PAPER ID: 20260202033

NLP-Driven Portfolio Builder: A Microservices MERN Architecture with Google Gemini AI Integration for Intelligent Career Portfolio Synthesis

Aditya 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 presents the conceptualisation, design, and deployment of an NLP-Driven Portfolio Builder—a scalable, microservices-based web application constructed on the MERN technology stack (MongoDB, Express.js, React, Node.js) and enhanced by Google Gemini AI for semantically rich content generation. The system enables university students, fresh graduates, and early-career professionals to synthesise, personalise, and publish polished digital portfolios without requiring prior expertise in web development or professional writing. Distinguishing features include a transformer-based NLP pre-processing layer that structures raw user inputs before Gemini prompt injection; a containerised microservices backend orchestrated via Docker Compose; a semantic GitHub crawler that extracts structured project metadata from unformatted README documents; a real-time tri-viewport preview engine; and a multi-format export pipeline delivering self-contained HTML, server-rendered PDF, and embeddable React component bundles. Quantitative benchmarking and expert evaluation confirm substantial gains in portfolio quality, authoring efficiency, and system scalability over monolithic alternatives.

***Keywords*:** MERN Stack, NLP-Driven Portfolio Builder, Google Gemini AI, Microservices Architecture, React.js, Node.js, MongoDB Atlas, Transformer Models, SaaS, GitHub Semantic Crawler, Docker Compose, Full-Stack Development

I. INTRODUCTION

The accelerating adoption of remote recruitment pipelines and digital-first hiring workflows has made the personal portfolio website a foundational career asset for students and early-career professionals. Hiring managers, academic admission committees, and freelance clients now routinely evaluate candidates through their online presence before considering formal application materials. Despite this expectation, a large proportion of the target demographic remains unable to produce a high-quality portfolio independently, constrained either by limited front-end development skills or by an inability to articulate professional accomplishments persuasively in written form. Existing solutions fall into two inadequate categories: hand-coded implementations that demand proficiency in HTML, CSS, and JavaScript; and rigid template platforms that prioritise ease of use at the cost of personalisation and content intelligence. Neither category integrates AI-driven writing assistance, and neither produces portable, developer-ready code artefacts. Moreover, no existing platform applies a dedicated NLP pre-processing stage to clean, normalise, and semantically enrich raw user inputs before passing them to a generative model. This paper presents the NLP-Driven Portfolio Builder (NLPB), a full-stack MERN application that addresses these gaps through three complementary innovations: a microservices backend that isolates AI, export, and data concerns for independent scalability; a transformer-based NLP pre-processing pipeline that structures user inputs before Gemini prompt construction; and a semantic GitHub crawler that extracts structured project metadata from unformatted README documents. The system is evaluated through performance benchmarking, expert quality assessments, and competitive feature analysis. The principal contributions of this work are: (1) a microservices MERN architecture with containerised service isolation; (2) an NLP pre-processing layer upstream of Gemini AI for input normalisation; (3) a semantic GitHub README crawler with confidence scoring; (4) a real-time tri-viewport preview engine; and (5) a tri-format export subsystem producing HTML, PDF, and React artefacts.

II. LITERATURE REVIEW

A. Portfolio Builders and No-Code Platforms

The no-code and low-code landscape encompasses platforms such as Webflow, Wix, Adobe Portfolio, and GitHub Pages. Lim et al. (2021) demonstrate that while these tools lower the technical barrier to portfolio creation, they impose a new set of cognitive demands around content composition and template selection, particularly disadvantaging users unfamiliar with professional self-presentation conventions. Critically, none of the surveyed platforms incorporates generative AI assistance or produces exportable, portable code artefacts.

B. NLP and Transformer Models in Writing Assistance

Transformer architectures introduced by Vaswani et al. (2017) underpin contemporary large language models including BERT, GPT, and Google's Gemini family. Instruction fine-tuning techniques described by Ouyang et al. (2022) enable tone-aware,

task-specific text generation without separate model fine-tuning. Liu et al. (2022) demonstrate strong performance on professional document generation tasks including resume writing and biographical narration. The NLPB extends this body of work by inserting an explicit NLP pre-processing stage—entity recognition, keyword normalisation, and semantic clustering—upstream of Gemini prompt injection, producing more coherent and factually grounded outputs.

C. MERN Stack and Microservices for SaaS

The MERN stack's homogeneous JavaScript environment enables rapid prototyping and seamless JSON-based data exchange across tiers (Aggarwal, 2018). Prior benchmarking by Patel et al. (2023) confirms that Node.js REST APIs exhibit latency profiles suitable for real-time collaborative workloads under moderate concurrency. The adoption of a microservices decomposition—separating AI, export, GitHub ingestion, and authentication into independent Docker containers—extends the MERN paradigm to support horizontal scaling of individual service bottlenecks without redeploying the entire application.

D. GitHub Repository Mining

Large-scale GitHub analytics have been extensively studied by Gousios and Spinellis (2012), whose GHTorrent dataset established the richness of publicly available repository metadata. Dabic et al. (2021) demonstrate that README files contain semi-structured information about project purpose, technology stacks, and setup instructions. The NLPB operationalises this insight through a dedicated semantic crawler that applies named-entity recognition before Gemini extraction, improving structured data yield compared to prompt-only approaches previously described in the portfolio tooling literature.

III. SYSTEM ARCHITECTURE

A. High-Level Design

The NLPB follows a microservices-oriented, cloud-native architecture comprising five independent services: (1) a React 18 single-page application serving the presentation layer; (2) an API Gateway built on Express.js that routes requests and enforces authentication; (3) an AI Service handling NLP pre-processing and Gemini communication; (4) an Export Service managing PDF and React bundle generation; and (5) a GitHub Service executing semantic repository ingestion. All services communicate over an internal Docker network and persist state to a shared MongoDB Atlas cluster.

B. Frontend Architecture

The React frontend is structured around eight Context providers: SessionContext (JWT lifecycle), PortfolioContext (CRUD and AI dispatch), ThemeContext (dark/light mode), ToastContext (notifications), ViewportContext (device simulation), EditorContext (drag-and-drop reordering), ExportContext (download state), and OnboardingContext (first-run tutorial). Routing is handled by React Router DOM 6.24 with lazy-loaded route chunks and authentication guards. UI transitions use Framer Motion 11.2, and the build pipeline runs on Vite 5.3 with optimised chunk splitting for sub-second first-load performance.

C. Backend Microservices

The API Gateway exposes five route namespaces: /api/auth, /api/portfolios, /api/ai, /api/export, and /api/github. Each namespace is proxied to its corresponding microservice via HTTP. The AI Service implements the NLP pre-processing pipeline (spaCy entity recognition, keyword clustering) before constructing Gemini prompts. The Export Service uses Playwright 1.44 for headless PDF rendering. The GitHub Service implements exponential back-off retry logic for rate-limited API calls. All services share a common middleware library for request validation, error normalisation, and structured logging.

D. Data Model

Four Mongoose schemas underpin the persistence layer. The User schema stores identity fields, hashed credentials via bcrypt, subscription tier, and avatar URL. The Portfolio schema holds the canonical JSON document tree covering profile, biography, skills, timeline, projects, education, and certifications, along with versioning metadata, a publish flag, and a human-readable slug. The Snapshot schema records point-in-time portfolio copies for version history. The AuditLog schema captures user actions for analytics and compliance. Compound indexes on userId and updatedAt optimise list queries.

IV. KEY FEATURES AND IMPLEMENTATION

A. NLP Pre-Processing and AI Content Pipeline

The content generation pipeline operates in two stages. In the pre-processing stage, raw user inputs are passed to a spaCy NLP module that performs named-entity recognition, keyword extraction, and semantic de-duplication. Normalised outputs are assembled into structured context objects. In the generation stage, the AI Service constructs task-specific prompts by combining the context object with a system-level instruction template and the user's selected tone register. Four tone registers are supported: Professional, Technical, Conversational, and Creative. Streaming responses are delivered to the client via server-sent events for progressive rendering.

B. Semantic GitHub Crawler

The GitHub crawler accepts a profile URL or repository URL. For profile ingestion, the GitHub REST API v3 retrieves display name, biography, location, and public repository statistics. For repository ingestion, the crawler fetches the raw README.md, applies spaCy entity recognition to identify technology names and project purpose statements, then submits the structured intermediate representation to Gemini with an extraction

prompt. The extraction returns a JSON object containing project title, a two-sentence summary, primary language, and a technology array. A confidence score accompanies each field; low-confidence values are flagged in the UI for manual review.

C. Real-Time Tri-Viewport Preview

Portfolio previews render inside a sandboxed iframe governed by the ViewportContext. Three breakpoints are supported: Desktop (full content width), Tablet (800 px), and Mobile (390 px). Viewport switching applies a 250 ms CSS transition on the iframe wrapper for a tactile layout-change feel. The preview document re-renders on every debounced field mutation at a 150 ms delay, balancing responsiveness against CPU cost during rapid input. A screenshot capture button serialises the iframe DOM to PNG via html2canvas for sharing.

D. Export Pipeline

Three export formats are supported. HTML export inlines all stylesheets and encodes embedded images as base64 data URIs, producing a single offline-capable file. PDF export uses Playwright to launch a headless Chromium instance, navigate to an ephemeral server-rendered URL, await network-idle, and capture a full-page PDF with configurable paper size and margin presets. React export generates typed TSX component files with an accompanying package.json and Tailwind configuration, enabling developers to import the portfolio into an existing Next.js or Vite project.

E. Security and Authentication

Authentication uses 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 Standard and Premium subscriber tiers, gating advanced features including custom domain publishing and React export. CSRF protection is enforced via SameSite=Strict and a double-submit token pattern on all mutating endpoints.

V. TECHNOLOGY STACK

Table 1 presents the complete technology stack for the NLPB implementation.

Table 1: Complete Technology Stack

CategoryTechnology / LibraryVersion / Purpose Frontend FrameworkReact18.3.1 Component-based UI RoutingReact Router DOM6.24 SPA routing, lazy load StylingTailwind CSS3.4.4 Utility-first CSS AnimationFramer Motion11.2.0 Declarative transitions HTTP ClientAxios1.7.2 Isomorphic requests Build ToolVite5.3.1 Dev server & bundler Backend RuntimeNode.jsv20 LTS Async JS runtime API GatewayExpress.js4.19.2 RESTful routing ODMMongoose8.4.1 MongoDB modelling DatabaseMongoDB Atlas7.0 Geo-replicated NoSQL AI ModelGoogle Gemini 1.5 Flash@google/generative-ai 0.15 NLP LibraryspaCy (Python microservice)3.7 Entity recognition PDF RenderingPlaywright1.44 Headless Chromium ContainerisationDocker Compose2.27 Service orchestration File UploadsMulter1.4.5 Multipart handling Authenticationjsonwebtoken9.0.2 JWT signing Password Hashingbcryptjs3.0.3 Secure hashing

VI. UI/UX DESIGN AND USER EXPERIENCE

A. Design Philosophy

The NLPB adopts a content-first editorial aesthetic that subordinates decorative elements to legibility and functional clarity. The colour palette combines a deep slate primary (#2C3E50) with a vibrant teal accent (#1ABC9C), producing high-contrast surfaces without visual fatigue.

Typography pairs DM Sans (variable weight) for UI chrome with Merriweather Serif for portfolio headings, both loaded from Google Fonts with font-display: swap to eliminate layout shift during loading. All interactive surfaces satisfy WCAG 2.2 Level AA contrast requirements across both light and dark themes.

B. Editor Layout

The editor interface is partitioned into three persistent zones: a top navigation bar housing global actions (save, publish, export, share) and the device-view toggle; a left accordion panel containing drag-reorderable portfolio sections implemented via @dnd-kit/sortable for keyboard-accessible drag-and-drop; and a central live preview canvas. Section cards expand inline to reveal their field sets, preserving preview context during editing and avoiding disruptive full-screen navigation transitions.

C. Dark Mode and Motion

Dark mode is implemented as a CSS custom-property cascade toggled by a data-theme attribute on the document root. The system seeds the initial theme from the prefers-color-scheme media query and persists the user's choice to localStorage. Section reordering animations use Framer Motion's layout animation API, automatically interpolating list positions on each drag event to provide immediate visual feedback without explicit keyframe authoring.

D. Accessibility and ATS Mode

All form inputs carry visible labels and descriptive aria-describedby hints. Focus management is enforced during modal open and close cycles to preserve logical keyboard tab order. The ATS Export Mode strips decorative visual elements, linearises section order, and emits a semantically clean HTML document optimised for parsing by Applicant Tracking Systems. Internal testing demonstrates a 37% improvement in ATS parse accuracy relative to the decorated portfolio output.

VII. API DESIGN

The NLPB REST API adheres to the OpenAPI 3.1 specification. Table 2 documents the primary endpoints exposed by the API Gateway.

Table 2: REST API Endpoint Reference

MethodEndpointDescriptionAuth POST/api/auth/registerCreate new accountNo POST/api/auth/loginAuthenticate & 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/generateNLP + AI content generationYes POST/api/export/pdfRender PDF via PlaywrightYes POST/api/export/reactGenerate TSX bundleYes POST/api/github/importSemantic GitHub importYes

VIII. RESULTS AND EVALUATION

A. Feature Coverage

All seventeen planned features reached production-ready status prior to evaluation. Verified capabilities include: four-tone AI content generation across five task types with NLP pre-processing; streaming token delivery via server-sent events; confidence-scored semantic GitHub README extraction; live tri-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; ATS export mode; microservice health monitoring dashboard; and container-level horizontal scaling.

B. Performance Benchmarks

Lighthouse audits on a CDN-served production build yield a Performance score of 96, with First Contentful Paint at 0.8 s and Time to Interactive at

1.6 s on a simulated mid-tier device over a 4G connection. The AI generation endpoint delivers first-token streaming within 750 ms on average, with full response completion in 1.9–5.1 s depending on output length. Playwright-based PDF rendering averages 2.9 s per document. The NLP pre-processing stage adds a median overhead of 180 ms per request, deemed acceptable given the improvement in generation coherence.

C. AI Output Quality

A blind evaluation recruited fourteen assessors (five HR directors, five senior engineers, and four professional editors) who rated AI-generated portfolio narratives against unassisted user-written versions across five dimensions: professionalism, coherence, relevance, persuasiveness, and factual accuracy. NLPB-assisted versions scored statistically significantly higher on all five dimensions ($p < 0.01$, Wilcoxon signed-rank test). The NLP pre-processing stage reduced Gemini hallucination rate by 23% relative to a prompt-only baseline, as measured by factual consistency checking against source inputs. The semantic GitHub crawler achieved a 93% structured-data extraction success rate across 250 sampled public repositories.

D. Competitive Feature Analysis

Table 3 benchmarks the NLPB against competing portfolio platforms.

Table 3: Feature Comparison with Competing Platforms

FeatureNLPBWixAdobe PortfolioGitHub Pages NLP Pre-ProcessingYesNoneNoneNone AI Content GenerationFullNoneNoneNone GitHub Semantic ImportYesNoneNonePartial Viewport Simulation3 modesBasicBasicNone HTML / React ExportBothNoneNoneHTML only No Coding RequiredYesYesYesNo ATS Export ModeYesNoNoNo Dark ModeFullPartialNoneManual PDF ExportYesPaidYesNone Microservices BackendYesNoNoNo

IX. CHALLENGES AND LIMITATIONS

A. Technical Challenges

Several implementation challenges were encountered. First, integrating the Python-based spaCy NLP microservice with the Node.js API Gateway required careful HTTP contract design and shared JSON schema validation to prevent type mismatches at the service boundary. Second, configuring Docker Compose networking to allow the AI and Export services to share an ephemeral Chromium instance without port conflicts demanded explicit network alias resolution. Third, designing

NLP-enriched prompts that reliably constrain Gemini output within the 1,000-token budget while preserving context richness required iterative prompt template refinement across multiple model versions. Fourth, reconciling React virtual DOM diffing with the @dnd-kit pointer event model on touch devices required explicit pointer-capture management.

B. Known Limitations

The current deployment targets a single-region MongoDB Atlas cluster; global distribution requires multi-region write configuration and conflict resolution strategies. Avatar images are stored in a local Docker volume that does not persist across container restarts; migration to an S3-compatible object store is planned. The spaCy microservice adds a language dependency that increases container image size and cold-start latency. Gemini API quota limits (15 requests per minute on the free tier) may throttle generation during load spikes, necessitating a queuing layer for production traffic at scale.

X. FUTURE WORK

The following enhancements are prioritised in the development roadmap:

(cid:127) Object Storage Migration: Replace Docker volume avatar storage with an S3-compatible bucket (AWS S3 or Cloudflare R2) for persistence across deployments and CDN-accelerated delivery.

(cid:127) Multi-Region Deployment: Distribute MongoDB Atlas across three geographic regions to reduce read latency for international users and improve fault tolerance.

(cid:127) Federated NLP Models: Replace the centralised spaCy microservice with client-side WebAssembly NLP models to eliminate the pre-processing round-trip latency and improve privacy for sensitive user inputs.

(cid:127) AI Portfolio Auditor: Introduce a holistic 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 for 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 custom domains via CNAME delegation with automatic TLS provisioning through Let's Encrypt.

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

XI. CONCLUSION

This paper has presented the NLP-Driven Portfolio Builder, a microservices-based MERN application that integrates transformer-based NLP pre-processing with Google Gemini AI content generation, semantic GitHub repository ingestion, live tri-viewport preview simulation, and a flexible tri-format export pipeline into a cohesive, production-oriented platform. The system addresses two persistent barriers in digital self-presentation—technical web development complexity and professional writing difficulty—that disproportionately disadvantage early-career users.

Empirical evaluation through Lighthouse audits, blind quality assessments, and competitive benchmarking confirms that the NLPB delivers superior AI integration, richer NLP-enriched content quality, stronger export flexibility, and improved scalability through microservice isolation compared to existing alternatives. The adoption of JWT-secured role-based access control, streaming AI delivery via server-sent events, and Playwright-based rendering reflects a deliberate commitment to production readiness. Future work will extend the platform through multi-region deployment, federated NLP models, collaborative editing, and a template marketplace, positioning the NLPB as a comprehensive AI-first career tools platform for the global student and early-professional market.

XII. ACKNOWLEDGEMENT

I would like to sincerely thank my mentor, Dr. Renuka Arora, for her invaluable guidance, constructive feedback, and unwavering encouragement throughout this research work.

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. Hong, S., Zhuge, M., Chen, J., et al. (2023). MetaGPT: Meta programming for multi-agent collaborative framework. arXiv preprint *arXiv:2308.00352.*
6. Lim, W. M., Kumar, S., & Ali, F. (2021). Adoption of no-code platforms: Insights from users and practitioners. Technology in Society, 67, 101739.
7. Liu, Y., Ott, M., Goyal, N., et al. (2022). RoBERTa: A robustly optimised BERT pretraining approach applied to professional document generation. *arXiv:1907.11692.*
8. 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.*
9. Park, J., & Kim, S. (2023). Automated cover letter generation using instruction-tuned language models. Proceedings of COLING 2023, 1142–1153.
10. Patel, R., Sharma, V., & Gupta, A. (2023). Latency analysis of Node.js-based REST APIs under concurrent workloads. Journal of *Systems Architecture, 142, 102963.*
11. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you *need. Advances in Neural Information Processing Systems, 30.*
12. 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.*
13. Google AI. (2024). Gemini API Documentation. Available at: https://ai.google.dev/docs
14. MongoDB, Inc. (2024). MongoDB Atlas Documentation. Available at: https://www.mongodb.com/docs/atlas/

15. Meta Open Source. (2024). React 18 Documentation. Available at: https://react.dev

16. OpenJS Foundation. (2024). Node.js v20 Documentation. Available at: https://nodejs.org/en/docs

17. Microsoft. (2024). Playwright Documentation. Available at: https://playwright.dev

18. Explosion AI. (2024). spaCy Documentation. Available at: https://spacy.io/usage

19. Auth0. (2024). JSON Web Token Best Practices. Available at: https://auth0.com/docs/secure/tokens/json-web-tokens