INSY 4325 — Information Systems Analysis & Design

Final Project Report

All-Stars IT Solutions

Internal IT Ticketing System

Prepared by:
Paula Aziz
Felix Cherian
Freddy Alex
John Haddock
Eduardo Hernandez

Table of Contents

Abstract

This Report outlines the design and prototype of the All-Stars IT Solutions Internal IT Ticketing System as part of INSY 4325. It is a role-based web application developed by a group of 5 Students that will replace a Shared-Inbox Support Workflow with structured ticket queue to serve three Roles: Employee, Technician and Manager. It includes creating/monitoring/tracking tickets and assigning tickets, a public conversation thread and private technicians notes (internal only), approval process for sensitive changes, file attachments and in-app notifications.

Deliverables included requirements analysis, four core screens design/UI, UML Class Diagram, Entity Relationship Diagram in Third Normal Form, working prototype built with React 19, Vite 7, Material UI 7, and React Router 7. Data persisted within browser storage allowing for a fully functional prototype without having to provision server side hosting. The Data Layer has been isolated behind a service module allowing for real backend to be swapped out without having to change UI.

Project Overview

This project was developed by a five member team who created All-Stars IT Solutions, enrolled in INSY 4325-Information Systems Analysis and Design. The purpose of the assignment was to design and prototype an entire internal information system, from start to finish; such as requirements, User Interface (UI), Data Model, and a fully operational build. The team chose an internal IT Ticketing System as the problem they would attempt to solve.

There were three main reasons why we selected this project. First, ticketing is one of those problems that touches upon all aspects of the coursework. There are clearly defined Functional Requirements, multiple different User Roles, an interesting Data Model, and real design trade-offs. Second, the solution shape to the problem is very well understood which allowed our team to focus their efforts on the Analysis and Design of the Solution rather than creating a completely new Domain. Third, the Prototype will be a piece of working software and a piece of work we can use to advance our career.

The System is outlined in the Report as if it were being delivered to a Small In-House IT Team since delivering the System in such a way ties the Requirements, UI, and Data-Model Decisions back into real workflow.

Problem Domain

Context

The process works well as long as you're receiving fewer than a few dozen email requests per week. Beyond that, it will start to fail.

You don't have a defined "owner" of your inbox. There is no single point where there's an official list of all issues by their priority/impact. There are two different worlds; the one the end-user can see and the world the IT support team uses (such as internal notes). There isn't a history/log trail that tracks the changes made.

This project addresses this exact problem space. It is designed to replace an unstructured "in-box" with a structured "ticket queue," but do so without requiring the user to pay or learn how to use a full-service commercial IT Service Management Platform.

Stakeholders

Role	Interaction & Success Criteria
Employee	Files tickets, tracks status, replies in the conversation thread. Wants quick acknowledgement and visibility into who is working on the issue.
Technician	Works the queue, replies, adds internal notes, requests manager actions. Wants a clear queue and the ability to keep working notes invisible to the requester.
IT Manager	Owns the queue. Assigns work, approves sensitive changes, monitors load. Wants visibility and an approval gate for changes that require oversight.
Project team (us)	The five students who designed and built the system. Responsible for analysis, UI, data model, and the working prototype.

Current Issues

- Tickets are lost or delayed because the shared inbox does not enforce ownership.
- No canonical capture of priority, impact, or category. There are critical issues compete with low-impact requests.
- Internal notes between technicians live in side-channel chat, so the next person to pick up a ticket has no context.
- No formal approval step for sensitive changes; sign-off happens verbally and leaves no trace.
- Reporting depends on a spreadsheet that drifts out of date within days.
- No audit trail; reconstructing the history of a ticket means searching scattered emails and chat logs.

Scope and Constraints

In Scope	Out of Scope
Ticket lifecycle (creation, assignment, status, resolution, closure); employee - technician conversation; technician internal notes; manager approval workflow; in-app notifications; role-based access control across three roles; file attachments; filtering and sorting; audit-friendly history of state changes.	Asset/inventory management; change-management board; problem & incident; knowledge base; email-to-ticket; SSO; native mobile apps; multi-tenant deployment; multi-language UI.

Constraints: the project is delivered by a small student team in one semester, with no budget for commercial licenses or paid hosting; the application must run in any modern browser; internal notes must never be visible to the requesting employee; status, priority, and approval changes must be timestamped; for the academic deliverable the system must run on a single laptop without server provisioning.

Objectives

The goals below have been created without referencing a specific technology; therefore these will continue to apply in the future when the delivery method is modified.

1. Replace the use of the common inbox for requests with a simple role aware web interface which creates all internal support requests as structured tickets.

2. Ensure ownership is clearly defined; each ticket has no more than one designated technician and the queue should reflect who owns it.

3. Audit Trail - record the creation date/time of the ticket, assignment of technician(s), status

4. Only visible to employees based upon their roles. Employees submitting a request should never see internal comments made about them or their submission.

5. Introduce an additional approval process for employee submitted requests requiring manager approval prior to making any changes to a request.

6. Simplify time from first response; surface new submissions at the top of the queue; send notifications to assigned technicians of all events related to a new submission.

7. Provide the basis for extending this project into the actual server backed solution without creating any need to redo the user interfaces.

Functional Requirements

The team analyzed and categorized all requirements into the six capability area requirements. Every requirement has a MoSCoW priority level, and has an Acceptance Criteria that can be unambiguously verified. Business Rules and Non-Functional Requirements cross cutting all of the other requirements will also be listed in the table below.

Access Control

Description	Priority	Acceptance Criterion
A user signs in with an email and is taken to a role-aware shell.	M	Login dispatches the session, sets the active role, and routes to My Tickets.
The system associates the active session with one of three roles: Employee, Technician, or Manager.	M	The active role is available on every page and gates every role-restricted control.
Employees cannot view, open, or modify any ticket they did not create.	M	Only the requester's tickets appear in My Tickets; direct URL access to another user's ticket is blocked.
Internal Notes are accessible only to Technicians and Managers.	M	The Internal Notes menu item is hidden from Employees; the route is gated by role.
A user can sign out, ending the active session.	M	Logout clears the session and returns the user to the login screen.

Ticket Submission

Description	Priority	Acceptance Criterion
An Employee can file a ticket with title, category, priority, impact, and description.	M	On submit a new ticket appears with status Open and a generated ID; a notification is dispatched to the manager.
The system validates required fields before accepting the ticket.	M	Submission is blocked until all required fields carry valid values; missing fields show inline errors.
An Employee can attach images to a ticket at submission.	S	Attachments are persisted with the ticket and visible on the ticket detail page.

Tracking and Status

Description	Priority	Acceptance Criterion
Every user sees a My Tickets list scoped to their role.	M	Employees see only their own tickets; Technicians see assigned tickets; Managers see every ticket.
The list supports search by title and filtering by status, priority, and scope.	M	Changing any filter narrows the list immediately; the active filters are visible in the header bar.

Description	Priority	Acceptance Criterion
Stat cards at the top show counts for Active, In Progress, Resolved, and Closed.	S	Counts update with the current filter and serve as one-click filters themselves.
Allowed status values are Open, In Progress, Resolved, and Closed.	M	Only these four values are stored or displayed; no other state appears in the UI.
Every status, priority, and impact change is recorded with actor and timestamp.	M	The conversation thread shows a system entry for each transition with the acting user's identity.

Assignment and Priority

Description	Priority	Acceptance Criterion
A Manager can assign a ticket to a Technician via an inline Select on the My Tickets row.	M	On change, the ticket's assignedTo updates and the new technician receives a "ticket-assigned" notification.
A Manager can reassign a ticket from the Actions menu.	M	The Reassign Ticket dialog opens; choosing a new technician updates the ticket and notifies the new assignee.
A Manager can change a ticket's priority directly.	M	Change Priority dialog applies the new value immediately; the change is timestamped and recorded.
A Manager can change a ticket's impact directly.	M	Change Impact dialog applies the new value; the change is recorded.
A Manager can change a ticket's status, including reopen.	M	Change Status dialog persists the new status; closed tickets show "Reopen Ticket" instead of "Close Ticket".
A Technician can request a priority change on a ticket they own.	M	The request is created in pending state; priority does not change until the Manager approves.
A Manager can delete a ticket.	C	Delete removes the ticket; the action is restricted to the Manager role.

Communication

Description	Priority	Acceptance Criterion
Employee, Technician, and Manager can post messages to the public conversation thread.	M	A new message appears immediately, labeled with author email, role, and timestamp.
A user can attach images to a conversation message.	M	Attachments render under the message and inherit its public visibility.
Technicians and Managers can add Internal Notes on a ticket.	M	Notes appear in the Internal Notes page (which opens in a new tab); they are never shown to the Employee.
A Technician can request a generic manager action with a free-text description.	S	A pending manager-request appears in the Manager's Pending Requests panel.

Description	Priority	Acceptance Criterion
In-app notifications are dispatched on every relevant event.	M	New ticket, assignment, replies, manager-requests, approvals, and declines each generate the appropriate notification type.

Resolution and Reporting

Description	Priority	Acceptance Criterion
An Employee can mark their own ticket as Resolved.	M	Status moves to Resolved with the current timestamp; the change is captured in the audit thread.
A Technician can request closure of a ticket.	S	The system auto-creates the closure request, posts a confirmation message in the thread, and notifies the manager.
A Manager can approve or decline any pending technician request.	M	On approve the change is applied; on decline no change occurs. The technician is notified of the decision.
Stat cards on My Tickets give a real-time view of workload by status.	S	The four counts (Active / In Progress / Resolved / Closed) reflect the live state of the queue.

Business Rules

1. Single owner: At any given time, there will be no more than one technician assigned to a ticket. Reassigning the technician logs that an assignment was made prior to that assignment.

2. Internal-note visibility: Only technicians and managers can see internal notes along with any files attached to them.

3. Approval gate: No changes will occur on a technician's request until manager approval occurs.

4. Audit trail: Any status, priority, impact, assignments or approvals of a ticket are logged and include who performed the action and when the action occurred.

5. Status set: Allowed status values are Open, In Progress, Resolved, and Closed. Resolved sets resolutionDate; Closed is terminal except by Manager reopen.

6. Controlled vocabularies: Category, priority, impact, and status are constrained to defined sets, not free text.

Non-Functional Requirements

1. Performance: List views & ticket detail renders quickly on prototype with realistic data volumes; image attachments are compressed client-side for local storage fit.

2. Availability: Browser-storage prototype available when static assets load; production target: 99.5% uptime during business hours.

3. Security: Role-based access control implemented at route level, component level, and data store layer; passwords not exposed to client.

4. Usability: keyboard reachable, status conveyed via both text and color.

5. **Scalability:** Data layer encapsulated in service module; persistence backend can be swapped out without changing UI.

UI Design

The user interface of this application has been developed using Material UI for its accessibility, consistency and familiarity in appearance. The overall design was done using a limited number of colors (one primary), a very narrow typographic range, an 8 pixel wide grid for all space elements and a consistent set of status chips that appear at the end of every list.

Additionally, the shell of the application changes based upon role. Employees have a left side bar that provides access to their dashboard and allows them to create new tickets. While managers and technicians do not have the side bar when working off the "my tickets" screen it makes it easier to focus while they are working through their queue of tickets.

Core Screens

My Tickets

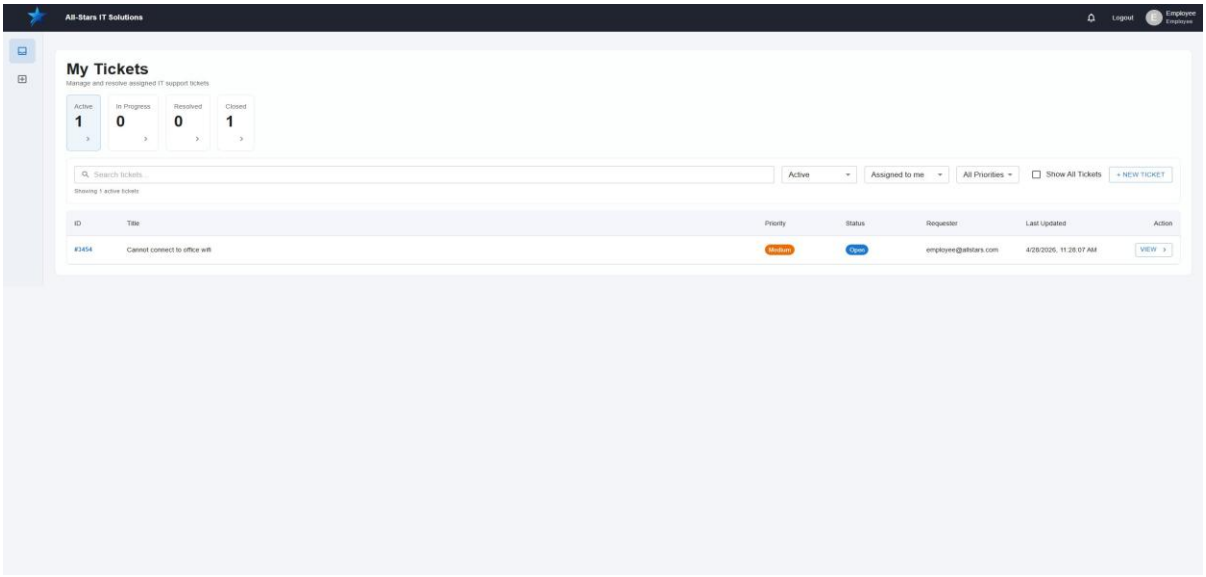

Figure 1. My Tickets — Employee view.

My Tickets is used by everyone involved with the system. Every row includes information about each ticket such as id, name, priority level, status, who requested it and when it was most recently updated. There are four stat cards located above the data rows that can be clicked once to provide a filtered view of all active, all in progress, all resolved or all closed tickets.

Create Ticket

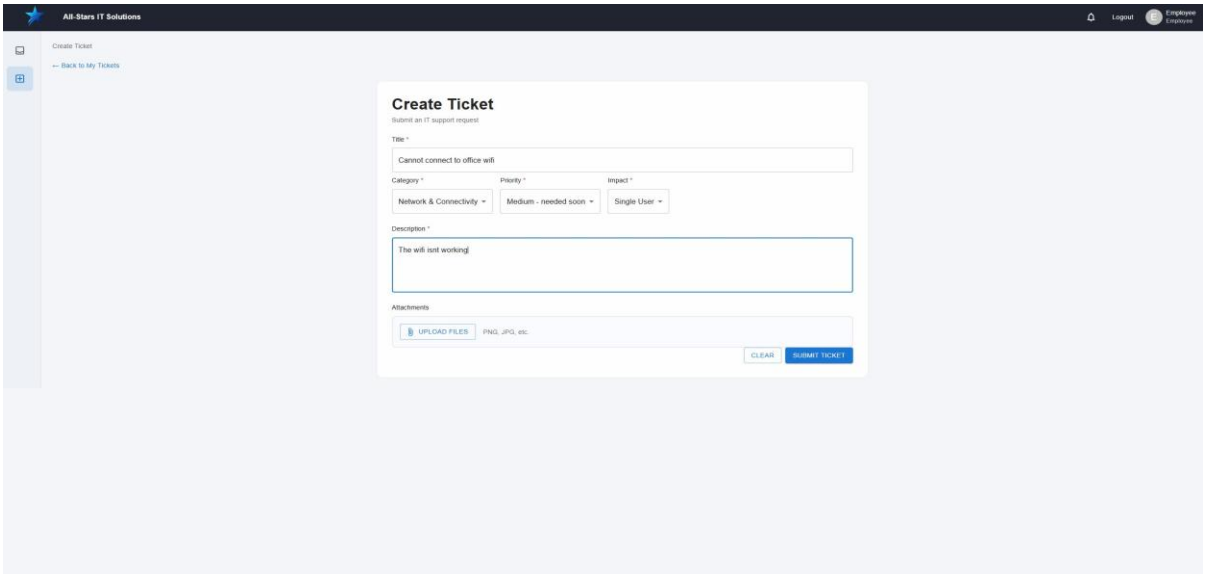

Figure 2. Create Ticket form (Employee). Required fields are marked with a red asterisk.

Ticket submission is a single screen activity. It requires users to enter a title for their issue, choose a Category from the list of categories available (Network & Connectivity, Hardware, Software, Accounts & Access, Other), a Priority level and an Impact Level, as well as a Description of the problem they are experiencing. Validation will be performed on all required fields, while submission validation will be provided in-line. Once submitted successfully, the form field will be cleared, a Green Success Alert box will appear and a "New-Ticket" notification will be sent to the Manager.

Ticket Detail

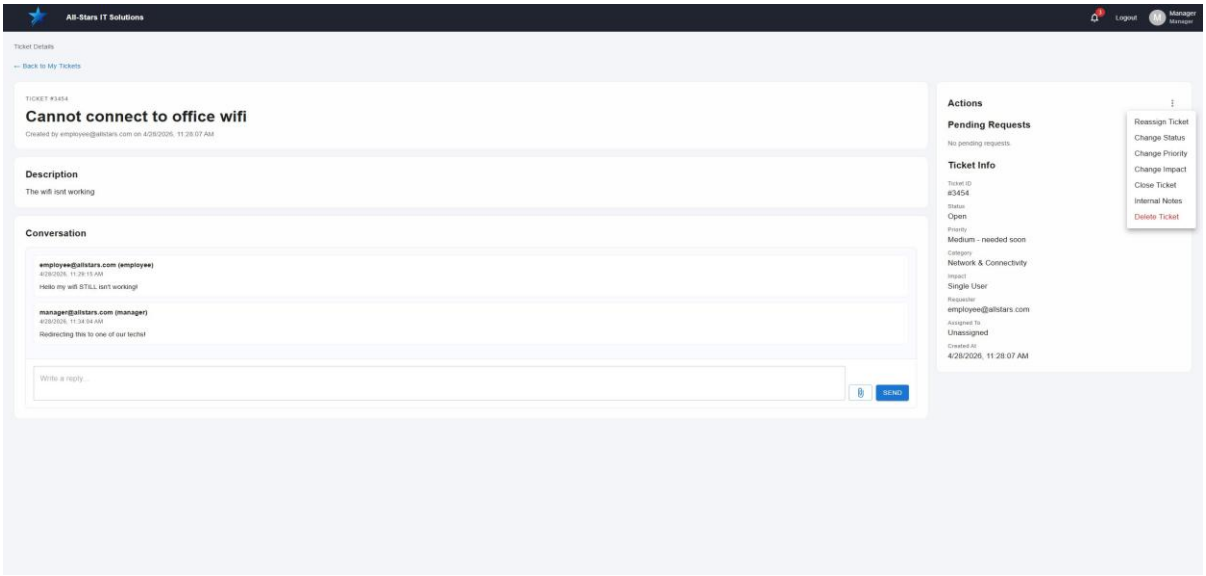

Figure 3. Ticket Detail — Manager view, with the Actions menu open showing the seven manager-only operations.

Ticket Detail is the central part of our application. It presents information about tickets as a two column grid. A header Card contains the Ticket Id, Title and Creation Metadata. Description Card describes what this ticket is about. The Conversation Panel shows all messages related to this ticket. In the Right Rail are placed the Actions, for employees "Mark Resolved", for technicians "Request Priority Change", "Request

Closure", "Request Manager Action" and "Internal Notes"; for managers there are "Reassign", "Change Status/Priority/Impact", "Close (or Reopen)", "Internal Notes" and "Delete" (in red). For every manager's action, it will be opened in a focused Dialog.

Internal Notes

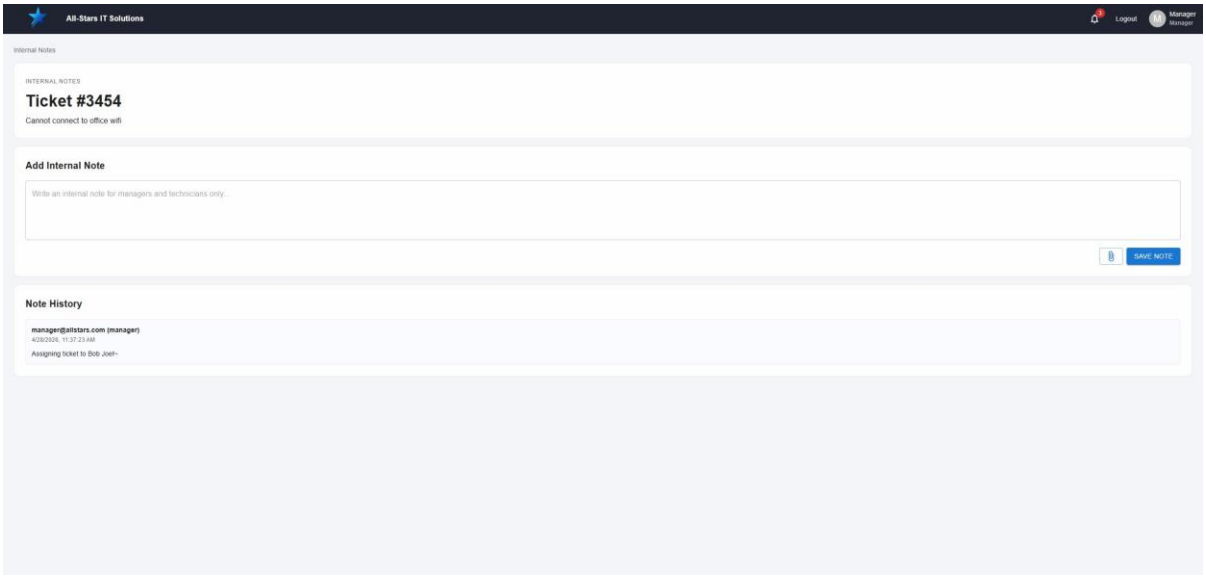

Figure 4. Internal Notes page (Manager view). Opened in a new browser tab; never shown to the Employee.

Internal Notes opens in a separate browser window or tab when selected via the action menu on the internal notes page. This page includes the Ticket Header at the top of the page, a form for adding an internal note along with the ability to attach files, and a chronological listing of all previous internal notes for the ticket. The Internal Notes route is also hidden from employees, and enforced by the application server side which ensures business rule 2 (internal-notes visible) is enforced at both the user interface layer and access control layers.

States, Validation and Feedback

All screens handle four distinct states. An empty state does not appear as a blank. Instead, the 'My Tickets' list will show you a copy of one of three messages depending upon your role (for example, "You have not submitted any tickets yet". Validation occurs within each field's border as well as in red helper text. In addition, the submit button is always disabled until all fields are validated. At the page level, errors and success are handled using snackbar messages located at the top of the page that auto-close but may be manually closed.

Usability and Accessibility

Every interactive feature can be accessed using tabbing with an obvious visual indicator of which element currently has focus; each field in your forms will have a corresponding label to tell screen readers what type of field it is; all chips convey their status via both text and color.

All pages use the same primary color, same chip styles, same spacing (eight pixels) for the entire grid to allow users that learn how to operate one screen easily navigate the others.

Alignment with Requirements

Screen	Notes
Login	Demo accounts seeded for the academic build.
Create Ticket	Inline validation; success snackbar; dispatches "new-ticket" notification.
My Tickets	Role-aware filters and stat cards; manager Select on each row.
Ticket Detail	Hosts the entire lifecycle; role-based controls hidden from unauthorized users.
Internal Notes	Separate tab; gated by role; never reachable from the Employee UI.

Class Diagram

The class diagram represents the static structural composition of the domain. The team selected a "User + Role" look up model instead of a large inheritance hierarchy to accommodate the dynamic nature of an organization's roles since members may transition from a contributor role to a manager role etc. A simple look up table will provide for clean representation of this dynamic relationship. Similarly, there were four small lookups (Categories, Priorities, Impact, Statuses) referenced on each ticket providing for controlled vocabulary constraints at the data level rather than simply through the user interface.

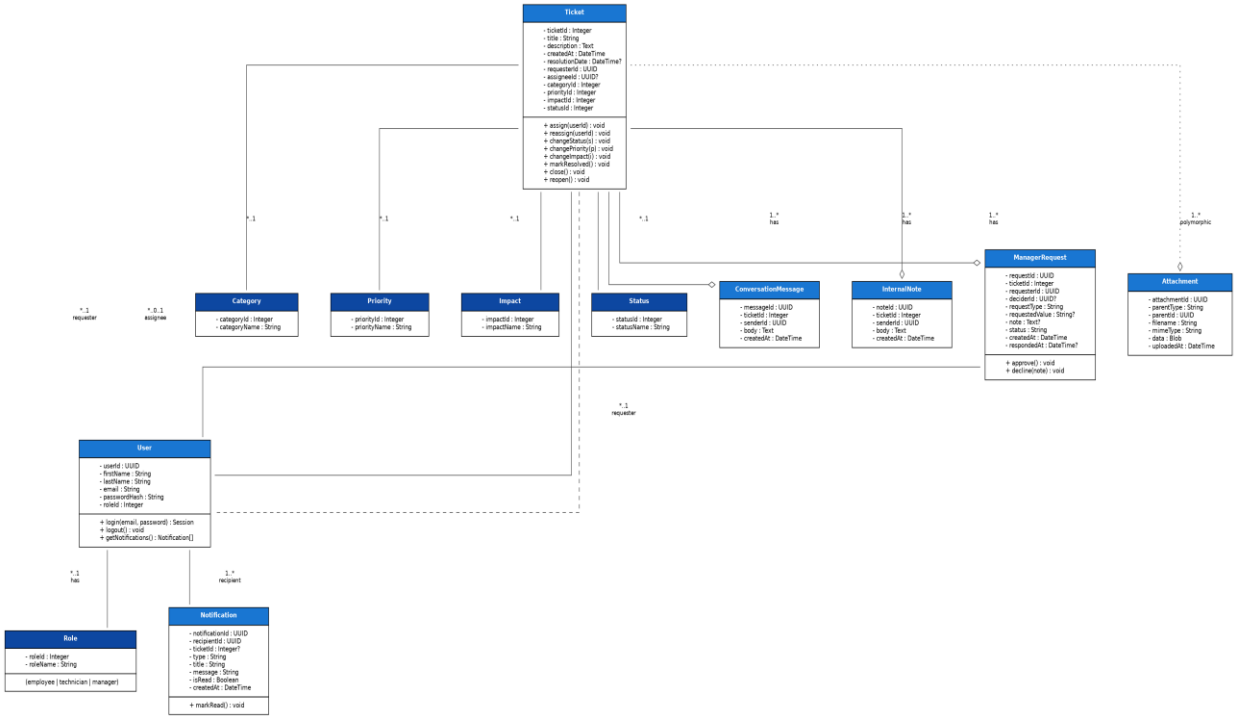

Figure 5. UML class diagram.

Classes and Attributes

Class	Key Attributes
Role	roleId, roleName ∈ {Employee, Technician, Manager}
User	userId, firstName, lastName, email, passwordHash, roleId (FK Role)
Category / Priority / Impact / Status	id, label — small lookup tables for the four controlled vocabularies referenced by Ticket.
Ticket	ticketId (auto, starts at 3452), title, description, requesterId (FK User), assignedToId (FK User, nullable), categoryId, priorityId, impactId, statusId, createdAt, resolutionDate
ConversationMessage	messageId, ticketId (FK), senderId (FK User), senderRole, text, createdAt
InternalNote	noteId, ticketId (FK), senderId (FK User), senderRole ∈ {tech, manager}, text, createdAt

Class	Key Attributes
ManagerRequest	requestId, ticketId (FK), requestedById (FK User, tech), type ∈ {change-priority, close-ticket, reassign, change-status, change-impact, reopen-ticket, other}, requestedValue, note, status ∈ {pending, approved, declined}, managerResponseNote, createdAt, respondedAt
Attachment	name, mimeType, data (compressed JPEG, ≤1200×1200) — embedded inside its parent record
Notification	notificationId, recipientEmail (FK User), type ∈ {new-ticket, ticket-assigned, manager-request, request-approved, request-declined, conversation-message}, title, message, ticketId, isRead, createdAt

Relationships, Multiplicities, and Inheritance

A User can have one Role; A Role can have many Users (1..*:1). A ticket can have no more than one requestor (1..*:1) and zero or one assignee (0..*: 0..1); a ticket also refers to all of the four lookup table values. A ticket contains many Conversations Messages, Internal Notes and Manager Requests (1:*). An Attachment exists within a conversation message or an internal note that has it as its owner (composition). Each notification is sent to one user Recipient.

No Class Hierarchy was created by the group based on the relationships between classes. If three subclasses of an abstract User were used (Employee, Technician, Manager) then you would be required to change a User's class type to change their role, this is very inconvenient when a User's roles shift in a real organization. The User + Role design allows for role change, keeps the DB schema simple, and puts all role-based logic into one place.

Methods

- **User:** login(email, password), logout(), getRole(), getNotifications().
- **Ticket:** create(...), assign(tech), updateStatus(s), markResolved(), delete().
- **ManagerRequest:** approve(mgr, note?), decline(mgr, note?). Approval applies the requested change and notifies the technician.

Database Diagram (ERD)

The persistent state as implemented is represented in twelve tables. UUIDs are used to identify users and child records; tickets have an integer primary key (the auto-increment IDs visible in the UI). Referential integrity is enforced through foreign keys; restricting the controlled vocabularies are the four small lookup tables (categories, priorities, impacts, statuses).

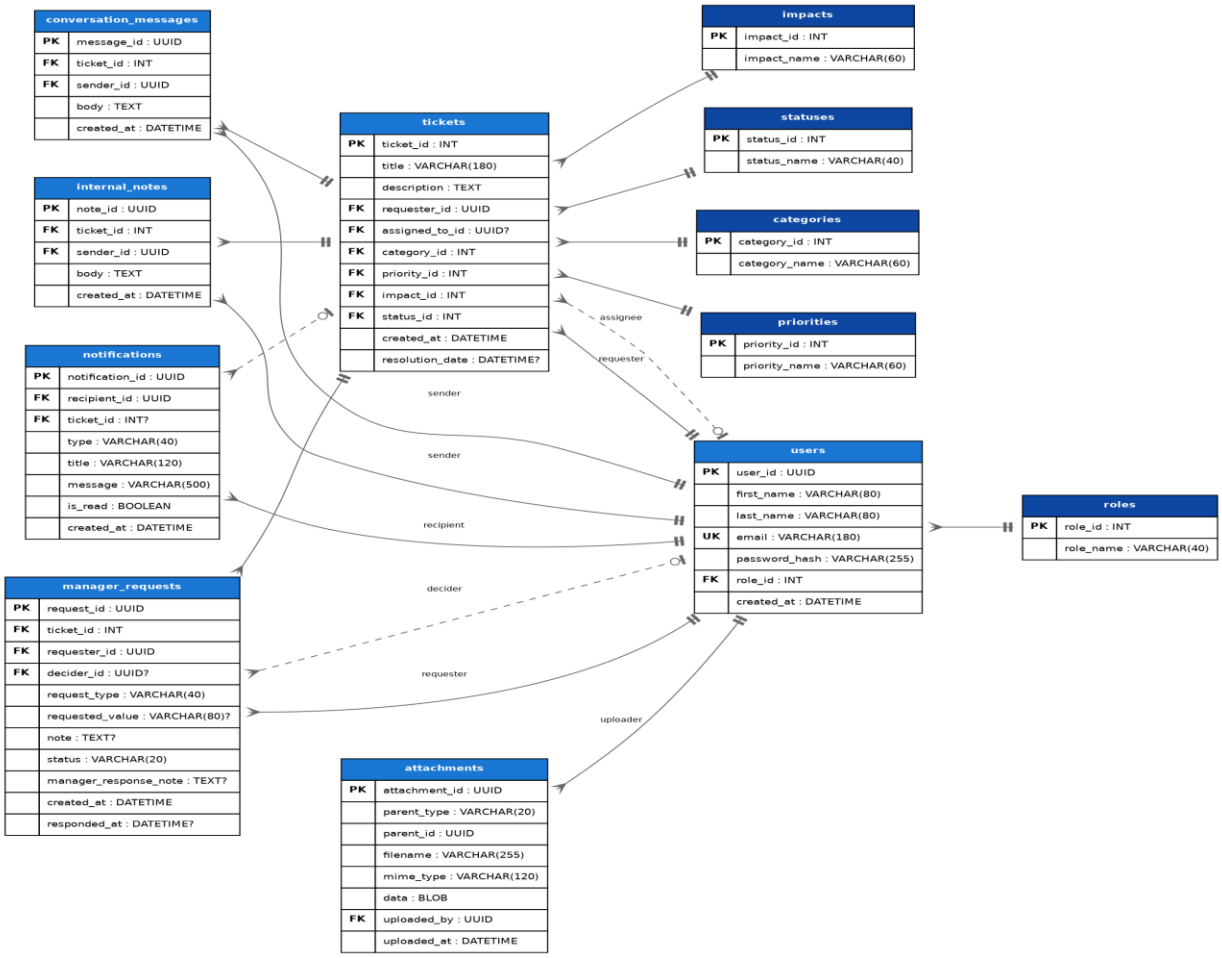

Figure 6. Entity-relationship diagram.

Tables and Columns

Table	Primary & Foreign Keys	Other Columns
roles	PK role_id	role_name (Employee, Technician, Manager)
users	PK user_id (UUID); FK role_id → roles; UK email	first_name, last_name, password_hash, created_at
categories	PK category_id	category_name (5 rows)
priorities	PK priority_id	priority_name (4 rows)

Table	Primary & Foreign Keys	Other Columns
impacts	PK impact_id	impact_name (4 rows)
statuses	PK status_id	status_name (Open, In Progress, Resolved, Closed)
tickets	PK ticket_id (INT); FK requester_id, assigned_to_id, category_id, priority_id, impact_id, status_id	title, description, created_at, resolution_date
conversation_messages	PK message_id; FK ticket_id, sender_id	body, created_at
internal_notes	PK note_id; FK ticket_id, sender_id	body, created_at — sender must be tech or manager
manager_requests	PK request_id; FK ticket_id, requester_id, decider_id	request_type, requested_value, note, status, manager_response_note, created_at, responded_at
attachments	PK attachment_id; FK uploaded_by	parent_type ∈ {message, note}, parent_id, filename, mime_type, data, uploaded_at
notifications	PK notification_id; FK recipient_id, ticket_id (nullable)	type, title, message, is_read, created_at

Cardinalities

- roles → users: 1 : N (a role has many users; a user has exactly one role).
- users → tickets (requester): 1 : N (a user files many tickets; a ticket has one requester).
- users → tickets (assignee): 1 : 0..N (a tech is assigned to many tickets; a ticket has 0 or 1 assignee).
- tickets → conversation_messages / internal_notes / manager_requests: 1 : N each.
- users → notifications: 1 : N (notifications are addressed to one recipient).

Normalization and Constraints

This design is in Third Normal Form. Each table has one column as its Primary Key (1NF); because each of the Primary Keys is a single column it can never be possible to have Partial Dependency (2NF); all four controlled vocabulary lists are separated from Tickets in a separate table (lookup table) so that the label for each list does not duplicate inside Tickets and therefore cannot exist as Transitive Dependency (3NF). The Constraints Layer includes: a UNIQUE constraint on email with case sensitivity ignored in the users table; an ON DELETE RESTRICT constraint on references from tickets to users; default values for created_at (the time when the ticket was created), status (status = Open), priority (priority = Medium), and impact (impact = Single User); and index recommendations for assigned_to_id and status_id in tickets, requester_id and status_id in tickets, and recipient_id and is_read in notifications to help with the queue query, my-tickets query, and unread-notifications query respectively.

Actual Implementation

Our current build is a single page application using React. The team has chosen a technology stack that is currently being used by many industries, it's easy to rapidly develop with, and very documented; therefore the majority of our project time was spent on the analysis, design, and UX aspects and not setting up the environment.

Layer	Choice	Rationale
UI library	React 19	Component model, hooks, current release.
Build tool	Vite 7	Fast dev server; minimal configuration.
Components	Material UI 7 + @emotion	Accessible primitives; consistent visual language.
Routing	React Router 7	Declarative client-side routing with nested routes.
Persistence (demo)	localStorage + sessionStorage	sessionStorage is per-tab so each browser tab can be a different role for side-by-side demonstration.

Application Behavior

Upon successful log-in, the email address and role of the signed in user are saved to sessionStorage to enable a single browser window to have multiple tabs open at one time; each tab will be logged in as a different demo account. Tickets, Manager Requests, and Notifications are stored in localStorage under static keys (allstars_tickets, allstars_notifications). All image attachments are also compressed prior to storing them via a pipeline built into the canvas object (the maximum size we compress images to is 1200x1200 pixels), this allows us to make use of localStorage in an educational prototype. All store calls are centralize within a service module so if we choose to swap out the backend to use a rest api, we will not need to touch any of the UI components.

Implemented Features

- Login with three demo accounts; tab-scoped sessions for side-by-side role demonstration.
- Create Ticket form with title, category, priority, impact, description, and image attachments.
- My Tickets list with stat cards, search, and filters; role-aware data scope; manager inline-assign Select.
- Ticket Detail with public conversation, role-aware action menu, Pending Requests / Manager Responses panel, and Ticket Info.
- Technician actions: Request Priority Change (dialog), Request Closure (auto-narrating), Request Manager Action, Internal Notes.
- Manager actions: Reassign, Change Status, Change Priority, Change Impact, Close/Reopen Ticket, Internal Notes, Delete.
- Internal Notes as a separate tabbed page, never reachable from the Employee UI.
- Six notification types with an unread badge in the header bar.

Known Limitations

- **No real authentication:** Passwords are accepted without verification because there is no server. Resolved in Phase 1 (real backend) and Phase 2 (SSO).
- **No multi-user persistence:** Each browser holds its own data. Resolved in Phase 1.
- **Attachment storage cap:** Compressed images are still base64-encoded in localStorage, capped near 5 MB total. Production routes attachments to object storage.
- **In-app notifications only:** Email delivery is in Phase 3.

Conclusions and Recommendations

The team's objective is to implement a purpose built web-based support system rather than relying on a shared inbox. In addition, they completed a prototype that meets each of the requirements. Notably three things were found. The single most impactful design decision was implementing role aware access for viewing Internal Notes. This results in one policy that resolves all related issues associated with confidentiality. The approval process provides a documented record of verbal approval by a manager when approving requests by technicians without adding additional steps. Also, controlling vocabulary (status, priority, impact, request type) will prevent the "free-form" entry of data into reports from creating degradation in report quality over time.

The prototype was successful and our team believes this systems should be the standard for internal technical support; however there are five key items that need to be added to make it suitable for production.

1. Stand up a real backend: Replace browser storage with a Node.js or Python service over PostgreSQL using the schema in Section 6.

2. Adopt single sign-on: Integrate with Microsoft Entra ID or Google Workspace.

3. Add a specific audit-events table: Keep a running log of changes that shows who made each change and what it was before and after.

4. Implement tracking and reporting: Set rules based on priority and impact, send alerts if something is late, and show a simple report (what's open, who's working on it, average fix time, and how long things have been waiting).

5. Move attachments to object storage: Store file info in the database, but keep the actual files in S3-style storage, and use secure links to access them.

High-Level Implementation Plan

The production deployment of this application will occur in four sequential phases. Each phase will produce an incremental version of the application that may be used in conjunction with the current email process until cutover occurs.

Phase	Duration	Deliverables and Success Criteria
1 — Backend & DB	6–8 weeks	Replace browser storage with a real backend server and a proper database, then host it online. **Success:** the system still does everything from Section 4, and your tickets show up no matter which computer or browser you log in from.
2 — Real Auth & SSO	3–4 weeks	Argon2 password hashing or SAML/OIDC against the corporate IdP; just-in-time provisioning; role assignment via IdP groups. **Success**: no accounts are created by hand; deactivating in the IdP removes access at next login.
3 — Deadlines, Reports & Email	6–8 weeks	Add deadline rules based on how urgent a ticket is, send a warning if a ticket is taking too long, and add basic reports (open tickets, who's working on what, average fix time). Also add email notifications. **Success:** the IT Manager can answer common questions ("how many tickets are open?", "who's busiest?") right inside the app, and emails arrive within a minute of an event.
4 — Mobile & PWA	4–6 weeks	Make the site work well on phones, let users "install" it to their home screen like an app, and add push notifications. **Success:** a technician can handle tickets from their phone, and the app feels like a native mobile app even though it runs in the browser.

Cutover and Risk

There are two main risks when switching from the old email system to the new one. The first is what to do with old emails and spreadsheet records that the team wants to keep. Rather than trying to rebuild full email conversations into proper tickets (which isn't worth the effort), we would import them as read-only "legacy" tickets marked as Closed, just so the history is searchable. The second risk is that people will keep using email out of habit. Technicians will reply directly to senders for the first few weeks even after the new system is live. To handle this, we would run a small pilot with one Tier-1 technician on the new system before the full switch, and give everyone a one-page cheat sheet showing how to do common tasks. Because all the data lives in a normal database, moving to a different platform later would be straightforward if the company ever decides to.

Summary of Project

This 5 member student group created and developed a fully functional version of an internal IT help desk ticketing system as part of their assignment for INSY 4325. A shared-inbox process is commonly used in small IT departments until they grow beyond this method. The group found that there are six common issues experienced when transitioning from a shared inbox process. Each issue was addressed through a role-based system.

Functional Requirements were determined and written based on the analysis of the six capability areas. Each area had a MoSCoW prioritization assigned along with a verifiable acceptance criterion. In addition, the team also developed six Business Rules and five Non-Functional Requirements. The Prototype's UI was centered around four main pages (Create Ticket, My Tickets, Ticket Details, Internal Notes). Material UI was selected for the UI to ensure consistent aesthetics throughout the application and to enhance accessibility.

A UML Class Diagram was created to model the Domain using a User + Role Lookup Design instead of Subclasses. A relational database schema composed of twelve tables was also created in Third Normal Form.

The Prototype was developed using the React Framework, Vite, Material UI, and React Router. The Data Layer persisted data to Browser Storage. Authentication was accomplished using Session-Scope Storage. Therefore, one Laptop could support Three Roles running simultaneously in Three Browsers Tabs. All requirements identified within scope were met. All Business Rules will be enforced at the Data Store Level versus just being applied at the User Interface Level. It is recommended to consider adopting the System in Production according to the Four Phase Plan.

References

Axelos. (2019). ITIL Foundation: ITIL 4 Edition. The Stationery Office.

Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387.

Dennis, A., Wixom, B. H., & Roth, R. M. (2021). Systems Analysis and Design (7th ed.). Wiley.

Material UI. (2026). Component documentation. https://mui.com/material-ui

OMG. (2017). Unified Modeling Language Specification, version 2.5.1. Object Management Group.

React. (2026). React documentation. Meta Platforms, Inc. https://react.dev

W3C. (2018). Web Content Accessibility Guidelines (WCAG) 2.1.

Appendices

Appendix A — Demo Accounts

Email	Role	Use
employee@allstars.com	Employee	File a ticket; track its progress; reply in the conversation thread.
bob.joe@allstars.com	Technician	Work the queue; reply, add internal notes, raise a manager request.
manager@allstars.com	Manager	Assign tickets; change priority/impact/status; approve or decline a request; delete a ticket.

Appendix B — Running the Demo Locally

Requires Node.js 18 or later.

1. npm install

2. npm run dev

Open the URL printed by Vite (localhost). Open additional browser tabs and sign in with a different account in each tab to see the three-role workflow side by side; sessionStorage scopes each session to its tab.