**Developer Tools**

This document contains a brief overview of some features of developer tools and their usefulness in debugging.

* Developer tools:
* Helps developers understand, debug, and optimize web pages.
* Used for checking the structure, behaviour, performance of the webpage.
* State management:
* State of an application: The data delivered to and produced by the application that represents the current condition or situation of that application, e.g. user input, data fetched from server, etc.
* State management: Process of maintain knowledge of an application’s inputs to understand the condition of the application at any given moment. This makes the state of an application visible and helps in keeping track of changes and update the user interface accordingly to ensure that the application behaves as expected.
* Purpose: Helps identify and fix bugs related to data flow and user interactions.
* Developer tools and debugging:

1. **Elements**

* To inspect and modify the HTML and CSS of a web page.
* Allows to view and edit structure and styles of a page in real-time.

1. The **HTML DOM structure** can be viewed (this includes changes made by the JS code too).
2. **Styles** tab: View the CSS styles applied to each HTML element.
3. **Computed**: Shows final computed styles (actual styles that an HTML element has after considering all the applied styles, including styles from external stylesheets, internal styles, and inline styles), along with the box-model.
4. List of all the **event listeners** applied to the page is available in the event-listeners section.
5. **Layout**: allows to view the grid and flexbox overlays of the webpage.
6. **Console**

* Allows to execute JS code, view console.log messages, and interact with the document using the command line

1. Logs all the **errors and warnings**, useful for debugging purposes.
2. Logs can be preserved between page navigations in order to track logs over time.
3. Context of the console can be changed to different embeds of the webpage, allowing debugging in different parts of the page.
4. **Sources**

* Contains all assets fetched by browser – ({} pretty print makes it readable).
* Usage: **debug, step through code, inspect variables**, etc.
* Provides a code editor for debugging.

1. **Page** sub-tab contains the source files.
2. **Snippets** sub tab - can save and run JavaScript code snippets.
3. **Watch** - To observe expressions and variables and how they change throughout the execution of the code.
4. **Breakpoints**: set breakpoints for pausing code execution, allows step-by-step debugging.
5. **Call stack** - helps check how a debugger point was reached in JS code execution.
6. **DOM breakpoints** - Allows to pause on specific DOM changes for debugging.
7. **Event listener breakpoints**: allows to pause execution on specific types of events for debugging event-related issues.
8. **Network**

* Contains information about **network activity.**
* Helps understand how web page interacts with server, identify failed requests, and more.
* Shows details about each request made by the application, like request and response headers, status code, timing, etc.
* Essential for optimizing page load times and debugging API calls.

1. **Logs** can be preserved.
2. Allows to test the application without network throttling, simulating fast and reliable network.

Different **network conditions and speeds can be simulated** like Fast 3G, offline, etc. to test application’s performance under various scenarios.

1. Requests can be **grouped and filtered**.
2. Details about each request can be viewed:
3. **Status** of the request
4. **Type** of the request (ex. ping, fetch)
5. **Initiator**- information about what initiated the network request
6. **Size** of the data transferred
7. **Timing** section - provides information that helps identify the bottleneck (point where flow or speed of data is significantly reduced)
8. **Waterfall chart** representing the timeline of network requests
9. **Headers** - Contains General details (request URL, status code, request method, remote address, etc.) that helps understand which request failed and on what path, Response headers, Request headers, etc.
10. **Preview** - contains a visual preview of the changes made on the webpage by the particular request
11. **Response** section contains the actual data that was sent by the server as a response
12. **Application**

* Provides info about **resources** a webpage uses, e.g. web app manifest (a JSON file that provides metadata about the web application) if present, registered service workers (background scripts in web application that enable features like offline functionality and act as intermediaries between the application, the browser, and the network), etc.
* Allows to clear storage types like cookies and cache.
* Storage mechanisms can be inspected, modified and cleared, helping in effective management of client-side state.
* Contains information about local and session storage key-value pairs on client side.

1. **Local** storage: Small amounts of data that persists on client browser across browser sessions.
2. **Session** storage: Data that persists for the duration of the tab or browser session.
3. **Indexed DB**: A client-side database for storing larger amounts of data.
4. **Cookies**: Small pieces of data sent from a website and stored on user’s device used for tasks like tracking user preferences, etc.
5. **Cache** storage: Assets like images, stylesheets etc. that are cached on the client browser to reduce need to downloading same resources repeatedly.
6. **Memory**

* Analyze and optimize memory usage of application, detect memory leaks (when browser uses more and more memory over time without releasing it back).

Types of memory profiling:

1. **Heap snapshot** allows to take a snapshot of the JavaScript heap at a specific moment to help identify memory-related issues, check spike in memory usage, etc.
2. **Allocation instrumentation**: Monitors memory allocation over time to help understand how memory is allocated during execution of the code.
3. **Allocation sampling**: samples memory allocation at regular intervals of the program’s execution and provides a statistical view of memory allocation patterns.
4. **Performance**

* Analyze runtime performance of webpage.
* Can record and analyze various **performance metrics** like CPU usage, network activity, etc. over time.