Improve Chromium Web Audio Testing

Summary

Project Proposal link

WebAudio and W3C test harness

This project aims to modernize and streamline Web Audio testing within Chromium by replacing the old, custom audit.js testing library with the widely-adopted, standard W3C test harness, testharness.js. Back in 2016, audit.js was created to fill a gap. It provided a way to run Web Audio tests sequentially and included some handy utilities for making audio-specific checks. It's been used in hundreds of tests inside Chromium's Blink rendering engine. But, now W3C's testharness.js has matured significantly. It's now the standard way to write tests for web platform features. The goal is to carefully transition those 300+ tests from audit.js to testharness.js.

Owner: Koushik
Contributors: Koushik Kumar Bug

Approver:

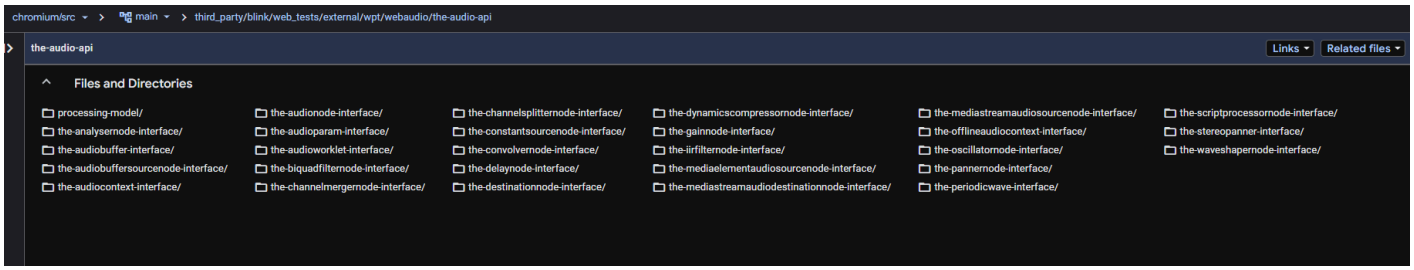
hongchan@google.com
mjwilson@google.com

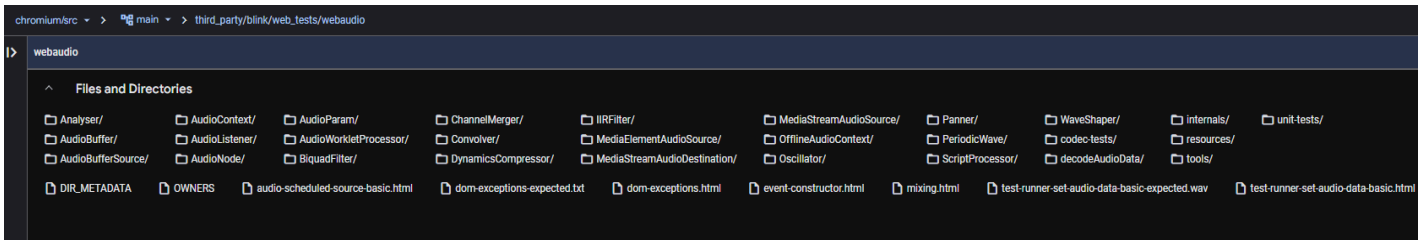
Status: Final
Created: 31-03-2025

Project Abstract

Chromium's Web Audio implementation relies heavily on thorough testing to ensure cross-browser compatibility and prevent regressions. Currently, many of these tests use a custom library called audit.js, which was created in 2016 to address specific needs at the time. However, the web platform testing landscape has evolved, and the W3C's testharness.js has become the recommended standard. This project aims to migrate existing Web Audio tests from audit.js to testharness.js.

This migration is not just a simple substitution. audit.js[1][2] provides some helpful utilities tailored to Web Audio, particularly for assertions that compare audio data. My approach will involve carefully analyzing each test, identifying the equivalent testharness.js functionalities, and, where necessary, extracting and adapting the relevant parts of audit.js into a small, reusable helper library within the testharness.js framework. This ensures that we retain the precision and expressiveness of the existing tests while aligning with modern testing best practices. The ultimate goal is to have a cleaner, more maintainable, and more standards-compliant testing infrastructure for Web Audio in Chromium.





Background

The Web Audio API is a complex and powerful part of the web platform. To ensure that Chromium's implementation of the Web Audio API works correctly and consistently across different systems, a comprehensive suite of tests is essential.
As mentioned, the existing tests, located in Chromium's blink/web_tests directory, currently use the audit.js library. testharness.js, on the other hand, is the W3C standard for web platform tests. It's widely used, actively maintained, and provides robust support for asynchronous testing, making it a natural fit for Web Audio. By migrating to testharness.js, Chromium's Web Audio tests will be more aligned with industry best practices, easier for external contributors to understand, and will benefit from ongoing improvements to the testharness.js framework itself.

testharness.js has several key advantages over custom testing solutions:

- *Standardization: It's the official W3C(World Wide Web Consortium) recommendation. Using it means that Chromium's tests are aligned with the broader web platform testing effort.*
- *Maintainability: testharness.js is actively maintained by a large community, meaning that bugs are fixed, and new features are added regularly. Chromium benefits from this ongoing work without having to dedicate its own resources to maintaining a separate testing library.*
- *Cross-Browser Compatibility: Tests written using testharness.js are more likely to be portable to other browsers. This is important for ensuring that the Web Audio API works consistently across the web.*
- *Modern Asynchronous Handling: testharness.js provides excellent support for asynchronous testing using Promises and async/await functions, which are now standard features of JavaScript. This simplifies the writing of asynchronous tests and makes them easier to understand.*
- *Extensibility: testharness.js can be built on to accommodate project specific requirements.*

Right now, Chromium's Web Audio tests are in a hybrid state. They use audit.js, a custom solution, while the rest of the web platform (and increasingly, other parts of Chromium) is moving towards testharness.js. This creates a few problems:

- *Duplication of Effort: Many of the core features of audit.js (like asynchronous test management) are already provided by testharness.js. Maintaining two systems that do similar things is inefficient.*
- *Increased Complexity: New contributors to Chromium's Web Audio code have to learn two different testing frameworks, increasing the learning curve.*
- *Potential for Divergence: As testharness.js continues to evolve, audit.js risks falling behind, potentially missing out on new features or bug fixes.*
- *CI Overhead: Running tests with audit.js adds a small but measurable overhead to Chromium's Continuous Integration (CI) system. Removing this overhead, even if it's small on a per-test basis, adds up when you consider the thousands of tests that are run every day.*

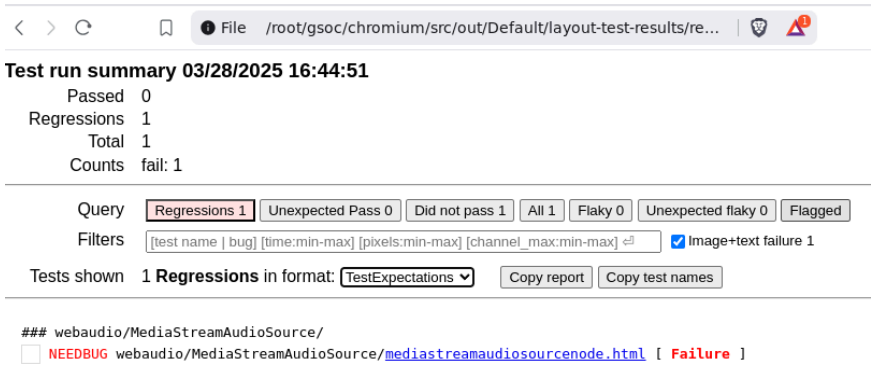
This project's goal is to transition Chromium's Web Audio tests from the older audit.js library to the modern, standardized testharness.js. This will involve:

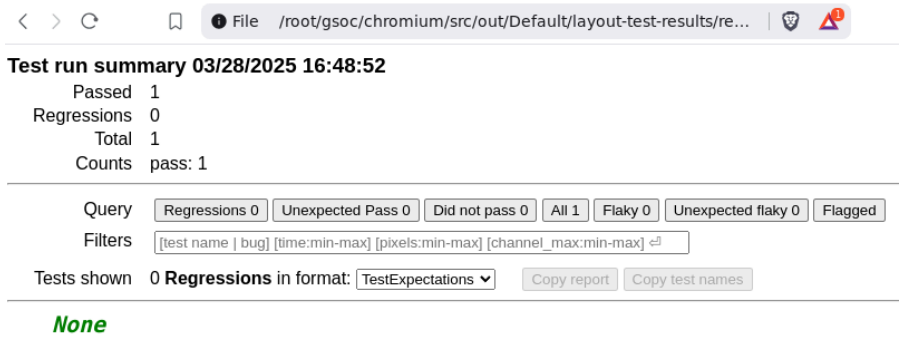
- *Careful Analysis: Each of the existing audit.js tests will be examined to understand how it works and what it's testing.*
- *testharness.js Equivalents: For each part of the audit.js test, the equivalent testharness.js functionality will be identified. This will often involve replacing audit.js's should() assertions with standard testharness.js assertions like assert_equals, assert_true, assert_throws_js, etc. Asynchronous tests will be rewritten using async_test and promise_test.*
- *Specialized Assertion Helper: For the audio-specific assertions in audit.js (like beCloseToArray), a small, focused helper library will be created. This library will not be a separate testing framework. Instead, it will be a set of utility functions that extend testharness.js, providing the specialized assertions needed for Web Audio testing within the standard framework. This approach keeps the testing infrastructure clean and consistent.*

```
// Return a string suitable for printing one failed element in
// |beCloseToArray|.
function _formatFailureEntry(index, actual, expected, abserr, threshold) {
  return '\t[' + index + ']\t' + actual.toExponential(16) + '\t' +
      expected.toExponential(16) + '\t' + abserr.toExponential(16) + '\t' +
      (abserr / Math.abs(expected)).toExponential(16) + '\t' +
      threshold.toExponential(16);
}

// Compute the error threshold criterion for |beCloseToArray|
function _closeToThreshold(abserr, relerr, expected) {
  return Math.max(abserr, relerr * Math.abs(expected));
}
```

- *Full Verification: After each test is rewritten, it will be carefully checked to ensure that it produces the same results as the original audit.js version. This is critical to prevent regressions.*
    - *How I expect to do this:*
        - *My main tool for this verification is Chromium's own test runner, run_web_tests.py.*
        - *I will run the original test using 'third_party/blink/tools/run_web_tests.py --build-directory=out/Default third_party/blink/web_tests/webaudio/MediaStreamAudioSource/mediastreamaudiosourcenode.html'*
        - *I will note its current state : pass ? fail ? or produce specific results.*
        - *Then, I'll carefully rewrite the test using testharness.js. The focus here is purely on replicating the original test's logic and checks using the new framework.*
        - *Finally, I will run the migrated test.*
        - *Below are the screenshot reports of 'mediastreamaudiosourcenode.html'. First when errors are encountered, second when errors have been fixed.*



- 

- *Gradual Rollout: The migration will happen in stages, starting with simpler tests and gradually moving to more complex ones. This allows for early detection of any problems and minimizes the risk of disrupting the overall testing process.*
    - *According to my investigations and understanding:*
        - *Simpler Test: Tests primarily using basic assertions like should().beEqualTo(), should().beTrue(), should().beFalse(), or should().throw() are generally simpler.*
        - *Moderate Test: Tests that use asynchronous operations using context.startRendering() or tests that depend heavily on utilities from audit-util.js. Also tests involving multiple sequential or nested asynchronous operations*
        - *Complex Test: Tests using the specialized audio assertions like beCloseToArray(), notGlitch(), beConstantValueOf() or containValues(). These will require using the new assertions helper library. Also Tests that rely on external resources loaded via Audit.loadFileFromUrl() add a bit of complexity.*

By the end of this project, Chromium's Web Audio tests will be fully integrated with the standard testharness.js framework, making them more maintainable, more consistent with the rest of the web platform, and easier for developers to work with. This represents a significant improvement to the long-term health and stability of Chromium's Web Audio implementation. It also aligns Chromium's testing practices with the broader web development community,

promoting interoperability and collaboration. The removal of a custom, less maintained solution streamlines the codebase and allows developers to focus on a single, well-documented testing framework.

Chromium Core Principles

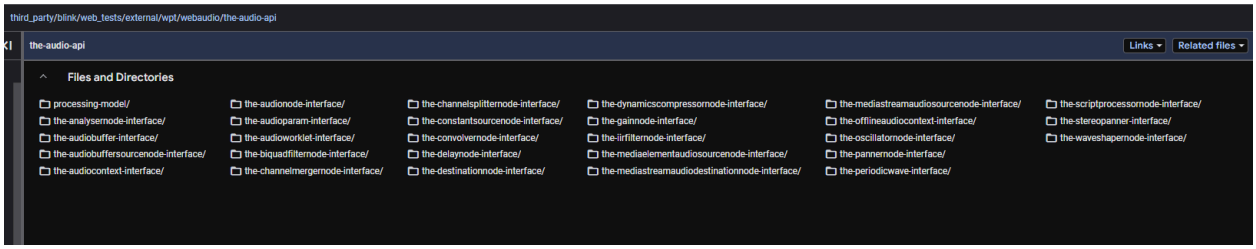
Fast, seamless, safe, and reliable testing experiences:

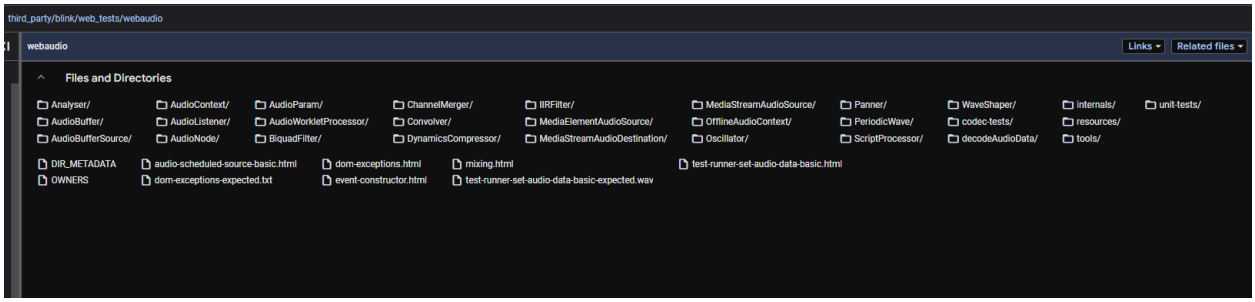
- *Speed: Faster web test runs for a smoother workflow.*
- *Simplicity: An easy-to-use testing framework.*
- *Security: Built on standards for safer tests.*
- *Stability: Reliable and maintainable tests.*

Design Ideas and Implementation

The core of this project is to translate audit.js based tests into their testharness.js equivalents. This will involve:

- *Test File Analysis: I will start by carefully reviewing each of the 300+ test files that currently use audit.js. I'll categorize them based on the audit.js features they use (e.g., specific assertion types, asynchronous task management).*
    - *Which audit.js test uses features like audit.define, task.done, should.beEqualTo, should.beCloseToArray, Audit.loadFileFromUrl etc.*
    - *The structure of the test (synchronous, asynchronous via callbacks, asynchronous via promises).*
    - *Test dependencies on audit-util.js.*





- *testharness.js Mapping: For each category, I'll identify the corresponding testharness.js features. For example:*
    - *audit.js's audit.define and task.done will typically be replaced with testharness.js's async_test and test.done.*
    - *audit.js's should().beEqualTo() will become assert_equals().*
    - *audit.js's should().throw() will become assert_throws_js() or assert_throws_dom().*
- *Replacing the test structure: audit.js uses a custom Task and TaskRunner system to manage the execution of tests, which are often asynchronous due to the nature of Web Audio processing. testharness.js provides built-in mechanisms for handling both synchronous and asynchronous tests, eliminating the need for the audit.js task management.*
  The core change here is replacing audit.define() and audit.run() with the appropriate testharness.js functions: test(), async_test(), and promise_test().
    - *Original synchronous audit.js test:*

```
const audit = Audit.createTaskRunner();
audit.define('Simple Test', (task, should) => {
  const context = new AudioContext();
  const oscillator = context.createOscillator();
  should(oscillator.type, 'Oscillator type').beEqualTo('sine');
  task.done();
});
audit.run();
```

Equivalent testharness.js test:

```
test(() => {
  const context = new AudioContext();
  const oscillator = context.createOscillator();
  assert_equals(oscillator.type, 'sine', 'Oscillator type');
}, 'Simple Test');
```

- *Original Asynchronous audit.js test:*

```
    mediastreamaudiosourcenode.html
  </title>
  <script src="../../resources/testharness.js"></script>
  <script src="../../resources/testharnessreport.js"></script>
  <script src="../resources/audit-util.js"></script>
  <script src="../resources/audit.js"></script>
</head>
<body>
  <script id="layout-test-code">
    let audit = Audit.createTaskRunner();

    audit.define(
        {
          label: 'test',
          description: 'Basic tests for MediaStreamAudioSourceNode API'
        },
        (task, should) => {
          should(
              () => {navigator.webkitGetUserMedia(
                  {audio: true},
                  (stream) => {
                    gotStream(stream, should);
                    task.done();
                  },
                  () => {
                    should(false, 'Stream generation')
                        .message('succeeded', 'failed');
                    task.done();
                  })},
              'getUserMedia()')
              .notThrow();
        });

    audit.run();
```

Functionally Equivalent testharness.js test:

```
  <title>MediaStreamAudioSourceNode: Test For API Functionality</title>
  <script src="../../resources/testharness.js"></script>
  <script src="../../resources/testharnessreport.js"></script>
</head>
<body>
  <script>
    'use strict';

    async function testMediaStreamAudioSourceNode() {
      let stream;
      try {
        stream =
            await navigator.mediaDevices.getUserMedia({ audio: true });
      } catch (error) {
        throw new TestAbortedError(
            `getUserMedia failed: ${error.name} - ${error.message}. ` +
            'Test did not run as expected'
        );
      }

      const context = new AudioContext();
      const mediaStreamSource =
          new MediaStreamAudioSourceNode(context, { mediaStream: stream });
```

- *Creating the assertions helper library: testharness.js provides a comprehensive set of built-in assertions (e.g., assert_true, assert_false, assert_equals, assert_throws_js, assert_throws_dom) which cover many of the cases used in audit.js. However, audit.js includes several specialized assertions tailored for Web Audio testing:*
  - *beCloseTo(expected, options): Checks if a value is close to an expected value within a given tolerance. Essential for floating-point comparisons in audio.*
  - *beCloseToArray(expected, options): Checks if all elements in an array are close to the corresponding elements in an expected array, within a tolerance. Used extensively for comparing audio buffer data.*
  - *notGlitch(threshold): Checks for sudden, large changes in an array of values (representing audio glitches).*
  - *beConstantValueOf(value): Checks if all elements in an array are equal to a specific value.*
  - *containValues(values): Checks that an array contains a specific sequence of values, though not necessarily consecutively.*

  Naming Convention:

  Use 'assert_' prefixes consistent with testharness.js, e.g: assert_array_approx_equals, assert_no_glitch etc.

  testharness.js Integration:

  Rewrite the extracted logic that uses standard testharness.js assertions (assert_true, assert_equals etc.) internally to report success or failure.

  Clear API & Documentation:

  Define clear function signatures and add JSDoc comments explaining parameters, purpose, and usage, similar to testharness.js itself.

  These audio-specific assertions do not have direct functional equivalents in the core testharness.js API. Therefore, I will create a helper library to provide this functionality. This library will be written in plain JavaScript and integrated with testharness.js.

- *Challenges to Tackle:*
  - *Accurate Equivalence: The biggest challenge will be ensuring that the rewritten testharness.js tests are exactly equivalent to the original audit.js tests. This requires careful attention to detail, especially when dealing with floating-point comparisons and asynchronous behaviour.*
  - *Assertion Helper Library: Creating a robust and well-tested assertion helper library for the audio-specific checks will be important.*
  - *Edge Cases: Some tests might rely on subtle or undocumented behaviour of audit.js. Identifying and handling these edge cases will require thorough investigation.*

- *Test Coverage: Ensuring test coverage is not reduced due to the migration.*

By addressing these design considerations, I can ensure a smoother and more effective migration process. The key is to combine a solid understanding of both audit.js and testharness.js with a proactive approach to identify and resolve potential issues.

Code Affected

- *third_party/blink/web_tests/webaudio/*
- *third_party/blink/web_tests/external/wpt/webaudio/the-audio-api/*

Alternatives Considered (Optional)

- *As of now, there is no automation for formatting html files in accordance with Google JS Style Guide. Infact, I faced lots of challenges in trying to understand how to write clean code. So, I plan to create an automation just like git cl format to help fellow contributors.*

Pre proposal Work

- *Contributions to Chromium:*
  - https://chromium-review.googlesource.com/c/chromium/src/+/6345010 *(GSOC Starter bug), (Merged)*
    Skills involved: HTML, JavaScript, Understanding of audit.js and testharness.js
    Project Briefing: Migrate webaudio/MediaStreamAudioSource to testharness.js
    - *The following test files have been deprecated from audit.js to testharness.js:*
      - *added testInvalidConstructor_TH() in audionodeoptions.js*
      - *renamed ctor-mediastreamaudiosourcenode.html to constructor.html*
      - *Mediastreamaudiosourcenode.html*

- https://chromium-review.googlesource.com/c/chromium/src/+/6462272
  (GSOC Starter bug), (Merged)
  Skills involved: HTML, JavaScript, Understanding of audit.js and testharness.js
  Project Briefing: Migrate webaudio/MediaStreamAudioSource to testharness.js
    - *The following test files have been deprecated from audit.js to testharness.js:*
      - *webaudio/Analyser/automatic-pull-node.html*

- https://chromium-review.googlesource.com/c/chromium/src/+/6345350
  (GSOC Starter Bug), (Work In Progress)
  Skills involved: Understanding of web audio libraries (audit.js and testharness.js) and HTML, JavaScript
  Project Briefing: Migrate webaudio/MediaStreamAudioDestination to testharness.js.
    - *The following test files have been deprecated from audit.js to testharness.js:*
      - *mediastreamaudiodestinationnode.html*

- https://chromium-review.googlesource.com/c/chromium/src/+/6345251
  (GSOC Starter Bug), (Work In Progress)
  Skills involved: Understanding of web audio libraries (audit.js and testharness.js) and HTML, JavaScript.
  Project Briefing: Migrate webaudio/Convolver to testharness.js.
    - *The following test files have been deprecated from audit.js to testharness.js:*
      - *unmodified-buffer.html*

- *I've built chromium on my local system (Intel 64 Linux) by reading documentation, interacting with mentors, and asking doubts in the Chromium-dev group. I have also read about mojo in C++ and learnt related documents to it. Mostly in my pre proposal period, I took guidance from my mentors, who helped me to navigate across the codebase and make contributions according to the style guide.*

Schedule of Deliverables (timeline)

This is a rough estimate of the things I am planning to do. I want to make it more detailed in my Community Bonding Period, by discussing it with my project mentors.

May 8 – June 1 (Community Bonding Period)

- *Community Bonding Period In the first one or two weeks, I want to do more research on the codebase and the code of conduct, strengthen the relationship with my mentors by getting to know what they expect, and have a better understanding of what we are going to accomplish so that we don't run into any problems later. I also want to talk about my alternative ideas and come up with a final design for this project. and after that, I want to make the most of the time I have left by starting to code in order to produce the intended outcome as accurately and efficiently as I can.*
- *I will thoroughly analyze the audit.js code and the existing Web Audio tests. Identify all the different assertion types and asynchronous patterns used.*
- *Create a detailed design document for the assertion helper library, specifying the functions it will provide and how they will map to the existing audit.js assertions.*

June 2 - July 14 (Phase I)

- *Week 1-2:*
  - *Implement the core of the assertion helper library (functions for beCloseToArray, notGlitch, and any other essential audio-specific functions or things).*
  - *Migrate a small set of "easy" tests (tests that use simple assertions and minimal asynchronous logic) to testharness.js and the new helper library.*
  - *Documentation on previous tasks which include analyzing audit.js and the existing Web Audio tests, identifying assertion patterns, and outlining the assertion helper library design.*
- *Week 3-4:*
  - *Migrate a larger set of tests, focusing on those that involve more complex asynchronous operations (using promise_test, step_timeout, etc.). Refine the helper library as needed.*
- *Week 5-6:*
  - *Handle those tests that use more obscure or potentially problematic features of audit.js. This is where I might need to ask the most questions to my mentors.*
  - *Begin addressing any edge cases or differences in behavior discovered during the migration.*

July 14 - August 25 (Phase II)

- *Week 7-8:*
  - *Continue migrating tests, aiming to complete the majority of the migration.*
  - *Focus on tests that involve complex audio processing graphs or interactions with other Web APIs.*
  - *Documentation done on the Phase I (Weeks 3–6) and Phase II (Week 7–8) tasks, which include migrating a larger set of tests, refining the helper library,*

addressing edge cases, and continuing the migration with complex audio processing graphs or interactions with other Web APIs.

- *Week 9-10:*
  - *Address any remaining edge cases or problematic tests.*
  - *Thoroughly review all migrated tests to ensure consistency and correctness.*
- *Week 11-12:*
  - *Complete final documentation by adding or cleaning up features as required.*
  - *Perform final testing and bug fixing.*
  - *Optimize performance.*
  - *Prepare a final report and presentation summarizing the project.*

August 25 - November 9 (For Extended Timelines)

- *Implement JS Style Guide auto formatting (optional).*
- *Explore additional improvements or optimizations to the Web Audio testing infrastructure, if time allows. This could include investigating ways to improve test performance or adding new tests for uncovered areas.*

Communications

I am flexible with any means of communication, and I have no problem adjusting to my mentor's time zone. In the summer, I have no obligations as such because of the summer vacation, and I am going to commit full time to this project and am ready to devote 40-50 hours per week (approximately 7-8 hours on weekdays and 5-6 hours on weekends). And in case If any delay occurs, I would first assess the situation and determine the reason for the delay. Then, I would communicate with the mentor to discuss the situation and see if any adjustments needed to be made to the timeline or deliverables.

- *Timezone: GMT +5:30 (Indian Standard Time)*
- *Working Hours: Flexible*
- *Email:* *Koushik*
- *Alternative email:* *mca40057.22@bitmesra.ac.in*
- *Phone: +91-7047274427*
- *Slack: koushikbug123@gmail.com*

If something else is needed, I will join it with pleasure.

About Me

I'm Koushik Kumar Bug, a recent graduate from BIT Mesra with experience in web development and cloud technologies. Currently, I am enrolled in a course certifications through online classes. My programming expertise includes C++, Solidity, JavaScript, TypeScript, and HTML/CSS, which are relevant to this project of chromium.Previously, I served as an intern as ML Ops in Amazon and prior that, also as an intern in PrepInsta Technologies Pvt.Ltd.
My journey with browsers began during my university days when I was fascinated by how web browsers interpret and render web content. Like, we all know that using CSS, we can provide colours in our websites. But , I was eager to know, how does the browser know that, this specific color is red? Why cannot I say, GREEN is RED or RED is GREEN? Seems dumb and funny! Right? But made very much sense to me at that time. Then I read the Google Chrome Comic Book .This interest led me to explore browser internals and eventually contribute to the Chromium project. I've made several contributions to the codebase, particularly in the 'web_tests' components and tab-related functionality.
During my internship at PrepInsta, I worked on web development projects that gave me a solid foundation in frontend technologies. Later, at Amazon Development Centre India (ADCI), I gained experience in ML ops, which helped me understand large-scale systems and their development processes.
I'm drawn to this project because it provides me an opportunity to improve the developer experience within a complex system. Migrating these tests to testharness.js, and especially crafting a well-designed helper library for audio-specific assertions, is a chance to directly address those large data buffers challenges, making the process of writing and maintaining robust Web Audio tests significantly easier and more efficient for all Chromium contributors. I'm eager about the opportunity to contribute to this aspect of Chromium's ecosystem.

Prior Experience with open source

- *GitHub Id:* https://github.com/KoushikBaagh

- *Contributions to Chromium:*
  - https://chromium-review.googlesource.com/c/chromium/src/+/6345010 (*GSOC starter bug)*
  - https://chromium-review.googlesource.com/c/chromium/src/+/6361177 (*GSOC Starter bug)*

- Other Contributions:
  - https://github.com/wootzapp/wootz-browser/pull/108

(Under Review)
Project Briefing: As, I was very much fascinated by how chromium works, I started figuring out chromium codebase. But as it was very large and complicated for a first-time contributor, I thought it was not my cup of tea. Luckily, I stumbled upon this wootzapp browser. I found it interesting as they were building a web3 browser on top of chromium. So, I reached out the CEO, providing him some of my proof of work and requesting him, if I can be of any use. He then, asked me to study brave browser's codebase and chromium codebase and understand the overall difference in their system design. It was a long one-month procedure. After that, he asked me to create a help-page WebUI and also told me to first understand the difference between web-page and WebUI. I then, started going through all the docs of chromium , especially "Creating WebUI Interfaces" and "WebUI BUILD.gn" . I then asked him, if I am on the right track? and he said yes. Finally I followed the guide and created a simple help-page UI.

I'm a beginner to open source; this is my first time participating in such a big open source program, and I hope this is a great kick start for me to learn and contribute to projects that will impact so many users for the better.

My Projects

- *Created a React-based frontend website for my residential society*
  - *GitHub Link: https://github.com/KoushikBaagh/Shilpakanan-Frontend*
  - *Deployed Link: https://shilpakanan.netlify.app/*

- *'Konversify' A real-time communicative platform for BIT Alumni's*
  - *Developed a communicative platform for BIT Alumni, integrating features such as asynchronous request handling, real-time texting, file sharing, video calling, and screen sharing. Scaled the support to over 70 concurrent users while maintaining latency below 50 milliseconds thereby achieving a 40% increase in alumni engagement and a 25% increase in networking efficiency. Built the platform using a comprehensive technology stack, including REACT, JavaScript, Node.js, Express.js, multer, Socket.IO, MongoDB, RESTful API, JWT, and Bcrypt.js.*
  - *GitHub Link: https://github.com/KoushikBaagh/konversify-chatApp*
  - *Deployed Link: https://konversify-x-bit-mesra.onrender.com/*

Why Chromium?

I have chosen the Chromium organization for Google Summer of Code because it is a well-established organization with a large community of developers working on a wide range of projects related to web technologies. As an open-source enthusiast, I am passionate about contributing to projects that have a significant impact on the web and the way we use it. Chromium is a project that fits this description perfectly, with its focus on creating an open-source browser that is fast, secure, and reliable.

Additionally, I am impressed by the Chromium project's commitment to community involvement and the support it provides to new contributors. Through my research, I have found that the organization has a vibrant community of developers, which makes it an ideal place for me to learn and grow as a developer. Overall, I believe that participating in the Google Summer of Code program with the Chromium organization will provide me with a unique opportunity to work with a talented group of developers and contribute to an important open-source project that can impact a large number of users for the better.

Feedback from Chromium

If you read this document, please provide your short general feedback in the section below. Please also feel free to make comments above.

Username	Date	Comment
Michael Wil...	2025-03-26	Thank you for your detailed proposal. I added some comments, and overall I think it's looking good.
Michael Wil...	2025-03-31	The current draft looks good to me, thank you!

I am 100% dedicated to Chromium and have no plans whatsoever to submit a proposal to any other organization.