IT Helpdesk System – Full Project Documentation

Author: Tshwarelo Lephoto
Project Type: Full-Stack Web Application
Tech Stack: ASP.NET Core MVC, Entity Framework Core, SQL Server, C#, Razor Pages, Bootstrap 5
Date Started: April 2025
Date Completed (Partial): January 2026

1. Project Overview

The **IT Helpdesk System** is a web application designed to manage IT support tickets in an organization. It allows users to:

- Submit IT requests/tickets
- Track the status of their requests
- View ticket history
- Manage roles and permissions (User/Admin)
- Administer users and roles

Technologies used: ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity, Razor Pages, Bootstrap 5.

The system provides role-based access to tickets and a responsive, professional user interface.

2. Project Setup

Tools and Technologies:

- **IDE:** Visual Studio 2022
- **Backend:** ASP.NET Core MVC
- **Frontend:** Razor Pages, Bootstrap 5, Google Material Symbols
- **Database:** SQL Server

- **ORM:** Entity Framework Core
- **Authentication:** ASP.NET Core Identity
- **Version Control:** Git

Initial Steps:

1. Created a new ASP.NET Core MVC project.
2. Added **Entity Framework Core** and configured **SQL Server** connection.
3. Installed **Identity scaffolding** for login, registration, and user management.
4. Added **Bootstrap** and **Material Symbols** for UI.

3. Database Design

Entities

ApplicationUser (inherits IdentityUser)

- `FullName` – string, full name of the user
- Roles managed through Identity

Ticket

- `TicketId` – int, primary key
- `Title` – string
- `Description` – string
- `Status` – string (Open, Closed, etc.)
- `CreatedAt` – DateTime
- `UpdatedAt` – DateTime
- `CreatedById` – string, foreign key to ApplicationUser
- `AssignedToId` – string, foreign key to ApplicationUser (optional)

Relationships:

- `Ticket.CreatedById` → ApplicationUser (1-to-many)
- `Ticket.AssignedToId` → ApplicationUser (1-to-many, optional)

Implementation Notes:

- Identity tables (`AspNetUsers`, `AspNetRoles`) manage users and roles.
- Added migrations and updated the database after model creation.

4. Identity & Roles Implementation

Roles

- **User** – Default role, can create tickets and view their own tickets.
- **Admin** – Can view all tickets, close tickets, delete tickets, and manage users/roles.

Steps Taken

1. Added `Roles.Admin` constant class.
2. Updated `Program.cs` to seed roles and an initial Admin user:

```
await SeedData.InitializeAsync(userManager, roleManager);
```

3. Modified `ApplicationUser` to include `FullName`.
4. Updated registration pages to capture `FullName`.
5. Implemented `[Authorize]` and role-based UI checks in controllers and views.

5. Controllers

TicketsController

Purpose: CRUD operations for tickets and role-based filtering.

Key Features:

- **Index:** Shows logged-in user's tickets (Admins can see all).
- **History:** Shows closed tickets.
- **Details:** Includes ticket details and restricts access if not owner or Admin.
- **Create:** Assigns `CreatedById` to current user.
- **Edit:** Updates ticket while preserving `CreatedAt` and `CreatedById`.
- **Close:** Only owner/Admin can close a ticket.
- **Delete:** Admin only.

Security Features:

- `[Authorize]` attribute on controller

- `User.IsInRole("Admin")` checks
- Restricts unauthorized access

Enhancements:

- EF Core `Include` used to load `CreatedBy` and `AssignedTo`.
- Guest access prevented.

AdminController (Planned)

Purpose: Admin panel for managing users, roles, and tickets.

- Access restricted using `[Authorize(Roles = "Admin")]`.
- Sidebar links for **Admin Panel** and **Users & Roles**.

6. Razor Views / UI

Layout:

- Responsive **sidebar** with toggler
- Primary navigation: Home, My Requests, Request History, Manage Requests (Admin only)
- Secondary navigation: Profile and Logout (or Login for guests)
- Dynamic display of **FullName** and Admin badge

Home Page:

- Hero section with welcome message
- Services section: Tool Access, Equipment, Software Troubleshooting, IT Support
- Dashboard cards: Create Ticket, My Requests, History, Help

UI Notes:

- Guests are redirected to login if accessing tickets.
- Sidebar updates dynamically based on role.

7. Routing and Middleware

Program.cs Setup:

```
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.MapRazorPages();
```

- Ensures Identity pages work
- Adds authentication and authorization middleware

8. Development Process – Step-by-Step

Step 1: Project Setup

- Created project, added EF Core, Identity, and Bootstrap

Step 2: Database & Models

- Added Ticket model
- Updated ApplicationUser with FullName
- Created migrations and updated DB

Step 3: Controller & CRUD

- TicketsController created
- Restricted actions based on user/admin roles
- Added EF Core queries with `Include`

Step 4: UI & Sidebar

- Responsive sidebar navigation
- Dynamic visibility based on roles
- Guest access handled

Step 5: Role Integration

- Seeded Admin/User roles
- Updated layout to show Admin links
- Updated dashboard and ticket visibility

9. Security & Best Practices

- `[Authorize]` and `[Authorize(Roles = "Admin")]` attributes
- Verified ownership before allowing edits/deletions
- Role-based UI for Admin vs User
- Prevented direct URL access for guests

10. Lessons Learned

- ASP.NET Core Identity is powerful but requires careful role planning.
- EF Core `Include` is essential for related entity data.
- Razor views can dynamically adapt to roles using proper `if` checks.
- Seeding users/roles early reduces future errors.
- Incremental development prevents being overwhelmed.

11. Known Limitations / Future Work

- AdminController incomplete (Users & Roles management pages).
- Ticket assignment to Admins not fully implemented.
- Some dashboard cards are placeholders.
- Notifications/email confirmations minimal (`NullEmailSender`).

12. Summary

Even though the project was not fully completed, the following features are functional:

- User registration, ticket creation, and history tracking

- Admin access to ticket management and role-aware UI
- Secure, responsive, role-based navigation
- Foundation for future expansion