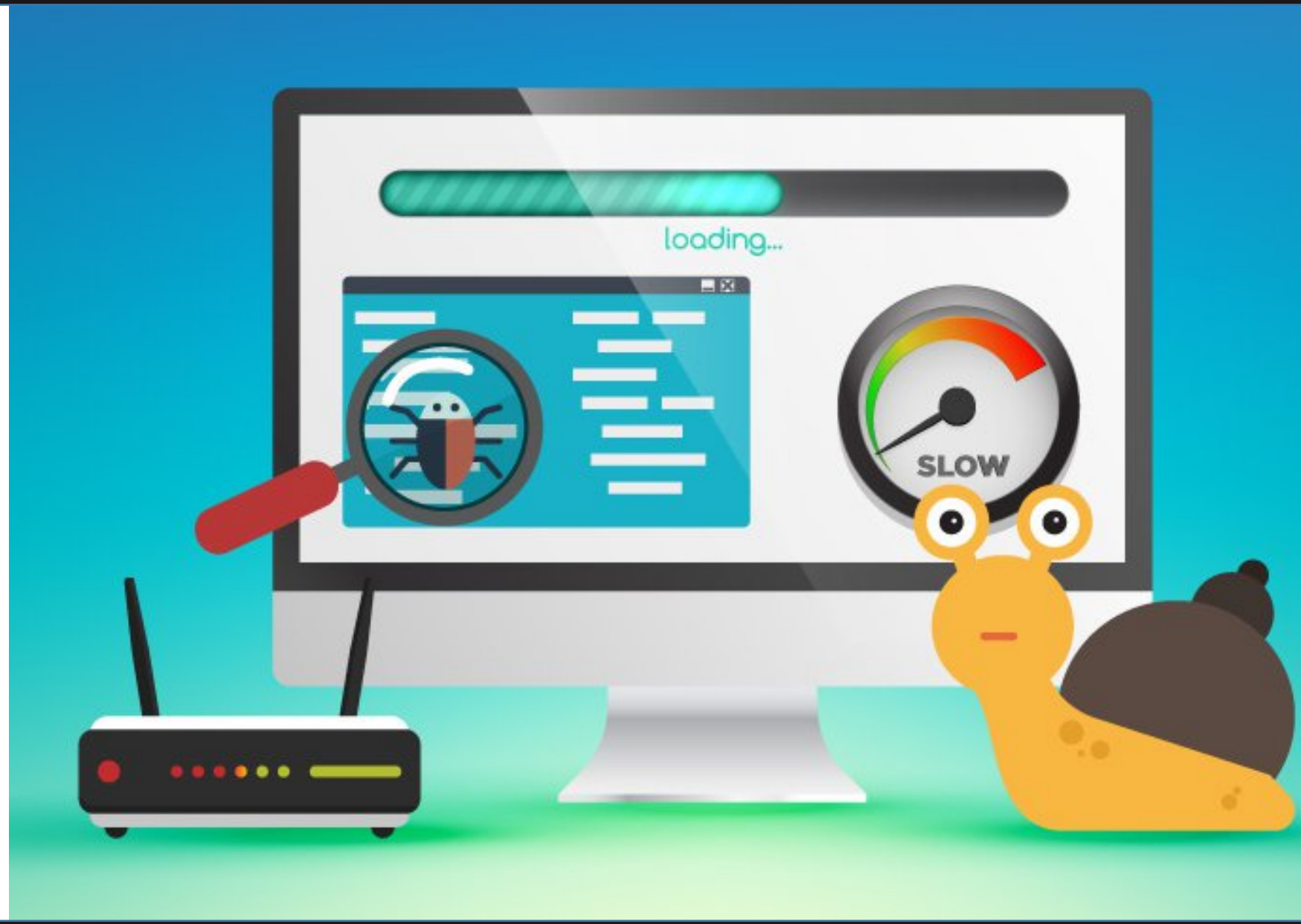
TESTES E PERFORMANCE WEB

TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A WEB

POLITÉCNICO DO PORTO
ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

M03 – PERFORMANCE TOOLS

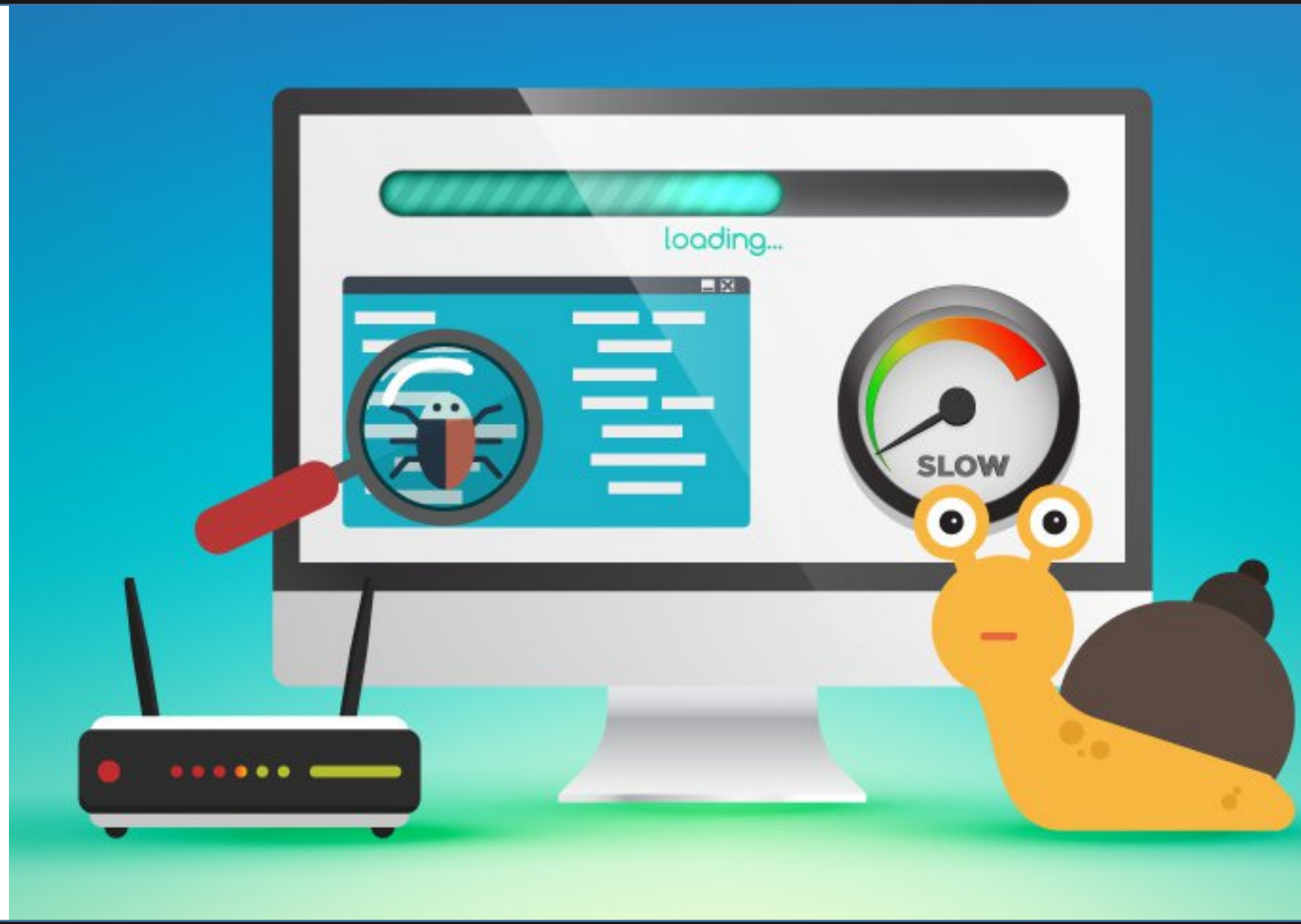
TSIW 2023/2024



AGENDA

1. Test Plan;
2. Performance Tools Concepts;
3. Main Tools:
    - 3.1. Google Lighthouse (Lab Data);
    - 3.2. PageSpeed Insights (Lab + Field Data);
    - 3.3. Chrome DevTools:
        - 3.3.1. Network Panel;
        - 3.3.2. Performance Monitor;
        - 3.3.3. Performance Panel.
    - 3.4. Performance Timing APIs.
4. Final remarks.

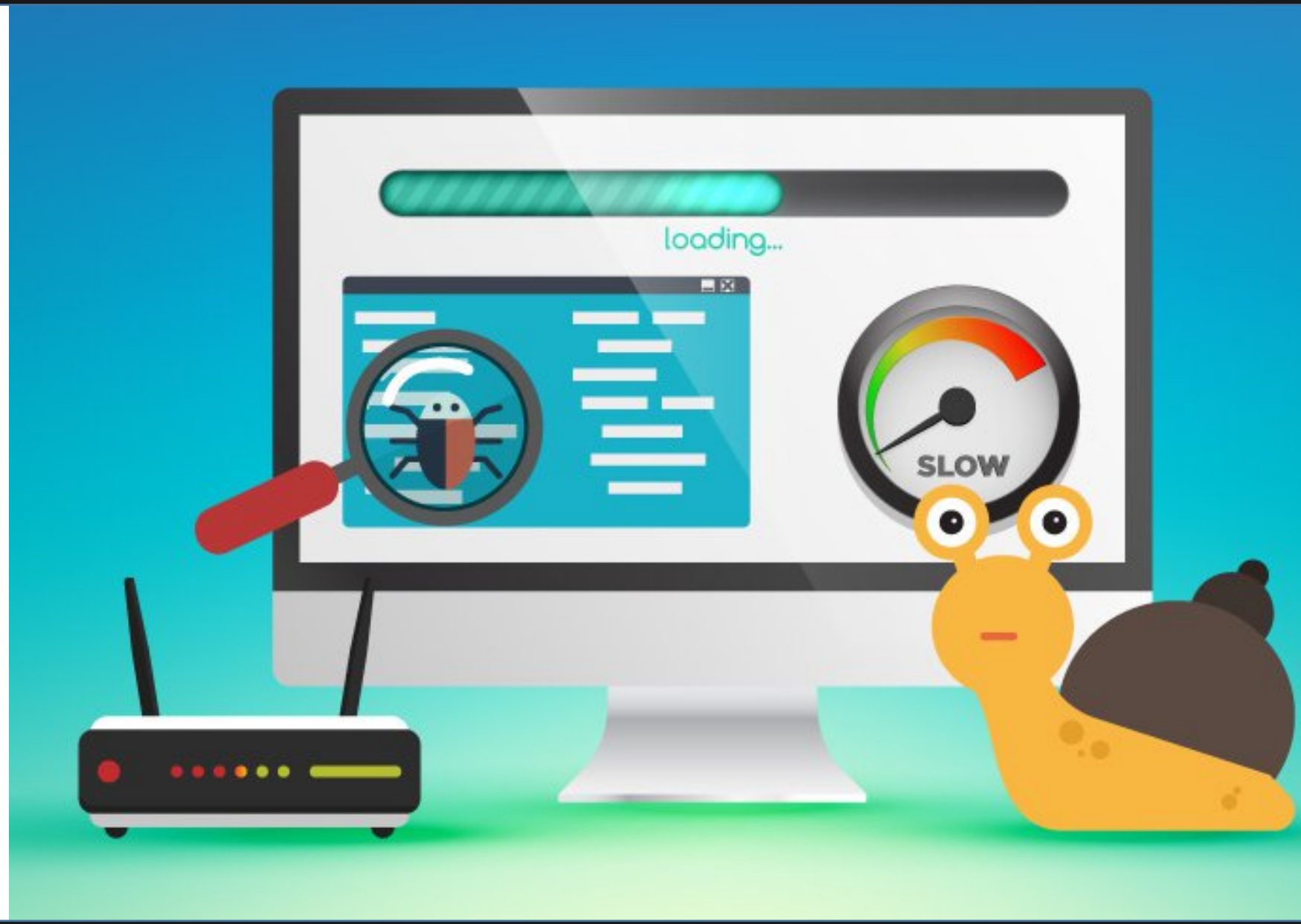
TEST PLAN



1. TEST PLAN

- If your website has a performance issue you need to recover it fast;
- A Test Plan is needed:
  - Who's responsible for running tests;
  - What tools and settings will be used for each test;
  - How results will be recorded and which feedback will be given to the development process.
- Use performance analysis tools manually at first to understand the reports and determine optimization priorities;
- As your process evolves, you should automate these tests so developers:
  - Are warned about potential problems;
  - Are blocked from committing poorly optimized code.

PERFORMANCE TOOLS CONCEPTS



2. PERFORMANCE TOOLS CONCEPTS

- Knowing that a website have a performance problem is important but finding and fixing the causes is more important;
- The first step is to identify whether the fault occurs server-side or client-side;
- A slow network response, either on the initial page load or during an HTTP request, will normally indicate a server-side issue;
- Client-side issues can be diagnosed using browser developer tools;
- Performance is affected when the browser has considerable work to do:
  - Long-running JavaScript functions;
  - DOM updates that causes the page to reload;
  - CSS changes that affects many elements.

2. PERFORMANCE TOOLS CONCEPTS

- Most of the tools diagnose a particular “page” in your website within the context of a web browser;
- They primarily analyse front-end performance, although a back-end server or database could be cause of a large or slow response.

2. PERFORMANCE TOOLS CONCEPTS

Browser Rendering Process:

- When a website is first accessed by a user, the following steps occur:
  1. The browser makes an HTTP request for a specific URL;
  2. The server receives and parses the request;
  3. The server returns a response (HTML);
  4. The browser starts to receive HTML data, which it parses and trigger additional HTTP requests for Images, CSS, and other files;
  5. Behind the scenes, starts building the DOM;
  6. Style calculations also determine which CSS rules applies to each DOM node, and a CSSDOM is created for JavaScript interaction;

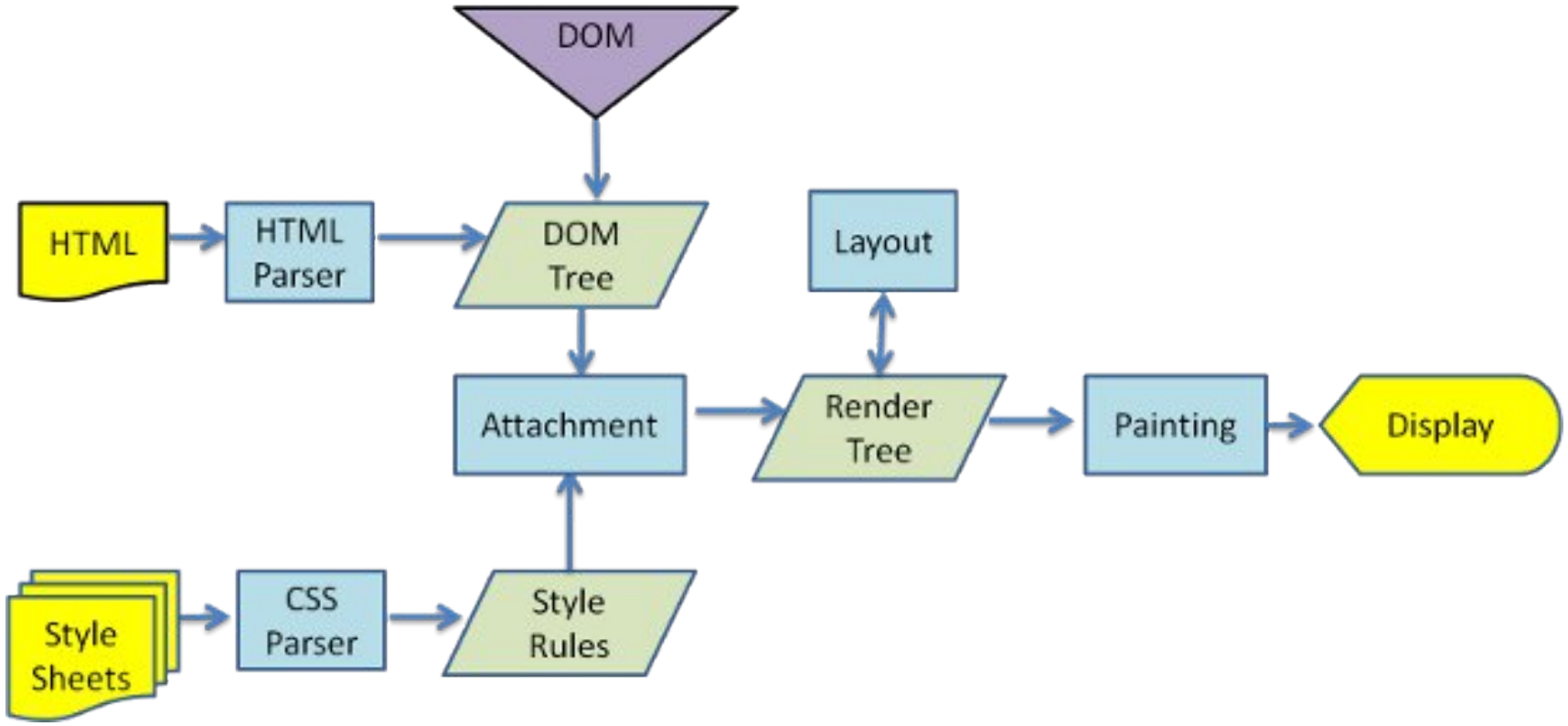
2. PERFORMANCE TOOLS CONCEPTS

Browser Rendering Process:

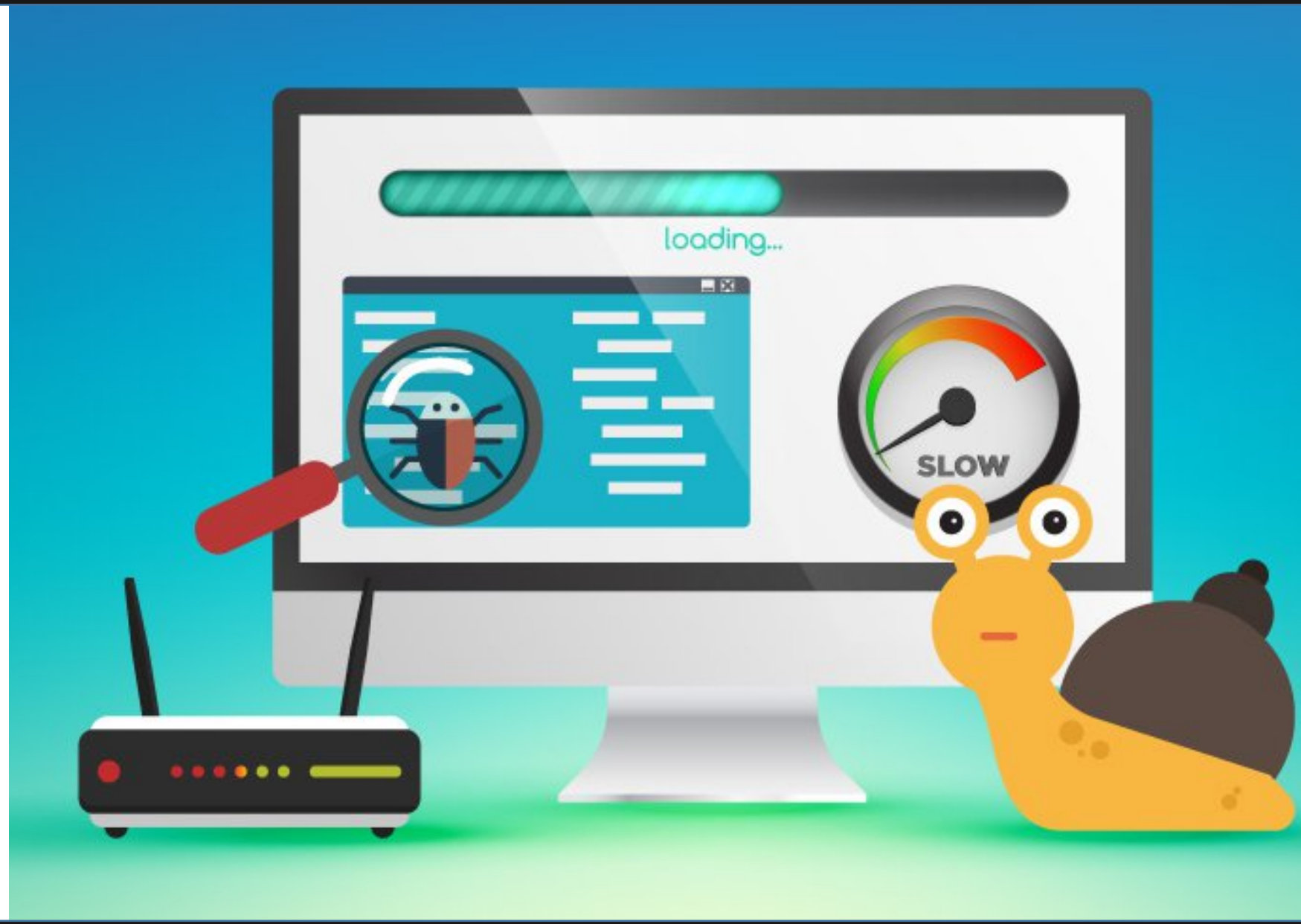
7. The layout (or re-flow) phase starts. This calculates the dimensions of each element and how it affects the size or positioning of elements around it;
8. The paint phase starts. This draws the visual parts of each elements onto separate layers – that is, text, colours, images, borders, shadows and so on;
9. The composite phase starts. This draws each layer to the screen in the correct order;
10. The page is now in an initial viewable state. During or after the render, JavaScript can run to make further HTTP requests, perform calculations, update the DOM, or apply CSS rules. This could trigger further layout, paint, and/or composite phases.

2. PERFORMANCE TOOLS CONCEPTS

Browser Rendering Process:

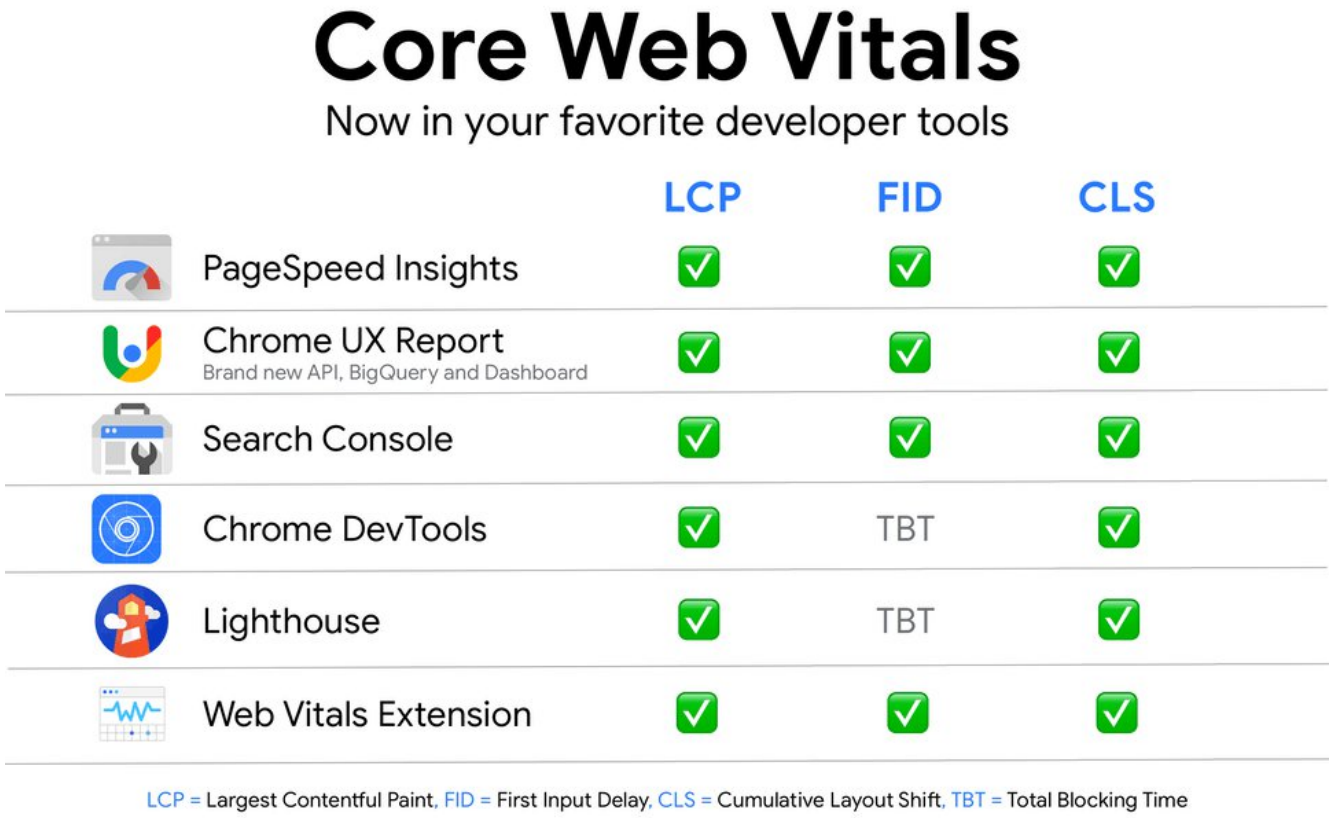


MAIN TOOLS



3. MAIN TOOLS

- Several tools to find performance issues;
- Web vitals to unify strategies.



3. MAIN TOOLS

- What should I use?
  - **Search Console** – to identify pages that require attention (field data);
  - **PageSpeed Insights** (Lighthouse and Chrome UX Report) – to diagnose lab and field issues on the pages;
  - **Lighthouse and Chrome DevTools** – to measure Core Web Vitals and get actionable guidance on exact what to fix;
  - **Web Vitals Chrome Extension** - for a real-time metrics (view on desktop);
  - **Chrome UX Report API** – to create a custom dashboard of Core Web Vitals (field data);
  - **PageSpeed Insights API** – to create a custom dashboard of Core Web Vitals (lab data);
  - **Lighthouse CI** – use on pull requests to ensure that there are no regressions in Core Web Vitals before you deploy a change to production.

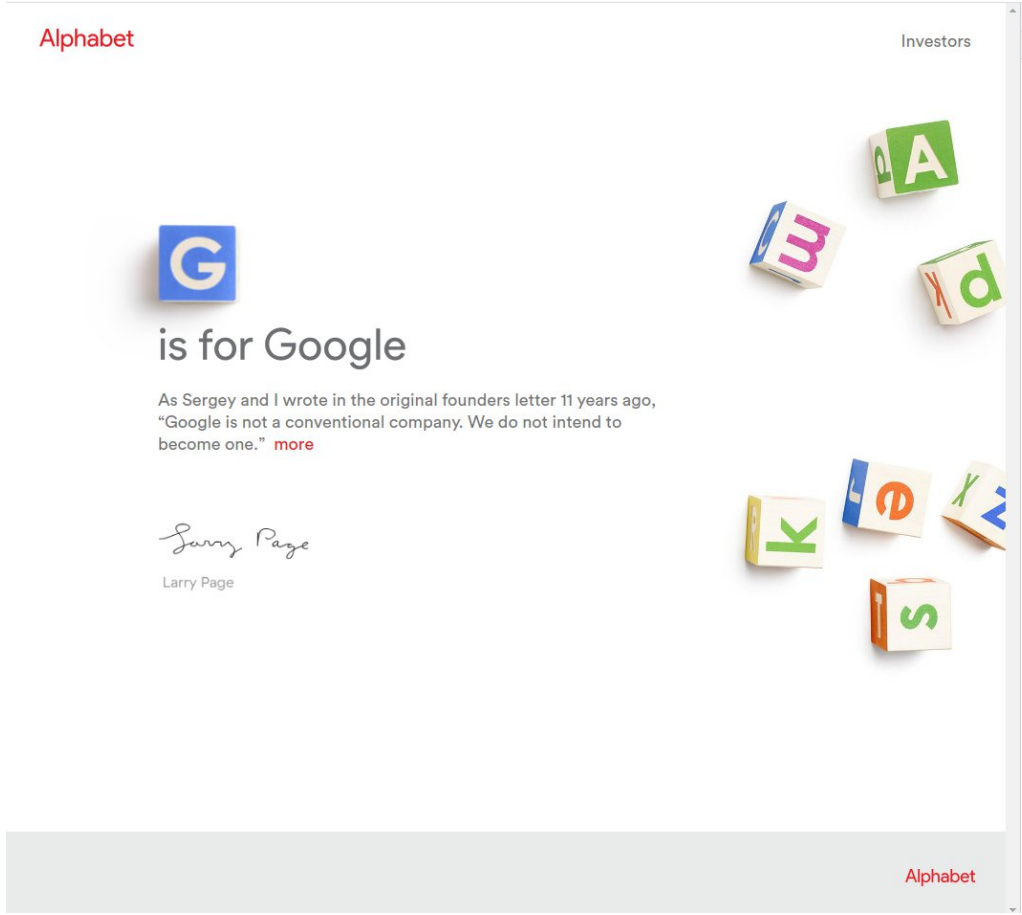
3. MAIN TOOLS

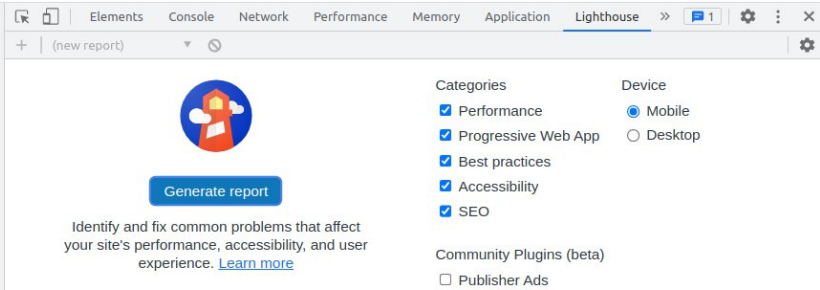
Google Lighthouse

- An open-source tool that helps evaluate the performance and quality of your page or app;
- It measures several dimensions:
  - Performance;
  - User Experience;
  - Accessibility.
- You can access it in the following ways:
  - From within Chrome DevTools;
  - From online web.dev or PageSpeed Insights;
  - From NodeJS module, through which command line and automated tests can be executed.

3. MAIN TOOLS

Google Lighthouse

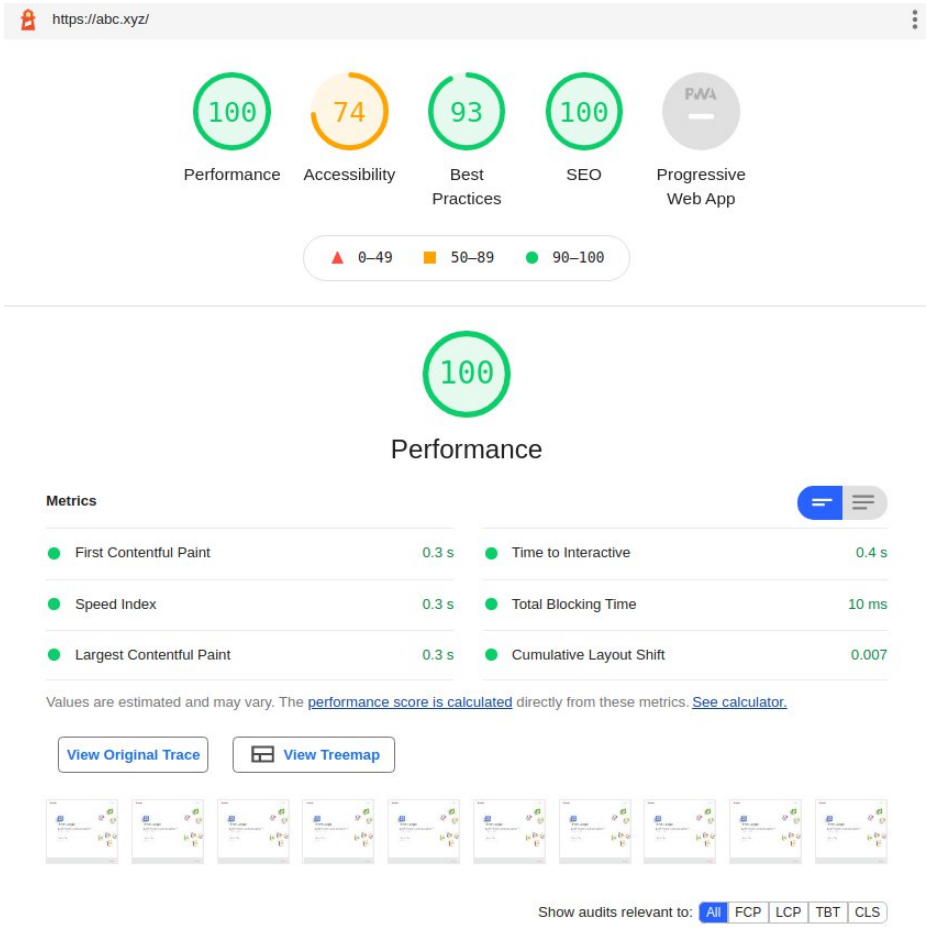


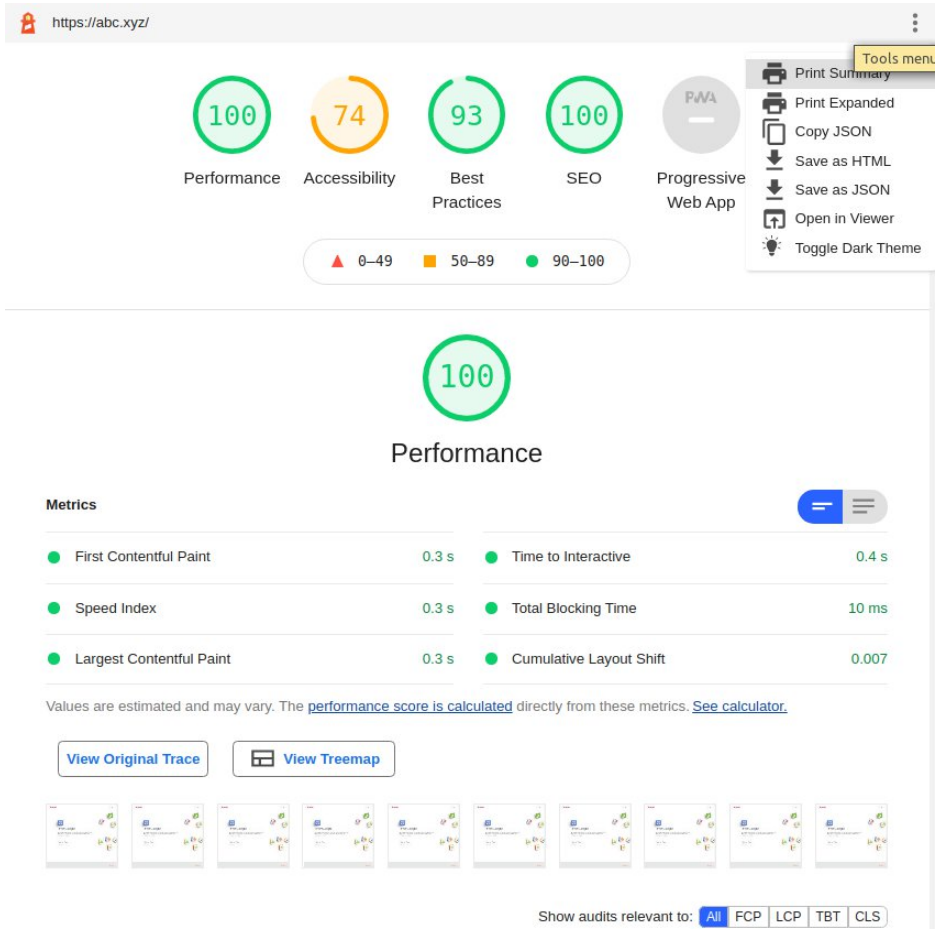


3. MAIN TOOLS

Google Lighthouse

- Report





3. MAIN TOOLS

Google Lighthouse

- In Lighthouse report:
  - The **opportunities** section has detailed suggestions and documentation on how to implement them;
  - The **diagnosis** section list additional guidance that developers can explore to further improve their performance;
  - The **approved audits** sections refer to the audits that passed in the Lighthouse analysis engine.

3. MAIN TOOLS

Google Lighthouse

Show audits relevant to: All | FCP | LCP | TBT | CLS

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Reduce unused JavaScript	0.18 s

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Ensure text remains visible during webfont load
- Serve static assets with an efficient cache policy — 39 resources found
- Avoid chaining critical requests — 23 chains found
- User Timing marks and measures — 4 user timings
- Keep request counts low and transfer sizes small — 53 requests • 1,257 KiB
- Largest Contentful Paint element — 1 element found
- Avoid large layout shifts — 5 elements found
- Avoid long main-thread tasks — 3 long tasks found

Passed audits (30)

- Eliminate render-blocking resources
- Properly size images
- Defer offscreen images
- Minify CSS
- Minify JavaScript
- Reduce unused CSS — Potential savings of 56 KiB
- Efficiently encode images — Potential savings of 37 KiB
- Serve images in next-gen formats — Potential savings of 133 KiB
- Enable text compression
- Preconnect to required origins
- Initial server response time was short — Root document took 20 ms
- Avoid multiple page redirects
- Preload key requests
- Use HTTP/2

3. MAIN TOOLS

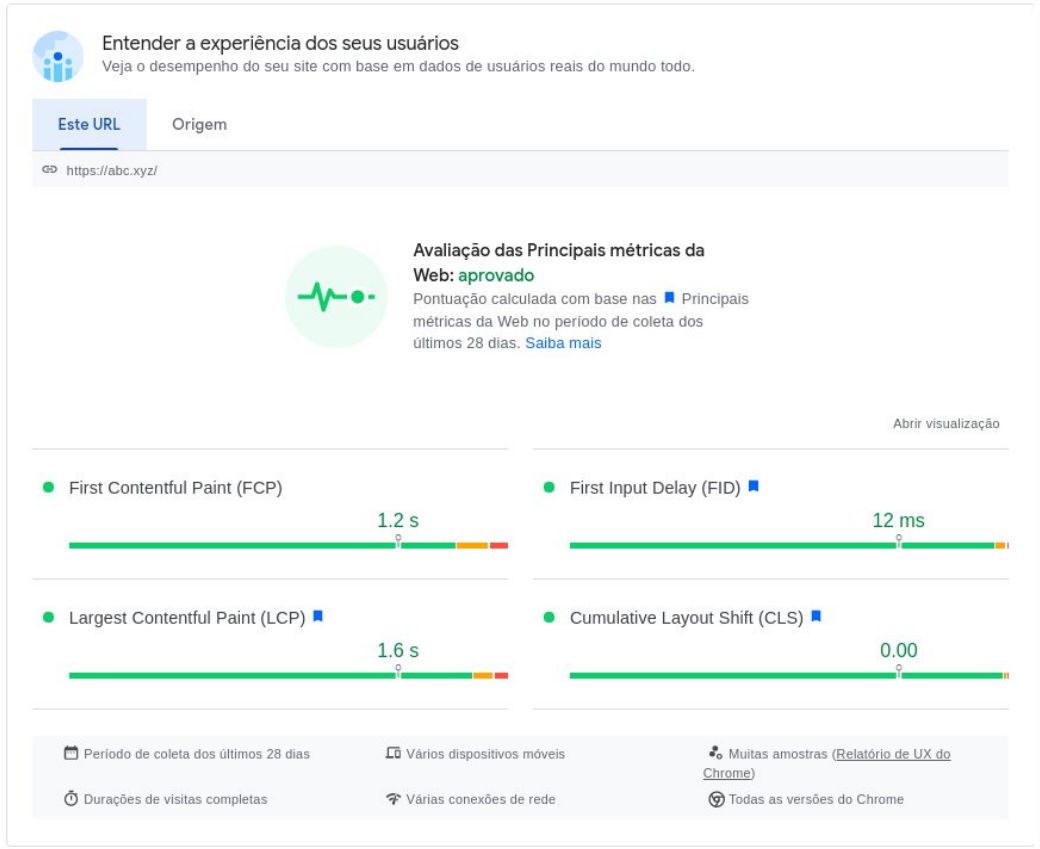
PageSpeed Insights

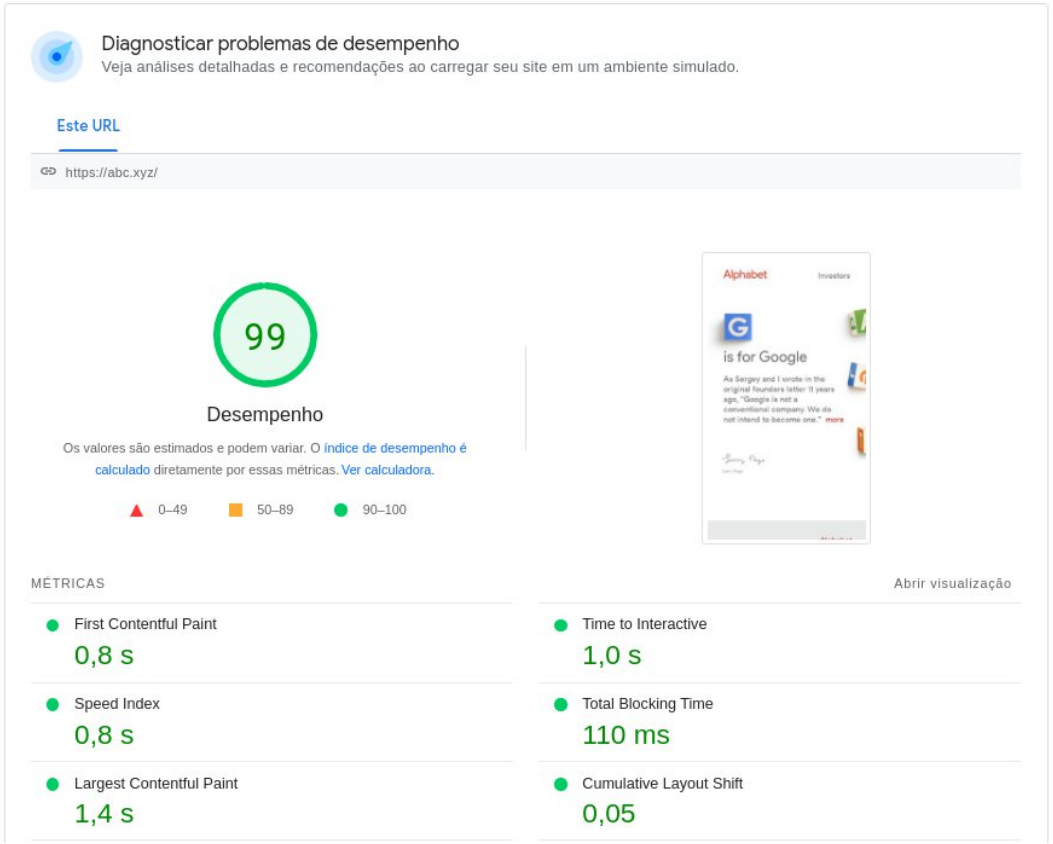
- Is a tool for web developers to understand what a page's performance is and how to improve it;
- It uses lab and field data:
  - Lab data – uses Google Lighthouse to audit the page and identify opportunities to improve performance;
  - Field data – uses Chrome UX Report to show how real users experience the page and the origin in aggregate.

3. MAIN TOOLS

PageSpeed Insights

- Report





3. MAIN TOOLS

PageSpeed Insights

- The PageSpeed Insights report provides a score which summarizes the page's performance;
- The score is determined by running Lighthouse to collect and analyse Lab data about the page;
- A score of 90 or above is considered good. Between 50 to 90 is a score that needs to be improved and a score bellow 50 is considered poor.

3. MAIN TOOLS

PageSpeed Insights

- PageSpeed Insights sets the following thresholds for good/needs improvement/poor, based on our analysis of the Chrome UX Report:

	Good	Needs Improvement	Poor
FCP	[0, 1000ms]	(1000ms, 3000ms]	over 3000ms
FID	[0, 100ms]	(100ms, 300ms]	over 300ms
LCP	[0, 2500ms]	(2500ms, 4000ms]	over 4000ms
CLS	[0, 0.1]	(0.1, 0.25]	over 0.25

3. MAIN TOOLS

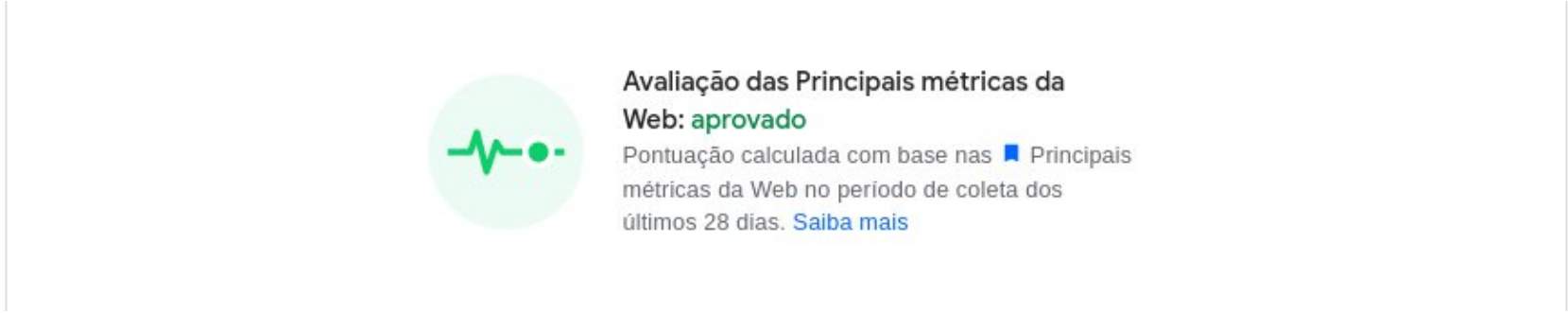
PageSpeed Insights

- Each metric are split into three categories:
  - Good (green bar);
  - Needs Improvement (yellow bar);
  - Poor (red bar).
- For example, seeing 49% within FCP's orange bar indicates that 49% of all observed FCP values fall between 1000 and 3000 milliseconds.

3. MAIN TOOLS

PageSpeed Insights

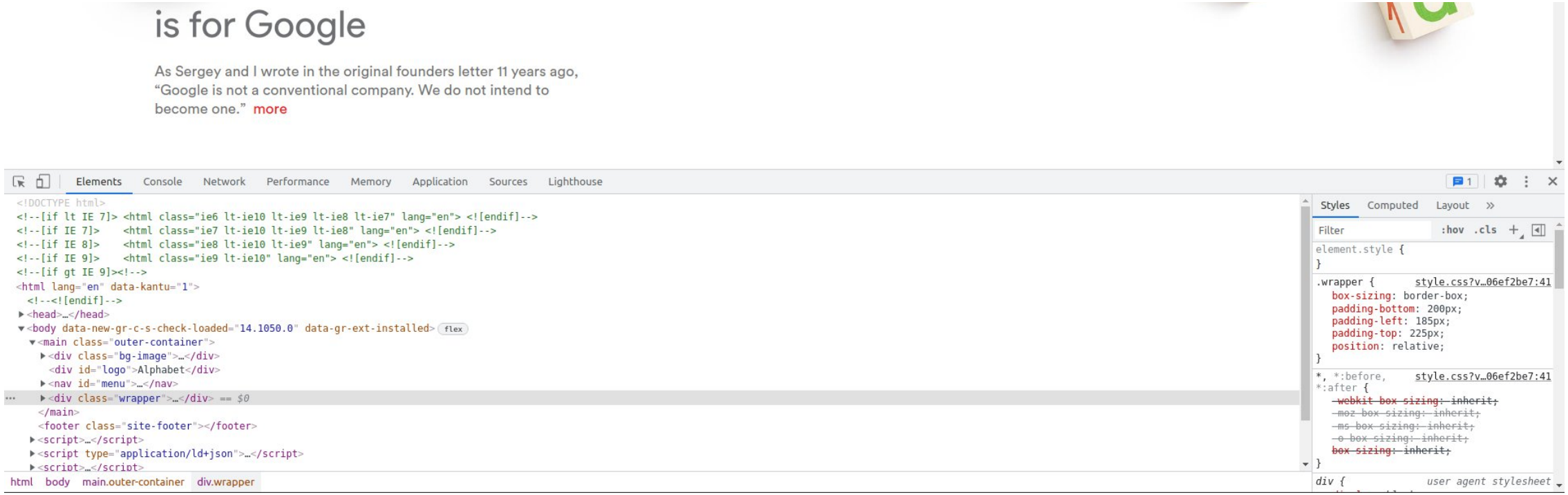
- This data represents an aggregate view of all page loads over the previous 28-day collection period;
- A page passes the Core Web Vitals assessment if the 75th percentiles of all three metrics are Good, otherwise the page does not pass the assessment.



3. MAIN TOOLS

Chrome DevTools

- Set of web developer tools built directly into the Google Chrome browser;
- Access to Crtl+Shift+I or F12.



3. MAIN TOOLS

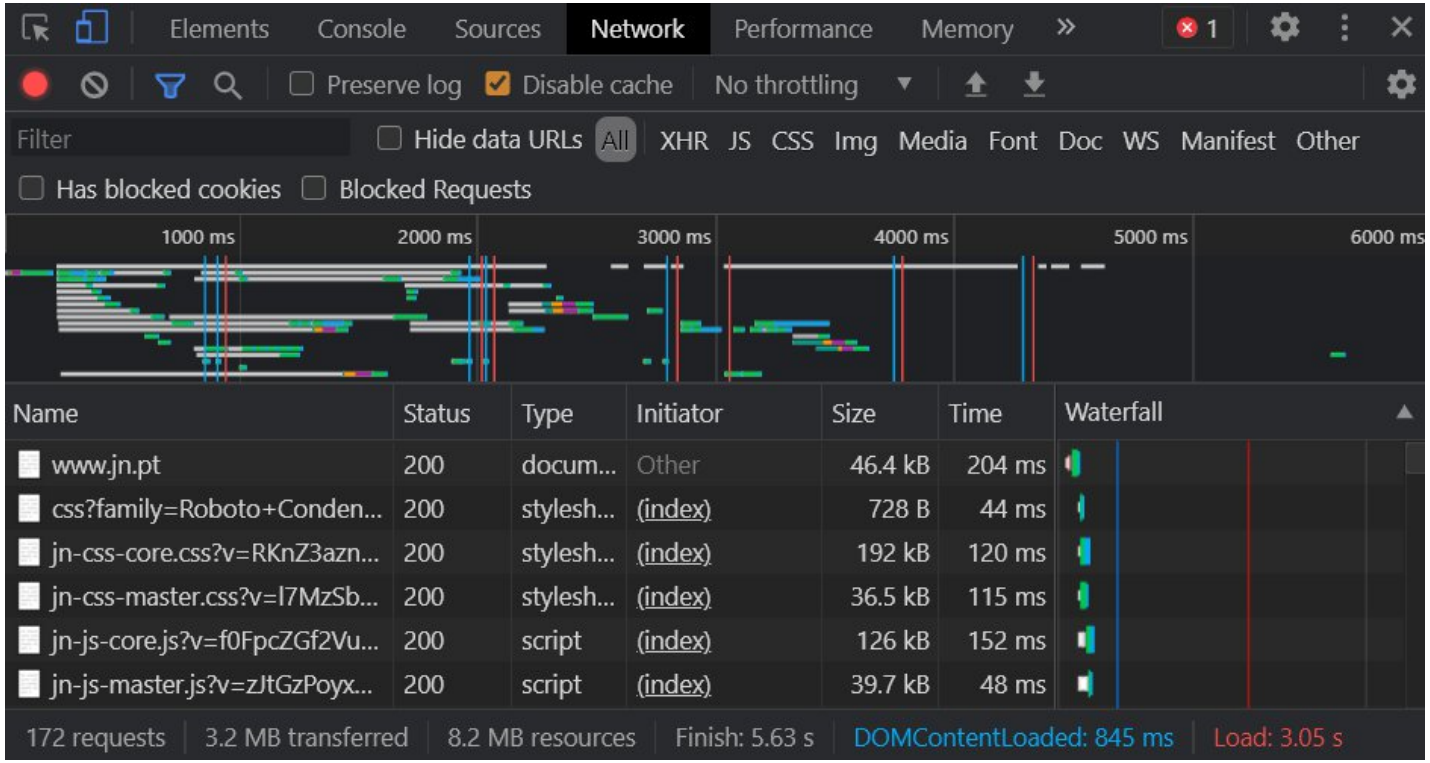
Chrome DevTools

- Currently, includes several tools for analyzing networking and performance:
  - Network;
  - Performance;
  - Lighthouse.

3. MAIN TOOLS

Chrome DevTools

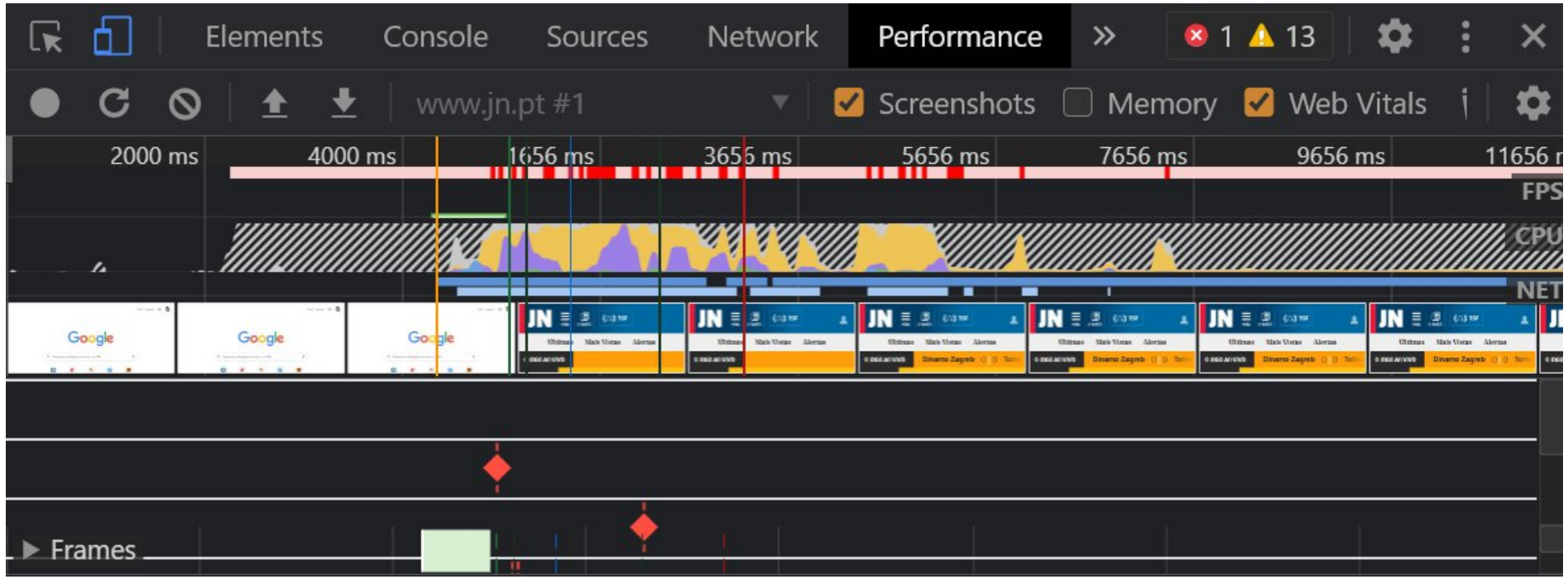
- Network Panel:



3. MAIN TOOLS

Chrome DevTools

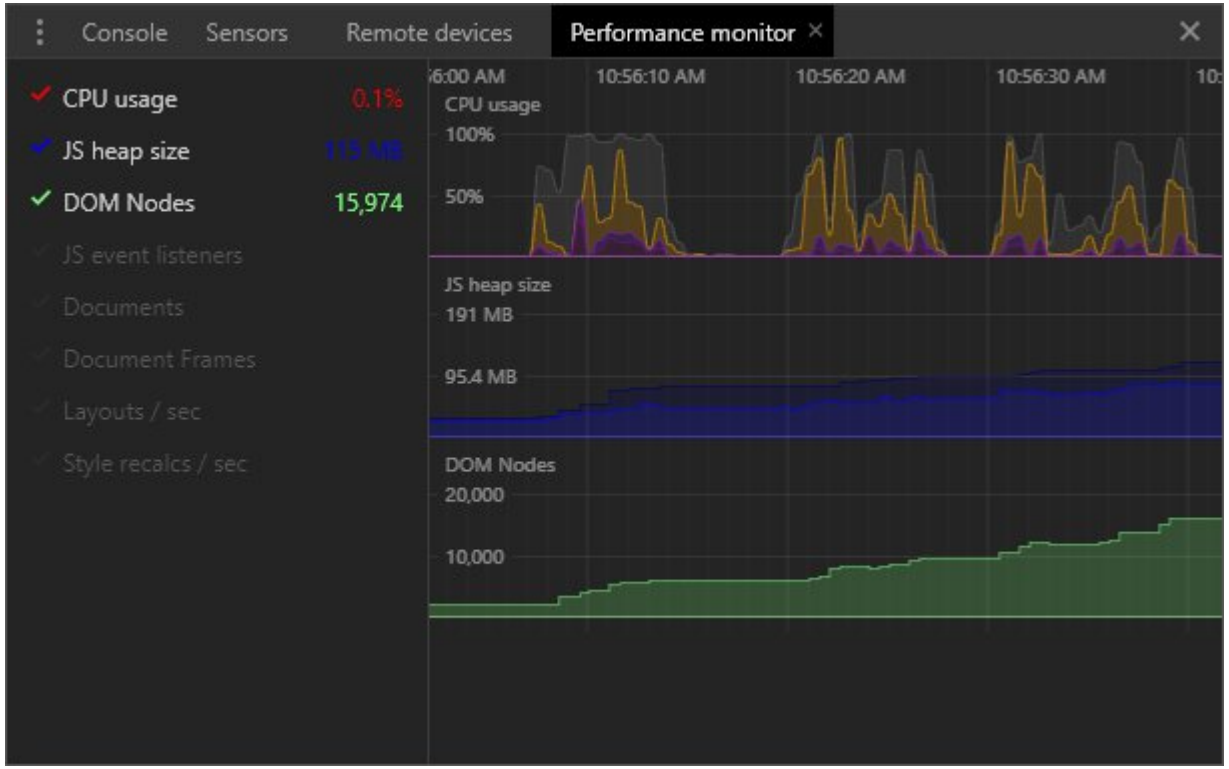
- Performance Panel:



3. MAIN TOOLS

Chrome DevTools

- Performance Monitor:



3. MAIN TOOLS

Chrome DevTools

- **Network Panel**
  - Use Cases for Network Panel are:
    - Making sure that resources are being uploaded/downloaded;
    - Inspecting properties of a resource (HTTP Headers, Content, Size, etc...).

3. MAIN TOOLS

Chrome DevTools

- **Network Panel**
  - Main actions:
    - Log network activity;
    - Show more information;
    - Simulate a slower network connection;
    - Capture screenshots;
    - Inspect a resource’s details;
    - Search network headers and responses;
    - Filter resources: by string, regular expression or property or by resource type;
    - Block requests.

3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Log Network Activity**
  - Reload page to see activity in a logs table;
  - Resources are listed chronologically;
    - The top resource is usually the main HTML document;
    - The bottom resource is whatever was requested last.

3. MAIN TOOLS

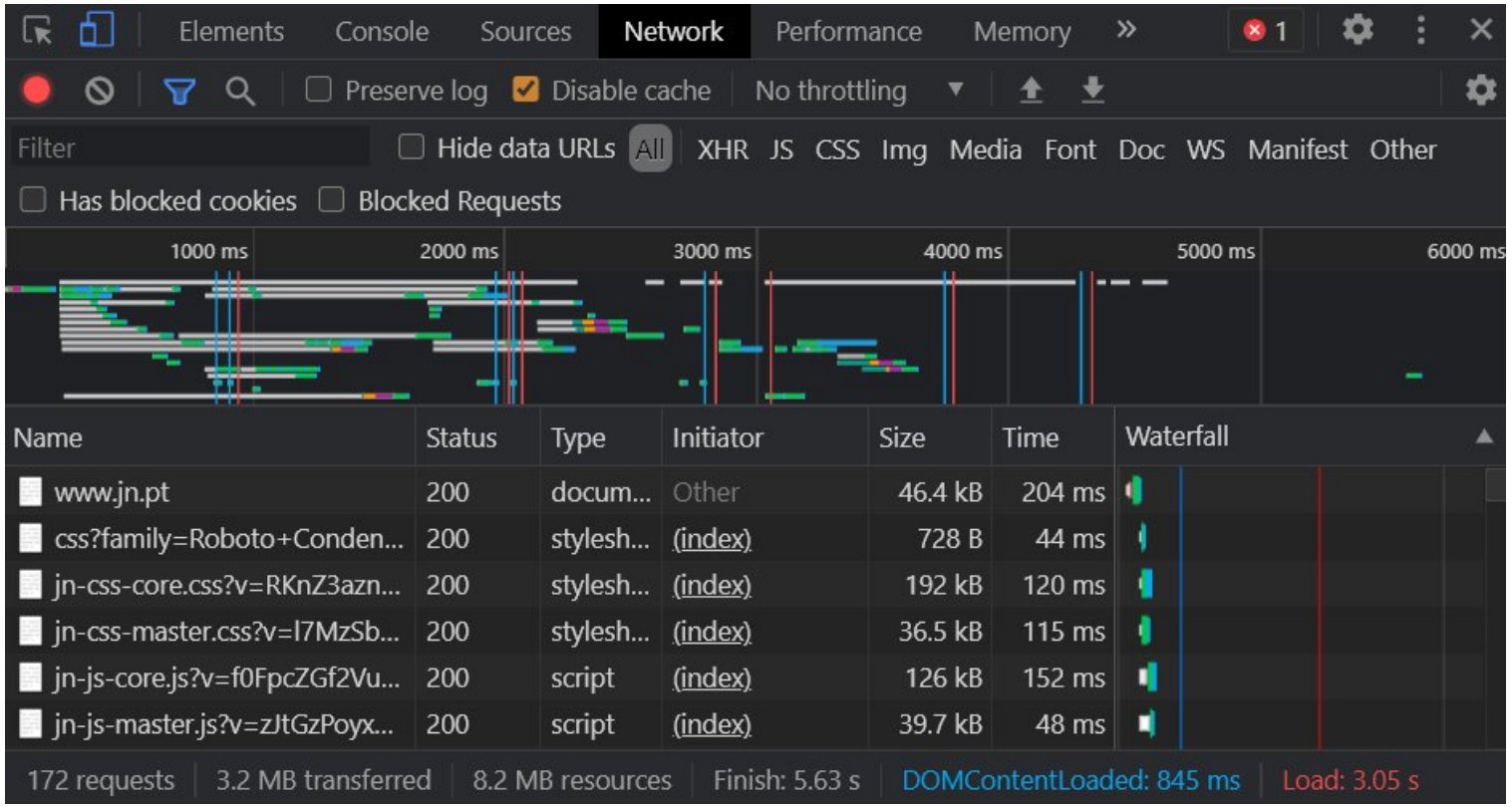
Chrome DevTools

- **Network Panel - Log Network Activity**
  - Each column represents information on resource:
    - Status - the HTTP response code;
    - Type - the resource type;
    - Initiator - what caused a resource to be requested;
    - Time - how long the request took;
    - Waterfall - a graphical representation of the different stages of the request.

3. MAIN TOOLS

Chrome DevTools

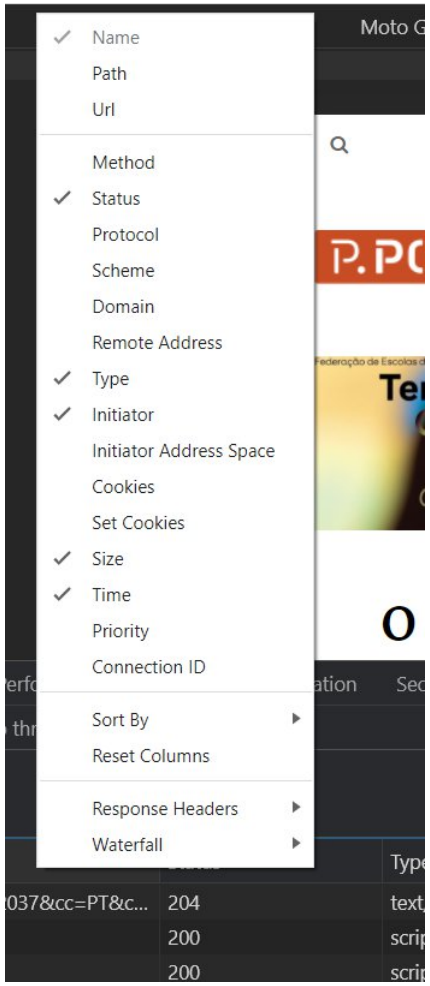
- **Network Panel - Log Network Activity**



3. MAIN TOOLS

Chrome DevTools

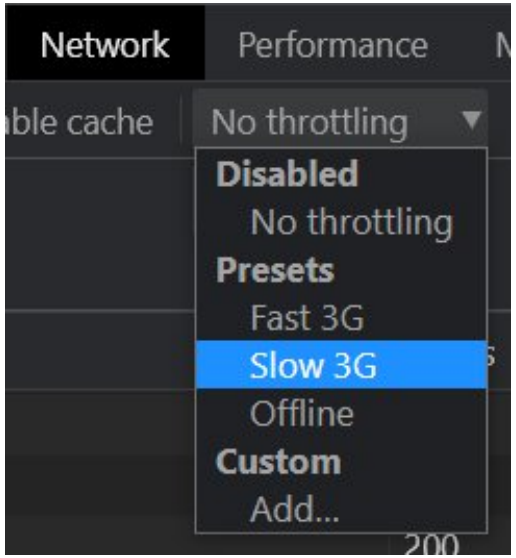
- **Network Panel - Show more information**
  - The columns of the Network Log are configurable;
  - You can hide columns that you are not using;
  - There are also many columns that are hidden by default that can be useful;
  - Right click on the header of the Network Log table to access all columns.

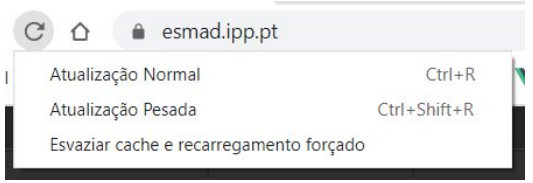


3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Simulate a slower network connection**
  - Network connection of the computer that you use to build sites is probably faster than the network connection of the mobile devices of your users;
  - By throttling the page you can get a better idea how long a page takes to load on a mobile device:
    - Click the throttling dropdown, which is “No throttling” by default and choose among the available presets;
    - Long-press or right-click on reload button and then select “Empty Cache and Hard Reload”.





3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Capture screenshots**
  - Screenshots let you see how a page looked over time while it was loading:
    - Go to settings;
    - Click “Capture Screenshots”;
    - Hit CRTL+R to reload and capture filmstrip;
    - Click the first thumbnail;
    - DevTools shows you what network activity was occurring at that moment in time.

3. MAIN TOOLS

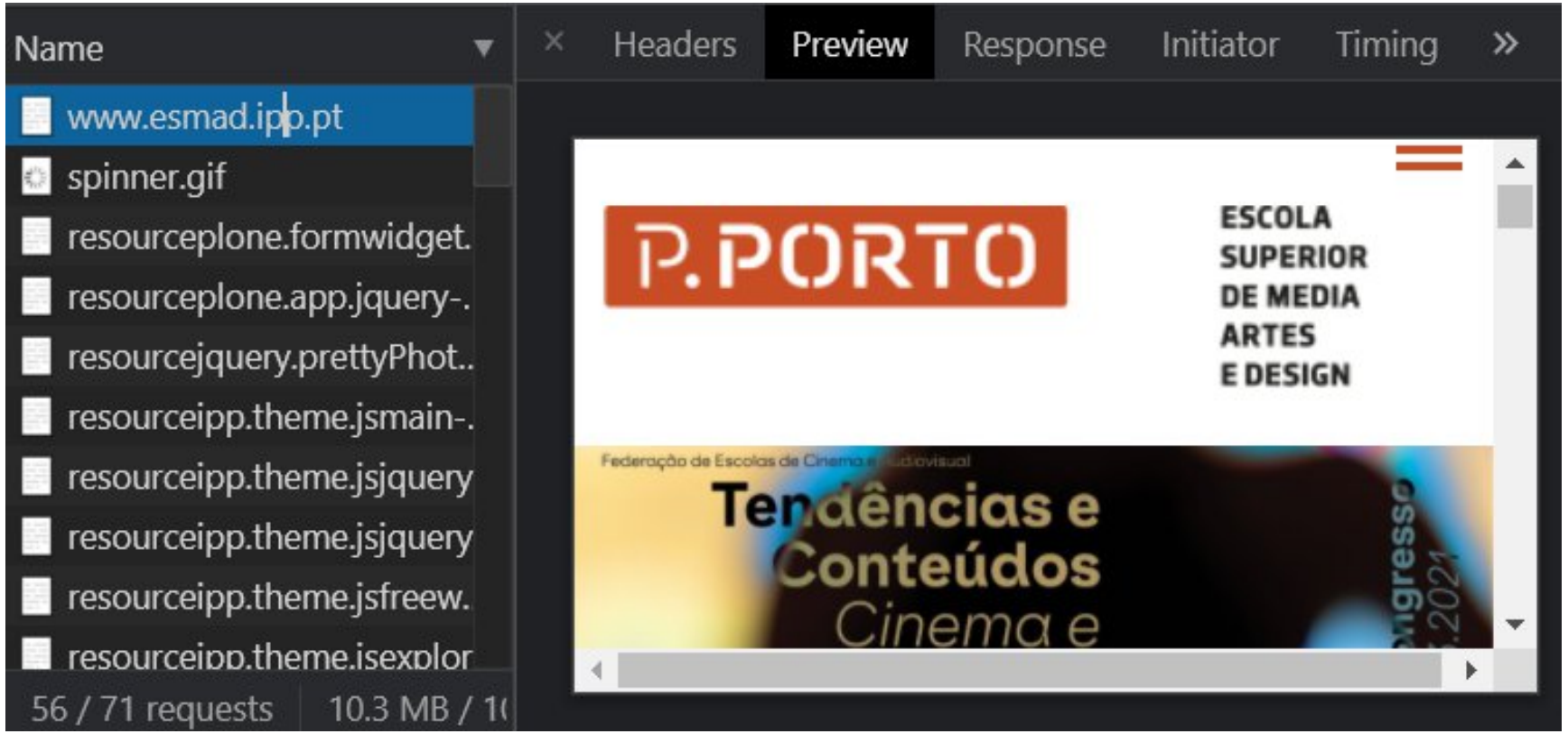
Chrome DevTools

- **Network Panel - Inspect a resource's details**
  - Click a resource to learn more information about it:
    - Headers - info on HTTP headers;
    - Preview - a basic rendering of the HTML is shown;
    - Response - the HTML source code is shown;
    - Initiator - presets a request initiator chain;
    - Timing - a breakdown of network activity is shown;
    - Cookie - list the cookies used in the resource;
    - Close - to view the Network Log again.

3. MAIN TOOLS

Chrome DevTools

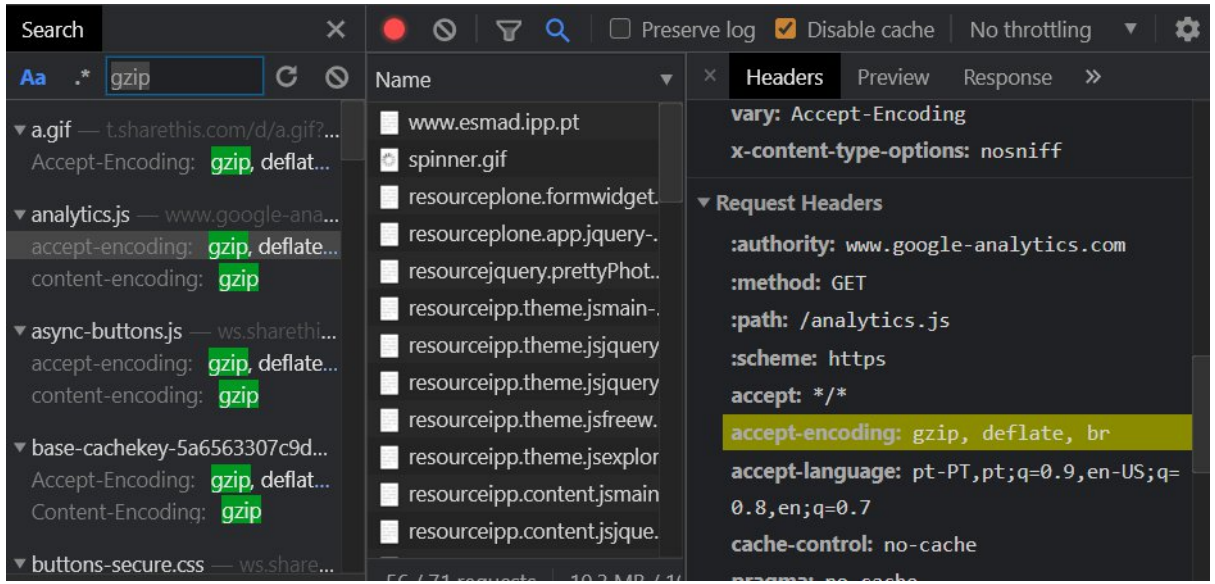
- **Network Panel - Inspect a resource's details**



3. MAIN TOOLS

Chrome DevTools

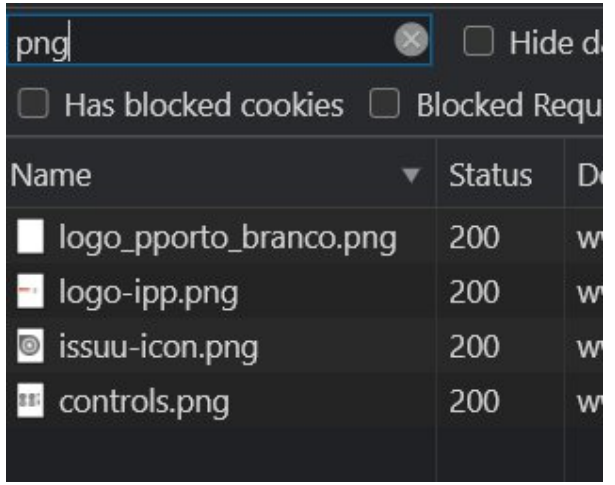
- **Network Panel - Search network headers and responses**
  - Search the HTTP headers and responses of all resources for a certain string or regular expression;
  - Click search to open the search pane;

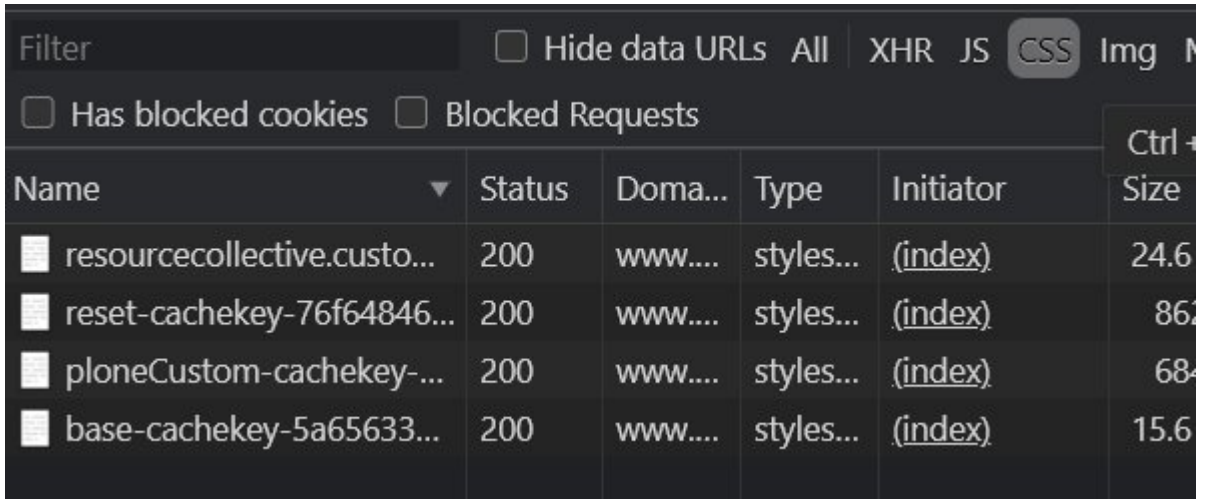


3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Filter results**
  - Click filter;
  - Filter types:
    - String, regular expression, or property;
    - Resource type.





3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Filter results**
  - Filters examples:
    - Is:running - show any incomplete or unresponsive requests;
    - larger-than:S - limit to files larger than S, which can be expressed as bytes (100), kilobytes (100k), or megabytes (100M);
    - smaller-than:S: limit to files smaller than S, which can be expressed as bytes (100), kilobytes (100k), or megabytes (100M);
    - domain:*.yourdomain.com - show third-party requests that are not from you primary domain.

3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Block requests**
  - How does a page look and behave when some of its resources are not available?
  - Does it fail completely, or is it still somewhat functional?
    - Press CRTL+SHIFT+P to open the command menu;
    - Type “block”, select “Show Network Request Blocking” and press Enter;
    - Click “Add Pattern”;
    - Add file to block.

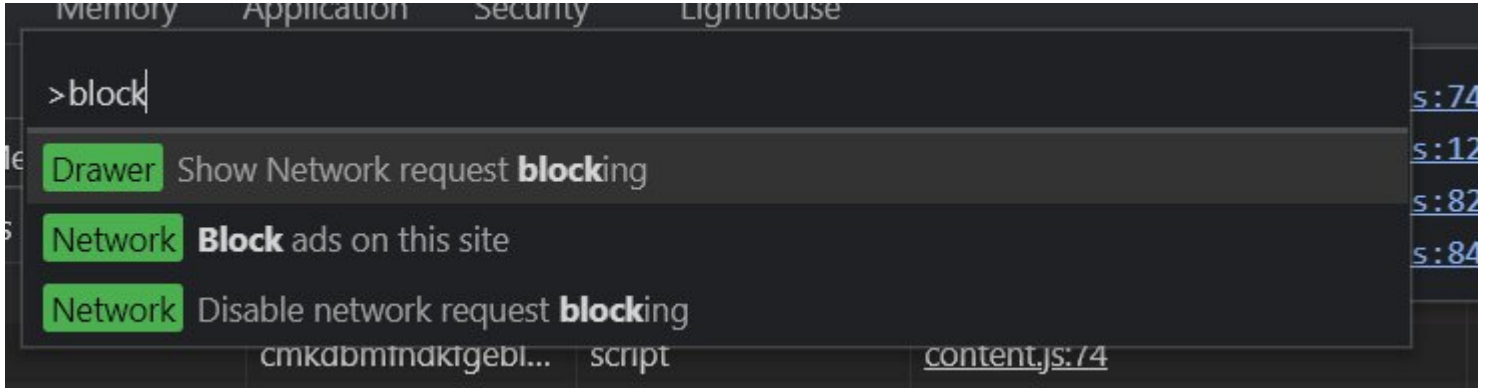
    OR

    - Select the resource;
    - Right click and choose between “Block request URL” or “Block request domain”.

3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Block requests**



3. MAIN TOOLS

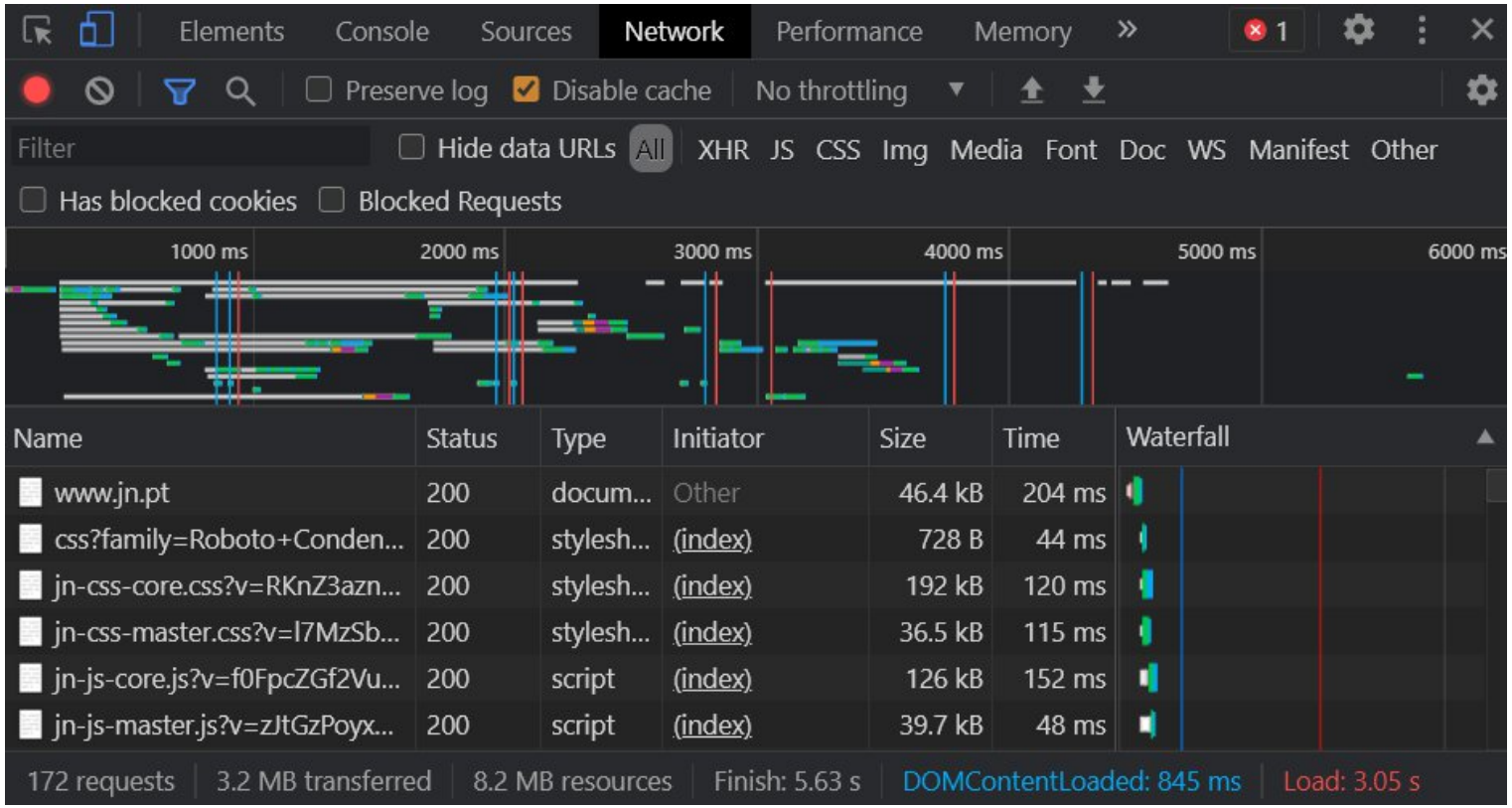
Chrome DevTools

- **Network Panel - Bars**
  - The status bar at the bottom summarizes:
    - The number of requests;
    - Total data transfer (possibly compressed);
    - The total size of all uncompressed resources;
    - The total download time;
    - The time when the document DOMContentLoaded and Window load events are triggered.

3. MAIN TOOLS

Chrome DevTools

- **Network Panel - Bars**



3. MAIN TOOLS

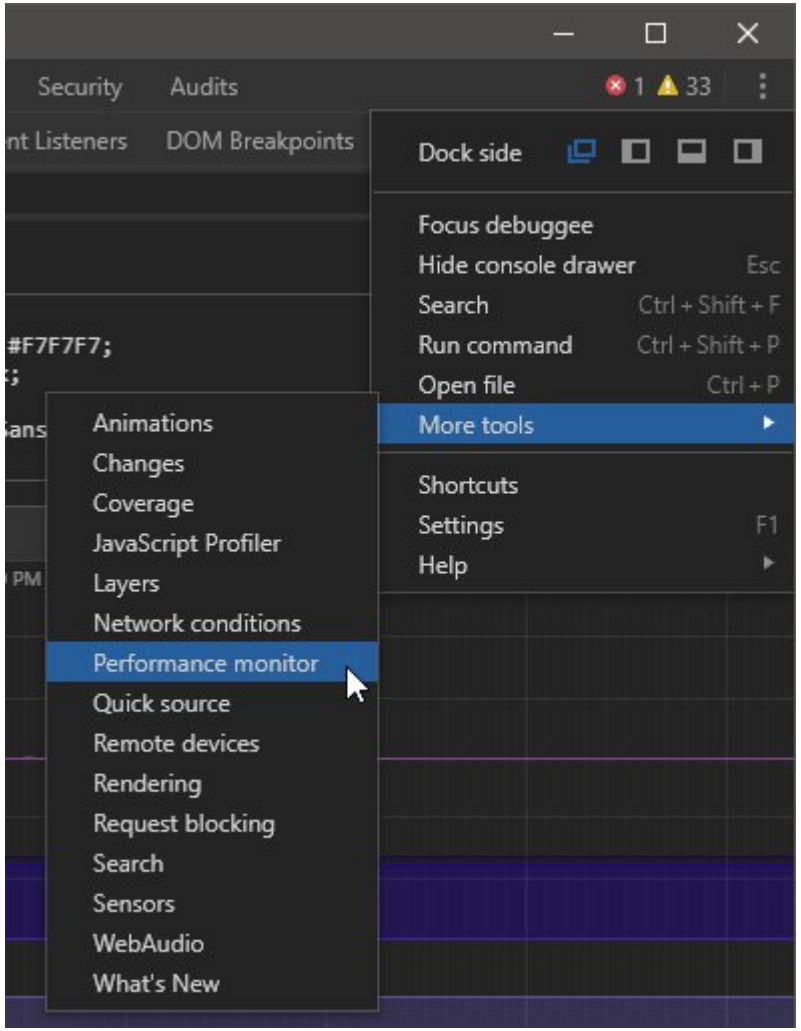
Chrome DevTools

- **Network Panel - Bars**
  - Further options are provided at the top:
    - Preserve logs - don’t clear the log between page loads;
    - Disable cache - load all files from network to make a better assessment of 1st time page access;
    - Throttle network speed – select or define download speed profiles.

3. MAIN TOOLS

Chrome DevTools

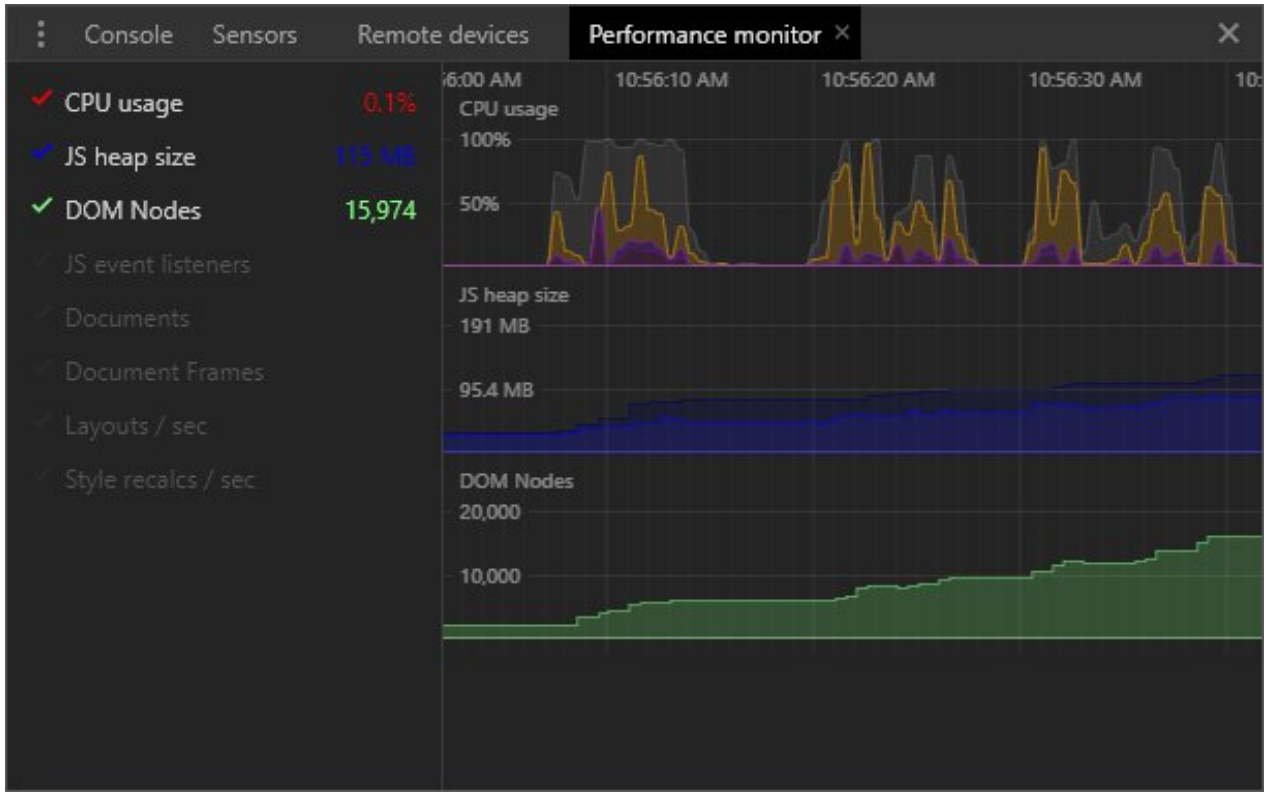
- **Performance Monitor**
  - Can be accessed from the DevTools' More Tools sub-menu;
  - Could be used do discover unusual spikes in activity, such as rising memory use or layout recalculations when an element has been added to the page;
  - Further investigation can then be carried out in the Performance Panel.



3. MAIN TOOLS

Chrome DevTools

- **Performance Monitor**
  - It appears in the lower console drawer panel, and charts are updated in real time as you use a page



3. MAIN TOOLS

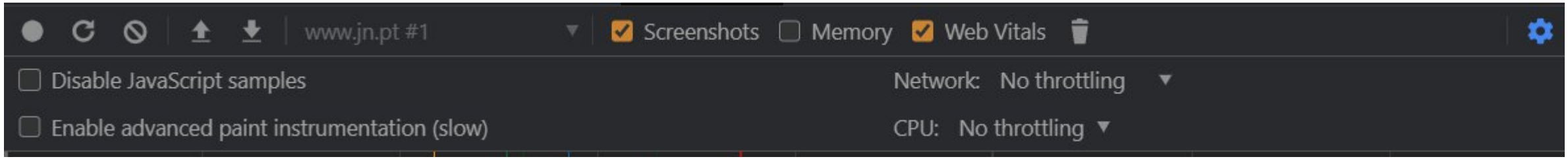
Chrome DevTools

- **Performance Monitor**
  - Several performance monitors:
    - **CPU usage:** processor utilization from 0% to 100%;
    - **JS heap size:** memory required for JavaScript Objects;
    - **DOM nodes:** the number of elements in the HTML document;
    - **JS event listeners:** the number of registered JavaScript listeners;
    - **Documents:** the number of document resources including the page, CSS, JS, etc…;
    - **Document frames:** the number of frames, iframes, and work scripts;
    - **Layout/sec:** the rate at which the browser has to re-layout the DOM;
    - **Style recalcs/sec:** the rate at which the browser has to recalculate styles.

3. MAIN TOOLS

Chrome DevTools

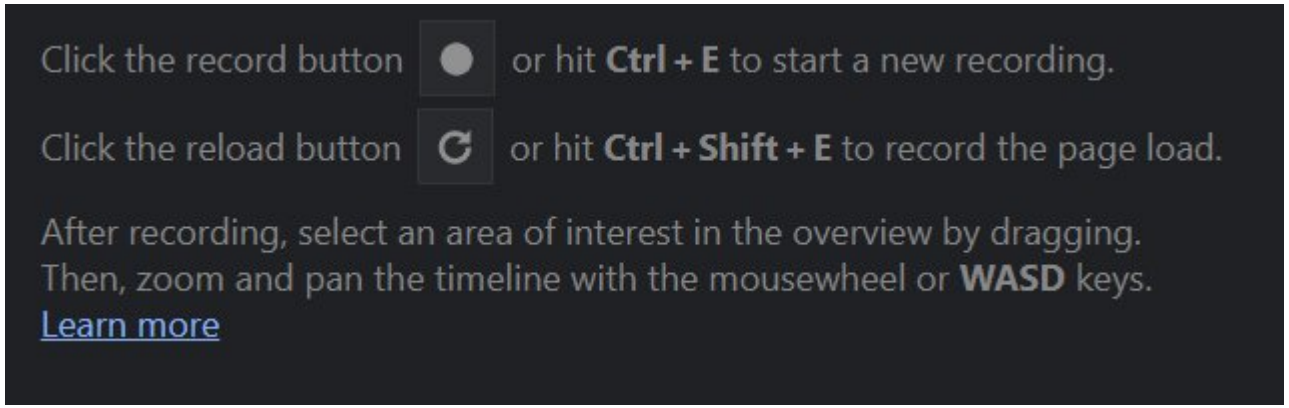
- **Performance Panel**
  - Use Cases for Performance Panel are:
    - Allows you to record a snapshot of browser activity when particular actions are made
  - Cog icon let us define throttling scenarios;



3. MAIN TOOLS

Chrome DevTools

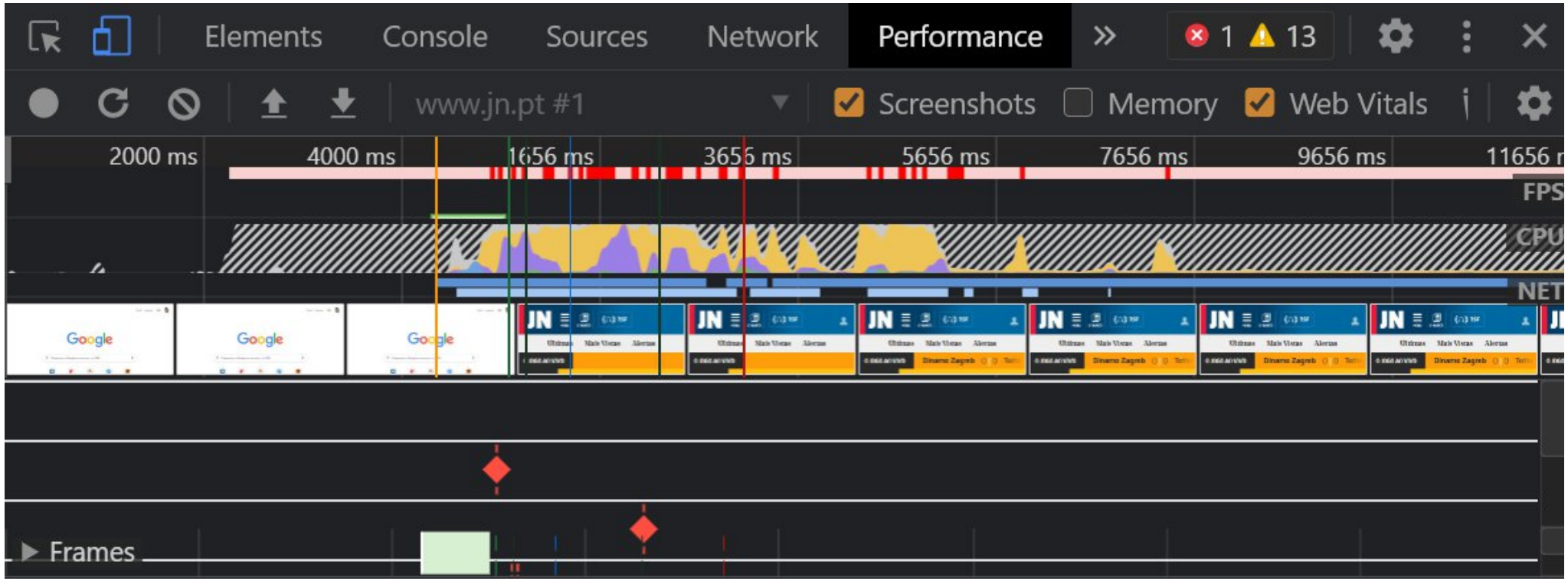
- **Performance Panel**
  - Record a profile before it can be analyzed;
  - Hit stop to generate the performance report.



3. MAIN TOOLS

Chrome DevTools

- **Performance Panel**



3. MAIN TOOLS

Chrome DevTools

- **Performance Panel**
  - The top timeline chart shows frames per second, CPU Usage, network usage, screenshots, and the heap memory size;
  - An area can be selected with the mouse to focus on a specific point;
  - Result panes can be expanded and collapsed. Panes may include:
    - Network - loading time for individual files;
    - Frames - screenshots at points on the timeline;
    - Interactions - input and animation timings;
    - Timings - events such as DOMContentLoaded, LCP, etc...;
    - Main - thread activities such as function calls and events handlers.

3. MAIN TOOLS

Performance Timing APIs

- Performing monitoring can help discover problems when specific actions are performed in a site or app;
- However, it may become necessary to profile JS execution by logging messages to the console when events occur;
- Modern browsers supports Performance Timing APIs to help to analyze code;
- Link: https://developer.mozilla.org/pt-BR/docs/Web/API/Performance;
- Most important methods:
  - Performance.now()
  - Performance.mark()
  - Performance.measure()

3. MAIN TOOLS

Performance Timing APIs

- **performance.now()**
  - Returns the elapsed time in milliseconds since the page was loaded;
  - Unlike Date.now(), it returns a floating-point number representing factions of a millisecond.

```
let t0 = performance.now();
doSomething();
let t1 = performance.now();
console.log(`doSomething() executed in ${t1 - t0}ms`);
```

3. MAIN TOOLS

Performance Timing APIs

- **performance.mark()**
  - Can become arduous to manage as an application grows;
  - The API also allows you to mark when an event occurs and measure the time elapsed between two marks;
  - A mark is defined by passing a name string to performance.mark().

```
performance.mark('script:start');

performance.mark('doSomething1:start');
doSomething1();
performance.mark('doSomething1:end');

performance.mark('doSomething2:start');
doSomething2();
performance.mark('doSomething2:end');

performance.mark('script:end');
```

3. MAIN TOOLS

Performance Timing APIs

- **performance.mark()**
  - A mark can be cleared with performance.clearMarks(markName);
  - All marks are cleared when no name is passed;
- **performance.measure()**
  - The elapsed time between two marks can be calculated by creating a performance.measure() by passing the measure name, start and end mark.

```
performance.measure('doSomething1', 'doSomething1:start', 'doSomething1:end');
performance.measure('script', 'script:start', 'doSomething1:end');
```

3. MAIN TOOLS

Performance Timing APIs

- **performance.measure()**
  - Omitting the start mark measures from all the moment the page loaded;
  - Omitting the end mark measures to the current time;
  - Each measure() call adds a PerformanceEntry object to the same array, which defines the name, StartTime, and duration;
  - A measure can be cleared with performanceClearMeasures(measuresName);
  - All measures are cleared when no name is passed.

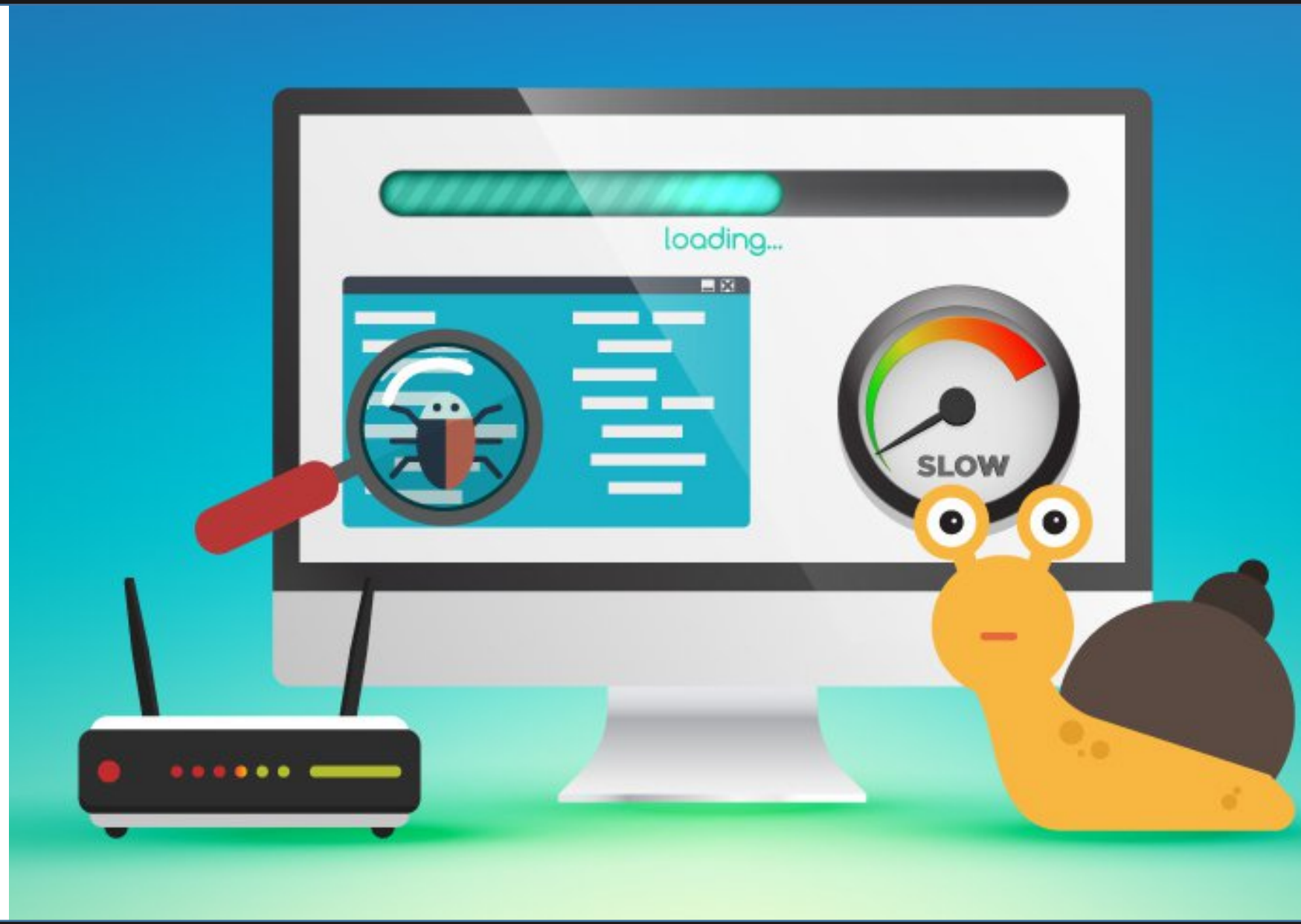
3. MAIN TOOLS

Performance Timing APIs

- Marks and measures can be accessed with:
  - performance.getEntriesByType(type) - type could be “mark” or “measure”;
  - performance.getEntriesByName(name) - where name is a mark or measure name;
  - performance.getEntries() - to access all items in the array.
- The name and duration of every measure can therefore be output:

```
performance.getEntriesByType('measure')
    .forEach(m => console.log(`${m.name}: ${m.duration}ms`));
```

FINAL REMARKS



4. FINAL REMARKS

- Use tools to automate the process of analysing performance in a website;
- More precisely:
  - Use **PageSpeed Insights** (Lighthouse and Chrome UX Report) to diagnose lab and field issues on the pages;
  - Use **Lighthouse** and **Chrome DevTools** to measure Core Web Vitals and get actionable guidance on exactly what to fix;
  - Use **Performance APIs** to profile JavaScript execution by logging messages to the console when events occur.