FSU ITS

2026 RISE Showcase

The

Transcripter

Ammiel Bowen

Special Projects Developer · Information Technology Services

Florida State University

About Me

Name

Ammiel Bowen

Major

Computer Science

University

Florida State University

Role

Special Projects Developer

Department

Information Technology Services

Who Am I?

I'm a Computer Science student at Florida State University working as a Special Projects Developer within the ITS department.

Over the past 6 months, I was given the opportunity to design, build, and deploy a full-stack internal web application from the ground up — with real stakeholders, real requirements, and real users.

I came in with intermediate experience in web development, but this project pushed me into backend architecture, database design, role-based security, and production-level thinking.

The Problem

Hundreds

of transcripts processed manually every week

No Central Storage

Transcript files scattered across emails, shared drives, and manual spreadsheets.

No Access Control

Anyone with drive access could view or modify sensitive student records.

No Standardized Process

Every employee handled transcripts differently — no consistent workflow.

Zero Analytics

Leadership had no visibility into submission volume, sources, or trends.

The Office of Admissions needed a purpose-built internal tool — not a workaround.

The Stakeholders

FSU Office of Admissions

Primary stakeholder and end user. The Admissions team receives, processes, and routes hundreds of college transcripts weekly as part of the student application and enrollment workflow.

Point of Contact

Bernica Rollison

Office of Admissions — FSU

What They Needed

- Centralized storage for transcript files
- Role-based access for staff teams
- Fast search, filtering, and editing
- Analytics on volume and sources

FSU Information Technology Services (ITS) — Supporting Department

ITS provided the development environment, infrastructure access, and project oversight. As Special Projects Developer, my role bridged both departments — translating Admissions' operational needs into a technical solution.

The Solution

Introducing the Transcripter

Intake

Structured form for submitting transcript data with validation

Storage

MongoDB + GridFS stores data and original PDF files

Access Control

Role-based system restricts actions by employee type

CRUD

Full create, read, update, and soft-delete lifecycle

Analytics

Live charts and KPI cards for leadership visibility

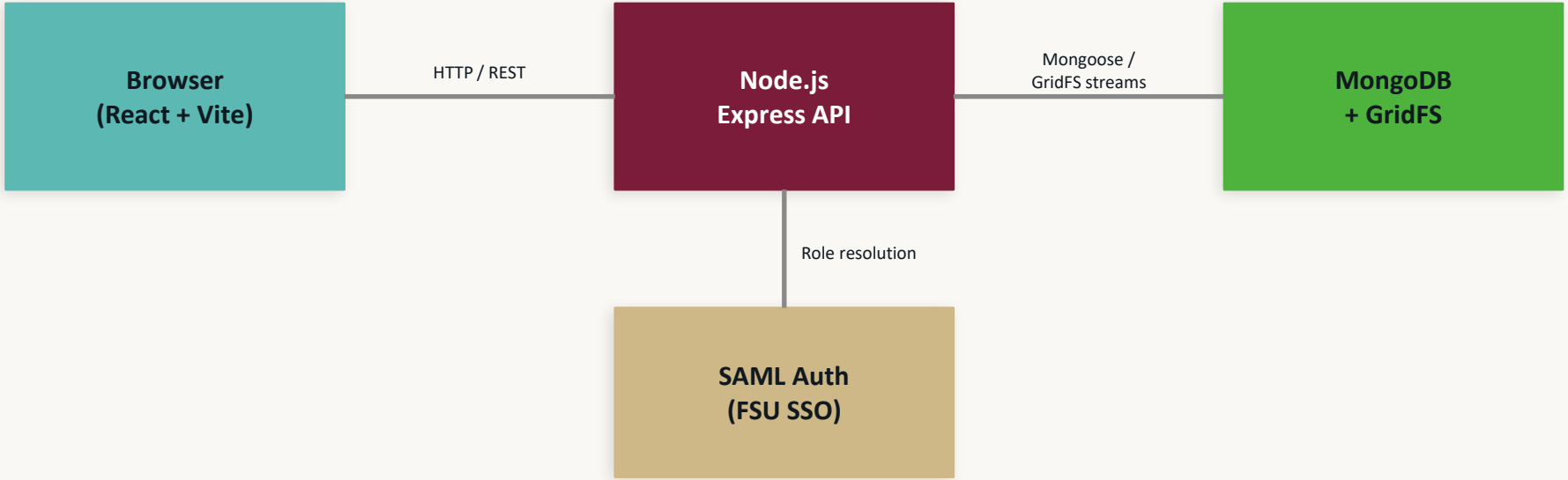
Audit Trail

Every action logged with timestamp and acting user

Tech Stack

FRONTEND	**React 19 + Vite**	Fast SPA build tooling, component-based UI, React Router for navigation, Recharts for analytics
BACKEND	**Node.js + Express**	RESTful API server, route-level middleware, centralized validation and error handling
DATABASE	**MongoDB + GridFS**	Document store for transcript metadata and employee records; GridFS streams raw PDF files
AUTH	**SAML (In Progress)**	FSU SSO integration — users will authenticate with their FSU credentials and auto-assign roles
TOOLING	**MongoDB Compass · PapaParser · react-select**	DB inspection, CSV parsing for 54K institutions, async searchable dropdowns

System Architecture



- The React frontend is served as a static build and communicates with the Express API via REST endpoints.
- MongoDB stores structured transcript and employee data; GridFS handles raw PDF binary storage.
- SAML will authenticate users against FSU's SSO — roles are assigned at login and enforced on every route.

Project Timeline

Phase 1

Foundation

Fall 2025

Phase 1: Schemas, barebones login, upload, and output. Identified core problems.

Phase 2

Backend Logic

Fall 2025

Centralized validation, soft delete, audit logging, destination ID rules.

Phase 3

Core UI

Fall 2025

Submission form, Edit Transcript page, dynamic ID prompts, full CRUD.

Phase 4

Employee System

Winter 2025

Register Employee, Employee List page, CRUD for users.

Phase 5

Admin Log

Winter 2025

Admin-only activity log: Created, Edited, Hidden, Restored events.

Phase 6

Analytics + Polish

Spring 2026

Analytics dashboard, date-range filters, UI polish and QA.

Phase 7

SAML Auth

Coming Soon

FSU SSO integration for production-ready authentication and role mapping.

Total project duration: Fall 2025 → Spring 2026 (~6 months)

Roles & Permissions

Why roles matter: Not every employee should see or do everything.
Role-based access ensures data integrity and limits human error.

Admin

- ✓ Full CRUD on all transcripts
- ✓ Register + manage employees
- ✓ Assign and modify roles
- ✓ View analytics dashboard
- ✓ Access admin activity log
- ✓ Can restore hidden transcripts

Imaging Staff / Technician

- ✓ Submit new transcripts
- ✓ Edit existing transcript records
- ✓ View all transcript data
- ✓ View analytics
- ✓ Cannot manage employees
- ✓ Cannot access admin log

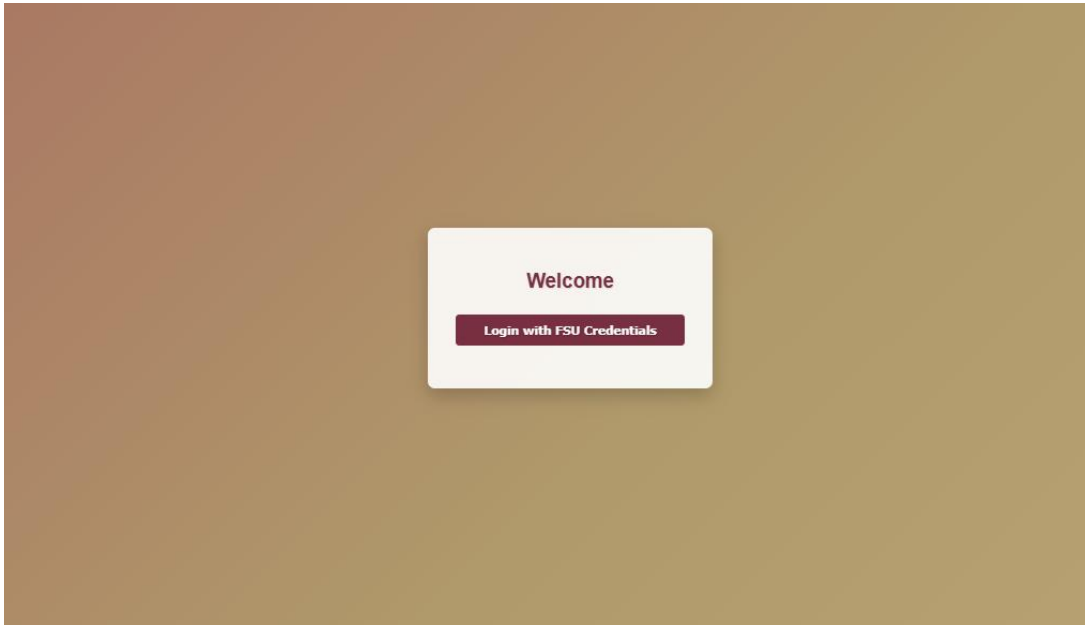
Viewer

- ✓ Read-only access to transcript list
- ✓ Search and filter records
- ✓ No submission ability
- ✓ No editing ability
- ✓ No analytics access
- ✓ No admin features

Screen Walkthrough

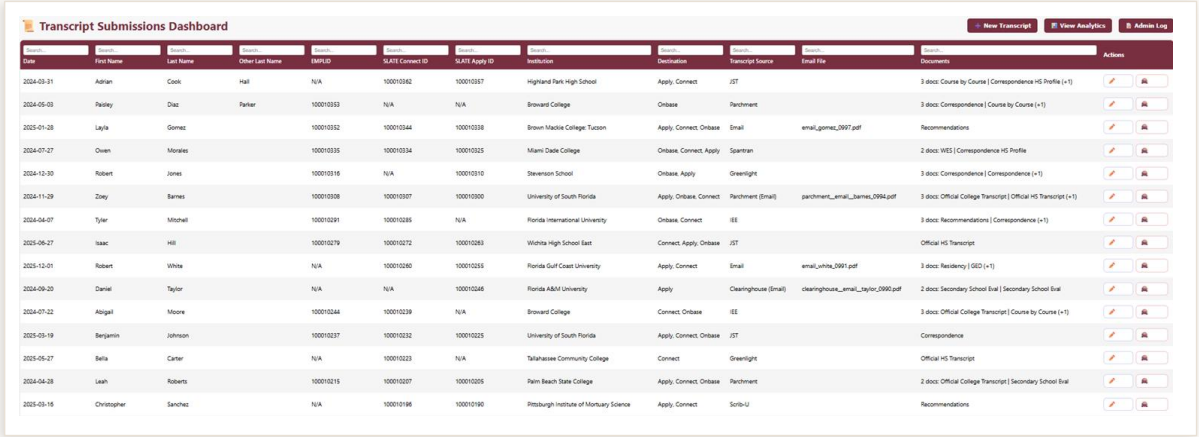
Every screen built over 6 months — purpose, design, and function

Login Screen



- Single-button FSU login
- SAML will identify user on sign-in
- Role determined at authentication
- Redirects to correct landing page
- Garnet & gold FSU brand colors
- Clean, minimal design

Transcript Submissions Dashboard

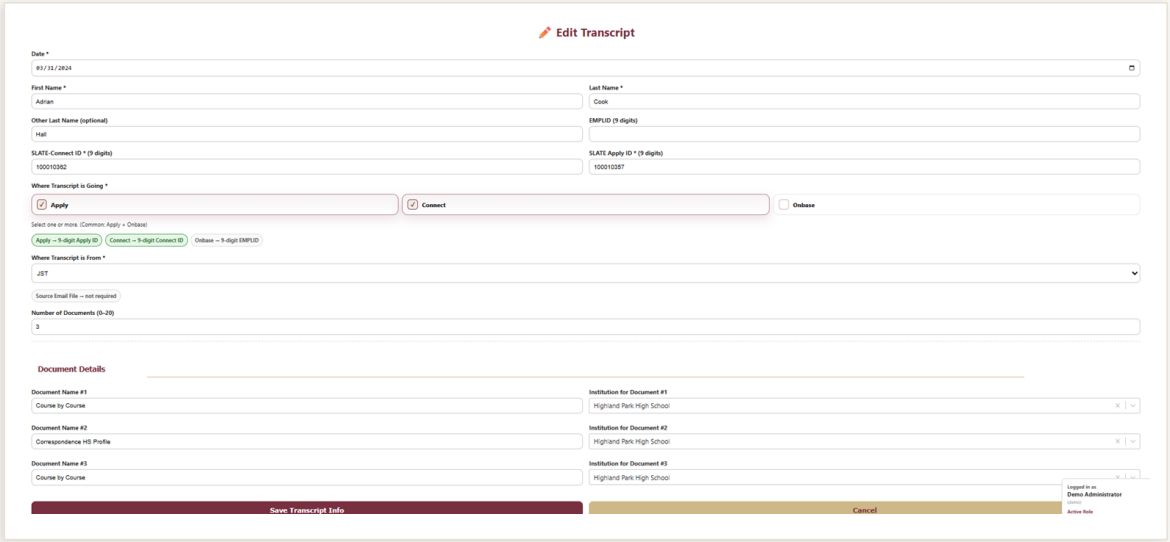


- Displays all transcript records
- Search bar on every column
- Edit icon per row (role-gated)
- PDF viewer button per record
- New Transcript + Analytics buttons
- Soft-deleted records are hidden

Transcript Submission Form

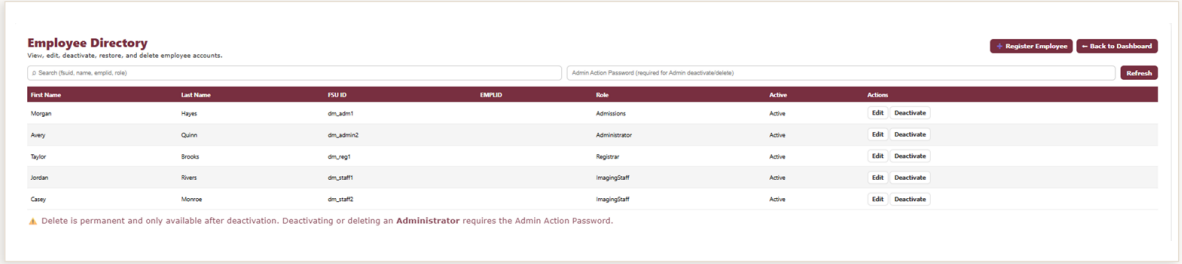
- Date, First, Last, Other Last Name
- EMPLID OR Slate Apply OR Connect ID
- Dynamic ID prompt by destination
- Multi-select destination (Apply/Connect/Onbase)
- Source dropdown (Parchment, Email, etc.)
- Document count + named doc list (max 20)

Edit Transcript



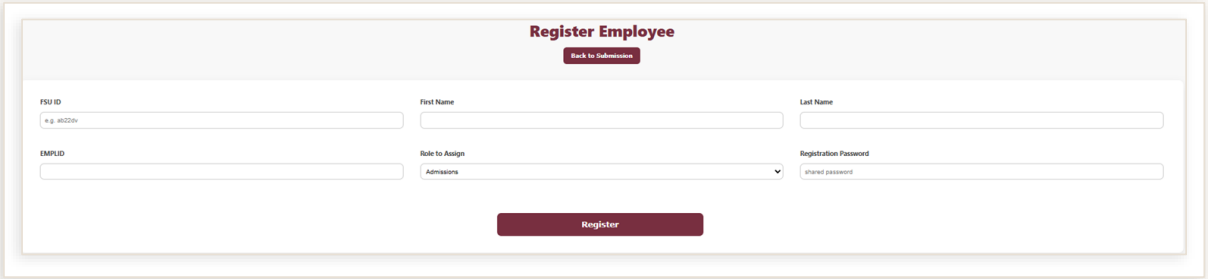
- Same layout as submission form
- Auto-populates existing record data
- Validates all fields before saving
- Updates Last Edited timestamp
- Writes 'Edited' event to audit log
- Cancel returns to dashboard safely

Employee Directory



- Lists all registered employees
- Columns: Name, FSUID, Role
- Admin can edit or delete records
- Accessible from Submission form
- Admin-only page (role-gated)
- Same table design language as Dashboard

Register Employee

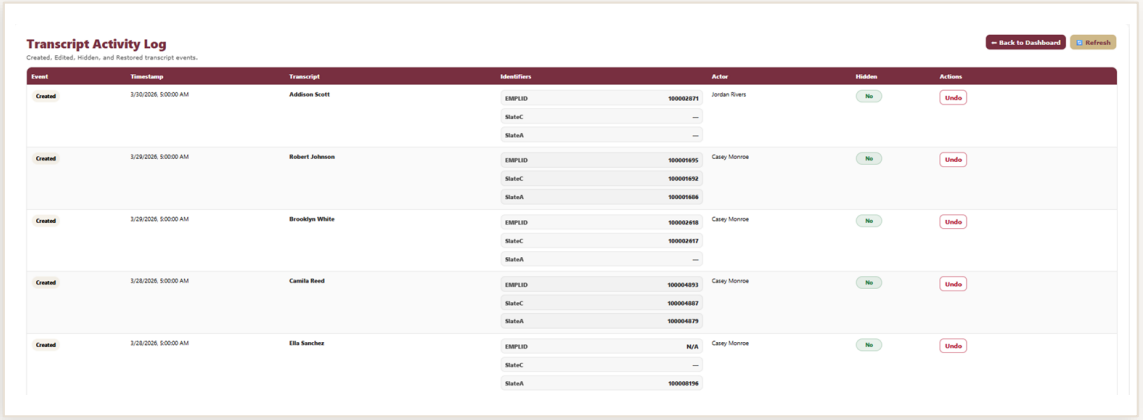


- Admin-only registration form
- Fields: First Name, Last Name, FSUID
- Role assignment dropdown
- Validates all required fields
- Submits to MongoDB User collection
- Used before SAML full rollout

Role Picker & Role Attach

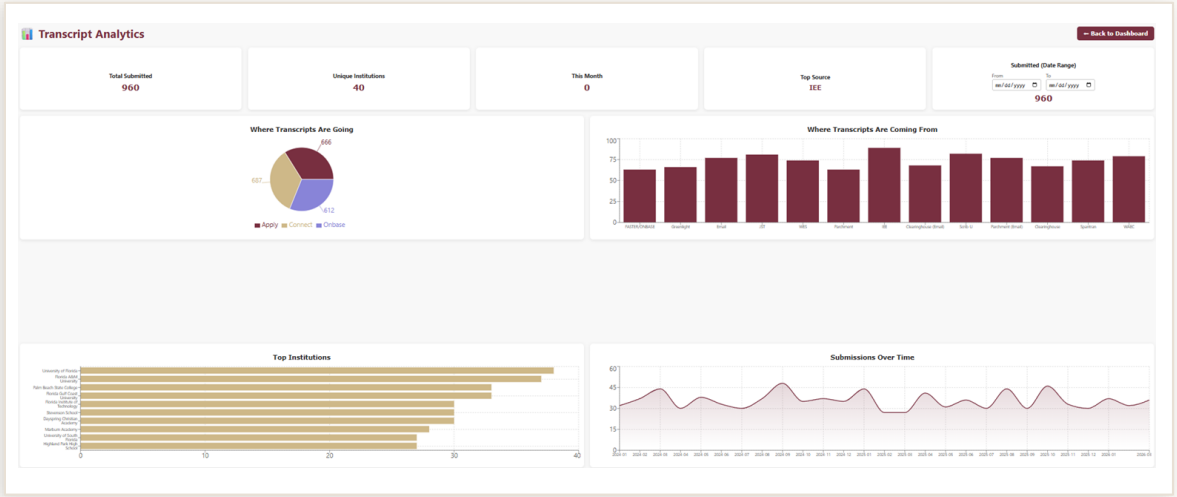
- Admin assigns roles to users
- Roles tie to SAML identity (FSU ID)
- Available roles: Admin, Imaging Staff, Viewer
- Role is enforced on every API route
- checkRole.js middleware protects backend
- Future: auto-assign via SAML attributes

Administrator Activity Log



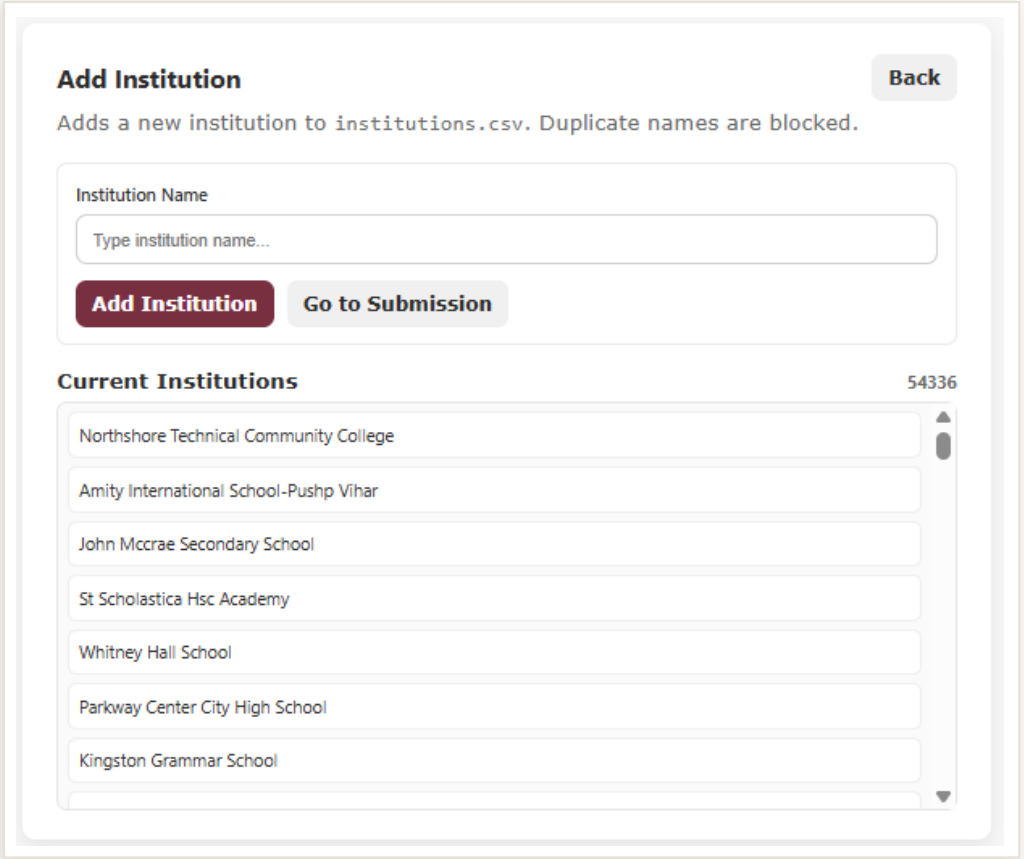
- Admin-only audit trail page
- Events: Created, Edited, Hidden, Restored
- Each entry: timestamp + acting user
- Restore button for hidden transcripts
- Restoring creates a new 'Restored' event
- Full accountability for every change

Transcript Analytics



- KPI cards: Total, Unique Institutions, Month
- Top Source and Date-Range submission count
- Pie chart: Where transcripts are going
- Bar chart: Where transcripts are coming from
- Top Institutions horizontal bar chart
- Submissions Over Time line chart

Add Institution



- Admin + Imaging Staff can add institutions
- New entries join the 54,334 institution list
- Available in Submission form immediately
- Prevents duplicate entries
- Backed by MongoDB institutions collection
- Async-select auto-filters as user types

Feature Deep Dives

The technical details that make the Transcripter unique

54,334 Institutions — Async Autocomplete

54,334

colleges & universities
in the institution dataset

The Challenge

Loading 54K records into a standard dropdown would freeze the UI entirely on every keystroke.

The Solution

react-select's AsyncSelect component fetches a filtered subset from the API on each keystroke — only 10-20 results at a time.

The Backend

The Express /institutions route accepts a search query param and returns only matching institutions, keeping payloads tiny.

The Dataset

institutions.csv was parsed with PapaParser and seeded into MongoDB — the single source of truth for all institution lookups.

Add Institution

Admins and Imaging Staff can add new institutions directly through the UI — no manual DB edits required.

Soft Delete & Audit Log System

Why it matters: In a system handling student records, hard deletes are dangerous. Every action must be traceable.

CREATED

Written when a new transcript is submitted. Captures submitting user and timestamp.

EDITED

Written every time a transcript is saved from the Edit page. Old data is never lost.

HIDDEN

Replaces hard delete. Sets Hidden=true on the record. Still stored — just not visible.

RESTORED

Admin clicks Restore on the log page. Sets Hidden=false and creates a Restored event.

Each log entry stores: Event Type · Timestamp · Acting User (First + Last) · Transcript Reference
This gives administrators a complete, tamper-evident history of every record in the system.

Unique Code Aspects

Exclusive ID Validation

Only ONE of EMPLID / Slate Apply ID / Slate Connect ID can be set at a time. The validation layer enforces mutual exclusivity — not the UI alone. Backend rejects any submission violating this rule.

validation.js

Role Middleware Chain

Every Express route passes through checkRole.js middleware before executing. Roles are read from the session and compared against allowed roles per route — unauthorized requests are rejected at the API level.

checkRole.js

Document Name Dropdowns (Max 20)

Users can add up to 20 named document entries per transcript submission. Each row is a controlled dropdown with a fixed list of valid document types, preventing free-text data quality issues.

TranscriptSubmission.jsx

Dynamic Destination Prompts

The submission form detects which destination (Apply/Connect/Onbase) is selected and dynamically shows which ID field is required. The UI prompt updates in real time with no page reload.

TranscriptSubmission.jsx

GridFS Binary PDF Storage

Transcript PDF files are streamed into MongoDB GridFS rather than stored on disk. This keeps the file system clean and makes the backend stateless and portable across environments.

server.js / transcript.js

Auto-Populate by EMPLID

If an employee enters an existing EMPLID during submission, the form auto-fills the applicant's name fields. This reduces data entry errors and speeds up repeat submissions for the same student.

TranscriptSubmission.jsx

Why This Color Scheme?

#7B1C3B

FSU Garnet

Primary brand color. Used for headers, buttons, accents, and all primary UI elements.

#CEB888

FSU Gold

Secondary brand color. Used for highlights, hover states, stat callouts, and analytics charts.

#FFFFFF

White

Card backgrounds and clean surfaces. Provides high contrast against garnet elements.

#F7F4F0

Off-White

Page background. Softer than pure white — reduces eye strain for long work sessions.

#1C1C1C

Near-Black

Body text. High legibility on light backgrounds throughout the application.

The garnet and gold palette was a deliberate choice to align the application with FSU brand standards — reinforcing that this is an official university tool, not a generic third-party system.

Infrastructure & Code Deep Dive

How the Transcripter is actually built under the hood

Project Folder Structure

Clean separation between frontend (app/) and backend (api/) — two independent systems communicating over HTTP.

app/ (Frontend — React + Vite)

```
app/
  src/
    pages/
      Dashboard.jsx
      TranscriptSubmission.jsx
      EditTranscript.jsx
      EmployeeList.jsx
      AdminLog.jsx
      AnalyticsPage.jsx
      AddInstitution.jsx
    CSS/
      [per-page stylesheets]
    utils/
      api.js
      roleUtils.js
    App.jsx
    main.jsx
  index.html
  vite.config.js
```

api/ (Backend — Node.js + Express)

```
api/
  models/
    Transcript.js
    User.js
    Log.js
  routes/
    transcript.js
    admin.js
    auth.js
    institutions.js
  middleware/
    checkRole.js
  utils/
    validation.js
  seed/
    demoSeed.js
  server.js
  .env
```

Data Model — Transcript.js

Every transcript record stored in MongoDB follows this Mongoose schema. Notice the shared validator — all three ID fields use identical rules.

api/models/Transcript.js

```
const optionalNineDigitOrNA = {
  type: String,  default: 'N/A'
  validate: {
    validator: (v) =>
      v === null || /^\d{9}$/.test(v) || v === 'N/A'
  }
};

const transcriptSchema = new Schema({
  date: String
  firstName: String,  lastName: String
  otherLastName: String   // optional
  emplid: optionalNineDigitOrNA  // shared validator
  slateC: optionalNineDigitOrNA
  slateA: optionalNineDigitOrNA
  documentNames: [String]
  transcriptDestination: [String]
  hidden: { type: Boolean, default: false }
}, { timestamps: true });
```

Shared Validator

One validator function covers EMPLID, SlateC, and SlateA — DRY principle. Any value must be null, 'N/A', or exactly 9 digits.

Soft Delete Flag

The `hidden` boolean replaces hard deletes. Setting it to true removes the record from all views while preserving it in the database.

Auto Timestamps

`{ timestamps: true }` tells Mongoose to auto-add createdAt and updatedAt fields — no manual date tracking needed.

Role Middleware — checkRole.js

Every protected API route passes through this middleware before executing. Roles are normalized, matched, and rejected at the network boundary.

api/middleware/checkRole.js

```
module.exports = function checkRole(...allowed) {
  // Normalize: 'Admin', 'admin', ' admin ' → 'admin'
  const allowedNormalized = allowed
    .flat().map(r => String(r).trim().toLowerCase());

  return (req, res, next) => {
    const rawRole = req.get('x-role');
    if (!rawRole) return res.status(401).json(...);

    const roles = rawRole.split(',')
      .map(r => r.trim().toLowerCase());

    const ok = roles.some(r =>
      allowedNormalized.includes(r));

    if (!ok) return res.status(403).json({
      message: 'Forbidden: insufficient role'
    });
    next();  //     Authorized — continue
  };
};
```

How it's used on routes:

routes/transcript.js

```
router.post('/upload'
  checkRole('ImagingStaff'),
  async (req, res) => {
    // route handler runs here
  });
```

Variadic Arguments

checkRole() accepts any number of role strings — e.g. checkRole('Admin', 'ImagingStaff') allows either role through.

Case-Insensitive Matching

'Admin', 'admin', and 'ADMIN' all resolve to the same match. Prevents header casing bugs from silently breaking access.

Centralized Validation — validation.js

All validation rules live in one module — both the API routes and the schema import from here. One source of truth, no duplication.

api/utils/validation.js

```
//     9 digits, 'N/A', or empty — nothing else allowed
const validateOptionalId = (label, value) => {
  const v = String(value || '').trim();
  if (!v) return { ok: true, value: null };
  if (v === 'N/A') return { ok: true, value: 'N/A' };
  if (!/^\d{9}$/.test(v)) return
    { ok: false, message: `${label} must be 9 digits` };
  return { ok: true, value: v };
};

//     Destination-based ID requirement enforcement
const validateDestinationRules = ({ dests, emplid, slateC, slateA }) => {
  if (dests.includes('Apply') && !isNineDigit(slateA))
    return { ok: false, message: 'Apply requires SlateA' };
  if (dests.includes('Connect') && !isNineDigit(slateC))
    return { ok: false, message: 'Connect requires SlateC' };
  if (/onbase/i.test(dest) && !isNineDigit(emplid))
    return { ok: false, message: 'OnBase requires EMPLID' };
  return { ok: true };
};
```

DRY Validation

One function handles EMPLID, SlateC, and SlateA identically. Before this, each was validated separately — a maintenance nightmare.

Destination Enforcement

Apply requires a SlateA ID. Connect requires SlateC. OnBase requires EMPLID. This is business logic — enforced on the server, not just the UI.

Backend-First Safety

Even if a user bypasses the frontend validation (via direct API call), the backend rejects any request that violates these rules.

Soft Delete & Logging — transcript.js

The DELETE route doesn't remove data — it marks it hidden. The createLog helper writes an audit event for every state change.

createLog helper

```
const createLog = async (req, transcriptId, eventType) => {
  const actor = {
    actorFsuid:     req.headers['x-fsuid'],
    actorFirstName: req.headers['x-first-name'],
    actorLastName:  req.headers['x-last-name'],
  };
  await Log.create({ transcript: transcriptId, eventType, ...actor });
};
```

Soft DELETE route

```
router.delete('/submissions/:id',
  checkRole('ImagingStaff'), async (req, res) => {
    const doc = await Transcript.findByIdAndUpdate(
      req.params.id,
      { hidden: true },  // ← soft delete
      { new: true }
    );
    await createLog(req, doc._id, 'Hidden');
    res.json({ ok: true });
});
```

Actor from Headers

The user's identity (FSUID, name) is passed in request headers, not the body. This prevents spoofing from the frontend and ties every log entry to a verified session.

hidden: true

The DELETE endpoint never calls .deleteOne() or .findByIdAndDelete(). It only sets hidden=true — the record is fully preserved and restorable.

Automatic Event Trail

createLog() is called after every Create, Edit, and Delete operation. Logging is never optional — it runs server-side regardless of the UI.

Frontend Route Protection — App.jsx

React Router wraps every protected page in <RequireAuth>. Unauthorized navigation is caught client-side before the page even renders.

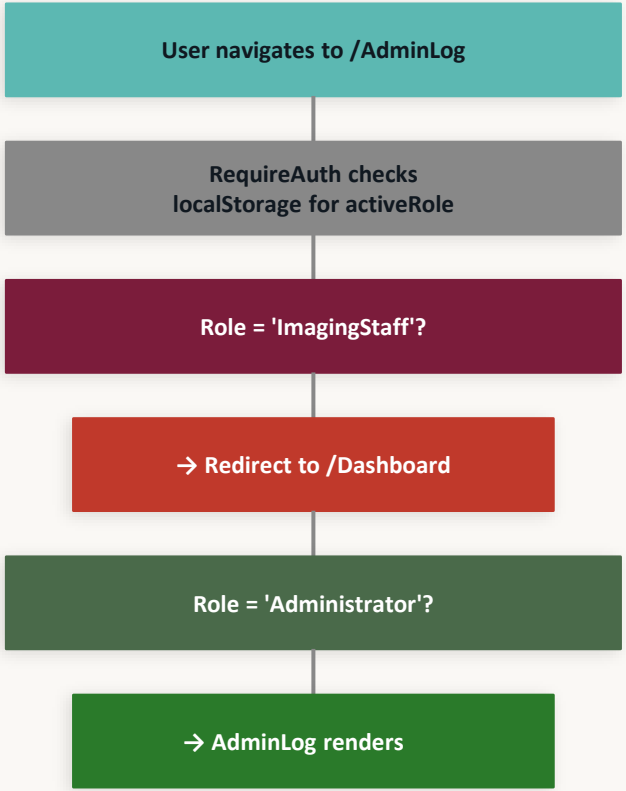
app/src/App.jsx (route structure)

```
<Routes>
  {/* Public */}
  <Route path="/" element={<Login/>} />

  {/*    Requires login */}
  <Route element={<RequireAuth/>}>
    <Route path="/Dashboard" element={<Dashboard/>} />

    {/*    Imaging + Admin only */}
    <Route element={<RequireAuth
      allowedRoles={['ImagingStaff', 'Administrator']} />}>
      <Route path="/TranscriptSubmission" ... />
    </>

    {/*    Admin only */}
    <Route element={<RequireAuth
      allowedRoles={['Administrator']} />}>
      <Route path="/AdminLog" ... />
      <Route path="/EmployeeList" ... />
    </>
  </>
</>
```



Note: Both frontend AND backend enforce roles independently — double-layered security.

Infrastructure & Deployment

Moving from localhost to a real FSU server environment was one of the most challenging parts of the project.

```
Production URL:        https://emt-transcripter.emt.fsu.edu
```

FSU ITS Server

Hosted on FSU EMT infrastructure. The Express backend runs as a Node.js process on an internal FSU server — not a public cloud.

Environment Config

.env file stores MONGO_URI, PORT, and secrets. devmode flag in App.jsx switches the API_BASE between localhost:5000 and the production URL.

CORS Configuration

Express CORS is configured with origin: true + credentials: true, allowing FSU internal domains to communicate while blocking external origins.

Static Build Serving

Vite builds the React app into /dist. The Express server (or a reverse proxy) serves the static files — the frontend and backend share one deployment.

MongoDB Atlas / Local

MONGO_URI connects to either a local MongoDB instance or Atlas cloud depending on environment. GridFS streams sit on top of the same connection.

SAML (Upcoming)

FSU SSO integration via SAML 2.0 will replace the manual login. Users will authenticate with their FSU credentials — roles resolved from directory attributes.

Before vs. After the Transcripter

BEFORE — Manual Process	AFTER — The Transcripter
✗ Transcripts received via email, drive links, or paper	✓ Structured submission form with required field validation
✗ Manually re-keyed into spreadsheets by staff	✓ Data stored instantly in MongoDB — zero re-keying
✗ No standard naming convention or file location	✓ Standardized document types and naming enforced by UI
✗ Anyone with shared drive access could view records	✓ Role-based access — only authorized staff can see records
✗ No way to know who changed what or when	✓ Every create, edit, and delete logged with user + timestamp
✗ Leadership had no visibility into volume or sources	✓ Analytics dashboard with live charts for leadership
✗ Duplicate entries and missing fields were common	✓ Duplicate detection and N/A defaulting built into schema
✗ Finding a specific transcript could take minutes	✓ Instant full-text search across all columns in milliseconds

Pitfalls & Challenges

What went wrong, what was hard, and what I had to figure out

Pitfalls & Challenges

Setting Up a Real Environment

Moving from localhost to a real FSU infrastructure environment exposed issues that never appeared locally — environment variables, port conflicts, CORS configurations, and network access rules all needed to be resolved.

Building for Stakeholders, Not Yourself

Requirements came from real users with real workflows. What made sense technically didn't always match how Admissions staff actually work. I had to learn to translate business logic into code — and re-translate it when requirements changed.

Institutions With Duplicate Names

Hundreds of colleges share similar or identical names. Early versions of the system had no way to distinguish them. The async-select approach with a deduplicated 54K dataset was the solution.

Changing Requirements Mid-Build

Fields were added, renamed, and removed across multiple meetings. Legacy fields like 'OnBase', 'Transient', and 'Both' had to be ripped out and replaced — requiring schema migrations and UI rewrites.

Authentication Before SAML

SAML integration with FSU SSO was not ready during initial development. A manual role registration system had to be built as a bridge — adding significant scope to the project.

ID Field Validation Complexity

EMPLID, Slate Apply ID, and Slate Connect ID all have different rules — 9 digits, mutually exclusive, destination-dependent. Getting this right on both frontend and backend without redundancy required a centralized validation module.

What I Learned

Technical Lessons

- ✓ Full-stack React + Node + MongoDB architecture
- ✓ GridFS for binary file storage in a NoSQL database
- ✓ Role-based access control with Express middleware
- ✓ Async search patterns for large datasets
- ✓ Soft delete and event sourcing / audit logging patterns
- ✓ Frontend validation + backend validation — why you need both

Professional Lessons

- ✓ Translating stakeholder requirements into technical specs
- ✓ Iterating on design through real user feedback
- ✓ Managing scope when requirements change mid-project
- ✓ Working within institutional IT constraints
- ✓ Writing code that others will maintain — not just code that works
- ✓ Presenting technical work to non-technical stakeholders

Key Skills Used

FRONTEND

React + Vite

85%

BACKEND

Node.js / Express

80%

DATABASE

MongoDB / Mongoose

78%

BACKEND

REST API Design

82%

FRONTEND

CSS / Responsive Design

80%

SECURITY

Role-Based Auth

75%

DATABASE

GridFS File Storage

70%

FULL-STACK

Data Validation

85%

TOOLING

Git / Version Control

88%

What Can Be Added Next

SAML / FSU SSO Integration

Full production authentication — users log in with FSU credentials, roles assigned automatically from directory attributes.

Email Notification System

Automated notifications when transcripts are submitted, edited, or flagged. Could alert the submitter and relevant staff.

CSV / Excel Export

Allow admins to export filtered transcript data to a spreadsheet for reporting, audits, or external processing.

Advanced Search & Filtering

Multi-field combined filters, saved filter presets, and date-range filtering across all columns — not just the Submitted count.

Mobile-Responsive Layout

The current UI is desktop-first. A responsive redesign would allow staff to look up and log transcripts from any device.

Auto-Population from PDF

The original vision included Python + pdfplumber to extract student name and institution directly from uploaded PDFs, reducing manual entry.

ADA Compliance & Accessibility

What's in place today — and the roadmap for full WCAG 2.1 AA compliance.

Currently in Place

Semantic Form Labels

All form inputs use <label> elements. Destinations group uses role="group" + aria-label so screen readers announce the fieldset correctly.

aria-hidden on Decorative Elements

Purely visual checkmark spans use aria-hidden="true" — screen readers skip them and avoid reading redundant information.

High-Contrast Color Palette

FSU Garnet (#7B1C3B) on white backgrounds maintains a contrast ratio above 4.5:1, meeting WCAG AA for normal text.

Responsive Layout Foundation

Flexbox and percentage-based widths provide a base for responsive scaling, reducing zoom-related usability barriers.

Planned Improvements

Full Keyboard Navigation

Tab order, focus trapping in modals, and keyboard-accessible dropdowns so every action can be completed without a mouse.

ARIA Live Regions

Status messages (save success, validation errors) will use aria-live="polite" so screen readers announce changes without interrupting flow.

Focus Indicators

Visible :focus outlines on all interactive elements — buttons, inputs, and table actions — to meet WCAG 2.4.7 (Focus Visible).

Accessible Charts (Analytics)

Recharts visuals will gain aria-label descriptions and data table fallbacks so chart data is consumable without sight.

Automated Accessibility Audits

Integrate axe-coreinto the build pipeline to catch regressions before each deployment.

Target standard: WCAG 2.1 Level AA · Internal tool used by FSU staff · Compliance supports employees with visual, motor, and cognitive disabilities

Live Demo

Demo URL

https://emt-transcripter.emt.fsu.edu/

Transcripter — FSU ITS Internal Transcript Management System

Acknowledgements

A project like this doesn't happen alone.

STAKEHOLDER & PROJECT LEAD

Bernica Rollison · FSU Office of Admissions

For defining requirements, providing real-world context, and trusting me with a project that genuinely matters to your team.

ITS DEPARTMENT

Humberto Messegeur · FSU Information Technology Services

For providing the development opportunity, infrastructure access, and support throughout the project lifecycle.

MENTORS & COLLEAGUES

Aleksandar Stavreski and William Callender · FSU ITS

For technical guidance, code reviews, and helping me navigate the realities of building software in a real institutional environment.

The

Transcripter

FSU ITS
2026

Thank You

Ammiel Bowen

Special Projects Developer · FSU ITS

2026 ITS RISE Showcase