ResolveDesk

Development Progress Record

Completed Phases 1 and 2

Help Desk Ticket System

Project owner	Katelyn Dempsey
Current status	Phase 1 complete; Phase 2 complete; Phase 3 next
Application type	Help desk ticketing system
Core stack	C#, .NET 10, ASP.NET Core, Entity Framework Core, PostgreSQL, Docker, ASP.NET Core Identity, JWT, Git
Purpose	Document the work completed and verified through the first two development phases.

1. Project Overview

ResolveDesk is a help desk ticketing application designed to reflect realistic internal IT support workflows. The first two completed phases established the backend architecture, database foundation, domain model, source-control history, user identity system, role-based authorization, and JWT authentication required for later ticket workflows, frontend development, testing, and deployment.

2. Progress at a Glance

Phase	Scope	Status
1	Foundation and initial data model	**Complete**
2	Identity, authentication, and authorization	**Complete**
3	Ticket API and business logic	**Next**

3. Phase 1 - Backend Foundation

Phase 1 established a functioning local backend and database environment. The complete solution builds successfully, the ASP.NET Core API runs locally, PostgreSQL runs in Docker, the database health endpoint reports status, and the initial help desk schema is represented through Entity Framework Core migrations.

Development Environment and Tooling

- Installed and verified the .NET 10 SDK, Git for Windows, Docker Desktop, Docker Compose, WSL 2, Ubuntu, and Visual Studio Code tooling.
- Diagnosed disabled hardware virtualization and enabled AMD SVM Mode in BIOS/UEFI so Docker could run.
- Configured the repository for C# development, Docker workflows, command-line builds, and local source control.

Solution Architecture

- Created a layered .NET solution with separate API, Application, Domain, Infrastructure, and Tests projects.
- Established project references and dependency direction to separate HTTP hosting, domain models, database concerns, and future business logic.
- Created a controller-based ASP.NET Core Web API and verified successful local execution and full-solution builds.

Database and Entity Framework Core

- Configured PostgreSQL 17 as a persistent Docker Compose service with environment-based settings, port mapping, and a named volume.
- Configured Entity Framework Core with the Npgsql PostgreSQL provider and registered the DbContext through dependency injection.
- Created and applied migrations for tickets, categories, comments, ticket history, and attachment metadata.
- Verified the resulting PostgreSQL tables directly using psql inside the running container.

Domain Model Created

Model	Implemented purpose
Ticket	Core ticket details, status, priority, category, comments, history, attachments, and lifecycle timestamps.
Category	Reusable ticket classification with active status and ticket relationships.
TicketComment	Ticket discussions, internal-note distinction, timestamps, and attachment relationships.
TicketHistory	Audit-history records with action and before/after values.
TicketAttachment	File metadata linked to a ticket and optionally to a comment.
TicketStatus / Priority	Enums for realistic workflow states and Low, Medium, High, and Critical priority levels.

Security and Reliability Practices

- Stored Docker database settings in a local .env file excluded from Git.
- Stored the API connection string using ASP.NET Core User Secrets instead of tracked project files.
- Added fail-fast configuration validation for a missing connection string.
- Registered an Entity Framework Core health check and mapped a /health endpoint.

Troubleshooting Completed

- Resolved Docker startup failure caused by disabled hardware virtualization.
- Installed WSL 2 and Ubuntu to satisfy Docker Desktop prerequisites.

- Updated a Microsoft.OpenApi package after identifying a security warning.
- Installed and configured Entity Framework tooling when migration commands could not run.
- Diagnosed an Unhealthy database status by checking API output and Docker Compose state, restarting PostgreSQL, and confirming the endpoint returned Healthy.

Git Checkpoints

- Initialized the local Git repository and renamed the default branch to main.
- Created and maintained .gitignore rules, including confirmation that sensitive .env data was excluded.
- Created meaningful commits for the backend solution, PostgreSQL foundation, and completed Phase 1 data model.
- Confirmed a clean working tree after the final Phase 1 checkpoint.

4. Phase 2 - Identity, Authentication, and Authorization

Phase 2 added the user and access-control foundation required for a multi-role help desk application. The API can now register and authenticate users, issue and validate JWTs, protect endpoints, and distinguish between employee, technician, and administrator access.

Area	Completed work
Identity	Added ASP.NET Core Identity users and roles.
Password security	Implemented secure password hashing through the identity system.
Role model	Created Employee, Technician, and Administrator roles.
JWT authentication	Implemented token creation and validation.
Registration	Added an endpoint for creating user accounts.
Login	Added an endpoint that authenticates users and returns a JWT.
Protected access	Added a current-user endpoint that requires authentication.
Verification	Completed successful authorized and unauthorized request tests.
Source control	Committed all completed Phase 2 changes to Git.

Verified Phase 2 Outcomes

- A user can register through the API.
- A registered user can log in and receive a JWT.
- The API validates the token on protected requests.
- An authenticated request can access the protected current-user endpoint.
- A request without valid authorization is rejected.
- Role definitions are in place for the three intended user types.

5. Combined Milestone After Phase 2

ResolveDesk now has a working backend foundation with:

- Layered ASP.NET Core architecture
- PostgreSQL running through Docker Compose
- Entity Framework Core migrations and ticket-related domain models
- Secure local configuration and database health monitoring
- ASP.NET Core Identity users and roles
- Employee, Technician, and Administrator access roles
- JWT creation, validation, registration, login, and protected access
- Verified authorized and unauthorized behavior
- Clean Git checkpoints for both completed phases

6. Next Planned Phase

Phase 3 - Ticket API and Business Logic

The next phase will add the core ticket workflow and business rules. Planned work includes ticket creation, retrieval, updates, status changes, technician assignments, role-based permissions, validation, and error handling.

This record is limited to completed and verified work from Phases 1 and 2, plus the explicitly planned next phase.