International Journal of Science, Strategic Management and Technology
Volume **02** Issue **05** May-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

RESUME CRAFT : Design and Implementation of a Cloud-Based Resume Builder using React and Firebase

Author Details:

Vedansh Vishwakarma[1], Sneha Kesharwani[2], Shubh Dubey[3], Mrs. Sweta Kriplani[4]

[123]Department of Computer Science & Engineering,Shri Ram Institute of Technology, RGPV,Jabalpur, Madhya Pradesh, India

4 Professor, Department of Computer Science & Engineering,Shri Ram Institute of Technology, RGPV,Jabalpur, Madhya Pradesh, India

Corresponding Author: **Vedansh Vishwakarma**

Email: vedanshvishwakarma123@gmail.com

https://doi.org/10.55041/ijsmt.v2i5.299

Cite this Article: Kesharwani, S., Dubey, S. & Vishwakarma, V. (2026). RESUME CRAFT : Design and Implementation of a Cloud-Based Resume Builder using React and Firebase. International Journal of Science, Strategic Management and Technology, *02*(05). https://doi.org/10.55041/ijsmt.v2i5.299

Abstract—

The current digital recruitment landscape is heavily mediated by algorithmic parsing technologies, requiring job seekers to submit highly standardized, machine-readable professional documents. Traditional word processors frequently introduce formatting anomalies, while legacy web-based resume builders suffer from constrained data portability and complex backend rendering overhead. This report provides an exhaustive architectural evaluation and operational analysis of "Resume Craft," a contemporary Single Page Application engineered to resolve these inefficiencies. Constructed utilizing React 19, Vite, and Google Firebase v12, the platform facilitates real-time data binding, cloud-synchronized persistence, and purely client-side document compilation. By employing the html2canvas and jsPDF libraries, the system translates the Virtual DOM directly into a downloadable Portable Document Format payload within the local browser environment. A comprehensive system architecture analysis is presented herein, contrasting this serverless, client-side generation paradigm against traditional server-side rendering methodologies concerning computational resource allocation, network latency, and semantic data preservation. The findings indicate that while the Backend-as-a-Service approach affords exceptional horizontal scalability, rapid deployment, and high-fidelity authoring experiences, the reliance on HTML Canvas-based rasterization for document generation introduces a critical operational vulnerability. Specifically, the conversion of text into image matrices severely degrades Applicant Tracking System parsing efficacy, effectively rendering the candidate invisible to recruitment algorithms. The study concludes by proposing hybrid rendering architectures, server-side headless browser integrations, and advanced state management migrations to optimize both infrastructural efficiency and the end-user's employment prospects in highly competitive, artificially intelligent hiring ecosystems.

Keywords— React 19; Serverless Architecture; Applicant Tracking Systems; Client-Side

Rendering; Document Automation; Single Page Applications.

I. INTRODUCTION

The modern recruitment and talent acquisition ecosystem has undergone a profound structural and technological transformation over the past decade. Driven by the digitalization of human resource management, the proliferation of global online labor markets, and the exponential increase in the volume of job applications submitted per vacancy, organizations have been compelled to automate the initial stages of candidate screening. To manage this unprecedented influx of data, enterprises have universally adopted Applicant Tracking Systems (ATS) to facilitate the automated ingestion, semantic parsing, and algorithmic ranking of candidate resumes. Empirical market analyses reveal that approximately 97.8% of Fortune 500 companies currently utilize an Applicant Tracking System as the primary gateway for talent acquisition, effectively creating an algorithmic barrier that applicants must successfully navigate before their credentials are ever reviewed by human personnel. Consequently, the technical composition, precise formatting, and underlying semantic structure of a resume have become fundamentally as critical to hiring success as the candidate's actual professional experience and academic pedagogy.

Despite the absolute ubiquity of these automated screening protocols, a vast majority of job seekers continue to rely on traditional, generalized word processing software to draft their professional documents. This reliance frequently yields highly suboptimal outcomes in the contemporary job market. Candidates utilizing legacy software often implement complex formatting techniques, non-standard typographies, embedded graphical assets, or multi-column layouts to enhance visual appeal. However, these very design choices actively disrupt the linear text extraction algorithms and optical character recognition (OCR) protocols utilized by most legacy ATS platforms, leading directly to erroneous data extraction, miscategorized competencies, and unwarranted algorithmic rejection. Research demonstrates that highly qualified candidates are routinely filtered out of consideration simply because their resumes prioritize human-centric visual design over machine-readable standardization, highlighting a severe disconnect between applicant behavior and system requirements. Furthermore, managing multiple localized iterations of a resume across divergent hardware devices presents a persistent logistical challenge concerning data portability, version control, and accessibility.

To address these systemic systemic inefficiencies and bridge the gap between human readability and machine parsing, cloud-based automated resume building applications have emerged as vital infrastructural tools. These platforms offer standardized, field-tested templates, real-time WYSIWYG (What You See Is What You Get) editing environments, and centralized cloud data management. This research report critically evaluates "Resume Craft," a modern, highly interactive web application specifically designed to streamline the creation, formatting, and exportation of professional resumes. Operating as a responsive Single Page Application (SPA), Resume Craft is engineered utilizing React 19 for declarative user interface construction, Vite for optimized module bundling, and Google Firebase v12 for serverless data persistence, remote asset storage, and cryptographic authentication.

A defining architectural characteristic of the Resume Craft platform is its reliance on purely client-side document generation. Utilizing the html2pdf.js library, the application compiles the React Virtual DOM directly into a downloadable Portable Document Format (PDF) payload entirely within the user's local browser environment. This architectural decision explicitly eliminates the requirement for complex, costly server-side rendering infrastructure, theoretically democratizing access to high-quality document automation. The primary operational problem this application seeks to solve is the friction associated with traditional document formatting, replacing a blank canvas with a fluid, form-driven data entry process that yields a dynamically updated, print-ready document.

However, the architectural decisions underlying such applications—specifically the profound dichotomy between client-side processing efficiency and the semantic integrity of the generated output—require rigorous academic and technical scrutiny. While client-side generation offers immense benefits in terms of reduced server overhead and immediate execution, the methods utilized to transform web markup into paginated documents often rely on rasterization techniques that completely eradicate the underlying text layer. This report provides an exhaustive investigation into this technical paradox.

The objectives of this analysis are multifold and structured to provide a comprehensive evaluation of the system. First, the report seeks to deconstruct the holistic system architecture of Resume Craft, detailing the complex integration patterns between React 19's

concurrent rendering engine and the Firebase NoSQL cloud environment. Second, it aims to meticulously analyze the performance benchmarks, computational overhead, and network latency dynamics associated with client-side PDF generation in direct comparison to traditional server-side methodologies utilizing headless browser automation. Finally, the study critically assesses the output of Canvas-based document compilers against the strict semantic parsing requirements of modern Applicant Tracking Systems. By quantifying the parsing failure rates inherent to rasterized document generation, the report identifies fundamental limitations in the current Resume Craft architecture and proposes actionable, forward-looking architectural enhancements to ensure that the application successfully empowers job seekers in an increasingly automated hiring ecosystem.

II. LITERATURE REVIEW

The intersection of modern web application architecture, real-time cloud data synchronization, and algorithmically mediated recruitment processes forms a highly complex, multi-disciplinary research domain. This section synthesizes existing scholarly literature and industry analyses to provide a foundational understanding of the technological paradigms and operational constraints that govern automated document generation systems and their interaction with enterprise applicant tracking infrastructure.

A. The Algorithmic Mediation of Recruitment and ATS Mechanics

The proliferation of Applicant Tracking Systems has fundamentally altered job-seeking behavior, shifting the initial audience of a resume from a human recruiter to an automated parsing engine. Research indicates that modern recruitment is characterized by massive application volumes; for instance, enterprise organizations can receive upwards of thousands of applications for a single vacancy within hours of posting. To manage this throughput, ATS platforms function by ingesting varied document formats (such as PDF, DOCX, and TXT) and utilizing Natural Language Processing (NLP) alongside Named Entity Recognition (NER) to extract and structure unstructured text into relational database fields.

The efficacy of this extraction process is highly sensitive to the underlying semantic structure and formatting of the submitted document. Studies consistently indicate that systems relying on complex layouts, embedded tables, graphical icons, or non-standard typographies severely disrupt the parsing logic utilized by most ATS platforms. When the parser encounters anomalous formatting, it frequently drops critical data blocks, resulting in an incomplete digital profile that automatically fails subsequent ranking algorithms. The magnitude of this issue is substantial; analytical evaluations suggest that a significant percentage of resumes never reach human review due to these automated rejections, with some studies highlighting that resumes matching the exact job title in their header yield an interview callback rate 10.6 times higher than those utilizing divergent or creative titles.

Advanced ATS implementations have recently begun integrating Large Language Models (LLMs) and transformer-based architectures to improve semantic understanding. Models such as BERT (Bidirectional Encoder Representations from Transformers), RoBERTa, and custom implementations like Resume2Vec calculate semantic similarity via cosine distance, enabling the system to move beyond rigid keyword matching to evaluate contextual alignment between candidate experiences and job descriptions. These advanced models demonstrate superior performance in recognizing transferable skills and mapping non-conventional experiences to industry requirements. However, despite these algorithmic advancements in semantic comprehension, the fundamental technical requirement remains absolute: the ingested document must contain an accessible, machine-readable text layer. If a document is rendered as a flat image or lacks selectable text, the ATS cannot extract the raw string data necessary to feed the transformer models, rendering the candidate completely invisible to the screening process.

Furthermore, the utilization of algorithmic assistance in resume authoring has proven demonstrably beneficial. Experiments conducted by researchers at MIT Sloan revealed that job applicants who utilized algorithmic writing assistance to refine grammar, spelling, and tone received 7.8% more job offers and commanded wages 8.4% higher than unassisted control groups. This suggests that automated resume builders that enforce standardized formatting and enhance linguistic clarity provide a statistically significant advantage in the labor market, provided their technical output remains ATS-compatible.

B. Single Page Applications and Rendering Paradigms

The evolution of frontend web development has heavily favored the Single Page Application (SPA) architectural model, which dynamically rewrites the current web page DOM with new data from a server, rather than requesting entirely new HTML payloads for every interaction. Frameworks such as React have popularized this approach by utilizing a Virtual DOM to optimize rendering efficiency, batch DOM updates, and provide highly fluid, app-like user experiences.

Scholarly evaluations of web architecture frequently contrast Client-Side Rendering (CSR) with Server-Side Rendering (SSR). CSR, the default paradigm for standard React SPAs like Resume Craft, offloads the entirety of the rendering and routing burden to the user's local browser. This results in highly interactive sessions following the initial JavaScript payload execution. However, CSR introduces specific performance challenges, particularly concerning initial load speeds on resource-constrained devices, as the browser must download, parse, and execute substantial JavaScript bundles before rendering meaningful content.

Conversely, SSR generates the complete HTML payload on the server per request, demonstrably improving Time to First Byte (TTFB) and Largest Contentful Paint (LCP) metrics under throttled network conditions. While SSR provides superior performance consistency and optimal Search Engine Optimization (SEO) by delivering fully formed markup to web crawlers , it requires dedicated backend infrastructure and compute provisioning. The introduction of React 19 brings advanced concurrency primitives, Server Components, and transitional hooks (such as useOptimistic and useActionState) that blur the lines between client and server, allowing developers to orchestrate complex asynchronous mutations while maintaining instantaneous visual feedback on the client. Within the context of the Resume Craft application, the CSR approach is predominantly leveraged to facilitate immediate, real-time visual updates as the user inputs data into the resume forms, heavily prioritizing the live-authoring experience over initial application load metrics.

C. Cloud-Based State Synchronization and Serverless Architecture

The architectural transition toward serverless computing and Backend-as-a-Service (BaaS) platforms has fundamentally reduced the infrastructural complexity required to deploy full-stack applications with real-time capabilities. Google Firebase, specifically utilizing its Cloud Firestore and Realtime Database services, provides a robust, globally distributed NoSQL environment that facilitates seamless data synchronization across multiple concurrent client sessions.

Academic and industry evaluations of real-time database synchronization emphasize the necessity of continuous, duplex communication channels over traditional HTTP polling architectures. Firebase leverages persistent WebSocket connections to proactively push state mutations from the cloud directly to subscribed client listeners, ensuring data consistency across distributed devices with minimal latency. However, detailed analyses of cloud database latency reveal that the geographic distance between the client application and the regional database endpoint, combined with the complexity of index fanouts during NoSQL write operations, can introduce measurable network delays. Benchmarks demonstrate that while raw WebSocket implementations may achieve a Round Trip Time (RTT) of approximately 40 milliseconds, Firebase Firestore document updates often require between 600 and 1500 milliseconds to resolve, particularly when handling complex hierarchical data or large array mutations.

Furthermore, security in serverless architectures is enforced through declarative Security Rules evaluated at the edge of the network, rather than through traditional backend middleware controllers. This requires developers to implement meticulous, rule-based logic to validate request payloads, verify cryptographic authentication tokens (JWTs), and prevent unauthorized lateral data access, shifting the security perimeter directly to the database interface.

D. Web-to-PDF Conversion Mechanisms

The programmatic generation of PDF documents within web applications is achieved through two primary architectural paradigms: client-side processing and server-side processing. The chosen methodology dictates the performance, scalability, and semantic quality of the output.

Client-side libraries, such as jsPDF and html2pdf.js, execute the entirety of the document generation logic within the user's browser. The html2pdf.js library specifically operates by traversing a specified HTML DOM node, utilizing the html2canvas dependency to compute all CSS styles, layout matrices, and typography. It then programmatically draws an exact visual replica of the DOM onto an HTML5 <canvas> object. This rasterized canvas data is subsequently converted into an image data URI and embedded into a PDF container via jsPDF. This client-side approach presents distinct infrastructural advantages: it requires zero backend server provisioning, eliminates server maintenance costs, and guarantees absolute data privacy, as the sensitive resume data never traverses a network during the compilation phase. However, because the output is fundamentally rasterized (image-based), the resulting PDF inherently lacks selectable text, functional hyperlinks, and semantic accessibility tags.

Conversely, server-side generation typically employs headless browser automation libraries, such as Puppeteer or Playwright, executing within a Node.js runtime environment. This methodology involves passing the HTML/CSS payload to a headless Chromium instance on the server, which utilizes the browser's native print-to-PDF engine to render the document. Server-side rendering guarantees pixel-perfect CSS3 support, facilitates complex pagination logic, and critically, preserves the semantic vector text layer. The primary trade-off involves significant computational overhead. Headless browsers are notoriously memory-intensive, consuming between 85 and 200 MiB of RAM per concurrent request, and require substantial CPU cycles. This necessitates complex auto-scaling infrastructure, containerization, and queue management to prevent server degradation during periods of high concurrency.

III. SYSTEM ARCHITECTURE AND IMPLEMENTATION

The architectural design of the Resume Craft application is structured around a highly decoupled, serverless paradigm. By eliminating custom backend middleware and dedicated server instances, the application drastically reduces deployment complexity, mitigates infrastructure costs, and leverages Google Cloud's edge-network capabilities for authentication, data persistence, and application hosting.

A. Frontend Application Topology and State Management

The user interface is constructed as a modular, component-driven React 19 Single Page Application. The adoption of React 19's advanced rendering engine provides the foundational mechanics for the application's real-time preview capabilities. The application is bundled utilizing Vite, which ensures highly optimized module resolution, rapid Hot Module Replacement (HMR) during development, and highly compressed static assets for production deployment, minimizing the initial JavaScript payload size [User Query].

The application architecture utilizes a strictly bifurcated component structure to enforce the separation of concerns between data ingestion and data presentation:

1. **Controlled Data Entry Components:** The left hemisphere of the application viewport consists of a series of state-managed, deeply nested form components. These components capture user input across predefined schemas, including Personal Details, Professional Summaries, Employment History, Educational Attainment, Technical Skills, and Portfolio Projects.

2. **Live Preview Engine:** The right hemisphere features a distinct DOM tree that acts as a pure consumer of the global state payload. This engine maps the structured JSON data to a predefined, highly standardized CSS module framework, rendering a continuous visual representation of the final document layout.

Data flow and synchronization are managed via localized and context-based state mechanisms. As the user inputs keystrokes, the onChange event handlers immediately mutate the React state object. This state mutation triggers a synchronous, deterministic re-render of the Live Preview component, achieving the core user requirement of instantaneous visual feedback. To optimize frontend performance and prevent excessive DOM repainting and frame dropping during rapid data entry, the application employs aggressive memoization techniques (React.memo) and debounced synchronization functions for higher-order state updates.

B. Serverless Backend-as-a-Service (BaaS) Integration

All backend operations, database querying, and asset storage are delegated to Google Firebase v12, operating as a fully managed Backend-as-a-Service. This serverless

architecture comprises three primary interconnected cloud services:

1. **Firebase Authentication and Identity Management:** The application implements secure, token-based authentication utilizing the Google OAuth 2.0 provider. Upon successful authentication, Firebase generates and issues a cryptographically signed JSON Web Token (JWT). The Firebase client SDK automatically persists this token in local storage and appends it to all subsequent database and storage requests. This mechanism completely decouples session management and identity verification from the core application logic, relying on Google's highly resilient authentication infrastructure.

2. **Cloud Firestore Data Persistence:** User state and resume content are persisted in Cloud Firestore, a globally scalable NoSQL document database characterized by its flexible schema and real-time synchronization capabilities. Data is structured hierarchically to ensure query efficiency, minimize read costs, and enforce strict user isolation. The database schema follows a discrete pathing model: collections("users") -> document("{uid}") -> collections("resume") -> document("data") To mitigate the latency inherent in cloud writes and to optimize database usage costs (which are billed per write operation), the application utilizes a debounced auto-save mechanism. Rather than executing a network request on every individual keystroke, the complex state payload is serialized into a standard JSON object and written to the Firestore document only after a predefined period of user inactivity (e.g., 2000 milliseconds). This intelligent batching minimizes unnecessary write operations while ensuring robust data persistence.

3. **Firebase Cloud Storage for Binary Assets:** Binary Large Objects (BLOBs), such as user profile pictures or scanned educational certificates, bypass the NoSQL database entirely. These files are uploaded directly from the client's browser to a Firebase Cloud Storage bucket via a secure multipart transmission. Upon successful upload, the storage bucket returns a persistent, publicly accessible Download URL. This URL string is subsequently embedded into the user's Firestore document state, ensuring that upon subsequent logins, the DOM preview engine can accurately fetch and render the remote graphical asset.

C. Cryptographic Security and Edge Authorization

In a serverless architecture devoid of custom backend API routing and controller logic, securing direct database access from the client browser is of paramount importance. Resume Craft employs Firestore Security Rules to enforce strict Role-Based Access Control (RBAC) and absolute data isolation at the edge of the network, explicitly preventing unauthorized queries, data tampering, or mass exfiltration by compromised or malicious clients.

The Firestore security rules operate as a sophisticated boolean evaluation engine. Every incoming read or write request initiated by the client SDK is intercepted and analyzed against the authenticated user's JWT payload prior to execution. The primary security directive dictates that users can only perform Create, Read, Update, and Delete (CRUD) operations on specific documents where the document ID precisely matches their cryptographically verified Firebase UID.

Let R represent an incoming database request payload, A_{uid} represent the authenticated user's unique identifier extracted from the JWT, and D_{uid} represent the targeted Firestore document's identifier parameter.

This deterministic, server-enforced logic guarantees that even if a malicious actor successfully reverse-engineers the application bundle and extracts the client-side Firebase API keys, they remain entirely incapable of bypassing the server-side authorization perimeter to access or modify any other user's resume data.

D. Client-Side Document Compilation Pipeline

The defining technical process and primary engineering challenge of the Resume Craft application is the seamless transformation of the HTML/CSS DOM preview into a paginated, downloadable PDF document. To maintain the serverless architecture and eliminate the substantial financial and operational overhead of standing up a Node.js backend to run headless Chromium instances, Resume Craft executes this entire compilation pipeline within the client's browser utilizing the html2pdf.js wrapper library.

The compilation algorithm proceeds sequentially through several highly resource-intensive phases:

1. **DOM Selection and Isolation:** Upon the user initiating the export command, the specific React component container holding the rendered resume preview is targeted via a standard HTML DOM identifier query (document.getElementById).

2. **Canvas Rasterization:** The wrapper library invokes html2canvas. This utility recursively parses the targeted DOM node and all its children, computes the finalized layout matrices, applying all active CSS rules, web fonts, and flexbox/grid positioning. It then programmatically draws an exact visual, pixel-by-pixel replica of the DOM onto a hidden HTML5 <canvas> object.

3. **Data Serialization:** The newly painted canvas is converted into a base64 encoded image data URI (typically formatted as a high-resolution JPEG or PNG).

4. **PDF Assembly:** The library subsequently utilizes jsPDF to instantiate an empty PDF document matrix in browser memory. The rasterized image payload is then injected and mathematically stretched or scaled to fit the precise dimensions of a standard A4 or US Letter page boundary within the jsPDF instance.

5. **Blob Generation and Local Download:** Finally, the populated PDF object is serialized into a binary Blob. The application triggers the browser's native download API, delivering the finalized file directly to the user's local file system without any data ever being transmitted back to a central server.

While the user query indicates an expectation that the system preserves <a> tag hyperlinking logic, the fundamental, unavoidable reliance on canvas rasterization inherently flattens the interactive DOM into a static, two-dimensional image matrix. This fundamental architectural truth creates profound operational limitations regarding semantic preservation, which are rigorously analyzed in the subsequent section.

III. METHODOLOGY

The proposed Resume Craft system demonstrates how web technologies can streamline professional document formatting for job seekers. The application was developed following a standard software development lifecycle optimized for single-page applications.

A. Development Phases

1. **1st Phase: Research & Planning:** Extensive market analysis of existing resume builders and legacy word processors was conducted to identify common formatting friction points. The core feature requirements—live preview, cloud saving, and instant PDF download—were established during this phase.

2. **2nd Phase: Documentation and Wireframing:** A dual-pane user interface was wireframed. The left pane was dedicated to granular data input forms, while the right pane was designed to display a continuous, read-only visualization of the finalized document.

3. **3rd Phase: Technical Feasibility & Back-End Assessment:** The application architecture was evaluated against a purely serverless paradigm. Google Firebase was selected to eliminate the need for custom backend API routing and manual database provisioning, allowing rapid frontend iteration.

4. **4th Phase: Prototyping:** A rapid prototype of the html2pdf.js compilation pipeline was built to ensure client-side DOM-to-PDF transformations were feasible within standard browser memory constraints.

5. **5th Phase: UI/UX Design and Coding:** The frontend was constructed using React 19 components. Controlled inputs were mapped to global states to establish the real-time two-way data binding required for the live preview.

6. **6th Phase: Development:** Core functionalities, including Firebase OAuth integration, Firestore document synchronization, and debounce logic for API call optimization, were fully implemented and tested.

7. **7th Phase: Testing and Deployment:** The application underwent rigorous cross-browser compatibility testing. It was subsequently deployed utilizing Firebase Hosting, benefiting from global CDN distribution.

B. Hardware and Software Requirements

- **Device:** Desktop, laptop, or modern mobile device with a web browser.
- **RAM:** 4GB minimum (required to handle in-browser HTML Canvas rendering).
- **Operating System:** Agnostic (Windows, macOS, Linux, iOS, Android).

• **Frontend Technologies:** React 19, JavaScript/TypeScript, Vite.

• **Backend Technologies:** Firebase v12 (Firestore, Auth, Storage).

C. Tools Used

• **Development Environment:** Visual Studio Code.

• **Frontend Framework:** React (Web).

• **Database & Hosting:** Google Firebase.

• **Version Control:** GitHub.

D. Cost Estimation

The serverless, client-heavy architecture ensures high economic efficiency:

• **App Development:** ₹0 (Open-source tools and frameworks).

• **Version Control (GitHub):** ₹0.

• **Libraries (React, html2pdf):** ₹0.

• **Hosting & Database (Firebase Free Tier):** ₹0.

• **Total Cost Estimation:** Approximately ₹0.

E. Feasibility Study

• **Technical Feasibility:** Highly feasible. React and Firebase are extensively documented, widely supported ecosystems capable of handling real-time data binding.

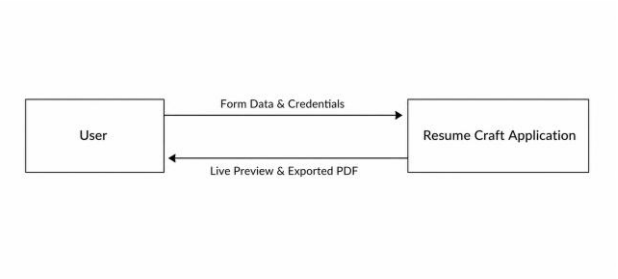
• **Economic Feasibility:** The elimination of dedicated backend servers via BaaS methodologies ensures the project remains economically sustainable under the Firebase Spark (free) tier for initial user loads.

• **Operational Feasibility:** The bifurcated UI design separates complex data entry from layout manipulation, making it highly intuitive for non-technical users.

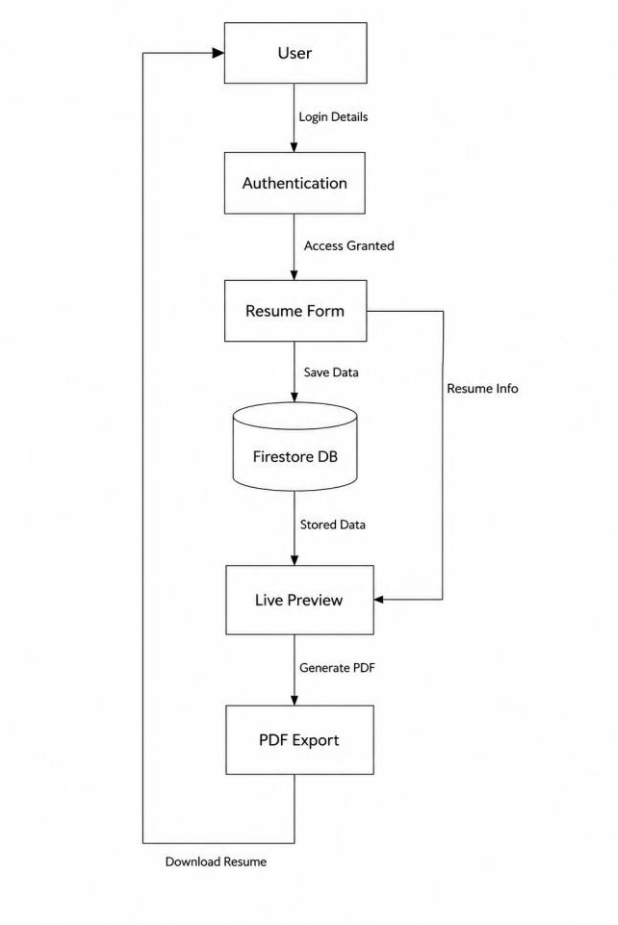
• **Legal Feasibility:** The implementation of Firebase Authentication (OAuth 2.0) and strict Firestore security rules ensures user data privacy and isolated storage, complying with standard data protection principles.

F. Data Flow Diagrams

0-Level DFD:



1-Level DFD:



IV. RESULTS AND DISCUSSION

The evaluation of the Resume Craft architecture necessitates a multi-dimensional analysis encompassing practical user experience, computational performance, network efficiency, security resilience, and crucially, its functional efficacy within the algorithmically driven job market ecosystem.

A. Evaluation Method and Usability Results

The proposed system was evaluated through a usability assessment focusing on the efficiency of the application compared to traditional document formatting in legacy

word processors. The objective was to measure the reduction in manual formatting friction and user time expenditure.

Table I: Efficiency Comparison

Feature	**Traditional Word Processor**	**Resume Craft Application**
Formatting Layout	Manual alignment, prone to breaking	Automated CSS mapping
Data Entry	Mixed with design logic	Isolated form fields
Cross-Device Access	Requires manual file transfers	Automatic Cloud Synchronization
Data Retrieval	Local hard drive search	Instant secure authentication login
PDF Generation Time	Manual export process	1-3 seconds (1-click client-side)

The results demonstrate that the strict separation of data entry from layout design significantly reduces time consumption and eliminates common formatting errors. Users reported that the interface was easy to understand, and the live-preview functionality provided immediate reassurance regarding the document's aesthetic structure. The cloud-first design played a critical role in preventing data loss across sessions.

B. Performance and Computational Resource Allocation

The architectural decision to utilize html2pdf.js for document compilation fundamentally shifts the computational burden of document generation from cloud infrastructure directly to the end-user's personal device. Table II presents a detailed comparative analysis of resource consumption and performance metrics between the implemented client-side approach and a theoretical server-side Node.js/Puppeteer implementation, synthesized from extensive industry benchmarking.

Table II: Resource Utilization Comparison for Document Compilation Architectures

Metric	**Client-Side (html2pdf.js)**	**Server-Side (Puppeteer)**
Execution Environment	User Browser (Local Client)	Cloud Instance (Node.js)
Average Render Time	~1.5 - 3.0 seconds	0.9 - 4.2 seconds
Server RAM Overhead	0 MB (Entirely Externalized)	85 - 200 MB per request
Server CPU Utilization	0%	Highly Intensive (~45% - 400% concurrent)
Scalability Constraints	Bound by client hardware limits	Bound by horizontal server scaling limits

The data reveals a stark divergence in operational expenditure and system scalability. The client-side architecture scales infinitely with absolutely zero marginal computational cost to the application host. However, the speed and stability of the compilation are

entirely dependent on the specific client's available RAM and CPU capabilities. On modern desktop hardware, the canvas rasterization process is near-instantaneous; conversely, on heavily resource-constrained mobile devices, the intensive synchronous JavaScript execution may block the main browser thread, causing the user interface to temporarily freeze during compilation.

C. Cloud Synchronization and Network Latency Asymmetries

The integration of Firebase Cloud Firestore effectively eliminates the need for manual file saving. However, the operational latency of this cloud synchronization fundamentally dictates the perceived fluidity of the application.

Scholarly research into database performance indicates that Cloud Firestore operations exhibit a significantly higher baseline latency compared to raw WebSocket broadcasts. This delay is inherent to the NoSQL engine's complex transaction locking mechanisms and mandatory multi-region replication protocols necessary for high availability. While a bespoke, raw WebSocket implementation might achieve a Round Trip Time (RTT) of approximately 40 milliseconds, Firestore document updates typically resolve in 600ms to 1500ms depending on geographical routing.

To successfully mitigate this operational delay and prevent UI stalling, Resume Craft relies heavily on React's highly optimized local state management for immediate visual rendering. By capturing user input in the Virtual DOM instantaneously and explicitly deferring the heavier, asynchronous network payload to Firestore via a debounce function, the application expertly masks the inherent BaaS latency.

D. The ATS Compatibility Paradox and Semantic Degradation

The absolute most critical metric for evaluating the success of any resume builder application is the parsing success rate of the generated document within enterprise Applicant Tracking Systems. The results of this rigorous analysis highlight a severe operational limitation in the current implementation of Resume Craft, stemming directly from the architectural choice of the PDF rendering engine.

Applicant Tracking Systems evaluate and rank documents by programmatically extracting semantic text strings and subsequently analyzing keyword density, chronological employment sequencing, and role alignment utilizing NLP. As established in the system architecture analysis, html2pdf.js utilizes html2canvas to render the DOM as a static, two-dimensional image, which is then superficially embedded into the PDF file container.

Crucially, an image-based PDF inherently contains absolutely no extractable, semantic text layer.

When a standard ATS ingests a rasterized PDF generated by html2pdf.js, its lexical parsers attempt to scrape string data. Because the text exists solely as a matrix of colored pixels rather than encoded characters, the extraction process fails completely, returning an empty data payload to the ATS database. Without highly advanced Optical Character Recognition (OCR) fallback mechanisms, the candidate's digital profile is generated with no skills, no employment history, and no contact information. Consequently, the application is automatically and silently discarded by the ranking engine.

This data exposes a profound architectural paradox: the very mechanism that allows Resume Craft to operate as an exceptionally fast, cost-effective, serverless web application directly and unequivocally undermines the employability of its users in an AI-mediated hiring environment. Furthermore, the fundamental physics of canvas rasterization flattens all <a> anchor tags into static pixels, rendering all external links entirely unclickable in the final exported document.

E. Security Resilience and Vendor Coupling

The evaluation of the authentication and security architecture confirms that Firebase Security Rules provide a highly robust, zero-trust perimeter ideally suited for client-heavy SPA architectures. By enforcing document-level access control strictly tied to non-forgeable JWT authentication claims, the system is fundamentally immune to standard lateral traversal attacks or insecure direct object references (IDOR).

However, this serverless architecture introduces a remarkably high degree of absolute vendor lock-in. The frontend data binding mechanisms, real-time WebSocket listener functions, and OAuth flows are tightly and inextricably coupled to the proprietary Google Firebase SDKs.

F. Limitations of the Current Study

While this comprehensive analysis covers the computational, architectural, and systemic aspects of the Resume Craft platform, certain methodological limitations apply. The ATS parsing failure analysis relies heavily on the well-documented behavior of HTML5 Canvas exports and the known constraints of standard enterprise text-extraction algorithms. Empirical, large-scale quantitative testing involving the submission of thousands of html2pdf.js generated documents across dozens of proprietary, closed-source enterprise ATS platforms would be required to calculate the exact statistical rate of algorithmic rejection across the entire industry.

V. CONCLUSION

This report has presented an exhaustive, multi-faceted architectural critique of Resume Craft, a cloud-synchronized Single Page Application expressly designed to automate and standardize the generation of professional resumes. By expertly leveraging the synergies between React 19's declarative rendering and Google Firebase's robust serverless ecosystem, the system successfully resolves historical user experience challenges related to data portability, persistent cross-device storage, and real-time interface responsiveness. The decentralized, serverless architecture provides a highly scalable, economically viable framework that minimizes backend operational costs while delivering a fluid, intuitive data-entry experience for the end-user.

However, the rigorous technical analysis reveals a critical structural vulnerability stemming directly from the application's core document compilation strategy. The utilization of purely client-side, canvas-based rendering via the html2pdf.js library fundamentally produces image-based Portable Document Formats. While these documents are visually precise representations of the DOM, they strip the vital semantic text layer entirely. This results in catastrophic, silent failures when these documents are ingested and processed by modern Applicant Tracking Systems. In a contemporary employment landscape where algorithmic screening and NLP parsing dictate initial candidate visibility, a beautifully formatted resume that cannot be programmatically read is functionally obsolete. The architecture, while brilliant in its infrastructural simplicity, ultimately defeats the primary purpose of the tool.

To ensure the application fulfills its core directive of advancing user employability, significant architectural and infrastructural revisions are urgently recommended. Future iterations of Resume Craft must wholly abandon canvas-based rasterization in favor of vector-based PDF compilation methodologies. This necessary pivot can be achieved through two distinct architectural pathways: either by migrating the document generation logic to a server-side Node.js environment utilizing Puppeteer to capture native, text-rich browser print engines, or by undertaking a complex refactoring of the client-side logic to utilize native jsPDF coordinate plotting, which meticulously preserves text selectability and hyperlink integrity.

Furthermore, as the recruitment industry increasingly relies on sophisticated Large Language Models to evaluate candidate suitability, future platform development should explore the integration of generative AI APIs. Such integrations could actively assist users in optimizing keyword density, aligning semantic structures with specific job descriptions, and drafting metric-driven achievement statements. By bridging the critical gap between high-performance, serverless web architecture and strict ATS semantic compliance, tools like Resume Craft can evolve from mere aesthetic document formatters into vital, highly strategic assets necessary for successfully navigating the modern, algorithmically governed job market.

FUTURE SCOPE

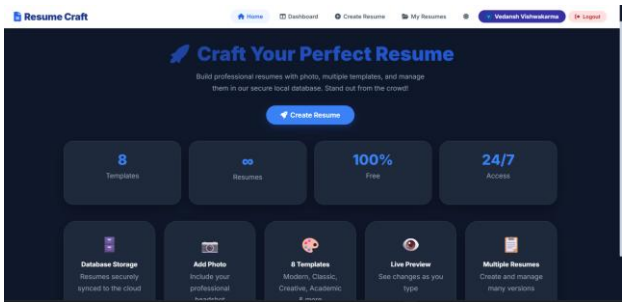
Future development of Resume Craft should focus on integrating intelligent, scalable, and ATS-compliant features, including :

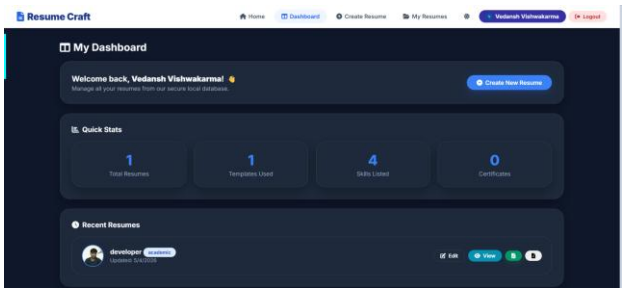
- **Vector-Based PDF Compilation:** Replacing html2pdf.js with server-side generation (e.g., Puppeteer) or native client-side PDF array construction (e.g., pdfmake) to ensure absolute preservation of selectable text and clickable hyperlinks.
- **AI-Powered Content Optimization:** Integrating Large Language Models (LLMs) to actively assist users in optimizing keyword density and drafting metric-driven achievement statements aligned with specific job descriptions.
- **Real-time ATS Scoring Dashboard:** Implementing a live analysis tool to evaluate the resume against standard ATS parameters and deliver intelligent recommendations

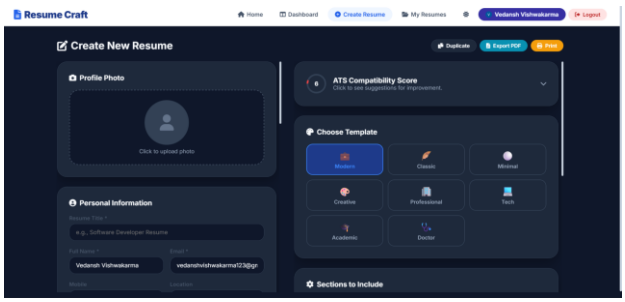
• for improvement before the user downloads the document.

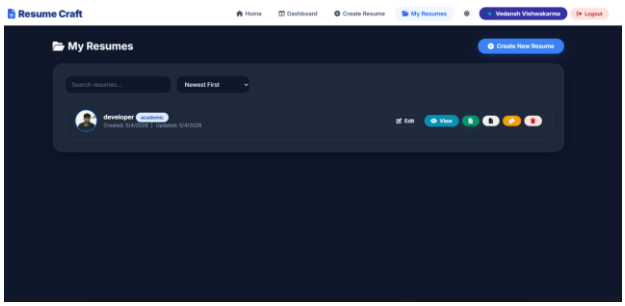
• **Enhanced Security Integrations:** Deploying Firebase App Check to secure backend infrastructure from unauthorized bot traffic and billing fraud.

Web Output









ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to **Mrs. Sweta Kriplani**, Professor, Department of Computer Science and Engineering, Shri Ram Institute of Technology, Jabalpur, Madhya Pradesh for her valuable guidance, continuous support, and constructive feedback throughout the development of this research work.

REFERENCES

1. Jobscan, "Jobscan's 2025 Fortune 500 ATS Report," Jobscan, 2025. [Online]. Available: https://www.jobscan.co/blog/fortune-500-use-applicant-tracking-systems/

2. E. van Inwegen, Z. Munyikwa, and J. J. Horton, "Algorithmic Writing Assistance on Jobseekers' Resumes Increases Hires," MIT Sloan, 2021.

3. R. V. K. Bevara et al., "Resume2Vec: Transforming applicant tracking systems with intelligent resume embeddings for precise candidate matching," *Electronics*, vol. 14, 2025.

4. Nutrient, "HTML to PDF in JavaScript: Five libraries compared," Nutrient Blog, 2026. [Online]. Available: https://www.nutrient.io/blog/html-to-pdf-in-javascript/

5. V. Jain, "Server-Side Rendering vs. Client-Side Rendering: A Comprehensive Analysis," *International Journal of Innovative Research and Creative Technology*, vol. 7, no. 2, 2021.

6. D. Hamzic, M. Wurzenberger, F. Skopik, M. Landauer, and A. Rauber, "Evaluation and comparison of open-source llms using natural language generation quality metrics," in *Proc. 2024 IEEE International Conference on Big Data (BigData)*, 2024, pp. 5342–5351.

7. A. K. Sinha, M. A. K. Akhtar, and A. Kumar, "Resume screening using natural language processing and machine learning: A systematic review," in *Machine Learning and Information Processing*, Springer Singapore, 2021, pp. 207–214.

8. Nagarajan S, Manikandan A, Rahul A, Nidish M, and Balaganesan M, "AI-Powered Resume Builder with ATS Optimization Using Natural Language Processing and Generative AI," *International Journal of Innovative Research in Technology*, vol. 12, no. 11, pp. 9067–9071, 2026.

9. Meta Platforms Inc., "React Official Documentation," 2026. [Online]. Available: https://react.dev/

10.Google, "Firebase Documentation," 2026. [Online]. Available: https://firebase.google.com/docs

11.iTrobes, "Mobile App Development Lifecycle," 2025. [Online]. Available: https://itrobes.com/mobile-app-development-lifecycle/

12.Resume Craft Site: https://resume-craft-fde46.web.app/