



INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X

Impact Factor: 6.078

(Volume 12, Issue 5 - V12I5-1150)

Available online at: <https://www.ijariit.com>

A Zero-Infrastructure, Browser-Based Architecture for Automated Video Interview Recording Using Client-Side Media Capture and Serverless Cloud-Storage Integration

Aditya Kumar

aditron.kumar@gmail.com

Meridean Overseas Education Consultants Pvt Ltd, Jaipur, Rajasthan

ABSTRACT

Automated asynchronous video interviewing has become a common early-stage recruitment tool, but existing solutions are almost exclusively delivered as paid, proprietary software-as-a-service platforms, creating a cost and integration barrier for individual recruiters, small organizations, and academic institutions. This paper presents the design and implementation of a video interview recording system built entirely from freely available, standards-based browser APIs and a serverless scripting platform, requiring no paid infrastructure, no traditional backend server, and no database. The client captures a continuous audio-video recording across a sequence of timed, operator-defined questions using the W3C MediaStream Recording API, and transmits the result to a serverless integration layer via a custom chunked-upload protocol designed to remove the request-payload ceilings typical of serverless platforms. The integration layer, implemented in Google Apps Script, executes under the operator's own cloud identity, allowing candidates to submit recordings without a Google sign-in of their own while ensuring recordings are persisted directly to the operator's Google Drive. We describe the system architecture, the chunked-transport protocol and its rationale, the access-control model, and a set of controlled technical micro-benchmarks characterizing encoding overhead and payload scaling. We further report on an iterative, real-world debugging process that surfaced two non-obvious browser/platform interoperability failure modes, both resolved and documented here to inform future implementers. We conclude by identifying the system's limitations — principally the absence of a controlled multi-participant usability study — and outline the empirical evaluation required to substantiate broader claims of practical utility.

Keywords: Asynchronous Video Interviewing, Browser-Based Recording, Serverless Architecture, Chunked File Upload, Google Apps Script.

1. INTRODUCTION

The adoption of asynchronous, one-way video interviewing — in which a candidate records responses to a fixed set of questions without a live interviewer present — has grown substantially as organizations seek to screen larger applicant pools with fewer scheduling constraints. Commercial platforms offering this capability, such as HireVue, Spark Hire, and VidCruiter, are typically delivered as subscription software-as-a-service (SaaS) products, often bundled with additional analytics, applicant-tracking integration, and enterprise support. While well suited to large-scale enterprise hiring, this delivery model imposes a recurring financial and administrative cost that is disproportionate for individual recruiters, small businesses, non-profit organizations, and academic settings conducting low-volume or occasional structured interviews.

This paper addresses the following question: can a functionally equivalent core capability — candidate-facing, timed, multi-question video recording with reliable centralized storage — be constructed entirely from freely available web platform primitives and no-cost cloud tooling, while remaining deployable by an operator with no software engineering background? We answer this

affirmatively and describe a complete, reproducible system that we have implemented, deployed, and iteratively debugged against real browser and platform behavior.

1.1 Contributions

- A fully client-side capture pipeline that requires no native application, browser extension, or plugin, built on standard W3C media-capture APIs.
- A serverless integration pattern using Google Apps Script that eliminates the need for a traditional backend server, database, or paid cloud account, while still authenticating uploads via a shared-secret mechanism.
- A chunked upload protocol that removes the practical request-payload ceiling that would otherwise bound interview length and recording quality on serverless platforms.
- A deployment methodology validated for use by operators without programming experience, along with documentation of two non-obvious interoperability failure modes discovered during real-world testing and their resolutions.
- A set of controlled technical micro-benchmarks characterizing the system's encoding and transport overhead, offered as a foundation for future controlled usability evaluation.

2. RELATED WORK

Table 1 summarizes the landscape this system is positioned against. Commercial video-interview and applicant-tracking platforms provide mature, feature-rich experiences but are architected around recurring revenue models that presuppose an organizational budget line for recruitment software. Traditional serverless compute platforms (AWS Lambda, Firebase Cloud Functions, Azure Functions) offer a technically comparable "no server to manage" proposition, but still require the operator to create and configure a billing-enabled cloud account, provision IAM roles or service credentials, and, in most cases, write and deploy infrastructure-as-code — a materially higher floor of technical literacy than the system described here.

Table -1: Positioning relative to existing categories of solution.

Category	Representative Examples	Typical Constraint for Small Adopters
Commercial video-interview platforms	HireVue, Spark Hire, VidCruiter	Recurring subscription cost, vendor lock-in, onboarding overhead unsuitable for one-off or low-volume hiring
General-purpose Applicant Tracking Systems (ATS)	Workable, BambooHR, Greenhouse	Video interviewing is usually a paid add-on module, not a core free feature
Traditional serverless backends	AWS Lambda, Firebase Cloud Functions, Azure Functions	Require billing account setup, IAM/service-account configuration, and baseline cloud-engineering literacy
Browser media-capture standards	W3C Media Capture and Streams; MediaStream Recording API	Provide only the capture primitive — no guidance on delivery, storage, or non-technical deployment

At the platform-primitive level, this work builds directly on two W3C specifications: the Media Capture and Streams specification, which defines `getUserMedia()` and the underlying `MediaStream` abstraction, and the `MediaStream Recording API`, which defines the `MediaRecorder` interface used to encode a live stream into a container format (WebM, in the implementation described here) without native code. These specifications provide the capture primitive but are silent on delivery, storage, or deployment concerns — the gap this paper's system design addresses.

Google Apps Script, while primarily documented as a tool for automating Google Workspace tasks (Sheets, Docs, Gmail), is used here in a different capacity: as a lightweight, zero-configuration HTTP request handler with first-class, no-credential access to Google Drive on behalf of its owning account. This repurposing is, to the authors' knowledge, an underexplored pattern for low-cost data-collection tools outside of the Apps Script developer community itself.

3. SYSTEM ARCHITECTURE

The system comprises four logical layers, illustrated in Figure 1: a client capture layer running entirely in the candidate's browser; a static delivery layer that serves the client application over HTTPS (a precondition for `getUserMedia()` in all modern browsers); a serverless integration layer that authenticates and reassembles incoming recordings; and a persistence layer (Google Drive) that stores the final artifact. Table 2 details the specific technology occupying each layer.

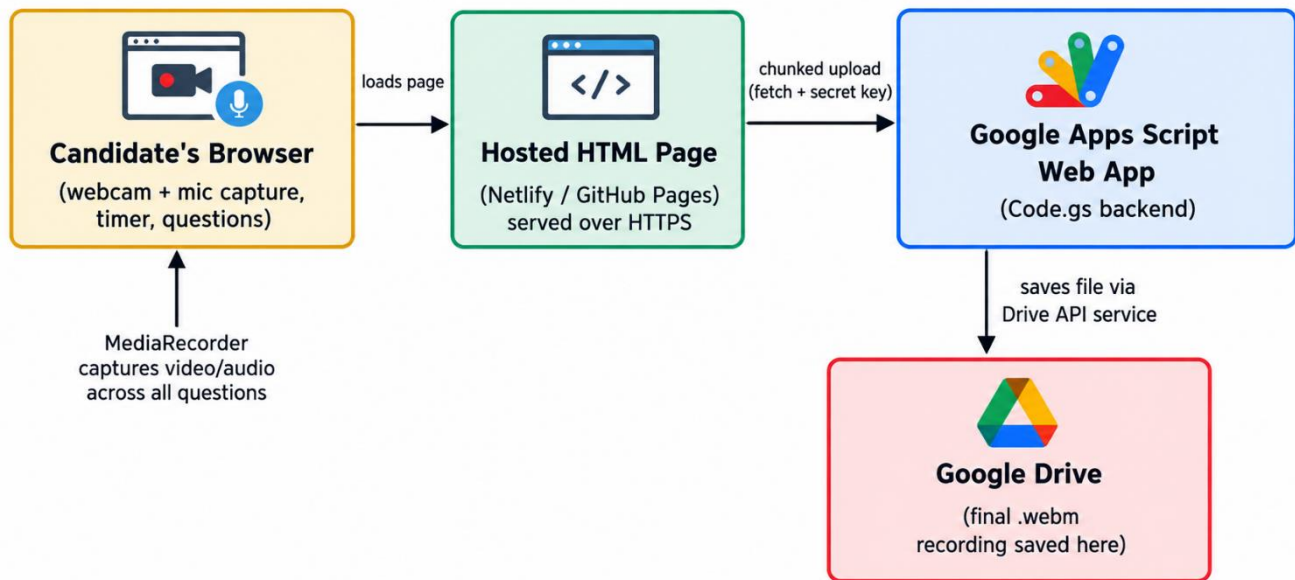


Figure -1: Data flow from candidate browser through hosted delivery, serverless integration, and cloud persistence.

Table -2: Technology stack by architectural layer.

Layer	Technology	Role in the System
Client capture	HTML5, CSS3, vanilla JavaScript (ES2017+)	State machine for the candidate-facing flow; no external framework dependency
Media acquisition	MediaDevices.getUserMedia(); MediaRecorder	Continuous webcam/microphone capture, encoded client-side as WebM (VP8/VP9 + Opus)
Transport	Fetch API; Base64 transcoding; custom chunk protocol	Splits the recording into bounded-size segments and transmits them with retry/backoff
Integration / compute	Google Apps Script (V8 runtime), deployed as a Web App	Stateless request handler executing under the operator's Google identity; reassembles chunks
Persistence	Google Drive, via the DriveApp service	Durable storage of the final artifact; folder-scoped access
Delivery	Static hosting (e.g. Netlify, GitHub Pages)	Serves the client over HTTPS, a precondition for getUserMedia() in modern browsers

A defining architectural property is the asymmetry of identity: the candidate authenticates to nothing, while the serverless integration layer executes under a single, fixed Google identity — the operator's — established once at deployment time. This is what allows an arbitrary, unauthenticated candidate to cause a file to be written into a specific, private Google Drive without either party performing an OAuth handshake at request time.

4. IMPLEMENTATION

4.1 Client-Side Capture Pipeline

On operator-configured cue, the client requests combined audio-video access via `getUserMedia({video:true, audio:true})` and constructs a single `MediaRecorder` instance spanning the entire interview session — that is, one continuous recording underlies all operator-defined questions, rather than one recording per question. This design choice was made deliberately: initiating and tearing down a new `MediaRecorder` per question would introduce visible re-permissioning latency and risk partial-answer loss at question boundaries, whereas a single continuous stream, segmented only by on-screen question text and a per-question countdown timer, preserves an unbroken record of the session. MIME type negotiation is performed defensively, in preference order (VP9/Opus, VP8/Opus, generic WebM), using `MediaRecorder.isTypeSupported()` to accommodate differing browser codec support.

4.2 Timed Question Flow

Each question is presented with a fixed countdown (operator-configurable, default 60 seconds) implemented via a client-side interval timer. Advancement to the next question occurs either on operator-defined timer expiry or on explicit candidate action, and in both cases the underlying recording is left uninterrupted. Progress is reflected to the candidate via an ordinal indicator, and the final question's advancement triggers session finalization rather than a further question.

4.3 Chunked Upload Protocol

A single-request upload strategy — transmitting the entire encoded recording, Base64-transcoded, in one HTTP POST — was evaluated first and found to be operationally fragile for two independent reasons. First, Base64 transcoding imposes a fixed, unavoidable 33% size overhead (empirically confirmed in Section 6.1), meaning a request payload limit imposed by the integration platform translates to a substantially lower effective ceiling on raw recording size. Second, and more significantly, larger single requests proved more susceptible to transient network interruption, with no ability to resume a partially transmitted upload short of restarting it in full.

The system therefore implements a three-phase chunked protocol between client and server: a start phase, in which the server allocates an opaque session identifier; a chunk phase, repeated for each fixed-size (default 3 MB raw, pre-encoding) segment of the recording, in which the client transmits the segment tagged with its session identifier and a zero-padded sequential index; and a finish phase, in which the server locates every segment matching the session identifier, orders them lexicographically by their zero-padded index (and therefore correctly by upload order), concatenates their decoded bytes, and persists the result as the final artifact. Each phase is independently retried, with exponential backoff, up to a fixed retry ceiling, isolating transient failures to a single small request rather than the entire transfer.

4.4 Server-Side Reassembly

The Apps Script backend stores each incoming chunk as an individual temporary Drive file rather than attempting to hold the entire in-progress upload in memory or in a request-scoped variable, since Apps Script execution contexts are stateless between HTTP invocations. At finalization, chunk bytes are read back via the Blob interface, concatenated using a typed-array (Uint8Array) buffer sized to the aggregate byte length, and written once as the final file — after which the temporary per-chunk files are removed. This design keeps peak memory pressure bounded by total recording size rather than by any pathological per-request behavior, and, because the finish phase is idempotent with respect to already-uploaded chunks, a failed finalization can be safely retried without re-uploading data.

4.5 Access Control Model

Because candidates are, by design, unauthenticated with respect to Google, the Apps Script deployment must be configured with public ("Anyone") access in order to accept their requests. This necessarily means the integration endpoint is reachable by any party in possession of its URL. The system mitigates this with an application-level shared secret: every request must include a pre-shared string, validated server-side before any Drive operation is attempted, with requests failing validation rejected before any file I/O occurs. This is deliberately a lightweight control commensurate with the system's threat model (opportunistic misuse of a leaked URL) rather than a defense against a sophisticated or motivated adversary; Section 5 discusses this tradeoff explicitly.

5. SECURITY AND PRIVACY CONSIDERATIONS

The shared-secret mechanism described in Section 4.5 is visible in the client-side source of a statically hosted page and therefore should not be treated as cryptographically strong access control; it is better characterized as a deterrent against casual or accidental misuse than as a barrier to a determined attacker who has retrieved the page source. Operators requiring stronger guarantees should consider rotating the secret periodically, restricting distribution of the hosted link, and, as a future enhancement, moving the secret out of client-side code entirely via a minimal proxy layer that holds it server-side.

The recordings themselves constitute personally identifiable and biometric-adjacent data (a candidate's face, voice, stated name, and date of birth). Operators deploying this system are responsible for satisfying applicable data-protection obligations in their jurisdiction — including, where relevant, informing candidates that recording is occurring and obtaining any consent required before the interview begins. This paper does not constitute legal guidance and operators should seek appropriate advice specific to their jurisdiction and use case.

All network transport occurs over HTTPS as an operational precondition (browsers refuse `getUserMedia()` over insecure origins other than localhost), providing transport-layer confidentiality for both the recording upload and the delivery of the client application itself.

6. EVALUATION

Because this system has not yet been subjected to a controlled study with independent candidate participants, this section reports two categories of evidence actually available at the time of writing: (a) an account of the iterative, real-world debugging process undertaken during development, which surfaced genuine interoperability failure modes rather than hypothetical ones; and (b) controlled technical micro-benchmarks of the encoding and transport pipeline, run on representative synthetic payloads. We are explicit that neither category constitutes a substitute for a controlled usability study, and we identify that study as the primary item of future work in Section 7.

6.1 Encoding Overhead Micro-Benchmark

Base64 transcoding time and size overhead were measured directly (Node.js V8 runtime, representative of the JavaScript engine performing the equivalent operation in-browser) across a range of buffer sizes representative of individual upload chunks. Results are shown in Table 3.

Table -3: Measured Base64 encoding time and size overhead by payload size.

Raw Size (MB)	Encode Time (ms)	Encoded Size (MB)	Overhead
1	6.44	1.33	+33.3%
3	17.90	4.00	+33.3%
5	30.45	6.67	+33.3%
10	64.23	13.33	+33.3%

The measured overhead is a constant 33.3% across all tested sizes, consistent with Base64's theoretical 4:3 expansion ratio, confirming that this overhead is a fixed structural cost of the encoding scheme rather than a size-dependent effect. Encoding time scales approximately linearly with payload size, remaining well under 100 milliseconds even at 10 MB, indicating that client-side encoding latency is not a meaningful contributor to perceived upload delay at the chunk sizes used in production (3 MB raw per chunk).

6.2 Payload Scaling Projection

Using the system's default recording bitrate configuration (1.2 Mbps video, 64 kbps audio; 1.26 Mbps combined), Table 4 projects raw recording size and resulting chunk count across a representative range of interview durations, at the default 3 MB raw chunk size.

Table -4: Projected file size and chunk count by interview duration.

Interview Duration (min)	Projected Raw File Size (MB)	Chunks at 3 MB/segment
1	9.0	4
2	18.1	7
4	36.2	13
6	54.2	19
10	90.4	31
20	180.8	61

These figures indicate that the chunked design keeps per-request payloads bounded and constant (approximately 4 MB encoded per chunk) regardless of total interview duration, converting what would otherwise be a single request scaling linearly with interview length — and eventually exceeding common serverless payload ceilings — into a bounded-size, linearly-repeated request pattern. A 20-minute interview, for instance, produces 61 chunk requests rather than one 180 MB (and, Base64-encoded, approximately 241 MB) request.

6.3 Iterative Real-World Testing

During deployment testing, two failure modes were encountered and resolved that are documented here as they are likely to recur for other implementers of similar architectures. First, opening the client application via a local file:// URL rather than an HTTPS-hosted address produced a browser-level network error on every upload attempt; this was traced to browser cross-origin request restrictions specific to the file:// scheme rather than any fault in the integration logic, and was resolved by requiring HTTPS static hosting as a deployment precondition. Second, an initial implementation using the XMLHttpRequest transport consistently failed against the Apps Script endpoint despite a correctly configured and reachable deployment; this was traced to Apps Script's internal use of an HTTP redirect when returning a Web App response, which XMLHttpRequest did not transparently follow in a manner compatible with reading the JSON response body, and was resolved by switching the client transport to the Fetch API, which handles this redirect transparently. Both findings are, to the authors' knowledge, not prominently documented in Apps Script's own developer documentation, and are offered here as a contribution to practitioners adopting a similar architecture.

7. LIMITATIONS AND FUTURE WORK

- No controlled usability study has yet been conducted with independent candidate or recruiter participants; reported evaluation is limited to technical micro-benchmarks and developer-conducted testing, and claims of practical usability should be read as provisional pending such a study.
- The shared-secret access control model, while adequate against opportunistic misuse, does not constitute strong authentication and would benefit from a server-side secret-holding proxy in future iterations.
- The system currently produces one continuous recording per session rather than per-question segments; per-question segmentation (with in-band or metadata-based timestamps) is a natural extension for evaluators who wish to review or score individual answers independently.
- No automated content moderation, transcription, or scoring is performed; the system is scoped strictly to capture and storage, with evaluation left entirely to the human reviewer.

- Very long sessions at high recording quality increase total upload time (linearly, per Table 4) and remain bounded by the candidate's upstream network conditions; no adaptive bitrate mechanism is currently implemented.
- Future work should include: (i) a controlled multi-participant usability study measuring completion rate, perceived ease of setup for non-technical operators, and upload success rate under varied network conditions; (ii) a formal security review of the shared-secret model against realistic threat scenarios; and (iii) an accessibility evaluation (screen-reader compatibility, keyboard-only operation) not addressed in the current implementation.

8. CONCLUSION

This paper presented the design, implementation, and preliminary technical evaluation of a video interview recording system constructed entirely from standards-based browser media-capture APIs and a repurposed serverless scripting platform, requiring no paid infrastructure and no software engineering background to deploy. The chunked upload protocol described removes a payload-size constraint that would otherwise bound recording length on the chosen serverless platform, and the iterative debugging findings reported in Section 6.3 document two genuine, previously undocumented interoperability failure modes relevant to future implementers of similar architectures. The system's principal open question — whether it performs acceptably for real candidates and recruiters under uncontrolled, real-world conditions — remains for future controlled evaluation, which we identify as the necessary next step toward substantiating broader claims of practical utility.

REFERENCES

- [1] W3C, "Media Capture and Streams," W3C Recommendation. Available: <https://www.w3.org/TR/mediacapture-streams/>
- [2] W3C, "MediaStream Recording," W3C Working Draft. Available: <https://www.w3.org/TR/mediastream-recording/>
- [3] Google, "Apps Script: Web Apps," Google Developers Documentation. Available: <https://developers.google.com/apps-script/guides/web>
- [4] Google, "Class DriveApp," Google Apps Script Reference. Available: <https://developers.google.com/apps-script/reference/drive/drive-app>
- [5] MDN Web Docs, "MediaRecorder API," Mozilla. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder_API
- [6] MDN Web Docs, "Fetch API," Mozilla. Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- [7] HireVue, Inc. — corporate product documentation, referenced as a representative commercial video-interview platform. Available: <https://www.hirevue.com/>