

- Selenium is one of the most popular frameworks for automating web browsers. It is widely used for testing web applications, and it offers several advantages that make it a preferred choice for developers and testers. Here are the key advantages of **Selenium**:

### **1. Open Source and Free**

* **Cost-Effective**: Selenium is completely free to use, as it is an open-source tool. This reduces the need for expensive testing tools and allows for broad adoption in various organizations, both small and large.
* **Community Support**: Being open-source, it has a large community of developers and testers who contribute to its improvement, making it continuously evolve with regular updates and bug fixes.

### **2. Cross-Browser Testing**

* **Browser Compatibility**: Selenium supports testing across multiple web browsers, such as:
  + Google Chrome
  + Mozilla Firefox
  + Safari
  + Microsoft Edge
  + Opera (via Chrome extensions)
* This cross-browser compatibility ensures that your application behaves consistently across different browsers.

### **3. Supports Multiple Programming Languages**

* Selenium supports multiple programming languages for writing test scripts, including:
  + Java
  + Python
  + JavaScript
  + C#
  + Ruby
  + PHP
* This flexibility allows teams to write tests in the language they are most comfortable with, or the one that best fits the project's needs.

### **4. Cross-Platform Testing**

* **Platform Compatibility**: Selenium can run on different operating systems like:
  + Windows
  + macOS
  + Linux
* This cross-platform support allows for testing web applications across various operating systems, ensuring that the application works seamlessly on different platforms.

### **5. Integration with Other Tools**

* **CI/CD Integration**: Selenium can easily integrate with continuous integration tools like Jenkins, TeamCity, and Bamboo. This allows automated tests to be part of the continuous integration/continuous delivery (CI/CD) pipeline.
* **Test Frameworks**: Selenium integrates with popular test frameworks such as:
  + JUnit
  + TestNG
  + NUnit
* **Reporting Tools**: It can be integrated with reporting tools like Allure and ExtentReports to generate detailed and customizable reports after test execution.

### **6. Support for Multiple Browsers and Devices**

* Selenium supports not only desktop browsers but also mobile browsers through tools like **Appium** (for mobile web testing). You can automate testing on real devices, as well as emulators and simulators for both Android and iOS platforms.

### **7. Parallel Test Execution**

* **Faster Testing**: Selenium supports parallel test execution through tools like **Selenium Grid**. This allows multiple tests to run concurrently across different machines or browsers, reducing the overall test execution time and speeding up the testing process.

### **8. Flexible and Scalable**

* **Selenium Grid**: It allows for distributed test execution, meaning you can run tests on different machines and browsers at the same time. This improves scalability and makes it easier to manage large test suites.
* **Modular Frameworks**: Selenium supports various frameworks and methodologies, such as **Page Object Model** (POM), **Data-Driven Testing**, and **Behavior-Driven Development** (BDD), offering flexibility in designing tests.

12. Why testers should opt for Selenium and not QTP?

- **Cost**: Selenium is free and open-source, while QTP/UFT requires expensive licenses.

* **Cross-Browser and Cross-Platform Testing**: Selenium offers superior support for multiple browsers and platforms.
* **Flexibility**: Selenium is more customizable and extensible due to its open-source nature.
* **Wide Language Support**: Selenium supports multiple programming languages, whereas QTP/UFT is primarily tied to VBScript.
* **Community and Ecosystem**: Selenium has a larger, more active community and ecosystem.
* **Mobile Testing**: Selenium works with Appium for mobile testing, while QTP/UFT requires additional tools and licenses.