IJIRAE:: International Journal of Innovative Research in Advanced Engineering ISSN: 2349-2163
Volume 12, Issue 11, November 2025 https://www.ijirae.com/archives
https://doi.org/10.26562/ijirae.2025.v1211.06

# Scholar Nest & Scholar Ship Agent: AI Powered Scholarship Eligibility Assessment System for Education Equity

**Roopesh Kumar BN**
Associate Professor Dept. of CSE
KS Institute of Technology, Bengaluru, India
roopeshkumarbn@ksit.edu.in,
**Anubhav Misra, Syed Ayan Hyder, Saketh AV, Thejus K**
Department of Computer Science & Engineering
KS Institute of Technology, Bengaluru, India
anubhavmisra_cse2022@ksit.edu.in, syedayanhyder_cse2022@ksit.edu.in
sakethav_cse2022@ksit.edu.in, thejusk_cse2022@ksit.edu.in

**Publication History**
Manuscript Reference No: IJIRAE/RS/Vol.12/Issue11/NVAE10085
Research Article| Open Access | Double-Blind Peer-Reviewed | Article ID: IJIRAE/RS/Vol.12/Issue11/NVAE10085
Received:22,October 2025,Revised:28,October 2025, Accepted:31, October 2025, Published Online: 21, November 2025.
https://www.ijirae.com/volumes/Vol12/iss-11/06.NVAE10085.pdf
**Citation:Roopesh,Thejus K,Anubhav Misra, Syed Ayan Hyder, Saketh (2025),**Scholar Nest & Scholar Ship Agent: AI Powered Scholarship Eligibility Assessment System for Education Equity, IJIRAE: International Journal of Innovative Research in Advanced Engineering, Volume 12, Issue 11 of 2025 pages 468-473
**doi:>**https://doi.org/10.26562/ijirae.2025.v1211.06
**BibTeX Key: Roopesh@2025** Scholar
IJIRAE papers should be cited as IJIRAE (International Journal of Innovative Research in Advanced Engineering, AM Publications, India 2025, ISSN 2349-2163, https://doi.org/10.26562/ijirae.2025.v1211.06 The journal's official abbreviation is IJIRAE. Orcid: https://orcid.org/0009-0004-9398-7488


**Abstract**: This paper presents the design, implementation and deployment of Scholar Nest, an AI powered web application that matches students in Karnataka with suitable scholarships. Scholar Nest combines a typed React frontend built with Vite, TypeScript, Tailwind CSS and shadcnui with a Supabase backend providing a PostgreSQL database, authentication, storage and row level security (RLS) policies. A separate Scholar Ship Agent, written in Python using Lang Chain and generative AI models, automates the discovery and extraction of scholarship information from official portals and documents. The system encodes student eligibility criteria, compares them to structured scholarship metadata and returns ranked results in a responsive dashboard. Implementation details, data model, agent pipeline, performance considerations, accessibility measures, security and deployment practices are documented in depth. The project demonstrates how open source technologies can be combined to promote education equity by reducing search time and improving access to financial aid opportunities.

## I. INTRODUCTION

Students in Karnataka face challenges when searching for scholarships due to scattered portals, lengthy eligibility criteria and a lack of visibility for niche programmes. The Scholar Nest project addresses this problem by building a web based platform that collects data from the State Scholarship Portal (SSP), National Scholarship Portal (NSP) and university bulletins, and matches student profiles against the eligibility conditions of each scholarship. The core idea is to encode student attributes such as gender, caste, religion, income, academic level and disability and compare them to structured scholarship metadata using similarity and rule based filters. The platform aims to reduce search time by 90%, improve accuracy of recommendations (self reported at 95 %) and increase access to financial aid for marginalized students. By automating data collection and matching, the project promotes education equity, reduces financial strain and supports colleges, government departments and NGOs. Scholar Nest comprises three major components: (i) a typed React front end built with Vite, TypeScript, Tailwind CSS and shadcnui; (ii) a Supabase backend providing a Post greSQL database, authentication, storage and RLS policies; and (iii) a Scholar Ship Agent implemented in Python that schedules scraping jobs, calls a large language model (LLM) to extract scholarship information and writes validated records back to the database. The integration of these components yields a full stack system that delivers a responsive user experience while enforcing strict access controls and enabling automation.

## II. RELATED WORK & TECHNOLOGY RATIONALE

Building an eligibility matching platform requires choices about the front end framework, backend services and automation tooling. Vite is chosen as the build tool because it provides a faster and leaner development experience for modern web projects; it uses native ES modules in the browser during development and employees build for pre bundling dependencies, resulting in sub second startup times.

IJIRAE:: International Journal of Innovative Research in Advanced Engineering ISSN: 2349-2163
Volume 12, Issue 11, November 2025 https://www.ijirae.com/archives
https://doi.org/10.26562/ijirae.2025.v1211.06

React offers a component based library for constructing user interfaces, and TypeScript adds static typing and tooling support that improves maintainability. Tailwind CSS is a utility first framework that accelerates custom UI design, and shadcnui provides headless RadixUI components styled with Tailwind. Supa base is selected for the back end because it bundles a hosted Post greSQL database, authentication and storage with a straightforward JavaScript client. Post greSQL supports row security policies that restrict which rows are returned or modified based on user identity, enabling fine grained access control. The Scholar Ship Agent uses Python with Lang Chain to orchestrate calls to generative models and scraping pipelines; although most Lang Chain documentation is inaccessible from within this environment, it is understood as a modular frame work for composing LLM calls and tools. The agent uses Google's Gemini model and a Tavily search API, and normalizes extracted data using Pydantic models and heuristics. Security guidance is drawn from the OWASP Injection Prevention cheat sheet, which recommends validating inputs, using parameterised queries and contextually escaping data. For accessibility, guidance from the WAIARIA specification is followed; ARIA roles provide semantic meaning to content so assistive technologies can interpret interactions.

## III. SYSTEM OVERVIEW

Scholar Nest is deployed as a static single page application on Vercel. The browser loads the React application, which uses the Supabase JavaScript client to authenticate users, fetch scholarship data, save user eligibility information and manage bookmarks. Supabase hosts a Postgre SQL database with RLS enabled on all user facing tables, an authentication service that issues JWTs, and storage buckets for document uploads. The Scholar Ship Agent runs as a scheduled Python process that discovers new scholar ships, extracts metadata using an LLM and writes validated records to the Supabase database. Figure1 shows the high level architecture, and the sequence in Figure4 illustrates how the agent and frontend interact through Supabase.

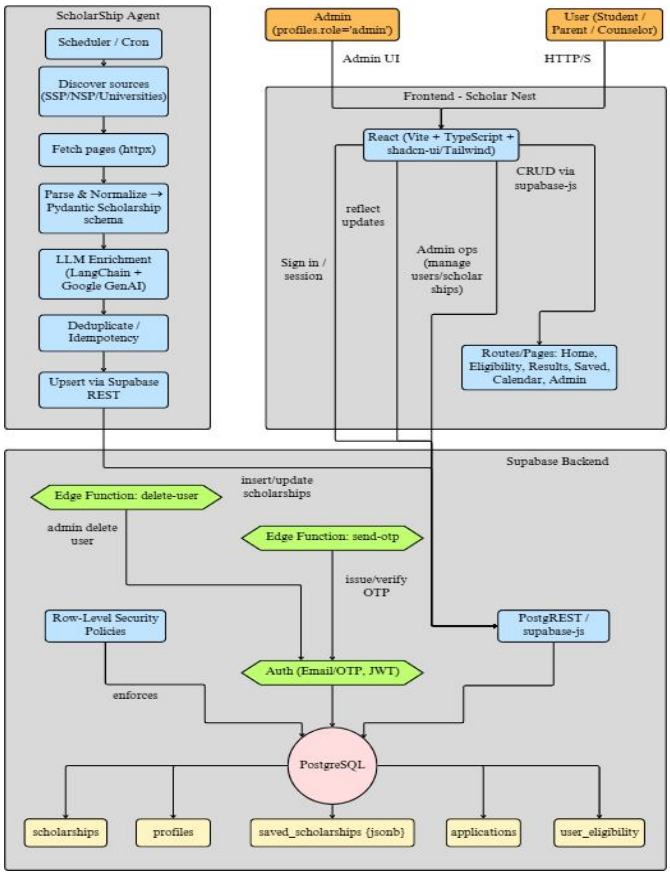


**Fig.1.** High Level Architecture Diagram

## IV. DATA MODEL & BACKEND

The Supabase schema evolved through a series of migrations. The key tables are summarised below and illustrated in Figure 2.

- profiles stores user information (id, email, role, full_name) and is used by Supabase Auth. Row security policy ensures users can only read their own profile.
- scholarships – holds scholarship metadata (id, name, provider, category, level, amount, application_deadline, eligibility JSON, requirements JSON). After a migration, the id changed from UUID to an auto increment integer for compatibility with third party imports. Public RLS allows anyone to read scholarships.
- saved_scholarships – maps users to scholarships they bookmark. Policies restrict inserts and deletes to the owner; administrators have full access.

IJIRAE:: International Journal of Innovative Research in Advanced Engineering ISSN: 2349-2163
Volume 12, Issue 11, November 2025 https://www.ijirae.com/archives
https://doi.org/10.26562/ijirae.2025.v1211.06

- Applications records scholarship applications submitted by users with fields for status, progress, number of documents submitted and timestamps. RLS restricts rows to the owner or admins.
- user_eligibility stores eligibility form responses (gender, caste, religion, class, percentage, income, disability, special reservations) and is updated whenever the user edits the form. Only the record owner can insert or update.
- admin_logs – logs actions performed through the admin panel with fields for admin_id, action, resource, timestamp and description. RLS restricts access to admins.
- One time pass codes but later dropped and replaced with secure functions. Edge functions like delete-user delete a user from Supabase Auth using a service role key, and send-otp calls a Resend API to email an OTP.

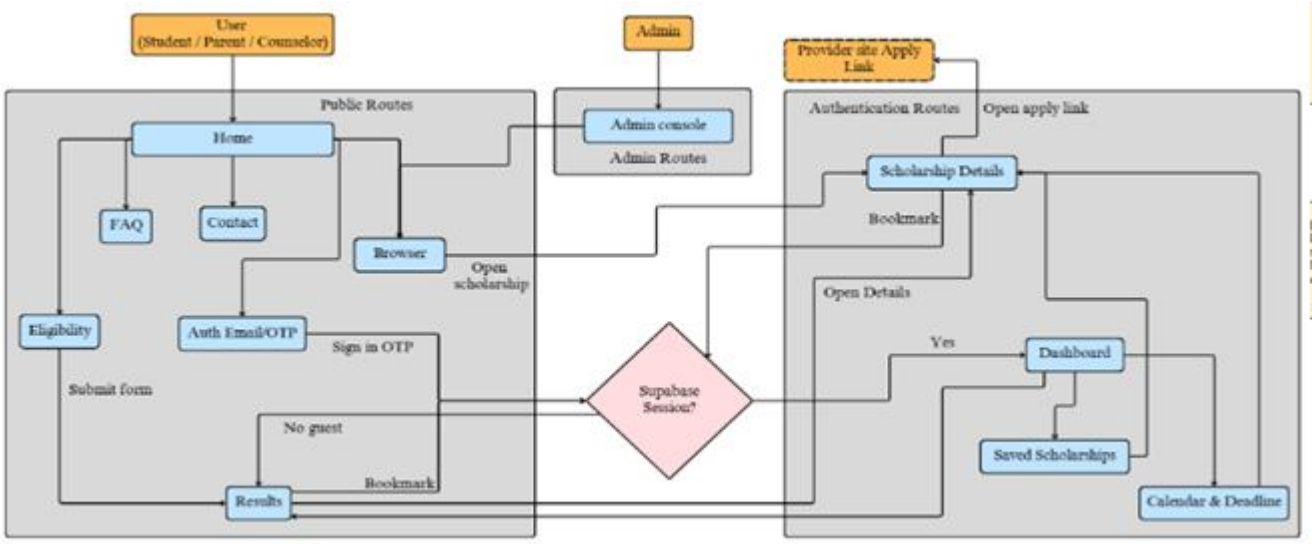


**Fig.3.**User Flow Diagram

## V. FRONT END IMPLEMENTATION (SCHOLAR NEST)

The front end is bootstrapped using Vite and Type Script. package.json lists dependencies such as React, React Router, SupabaseJS, TanStack React Query and shadcnui, while dev dependencies include Tailwind CSS, Post CSS and ESLint. vite.config.ts configures the React plugin and defines an alias for @/pointing to the src directory. Tailwind is configured in tailwind.config.ts with dark mode support, custom colours (e.g., scholar blue, scholar purple) and animation key frames.

### A. Routing and Pages

React Router defines routes for the home page (Index.tsx), authentication (Auth.tsx), eligibility form (Eligibility.tsx), results (Results.tsx), scholarship browsing (Browse.tsx), calendar view (Calendar.tsx), saved scholarships (SavedScholarships.tsx), user dashboard (Dashboard.tsx), admin panel (Admin.tsx) and informational pages such as FAQ and Contact. The eligibility form collects gender, caste, religion, academic level, marks, and income, disability and reservation details and stores them in user_eligibility via the use Eligibility hook. The results page loads the form data from local Storage, filters the imported scholarships Data array using multiple conditions and displays eligible scholarships.

### B. State Management and Hooks

Custom hooks wrap Supabase queries and React Query for caching. use Auth manages authentication state and exposes sign Up, sign In and sign Out functions. Use Applications provides CRUD operations on the applications table and joins scholarship details. Use Bookmarks synchronises bookmarks between saved_scholar ships and local Storage, enabling anonymous users to bookmark. Use Eligibility fetches and updates the user's eligibility form, while use Email Notifications mocks a notification scheduler using local Storage. A toast system built with a reducer (useToast) offers global notifications.

### C. Components and UI

The application uses shadcnui components (Accordion, Alert, Button, Card, form controls, Tabs, etc.) styled with Tailwind. The Navbar renders dynamic links and checks if a signed in user has an admin profile by querying the profile stable. Scholar ship Card displays scholarship information with badges for category and level, deadlines, amounts and a bookmark button. The Dashboard shows an overview of applications, statuses and upcoming deadlines and provides quick actions for eligibility check, browsing and saved scholar ships. Calendar and FAQ pages render dynamic lists and user input. The admin panel uses tabs to switch between analytics, scholarships, users, logs and settings.

### D. Accessibility

All interactive elements use semantic HTML elements and ARIA roles. Components like Accordion and Tabs are built atop Radix primitives that implement key board navigation and ARIA roles. MDN notes that ARIA roles add semantic meaning to content and help assistive technologies interpret interactions. Form fields use <label>elements connected to inputs, and toast notifications announce status changes. Colour palettes meet contrast ratios for light and dark themes.

IJIRAE:: International Journal of Innovative Research in Advanced Engineering ISSN: 2349-2163
Volume 12, Issue 11, November 2025 https://www.ijirae.com/archives
https://doi.org/10.26562/ijirae.2025.v1211.06

The application supports dark mode and responsive design through Tailwind's utility classes and custom hooks (e.g., usels Mobile).

## VI. SCHOLARSHIP AGENT IMPLEMENTATION

The Scholar Ship Agent is a Python package that automates the discovery and extraction of scholarship information. It uses a configuration file (config.py) to load environment variables such as GOOGLE_API_KEY, TAVILY_API_KEY and model names. The agent's pipeline comprises the following modules:

- discover.py – builds search queries and calls the Tavily search API to retrieve candidate URLs. The discover_urls function filters duplicates, favours official domains and ensures URLs return an HTTP 200 response.
- extract.py – contains heuristics to score official pages, functions to fetch and parse HTML/PDF content, and a Lang Chain based extractor chain. The extract_from_url function uses the model to extract scholarship fields (name, provider, amount, eligibility, etc.), canonicalises dates and deduplicates results. Pydantic models defined in models.py validate the extracted data.
- utils.py – provides helper functions for URL normalisation, deduplication and HTTP checks.
- mcp_client.py – interacts with an optional MCP server to fetch and crawl pages when remote retrieval is enabled.
- main.py – exposes a CLI that accepts optional filters (state, category, level), orchestrates discovery and extraction, deduplicates results and writes the output to a JSON file.

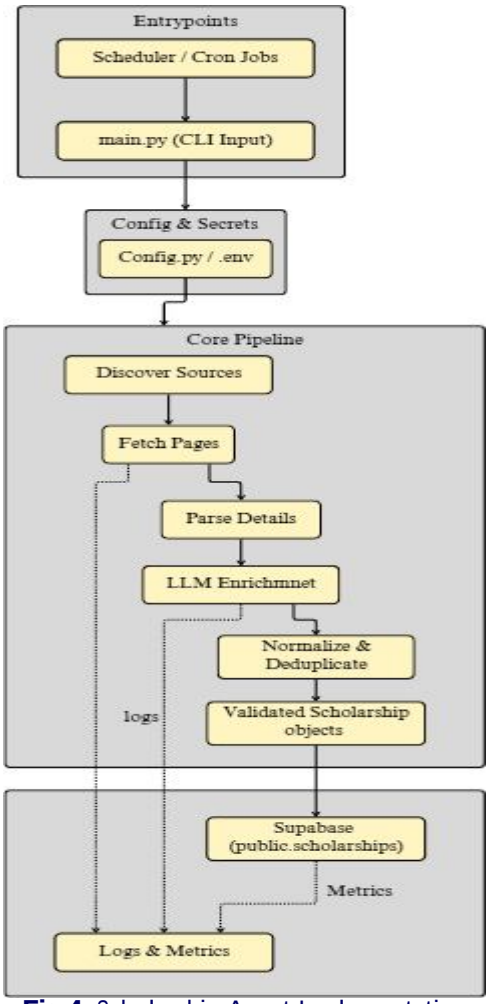


**Fig.4.** Scholarship Agent Implementation

The agent runs on a scheduler (cronjob or serverless function) and writes newly extracted scholarships to the Supabase scholarships table. It uses the Supabase Python client with a service role key to bypass RLS for inserts. To avoid duplicates, it normalizes URLs and names and checks for existing records. When vector similarity search is enabled (future work), the agent could embed scholarship descriptions and perform nearest neighbour search for improved matching. Secret keys are stored in environment variables and loaded via Pydantic settings, adhering to least privilege principles.

## VII. PERFORMANCE

During development, Vite leverages native ES modules and esbuild, achieving sub second startup times. Production builds are optimized by Rollup, and dynamic imports ensure that only pages visited by the user are loaded. TanStack React Query caches data and reduces network round trips. Supa base queries are indexed on primary and foreign keys, and updated_at triggers update timestamps automatically.

The agent batches HTTP requests and uses asynchronous I/O (aiohttp) and concurrency controls to avoid blocking. It uses heuristic scoring to prioritise official sources and de duplicates results, reducing processing time. Al though the application does not yet include comprehensive metrics, we propose using Lighthouse and Web Vitals for front end performance assessment and tracking job latency and throughput for the agent. Future work could introduce server side rendering, caching strategies and queue based job orchestration to improve scalability.

## VIII. ACCESSIBILITY(A11Y)

Scholar Nest adopts accessibility best practices through shadcnui and Tailwind. Components provide keyboard navigation, focus outlines and ARIA attributes. Radix primitives used in accordions and tabs apply the correct roles and states; ARIA roles convey semantic meaning to assistive technologies. Form fields use labels and required indicators; colour contrasts are defined via Tailwind tokens. The platform supports dark mode and responsive design through Tailwind's utility classes and custom hooks (e.g., useIs Mobile). The project plans to integrate automated testing (e.g., axe) and manual testing by people with disabilities to identify further improvemens.

## IX.SECURITY & PRIVACY

Security is enforced at multiple layers. Supabase implements RLS on every user facing table; policies ensure that users can only access their own records, while auth.uid() and get_current_user_role() functions differentiate between regular users and admins. JWT tokens carry user claims and are validated by the Supabase client. Edge functions use a service role key securely stored in environment variables and are executed with the least privilege required. The OTP feature originally used an otp_codes table, but was refactored into secure functions with search path isolation to prevent SQL injection. Admin email was initially hardcoded in the frontend, which posed a security risk; later migrations introduced a get_current_user_role function and updated policies to decouple admin privileges from email. On the frontend, no secrets are exposed; the Supabase publishable key is safe for client use. Inputs are validated in forms and sanitised before being sent to the database. The OWASP Injection Prevention cheat sheet recommends performing input validation, using parameterised queries and contextually escaping user data. The agent uses environment variables for API keys and avoids logging sensitive data. Access to the agent's scheduler and logs should be restricted using IAM roles, and rate limiting should be applied to external requests to prevent abuse.

## X. DEPLOYMENT & DEVOPS

The frontend is deployed on Vercel using npm run build and vercel commands. Vite generates static assets,and environment variables (Supabase URL and a non key) are configured through Vercel's dashboard. Supabase is provisioned via the Supabase CLI; migrations in the supabase/migrations directory are applied using supabase db push, and storage buckets are created for document uploads. Edge functions are deployed with supabase functions deploy. The ScholarShip Agent is containerized and can be deployed on a server or serverless platform. A cron job triggers python scholarship_agent/main.py at a configured interval. Logs are written to std out, and any errors trigger retries with exponential backoff. Environment variables (e.g., GOOGLE_API_KEY, TAVILY_API_KEY,SUPABASE_SERVICE_KEY) are Injected securely. Continuous integration tasks (linting, testing and type checking) can be configured in GitHub Actions. Monitoring and alerts should be added to detect failures in the scraping pipeline.

Table I. ENVIRONMENT ALVARIABLES

| Variable | Repository | Purpose |
|---|---|---|
| VITE_SUPABASE_URL | Scholar nest | URL of the Supa base project; Used by the JS client |
| VITE_SUPABASE_ANON_KEY | Scholar nest | Publish able an on key for Supa base |
| SUPABASE_URL | Scholar Ship Agent | Project URL usedby the Python client |
| SUPABASE_SERVICE_KEY | Scholar Ship Agent | Service role key with elevated privileges |
| GOOGLE_API_KEY | Scholar Ship Agent | API key for the Gemini model |
| TAVILY_API_KEY | Scholar Ship Agent | API key for the Tavily search API |
| RESEND_API_KEY | Scholar nest | Key for sending OTP emails via Resend |
| RESEND_EMAIL | Scholar nest | Sender email address for OTP emails |

## XI.CROSS REPOSITORY WORK FLOW

Figure 4 illustrates an end to end workflow involving both the Scholar Ship Agent and the Scholar Nest front end. A student submits an eligibility form, which the front end stores in the user_eligibility table. The agent periodically discovers scholarship URLs, extracts structured data with the LLM, and inserts or updates records in the scholarships table. When the student browses results, the front end queries Supa base using filters derived from the eligibility form and displays a ranked list. If the student bookmarks a scholarship, a row is inserted into saved_scholarships. Administrators can use the admin panel to review logs and modify scholarships or user roles.

IJIRAE:: International Journal of Innovative Research in Advanced Engineering ISSN: 2349-2163
Volume 12, Issue 11, November 2025 https://www.ijirae.com/archives
https://doi.org/10.26562/ijirae.2025.v1211.06

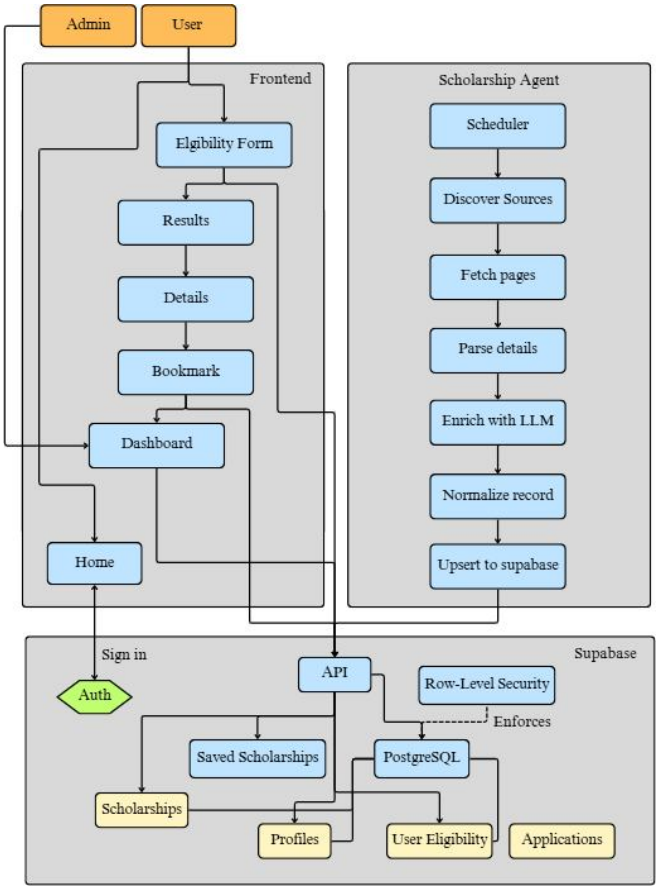


**Fig.5.** Cross-Repository Sequence Diagram

## XII. LIMITATIONS & FUTURE WORK

Although Scholar Nest delivers a functional scholarship matching platform, several limitations remain. The eligibility matching currently uses rule based filtering on structured fields and a small data set; implementing true vector similarity search with embeddings would improve recommendations but requires a vector database and embedding service. The front end is a client side rendered single page, which limits search engine optimization and initial load performance; future work could explore server side rendering or static pre rendering. The admin email is still hard coded in some front end logic (pending removal), and the system lacks comprehensive testing and analytics. Accessibility testing should be expanded, and language localization added to support other Indian languages. On the agent side, reliability could be improved by using message queues and task orchestration frameworks (e.g., Celery), implementing caching for previously crawled pages, and adding evaluation harnesses to measure extraction accuracy. Finally, integration with government APIs could replace scraping and provide authoritative data.

## XIII.CONCLUSION

Scholar Nest demonstrates how modern web frameworks, managed databases and LLM powered agents can be combined to address the challenge of scholarship discovery. Through a typed React front end, a secure Supa base backend and an automated Python agent, the system encodes student eligibility criteria and matches them again saturated set of scholar ships, delivering results in a responsive dashboard. RLS policies protect user data, while scheduled scraping and extraction keep the database upto date. By reducing search time and increasing access to funding, the project supports education equity and sets a foundation for future research in AI assisted eligibility matching.

## REFERENCES

1. This Dot Labs,“ Introduction to Vite–Next Generation Front end Tooling,”2023.
2. MDN WebDocs, “Getting started with React,”2024.
3. Type Script Team, “What isType Script ?,”2024.
4. Tailwind Labs,“ Utility-FirstCSS,” 2024.
5. Post gre SQL Documentation,“ CREATE POLICY— Row security policies,”2024.
6. OWASP,“ Injection Prevention Cheat Sheet,”2025
7. MDN Web Docs,“WAI-ARIARoles,”2024.