Project Title: SkillLedger: Career Growth Platform

Video Demonstration Link: https://youtu.be/unPxHn7n2Qs

GitHub Repository Link: https://github.com/mrudulaeluri29/skillledger.git

1. Overview of the Application

SkillLedger is a comprehensive web-based career growth platform designed to address the critical challenge of credible skill verification and professional experience tracking in the modern job market. The application serves as a centralized ecosystem that connects students, mentors, companies, and academic administrators to create a transparent and verifiable record of internship experiences and professional skills.

The platform leverages a blockchain-inspired verification ledger using SHA-256 hash chaining to ensure tamper-proof documentation of student achievements. This creates an immutable trail of verified internships and skill endorsements that students can confidently showcase to potential employers. The application goes beyond simple record-keeping by integrating social networking features, allowing students to share their verified experiences on a public feed where peers can engage through likes and comments, fostering a collaborative learning community.

SkillLedger addresses several gaps in the current internship and early-career process:

- **Unverified credentials** – Resumes and LinkedIn profiles rarely include trusted third-party verification, making it hard for employers to trust claims about projects and skills.
- **Fragmented tracking** – Students track internships, certificates, and feedback across email, cloud folders, and social platforms, resulting in incomplete or lost records.
- **Limited mentor tools** – Mentors lack a structured workflow to review, rate, and track student internships over time.
- **Low visibility for institutions** – Departments have little aggregated data on which skills are growing, which industries students enter, or how effective programs are.
- **Inefficient recruiting** – Companies must spend extra effort validating candidate backgrounds because academic verification is not integrated with hiring tools.

The platform serves four distinct user roles, each with specific workflows and benefits:

Students: They are at the center of SkillLedger's ecosystem. They use the platform to:

- Register and create their professional profile
- Log internship experiences with detailed descriptions, start/end dates, and skills gained
- Upload supporting documents such as offer letters, certificates, and completion letters
- Submit internships for mentor verification
- Track verification status in real-time through their dashboard
- Build a comprehensive skill portfolio validated by industry professionals
- Apply to company-posted internship opportunities directly through the platform

- Share verified experiences on a public social feed to inspire peers
- Engage with peer content through likes and comments
- Export their verified profile for job applications and interviews

Use-Case Scenario: Sarah, a computer science student, completed a summer internship at a tech startup. She logs the experience in SkillLedger, uploading her offer letter and completion certificate. Her assigned academic mentor, Dr. Johnson, reviews her submission and verifies it with a 4/5 rating and constructive feedback. The verification is permanently recorded in the blockchain ledger. Sarah then shares this achievement on the public feed, inspiring her peers. When applying for full-time positions after graduation, she provides employers with her SkillLedger profile showing multiple verified internships, giving her a significant competitive advantage.

Mentors (Verification Authorities): Academic advisors, faculty members, or industry professionals who:

- Receive system login credentials from administrators
- View their assigned students and their pending verification requests
- Review internship submissions including descriptions, dates, and uploaded certificates
- Approve or reject verification requests with detailed feedback
- Provide ratings (1-5 scale) reflecting the quality and authenticity of experiences
- Track their verification history and student progress over time
- Maintain accountability through the immutable verification ledger

Companies (Recruiters and Hiring Managers): Organizations seeking qualified interns who:

- Register company profiles with business details and contact information
- Await administrator verification to ensure legitimacy
- Once verified, post internship opportunities with detailed requirements, duration, and application deadlines
- Review applications from interested students, viewing their verified profiles and skill portfolios
- Update application statuses (under review, interview scheduled, accepted, rejected)
- Access a talent pool of pre-verified candidates, reducing recruitment time and costs
- Build long-term relationships with academic institutions for continuous talent pipeline

Use-Case Scenario: TechCorp Inc. registers on SkillLedger and is verified by the platform administrator. The HR manager posts a "Data Science Intern - Summer 2025" position requiring Python, SQL, and machine learning skills. Twenty students apply, and the HR team can immediately see their verified internship history, mentor ratings, and documented skills. They shortlist five candidates whose verified experiences align with the role, significantly reducing screening time. TechCorp updates

application statuses in the system, keeping applicants informed throughout the process.

Administrators (Academic Institution Staff): University staff members or platform administrators:

- Manage the entire ecosystem with comprehensive oversight
- Create and manage mentor accounts, assigning them to students
- Verify company registrations to prevent fraudulent postings
- Access analytics dashboards showing:
    - Most in-demand skills among students and department-wise internship distribution
    - Mentor performance and verification statistics and company engagement metrics
- Manage the skills catalog, adding new relevant skills as industries evolve
- View students without assigned mentors and facilitate mentor-student matching
- Monitor user activity and ensure data integrity and resolve disputes if necessary

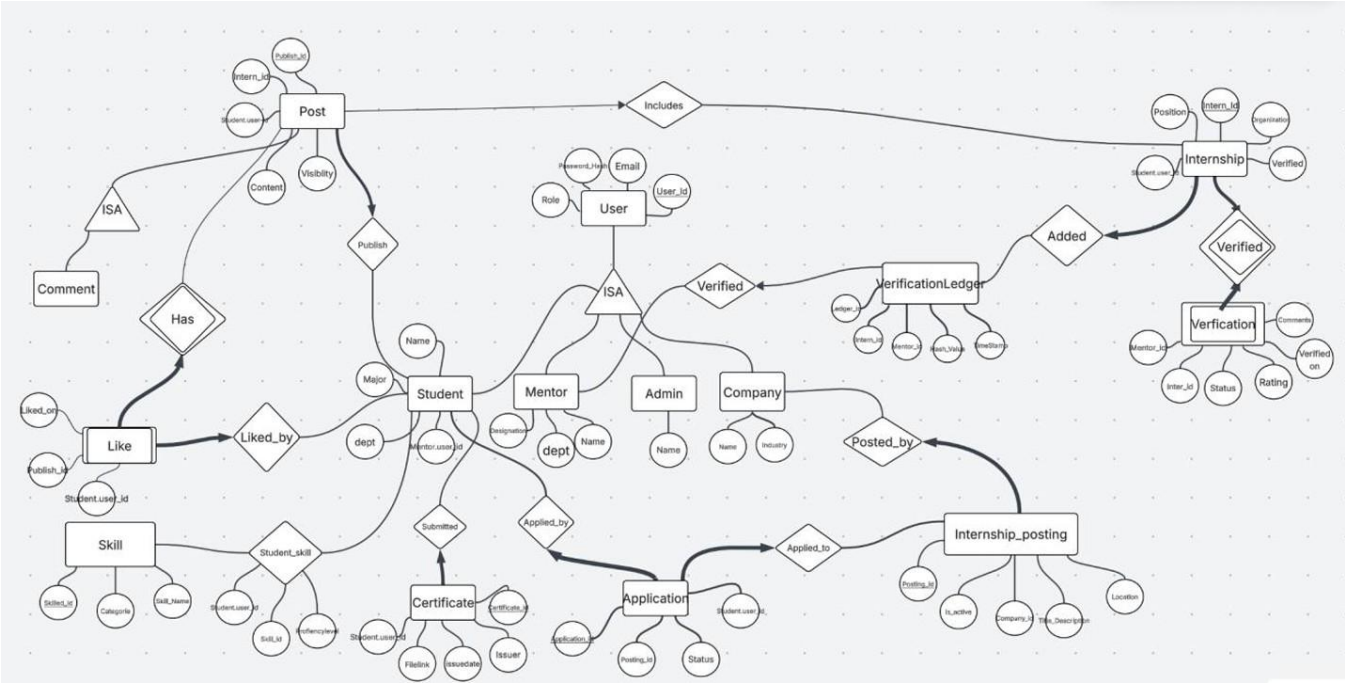
Use-Case Scenario: Dean Martinez, the administrator for the Computer Science department, logs into SkillLedger's admin dashboard. She reviews the analytics and notices that "Cloud Computing" and "Kubernetes" are trending skills with 40% year-over-year growth. She decides to advocate for a new cloud computing elective. She also sees that 15 new students haven't been assigned mentors yet, so she matches them with available faculty based on their declared majors. When a new company "StartupXYZ" registers, she verifies their business credentials before approving them to post internships, protecting students from potentially illegitimate opportunities.

SkillLedger’s power comes from how its roles connect in a single end-to-end workflow. Students and companies onboard to the system while admins verify companies and create mentor accounts. Students log internships and submit them for mentor review; once verified, the system writes an immutable ledger entry and marks the experience as shareable. Verified companies then post roles, review applicants’ verified profiles, update application statuses, and students receive status notifications. As students share achievements on the public feed and interact with peers, all activity feeds into admin analytics dashboards, supporting curriculum improvements and stronger industry partnerships.

2. Database Design Summary

The SkillLedger database is implemented in PostgreSQL under the `skillledger` schema, designed to support a comprehensive career growth platform with robust data integrity and relationships. The schema consists of 15 interconnected tables organized around four primary user types (Student, Mentor, Administrator, Company) and their associated activities including internship management, skill tracking, verification workflows, job applications, and social interactions.

ER diagram:



Key Entities, Attributes, and Constraints

Student Attributes:

student_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	year (INT, NOT NULL, CHECK(year BETWEEN 1 AND 6))
name (VARCHAR(100), NOT NULL)	department (VARCHAR(100), NOT NULL)
email (VARCHAR(255), UNIQUE, NOT NULL)	role (VARCHAR(20), NOT NULL, CHECK(role = 'STUDENT'))
password_hash (VARCHAR(255), NOT NULL)	mentor_id (Foreign Key → Mentor.mentor_id, NULL)
major (VARCHAR(100), NOT NULL)	

Constraints:

student_id is unique and auto generated and email is also unique	mentor_id is optional so a student can exist without a mentor

password_hash stores a secure salted hash	on delete of Mentor either SET NULL on Student.mentor_id or RESTRICT deletion

Mentor Attributes:

mentor_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	name (VARCHAR(100), NOT NULL)
email (VARCHAR(255), UNIQUE, NOT NULL)	password_hash (VARCHAR(255), NOT NULL)
department (VARCHAR(100), NOT NULL)	designation (VARCHAR(100), NULL)
role(VARCHAR(20),NOT NULL,CHECK(role = 'MENTOR'))	

Constraints:

mentor_id is unique and auto generated and email is unique	password_hash stores a secure salted hash

Administrator Attributes:

admin_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	name (VARCHAR(100), NOT NULL)
email (VARCHAR(255), UNIQUE, NOT NULL)	password_hash (VARCHAR(255), NOT NULL)
role (VARCHAR(20), NOT NULL, CHECK(role = 'ADMIN'))	

Constraints:

admin_id is unique and auto generated and email is unique	password_hash stores a secure salted hash

Skill Attributes:

skill_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	skill_name (VARCHAR(100), NOT NULL, UNIQUE)
category (VARCHAR(50), NOT NULL)	

StudentSkill Attributes:

student_id (Foreign Key → Student.student_id, NOT NULL)	skill_id (Foreign Key → Skill.skill_id, NOT NULL)
proficiency_level (INT, NOT NULL, CHECK(proficiency_level BETWEEN 1 AND 5)	

Constraints:

Primary Key is composite (student_id, skill_id)	on delete of Student cascade delete StudentSkill rows

Internship Attributes:

internship_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	student_id (Foreign Key → Student.student_id, NOT NULL)
organization (VARCHAR(150), NOT NULL)	position (VARCHAR(150), NOT NULL)
start_date (DATE, NOT NULL)	end_date (DATE, NOT NULL, CHECK(end_date >= start_date)
verified (BOOLEAN, NOT NULL, DEFAULT FALSE)	status (VARCHAR(20), NOT NULL, CHECK(status IN ('Draft', 'Submitted', 'Verified', 'Rejected')))
description (TEXT, NULL)	

Constraints:

status and verified are synchronized from Verification by triggers or stored procedures	verified becomes TRUE only when the associated Verification.status is Approved
each internship belongs to one student	

Verification Attributes:

verification_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	internship_id (Foreign Key → Internship.internship_id, NOT NULL, UNIQUE)
mentor_id (Foreign Key → Mentor.mentor_id, NOT NULL)	status (VARCHAR(20), NOT NULL, CHECK(status IN ('Pending', 'Approved', 'Rejected')))
comments (TEXT, NULL)	verified_on (TIMESTAMP, NULL)
rating (INT, NULL, CHECK(rating BETWEEN 1 AND 5))	

Constraints:

one to one with Internship enforced by UNIQUE on Verification.internship_id	business rule trigger updates Internship.status and Internship.verified based on Verification.status
if status is Approved or Rejected then verified_on must be set	

Certificate Attributes:

certificate_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	Student.user_id (Foreign Key, NOT NULL)
file_link (VARCHAR(500), NOT NULL)	issue_date (DATE, NOT NULL)

certificate_type (VARCHAR(50), NULL)	issuer (VARCHAR(150), NULL)

Constraints:

an internship may have zero or many certificates	on delete of Internship cascade delete Certificates

VerificationLedger Attributes:

ledger_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	action (VARCHAR(50), NOT NULL, CHECK(action IN ('APPROVE', 'REJECT', 'CERT_ATTACH')))
internship_id (Foreign Key → Internship.internship_id, NOT NULL)	mentor_id (Foreign Key → Mentor.mentor_id, NOT NULL)
hash_value (CHAR(64), NOT NULL)	previous_hash (CHAR(64), NULL)
timestamp (TIMESTAMP, NOT NULL)	

Constraints:

hash_value is computed from immutable fields such as internship_id, mentor_id, action, timestamp and certificate link if present	previous_hash points to the prior row's hash_value to form a chain
append only log so rows are not updated after insert	

Company Attributes:

company_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	verified_status (VARCHAR(20), NOT NULL, CHECK(verified_status IN ('Pending', 'Verified', 'Suspended')))
name (VARCHAR(150), NOT NULL, UNIQUE)	email (VARCHAR(255), UNIQUE, NOT NULL)
password_hash (VARCHAR(255), NOT NULL)	industry (VARCHAR(100), NULL)

Constraints:

only companies with verified_status = 'Verified' can create active postings	name and email are unique

InternshipPosting Attributes:

posting_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	company_id (Foreign Key → Company.company_id, NOT NULL)
title (VARCHAR(150), NOT NULL)	description (TEXT, NOT NULL)
location (VARCHAR(150), NULL)	duration (VARCHAR(100), NULL)

application_deadline (DATE, NULL)	is_active (BOOLEAN, NOT NULL, DEFAULT TRUE)

Constraints:

business rule prevents is_active = TRUE unless Company.verified_status = 'Verified'	a company can have many postings

Application Attributes:

application_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	status (VARCHAR(20), NOT NULL, CHECK(status IN ('Applied', 'UnderReview', 'Interview', 'Offer', 'Rejected', 'Withdrawn')))
posting_id (Foreign Key → InternshipPosting.posting_id, NOT NULL)	student_id (Foreign Key → Student.student_id, NOT NULL)
applied_on (TIMESTAMP, NOT NULL)	

Constraints:

unique constraint on (posting_id, student_id) prevents duplicate applications by the same student	on delete of InternshipPosting cascade delete Applications
on delete of Student cascade delete Applications	

Post Attributes:

post_id (Primary Key, INT, GENERATED ALWAYS AS IDENTITY)	internship_id (Foreign Key → Internship.internship_id, NOT NULL)
visibility (VARCHAR(20), NOT NULL, CHECK(visibility IN ('Public', 'Campus', 'Private')))	student_id (Foreign Key → Student.student_id, NOT NULL)
content (TEXT, NOT NULL)	created_at (TIMESTAMP, NOT NULL)

Constraints:

a database trigger on insert or update checks that the referenced Internship has verified = TRUE	a student can create many posts and each post references one internship owned by the same student

Relationships:

- Student to Mentor is optional many to one:

A student may have zero or one mentor. A mentor can supervise many students.	Foreign key Student.mentor_id references Mentor.mentor_id. Deletion behavior is either RESTRICT or SET NULL.

- Student to Internship is one to many:

A student can have many internships. Foreign key Internship.student_id references Student.student_id.	On delete of Student cascade related internships and their dependent rows.

- Internship to Verification is one to one:

Each internship has at most one verification. Enforced by UNIQUE on Verification.internship_id.	Triggers synchronize Internship.status and Internship.verified with Verification.status.

- Mentor to Verification is one to many: A mentor can verify many internships. Foreign key Verification.mentor_id references Mentor.mentor_id.
- Internship to Certificate is one to many: An internship can have multiple certificates. Foreign key Certificate.internship_id references Internship.internship_id.
- Internship to VerificationLedger is one to many: Each verification action produces a ledger entry. Foreign keys to Internship and Mentor.
- Company to InternshipPosting is one to many: A company can create many postings. Foreign key InternshipPosting.company_id references Company.company_id.
- InternshipPosting to Application is one to many and Student to Application is one to many:

A posting receives many applications. A student can submit many applications.	Unique constraint on (posting_id, student_id) prevents duplicates.

- Internship to Post is one to many and Student to Post is one to many: Only verified internships may be posted. Enforced by a trigger that checks Internship.verified = TRUE.
- Post to Comment is one to many and Student to Comment is one to many: Comments belong to a post and are authored by students.
- Post to Like is one to many and Student to Like is one to many with a composite pk:

The database architecture emphasizes data integrity through constraints, referential integrity through foreign keys,& business logic enforcement through triggers.	A like is uniquely identified by the pair (post_id, student_id).

3. Technology Stack

Backend Components

- Flask 3.0.0 - Python web framework providing routing, request handling, and template rendering through Jinja2. The application uses a Blueprint architecture with 6 modules (auth, student, mentor, admin, company, feed) for modular organization.
- Werkzeug 3.0.1 - WSGI toolkit handling password hashing using PBKDF2-SHA256 for secure credential storage.
- Flask-Login 0.6.3 - User session management with polymorphic authentication supporting four user roles (Student, Mentor, Administrator, Company).

Database System

- PostgreSQL 12+ - Relational database (skillledger_db) with the skillledger schema containing 15 tables. Utilizes triggers for status synchronization, CHECK constraints for validation, and foreign keys for referential integrity.
- Flask-SQLAlchemy 3.1.1 - ORM layer mapping Python classes to database tables, providing parameterized queries for SQL injection prevention.
- psycopg2-binary 2.9.9 - PostgreSQL db adapter enabling Python-PostgreSQL connectivity.

Frontend Components

- Bootstrap 5.3.0 - CSS framework (CDN-hosted) providing responsive UI components including navigation bars, cards, forms, tables, modals, and alerts.
- HTML5/CSS3 - Semantic markup with custom styling for status badges and responsive layouts.
- JavaScript (ES6) - Client-side scripting for form validation and confirmation dialogs.

APIs and Libraries

- python-dotenv 1.0.0 - Environment variable management for database credentials, secret keys.
- email-validator 2.1.0 - Email address validation for user registration.
- hashlib (Python built-in) - SHA-256 hashing for blockchain-inspired VerificationLedger
- datetime (Python built-in) - Date and time handling for internship records and timestamps.

Development Tools

- Python 3.8+ - Core programming language
- Git - Version control
- pgAdmin 4 - PostgreSQL database management
- Flask development server - Local testing environment

4. Setup Instructions

Prerequisites

- Python 3.8 or higher, PostgreSQL 12+, Git and pip

Steps:

- Clone the github Repository
- Install Python Dependencies: pip install -r requirements.txt
- Open pgAdmin or psql terminal and execute: CREATE DATABASE skillledger_db;
- Run the DDL script to create the schema and all 15 tables: psql -U postgres -d skillledger_db -f skillledger_phase2_ddl.sql
- You can find the database backup.sql file in the database files folder.
- Populate the database with test data: psql -U postgres -d skillledger_db -f seed_data.sql
- Create or verify the .env file in the project root with your database credentials: DATABASE_URL=postgresql://postgres:YOUR_PASSWORD@localhost:5432/skillledger_db
  SECRET_KEY=your-secret-key-here
  FLASK_ENV=development
- Start the Flask development server: python app.py

Expected output: The terminal should display the following output:

* Serving Flask app 'app'

* Debug mode: on

* Running on http://127.0.0.1:5000

Press CTRL+C to quit

Open your web browser and navigate to http://127.0.0.1:5000

You should see the home page: Displays welcome screen with login/register options.

Default Test Accounts (if seed data was loaded):

Student: student@asu.edu, password123

Mentor:mentor@asu.edu, password123

Admin: admin@skillledger.com, admin123

Company: company@example.com, password123

Administrators must be created directly in the database or through SQL inserts as they cannot self-register. Mentors can be created directly in the database or through administrators dashboard.

5. Feature List

Student Features

- Add Internship Record (INSERT): Students can log new internship experiences by providing organization name, position, start/end dates, and description. They can upload supporting certificates (PDF/image files) with links stored in the database. Status is initially set to 'Draft' until submitted for verification.
- View Internship Records (SELECT): Students access a dashboard displaying all their internships with status indicators (Draft, Submitted, Verified, Rejected). They can view detailed information including mentor feedback, ratings (1-5 scale), and verification timestamps.
- Update Internship Details (UPDATE): Students can edit internship records that are in 'Draft' status, modifying organization, position, dates, or description before submission. Once submitted, records become read-only to maintain verification integrity.
- Delete Internship (DELETE): Students can remove internship records in 'Draft' status. Cascade deletion automatically removes associated certificates. Verified internships cannot be deleted to preserve the verification ledger integrity.
- Apply to Job Postings (INSERT): Students browse active internship postings from verified companies and submit applications. A unique constraint prevents duplicate applications to the same posting. Application status updates (Applied → Under Review → Interview → Offer/Rejected) are tracked.
- Manage Skills Portfolio (INSERT/UPDATE/DELETE):Students add skills from the centralized catalog with proficiency levels (1-5). They can update proficiency ratings or remove skills as their expertise evolves.
- Share Verified Experiences (INSERT - Social Feed): Students create posts on the public feed showcasing verified internships. Only internships with verified = TRUE can be posted (enforced by database trigger). Posts include content, visibility settings (Public/Campus/Private), and timestamps.
- Engage with Peer Content (INSERT/DELETE): Students like posts (composite primary key prevents duplicate likes) and comment on peer experiences. Comments support threaded discussions about internship insights.

Mentor Features

- View Assigned Students (SELECT with JOIN): Mentors access a list of students assigned to them (foreign key relationship Student.mentor_id), viewing student profiles.
- Review Verification Requests (SELECT with JOIN): Mentors query pending verifications for

their assigned students, joining Internship, Verification, and Certificate tables to review complete submission packages including uploaded documents.

- Approve/Reject Internships (UPDATE + INSERT): Mentors update Verification.status to 'Approved' or 'Rejected', provide ratings (1-5), and add feedback comments. This triggers: Automatic update of Internship.status and Internship.verified via database trigger. Creation of immutable VerificationLedger entry with SHA-256 hash chain linking to previous entry
- Track Verification History (SELECT): Mentors view their complete verification history with timestamps, ratings, and outcomes for accountability and performance tracking.

Administrator Features

- User Management (INSERT/UPDATE/DELETE): Admins create mentor accounts, assign mentors to students (updating Student.mentor_id), and manage user statuses. They view students without assigned mentors using WHERE mentor_id IS NULL queries.
- Company Verification (UPDATE): Admins review company registrations and update Company.verified_status from 'Pending' to 'Verified' or 'Suspended'. Only verified companies can create active postings (trigger-enforced).
- Analytics Dashboard (SELECT with Aggregations)
    - Admins access data insights through complex queries: Top Skills: SELECT skill_name, COUNT(*) FROM StudentSkill GROUP BY skill_id ORDER BY COUNT DESC
    - Department Distribution: Internship counts grouped by Student.department
    - Mentor Performance: Average ratings and verification counts per mentor
    - Company Engagement: Application counts per posting
- Skills Catalog Management (INSERT/UPDATE/DELETE): Admins add new skills to the central catalog with categories (Technical, Soft Skills, Languages), update skill names, or archive obsolete skills (cascade impacts StudentSkill entries).

Company Features

- Create Job Postings (INSERT): Verified companies create internship postings with title, description, location, duration, and application deadlines. Database trigger prevents is_active = TRUE unless Company.verified_status = 'Verified'.
- View Applications (SELECT with JOIN): Companies query applications for their postings, joining with Student, StudentSkill, and Internship tables to review applicant profiles, verified experiences, and skill portfolios.
- Update Application Status (UPDATE): Companies modify Application.status through workflow stages (Applied → UnderReview → Interview → Offer/Rejected), keeping applicants informed of their progress.
- Manage Postings (UPDATE/DELETE): Companies can edit posting details, toggle is_active

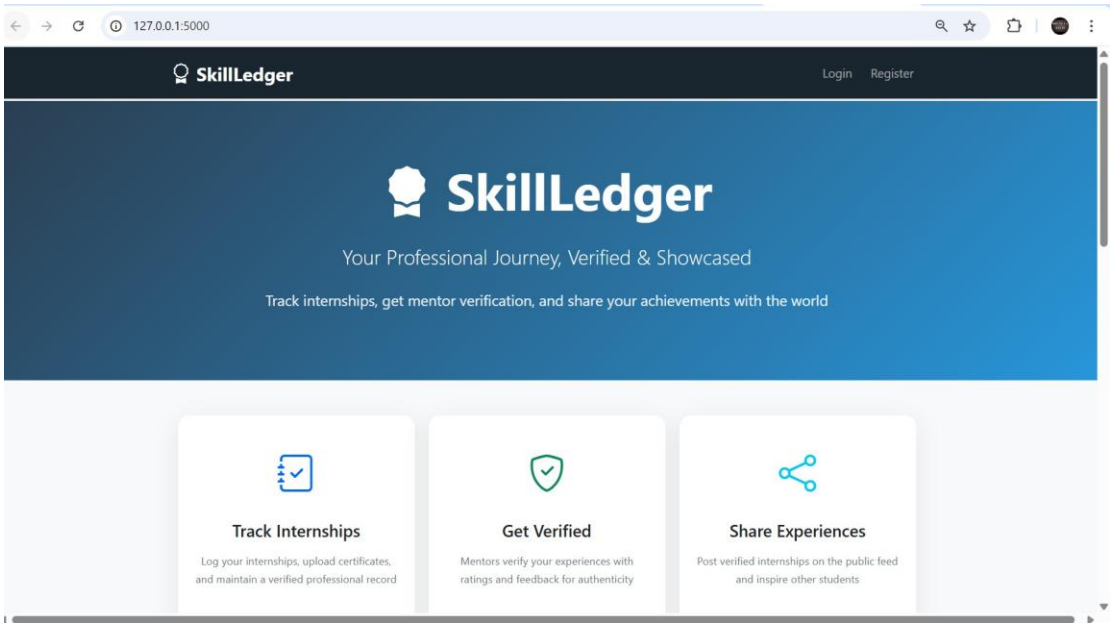
status to close applications, or delete postings (cascade deletes associated applications).

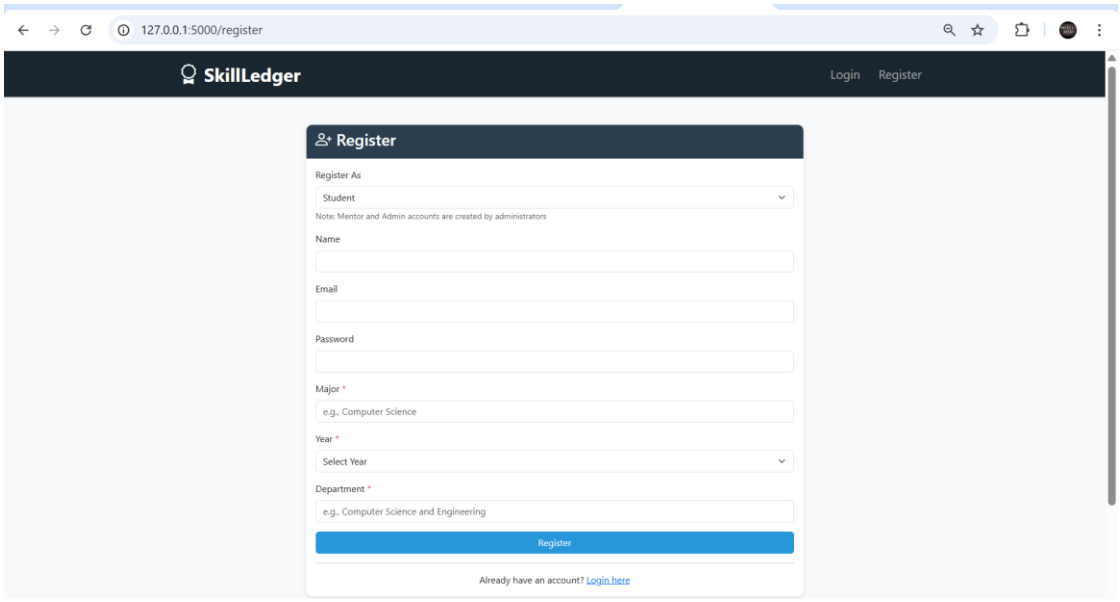
Special Features

- Blockchain-Inspired Verification Ledger (INSERT - Append-Only): Every verification action creates a VerificationLedger entry with:
    - hash_value: SHA-256 hash of (internship_id + mentor_id + action + timestamp + previous_hash)
    - previous_hash: Links to prior entry creating tamper-proof chain
- Immutable records (no UPDATE/DELETE allowed) ensure audit trail integrity
- polymorphic Authentication (SELECT with Dynamic Routing): Login system identifies user role and loads appropriate model (Student, Mentor, Administrator, Company) using composite ID format (student_34, mentor_12). Flask-Login's user_loader dynamically queries the correct table.
- Trigger-Based Status Synchronization (Automated UPDATE): Database triggers maintain consistency:
    - If Verification.status changes, Internship.status & Internship.verified auto-update
    - Posts require Internship.verified = TRUE (trigger blocks invalid posts)
    - Active postings require Company.verified_status = 'Verified' (trigger enforcement)
- Cascade Deletion Protection (Automated DELETE): Foreign key cascade behaviors ensure data consistency:
    - Deleting internships removes certificates, posts, and ledger entries
    - Deleting posts removes comments and likes
    - Deleting postings removes applications
    - Prevents orphaned records while maintaining referential integrity

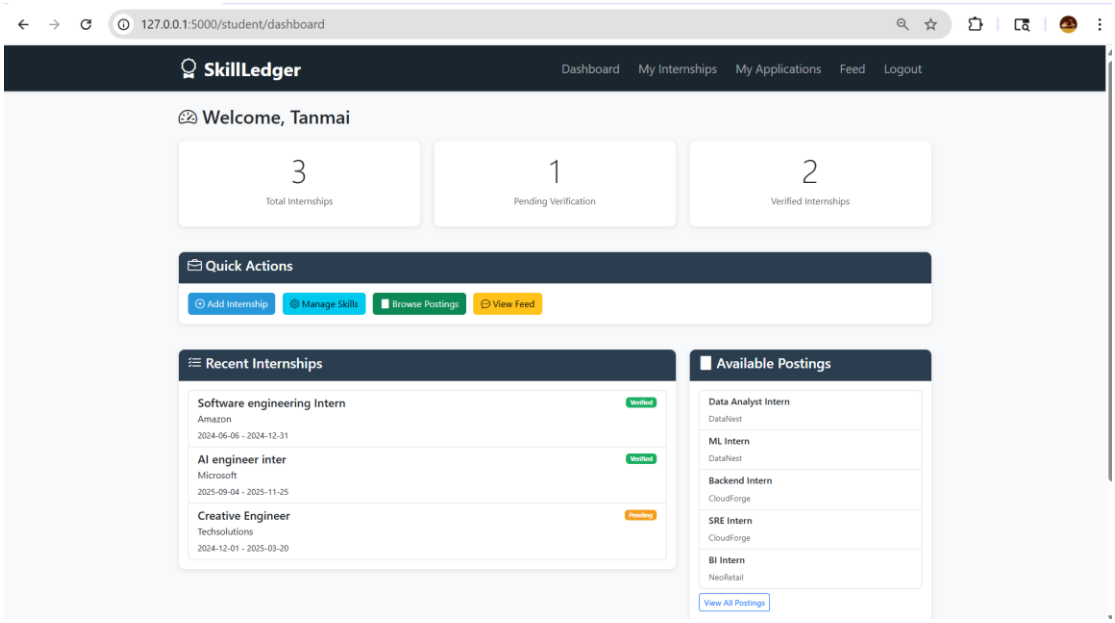
6. Screenshots

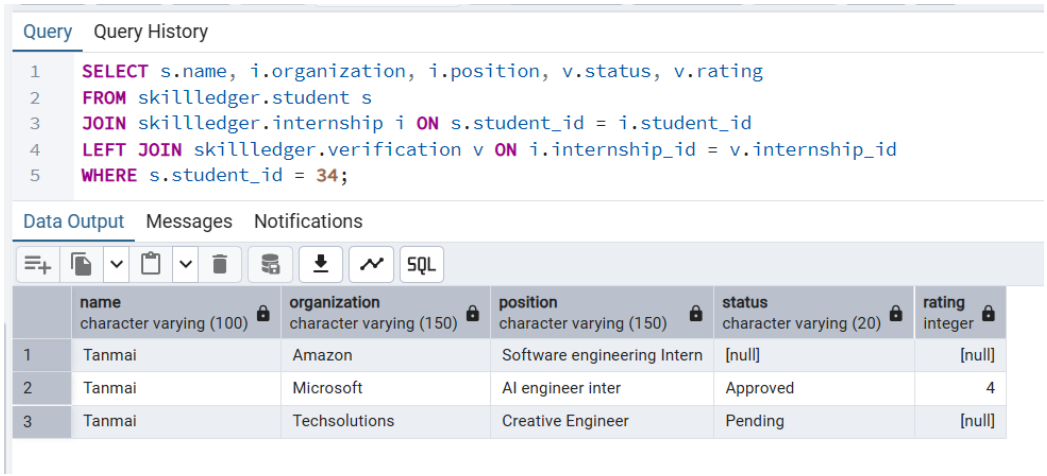
Home screen showing navigation and user authentication portal. Each registration directly updates either the company or student table respectively.





Student dashboard displaying internships with verification status and available posting by companies. In the verification table, once the mentor approves an internship, the verification becomes true.





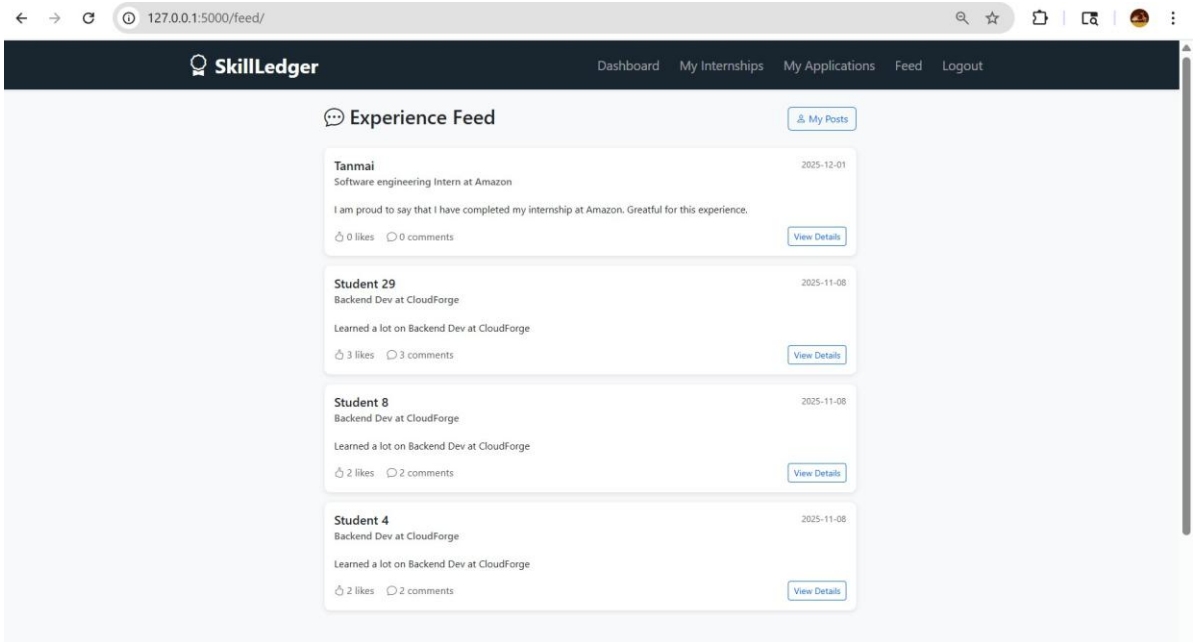
Query | Query History

```
1  SELECT s.name, i.organization, i.position, v.status, v.rating
2  FROM skillledger.student s
3  JOIN skillledger.internship i ON s.student_id = i.student_id
4  LEFT JOIN skillledger.verification v ON i.internship_id = v.internship_id
5  WHERE s.student_id = 34;
```

Data Output | Messages | Notifications

	name character varying (100)	organization character varying (150)	position character varying (150)	status character varying (20)	rating integer
1	Tanmai	Amazon	Software engineering Intern	[null]	[null]
2	Tanmai	Microsoft	AI engineer inter	Approved	4
3	Tanmai	Techsolutions	Creative Engineer	Pending	[null]

Social feed displaying verified internship posts with engagement features (likes, comments) and visibility indicators



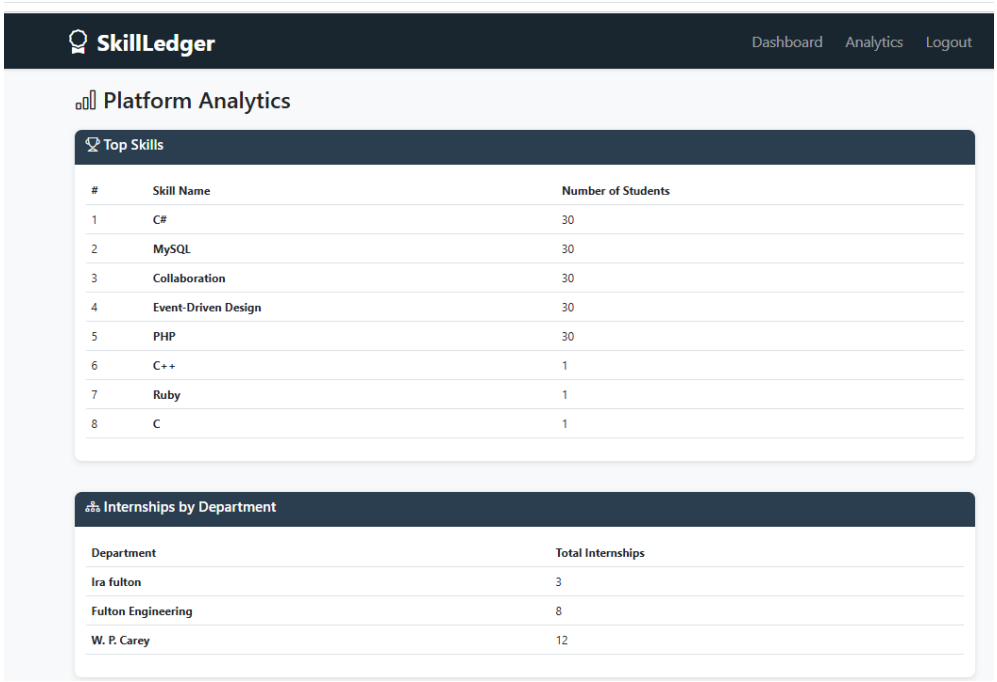
```sql
SELECT p.post_id, s.name AS student_name,i.position, i.organization, p.content,COUNT(DISTINCT l.student_id) AS like_count,
  COUNT(DISTINCT c.comment_id) AS comments
FROM skillledger.post p
JOIN skillledger.student s ON p.student_id = s.student_id
JOIN skillledger.internship i ON p.internship_id = i.internship_id
LEFT JOIN skillledger.like l ON p.post_id = l.post_id
LEFT JOIN skillledger.comment c ON p.post_id = c.post_id
WHERE p.visibility = 'Public'
GROUP BY p.post_id, s.name, i.position, i.organization, p.content, p.created_at
ORDER BY p.created_at DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

	post_id integer	student_name character varying (100)	position character varying (150)	organization character varying (150)	content text	like_count bigint	comments bigint
1	9	Tanmai	Software engineering Intern	Amazon	I am proud to say that I have completed my internship at Amazon. Greatful for this experienc…	0	0
2	4	Student 29	Backend Dev	CloudForge	Learned a lot on Backend Dev at CloudForge	3	3
3	5	Student 8	Backend Dev	CloudForge	Learned a lot on Backend Dev at CloudForge	2	2
4	6	Student 4	Backend Dev	CloudForge	Learned a lot on Backend Dev at CloudForge	2	2

Admin dashboard which has the analytics:




```sql
SELECT
    'Internships by Department' AS metric_type,
    s.department AS category,
    COUNT(i.internship_id) AS count_value
FROM skillledger.student s
LEFT JOIN skillledger.internship i ON s.student_id = i.student_id
GROUP BY s.department
HAVING COUNT(i.internship_id) > 0
ORDER BY COUNT(i.internship_id) DESC;
```

Data Output | Messages | Notifications

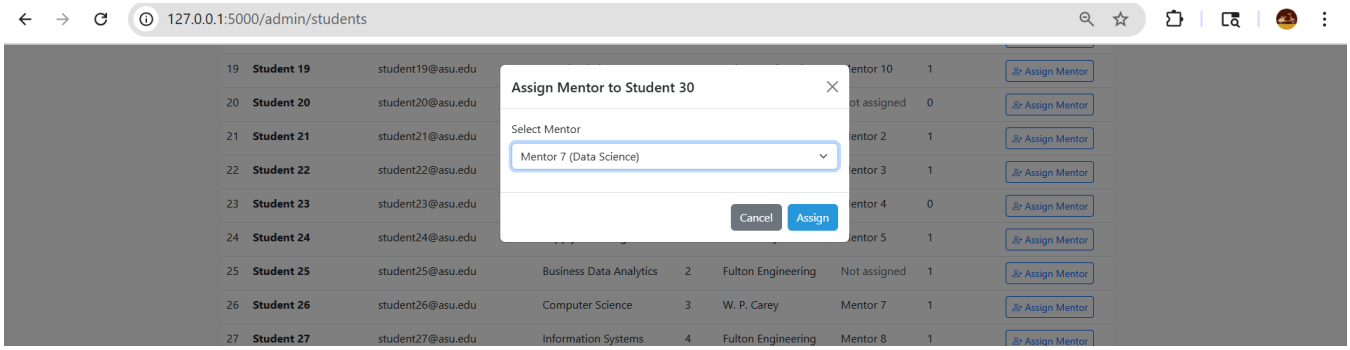
	metric_type text	category character varying (100)	count_value bigint
1	Internships by Department	W. P. Carey	12
2	Internships by Department	Fulton Engineering	8
3	Internships by Department	Ira fulton	3

```sql
SELECT
    sk.skill_id,
    sk.skill_name AS "Skill Name",
    COUNT(ss.student_id) AS "Number of Students"
FROM skillledger.skill sk
LEFT JOIN skillledger.student_skill ss ON sk.skill_id = ss.skill_id
GROUP BY sk.skill_id, sk.skill_name
ORDER BY COUNT(ss.student_id) DESC, sk.skill_name
LIMIT 10;
```

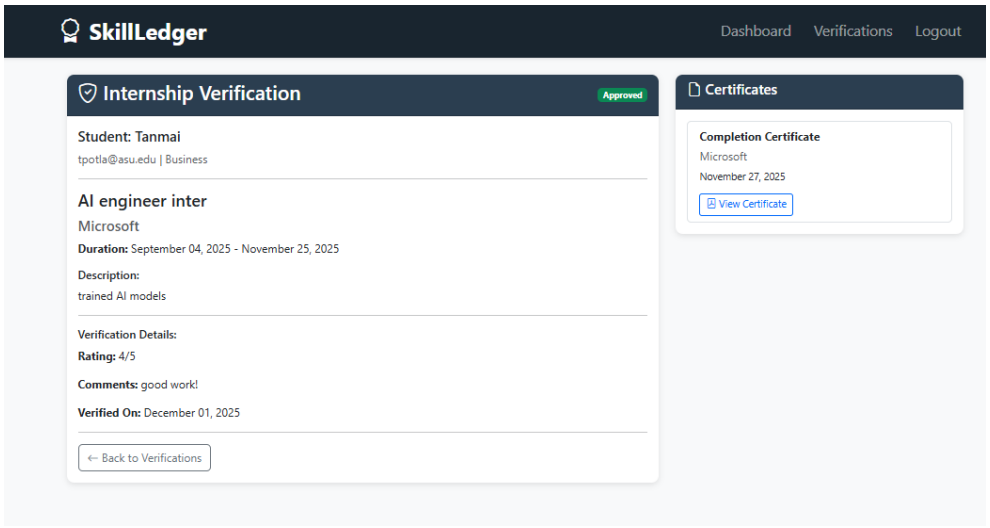
Data Output | Messages | Notifications

	skill_id [PK] integer	Skill Name character varying (100)	Number of Students bigint
1	7	C#	30
2	77	Collaboration	30
3	72	Event-Driven Design	30
4	43	MySQL	30
5	14	PHP	30
6	5	C	1
7	6	C++	1
8	15	Ruby	1

Admin can assign mentors to students, db updates mentor assignment from null to mentor_id



Mentor verifies the internship posted by students and can also rate it and provide a comment:



VerificationLedger demonstrating blockchain-inspired SHA-256 hash chaining - previous_hash links to prior entry's hash_value creating immutable audit trail. Append-only table (enforced by database rules) ensures tamper-proof verification history.

Query | Query History

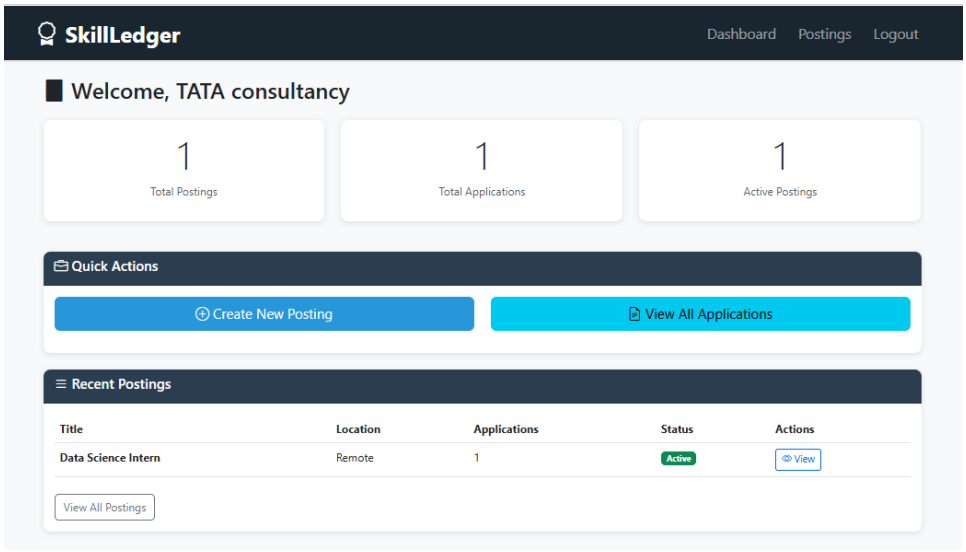
```
SELECT vl.ledger_id AS "ID",i.organization AS "Company", i.position AS "Position",m.name AS "Mentor",vl.action AS "Action",
LEFT(vl.hash_value, 16) || '...' AS "Hash", CASE
        WHEN vl.previous_hash IS NULL THEN 'Genesis (First)'
        ELSE LEFT(vl.previous_hash, 16) || '...'
    END AS "Previous Hash",
    TO_CHAR(vl.timestamp, 'YYYY-MM-DD HH24:MI') AS "Timestamp"
FROM skillledger.verification_ledger vl
JOIN skillledger.internship i ON vl.internship_id = i.internship_id
JOIN skillledger.mentor m ON vl.mentor_id = m.mentor_id
ORDER BY vl.ledger_id LIMIT 6;
```

Data Output | Messages | Notifications

Showing rows: 1 to 6 | Page No: 1

	ID integer	Company character varying (150)	Position character varying (150)	Mentor character varying (100)	Action character varying (50)	Hash text	Previous Hash text	Timestamp text
1	1	CloudForge	Data Analyst	Mentor 8	APPROVE	43a738cd55ac7c87...	Genesis (First)	2025-11-08 21:04
2	2	CloudForge	Data Analyst	Mentor 8	APPROVE	4eadc91558971115...	Genesis (First)	2025-11-08 21:04
3	3	CloudForge	Backend Dev	Mentor 7	APPROVE	22eccd9e810eb525...	Genesis (First)	2025-11-08 21:04
4	4	CloudForge	Backend Dev	Mentor 3	APPROVE	f559f47f371b8bef...	Genesis (First)	2025-11-08 21:04
5	5	CloudForge	Backend Dev	Mentor 6	APPROVE	d0f8085f8e89320b...	Genesis (First)	2025-11-08 21:04
6	6	CloudForge	Backend Dev	Mentor 2	APPROVE	ad70b44ec0be0ede...	Genesis (First)	2025-11-08 21:04

Company dashboard: Here, they can create new posts and view the applicants to their posts.



Query | Query History

```
SELECT c.company_id, c.name AS company_name, c.verified_status, COUNT(DISTINCT ip.posting_id) AS total_postings,
    COUNT(DISTINCT CASE WHEN ip.is_active = TRUE THEN ip.posting_id END) AS active_postings,
    COUNT(DISTINCT a.application_id) AS total_applications
FROM skillledger.company c
LEFT JOIN skillledger.internship_posting ip ON c.company_id = ip.company_id
LEFT JOIN skillledger.application a ON ip.posting_id = a.posting_id
WHERE c.company_id = 10
GROUP BY c.company_id, c.name, c.verified_status;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page

	company_id [PK] integer	company_name character varying (150)	verified_status character varying (20)	total_postings bigint	active_postings bigint	total_applications bigint
1	10	TATA consultancy	Verified	1	1	1

References:

Flask Documentation. (2023). *Flask web development framework* https://flask.palletsprojects.c