Issues Faced and Resolution Documentation

Smart Ticket Issue Management System

Issues Faced During Development

Issue 1: Background Worker Crash Due to Incorrect Dependency Lifetime

Problem: Application failed to start and threw `System.AggregateException` during startup. The BackgroundService could not resolve dependencies.

Cause: A Singleton `TicketEventWorker` attempted to inject a Scoped service `ITicketHistoryService`, which internally depends on DbContext. This violated .NET DI lifetime rules where scoped services cannot be injected into singletons, risking memory leaks and thread-safety issues.

Fix: Injected `IServiceScopeFactory` instead and manually created a scope inside `ExecuteAsync()` to obtain fresh scoped services per execution cycle. This ensures proper DbContext lifetime management and resource disposal.

Issue 2: Event Queue Limited to Single Event Type

Problem: The event queue was tightly coupled to `TicketUpdatedEvent`, making it impossible to extend for comments, notifications, or other domain events without major refactoring.

Cause: Queue was designed to accept a concrete event type instead of a generic contract, creating inflexibility and violating the Open-Closed Principle.

Fix: Introduced a generic `IDomainEvent` marker interface that all events implement. Refactored the queue to accept and process any event implementing this interface, enabling polymorphic event handling.

Issue 3: Pattern Matching Failure in Event Handler

Problem: The switch statement in the background worker failed to correctly match and route events to their handlers, causing events to be ignored.

Cause: Events did not share a common base interface, and the inheritance hierarchy was inconsistent, leading to runtime type checking failures.

Fix: Made all event classes sealed and ensured they implement `IDomainEvent`. Used proper C# pattern matching with type checks in the switch statement to reliably route events.

Issue 4: Lucene Auto-Assignment Misclassification

Problem: Lucene classifier assigned incorrect categories to tickets when content was ambiguous, causing misrouting and agent workload imbalance.

Cause: The ML model always returned a category even when confidence scores were extremely low, leading to random assignments for tickets that didn't match training data patterns.

Fix: Implemented a confidence threshold check. When the score falls below the threshold, tickets are assigned to "Miscellaneous" category for manual review instead of incorrect auto-assignment.

Issue 5: Identity Column Conflict with Snowflake IDs

Problem: SQL Server threw primary key insertion errors when attempting to insert tickets with Snowflake-generated IDs.

Cause: The TicketId column was configured as IDENTITY in SQL Server, which auto-generates values and prevents manual ID insertion. This conflicted with externally generated Snowflake IDs.

Fix: Marked the TicketId property with `[DatabaseGenerated(DatabaseGeneratedOption.None)]` to disable identity generation. Dropped and recreated the column to apply changes.

Issue 6: EF Core Seeding Failed with Identity Columns

Problem: Database seeding for master data tables (Roles, TicketStatus, TicketPriority, TicketCategory) failed with "Cannot insert explicit value for identity column" errors.

Cause: Master tables were configured with IDENTITY columns but seeding logic attempted to insert explicit IDs to align with enum values for cleaner lookups.

Fix: Configured these entities with `ValueGeneratedNever()` in the model builder to disable identity generation. Recreated the database to apply schema changes.

Issue 7: Migration Failures Due to Schema Conflicts

Problem: EF Core migrations failed during deployment with foreign key constraint errors when trying to apply changes to existing database.

Cause: Migrations were created in wrong order, and some referenced tables that didn't exist yet. Manual database changes were made outside of migrations during testing.

Fix: Dropped development database and regenerated migrations in correct order. Established rule to never modify database schema manually, always through migrations.

Issue 8: Validation Errors Not Displayed to User

Problem: When ticket creation failed due to validation errors, users only saw generic "Bad Request" message without knowing what was wrong.

Cause: API returned 400 status but ModelState errors weren't properly serialized into response. Frontend didn't handle validation error structure.

Fix: Implemented custom validation error response format that includes field names and specific error messages. Updated frontend to parse and display these errors next to relevant form fields.

Issue 9: Lucene Search Not Finding Recently Created Tickets

Problem: Tickets created within the last few seconds wouldn't appear in search results, requiring users to refresh multiple times.

Cause: Lucene index wasn't being committed immediately after document addition. Changes were buffered in memory and only persisted periodically.

Fix: Added explicit index commit after ticket creation and updates. Balanced commit frequency to maintain search freshness without excessive I/O overhead.

Issue 10: Immediate Ticket Updates Triggered on Dropdown Selection

Problem: Ticket status, priority, and assignee were updated immediately upon menu selection, causing accidental changes without user confirmation.

Cause: UI directly invoked the update API inside menu click handlers instead of staging changes.

Fix: Introduced a staged update mechanism using a local `stagedUpdate` state and `editMode`. Changes are now applied only after explicitly clicking the **Update Ticket** button.

Issue 11: SignalR Event Listeners Caused Duplicate Data Reloads

Problem: Ticket history and comments were reloaded multiple times for a single update event.

Cause: SignalR subscriptions were not filtered by ticket ID.

Fix: Added ticket ID checks before handling SignalR events, ensuring updates are processed only for the active ticket.
