Project – 1

Exploring Chrome Developer Tools: An In-Depth Internship Program on Google Chrome's Inspect Feature

Submitted By:

Nishigandha Patil

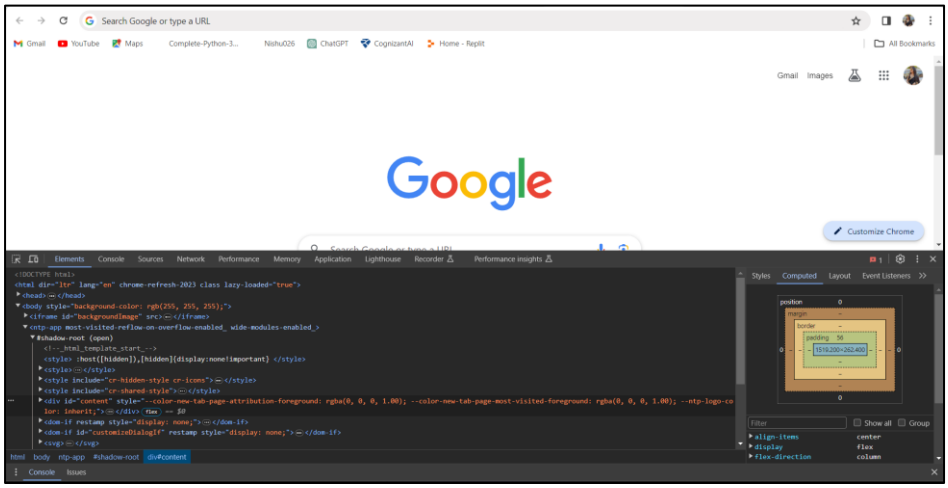
INDEX

INTRODUCTION TO CHROME'S DEVELOPER TOOLS

What is Chrome Developer Tools?

Chrome's Developer Tools are a set of utilities integrated directly into the Google Chrome browser. DevTools allows developers to build, debug, and optimize web pages and applications. These tools provide a suite of functionalities that enable developers to view and change/manipulate the DOM, change a page's styles (CSS) in a preview environment, and work with JavaScript by allowing you to debug, view messages and run JavaScript code in the console and much more, all from within the browser itself.

- Here's what you'll see when you open Developer Tools:



Accessing the Developer Tools:

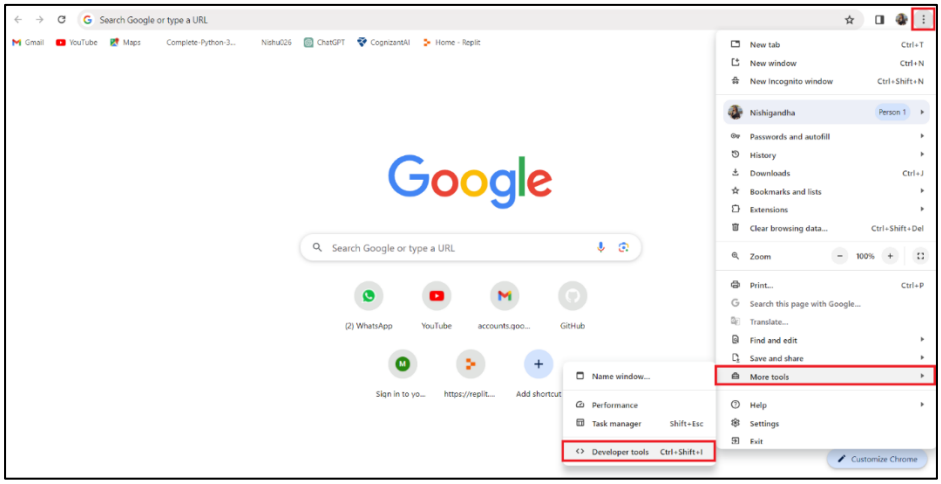
You can access Developer Tools in many different ways:

1. The keyboard Shortcuts:

Sr. No	Action	Mac OS	Windows/Linux
1	To open whatever panel, you used last	Command+Option+I	F12 or Control+Shift+I
2	To open "Elements panel"	Command+Shift+C or Command+Option+C	Control+Shift+C
3	To open "Console Panel"	Command+Option+J	Control+Shift+J

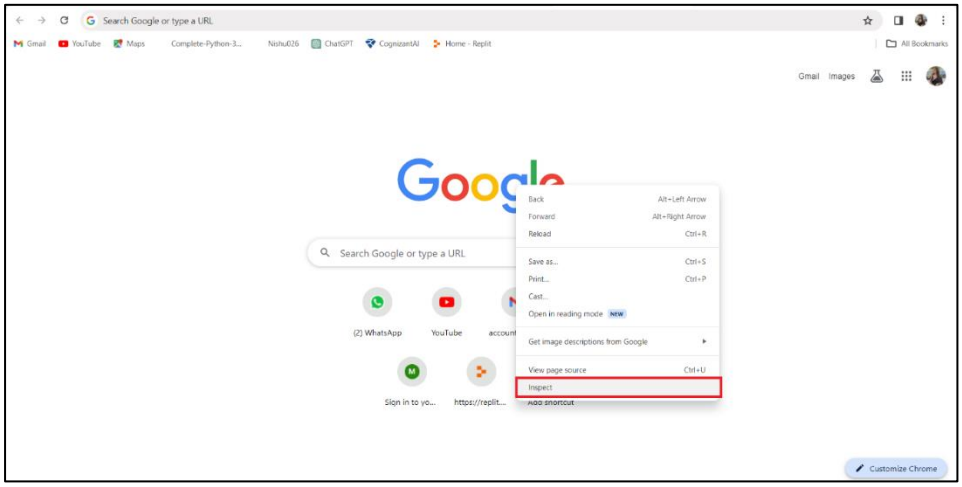
2. From the Chrome menu:

Open the Chrome menu and go to “More Tools” > “Developer Tools.”



3. Right-click on element > “Inspect”:

Finally, you can right-click on an element on the web page and select “Inspect” to open Developer Tools.



PANELS IN DEVTOOLS

Chrome's Developer Tools consist of several panels, each offering specific functionalities to aid in web development, debugging, and optimization etc. Here is an overview of the main panels:

1. ELEMENTS PANEL:

The Elements panel is a powerful tool for inspecting and manipulating the structure and styles of a web page. It provides a visual representation of the Document Object Model (DOM) and allows developers to view and edit the HTML and CSS of a webpage in real-time.

➢ **Features:**

A. Viewing DOM:

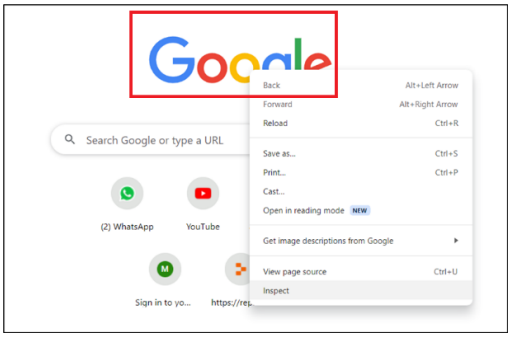
- **What is DOM:**

DOM stands for Document Object Model. It is a programming interface for documents. DOM is a tree-like structure that represents the elements such as paragraphs, headings, images, etc. and their relationships within an HTML page. The DOM represents the document as nodes and objects. A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated.

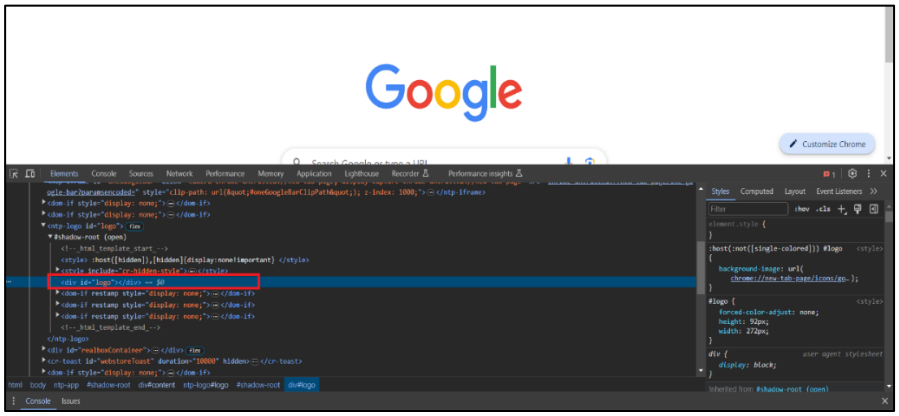
1. Viewing DOM Nodes:

a) When you're interested in a particular DOM node, you can right-click on that node/element and "Inspect" it.

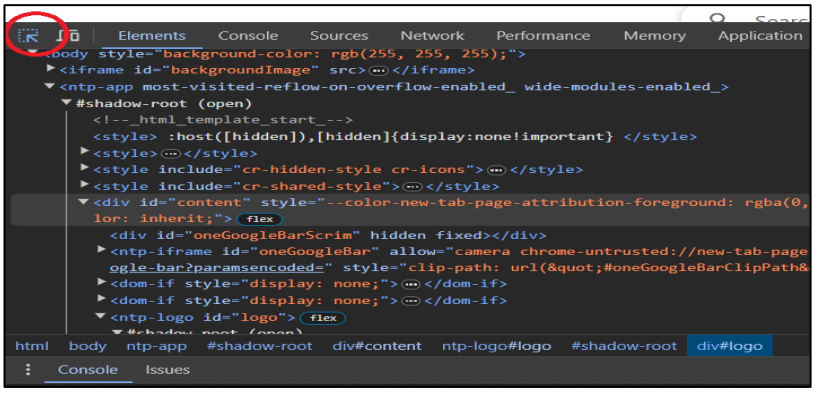
Eg. Inspecting the element- "Google"



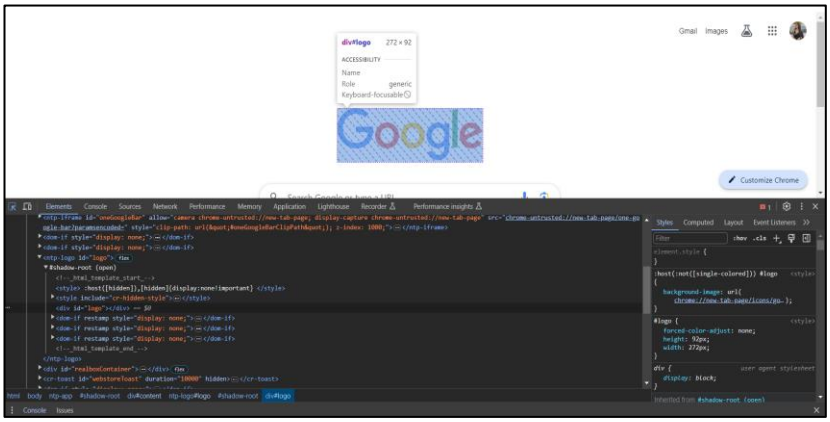
The Elements panel opens and "Google" is highlighted in DOM Tree



b) Click on the “Inspect Icon” in the top-left corner of DevTools highlighted below.



The element “Google” is highlighted with specifications.

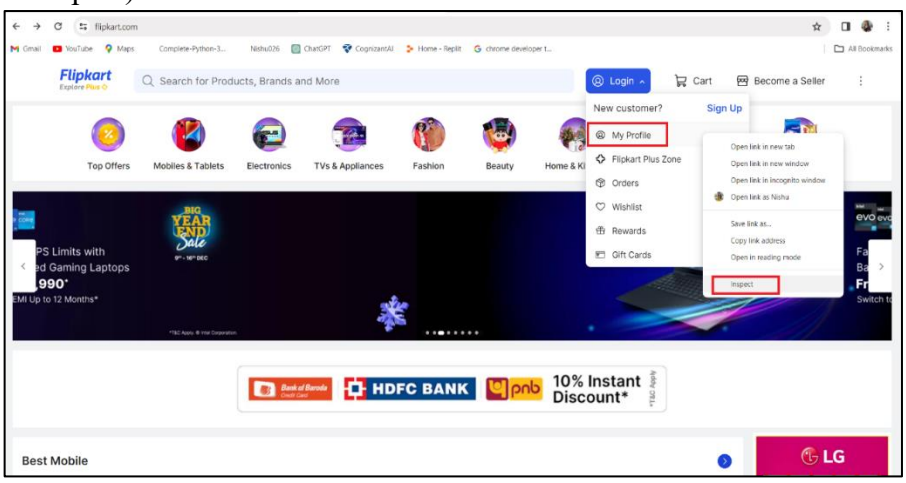


2. Navigating the DOM Tree with keyboard

Once you've selected a node in the DOM Tree, you can navigate the DOM Tree with keyboard.

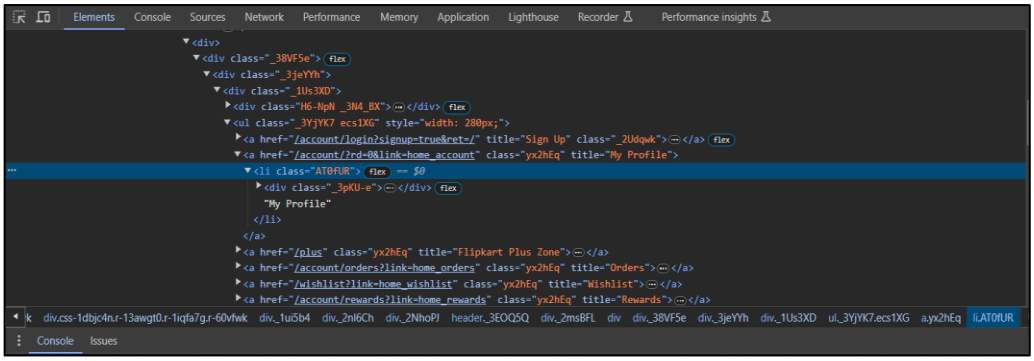
Steps:

i. Right-click on “My Profile” below and select Inspect (select any node you want to inspect).



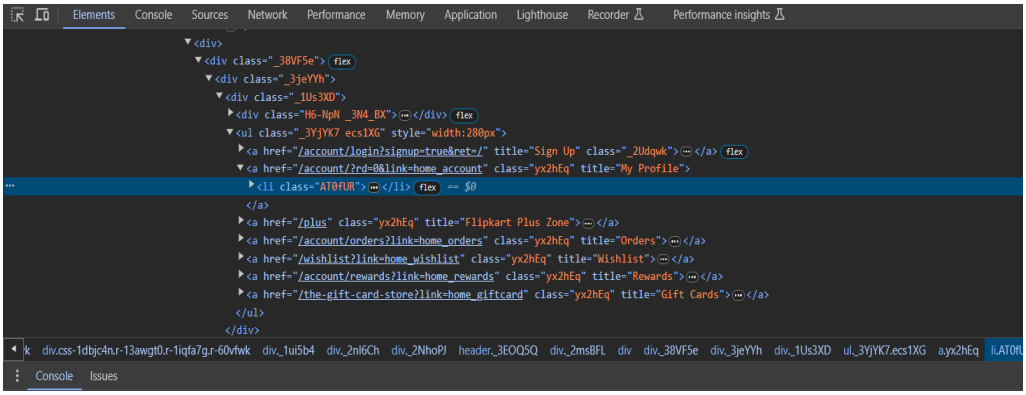
“My Profile” is selected in the DOM Tree.

ii. Press the right-arrow key > to expand the list.



iii. Press the left-arrow key <. The list collapses.

iv. Press the up-arrow key to go up and down-arrow key to go down in the DOM Tree.



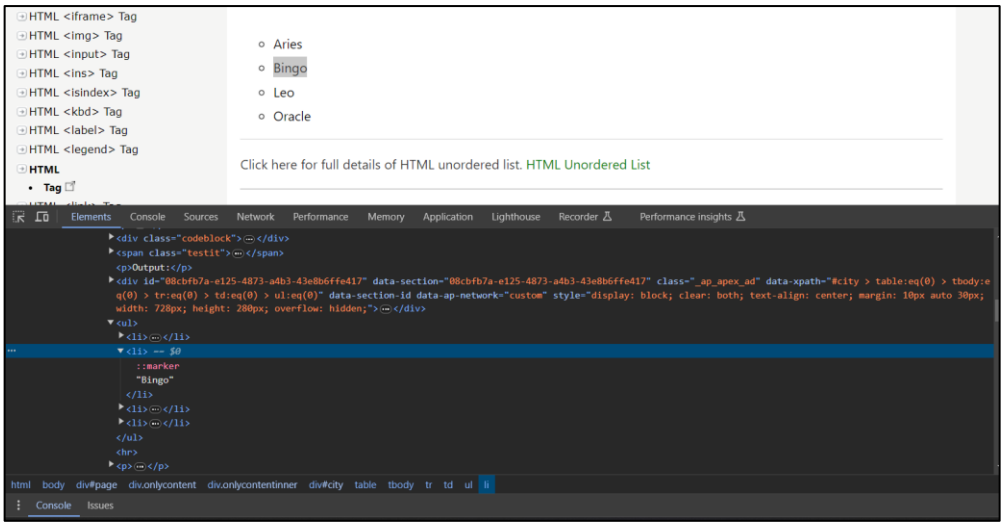
B. Editing the DOM

1. Editing content:

Steps:

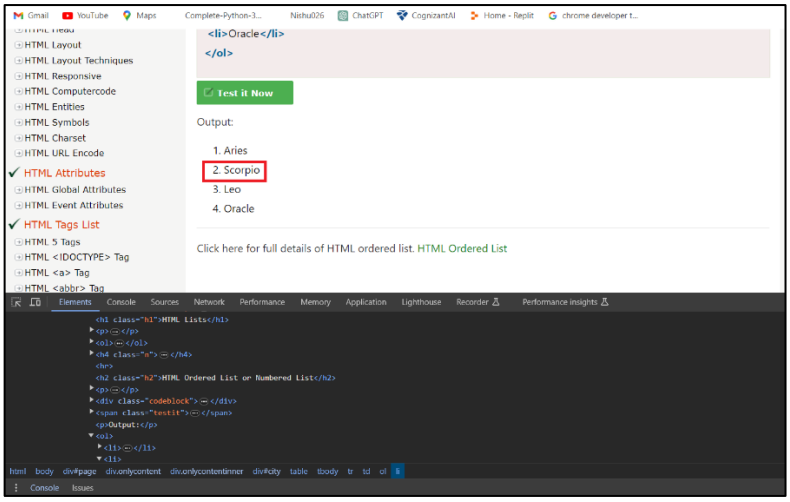
i. Right-click on the DOM node (eg. “Bingo”) and select Inspect.

ii. To edit a node's content, double-click the content in the DOM Tree.



iii. Delete Bingo and type “Scorpio” or any text.

The text Bingo Changes to Scorpio.



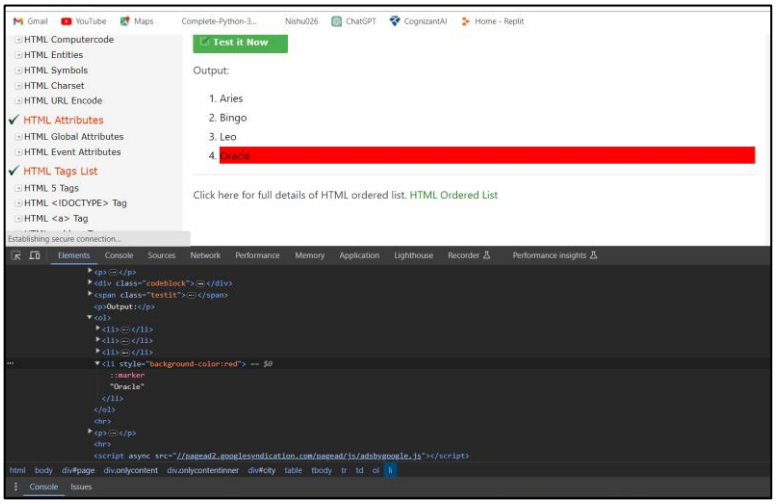
2. **Editing Attributes:**

Steps:

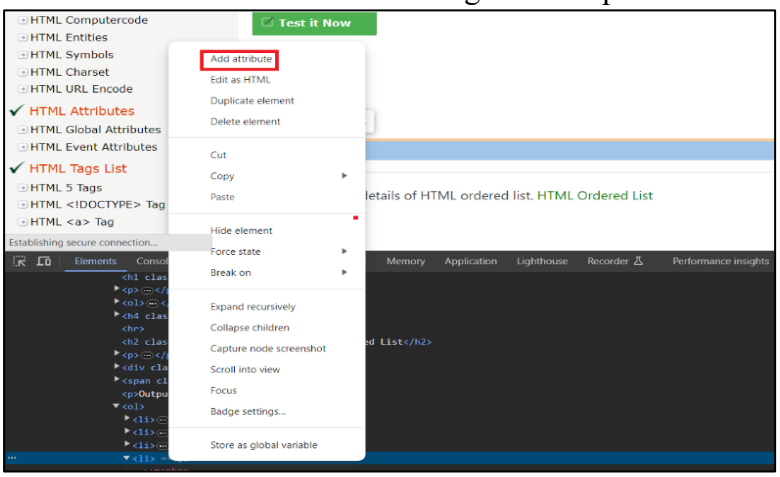
i. To edit or to add new attributes, inspect the node (eg., Oracle)

ii. Double-click <li>. The text is highlighted to indicate that the node is selected.

iii. Type style="background-color:red", and then press Enter. The background color of the node changes to red.



- You can also use the Edit attribute right-click option.



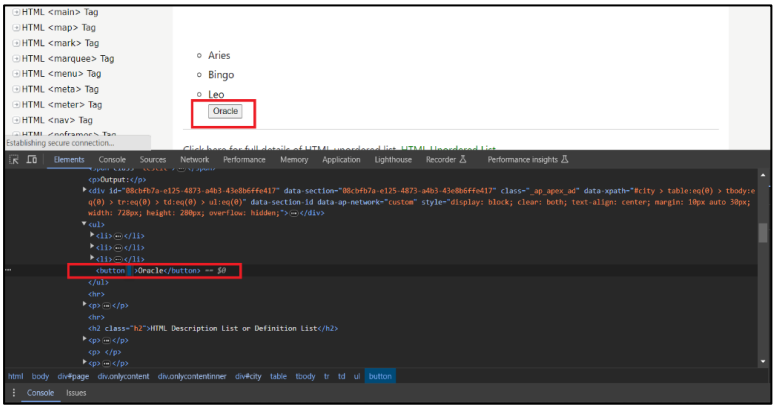
3. **Editing Node type:**

Steps:

i. To edit a node's type, double-click the existing type and then type in the new type.

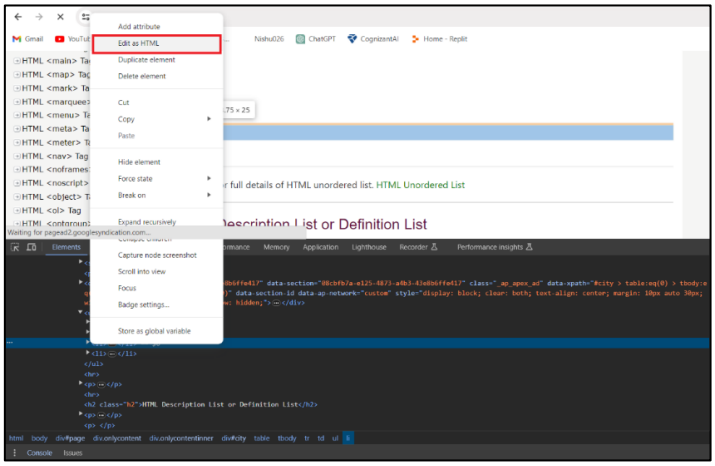
ii. Double-click <li>. The text li is highlighted.

iii. Delete li, type button, then press Enter. The <li> node changes to a <button> node.



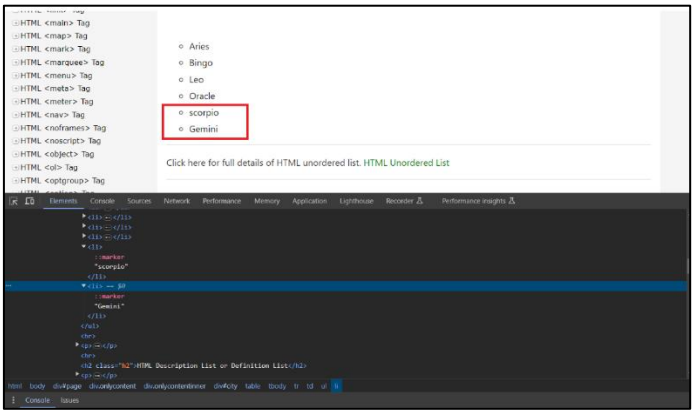
4. **Edit as HTML**

To edit nodes as HTML with syntax highlighting and autocomplete, select Edit as HTML from the node's drop-down menu. DevTool highlights HTML syntax and autocompletes tags for you.



Type: Scorpio

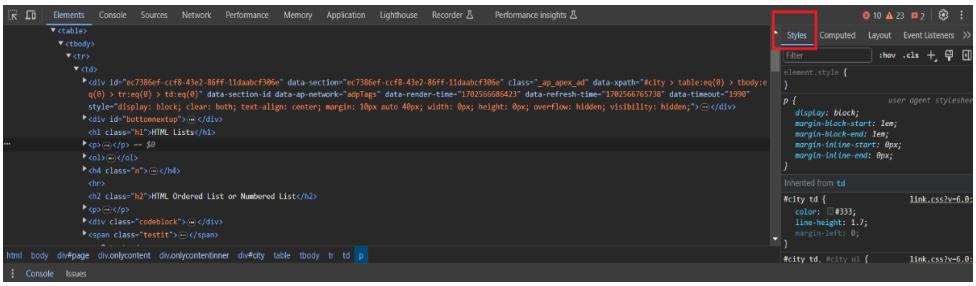
Gemini New items will be added to the list.



C. Viewing CSS:

1. Viewing CSS of an element:

The "Styles" pane in the Elements panel of Chrome's Developer Tools allows you to view and modify CSS styles applied to the selected HTML element. It offers a detailed view of the element's styles and provides tools for real-time editing and debugging.



D. Changing CSS:

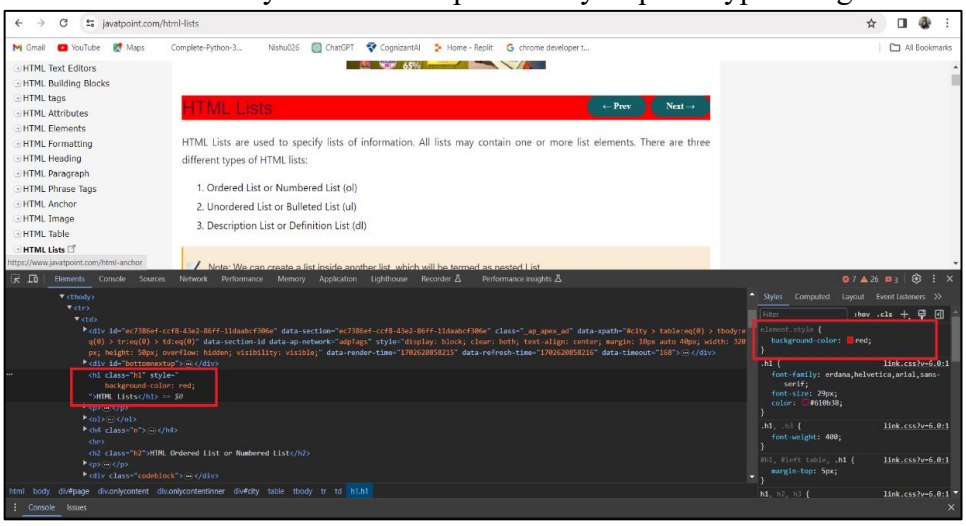
1. Adding a CSS declaration to an element

Steps: [Changing background color of an element]

i. Use the Styles pane when you want to change or add CSS declarations to an element.

ii. Inspect the element to change the background color of HTML element ("HTML List") highlighted below.

iii. Click "element.style" near the top of the Styles pane.Type background-color:red;



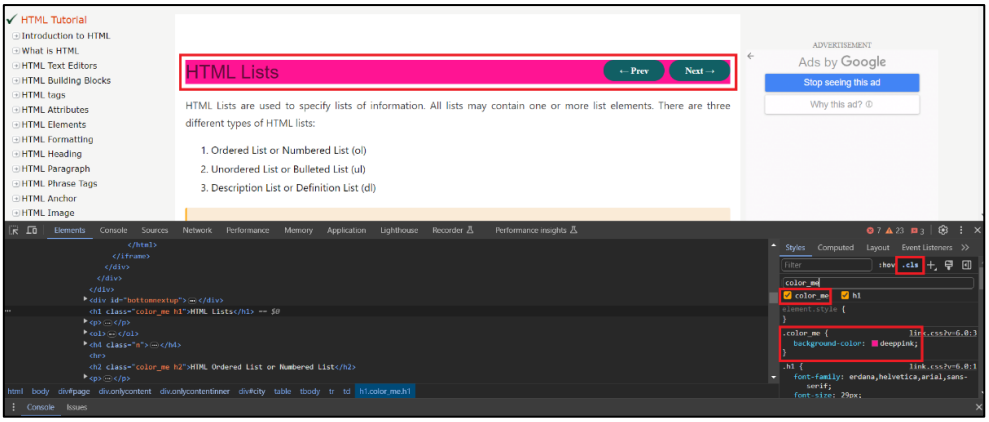
2. Adding a CSS class to an element

Steps:

i. Inspect an element.

ii. Click on ".cls" DevTools reveals a text box where you can add classes to the selected element.

iii. Type class_name in the Add new class text box and then press Enter. A checkbox appears below the Add new class text box, where you can toggle the class on and off.



3. Pseudoclass Editing:

Use the Styles pane to permanently apply a CSS pseudostate to an element. DevTools supports :active, :focus, :hover, :visited, and others.

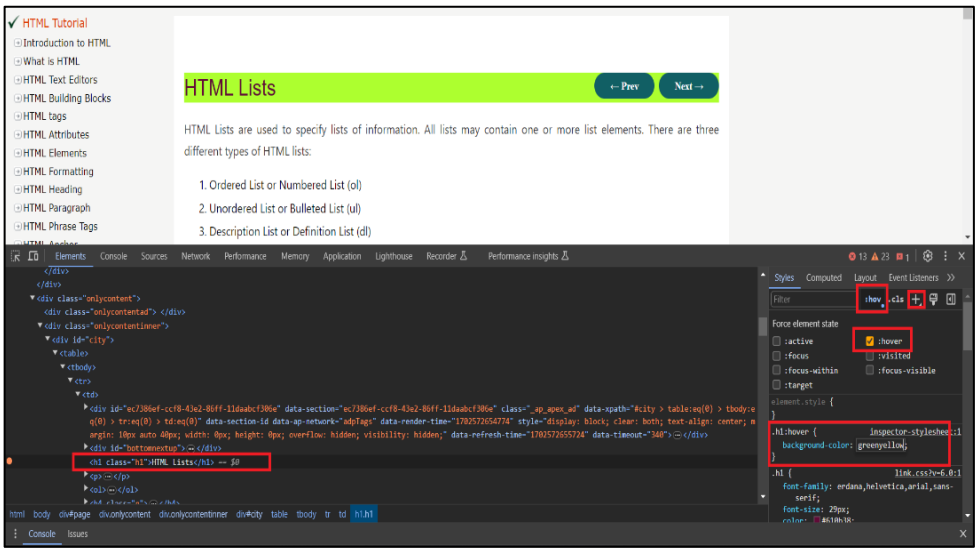
Steps:

i. Right-click the "HTML Lists" text and select Inspect.

ii. In the Styles pane, click :hov.

iii. Check the :hover checkbox.

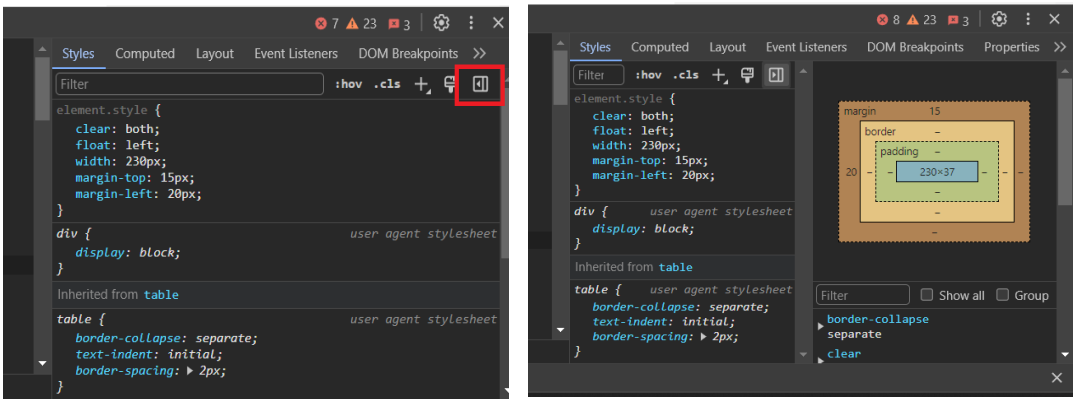
iv. Click on the "+" option to add hovering style to the selected element. The background color changes like before, even though you're not actually hovering over the element.



4. Changing the dimensions of an element:

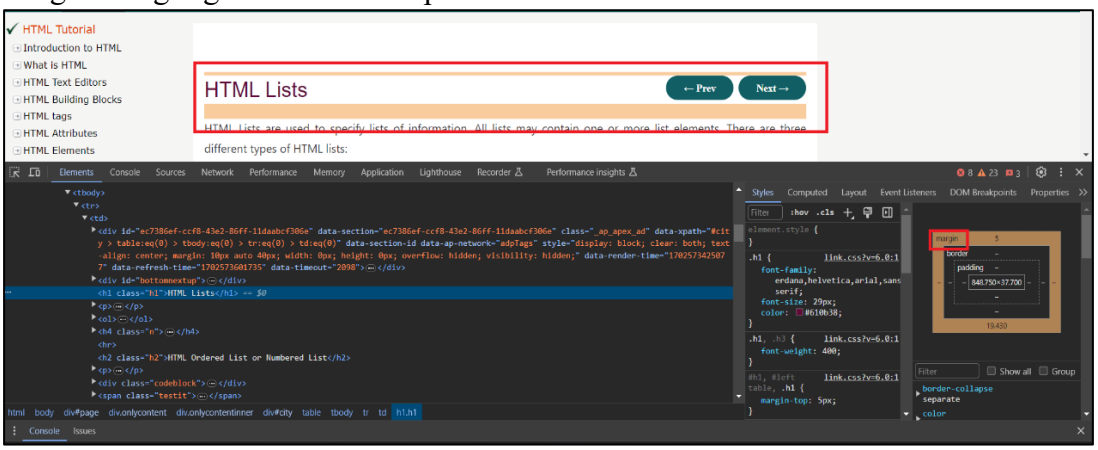
Use the Box Model interactive diagram in the Styles pane to change the width, height, padding, margin, or border length of an element.

To see the Box Model, click the Show sidebar. Show sidebar button in the action bar on the Styles pane.



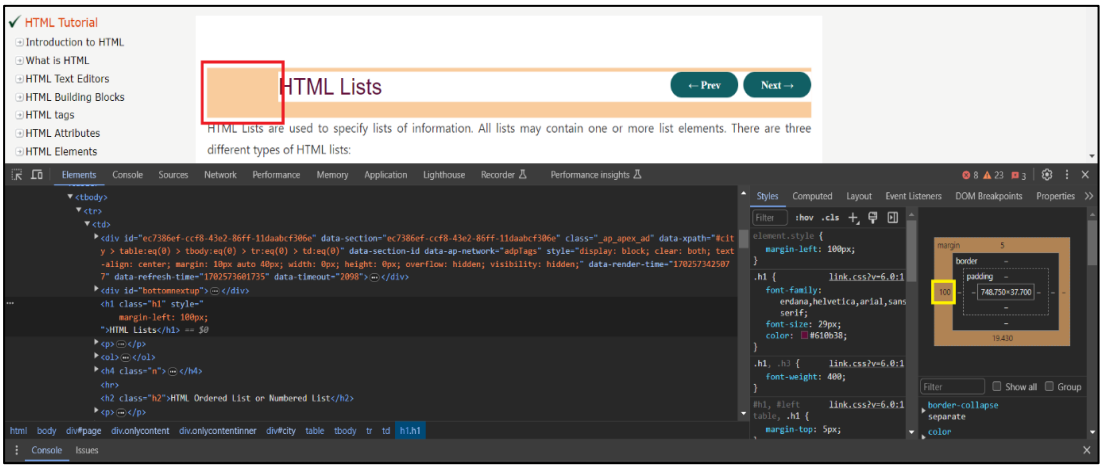
Steps:

i. In the Box Model diagram in the Styles pane, hover over margin. The element's margin is highlighted in the viewport.



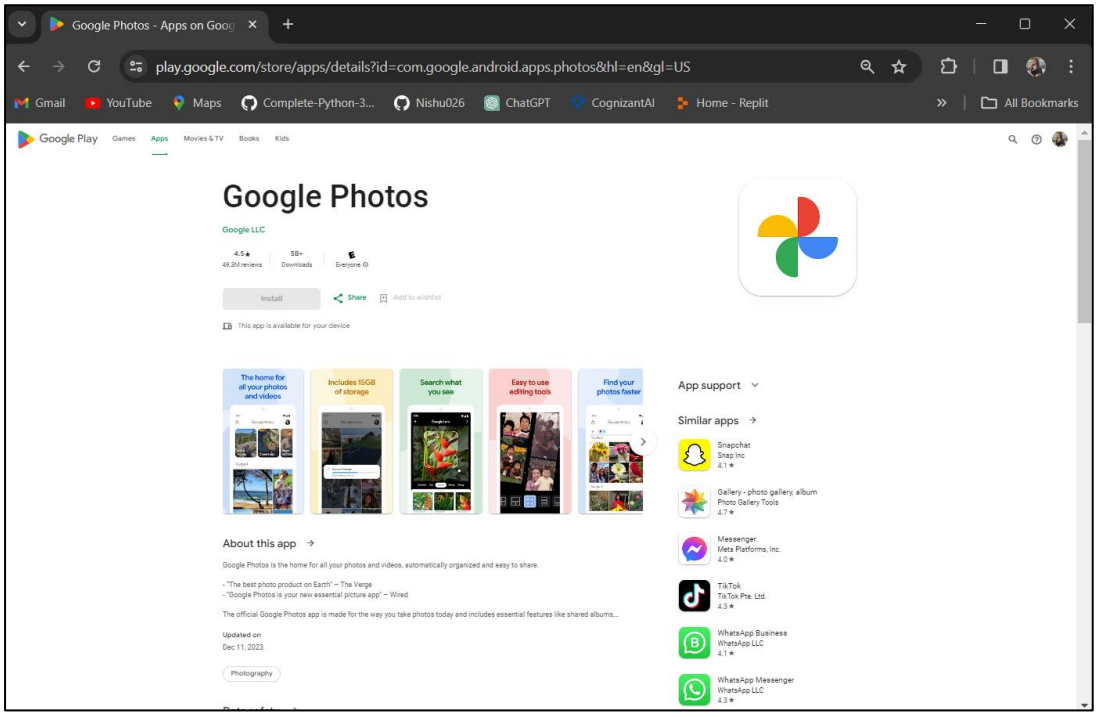
ii. Double-click the top margin in the Box Model. The element currently doesn't have left and right margins, so the left-margin has a value of -.

iii. Type 100 and press Enter. (100 px margin will be added)

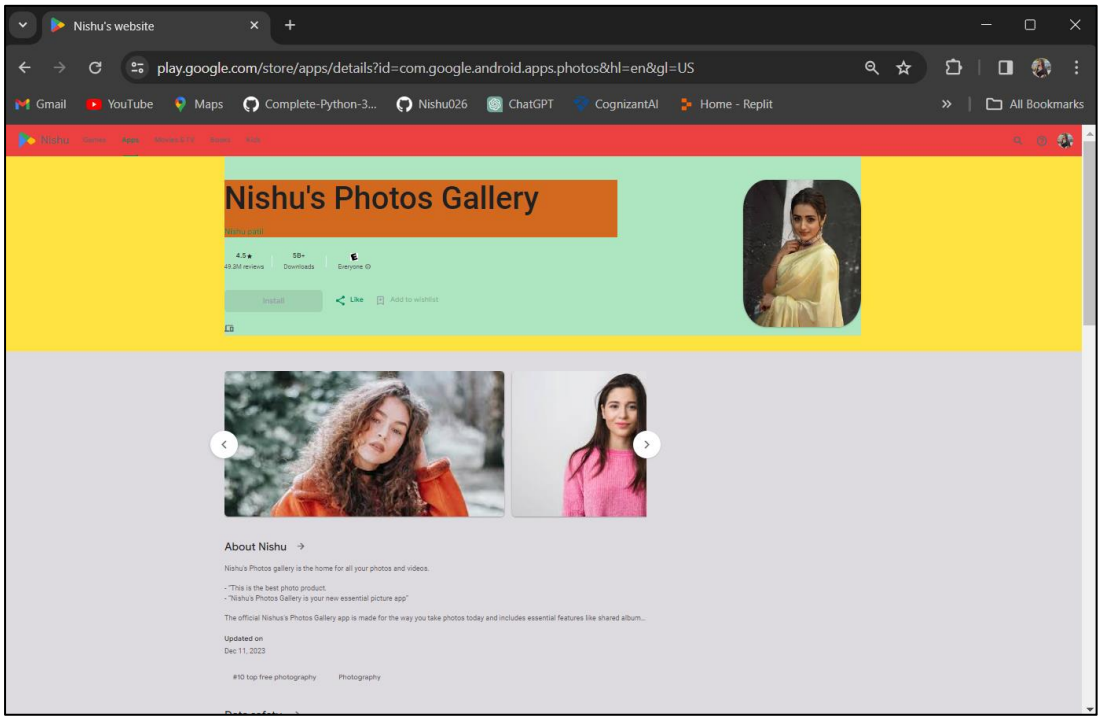


Example: Changing the front-end of the Google Photos website using Elements Panel.

❖ **Before:**



❖ **After DOM and CSS manipulation:**



2. CONSOLE PANEL:

The Console panel in Chrome's Developer Tools is a command-line interface where developers can execute JavaScript, log information, debug code, and diagnose issues within a web page or application.

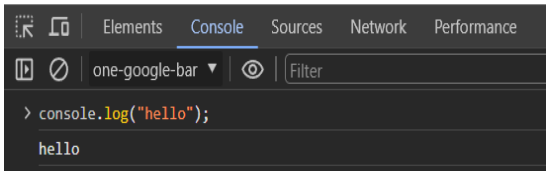
- **Features:**

A. Console for debugging:

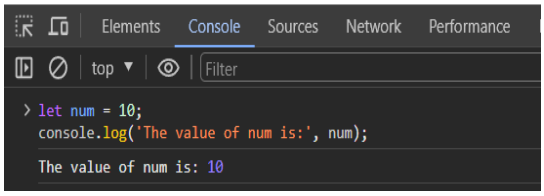
It offers various methods that can significantly aid in identifying and fixing issues in your JavaScript code. Here are some ways you can use the console for debugging:

1. **Logging messages:** Logging messages to the console panel in various programming languages or environments is a common practice for debugging and monitoring code.

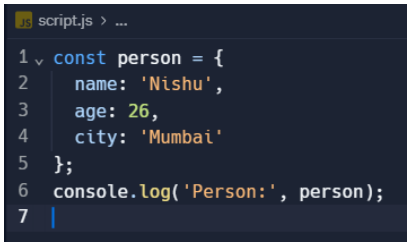
a) Using console.log (): Logs a simple message

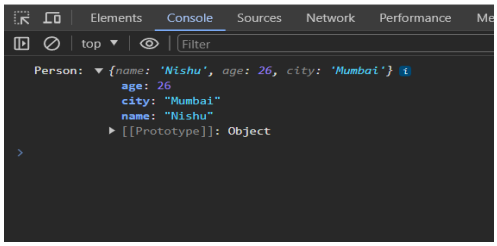


b) Logging Variables: Logs a variable value

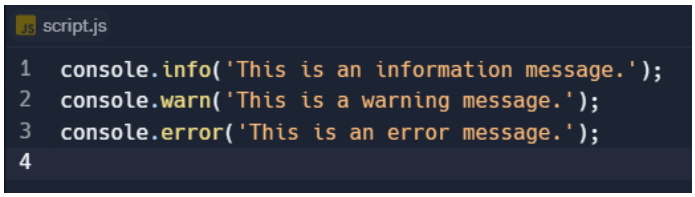


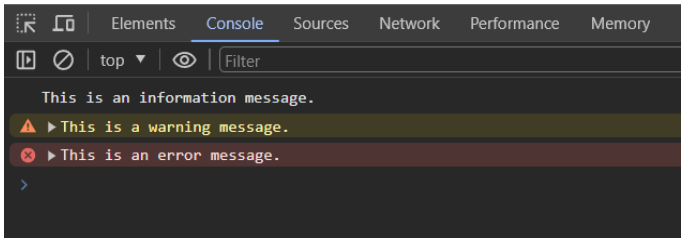
c) Logging Objects: Logs an object





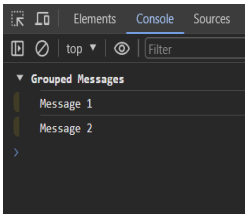
d) Info, Warning, Error:





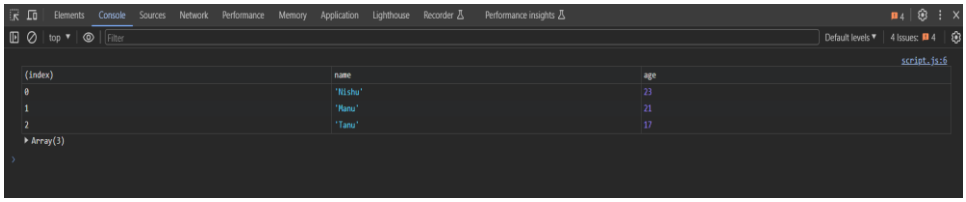
e) **Grouping Messages:**

```
script.js
1  console.group('Grouped Messages');
2  console.log('Message 1');
3  console.log('Message 2');
4  console.groupEnd();
```

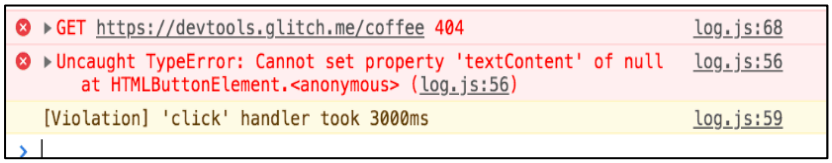


f) **Logging Objects as Tables:** Logs objects as a table.

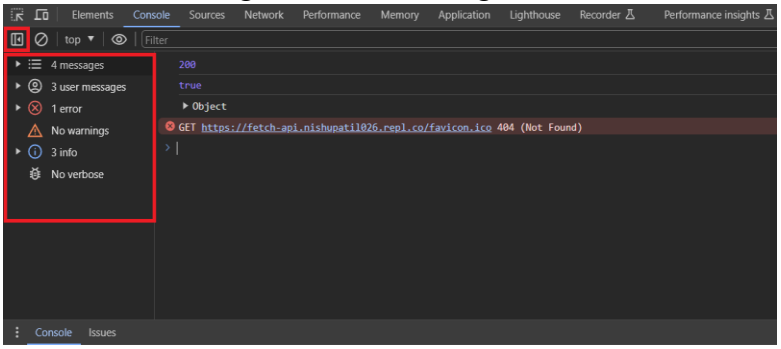
```
script.js > ...
1  const people = [
2    { name: 'Nishu', age: 23 },
3    { name: 'Manu', age: 21 },
4    { name: 'Tanu', age: 17 }
5  ];
6  console.table(people);
7
```



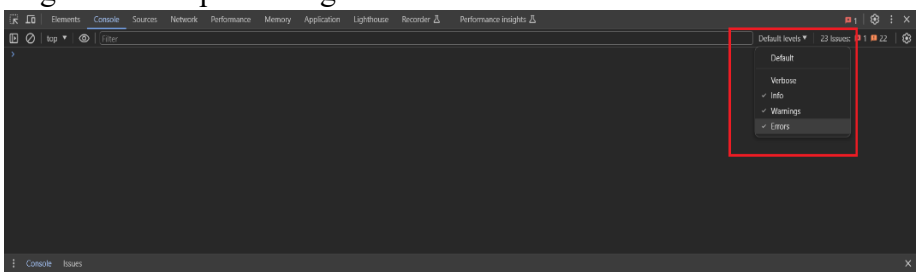
2. **Viewing messages logged by the browser:**
   The browser logs messages to the Console, too. This usually happens when there's a problem with the page.
   - **a)** Click **Cause 404**: The browser logs a 404-network error because the page's JavaScript tried to fetch a file that doesn't exist.
   - **b)** Click **Cause Error**: The browser logs an uncaught TypeError because the JavaScript is trying to update a DOM node that doesn't exist.
   - **c)** Click **Cause Violation**: The page becomes unresponsive for a few seconds and then the browser logs the message [Violation] 'click' handler took 3000ms to the Console. The exact duration may vary.



- Click Show Console Sidebar Show Console Sidebar. Expand each message to see the user messages, errors, warnings, etc.

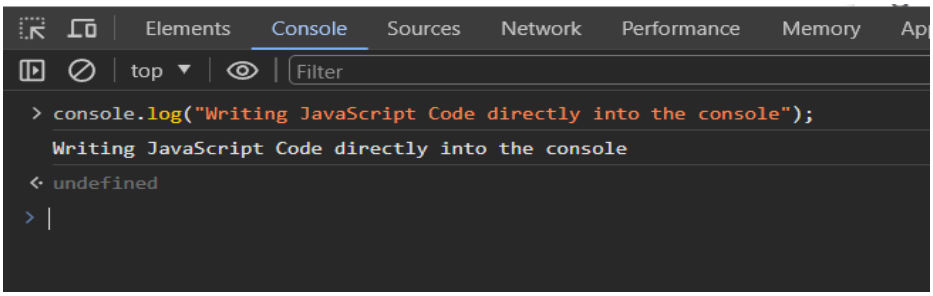


3. **Filtering messages:**
   Each console.* method is assigned a severity level: **Verbose, Info, Warning** and **Error.** For example, console.log() is an Info-level message, whereas console.error() is an Error-level message.
   You can select the log level you want to see or deselect to hide those messages.A level is disabled when there is no longer a checkmark next to it. You can click the Log Levels drop-down again to re-enable it.

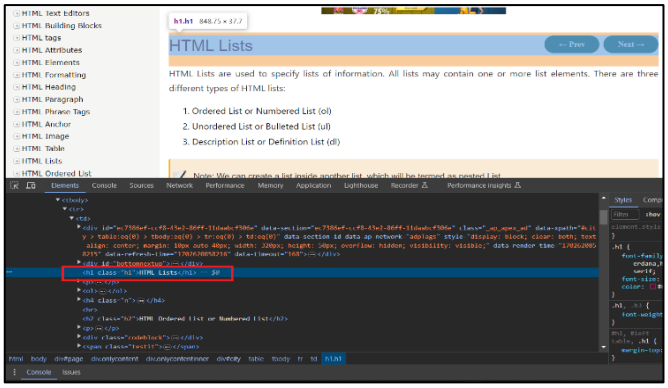


   a) **Filter by text:** When you want to only view messages that include an exact string, type that string into the Filter text box.
   b) **Filter by regular expression:** When you want to show all messages that include a pattern of text, rather than a specific string, use a regular expression.
   c) **Filter by message source:** When you want to only view the messages that came from a certain URL, use the Sidebar.

B. Running JavaScript:
Developers can directly enter and execute JavaScript commands in the console, which can manipulate the webpage in real-time. This is useful for testing code snippets or quickly experimenting with DOM manipulation and other JavaScript functionalities.
Once the console is open, you can start typing JavaScript code directly into the input area.

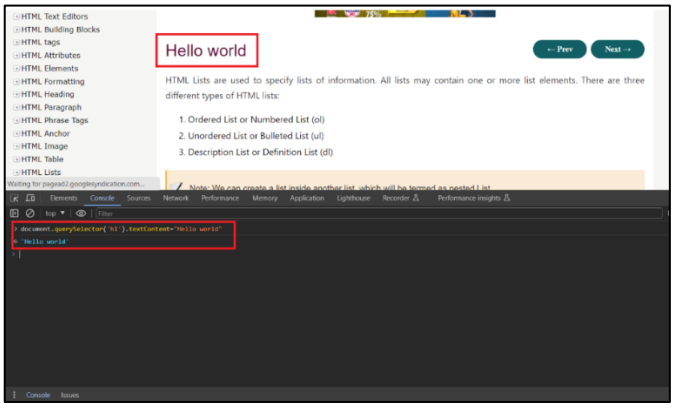


1. **Interacting with the DOM:**
   You can access and manipulate the Document Object Model (DOM) using JavaScript in the console.
   Steps:
   i. Inspect an element (“HTML Lists”). Element is highlighted in th DOM tree.



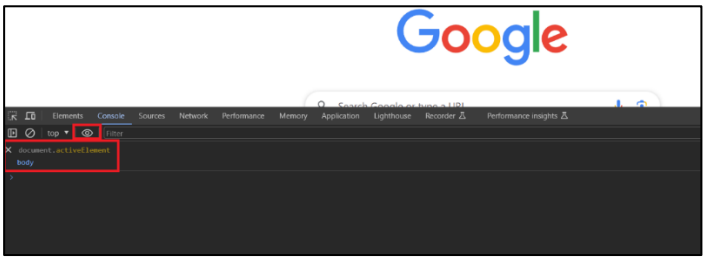
ii. Click on Console to change the page's title from “HTML Lists” to “Hello World”

Type: document.querySelector(‘h1’).textContent=“Hello World”



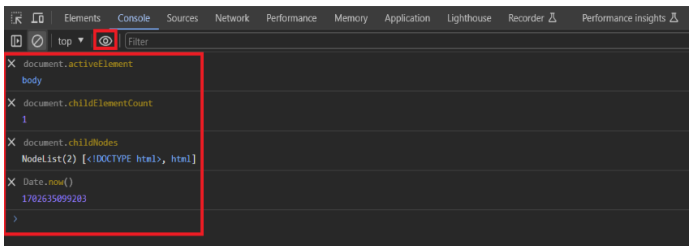
2. **Create a live expression:**

Click Create Live Expression . The Live Expression text box appears.Type your expression in the text box. For example, you can use a live expression to track element focus.



a) Adding multiple expressions:

To pin multiple expressions in parallel, click the Create a live expression button as many times as you need. You can only see several pinned expressions at a time.



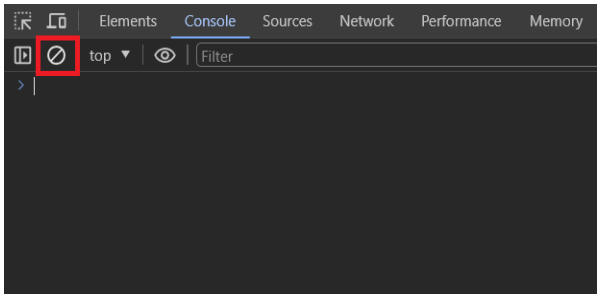
b) Remove expressions:

To remove an expression, click the Close button next to it.

3. **Clearing Console:**

Type: **console.clear()** in console panel to clear console.

Or click on the clear Console icon

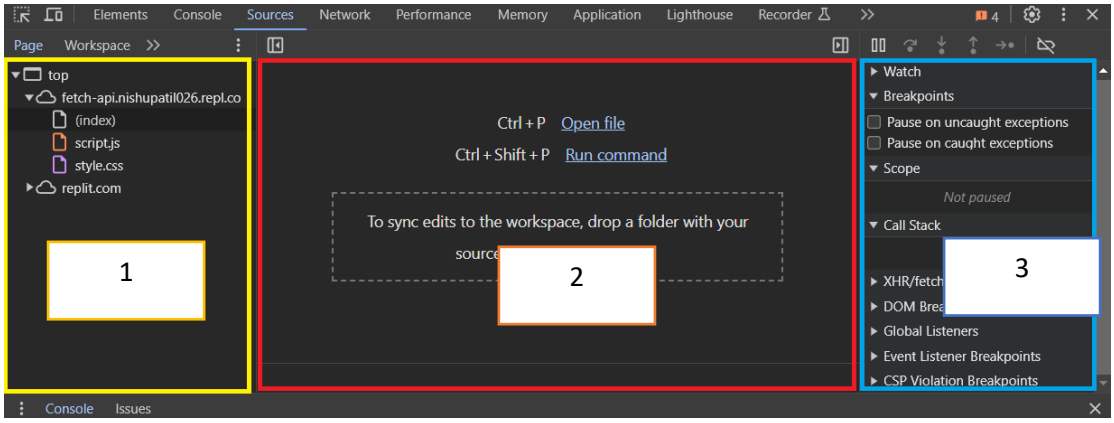


3. SOURCES PANEL:

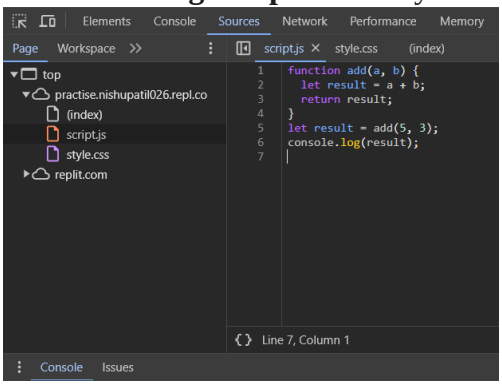
The "Sources" panel is a powerful tool primarily used for debugging and analyzing the source code of web applications.

➢ **Features:**

A. The Sources panel UI has 3 parts:



1. **The File Navigator pane:** Every file that the page requests is listed here.

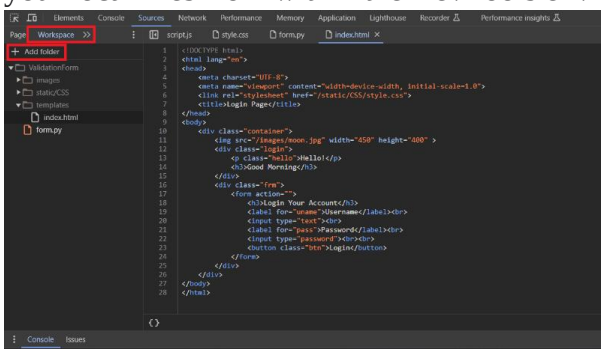


Features:

a) **Code Navigation:** Click on individual files to view their content in the code editor. Search for specific files based on file types to locate what you need.

b) **Debugging and Editing:** Set breakpoints directly within the code editor by clicking on line numbers. Modify code in files (if enabled) and save changes directly from DevTools.

c) **Adding Workspace:** The "Workspace" option within the Sources panel allows you to link local files from your computer to the files served by a website. This integration enables you to edit and save changes directly to your local files from within the DevTools environment.



2. **The Code Editor pane:** After selecting a file in the File Navigator pane, the contents of that file are displayed here.

   Features:

   a) **Code Display:**
   Syntax Highlighting: Displays code with color-coded elements for improved readability.
   Line Numbers: Facilitates easy reference and navigation within the code.

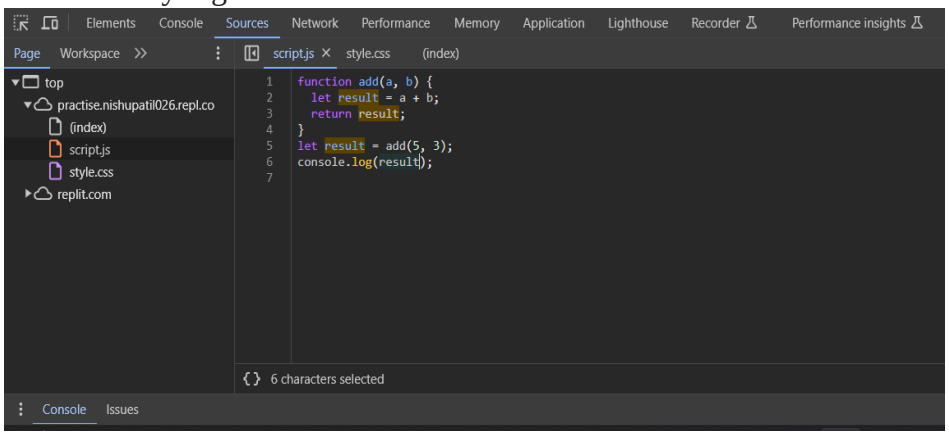
   b) **Code Navigation:**
   File Selection: Allows selection of different files to view their contents.
   Go to Line: Jump directly to a specific line within the code.

   c) **Debugging Integration:**
   Setting Breakpoints: Clicking on line numbers to set breakpoints for debugging purposes.
   Conditional Breakpoints: Set breakpoints to pause execution based on certain conditions.

   d) **Code Modification:**
   Live Editing: Modify HTML, CSS, or JavaScript files directly within the editor.
   Undo/Redo: Allows reverting or reapplying changes made to the code.

   e) **Search and Find:**
   Find Functionality: Search for specific text within the code.
   Search and Replace: Replace text occurrences within the code.

   f) **Code Formatting and Indentation:**
   Auto-Indentation: Automatically adjusts code indentation for better readability.
   Formatting Options: Offers options for code formatting to maintain consistent styling.



3. **The JavaScript Debugging pane:** Various tools for inspecting the page's JavaScript. If your DevTools window is wide, this pane is displayed to the right of the Code Editor pane.

   Features:

   a) **Breakpoints:** Place breakpoints in your code to pause execution at specific points. This allows you to inspect variables and step through code.

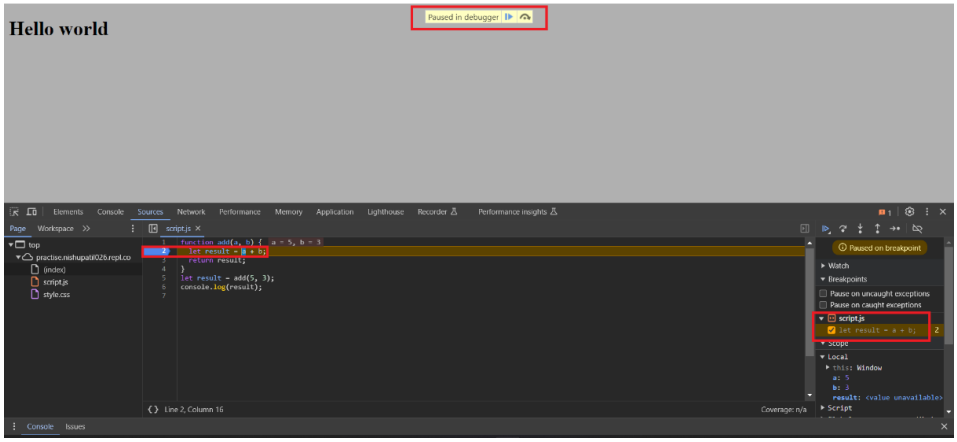
b) Setting a Breakpoint:

Steps:

i. Open your JavaScript file in the Code Editor pane of "Sources" panel.

ii. Find the line number where you want to place the breakpoint.

iii. Click on the line number to set a breakpoint (you should see a blue marker 2 indicating the breakpoint).

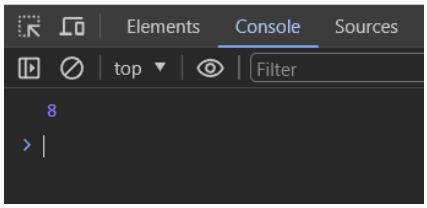


Once the breakpoint is set, perform an action that triggers the function add(), such as calling it with specific values:

When the code execution reaches the line with the breakpoint, the browser will pause, and the DevTools will open if they aren't already open. You can then Inspect Variables, Step Through Code and Evaluate Expressions to check the values of a, b, and result.

Click Resume script execution  The script continues executing until it reaches line 6.

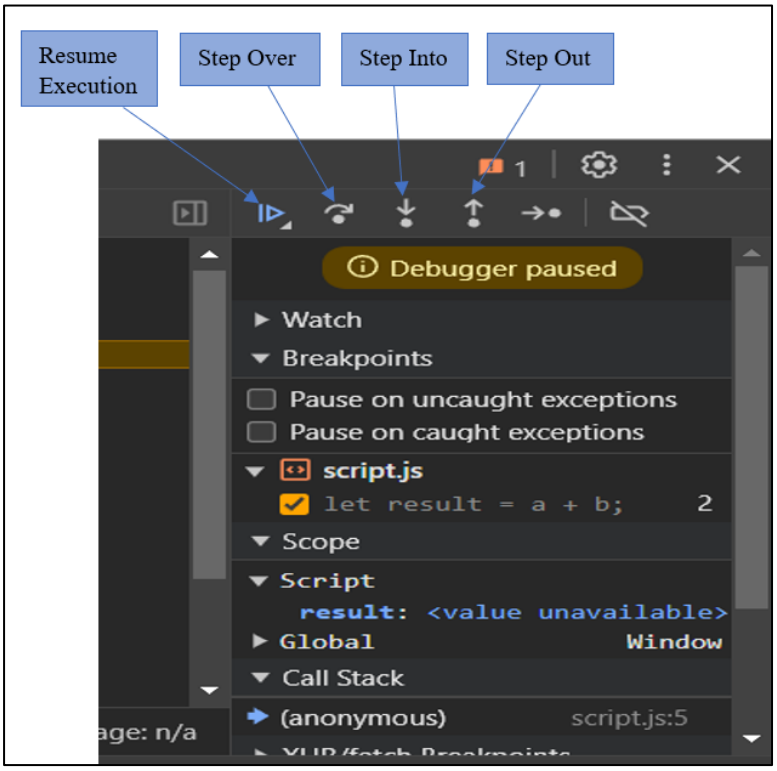
And you can see the result in console:



c) Types of Breakpoints:

Breakpoint	Usage
Line-of-code	Pause on an exact region of code
Conditional line-of-code	Pause on an exact region of code, but only when some other condition is true.
DOM Breakpoint	Pauses execution when a particular DOM event is triggered on a specified DOM element.
XHR/Fetch Breakpoint	Helpful for debugging network-related issues or inspecting AJAX requests
Event Listener Breakpoint	Pause on the code that runs after an event, such as click, is fired.
Trusted Type	Pause on Trusted Type Violations

B. Debugging Controls:



a) **Resume Execution:** Continues code execution until the next breakpoint is encountered.

b) **Step Over Button:** Executes the current line of code and pauses at the next line.

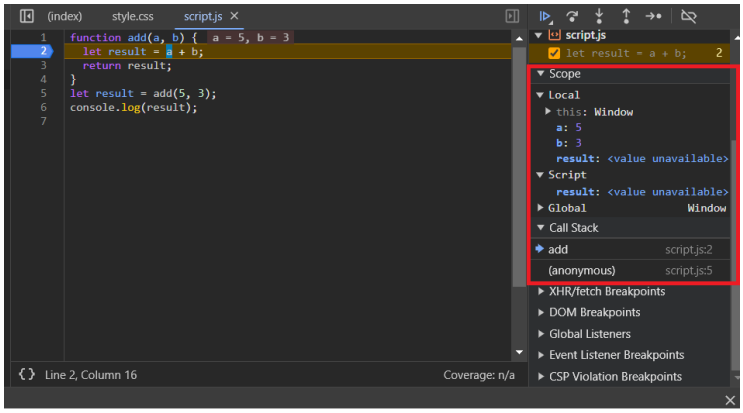
c) **Step Into Button:** If the current line calls a function, it will step into that function.

d) **Step Out Button:** Steps out of the current function and stops at the line following the function call.

C. Scope Inspection and Call Stack:

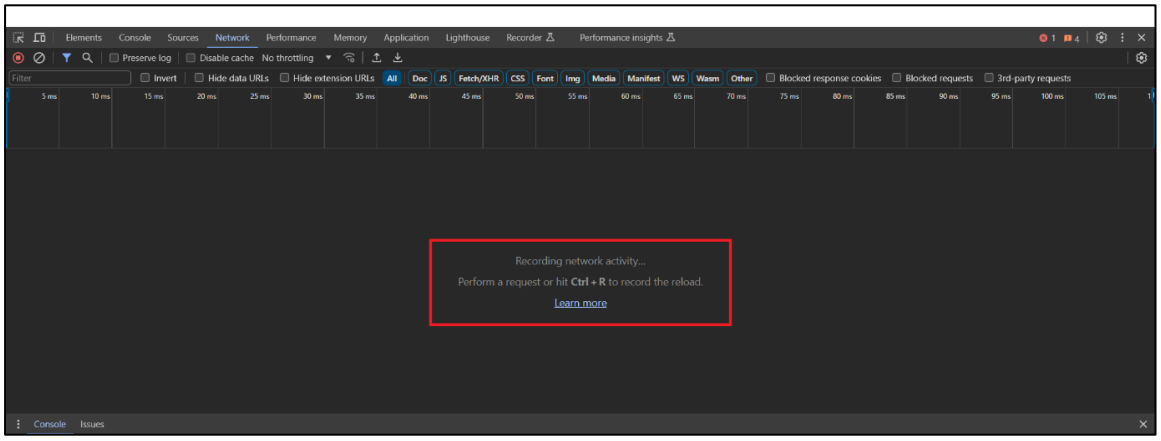
a) Scope Variables: Inspect variables' values within different scopes (local, closure, global) at each breakpoint.

b) Call Stack: Visualizes the function call hierarchy, showing the path through which, the code was executed.



4. NETWORK PANEL:

The Network panel in browser DevTools provides a detailed overview and analysis of the network activity of a web application. It captures and displays information about each network request made by the webpage, including resources like HTML, CSS, JavaScript files, images, API requests, and more.



Right now the Network panel is empty. That's because DevTools only logs network activity while it's open and no network activity has occurred since you opened DevTools.

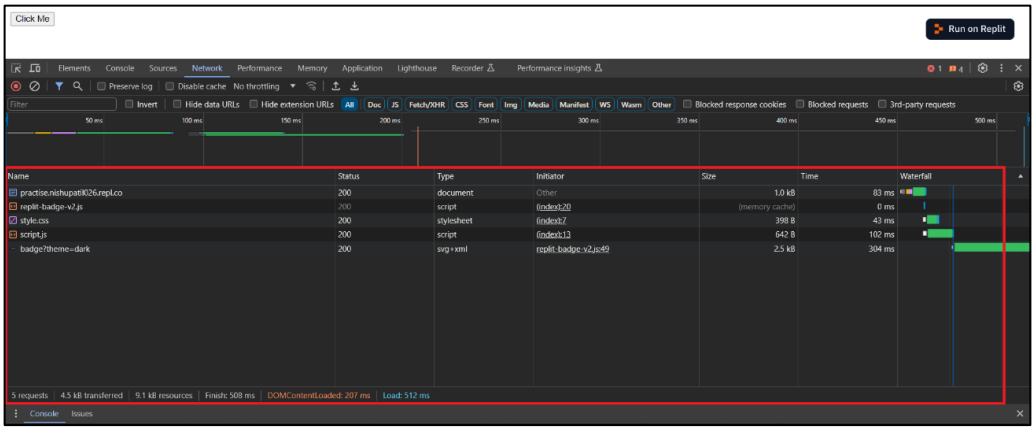
➢ **Features:**

A. Network log:

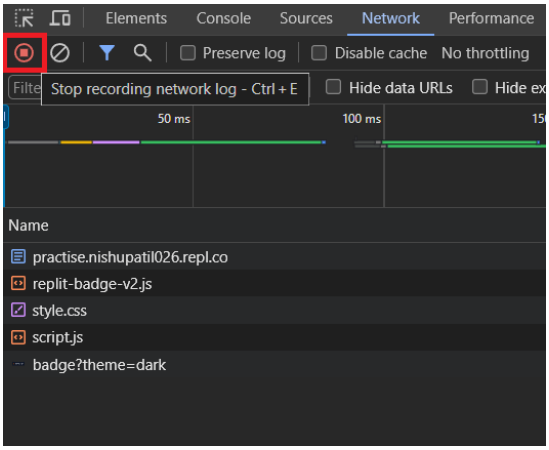
To view the network activity that a page causes: Reload the page. The Network panel logs all network activity in the Network Log. Each row of the Network Log represents a resource. By default, the resources are listed chronologically. Each column represents information about a resource.

The default columns in Network log:

- **Name:** Name of the resources.
- **Status**: The HTTP response code.
- **Type:** The resource types.
- **Initiator**: What caused a resource to be requested. Clicking a link in the Initiator column takes you to the source code that caused the request.
- **Size**: Size of the resources.
- **Time:** How long the request took.
- **Waterfall**: A graphical representation of the different stages of the request. Hover over a Waterfall to see a breakdown.

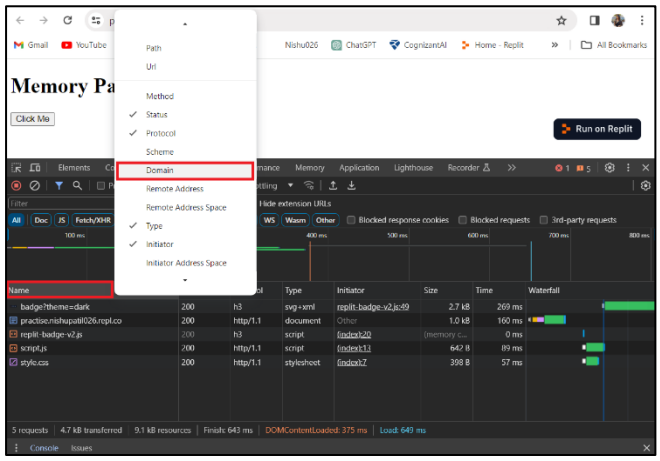


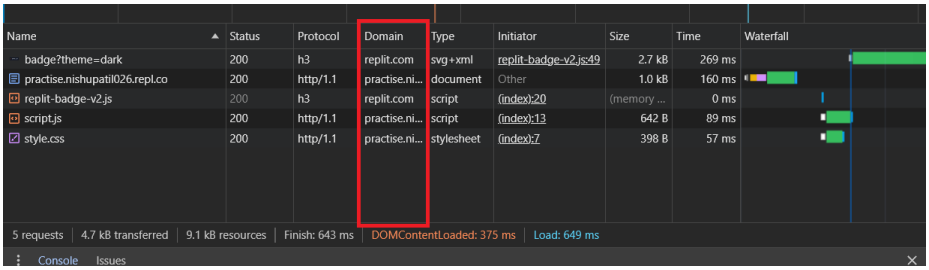
To stop the network recording click on the stop network recording option (ctrl+E):



B. Show more information:

The columns of the Network Log are configurable. You can hide columns that you're not using. There are also many columns that are hidden by default which you may find useful. Right-click the header of the Network Log table and select any required column. Eg., Domain. The domain of each resource is now shown

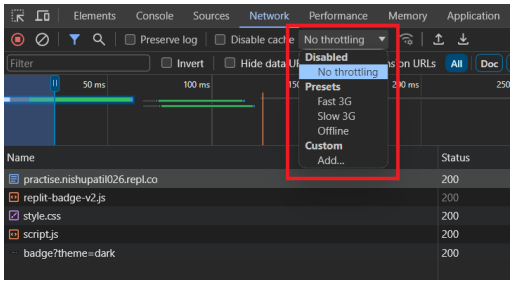




To hide the Domain column again click on header of network log and uncheck the domain.

C. Network Throttling:

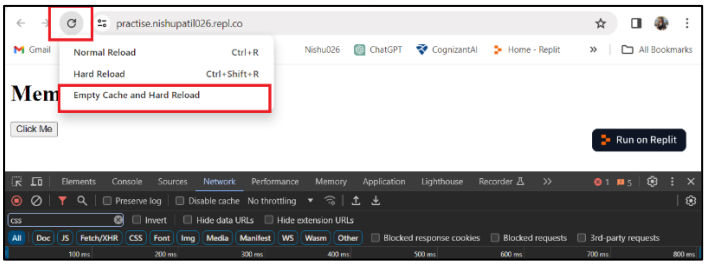
Simulates different network conditions like slow 3G, fast 3G, or offline mode to test and optimize performance under varied circumstances. By throttling the page, you can get a better idea of how long a page takes to load on a mobile device.



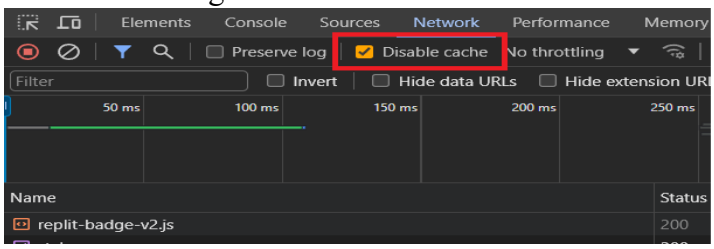
D. Empty Cache And Hard Reload

On repeat visits, the browser usually serves some files from its cache, which speeds up the page load. "Empty Cache And Hard Reload" forces the browser to go the network for all resources. This is helpful when you want to see how a first-time visitor experiences a page load.

Long-press Reload and then select "Empty Cache And Hard Reload".



- Disabling Cache:

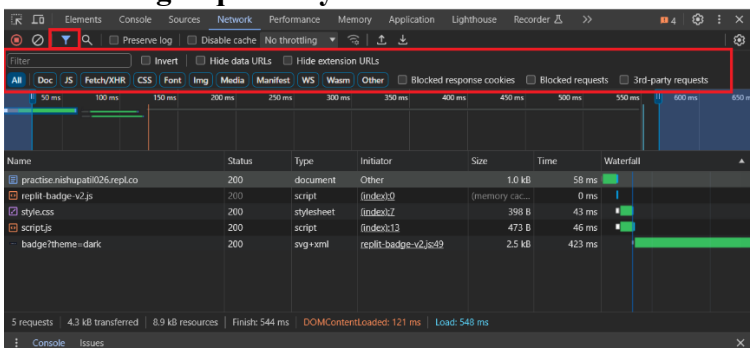


E. Filtering resources:

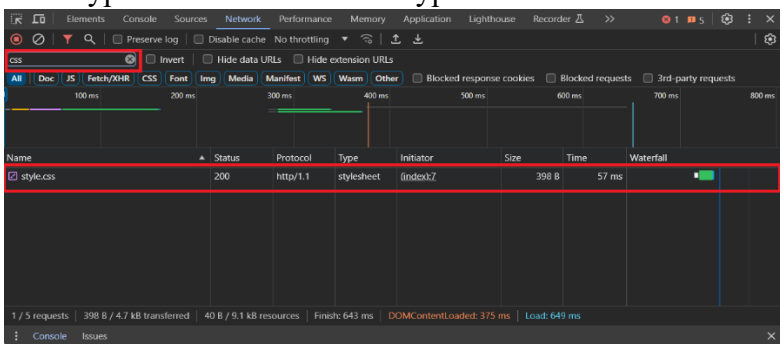
DevTools provides numerous workflows for filtering out resources that aren't relevant to the task at hand. The **Filters** toolbar should be enabled by default. If not: Click **Filter** to show it.

The Filter text box supports many different types of filtering such as filtering by string(.png), regular expression(/.*\.[cj]s+$/), or property(repl.co).

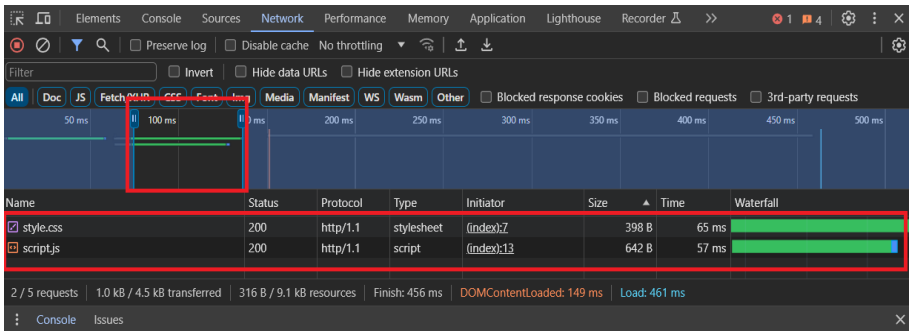
1. Filtering requests by resources:



If we type “CSS” All other file types are filtered out.



2. **Filtering requests by time:** Any request that was active during the highlighted time is shown.



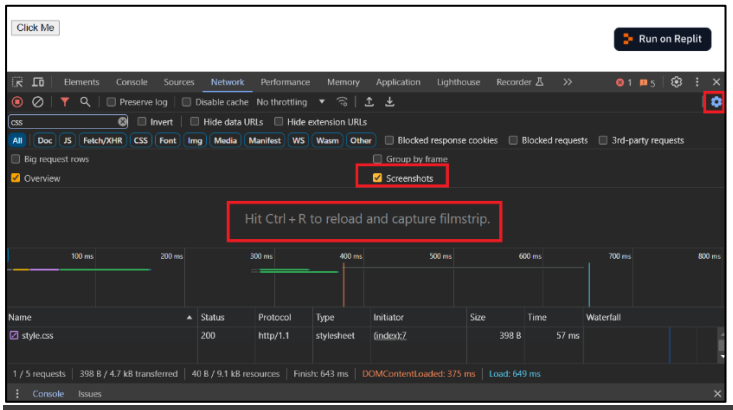
F. Capture screenshots:

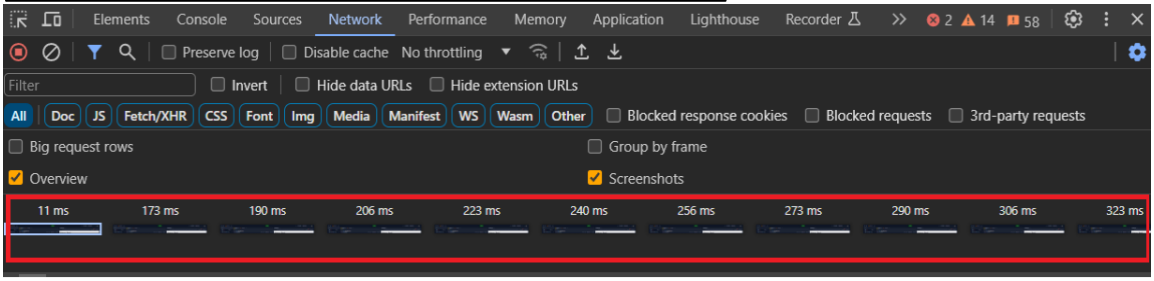
The Screenshots pane provides thumbnails of how the page looked at various points during the loading process.

Steps:

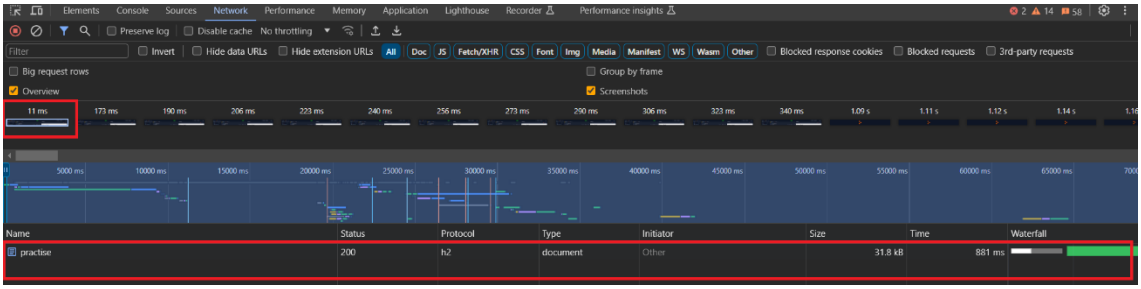
i. Click on settings of network panel.

ii. Check the “Screenshots” checkbox and press ctrl+R to reload a page.





iii. Click the first thumbnail. DevTools shows you what network activity was occurring at that moment in time.



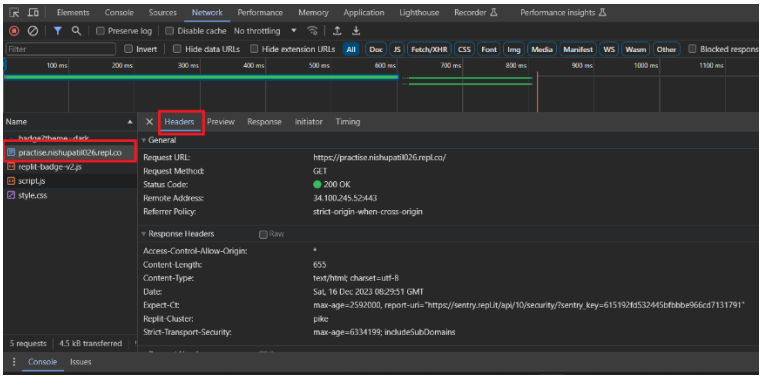
iv. Uncheck Screenshots again to close the Screenshots pane.

v. Reload the page again.

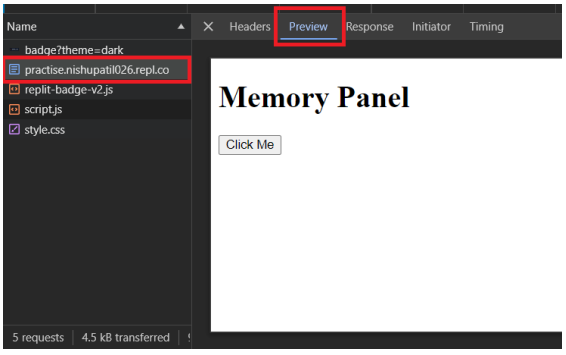
G. Inspect a resource's details:

Click a resource to learn more information about it.

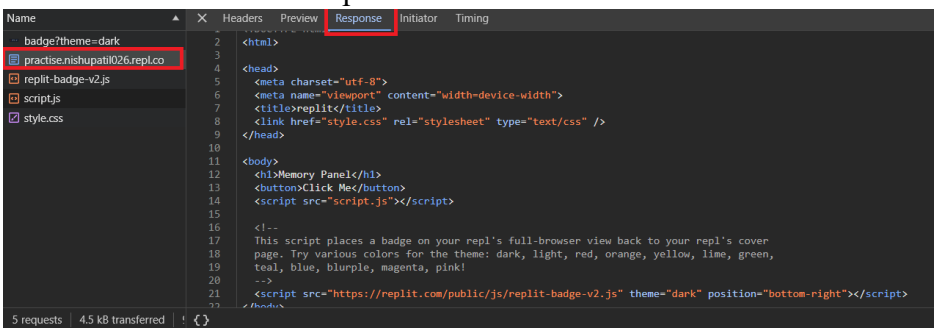
1. **Headers:** Use this tab to inspect HTTP headers.
   Click on HTML file > Headers



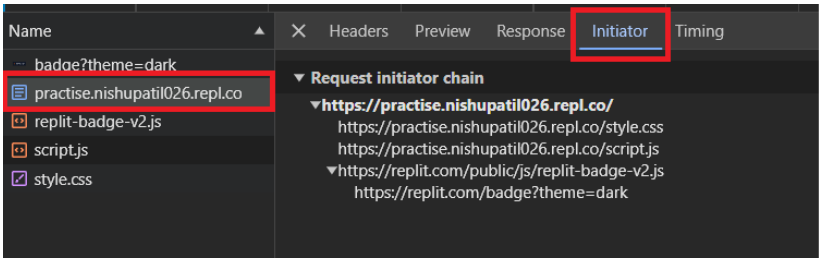
2. **Preview:** A basic rendering of HTML is shown.
   Click on HTML file > Preview



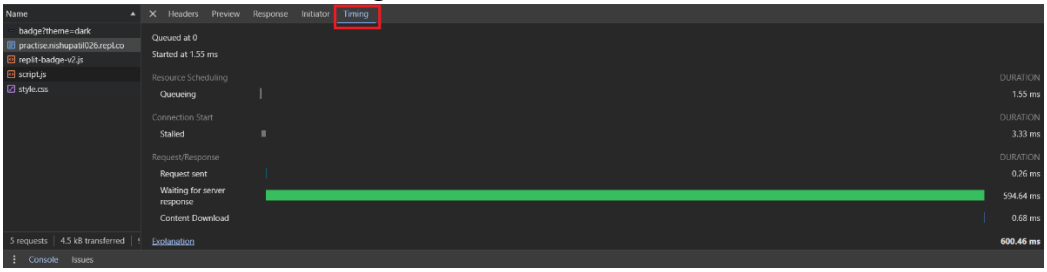
3. **Response:** The HTML source code is shown
   Click on HTML file > Response



4. **Initiator:** source of the resource.
   Click on HTML file > Initiator



5. **Timing:** A breakdown of the network activity for this resource is shown.
   Click on HTML file > Timing



   Click on close to close the network log.

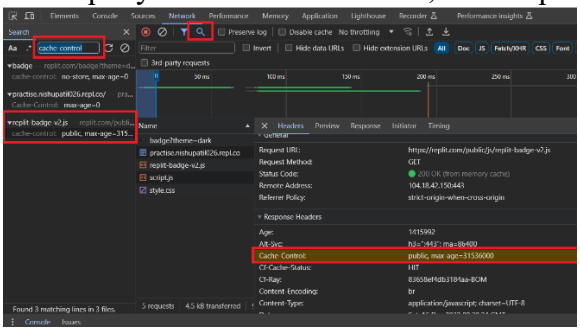
H. Search network headers and responses:

Steps:

i. Click Search. The Search pane opens to the left of the Network log.

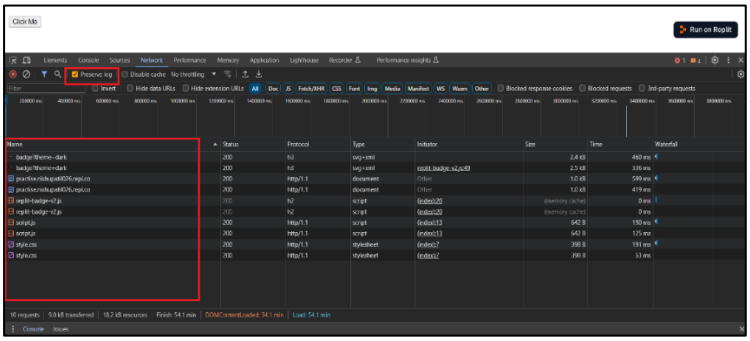
ii. Type “cache-control”. The Search pane lists all instances of Cache-Control that it finds in resource headers or content.

iii. Click a result to view it. If the query was found in a header, the Headers tab opens. If the query was found in content, the Response tab opens.



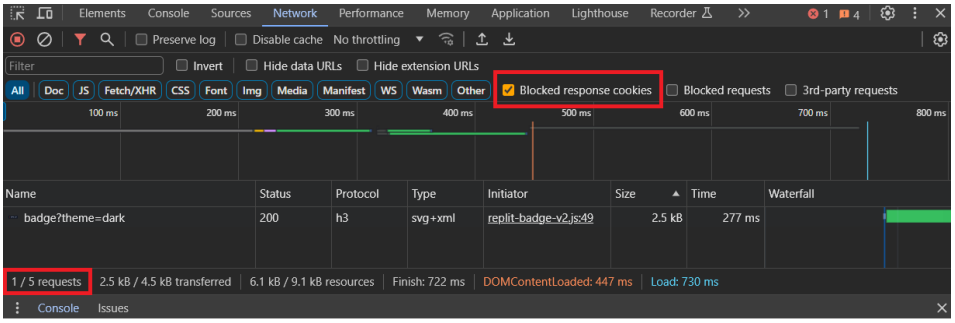
I. Save requests across page loads:

To save requests across page loads, check the Preserve log checkbox on the Network panel. DevTools saves all requests until you disable Preserve log.

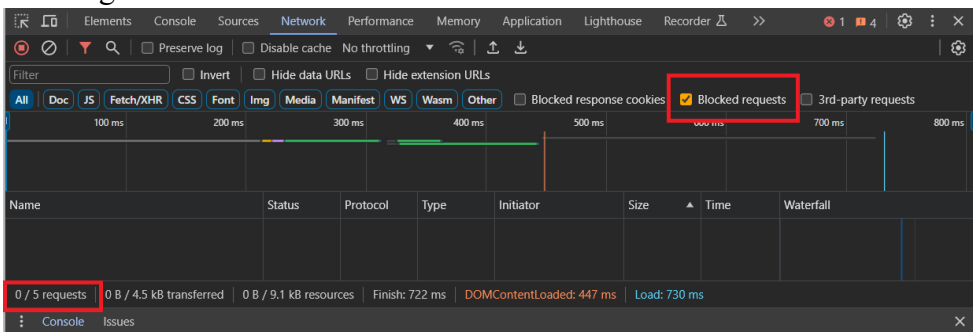


J. Blocked response cookies, Blocked requests, 3rd-party requests:

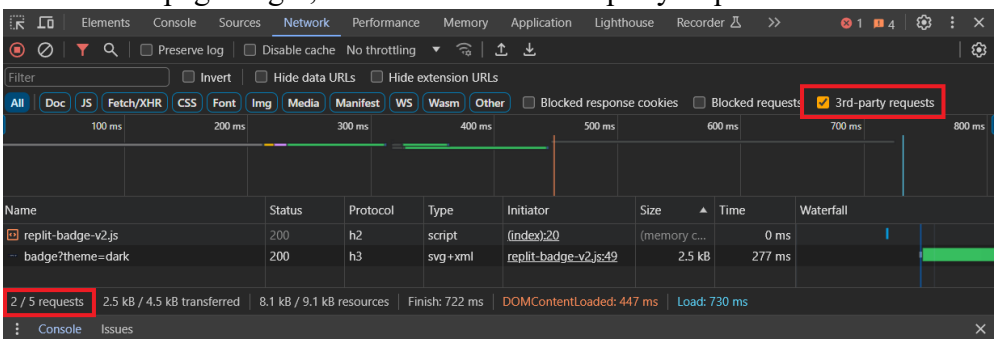
1. **Blocked response cookies:**
   To filter out everything except the requests with response cookies blocked for any reason, check Checkbox. Blocked response cookies.



2. **Blocked requests:** To filter out everything except blocked requests, check Checkbox. Blocked requests. To test this, you can use the Network request blocking drawer tab.

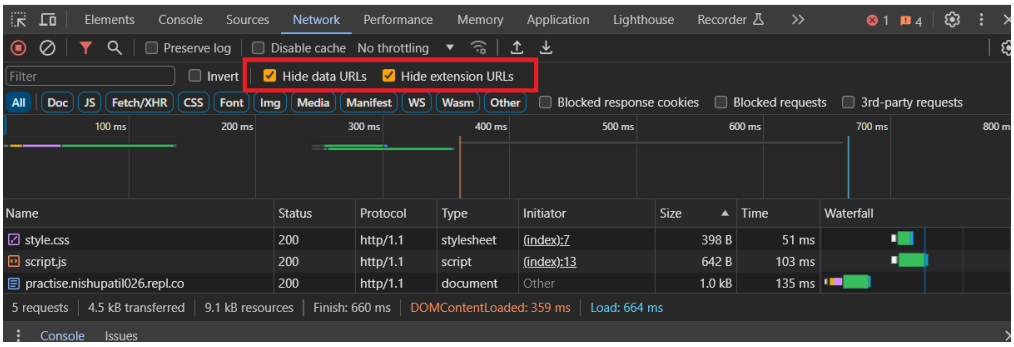


3. **3rd-party requests:** To filter out everything except the requests with origin that differs from page origin, check Checkbox. 3rd-party requests.



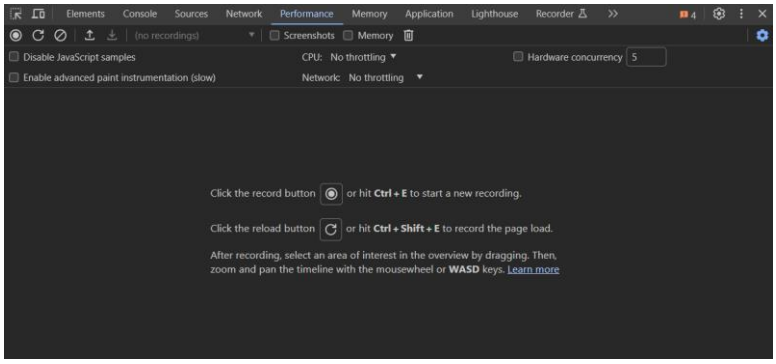
K. Hide data URLs and extension URLs:

1. **Hide data URLs:** Data urls are small files embedded into other documents. Any request that you see in the Requests table that starts with data: is a data URL. To hide these requests, check Checkbox. Hide data URLs.Data URLs hidden from the Requests table.The status bar at the bottom displays the number of the shown requests out of the total.
2. **Hide extension URLs**: To focus on the code you author, you can filter out irrelevant requests sent by extensions you may have installed in Chrome. Extension requests have URLs that start with chrome-extension://.To hide extension requests, check Checkbox. Hide extension URLs.


Elements
Console
Sources
Network
Performance
Memory
Application
Lighthouse
Recorder
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
Hide extension URLs
All
Doc
JS
Fetch/XHR
CSS
Font
Img
Media
Manifest
WS
Wasm
Other
Blocked response cookies
Blocked requests
3rd-party requests
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
Name
Status
Protocol
Type
Initiator
Size
Time
Waterfall
style.css
200
http/1.1
stylesheet
(index):7
398 B
51 ms
script.js
200
http/1.1
script
(index):13
642 B
103 ms
practise.nishupatil026.repl.co
200
http/1.1
document
Other
1.0 kB
135 ms
5 requests
4.5 kB transferred
9.1 kB resources
Finish: 660 ms
DOMContentLoaded: 359 ms
Load: 664 ms
Console
Issues

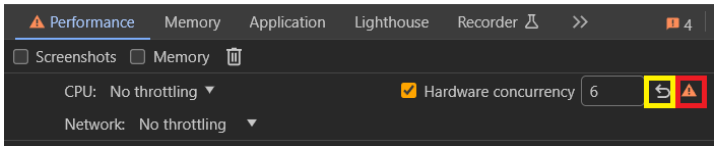
5. PERFORMANCE PANEL:

The Performance panel in browser DevTools is a powerful tool for analyzing and optimizing the performance of web pages and web applications. It provides insights into various aspects of page rendering, JavaScript execution, loading times, and more.



In above figure you can enable the checkboxes according to the need.

1. Enable the **Screenshots** checkbox to capture a screenshot of every frame while recording.
2. Click on **Disable JavaScript** samples to disable JS.
3. You can set the **CPU** and **Network** Throttling options.
4. To test application performance with different numbers of processor cores, you can configure the **Hardware concurrency** value. DevTools displays a warning icon next to the Performance tab to remind you that hardware concurrency emulation is enabled. To revert to the default value, click the Revert button.



Example: (Analyzing performance of a website)

- **Setting up the demo:**

i. Load the following page in your Incognito window. The page shows a bunch of little blue squares moving up and down.
https://googlechrome.github.io/devtools-samples/jank/

ii. Keep clicking Add 10 until the blue squares move noticeably slower than before. On a high-end machine, it may take about 20 clicks.
Click Optimize. The blue squares should move faster and more smoothly.

iii. Click Un-Optimize. The blue squares move slower and with more jank again.

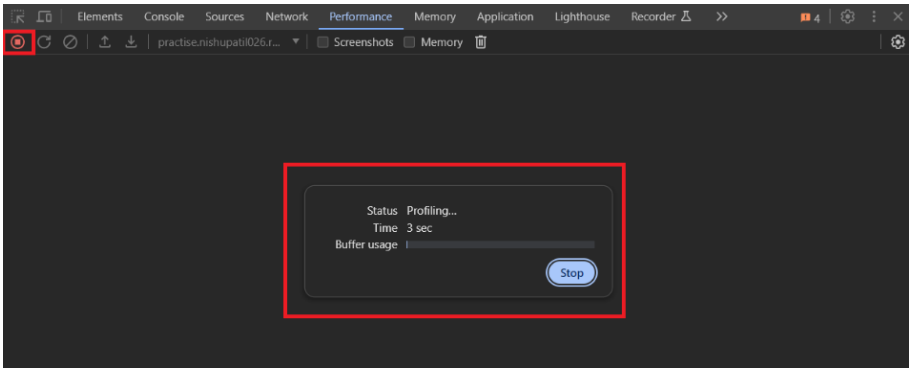
➢ **Features:**

A. Recording Timeline:

Captures and visualizes the performance timeline, displaying activities like loading, scripting, rendering, painting, and more. Shows key events like DOMContentLoaded, Load, and major rendering activities.

ii. Click Record option. DevTools captures performance metrics as the page runs.

iii. Click Stop. DevTools stops recording, processes the data, then displays the results on the Performance panel.



Result of Profiling:

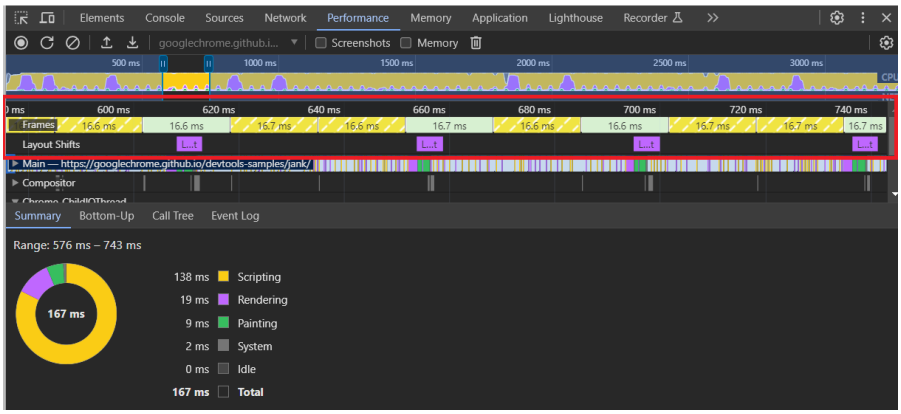
1. Analyzing the results:

Once you've got a recording of the page's performance, you can measure how poor the page's performance is, and find the cause(s).

a) Analyze frames per second:

The main metric for measuring the performance of any animation is frames per second (FPS). Users are happy when animations run at 60 FPS.

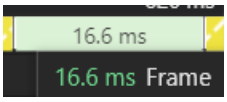
Look at the FPS chart. Whenever you see a red bar above FPS, it means that the framerate dropped so low that it's probably harming the user experience. In general, the higher the green bar, the higher the FPS.



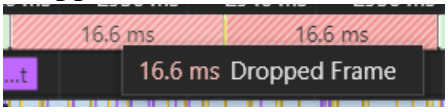
In the Frames section, hover your mouse over one of the squares. DevTools shows you the FPS for that particular frame. Each frame is probably well below the target of 60 FPS.

Partially Presented Frame

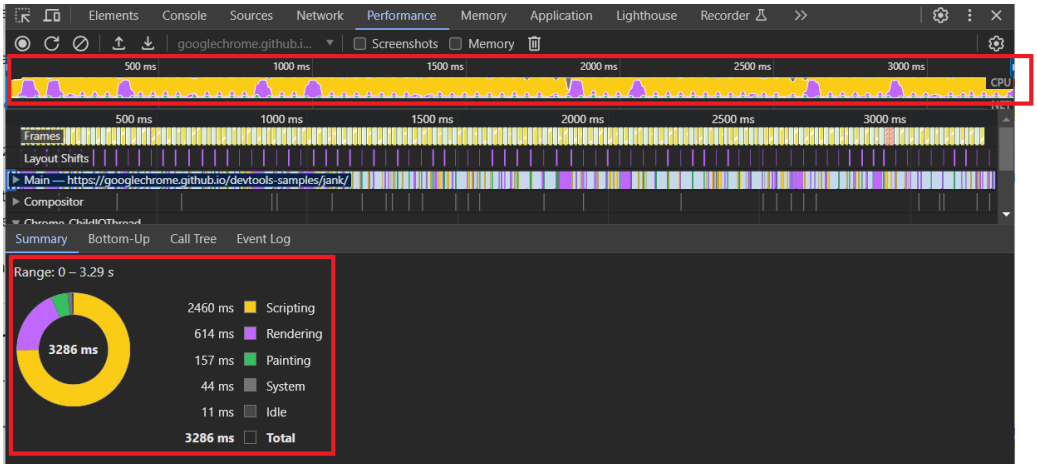
Frame



Dropped Frame



Below the FPS chart you see the CPU chart. The colors in the CPU chart correspond to the colors in the Summary tab, at the bottom of the Performance panel. The fact that the CPU chart is full of color means that the CPU was maxed out during the recording. Whenever you see the CPU maxed out for long periods, it's a cue to find ways to do less work.



b) Open the FPS meter:

Another handy tool is the FPS meter, which provides real-time estimates for FPS as the page runs.

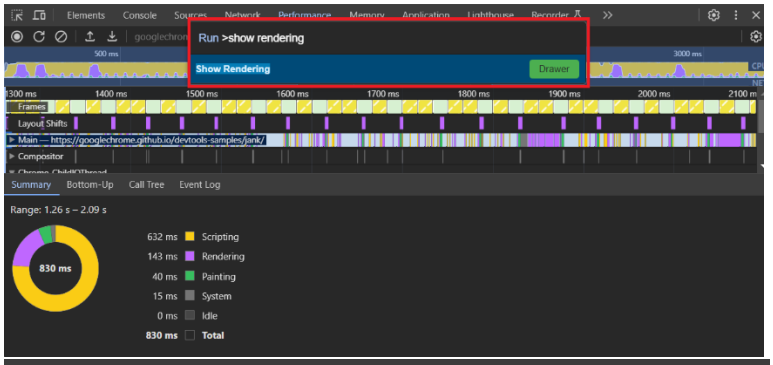
Steps:

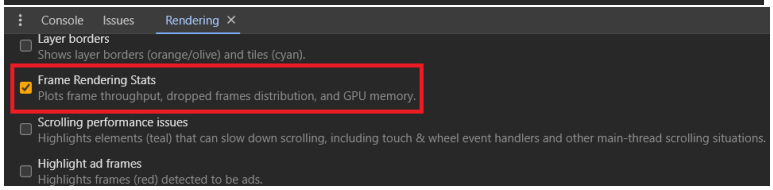
i. Press Command+Shift+P (Mac) or Control+Shift+P (Windows, Linux) to open the Command Menu.

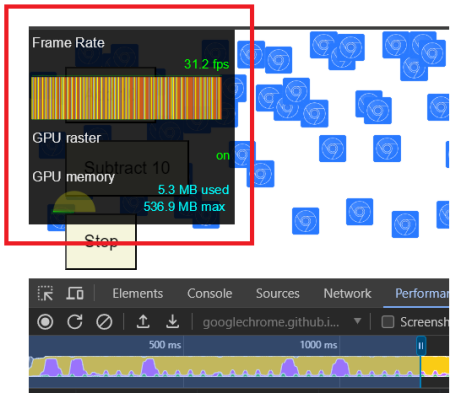
ii. Type show rendering in the Command Menu and click on Drawer.

iii. The Rendering tab will open, enable FPS rendering stats. A new overlay appears in the top-right of your viewport.

iv. Disable it.

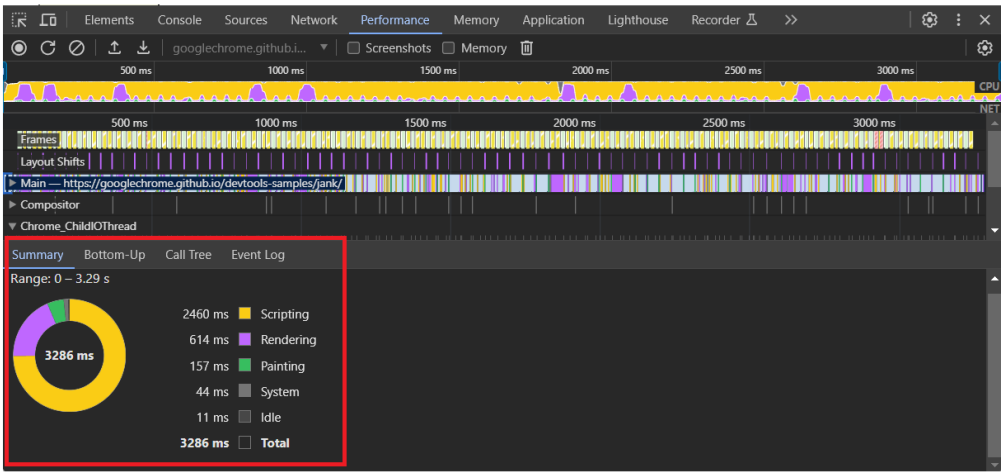




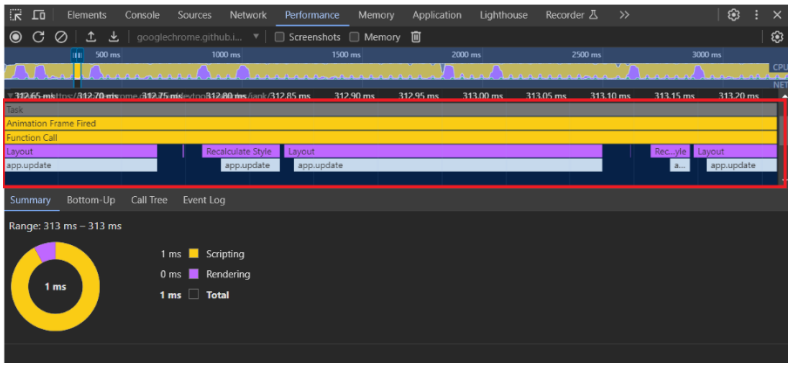


B. Finding the bottleneck:

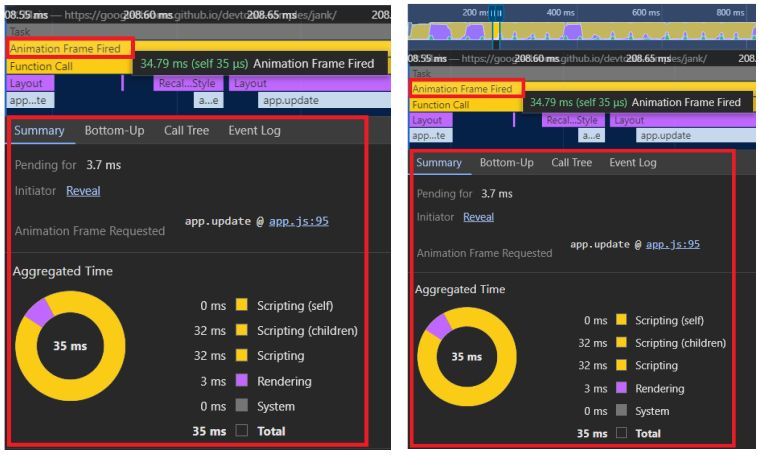
1. When no events are selected, summary tab shows a breakdown of activity. The page spent most of its time scripting.



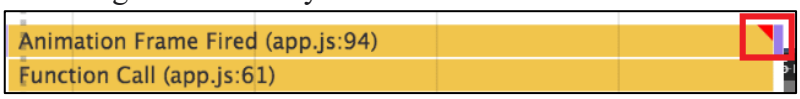
2. Zoom in on a single **Animation Frame Fired** event by clicking, holding, and dragging your mouse over the Overview, which is the section that includes the FPS, CPU, and NET charts. The Main section and Summary tab only display information for the selected portion of the recording.



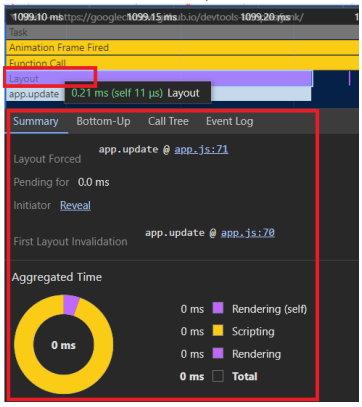
3. Click the **Animation Frame Fired** event. The Summary tab now shows you information about that event. Note the reveal link. Clicking that causes DevTools to highlight the event that initiated the Animation Frame Fired event. Also note the app.js:95 link. Clicking that jumps you to the relevant line in the source code. Similar for the function call.



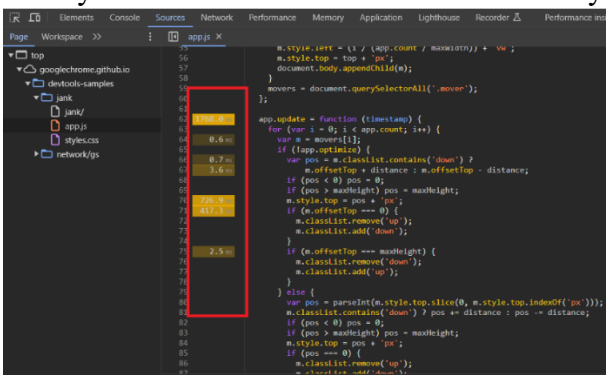
If there is a red triangle in the top-right of the **Animation Frame Fired** event, it's a warning that there may be an issue related to this event.



4. Click one of the **purple Layout** events now. DevTools provides more information about the event in the Summary tab. Indeed, there's a warning about forced reflows (another word for layout).

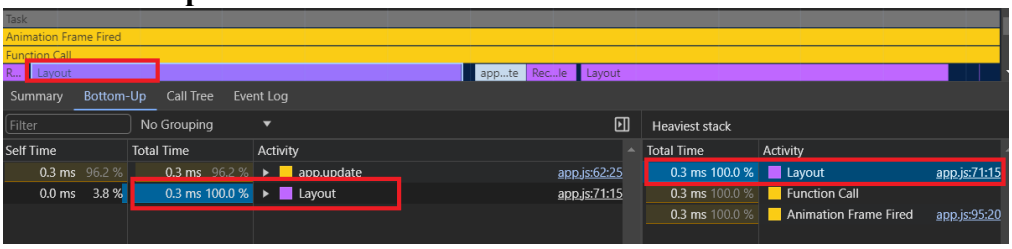


5. In the Summary tab, click the app.js:70 link under Layout Forced. DevTools takes you to the line of code that forced the layout



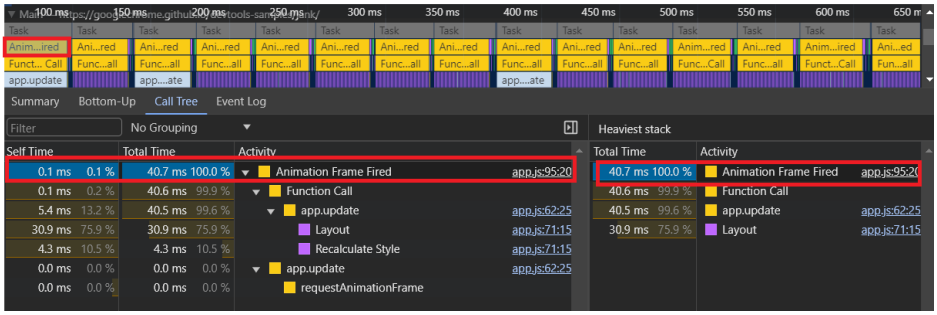
C. View activities in a table (Main):

1. When you want to view the activities where the most time was directly spent, use the **Bottom-Up tab.**

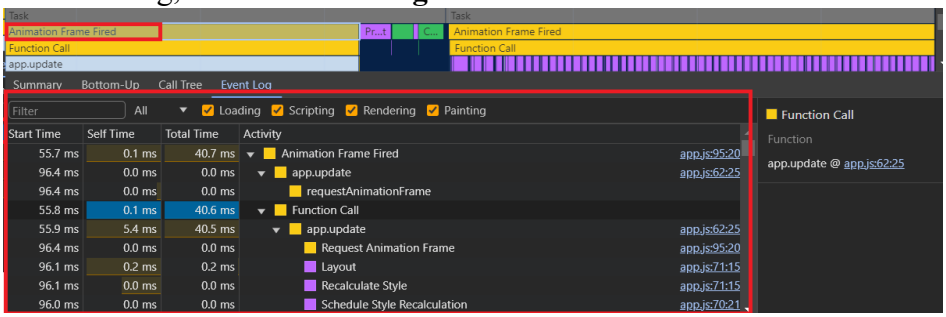


- The **Self Time** column represents the aggregated time spent directly in that activity, across all of its occurrences.
- The **Total Time** column represents aggregated time spent in that activity or any of its children.

2. When you want to view the root activities that cause the most work, use the **Call Tree tab.**



3. When you want to view the activities in the order in which they occurred during the recording, use the **Event Log tab.**



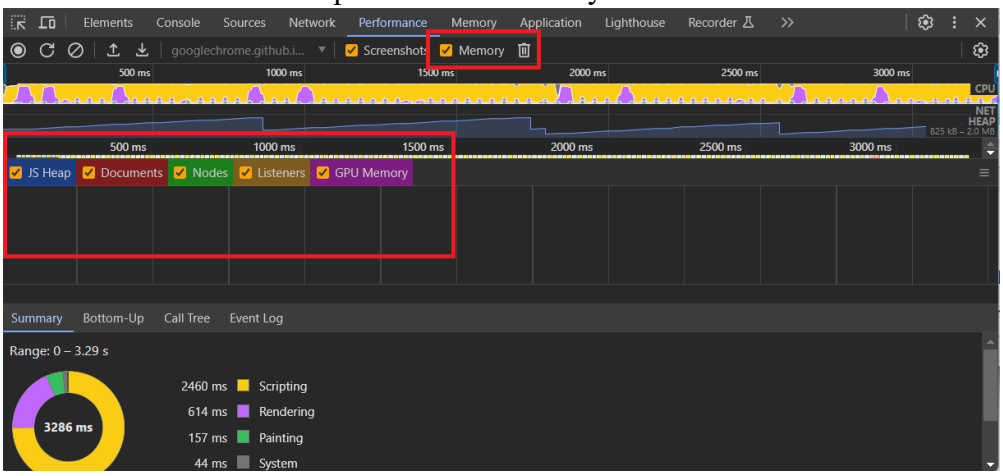
- The **Start Time** column represents the point at which that activity started, relative to the start of the recording.
- The **Self Time** column represents the time spent directly in that activity.
- The **Total Time** columns represents time spent directly in that activity or in any of its children.
- Click Start Time, Self Time, or Total Time to sort the table by that column.
- Use the Filter text box to filter activities by name.
- Use the Duration menu to filter out any activities that took less than 1 ms or 15 ms. By default the Duration menu is set to All, meaning all activities are shown.
- Disable the Loading, Scripting, Rendering, or Painting checkboxes to filter out all activities from those categories.

Each tabular view in the Performance panel shows links for activities such as functions calls. To help you debug, DevTools finds the corresponding function declarations in source files. Additionally, if the appropriate source maps are present and enabled, DevTools automatically finds the original files.

Click a link to open a source file in the Sources panel.

D. View Memory Metrics:

DevTools displays a new Memory chart, above the Summary tab. There's also a new chart below the NET chart, called HEAP. The HEAP chart provides the same information as the JS Heap line in the Memory chart.



6. MEMORY PANEL:

The Memory panel in browser DevTools is a tool designed to help developers analyze and optimize memory usage in web applications. It provides insights into JavaScript memory allocation, heap snapshots, and helps identify memory leaks and inefficient memory usage.

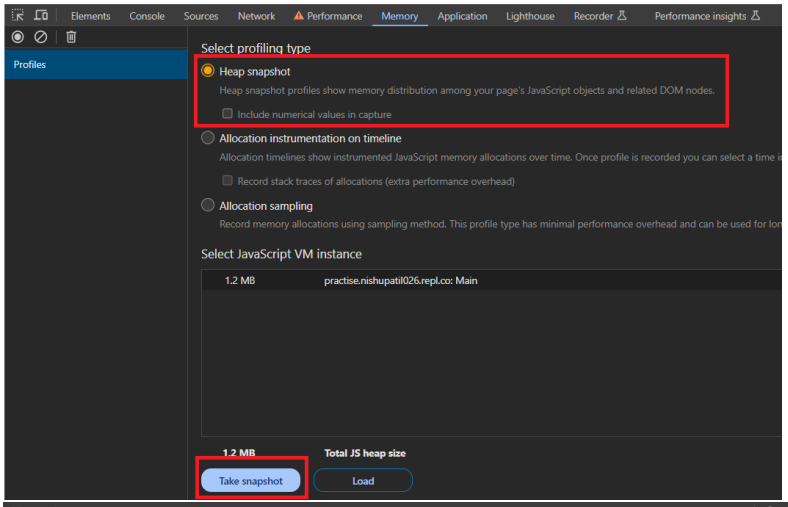
➢ **Features:**

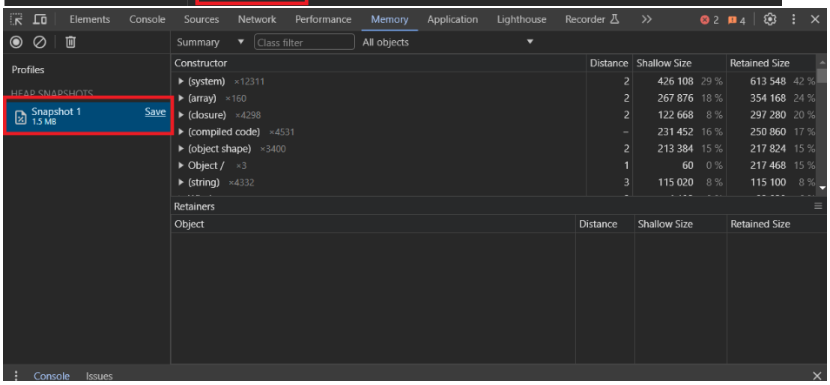
A. Heap Snapshot:

Captures a snapshot of the JavaScript memory heap at a specific point in time.Shows memory allocation details for objects, functions, and closures, aiding in memory analysis.

1. Take Snapshot:

To create a snapshot, select the Heap Snapshot radio button, and then press the "Take Snapshot" button.





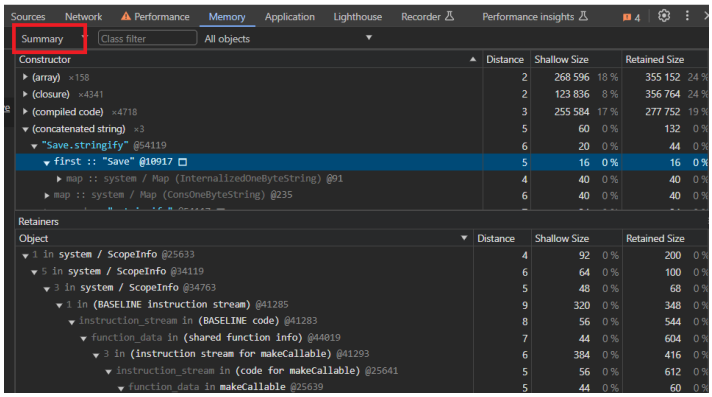
Here is a short overview of the columns in this table:

- **Constructor:** JavaScript function used to construct the objects.
- **Distance:** distance from the root. The greater the distance is, the longer the object takes to load and process.
- **Objects Count:** the number of objects created by the specified constructor.
- **Shallow Size:** displays the sum of shallow sizes of all objects created by a certain constructor function. The shallow size is the size of memory held by an object itself (generally, arrays and strings have larger shallow sizes).
- **Retained Size:** displays the maximum retained size among the same set of objects. The size of memory that can be freed once an object is deleted (and this its dependents made no longer reachable) is called the retained size.

2. **View snapshots:**

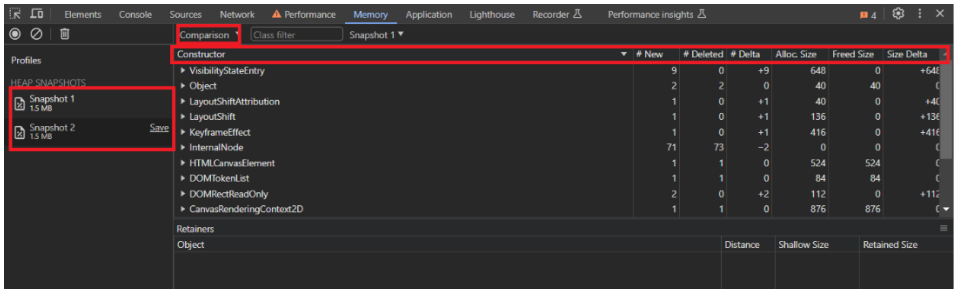
View snapshots from different perspectives for different tasks.

a) **Summary view** shows objects grouped by the constructor's name. Use it to hunt down objects (and their memory use) based on type grouped by constructor name. It's particularly helpful for tracking down DOM leaks.



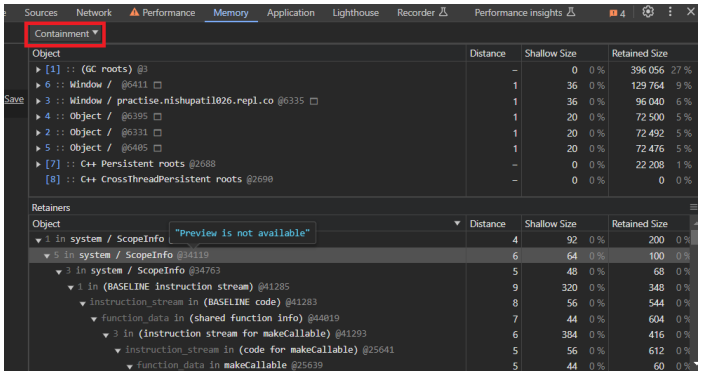
b) **Comparison View** displays difference between two snapshots. Use it to compare two (or more) memory snapshots of before and after an operation. Inspecting the delta in freed memory and reference count lets you confirm the presence and cause of a memory leak.

Take a second heap snapshot and change the view of this one to Comparison, comparing it to snapshot 1.

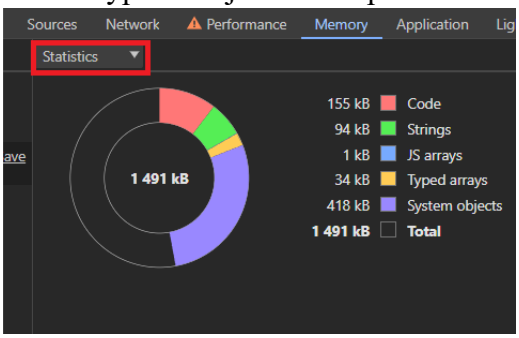


Here is a short overview of the table columns in the Comparison view:
Constructor: the JavaScript function used to construct a set of objects.
#New: how many new objects have been created.
#Deleted: how many objects have been deleted.
#Delta: the change in the overall number of objects.
Alloc. Size: how much memory has been allocated to be used.
Freed Size: how much memory has been freed for new objects.
Size Delta: the change in the overall amount of free memory.

c) **Containment view** allows exploration of heap contents. It provides a better view of object structure, helping analyze objects referenced in the global namespace (window) to find out what is keeping them around. Use it to analyze closures and dive into your objects at a low level.

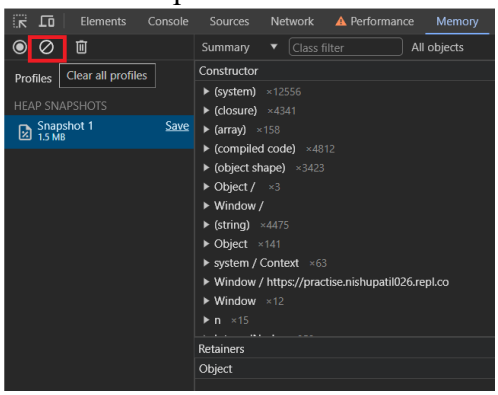


d) Statistics View displays a circular chart, which shows how much memory each type of object that is present takes up overall.

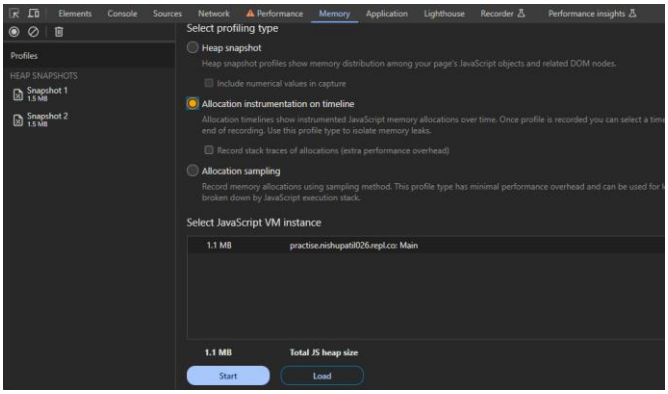


3. Clearing snapshots:

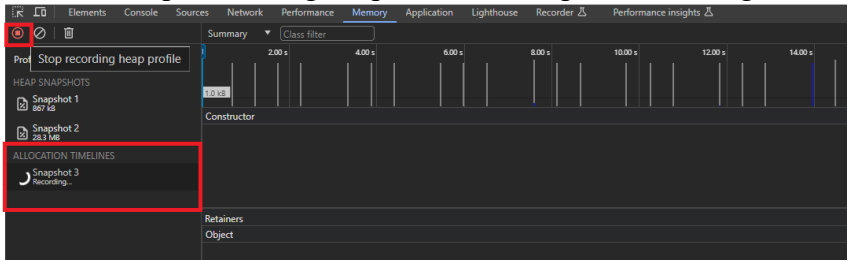
Remove snapshots (both from DevTools and renderers memory) by pressing the Clear all profiles icon:



B. Allocation instrumentation on timeline: Visualizes memory allocation events and helps track memory usage over time. Identifies when and where memory is allocated and deallocated. The Allocations Timeline shows real-time allocation activity during interaction. Spikes may indicate inefficient operations. Filtering to a specific component isolates its impact. Select the **Allocation instrumentation on timeline** radio button, and then press the "Start" button.

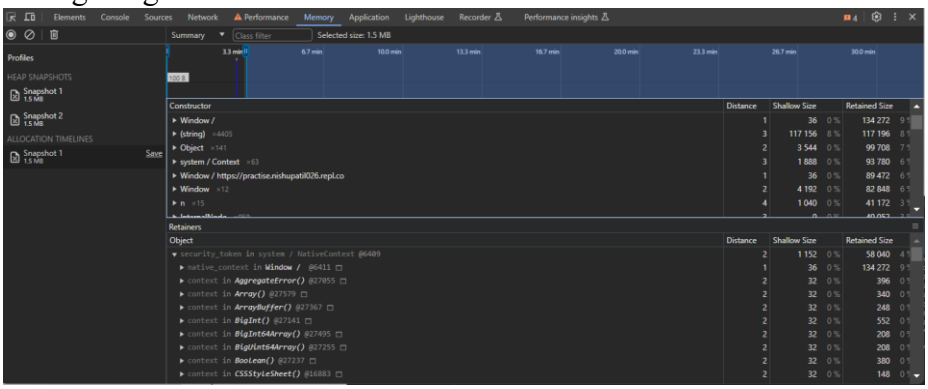


Click on Stop Recording heap Profile to stop the recording and to get the results.



This profile shows where objects are being created and identifies the retaining path. In the snapshot below, the bars at the top indicate when new objects are found in the heap.

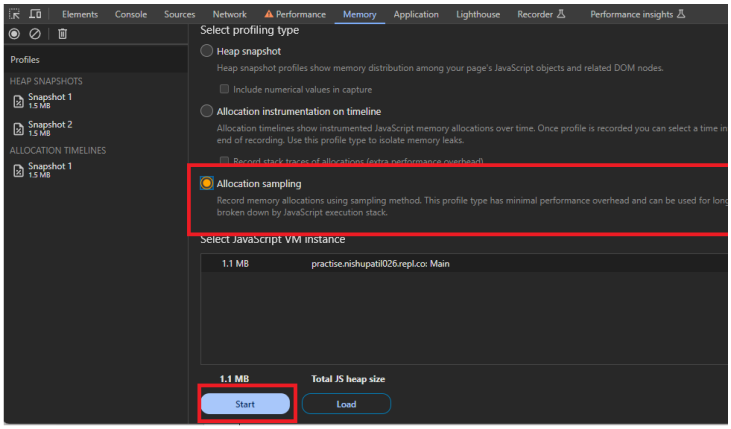
The height of each bar corresponds to the size of the recently allocated objects, and the color of the bars indicate whether or not those objects are still live in the final heap snapshot. Blue bars indicate objects that are still live at the end of the timeline, gray bars indicate objects that were allocated during the timeline, but have since been garbage collected:



C. Allocation sampling:

Monitors memory allocation patterns and provides insights into memory usage trends. Helps identify memory-heavy operations or code segments. The Allocation Sampling view maps memory usage to individual page components like documents, frames, web workers, and graphics layers. This reveals the source of any high usage. This combines the detailed snapshot information of the heap profiler with the incremental updating and tracking of the Performance panel.

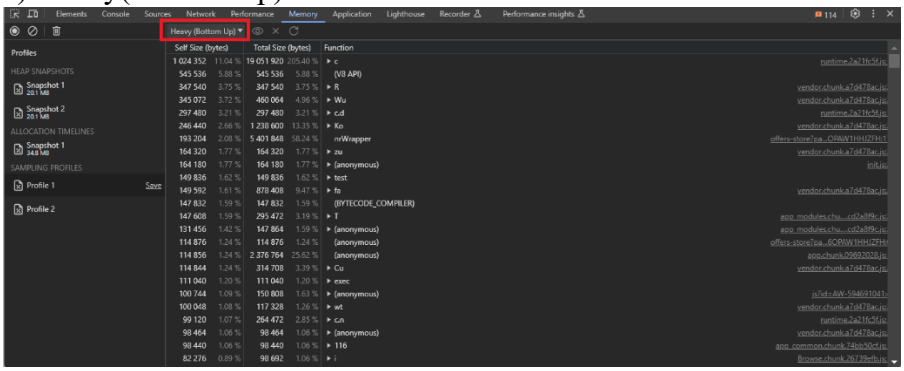
Select the **Allocation sampling** radio button, and then press the “Start” button.



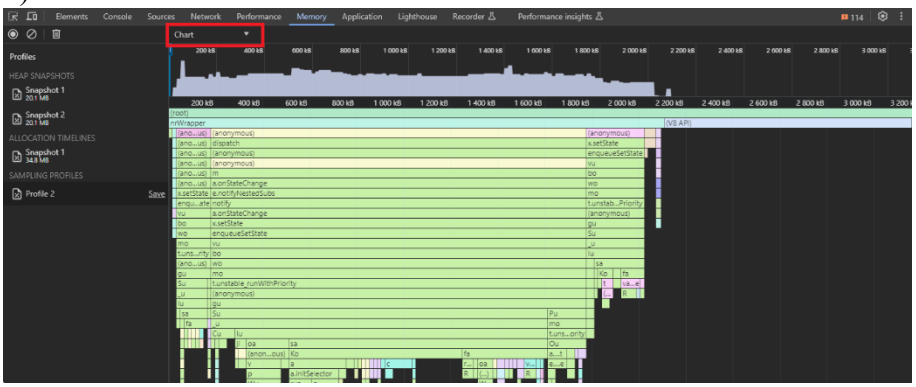
Tracking objects' heap sampling usually involves starting a recording, performing a sequence of actions, then stop the recording for analysis.

Views:

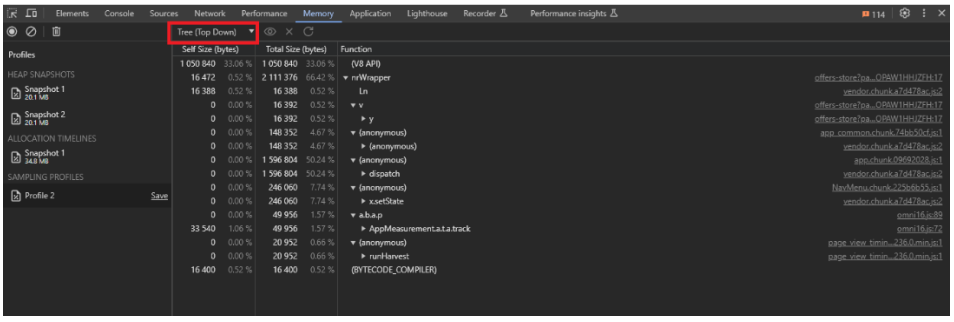
a) Heavy(Bottom Up):



b) Chart

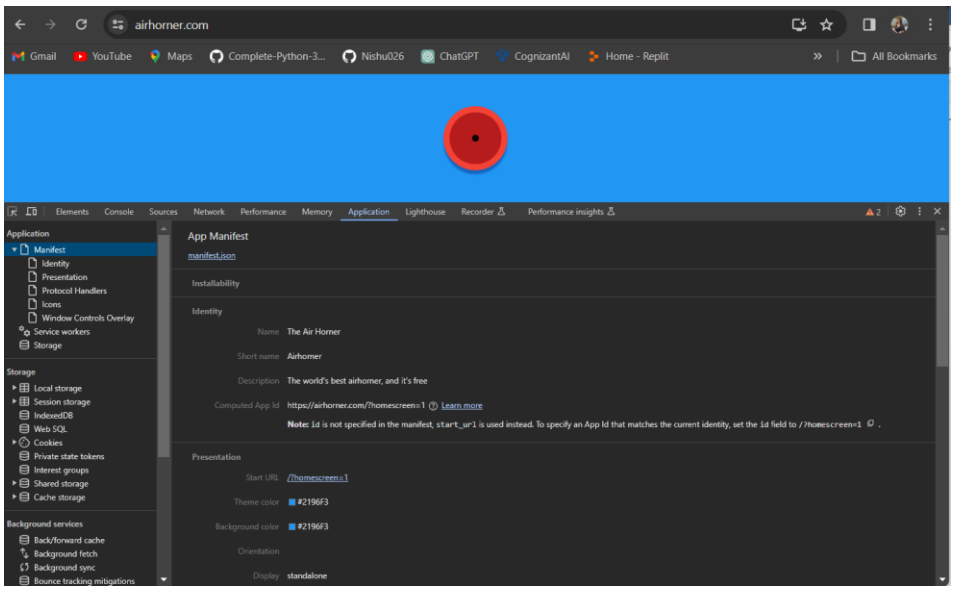


c) Tree (Top Down)



7. APPLICATION PANEL:

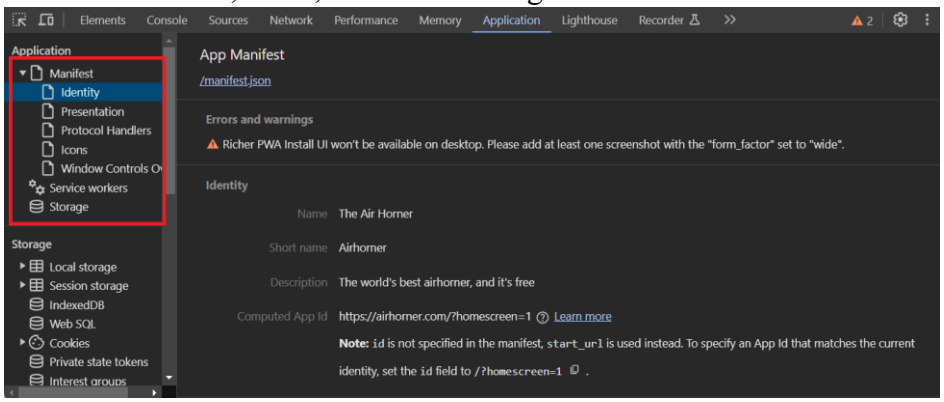
The Application panel in browser developer tools is a powerful toolset that helps developers inspect, debug, and manage web applications. It primarily focuses on handling various aspects related to the application lifecycle, such as managing storage, debugging service workers, examining caches, managing cookies, and more.



The application panel consist of:

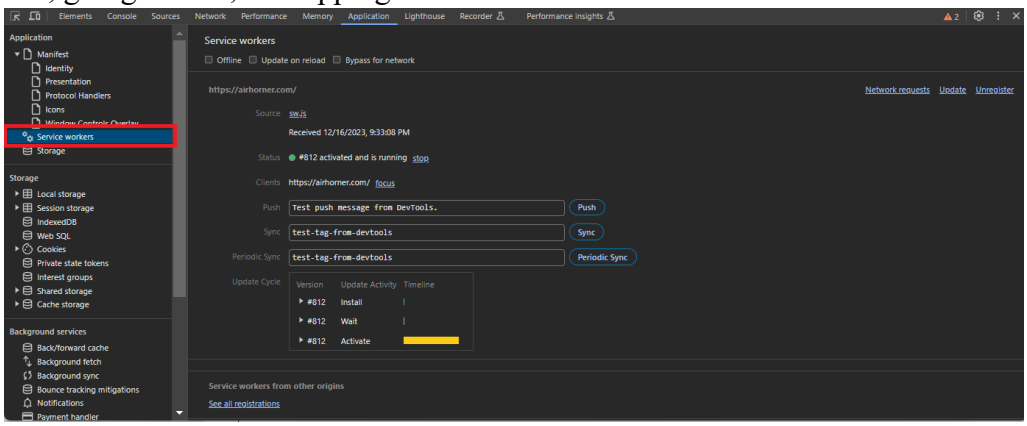
A. Application:

1. **Manifest:** Use the Manifest pane to inspect your web app manifest.If your web app has a manifest file for Progressive Web Apps (PWAs), this section shows details like the name, icons, and other settings defined in the manifest.

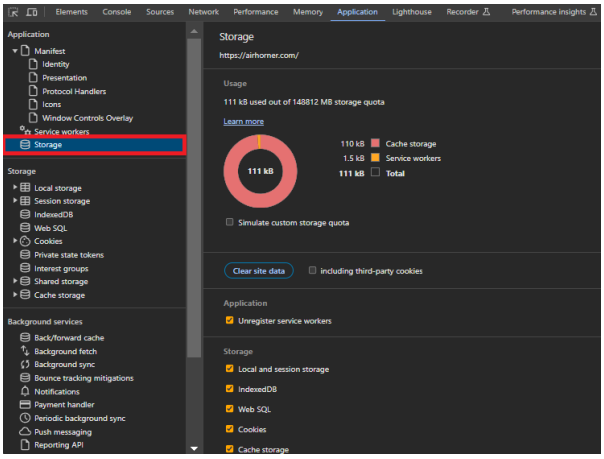


   - The **Identity** and **Presentation** sections just display fields from the manifest source in a more user-friendly way.
   - The **Protocol Handlers** section lets you to test the URL protocol handler registration of your PWA with a click of a button. To learn more, see Test URL protocol handler registration.
   - The **Icons** section displays every icon that you've specified and lets you check their masks.
   - The Shortcut #N set of sections displays information on all your shortcut objects.
   - The Screenshot #N set of section displays the screenshots for a richer installation UI of your app.

2. **Service workers:** Use the Service Workers pane for a whole range of service-worker-related tasks, like unregistering or updating a service, emulating push events, going offline, or stopping a service worker.

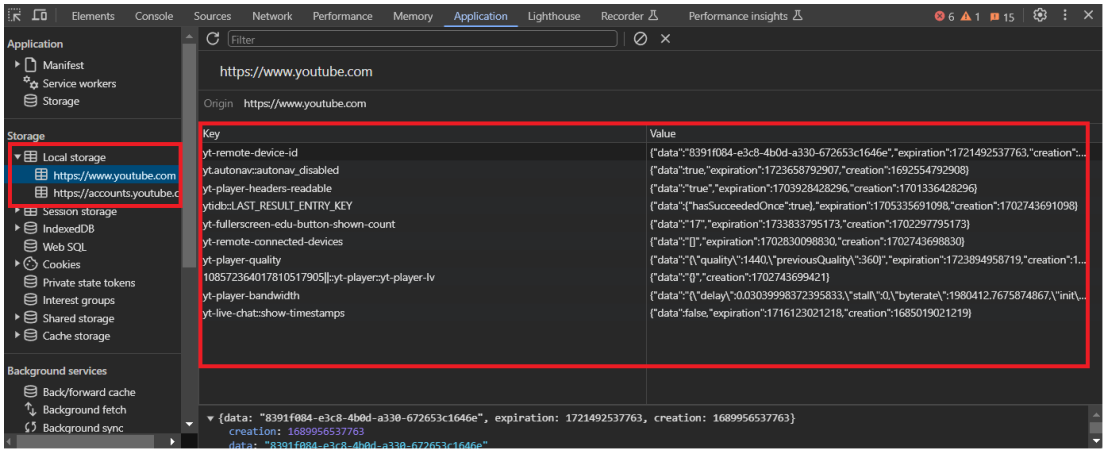


3. **Storage:** View your service worker cache from the Cache Storage pane. Unregister a service worker and clear all storage and caches with a single button click from the Clear storage pane.

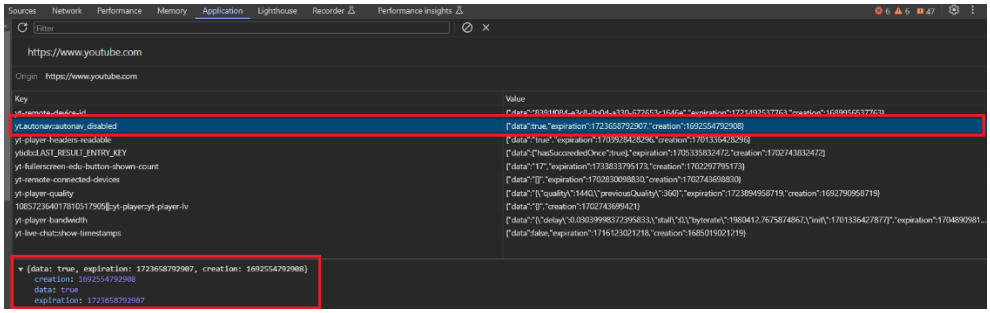


B. Storage Inspection:

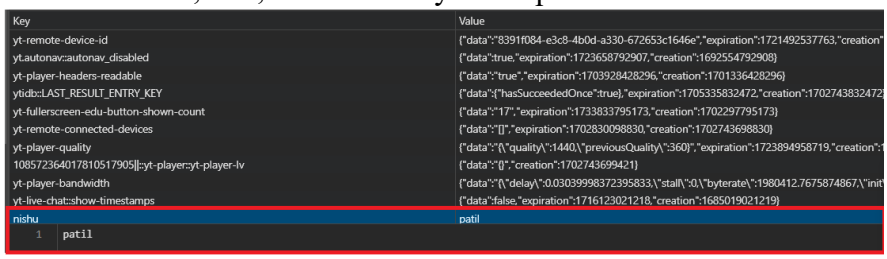
1. **LocalStorage/SessionStorage:** View and manipulate data stored in these storage mechanisms.



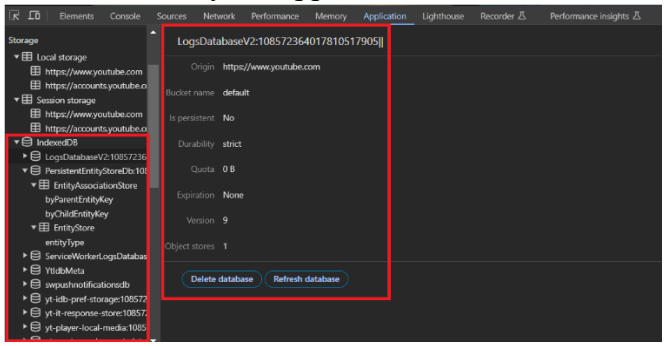
To preview the value below the table, select a pair.



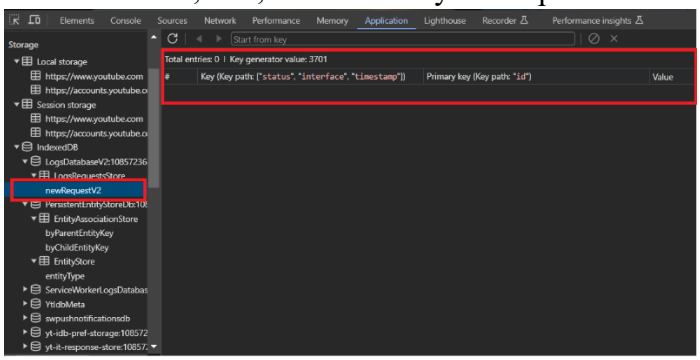
You can create, edit, delete the key value pairs.



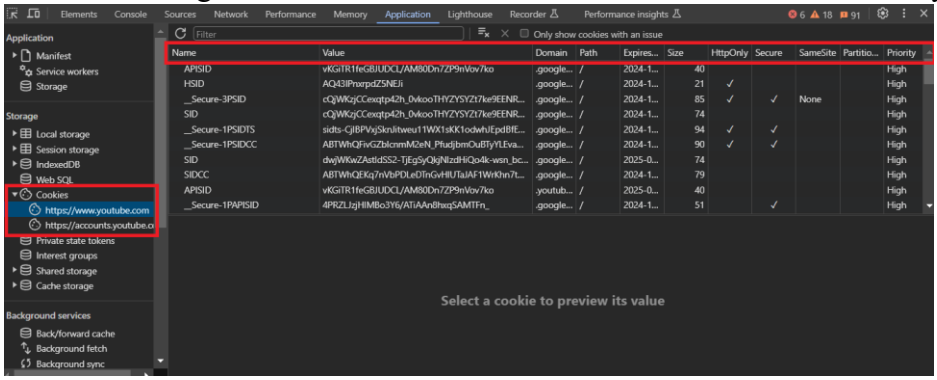
2. **IndexedDB/WebSQL:** Inspect and modify data stored in IndexedDB or WebSQL databases used by the application.



Click an object store to see its key-value pairs.
You can create, edit, delete the key-value pairs.



3. **Cookies:** Manage cookies, view their details, and delete them if necessary.



The Cookies table contains the following fields:

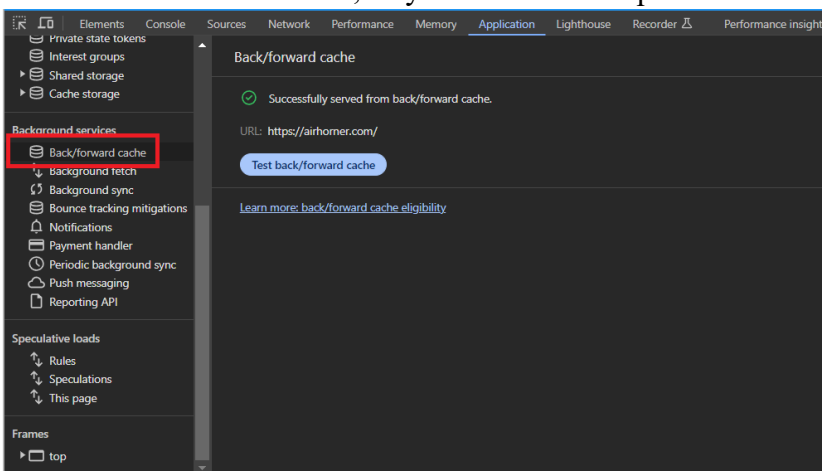
- **Name:** The cookie's name.
- **Value:** The cookie's value.
- **Domain**: The hosts that are allowed to receive the cookie.
- **Path**: The URL that must exist in the requested URL in order to send the Cookie header.
- **Expires / Max-Age**: The cookie's expiration date or maximum age. For session cookies this value is always Session.
- **Size:** The cookie's size, in bytes.
- **HttpOnly**: If true, this field indicates that the cookie should only be used over HTTP, and JavaScript modification is not allowed.
- **Secure**: If true, this field indicates that the cookie can only be sent to the server over a secure, HTTPS connection.
- **SameSite**: Contains Strict or Lax if the cookie is using the experimental SameSite attribute.
- **Partition Key:** For cookies with independent partition state, the partition key is the site of the top-level URL the browser was visiting at the start of the request to the endpoint that set the cookie.
- **Priority:** Contains Low, Medium (default), or High if using deprecated cookie Priority attribute.
- You can create, edit, delete the cookies.

C. Background Services:

The Background services section of Chrome DevTools is a collection of tools for the JavaScript APIs that enables your website to send and receive updates even when a user does not have your website open. A background service is functionally similar to a background process.

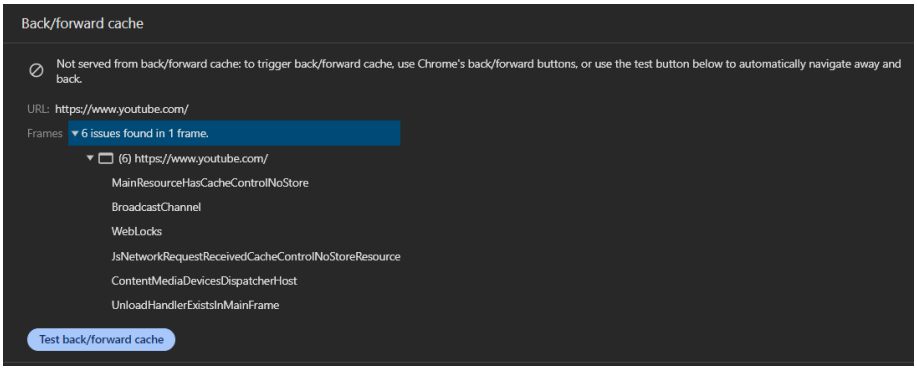
The Background services section lets you debug the following background services:

1. **Back/forward cache:** Back/forward cache (or bfcache) is a browser optimization that enables instant back and forward navigation. It significantly improves the browsing experience for users—especially those with slower networks or devices. As web developers, it's critical to understand how to optimize your pages for bfcache across all browsers, so your users can reap the benefits.

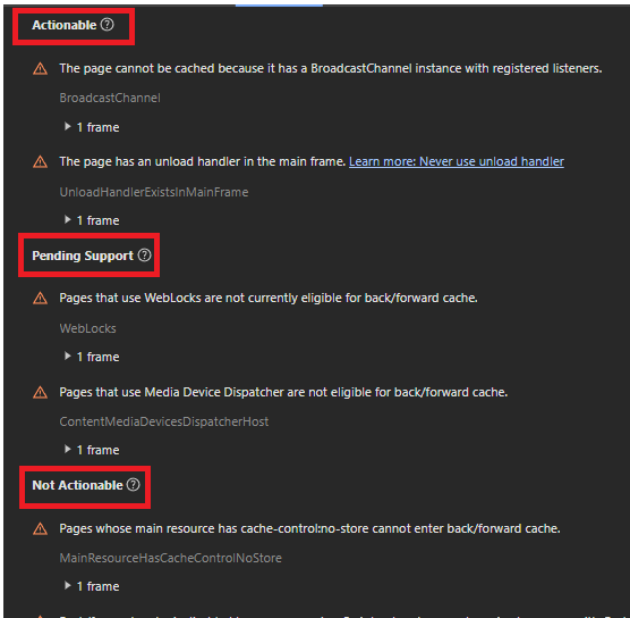


Otherwise, you'll see a list of issues.

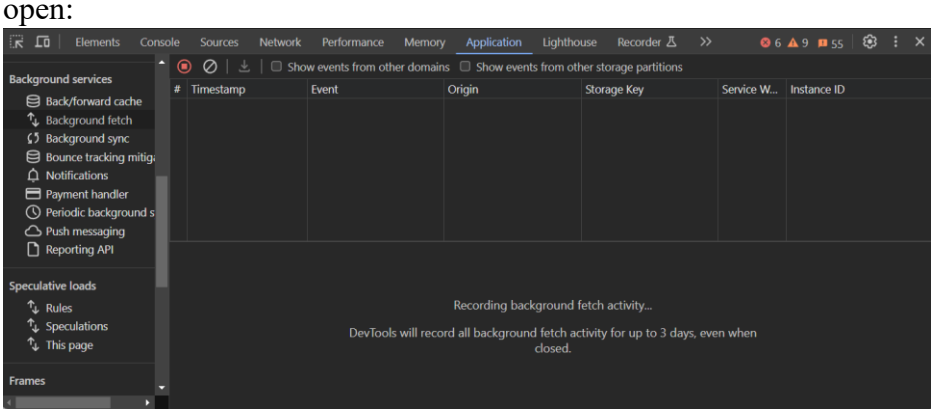
To find out which issues affect which frames, expand the Frames Expand icon. section.



If your page isn't eligible for back/forward caching, the Back/forward cache tab shows you a list with three types of causes:

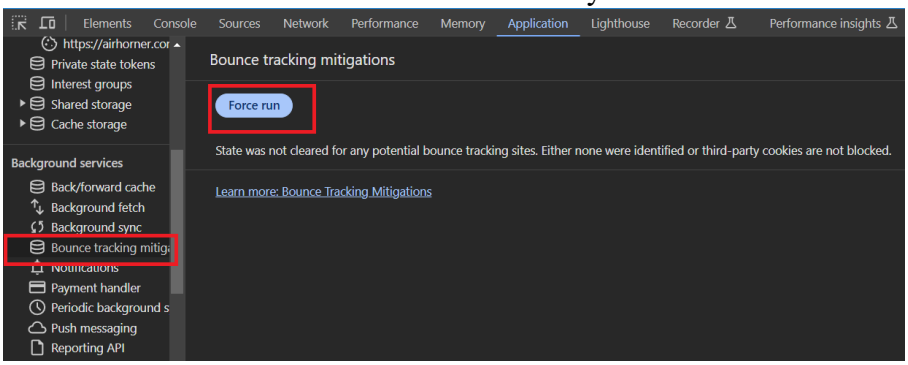


2. **Background fetch:**The Background Fetch API enables a service worker to reliably download large resources, like movies or podcasts, as a background service. To log background fetch events for three days, even when DevTools isn't open:



3. **Background sync:**The Background Sync API enables an offline service worker to send data to a server once it has re-established a reliable internet connection. To log background sync events for three days, even when DevTools isn't open.

4. **Bounce tracking mitigations:** Bounce tracking mitigations experiment in Chrome lets you identify and delete the state of sites that appear to perform cross-site tracking using the bounce tracking technique. You can manually force tracking mitigations and see a list of sites whose states were deleted.
   To force tracking mitigations:
   Block third-party cookies in Chrome. Navigate to and enable Three-dot menu. > Settings > Security. Privacy and security > Cookies and other site data > Radio button checked. Block third-party cookies.
   In chrome://flags, set the Bounce tracking mitigations experiment to Enabled With Deletion.
   Open DevTools, for example, on demo page, and navigate to Application > Background services > Bounce tracking mitigations.
   On the demo page, click a bounce link and wait (10 seconds) for Chrome to record the bounce. The Issues tab warns you about the upcoming state deletion.
   Click Force run to delete the state immediately.

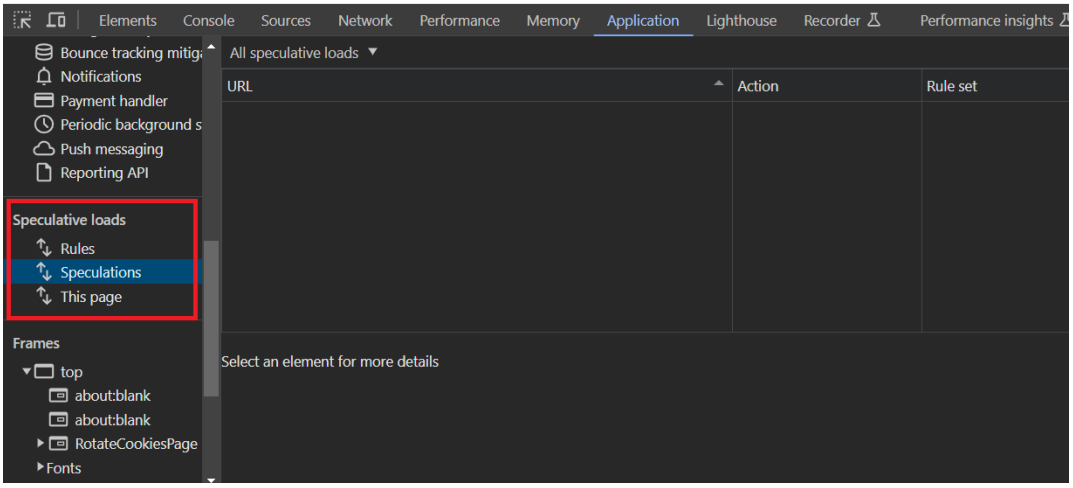


5. **Notifications:**After a service worker has received a Push Message from a server, the service worker uses the Notifications API to display the data to a user.
   To log Notifications for three days, even when DevTools isn't open:
   Open DevTools, for example, on this demo page. Navigate to Application > Background services > Notifications and click Record.
   The Notifications pane: On the demo page, click Schedule Notification and Allow when prompted. Wait for the notification to appear. DevTools logs the notification events to the table. Click an event to view its details in the space below the table.
   You can close DevTools and leave the recording run for up to three days. To stop recording, click Stop.
6. **Push messaging:** To display a push notification to a user, a service worker must first use the Push Message API to receive data from a server. When the service worker is ready to display the notification, it uses the Notifications API. To log push messages for three days, even when DevTools isn't open.
7. **Report API:**The Reporting API is designed to help you monitor deprecated API calls, security violations of your page, and more. You can set up reporting as described in Monitor your web application with the Reporting API.
   To view the reports generated by a page:Go to chrome://flags/#enable-experimental-web-platform-features, set Experimental Web Platform features to Enabled, and restart Chrome.
   Open DevTools and navigate to Application > Background services > Reporting API. For example, you can check out reports on this demo page.

D. Speculative Loads:

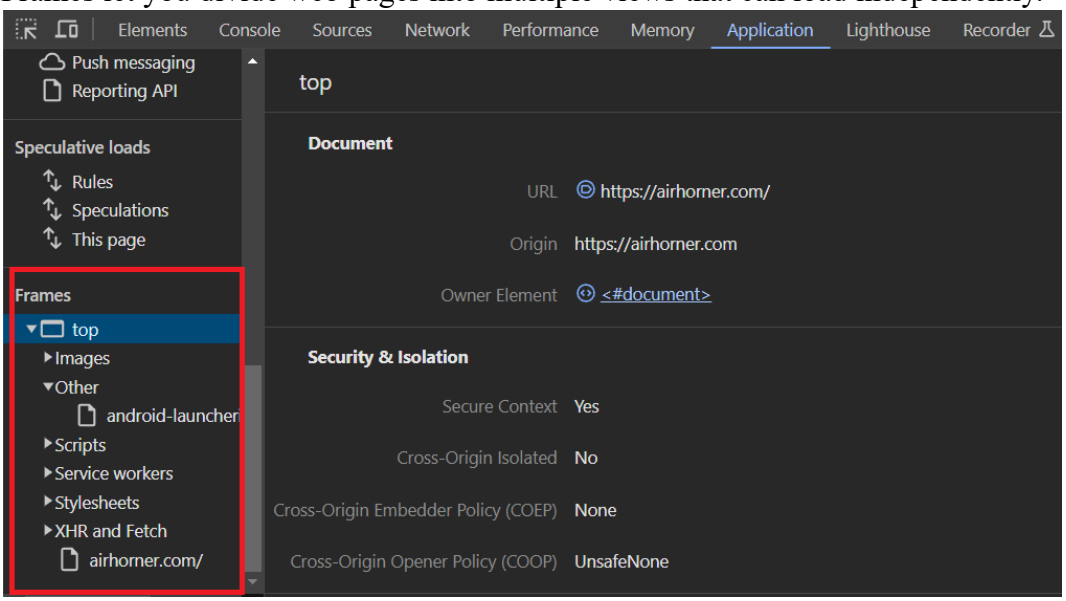
Speculative loads allow a near-instant page load based on speculation rules that you define. This lets your website prefetch and prerender most navigated-to pages. Prefetch fetches a resource in advance and prerender goes a step beyond and renders the whole page in a hidden background renderer process.

- **Speculative loads:** Contains the speculative status for the current page, current URL, pages that the current page attempts to load speculatively, and their statuses.
- **Rules:** Contains the rule sets on the current page in the Elements panel and the overall status of speculations.
- **Speculations:** Contains a table with information on speculative loading attempts and their statuses. If an attempt failed, you can click it in the table to see detailed information and failure reason.



E. Frames:

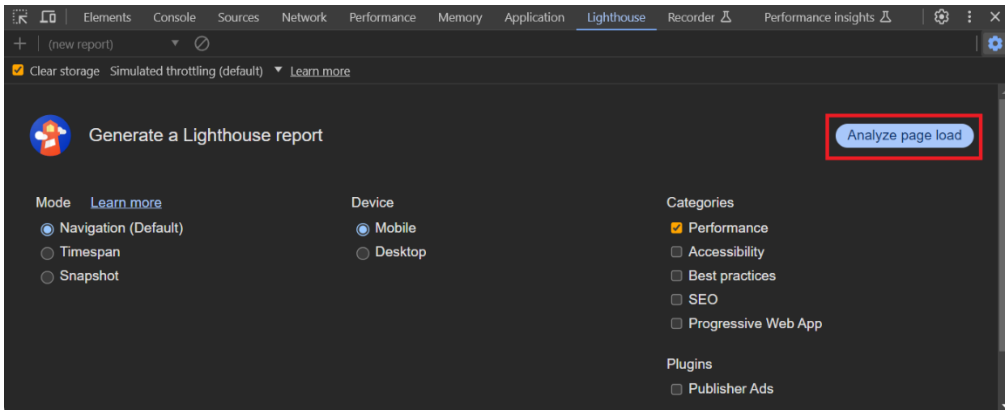
Frames let you divide web pages into multiple views that can load independently.



To see the details of a frame, select it in the Frames section.
The details include images, documents, script, stylesheets etc.

8. LIGHTHOUSE PANEL:

Lighthouse is a powerful tool in Google Chrome's DevTools that helps developers improve the quality of web pages by auditing various aspects of performance, accessibility, best practices, SEO, and more. The Lighthouse panel runs audits on a web page and generates a report with suggestions on how to enhance its performance and user experience.



Lighthouse report configuration settings consist of:

a) **Clear Storage:** Enabling this checkbox clears all storage associated with the page before every audit. Leave this setting on if you want to audit how first-time visitors experience your site. Disable this setting when you want the repeat-visit experience.

b) **Simulated throttling (default):** This option simulates the typical conditions of browsing on a mobile device. It's called "simulated" because Lighthouse doesn't actually throttle during the auditing process. Instead, it just extrapolates how long the page would take to load under mobile conditions. The DevTools throttling (advanced) setting, on the other hand, actually throttles your CPU and network, with the tradeoff of a longer auditing process.

c) **Mode > Navigation (Default):** This mode analyses a single page load. Navigation mode analyzes a single page load.
   - **Timespan mode** analyzes an arbitrary period of time, typically containing user interactions.
   - **Snapshot mode** analyzes the page in a particular state.

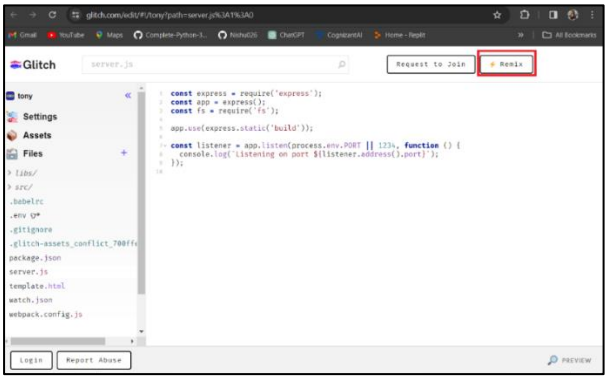
d) **Device > Mobile:** The mobile option changes the user agent string and simulates a mobile viewport. The desktop option pretty much just disables the mobile changes.

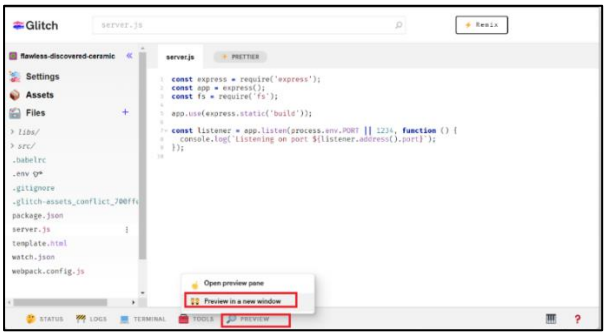
e) **Categories >Performance:** A single enabled category makes Lighthouse generate a report only with the corresponding set of audits. You can leave the other categories enabled, if you want to see the types of recommendations they provide. Disabling irrelevant categories slightly speeds up the auditing process.

f) **Click Analyze page load:** After 10 to 30 seconds, the Lighthouse panel shows you a report of the site's performance.

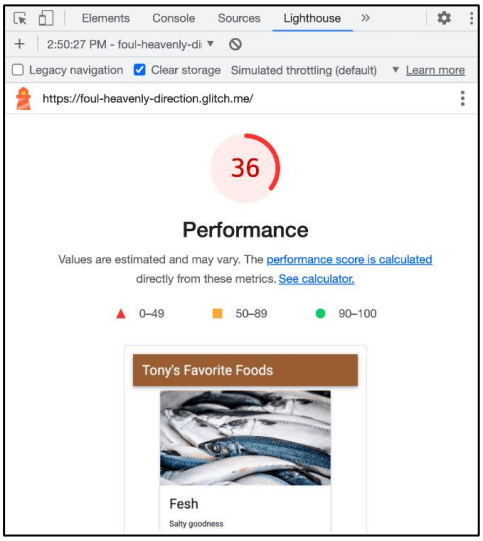
Example: Optimizing website speed.

i. Go to https://glitch.com/edit/#!/tony
ii. Click on Remix. Your new project opens in a tab. This tab will be referred to as the editor tab.



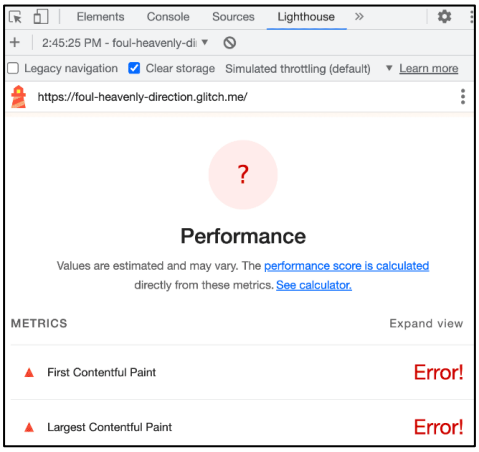


iii. Open DevTools > lighthouse.
iv. Click Analyze page load. After 10 to 30 seconds, the Lighthouse panel shows you a report of the site's performance.



Handling report errors:
If you ever get an error in your Lighthouse report, try running the demo tab from an incognito window with no other tabs open. This ensures that you're running Chrome

from a clean state. Chrome Extensions in particular can interfere with the auditing process.

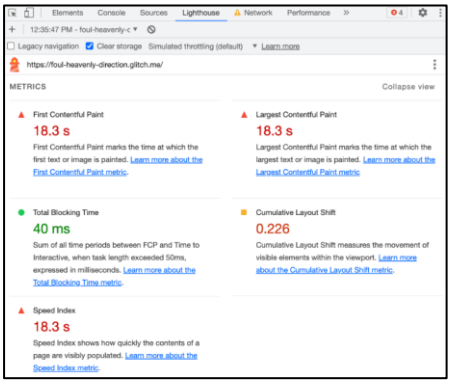


Understand your report:

The number at the top of your report is the overall performance score for the site. Later, as you make changes to the code, you should see this number rise. A higher score means better performance.

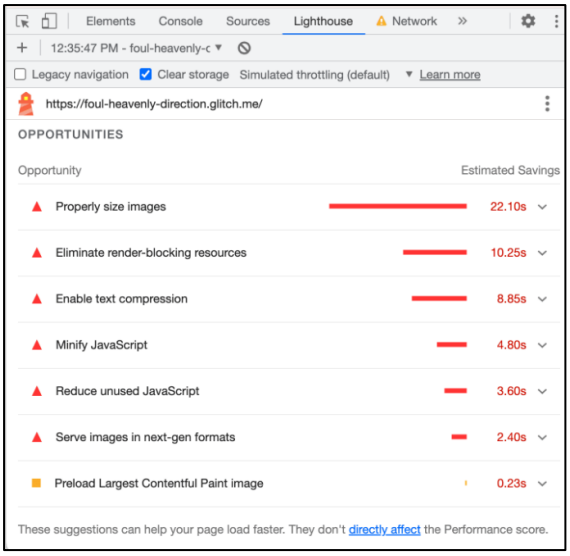
Metrics:

Scroll down to the Metrics section and click Expand view. To read documentation on a metric. This section provides quantitative measurements of the site's performance. Each metric provides insight into a different aspect of the performance.



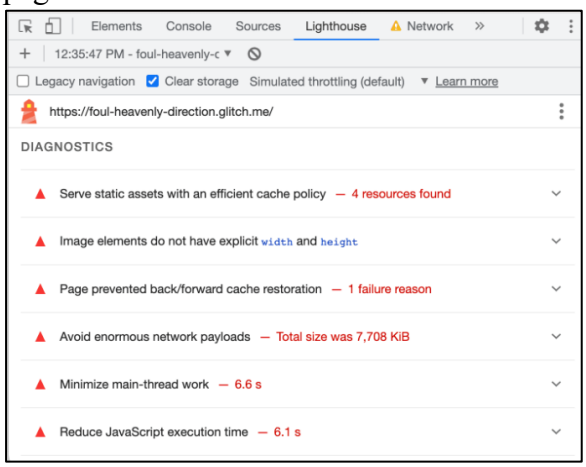
Opportunities:

Next is the Opportunities section that provides specific tips on how to improve this particular page's load performance. Click an opportunity to learn more about it.



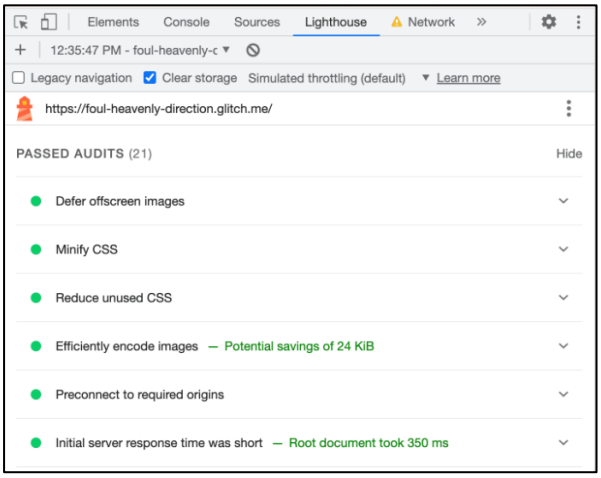
Diagnostics:

The Diagnostics section provides more information about factors that contribute to the page's load time.

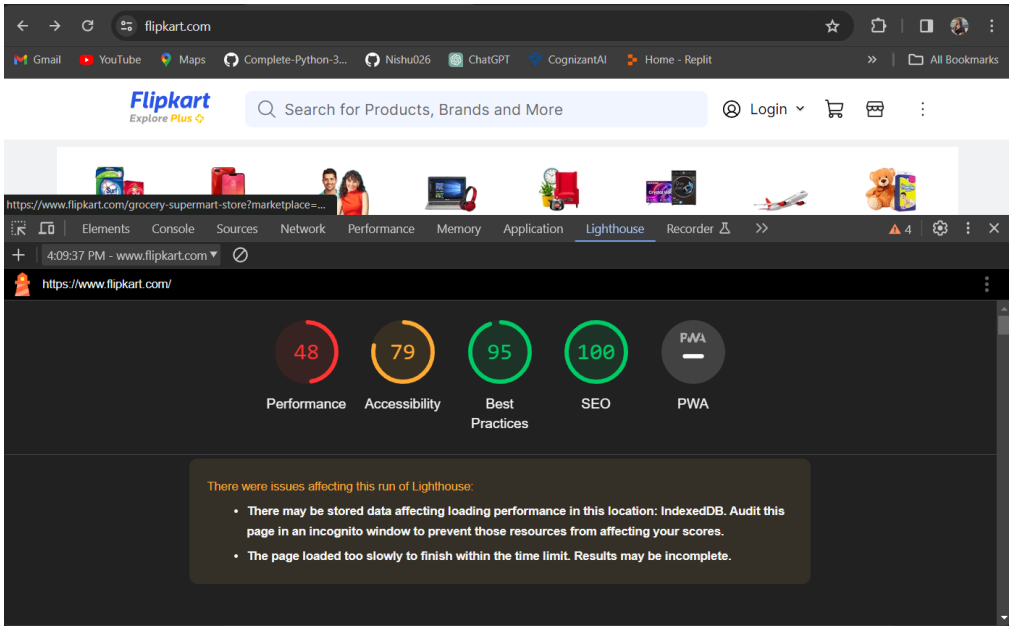


Passed audits:

The Passed audits section shows you what the site is doing correctly. Click to expand the section.



Performance insights of Flipkart:

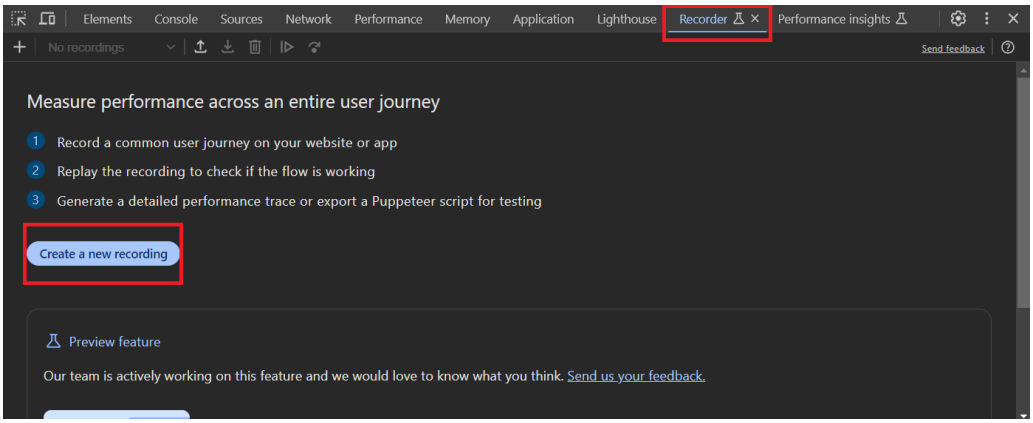


Click on + icon to create new audit.

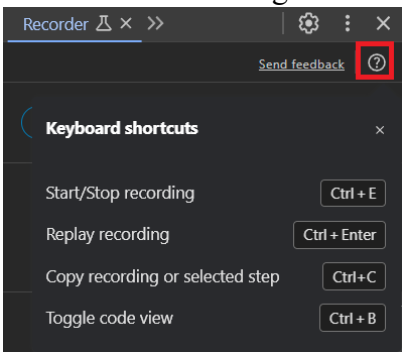
Go to this site to explore more about lighthouse:
https://developer.chrome.com/docs/devtools/lighthouse

9. RECORDER (PREVIEW PANEL):

Recorder API is a preview feature that allows you to extend the Recorder panel in Chrome DevTools.



Use shortcuts to navigate the Recorder faster.



Example: Recording a execution flow of a website.

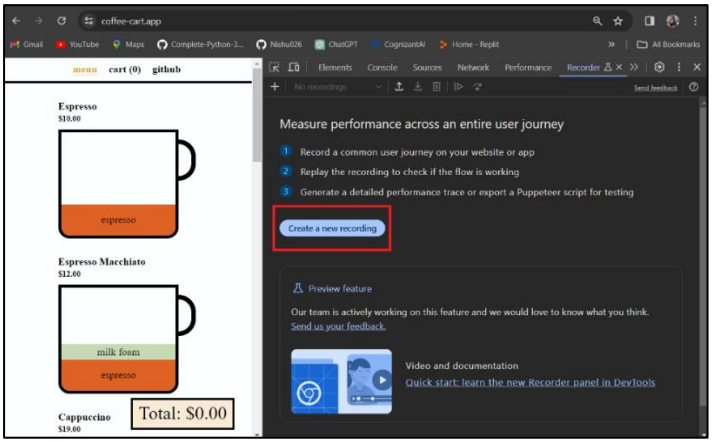
We will be using the coffee ordering demo page.

1. Record a user flow:

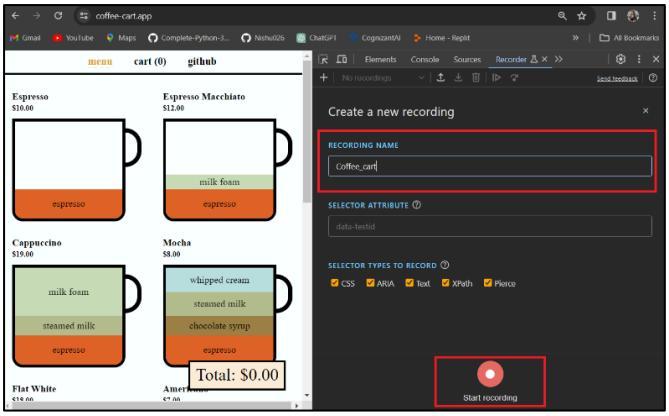
Steps:

i. Go to https://coffee-cart.app/

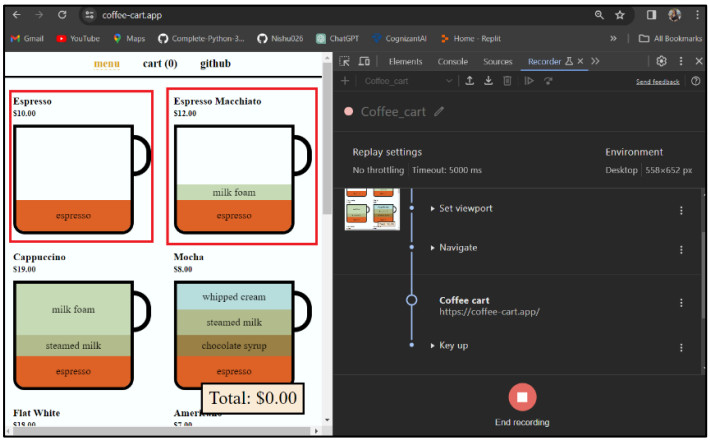
ii. Go to recorder panel. Click on the “Create a new recording” button.



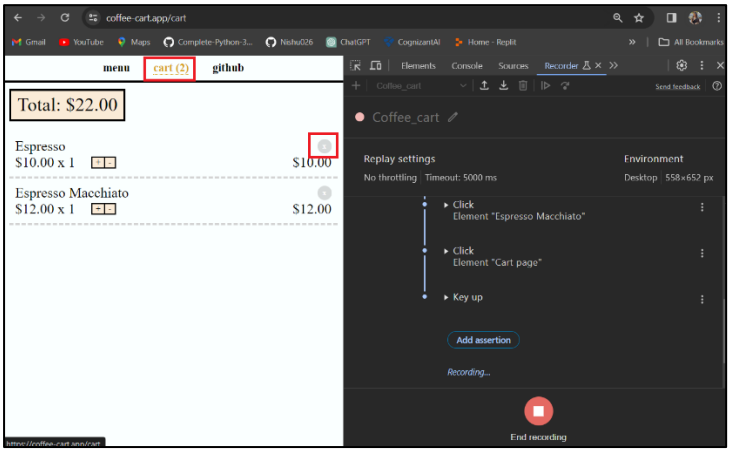
iii. Type the name of the recording(“Coffee_cart”). The recording is started. The panel is showing Recording... indicating the recording is in progress.



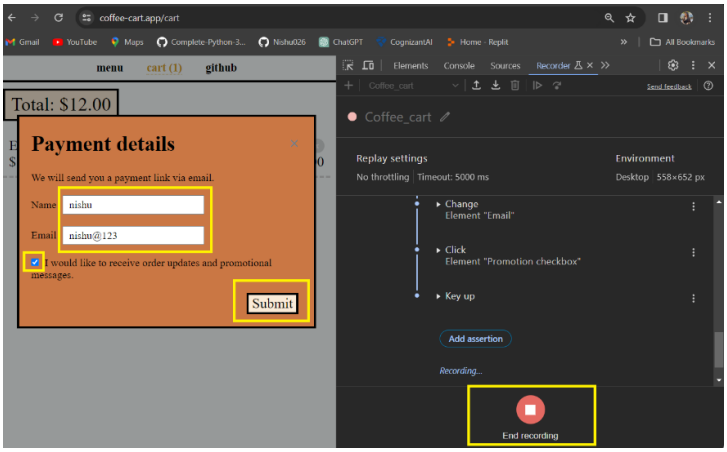
iii. Add a coffee to the cart (Espresso). Add another coffee to the cart (Espresso Macchiato).



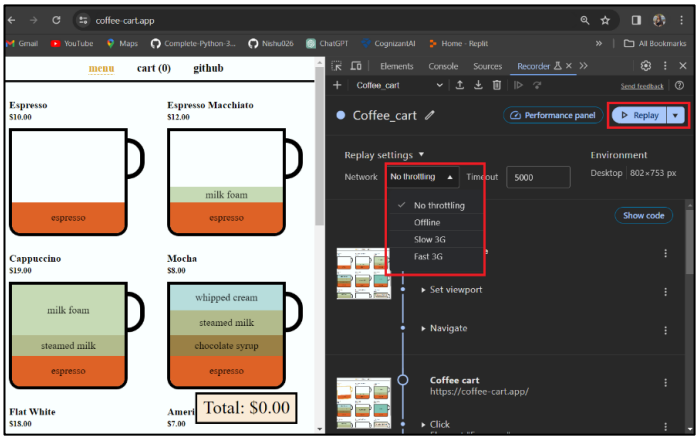
iv. Go to the cart page. Remove one coffee from the cart (Espresso).



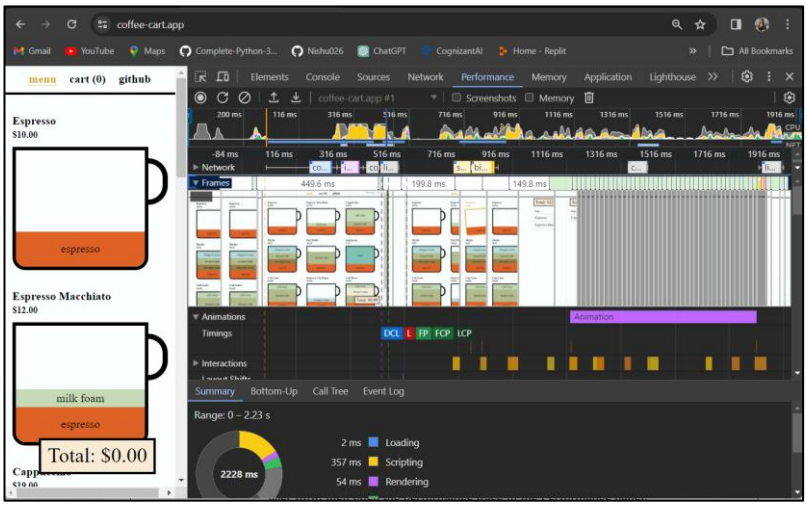
v. Start the checkout process and fill in payment details.
vi. Click on Submit.
vii. Click on End recording to end the recording.



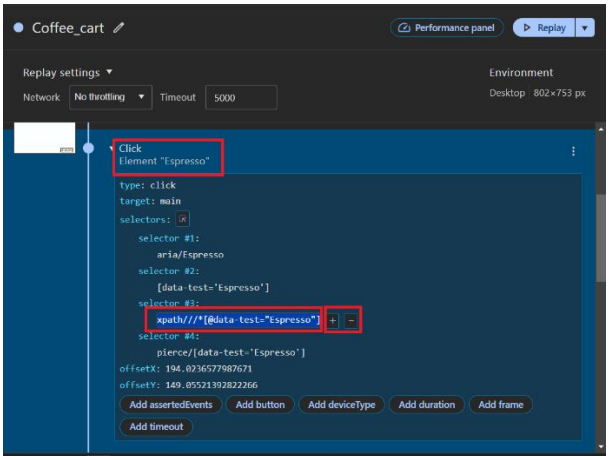
2. **Replay a user flow:**
   After recording a user flow, you can replay it by clicking on the Replay button.You can simulate a slow network connection by configuring the Replay settings.



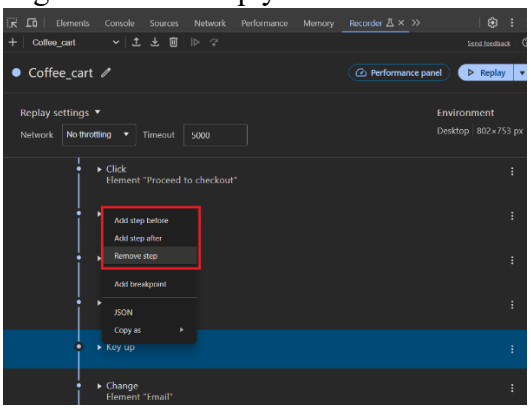
3. **Measure a user flow:**
   You can measure the performance of a user flow by clicking on the Measure performance button. For example, checkout is a critical user flow of a shopping website. With the Recorder panel, you can record the checkout flow once and measure it regularly. Clicking on the Measure performance button will first trigger a replay of the user flow, then open the performance trace in the Performance panel.



4. **Edit steps:**
   Expand each step to see the details of the action. For example, expand the Click Element "Espresso" step. You can add or remove any selectors The selector is editable. In the Selector types to record set of checkboxes, choose the types of selectors to detect automatically:
   - **CSS:** Syntactic selectors.
   - **ARIA:** Semantic selectors.
   - **Text:** Selectors with the shortest unique text if available.
   - **XPath:** Selectors that use XML Path Language.
   - **Pierce:** Selectors similar to the CSS ones but that can pierce shadow DOM.

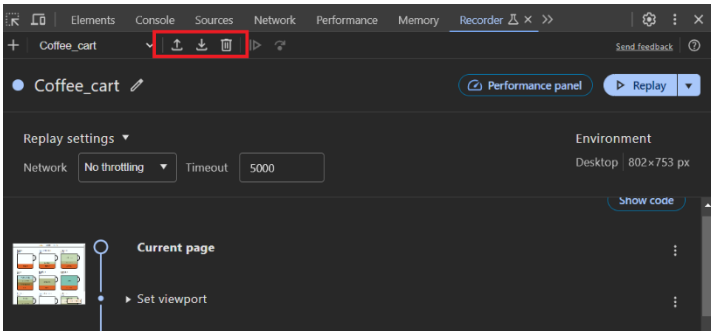


5. **Add and remove steps:**
   There are options to add and remove steps too. This is useful if you want to add an extra step or remove an accidentally added step. Instead of re-recording the user flow, you can just edit it:
   Right-click the step you want to edit or click the Three-dot menu next to it.



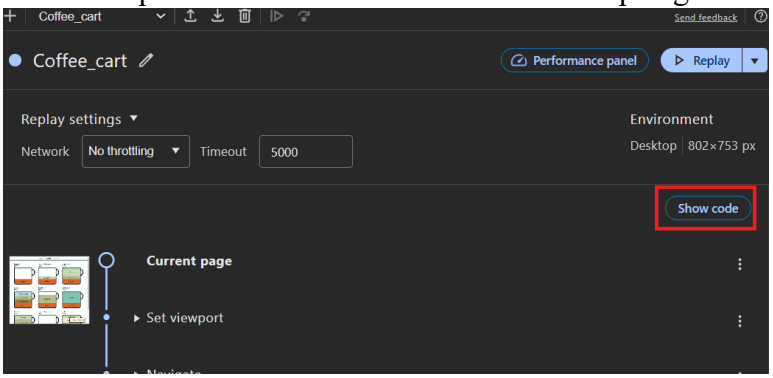
6. **Export a recording:** To further customize the script or share it for bug reporting purposes, you can export the user flow in one of the following formats:
   - JSON file.
   - @puppeteer/replay script.
   - [Puppeteer](/docs/puppeteer/ script.
   - Puppeteer (including Lighthouse analysis).
7. **Import a recording:** Only in JSON format.

8. **Delete a recording:** Delete the selected recording.

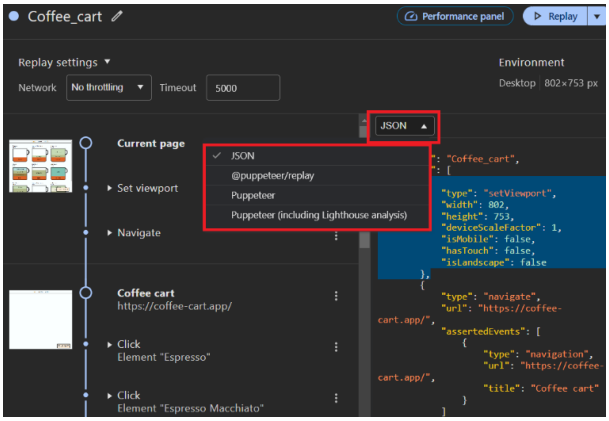


9. **Inspect code:**

To inspect the code of a user flow in various formats. Open a recording in the Recorder panel. Click Show code in the top right corner of the steps list.



The Recorder shows a side-by-side view of the steps and their code.As you hover over a step, the Recorder highlights its respective code in any format, including those provided by extensions.Expand the format drop-down list to select a format that you use to export user flows.



10. **Extensions:**

Installing extensions:

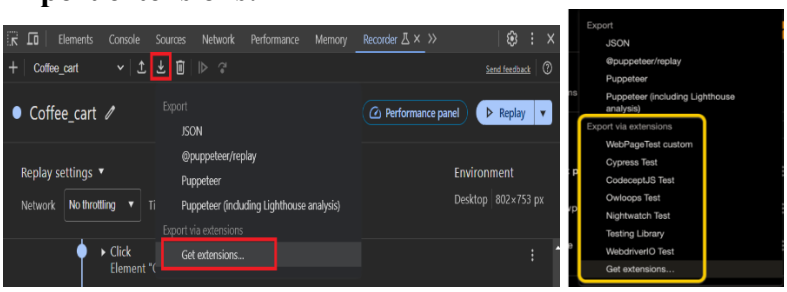
To integrate the Recorder with your tools, install extensions:

i. Choose an extension from the list of known ones and click its link.

ii. In Chrome Web Store, click Add extension.

iii. Open DevTools in a new tab and find new custom options in the Recorder.

Export extensions:



- **CodeceptJSextension:**
  This extension lets you export JSON user flows as CodeceptJS test script. CodeceptJS is a supercharged End 2 End Testing.
  Link:https://chrome.google.com/webstore/detail/codeceptjs-chrome-recorde/jgdcobhagdbipacidhfnoaccgjooebam
- **Cypress extension:**
  This extension lets you export JSON user flows as Cypress test script. Cypress is a front-end testing tool built for the modern web.
  Link:https://chromewebstore.google.com/detail/cypress-chrome-recorder/fellcphjglholofndfmmjmheedhomgin
- **Nightwatch extension**
  This extension lets you export JSON user flows as Nightwatch test script. Nightwatch is an end-to-end testing solution for web applications and websites.
  Link:https://chrome.google.com/webstore/detail/nightwatch-chrome-recorde/nhbccjfogdgkahamfohokdhcnemjafjk/
- **Owloops extension:**
  This extension lets you export recordings as Owloops tests.
  Link:https://chromewebstore.google.com/detail/owloops-chrome-recorder/kojnjjbhkfcpjhppocjocdkjkbbkhimh
- **Testing Library extension**
  This extension lets you export JSON user flows as Testing Library script. Testing Library has simple and complete testing utilities that encourage good testing practices.
  Link:https://chromewebstore.google.com/detail/testing-library-recorder/pnobfbfcnoeealajjgnpeodbkkhgiici
- **WebdriverIO extension:**
  This extension lets you export JSON user flows as WebdriverIO test script. WebdriverIO is an end-to-end testing solution for web, mobile and IoT applications and websites.
  Link:https://chromewebstore.google.com/detail/testing-library-recorder/pnobfbfcnoeealajjgnpeodbkkhgiici
- **WebPageTest extension**
  This extension lets you export user flows from the Recorder directly as WebPageTest Custom scripts to measure site's performance. See Converting user flows to WebPageTest custom scripts to learn more.
  Link:https://chromewebstore.google.com/detail/eklpnjohdjknellndlnepihjnhpaimok

10. PERFORMANCE INSIGHTS (PREVIEW PANEL):

The new Performance insights panel is an experiment to address these 3 developer pain points when working with the existing Performance panel:

Too much information. With the redesigned UI, the Performance insights panel streamlines the data and displays only relevant information.

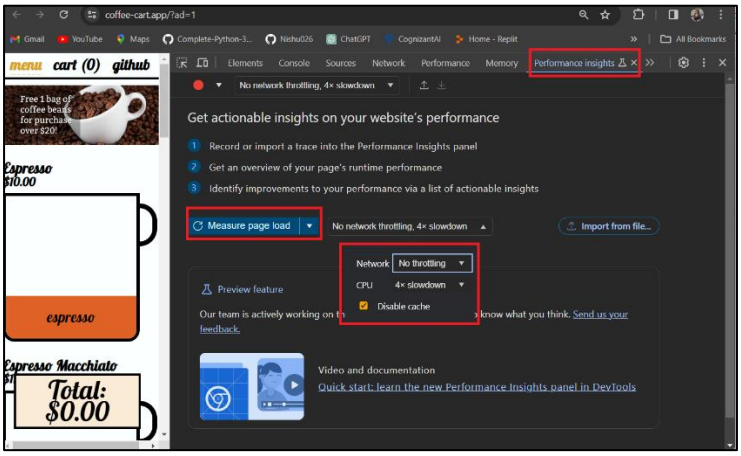
Hard to distinguish between use cases. The Performance insights panel supports use-case-driven analysis. It only supports page load use-case at the moment, with more to come in the future based on your feedback, for example, interactivity.

Requires deep expertise of how browsers work to use effectively. The Performance insights panel highlights the key insights in the Insights pane, with actionable feedback on how to fix issues.

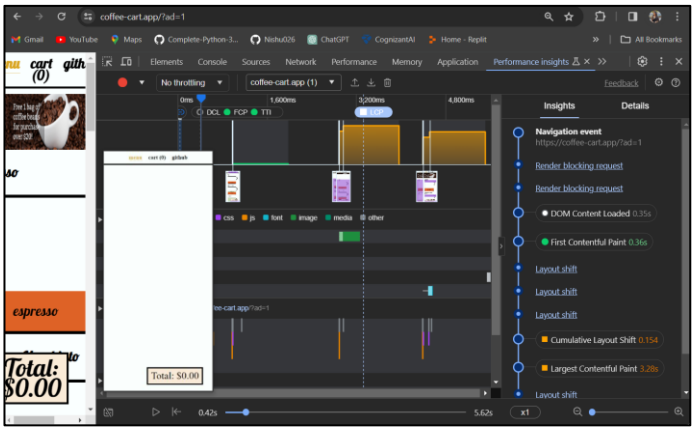
Example: Finding performance insights of a website

i. Go to https://coffee-cart.app/?ad=1 open the Performance insights panel.

ii. You can throttle the network and CPU while recording. For this, check Disable cache and set CPU to 4x slowdown in the drop-down menu and click on "Measure page load". DevTools records performance metrics while the page reloads and then automatically stops the recording a couple seconds after the page load has finished.



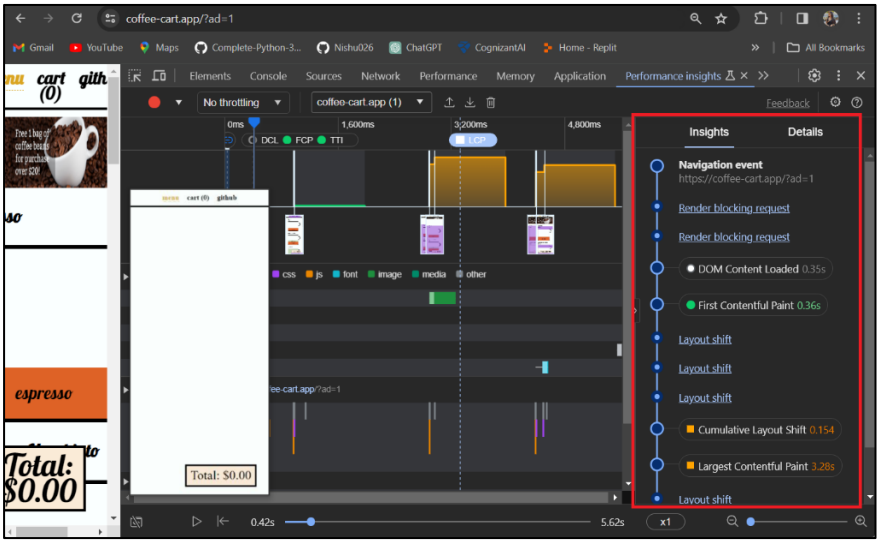
1. Output of performance insights:



Use the controls at the bottom to control the replay of your recording.

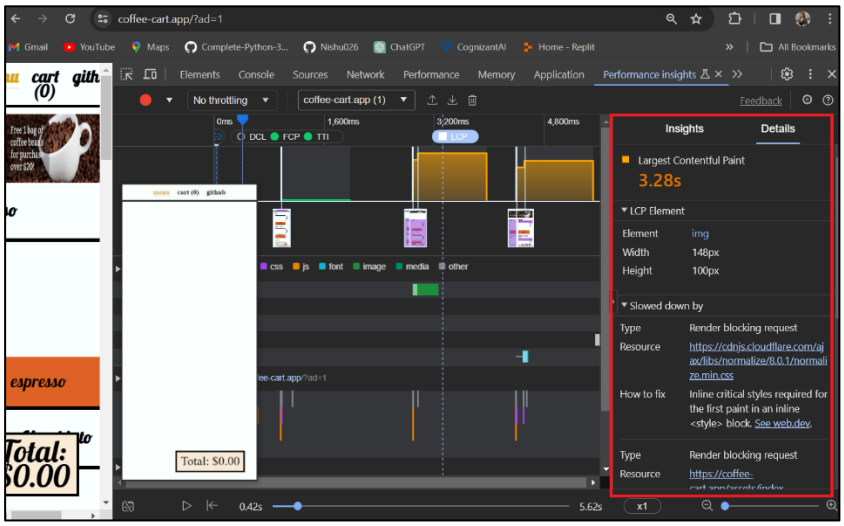
2. **View performance insights:**

Get a list of performance insights in the Insights pane. Identify and fix potential performance issues.

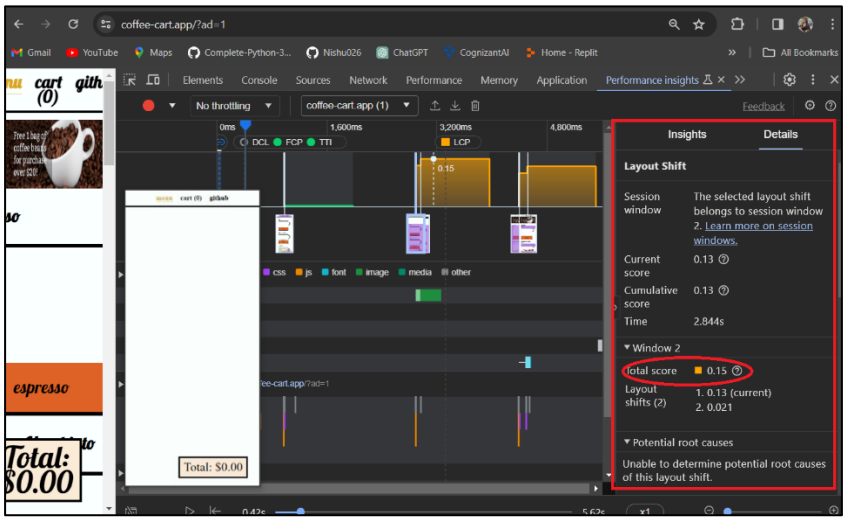


- **Largest Contentful Paint (LCP)** is one of the Core Web Vitals metrics. It reports the render time of the largest image or text block visible within the viewport, relative to when the page first started loading. A good LCP score is 2.5 seconds or less.

Click on the Largest Contentful Paint(LCP) to get the further details.



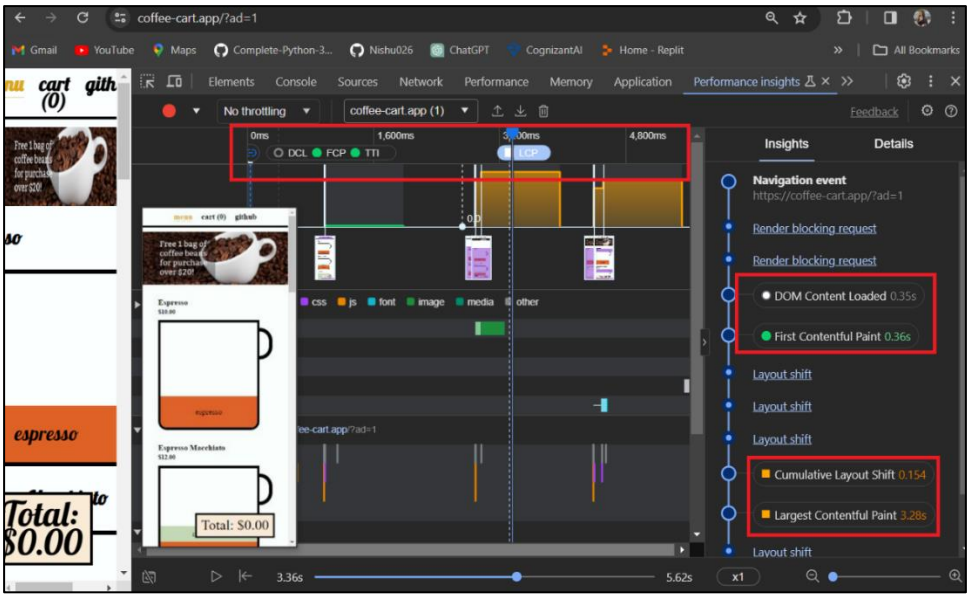
- **Cumulative Layout Shifts (CLS)** is one of the core web vitals metrics. Use the Layout Shifts track to identify potential issues and causes of layout shifts.



3. View Web Vitals performance metrics

Web Vitals is an initiative by Google to provide unified guidance for quality signals that are essential to delivering a great user experience on the web.

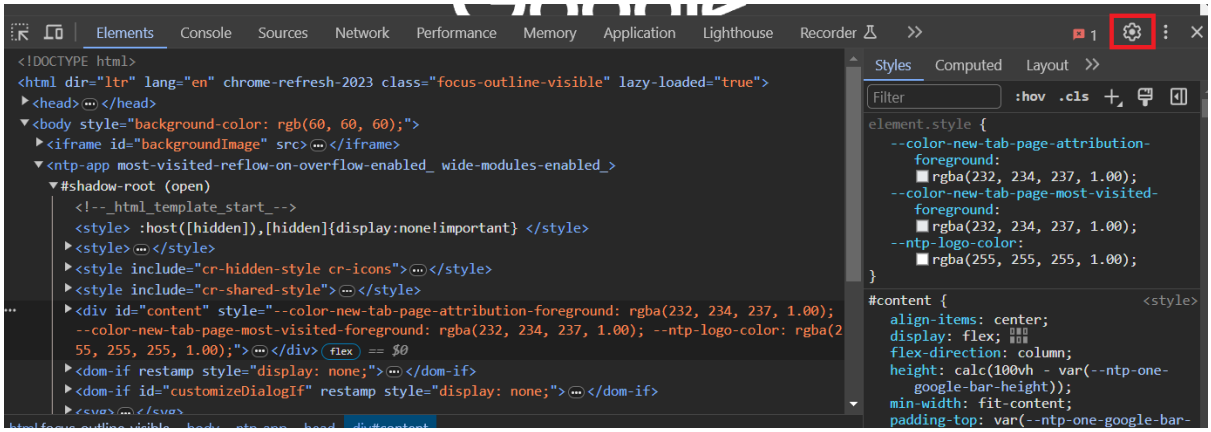
You can view these metrics on the Timeline and Insights pane.

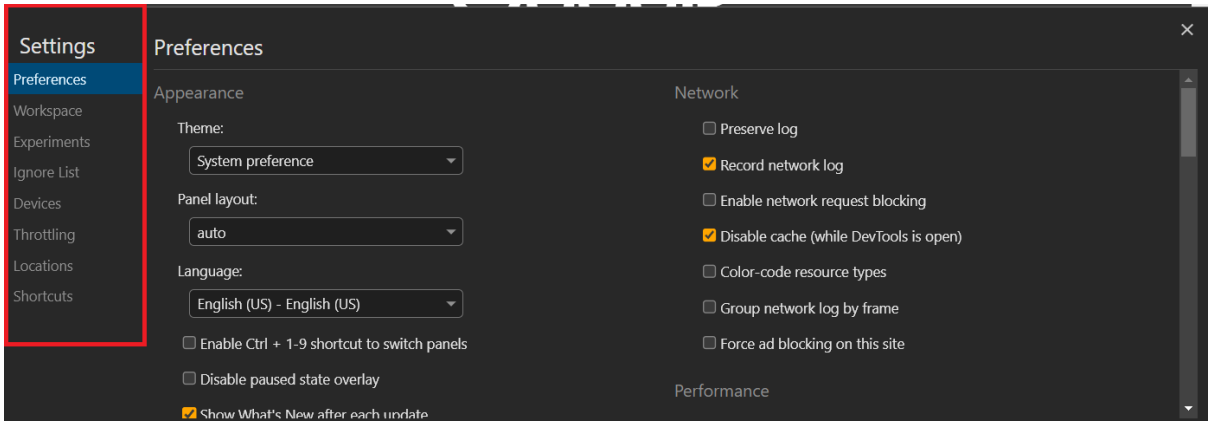


DEVTOOLS SETTINGS

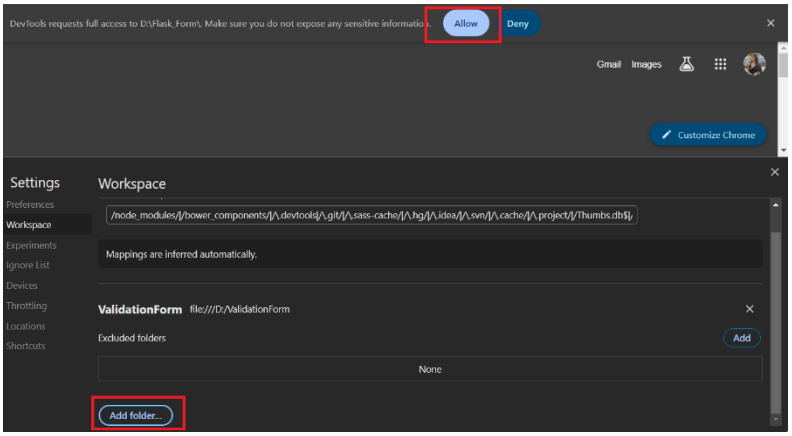
DevTools settings let you control the behaviour of both individual panels and DevTools in general.

The devTools Settings panel has several tabs described in the following sections:

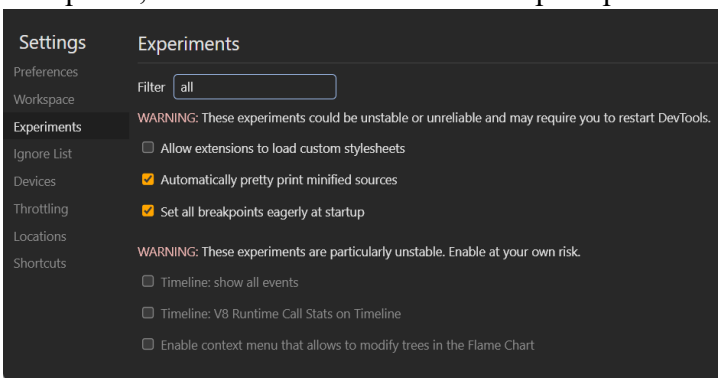




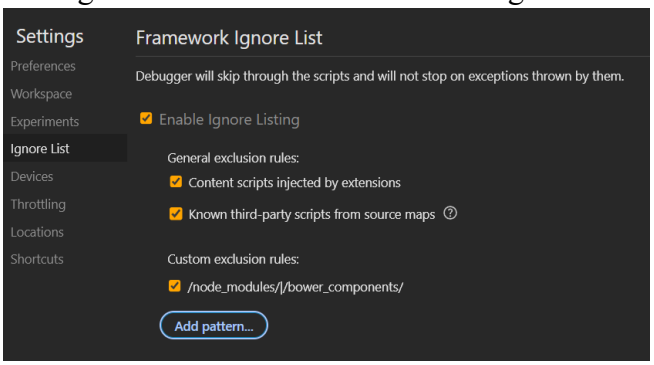
1. **Preferences:** Configure the appearance and behavior of DevTools and its panels using Settings > Preferences. This tab lists both general customization options and panel-specific ones.
2. **Workspace:** Settings > Workspace lets you save changes that you make within DevTools to source code that's stored on your computer.
   To add a new Workspace:
   i. In the Workspace tab, click Add folder.
   ii. Select the folder with your sources.
   ii. Click Allow in the prompt at the top to let DevTools make changes to sources.



3. **Experiments:** Settings > Experiments let you enable and disable experimental features of Chrome DevTools.
   i. In the Experiments tab, search for the experiment you would like to try in the Filter textbox.
   ii. Enable the checkbox next to the experiment.
   Close Settings.
   If required, click Reload DevTools in the prompt at the top.

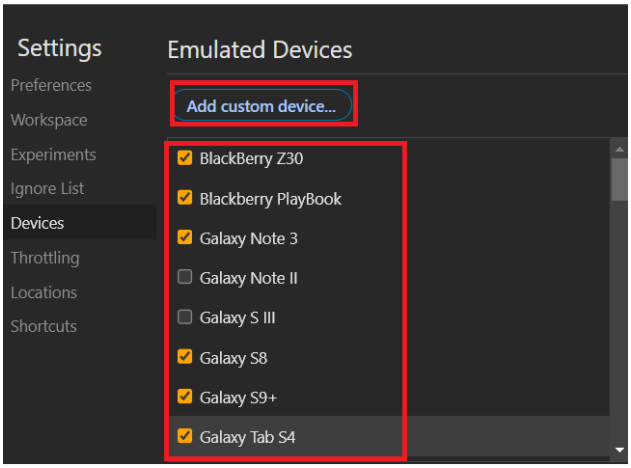


4. **Ignore List:** Settings > Ignore List lets you configure the list of scripts the debugger ignores.
   <u>To enable or disable all ignore listing for the debugger:</u>
   i. Open Settings.
   ii. In the Ignore List tab, check or clear Settings. Settings > Checkbox. Enable Ignore Listing. This is the main switch for all ignore-listing capabilities.



5. **Devices:** Settings > Devices list devices and their dimensions. You can select these devices from the Dimensions drop-down list in device mode.
   <u>To add a device to the list:</u>
   i. Open Settings.

ii. In the Device tab, enable the checkbox next to a device you want to add. You can add custom devices.



6. **Throttling:** Settings > Throttling lists custom throttling profiles. You can use these profiles to test custom connection speeds in the Network panel.

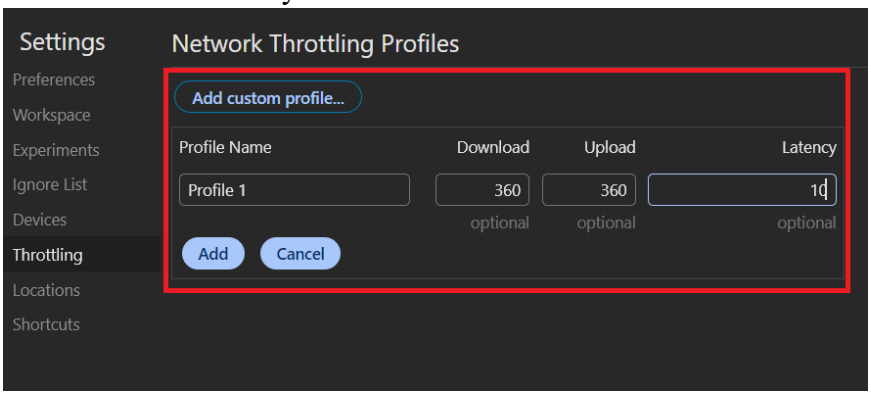
   To add a custom profile:

   i. Open Settings.

   ii. In the Throttling tab, click Add custom profile.

   iii. Specify the following values for the new entry:

   - Profile Name.
   - Download and Upload speeds in Kbps.
   - Latency in milliseconds.



7. **Locations:** Settings > Locations list geolocation presets. You can use these presets to override geolocation in Chrome. You can also populate the list with your own preset that you use frequently.

   To add a custom preset:

   i. In the Locations tab, click Add location.

   ii. Specify the following values for the new entry. For example, let's add New York as a new location.

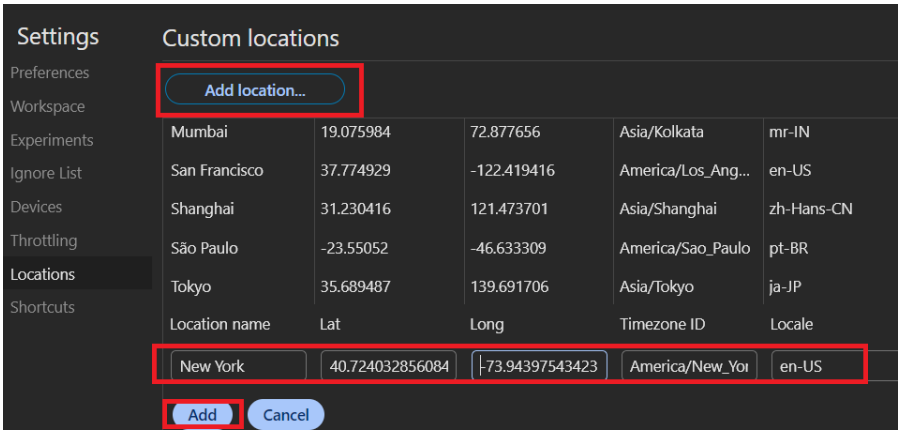
   Location name: New York.

   Latitude: 40.72403285608484.

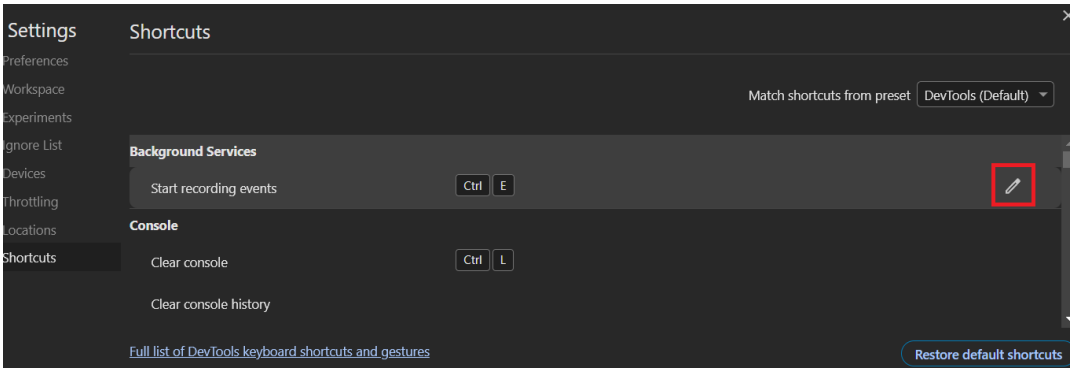
   Longitude: -73.94397543423175.

   Timezone ID: America/New_York as defined in the latest release of the Time Zone Database.

   Locale: en-US as defined by BCP47.



8. **Shortcuts:** Settings > Shortcuts lists default shortcuts you can use while focused in DevTools to speed up your workflow.
   when focused in DevTools, press:
   ?
   F1 on Windows or Linux
   Fn + F1 on Mac
   The Settings panel opens.
   Customizable shortcuts.



CONCLUSION

DevTools have proven to be an indispensable asset in modern web development, providing a suite of powerful functionalities that streamline the debugging, testing, and optimization processes. Through its array of features like inspecting and editing HTML, CSS, and JavaScript in real-time, monitoring network activity, profiling performance, and offering a multitude of extensions, DevTools empowers developers to create more efficient, responsive, and visually appealing web applications.

Moreover, DevTools' diagnostic features help identify bottlenecks and inefficiencies, guiding developers toward solutions that enhance speed, responsiveness, and overall user satisfaction. The integration of accessibility auditing tools ensures that applications are inclusive and usable for all individuals, complying with standards and best practices.

DevTools, with their comprehensive set of features and real-time insights, prove invaluable in real-life scenarios across the spectrum of web development. From debugging complex issues to optimizing performance and ensuring seamless user experiences, their practical utility is evident in various contexts.

In troubleshooting, DevTools serve as a detective's magnifying glass, allowing developers to delve deep into code, inspect elements, and identify issues efficiently. This capability becomes crucial when diagnosing layout discrepancies, JavaScript errors, or network-related problems, enabling swift resolution and minimizing downtime.

In the realm of design and optimization, DevTools act as a designer's canvas. They facilitate live editing of styles, offering immediate visual feedback, which is incredibly beneficial when fine-tuning layouts or testing responsiveness across multiple devices and screen sizes. Performance profiling tools aid in optimizing speed, ensuring that websites and applications load swiftly, providing a seamless user experience.

Accessibility auditing through DevTools ensures inclusivity, allowing developers to identify and rectify potential barriers for users with disabilities, ensuring compliance with accessibility standards.

Additionally, in collaborative settings, DevTools serve as an educational resource, enabling developers to share insights and work collaboratively on code. They allow for real-time inspection and editing, making it easier for teams to troubleshoot issues collectively and learn from each other's approaches.

Ultimately, the real-life utility of DevTools lies in their ability to streamline workflows, enhance collaboration, and empower developers to create robust, visually appealing, and performant web applications. They stand as an indispensable ally in the everyday challenges faced in web development, fostering efficiency, innovation, and continuous improvement.