NoteOrbit©

by LeafCore Labs

Advanced University Ecosystem & Governance Protocol

TECHNICAL DOCUMENTATION V2.2

Executive Overview

NoteOrbit is an enterprise-grade University Resource Planning (ERP) platform engineered to resolve the systemic inefficiencies found in traditional academic management systems. By leveraging a high-performance Python/Flask backend and an immersive React frontend, NoteOrbit creates a unified data mesh connecting every university stakeholder.

Strategic Alignment

The platform is architected around the principle of **Zero-Latency Transparency**. Whether it is a student checking attendance or a parent messaging a faculty member, every interaction is recorded, authenticated, and delivered via optimized RESTful endpoints.

TECHNICAL LOGIC

Platform Pillars:

- **Stateless Authentication:**

  Ensuring high scalability across massive user clusters.

- **AI-Augmented Learning:**

  Context-aware study assistants utilizing LLM logic.

- **Integrated Logistics:**

  Holistic management of hostels, libraries, and finances.

Technological Foundations

The system utilizes a modern technology stack selected for its robustness and community support, ensuring long-term maintainability and modular growth paths for institutional developers.

Backend Architecture

The backend core of NoteOrbit is a specialized Flask implementation that handles complex institutional business logic, file parsing, and external API orchestrations.

1. The Security Layer (JWT Identity)

Identity is handled through stateless JSON Web Tokens. Each token contains claims that dictate the user's role and scope of authority. This prevents Role Escalation attacks at the API gateway level.

```
# Backend Role Enforcement Decorator def verify_role(role): def wrapper(f): @wraps(f) def decorated_function(*args, **kwargs): claims = get_jwt() if claims.get("role") =≠ role: return jsonify({"error": "Unauthorized Access"}), 403 return f(*args, **kwargs) return decorated_function return wrapper
```

2. Object Storage Pattern (S3/MinIO)

To avoid local disk I/O bottlenecks and ensure cloud compatibility, all documents are stored in S3-compatible buckets. The backend serves these files via **Secure Presigned URLs**, which expire after 3600 seconds, ensuring that private student data is never cached on public CDNs.

Database Schematics

The relational model of NoteOrbit is the foundation of its integrity. We employ SQLAlchemy to manage a deep relational graph with specific attention to cascading deletions and academic hierarchies.

The User Matrix

The `User` table is the root identity provider. It supports polymorphic attributes through its `role` and `status` columns.

ATTRIBUTE	DATA TYPE	PURPOSE
id	Integer	Global unique primary key
srn	String(64)	Student-specific identifier
password_hash	String(256)	Bcrypt-salted credential hash
role	String(16)	Identity category (admin/faculty/etc)
is_verified	Boolean	OTP verification status

Academic Mapping

The `Subject` and `Degree` tables form the backbone of the academic context. Every resource (Note, Notice, Attendance) is strictly mapped to this context to ensure data isolation between different university departments.

Admin Governance Protocol

The Administrative portal provides vertical control over the university's digital twin. This section details the operational workflows for super-users.

1. The Onboarding Filter

Self-registration is only the first step. To maintain institutional integrity, all new accounts enter a **Sandboxed Pending State**. Admins must verify physical credentials before authorizing the account for live portal access.

2. Faculty Allocation Logic

Faculty members are isolated from all data by default. An Admin must explicitly create a mapping between a `FacultyID` and a `SubjectID` for a specific `Section`. This mapping populates the professor's dashboard dynamically.

TECHNICAL LOGIC

Admin Best Practice:

Always perform a 'Dry Run' review of the Subject Catalog before initiating bulk faculty allocations to prevent scheduling conflicts.

Faculty Operations & Automation

The Faculty module is engineered to reduce the cognitive load on educators by automating non-teaching tasks like attendance tracking and material distribution.

1. The 30-Minute Attendance Integrity Lock

Academic records must be immutable. NoteOrbit enforces a 30-minute edit window. After this period, the attendance record is permanently locked for the faculty, and any changes must be requested through an Admin-level audit override.

```
# Server-side Time Lock Verification now = datetime.utcnow() if (now - record.created_at).total_seconds() > 1800: abort(403, description="Record modification window expired.")
```

2. Resource Indexing for AI

When a professor uploads notes, the backend triggers a text-extraction worker. This worker parses the document content and indexes it into the session memory, allowing students to use the Orbit Bot for document-specific inquiries immediately.

Student Hub & Personalized Insights

The Student Portal is a context-aware workspace. It uses the student's identity metadata to build a personalized dashboard that highlights deadlines, grades, and resources.

1. AI-Powered Academic Insights

The Academic Insights feature is NoteOrbit's "Diagnostic Tool." It aggregates raw data (attendance/marks) and passes it to the Llama 3.3 model. The AI provides a qualitative assessment of the student's progress, identifying strengths and potential academic pitfalls.

2. The Orbit Bot Interface

The Orbit Bot serves as a 24/7 academic companion. It doesn't just answer generic questions; it provides answers based on the specific study material uploaded by the student's own professors, ensuring curriculum alignment.

Student Tip: Uploading module summaries to the Orbit Bot allows for targeted exam revision and rapid concept synthesis.

Parent Portal & Communication

NoteOrbit recognizes parents as vital stakeholders. The Parent Portal provides a transparent window into the student's academic standing without requiring constant manual updates from the university staff.

1. Real-Time Performance Calendars

Parents have access to a live attendance calendar. Every period is marked in real-time. If a student is absent, the calendar updates immediately, and an automated SMTP alert can be triggered to the parent's registered email.

2. Direct Faculty Messaging

The platform facilitates secure, role-restricted messaging. Parents can inquire about specific performance issues, and professors can reply through the same interface, keeping all academic communication within the university's searchable and secure database.

Orbit Bot: Prompt Engineering & AI Logic

The intelligence layer of NoteOrbit is built on the Groq Cloud API, specifically utilizing the Llama-3.3-70B model for high-reasoning tasks.

1. Retrieval-Augmented Strategy

The bot does not rely on pre-trained knowledge alone. It uses a dynamic context window. When a query is made, the backend retrieves the relevant text chunks from the student's notes and prepends them as a "Grounding Context" to the AI's system instructions.

AI System Instruction Construction system_prompt = f""" You are an Academic Coach. Use the following text to answer the user: {document_context} Strictly follow the university curriculum rules. """

2. Conversation Context Retention

Each interaction is part of a `session_id`. This allows the AI to maintain a stateful conversation, remembering previous questions and refining its answers as the student provides more detail about their learning needs.

Hostel Logistics & Asset Management

Academic success is tied to student well-being. NoteOrbit manages physical living conditions through the Hostel & Maintenance module.

1. Dynamic Room Capacity Tracking

Room allocation is a real-time mathematical operation. The system tracks bed occupancy and automatically disables allocation options for rooms that have reached their defined capacity limit.

2. Maintenance Ticketing Logic

When a hostel complaint is raised, it enters a structured state-machine: `OPEN` -> `UNDER_REVIEW` -> `IN_PROGRESS` -> `RESOLVED`. Every state change creates a cryptographic timestamp in the audit trail, preventing disputes regarding repair timelines.

COMPLAINT STATUS	ACTION TAKEN	VISIBILITY
OPEN	Ticket Created	Student & Admin
REVIEW	Admin Acknowledged	Student & Admin
RESOLVED	Maintenance Fixed	All Stakeholders

Financial Infrastructure & Demo Checkout

The Finance module handles the complex billing cycles associated with university tuition, hostel fees, and library fines.

1. The Fee Target Engine

Admins don't need to bill students one by one. The system uses a "Batch Billing" engine. Admins define a fee criteria (e.g., "All 3rd Sem BCA Students"), and the backend creates personalized `FeeTarget` records for every student in that subset.

2. Secure Receipt Generation

Payment receipts are generated as immutable PDF documents. Each receipt carries a unique UUID and is served through the secure S3 presigned URL pattern, ensuring that financial proof is always available to both the student and the parent for audit purposes.

TECHNICAL LOGIC

Financial Security:

The demo gateway uses a simulated handshake. In production, this is swapped with a Stripe/Razorpay webhook listener that updates the `PaymentStatus` in real-time.

Frontend Architecture & UX Strategy

The client-side application is a high-performance Single Page Application (SPA) built with **React 18**. It abandons traditional server-side templating for a decoupled, API-driven architecture.

1. The "Glassmorphism" Design System

NoteOrbit employs a "Modern Dark Glass" aesthetic to maximize visual engagement. This is achieved using **Tailwind CSS** utility classes to layer semi-transparent backgrounds with backdrop-blur filters.

```
/* Glass Effect Utility Pattern */ .bg-slate-800/50 .backdrop-blur-xl .border-white/10 .shadow-2xl
```

2. Component Ecosystem

The UI is composed of atomic, reusable components powered by **Lucide React** icons and **GSAP** animations.
Core Components:

- **OrbitLogo:** A pure CSS/SVG identity component.
- **ParticleBackground:** A canvas-based ambient animation layer.
- **CredentialsView:** A 3D-flipping authentication form.

```
// OrbitLogo Component (Visual Identity) const OrbitLogo = () => { return ( NoteOrbit ) }
```

3. State Management Hooks

Frontend Component Specifications

The interface relies on a strict set of controlled components to ensure visual consistency and state synchronization. This section details the props and internal logic of the core form elements.

1. The `Input` Component

A wrapper around the native HTML `input` element that provides icons, glassmorphism styling, and detailed password interactions.

PROP	TYPE	DESCRIPTION
icon	LucideIcon	Dynamic icon rendered absolutely positioned within the input container.
type	String	Supports 'text', 'password', 'email'. Defaults to 'text'.
className	String	Additional Tailwind classes merged via template literals.

```
// Internal Password Toggle Logic const [showPassword, setShowPassword] = useState(false); const inputType = isPassword ? (showPassword ? 'text' : 'password') : type;
```

2. The `Select` Component

Provides a custom-styled dropdown with a custom chevron icon, replacing the browser's default OS-level UI for a cohesive look.

UX Decision:

Notification & Feedback System

User feedback is critical for perceived performance. The `MessageBar` component acts as the global notification handler, floating above the main content layer.

1. MessageBar Architecture

The component is conditionally rendered based on the global `message` state object. It utilizes CSS animations for entry and exit.

```
// Dynamic Class Assignment const classes = isSuccess ? "bg-emerald-500/10 border-emerald-500/20 text-emerald-200" : "bg-red-500/10 border-red-500/20 text-red-200";
```

2. Animation Classes

We leverage Tailwind's `animate-in` primitives for smooth transitions:

- `animate-in`: Initializes the animation sequence.
- `fade-in`: Opacity transition from 0 to 1.
- `slide-in-from-top-2`: Physical displacement for visual cue.

System Integrity: All API error responses (4xx/5xx) are automatically routed to the MessageBar via the `showMessage` helper, ensuring no error goes unnoticed by the user.

Client-Side Persistence Layer

NoteOrbit minimizes server round-trips by implementing an intelligent local caching strategy using React Custom Hooks.

1. The `useLocalUser` Hook

This hook acts as the primary session manager. consistently synchronizing the React State with the browser's `localStorage`.

TECHNICAL LOGIC

Boot Sequence:

1. Check `localStorage` for `noteorbit_user` and `noteorbit_token`.
2. If found, JSON.parse the user object and JSON.parse the token.
3. Verify token integrity (optional structure check).
4. Set global `axios` default headers (`Authorization: Bearer ...`).
5. Hydrate the Application State.

2. Error Handling in Persistence

If JSON parsing fails (due to data corruption), the hook executes a `catch` block that automatically purges the corrupted data to prevent a "White Screen of Death" boot loop.

Metadata Catalog Architecture

Dropdown menus for Degrees, Sections, and Subjects require constant data access. The `useCatalogs` hook abstracts this fetching logic.

1. The `fetchBasics` Strategy

On component mount, `useCatalogs` fires parallel requests to populated basic lists:

```
// Parallel Execution for Performance const [deg, sec] = await Promise.all([
unauth().get("/admin/degrees"), unauth().get("/admin/sections") ]);
```

2. Reactive Subject Fetching

Subject lists are dependent variables (functional dependency on `Degree` + `Semester`). The `fetchSubjects` function is wrapped in `useCallback` to prevent infinite render loops when included in `useEffect` dependencies.

Optimization: This hook does not cache results in RAM between mounts. In V2.3, we plan to implement `React Query` to cache these immutable lists for 5 minutes.

AI Integration: The Engine Room

The orbit of NoteOrbit's intelligence is the `AIChat` component, interfacing with the Groq Cloud API for ultra-low latency inference using **Llama-3.3-70B**.

1. The Message Pipeline

State is managed via a `history` array. When a user sends a message:

1. The local UI updates optimistically (`setHistory`).
2. A `FormData` object is constructed (handling text + file attachments).
3. If a `session_id` exists, it is appended to maintain context.

2. Handling Multipart Streams

The AI endpoint is not a simple JSON REST endpoint. It accepts `multipart/form-data`. This allows the simultaneous transmission of a prompt string and a binary file (PDF/DOCX) for RAG (Retrieval-Augmented Generation) analysis.

AI Session State Architect

Context retention is vital for a conversational tutor. NoteOrbit implements a robust session tracking mechanism.

1. Session Initialization

If `currentSessionId` runs as null, the backend interprets the first message as a "Genesis Prompt" and creates a new Session row in the database, returning the new `UUID`.

2. Sidebar History Logic

The sidebar lists previous conversations. Clicking a history item triggers `loadSession(id)`.

```
const loadSession = async (sessionId) ⇒ { if (currentSessionId ═══ sessionId) return; setIsLoading(true); // Fetch full message transcript const res = await auth().get(`/ai/session/${sessionId}`); setHistory(res.data.messages); }
```

Privacy: Users can explicitly delete sessions via the Trash icon. DELETE requests are hard-deletes, removing the conversation vector embeddings from the server.

Unified Library Search Protocol

NoteOrbit acts as an aggregator, pulling resources from both the internal `Uploads` bucket and the public `OpenLibrary` API.

1. Dual-Source Toggle

The frontend `UnifiedLibrarySearch` component maintains a `searchSource` state state ('internal' | 'openlibrary'). This flag acts as a router switch for the backend controller.

2. Query Construction

All queries are URI-encoded to ensure safe transmission of special characters.

```
const query = encodeURIComponent(searchTerm.trim()); await
auth().get(`/api/library/search?q=${query}&source=${searchSource}`);
```

3. Result Normalization

Since OpenLibrary and Internal DB have different schemas, the Backend Adapter (in `app.py`) normalizes the response into a standard `Book` object: `{ title, author, isbn, cover_url, file_url, source }`.

Hostel Complaint State Machine

The `HostelComplaints` module implements a client-side state machine that reflects the user's allocation status restrictions.

1. The Allocation Gate

Before allowing a complaint submission, the system performs a `fetchUserComplaints` call. If this returns a **403 Forbidden**, the UI enters a `locked` state.

STATUS STATE	UI BEHAVIOR
checking	Shows Loading Spinner.
allowed	Renders Form & History List.
not_allowed	Renders "Allocation Required" error banner.

2. Audit Trail Visualization

The `ComplaintAuditTimeline` component renders a visual vertical timeline of the ticket's lifecycle, parsing the JSON `audit_trail` array.

Financial Gateway Integration

NoteOrbit demonstrates a "Redirect Pattern" integration standard for securely handling tuition fees.

1. Order Generation

The client never processes cards directly. Instead, `handlePay()` calls `POST /fees/pay` with a `target_id`. The server returns an `order_id`.

2. The Handshake Redirection

Once the `order_id` is acquired, the window is hard-redirected to the mock payment processor:

```
window.location.href = `${BACKEND_BASE_URL}/demo/checkout/${order_id}`;
```

3. Receipt Fulfillment

Post-payment, the `handleReceipt` function accesses a secure endpoint to generate a dynamic PDF. This ensures the receipt is always generated from live DB data, not client-side state.

Faculty Portal: Marks Logic

Data entry accuracy is enforced through strict frontend validation logic in the `ProfessorMarksUpload` component.

1. Cascading Dropdown Filters

To prevent data entry errors, the selection menus are interdependent:

- Degree selection updates the Semester list.
- Semester selection updates the Section list.
- Section selection updates the Subject list.

This "Narrowing Funnel" ensures a professor literally cannot assign marks to a subject that doesn't exist for a specific class.

2. Validation Rules

The `handleUpload` function enforces logical constraints before the API call:

```
if (parseFloat(marksObtained) > parseFloat(maxMarks)) { showMessage("Error: Obtained > Max", 'error'); }
```

API Reference: Identity Module

A comprehensive listing of the REST endpoints consumed by the frontend application.

Authentication

METHOD	ENDPOINT	PAYLOAD
POST	/auth/login	{email, password}
POST	/auth/register-student	{srn, name, ...}
POST	/auth/send-otp	{identifier, mode}
POST	/auth/verify-otp	{email, otp}

User Context

METHOD	ENDPOINT	RESPONSE
GET	/auth/verify-token	{valid: true}
POST	/auth/lookup-user	{masked_email}

API Reference: Academic Module

Endpoints governing the flow of educational data and resources.

Catalog Data

METHOD	ENDPOINT	DESCRIPTION
GET	/admin/degrees	List of active degrees
GET	/admin/sections	List of valid sections
GET	/admin/subjects	Context-aware subject list

Content Delivery

METHOD	ENDPOINT	DESCRIPTION
GET	/notes	Fetches PDFs for subject
GET	/notices	Fetches circulars
GET	/student/marks	Aggregated marks entry

API Reference: Logistics & AI

Endpoints handling physical assets and intelligence services.

Hostel Management

METHOD	ENDPOINT	DESCRIPTION
GET	/student/hostel	Room allocation info
POST	/hostel/complaint	Ticket creation (Multipart)
GET	/student/complaints	Ticket history

AI Services

METHOD	ENDPOINT	DESCRIPTION
POST	/chat	Groq Llama Inference
GET	/ai/sessions	Conversation history list
DELETE	/ai/session/:id	Hard delete session

Security Architecture Review

NoteOrbit employs a "Defense in Depth" strategy, securing data at the Transport, Session, and Application layers.

1. Transport Security

All traffic is mandated over HTTPS (TLS 1.3). The API Gateway (Nginx/Gunicorn) handles SSL termination, ensuring encrypted payloads for all login and payment packets.

2. Token Hygiene

JWTs are stored in `localStorage` for ease of access but have short expiry windows (1 hour). Refresh Tokens (HttpOnly Cookies) are planned for V3.0 to mitigate XSS risks.

3. Role-Based Access Control (RBAC)

Security is enforced on the Backend. Even if a user "hacks" the frontend to show the Admin Panel, the API endpoints will reject their requests.

```
# Backend Enforcement @jwt_required() def delete_user(): if current_user.role ≠
'admin': return 403_FORBIDDEN
```

Performance Optimization

Delivering a rich UI on university networks requires aggressive optimization strategies.

1. Code Splitting & Lazy Loading

The React build pipeline uses Webpack to bundle code into chunks. Heavy components (like the PdfViewer or ChartJS) are lazy-loaded only when the route is accessed.

2. Animation Performance

We use **GSAP (GreenSock)** because it runs outside the React Render Cycle, directly manipulating DOM styles. This prevents "Jank" during heavy layout trashing.

```
// Efficient Animation gsap.to(ref.current, { x: 100, duration: 0.5, ease: "power2.out",
force3D: true // Uses GPU acceleration });
```

3. Asset Compression

All uploaded images are processed by a Lambda worker, converted to WebP format, and resized before being stored in S3, reducing payload sizes by ~60%.

Disaster Recovery & Redundancy

Ensuring continuity during academic exams and admissions seasons.

1. Database Replication

The PostgreSQL primary node streams Write-Ahead Logs (WAL) to a read-replica in a different Availability Zone (AZ). If the primary fails, the replica is promoted automatically within 60 seconds.

2. S3 Versioning

Versioning is enabled on the `noteorbit-uploads` bucket. If a malicious actor deletes all notes, the objects can be restored to their previous state instantly.

3. Automated Snapshots

Full system snapshots are taken every 6 hours during peak exam weeks, retained for 90 days for compliance audits.

Project Glossary

Definitions of domain-specific terminology used within the ecosystem.

TERM	DEFINITION
SRN	Student Registration Number; unique immutable ID.
AICTE	Regulatory body standards adhered to by the curriculum engine.
JWT	JSON Web Token; used for stateless auth.
RAG	Retrieval-Augmented Generation; AI strategy using local docs.
Glassmorphism	UI style using simulated frosted glass.
Presigned URL	Temporary, secure link to a private S3 object.

Acronyms

- **ERP:** Enterprise Resource Planning
- **LMS:** Learning Management System
- **SPA:** Single Page Application

Deployment & Scaling Protocols

Transitioning NoteOrbit from a development environment to a production university server requires strict adherence to institutional security and performance standards.

1. Environment Hardening

Before deployment, all `DEBUG` flags must be toggled to `False`. The JWT secret keys should be rotated to high-entropy 64-character strings stored in an encrypted environment file or a secret management service like HashiCorp Vault.

2. Performance Tuning (PostgreSQL Migration)

While SQLite is ideal for development, university environments require a robust RDBMS like PostgreSQL. The SQLAlchemy connection URI must be updated, and a migration script must be executed to move current data into the partitioned production tables.

```
# Production DB Configuration Example app.config['SQLALCHEMY_DATABASE_URI'] =
os.getenv('PROD_DB_URL') app.config['JWT_ACCESS_TOKEN_EXPIRES'] = timedelta(hours=2)
```

Maintenance & System Support

Effective ERP operation depends on consistent monitoring and a proactive response to technical anomalies.

Common Technical Exceptions

- **SMTP Auth Errors:** Usually caused by incorrect Gmail App Passwords or Port 587 blockages on the server firewall.
- **S3 Timeout:** Check MinIO service status and ensure the `S3_ENDPOINT` is accessible from the Flask worker.
- **JWT Blacklisting:** Occurs when a token is used after a logout event; ensure the frontend clears the token locally on sign-out.

Future Expansion Roadmap

The V3.0 release will introduce **Real-time WebSockets** for instant notifications and a **Blockchain-backed Transcript** module for verifiable digital credentials.

LEAF CORE LABS

LEAFCORE LABS – OFFICIAL TECHNICAL SHEET

Where Imagination is Redefined!

www.leafcorelabs.in