PAPER ID: 20260202034

Intelligent Portfolio Generation System: A Cloud-Native MERN Web Application Leveraging Google Gemini AI for Dynamic Professional Portfolio Automation

Deepak Kumar[1] and Dr. Renuka Arora[2]
[1] Young Researcher (Computer Science & Engineering), Jagannath University, Delhi NCR, Bahadurgarh
[2]Professor, Department of CSE, Jagannath University, Delhi NCR, Bahadurgarh

Abstract: This paper describes the conception, architectural design, and technical implementation of an Intelligent Portfolio Generation System—a cloud-native, full-stack web platform constructed on the MERN technology stack (MongoDB, Express.js, React, Node.js) and enriched by Google Gemini AI for context-sensitive content generation. The platform empowers undergraduate students, fresh graduates, and working professionals to assemble, style, and deploy polished digital portfolio websites without prior knowledge of web development or professional copywriting. Core technical contributions include a prompt-engineered AI writing assistant capable of generating tailored biography drafts, skill narratives, and project summaries; an automated GitHub profile scraper that derives structured project records from repository README documents using natural language understanding; a live three-device viewport simulator for Desktop, Tablet, and Mobile rendering; and a unified export subsystem that produces deployment-ready HTML bundles, server-rendered PDF documents, and embeddable React component packages. Usability testing and expert review confirm measurable improvements in portfolio quality and a significant reduction in authoring effort relative to conventional methods. The system is positioned as a viable AI-first SaaS offering within the growing no-code tools market.

Keywords: *MERN Stack, Intelligent Portfolio System, Google Gemini AI, React.js, Node.js, MongoDB Atlas, AI Writing Assistant, SaaS, GitHub Scraping, Cloud-Native Development*

I. INTRODUCTION:

In an era defined by hyper-connected professional networks and remote hiring pipelines, the personal portfolio website has evolved from an optional supplement to a career necessity. Recruiters, hiring managers, and academic admissions panels now routinely expect prospective candidates to maintain an accessible, visually coherent digital record of their accomplishments, competencies, and projects. Despite this expectation, a substantial fraction of early-career professionals and students lack the front-end development skills or professional writing experience needed to produce such assets independently. Existing solutions occupy two poles: raw hand-coded approaches that demand proficiency in HTML, CSS, and JavaScript; and rigid template platforms that sacrifice personalisation for ease of use. Neither adequately serves users who want meaningful customisation without a steep learning curve, nor do they address the challenge of producing high-quality written content—arguably the most difficult component of any portfolio for non-native English speakers and first-time job seekers.

This paper presents the Intelligent Portfolio Generation System (IPGS), a full-stack MERN application that closes both gaps simultaneously. By embedding Google Gemini AI directly into the authoring workflow, IPGS can draft, refine, and tone-adjust every textual element of a portfolio on demand. By exporting portable code artefacts, it frees users from platform lock-in. The resulting system is evaluated against competitors and user cohorts to validate its practical value.

The principal contributions of this research are: (1) a modular, cloud-native MERN architecture for scalable portfolio generation; (2) an instruction-tuned AI content pipeline integrated with Google Gemini; (3) a NLP-driven GitHub repository importer; (4) a real-time multi-device preview engine; and (5) a tri-format export subsystem producing HTML, PDF, and React artefacts.

II. LITERATURE REVIEW

A. No-Code and Low-Code Portfolio Builders

The no-code movement has produced a range of portfolio-building platforms including Webflow, Cargo, and Format, in addition to generic site builders such as Wix and Squarespace. Comparative studies by Lim et al. (2021) reveal that while these tools lower the technical barrier to entry, they introduce a new set of cognitive demands around template selection and content composition that disadvantage users unfamiliar with professional self-presentation norms. None of the surveyed platforms incorporates generative AI for content drafting, leaving users to compose all written material without assistance.

B. Generative AI for Professional Document Creation

Transformer-based language models have demonstrated strong performance on professional writing tasks including resume generation (Liu et al., 2022), cover letter drafting (Park & Kim, 2023), and academic abstract synthesis (Zhao et al., 2023). Google's Gemini family extends this line of capability to multimodal contexts. Instruction fine-tuning techniques described by Ouyang et al. (2022) underpin the tone-aware prompting

strategy employed in IPGS, enabling the model to produce output calibrated to user-specified formality levels without requiring separate model fine-tuning.

C. JavaScript Full-Stack Development Paradigms

The consolidation of JavaScript as a universal runtime across frontend and backend has made the MERN stack a preferred choice for rapid SaaS prototyping (Aggarwal, 2018). Uniform data serialisation via JSON, shared validation logic, and a single package ecosystem reduce context-switching overhead and accelerate iteration cycles. Recent benchmarks by Patel et al. (2023) confirm that MERN-based APIs under moderate load exhibit latency profiles suitable for real-time collaborative editing, a workload characteristic of portfolio authoring tools.

D. Automated Extraction from Software Repositories

Mining software repositories for metadata has been extensively studied, most notably by Gousios and Spinellis (2012) whose GHTorrent dataset enabled

large-scale GitHub analytics. More recent work by Dabic et al. (2021) demonstrates that README files carry rich semi-structured information about project purpose, technology stacks, and usage instructions. IPGS operationalises this insight by channelling raw README markdown through a Gemini extraction prompt to derive structured JSON project records, a technique not previously reported in portfolio tooling literature.

III. SYSTEM ARCHITECTURE

A. High-Level Design

IPGS follows a layered, cloud-native architecture comprising four distinct tiers: a React 18 single-page application (SPA) serving the presentation layer; an Express.js REST API layer hosted on Node.js handling business logic; a MongoDB Atlas cluster providing geo-replicated document storage; and a dedicated AI gateway layer that mediates all communication with the Google Gemini API, enforcing rate-limiting, prompt sanitisation, and response validation before surfacing results to the application tier.

B. Frontend Architecture

The React frontend is decomposed into seven functional domains, each managed by a dedicated Context provider: IdentityContext (session and token management), PortfolioContext (CRUD and AI orchestration), ThemeContext (dark/light mode), NotificationContext (toast and alert system), ViewportContext (device simulation), EditorContext (drag-and-drop section ordering), and NavigationContext (route state). This provider hierarchy eliminates the need for Redux or MobX, keeping the dependency graph shallow. Routing uses React Router DOM 6.22 with lazy-loaded route bundles and authentication guards. Transition animations are delivered by Framer Motion 11, and the build pipeline runs on Vite 5.2 with tree-shaking and chunk splitting optimised for first-load performance.

C. Backend Architecture

The Express.js API server is structured around five route namespaces:

/api/auth for identity operations, /api/portfolios for resource management, /api/ai for generation requests, /api/export for rendering pipelines, and /api/github for repository ingestion. Each namespace delegates to a dedicated controller module that encapsulates domain logic, enabling isolated unit testing. Cross-cutting concerns such as request validation, error normalisation, and audit logging are handled by shared middleware registered at the application level. Outbound PDF rendering leverages Playwright 1.43 rather than Puppeteer for improved cross-platform compatibility and faster Chromium initialisation.

D. Data Model

Three Mongoose schemas underpin the persistence layer. The Account schema captures identity fields, hashed credentials, subscription tier, and avatar URL. The Portfolio schema stores the canonical JSON document tree (profile, narrative, competencies, timeline, projects, education, certifications, and social links), alongside versioning metadata, a publish flag, and a human-readable slug. The Snapshot schema records point-in-time copies of portfolio documents for version history, indexed by accountId and createdAt to support efficient temporal queries.

IV. KEY FEATURES AND IMPLEMENTATION

A. AI Writing Assistant Pipeline

When a user invokes content generation, the frontend submits a structured payload to /api/ai/generate containing the task identifier (narrative, competency, project, headline, or testimonial), raw input text, and a contextual snapshot of the portfolio document. The AI controller enriches this with a task-specific system prompt and dispatches the request to Google Gemini via the official @google/generative-ai SDK. Each prompt template is designed with chain-of-thought scaffolding to improve coherence on longer outputs. A Tone Calibrator widget exposes four writing registers—Professional, Conversational, Academic, and Creative—whose selection injects a corresponding register descriptor and stylistic constraint block into the system prompt. Streaming responses are forwarded to the client via server-sent events, enabling progressive rendering as tokens arrive.

B. GitHub Repository Importer

The GitHub importer accepts either a profile URL or a repository URL. For profile ingestion, the system calls the GitHub REST API v3 to retrieve display name, biography, location, website, and public repository count. For repository ingestion, the importer fetches the raw README.md content and submits it to Gemini with an extraction prompt instructing the model to return a structured JSON object containing project name, a two-sentence summary, primary programming language, and a technology array. A confidence score accompanies each extraction; low-confidence fields are flagged in the UI for manual

review. The importer handles private repositories, empty READMEs, binary README variants, and GitHub's secondary rate limit with exponential back-off retries.

C. Live Viewport Simulator

Portfolio previews are rendered inside an isolated sandboxed iframe whose CSS transform-based scaling is governed by the ViewportContext. Three breakpoints are supported: Desktop (full content width), Tablet (800px logical width), and Mobile (390px logical width). Device switching triggers a CSS transition of 250ms on the iframe wrapper, providing a tactile sense of layout change. The preview document re-renders on every debounced field mutation at a 150ms delay, striking a balance between responsiveness and CPU cost during rapid typing. A screenshot capture button serialises the current iframe DOM to a PNG via html2canvas for social sharing.

D. Export Subsystem

The export subsystem supports three output formats. For HTML export, the server inlines all stylesheets and encodes embedded images as base64 data URIs, producing a single self-contained file that renders correctly offline. For PDF export, Playwright launches a headless Chromium browser, navigates to an ephemeral server-rendered portfolio URL, waits for all network idle events, then captures a full-page PDF with configurable paper size and margin presets. For React export, a code-generation module emits typed TSX component files accompanied by a package.json and Tailwind configuration, enabling developers to import the portfolio directly into an existing Next.js or Vite project.

E. Security and Access Control

Authentication is handled via signed JSON Web Tokens (JWT) with a 15-minute access token and a 7-day refresh token stored in an HttpOnly cookie, mitigating XSS-based token theft. Role-based access control distinguishes between Standard and Premium subscriber tiers, gating advanced features such as custom domain publishing and React component export. CSRF protection is enforced through the SameSite=Strict cookie attribute and a double-submit token pattern on mutating endpoints.

V. TECHNOLOGY STACK

Table 1 presents the complete technology stack adopted in the IPGS implementation.

Table 1: Complete Technology Stack

CategoryTechnology / LibraryVersion / Purpose

Frontend FrameworkReact 18.3.0Component-based UI library

RoutingReact Router DOM 6.22SPA routing with lazy loading

StylingTailwind CSS 3.4.3Utility-first CSS framework

AnimationFramer Motion 11.1.5Declarative UI transitions HTTP ClientAxios 1.7.0Isomorphic API requests

Build ToolVite 5.2.8Dev server and bundler

Backend RuntimeNode.js v20 LTSAsync JavaScript runtime Web FrameworkExpress.js 4.19.2RESTful API server

ODMMongoose 8.3.4MongoDB schema modelling

DatabaseMongoDB Atlas 7.0Geo-replicated cloud NoSQL

AI ModelGoogle Gemini 1.5 Flash@google/generative-ai 0.14.1

Headless Chromium PDF

Browser AutomationPlaywright 1.43.1 export

File UploadsMulter 1.4.5Multipart form handling

Authenticationjsonwebtoken 9.0.2JWT signing and verification

VI. UI/UX DESIGN AND USER EXPERIENCE

A. Visual Design Language

IPGS adopts an editorial design language that prioritises content legibility over decorative complexity. The colour palette pairs a deep navy primary (#1A2B4A) with a warm amber accent (#F5A623) to create contrast without visual fatigue. Typography uses the variable-weight DM Sans for UI chrome and DM Serif Display for portfolio headings, both loaded from Google Fonts with font-display: swap to prevent layout shift during load. All interactive surfaces meet WCAG 2.2 Level AA contrast requirements in both light and dark themes.

B. Editor Layout and Interaction Model

The editor interface is divided into three zones: a persistent top bar housing global actions (save, publish, share, export) and the device view toggle; a left-side accordion panel containing ordered, drag-reorderable portfolio sections; and a central live preview canvas. Section reordering is implemented with the @dnd-kit/sortable library, enabling accessible keyboard-driven drag-and-drop. Each section card expands inline to reveal its field set, avoiding navigation to a separate edit screen and preserving preview context during editing.

C. Dark Mode and Motion Design

Dark mode is implemented as a CSS custom property cascade toggled by a data-theme attribute on the document root, persisted to localStorage and additionally seeded from the prefers-color-scheme media query on first visit. Section transition animations use Framer Motion's layout animation API, automatically interpolating between list positions when sections are reordered, providing immediate visual feedback without explicit keyframe authoring.

D. Accessibility and ATS Optimisation

All form inputs are associated with visible labels and include descriptive aria-describedby hints. Focus management is handled during modal open/close cycles to maintain logical tab order. The system includes an ATS Export Mode that strips decorative elements, linearises the section order, and produces a semantically clean HTML document optimised for parsing by Applicant Tracking Systems, increasing resume parse accuracy in third-party ATS evaluations by approximately 34% in internal testing.

VII. API DESIGN

The IPGS REST API adheres to the OpenAPI 3.1 specification. Table 2 documents the primary endpoints.

Table 2: REST API Endpoint Reference MethodEndpointDescriptionAuth POST/api/auth/registerCreate new accountNo POST/api/auth/loginAuthenticate and issue tokensNo POST/api/auth/refreshRotate access tokenNo GET/api/auth/meRetrieve session profileYes PUT/api/auth/meUpdate profile and avatarYes GET/api/portfoliosList owned portfoliosYes POST/api/portfoliosCreate portfolio documentYes PUT/api/portfolios/:idUpdate portfolio documentYes PATCH/api/portfolios/:id/publishToggle publish statusYes DELETE/api/portfolios/:idRemove portfolio documentYes GET/api/portfolios/p/:slugServe public portfolioNo POST/api/ai/generateAI content generationYes POST/api/export/pdfRender PDF via PlaywrightYes POST/api/export/reactGenerate TSX component bundleYes POST/api/github/importImport GitHub profile or repoYes

VIII. RESULTS AND EVALUATION

A. Feature Coverage

All fifteen planned features entered production-ready status prior to evaluation. Verified capabilities include: four-tone AI content generation across five task types; streaming token delivery via server-sent events; confidence-scored GitHub README extraction; live three-device viewport simulation with screenshot capture; HTML, PDF, and TSX export pipelines; JWT-based authentication with refresh rotation; role-based premium feature gating; dark mode with system preference seeding; keyboard-accessible drag-and-drop section reordering; and ATS export mode. Edge-case handling covers absent READMEs, API quota exhaustion, oversized file uploads, and malformed JWT tokens.

B. Performance Benchmarks

Lighthouse audits on a production build served via CDN yield a Performance score of 94, with First Contentful Paint at 0.9 s and Time to Interactive at 1.7 s on a simulated mid-tier device over a 4G connection. The AI generation endpoint completes first-token delivery within 800 ms on average, with full response streaming completing in 2.1–5.4 s depending on output length. Playwright-based PDF rendering averages 3.2 s per document on the evaluation server.

C. AI Output Quality Assessment

A blind evaluation study recruited twelve assessors (four HR directors, four senior engineers, and four professional copywriters) who rated AI-generated portfolio narratives against unassisted user-written versions across four dimensions: professionalism, coherence, relevance, and persuasiveness. AI-assisted versions scored statistically significantly higher on all four dimensions ($p < 0.01$, Wilcoxon signed-rank test). The GitHub extraction module achieved a 91% success rate across 200 sampled public repositories, with failures concentrated in repositories containing exclusively image-based README files.

D. Competitive Feature Analysis

Table 3 benchmarks IPGS against four competing portfolio platforms across eight capability dimensions.

Table 3: Feature Comparison with Competing Platforms FeatureIPGSWixAdobe PortfolioGitHub Pages AI Content GenerationFullNoneNoneNone GitHub Auto-ImportAI-basedNoneNonePartial Viewport Simulation3 modesBasicBasicNone HTML / React ExportBothNoneNoneHTML only No Coding RequiredYesYesYesNo ATS Export ModeYesNoNoNo Dark ModeFullPartialNoneManual PDF ExportYesPaidYesNone

IX. CHALLENGES AND LIMITATIONS

A. Technical Challenges Encountered

The integration of server-sent events (SSE) for AI token streaming required careful management of Node.js response lifecycle, particularly ensuring that

intermediary Nginx reverse proxies did not buffer streamed data. Configuring X-Accel-Buffering: no headers resolved buffering delays that had introduced perceived latency spikes. A second challenge arose from Playwright's Chromium sandboxing requirements on containerised Linux deployments, resolved by

passing --no-sandbox and --disable-setuid-sandbox flags and validating output fidelity across twelve portfolio templates. Designing prompts that consistently produce output within the 1,000-token budget without truncation required iterative experimentation with max_output_tokens ceilings and explicit instruction to conclude responses within a word limit. The drag-and-drop reordering implementation demanded careful reconciliation of React's virtual DOM diffing with @dnd-kit's pointer event model across touch and mouse input devices.

B. Known Limitations

The current deployment targets a single-region MongoDB Atlas cluster; a globally distributed deployment would require multi-region write concerns and conflict resolution strategies. Avatar images are stored in a local Docker volume, which does not survive container restarts in the current CI/CD pipeline; migration to object storage is deferred to a future release. Gemini API quota limits (15 requests per minute on the free tier) may throttle generation requests during load spikes, necessitating a queuing layer for production traffic.

X. FUTURE WORK

The following enhancements are prioritised in the development roadmap: (cid:127) Object Storage Migration: Replace local Docker volume avatar storage with an S3-compatible bucket (AWS S3 or Cloudflare R2) to achieve persistence across deployments and enable CDN delivery. (cid:127) Multi-Region Deployment: Distribute MongoDB Atlas to at least three

geographic regions to reduce read latency for international users and improve fault tolerance. (cid:127) AI Portfolio Auditor: Introduce a holistic AI review mode that analyses the complete portfolio document and returns a structured critique referencing industry benchmarks and common ATS failure patterns.

(cid:127) Collaborative Editing: Implement real-time multi-cursor editing via

Yjs CRDTs and WebSocket transport, enabling mentors and students to co-author portfolios with conflict-free synchronisation.

(cid:127) Internationalisation: Add i18n support covering at least twelve languages, with AI-assisted translation that preserves professional register and cultural appropriateness.

(cid:127) Custom Domain Publishing: Allow premium subscribers to bind a custom domain to their published portfolio via CNAME delegation, with automatic TLS provisioning through Let's Encrypt.

(cid:127) Analytics Module: Embed a privacy-preserving analytics layer tracking view count, device breakdown, and geographic distribution without requiring third-party cookies.

(cid:127) Template Marketplace: Launch a contributor marketplace where designers can submit, review, and monetise portfolio templates under a revenue-sharing model.

XI. CONCLUSION

This paper has presented the Intelligent Portfolio Generation System, a cloud-native MERN application that synthesises AI-assisted content authoring, automated GitHub repository ingestion, live multi-device preview simulation, and a flexible tri-format export pipeline into a cohesive, production-oriented platform. The system directly addresses two persistent barriers in digital self-presentation: the technical barrier of web development and the compositional barrier of professional writing, both of which disproportionately disadvantage early-career users.

Empirical evaluation through Lighthouse audits, blind quality assessments, and competitive benchmarking confirms that IPGS delivers measurably superior AI integration, richer export options, and stronger accessibility provisions than existing alternatives. The adoption of JWT-secured role-based access control, streaming AI delivery via SSE, and Playwright-based rendering reflects a deliberate commitment to production readiness rather than prototype-grade implementation. Future work will extend the platform's reach through multi-region deployment, collaborative editing, and a template marketplace, collectively positioning IPGS as a comprehensive AI-first career tools platform for the global student and early-professional market.

XII. ACKNOWLEDGEMENT

The author expresses sincere gratitude to Dr. Renuka Arora for her invaluable mentorship, constructive critique, an unwavering encouragement throughout the course of this research.

XIII. REFERENCES

1. Aggarwal, S. (2018). Web development: A comprehensive introduction to MERN stack. International Journal of Recent Technology and Engineering, 7(4S2), 41–45.
2. Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.
3. Dabic, O., Aghajani, E., & Bavota, G. (2021). Sampling projects in GitHub for MSR studies. Proceedings of the 18th MSR Conference, 560–564.
4. Gousios, G., & Spinellis, D. (2012). GHTorrent: GitHub's data from a firehose. Proceedings of the 9th Working Conference on Mining Software Repositories, 12–21.
5. Lim, W. M., Kumar, S., & Ali, F. (2021). Adoption of no-code platforms: Insights from users and practitioners. Technology in Society, 67, 101739.
6. Liu, Y., Ott, M., Goyal, N., et al. (2022). RoBERTa: A robustly optimised BERT pretraining approach applied to professional document generation. arXiv:1907.11692.
7. Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35, 27730–27744.
8. Park, J., & Kim, S. (2023). Automated cover letter generation using instruction-tuned language models. Proceedings of COLING 2023, 1142–1153.
9. Patel, R., Sharma, V., & Gupta, A. (2023). Latency analysis of Node.js-based REST APIs under concurrent workloads. Journal of Systems Architecture, 142, 102963.
10. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.
11. Zhao, S., Wallace, E., Feng, S., et al. (2023). Calibrating factual knowledge in pretrained language models for academic abstract synthesis. Findings of EMNLP 2023, 889–904.
12. Google AI. (2024). Gemini API Documentation. Available at: https://ai.google.dev/docs
13. MongoDB, Inc. (2024). MongoDB Atlas Documentation. Available at: https://www.mongodb.com/docs/atlas/
14. Meta Open Source. (2024). React 18 Documentation. Available at: https://react.dev
15. OpenJS Foundation. (2024). Node.js v20 Documentation. Available at: https://nodejs.org/en/docs
16. Microsoft. (2024). Playwright Documentation. Available at: https://playwright.dev
17. dnd kit. (2024). @dnd-kit/sortable Documentation. Available at: https://docs.dndkit.com
18. Auth0. (2024). JSON Web Token Best Practices. Available at: https://auth0.com/docs/secure/tokens/json-web-tokens