GOVERNMENT POLYTECHNIC, CHHATRAPATI SAMBHAJINAGAR
2024–25

A
PROJECT REPORT
ON

"SMARTINTERN – PERSONALIZED INTERNSHIP & SCHOLARSHIP PORTAL"

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DIPLOMA IN ARTIFICIAL INTELLIGENCE & MACHINE LEARNING DEPARTMENT

GUIDED BY
Prof. P. V. Sontakke

SUBMITTED
TO
GOVERNMENT POLYTECHNIC, CHHATRAPATI SAMBHAJINAGAR

SUBMITTED BY

Dipali Sanjay Khosare	234019
Prajakta Jagannath Mane	234020
Ayush Sanjay Parkhe	234023
Vaishnavi Manik Patil	234024
Dipika Anil Warade	234035
Dipali Gajanan Sanap	244203

GOVERNMENT POLYTECHNIC, CHHATRAPATI SAMBHAJINAGAR 2024–25

CERTIFICATE

This is to certify that the project report entitled "**SMARTINTERN – PERSONALIZED INTERNSHIP & SCHOLARSHIP PORTAL**"was successfully completed by the students of Sixth Semester Diploma in Artificial Intelligence & Machine Learning department:

Dipali Sanjay Khosare	234019
Prajakta Jagannath Mane	234020
Ayush Sanjay Parkhe	234023
Vaishnavi Manik Patil	234024
Dipika Anil Warade	234035
Dipali Gajanan Sanap	244203

In partial fulfillment of the requirements for the award and submitted to the Department of Artificial Intelligence & Machine Learning, Government Polytechnic, Chhatrapati Sambhajinagar. The work was carried out during the academic year 2025-26 as per the curriculum.

Prof. P. V. Sontakke
(Project Guide)

Dr. M. A. Ali
(HOD AN)

External Examiner

Dr. A. M. Jinturkar
(Principal)

ACKNOWLEDGEMENT

We are pleased to present this project report on “SMARTINTERN”, which was completed as a part of the course Project (7G401) for the academic year 2025–26. We would like to express our sincere gratitude to our respected Principal, Dr. A. M. Jinturkar, and Head of Department, Dr. M. A. Ali, Department of Artificial Intelligence and Machine Learning, Government Polytechnic College, Chhatrapati Sambhaji Nagar, for their valuable guidance, encouragement, and support throughout the completion of this project.

We would also like to thank Smt. P. V. Sontakke Mam for providing us with the opportunity to explore practical knowledge and understand the real-world applications related to our project. Our heartfelt thanks go to our project guide and faculty members for their continuous support, valuable suggestions, and encouragement which helped us to understand the topic better and complete the project successfully.

We would also like to acknowledge the management and administration of my college for providing the necessary facilities and a supportive learning environment to complete this project. The resources and infrastructure provided by the institution helped me conduct my research and prepare the report efficiently. The encouragement given by the college played an important role in motivating students like me to participate in such valuable internship programs.

Finally, we would like to thank our parents, friends, and all the staff members of the Artificial Intelligence and Machine Learning Department for their support and cooperation during the development of this project.

ABSTRACT

The transition from academia to the professional workforce presents a significant bottleneck for college students, who must simultaneously navigate career advancement and secure financial aid. Traditional discovery platforms and job portals are heavily fragmented and rely on static, keyword-based search methodologies. These legacy systems lack semantic understanding, leading to inefficient matching, and fail to accommodate the rigid demographic criteria required for scholarship distribution. Furthermore, modern platforms attempting to utilize machine learning frequently succumb to the "Cold Start" problem, where algorithms fail due to a lack of initial user interaction data.

To address these systemic inefficiencies, this project introduces **SmartIntern**, an intelligent, centralized recommendation ecosystem tailored for the collegiate demographic. SmartIntern shifts the paradigm from manual searching to automated discovery through a highly scalable, dual-engine architecture. The primary recommendation engine utilizes Natural Language Processing (NLP)—specifically Term Frequency-Inverse Document Frequency (TF-IDF) and Cosine Similarity—to mathematically calculate the semantic alignment between unstructured student skill sets and internship requirements. Concurrently, a secondary heuristic engine applies rule-based Regular Expressions (Regex) to process complex demographic constraints, ensuring 100% deterministic accuracy for financial aid and scholarship eligibility.

Developed using a lightweight Python (Flask) backend and a serverless PostgreSQL database (Supabase), the system ensures high concurrency and stateless cloud scalability. Crucially, the platform actively mitigates the User Cold Start vulnerability by enforcing a strict, middleware-driven profile gatekeeper, guaranteeing the algorithmic matrices have baseline vector data prior to execution. The resulting application successfully eliminates cognitive overload, providing students with a unified, highly personalized dashboard that proactively delivers precision-matched career and financial opportunities.

LIST OF FIGURES

LIST OF SYMBOLS , ABBREVIATIONS , & NOMENCLATURE

1. Abbreviations & Acronyms:

Abbreviation	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
CI / CD	Continuous Integration / Continuous Deployment
CSP	Constraint Satisfaction Problem
ERD	Entity-Relationship Diagram
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
ML	Machine Learning
NLP	Natural Language Processing
OAuth	Open Authorization
OCR	Optical Character Recognition
PaaS	Platform as a Service
PwD	Persons with Disabilities
REST	Representational State Transfer

Abbreviation	Full Form
SaaS	Software as a Service
SDLC	Software Development Life Cycle
SQL	Structured Query Language
UI	User Interface
UUID	Universally Unique Identifier
UX	User Experience

2. Mathematical Nomenclature

The following symbols are utilized within the algorithmic logic and vector mathematics of the platform's recommendation engines.

Symbol	Definition
$\mathbf{A}$, $\mathbf{B}$	Vector representations of unstructured text arrays in multi-dimensional space.
$\cos(\theta)$	Cosine Similarity; the geometric angle calculated between two vectors.
d	A specific document (e.g., an internship description or a user's skill profile).
$DF(t)$	Document Frequency; the total number of documents within the database containing term t.
IDF	Inverse Document Frequency.
N	The total number of documents (internships) present in

Symbol	Definition
	the active corpus.
$S_{interests}$	The isolated Cosine Similarity match score calculated for career interests.
$S_{location}$	The isolated Cosine Similarity match score calculated for geographical location.
S_{skills}	The isolated Cosine Similarity match score calculated for technical skills.
t	A specific term, keyword, or feature token extracted from a document.
TF	Term Frequency.
$W(t, d)$	The calculated mathematical vector weight of term t in document d.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Overview

The transition from an academic environment to the professional industry represents a critical bottleneck in the modern educational lifecycle. Every year, millions of college students enter the highly competitive job market seeking internships to gain practical experience, while simultaneously searching for scholarships to ease the financial burden of higher education.

Despite the proliferation of digital job boards and government aid websites, the ecosystem bridging students to these vital opportunities remains heavily fragmented and inefficient. Students are frequently subjected to information overload, manually sifting through thousands of generic listings that do not align with their specific technical skills or demographic backgrounds.

To resolve this, the "SmartIntern" platform was conceptualized. SmartIntern is an AI-driven, centralized recommendation system designed exclusively for the collegiate demographic. By leveraging Natural Language Processing (NLP) and deterministic rule-based algorithms, the system shifts the paradigm from "manual searching" to "intelligent discovery," ensuring students are proactively matched with the most relevant career and financial opportunities based on their unique, granular profiles.

1.2 Problem Statement

Currently, students rely on a disconnected mix of platforms to navigate their career and financial needs, resulting in a high-friction user experience. The primary issues this project addresses are:

- **The Literal Keyword Trap:** Existing internship portals rely heavily on exact string matching. If a student lists "React" as a skill, but an employer posts a role for a "Frontend Developer," traditional systems fail to connect them due to a lack of semantic understanding.
- **Disconnected Ecosystems:** Career advancement (internships) and financial survival (scholarships) are treated as entirely separate domains, forcing students to maintain multiple profiles across disparate websites.

- **Ignored Demographic Nuances:** While existing portals match based on technical skills, they fail to process complex demographic criteria (such as family income limits, gender, or specific social categories), which are strictly required for accurate scholarship matching.
- **The Cold Start Problem:** Modern recommendation algorithms often fail when a new user registers but has no historical interaction data (clicks or applications) for the system to analyze.

1.3 Objectives

The primary objective of this project is to develop a highly scalable, personalized platform that resolves the inefficiencies of traditional opportunity discovery. The specific objectives include:

1. **Develop an AI Recommendation Engine:** To implement a Content-Based Filtering algorithm utilizing TF-IDF (Term Frequency-Inverse Document Frequency) and Cosine Similarity to mathematically match unstructured user skills with internship requirements.
2. **Develop a Heuristic Financial Aid Engine:** To design a secondary, deterministic algorithm utilizing Regular Expressions (Regex) to parse complex demographic eligibility criteria and match students with 100%-applicable scholarships.
3. **Solve the User Cold Start Problem:** To engineer a mandatory profile-completion gateway that structurally enforces data collection, ensuring the recommendation matrices always have baseline vector data upon a user's first login.
4. **Ensure Cloud Scalability:** To deploy the backend architecture using stateless RESTful API protocols over a serverless PostgreSQL database (Supabase), eliminating the risk of connection pool exhaustion during heavy data processing.

1.4 Scope

The scope of the SmartIntern project encompasses the development of a full-stack web application tailored for college students and recent graduates.

- **In-Scope:** The system includes secure user authentication (Google OAuth 2.0), comprehensive profile management (skills, interests, and demographic data collection), a dynamic dashboard rendering personalized internship and scholarship matches, and an automated backend data-processing pipeline. The system processes text data to establish semantic relationships and demographic eligibility.
- **Out-of-Scope:** The platform acts as a discovery and recommendation hub; it does not host native video interviewing, in-app messaging between recruiters and students, or direct payment gateways for scholarship disbursement. Users are matched internally but redirected to the original source links for final application submission.

CHAPTER 2: LITERATURE SURVEY

The literature survey plays a critical role in identifying the existing work, technologies, and research studies related to our project. By exploring scholarly articles, whitepapers, and implemented systems, we identify trends, similarities, and limitations in the current approaches. This chapter reviews the academic foundation of Natural Language Processing (NLP) in recruitment, the mathematical logic behind recommendation engines, and the documented challenges of user data acquisition, which collectively justify the architecture of the SmartIntern platform.

2.1 Literature Survey

[1] K. S. Varshith Reddy, et al., "Resume Analyzer and Job Recommendation System" (2025) This paper explores the transition from manual applicant screening to automated, AI-driven evaluation. The authors highlight the inefficiency of handling unstructured candidate data. To resolve this, they proposed a system that leverages Natural Language Processing (NLP) to extract core skills and education. Crucially for our project, the methodology relies on Term Frequency-Inverse Document Frequency (TF-IDF) combined with Cosine Similarity to calculate a precise mathematical match between a candidate's profile and a job description. The outcome of this research proves that vector-based text analysis significantly outperforms traditional keyword-based database queries in terms of recommendation accuracy.

[2] Hongli Yuan and Alexander A. Hernandez, "User Cold Start Problem in Recommendation Systems: A Systematic Review" (2023) While machine learning provides excellent recommendation capabilities, this paper addresses its greatest vulnerability: the User Cold Start Problem. The authors conducted a systematic review revealing that recommendation engines (particularly collaborative filtering models) fail completely when a new user registers but has no historical data, clicks, or profile information to process. The study explores various mitigation strategies, ultimately concluding that forcing initial data collection or using hybrid content-based filtering is required. This research directly justifies the architectural decision in SmartIntern to implement a "profile-completion

gatekeeper" that requires users to input technical skills before accessing the recommendation dashboard.

[3] Gugasree S. and Dr. N. Saranya, "AI-Based Skill Matching and Job Recommendation Hub" (2026) This recent study reinforces the necessity of replacing manual job searching with proactive recommendation hubs. The authors implemented a system where user resumes are parsed and the text is converted into numerical representations. By applying Cosine Similarity to these TF-IDF vectors against a dataset of job descriptions, the system successfully generated highly pertinent job openings. The methodology confirms that a lightweight, content-based NLP approach is highly effective for student-centric platforms, drastically reducing the friction and effort required in manual job filtering.

[4] Priyanka Singla and Vishal Verma, "An Intelligent Job Recommendation System based on Semantic Embeddings and Machine Learning" (2025) This research identifies the critical shortcomings of traditional job portals, which rely heavily on static, exact-keyword matching. The authors note that if a candidate and an employer use different terminologies for the same skill, traditional systems fail to connect them. To solve this, their methodology incorporates advanced NLP techniques, including Bag of Words (BoW) and semantic embeddings, to build a rich feature set. The outcome demonstrates that moving beyond literal text and analyzing the *semantic meaning* of skills provides much more robust and dynamic job recommendations. This justifies SmartIntern's use of domain-knowledge expansion (e.g., mapping specific frameworks to broader industry terms) prior to TF-IDF vectorization.

2.2 Inferences Drawn

From the detailed analysis of the above literature, several critical architectural inferences have been drawn that directly shape the SmartIntern platform:

1. **Obsolescence of Simple Keyword Queries:** Traditional portals are highly inefficient due to their reliance on exact string matching. As proven by

Singla and Verma [4], modern systems must employ semantic awareness to prevent missed opportunities.

2. **Efficiency of Vector Mathematics:** Papers [1] and [3] universally agree that transforming unstructured text (skills and interests) into numerical vectors via TF-IDF, and calculating their Cosine Similarity, is the most lightweight, scalable, and accurate method for content-based recommendation matching.
3. **Structural Mitigation of AI Failure:** As highlighted by Yuan and Hernandez [2], the "Cold Start" problem is fatal to recommendation engines. Therefore, SmartIntern cannot rely purely on user behavior tracking; it must enforce mandatory skill data collection during onboarding to guarantee the TF-IDF engine always has baseline matrices to process.
4. **The Need for a Hybrid Ecosystem:** While the literature thoroughly proves the efficacy of NLP for skill-based job matching, it exposes a gap regarding deterministic demographic matching. Because scholarships require rigid rule-based filtering (e.g., category, income) rather than fuzzy NLP matching, SmartIntern must implement a dual-engine architecture to serve both career and financial needs simultaneously.

CHAPTER 3: REQUIREMENTS GATHERING AND ANALYSIS

3.1 Introduction and Problem Definition

Requirements gathering is the foundational phase of the Software Development Life Cycle (SDLC), wherein the abstract concepts of a proposed system are translated into definitive technical specifications. The problem defined for the SmartIntern platform is the acute inefficiency in opportunity discovery for college students. The system must process highly unstructured text data (resumes and skills) and complex heuristic data (demographic constraints) to output personalized, highly accurate career and financial aid recommendations. This chapter outlines the feasibility of the proposed system, the hardware and software prerequisites, and the strict functional and non-functional requirements necessary to execute the architecture.

3.2 Feasibility Study

Before finalizing the system requirements, a feasibility study was conducted to ensure the proposed SmartIntern platform could be realistically developed, deployed, and maintained.

- **Technical Feasibility:** The project is technically highly feasible. By leveraging a stateless cloud infrastructure (Supabase REST API) and a lightweight Python (Flask) backend, the system bypasses the need for heavy enterprise server clusters. Utilizing pre-trained Natural Language Processing (NLP) techniques like TF-IDF instead of deep neural networks ensures the AI engine can run efficiently on standard cloud hardware.
- **Economic Feasibility:** The system is economically viable as it relies entirely on open-source libraries (Scikit-Learn, Pandas) and cloud platforms offering generous academic/developer tiers (Render, Supabase). The elimination of persistent SQL connection pooling further reduces the required compute hours, keeping hosting costs negligible.
- **Operational Feasibility:** The operational workflow is designed to be highly automated. By implementing a "Cold Start Gatekeeper," the system forces

users to provide necessary data upfront, drastically reducing the operational overhead of managing incomplete profiles or debugging AI mathematical failures.

3.3 Hardware Requirements

The hardware requirements are divided into two environments: the local development environment used to build the system, and the baseline cloud server requirements for deployment.

Development Environment (Minimum Specifications):

- **Processor:** Intel Core i5 (8th Gen) / AMD Ryzen 5 or higher.
- **RAM:** 8 GB (16 GB recommended for handling large Pandas DataFrames during local testing).
- **Storage:** 256 GB SSD for fast read/write operations during matrix compilation.
- **Internet:** Stable broadband connection for Supabase SDK synchronization and OAuth callback testing.

Deployment Environment (Cloud Host Minimum):

- **Compute:** 1 vCPU.
- **Memory:** 512 MB RAM (Optimized due to the stateless database connection).
- **Database:** Cloud-hosted PostgreSQL instance (Supabase) with vector support.

3.4 Software Requirements

The SmartIntern platform is built on a modern, full-stack Python architecture. The following software tools, libraries, and frameworks are strictly required:

- **Operating System:** Windows 10/11, macOS, or Linux (Ubuntu 20.04+).
- **Programming Languages:** Python 3.10+, HTML5, CSS3, JavaScript (ES6).

- **Backend Framework:** Flask (for WSGI routing and secure session management).
- **Database & BaaS:** Supabase (PostgreSQL) integrated via the supabase Python SDK.
- **Authentication:** Google OAuth 2.0 configured via the Authlib library.
- **Machine Learning Ecosystem:** scikit-learn (for TfidfVectorizer and similarity metrics), pandas (for in-memory tabular data manipulation), and numpy.
- **Environment Management:** python-dotenv for secure environment variable loading.

3.5 Functional Requirements

Functional requirements define the specific behaviors, tasks, and data processing pipelines the system must execute.

1. **Secure Authentication and Session Management:** The system must allow users to register and log in via Google OAuth 2.0. The system must generate secure server-side sessions and successfully manage callback state tokens to prevent CSRF (Cross-Site Request Forgery) attacks.
2. **Profile Data Enforcement (Cold Start Mitigation):** The system must intercept user navigation. If a user attempts to access the recommendation dashboard without a minimum of three technical skills saved in the database, the system must forcefully redirect them to the onboarding gateway.
3. **AI Internship Recommendation Engine:** The system must fetch all active internships and the user's saved skills. It must generate numerical vectors using TF-IDF and calculate the mathematical alignment utilizing the Cosine Similarity formula:
   $$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$
   The system must then rank these scores and render the Top 10 matches dynamically to the frontend dashboard.
4. **Heuristic Scholarship Matching:** The system must apply rule-based logic to parse unstructured scholarship eligibility criteria (using Regular

Expressions). It must perform a boolean check against the user's demographic data (e.g., gender, social category) and reject any matches that do not yield a 100% eligibility score.

3.6 Non-Functional Requirements

Non-functional requirements define the quality attributes, performance goals, and security standards the system must uphold.

- **Performance and Scalability:** The backend must execute database read/write operations without locking server threads. This is achieved by utilizing the stateless Supabase HTTP REST API over Port 443 rather than traditional persistent psycopg2 TCP connections, ensuring the system can scale to hundreds of concurrent users without connection exhaustion.
- **Security and Cache Control:** Sensitive dashboard data must be protected post-logout. The system must inject strict HTTP headers (Cache-Control: no-store, no-cache, must-revalidate) into all dynamic templates to prevent modern browsers' Back-Forward Cache (bfcache) from exposing data.
- **Usability:** The user interface must be responsive and intuitively categorized. The system must process complex background AI mathematics in under 3 seconds and render the results without cognitive overload, segregating data into "Urgent Action," "Top AI Matches," and "Financial Aid" sections.

CHAPTER 4: METHODOLOGY

The development of a complex software system requires a structured framework to ensure that the final product meets all technical and user-centric requirements. Because the SmartIntern platform heavily integrates machine learning (TF-IDF vectorization) and strict heuristic rule-matching (Regex for scholarships), a rigid, linear approach was deemed inappropriate. Instead, an iterative, phase-based methodology was required. This approach allowed for continuous testing, refinement of algorithmic outputs, and architectural pivoting based on real-world system behavior before committing to the final deployment.

4.1 System Development Approach

The development of SmartIntern was executed through a series of iterative cycles. This method involves building an initial functional version of the system, testing it against core requirements, and then reworking the architecture as necessary until the system reaches production-level stability.

The methodology was executed in the following progressive phases:

Phase 1: Requirement Gathering and Analysis The initial phase involved identifying the core problems with traditional portals. The requirements dictated that the system must process unstructured text (user skills) and deterministic criteria (user demographics). The necessary mathematical algorithms (Cosine Similarity) and database constraints were clearly defined during this stage.

Phase 2: Architectural and UI Design Before writing the core backend logic, a structural design was formulated. This involved sketching the Entity-Relationship Diagram (ERD) for the database and wireframing the user dashboard. The goal was to conceptually map out how the complex matrix data generated by the AI would be visually segregated (e.g., separating "Top AI Matches" from "Financial Aid") to prevent cognitive overload for the end-user.

Phase 3: Building the Initial Iteration A localized, baseline version of the application was constructed to test core concepts. In this phase:

- A basic Flask backend was instantiated.
- The TfidfVectorizer from Scikit-Learn was implemented against a static, local CSV file of dummy internships to test the vector math.
- A standard local database was connected to test the flow of user registration.

Phase 4: Algorithmic Testing and Evaluation The initial build was subjected to rigorous testing, which exposed several critical flaws that a traditional linear development model would have missed until final deployment:

- **The Cold Start Failure:** Testing revealed that if a user navigated to the dashboard with an empty profile, the Pandas dataframes crashed, causing a fatal server error.
- **The Literal Keyword Flaw:** The initial TF-IDF math proved too rigid; it failed to match a user with "React" skills to a "Frontend" internship.
- **Database Connection Leaks:** When the local build was briefly tested on a cloud server using persistent psycopg2 connections, the system experienced severe connection pool exhaustion and latency.

Phase 5: Architectural Refinement Based on the evaluations in Phase 4, the application was heavily re-engineered to resolve the identified bottlenecks:

- **Cloud Database Pivot:** The database interaction layer was completely rewritten. Persistent SQL connections were discarded in favor of the stateless Supabase Python SDK (REST API), permanently solving the connection leak issue.
- **Algorithmic Enhancement:** A "Smart Dictionary" (domain knowledge expansion) was injected into the NLP pipeline to expand acronyms and related terms prior to vectorization, significantly increasing match accuracy.
- **Cold Start Gatekeeper:** Strict Flask middleware (@login_required and is_profile_complete) was developed to forcefully redirect users with incomplete profiles, ensuring the AI matrices always had baseline data.

- **The "Stale Data" Bottleneck:** Early testing showed that relying purely on manual employer uploads resulted in an empty, stale database, severely limiting the AI's ability to recommend diverse roles. A dedicated, automated

data-aggregation pipeline was required to keep the platform populated without human intervention.

Phase 6: Engineering the Final Product Once the refined architecture successfully handled high concurrency, dynamic AI scoring, and strict heuristic scholarship matching without errors, it was solidified into the final production system. Security headers, OAuth 2.0 state validation, and responsive UI components were then finalized for the ultimate deployment.

4.2 Rationale for the Iterative Approach

An adaptive, iterative methodology was the optimal choice for the SmartIntern project for several specific technical reasons:

1. **High Algorithmic Uncertainty:** In machine learning and NLP, it is impossible to predict the exact accuracy of Cosine Similarity matching without testing it against a live corpus of text. An iterative approach allowed for the manual adjustment of feature weights (e.g., weighting skills at 45% and interests at 40%) based on observed output quality.
2. **Complex UI/UX Requirements:** Presenting mathematical AI outputs (match percentages, parsed criteria, deadlines) requires careful user interface design. Continuous iteration allowed for visual refinement to ensure the dashboard remained intuitive rather than overwhelming.
3. **High Technical Risk (Cloud Integration):** Transitioning from stateful local development to a stateless cloud database (Supabase) carried significant architectural risk. Testing the system in cycles allowed for isolated integration testing, ensuring the HTTP REST APIs could handle bulk data fetching before the entire application was built around them.
4. **Uncovering Edge Cases:** Complex logical hurdles, such as the User Cold Start problem and OAuth CSRF state mismatches, were not fully apparent during the initial theoretical design. The iterative testing phases exposed these edge cases early, allowing the architecture to be fortified proactively.

4.3 Automated CI/CD Data Aggregation Pipeline

A critical realization during the iterative testing phases was that a recommendation engine is fundamentally useless without a continuous influx of fresh, high-quality data. Relying solely on manual employer registrations created a severe "Stale Data" bottleneck. To resolve this, the methodology was expanded to include the engineering of an automated web-scraping and data-ingestion pipeline utilizing GitHub Actions.

Rather than hosting persistent background processes on the Flask web server—which would consume unnecessary compute resources and risk crashing the main user application—the data aggregation logic was decoupled into standalone Python scripts. These scripts were then integrated into a CI/CD (Continuous Integration / Continuous Deployment) workflow using GitHub Actions, effectively utilizing Microsoft's infrastructure as a serverless cron-job execution environment.

4.3.1 Workflow Scheduling Strategy To maintain a real-time dataset while respecting the rate limits of external source portals and optimizing Supabase database writes, two distinct, staggered workflows were engineered:

- **The Internship Aggregator (High Frequency):** Because career opportunities open and close rapidly, the internship web-scraping workflow is configured to execute every 3 hours. Using POSIX Cron syntax (`0 */3 * * *`), the GitHub Action automatically initializes a virtual Ubuntu runner, installs necessary Python dependencies (like `BeautifulSoup4` and `Playwright`), and aggregates fresh job postings.
- **The Scholarship Aggregator (Low Frequency):** Financial aid opportunities have much longer lifecycles and application windows. Therefore, the scholarship aggregation workflow is configured to execute every 6 hours (`0 */6 * * *`). This reduces unnecessary database queries while ensuring no new government or private grants are missed.

4.3.2 The Execution Architecture The automated methodology follows a strict, fail-safe execution flow:

1. **Initialization:** The GitHub Action triggers precisely on schedule, checking out the repository code and securely injecting environment variables (Supabase URL and API Keys) stored in GitHub Secrets.
2. **Data Extraction & Cleaning:** The Python scripts extract unstructured HTML data from targeted external portals, clean the text, and format it into the normalized schema required by the SmartIntern database.
3. **Cryptographic Deduplication (`content_hash`):** Before making any network calls to the database, the script generates a unique cryptographic hash (`content_hash`) for every scraped item.
4. **Upsertion:** The script utilizes the Supabase Python SDK to perform an "upsert" (Update or Insert) operation. If the `content_hash` already exists in the database, the row is ignored or updated; if it is new, it is inserted. This guarantees that the automated workflows never duplicate data, even if they run thousands of times a month.

By integrating GitHub Actions into the SDLC, the SmartIntern platform achieves true automation. It successfully transforms from a static application into a dynamic, self-sustaining ecosystem that continuously feeds fresh text data into the Natural Language Processing AI matrices.

CHAPTER 5: SYSTEM DESIGN

System design is a critical phase of the Software Development Life Cycle (SDLC) where the conceptual architecture of the application is translated into structured, logical models. For the SmartIntern platform, modeling the flow of unstructured text data through the machine learning pipeline is just as important as modeling the user interaction. This chapter utilizes both Data Flow Diagrams (DFDs) to map the trajectory of data and Unified Modeling Language (UML) diagrams to visualize the dynamic and static behaviors of the Flask backend and AI recommendation engines.

Architecuture Flow:

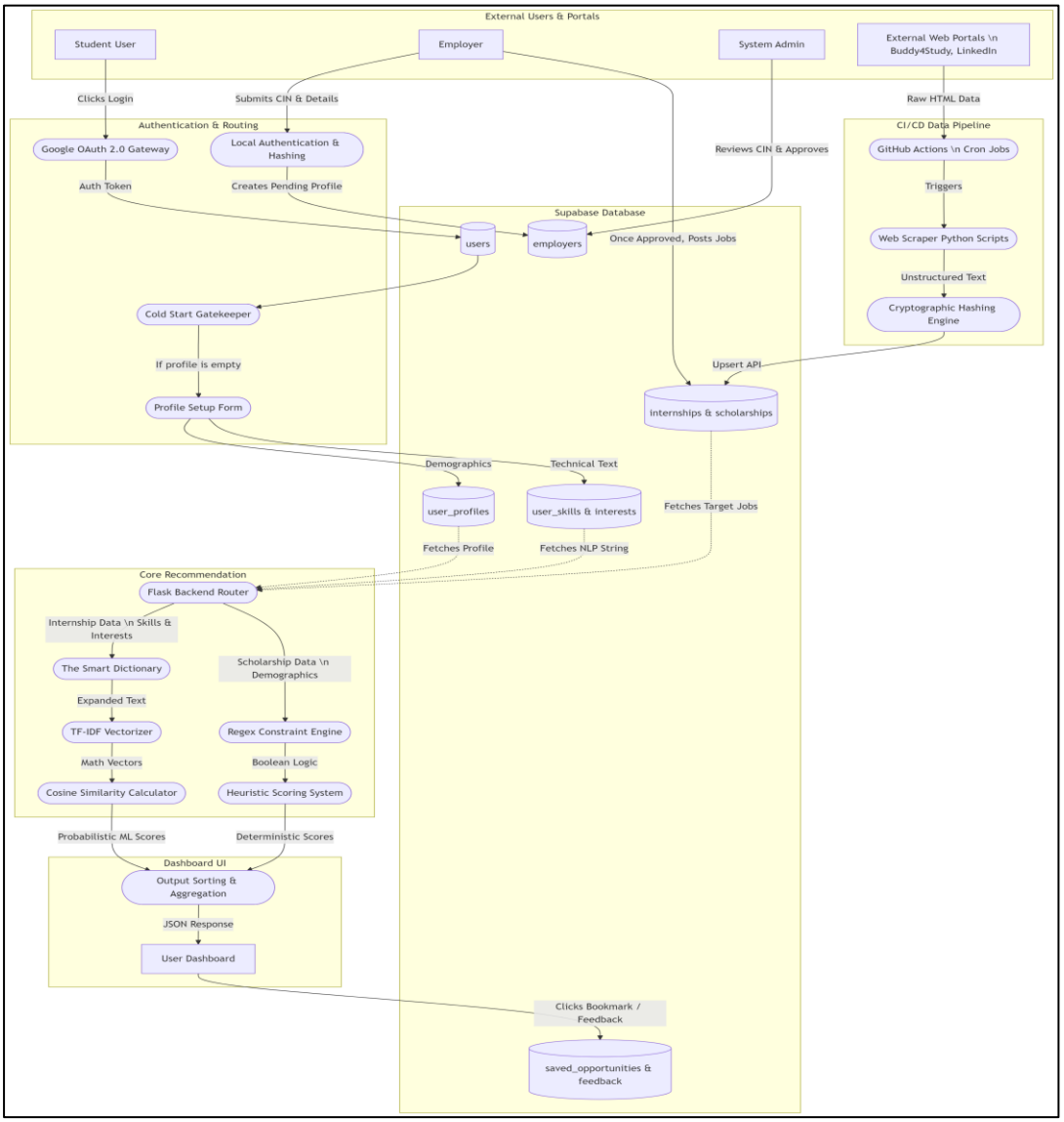


***Figure 5.1**: Detailed System Architecture and Data Flow Diagram illustrating the end-to-end processing pipeline of the SmartIntern platform.*

5.1 Level 0 DFD (Context Diagram)

The Zero Level Data Flow Diagram, or Context Diagram, represents the entire SmartIntern system as a single, high-level process. It identifies the external entities that interact with the system and the fundamental flow of data between these entities and the platform, ignoring internal routing logic.

This is the highest-level view of the system's **Data Boundaries**.

- **Central Process:** The "SmartIntern System" is a single bubble representing the entire application.
- **External Entities:** It shows how data flows between the system and external actors. The **Student** provides preferences and receives recommendations, while **External Data Sources** feed internship and scholarship data into the system.
- **Admin Control:** The Admin receives system status reports and sends back control policies (like scraping or cleanup rules).

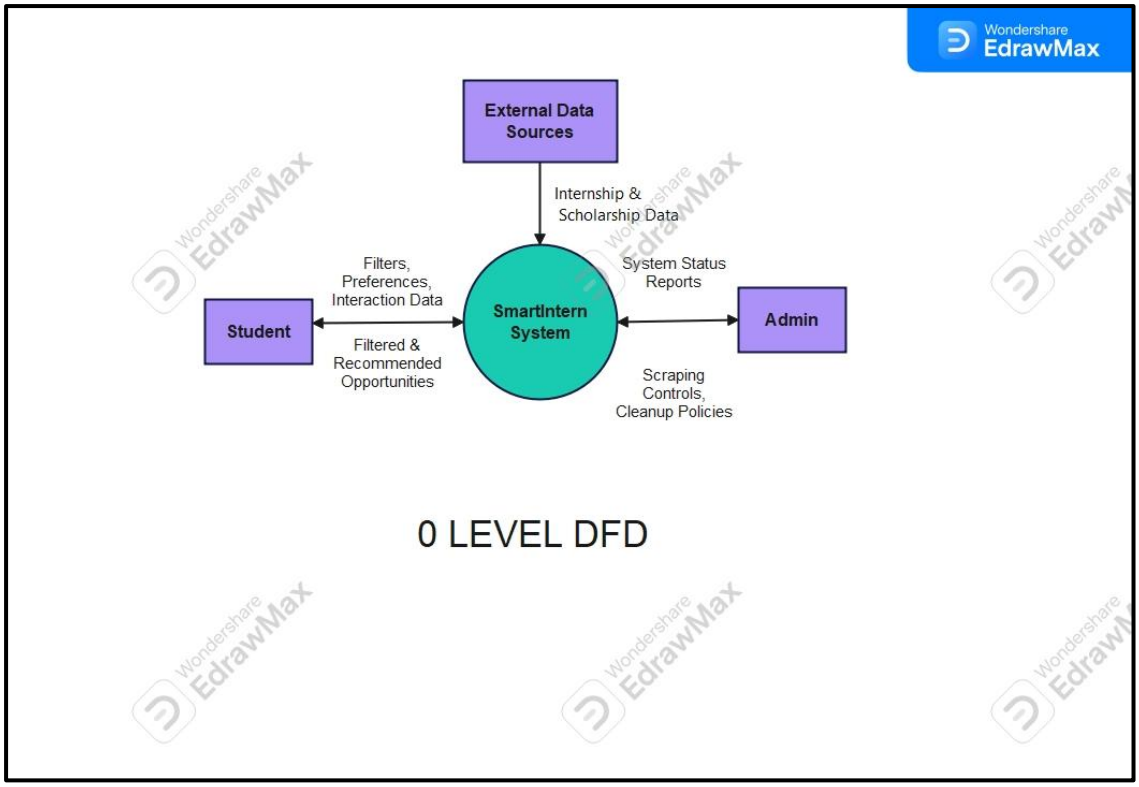


Figure 5.2: 0-Level Data Flow Diagram (Context Diagram) *– Provides a high-level overview of the SmartIntern system boundaries and its data exchange with external entities.*

5.2 Level 1 DFD

The Level 1 DFD breaks down the central context process into its primary sub-processes. It highlights how the platform manages authentication separately from the AI and heuristic mathematical engines, and introduces the internal data stores.

This breaks down the internal **Major Processes** of the SmartIntern system.

- **Process Flow:** Data moves from `1. User Authentication` to `2. Profile & Skills Processing`.
- **Data Stores:** It introduces databases (D1 for Users, D2/D3 for Internships/Scholarships).
- **The Engine:** Process `3. Recommendation Engine` pulls data from the opportunity databases and matches it against the user's processed profile to generate a list for Process `4. Display Recommendations`.

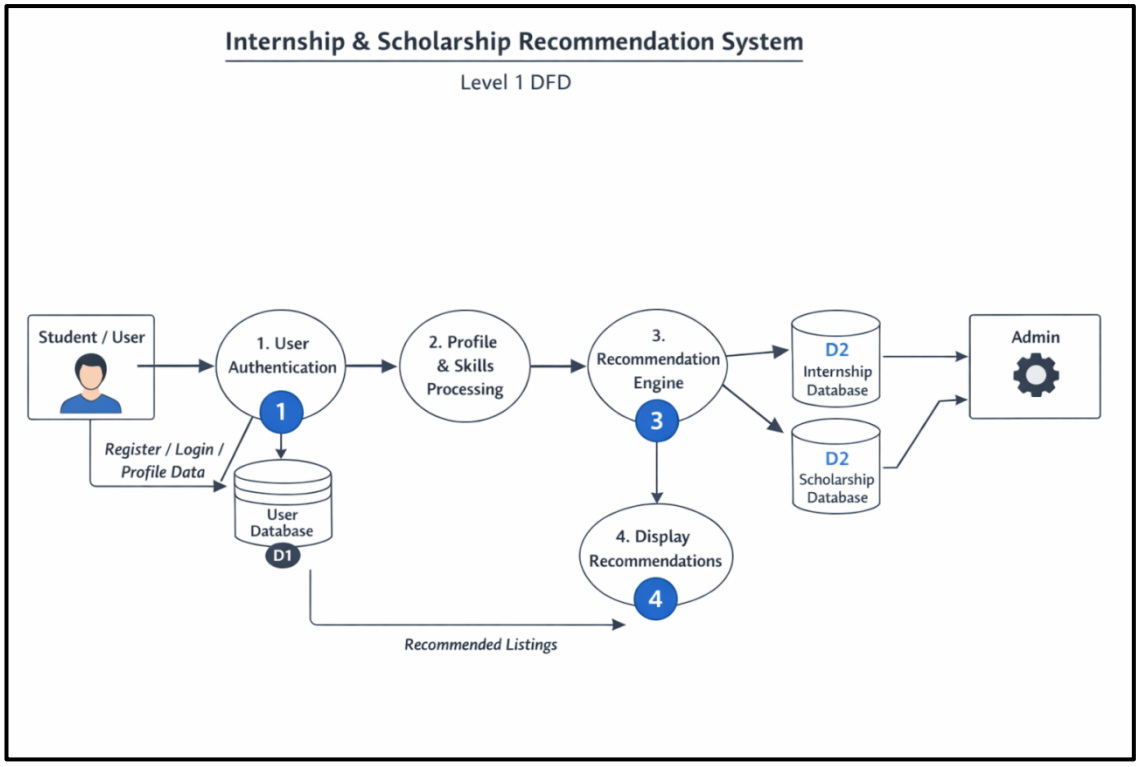


Figure 5.3: Level 1 Data Flow Diagram – Details the primary internal processes of SmartIntern, including authentication, profile processing, and the flow of information between data stores.

5.3 Level 2 DFD

The Level 2 DFD dives deeply into one specific sub-process from Level 1 to show the granular computational logic. For SmartIntern, the most critical process to expand is 3.0 NLP Internship Matching Engine, illustrating exactly how the AI transforms text into match percentages.

This is a **Functional Decomposition** specifically for Process 3 (the Recommendation Engine).

- **Granular Logic:** It reveals the "brain" of SmartIntern. It isn't just one step; it involves `3.1 Skill & Profile Matching`, followed by `3.3 Rank Recommendations`.
- **Ranking:** This shows that the system doesn't just find any match—it evaluates the "Matched Opportunities" to prioritize the best ones for the student before they are displayed.

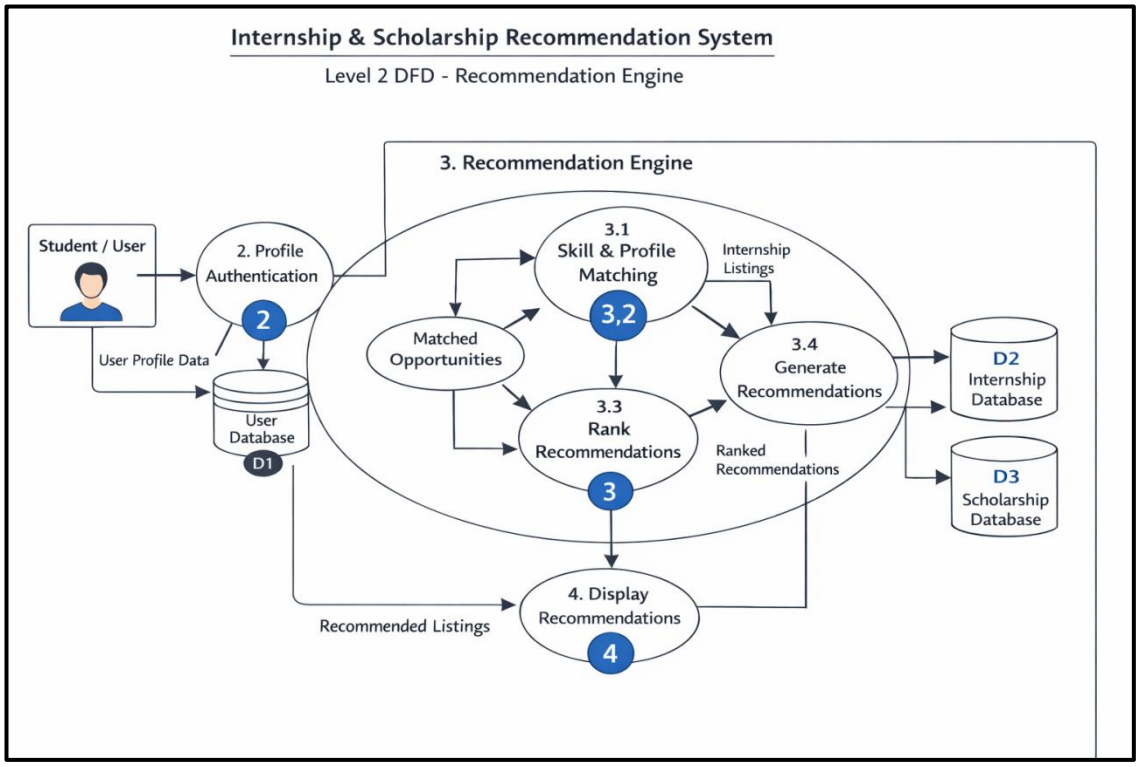


Figure 5.4: Level 2 Data Flow Diagram (Recommendation Engine) – A decomposed view of the recommendation process, highlighting the internal logic of skill matching and opportunity ranking.

5.4 Use Case Diagram

A Use Case Diagram represents the functional requirements of the system from the user's perspective. It defines the "Actors" and the "Use Cases" (actions) without detailing the internal database interactions.

This diagram describes the **Functional Requirements** from the perspective of different users (Actors).

- **Student:** The primary actor who interacts with the system to `Register/Login`, `Update Profile`, and `View/Apply` for opportunities.
- **Employer/SmartIntern:** Responsible for the supply side, specifically the `Add Internship` and `Add Scholarship` functions.
- **Admin:** Oversees the system by managing (Update/Delete) existing opportunities to ensure quality and relevance.

- **Relationships:** The «includes» and «extends» relationships show dependencies—for example, a user must be logged in to update their profile or receive specific recommendations.

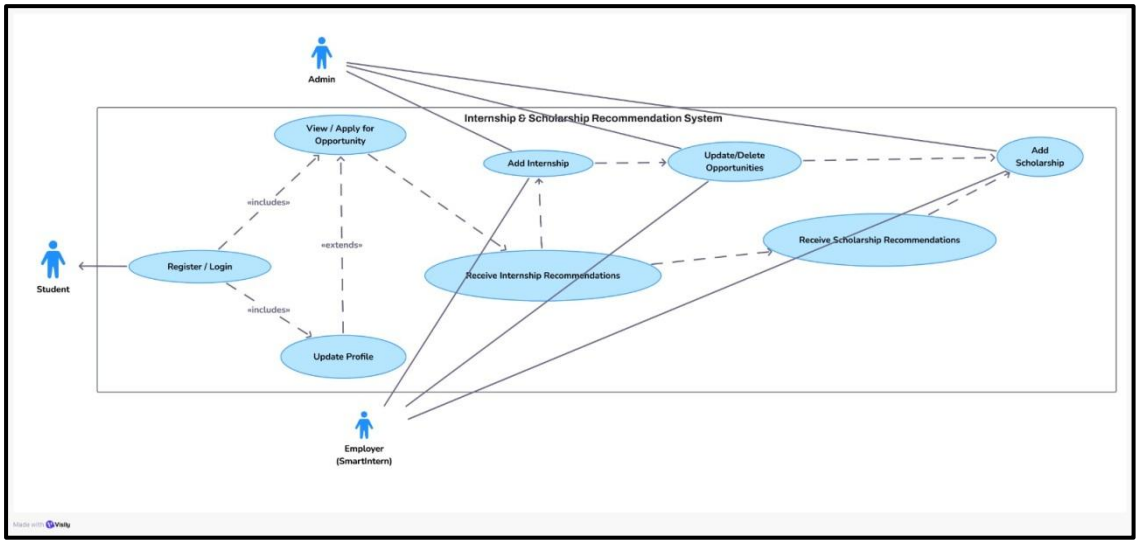


Figure 5.5: Use Case Diagram – Represents the functional requirements of the system, depicting the interactions between Students, Admins, and Employers with various system features.

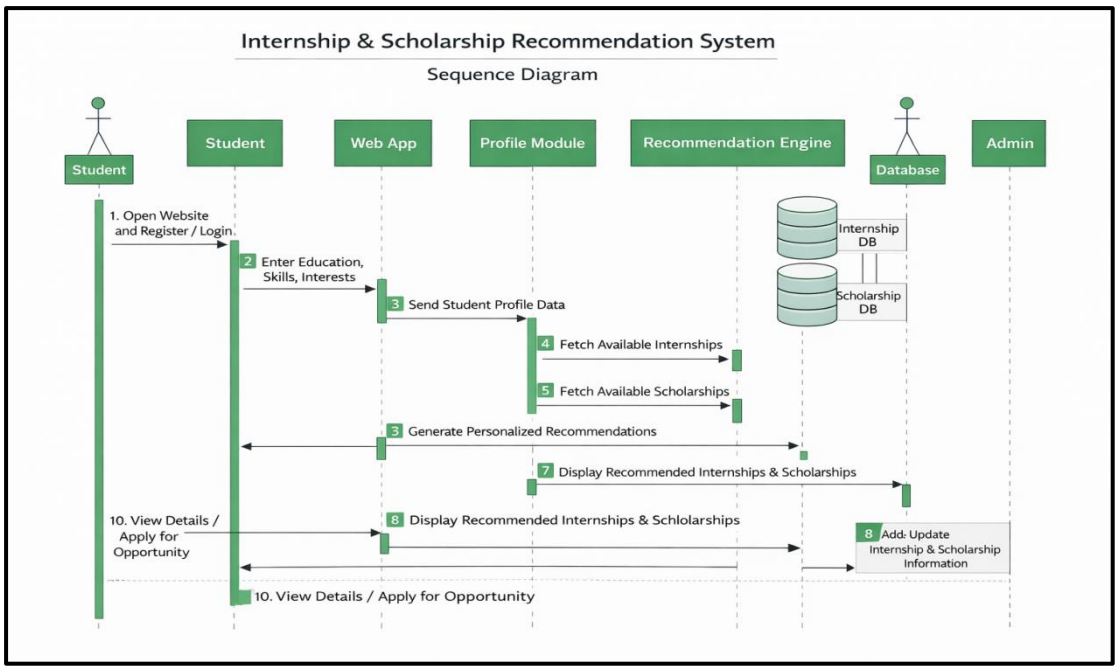
5.5 Sequence Diagram

A Sequence Diagram illustrates the interactions between system objects in a sequential, time-based order. It shows the step-by-step flow of messages passed between the frontend, the Flask backend, and the database over the lifespan of a single HTTP request.

This illustrates the **Dynamic Behavior** and the timing of interactions between objects.

- **Chronological Flow:** It reads from top to bottom. For instance, the **Student** must "Enter Education/Skills" (Step 2) before the **Profile Module** can "Fetch Available Internships" (Step 4).
- **Back-end Interaction:** It clearly shows the communication between the Web App, the Recommendation Engine, and the Database.
- **Completion:** The sequence ends with the student viewing details or applying, completing the primary user journey.

Figure 5.6: Sequence Diagram – Visualizes the chronological flow of interactions and message exchanges between the Student, Web App, and Backend modules during a typical session.



5.6 Class Diagram

The Class Diagram visualizes the static structure of the application. It models the core entities (classes) within the Python architecture, detailing their internal attributes, operational methods, and the structural relationships between them.

This diagram represents the **Static Structure** of SmartIntern. It defines the blueprint of the system by showing the attributes and methods of various objects and how they interact.

- **Core Entities:** It identifies key classes like `User`, `Admin`, `Employer`, `Scholarship`, and `Internship`.
- **Recommendation Logic:** The `RecommendationEngine` class is central, containing methods like `analyzeUserProfile()` and `matchInternships()`. This shows that the system logic is decoupled from the data entities, allowing for easier algorithm updates.
- **Relationships:** You can see how the `Application` class acts as a bridge between the `User` and the opportunities they apply for, tracking the `applyDate` and `status`.

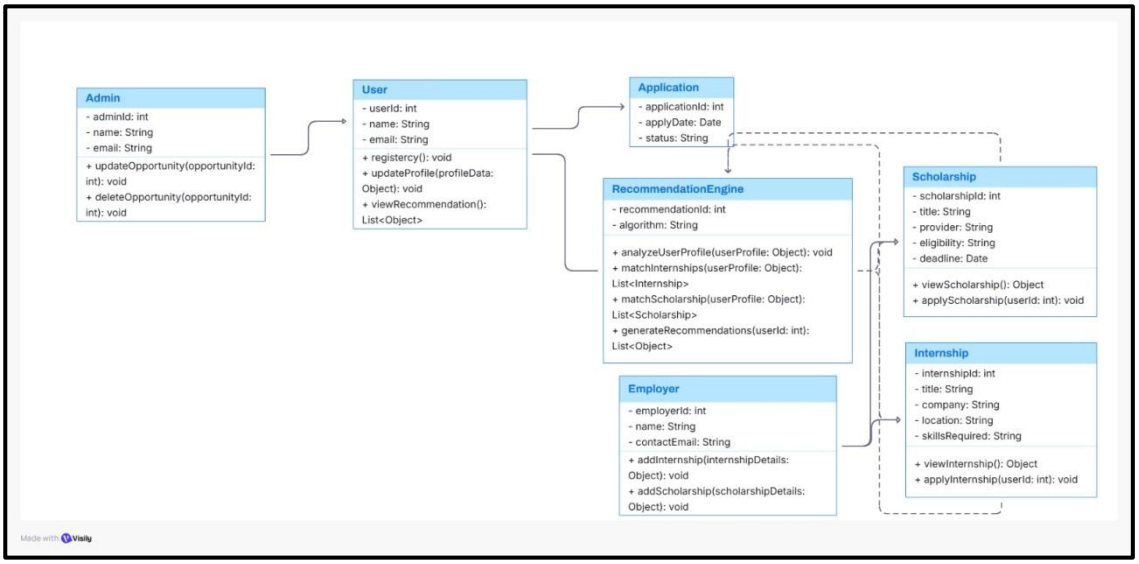


Figure 5.7: Class Diagram – Illustrates the static structure of SmartIntern, defining the attributes, methods, and relationships between core entities like Users, Admins, and the Recommendation Engine.

CHAPTER 6: DATABASE SCHEMA

6.1 Introduction to the Data Architecture

The database schema serves as the logical blueprint of the SmartInt ern platform. Hosted on Supabase (PostgreSQL), the schema is highly normalized. It strategically separates authentication from demographic data and normalizes unstructured text (like skills and interests) into dedicated relational tables. This architecture ensures referential integrity, minimizes storage redundancy, and optimizes the complex database queries required before feeding data into the machine learning (TF-IDF) and heuristic (Regex) engines.

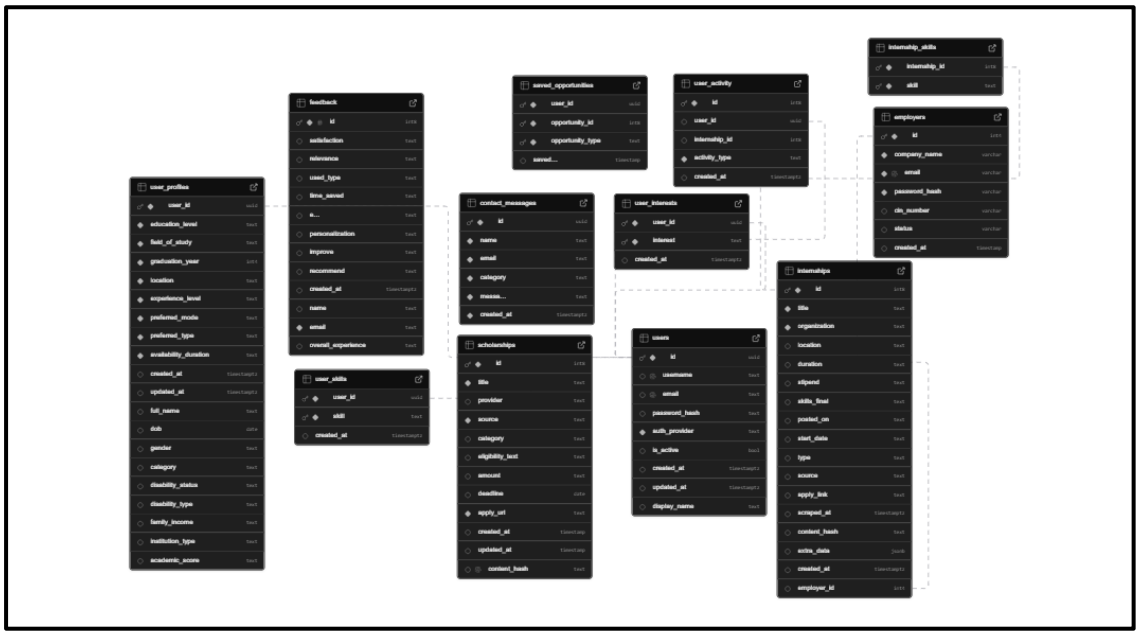
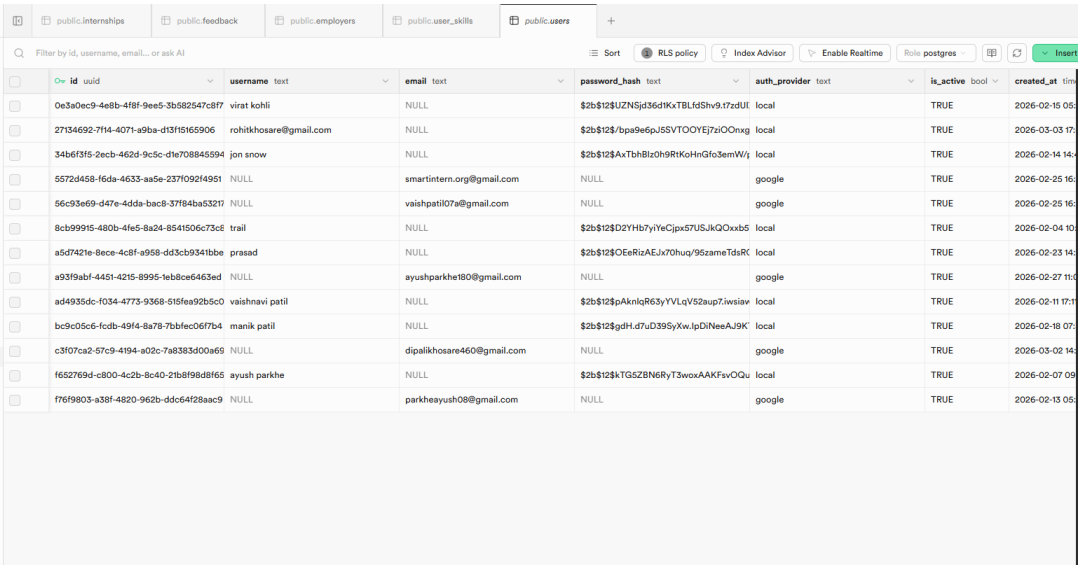


Figure 6.1: Complete Entity-Relationship Diagram (ERD) of the SmartIntern Database Architecture.

6.2 Table 1: users (Core Authentication)

Description and Business Logic: The `users` table handles the strict authentication layer of the platform. It accommodates both traditional email/password logins and Google OAuth pipelines. The schema enforces strict data integrity through table-level constraints, ensuring that a user must have either a username or an email, and restricting the authentication provider strictly to recognized platforms.



id uuid	username text	email text	password_hash text	auth_provider text	is_active bool	created_at
0e3a0ec9-4e8b-4f8f-9ee5-3b582547c8f7	virat kohli	NULL	$2b$12$UZNSjd36d1KxTBLfdShv9.t7zdUI	local	TRUE	2026-02-15 05:
27134692-7f14-4071-a9ba-d13f15165906	rohitkhosare@gmail.com	NULL	$2b$12$/bpa9e6pJSSVTOOYEj7ziOOnxg	local	TRUE	2026-03-03 17:
34b6f3f5-2ecb-462d-9c5c-d1e708845594	jon snow	NULL	$2b$12$AxTbhBlz0h9RtKoHnGfo3emW/	local	TRUE	2026-02-14 14:
5572d458-f6da-4633-aa5e-237f092f4951	NULL	smartintern.org@gmail.com	NULL	google	TRUE	2026-02-25 16:
56c93e69-d47e-4dda-bac8-37f84ba53217	NULL	vaishpatil07a@gmail.com	NULL	google	TRUE	2026-02-25 16:
8cb99915-480b-4fe5-8a24-8541506c73c8	trail	NULL	$2b$12$D2YHb7yiYeCjpx57USJkQOxxb5	local	TRUE	2026-02-04 10:
a5d7421e-8ece-4c8f-a958-dd3cb9341bbe	prasad	NULL	$2b$12$OEeRizAEJx70huq/95zameTdsR(	local	TRUE	2026-02-23 14:
a93f9abf-4451-4215-8995-1eb8ce6463ed	NULL	ayushparkhe180@gmail.com	NULL	google	TRUE	2026-02-27 11:
ad4935dc-f034-4773-9368-515fea92b5c0	vaishnavi patil	NULL	$2b$12$pAknIqR63yYVLqV52aup7.iwsiav	local	TRUE	2026-02-11 17:
bc9c05c6-fcdb-49f4-8a78-7bbfec06f7b4	manik patil	NULL	$2b$12$gdH.d7uD39SyXw.IpDiNeeAJ9K'	local	TRUE	2026-02-18 07:
c3f07ca2-57c9-4194-a02c-7a8383d00a69	NULL	dipalikhosare460@gmail.com	NULL	google	TRUE	2026-03-02 14:
f652769d-c800-4c2b-8c40-21b8f98d8f65	ayush parkhe	NULL	$2b$12$kTG5ZBN6RyT3woxAAKFsvOQu	local	TRUE	2026-02-07 09:
f76f9803-a38f-4820-962b-ddc64f28aac9	NULL	parkheayush08@gmail.com	NULL	google	TRUE	2026-02-13 05:

Figure 6.2: Data snapshot of the `users` table showing authentication records.

Column Specifications:

- **`id` (Primary Key, UUID, Not Null):** Unique identifier automatically generated using the `gen_random_uuid()` function upon account creation.
- **`username` (TEXT, Unique, Nullable):** The user's chosen login handle. A table-level constraint (`user_identifier_check`) ensures that either this column or the email column must be populated.
- **`email` (TEXT, Unique, Nullable):** The user's contact and login email address.
- **`password_hash` (TEXT, Nullable):** Stores the securely encrypted password for users authenticating locally. This remains null for users logging in via third-party OAuth.
- **`auth_provider` (TEXT, Not Null):** Specifies the authentication method. A strict constraint (`users_auth_provider_check`) forces this value to be either `'local'` or `'google'`.
- **`is_active` (BOOLEAN, Default True):** A system flag indicating whether the account is currently active, suspended, or deactivated.
- **`created_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Automatically logs the exact time the user registered on the platform.

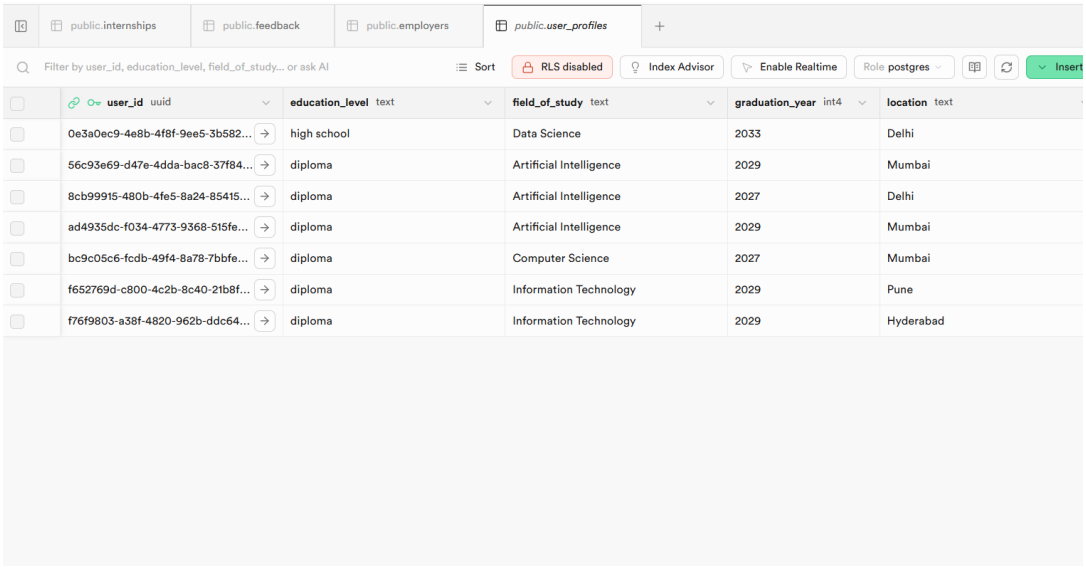
- **`updated_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Automatically tracks the timestamp of the most recent modification to the user's authentication data.
- **`display_name` (TEXT, Nullable):** The complete name of the user rendered on the frontend dashboard UI.

Table Constraints and Relationships:

- **Primary Key:** `users_pkey` on (`id`).
- **Unique Constraints:** `users_email_key` on (`email`), `users_username_key` on (`username`).
- **Relationships:** Acts as the parent table (1:1 with `user_profiles`, 1:N with `saved_opportunities`, `user_skills`, etc.).

6.3 Table 2: user_profiles (Demographic & Academic Data)

Description and Business Logic: The `user_profiles` table acts as the comprehensive repository for a student's personal, academic, and demographic information. By strictly separating this data from the `users` authentication table, the schema achieves excellent normalization. The demographic columns (`gender`, `category`, `family_income`, `disability_status`) are critical inputs for the Heuristic Engine to determine strict deterministic scholarship eligibility. Meanwhile, the academic and preference columns act as secondary filters for the AI internship matching pipeline.



user_id uuid	education_level text	field_of_study text	graduation_year int4	location text
0e3a0ec9-4e8b-4f8f-9ee5-3b582...	high school	Data Science	2033	Delhi
56c93e69-d47e-4dda-bac8-37f84...	diploma	Artificial Intelligence	2029	Mumbai
8cb99915-480b-4fe5-8a24-85415...	diploma	Artificial Intelligence	2027	Delhi
ad4935dc-f034-4773-9368-515fe...	diploma	Artificial Intelligence	2029	Mumbai
bc9c05c6-fcdb-49f4-8a78-7bbfe...	diploma	Computer Science	2027	Mumbai
f652769d-c800-4c2b-8c40-21b8f...	diploma	Information Technology	2029	Pune
f76f9803-a38f-4820-962b-ddc64...	diploma	Information Technology	2029	Hyderabad

Figure 6.3: Data snapshot of the `user_profiles` table displaying heuristic demographic criteria and academic preferences.

Column Specifications:

- **`user_id` (Primary Key & Foreign Key, UUID, Not Null):** Serves as both the primary key for this table and a foreign key linking back to the `users` table.
- **`education_level` (TEXT, Not Null):** The student's current academic standing (e.g., Undergraduate, Postgraduate).
- **`field_of_study` (TEXT, Not Null):** The major or domain of education (e.g., Computer Science, Mechanical).
- **`graduation_year` (INTEGER, Not Null):** The expected or actual year of graduation. A strict table-level constraint (`graduation_year_check`) ensures this value must fall between 2015 and 2035.
- **`location` (TEXT, Not Null):** The user's current city or region, used for geographical internship filtering.
- **`experience_level` (TEXT, Not Null):** Categorizes the user's professional background (e.g., Fresher, Intermediate).
- **`preferred_mode` (TEXT, Not Null):** Indicates work style preference (e.g., Remote, On-site, Hybrid).
- **`preferred_type` (TEXT, Not Null):** Indicates opportunity type preference (e.g., Full-time, Part-time, Internship).

- **`availability_duration` (TEXT, Not Null):** The timeframe the student is available to work (e.g., 3 months, 6 months).
- **`created_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Automatically logs the exact time the profile was generated.
- **`updated_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Automatically tracks the timestamp of the most recent profile modification.
- **`full_name` (TEXT, Nullable):** Redundant display name for optimized UI querying.
- **`dob` (DATE, Nullable):** Date of Birth, used for age-restricted scholarship criteria.
- **`gender` (TEXT, Nullable):** Required for gender-specific demographic scholarship filtering.
- **`category` (TEXT, Nullable):** Social classification (e.g., Open, OBC, SC/ST) processed by the backend Regex engine.
- **`disability_status` (TEXT, Nullable):** Boolean-style flag indicating if the user qualifies for PwD (Persons with Disabilities) schemes.
- **`disability_type` (TEXT, Nullable):** Specifies the nature of the disability for granular grant matching.
- **`family_income` (TEXT, Nullable):** Income threshold metric for financial aid eligibility.
- **`institution_type` (TEXT, Nullable):** Classification of the college (e.g., Government, Private).
- **`academic_score` (TEXT, Nullable):** CGPA or percentage metric.

Table Constraints and Relationships:

- **Primary Key:** `user_profiles_pkey` on (`user_id`).
- **Foreign Key:** `user_profiles_user_id_fkey` links `user_id` to `users(id)` with an `ON DELETE CASCADE` rule. If a user deletes their main account, their demographic profile is automatically wiped to ensure data privacy.
- **Check Constraints:** `graduation_year_check` enforces valid date ranges (2015-2035).

Indexes and Database Triggers:

- **Indexes:** A B-Tree index (`idx_user_profiles_user`) is applied to the `user_id` column to drastically optimize read speeds during dashboard rendering.
- **Triggers:** A database trigger (`trg_user_profiles_updated`) automatically executes the `update_updated_at_column()` function before any row update, ensuring the `updated_at` timestamp is always perfectly accurate without requiring backend Python logic.

6.4 Table 3: employers (Employer Authentication and Verification)

Description and Business Logic: The `employers` table serves as the primary authentication and metadata repository for companies looking to post internships on the platform. Unlike student users who heavily utilize Google OAuth, this table manages local authentication via password hashing. Crucially, the schema includes a robust verification mechanism using the Corporate Identification Number (CIN) and an administrative approval status workflow. This ensures that only legitimate, verified organizations can access the recruitment dashboard and post opportunities.

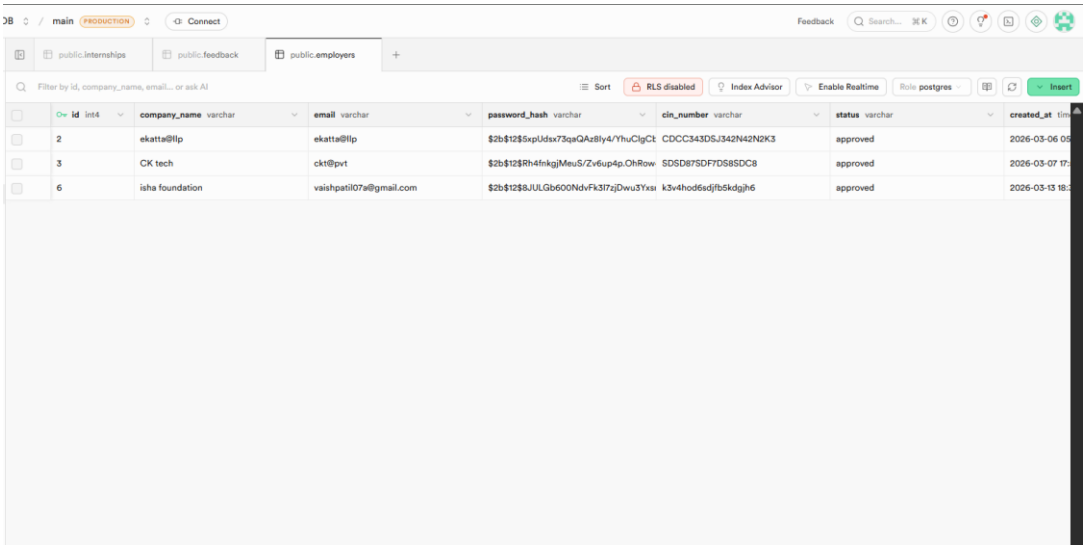


Figure 6.4: Data snapshot of the `employers` table showing authentication records and verification statuses.

Column Specifications:

- **`id` (Primary Key, SERIAL, Not Null):** An auto-incrementing integer identifier for the employer. Utilizing a serial integer here optimizes the indexing for employer-specific queries.
- **`company_name` (VARCHAR(255), Not Null):** The official, legally registered name of the organization.
- **`email` (VARCHAR(255), Unique, Not Null):** The official outreach and login email for the company representative.
- **`password_hash` (VARCHAR(255), Not Null):** Securely encrypted password string for local authentication, preventing plain-text credential exposure in the database.
- **`cin_number` (VARCHAR(50), Nullable):** The Corporate Identification Number. This is utilized by platform administrators to legally verify the company against government registries.
- **`status` (VARCHAR(20), Default 'pending'):** A state-management flag (e.g., 'pending', 'approved', 'rejected'). The default 'pending' state restricts employer privileges (like posting jobs) until administrative verification is complete.
- **`created_at` (TIMESTAMP WITHOUT TIME ZONE, Default `CURRENT_TIMESTAMP`):** Automatically logs the exact time the employer registered on the platform.

Table Constraints and Relationships:

- **Primary Key:** `employers_pkey` applied to (`id`).
- **Unique Constraints:** `employers_email_key` applied to (`email`) ensures no duplicate employer accounts can be created with the same contact address.
- **Relationships:** Acts as the parent table (1:N) for the `internships` table, linking every posted job to a verified company.

6.5 Table 4: internships (Career Opportunities & AI Target Dataset)

Description and Business Logic: The `internships` table serves as the central repository for all career opportunities on the platform and acts as the primary target dataset for the Natural Language Processing (TF-IDF) engine. The architectural brilliance of this schema lies in its dual-ingestion capability. It natively supports direct job postings from verified `employers` (via the `employer_id` foreign key) while simultaneously supporting automated data aggregation from external sources (via web scraping). To maintain high data integrity during aggregation, it utilizes a `content_hash` and strict compound indexes to completely eliminate duplicate job postings.

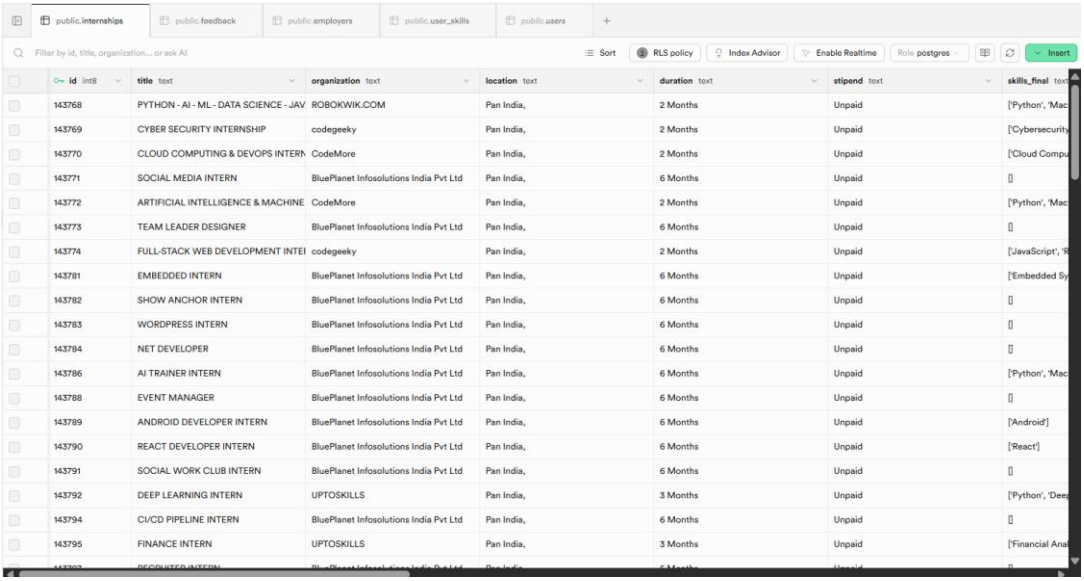


Figure 6.5: Data snapshot of the `internships` table showing both locally hosted and aggregated job postings.

Column Specifications:

- **`id` (Primary Key, BIGSERIAL, Not Null):** A massive auto-incrementing integer identifier capable of handling millions of rows, future-proofing the table for large-scale data scraping.
- **`title` (TEXT, Not Null):** The official role title (e.g., "Software Engineering Intern").

- **`organization` (TEXT, Not Null):** The name of the company offering the role.
- **`location` (TEXT, Nullable):** The geographical location or "Remote" designation.
- **`duration` (TEXT, Nullable):** The expected length of the internship (e.g., "3 Months", "6 Months").
- **`stipend` (TEXT, Nullable):** The financial compensation. Stored as text to accommodate ranges or specific phrases (e.g., "Unpaid", "10k - 15k / month").
- **`skills_final` (TEXT, Nullable): The critical AI target column.** This houses the explicit technical requirements expected by the employer, which the backend Cosine Similarity engine actively matches against the `user_skills` table.
- **`posted_on` & `start_date` (TEXT, Nullable):** Chronological metadata for the opportunity.
- **`type` (TEXT, Nullable):** Classification of the role (e.g., Full-time, Part-time).
- **`source` (TEXT, Nullable):** Identifies the origin of the data (e.g., "Internal", "LinkedIn", "Internshala"), tracking whether the job was posted natively or scraped.
- **`apply_link` (TEXT, Nullable):** The external URL for scraped jobs where the student is ultimately redirected to apply.
- **`scraped_at` (TIMESTAMP WITH TIME ZONE, Nullable):** Logs the exact execution time of the web scraping script that fetched the row.
- **`content_hash` (TEXT, Nullable):** A unique cryptographic hash generated from the job's core details. This is used programmatically during data insertion to instantly identify and reject duplicate entries.
- **`extra_data` (JSONB, Nullable):** A highly flexible, NoSQL-style JSON binary column. This allows the system to store unpredictable or heavily nested data from external APIs without requiring constant schema migrations.
- **`created_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Record creation timestamp.

- **`employer_id` (INTEGER, Nullable):** The Foreign Key linking a natively posted job to a verified internal employer.

Table Constraints and Relationships:

- **Primary Key:** `internships_pkey` applied to (`id`).
- **Foreign Key:** `internships_employer_id_fkey` connects to `employers(id)` with an `ON DELETE CASCADE` rule (if an employer is removed, all their natively posted jobs are deleted).
- **Relationships:** Acts as the child table (N:1) to `employers`, and the parent dataset for user AI matching.

Indexes and Database Optimizations:

- **Compound Unique Indexes:** The database enforces uniqueness using a B-Tree index (`uniq_title_organization`) spanning both the `title` and `organization` columns. This guarantees at the database level that the same company cannot have multiple identical job listings, keeping the AI matrices clean and efficient.

6.6 Table 5: scholarships (Deterministic Financial Aid & Aggregation)

Description and Business Logic: The `scholarships` table stores financial aid opportunities and operates strictly alongside the Heuristic Matching Engine. Unlike the fuzzy logic utilized for internship matching, scholarships require absolute deterministic filtering. The backend Regular Expressions (Regex) engine parses columns like `category` and `eligibility_text` to verify against a student's demographic profile. Furthermore, the schema mirrors the aggregated architecture of the internships table, utilizing cryptographic hashing and strict indexing to deduplicate financial aid data scraped from various external portals.

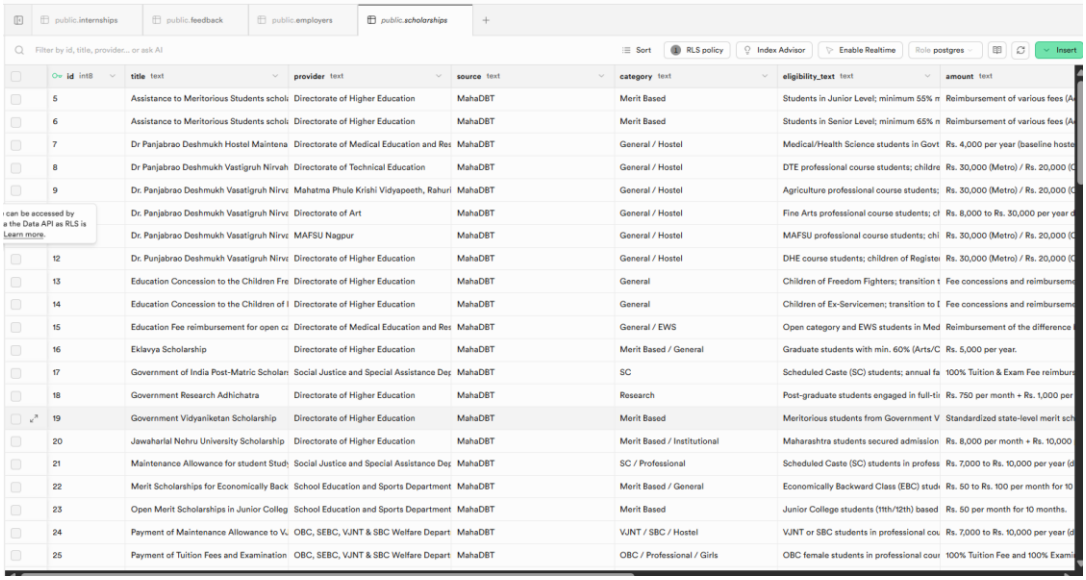


Figure 6.6: Data snapshot of the `scholarships` table showcasing deterministic eligibility criteria and aggregated sources.

Column Specifications:

- **`id` (Primary Key, BIGSERIAL, Not Null):** A massive auto-incrementing integer identifier, efficiently indexing the table for high-volume automated data insertion.
- **`title` (TEXT, Not Null):** The official name of the scholarship or grant program.
- **`provider` (TEXT, Nullable):** The government body, NGO, university, or private institution funding the financial aid.
- **`source` (TEXT, Not Null):** Identifies the origin of the data (e.g., "National Scholarship Portal", "Buddy4Study"), tracking the specific external site from which the grant was aggregated.
- **`category` (TEXT, Nullable):** The specific social or demographic classification the scholarship is targeted toward (e.g., "OBC", "Women in STEM", "EWS"). This is a primary target for the regex filtering engine.
- **`eligibility_text` (TEXT, Nullable):** Unstructured text outlining the exact academic or demographic requirements.
- **`amount` (TEXT, Nullable):** The total monetary value. Stored as text to accommodate variable formats like ranges or recurring stipends (e.g., "Up to ₹50,000", "₹10,000/year").

- **`deadline` (DATE, Nullable):** The expiration date of the scholarship. The backend utilizes this to programmatically filter out expired opportunities from the user dashboard.
- **`apply_url` (TEXT, Not Null):** The external link where the student is redirected to formally submit their application.
- **`created_at` (TIMESTAMP WITHOUT TIME ZONE, Default `CURRENT_TIMESTAMP`):** Automatically logs the time the record was inserted into the database.
- **`updated_at` (TIMESTAMP WITHOUT TIME ZONE, Default `CURRENT_TIMESTAMP`):** Tracks the last modification to the scholarship's details.
- **`content_hash` (TEXT, Nullable):** A unique cryptographic hash generated from the scholarship's core details, utilized to programmatically detect and reject duplicate scraping insertions.

Table Constraints and Relationships:

- **Primary Key:** `scholarships_pkey` applied to (`id`).
- **Unique Constraints:** `scholarships_content_hash_unique` applied to (`content_hash`) ensures absolute data uniqueness at the hash level.
- **Compound Unique Constraints:** `scholarships_identity_unique` applied to (`title`, `source`) acts as a secondary failsafe, ensuring that the same source cannot list the exact same scholarship title twice.

6.7 Table 6: user_skills (Student NLP Input)

Description and Business Logic: The `user_skills` table is a highly normalized entity designed specifically to feed unstructured text into the backend Natural Language Processing pipeline. Instead of storing a user's technical abilities as a comma-separated string within the main profile, this table isolates each skill into its own row. To maintain strict data integrity and prevent AI weighting errors caused

by duplicate entries, the table utilizes a composite primary key. The extracted text from this table acts as the foundational baseline vector for the TF-IDF and Cosine Similarity math.

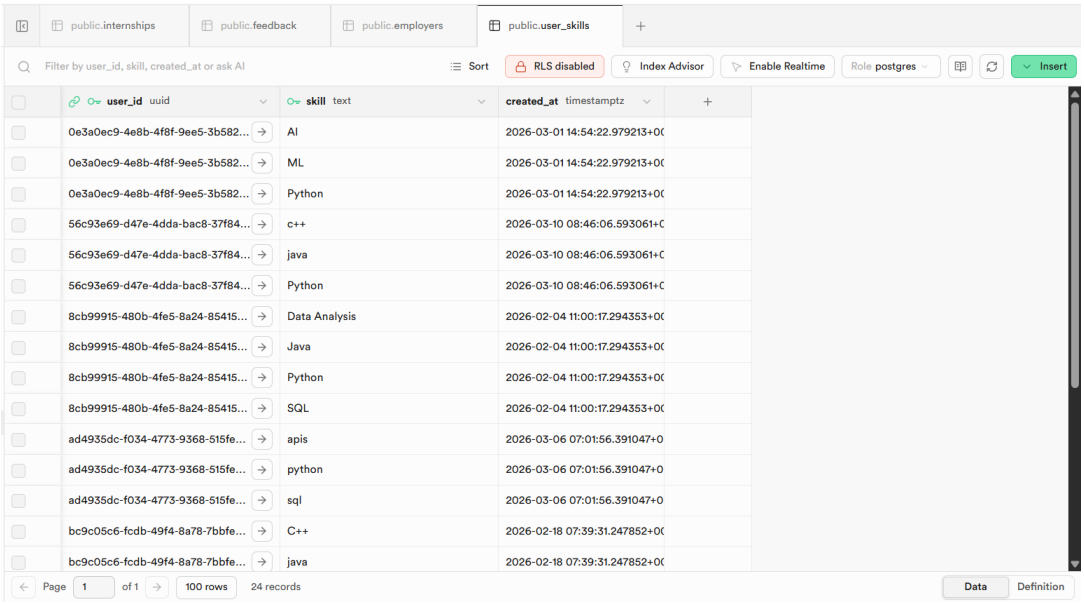


Figure 6.7: Data snapshot of the `user_skills` table highlighting isolated NLP text inputs.

Column Specifications:

- **`user_id` (Primary & Foreign Key, UUID, Not Null):** The unique identifier linking the skill back to the specific student.
- **`skill` (Primary Key, TEXT, Not Null):** The actual technical competency or tool (e.g., "Python", "React", "Machine Learning"). This column provides the raw string data that is transformed into numerical matrices.
- **`created_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Automatically logs the exact time the student added the skill to their profile.

Table Constraints and Relationships:

- **Composite Primary Key:** `user_skills_pkey` applied to the combination of (`user_id`, `skill`). This enforces absolute uniqueness; a single user ID cannot have two identical skill rows.
- **Foreign Key:** `user_skills_user_id_fkey` connects `user_id` to `users(id)` with a strict `ON DELETE CASCADE` rule. If a user deletes

their account, all associated NLP skill vectors are instantly purged from the system.

- **Relationships:** Acts as a child table (N:1) to the central `users` table.

Indexes and Database Optimizations:

- **User Index:** A B-Tree index (`idx_user_skills_user`) is applied to the `user_id` column. This optimizes the system's ability to rapidly fetch all skills belonging to a single user when generating their personalized recommendation dashboard.
- **Skill Index:** A B-Tree index (`idx_user_skills_skill`) is applied to the `skill` column. This optimizes reverse-lookups, allowing the database to instantly group all users who possess a specific technology (which is highly beneficial for future Collaborative Filtering algorithms).

6.8 Table 7: internship_skills (Target NLP Input)

Description and Business Logic: The `internship_skills` table acts as the target dataset for the backend Natural Language Processing engine. Instead of storing the employer's required skills as an unstructured block of text within the main internship record, this highly normalized table breaks them down into individual, queryable rows. By utilizing a composite primary key, the database intrinsically prevents identical skills from being tied to the same job posting, ensuring the mathematical weights of the AI matrices remain accurate and unskewed.

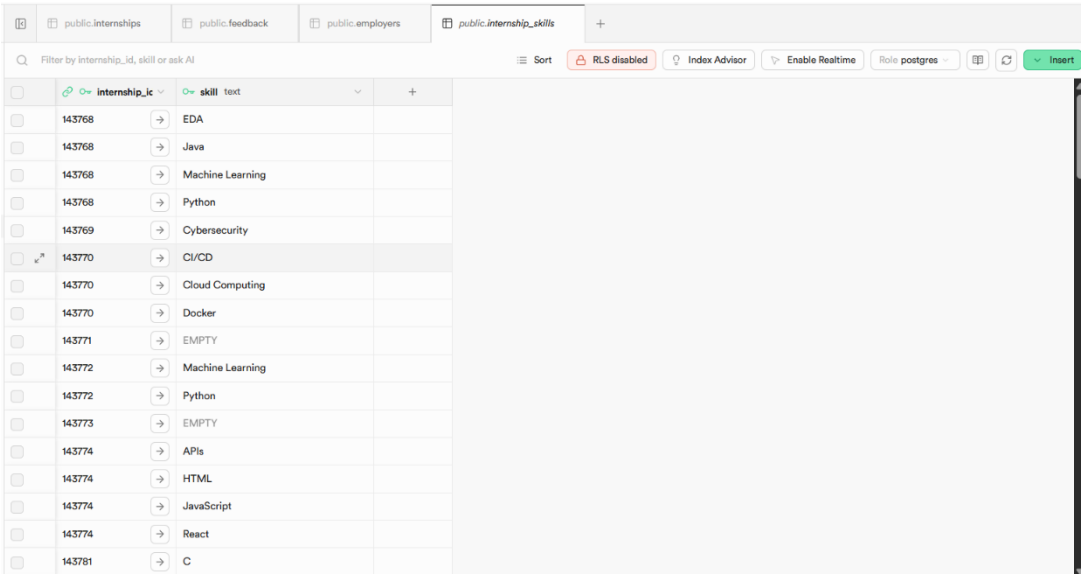


Figure 6.8: Data snapshot of the `internship_skills` table showing normalized employer requirements.

Column Specifications:

- **`internship_id` (Primary & Foreign Key, BIGINT, Not Null):** The unique integer identifier linking the required skill back to the specific career opportunity.
- **`skill` (Primary Key, TEXT, Not Null):** The explicitly required technology or competency (e.g., "React", "SQL", "Flask"). This column forms the target baseline string that the AI compares against the student's profile.

Table Constraints and Relationships:

- **Composite Primary Key:** `internship_skills_pkey` applied to the combination of (`internship_id`, `skill`). This strictly enforces uniqueness; a single internship cannot list the exact same required skill twice.
- **Foreign Key:** `internship_skills_internship_id_fkey` connects `internship_id` to `internships(id)` with a strict `ON DELETE CASCADE` rule. If an internship posting expires or is deleted by an employer, all its associated required skills are instantly removed from the database, preventing orphaned data.

- **Relationships:** Acts as a child table (N:1) to the central `internships` table.

Indexes and Database Optimizations:

- **Skill Index:** A B-Tree index (`idx_internship_skills_skill`) is applied to the `skill` column. This is a critical database optimization. When a student logs in, this index allows the backend to instantly scan thousands of internships and rapidly isolate the ones that contain the student's specific skills, drastically reducing the AI computation time.

6.9 Table 8: user_interests (Secondary AI Weighting)

Description and Business Logic: While technical skills dictate a student's capability, their interests dictate their career trajectory and passion. The `user_interests` table provides supplementary Natural Language Processing (NLP) text data (e.g., "Artificial Intelligence", "Open Source", "FinTech"). By factoring these interests into the TF-IDF vectorization process, the AI can weight recommendations not just on what a student is qualified for, but what they actively want to pursue. Similar to the skills tables, this schema enforces strict data normalization utilizing a composite primary key to prevent duplicate entries from skewing the mathematical weights.

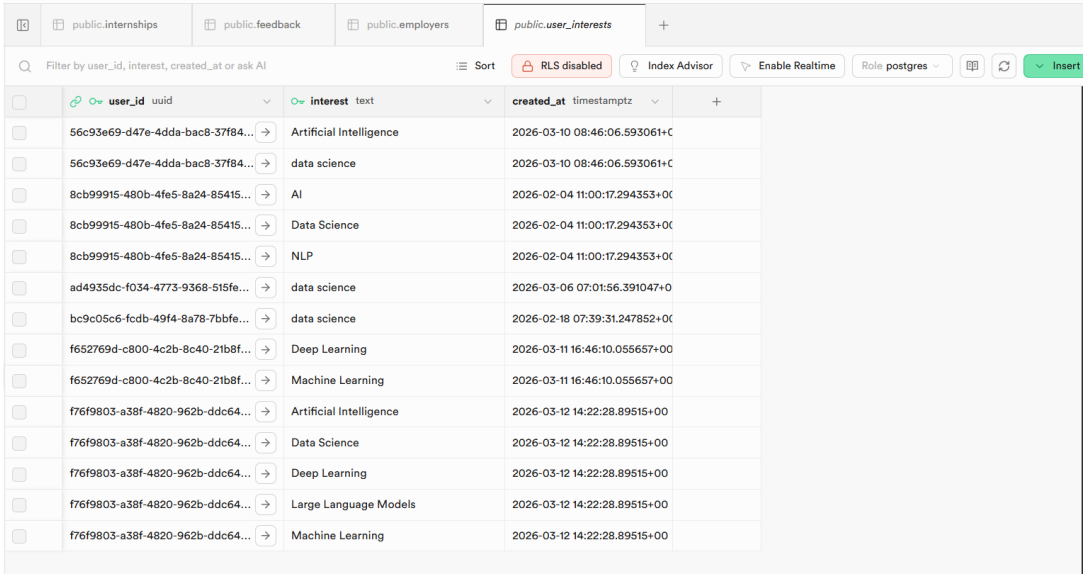


Figure 6.9: Data snapshot of the `user_interests` table used for supplementary AI weighting.

Column Specifications:

- **`user_id` (Primary & Foreign Key, UUID, Not Null):** The unique identifier linking the specific interest back to the student's central authentication record.
- **`interest` (Primary Key, TEXT, Not Null):** The actual domain, hobby, or career focus string. This acts as secondary text data injected into the AI's Cosine Similarity matrix.
- **`created_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Automatically logs the exact time the student added the interest to their profile.

Table Constraints and Relationships:

- **Composite Primary Key:** `user_interests_pkey` applied to the combination of (`user_id`, `interest`). This strict database constraint prevents a user from adding the exact same interest twice, maintaining clean AI vectors.
- **Foreign Key:** `user_interests_user_id_fkey` connects `user_id` to `users(id)` with an `ON DELETE CASCADE` rule. If a user deletes their account, their entire web of interests is securely erased alongside their identity.

- **Relationships:** Acts as a child table (N:1) to the central `users` table.

6.10 Table 9: saved_opportunities (Transactional Mapping & Engagement)

Description and Business Logic: The `saved_opportunities` table is a transactional mapping entity that tracks user engagement (bookmarks). Architecturally, it utilizes a "Polymorphic Association" design pattern. Instead of creating separate tables for saved internships and saved scholarships, this single table handles both by utilizing an `opportunity_type` discriminator column. This table is not only critical for rendering the user's saved dashboard UI, but it also serves as the foundational interaction dataset required to implement Collaborative Filtering AI algorithms in future project iterations.

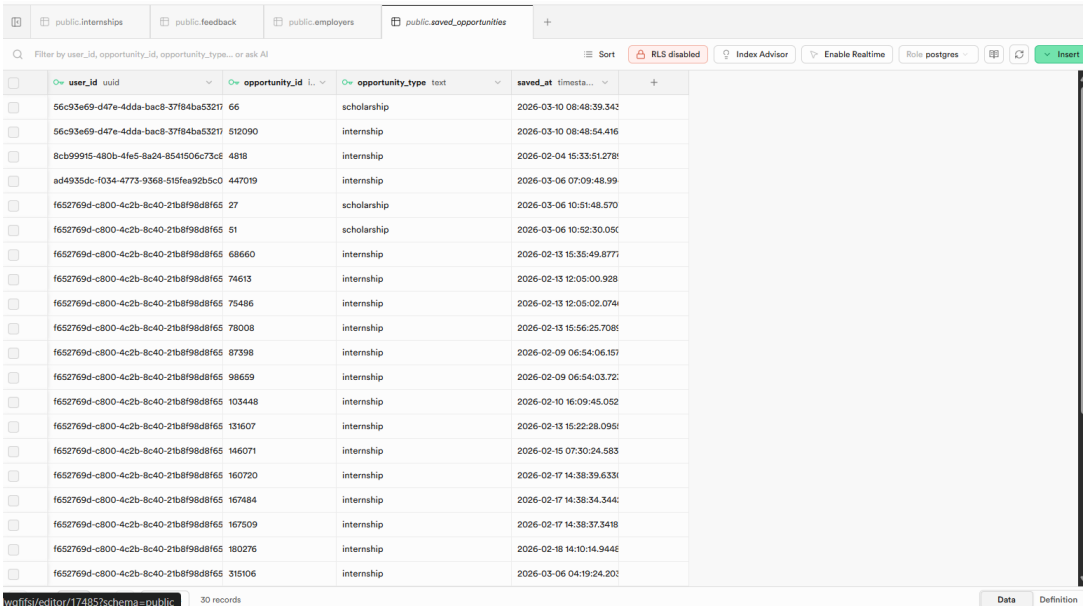


Figure 6.10: Data snapshot of the `saved_opportunities` mapping table tracking polymorphic user engagement.

Column Specifications:

- **`user_id` (Primary Key, UUID, Not Null):** The unique identifier representing the student taking the save/bookmark action.
- **`opportunity_id` (Primary Key, BIGINT, Not Null):** The target ID of the saved item. Because this ID can point to either an internship or a

scholarship, it utilizes the `BIGINT` format to match the primary keys of both target tables.

- **`opportunity_type` (Primary Key, TEXT, Not Null):** The discriminator column (e.g., 'internship' or 'scholarship'). This explicitly tells the backend Flask router which database table to query when fetching the full details of the saved item.
- **`saved_at` (TIMESTAMP WITHOUT TIME ZONE, Default `CURRENT_TIMESTAMP`):** Automatically records the exact time the interaction occurred, essential for chronological sorting on the frontend UI.

Table Constraints and Relationships:

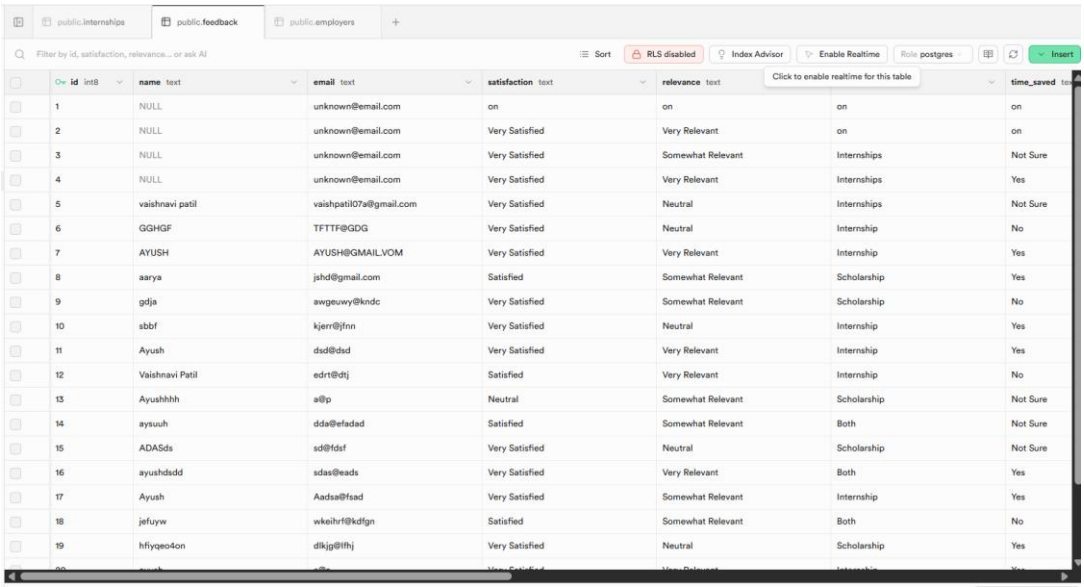
- **Composite Primary Key:** `saved_opportunities_pkey` applied to a three-part combination of (`user_id`, `opportunity_id`, `opportunity_type`). This is a brilliant structural constraint; it prevents a user from saving the same opportunity twice, while safely allowing a user to save an internship with ID '5' and a scholarship with ID '5' without a primary key collision.
- **Relationships:** Acts as the Many-to-Many (N:M) resolution table connecting the `users` table to both the `internships` and `scholarships` tables.

Indexes and Database Optimizations:

- **Composite B-Tree Index:** An index (`idx_saved_user`) is applied across both (`user_id`, `opportunity_type`). This optimization allows the database to instantly retrieve and filter a specific user's saved items by category (e.g., instantly fetching "User A's Saved Scholarships") without performing a full table scan.

6.11 Table 10: feedback (System Evaluation & User Experience)

Description and Business Logic: The `feedback` table acts as the quantitative and qualitative evaluation repository for the SmartIntern platform. Collecting direct user feedback is a critical component of the iterative Software Development Life Cycle (SDLC). By capturing granular metrics such as algorithmic relevance, time saved, and UI ease-of-use, the system administrators can continuously refine the recommendation matrices and user interface based on empirical data rather than assumptions.



id int8	name text	email text	satisfaction text	relevance text		time_saved
1	NULL	unknown@email.com	on	on	on	on
2	NULL	unknown@email.com	Very Satisfied	Very Relevant	on	on
3	NULL	unknown@email.com	Very Satisfied	Somewhat Relevant	Internships	Not Sure
4	NULL	unknown@email.com	Very Satisfied	Very Relevant	Internships	Yes
5	vaishnavi patil	vaishpatil07a@gmail.com	Very Satisfied	Neutral	Internships	Not Sure
6	GGHGF	TFTTF@GDG	Very Satisfied	Neutral	Internship	No
7	AYUSH	AYUSH@GMAIL.VOM	Very Satisfied	Very Relevant	Internship	Yes
8	aarya	jshd@gmail.com	Satisfied	Somewhat Relevant	Scholarship	Yes
9	gdja	awgeuwy@kndc	Very Satisfied	Somewhat Relevant	Scholarship	No
10	sbbf	kjerr@jfnn	Very Satisfied	Neutral	Internship	Yes
11	Ayush	dsd@dsd	Very Satisfied	Very Relevant	Internship	Yes
12	Vaishnavi Patil	edrt@dtj	Satisfied	Very Relevant	Internship	No
13	Ayushhhh	a@p	Neutral	Somewhat Relevant	Scholarship	Not Sure
14	aysuuh	dda@efadad	Satisfied	Somewhat Relevant	Both	Not Sure
15	ADASds	sd@fdsf	Very Satisfied	Neutral	Scholarship	Not Sure
16	ayushdsdd	sdas@eads	Very Satisfied	Very Relevant	Both	Yes
17	Ayush	Aadsa@fsad	Very Satisfied	Somewhat Relevant	Internship	Yes
18	jefuyw	wkeihrf@kdfgn	Satisfied	Somewhat Relevant	Both	No
19	hfiyqeo4on	dlkjg@fhj	Very Satisfied	Neutral	Scholarship	Yes

Figure 6.11: Data snapshot of the `feedback` table capturing quantitative UI/UX and algorithmic evaluation metrics.

Column Specifications:

- **`id` (Primary Key, BIGINT, Not Null):** The unique identifier for the feedback submission. This column utilizes the modern PostgreSQL `GENERATED ALWAYS AS IDENTITY` constraint, which is a highly secure, SQL-compliant method of auto-incrementing integers that prevents manual primary key overriding.
- **`name` (TEXT, Nullable):** The user's name (optional, to allow for anonymous feedback).
- **`email` (TEXT, Not Null):** The contact address of the user submitting the evaluation, ensuring administrators can follow up on critical bug reports.

- **`satisfaction & overall_experience` (TEXT, Nullable):** High-level metrics tracking general user sentiment.
- **`relevance` (TEXT, Nullable):** A critical metric evaluating the perceived accuracy of the AI recommendation engine's TF-IDF outputs.
- **`used_type` (TEXT, Nullable):** Identifies whether the user is primarily evaluating the internship engine, the scholarship engine, or both.
- **`time_saved & ease` (TEXT, Nullable):** Metrics evaluating the efficiency and usability of the platform compared to traditional manual searching.
- **`personalization` (TEXT, Nullable):** Evaluates how well the Cold Start mitigation strategies and AI tailoring performed.
- **`improve & recommend` (TEXT, Nullable):** Free-form text fields for qualitative feature requests and Net Promoter Score (NPS) tracking.
- **`created_at` (TIMESTAMP WITH TIME ZONE, Default `now()`):** Automatically logs the exact time the feedback was submitted.

Table Constraints and Relationships:

- **Primary Key:** `feedback_pkey` applied to (`id`).
- **Identity Constraint:** `GENERATED ALWAYS AS IDENTITY` ensures the database strictly controls the auto-increment sequence, preventing application-level insertion errors.

6.12 Table 11: contact_messages (Administrative Routing & Support)

Description and Business Logic: The `contact_messages` table handles asynchronous communication between end-users and the platform administrators. It acts as the backend storage for the application's "Contact Us" or support forms. A critical architectural feature of this table is its native anti-spam mechanism. By implementing a compound unique constraint on the sender's email and the message body, the database inherently prevents duplicate submissions caused by network

latency or accidental double-clicks on the frontend UI, ensuring administrative dashboards remain clean.

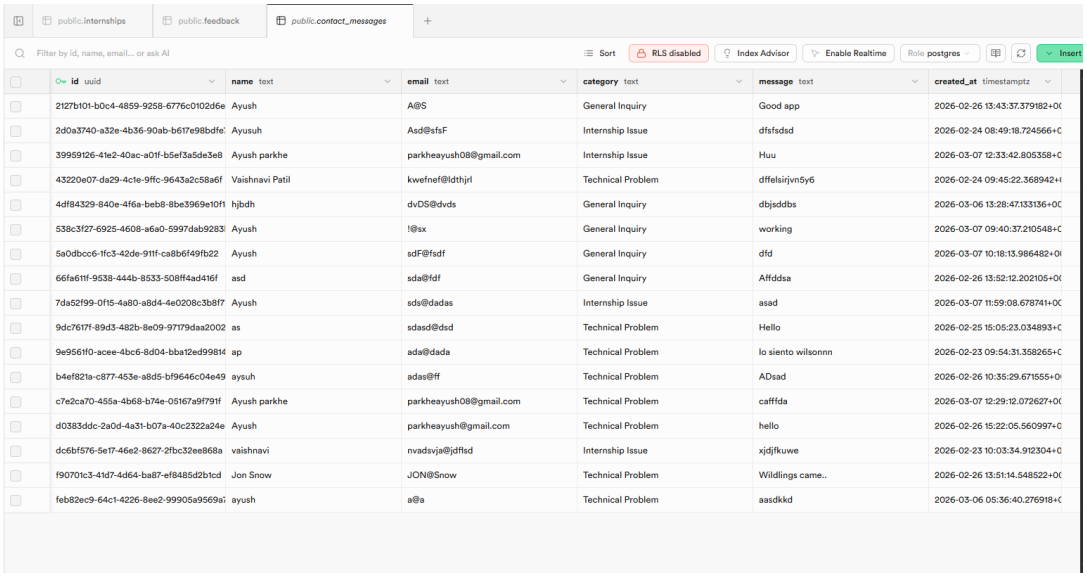


Figure 6.12: Data snapshot of the `contact_messages` table utilized for administrative inquiry routing and support.

Column Specifications:

- **`id` (Primary Key, UUID, Not Null):** A universally unique identifier generated automatically using the `gen_random_uuid()` cryptographic function upon record insertion.
- **`name` (TEXT, Not Null):** The name of the user or visitor submitting the inquiry.
- **`email` (TEXT, Not Null):** The contact address of the sender, allowing administrators to route replies effectively.
- **`category` (TEXT, Not Null):** The classification of the inquiry (e.g., 'Technical Support', 'Partnership', 'Bug Report'). This allows the administrative backend to sort and prioritize incoming tickets.
- **`message` (TEXT, Not Null):** The full, unstructured content of the user's inquiry.
- **`created_at` (TIMESTAMP WITH TIME ZONE, Not Null, Default `now()`):** Automatically logs the exact time the message was securely recorded by the system.

Table Constraints and Relationships:

- **Primary Key:** `contact_messages_pkey` applied to (`id`).
- **Compound Unique Constraint:** `unique_email_message` applied to the combination of (`email`, `message`). This strict database-level constraint serves as an automated debounce and anti-spam filter, outright rejecting duplicate identical submissions from the same user.

CHAPTER 7: RECOMMENDATION ENGINE ARCHITECTURE & ALGORITHMS

The core technical achievement of the SmartIntern platform is its dual-engine recommendation architecture. Because the system handles two fundamentally different types of opportunities—internships (which require probabilistic matching) and scholarships (which require strict deterministic filtering)—a monolithic algorithmic approach was structurally unviable.

To resolve this, the system routes data through two distinct mathematical pipelines. This chapter details the theoretical foundations, data preprocessing steps, and mathematical formulas utilized to generate the final personalized user dashboard.

7.1 Theoretical Foundations of the Recommender System

Recommender systems generally fall into three categories: Collaborative Filtering (behavior-driven), Content-Based Filtering (metadata-driven), and Hybrid systems.

Given the "User Cold Start" problem inherent in new platforms lacking historical application data, SmartIntern was engineered utilizing a strict **Content-Based Filtering** paradigm for internships, and a **Rule-Based Constraint Satisfaction** paradigm for scholarships. By treating the student's technical skills and the employer's requirements as unstructured text documents, the system leverages Natural Language Processing (NLP) to mathematically calculate the semantic alignment between the two entities.

7.2 The Natural Language Preprocessing Pipeline

Before any mathematical vectorization can occur, the raw text data extracted from the `user_skills`, `user_interests`, and `internships` tables must be normalized. Raw text contains noise that can severely skew the vector matrices. The system executes the following preprocessing steps sequentially:

1. **Tokenization & Lowercasing:** All incoming text strings are converted to lowercase to ensure case insensitivity (e.g., treating "Python" and "python"

as identical entities). The text is then tokenized, breaking sentences into individual algorithmic features.

2. **Noise Reduction:** Special characters, punctuation, and numerical artifacts are stripped from the strings.
3. **Stop-Word Removal:** Common English words that carry zero semantic weight (e.g., "the," "and," "is," "required") are programmatically purged from the dataset. This ensures the AI only processes high-value technical vocabulary.

7.3 Domain Knowledge Injection (The "Smart Dictionary" Algorithm)

A fundamental limitation of standard NLP is the "Semantic Gap"—the inability of an algorithm to understand that two linguistically different words represent the same concept. For example, if a student inputs the skill "React" and an employer requires "Frontend UI," standard vectorization yields a similarity score of exactly zero.

To bridge this gap, a "Smart Dictionary" algorithm was engineered into the preprocessing pipeline. Before vectorization, the system evaluates the student's input against a predefined domain mapping matrix.

Algorithmic Execution:

- The system parses the cleaned user string.
- It searches for root technical nodes (e.g., "Node.js").
- Upon detection, it dynamically concatenates predefined semantic synonyms directly into the underlying text string (e.g., appending "backend," "server," "express," "javascript").
- This transforms a sparse, single-word input into a rich, dense semantic document, drastically improving the accuracy of the subsequent mathematical processing.

7.4 Mathematical Formulation of the Content-Based Engine

Once the text is expanded and cleaned, the system converts the unstructured strings into numerical matrices using **Term Frequency-Inverse Document Frequency (TF-IDF)**, a statistical measure used to evaluate how important a word is to a document within a larger corpus.

7.4.1 Term Frequency (TF)

Term Frequency measures the raw occurrence of a specific skill within a document. However, to prevent bias toward longer job descriptions, the frequency is normalized by the total length of the document:

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in a document } d}{\text{Total number of terms in the document } d}$$

7.4.2 Inverse Document Frequency (IDF)

While TF measures occurrence, IDF measures the informational rarity of the word. If a word like "software" appears in 90% of all internships, it is not a strong differentiator. Conversely, a highly specific skill like "TensorFlow" appearing in only 2% of postings is highly valuable. The IDF formula penalizes common terms and boosts rare ones:

$$IDF(t) = \log_{10}\left(\frac{N}{DF(t)}\right)$$

(Where N is the total number of internship postings, and $DF(t)$ is the number of postings containing the specific term t).

7.4.3 The Final TF-IDF Matrix

The final vector weight for every single word is calculated by multiplying these two metrics:

$$W(t,d) = TF(t,d) \times IDF(t)$$

7.4.4 Cosine Similarity Calculation

Once the TF-IDF matrices are generated—representing the student's profile as Vector {A} and the internship as Vector {B}—the system calculates the geometric angle between them in multi-dimensional space. Cosine Similarity is utilized over Euclidean Distance because it measures the *orientation* rather than the magnitude, making it immune to variations in text length:

$$\text{Similarity Score} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

This formula outputs a raw decimal score between 0.0 (no match) and 1.0 (perfect match).

7.5 The Weighted Ensemble Aggregation

To provide highly personalized recommendations, the AI does not rely on a single vector comparison. Relying solely on skills ignores a user's passion, while relying solely on interests ignores their technical capability.

Instead, the algorithm calculates three separate Cosine Similarity scores and combines them into a final weighted ensemble. The weights were fine-tuned during the iterative testing phase:

- **S_{skills} (Technical Capability):** Weighted at 45% (0.45)
- **$S_{interests}$ (Career Passion):** Weighted at 40% (0.40)
- **$S_{location}$ (Geographical Feasibility):** Weighted at 15% (0.15)

The final output score is calculated via the following linear equation:

$$\text{Final Match Score} = (S_{skills} \times 0.45) + (S_{interests} \times 0.40) + (S_{location} \times 0.15)$$

Internships scoring below a baseline threshold of \$0.01\$ are programmatically discarded. The remaining opportunities are sorted in descending order and pushed to the frontend UI.

7.6 Deterministic Rule-Based Engine for Scholarships

Unlike internships, the distribution of financial aid is bound by strict legal and demographic constraints. A probabilistic 95% mathematical match is unacceptable if a grant requires a specific demographic category. Therefore, the secondary engine bypasses the TF-IDF NLP pipeline entirely and utilizes a **Constraint Satisfaction Problem (CSP)** framework executed via Regular Expressions.

The system executes a sequential, heuristic decision tree against the eligibility criteria of every aggregated scholarship in the database:

- **Phase 1: Boolean Gender Exclusion** The algorithm scans the text for explicit gender requirements. If a scholarship is tagged exclusively for female applicants, the system cross-references the student's profile constraint. If the student evaluates to male, the algorithm assigns an absolute `False` boolean, forcefully terminating the evaluation of that specific grant and removing it from the queue.
- **Phase 2: Demographic Category Prioritization**

  The algorithm cross-references the student's social category against the grant provider's criteria. Explicit string matches trigger a positive score multiplier (\$+50\$ points). Conversely, if the student belongs to the 'Open/General' category and the engine detects minority-exclusive keywords within the string, a heavy penalty (\$-50\$ points) is applied.

- **Phase 3: Disability Grant Isolation**

  To support accessibility, the engine actively scans the criteria for specialized disability vocabulary. Upon detection, and verifying the student's

`disability_status` boolean is `True`, it grants a maximum score boost. This guarantees that these highly specific, high-value grants override standard scoring and appear at the absolute top of the student's feed.

Opportunities that successfully navigate this strict heuristic constraint matrix are tabulated by their accumulated score, sorted, and rendered on the scholarship dashboard.

CHAPTER 8: SYSTEM FLOW WITH OUTPUTS

This chapter provides a visual walkthrough of the SmartIntern platform, demonstrating the execution of the system architecture in a live environment. The system flow illustrates the complete user journey from authentication and data collection to the final algorithmic rendering of personalized career and financial aid recommendations.

8.1 User Authentication (Google OAuth)

The system strictly relies on third-party authentication to ensure security. This output demonstrates the initial landing page where the user is prompted to log in using their existing Google credentials, bypassing the need for manual password management.

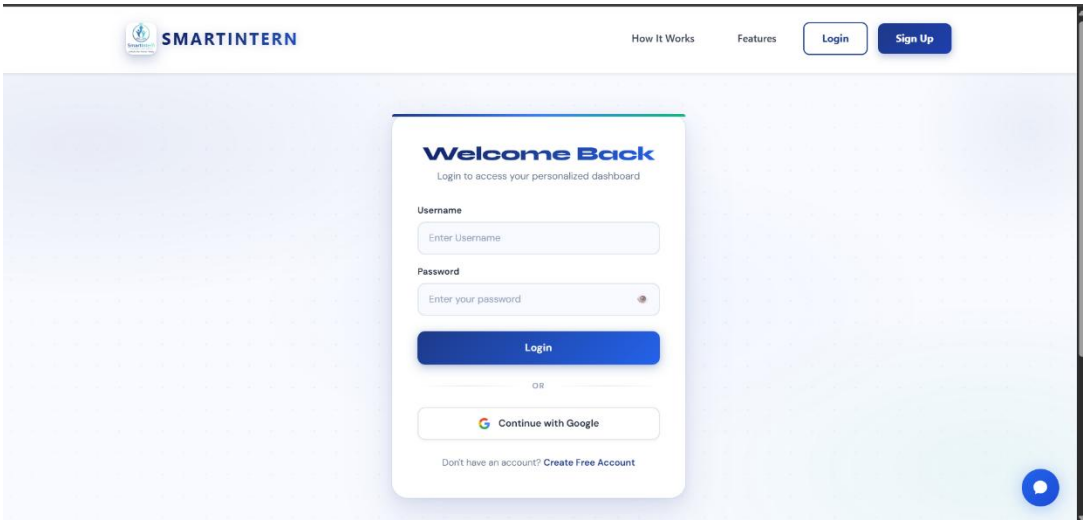


Figure 8.1: The login screen/Google Sign-in button

8.2 Profile Creation & Cold Start Mitigation

Once authenticated, if a user has no existing data, the backend middleware intercepts their navigation. This output shows the mandatory onboarding form where the user must input their technical skills (for AI matching) and demographic details (for heuristic matching) before the system allows access to the dashboard.

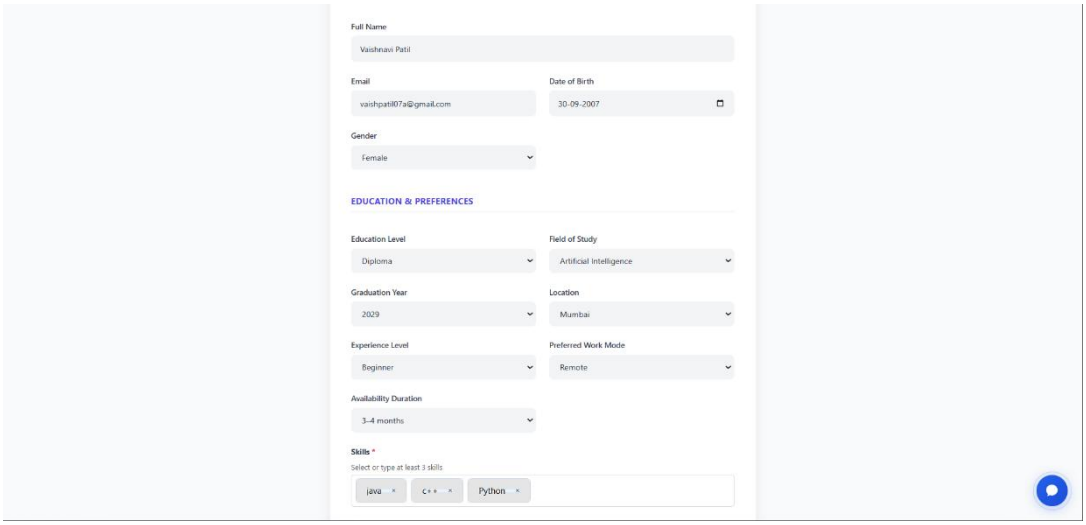


Figure 8.2: The profile form where you enter skills, category, gender, etc.

8.3 The SmartIntern Dashboard (Unified View)

This output displays the central hub of the application. It visualizes how the system curates complex data into an intuitive UI, preventing cognitive overload by separating AI internship matches from financial aid opportunities.

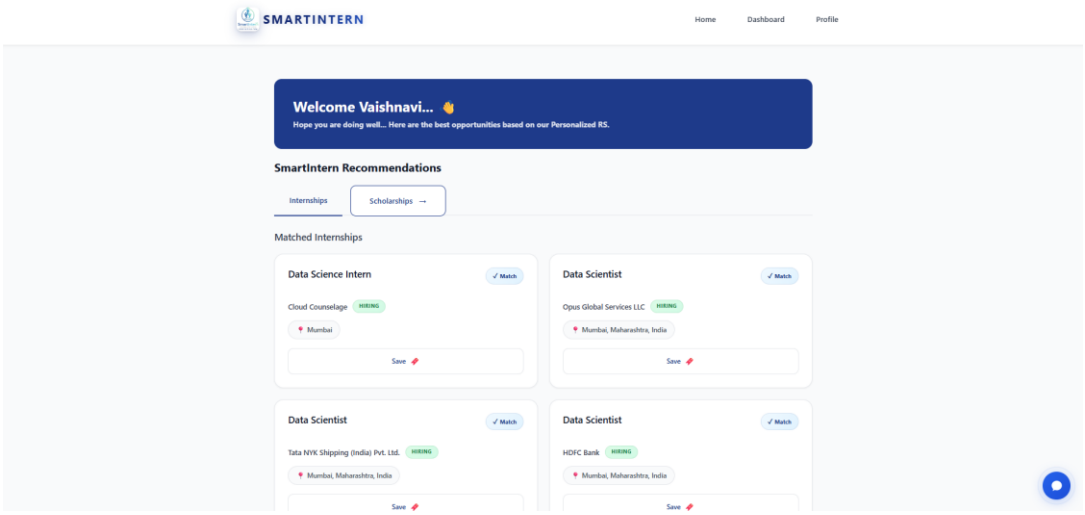


Figure 8.3: A wide shot of your main dashboard

8.4 AI-Driven Internship Recommendations

This output focuses on the results of the TF-IDF and Cosine Similarity engine. The system dynamically renders the top internship matches based on the mathematical alignment between the unstructured text in the user's profile and the employer's job description.

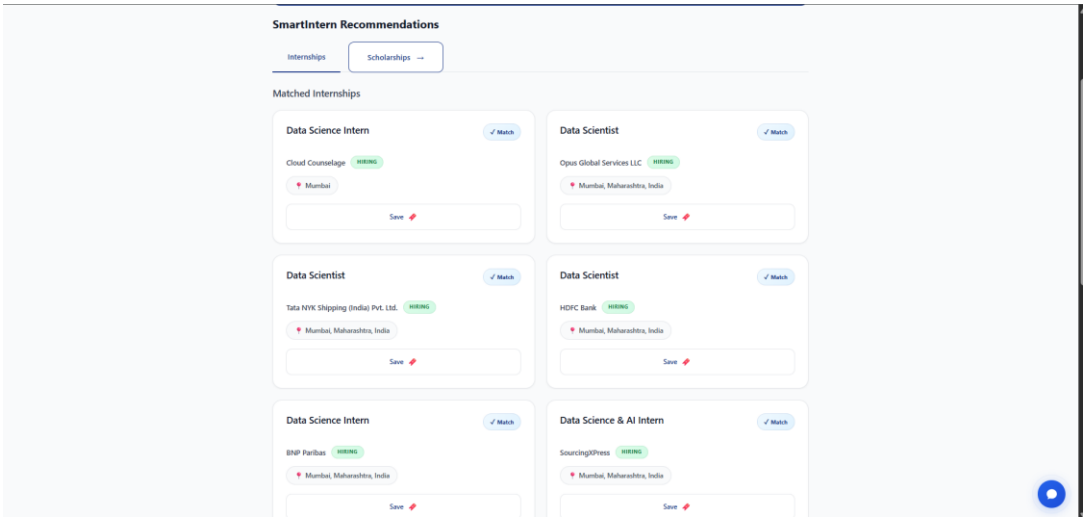


Figure 8.5: A close-up of the Internship cards/list showing the recommended jobs

8.5 Heuristic Scholarship Matching

This output highlights the secondary matching engine. It displays the scholarships the user is explicitly eligible for, generated by applying deterministic Regular Expressions (Regex) against the demographic data provided during the profile creation phase.

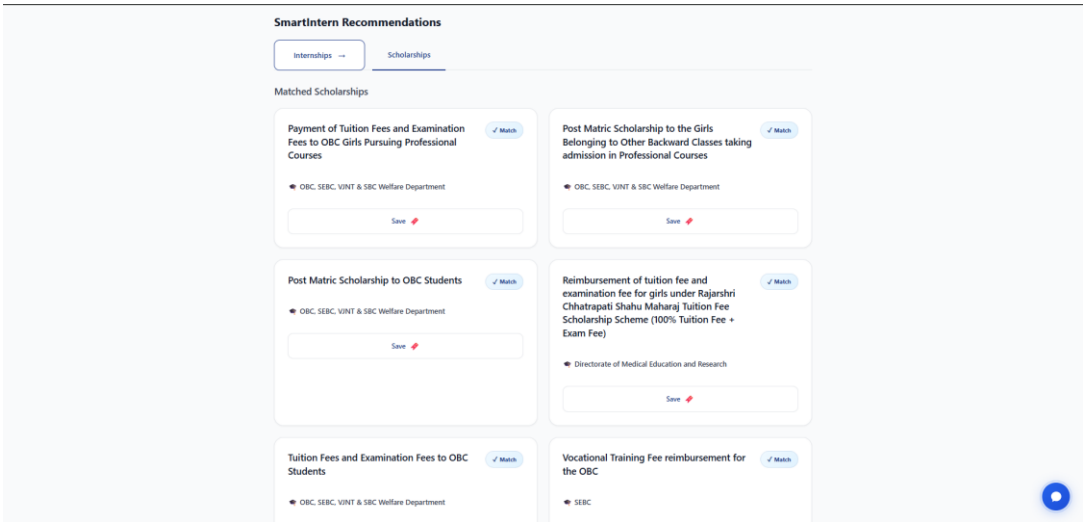


Figure 8.5: A close-up of the Scholarship cards/list

CHAPTER 9: CONCLUSION & FUTURE SCOPE

9.1 Conclusion

The transition from academia to the professional workforce is a critical juncture for college students, often hindered by fragmented, inefficient, and rigidly static discovery platforms. The objective of the SmartIntern project was to architect a centralized, intelligent ecosystem capable of resolving these systemic inefficiencies. Through the rigorous application of the Software Development Life Cycle, this objective was successfully met.

The SmartIntern platform completely shifts the paradigm of opportunity discovery from "manual keyword searching" to "automated semantic matching." By engineering a dual-engine architecture, the system successfully bridges the gap between career advancement and financial aid. The integration of Natural Language Processing (TF-IDF and Cosine Similarity) ensures that students are matched with internships based on the mathematical alignment of their skills, bypassing the limitations of literal string matching. Concurrently, the heuristic Regex engine ensures that complex, deterministic demographic criteria are strictly adhered to for scholarship eligibility.

Furthermore, the system’s architecture demonstrates a high level of resilience. By preemptively identifying the User Cold Start problem, the Flask routing was fortified with a strict middleware gatekeeper, ensuring the AI matrices never fail due to a lack of baseline data. Coupled with a stateless, serverless Supabase PostgreSQL backend, the final deployed application is highly scalable, secure, and capable of serving the complex needs of the collegiate demographic without performance degradation.

9.2 Future Scope

The SmartIntern architecture is built to scale and will evolve beyond its current core capabilities through four primary upgrades:

- **Advanced AI Matching:** The system will upgrade from frequency-based text matching (TF-IDF) to deep learning transformer models (like BERT) to understand the true semantic context of skills and job descriptions.
- **Behavior-Based Recommendations:** As user data grows, the platform will introduce Hybrid Collaborative Filtering. This will complement the current text-matching approach by recommending internships based on the application behavior of similar students.
- **Frictionless Onboarding:** Automated resume parsing via OCR and PDF-extraction tools will be implemented, allowing the system to automatically identify and populate a user's skills and education from an uploaded resume.
- **End-to-End Native Experience:** The platform will transition from a simple discovery hub into a comprehensive ecosystem featuring native application tracking, direct recruiter messaging, and in-app interview scheduling.

BIBLIOGRAPHY

[1] K. S. Varshith Reddy, et al., "Resume Analyzer and Job Recommendation System" (2025)

[2] Hongli Yuan and Alexander A. Hernandez, "User Cold Start Problem in Recommendation Systems: A Systematic Review" (2023).

[3] Gugasree S. and Dr. N. Saranya, "AI-Based Skill Matching and Job Recommendation Hub" (2026)

[4] Priyanka Singla and Vishal Verma, "An Intelligent Job Recommendation System based on Semantic Embeddings and Machine Learning" (2025)

[5] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[7] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, Vol. 12, pp. 2825-2830, 2011.

[8] Y. Kino, H. Kuroki, T. Machida, N. Furuya, and K. Takano, "Text analysis for job matching quality improvement," *Procedia Computer Science*, 112, 1523-1530, 2017.

[9] R. Liu, W. Rong, Y. Ouyang, and Z. Xiong, "A hierarchical similarity-based job recommendation service framework for university students," *Frontiers of Computer Science*, 11, 912-922, 2017.

[10] S. Yang, M. Korayem, K. AlJadda, T. Grainger, and S. Natarajan, "Combining content-based and collaborative filtering for job recommendation system: A cost-sensitive Statistical Relational Learning approach," *Knowledge-Based Systems*, 136, 37-45, 2017.

[11] G. M. Sridevi and S. K. Suganthi, "AI-based suitability measurement and prediction between job description and job seeker profiles," *International Journal of Information Management Data Insights*, 2(2), 100109, 2022.

[12] Y. C. Chou and H. Y. Yu, "Based on the application of AI technology in resume analysis and job recommendation," *2020 IEEE International Conference on Computational Electromagnetics (ICCEM)*, pp. 291-296, 2020.

[13] S. Guo, F. Alamudun, and T. Hammond, "RésuMatcher: A personalized résumé-job matching system," *Expert Systems with Applications*, 60, 169-182, 2016.

[14] T. Keim, "Extending the applicability of recommender systems: A multilayer framework for matching human resources," *2007 40th Annual Hawaii International Conference on System Sciences*, 2007.

[15] D. Mhamdi, R. Moulouki, M. Y. El Ghoumari, M. Azzouazi, and L. Moussaid, "Job recommendation based on job profile clustering and job seeker behavior," *Procedia Computer Science*, 175, 695-699, 2020.

[16] T. V. Yadalam, V. M. Gowda, V. S. Kumar, D. Girish, and M. Namratha, "Career recommendation systems using content-based filtering," *2020 5th International Conference on Communication and Electronics Systems*, pp. 660-665, 2020.

[17] L. Wu, Z. Qiu, Z. Zheng, H. Zhu, and E. Chen, "Exploring large language model for graph data understanding in online job recommendations," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, No. 8, pp. 9178-9186, 2024.

[18] X. Shi, Q. Wei, and G. Chen, "A bilateral heterogeneous graph model for interpretable job recommendation considering both reciprocity and competition," *Frontiers of Engineering Management*, 11(1), 128-142, 2024.

APPENDIX

Appendix A: Web Application Access & Deployment Specifications

This section outlines the accessibility requirements and cloud deployment architecture of the SmartIntern platform. Because the system was engineered using a modern, cloud-native Software-as-a-Service (SaaS) architecture, it completely eliminates the need for end-users to download, install, or locally configure any software.

A.1 Client-Side Prerequisites

To ensure a seamless and responsive user experience, the client device must meet the following baseline requirements to interact with the cloud servers:

- **Hardware:** Any internet-enabled device (Desktop, Laptop, Tablet, or Smartphone) capable of rendering modern HTML5 and CSS3.
- **Web Browser:** A modern, actively supported web browser such as Google Chrome, Mozilla Firefox, Apple Safari, or Microsoft Edge.
- **JavaScript Enablement:** The client's browser must have JavaScript execution enabled to allow the dynamic rendering of the AI recommendation carousels and asynchronous API fetching.
- **Authentication Prerequisite:** An active Google Account is strictly required to bypass the OAuth 2.0 gateway.

A.2 Accessing the Platform

Since the application operates in a completely serverless client-server paradigm, end-users can access the platform globally via the following steps:

1. **Network Connection:** Ensure the device is connected to a stable internet network.
2. **URL Navigation:** Open the web browser and navigate to the official SmartIntern production URL *(➔ Note: You can insert your actual hosted URL here, e.g., https://smartintern.onrender.com).*

3. **Secure Authentication:** Upon reaching the landing page, the user must click the "Login with Google" button. The system will securely redirect the user to Google's OAuth consent screen. Upon authorization, the user is instantly redirected back to their personalized SmartIntern dashboard.

A.3 Cloud Deployment Architecture (Technical Note)

For administrative and evaluation purposes, the SmartIntern platform is hosted entirely in the cloud, utilizing a decoupled architecture to ensure high availability and scalability:

- **The Backend (Application Logic):** The Python (Flask) backend, including the NLP preprocessing pipeline and Cosine Similarity matrices, is deployed on a scalable Platform-as-a-Service (PaaS) provider. This ensures the heavy mathematical computations are offloaded entirely from the user's local device to the cloud servers.
- **The Database Layer:** The database architecture is hosted on Supabase, utilizing a serverless PostgreSQL infrastructure. The web application communicates with this database statelessly via secure REST APIs, completely isolating the backend routing from the data storage layer.
- **Continuous Integration:** The automated web-scraping pipelines (for internship and scholarship aggregation) are executed serverlessly via GitHub Actions, pushing fresh data directly to the cloud database without requiring localized cron jobs.