International Journal of Creative and Open Research in Engineering and Management
ISSN: 3108-1754 (Online)
Volume 02 Issue 09 September-2026 | Impact Factor: 3.5

FormOS: A Unified Platform for Centralized Identity Document Management and Intelligent Web Form Autofill

Tanya Jha
INFT Department
VIT, Wadala
Mumbai, India

Sejal Dhanve
INFT Department
VIT, Wadala
Mumbai, India

Mansi Jadhav
INFT Department
VIT, Wadala
Mumbai, India

Shakir Kalu
INFT Department
VIT, Wadala
Mumbai, India

Prof. Shashikant Mahajan
INFT Department (Guide)
VIT, Wadala
Mumbai, India

How to Cite this Article:
Jha, T., Dhanve, S., Jadhav, M. & Kalu, S. (2026). FormOS: A Unified Platform for Centralized Identity Document Management and Intelligent Web Form Autofill. International Journal of Creative and Open Research in Engineering and Management, 02(9), 1-9. https://doi.org/10.55041/ijcope.v2i9.034

https://doi.org/10.55041/ijcope.v2i9.034

Abstract*—Users of government portals, banking platforms, job boards, and educational institutions routinely re-enter identical identity information across unrelated web forms, a repetitive process that is time-consuming and error-prone. Existing tools address this only partially: password managers and browser-native autofill handle basic contact fields but cannot process identity documents, while repositories such as DigiLocker provide storage without autofill or query capability. This paper presents FormOS, a full-stack platform that lets a user upload identity documents once and subsequently access, query, and deploy that data across the web. FormOS combines an OCR pipeline (Tesseract.js) for structured field extraction, a schema-grounded Retrieval-Augmented Generation (RAG) assistant for natural-language querying, a Form Analyzer using synonym and fuzzy field-matching for autofill-readiness scoring, and a Chrome Man-ifest V3 extension for one-click form completion. Its contribution is the integration of these established components around a single, securely governed personal identity store, rather than novelty in any individual algorithm. The platform is built on Next.js 14, Node.js/Express.js, and MongoDB, with a multi-layered security model comprising JWT authentication, bcrypt-hashed PINs, TOTP-based MFA, and AES-256 field-level encryption. A feasibility analysis spanning technical, economic, operational, and legal dimensions supports real-world viability. As the system is at an early implementation stage, this paper reports a qualitative, architecture- and feasibility-based evaluation with module-level functional validation rather than fabricated performance bench-marks, and identifies empirical validation as future work.

***Index Terms*—Optical Character Recognition, Retrieval-Augmented Generation, Form Autofill, Identity Document Man-agement, Chrome Extension, Data Privacy, Multi-Factor Authen-tication, Web Automation, Systems Integration**

I. Introduction

Nearly every interaction between an individual and a digital institution — opening a bank account, applying for a job, registering for a government scheme, enrolling in a course — begins with the same act: manually re-typing a name, date of birth, address, and identity numbers already entered elsewhere. The underlying data rarely changes; what is missing is a mechanism to read it once and make it available everywhere it is subsequently needed.

FormOS — Form Operating System — is a unified personal data platform addressing this gap. A user uploads identity documents (Aadhaar, PAN, Passport, Driving Licence) once; an OCR pipeline extracts structured fields, stores them in an encrypted centralized profile, and exposes that profile through two interfaces: a conversational AI assistant answering natural-language questions about the user's own data, and a browser extension that detects and completes form fields on arbitrary websites.

The system is a three-tier web application. The client tier comprises a Next.js 14 frontend and a Chrome Manifest V3 extension; the application tier is a Node.js/Express.js REST API handling business logic, OCR orchestration, and AI-provider integration; the data tier is MongoDB (via Mon-goose), storing profiles, document metadata, extracted data, and sessions. Security is first-class throughout: JWT session management, a bcrypt-hashed 6-digit PIN, TOTP-based MFA, AES-256 encryption at rest, and HTTPS in transit.

The paper's central contribution is the integration, around a single securely governed personal identity store, of five capabilities individually well established but not previously combined end-to-end: (i) secure identity-document intelli-gence — OCR-driven conversion of uploaded documents into structured, encrypted profile data; (ii) deterministic, schema-grounded RAG — a conversational assistant retrieving directly from a structured MongoDB schema rather than a general cor-pus; (iii) intelligent semantic form-field matching — reconcil-ing heterogeneous, arbitrarily named web-form fields against a fixed profile-field set via synonym and fuzzy matching; (iv) browser-based autofill using the matched fields, without per-site configuration; and (v) secure, centralized identity manage-ment governing the pipeline through layered authentication, encryption, and consent. No individual component is a novel algorithm; each builds on established OCR, RAG, string-similarity, and web-security techniques. Concretely, the paper: describes an OCR-driven extraction pipeline converting docu-ment images/PDFs into queryable key-value data; presents a RAG-based conversational interface with sensitive-field mask-ing; implements a Form Analyzer and Multi-Document Merge Engine reconciling heterogeneous field names and conflicting multi-document data via confidence-based prioritization; de-tails a Chrome extension performing field-level autofill across arbitrary sites; and presents a multi-layered security architec-

ture aligned with India's Digital Personal Data Protection Act, 2023.

The paper proceeds as follows: Section II reviews related work; Section III states the problem; Section IV lists objectives; Section V describes the methodology; Section VI details the architecture; Section VII covers implementation; Section VIII formalizes the core algorithms; Section IX describes security; Section X presents evaluation; Sections XI–XII discuss advantages and limitations; Section XIII addresses threats to validity; Section XIV outlines future scope; Section XV concludes.

II. Literature Review

A. Password Managers and Browser-Native Autofill

Password managers such as LastPass, 1Password, and Bitwarden pioneered form-autofill alongside credential storage [1], but their autofill is restricted to usernames, passwords, and basic contact fields, with no ability to process identity documents, no OCR, and no AI querying. Browsers provide similarly scoped built-in autofill for addresses and payment fields [2], but cannot read uploaded documents, cannot handle India-specific fields such as Aadhaar or PAN, and store values in plaintext within the browser profile. Both categories leave identity-document data management unsolved.

B. Government and General-Purpose Document Repositories

DigiLocker enables citizens to store and share official documents digitally [3] but offers no form-filling or AI-query capability and is restricted largely to government-issued records. General-purpose platforms (Google Drive, Dropbox, Evernote) allow file storage with rudimentary OCR-based search, but none extract structured fields or provide autofill integration. Both categories function as storage systems rather than data-management systems.

C. Intelligent Document Processing and OCR Research

Intelligent Document Processing (IDP) integrates OCR, ML, and NLP, and is characterized as an evolution beyond template-based automation [4], offering advantages over conventional RPA for unstructured documents such as invoices [5]. Public-administration applications have been explored [6], and LLM–OCR integration has been proposed for degraded-text extraction, including immigration processing [7]; this paper does not adopt the specific performance figures from [6], [7] without independent verification. No-code agentic IDP frameworks have also emerged [8]. FormOS occupies a narrower niche — a browser-based, personal-identity-focused OCR pipeline (Tesseract.js) rather than an enterprise classification system — and Section X reports no empirical OCR accuracy figures for FormOS itself.

D. Retrieval-Augmented Generation and Semantic Retrieval

Lewis et al. introduced RAG, showing that grounding generation in retrieved documents improves factual accuracy over parametric generation alone [9]. Sawarkar et al. blended dense and sparse retrieval with hybrid queries [10]; surveys consolidate the broader evolution of RAG architectures [11], [12]. Personalization-oriented RAG has been explored for question-answering over individual databases [13], graph-based retrieval for context-aware document retrieval [14], and domain-specific RAG in legal prediction [15] and cybersecurity intelligence [16]. FormOS applies the core RAG principle in a narrower, personal-data context: rather than corpus-scale dense/sparse retrieval, it retrieves fields directly from the user's own structured MongoDB profile via deterministic, schema-based lookup, without a vector database or embeddings. The shared principle is retrieval-grounded generation, not the specific mechanism (Section VIII, Algorithm 4).

E. Web Automation and Programming-by-Demonstration

Web automation traditionally relies on Selenium-based testing/data-entry frameworks [17]. Recent work explores natural-language-driven automation: DiLogics segments task steps via NLP and demonstration [18], Steward extends this to plain-English instructions [19], and Semanticon proposes semantic conditions for automation programs [20]; ML-based approaches also improve web-navigation accessibility [21]. A consistent limitation is handling semantic variability of form fields. FormOS's Form Analyzer targets a narrower problem — matching a fixed personal-identity field set to arbitrary forms — via synonym and fuzzy matching (Section VIII, Algorithm 2).

F. Security and Privacy in Document and RAG Systems

Cloud document-storage research has proposed cryptographic integrity-checking for multi-tenant privacy [22], and secure, self-hosted retrieval has been proposed as an alternative to cloud-hosted RAG for privacy-sensitive composition [23]; digital-transformation research emphasizes security-aware design as document workflows move online [24]. FormOS's security model (Section IX) is consistent with this direction toward layered, encryption-first design, though it does not implement the specific schemes of [22], [23].

G. Research Gap

Three gaps emerge. The RAG literature [9]–[16] is oriented toward question-answering, not driving external actions such as form completion. The web-automation literature [17]–[21] maps instructions to page actions but not the narrower problem of reconciling a user's structured identity data — often spread across conflicting documents — with heterogeneous field naming. The IDP literature [4]–[8] targets enterprise/government workflows without autofill integration. FormOS sits at this intersection without proposing a new retrieval, automation, or classification algorithm, integrating instead structured OCR extraction, retrieval-grounded querying, and field-level autofill around one securely stored personal profile.

H. Feature Comparison with Existing Solutions

Table I compares FormOS against representative systems from each category above, based on documented feature sets rather than independent hands-on testing.

TABLE I
FEATURE COMPARISON BETWEEN FORMOS AND EXISTING SOLUTIONS

Feature	Chrome Autofill	Password Managers	DigiLocker	Google Drive	FormOS
OCR Extraction	No	No	No	Partial (search only)	✓
AI Conversational Query	No	No	No	No	✓
RAG-Grounded Responses	No	No	No	No	✓
Identity Document Support	No	No	✓	Partial	✓
Structured Data Extraction	No	No	No	No	✓
Browser Autofill	✓	✓	No	No	✓
Intelligent Field Matching	Partial (exact match)	Partial (exact match)	No	No	✓
Multi-Document Merge	No	No	No	No	✓
Encryption at Rest	Partial	✓	✓	✓	✓(AES-256)
Multi-Factor Authentication	No	Partial	✓	✓	✓
Cloud Storage	No	Partial	✓	✓	✓(AWS S3)

No reviewed system combines identity-document OCR, structured extraction, RAG-based querying, intelligent field matching, and browser autofill; this is the integration gap FormOS targets.

I. Technology Selection Rationale

Tesseract.js was chosen over Google Vision API/AWS Textract for open-source licensing, cost, and offline capability, with cloud OCR planned as a future enhancement [25]. Gemini was chosen as primary AI provider for its free tier, with OpenAI/OpenRouter as configurable alternatives (implementation documentation) [26], [27]. Next.js 14 was chosen for App Router/SSR [28]; Node.js/Express.js for ecosystem maturity and async I/O; MongoDB for schema flexibility across heterogeneous document types [29]. As Table I shows, no existing product combines document storage, OCR, AI querying, and browser autofill in one platform.

III. PROBLEM STATEMENT

Every application for employment, education, banking, or a government scheme requires re-typing identity information — name, DOB, address, Aadhaar, PAN — unchanged since prior submissions, concentrated in exactly the fields most prone to transcription error. This paper does not adopt a specific error-rate figure without empirical verification (Section X).

Individuals also lack a single organized location for personal information; scanned documents are scattered across email, chat, and phone galleries, and storing them unencrypted creates a security risk — a single compromised account can expose Aadhaar, PAN, and passport data. Existing autofill tools compound this: browser built-ins and password managers handle only basic contact fields and cannot map identity data to varied form structures, leaving the most sensitive fields to manual entry regardless. FormOS addresses these three interlinked problems — repetition, disorganization, and insecure storage — within one platform.

IV. OBJECTIVES

FormOS aims to consolidate a user's identity data, extract it automatically via OCR, expose it through a natural-language AI interface, and apply it to fill web forms without manual effort, decomposed into seven objectives: (1) a unified, encrypted repository for Aadhaar, PAN, Passport, Driving Licence, and custom documents (PDF/PNG/JPG, local or AWS S3); (2) OCR-based extraction of name, DOB, ID numbers, address, and contact fields into queryable structured data; (3) a RAG-based chat interface answering natural-language queries with sensitive-field masking; (4) a Chrome extension mapping and filling detected fields despite varied naming conventions, with visual feedback; (5) JWT, bcrypt PIN, TOTP MFA, AES-256 encryption, HTTPS, rate limiting, and step-up authentication; (6) a clean, responsive, dark-themed interface with real-time status feedback; and (7) a modular, Docker-containerized, horizontally scalable architecture with efficient database indexing.

V. PROPOSED METHODOLOGY

Users upload documents through the web interface; files are stored in AWS S3 or local storage and processed via Tesseract.js and PDF-parsing libraries to extract raw text. The extracted text is parsed into structured key-value data (name, DOB, address, ID numbers) and persisted in MongoDB. A Form Analyzer evaluates target forms against the stored profile, flagging autofillable versus low-confidence fields using synonym mapping and fuzzy matching.

A Multi-Document Merge Engine combines data from multiple uploaded documents into one unified profile, resolving conflicts via priority rules and confidence scoring. A RAG-based AI assistant retrieves relevant fields from the user's own data store and generates a context-grounded response rather than relying on parametric knowledge alone. A Chrome extension completes forms using the same field-mapping logic as the Form Analyzer. Security constraints — encrypted communication, sensitive-data masking, re-authentication for

critical actions — apply throughout the pipeline rather than as a final step. Development followed an Agile iterative process (Section X).

VI. System Architecture

A. Overview

FormOS is a three-tier application (Fig. 1). The client tier is a Next.js frontend and Chrome extension [2], [28]; the application tier is a Node.js/Express.js API handling business logic, external integrations, and OCR orchestration; the data tier is MongoDB via Mongoose, storing profiles, metadata, extracted data, and sessions [29]. External dependencies include AI providers (Gemini, OpenAI, OpenRouter) [26], [27], AWS S3 [30], and the Resend API for email.

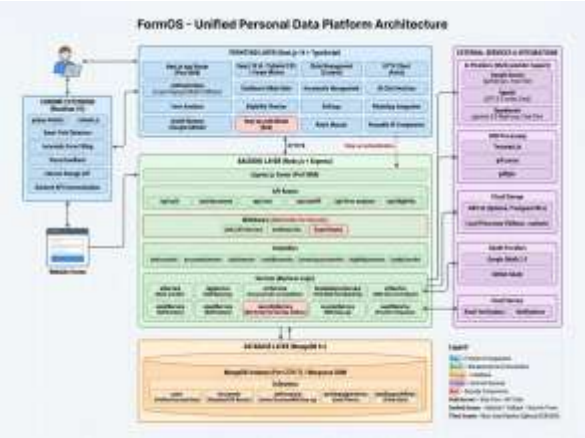

Fig. 1. System architecture of FormOS, showing the client, application, and data tiers and their external dependencies.

The request path: the Frontend or Extension issues an authenticated HTTPS request to the REST API. For uploads, the API hands the file to the OCR Pipeline, which persists it, extracts text, and returns candidate fields; the API writes these to MongoDB, invoking the Merge Engine where multiple documents overlap. For form-filling, the Extension sends detected DOM fields to the Form Analyzer, which queries the profile in MongoDB and returns a mapping report for injection. For conversational queries, the API retrieves relevant fields via schema-based lookup and forwards a context-augmented prompt to the AI Provider. Every path is gated by the authentication and authorization checks of Section IX before profile data is read or written.

B. Core Data Flow

The data flow (Fig. 2) proceeds through eight stages: (1) **Upload** — the user submits a document via drag-and-drop; (2) **Storage** — persisted to AWS S3, with local-filesystem fallback; (3) **OCR** — Tesseract.js (with pdf2pic conversion where needed) produces raw text; (4) **Structured Parsing** — raw text is parsed into key-value fields and mapped to a document-type schema; (5) **Merge Engine** — conflicting values across documents are reconciled via priority rules and confidence scoring; (6) **Database** — the unified profile is persisted in MongoDB, field-level encrypted; (7) **AI Chat** — the RAG assistant retrieves relevant fields and generates a grounded response; (8) **Chrome Autofill** — the Form Analyzer maps detected fields to the profile and the extension injects matched values.

Fig. 2. Data flow diagram of FormOS, corresponding to the eight stages of Section VI-B.

C. Technology Stack

TABLE II
Technology Stack by System Layer

Layer	Technology	Purpose
Frontend	Next.js 14, React 18, Tailwind CSS	UI, routing, responsive design
Backend	Node.js, Express.js	REST APIs, orchestration
Database	MongoDB, Mongoose ODM	Users, documents, sessions
AI & OCR	Tesseract.js, pdf-parse, Gemini/OpenAI/OpenRouter	Extraction, RAG querying
Storage	AWS S3 / local filesystem	Secure document storage
Auth & Security	JWT, bcryptjs, TOTP (speakeasy)	Authentication, hashing, MFA

VII. Implementation

A. Development Environment

The frontend uses Next.js 14 (App Router), TypeScript, Tailwind CSS, Framer Motion, Zustand, and Axios [28].

The backend uses Node.js/Express.js with Helmet.js, CORS middleware, and express-validator. Text extraction uses Tesseract.js with pdf-parse and pdf2pic [25].

B. Authentication, Onboarding, and Document Pipeline

Account creation captures email, name, phone, and a bcrypt-hashed 6-digit PIN. Login issues a JWT with a 7-day expiry [31]; OAuth 2.0 (Google, GitHub) is supported as an alternative onboarding path. Uploaded files (PDF/PNG/JPG) are persisted to AWS S3 with local-filesystem fallback [30] and queued for asynchronous OCR with client-visible status tracking; extracted fields are written to MongoDB collections accommodating the varying schema of Aadhaar, PAN, Passport, and Driving Licence documents [29].

C. Chrome Extension and Deployment

The autofill client is a Manifest V3 extension [2] with a popup UI and a content script performing DOM-level field detection and value injection, authenticated via the same JWT mechanism as the web frontend, using chrome.storage/chrome.runtime for state and messaging. A Docker Compose configuration containerizes the frontend, backend, and MongoDB; recommended production targets are Vercel (frontend), Railway/Heroku/DigitalOcean (backend), and MongoDB Atlas (database). The project's earlier documentation contained two resolvable inconsistencies — differing quick-start versus Docker-Compose port references, and README language describing the AI chat as "OpenAI-powered" rather than multi-provider — both of which should be reconciled in the project's README ahead of formal deployment.

VIII. Algorithms

This section formalizes, in IEEE algorithmic pseudocode, the four core routines described in Section V. These reflect the level of detail available in the source documentation; exact scoring formulae and thresholds are undocumented implementation parameters and are not invented here.

Algorithm 1 Document OCR and Structured Field Extraction

Require: Uploaded file F (PDF, PNG, or JPG)
Ensure: Structured field set S, persisted to MongoDB
procedure ExtractFields(F)
  Persist F to AWS S3 or local storage
  if F is a PDF **then**
    Convert pages to images via pdf2pic
  else
    Use F directly as image input
  end if
  $T \leftarrow$ Tesseract.js OCR output on the image(s)
  $C \leftarrow$ candidate fields parsed from T
  Map C to a document-type schema
  $S \leftarrow$ structured key–value fields from C
  Persist S to MongoDB; update processing status
  return S
end procedure

Complexity. Algorithm 1's cost is dominated by OCR engine time on the input image(s). Algorithm 2 is $O(|I| \times |P|)$ per form, comparing each detected field against the profile field set. Algorithm 3 is $O(n \times k)$ for n documents and k logical fields. Algorithm 4 is $O(|F|)$, bounded by the number of schema fields retrieved. These are asymptotic characterizations of the described logic, not measured runtimes.

Algorithm 2 Form Analyzer Field Matching

Require: Webpage DOM D; stored profile field set P
Ensure: Field-mapping report R
procedure AnalyzeForm(D, P)
  $I \leftarrow$ input fields enumerated from D
  for each field $f \in I$ **do**
    $\ell \leftarrow$ label/name/placeholder of f
    $m \leftarrow$ synonym lookup of ℓ against P
    if m is empty **then**
      $m \leftarrow$ fuzzy match of ℓ against P
    end if
    $c \leftarrow$ confidence score of mapping m
    Classify f as autofillable/low-confidence/unmapped using c
    $R \leftarrow R \cup \{(f, m, c)\}$
  end for
  return R
end procedure

Algorithm 3 Multi-Document Merge Engine

Require: Field sets $S_1, \ldots, S_n$ from n documents
Ensure: Unified profile U
procedure MergeDocuments($S_1, \ldots, S_n$)
  Group candidate values by logical field k
  for each logical field k **do**
    if one candidate exists **then**
      $U[k] \leftarrow$ that value
    else
      Apply priority rules and confidence scores; $U[k] \leftarrow$ best candidate
    end if
  end for
  Persist U; retain per-document sources
  return U
end procedure

For Algorithm 2, matching is illustrative rather than exhaustively documented: a page field labeled "Full Legal Name" matches profile field `name` via synonym lookup, while a field labeled "DOB" matches `date_of_birth` via fuzzy string similarity when no synonym entry exists. The specific similarity metric and confidence thresholds separating autofillable, low-confidence, and unmapped classifications are undocumented implementation parameters; their disclosure is identified as necessary for reproducibility (Section XIV), as is the document-priority ordering and scoring formula referenced in Algorithm 3.

As discussed in Section II-D, retrieval in Algorithm 4 is deterministic schema-based lookup, not vector similarity search; FormOS implements no vector database or embeddings. This is narrower than corpus-scale RAG but shares its defining principle [9]: output grounded in retrieved data rather than parametric knowledge alone.

A. Conceptual Formulation of Confidence-Based Matching

The field-matching confidence referenced in Algorithms 2–3 can be expressed conceptually as a weighted aggregation of per-feature similarity scores:

Algorithm 4 RAG-Based Conversational Query Resolution

Require: Query q; profile store U
Ensure: Grounded response r
1: **procedure** RESOLVEQUERY(q, U)
2: $F \leftarrow$ schema-based lookup of fields in U relevant to q
3: Apply sensitive-field masking to F
4: $p \leftarrow$ prompt combining q and F
5: $r \leftarrow$ AI-provider response given p
6: **return** r
7: **end procedure**

$$\text{Conf} = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i} \quad (1)$$

where s_i is the similarity score of feature i (e.g., label-string similarity, synonym-dictionary agreement, field-type compatibility) and w_i its weight. For example, with w_1=0.6, s_1=0.9 for label-string similarity and w_2=0.4, s_2=0.7 for synonym agreement, Eq. (1) yields Conf $\approx$ 0.82. This is a conceptual illustration of how such scoring is typically structured, *not* the exact formula implemented in FormOS, whose weights and feature set are undocumented (Section XIII).

IX. Security Mechanisms

FormOS applies controls at the authentication, transport, storage, and application layers (Table III). Authentication combines JWT tokens [31] with TOTP-based MFA [32]; application-layer mitigations are informed by the OWASP Top Ten [33].

TABLE III
Security Mechanisms by Layer

Layer	Mechanism	Purpose
Auth	JWT (7-day expiry)	Session management
Auth	Bcrypt-hashed 6-digit PIN	Secondary login factor
Auth	TOTP MFA (speakeasy)	Time-based verification
Auth	OAuth 2.0 (Google, GitHub)	Federated sign-in
Transport	HTTPS enforcement	Prevents interception
Storage	AES-256 encryption	Protects data at rest
App	Data masking (****1234)	Limits UI/chat exposure
App	express-validator	Mitigates injection
App	Helmet.js, CORS	Hardens HTTP layer
App	Rate limiting	Mitigates abuse
App	Step-up re-authentication	Verifies sensitive actions

A. End-to-End Security Workflow

The layers in Table III operate as a single ordered workflow rather than independent checks: User Request → Authentication → JWT Issuance/Validation → PIN Verification → TOTP Check → Authorization → AES-256 Encrypted Storage/Retrieval → MongoDB.

No single layer is sufficient alone: a valid JWT does not authorize unmasked-data access, and a valid PIN does not bypass TOTP for sensitive operations. This defense-in-depth means that compromise of any one factor — a leaked token, a guessed PIN, an intercepted request — does not by itself expose plaintext data, since encryption at rest, step-up re-authentication, and masking continue to constrain what an attacker holding only that factor can retrieve. Step-up re-authentication gates unmasked-data viewing, document export, and high-sensitivity autofill. Consent is captured at registration, users may delete their data at any time, and the data-handling model aligns with India's DPDP Act, 2023 [34]. All security dependencies (JWT, bcryptjs, speakeasy) are MIT-licensed.

X. Experimental Evaluation

As FormOS is an in-progress capstone implementation without deployed usage logs, this section reports qualitative feasibility and module-level functional validation rather than empirical performance metrics. No OCR accuracy, autofill success-rate, or RAG-relevance benchmarks were available and none are asserted here; quantitative validation against a labeled corpus and a live-form test suite is future work (Section XIV).

A. Technical and Economic Feasibility

Every stack component is production-proven (Table IV); no experimental component sits on the critical path. Identified risks are mitigated specifically: OCR degradation via pre-processing and manual override; AI-provider outages via multi-provider failover; scalability via database indexing and a stateless, horizontally scalable API.

TABLE IV
Technical Feasibility by Component

Component	Technology	Rating
Frontend	Next.js 14	High
Backend	Node.js / Express.js	High
Database	MongoDB 6.0	High
OCR Engine	Tesseract.js 5.0	High
AI Integration	Gemini / OpenAI	High
File Storage	AWS S3	High
Extension	Chrome Manifest V3	High
Authentication	JWT + bcrypt + TOTP	High

Economically, core cloud services (MongoDB Atlas, Vercel, Gemini) offer sufficient free tiers, with domain registration the only recurring cost. Operationally, onboarding is reported to complete in under five minutes on a modular codebase. Legally, feasibility rests on explicit consent, user-initiated deletion, AES-256 encryption in transit and at rest, DPDP Act alignment, and MIT-licensed dependencies.

B. Functional Validation

In place of quantitative benchmarking, each core module was exercised during development to confirm it performs its intended function — functional validation, not measured accuracy or latency (Table V). A "Passed" status indicates the

module performed its intended function during this qualitative, development-time exercising (e.g., an uploaded document produced extracted text; a chat query produced a grounded response; the extension injected values into a test form); it is not a measured pass rate over a labeled test set, and no precision, recall, latency, or success-rate figure is claimed for any module.

TABLE V
MODULE-LEVEL FUNCTIONAL VALIDATION

Module	Status	Validation Basis
Authentication (JWT/PIN/TOTP)	Passed	Manual walkthrough
OCR Pipeline	Passed	Manual walkthrough
Document Upload	Passed	Manual walkthrough
AI Chat (RAG Query)	Passed	Manual walkthrough
Chrome Extension (Autofill)	Passed	Manual walkthrough
Merge Engine	Passed	Manual walkthrough
AES-256 Encryption	Passed	Manual walkthrough

C. Schedule Feasibility

All planned milestones — architecture and planning, authentication, OCR pipeline, AI chat integration, Chrome extension, testing, and deployment — are reported achieved. The source documentation presents two differing schedule framings (a twelve-week sprint characterization versus an eight-month, thirty-two-week schedule) which should be reconciled in future project documentation.

XI. ADVANTAGES

- Consolidates identity-document collection into one upload event, reducing repetitive annual form-entry burden and transcription errors from a single verified profile rather than repeated free-text input.
- Integrates document storage, OCR, AI querying, and browser autofill within one platform — a combination not identified in any single reviewed system (Table I).
- Multi-provider AI integration (Gemini, OpenAI, OpenRouter) reduces single-vendor dependency and provides outage failover.
- Layered security (JWT, PIN, TOTP MFA, AES-256, OAuth, masking, step-up authentication) provides defense-in-depth beyond single-factor browser autofill.
- Modular, containerizable architecture supports horizontal scaling and multi-cloud deployment without redesign.

XII. LIMITATIONS

- No empirical performance data (OCR accuracy, autofill success rate, RAG relevance) is available; evaluation is qualitative and feasibility-based, supplemented by functional validation only.
- OCR accuracy depends on scan/photo quality; degraded input may require manual correction despite preprocessing.
- The autofill extension targets Chrome (Manifest V3) exclusively.
- Reliance on third-party AI APIs introduces cost, rate-limit, and availability dependencies only partially mitigated by failover.
- Supported document schemas (Aadhaar, PAN) are India-specific, constraining applicability elsewhere.
- The similarity metric, confidence thresholds, and merge-priority rules underlying Algorithms 2–3 are undocumented, limiting independent reproducibility.

XIII. THREATS TO VALIDITY

Construct validity. The "Passed" status in Table V reflects functional exercising during development, not a formally specified test protocol with predefined pass/fail criteria, and should not be read as a unit- or integration-test suite result.

Internal validity. Structured-field extraction (Algorithm 1) depends entirely on OCR quality, so downstream Form Analyzer and Merge Engine conclusions are conditional on input quality, which was not systematically varied or measured.

Absence of quantitative evaluation. No OCR accuracy, autofill success rate, RAG relevance, or latency measurement has been collected; feasibility and advantages in Sections X–XI are architectural and qualitative claims, not empirically demonstrated ones.

XIV. FUTURE SCOPE

A. Near-Term

- DigiLocker integration for verified document retrieval beyond user-uploaded scans.
- Cloud-based and improved OCR models (e.g., Google Vision API) to raise extraction accuracy on lower-quality scans.
- Vector-database and embedding-based retrieval to complement the current schema-based RAG lookup (Algorithm 4), situated against the retrieval literature of Section II-D rather than replacing it outright.
- Disclosure of the concrete matching parameters and merge-priority rules identified as a reproducibility gap (Sections VIII, XII).
- Structured empirical evaluation: OCR accuracy against a labeled corpus, autofill success-rate testing across real-world forms, and a structured user study (Section XIII).

B. Long-Term

- A React Native mobile application and multi-language OCR support.
- NFC-based identity cards for tap-to-autofill interaction.
- IoT-based smart document scanning to reduce reliance on phone-camera quality.
- A voice-based, multilingual AI assistant for improved accessibility.
- Team-account functionality and enterprise API access.

XV. Conclusion

This paper presented FormOS, a systems-integration platform eliminating repetitive identity-data entry by combining OCR-based extraction, a schema-grounded RAG conversational interface, a Form Analyzer with a Multi-Document Merge Engine, and a Chrome extension for automated form completion, on a three-tier Next.js/Node.js/MongoDB architecture secured by JWT, PIN/TOTP MFA, and AES-256 encryption aligned with India's DPDP Act, 2023.

A feasibility analysis spanning technical, economic, operational, schedule, and legal dimensions, together with module-level functional validation (Section X-B), indicates FormOS is viable for real-world deployment on production-proven technologies. As the platform has not undergone empirical benchmarking, this paper deliberately reports a qualitative, architecture-grounded evaluation rather than fabricated performance figures, identifying structured quantitative validation — OCR accuracy measurement, autofill success-rate testing, and a formal user study (Section XIV) — as the essential next step. The architecture, security model, and integration approach described here provide a practical, extensible foundation for more rigorously validated, next-generation secure AI-powered identity management systems, addressing a widespread and currently underserved problem in everyday digital interaction.

Acknowledgment

The authors thank their project guide, Prof. Shashikant Mahajan, for mentorship and technical oversight, and the Head of Department, Dr. Vidya Chitre, and Principal, Dr. Sangeeta Joshi, for departmental support throughout this project.

References

[1] LastPass, 1Password, and Bitwarden — Password manager platforms with autofill functionality. [Online]. Available: respective vendor documentation.

[2] Chrome Extensions, Manifest V3 developer guide and API reference. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3

[3] DigiLocker, National e-Governance Division, Ministry of Electronics and Information Technology, Government of India. [Online]. Available: https://www.digilocker.gov.in

[4] K. Kapula, "Intelligent document processing: The new frontier of automation," *World Journal of Advanced Engineering Technology and Sciences*, 2025. DOI: 10.30574/wjaets.2025.17.1.1360.

[5] V. B and J. Krithika, "An Overview on IDP and Its Significance in Business," *International Journal for Science Technology and Engineering*, 2022. DOI: 10.22214/ijraset.2022.47001.

[6] R. Pingili, "AI-driven intelligent document processing in government and public administration," *World Journal of Advanced Engineering Technology and Sciences*, 2025. DOI: 10.30574/wjaets.2025.15.3.1059.

[7] N. Abdel-Rahman and H. Ali, "ERPA: Efficient RPA Model Integrating OCR and LLMs for Intelligent Document Processing," arXiv:2412.19840, 2024.

[8] G. Vignesh, H. P. M, S. Reddy, S. Gopalakrishnan, and V. Vaddina, "IDPFlow: A No-Code Agentic Framework for Multimodal Intelligent Document Processing," in *Proc. 33rd ACM Int. Conf. Multimedia*, 2025. DOI: 10.1145/3746027.3761838.

[9] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474.

[10] K. Sawarkar, A. Mangal, and S. R. Solanki, "Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers," in *Proc. 2024 IEEE 7th Int. Conf. Multimedia Information Processing and Retrieval (MIPR)*, 2024, pp. 155–161. DOI: 10.1109/MIPR62202.2024.00031.

[11] S. Gupta, R. Ranjan, and S. N. Singh, "A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions," arXiv:2410.12837, 2024.

[12] W. Ke et al., "Large Language Models in Document Intelligence: A Comprehensive Survey, Recent Advances, Challenges and Future Trends," *ACM Transactions on Information Systems*, 2025. DOI: 10.1145/3768156.

[13] J. Byun, B. Kim, K.-A. Cha, and E. Lee, "Design and Implementation of an Interactive Question-Answering System with Retrieval-Augmented Generation for Personalized Databases," *Applied Sciences*, vol. 14, no. 17, 2024. DOI: 10.3390/app14177995.

[14] V. Kamra, L. Gupta, D. Arora, and A. K. Yadav, "Enhancing document retrieval using AI and graph-based RAG techniques," 2024. DOI: 10.1109/c2i663243.2024.10895931.

[15] S. K. Nigam, B. D. Patnaik, S. Mishra, A. V. Thomas, N. Shallum, K. Ghosh, and A. Bhattacharya, "NyayaRAG: Realistic Legal Judgment Prediction with RAG under the Indian Common Law System," arXiv:2508.00709, 2025.

[16] S. T. Gandhi, "RAG-driven cybersecurity intelligence: leveraging semantic search for improved threat detection," 2023. DOI: 10.15662/ijrai.2023.0603003.

[17] G. Chen, G. Chen, D. Wu, Q. Liu, L. Zhang, and X. Fan, "A Selenium-based web application automation test framework," in *Proc. 2021 Int. Conf. Information and Business Applications (ICIBA)*, 2021. DOI: 10.1109/ICIBA52610.2021.9688190.

[18] D. Rui, W. Xinyu, C. Yan, and G. Tovi, "DiLogics: Creating web automation programs with diverse logics," arXiv:2308.05828, 2023.

[19] B. Tang and K. G. Shin, "Steward: Natural language web automation," arXiv:2409.15441, 2024.

[20] K. Pu et al., "Semanticon: Specifying content-based semantic conditions for web automation programs," 2022. DOI: 10.1145/3526113.3545691.

[21] T. Makati, "Machine learning for accessible web navigation," 2022. DOI: 10.1145/3493612.3520463.

[22] K. Gokulkannan, M. Parthiban, S. Jayanthi, and M. K. T, "Cost effective cloud-based data storage scheme with enhanced privacy preserving principles," *The Scientific Temper*, 2024. DOI: 10.58414/scientifictemper.2024.15.2.21.

[23] A. Chen and S. Tran, "Supercharging Document Composition with Generative AI: A Secure, Custom Retrieval-Augmented Generation Approach," 2024. DOI: 10.1109/sds60720.2024.00025.

[24] S. Jordan, M. Vodopivec, and colleagues, "Document Management System – A Way to Digital Transformation," 2022. DOI: 10.2478/ngoe-2022-0010.

[25] Tesseract.js, JavaScript OCR library documentation (implementation documentation). [Online]. Available: https://tesseract.projectnaptha.com

[26] Google Generative AI, Google Gemini API developer documentation (implementation documentation). [Online]. Available: https://ai.google.dev/docs

[27] OpenAI API Reference, OpenAI platform developer documentation (implementation documentation). [Online]. Available: https://platform.openai.com/docs

[28] Next.js Documentation, Next.js 14 App Router and deployment reference (implementation documentation). [Online]. Available: https://nextjs.org/docs

[29] MongoDB Documentation, MongoDB 6.0 reference and Mongoose ODM guide (implementation documentation). [Online]. Available: https://www.mongodb.com/docs

[30] AWS S3 Documentation, Amazon Simple Storage Service developer guide (implementation documentation). [Online]. Available: https://docs.aws.amazon.com/s3

[31] RFC 7519, "JSON Web Token (JWT)," IETF, 2015. DOI: 10.17487/RFC7519. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519

[32] RFC 6238, "TOTP: Time-Based One-Time Password Algorithm," IETF, 2011. DOI: 10.17487/RFC6238. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6238

[33] OWASP Foundation, "OWASP Top Ten," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[34] Digital Personal Data Protection Act, 2023, Act No. 22 of 2023, Ministry of Electronics and Information Technology, Government of India.