

Testing Your Application



Testing APIs with Postman



Introduction

- An API, as well as an application, must be testable.
- It must be possible to test the different methods of an API in order to verify that it responds without error.
- And the response of the API is equal to the expected result.
- Proper functioning of an API is much more critical to an application, because this API is potentially consumed by several client applications, and if it does not work, it will have an impact on all of these applications.

Why Postman?

Following are the reasons to use Postman tool.

- **Accessibility:** One can use it anywhere after installing Postman into the device by simply logging in to the account.
- **Use Collections:** Postman allows users to build collections for their API-calls. Every set can create multiple requests and subfolders. It will help to organize the test suites.
- **Test development:** To test checkpoints, verification of successful HTTP response status shall be added to every API- calls.
- **Automation Testing:** Tests can be performed in several repetitions or iterations by using the Collection Runner or Newman, which saves time for repeated tests.

Why Postman?

- **Creating Environments:** The design of multiple environments results in less replication of tests as one can use the same collection but for a different setting.
- **Debugging:** To effectively debug the tests, the postman console helps to track what data is being retrieved.
- **Collaboration:** You can import or export collections and environments to enhance the sharing of files. You may also use a direct connection to share the collections.
- **Continuous integration:** It can support continuous integration.

Benefits of API Testing Using Postman tool

- **Store information in an organized way:** For subsequent API calls, you can use the stored response as part of a request header. You can use this when API requires data that is received from another API.
- **Effective integration with systems:** Postman's command-line tool makes it easy to integrate with build systems, such as Jenkins.
- **Comfortable business logic implementation:** On the Business logic layer, Postman testing improves the application test coverage.
- **On-the-fly testing:** Postman makes sure that there aren't any bugs or typos by incrementally running the written code.

Benefits of API Testing Using Postman tool

- **Useful scripts:** You have the leverage to manipulate the received data in different ways as the test and test assertions are written in JavaScript.
- **Tons of resources:** Postman also provides you with tons of code snippets with examples if you are new to writing test scripts.
- **Easy move to code repositories:** You can use traditional file sharing to export a test collection and move it from the environment to the environment.
- **Easy Creation of test suites:** To ensure your API is working as expected you can easily create a collection of integration tests in Postman.

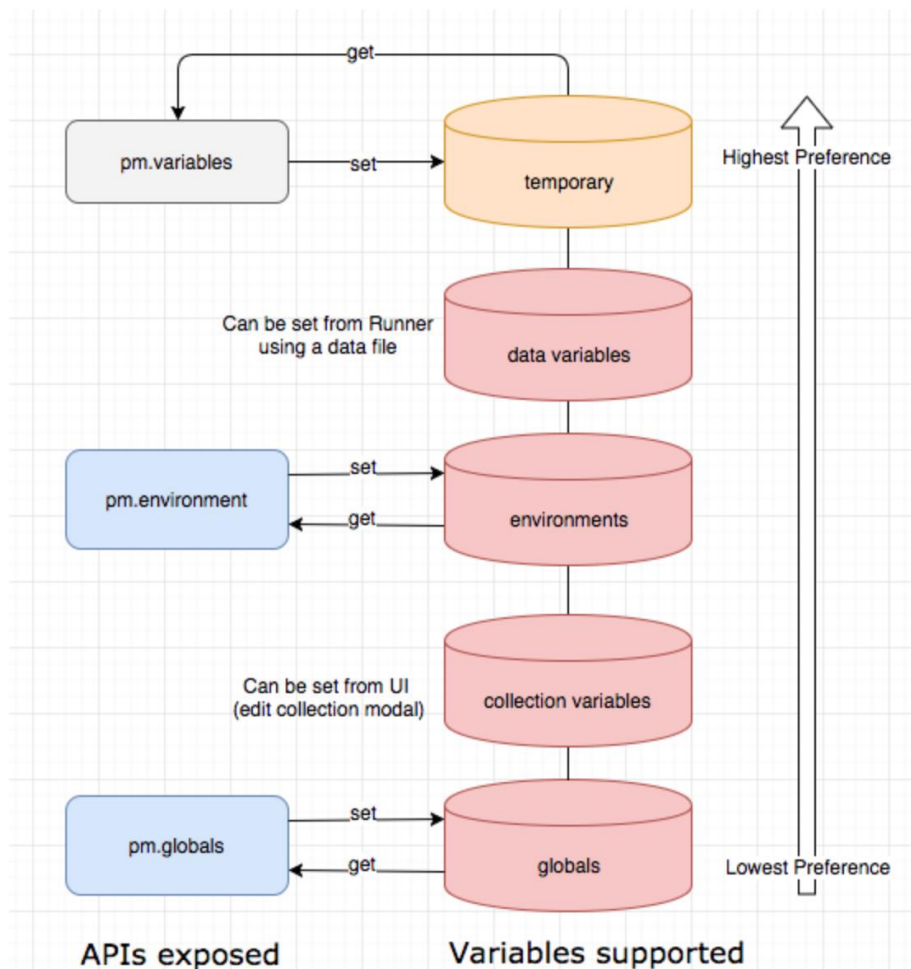
Benefits of API Testing Using Postman tool

- **Capable of running tests in different environments:** Postman inserts the correct environment configuration into your test automatically and allows you to store specific information about different environments.
- **Data storage for Multiple tests:** Postman permits you to store data from the previous test into global variables which can exactly be used as environmental variables for subsequent API calls.

Variable Scopes

- Postman supports variables at different scopes, allowing you to tailor your processing to a variety of development, testing, and collaboration tasks.
- Scopes in Postman relate to the different contexts that your requests run in, and different variable scopes are suited to different tasks.
- In order from broadest to narrowest, these scopes are:
 1. Global
 2. Collection
 3. Environment
 4. Data
 5. Local.

Variable Scopes



Variable Scopes

- **Global Variables:**

- Enable you to access data between collections, requests, test scripts, and environments.
- These are available throughout a workspace.
- Have the broadest scope available in Postman and are well-suited for testing and prototyping.

- **Collection Variables:**

- Available throughout the requests in a collection and are independent of environments.
- Collection variables don't change based on the selected environment.
- Collection variables are suitable if you're using a single environment, for example for auth or URL details.

Variable Scopes

- **Environment Variables:**

- Enable you to scope your work to different environments, for example local development versus testing or production.
- One environment can be active at a time.
- If you have a single environment, using collection variables can be more efficient, but environments enable you to specify role-based access levels.

- **Data Variables:**

- These come from external CSV and JSON files to define data sets you can use when running collections with Newman or the Collection Runner.
- Data variables have current values, which don't persist beyond request or collection runs.

Variable Scopes

- **Local Variables:**
 - These are temporary variables that are accessed in your request scripts.
 - These variable values are scoped to a single request or collection run, and are no longer available when the run is complete.
 - Suitable if you need a value to override all other variable scopes but don't want the value to persist once execution has ended.

Variable Scopes

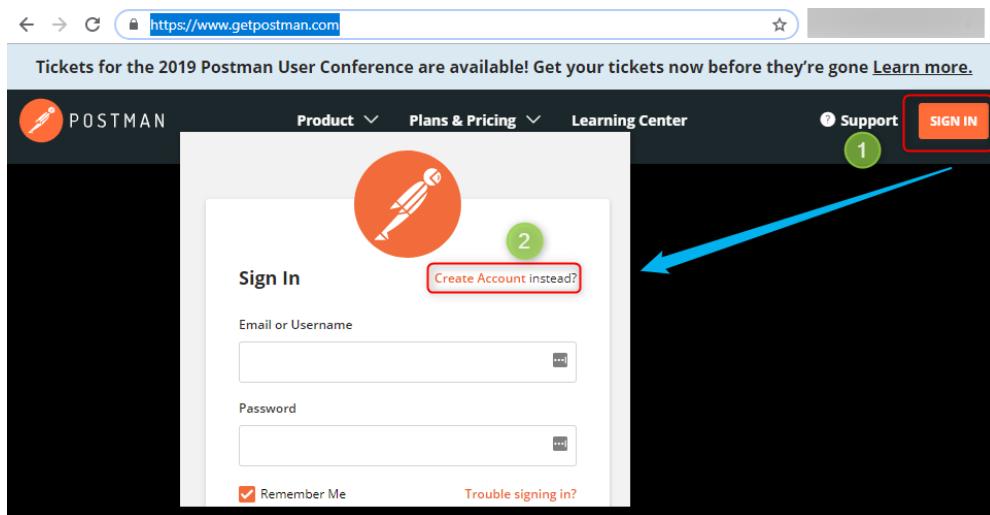
- **Local Variables:**
 - These are temporary variables that are accessed in your request scripts.
 - These variable values are scoped to a single request or collection run, and are no longer available when the run is complete.
 - Suitable if you need a value to override all other variable scopes but don't want the value to persist once execution has ended.

Creating a Postman collection with requests

- Postman is a free client tool in a graphical format that can be installed on any type of OS.
- Its role is to test APIs through requests.
- Allows to dynamize API tests through the use of variables and the implementation of environments.
- Postman is famous for its ease of use, but also for the advanced features that it offers.

Creating a Postman collection with requests

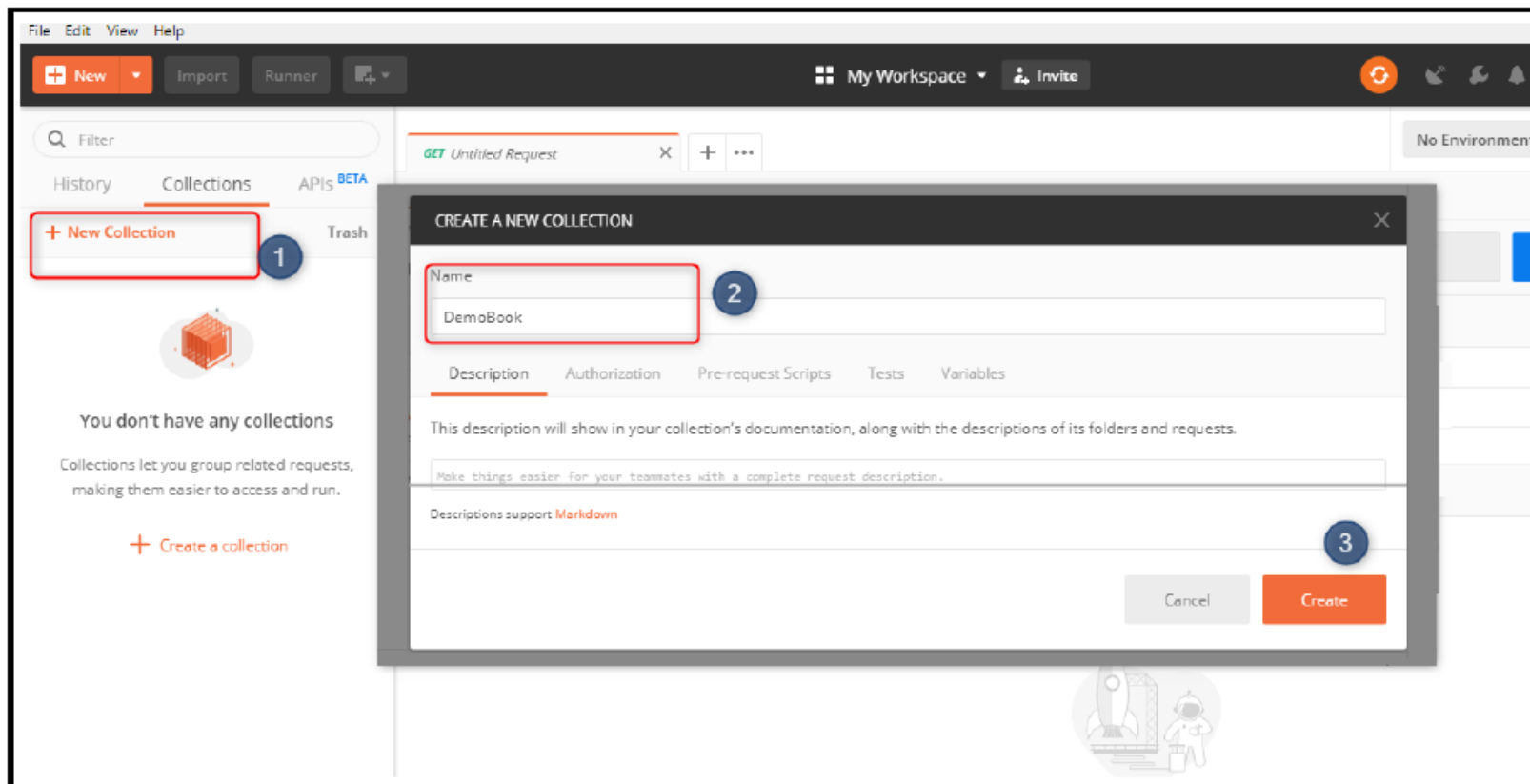
- Create a Postman account by going to <https://www.getpostman.com/> and clicking on the SIGN IN button.
- Then, in the form, click on the Create Account link, as shown in the following screenshot:



Creating a Collection

- In Postman, any request that we test must be added into a directory called Collection.
- This provides storage for requests and allows for better organization.
- Create a DemoBook collection that will contain the requests to the demo API, and perform the following tasks:
 1. In Postman, in the left-hand panel, click on the Collections | New Collection button.
 2. Once the form opens, enter the name as DemoBook, and validate it by clicking on the Create button.

Creating a Collection



Creating a Collection

- This collection will allow to organize the requests of our API tests.
- Also possible to modify its properties in order to apply a certain configuration to all the requests that will be included in this collection.
- These properties include request authentication, tests to be performed before and after requests, and common variables to all requests in this collection.
- To modify the settings and properties of this collection:
 1. Click on the ... button of the context menu of the collection.
 2. Choose the Edit option, and the edit form appears, change all the settings that will apply to the requests in this collection.

Creating a Collection

The screenshot illustrates the process of creating and editing a collection in Postman. On the left sidebar, the 'Collections' tab is active. Under the 'DemoBook' collection, a red box with a '1' highlights the three-dot menu. A dropdown menu is open, and a red box with a '2' highlights the 'Edit' option. The main panel shows the 'EDIT COLLECTION' dialog. The 'Name' field contains 'DemoBook'. Below it, a red box with a '3' highlights the 'Description' tab, which is selected. The description text area contains the placeholder text: 'This description will show in your collection's documentation, along with the descriptions of its folders and requests. Make things easier for your teammates with a complete request description.' At the bottom right, a red box with a '4' highlights the 'Update' button.

History Collections APIs **BETA**

+ New Collection Trash

DemoBook ☆

requests

...

Share Collection

Manage Roles

Rename Ctrl+E

Edit

Create a fork

Merge changes

EDIT COLLECTION

Name

DemoBook

Description Authorization Pre-request Scripts Tests Variables

This description will show in your collection's documentation, along with the descriptions of its folders and requests.

Make things easier for your teammates with a complete request description.

Descriptions support **Markdown**

Cancel Update

Creating our first request

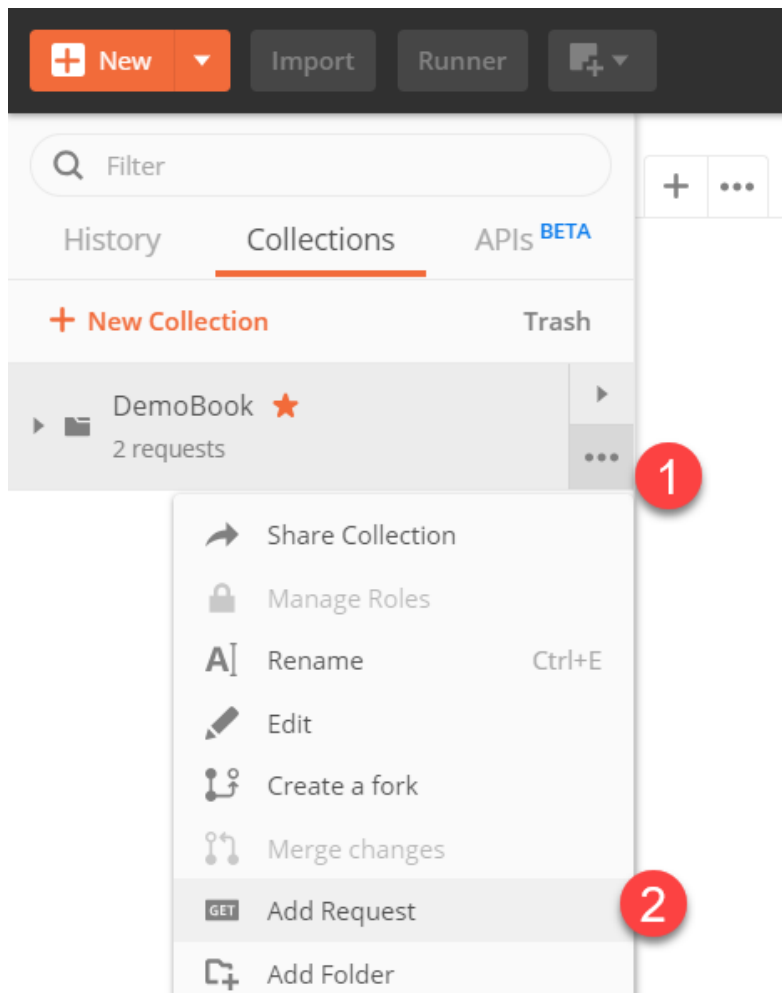
- In Postman, the object that contains the properties of the API to be tested is called a request.
- This request contains the configuration of the API itself, but it also contains the tests that
- are to be performed to check it's functioning properly.
- The main parameters of a request are as follows:
 - The URL of the API
 - Its method: Get/POST/DELETE/PATCH
 - Its authentication properties
 - Its querystring keys and its body request
 - The tests that are to be performed before or after the execution of the API

Creating our first request

- The creation of a request is done in two steps: its creation in the collection and its configuration.

1. The creation of the request: To create the request of our API, here are the steps that need to be followed:

- 1. We go to the context menu of the DemoBook collection and click on the Add Request option:



Creating our first request

- 2. Then, in the form, enter the name of the request, Get all posts. Finally, validate the form by clicking on the Save to DemoBook button, as shown in the following screenshot:

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
[Learn more about creating collections](#)

Request name

Request description (Optional)

Descriptions support [Markdown](#)

Select a collection or folder to save to:

◀ DemoBook

+ Create Folder

Cancel

Save to DemoBook

Creating our first request

2. The configuration of the request: After creating the request, configure it by entering the URL of the API to be tested, which is `https://jsonplaceholder.typicode.com/posts`, in the GET method.

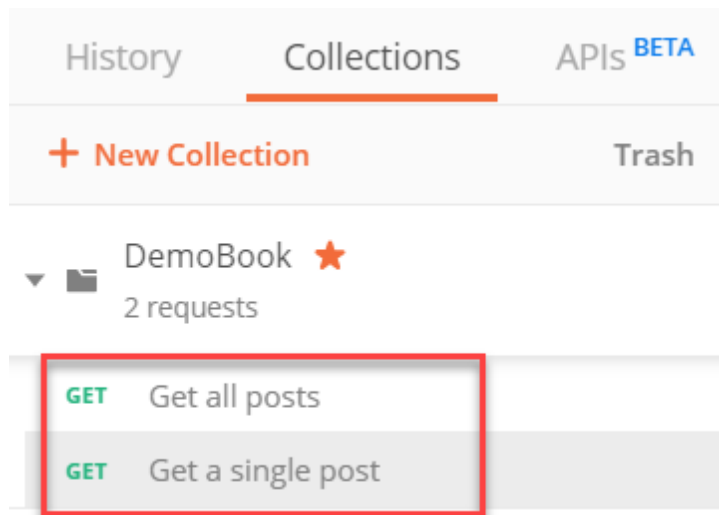
- After entering the URL, save the request configuration by clicking on Save button.

The screenshot displays the REST Client application interface. On the left sidebar, the 'Collections' tab is active, showing a collection named 'DemoBook' with 1 request. A red arrow points to the 'GET Get all posts' request in the list. The main panel shows the configuration for this request. The method is set to 'GET' and the URL is 'https://jsonplaceholder.typicode.com/posts'. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is selected, showing a table for 'Query Params' with columns 'KEY' and 'VALUE'. The table has one row with 'Key' and 'Value'.

KEY	VALUE
Key	Value

Creating our first request

- Finally, to complete the tests, and to add more content to lab, add a second request to the collection, which will be called Get a single post.
- It will test another method of the API, and it will also ensure to be configured it with the `https://jsonplaceholder.typicode.com/posts/<ID of post>` URL.



Using environments and variables to dynamize requests

- To test an API, test it on several environments for better results.
- For example, test it on local machine and development environment, and then also on the QA environment.
- To optimize test implementation times and to avoid having a duplicate request in Postman, inject variables into this same request in order to make it testable in all environments.
- So, in the following steps, improve the requests by creating an environment and two variables; then, modify the requests in order to use these variables:
 1. In Postman, start by creating an environment that is called as Local:

Using environments and variables to dynamize requests

The screenshot illustrates the process of creating a new environment in Postman. The interface is divided into a top toolbar, a left sidebar, and a main workspace. The 'MANAGE ENVIRONMENTS' dialog is open, showing the 'Add Environment' form. The form includes a text input field for the environment name (containing 'Local'), a table for adding variables, and a bottom section with a warning message and 'Cancel'/'Add' buttons. Red boxes and numbers 1-4 highlight the following steps:

1. Click the gear icon in the top right corner of the workspace to open the environment settings.
2. Click the 'Add' button in the bottom bar to create a new environment.
3. Click the 'Add Environment' button in the 'MANAGE ENVIRONMENTS' dialog.
4. Click the 'Add' button in the bottom right corner of the 'MANAGE ENVIRONMENTS' dialog to save the new environment.

MANAGE ENVIRONMENTS

An environment is a set of variables that allow you to switch the context of your requests. Environments can be shared between multiple workspaces. [Learn more about environments](#)

You can declare a variable in an environment and give it a starting value, then use it in a request by putting the variable name within curly-braces. [Create an environment](#) to get started.

Add Environment

Local

VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
Add a new variable					

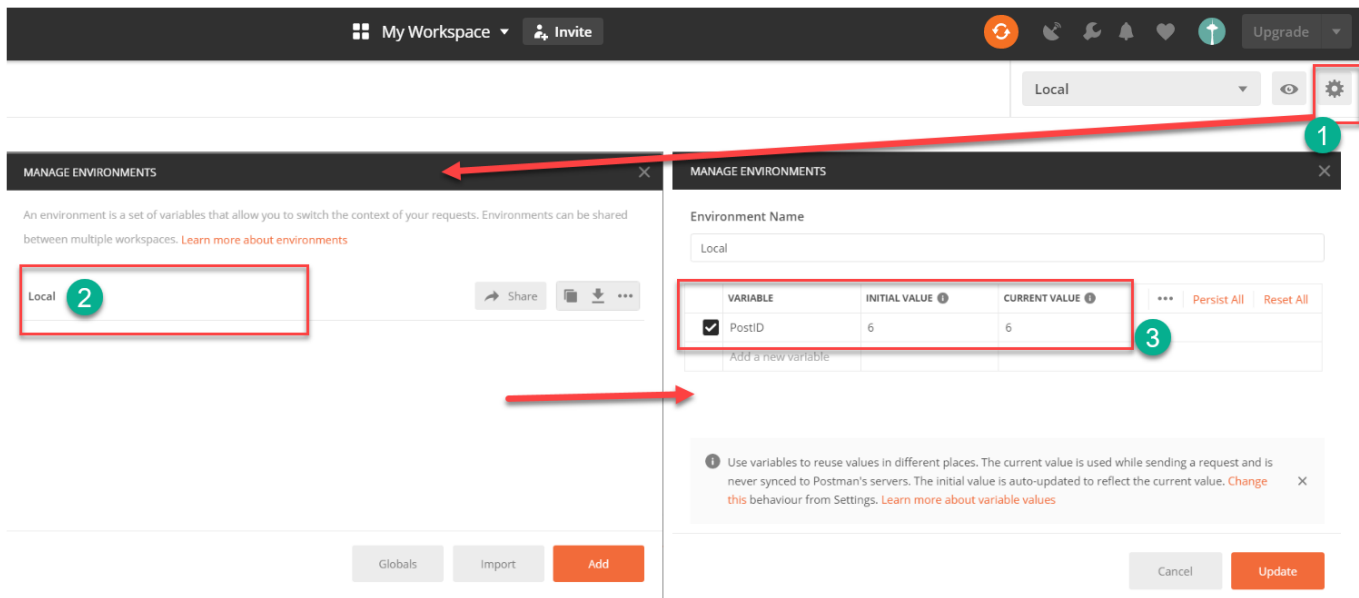
ⓘ Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. [Learn more about variable values](#)

Cancel Add

Using environments and variables to dynamize requests

2. Then, in this Local environment, insert a variable named PostID, which will contain the value to pass in the URL of the request.

- This following screenshot shows the creation of the PostID variable:



Using environments and variables to dynamize requests

3. Thus, for the Local environment, the value of the PostID variable is 6.

- To have a different value for other environments, it is necessary to create other environments using the same steps, and then adding the same variables (with the same name) and their corresponding values.

MANAGE ENVIRONMENTS

Environment Name

QA

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	
<input checked="" type="checkbox"/>	PostID	7	7	
	Add a new variable			

⋮

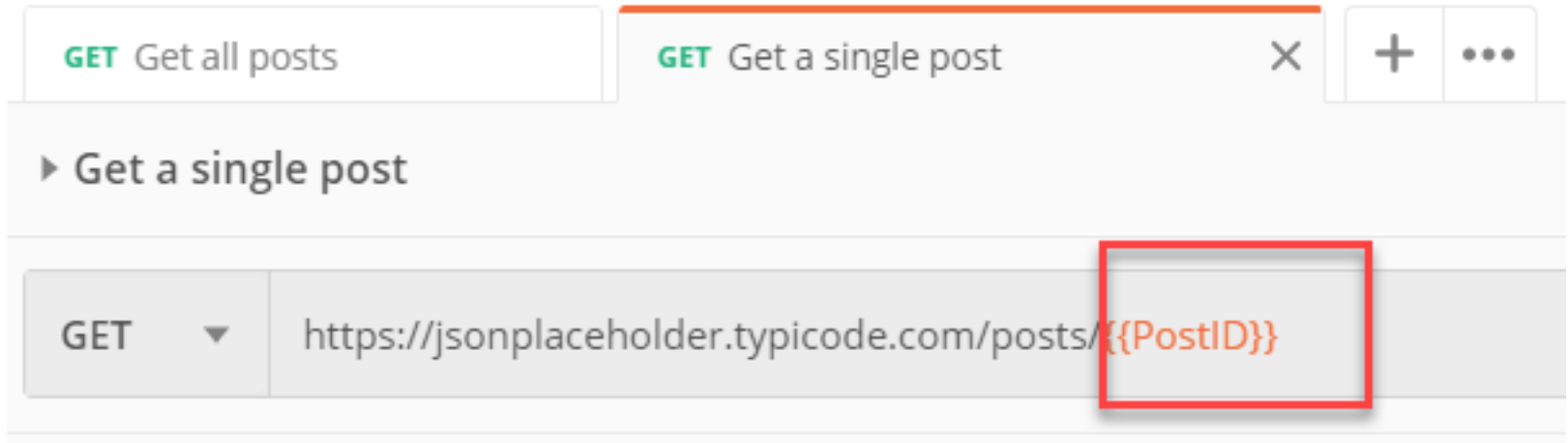
Persist All

Reset All

Using environments and variables to dynamize requests

- Finally, modify the request in order to use the variable that was just declared.
- In Postman, the usage of a variable is done using the `{{variable name }}` pattern.
- So first, select the desired environment from the dropdown in the top-right corner.
- Then, in the request, replace the post's ID at the end of the URL with `{{PostID}}`

Using environments and variables to dynamize requests

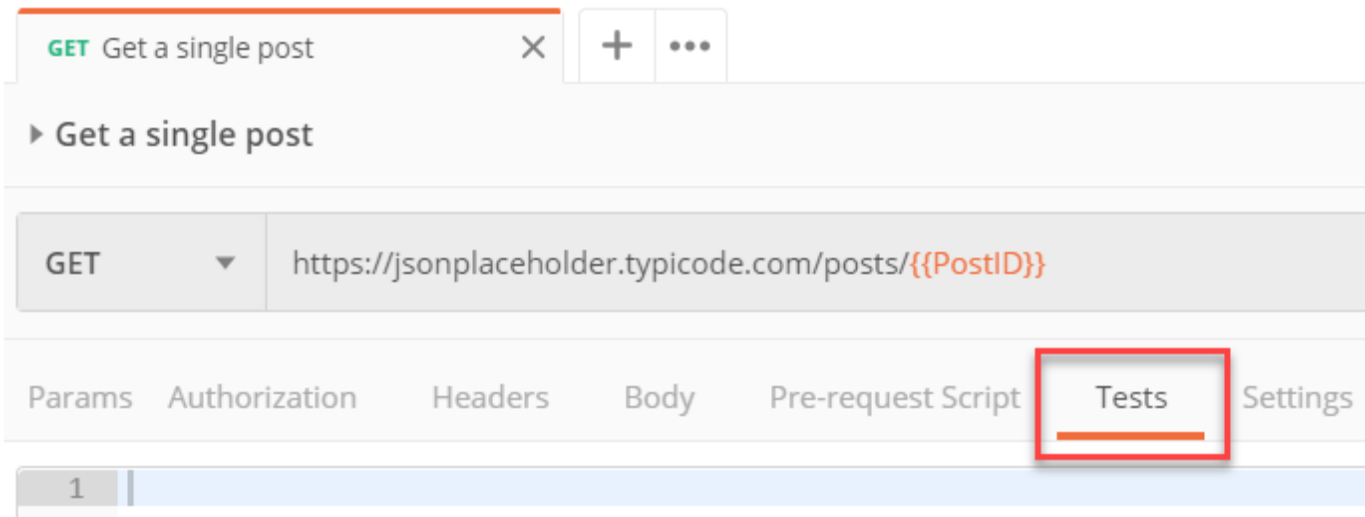


Writing Postman tests

- Testing an API is about checking that its call returns a return code of 200, that is, that the API responds well.
- Also its return result corresponds to what is expected, or that its execution time is not too long.
- For example, consider an API that returns a response in JSON format with several properties.
- In the tests of this API, it will be necessary to verify that the result returned is a JSON text that contains the expected properties, and even more so to verify the values of these properties.

Writing Postman tests

- In Postman, it is possible to write tests that will ensure that the response of the request corresponds to the expected result in terms of return or execution time using the JavaScript language.
- Postman tests are written in the Tests tab of the request:



Writing Postman tests

- To test few request, write several tests, which are as follows:
- That the return code of the request is 200

The following code illustrates the return code of the request:

```
pm.test("Status code is 200", function () {  
  pm.response.to.have.status(200);  
});
```

Writing Postman tests

- To test few request, write several tests, which are as follows:
- That the response time of the request is less than 400 ms

The following code illustrates a response time of less than 400 ms:

```
pm.test("Response time is less than 400ms", function () {  
  pm.expect(pm.response.responseTime).to.be.below(400);  
});
```

Writing Postman tests

- To test few request, write several tests, which are as follows:
- That the answer is not an empty JSON

The following code illustrates that the response in JSON format is not empty:

```
pm.test("Json response is not empty", function () {  
  pm.expect(pm.response).to.be.json;  
});
```

Writing Postman tests

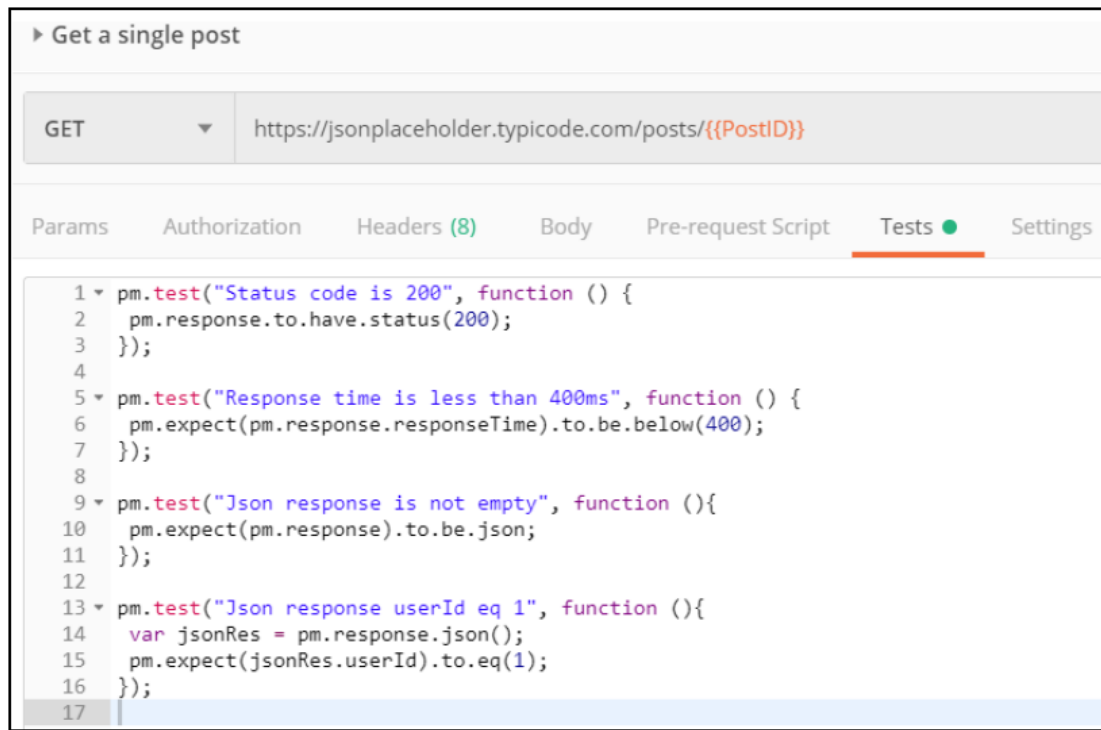
- To test few request, write several tests, which are as follows:
- That the return JSON response contains the `userId` property, which is equal to 1.

The following code illustrates that, in the JSON response the `userId` property is equal to 1:

```
pm.test("Json response userId eq 1", function () {  
  var jsonRes = pm.response.json();  
  pm.expect(jsonRes.userId).to.eq(1);  
});
```

Writing Postman tests

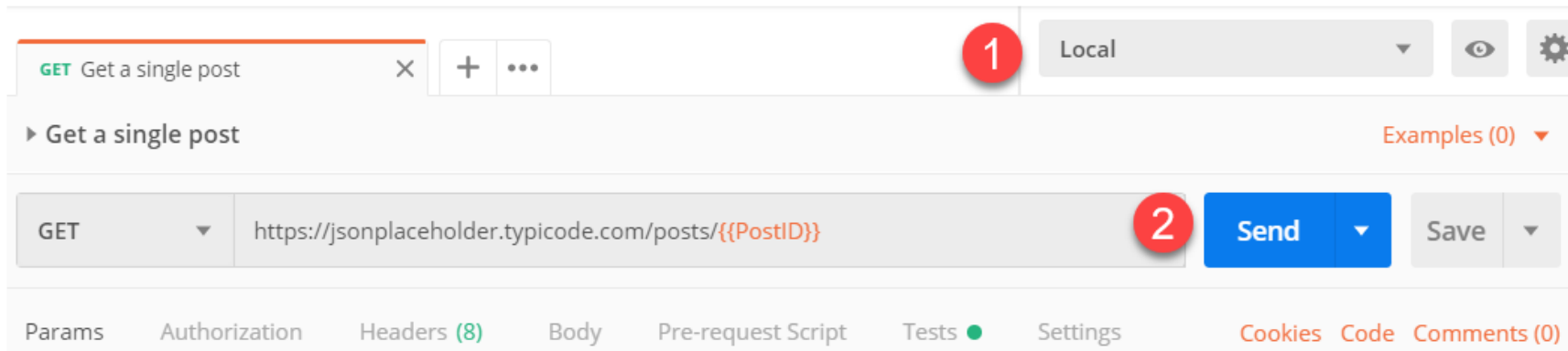
- And so, finally, the Tests tab of the request, which tests the API, contains all this code, as shown in the following screenshot:



Executing Postman request tests locally

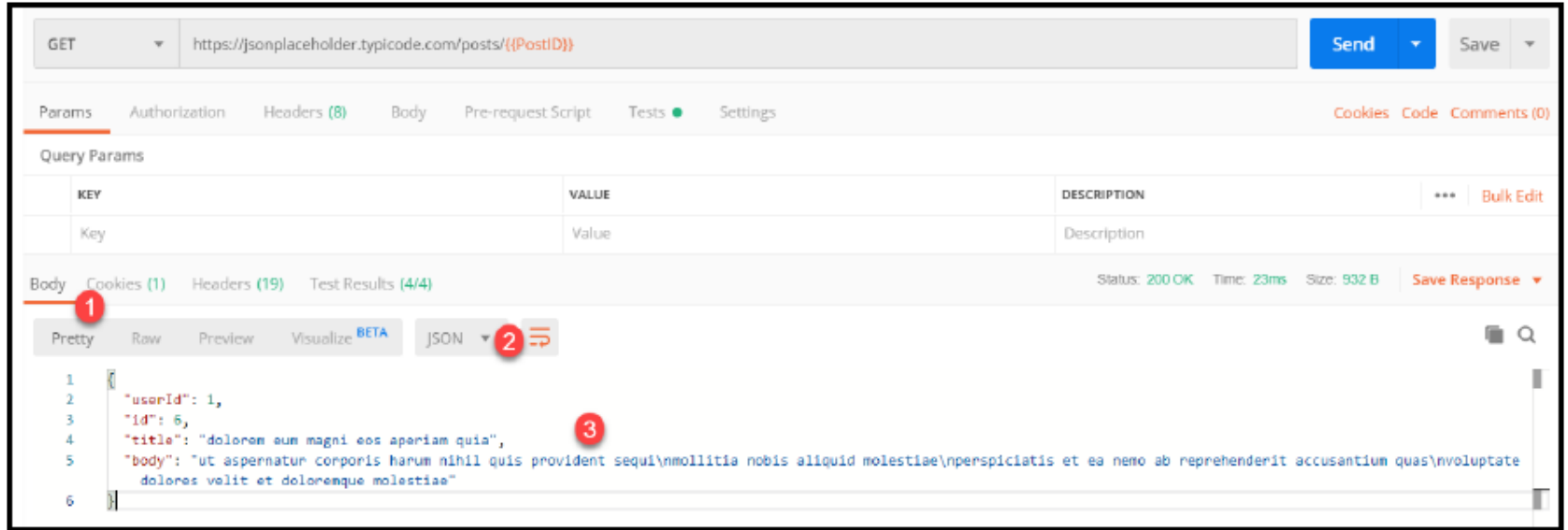
- To test the proper functioning of the APIs with their parameters and tests, execute the requests that are in Postman.
- Note that it will only be at the end of this execution that we will know whether the APIs correspond to our expectations.
- To execute a Postman request, perform the following actions:
 1. First choose the desired environment.
 2. Click on the Send button of the request, as shown in the following screenshot:

Executing Postman request tests locally



Executing Postman request tests locally

3. In the Body tab, view the content of the query response, and to display it in JSON format, choose the display format.
- The following screenshot shows the response of the request displayed in JSON format:



Executing Postman request tests locally

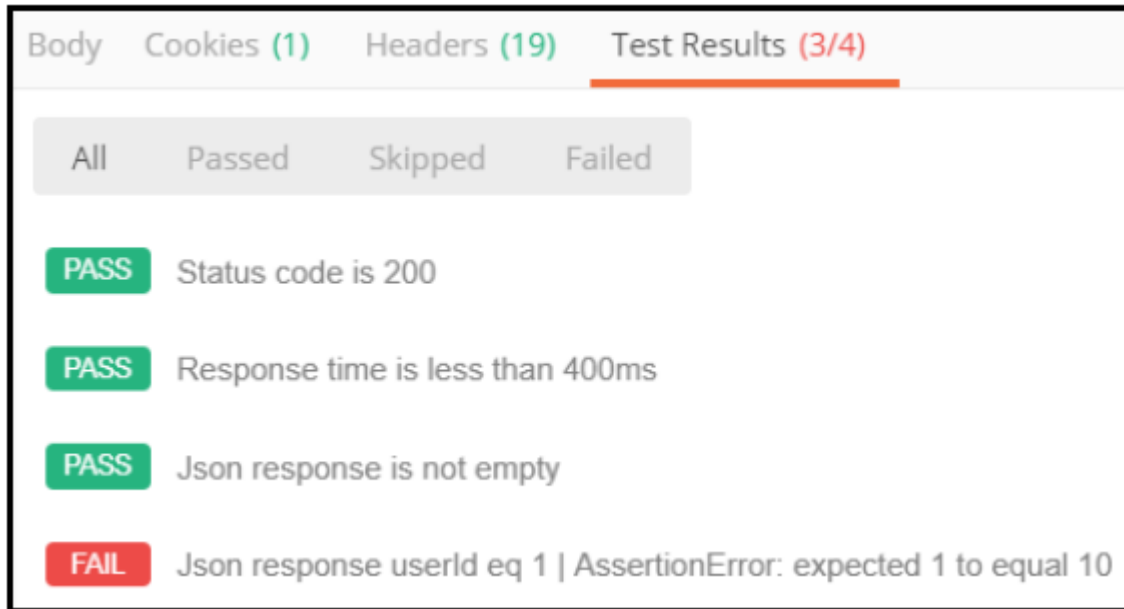
4. The Test Results tab displays the results of the execution of the tests that was previously written, and the four tests have been executed correctly—they are all green, as shown in the following screenshot:

The screenshot shows the Postman Test Results tab. At the top, there is a table with columns: KEY, VALUE, and DESCRIPTION. Below this, there is a tab bar with options: Body, Cookies (1), Headers (19), Test Results (4/4), and a red circle with the number 1. To the right of the tab bar, there is a status bar showing: Status: 200 OK, Time: 23ms, Size: 932 B, and a Save Response button. Below the tab bar, there are filters: All, Passed, Skipped, and Failed. A red circle with the number 2 is next to the Passed filter. The main area displays four test results, each with a green PASS button and a description:

- PASS Status code is 200
- PASS Response time is less than 400ms
- PASS Json response is not empty
- PASS Json response userid eq 1

Executing Postman request tests locally

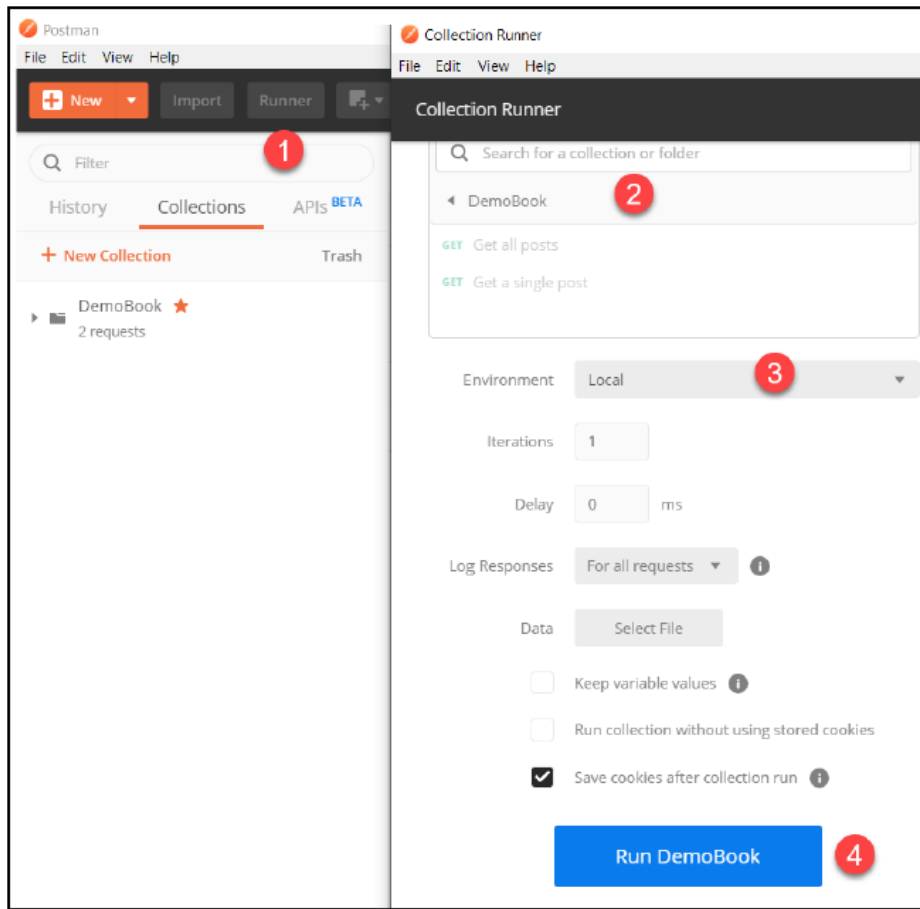
- In the event that one of the tests fails, it will be displayed in red in order to clearly identify it.
- An example of a failed test is shown in the following screenshot:



Executing Postman request tests locally

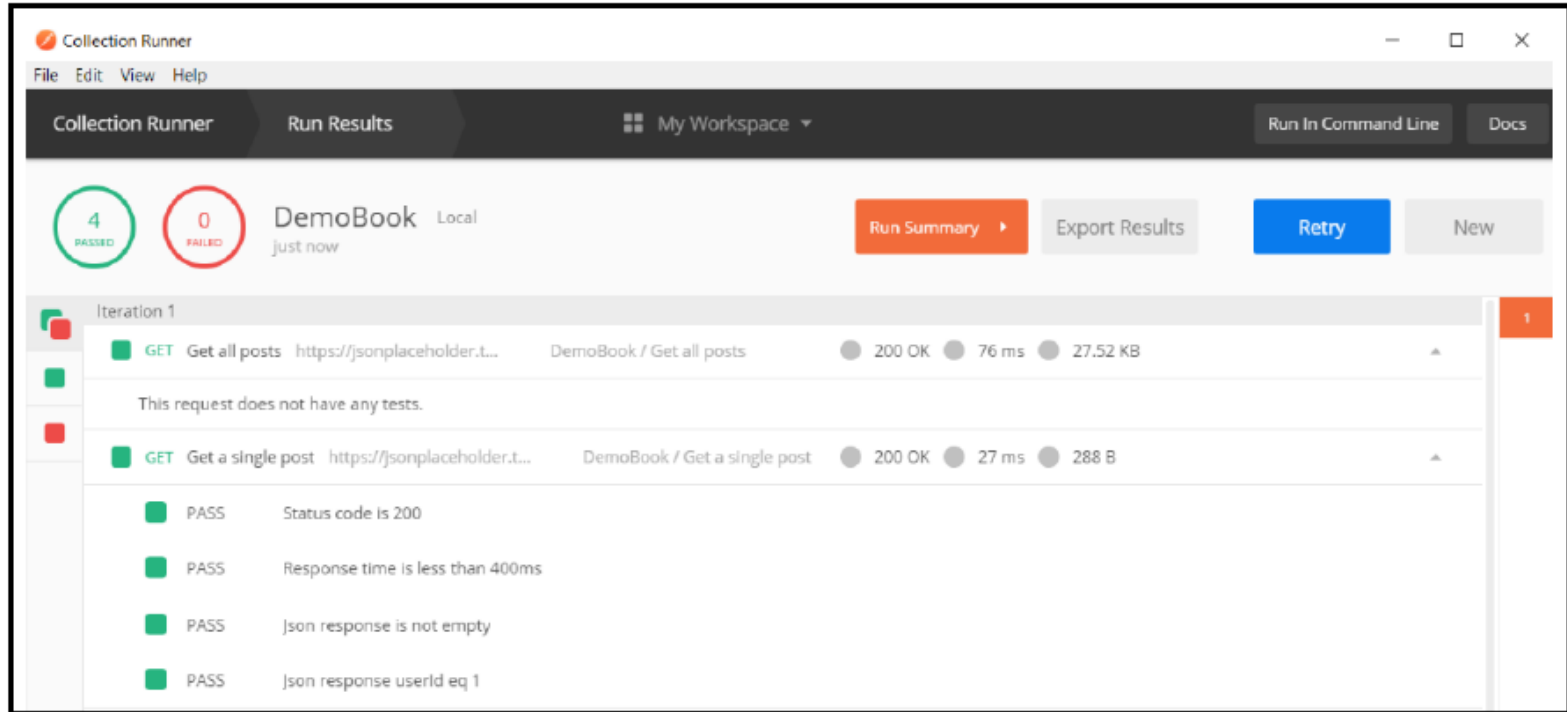
- We have just seen the execution of a Postman request to test an API, but this execution is only for the current request.
- To execute all Postman requests in a collection, use the Postman Collection Runner.
- The Postman Collection Runner is a Postman feature that automatically executes all the requests in a collection in the order in which they have been organized.
- The next two screenshots show the Runner execution steps, in which we choose the collection to execute, the environment, and the number of iterations.
- To start its execution, click on the Run DemoBook button:

Executing Postman request tests locally



Executing Postman request tests locally

- In the Collection Runner screen, see the test execution result of all the requests in the collection, as shown in the following screenshot:



Understanding the Newman concept

- Postman is used locally to test the APIs that was developed or consumed.
- But, what is important in unit, acceptance, and integration tests is that they are automated in order that they are able to be executed within a CI/CD pipeline.
- Postman, as such, is a graphical tool that does not automate itself, but there is another tool called **Newman** that automates tests that are written in Postman.
- Newman is a free command-line tool that has the great advantage of automating tests that are already written in Postman.
- It allows us to integrate API test execution into CI/CD scripts or processes.

Understanding the Newman concept

- In addition, it offers the possibility of generating the results of the tests of reports of different formats (HTML, JUnit, and JSON).
- Newman does not allow to do the following:
 - To create or configure Postman requests as requests that are executed by Newman will be exported from Postman.
 - To execute only one request that is in a collection—it executes all the requests in a collection.

Understanding the Newman concept

- In order to use Newman, need to have installed Node.js and npm.
- Then, to install Newman, must execute the command in Terminal:
- *`npm install -g newman`*

```
>npm install -g newman
\AppData\Roaming\npm\newman -> C:\Users\MikaelKRIEF\AppData\Roaming\npm\node_modules\newman\bin\newman.js
+ newman@4.5.1
added 152 packages from 191 contributors in 13.755s
```

- This command installs the npm newman package and all its dependencies globally, that is, it is accessible on the entire local machine.

Understanding the Newman concept

- Once installed, test its installation by running the *newman --help* command.
- It displays the arguments and options to use, as shown in the following screenshot:

```
>newman --help
Usage: newman [options] [command]

Options:
  -v, --version          output the version number
  -h, --help             output usage information

Commands:
  run [options] <collection> URL or path to a Postman Collection.

To get available options for a command:
  newman [command] -h
```

Preparing Postman collections for Newman

- Newman is Postman's client tool, and in order to work, it needs the configuration of the collections, requests, and environments that was created in Postman.
- So before running Newman, export Postman's collection and environments.
- And this export will serve as Newman's arguments.
- Let's start exporting the DemoBook collection that was created in Postman.

Exporting the collection

- The export of a Postman collection consists of obtaining a JSON file that contains all the settings of this collection and the requests that are inside it.
- It is from this JSON file that Newman will be able to run the same API tests.
- To do this export, we perform the following tasks:
 1. Go to the context menu of the collection that we want to export.
 2. Choose the Export action.
 3. Then, in the window that opens, uncheck the Collection v2.1 (recommended) checkbox.
 4. Finally, validate by clicking on the Export button.

Exporting the collection

The screenshot illustrates the process of exporting a collection in Postman. On the left, the 'DemoBook' collection is selected, and the menu icon (three dots) is highlighted with a red box and a red circle containing the number '1'. The menu is open, showing various options. The 'Export' option, represented by a download icon, is highlighted with a red box and a red circle containing the number '2'. On the right, the 'EXPORT COLLECTION' dialog is shown. It displays the text 'DemoBook will be exported as a JSON file. Export as:' followed by three radio button options: 'Collection v1 (deprecated)', 'Collection v2', and 'Collection v2.1 (recommended)'. The 'Collection v2.1 (recommended)' option is selected, indicated by a red circle with the number '3'. At the bottom right of the dialog, the 'Export' button is highlighted with a red circle and the number '4'.

EXPORT COLLECTION

DemoBook will be exported as a JSON file. Export as:

- ☐ Collection v1 (deprecated)
- ☐ Collection v2
- ☒ Collection v2.1 (recommended)

[Learn more about collection formats](#)

Cancel Export

Exporting the collection

- Clicking on the Export button exports the collection to a JSON format file, DemoBook.postman-collection.json, which is saved in a folder that is dedicated to Newman.
- After exporting the collection, need to export the environment and variable information, because the requests in the collection depend on it.

Exporting the Environments

- Newman's configuration is not easy as the problem is that the Postman requests use variables that are configured in environments.
- Therefore, for this reason export the information from each environment in JSON format, to pass it on as an argument to Newman.
- To export the environments and their variables, perform the following tasks:
 1. We will open the MANAGE ENVIRONMENTS from settings in Postman.
 2. Then, click on the download environment button.

Exporting the Environments

The screenshot shows the 'MANAGE ENVIRONMENTS' dialog box. At the top, there's a header bar with a dropdown menu set to 'No Environment' and a gear icon labeled with a red circle '1'. A red arrow points from this gear icon to the dialog. Inside the dialog, there's a description: 'An environment is a set of variables that allow you to switch the context of your requests. Environments can be shared between multiple workspaces. [Learn more about environments](#)'. Below this, there's a table with two rows: 'Local' and 'QA'. Each row has a 'Share' button, a download icon (labeled with a red circle '2'), and a more options menu. The download icon in the 'Local' row is highlighted with a red box, and two blue arrows point from it to the download icons in the 'QA' row.

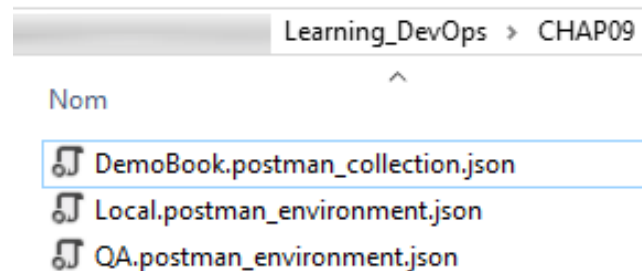
MANAGE ENVIRONMENTS

An environment is a set of variables that allow you to switch the context of your requests. Environments can be shared between multiple workspaces. [Learn more about environments](#)

Local	Share	Download	More
QA	Share	Download	More

Exporting the Environments

- So, for each environment, export their configurations in a JSON file, which was saved in the same folder where the collection was exported.
- Finally, have a folder on the machine that contains three Postman JSON files:
 1. One JSON file for the collection
 2. One JSON file for the Local environment
 3. One JSON file for the QA environment



Running the Newman command line

- After exporting the Postman configuration, run the Newman utility on the local machine.
- To execute Newman, go to Terminal, then to the folder where the JSON configuration files are located, and execute the following command:

```
newman run DemoBook.postman_collection.json -e  
Local.postman_environment.json|
```

Exporting the Environments

- The newman run command takes the JSON file of the collection that was exported as an argument, and a parameter, -e, which is the JSON file of the exported environment.
- Newman will execute the Postman requests from the exported collection.
- It will also use the variables of the exported environment and will also perform the tests written in the request.

Exporting the Environments

```
C:\Windows\System32\cmd.exe

C:\Users\Mikael\Documents\Postman>newman run DemoBook.postman_collection.json -e Local.postman_environment.json
newman

DemoBook

→ Get all posts
GET https://jsonplaceholder.typicode.com/posts [200 OK, 27.64KB, 194ms]

→ Get a single post
GET https://jsonplaceholder.typicode.com/posts/6 [200 OK, 932B, 27ms]
✓ Status code is 200
✓ Response time is less than 400ms
✓ Json response is not empty
✓ Json response userId eq 1
```

	executed	failed
iterations	1	0
requests	2	0
test-scripts	1	0
prerequisite-scripts	0	0
assertions	4	0

```
total run duration: 338ms
total data received: 27.16KB (approx)
average response time: 110ms [min: 27ms, max: 194ms, s.d.: 83ms]
```

Exporting the Environments

- This screenshot shows the result of its execution in case there is an error in the test:

```
C:\Windows\System32\cmd.exe

C:\Users\Mikael\Documents\Postman>newman run DemoBook.postman_collection.json -e qa.postman_environment.json
newman

DemoBook

→ Get all posts
  GET https://jsonplaceholder.typicode.com/posts [200 OK, 27.64KB, 254ms]

→ Get a single post
  GET https://jsonplaceholder.typicode.com/posts/7 [200 OK, 868B, 28ms]
  ✓ Status code is 200
  ✓ Response time is less than 400ms
  ✓ Json response is not empty
  1. Json response userId eq 1
```

	executed	failed
iterations	1	0
requests	2	0
test-scripts	1	0
prerequisite-scripts	0	0
assertions	4	1

total run duration: 412ms
total data received: 27.09KB (approx)
average response time: 141ms [min: 28ms, max: 254ms, s.d.: 113ms]

```
# failure detail
1. AssertionError      Json response userId eq 1
                        expected 1 to equal 10
                        at assertion:3 in test-script
                        inside "Get a single post"
```

Integration of Newman in the CI/CD pipeline process

- Newman is a tool that automates the execution of Postman requests from the command line.
- This will quickly allow to integrate it into a CI/CD pipeline.
- To simplify its integration into a pipeline, go to the first step in the directory that
- contains the JSON files that were exported from Postman, and create an npm configuration file—package.json.

Integration of Newman in the CI/CD pipeline process

This will have the following content:

```
{
  "name": "postman",
  "version": "1.0.0",
  "description": "postmanrestapi",
  "scripts": {
    "testapilocal": "newman run DemoBook.postman_collection.json -e
Local.postman_environment.json -r junit,cli --reporter-junit-export result-
tests-local.xml",
    "testapiQA": "newman run DemoBook.postman_collection.json -e
QA.postman_environment.json -r junit,cli --reporter-junit-export result-
tests-qa.xml"
  },
  "devDependencies": {
    "newman": "^4.5.1"
  }
}
```

Integration of Newman in the CI/CD pipeline process

- In the scripts section, put the two scripts that will be executed with the command lines.
- And add to them the -r argument, which allows the output of the command with reporting in JUnit format.
- In the DevDependencies section, indicate that there is a need for the Newman package.
- To show Newman's integration into a CI/CD pipeline, Azure Pipelines—an
- Azure DevOps service is used in this case.

Static Code Analysis with SonarQube

Introduction

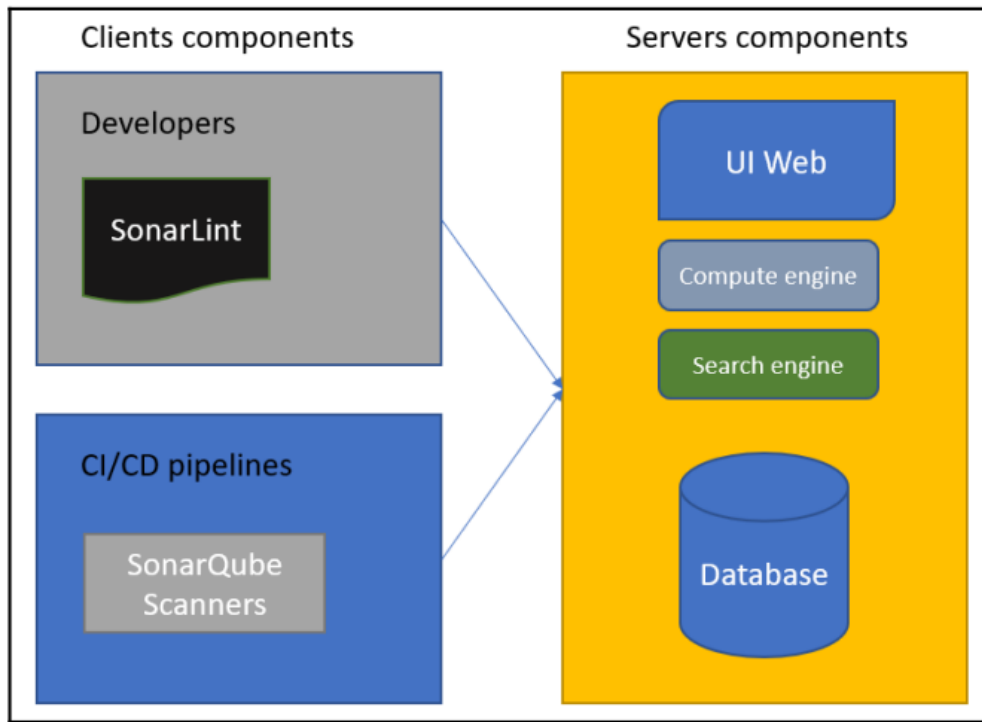
- Static code analysis is conducted with a well-known tool called SonarQube.
- SonarQube is an open source tool from SonarSource and is written in Java.
- It performs static code analysis to verify the quality and security of an application's code.
- SonarQube is designed for developer teams and provides them with a dashboard and reports that are customizable so that they can present the quality of the code in their applications.
- Allows for the analysis of static code in a multitude of languages (over 25), such as PHP, Java, .NET, JavaScript, Python, and so on.

Introduction

- Apart from code analysis with security issues, code smell, and code duplication, SonarQube also provides code coverage for unit tests.
- Finally, SonarQube integrates very well into CI/CD pipelines.
- Helps to automate code analysis during developer code commits.
- Reduces the risk of deploying an application that has security vulnerabilities or code complexity that is too high.

Overview of the SonarQube Architecture

- SonarQube is a client-server tool, and its architecture is composed of artifacts on the server side and also on the client side.



Overview of the SonarQube Architecture

- The components that make up SonarQube on the server side are as follows:
 - A SQL Server, MySQL, Oracle, or PostgreSQL database that contains all the analysis data.
 - A web application that displays dashboards.
 - The compute engine, which is in charge of retrieving the analysis and processes.
 - It puts them in the database.
 - A search engine built with Elasticsearch.

Overview of the SonarQube Architecture

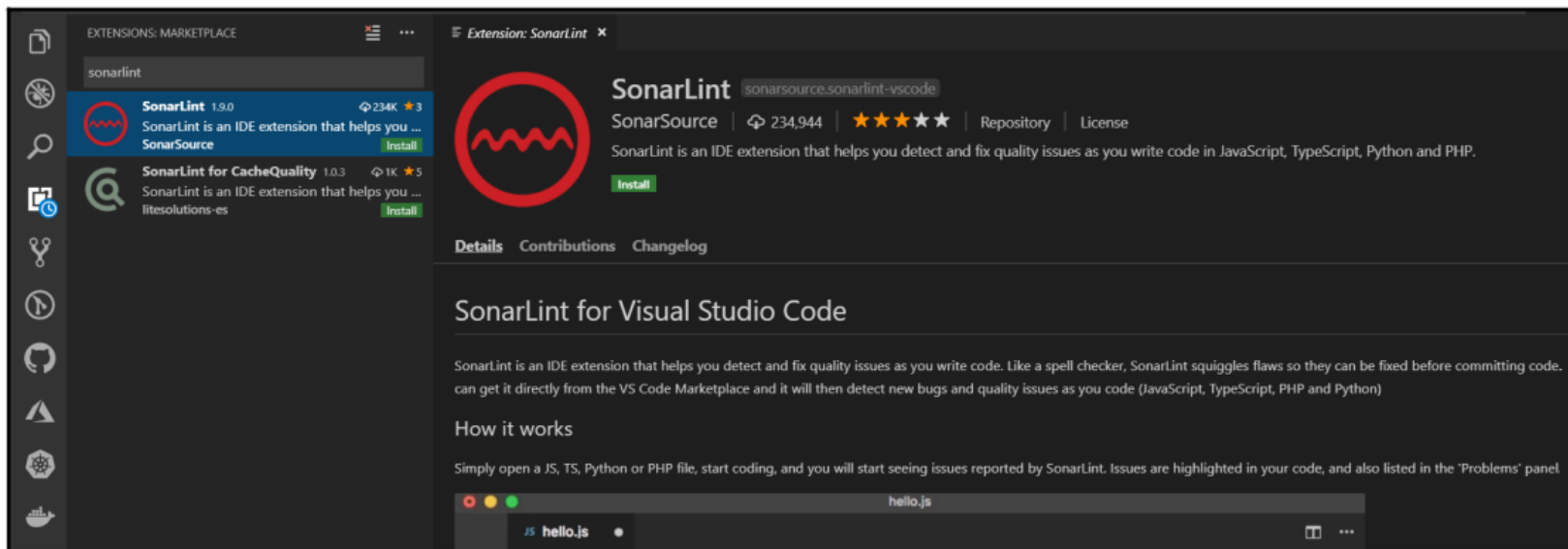
- The client-side components are as follows:
 - The scanner, which scans the source code of the applications and sends the data to the compute engine.
 - The scanner is usually installed on the build agents that are used to execute CI/CD pipelines.
 - SonarLint is a tool that's installed on developers' workstations for real-time analysis.

Real-time analysis with SonarLint

- Developers using SonarQube in a continuous integration context face the problem of having to wait too long before they get the results of the SonarQube analysis.
- They must commit their code and wait for the end of the continuous integration pipeline before they get the results of the code analysis.
- To address this problem and improve the daily lives of developers, SonarSource – the editor of SonarQube – provides another tool, **SonarLint**.
- SonarLint allows realtime code analysis.
- SonarLint is a free and open source tool.
- SonarLint is available for Eclipse, IntelliJ, Visual Studio, and Visual Studio Code IDEs.

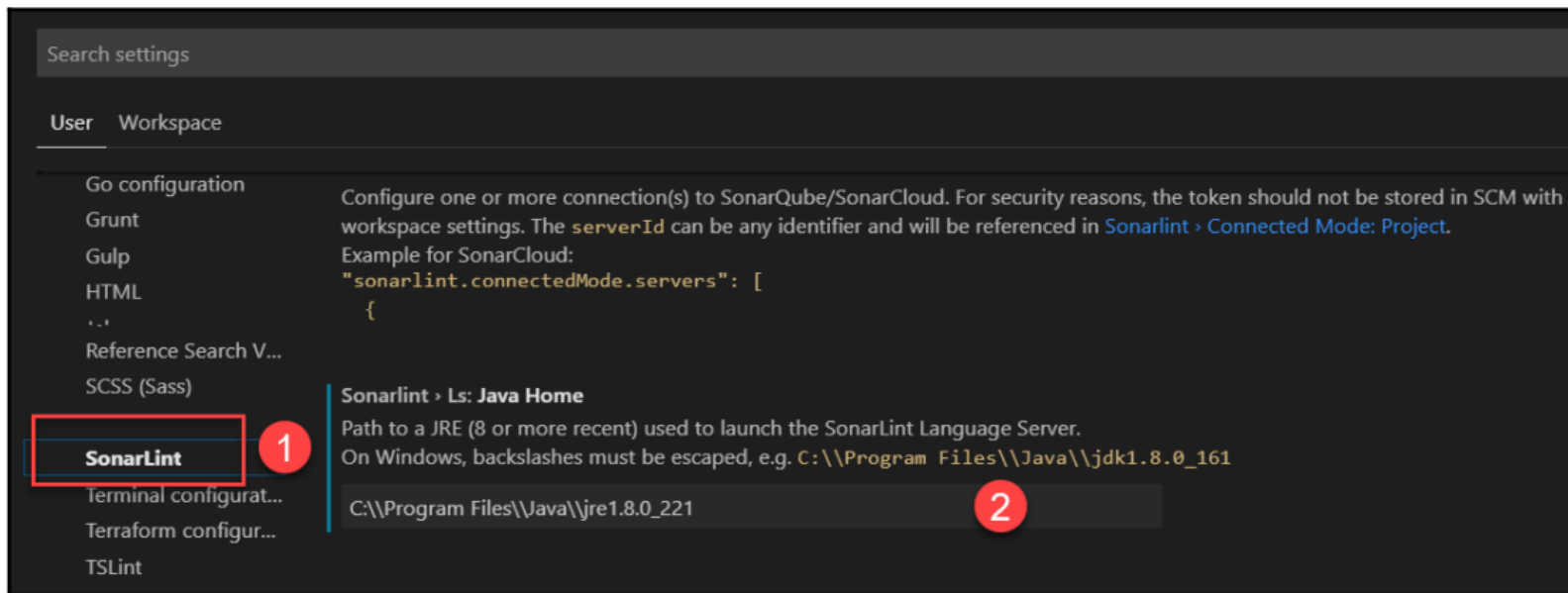
Real-time analysis with SonarLint

- To learn more about the concrete use of SonarLint, follow these steps:
 1. In Visual Studio Code, install the SonarLint extension by going to the following page on the Azure Marketplace:



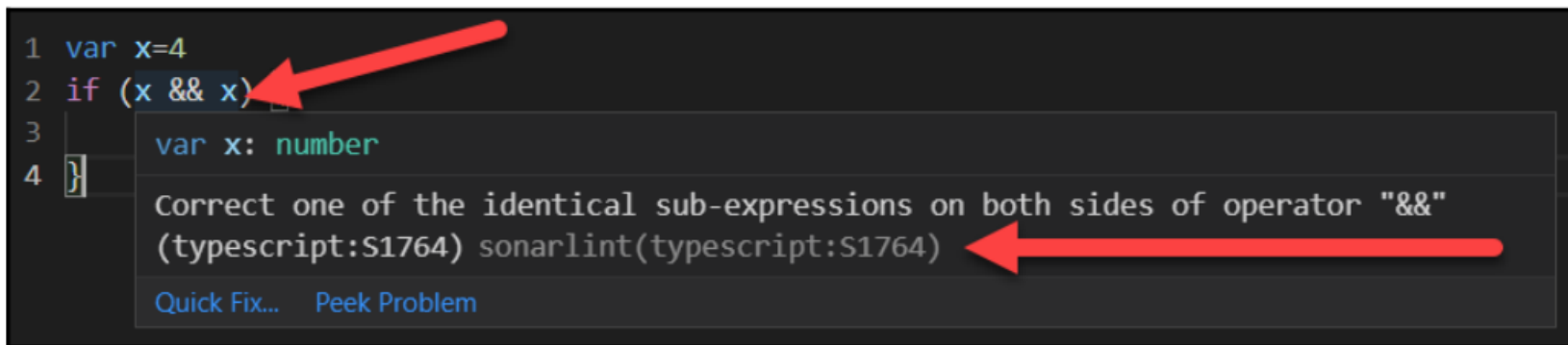
Real-time analysis with SonarLint

- To learn more about the concrete use of SonarLint, follow these steps:
2. Then, in Visual Studio Code, in the User settings, configure the extension with the installation path of the JRE, as follows :



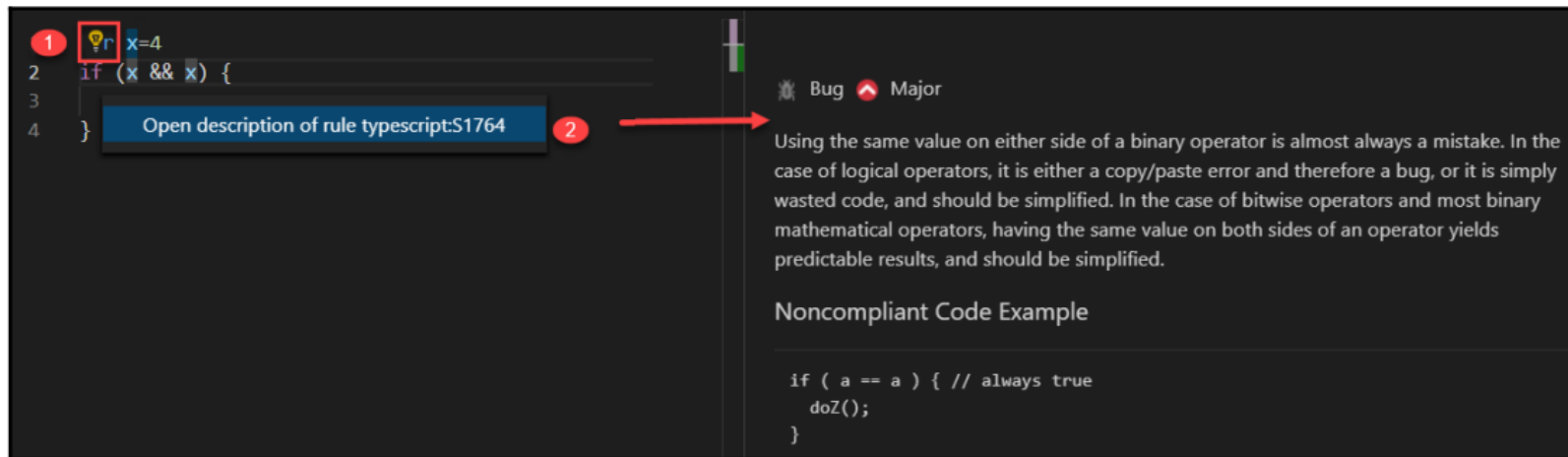
Real-time analysis with SonarLint

- To learn more about the concrete use of SonarLint, follow these steps:
3. In the project, create a tsApp folder. Inside that folder, create an app.ts file that contains the code of our application:
 4. Note that in this SonarLint code, it states that the code is not correct, as shown in the following screenshot:



Real-time analysis with SonarLint

- To learn more about the concrete use of SonarLint, follow these steps:
5. SonarLint allows to learn more about this error by displaying detailed information regarding the error and how to fix it, as shown in the following screenshot:



Real-time analysis with SonarLint

- SonarLint and its integration with various IDEs allows us to detect static code errors in real time as soon as possible, that is, while the developer writes their code and before they commit it in source control version.

Executing SonarQube in Continuous Integration

- Look at how to perform code analysis during continuous integration to ensure that each time a code commit is made, check the application code that's provided by all team members.
- To integrate SonarQube into a Continuous Integration process, perform the following actions:
 - 1. Configure SonarQube by creating a new project.
 - 2. Create and configure a continuous integration build in Azure Pipelines.

Configuring SonarQube

- SonarQube's configuration consists of creating a new project and retrieving an identification token.
- To create a new project, follow these steps:
 1. Click on the Create new project link on the dashboard.
 2. Then, in the form, enter a unique demobook key and a name for this project as demo-book.

Configuring SonarQube

3. To validate this, click on the Set Up button to create the project. These steps are shown in the following screenshot:

Sort by: Name

Search by project name or key

Once you analyze some projects, they will show up here.

Here is how you can analyse new projects

Create new project

Create new project

Project key*

demobook

Up to 400 characters. All letters, digits, dash, underscore, period

Display name*

demo-book

Up to 255 characters

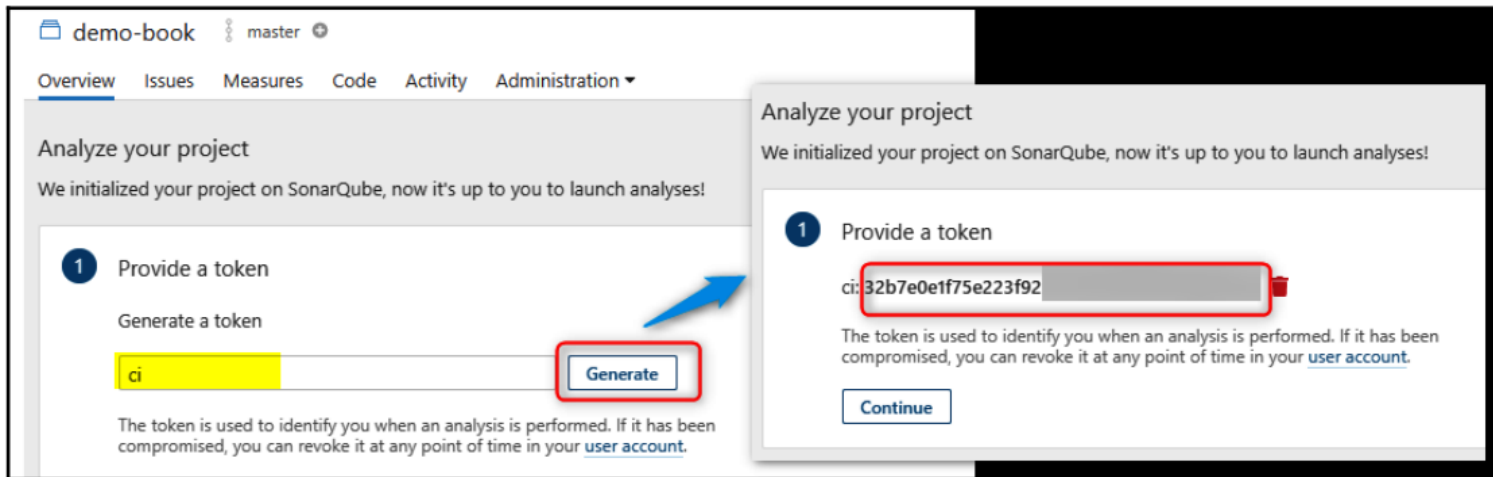
Set Up

Configuring SonarQube

- As soon as the project is created, the SonarQube assistant proposes to create a token (unique key) that will be used for analysis.
- To generate and create this token, follow these steps:
 1. In the input, type a desired token name.
 2. Then, validate it by clicking on the Generate button.
 3. The unique key is then displayed on the screen. This key is our token, and we must keep it safe. The following screenshot shows the steps for generating the token:

Configuring SonarQube

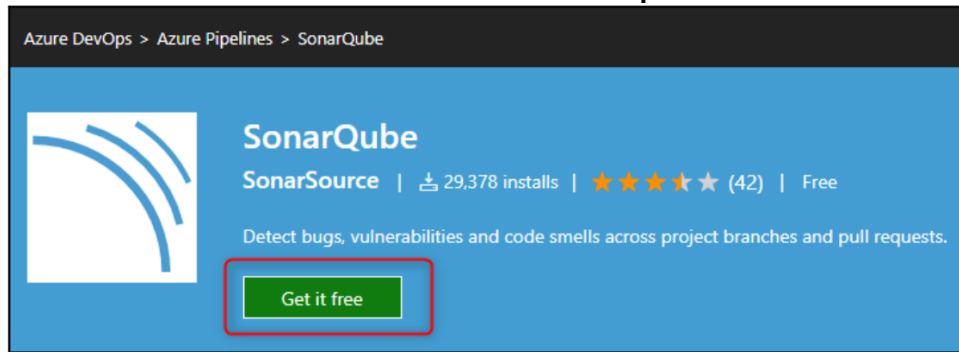
- Generating and creating the token:



- The configuration of SonarQube with our new project is complete.
- Now configure our CI pipeline to perform the SonarQube analysis.

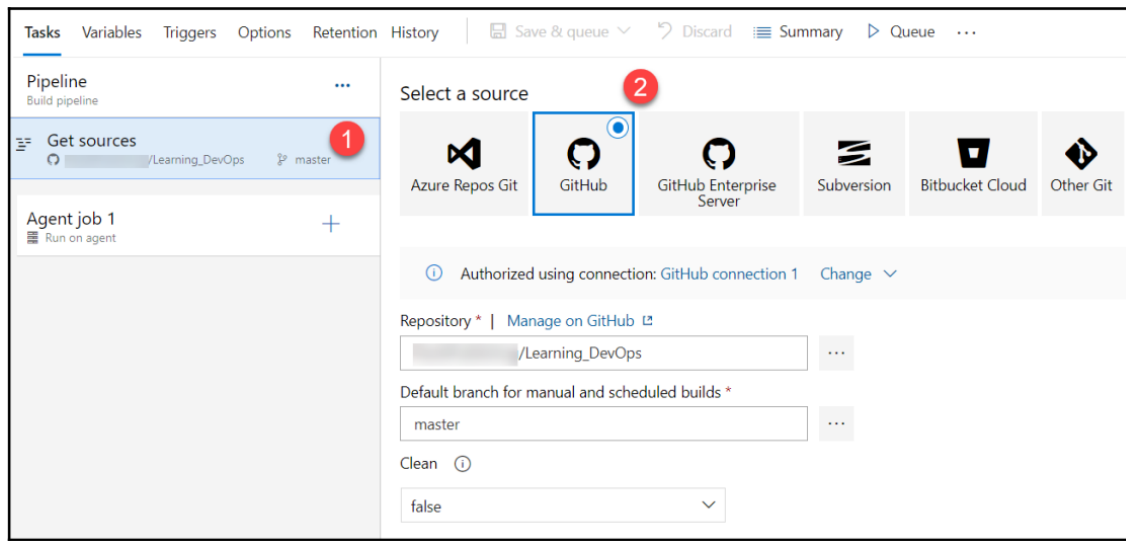
Creating a CI pipeline for SonarQube in Azure Pipelines

- To illustrate the integration of a SonarQube analysis into a CI pipeline, use Azure Pipelines.
- The application that will be used as an example has been developed in Node.js, which is a simple calculator that contains some methods, including unit test methods.
- The following screenshot shows the header and button to install the extension from the Visual Studio Marketplace:



Creating a CI pipeline for SonarQube in Azure Pipelines

- Once the extension has been installed, configure CI build.
- In Azure Pipelines, create a new build definition:
 1. In the Get Sources tab, select the repository and the branch that contains the source code of the application, as shown in the following screenshot:



Creating a CI pipeline for SonarQube in Azure Pipelines

2. Then, in the Tasks tab, configure the schedule of the tasks, as follows:

The screenshot displays the Azure Pipelines configuration interface for the 'Prepare analysis on SonarQube' task. The left sidebar shows the task list with 'Prepare analysis on SonarQube' highlighted and numbered 1. The main configuration area shows the following settings:

- Display name ***: Prepare analysis on SonarQube
- SonarQube Server Endpoint ***: SonarQube Demo (highlighted with a red box and a blue arrow pointing to the 'Add SonarQube service connection' dialog)
- Choose the way to run the analysis ***: Use standalone scanner (highlighted with a red box)
- Mode ***: Manually provide configuration
- Project Key ***: demobook
- Project Name**: demo-book
- Project Version**: \$(Build.BuildNumber)
- Sources directory root ***: CHAP10/TsApp2/lib/

The 'Add SonarQube service connection' dialog box is open, showing the following details:

- Connection name**: SonarQube
- Server Url**: http://168.63...
- Token**: *****
- Allow all pipelines to use this connection**: checked

Creating a CI pipeline for SonarQube in Azure Pipelines

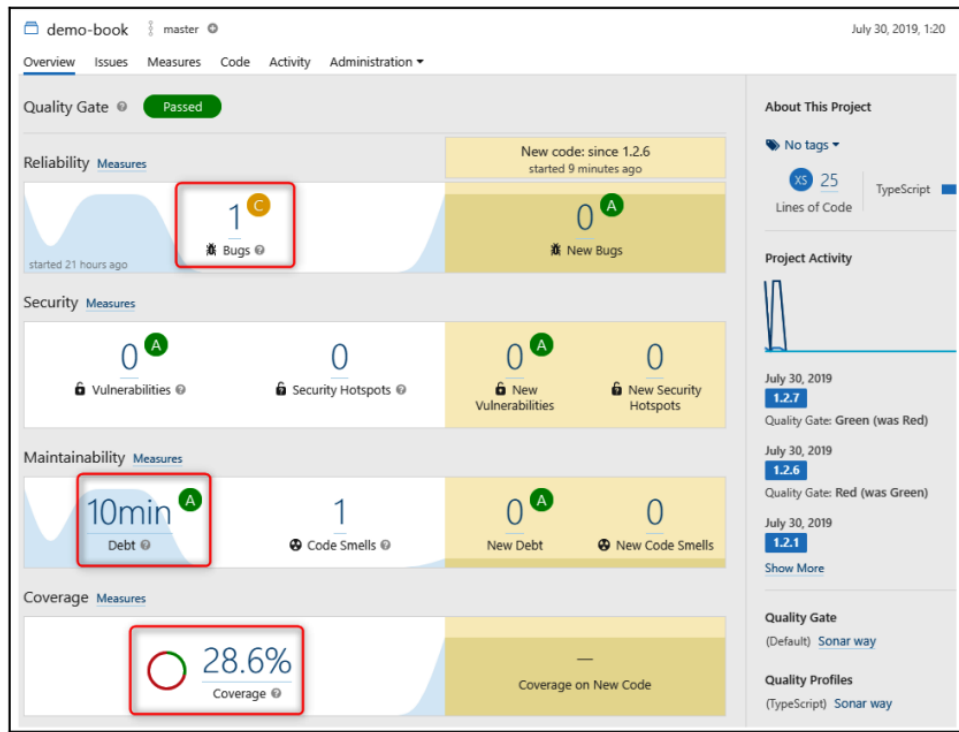
- Here are the details of the configuration of these tasks:
 1. The Prepare analysis for SonarQube task includes configuring SonarQube with the following:
 - An endpoint service, which is the connection to SonarQube with its URL and token that we generated previously in the SonarQube configuration
 - The key and name of the SonarQube project
 - The version number of the analysis

Creating a CI pipeline for SonarQube in Azure Pipelines

- Here are the details of the configuration of these tasks:
- 2. Then, we build and execute the unit tests of the application with npm build and npm test.
- 3. The Run Code Analysis task retrieves the test results, analyzes the TypeScript code of our application, and sends the data from the analysis to the SonarQube server.
- Then, we save, start executing the CI build, and wait for it to finish.

Creating a CI pipeline for SonarQube in Azure Pipelines


- The SonarQube dashboard has been updated with the code analysis, as shown in the following screenshot:



Summary

- How to analyze the static code of an application using SonarQube.
- This analysis can detect and prevent code syntax problems, vulnerabilities in the code, and also indicate the code coverage provided by unit tests.
- SonarLint allows developers to check their code in real time as they write their code.
- Details of the configuration of SonarQube and its integration into a continuous integration process to ensure continuous analysis that will be triggered at each code commit of a team member.

Security and Performance Tests

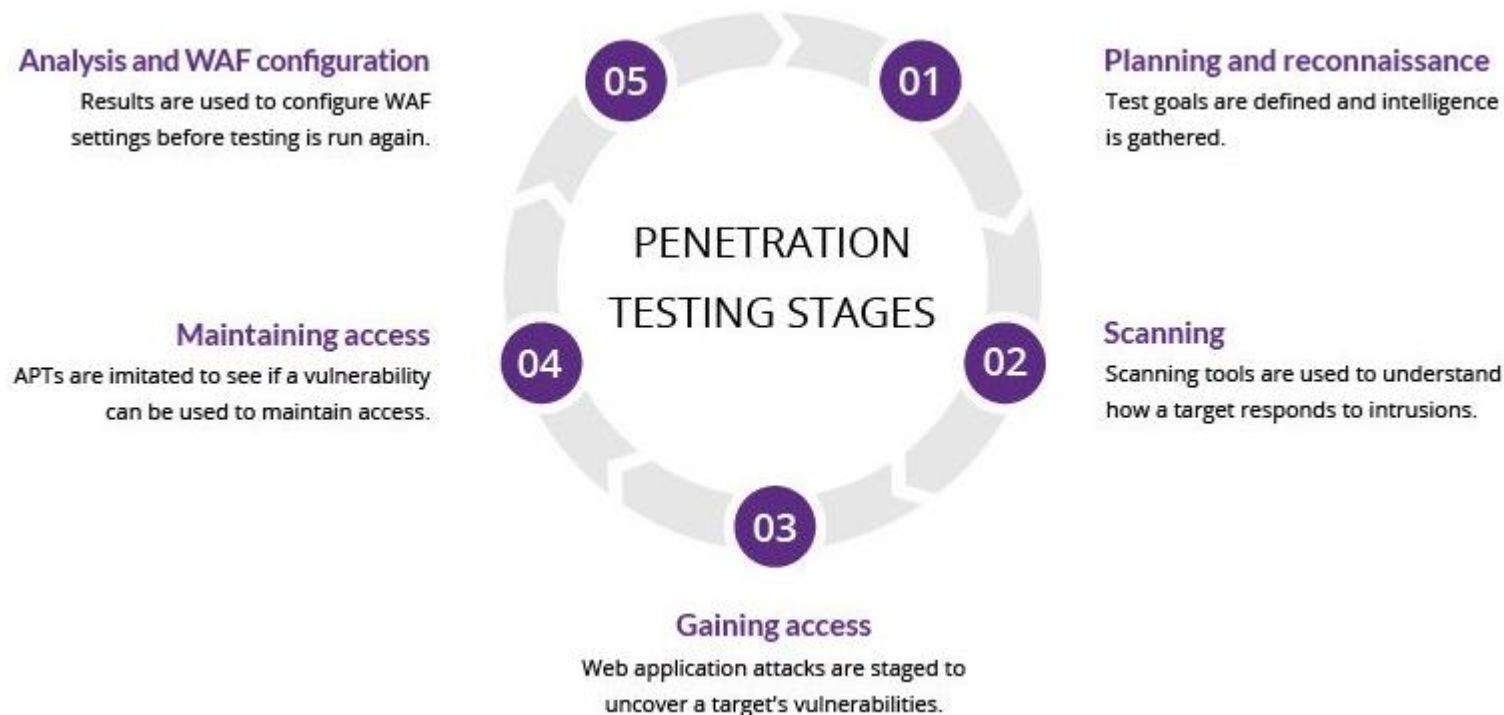


Introduction

- Perform security and penetration tests on a web application using the ZAP tool based on the OWASP recommendations.
- Penetration testing (or pen testing) is a security exercise where a cybersecurity expert attempts to find and exploit vulnerabilities in a computer system.
- The purpose of this simulated attack is to identify any weak spots in a system's defenses which attackers could take advantage of.
- Pen testing involves the attempted breaching of any number of application systems, (e.g., application protocol interfaces (APIs), frontend/backend servers) to uncover vulnerabilities, such as unsanitized inputs that are susceptible to code injection attacks.

Penetration testing stages

- The pen testing process can be broken down into five stages:



Penetration testing stages

- The pen testing process can be broken down into five stages:

1. **Planning and reconnaissance:** The first stage involves:

- Defining the scope and goals of a test, including the systems to be addressed and the testing methods to be used.
- Gathering intelligence (e.g., network and domain names, mail server) to better understand how a target works and its potential vulnerabilities.

2. **Scanning:** The next step is to understand how the target application will respond to various intrusion attempts. This is typically done using:

- **Static analysis:** Inspecting an application's code to estimate the way it behaves while running. These tools can scan the entirety of the code in a single pass.

Penetration testing stages

2. **Scanning:** The next step is to understand how the target application will respond to various intrusion attempts. This is typically done using:

- **Dynamic analysis:** Inspecting an application's code in a running state. This is a more practical way of scanning, as it provides a real-time view into an application's performance.

3. Gaining Access:

- This stage uses web application attacks, such as cross-site scripting, SQL injection and backdoors, to uncover a target's vulnerabilities.
- Testers then try and exploit these vulnerabilities, typically by escalating privileges, stealing data, intercepting traffic, etc., to understand the damage they can cause.

Penetration testing stages

4. Maintaining access:

- The goal of this stage is to see if the vulnerability can be used to achieve a persistent presence in the exploited system long enough for a bad actor to gain in-depth access.
- The idea is to imitate advanced persistent threats, which often remain in a system for months in order to steal an organization's most sensitive data.

5. **Analysis:** The results of the penetration test are compiled into a report:

- Specific vulnerabilities that were exploited
- Sensitive data that was accessed
- The amount of time the pen tester was able to remain in the system undetected.

Penetration Testing Methods

- **External Testing:**

- External penetration tests target the assets of a company that are visible on the internet, e.g., the web application itself, the company website, and email and domain name servers (DNS).
- The goal is to gain access and extract valuable data.

- **Internal Testing:**

- In an internal test, a tester with access to an application behind its firewall simulates an attack by a malicious insider.
- This isn't necessarily simulating a rogue employee.
- A common starting scenario can be an employee whose credentials were stolen due to a phishing attack.

Penetration Testing Methods

- **Blind Testing:**

- In a blind test, a tester is only given the name of the enterprise that's being targeted.
- This gives security personnel a real-time look into how an actual application assault would take place.

- **Double-blind Testing:**

- In a double blind test, security personnel have no prior knowledge of the simulated attack.
- As in the real world, they won't have any time to shore up their defenses before an attempted breach.

Penetration Testing Methods

- Targeted Testing:
 - In this scenario, both the tester and security personnel work together and keep each other appraised of their movements.
 - This is a valuable training exercise that provides a security team with real-time feedback from a hacker's point of view.

Performance Testing

- Performance Testing is a software testing process used for testing the speed, response time, stability, reliability, scalability, and resource usage of a software application under a particular workload.
- The main purpose of performance testing is to identify and eliminate the performance bottlenecks in the software application.
- It is a subset of performance engineering and is also known as “Perf Testing”.
- The focus of Performance Testing is checking a software program’s
 1. **Speed** : Determines whether the application responds quickly

Performance Testing

- The focus of Performance Testing is checking a software program's
 2. **Scalability** : Determines the maximum user load the software application can handle.
 3. **Stability** : Determines if the application is stable under varying loads.

Types of Performance Testing

- **Load Testing:**
 - Checks the application's ability to perform under anticipated user loads.
 - The objective is to identify performance bottlenecks before the software application goes live.
- **Stress Testing:**
 - Involves testing an application under extreme workloads to see how it handles high traffic or data processing.
 - The objective is to identify the breaking point of an application.

Types of Performance Testing

- **Endurance Testing:**
 - This is done to make sure the software can handle the expected load over a long period of time.
 - Spike Testing: This tests the software's reaction to sudden large spikes in the load generated by users.
- **Volume Testing:**
 - Under Volume Testing large no. of. Data is populated in a database, and the overall software system's behavior is monitored.
 - The objective is to check software application's performance under varying database volumes.

Types of Performance Testing

- Scalability Testing:
 - The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load.
 - It helps plan capacity addition to your software system.

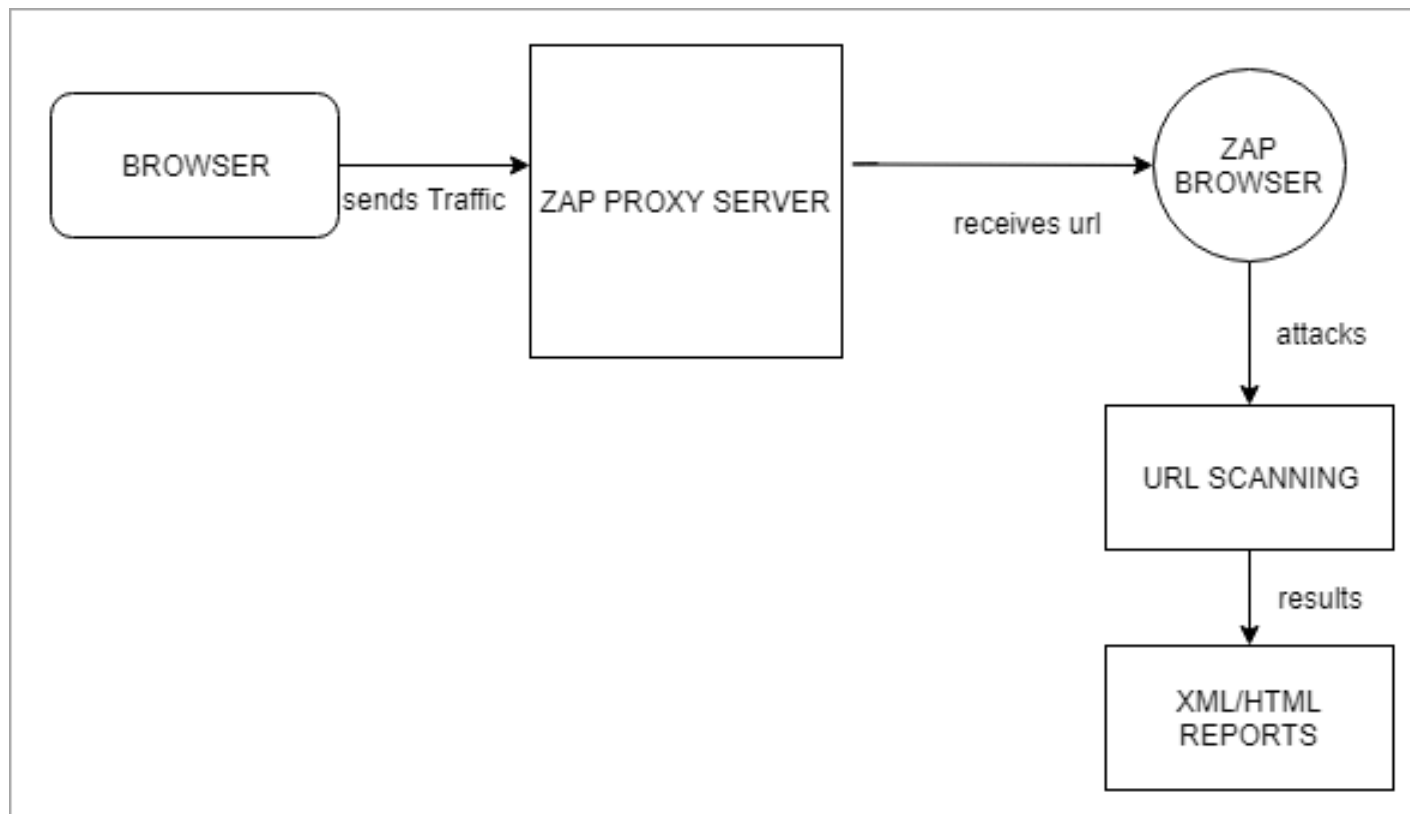
OWASP ZAP

- Penetration testing helps in finding vulnerabilities before an attacker does.
- OSWAP ZAP is an open-source free tool and is used to perform penetration tests.
- The main goal of Zap is to allow easy penetration testing to find the vulnerabilities in web applications.
- Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project (OWASP).
- ZAP is designed specifically for testing web applications and is both flexible and extensible.

OWASP ZAP

- ZAP advantages:
 - Zap provides cross-platform i.e. it works across all OS (Linux, Mac, Windows)
 - Zap is reusable
 - Can generate reports
 - Ideal for beginners
 - Free tool

Working of ZAP



ZAP Terminologies

- Perform security and penetration tests on a web application using the ZAP tool based on the OWASP recommendations.
- ZAP is a 'man-in-the-middle proxy'.
- This means that it runs behind the browser, but before the audited application.
- All information exchanged between the browser and the application therefore first passes through ZAP.



ZAP Terminologies

- **Active Scan:** Active scanning seeks out potential vulnerabilities using known attacks. It's worth noting that Active Scan can only find certain vulnerabilities. Errors in application logic cannot be found by any active or automatic vulnerability scan. This is only possible during a manual audit.
- **Passive Scan:** ZAP by default scans all HTTP requests and responses sent and received from the application. Passive scanning doesn't affect their content. In this case, we can additionally add tags or alerts which will inform us about potential errors. This is enabled by default, but - as with most features - it can be configured.

ZAP Terminologies

- **Spider:** Spider is a crawler, a tool that allows you to discover and map all the links available in the application. The list of discovered links is later saved and can be used to discover additional information about the audited application or for further passive or active scans.
- **Fuzzer:** This is a technique that involves sending a lot of incorrect or unexpected data to the tested application. OWASP ZAP allows fuzzing. We can choose one of the built-in payloads, download those provided by the ZAP community and available in add-ons, or create our own ones.

ZAP Terminologies

- **API:** ZAP provides an API that allows other programs to interact with it. It accepts JSON, HTML, and XML formats. ZAP presents a simple page where we can see the functionality of the API. By default, only the machine on which ZAP is running can connect to the API, but you can allow other machines to contact it in the configuration options.
- **Authentication:** If the application under attack requires authentication, it can be configured. ZAP supports different types of authentication methods. The list includes manual authentication, form-based authentication, JSON or HTTP/NTLM-based authentication, and script-based authentication.

ZAP Terminologies

- **Session:** Session simply means to navigate through the website to identify the area of attack. For this purpose, any browser like Mozilla Firefox can be used by changing its proxy settings. Or else we can save zap session as .session and can be reused.
- **Context:** It means a web application or a set of URLs together. The context created in the ZAP will attack the specified one and ignore the rest, to avoid too much data.
- **Alerts:** Website vulnerabilities are flagged as high, medium and low alerts.

Running performance tests with Postman

- Among the tests that need to be done to guarantee the quality of applications are functional, code analysis, and security tests.
- But there are also performance tests.
- The purpose of performance testing isn't to detect bugs in applications; it's to ensure that the application (or API) responds within an acceptable time frame to provide a good user experience.

Running performance tests with Postman

- The performance of an application is determined by metrics such as the following:
 1. Its response time
 2. The use of resources (CPU, RAM, and network)
 3. The error rate
 4. The number of requests per second

Running performance tests with Postman

- Performance tests are divided into several types of tests, such as load tests, stress tests, and scalability tests.
- There are many tools available to perform performance tests.
- Among them Postman is one such tool.
- Postman is not really a dedicated tool for performance testing, especially since it focuses mainly on APIs and not on monolithic web applications.
- However, Postman can provide a good indication of the performance of the API.

Running performance tests with Postman

- When executing a request that tests an API in a unitary way, Postman provides the execution time of that API.

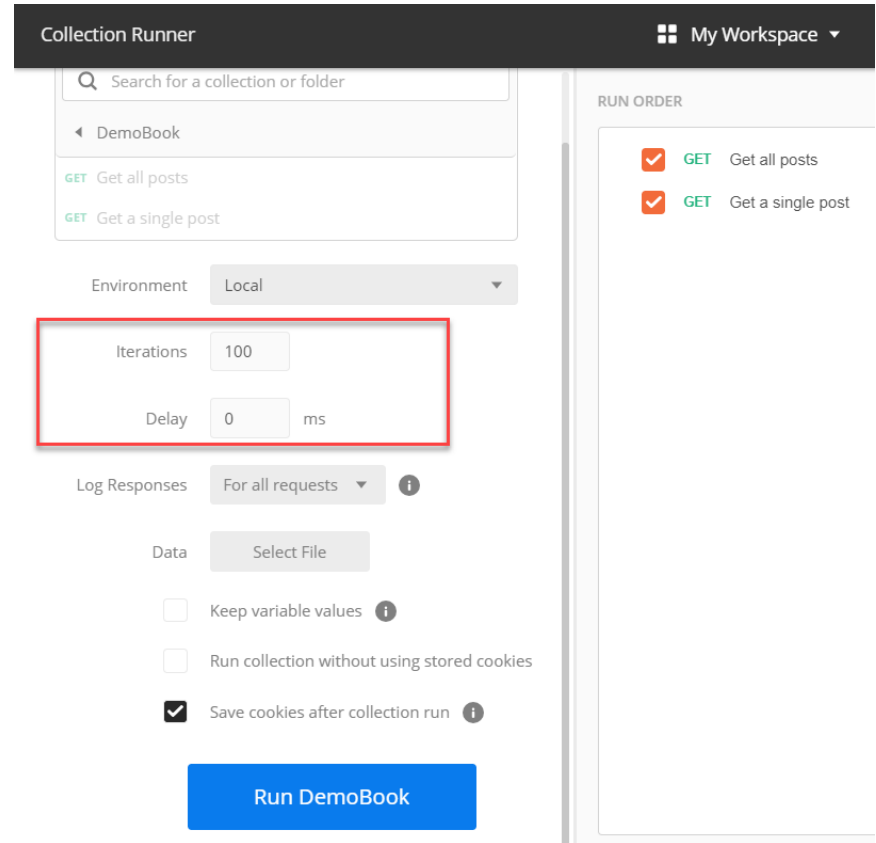
The screenshot displays the Postman application interface. On the left, the 'Collections' tab is active, showing a collection named 'DemoBook' with 2 requests. The selected request is 'GET Get a single employee data'. The main panel shows the request details for 'GET Get a single employee data' with the URL 'http://dummy.restapiexample.com/api/v1/employee/{{EmployeeID}}'. The 'Send' button is visible. Below the URL bar, the 'Params' tab is selected, showing a table for 'Query Params'.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

At the bottom of the main panel, the response status is 'Status: 200 OK'. The 'Time' is '390ms', which is highlighted with a red box. Other response details include 'Size: 469 B' and a 'Save Response' button. The bottom of the interface shows the response body in 'Pretty' format, displaying '1 false'.

Running performance tests with Postman

- In addition, in Postman's Collection Runner, it is possible to execute all the requests of a collection, indicating the number of iterations.
- This simulates several connections that call the API, and it is also where the execution time rendered by Postman becomes very interesting.



Running performance tests with Postman

- And the following screenshot shows the results of the Collection Runner:

The screenshot displays the Postman Collection Runner interface. On the left, the 'Collection Runner' panel shows the 'DemoBook' collection with two requests: 'Get all employee' and 'Get a single employee data'. The 'Iterations' field is set to 100, and the 'Delay' is 0 ms. The 'Run DemoBook' button is at the bottom. On the right, the 'Run Results' panel shows the test results for 'DemoBook' at 2:00 PM. The results indicate 200 PASSED and 0 FAILED. The test results are organized into iterations, showing the status, response time, and size for each request.

Collection Runner

Choose a collection or folder

Search for a collection or folder

DemoBook

GET Get all employee

GET Get a single employee data

Environment: Local

Iterations: 100

Delay: 0 ms

Log Responses: For all requests

Data: Select File

☐ Keep variable values

☐ Run collection without using stored cookies

☒ Save cookies after collection run

Run DemoBook

Collection Runner **Run Results** My Workspace

200 PASSED **0 FAILED** **DemoBook** Local **4** **Run Summary**

Iteration 1

GET Get all employee http://dummy.restapiexa... DemoBook / Get all employee 200 OK 822 ms 576.367 KB

This request does not have any tests.

GET Get a single employee data http://dummy.restapiexa... ...Book / Get a single employee data 200 OK 271 ms 100 B

PASS Status code is 200

PASS Response time is less than 400ms

PASS Json response is not empty

PASS Json response employee_age eq 182

Iteration 2

GET Get all employee http://dummy.restapiexa... DemoBook / Get all employee 200 OK 335 ms 576.367 KB

This request does not have any tests.

GET Get a single employee data http://dummy.restapiexa... ...Book / Get a single employee data 200 OK 292 ms 100 B

PASS Status code is 200

PASS Response time is less than 400ms

PASS Json response is not empty

PASS Json response employee_age eq 182

Iteration 3

Summary

- Uses of ZAP.
- It is a tool developed by the OWASP community to automate the execution of web application security tests.
- Postman can provide information on API performance.

Thank You
