International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Smart Scheme & Scholarship Recommendation Portal

Sanjay Krishna[1], Mavyashri M[2], Manish Bharathi G[3], Mohamed Faizal S[4], Navin R[5]

[1] Assistant Professor, Dept. of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India. Email: sanjaykrishnacse@siet.ac.in

[2] Student (714025104139), Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India. Email: mavya.codes@gmail.com

[3] Student (714025104133), Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India. Email: g.manishbharathi7@gmail.com

[4] Student (714025104145), Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India. Email: faiz.afsal7@gmail.com

[5] Student (714025104162), Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India. Email: rnavin1409@gmail.com

November 2026

ABSTRACT

The Smart Scheme & Scholarship Recommendation Portal is a decision-support web platform that helps students and citizens discover government schemes, scholarships, welfare programmes, subsidies and educational benefits for which they may qualify. Information about such benefits is spread across many ministries, departments and portals, and eligibility conditions are often written in dense administrative language. As a result, eligible people frequently do not apply. The portal addresses this by combining a verified scheme catalogue, a structured citizen profile and a deterministic eligibility engine that explains every outcome in plain language.

The eligibility engine evaluates each scheme's rules against normalised citizen facts using three-valued (Kleene) logic, so that missing information produces an explicit "unknown" instead of a silent pass or fail. Every scheme is classified as Eligible, Likely Eligible, Needs Verification or Not Eligible, with matched, failed and unknown conditions listed. A document vault supports secure upload, signature-checked storage, field extraction and consent-based confirmation of extracted values. A notification module issues deadline, opening and document-expiry alerts. A grounded AI assistant explains results and guides applications, but it is never allowed to decide eligibility. An analytics and audit module gives administrators platform insight and an append-only security trail.

The system is a modular monolith built with React 18, TypeScript, Vite and Tailwind CSS on the frontend, and Node.js, Express, Prisma and PostgreSQL on the backend. It uses short-lived JSON Web Tokens with rotating refresh tokens, bcrypt password hashing, role-based access control, Helmet, strict CORS, rate limiting, structured logging and Docker-based deployment. The repository records 146 backend and 64 frontend automated tests, a seed catalogue of 162 schemes across 14 categories, and ten completed modules. These results describe software behaviour; they are not an independent measurement of recommendation accuracy, and the portal gives indicative guidance only. Final approval always rests with the responsible department or institution.

Keywords: *scheme recommendation, scholarship discovery, rule-based eligibility, three-valued logic, explainable decision support, grounded AI assistant, document vault, role-based access control, e-governance.*

I. INTRODUCTION

Welfare and education schemes are meant to lower financial and social barriers, but a large share of eligible people never benefit from them. The obstacles are mostly practical. Details are released separately by central ministries, state departments and individual institutions, each with its own page, application window, document list and eligibility wording. A student who qualifies for a post-matric scholarship, a state fee concession and a skill-training programme may have to read dozens of pages, interpret terms such as "annual family income" or "creamy layer", and track several deadlines. Some abandon the effort, some miss the window and some submit the wrong papers.

This project aims to lighten that burden with one dependable portal. A citizen enters their details once and receives a list of schemes that look suitable, the reasons behind each result, the facts still missing, the certificates to prepare and the dates that matter. Three design priorities set it apart from typical consumer recommenders: **trust**, **explainability** and **modularity**.

Trust. Scheme data is held in a managed catalogue with verification status, department ownership, official application links and versioned eligibility rules. The portal does not scrape or guess. Its guidance is labelled as indicative, and it states clearly that approval and disbursement are decided by the relevant authority.

Explainability. An opaque score such as "87% match" is unhelpful to a citizen deciding whether to gather certificates. The portal instead shows the individual conditions that matched, the conditions that failed, and the conditions that could not be assessed because the profile lacks information. A person can see exactly what to correct or supply.

Modularity. The system is organised into ten modules with clear ownership, from foundation and authentication to analytics and production hardening. Modules that establish facts (profile, scheme rules, eligibility,

documents, notifications) are separated from modules that merely assist the user (discovery interface, AI guidance). This separation is the basis of the portal's safety argument: assistance modules can fail or be switched off without corrupting ground truth.

A. Problem Statement

Large numbers of eligible students and citizens never claim central or state support, mainly because the relevant information is scattered, the eligibility wording is hard to interpret and it is rarely clear why a given decision was reached. Current discovery tools tend to fall into two groups: those that list schemes without tailoring them to the person, and those that rank schemes opaquely and ignore what the applicant has not yet told them. Neither group tells a user, with reasons, whether applying is worthwhile.

B. Objectives

The project set out to meet the following objectives:

1. Provide a centralised, verified catalogue of schemes and scholarships with department attribution, benefits, application periods, document requirements and official links.
2. Capture citizen facts as structured, machine-readable attributes (location, education, income band, social category, occupation) with explicit consent.
3. Evaluate eligibility deterministically from versioned rules, with a clear distinction between failed and unknown conditions.
4. Explain each result in plain language and list the information that would change the outcome.
5. Offer discovery, search, filtering, saving and side-by-side comparison of schemes.
6. Provide a secure document vault with field extraction and user-confirmed mapping to profile attributes.
7. Deliver timely deadline, opening and document-expiry notifications.
8. Offer a guarded AI assistant that explains and translates but never determines eligibility or invents deadlines.
9. Give administrators analytics, append-only audit trails and system-health diagnostics.
10. Harden the platform for production with secure defaults, containerised deployment, backups and a recovery plan.

C. Scope and Intended Users

The portal is intended for students seeking scholarships, citizens seeking welfare entitlements, and administrators who maintain the scheme catalogue. Four privileged roles support the administrative workflow: scheme manager, verifier, administrator and super administrator. The system supports assistance with discovery and preparation. It does not submit applications on behalf of users, it does not authenticate identity against government databases, and it does not replace the decision of the awarding authority.

D. Organisation of the Report

Section II reviews existing approaches and related technology. Section III describes the proposed methodology and the ten-module design. Section IV describes the implementation, covering the technology stack, data model, API, rule engine, document pipeline, notifications, assistant, analytics and deployment. Section V lists the advantages of the approach. Section VI presents results and analysis, including verification evidence and an analysis of the eligibility engine. Sections VII and VIII give the conclusion and future work. References follow.

II. EXISTING SYSTEM AND LITERATURE SURVEY

A. Existing Approaches

People currently discover schemes through four broad routes: word of mouth and local offices, individual department websites, national portals that aggregate information or applications, and general web search or social media. Each route has strengths but also characteristic weaknesses.

Word of mouth and local offices. Local officials and community networks can interpret context and help with paperwork, but coverage is uneven. Awareness depends on whom a person knows, and the advice may be out of date.

Department websites. These are authoritative for a single scheme, but they require the citizen to already know the scheme exists. Eligibility criteria are written for administrators, not for applicants, and the format differs from site to site.

Aggregating national portals. Government platforms such as the National Scholarship Portal [10] and the mySchemе discovery service [11] centralise information or applications and are valuable public infrastructure. Their emphasis is on cataloguing and applying. A citizen may still need to interpret each scheme's criteria, and may not learn precisely which missing fact or certificate blocks a decision.

General search and social media. Search engines return a mixture of official pages, agent sites and outdated blog posts. Misinformation about deadlines and amounts is common, and there is no assurance that a page is current or official.

The table below summarises the differences that motivated this project.

TABLE I. COMPARISON OF BENEFIT-DISCOVERY APPROACHES

Approach	Strength	Limitation addressed by this project
Local offices and word of mouth	Contextual human help	Uneven coverage; no record or consistency
Individual department sites	Authoritative for one scheme	Citizen must already know the scheme; dense language
Aggregating national portals	Broad catalogue; official	Limited personalised explanation of failed or unknown criteria
Web search and social media	Easy to reach	Unverified, outdated or misleading content
Generic AI chatbots	Fluent answers	Can invent eligibility rules, amounts and deadlines
Proposed portal	Verified catalogue, deterministic explainable eligibility, document readiness, alerts	Indicative guidance only; catalogue requires continued curation

B. Limitations of Existing Systems

The limitations of the present landscape can be stated as workflow risks rather than as criticisms of any single service.

1. **Fragmentation.** Information is distributed across many sites with inconsistent structure, so a citizen must search repeatedly.
2. **Opaque eligibility.** Criteria are expressed in prose. Users cannot easily tell which of several conditions they fail.
3. **Treatment of missing information.** Ordinary filters either exclude a scheme when a field is empty or include it without warning. Both outcomes mislead. A missing income band is neither a pass nor a fail.
4. **No document readiness view.** Users learn about required certificates only at the application stage, when a delay may cause them to miss a deadline.
5. **Weak deadline support.** Application windows open and close on different dates, and changes are not pushed to the user.
6. **Unsafe AI use.** A general-purpose language model asked "am I eligible for this scholarship?" may produce a confident but fabricated answer. In a public-service setting this can cause real harm.
7. **Privacy and security.** Documents such as income and community certificates are sensitive. Careless handling of uploads, tokens or logs creates risk.

C. Related Technologies and Literature

Rule-based eligibility and expert systems. Rule-based systems write domain knowledge down as explicit conditions, which suits regulated settings where a decision has to be justified. Entitlement to a public benefit usually reduces to combinations of attribute tests, a form that translates directly into rules. A common flaw in simple implementations is that an absent fact is silently read as false. This project adopts a three-valued interpretation so that absent facts propagate as "unknown".

Three-valued logic. Kleene's strong three-valued logic [1] extends the usual Boolean tables for AND, OR and NOT with a third value, unknown. A conjunction that contains a definite false is false whatever the unknown operands are, whereas true combined with unknown stays unknown; a disjunction that contains a true is true whatever the unknowns are. This mirrors how eligibility is actually judged: a single failed mandatory condition rules a scheme out, while an unknown one only postpones the decision. The portal's logical evaluator implements exactly these tables.

Recommender systems. Collaborative and content-based recommenders [2] rank items using behaviour or similarity. These methods work well for media and shopping, but they fit eligibility badly: here a correct answer matters more than a relevant-looking one, and a plausible but ineligible suggestion wastes a citizen's effort. The portal therefore uses deterministic rule evaluation as the recommendation core and reserves ranking and sorting for presentation.

Retrieval-augmented and grounded generation. Lewis et al. [3] describe combining retrieved knowledge with language generation for knowledge-intensive tasks. Grounding reduces, but does not eliminate, unsupported statements. The portal applies a stricter pattern: the assistant receives only data fetched from the catalogue and rule engine for the current user, and its output is checked against those records before delivery. Eligibility decisions are never delegated to the model.

Optical character recognition. OCR engines such as Tesseract [4] turn images of text into machine-readable characters, with accuracy that varies by scan quality, language and page layout. For certificates, the extracted values are best treated as suggestions that a person then confirms. The portal's document module follows this principle through a mandatory confirmation step with consent, and its extraction layer is defined behind a provider interface so that a production OCR engine can replace the current local heuristic provider.

Web security practice. The OWASP Top 10 [5] lists the categories of weakness that most often affect web applications, among them broken access control, cryptographic failures, injection and misconfiguration. The portal's mitigations are organised around that list. JSON Web Tokens [6] carry compact signed credentials, and adaptive password hashing [7] such as bcrypt makes offline guessing expensive. India's Digital Personal Data Protection Act, 2023 [8] sets expectations for consent and purpose limitation, which shape the portal's consent records and data-minimisation rules.

Modern web frameworks. The interface is built from React [9] components checked by TypeScript. Persistence rests on PostgreSQL [13], accessed through the Prisma [12] typed ORM, which supplies referential integrity, indexes and migrations. Zod [14] schemas serve a dual purpose here: one definition validates incoming API data and also yields the matching TypeScript types.

D. Research Gap and Contribution

None of the individual technologies above is new. What this project contributes is the way they are combined into a public-service workflow with the following safeguards:

- eligibility computed by a pure, side-effect-free rule engine using three-valued logic, with four explicit outcomes;
- explanations that distinguish matched, failed and unknown conditions and list the missing information;
- versioned scheme rules so that a recommendation can be related to the exact rule set used;
- a document vault that links extracted values to profile attributes only after user confirmation;
- an AI assistant restricted to explanation and translation, with intent detection, tool-based grounding and rate limiting;
- an append-only audit trail and graceful-degradation plan so that failure of an assistive component does not remove core functionality.

The portal does not claim a new machine-learning method, and it does not claim to be more accurate than other portals. Its distinction is that unavailable information is surfaced as uncertainty and never converted into a favourable result.

III. PROPOSED METHODOLOGY

A. Overview

The portal follows a pipeline in which each stage produces records that the next stage can inspect. A citizen registers and grants consent. The citizen supplies profile facts, either by typing them or by confirming values extracted from uploaded documents. The recommendation service loads all active, published schemes with their current versioned rules, normalises the citizen's facts, and evaluates every rule deterministically. The result for each scheme is stored with its explanation. The dashboard, discovery pages, notifications and assistant then read these stored results rather than recomputing or reinterpreting them. Administrative actions and sensitive reads are recorded in an append-only audit log.

The design principle is a strict boundary between **ground truth** and **assistance**. Profile, scheme rules, eligibility evaluation, documents and notifications are ground truth: they must be correct, auditable and reproducible. The discovery interface and the AI assistant are assistance: they present or explain ground truth but cannot alter it.

B. Ten-Module Architecture

The system is delivered as ten modules. Each has a defined responsibility and a stated relationship to truth.

TABLE II. THE TEN MODULES OF THE PORTAL

Module	Name	Core responsibility	Truth or assistance
1	Foundation & Architecture	Project structure, configuration, health probes, error handling, authentication base	Ground truth
2	Authentication & User Profile	Registration, login, token rotation, roles, normalised citizen profile, consent	Ground truth
3	Scheme Management	Scheme catalogue, departments, benefits, document requirements, application periods, eligibility rules, versions	Ground truth
4	Recommendation Engine	Deterministic rule evaluation, four eligibility outcomes, explanations	Ground truth
5	Discovery & Dashboard	Search, filters, comparison, saved schemes, personalised dashboard	Assistance and user experience
6	Documents & OCR	Secure vault, signature validation, extraction, confirmation, profile mapping	Ground truth
7	Notifications & Alerts	Deadline, opening and document-expiry alerts; channel dispatch; preferences	Ground truth
8	AI Assistant & Guidance	Grounded explanations, multilingual guidance, conversation history	Assistance and user experience
9	Analytics & Audit	Administrative metrics, funnels, audit logs, system health	Observability
10	Production & Security	Hardening, containerisation, backups, recovery, continuous integration	Productionisation

C. Citizen Profile and Consent

Eligibility conditions in government schemes refer to a small, recurring set of facts. The portal therefore stores profile data as structured attributes, decoupled from any particular scheme's logic. The profile is divided into the following groups:

- **Personal:** basic demographic attributes such as gender and date of birth.
- **Location:** state, district and urban or rural area type.
- **Education:** current education level, institution type and relevant academic attributes.
- **Financial:** annual household income expressed in bands.
- **Social eligibility:** social category (for example general, OBC, SC, ST) and related criteria.
- **Occupation:** occupation type and, where relevant, farmer category.
- **Preferences:** categories of interest and notification choices.

Decoupling means that when a scheme manager writes a rule such as "annual income at most a stated limit", the rule refers to a named fact. The same fact can be filled from the profile form or from a confirmed document. A separate **consent** record captures each type of permission the citizen

grants, so that processing can be tied to a stated purpose and withdrawn.

A key rule applies to every fact: it is in one of three states, **known**, **unknown** or **not applicable**. This state travels with the value into the rule engine, and it is what allows the engine to say "unknown" truthfully.

D. Scheme Catalogue and Versioned Rules

Each scheme record carries its department, programme type, categories, state scope, status, verification status, publication flag, benefits (type and frequency), document requirements, application periods and application method with official link. Eligibility is expressed as a list of rule records, each with:

- a **field** naming the citizen fact being tested;
- an **operator** from the set EQUALS, NOT_EQUALS, GREATER_THAN, GREATER_THAN_OR_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, IN, NOT_IN, BETWEEN, CONTAINS, EXISTS and NOT_EXISTS;
- a **value** and a **data type** for comparison;
- a **mandatory** flag distinguishing essential conditions from supporting ones;
- a human-readable **description**, an **order**, an optional **logical operator** and an optional **rule group**.

When rules change, a new **scheme version** is created. A recommendation stores the identifier and number of the version that produced it. This makes results reproducible and allows the system to detect when an earlier recommendation was based on superseded rules, which in turn can trigger a notification that the recommendation has changed.

Administrative workflows allow scheme managers to create and edit schemes, and verifiers to review authenticity. Verification status is shown to citizens so that they can distinguish verified from unverified entries.

E. Deterministic Eligibility Algorithm

The recommendation engine is implemented as a pure function: given a list of rules and a set of normalised facts, it returns a complete evaluation and has no side effects. This makes it straightforward to test in isolation. The algorithm has four steps.

Step 1: Fact normalisation. Raw profile values are converted into comparable forms. Numbers are parsed, booleans are interpreted, and strings are normalised for case and spacing. A field definition catalogue supplies a label, category and description for each fact, which are later used in explanations and in the list of missing information.

Step 2: Condition evaluation. Each rule is evaluated against its fact and produces one of three results:

- **MATCH** when the fact is known and satisfies the operator;
- **FAIL** when the fact is known and does not satisfy the operator;
- **UNKNOWN** when the fact is missing, in the unknown state, or empty, and the operator is a comparison.

The existence operators are treated specially. EXISTS matches when a value is present and fails otherwise, and NOT_EXISTS does the reverse. Every evaluation records the field, operator, required value, actual value, mandatory flag, description and a plain-language explanation. Where the result is UNKNOWN, a reason states that the profile lacks the information.

Step 3: Three-valued combination. Each condition result is mapped to a three-state value (TRUE, FALSE, UNKNOWN). The logical evaluator implements Kleene's strong tables [1]:

TABLE III. KLEENE STRONG THREE-VALUED TRUTH TABLES

A	B	A AND B	A OR B
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

NOT TRUE is FALSE, NOT FALSE is TRUE and NOT UNKNOWN is UNKNOWN. An empty AND is TRUE. In the current implementation these tables are provided and unit-tested, but the overall status in Step 4 is derived from the mandatory and optional counters; combining rule groups with logical OR is not yet used in the status decision and is listed as future work.

Step 4: Status decision. The engine counts mandatory failures, mandatory unknowns, optional failures and optional unknowns, and then applies the following precedence:

TABLE IV. STATUS DECISION PRECEDENCE

Priority	Condition	Resulting status
1	One or more mandatory conditions definitively fail	NOT_ELIGIBLE
2	No mandatory failure, but one or more mandatory conditions are unknown	NEEDS_VERIFICATION
3	All mandatory conditions match, but an optional condition fails or is unknown	LIKELY_ELIGIBLE
4	All mandatory conditions match and nothing remains unknown or failed	ELIGIBLE

A scheme with no rules is treated as open to all and is returned as ELIGIBLE with an explanatory summary. The result also contains the matched, failed and unknown condition lists, a list of missing-information items (field, label, category and description), the number of rules evaluated and a trace of the four counters.

Worked illustration. The following scenario is hypothetical and is included only to show how the algorithm behaves. Suppose a scheme has three mandatory rules (annual income within a stated limit, social category in a stated list, education level equal to undergraduate) and one optional rule (state of residence). A citizen has entered income and social category that satisfy the first two rules, has not entered an education level, and lives in the required

state. The two known mandatory conditions are MATCH, the third is UNKNOWN, and the optional condition is MATCH. There are no mandatory failures but one mandatory unknown, so the status is NEEDS_VERIFICATION, and the missing-information list asks for education level. Once the citizen supplies it and it satisfies the rule, the status becomes ELIGIBLE. If instead it fails the rule, the status becomes NOT_ELIGIBLE, with the failing condition named. In no case is a missing fact silently treated as a pass.

F. Discovery, Comparison and Dashboard

The discovery interface lets users browse the catalogue by category, programme type, department, state scope and status, and search by keyword. Users can save schemes, which also records recently viewed items. A comparison view places multiple schemes side by side on benefits, eligibility, documents and application periods. The dashboard aggregates counts of eligible and likely eligible schemes, items that need information, upcoming deadlines and recent notifications. These views present stored evaluation results and do not redefine them.

G. Document Vault and Extraction

The document module pursues two goals: helping citizens know which certificates they need, and reducing manual entry by suggesting profile values from uploaded documents. The workflow is:

1. **Upload.** The file's declared MIME type and extension must be on an allowlist (PDF, JPEG and PNG). A size ceiling applies, and the server inspects the file's leading bytes to confirm that the content matches the claimed type, defeating simple extension spoofing.
2. **Private storage.** Files are written under generated storage keys that do not depend on the user's file name, with a SHA-256 checksum recorded. They are never served by direct link.
3. **Extraction.** A document-type catalogue defines which fields are extractable for each certificate type (for example income certificate, community certificate and marksheet) and how each maps to a profile attribute. An extraction provider returns text, structured fields, per-field confidence, provider name and version, and a low-confidence flag.
4. **Confirmation.** The citizen reviews the extracted fields and explicitly confirms them with consent. Only confirmed values are mapped to the profile.
5. **Verification state.** A document moves through unverified, pending review, verified for use and rejected states, with a document-level audit log.
6. **Access.** Downloads use short-lived signed tokens rather than permanent links.

The extraction layer is defined behind a provider interface. In the current implementation the registered provider is a local heuristic provider that returns type-specific sample values with confidence scores, and it simulates low confidence for specially named test files. It is intended to demonstrate the confirmation workflow and to be replaced by a production OCR engine such as Tesseract [4]. This limitation is discussed again in Sections VI and VIII.

H. Notification Methodology

Notifications are generated by scheduled jobs and by events. Four jobs are defined: a deadline reminder, an upcoming-opening alert, a document-expiry check and a cleanup task. Sixteen notification types cover scheme opening soon, deadline approaching, deadline today, deadline changed, scheme updated, scheme closed, recommendation available, recommendation changed, profile information needed, document expiring, document expired, document processing completed, processing failed, review required, system announcement and security alert. Each has a severity. Delivery goes through a channel interface with in-app and email implementations. Users choose their preferences, and the in-app bell remains available even if the email channel fails.

I. Grounded AI Assistant

The assistant is designed on the premise that a language model must not be the source of eligibility truth. The methodology has five parts.

1. **Intent detection.** The incoming message and any explicit context type (recommendation, document, notification or comparison) are classified into intents such as comparing schemes, explaining application steps, summarising a scheme, explaining a notification, explaining documents or OCR results, and explaining eligibility.
2. **Tool-based grounding.** For the detected intent, tools fetch only the necessary records for the current user: scheme details, stored recommendations, document status, notifications and profile facts. The model is given this data and no more.
3. **Constrained prompting.** Prompts instruct the model to answer only from supplied data, to say when information is unavailable and to avoid inventing amounts or dates.
4. **Output validation.** Responses are validated against the grounded data and the stored rule outputs before being returned. Source references are attached.
5. **Operational safeguards.** A separate rate limiter applies to assistant requests, usage is recorded, and a mock provider allows deterministic operation when no external model is configured. If the provider is unavailable, the assistant returns an advisory message while the remainder of the portal continues to work.

J. Analytics and Audit

Analytics events are recorded asynchronously so that a telemetry failure never blocks a citizen action. Administrators can view overview metrics, user demographics and profile-completion buckets, scheme discovery funnels, eligibility distributions, document and OCR outcomes, notification dispatch and read rates, assistant usage and latency, and search terms including zero-result queries. Exports are themselves audited. In the current code the audit log records privileged reads of analytics and audit data and every export, and viewing the audit log is also logged; scheme and account changes are written to structured application logs, and document actions go to a separate document audit log. Sensitive items

such as passwords, tokens, identity numbers, bank details and raw OCR payloads are excluded from events and logs.

K. Security Methodology

Security measures are organised against common web risks [5].

TABLE V. SECURITY CONTROLS BY RISK AREA

Risk area	Control in the portal
Authentication	bcrypt hashing with cost factor 12 [7]; 15-minute JWT access tokens [6]; 7-day refresh tokens stored as SHA-256 hashes with rotation; enumeration-safe responses
Brute force	Rate limiting on authentication routes (20 requests per 15 minutes per IP); general API limiter; separate assistant limiter
Access control	Role-based permissions for citizen, scheme manager, verifier, administrator and super administrator; citizen data restricted to the owner
Input handling	Zod validation of body, query and parameters on every route
Transport and headers	Helmet [22] for content security policy, frame options, content-type sniffing prevention, referrer policy and permissions policy; strict CORS without wildcards on authenticated routes
Uploads	Allowlist, size limit, signature check, generated storage keys, short-lived signed download tokens
Logging	Structured Pino [21] logs with request identifiers and redaction of sensitive fields
Error handling	Stack traces and database errors suppressed in staging and production
Privacy	Consent records, data minimisation and alignment with the DPDP Act principles [8]

L. Deployment and Recovery Methodology

The frontend, backend and PostgreSQL database are packaged as containers [20] and started together with a compose file. Liveness and readiness probes support orchestration. Scripts provide database backup with compression and retention pruning, restoration with an automatic integrity check, and a smoke test. The recovery plan sets a recovery point objective of under 24 hours and a recovery time objective of under 2 hours, and it documents graceful degradation: if the AI provider, OCR engine, email channel or analytics pipeline fails, discovery, rule-based recommendation, document storage and official redirection remain available.

IV. SYSTEM IMPLEMENTATION

A. Architectural Style

The application is a **modular monolith** with a separate single-page frontend. A modular monolith keeps deployment simple, with one backend process and one database, while preserving internal boundaries: each backend module owns its routes, controller, service, repository, schemas and types, and modules communicate through service interfaces rather than reaching into each other's tables. This arrangement suits a project of this size and allows a module to be extracted later if scale demands it. The browser communicates with the backend only through a versioned REST API under the path prefix /api/v1, using bearer tokens.

B. Technology Stack

TABLE VI. TECHNOLOGY STACK

Layer	Technology	Purpose
Frontend core	React 18 [9], TypeScript, Vite [16]	Component UI, static typing, fast build and development server
Frontend styling	Tailwind CSS [18]	Utility-first styling with a high-accessibility government-service theme
Routing	React Router 6	Client routes with protected-route guards that preserve navigation state
Data fetching	TanStack Query 5 [17], Axios	Caching, background refresh, centralised interceptors and token attachment
Forms and validation	React Hook Form, Zod [14]	Typed forms and schema validation
Icons	Lucide React	Interface icons
Frontend testing	Vitest [19], React Testing Library, jsdom	Component and page tests
Backend runtime	Node.js 18 or later, TypeScript	Server runtime with static typing
Server framework	Express 4 [15]	HTTP routing and middleware
Database and ORM	PostgreSQL [13], Prisma 5 [12]	Relational storage, typed queries, migrations, pooling
Authentication	jsonwebtoken, bcryptjs	Signed tokens and password hashing
Uploads	Multer	Multipart file handling
Security	Helmet, CORS, express-rate-limit	Headers, origin control and throttling
Logging	Pino and pino-http [21]	Structured logs with request correlation
Validation	Zod [14]	Request body, query and parameter validation
AI	Provider interface with Gemini [23] and mock implementations	Optional language generation
Backend testing	Vitest, Supertest, embedded PostgreSQL	API and integration tests against a real database engine
Delivery	Docker and Docker Compose [20], GitHub Actions	Containers, environment-specific stacks, CI and release workflows

C. Repository Organisation

The repository is a monorepo with a root package that runs both services together. A single setup command installs dependencies and generates the Prisma client, and a single development command starts the backend on port 5001 and the frontend on port 5173.

TABLE VII. REPOSITORY ORGANISATION

Path	Content
backend/src/modules	Ten feature modules (analytics, assistant, auth, dashboard, documents, health, notifications, profile, recommendations, saved-schemes, schemes)
backend/src/middleware	Authentication, authorisation, CORS, error handling, request logging, request identifiers, rate limiting, validation
backend/src/config	Environment, database, security, storage, email, AI and application configuration
backend/prisma	Database schema, seed scripts and scheme data files
backend/tests	Sixteen test files
frontend/src/features	Feature folders for auth, profile, schemes, recommendations, documents, notifications, assistant, analytics, saved schemes and dashboard
frontend/src/components	Shared user-interface components and error and route guards
docs	Architecture, API, database, security, environment, deployment, operations and recovery guides
scripts	Backup, restore and smoke-test scripts
.github/workflows	CI, staging and production workflows

The backend source and tests together comprise roughly 22,000 lines of TypeScript, organised so that every feature has the same internal layout.

D. Backend Request Pipeline

Every request passes through a fixed middleware chain. A request-identifier middleware assigns a correlation identifier that appears in all log lines for that request. The HTTP logger records structured access logs with sensitive headers redacted. Helmet applies security headers. The CORS middleware allows only configured origins. The rate limiter applies the general, authentication or assistant policy according to the route. The authentication middleware verifies the bearer token and loads the principal. The authorisation middleware checks the role against the route's requirement. The validation middleware parses and coerces the body, query and parameters using Zod schemas, rejecting malformed input before it reaches business logic. The controller then delegates to a service, which uses a repository for persistence. A central error handler converts typed application errors (validation, not found, forbidden and so on) into consistent JSON responses and hides internal details in staging and production. A not-found handler covers unknown routes.

E. Data Model

The Prisma schema defines 32 models and 39 enumerations. The models are grouped as follows.

TABLE VIII. DATA MODEL GROUPS

Group	Models
Identity and profile	User, RefreshToken, PasswordResetToken, UserProfile, LocationProfile, EducationProfile, FinancialProfile, SocialEligibilityProfile, OccupationProfile, UserPreference, Consent
Schemes and rules	Department, Scheme, Benefit, DocumentRequirement, ApplicationPeriod, EligibilityRule, SchemeVersion
Engagement	Recommendation, SavedScheme, RecentlyViewedScheme
Documents	UserDocument, DocumentExtraction, DocumentAuditLog
Notifications	Notification, NotificationPreference
Assistant	Conversation, ConversationMessage, AiUsage
Observability	AnalyticsEvent, AuditLog, SystemHealthCheck

Indexes are aligned with query patterns. Users are indexed on email, role, status and creation time for login and administrative filtering. Schemes are indexed on status, verification status, programme type, department, state scope and publication flag for discovery. Recommendations are indexed on user, scheme and eligibility status. Saved schemes have a unique user and scheme pair. Documents are indexed on user, type and verification status. Notifications are indexed on user, read flag and creation time to serve the bell dropdown. Analytics events and audit logs are indexed on type, category, actor, action, entity and timestamp. Connection pooling is configured with minimum 4 and maximum 20 connections and a ten-second timeout, and the client retries with exponential backoff.

F. Authentication and Account Lifecycle

Registration creates a user with a hashed password (bcrypt, cost factor 12, as set in the authentication utilities). Login validates the credentials and returns a pair of tokens: a signed access token that expires after 15 minutes and a refresh token valid for seven days. Only a SHA-256 hash of the refresh token is stored, so a database leak does not expose usable tokens. Refresh requests rotate tokens, and logout revokes the active refresh token. Password reset uses a single-use token record with expiry. Responses on login, registration and password-reset requests are written to avoid revealing whether an email address is registered. The frontend authentication context stores state, attaches the bearer token through an Axios interceptor and handles refresh, while a protected-route component redirects unauthenticated users and preserves their intended destination.

Five roles exist: USER, SCHEME_MANAGER, VERIFIER, ADMIN and SUPER_ADMIN. The authorisation matrix is summarised below.

TABLE IX. AUTHORISATION MATRIX

Endpoint group	Citizen	Scheme manager	Verifier	Admin / Super admin
Public scheme catalogue	Read	Read	Read	Read
Own profile and documents	Self only	Denied	Denied	Denied
Create or edit schemes	Denied	Allowed	Denied	Allowed
Verification reviews	Denied	Denied	Allowed	Allowed
Analytics overview	Denied	Allowed	Allowed	Allowed
Security audit logs	Denied	Denied	Denied	Allowed
System health diagnostics	Denied	Denied	Denied	Allowed

G. Scheme Catalogue Implementation

The repository includes a seed catalogue documented as 162 schemes drawn from central ministries and state welfare directorates, loaded by a seed script in four data files, plus a minimal production baseline seed. The catalogue is organised into 14 categories, and the project documentation records a minimum of 20 schemes per category (a scheme may belong to more than one category, so the category counts below sum to more than the number of distinct schemes).

TABLE X. SEED CATALOGUE BY CATEGORY

Category	Schemes	Examples named in the project documentation
Social Security	45	Atal Pension Yojana, Antyodaya Anna Yojana, NSAP
Financial Assistance	40	PM-KISAN, MUDRA, Stand-Up India, PM SVANidhi
Education	33	Post-Matric scholarships, AICTE Pragati, National Overseas Scholarship
Entrepreneurship	28	PMEGP, Startup India Seed Fund, CGTMSE
Health	26	Ayushman Bharat PM-JAY, Janani Suraksha Yojana
Women Empowerment	24	Sukanya Samriddhi, Beti Bachao Beti Padhao, Mission Shakti
Student Support	22	NMMSS, PMRF, Ishan Uday, CBSE Udaan
Employment	21	MGNREGA, PM Vishwakarma, NAPS
Agriculture and Farming	20	PM Fasal Bima Yojana, Kisan Credit Card, PM-KUSUM
Housing and Shelter	20	PMAY-G, PMAY-U 2.0, PM Surya Ghar
Persons with Disabilities	20	ADIP, Divyangjan Swavalamban, UDID
Senior Citizens	20	PM Vaya Vandana, SCSS, RVY
Skill Development	20	PMKVY 4.0, PM-DAKSH, DDU-GKY
Child Welfare	20	PM CARES for Children, Poshan 2.0, Mission Vatsalya

A data-integrity script checks relationships such as schemes without a valid department and orphaned recommendations, saved schemes, documents and notifications; it does not yet validate rule field names against the field catalogue. Because scheme conditions, amounts and dates change, the seeded data is a starting point that must be re-verified against official sources before any public release.

H. Recommendation Module Implementation

The recommendation module is divided into an engine folder and a service layer. The engine contains the value normaliser, condition evaluator, logical evaluator, explanation generator and rule-engine service. The service layer loads the user's facts and the schemes, calls the engine for each scheme, determines the application status from the scheme's active period, flags schemes expiring soon, and persists the recommendation with the scheme version identifier. Two endpoints expose results: one lists personalised matches, and one returns the full breakdown for a single recommendation.

The explanation generator produces a summary for each result and an explanation for each condition. For a failed condition it states what was required and what the profile contains. For an unknown condition it states what information is needed. The frontend recommendation pages present a status badge, the match explanation, and a missing-information card that links the citizen back to the relevant part of the profile.

I. Frontend Implementation

The frontend defines routes for the home page, authentication (login, registration, forgot and reset password), dashboard, profile, documents, notifications and notification settings, assistant, scheme catalogue and detail, recommendations and detail, saved schemes, comparison, and three administrative areas for scheme management, analytics and audit logs, and system health. Routes that require login are wrapped in a protected-route guard, and a root error boundary prevents a component failure from blanking the entire application.

Shared interface components (button, input, select, textarea, card, badge, modal, alert, toast, skeleton, spinner, empty state, error state and page container) provide a consistent, accessible appearance and standard loading, empty and error behaviour. Feature code is organised by domain with its own API client, hooks, components, pages and tests. TanStack Query manages server state, so that lists are cached and refreshed and mutations invalidate affected queries. Types for API payloads are declared centrally.

Specific screens include the scheme catalogue with filters, scheme detail with benefits, documents and a verification modal, a compare tray that allows several schemes to be added and compared, a recommendation list with status badges, a document checklist that shows which required documents the citizen has, a document upload modal and an extraction review modal for confirming fields, a notification bell with dropdown and unread badge, a notification page with filters, a floating assistant button with a chat panel, suggested questions and source

references, and administrative dashboards with metric cards, charts, an audit-log table and system-health status.

J. Document Module Implementation

The document service coordinates the catalogue, storage, OCR provider, repository and notifications. The storage service resolves a configurable root directory, validates the magic bytes of PDF, JPEG and PNG files, computes a SHA-256 checksum, generates a non-user-controlled storage key and writes the file. Download tokens are signed JSON Web Tokens that carry the user, document and storage key with a five-minute lifetime, and they are verified before streaming the file. A cleanup routine removes orphaned temporary uploads. After extraction, the service stores a document extraction record with structured fields, confidence, provider and version. If confidence is low the user is told to review carefully, and if processing fails a notification is created. Confirming fields maps values to profile attributes through the catalogue's mapping and writes a document audit entry.

The extraction provider is an interface with a single method, which returns text, structured fields, confidence, provider identity and a low-confidence flag. The registered implementation is the local heuristic provider described earlier. It does not read the pixels of the uploaded file. For this reason, the project treats real optical character recognition as outstanding work, and the confirmation step is mandatory so that no extracted value reaches a profile without a person approving it.

K. Notification Implementation

A scheduler class defines the four jobs, which an administrator can run on demand through an endpoint; the repository does not yet attach them to an automatic timer. The deadline-reminder job inspects application periods and creates alerts at configured thresholds, with a separate alert on the day of closure. The upcoming-opening job notifies users who have saved or been recommended a scheme when its window is about to open. The document-expiry job uses each document type's default validity (for example, twelve months for an income certificate) to warn before and after expiry. The cleanup job removes old notifications. An event service generates notifications when recommendations change or documents finish processing. Notifications are de-duplicated and respect each user's preferences, and the channel interface keeps dispatch separate from creation so that a failing email provider does not lose the in-app record.

L. Assistant Implementation

The assistant module has a controller, service, grounding service, prompt library, repository, validators and a provider factory. The grounding service detects the intent from explicit context or from key phrases in the message, then calls the relevant tool: scheme, recommendation, document, notification or profile. The provider factory returns either the Gemini provider or the mock provider according to configuration. Conversations and messages are stored so that history can be listed, reopened or deleted by the user. Usage records capture requests, latency and token use for the analytics module. The frontend panel shows messages with source references and suggested follow-up questions.

M. Analytics and System Health Implementation

The analytics service computes aggregate queries for each dashboard tab, and the export service produces CSV or JSON files and writes an audit entry for each export. The system-health service measures database latency and storage read and write access. The OCR, AI, notification and scheduler entries, and the latency percentiles it reports, are currently fixed placeholder values rather than live probes, and results are not yet written to the health table. Public probes expose process liveness and service readiness for orchestrators, while the detailed diagnostics endpoint requires an administrator.

N. Configuration, Environments and Delivery

Environment variables are validated at start-up with a schema so that misconfiguration fails fast. Separate environment templates exist for development, staging and production, and a reference document lists the variables. The production compose file defines the frontend, backend and database services and runs application containers as a non-root user. Three GitHub Actions workflows cover continuous integration, staging and production. The continuous-integration workflow is intended to lint, type-check, test and build both applications. Operational scripts handle database backup with thirty-day retention, restoration with the integrity validator, and a smoke test against a running server. Secrets such as the JWT secrets, database password and AI key are supplied through environment variables and are not committed in the templates; a deployment must generate strong values, for example with a cryptographic random generator.

O. Code Quality Practices

The backend uses ESLint with Prettier, and the frontend uses ESLint with a zero-warning policy. TypeScript is configured in strict mode on both sides. Formatting checks run through a single command. Because validation, authorisation and error handling are centralised in middleware, individual features contain little repeated security code.

Figures 1–4 summarise the processing pipeline, the data relationships, the eligibility decision and the verification boundaries. The backend enforces validation, authorisation and consent; the frontend exposes explanations, uncertainty and permitted actions.

V. ADVANTAGES

1. Explainable and Honest Eligibility

Condition-level explanations. A citizen sees which conditions matched, which failed and which could not be assessed, and a plain-language reason for each. This turns a verdict into guidance on what to do next.

Uncertainty is visible. Missing profile information produces UNKNOWN rather than a silent pass or fail, and the portal lists the facts that would resolve the uncertainty. This avoids both false encouragement and false discouragement.

Reproducibility. Because evaluation is a pure function of the rules and facts, and because each recommendation stores the scheme version used, the same inputs always give the same result and the basis of a result can be reconstructed.

2. Safe Use of AI

AI is confined to assistance. The assistant explains, summarises and guides, but it cannot change an eligibility status, and its answers are built from data fetched for the current user.

Graceful degradation. If the model provider is unavailable, an advisory message is returned and every core function continues to work.

Cost and control. A dedicated rate limiter and usage records bound consumption and make assistant behaviour measurable.

3. Document Readiness

Checklist before application. Citizens can see which certificates a scheme needs and which they already hold, reducing last-minute failures.

Human confirmation. Extracted values reach the profile only after explicit user confirmation with consent, which reduces the effect of extraction errors.

Protected storage. Signature validation, generated storage keys, checksums and short-lived signed download tokens reduce upload-related risks.

4. Timely Notifications

Scheduled jobs warn users of openings, approaching and same-day deadlines, deadline changes and document expiry. Preferences allow users to control channels, and the in-app channel is a reliable fallback.

5. Security and Privacy by Design

The design uses short-lived tokens with rotation, hashed refresh tokens, adaptive password hashing, role-based access, validated input, hardened headers, strict origins, rate limiting, enumeration-safe responses, redacted logs. An append-only audit log records access to administrative analytics, audit views and exports. Consent records and data minimisation follow the principles of the DPDP Act [8].

6. Maintainability and Extensibility

Because the code is split into feature folders with typed schemas, shared middleware and provider interfaces, a newcomer can follow it quickly. Swapping in a real OCR engine, another AI provider or a new notification channel means writing one implementation of an existing interface; callers stay unchanged.

7. Operational Readiness

The repository ships container definitions, per-environment templates, health probes, structured logs, backup and restore scripts, a smoke test, a written recovery plan and CI workflows, which together let a team run it. Recovery targets are stated outright, and the failure of assistive components is handled by design.

8. Administrative Insight

Administrators can see funnels, eligibility distributions, document outcomes, notification read rates, assistant usage and zero-result searches. Such figures show where citizens get stuck, for instance which missing profile fields most often hold up a decision, and point to where the catalogue needs work.

Figure 1 Portal processing
Consented profile to an explained recommendation



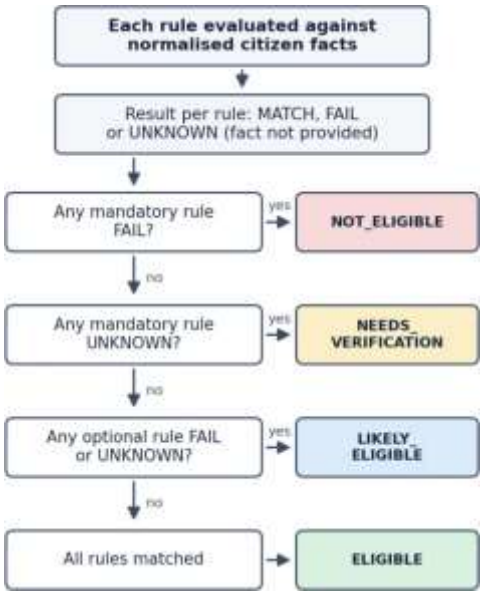
Missing information is reported as unknown; assistance modules do not decide eligibility.

Figure 2 Data relationships
Durable records with explicit ownership



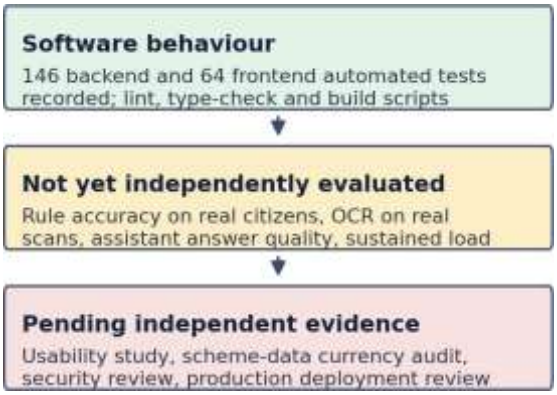
Each recommendation stores the scheme version it was evaluated against.

Figure 3 Eligibility decision
Failure dominates; unknown never passes



Applied to every scheme; status describes rule outcomes, not approval.

Figure 4 Verification boundaries
Measured local results and their scope



Tests do not establish accuracy on real citizens or deployment readiness.

VI. RESULTS AND ANALYSIS

This section reports what the project's records and code demonstrate. It separates **implementation completeness**, **automated verification** and **behavioural analysis of the eligibility engine**, and it states clearly what has not been measured. The test counts below are those recorded in the repository's documentation and test files; they should be re-run and confirmed on the submission date.

A. Implementation Completeness

All ten planned modules are marked complete in the project documentation. The delivered system comprises the following artefacts.

TABLE XI. DELIVERED ARTEFACTS

Item	Result
Modules completed	10 of 10
Database models / enumerations	32 models, 39 enumerations
Seed catalogue	162 schemes documented, in 14 categories
Eligibility operators	12 (EQUALS through NOT_EXISTS)
Eligibility outcomes	4 (Eligible, Likely Eligible, Needs Verification, Not Eligible)
Notification types	16
User roles	5
Frontend routes	25 route definitions including administration and a not-found page
REST endpoint groups	10, under /api/v1
Backend test files / tests	16 files, 146 tests
Frontend test files / tests	20 files, 64 tests (as recorded in the documentation)
Deployment	Docker Compose for development and production; CI, staging and production workflows

Figures 5 to 11 show the main screens of the running system: the home page, citizen sign-in, dashboard, scheme catalogue, comparison matrix, saved schemes and personalised recommendations. For the demonstration profile shown in the dashboard and recommendation screens, the portal evaluated 162 schemes and reported 64 Eligible, 0 Likely Eligible, 37 Needs Verification and 61 Not Eligible. This is a single-profile observation that shows the four outcomes in use; it is not a measure of recommendation accuracy.

B. Automated Verification

The repository records a combined 210 automated tests (146 backend and 64 frontend) and a zero-warning lint policy. The backend suite uses Vitest and Supertest with an embedded PostgreSQL engine so that API tests exercise real SQL behaviour. The distribution of backend tests across test files is shown below.

TABLE XII. BACKEND AUTOMATED TESTS BY FILE

Backend test file	Tests	Area covered
notifications.test.ts	20	Alert creation, jobs, preferences
analytics.test.ts	19	Metrics, audit logs, export
documents.test.ts	18	Upload, validation, extraction, confirmation
assistant.test.ts	14	Grounding, intent, provider behaviour
recommendations.test.ts	14	Rule engine and recommendation API
schemes.test.ts	12	Catalogue, search, administration
production-config.test.ts	10	Production configuration safeguards
saved-schemes.test.ts	9	Saving and listing
auth.test.ts	6	Registration, login, tokens
schemes-compare.test.ts	5	Comparison
authorization.test.ts	4	Role enforcement
health.test.ts	4	Probes
profile.test.ts	4	Profile updates
dashboard.test.ts	3	Dashboard aggregation
env.test.ts	2	Environment validation
errors.test.ts	2	Error handling
Total	146	

Frontend tests cover the button component, error boundary, protected route, login page, schemes catalogue, scheme detail and comparison pages, recommendation pages, document checklist and documents page, notification bell, pages and settings, assistant panel and page, saved schemes, audit log table, the administrative analytics dashboard and the dashboard page.

These results show that the implemented behaviours match the tests written for them. They do not show that every possible path is covered, and no coverage percentage is reported here because a coverage run is not part of the recorded results.

C. Behavioural Analysis of the Eligibility Engine

Because the engine is deterministic, its behaviour can be analysed by cases rather than statistical sampling. The following table lists representative situations and the result the algorithm must produce. They are derived from the algorithm's definition and are illustrative; they are not field measurements.

TABLE XIII. BEHAVIOURAL CASES OF THE ELIGIBILITY ENGINE

Situation	Mandatory failures	Mandatory unknowns	Optional issues	Result
All mandatory rules match; all optional rules match	0	0	0	ELIGIBLE
All mandatory rules match; one optional rule fails	0	0	1	LIKELY_ELIGIBLE
All mandatory rules match; an optional fact is missing	0	0	1	LIKELY_ELIGIBLE
One mandatory fact missing; others match	0	1	0	NEEDS_VERIFICATION
One mandatory rule fails; another mandatory fact missing	1	1	0	NOT_ELIGIBLE
Income above the stated limit	1	0	0	NOT_ELIGIBLE
Scheme has no rules	0	0	0	ELIGIBLE (open to all)

Three properties follow directly from the design.

1. **Monotonic safety with respect to missing data.** A missing mandatory fact can never produce ELIGIBLE. It produces NEEDS_VERIFICATION unless another mandatory condition definitely fails.
2. **Definite failure dominates.** A single failed mandatory condition yields NOT_ELIGIBLE regardless of

unknowns, which prevents users spending effort on schemes they cannot receive.

3. **Determinism.** Re-evaluating the same facts against the same scheme version always gives the same result, and the trace counters make the decision auditable.

In practice, profile completeness therefore determines how many schemes appear as NEEDS_VERIFICATION. A user who completes their profile moves schemes out of that status into a definite one. The administrative recommendation analytics are designed to report the distribution of outcomes and the most frequent failure criteria, which would allow this effect to be measured once real usage data exists.

D. Security Posture Analysis

The implemented controls address the principal OWASP categories [5] as follows.

TABLE XIV. MITIGATION OF OWASP RISK CATEGORIES

OWASP risk	Mitigation implemented
Broken access control	Role middleware on routes; citizen data scoped to owner; administrative routes restricted
Cryptographic failures	bcrypt passwords; signed tokens; hashed refresh tokens; signed, expiring download tokens
Injection	Prisma parameterised queries; Zod validation of all inputs
Insecure design	Separation of ground truth and assistance; consent; data minimisation; graceful degradation
Security misconfiguration	Helmet headers; schema-validated environment; error sanitisation; non-root containers
Identification and authentication failures	Rate-limited authentication routes; rotating refresh tokens; enumeration-safe responses
Software and data integrity	Checksums on stored files; file signature validation; audit trail
Logging and monitoring failures	Structured, redacted logs with correlation identifiers; append-only audit log; health probes

The repository documents a strict policy that identity numbers, bank details, tokens and raw OCR payloads are not logged or placed in analytics events. These controls were implemented and tested at the level described in Section VI-B. No independent penetration test has been carried out, so the results should not be read as a security certification.

E. Performance Considerations

The documentation states a design target of sub-100-millisecond responses for indexed queries, supported by indexes matched to query patterns and by connection pooling. No benchmark or load-test result is recorded in the repository, so this figure is a design goal and is not reported here as a measured result. Evaluation of every scheme for a user is a bounded computation that grows linearly with the number of schemes and rules, and the catalogue of about 160 schemes is small relative to the capacity of the database and runtime.

F. Limitations Identified

A candid analysis of the delivered system identifies the following limitations.

1. **Extraction is a stand-in.** The registered OCR provider returns type-specific sample values and does not analyse image content. The confirmation workflow is complete, but real extraction accuracy has not been evaluated. The system should not be presented as performing true OCR until a real engine is integrated and measured.
2. **Rule accuracy depends on curation.** Eligibility rules were authored from published scheme descriptions. They have not been independently reviewed by the awarding departments, and scheme terms change over time.
3. **No user study.** Usability, comprehension of explanations and the effect on application rates have not been measured with real users.
4. **No recorded load or penetration testing.** Scalability and security assurance beyond the implemented controls remain unmeasured.
5. **Assistant quality is unmeasured.** Grounding and validation reduce unsupported statements, but multilingual answer quality has not been systematically assessed.
6. **Documentation inconsistencies.** The security guide states a bcrypt cost of 10 whereas the authentication code uses 12, and some documentation refers to services by their planned deployment rather than the tested local setup. The architecture guide also describes an audit entry for every administrative mutation, whereas the code writes audit entries only for analytics, audit-view and export actions, and the security guide mentions encrypted document directories although the storage layer writes files without application-level encryption. All of these should be aligned before submission.
7. **Seed rules use fields the engine does not recognise.** Of the 212 rules in the seed catalogue, 37 (in 37 schemes) test the fields is_disabled, farmer_status or caste_category, which are absent from the field catalogue and from the fact normaliser, so they always evaluate to UNKNOWN. For a complete profile this forces NEEDS_VERIFICATION on those schemes unless another mandatory rule fails, and it explains why the demonstration profile shows 37 Needs Verification results. Mapping the three fields to the existing disability, social category and occupation facts, or renaming them in the seed data, would resolve this; this was identified by running the rule engine over the seeded catalogue.
8. **Seed document types do not match the upload catalogue.** Only 79 of the 231 document requirements in the seed data (34.2 per cent) use one of the 11 types accepted by the upload catalogue, and 120 schemes list at least one mandatory requirement that a citizen cannot upload under that type name, so the readiness checklist will report it as missing.
9. **All seeded rules are mandatory.** Every one of the 212 seed rules carries the mandatory flag, so the LIKELY_ELIGIBLE outcome cannot occur on seed data, which agrees with the 0 Likely Eligible count in

the demonstration. Geographic scope is also not tested: no seed rule refers to state or district, so a scheme limited to one region can appear as Eligible for a citizen elsewhere.

10. **Indicative output only.** Results are guidance and never a guarantee of approval.

These limitations do not undermine the central design result, which is that the engine, the document workflow and the assistant behave according to their specifications and that uncertainty is surfaced rather than hidden. They define the evaluation agenda for future work.

VII. CONCLUSION

The Smart Scheme & Scholarship Recommendation Portal shows one workable way to help citizens find and prepare for government benefits while remaining trustworthy. Its core is a deterministic, explainable eligibility engine that uses three-valued logic, so that a missing fact becomes an explicit request for information instead of a misleading verdict. Around this core, the system provides a verified, versioned scheme catalogue, a structured and consent-based citizen profile, a secure document vault with a mandatory confirmation step, scheduled deadline and expiry notifications, a grounded AI assistant that may explain but never decide, and administrative analytics with an append-only audit trail.

The architecture is a modular monolith in React, TypeScript, Express, Prisma and PostgreSQL, organised into ten modules with a deliberate separation between ground truth and assistance. This separation allows assistive components such as the AI provider or OCR engine to fail without affecting discovery or eligibility. Security and operational concerns are treated as design requirements, with short-lived and rotating tokens, role-based access, validated input, hardened headers, rate limiting, redacted logging, containerised deployment, backups and a documented recovery plan. The repository records 210 automated tests and a zero-warning lint policy.

The work also has clear boundaries. The document extraction provider is currently a heuristic stand-in, rules have not been validated by awarding authorities, and no user study, load test or penetration test has been performed. The portal's guidance is indicative, and final decisions remain with the responsible departments and institutions. Within these limits, the project shows that explainable rule evaluation, careful handling of missing information and restricted use of AI can be combined into a platform that is both useful and honest about what it does not know.

VIII. FUTURE WORK

Real document extraction. The highest-priority technical task is to replace the heuristic provider with a real OCR engine, such as Tesseract [4] or a managed document service, behind the existing provider interface. Extraction accuracy should then be measured on consented, redacted samples of income, community and academic certificates, reporting field-level precision and recall by document type and scan quality.

Rule validation and catalogue governance. Eligibility rules should be reviewed against official notifications by independent reviewers, with disagreement recorded. A scheduled re-verification process, change alerts for source pages and an editorial workflow with approval would keep the catalogue current. Rule authoring could be supported by a visual builder that previews how a rule evaluates for sample profiles. The three-valued AND and OR evaluator should also be wired into rule-group evaluation so that OR clusters of conditions can influence the final status.

User evaluation. A usability study with students and citizens should measure whether explanations are understood, whether profile completion improves, how often users reach an official application, and where they abandon the flow. The administrative analytics already capture funnels that can support this study.

Performance and security assurance. Load tests should establish response times and capacity with the full catalogue and many concurrent users. An independent penetration test and dependency audit should precede any public release, and refresh-token theft detection, multi-factor authentication and field-level encryption of sensitive attributes should be considered.

Assistant quality. The assistant should be evaluated with a labelled question set covering eligibility explanation, application steps and document guidance, measuring groundedness and refusal behaviour. Multilingual output, particularly for Tamil and Hindi, requires review by native speakers, and voice input could improve access for low-literacy users.

Reach and accessibility. Offline scheme browsing through a progressive web app, SMS or WhatsApp alerts, formal accessibility audits and a low-bandwidth mode would help the portal reach rural users.

Integration. Where official interfaces allow, the portal could link to application portals with pre-filled information, support document verification through authorised services, and import scheme updates from official feeds. Any such integration would require careful legal and consent review.

Operational maturity. A staged deployment with monitoring, alerting, secret management, regular restore drills and a documented incident process would be required before public operation. Documentation should be aligned with the implemented configuration, and secret values should be kept out of version control.

Throughout, future development should preserve the central principle of the project: information that is unavailable must be shown as unknown, never as a favourable outcome.



Fig. 5. Home page of the portal.



Fig. 6. Citizen sign-in page.

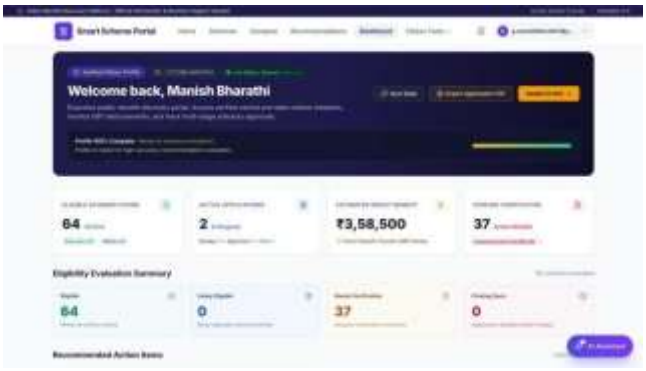


Fig. 7. Citizen dashboard with eligibility summary.



Fig. 8. Public scheme and scholarship discovery directory.



Fig. 9. Statutory scheme comparison matrix.

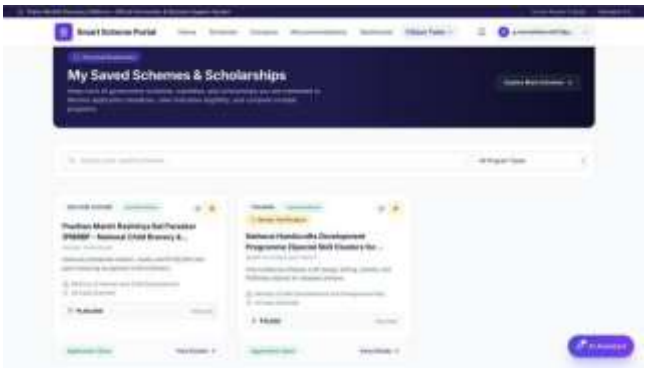


Fig. 10. Saved schemes and scholarships.

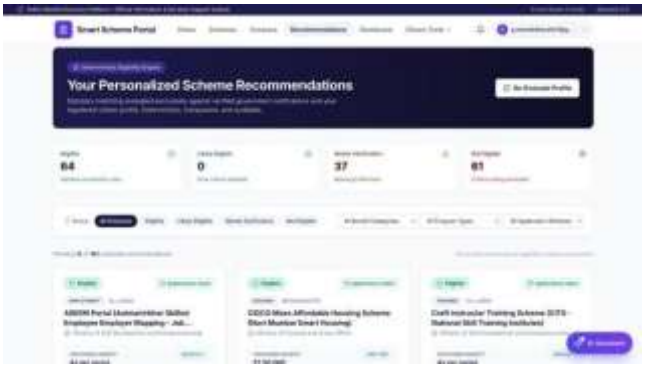


Fig. 11. Personalised scheme recommendations with four eligibility outcomes.

REFERENCES

[1] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

[2] F. Ricci, L. Rokach and B. Shapira (eds.). *Recommender Systems Handbook*, 2nd ed. Springer, 2015.

[3] P. Lewis et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33, 2020, pp. 9459-9474. https://arxiv.org/abs/2005.11401

[4] Tesseract OCR contributors. Tesseract User Manual. Official documentation. https://tesseract-ocr.github.io/tessdoc/

[5] OWASP Foundation. OWASP Top 10 Web Application Security Risks. https://owasp.org/www-project-top-ten/

[6] M. Jones, J. Bradley and N. Sakimura. JSON Web Token (JWT). RFC 7519, IETF, 2015. https://www.rfc-editor.org/rfc/rfc7519

[7] N. Provos and D. Mazieres. A Future-Adaptable Password Scheme. *Proceedings of the USENIX Annual Technical Conference*, 1999.

[8] Government of India. The Digital Personal Data Protection Act, 2023. Ministry of Electronics and Information Technology.

[9] Meta Open Source. React Documentation. https://react.dev

[10] Government of India. National Scholarship Portal. https://scholarships.gov.in

[11] Government of India. myScheme: Scheme Discovery Platform. https://www.myscheme.gov.in

[12] Prisma. Prisma ORM Documentation. https://www.prisma.io/docs

[13] PostgreSQL Global Development Group. PostgreSQL Documentation. https://www.postgresql.org/docs/

[14] Zod contributors. Zod: TypeScript-first Schema Validation. https://zod.dev

[15] OpenJS Foundation. Express Documentation. https://expressjs.com

[16] Vite contributors. Vite Documentation. https://vite.dev

[17] TanStack. TanStack Query Documentation. https://tanstack.com/query/latest

[18] Tailwind Labs. Tailwind CSS Documentation. https://tailwindcss.com/docs

[19] Vitest contributors. Vitest Documentation. https://vitest.dev

[20] Docker Inc. Docker Documentation. https://docs.docker.com

[21] Pino contributors. Pino Logger Documentation. https://getpino.io

[22] Helmet contributors. Helmet.js Documentation. https://helmetjs.github.io

[23] Google. Gemini API Documentation. https://ai.google.dev/gemini-api/docs

[24] Project implementation records. Architecture, Security, Database, API, Operations, Deployment and Disaster Recovery guides, and README, Smart Scheme & Scholarship Recommendation Portal repository, 2026.