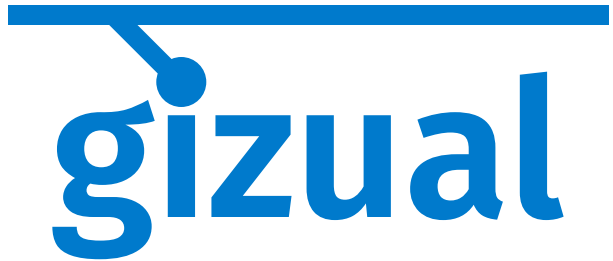



# Browser-Based Git Repository Visualisation

## with WebAssembly and Web Workers

https://gizual.com/

GrazJS, Tue 25 Jun 2024

Github: https://github.com/gizual/gizual
Resources: https://gizual.com/resources

# About us

## Stefan Schintler

steschi

Omnity e.U. since 2014

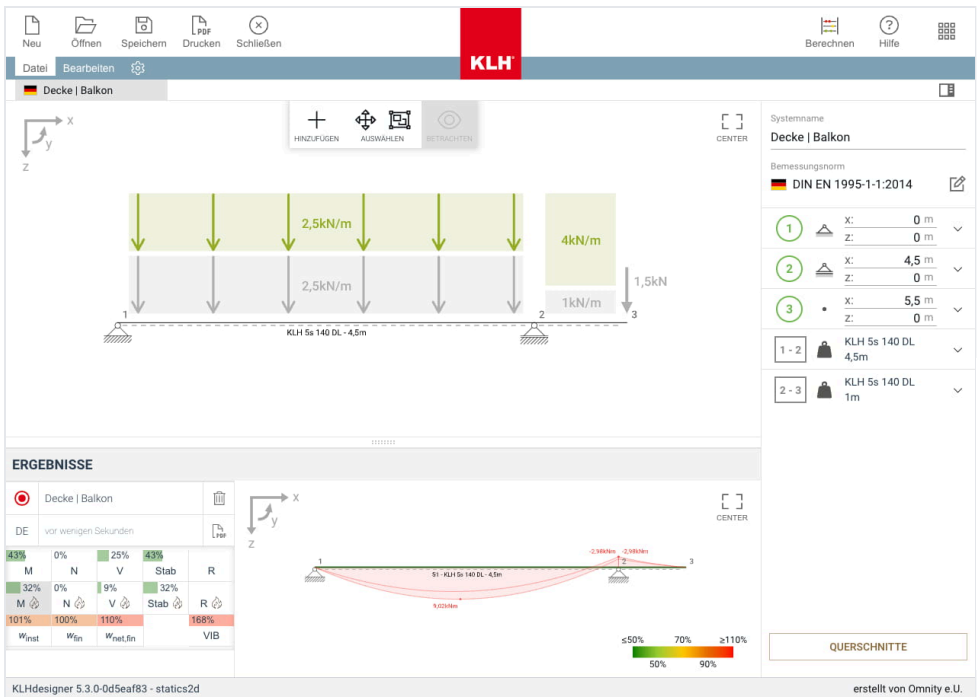
## Andreas Steinkellner

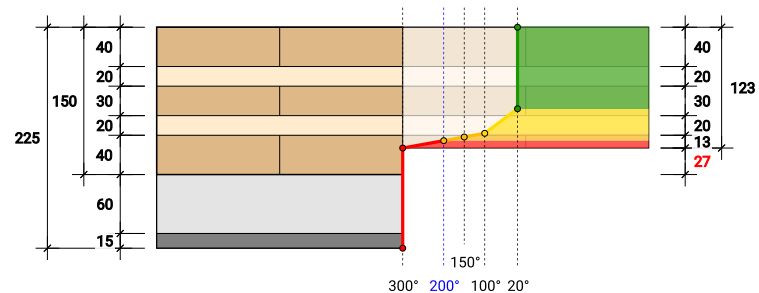
NarrowCode

Solasit e.U. since 2022

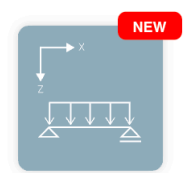
# What we do

## KLH Massivholz GmbH



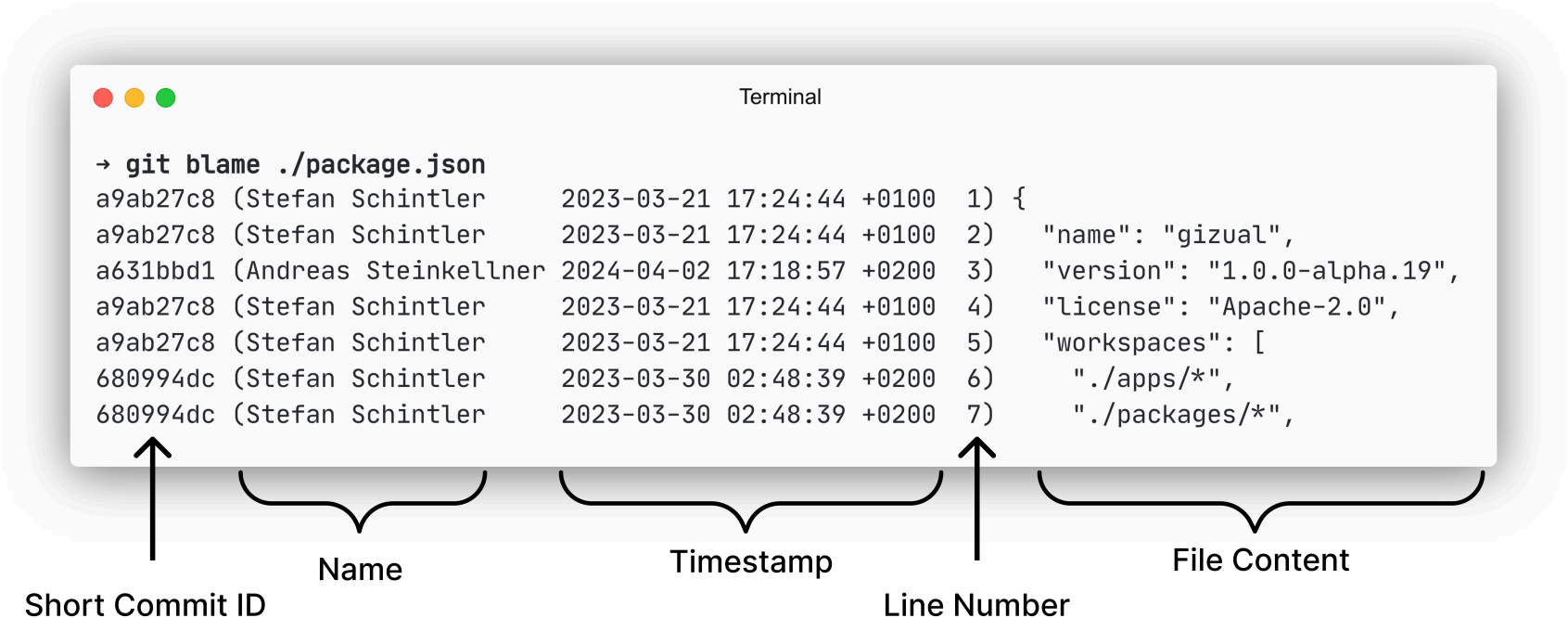

data+




statics2d

co2

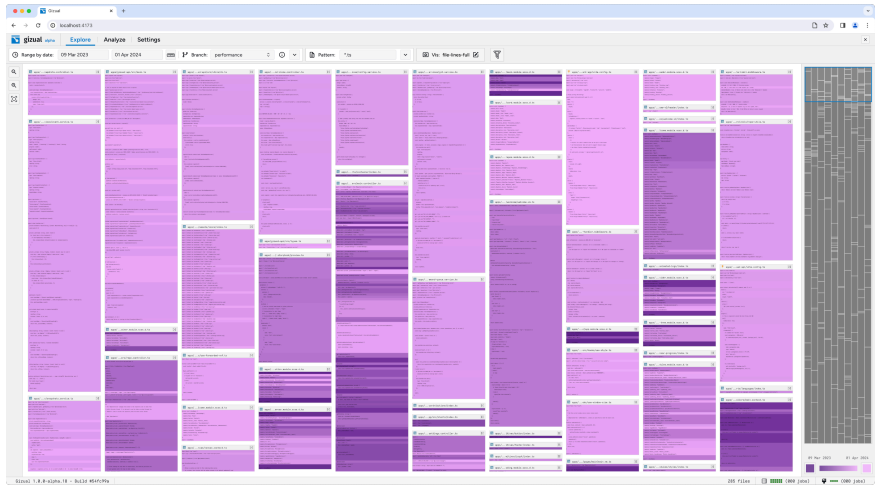
# Git Blame [2006]




*Git blame, a tool for assigning blame.*; Fredrik Kuivinen;
Git commit cbfb73d; 21 Feb 2006. https://github.com/git/git/commit/cbfb73d73f272f194bafa70d5be1f35abc265c79

# Gizual [2022→]

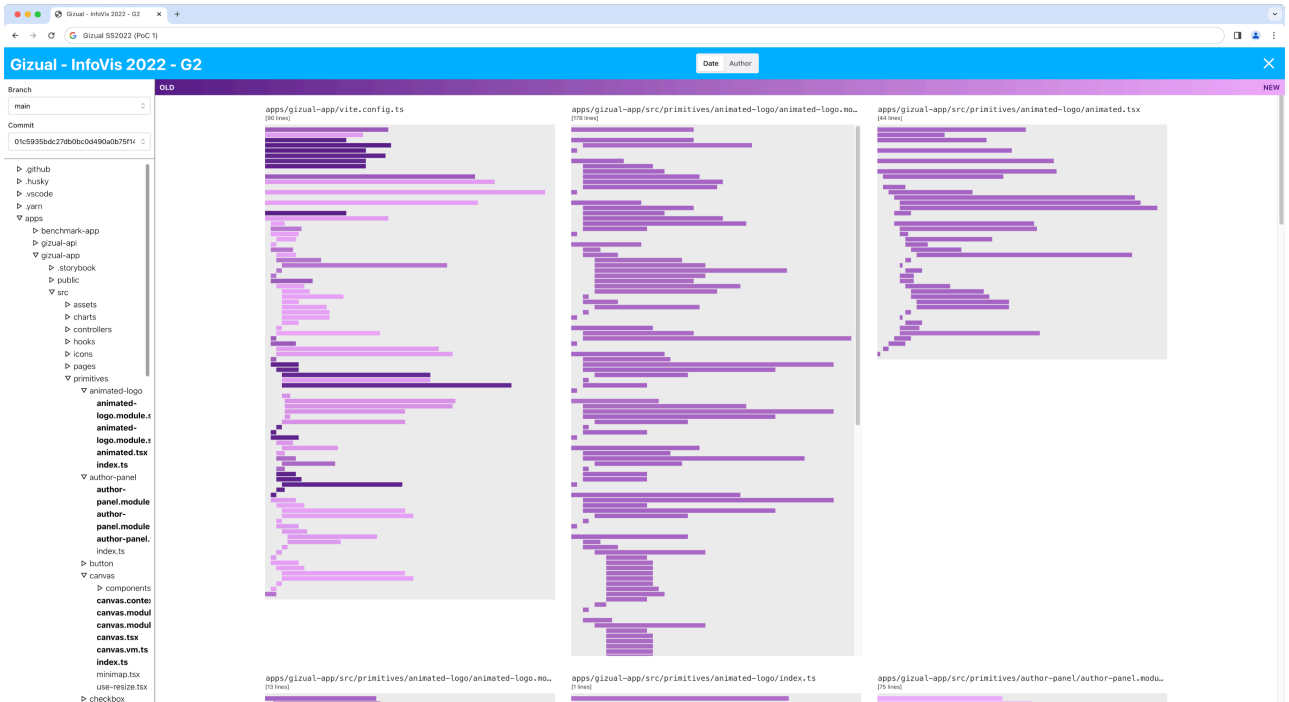
- Git visualisation inside browser
- Browsers and devices are both more powerful.
- Native code inside browser via WebAssembly
- Backend: not necessary
- Frontend: TypeScript, React, Mantine, WebAssembly (libgit2), Web Workers



Gizual, 2024.

# Project History

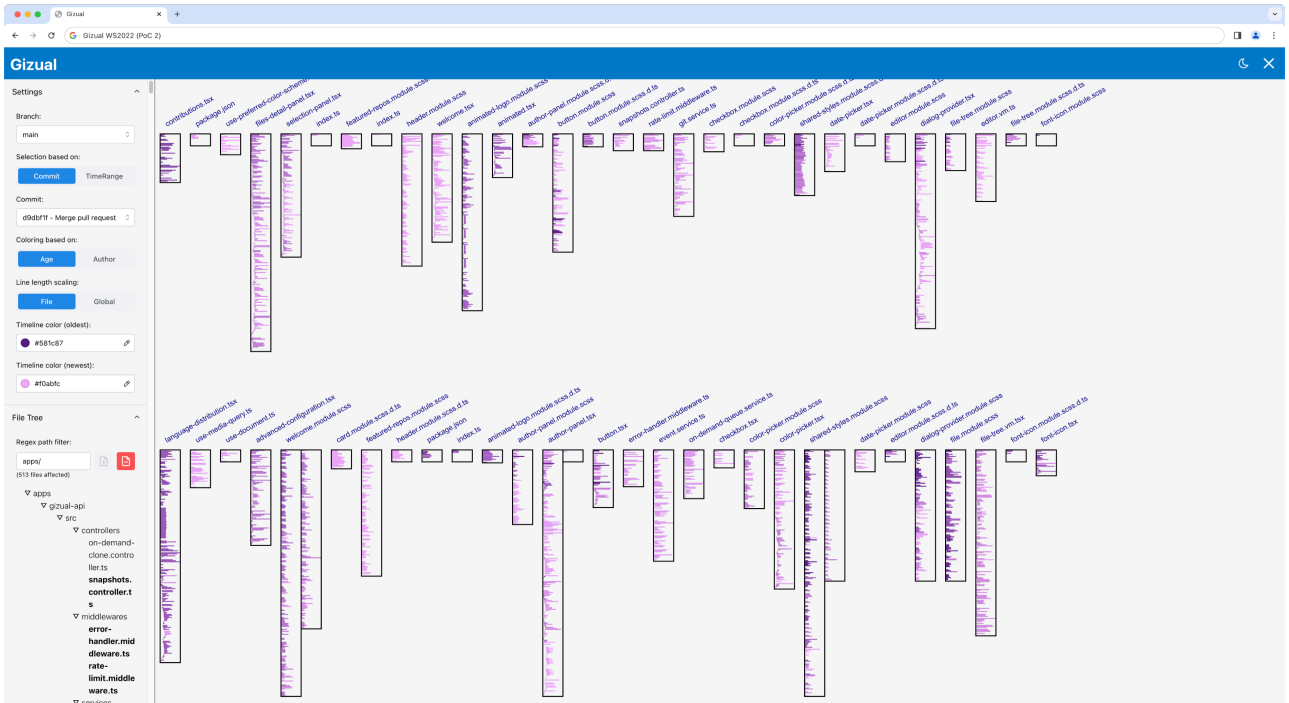
# Proof of Concept 1 [SS 2022]




POC1[2] [May 2022 → June 2022]

1. Gizual - Repository Visualization for Git (SS 2022), seminar paper - https://gizual.com/resources/gizual-paper-ss2022.pdf

# Proof of Concept 2 [WS 2022/23]

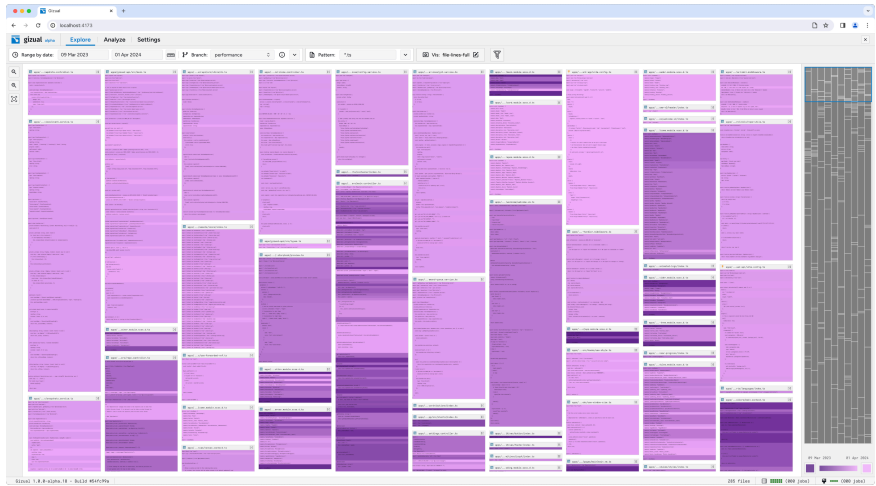



POC2[1] [Dec 2022 → Jan 2023]

1. Gizual - Repository Visualization for Git (WS 2022), seminar paper - https://gizual.com/resources/gizual-paper-ws2022.pdf

# Gizual v1.0 (Master's Theses) [SS 2023→]

- Reimplement:
  - Git core (C / Rust)
  - Project architecture (multiple times 😅)
  - Frontend and visualization core
  - Better performance
- Extend:
  - More visualization options
  - Cross-browser, mobile and PC

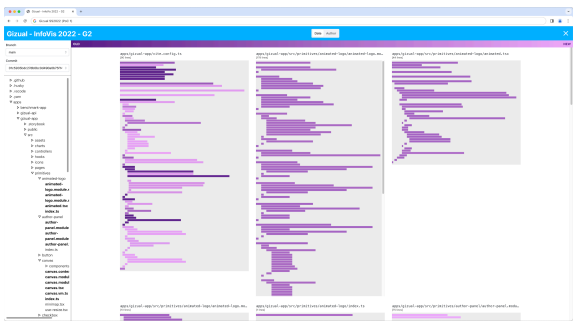


Gizual v1.0[1] [Mar 2023→]

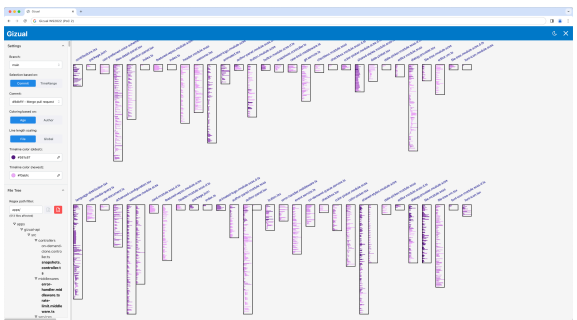
1. Gizual - Browser-based Git repository visualization, website - https://gizual.com

# Lessons from 3 Iterations

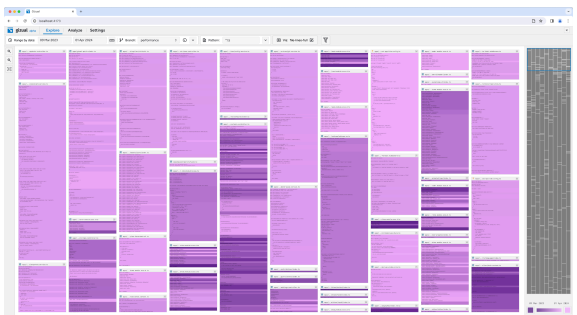
- 🕰️ Creating performant visualizations with good UX is time-consuming.
- 🙊 Feature parity in browsers is a myth.
- 📈 Profiling complex architecture is hard.



POC1



POC2



Gizual v1.0

# Technical Goals

- 📈 **Fast**: Provide results quickly and incrementally.
- 🔀 **Local**: Avoid uploading to a server.
- 📱 **Scalable**: Support large repositories:

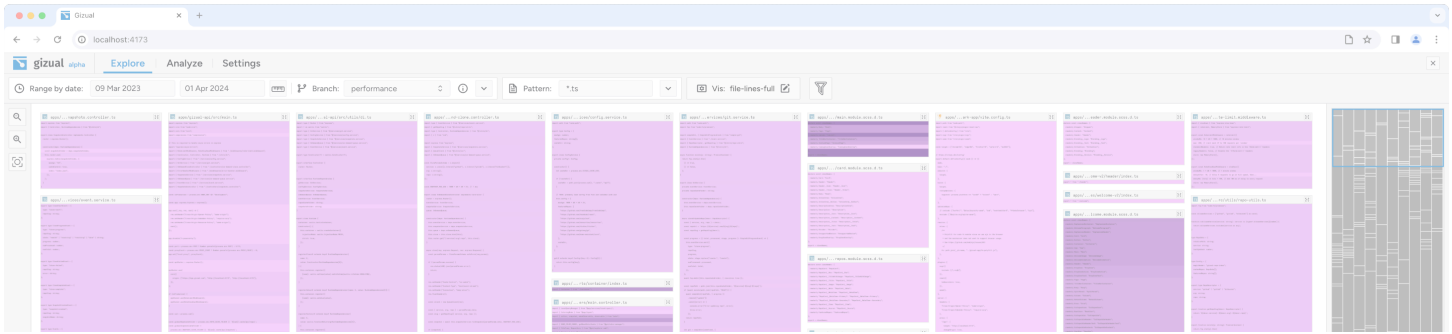
| | Commits | Size | 🖥️ | 📱 |
|---|---|---|---|---|
| vuejs/vue | 3,590 | 33 MB | ✅ | ✅ |
| facebook/react | 16,606 | 445 MB | ✅ | 🚧 |
| microsoft/vscode | 120,484 | 959 MB | ✅ | 🚧 |
| torvalds/linux[1] | 1,264,922 | 5,426 MB | 🤷‍♂️ | 🤷‍♂️ |

1. Only counting data within GitHub's mirror

🖥️ Google Chrome Version 123.0.6312.87 on MacBook Pro 2023, M3 Max, 48GB RAM.
📱 Safari on iPhone 15 Pro Max, iOS 17
✅ Supported, 🚧 Work in Progress / Frequent Crashes, 🤷‍♂️ Probably not feasible

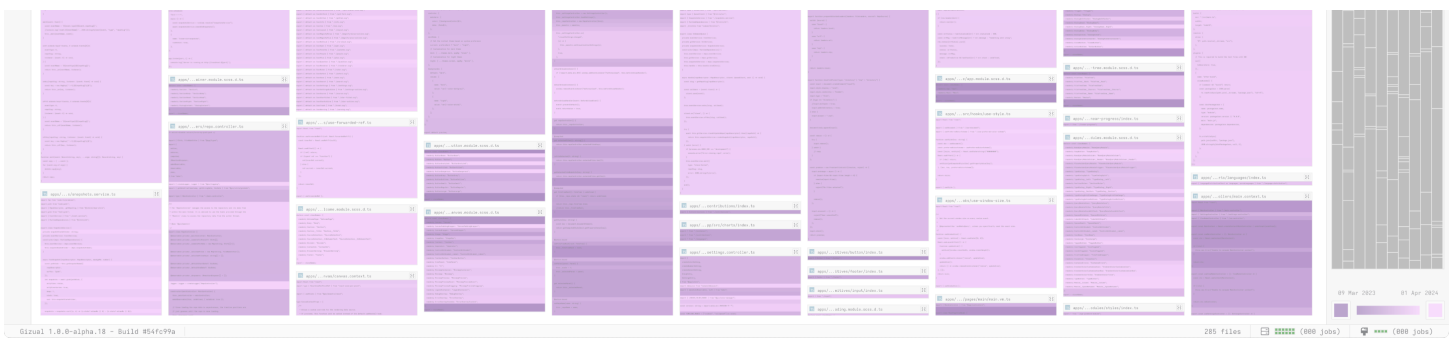
# Demo

# Showcase



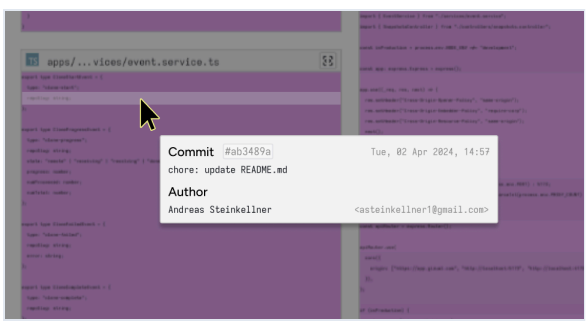

This video is accessible at:

https://gizual.com/glt24

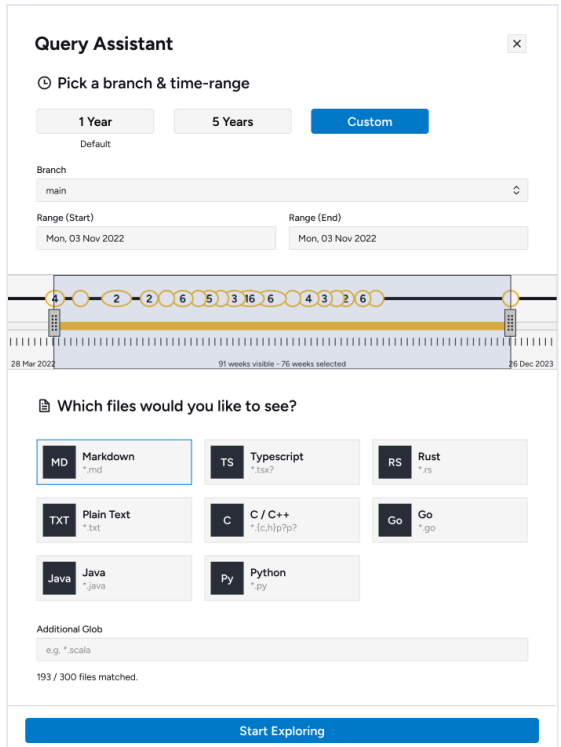


# Coming Soon ™

- More visualizations
- SVG export
- Git blame within editor view
- Query assistant
- Line-of-Code tooltips




Mockup - Line-of-Code tooltips




Mockup - Query assistant

# Upcoming Event

UI deep-dive @ CSS-in-Graz `meetup.com/css-in-graz`

# File-IO and FileSystems

## within the Browser Sandbox

# The File Input Element

- Proposed in RFC 1867 (1995)
- Standardized in HTML 3.2 (1997)
- Nested in a `<form>` element
- enctype="multipart/form-data"
- Used to upload files to a server

```
<form
  method="post"
  enctype="multipart/form-data"
  >
  <input
    type="file"
    name="my-file"
  />
  <button type="submit">
    Upload
  </button>
</form>
```

# Client First Approach

- Validate before Upload
- Offline Mode
- Progressive Web Apps (PWA)
- Reduced Server Load
- Faster User Interactions
- Less Data Transfer
- Privacy

# File API: 1. Reading a Local File

```
<label for="file-picker">Select a file:</label>
<input id="file-picker" type="file" accept=".txt,.md" />
<pre><code id="output"></code></pre>
<script>
  const picker = document.querySelector('input');
  picker.addEventListener('change', (e) => {
    for (let file of e.target.files) {
        const file = event.target.files[0];
        const text = await file.text();
        document.querySelector("#output").innerHTML = text;
    }
  });
</script>
```

Open Codepen

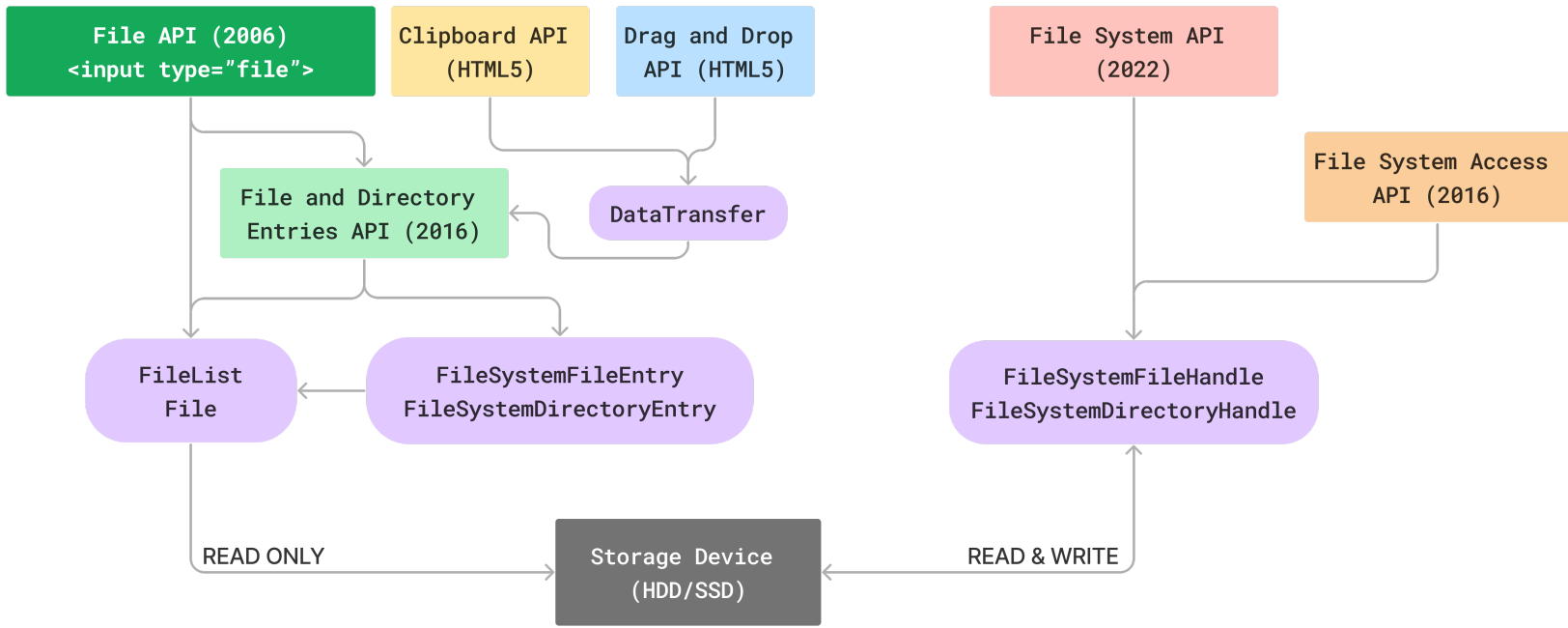
# Typical File Operations

- READ file
- WRITE / CREATE file
- MOVE file
- DELETE file
- LIST directory

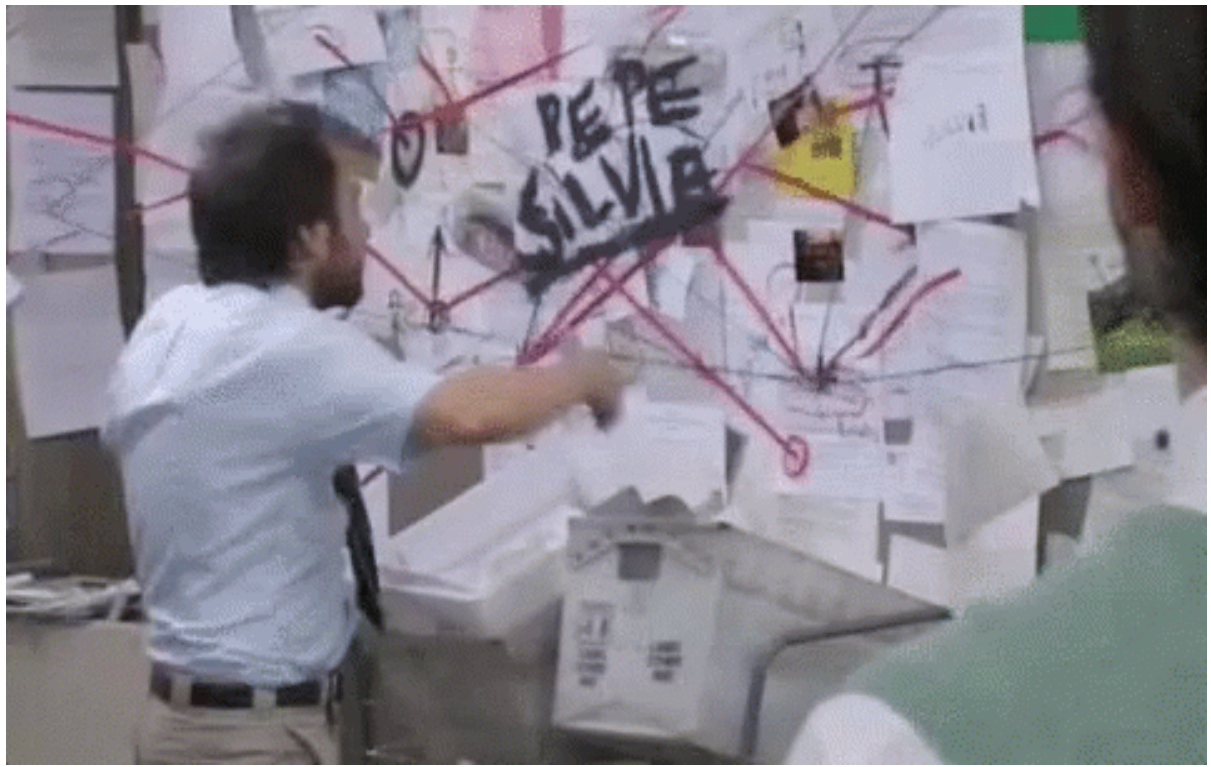
# How many Web APIs for files are there?

- File API
- Drag and Drop API
- Clipboard API
- File and Directory Entries API
- File System API
- File System Access API

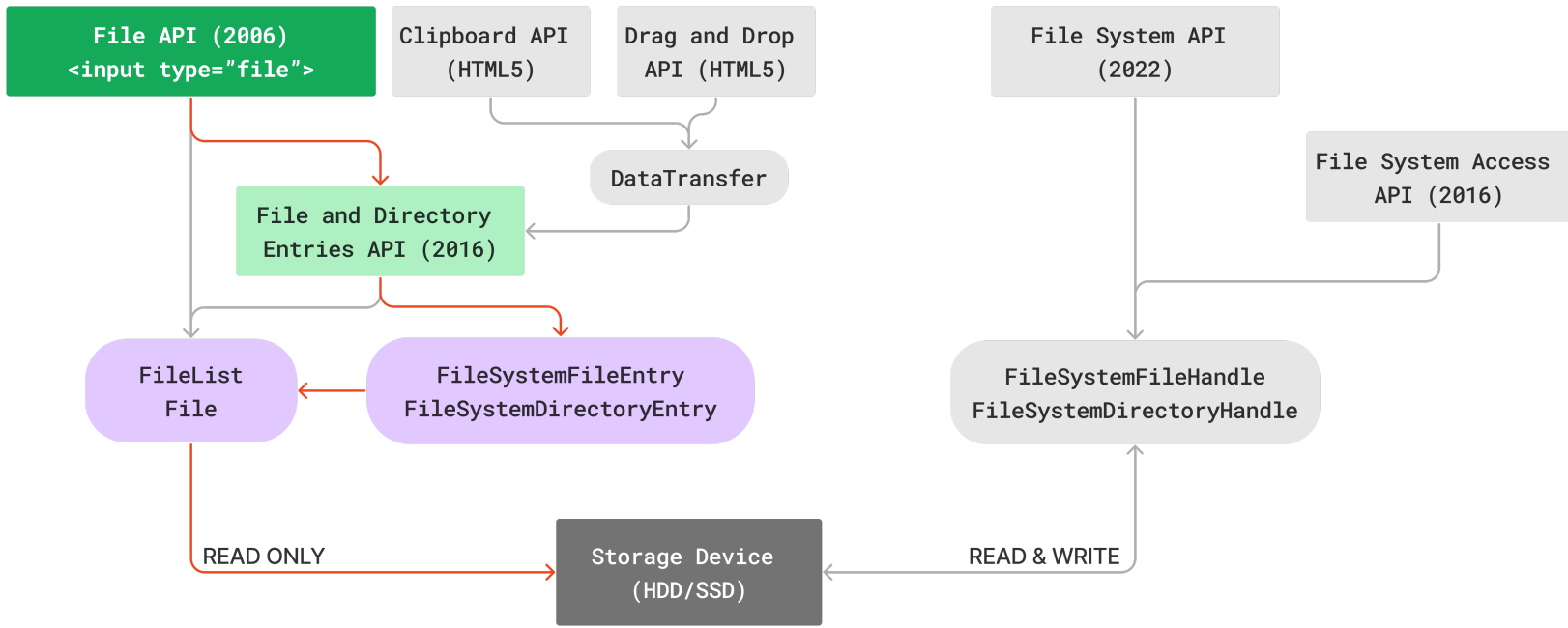
# Web APIs for Files



# It’s complicated



# File API[1]




1. File API https://developer.mozilla.org/en-US/docs/Web/API/File_API

# File API: Scanning a Local Directory (1)

```
<input type="file" webkitdirectory />
<script>
  const picker = document.querySelector('input');
  picker.addEventListener('change', (e) => {
    for (let file of picker.files) {
      console.log(file.webkitRelativePath);
      // 'repo/README.md'
      // 'repo/.git/HEAD'
      // 'repo/.git/objects/22/ee7b7ca19c25..'
      //  and many more ...
    }
  });
</script>
```

Open Codepen

# File API: 2. Scanning a Local Directory (2)

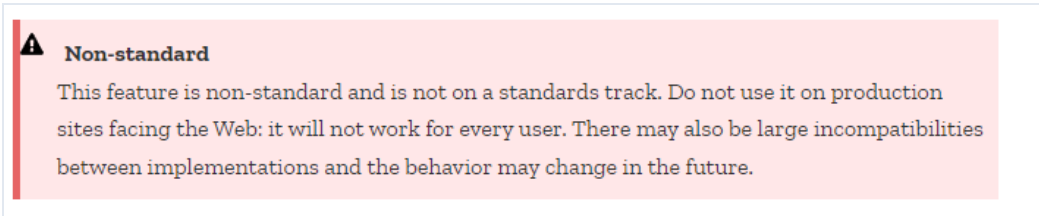



**Fig. 1:** Screenshot taken from: https://udn.realityripple.com/

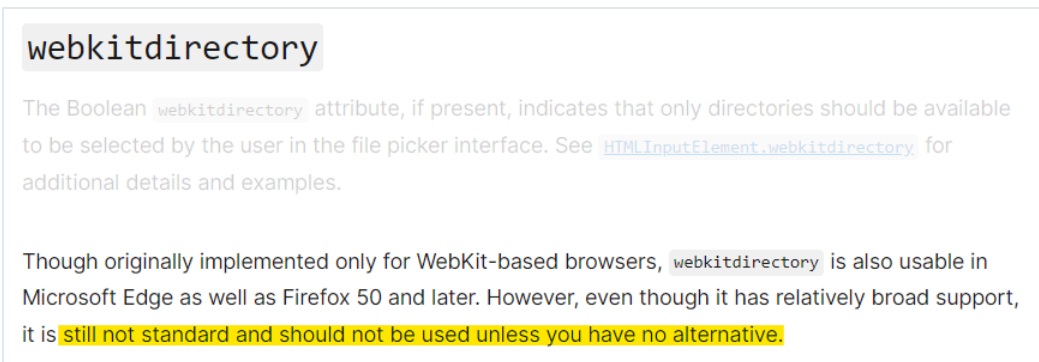
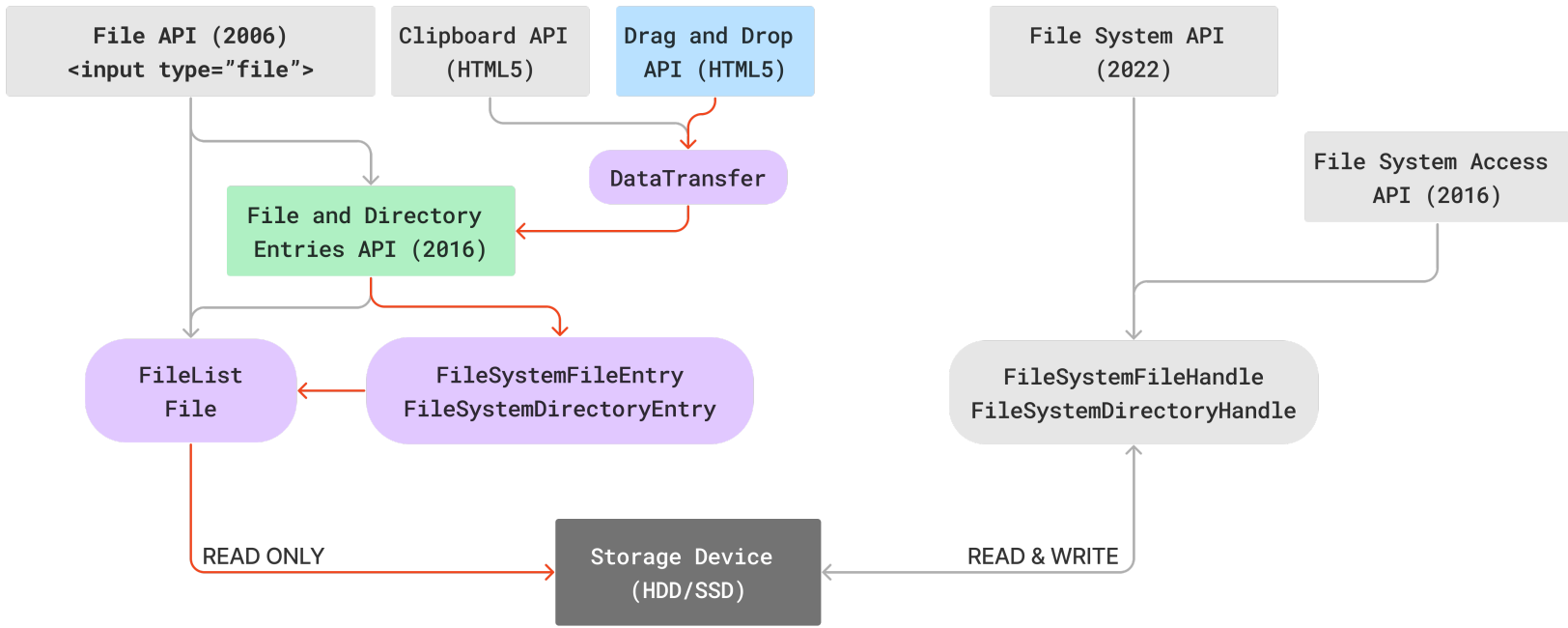



**Fig. 2:** Screenshot taken from: https://developer.mozilla.org/

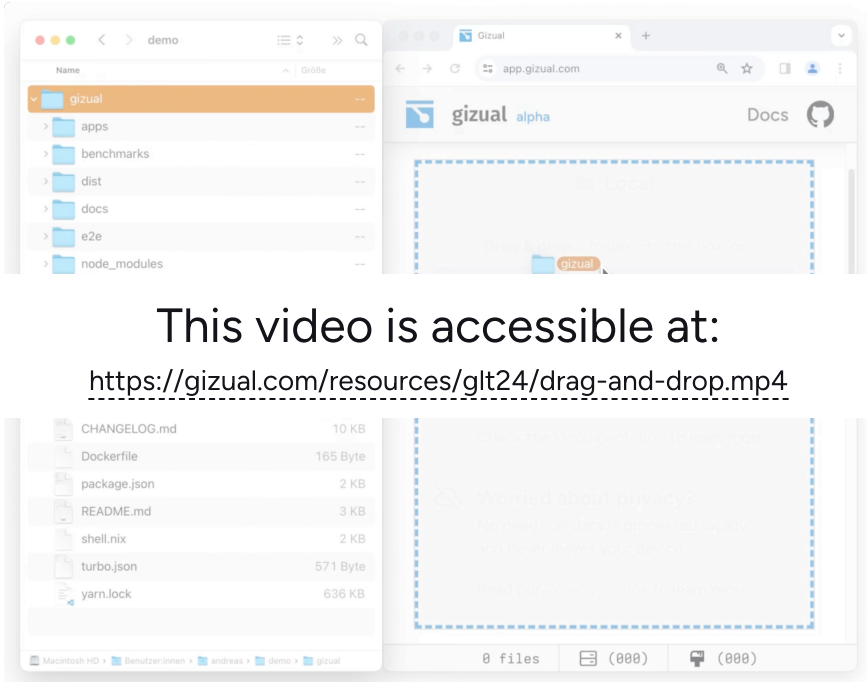
# Drag and Drop API[1]




1. Drag and Drop API https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API

# Drag and Drop API (1)

- Returns single `FileSystemDirectoryEntry` for a directory
- No additional user confirmation
- Allows to ignore paths we are not interested in
- Can not be transferred to a Web Worker



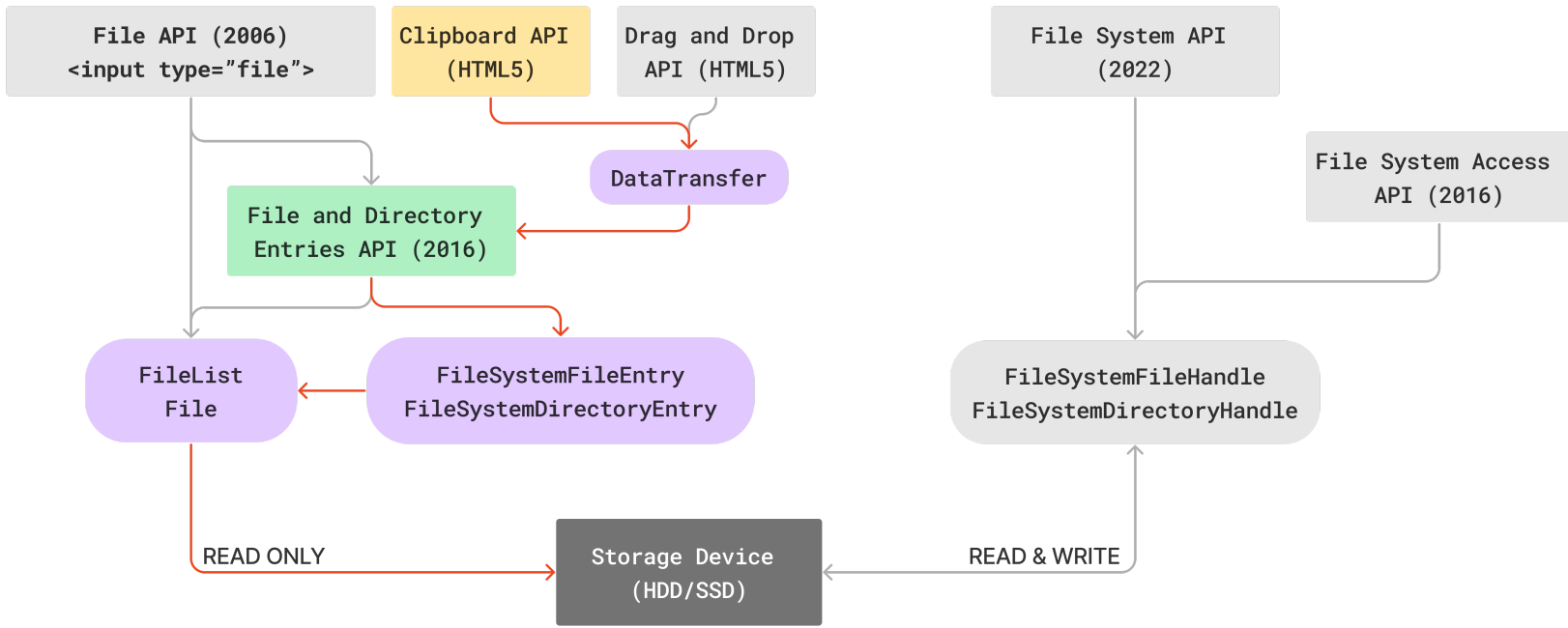

**Video 1:** Demo of drag and drop functionality

# Drag and Drop API (2)

```
const dropArea = document.querySelector("#drop-area");
dropArea.addEventListener("dragover", (event) => {
  event.preventDefault();
});
dropArea.addEventListener("drop", async (event) => {
  event.preventDefault();
  const item = event.dataTransfer.items[0].webkitGetAsEntry();
  scanFiles(item, document.querySelector("#output"));
});
```

Open Codepen

# Clipboard API[1]




1. Clipboard API https://developer.mozilla.org/en-US/docs/Web/API/Clipboard_API

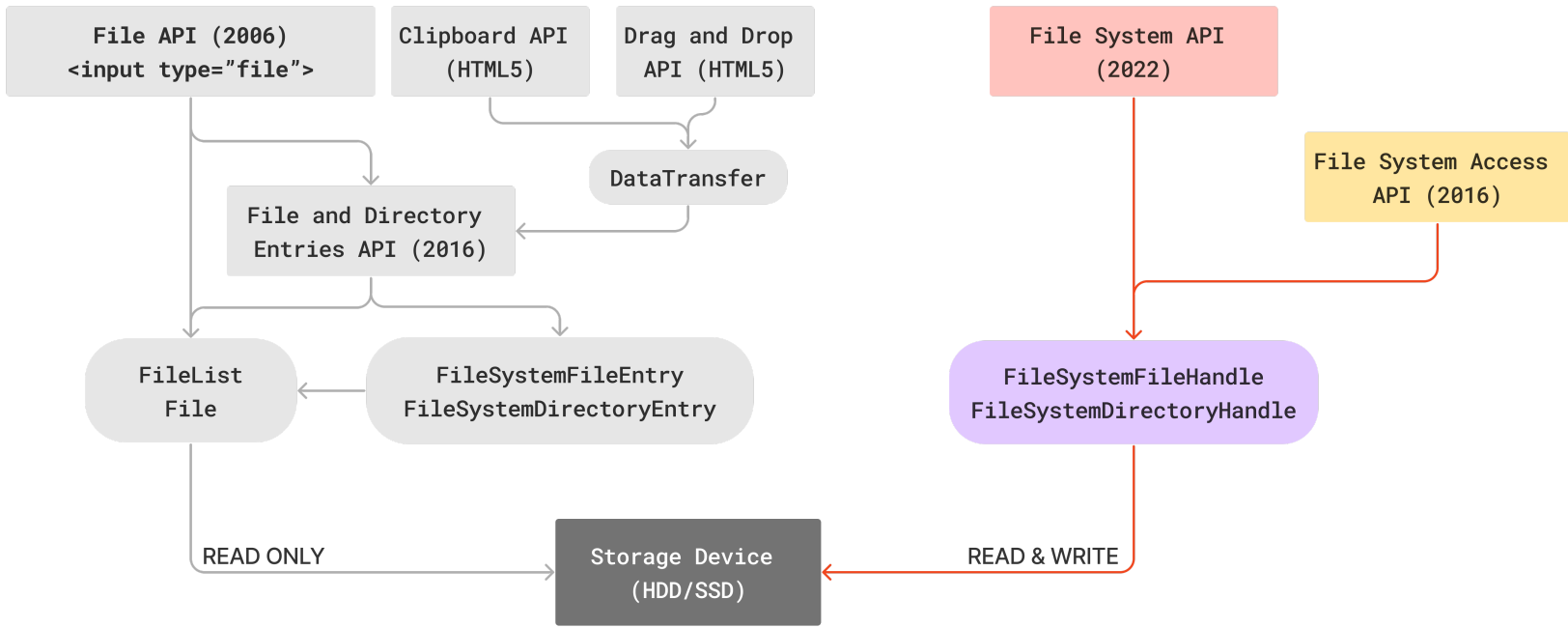
# Clipboard API

- Listen for `paste` event
- `clipboardData` property = `DataTransfer`
- Proceed as with Drag and Drop API

```
const output = document.querySelector("#output");
window.addEventListener('paste', e => {
  output.innerHTML = "";
  const item = e.clipboardData.items[0].webkitGetAsEntry();
  scanFiles(item, document.querySelector("#output"));
});
```

Open Codepen

# File System API[1]




1. File System API - https://fs.spec.whatwg.org/

# File System API

- Abstraction of a file system.
- Requires secure context.
- 2 Implementations:
  - File System Access API[1]
    - Access local files directly.
    - Supported by Chromium.
  - Origin Private File System[2]
    - Sandboxed file system.
    - Supported by Chromium, Firefox, Safari.

1. File System Access API - https://wicg.github.io/file-system-access/
1. File System Access API (2) - https://developer.chrome.com/docs/capabilities/web-apis/file-system-access
2. Origin Private File System - https://developer.mozilla.org/en-US/docs/Web/API/File_System_API/Origin_private_file_system

# Origin Private File System

```
const button = document.querySelector("#open-dir");

button.addEventListener("click", async () ⇒ {
  const opfsHandle = await navigator.storage.getDirectory();
  for await (let [name, handle] of opfsHandle) {
    if (name === "dir_to_delete") { handle.remove({recursive: true}); }
  }
  const newDir = opfsHandle.getDirectoryHandle("new_dir", {create: true});
  const fileHandle = await newDir.getFileHandle("file.txt", {create: true});
  const writable = await fileHandle.createWritable();
  await writable.write("Hello World");
  await writable.close();
});
```

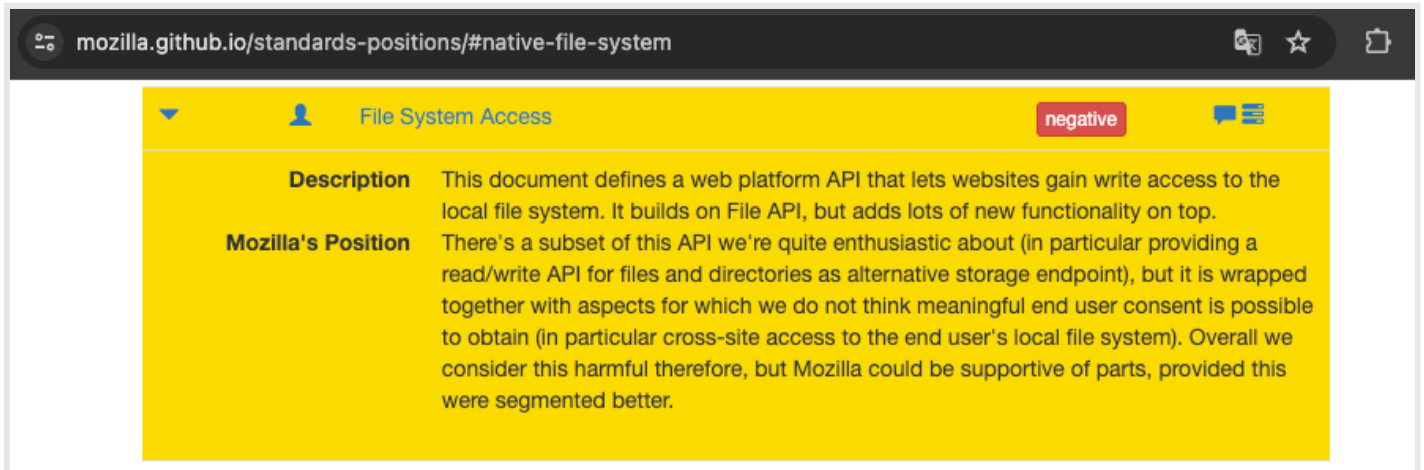
# File System Access API

```
1  const button = document.querySelector("#open-dir");
2
3  button.addEventListener("click", async () ⇒ {
4    const root = await window.showDirectoryPicker();
5
6    const newDir = await root.getDirectoryHandle("new_dir", {create: true});
7    const fileHandle = await newDir.getFileHandle("file.txt", {create: true});
8    const writable = await fileHandle.createWritable();
9    await writable.write("Hello World");
10   await writable.close();
11 });
```

Open Local Demo

# Why not just use the File System Access API?

# Why not just use the File System Access API?




Firefox - Standards Position - https://mozilla.github.io/standards-positions/#native-file-system
Firefox - Blog Post - https://hacks.mozilla.org/2012/07/why-no-filesystem-api-in-firefox/

# Gizual's Approach (Best Case)

| | Input | Storage |
|---|---|---|
| Chromium<br>Version 106+, Sep. 2022 | File System Access API | - |
| Firefox<br>Version 110+, Feb 2023 | Drag & Drop API | Origin Private File System |
| Safari<br>Version 16.4+, March 2023 | Drag & Drop API | WIP 🚧 🙄[1] |

1. Safari - Incomplete OPFS Implementation - https://bugs.webkit.org/show_bug.cgi?id=254726

# Discussion

Have you used any of these APIs before?

# Thank you!