Software Design Document

Prodigy: Support Tracker

Reference:	SDD-INT-SB-230718.0
Revision:	0
Revision Date:	7th August 2023
Revision Author:	Shaun Broomfield

Tascomp Limited

Industrial Software & Systems

Newburgh Court
Belasis Hall Technology Park
Billingham, Stockton-on-Tees
TS23 4EE. UK

Tel: +44 (0) 1642 370666
Fax: +44 (0) 1642 370012
Email: sales@tascomp.com
Web: www.tascomp.com

Revision History

Revision	Revision Date	Revision Authors	Reviewed by	Review Date
0	7th August 2023	Shaun Broomfield		
Description				
The initial version of the Proposal Document created.				

Contents

1. Introduction

Tascomp's commitment to offering exceptional support for our Prodigy, PlantRun, and PAMS systems has been a cornerstone of our success. Through the years, support events and request from our diverse clientele, including several government-affiliated agencies, have highlighted the need for a streamlined and dedicated ticketing system.

Our current support system, while effective for its original purposes, has shown limitations as a ticketing platform. This gap in our operations has underscored the necessity for the 'Support Tracker', a dedicated tool designed to efficiently manage and track customer inquiries, technical issues, and support requests.

By introducing the Support Tracker, our objective is clear: enhance communication, improve tracking, and reduce resolution times for reported issues. While we acknowledge that achieving full ITIL standards is beyond the immediate scope, the Support Tracker aims to fulfil the fundamental needs of a standard ticketing system.

This document will delve deeper into the design and operational aspects of the Support Tracker, reflecting Tascomp's dedication to enhancing its support mechanisms.

2. Overview

Support Tracker is Tascomp's answer to the growing demand for a specialised and efficient support ticketing solution. As our clientele base and their needs have evolved, especially users of our Prodigy, PlantRun, and PAMS systems, so has our recognition of the importance of an improved support infrastructure. Here's a succinct overview of what the Support Tracker brings to the table:

1. **Purpose & Justification:**
    - **Primary Aim:** Facilitate effective management and tracking of customer inquiries, technical glitches, and support requests.
    - **Client Requirements:** Cater to the specific needs of clients, notably those affiliated with government agencies, by offering distinct support ticket IDs.
    - **Operational Enhancement:** Replace the current system's `*Support Manager's events*` which, while functional for its original design, falls short in the realm of ticketing.
2. **Key Features:**
    - **Streamlined Communication:** Equipped to ensure clarity, reduce bottlenecks, and provide instant updates.
    - **Efficient Tracking:** Every ticket will have a clear trail of interactions, updates, and resolutions.
    - **Rapid Resolution:** Focus on minimising downtime or disruptions for clients by expediting issue resolutions.
3. **Scope:**
    - **Internal Usage:** This will be an in-house tool used exclusively by Tascomp.
    - **ITIL Standards:** While complete ITIL compliance is a broader objective, Support Tracker is designed to meet essential standards for a ticketing system.
4. **Anticipated Impact:**
    - **Enhanced Support Structure:** Strengthening Tascomp's commitment to client satisfaction.
    - **Operational Efficiency:** A significant uplift in handling support queries and managing technical issues.
    - **Client Trust:** Fulfilling and potentially surpassing client expectations, thereby cementing our reputation.

In subsequent sections of this document, we will delve into each of these areas in detail, providing a comprehensive understanding of the system's design, deliverables, and requirements. The journey towards a refined support process begins with the Support Tracker, and this overview aims to shed light on its key aspects.

3. Architecture

The C4 model is a hierarchical framework for visualising and documenting a system's architecture. It describes a system at four levels: Context, Containers, Components, and Code. Starting from a bird's-eye view at the Context level, it progressively delves deeper, culminating in the specifics at the Code level.

Note:

Within this design document, the 4th level diagrams are located in their respective sections, detailing the intricate aspects of individual components.

3.1 Context Scope

This diagram provides a high-level overview of the "Support Tracker System", detailing its interactions with both external actors and libraries.

Core Components:

- **Tool Prototype:**
  This represents an internal tool residing on the TSC-Server, which is tasked with the management and tracking of support tickets and associated events.

- **Support Tracker System API:**
  These are the core libraries that constitute the majority of the system's functionality.

- **TSC-Server:**
  An MS SQL Server database that hosts data for internal tools.

External Libraries:

The system also leverages "Tascomp Internal Libraries", which are external libraries utilised by the Support Tracker system.

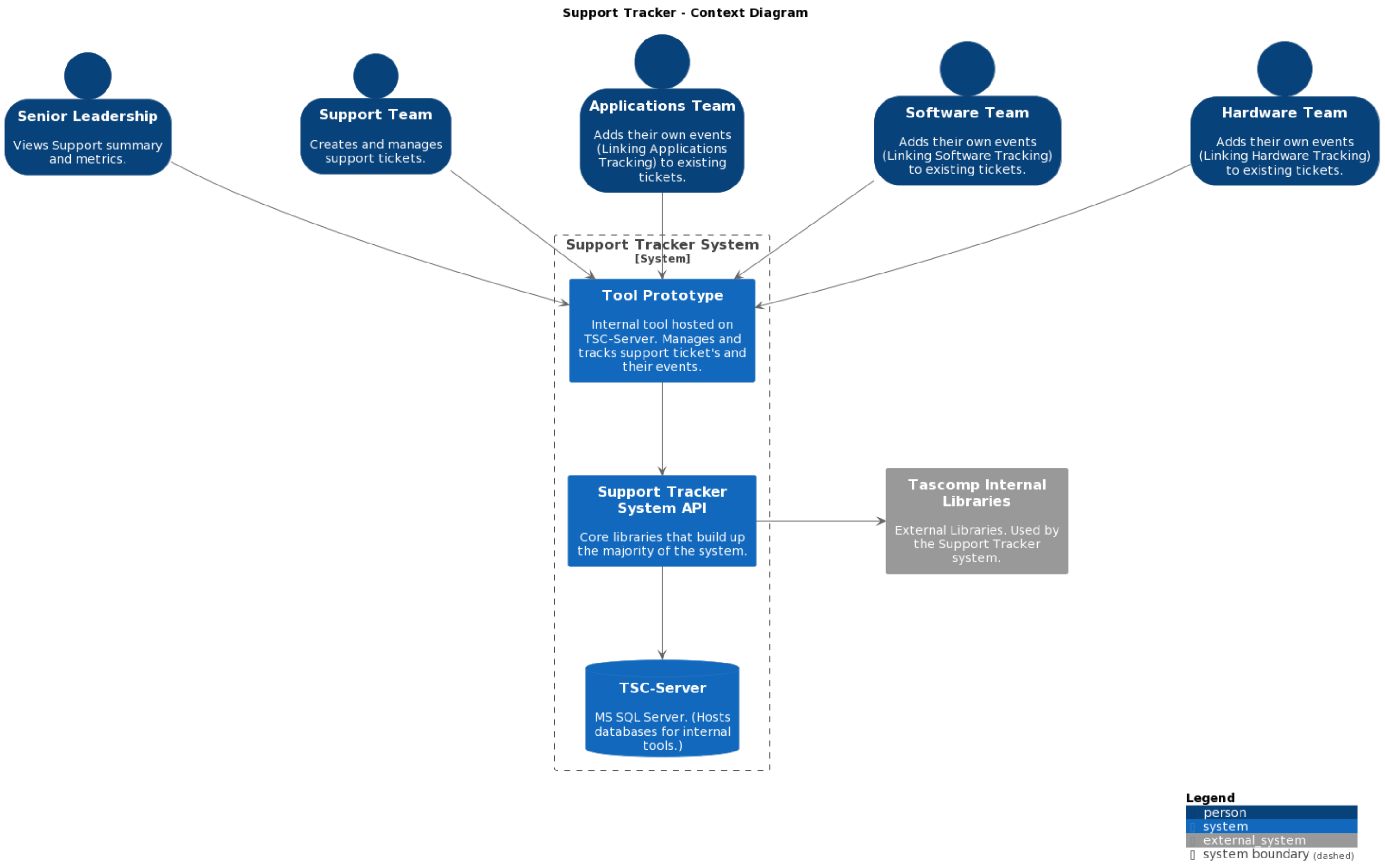
Actors and Their Interactions:

- **Senior Leadership:**
  Primarily concerned with viewing summaries and metrics related to support.

- **Support Team:**
  Engages directly with the Tool Prototype for creating and managing support tickets.

- **Applications Team, Software Team & Hardware Team:**
  These teams interface with the Tool Prototype to append their specific events, thereby linking application, software, and hardware tracking respectively to existing support tickets.

In essence, the diagram showcases the primary entities of the Support Tracker System and the various interactions taking place between them, offering a bird's eye view of the entire ecosystem.


Support Tracker - Context Diagram
Senior Leadership
Views Support summary and metrics.
Support Team
Creates and manages support tickets.
Applications Team
Adds their own events (Linking Applications Tracking) to existing tickets.
Software Team
Adds their own events (Linking Software Tracking) to existing tickets.
Hardware Team
Adds their own events (Linking Hardware Tracking) to existing tickets.
Support Tracker System
[System]
Tool Prototype
Internal tool hosted on TSC-Server. Manages and tracks support ticket's and their events.
Support Tracker System API
Core libraries that build up the majority of the system.
Tascomp Internal Libraries
External Libraries. Used by the Support Tracker system.
TSC-Server
MS SQL Server. (Hosts databases for internal tools.)
Legend
person
system
external system
system boundary (dashed)

3.2 Container Scope

The Container Diagram offers a deeper dive into the architecture of the “Support Tracker System”, particularly elucidating the major software containers or components and how they integrate with each other.

Main Tool Prototype:

- **Tool Prototype:**
  This container represents an executable (MFC EXE) with a primary function to provide a user interface made predominantly of controls derived from the “User-Interface Components”.

Support Tracker System API Components:

- **User-Interface Components:**
  An MFC DLL component filled with windows forms and user controls. It communicates directly with the “Data Access Layer” for data-driven operations.

- **Data Access Layer:**
  Another MFC DLL component designed to read and write data. It interfaces with the MS SQL Server and the domain model to ensure proper data flow.

- **Domain:**
  As an MFC DLL, this container serves as a domain model, hosting reference classes and structures vital to the system.

- **TSC-Server:**
  An MS SQL Server database that plays the role of the storage layer and hosts databases pertinent to internal tools.

External Libraries:

The system leverages the “Tascomp Internal Libraries”, which are tapped by both the “User-Interface Components” and the “Data Access Layer”.

Actors and Their Roles:

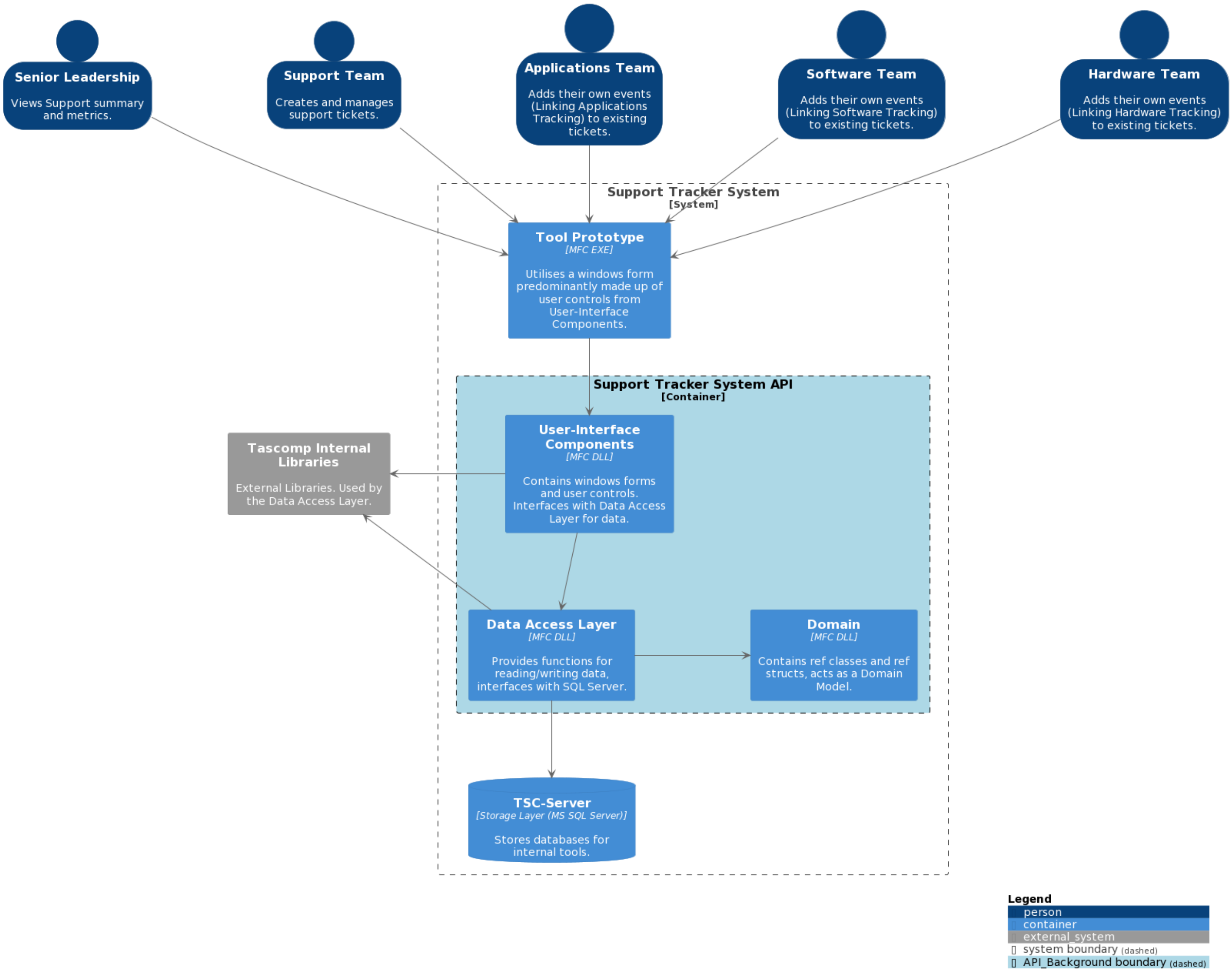
- **Senior Leadership:**
  Engages with the “Tool Prototype” primarily to view support summaries and metrics.

- **Support Team:**
  Directly interacts with the “Tool Prototype” to create and manage support tickets.

- **Applications, Software & Hardware Teams:**
  These teams interact with the “Tool Prototype”, enriching it with their respective events which correlate to application, software, and hardware tracking for the support tickets.

In summary, this diagram sheds light on the integral software containers within the “Support Tracker System”, portraying the links between them and illustrating the interactions between different user roles and the software components.

Support Tracker - Container Diagram



Senior Leadership
Views Support summary and metrics.
Support Team
Creates and manages support tickets.
Applications Team
Adds their own events (Linking Applications Tracking) to existing tickets.
Software Team
Adds their own events (Linking Software Tracking) to existing tickets.
Hardware Team
Adds their own events (Linking Hardware Tracking) to existing tickets.
Support Tracker System
[System]
Tool Prototype
[MFC EXE]
Utilises a windows form predominantly made up of user controls from User-Interface Components.
Support Tracker System API
[Container]
User-Interface Components
[MFC DLL]
Contains windows forms and user controls. Interfaces with Data Access Layer for data.
Tascomp Internal Libraries
External Libraries. Used by the Data Access Layer.
Data Access Layer
[MFC DLL]
Provides functions for reading/writing data, interfaces with SQL Server.
Domain
[MFC DLL]
Contains ref classes and ref structs, acts as a Domain Model.
TSC-Server
[Storage Layer (MS SQL Server)]
Stores databases for internal tools.
Legend
person
container
external_system
system boundary (dashed)
API_Background boundary (dashed)

3.3 Component Scope

Component diagrams visualise a system’s high-level components and their interactions. Within the C4 model of system architecture design, these diagrams reside at the third level, detailing the inner workings of containers. They offer insight into the interrelationship of components, aiding in understanding the system’s functionality and structure.

3.3.1 User-Interface Components

This diagram drills deeper into the architecture of the “User Interface Components” container, focusing on individual components and their inter-relationships.

Main UI Components:

- **Main View:**
  Serves as the primary window for user interactions. It contains the “Tickets View”, which displays a list of tickets, and further permits ticket filtering.

- **Tickets View:**
  Interfaces directly with “Ticket View”, which provides comprehensive details for individual tickets.

- **Ticket View:**
  This component interfaces with both the “Event View”, which exhibits details of a distinct event within a ticket, and with windows forms like “Linked Files View” and “Mantis References View”, which manage linked files and Mantis references, respectively.

- **Event View:**
  Likewise, has the ability to launch both the “Linked Files View” and “Mantis References View”.

- **Linked Files View and Mantis References View:**
  Both of these windows forms inherit from “Base Links View”, forming a foundational view for linked items.

Data Access Layer (DAL):

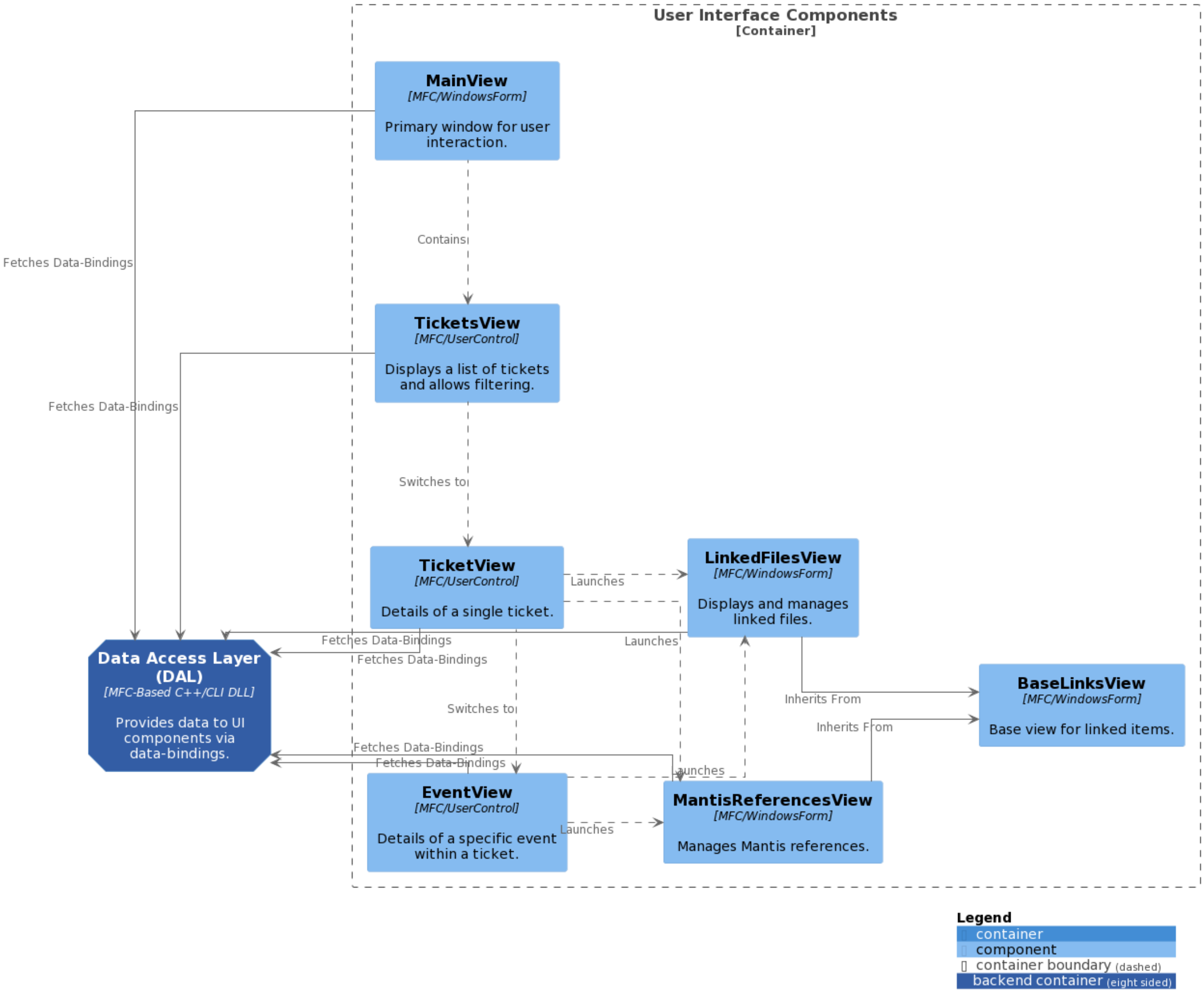
- Represented as a backend container, the Data Access Layer is a crucial MFC-Based C++/CLI DLL that facilitates data provisioning to the UI components using data-bindings.

Interactions:

- Each of the primary UI components, including “Main View”, “Tickets View”, “Ticket View”, “Event View”, “Linked Files View”, and “Mantis References View”, fetch their requisite data-bindings from the Data Access Layer (DAL).

To encapsulate, this diagram visually encapsulates the intricate layout of the User Interface Components, portraying how individual components are interrelated and how they tap into the Data Access Layer to derive their required data.

Support Tracker - User-Interface Components Diagram



3.3.2 Data Access Layer

This diagram offers a detailed view into the components that constitute the “Data Access Layer (DAL)” of the Support Tracker and describes how they cooperate with each other and interface with external components.

DAL Components:

- **Database Interaction:**
  This C++/CLI component directly handles CRUD (Create, Read, Update, Delete) operations in interaction with the SQL Database. It also engages with the “Domain Data Mapper” to dispatch raw data.

- **UI Contract:**
  Offers a higher abstraction layer with C++/CLI operations for the User Interface data bindings. It exploits the functionalities from both the “Database Interaction” and “Domain Data Mapper” components.

- **Domain Data Mapper:**
  The responsibility of this C++/CLI component is to map data between the raw SQL results and domain entities, ensuring consistency and proper data translation.

External Elements:

- **User-Interface Components:**
  A DLL component primarily created using MFC-Based C++/CLI. This holds windows forms and user controls which interface directly with the Data Access Layer to obtain the necessary data.

- **External Libraries:**
  These are pre-existing libraries from the company that perform specialised operations. Both the “Database Interaction” and “Domain Data Mapper” components leverage these libraries for certain operations and data conversions.

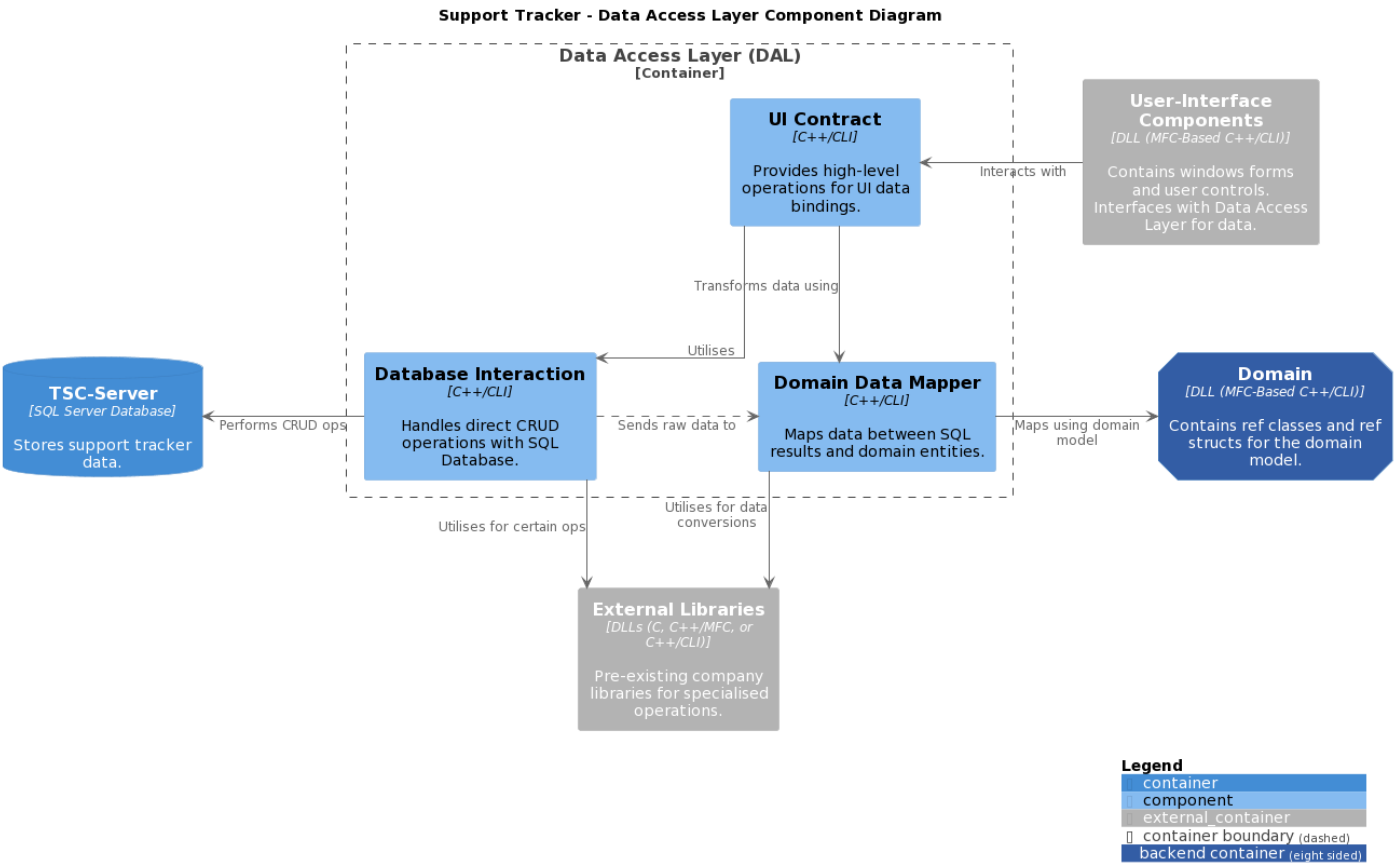
- **Domain:**
  As an MFC-Based C++/CLI DLL, the Domain consists of ref classes and ref structs, underpinning the domain model of the system.

- **TSC-Server:**
  This SQL Server Database component acts as the repository, storing the support tracker’s data. Direct CRUD operations are performed on it via the “Database Interaction” component.

Interactions:

- “Domain Data Mapper” uses the “External Libraries” for data conversions.
- “Database Interaction” leverages “External Libraries” for particular operations.
- “Database Interaction” undertakes CRUD operations on the “TSC-Server”.
- “Domain Data Mapper” cooperates with the “Domain” to map using the domain model.
- “User-Interface Components” directly interface with the “UI Contract”.

In summary, this diagram elucidates the layered and interactive structure of the Data Access Layer, illustrating how data flows, is transformed, and is made accessible to the user interface.



Support Tracker - Data Access Layer Component Diagram
Data Access Layer (DAL)
[Container]
UI Contract
[C++/CLI]
Provides high-level operations for UI data bindings.
User-Interface Components
[DLL (MFC-Based C++/CLI)]
Contains windows forms and user controls. Interfaces with Data Access Layer for data.
Interacts with
Transforms data using
Utilises
TSC-Server
[SQL Server Database]
Stores support tracker data.
Performs CRUD ops
Database Interaction
[C++/CLI]
Handles direct CRUD operations with SQL Database.
Sends raw data to
Domain Data Mapper
[C++/CLI]
Maps data between SQL results and domain entities.
Maps using domain model
Domain
[DLL (MFC-Based C++/CLI)]
Contains ref classes and ref structs for the domain model.
Utilises for certain ops
Utilises for data conversions
External Libraries
[DLLs (C, C++/MFC, or C++/CLI)]
Pre-existing company libraries for specialised operations.
Legend
container
component
external_container
container boundary (dashed)
backend container (eight sided)

3.3.3 Domain

This diagram offers a deep dive into the components that make up the “Domain” of the Support Tracker. It provides a clear view of how they interact with one another and with external components.

Domain Components:

- **Ticket Component:**
  Responsible for managing all ticket-related operations, this C++/CLI component is at the heart of the domain. It also uses and interacts with all other domain components.

- **Event Component:**
  This C++/CLI component manages event operations linked to tickets. Events within the system are integral parts of tickets and might contain linked files and Mantis references.

- **Linked File Component:**
  An important C++/CLI component that manages all the files linked to both tickets and events.

- **Linked Mantis Reference Component:**
  Another vital C++/CLI component that is responsible for managing all Mantis references associated with tickets or events.

- **Audit Action Component:**
  This component, built with C++/CLI, tracks, and manages auditing actions associated with entities within the system. Nearly every primary component in the domain utilizes it.

Interactions Within Domain:

- “Ticket Component” has a strong association with other components since it can contain events, linked files, and Mantis references. Additionally, it uses the “AuditAction Component” for auditing.

- “Event Component” can contain both linked files and Mantis references, and it utilizes the “AuditAction Component” for event-related auditing actions.

- Both the “LinkedFile Component” and “LinkedMantisReference Component” use the “AuditAction Component” to audit any actions related to linked files or Mantis references respectively.

External Elements:

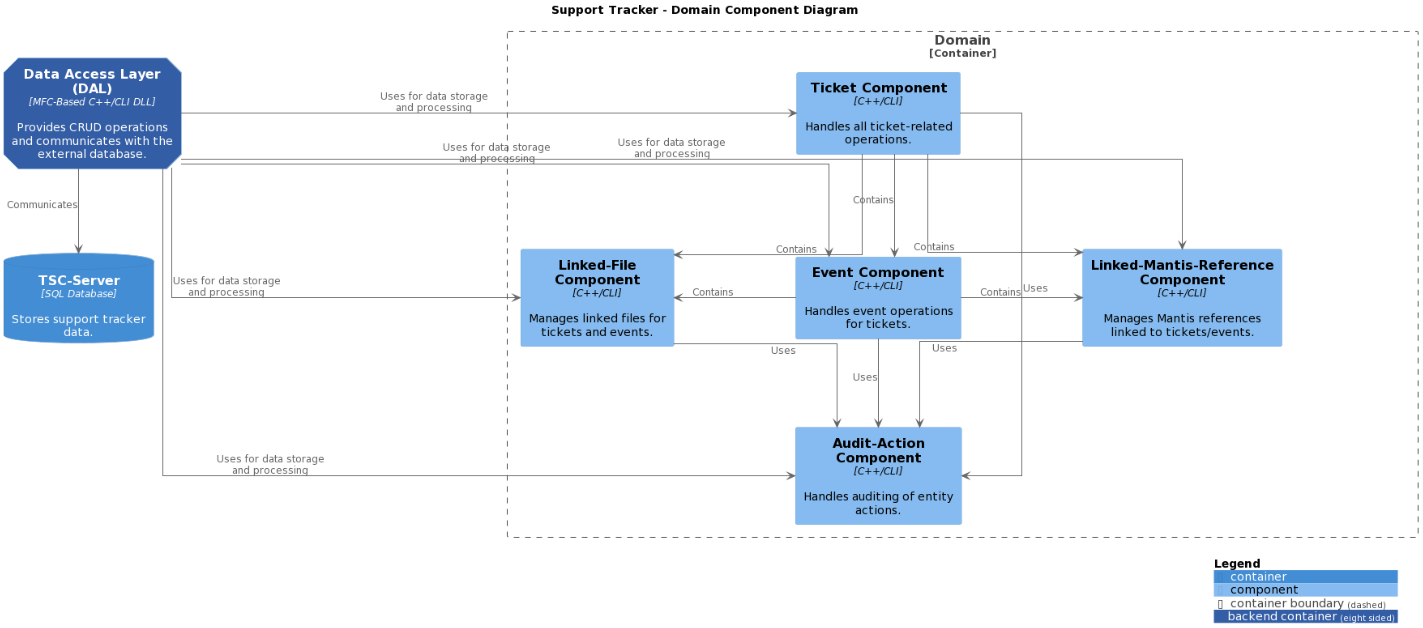
- **Data Access Layer (DAL):**
  This MFC-Based C++/CLI DLL component offers CRUD (Create, Read, Update, Delete) operations and serves as the bridge between the domain components and the external SQL database.

- **TSC-Server:**
  A Microsoft SQL Database that acts as the data repository, storing all support tracker data.

Interactions with External Elements:

- “Data Access Layer (DAL)” utilizes each domain component for data storage and processing, making it a crucial component that effectively interfaces between the application and the database.
- There’s also a direct communication established between “Data Access Layer (DAL)” and the “TSC-Server” SQL database.

In summary, this diagram uncovers the integral architecture of the Domain layer of the Support Tracker system, emphasizing the internal component interactions and the exchanges with external components.


Support Tracker - Domain Component Diagram
Domain
[Container]
Data Access Layer (DAL)
[MFC-Based C++/CLI DLL]
Provides CRUD operations and communicates with the external database.
Communicates
TSC-Server
[SQL Database]
Stores support tracker data.
Uses for data storage and processing
Uses for data storage and processing
Uses for data storage and processing
Uses for data storage and processing
Uses for data storage and processing
Ticket Component
[C++/CLI]
Handles all ticket-related operations.
Contains
Contains
Contains
Contains
Contains
Linked-File Component
[C++/CLI]
Manages linked files for tickets and events.
Event Component
[C++/CLI]
Handles event operations for tickets.
Linked-Mantis-Reference Component
[C++/CLI]
Manages Mantis references linked to tickets/events.
Uses
Uses
Uses
Uses
Audit-Action Component
[C++/CLI]
Handles auditing of entity actions.
Legend
container
component
container boundary (dashed)
backend container (eight sided)

3.3.4 Storage Layer

This diagram provides a comprehensive representation of the storage components in the TSC-SERVER system and how they interact.

TSC-SERVER Storage Layer:

The central entity of this diagram is the TSC-SERVER storage layer. It's an SQL Server that hosts multiple databases, serving as the backbone of the Support Tracker system.

Support Tracker Database:
This is the primary database of the Support Tracker application. It also contains a special view:

- **Support Tracker View:**
  An SQL view within the Support Tracker Database that amalgamates data from the main Support Tracker Database and other external database views.

- **Sub-views Inside Support Tracker Database:**
  These are specific SQL views designed to pull data from external databases into the main Support Tracker Database. They include:

  - **Support Manager View:**
    Reads data from the Support Manager tool's database.

  - **Order Manager View:**
    Pulls data from the Order Manager tool's database.

  - **Mantis View:**
    Gathers data from the Mantis bug-tracking tool's database.

External Databases:

- **Support Manager Database:**
  The database dedicated to the Support Manager tool.

- **Order Manager Database:**
  The database behind the Order Manager tool.

- **Mantis Database:**
  The storage layer for the Mantis bug tracking tool.

Relationships within TSC-SERVER:

The Support Tracker View relies on the three sub-views (Support Manager View, Order Manager View, and Mantis View) to aggregate data. Furthermore, this main view also reads directly from the Support Tracker Database.

Each of the three sub-views correspondingly reads from their respective external databases (Support Manager, Order Manager, and Mantis databases).

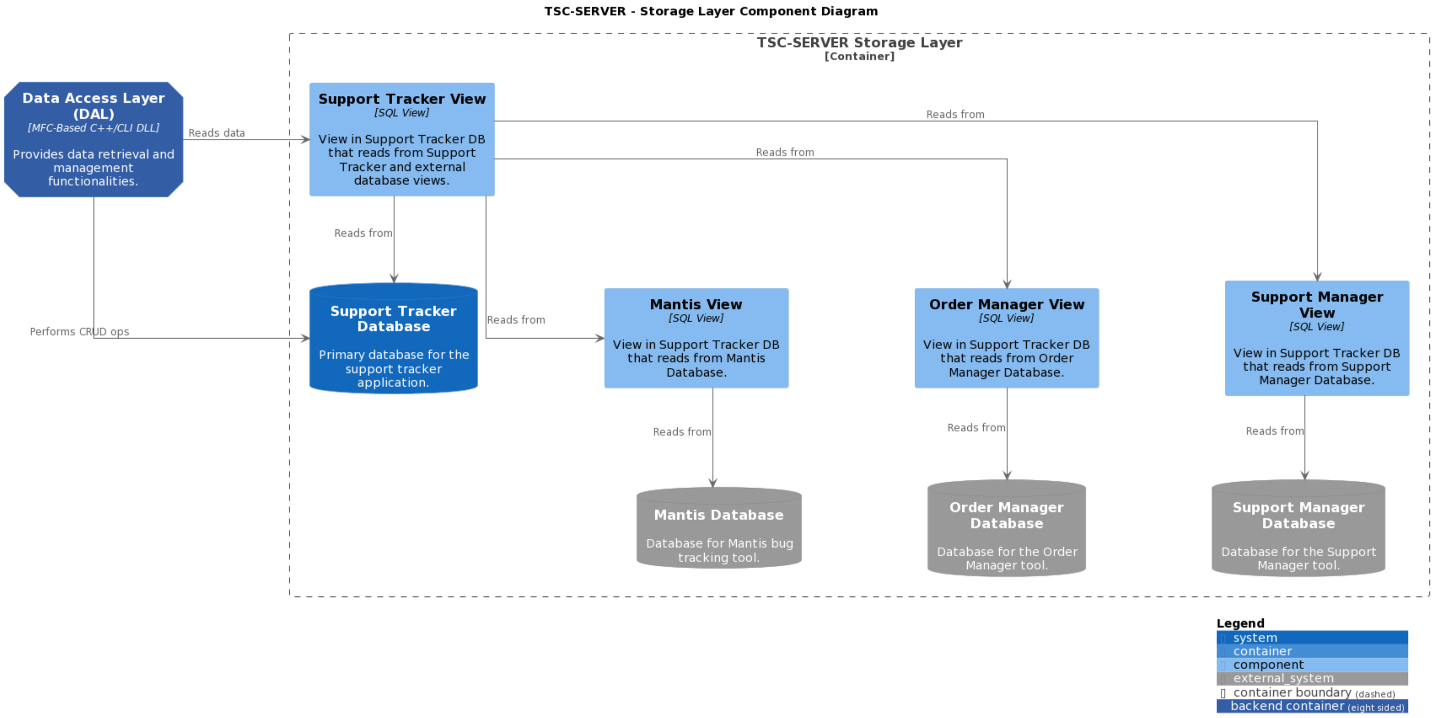
External Containers and Systems:

Data Access Layer (DAL): An MFC-Based C++/CLI DLL that is responsible for data retrieval and management operations.

Interactions with DAL:

- The Data Access Layer (DAL) has a direct connection with the Support Tracker Database for CRUD operations. It also reads data from the Support Tracker View.

This diagram effectively encapsulates the SQL Server architecture of the TSC-SERVER system, emphasizing the internal components and their connections. It also showcases how the system integrates data from various external sources into a unified, cohesive storage solution.


TSC-SERVER - Storage Layer Component Diagram
TSC-SERVER Storage Layer
[Container]
Data Access Layer (DAL)
[MFC-Based C++/CLI DLL]
Provides data retrieval and management functionalities.
Reads data
Performs CRUD ops
Support Tracker View
[SQL View]
View in Support Tracker DB that reads from Support Tracker and external database views.
Reads from
Reads from
Reads from
Reads from
Support Tracker Database
Primary database for the support tracker application.
Mantis View
[SQL View]
View in Support Tracker DB that reads from Mantis Database.
Order Manager View
[SQL View]
View in Support Tracker DB that reads from Order Manager Database.
Support Manager View
[SQL View]
View in Support Tracker DB that reads from Support Manager Database.
Reads from
Reads from
Reads from
Mantis Database
Database for Mantis bug tracking tool.
Order Manager Database
Database for the Order Manager tool.
Support Manager Database
Database for the Support Manager tool.
Legend
system
container
component
external_system
container boundary (dashed)
backend container (eight sided)

4. User Interface

This software will facilitate the management, visualisation, and effective utilisation of the data stored in the database. It has to provide an intuitive interface that will allow users to interact with the data with ease.

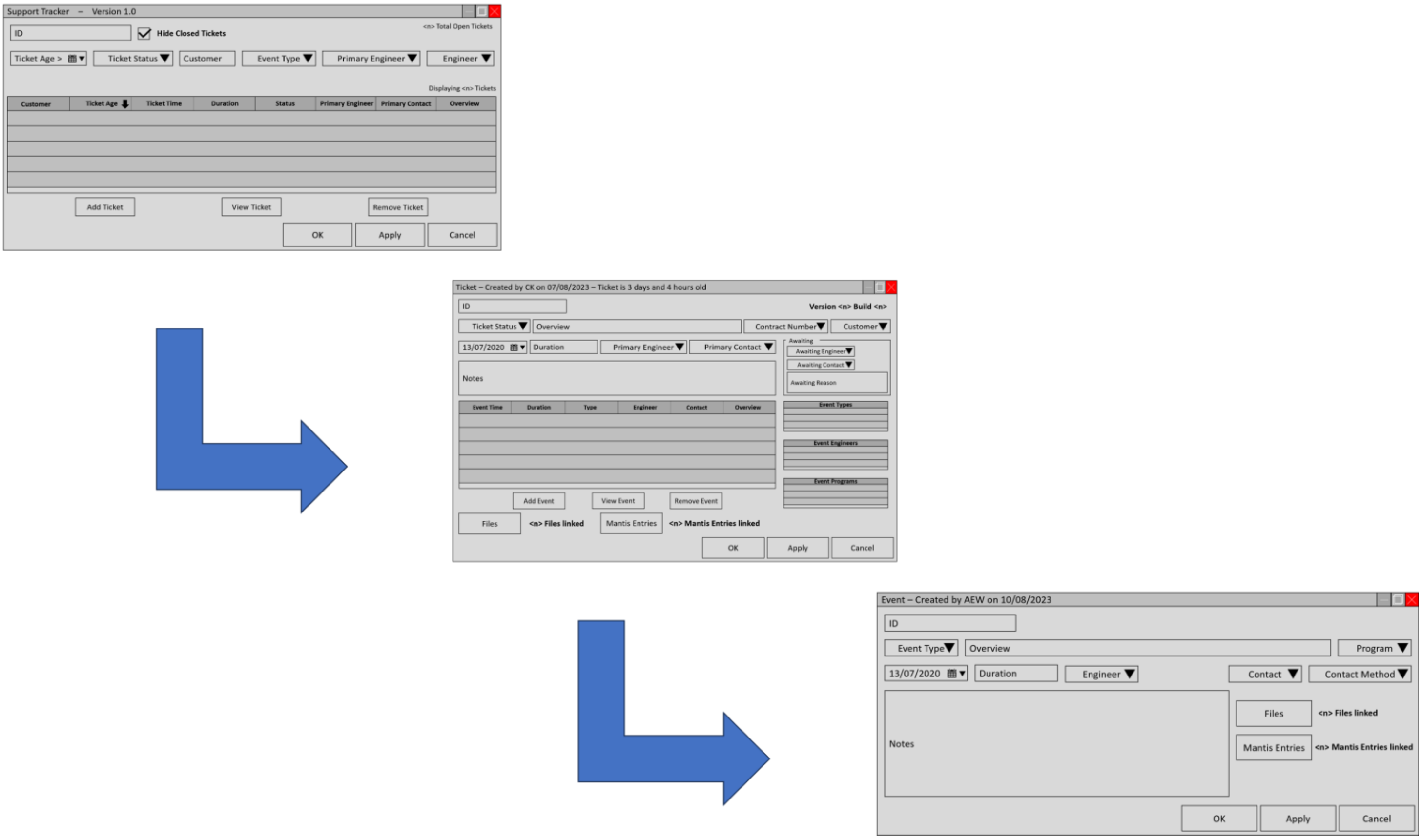
The main views for the user-interface will be nested user controls within a single windows form.

The views are as follows:

- Tickets View
    - Tickets Filtering
    - Tickets List
- Ticket View
    - Ticket Details
    - Ticket Events List
- Event View

There will also be a sub-form, a windows form that will be launched by the main form.
This sub-form will have two derived views:

- Linked Files View
- Mantis References View

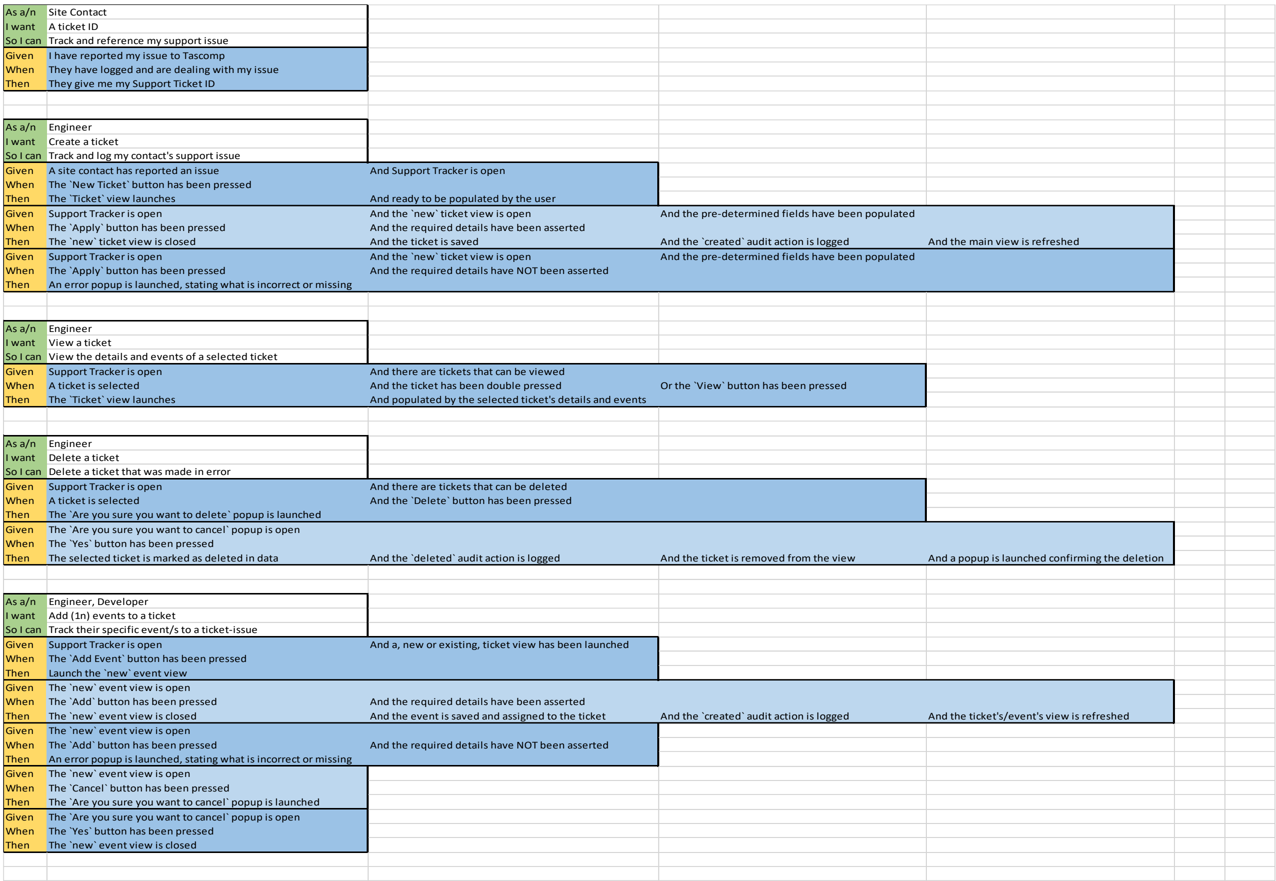


4.1 User Interactions

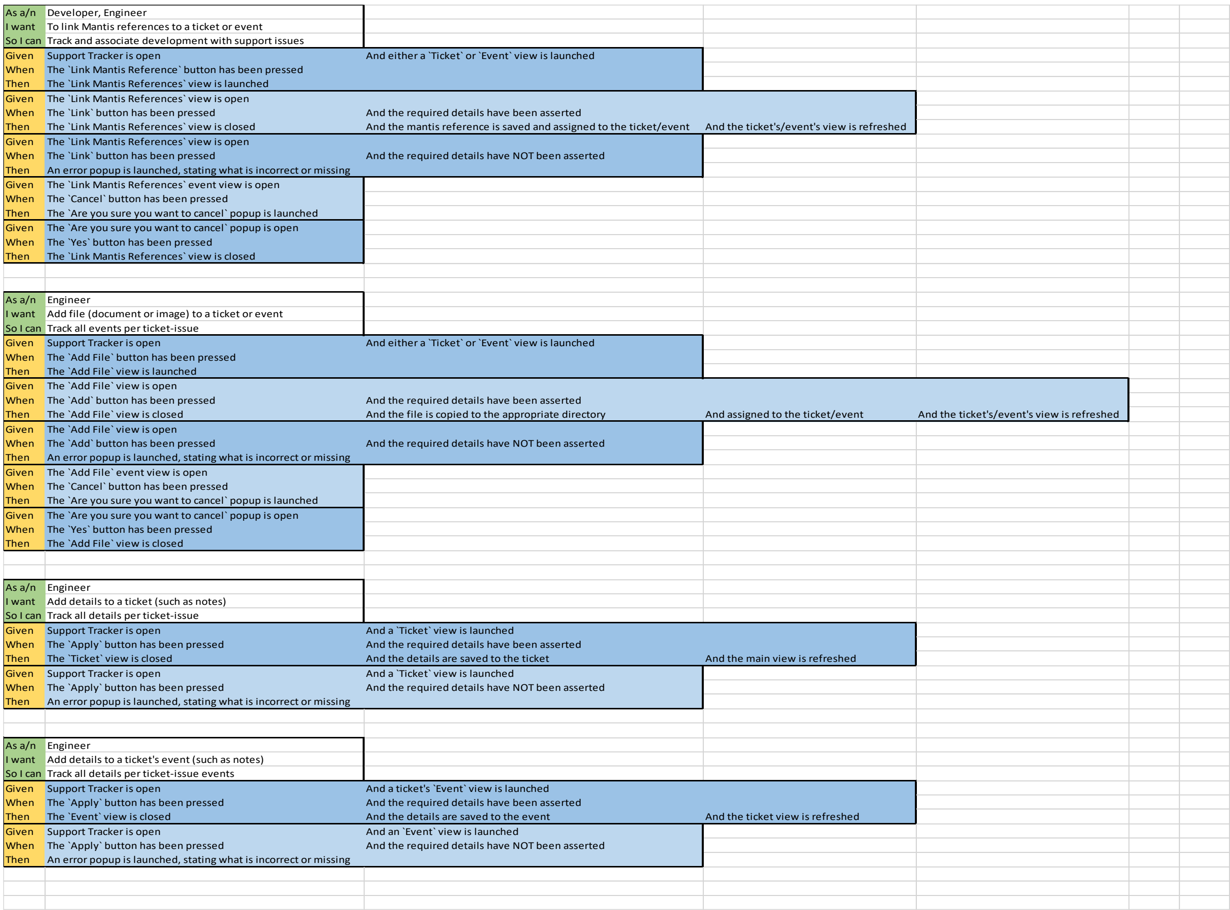
The following spreadsheet is a collection of what are known as `user stories`.

User stories are concise, plain-language descriptions of features or functionalities from the perspective of an end-user. They typically capture what the user wants to achieve, often formatted as ‘As a *[type of user]*, I want *[a specific functionality]* so that *[I achieve some benefit or value].*’

Within the context of a software design document, user stories play an essential role in bridging the gap between technical specifications and user requirements. They ensure that the software developed aligns closely with the user’s needs and expectations. By embedding them within the design document, stakeholders and maintainers can get a clearer understanding of the intended user experience and functionalities, allowing for a more user-centric design and development process.



As a/n	Site Contact			
I want	A ticket ID			
So I can	Track and reference my support issue			
Given	I have reported my issue to Tascomp			
When	They have logged and are dealing with my issue			
Then	They give me my Support Ticket ID			
As a/n	Engineer			
I want	Create a ticket			
So I can	Track and log my contact's support issue			
Given	A site contact has reported an issue	And Support Tracker is open		
When	The `New Ticket` button has been pressed			
Then	The `Ticket` view launches	And ready to be populated by the user		
Given	Support Tracker is open	And the `new` ticket view is open	And the pre-determined fields have been populated	
When	The `Apply` button has been pressed	And the required details have been asserted		
Then	The `new` ticket view is closed	And the ticket is saved	And the `created` audit action is logged	And the main view is refreshed
Given	Support Tracker is open	And the `new` ticket view is open	And the pre-determined fields have been populated	
When	The `Apply` button has been pressed	And the required details have NOT been asserted		
Then	An error popup is launched, stating what is incorrect or missing			
As a/n	Engineer			
I want	View a ticket			
So I can	View the details and events of a selected ticket			
Given	Support Tracker is open	And there are tickets that can be viewed		
When	A ticket is selected	And the ticket has been double pressed	Or the `View` button has been pressed	
Then	The `Ticket` view launches	And populated by the selected ticket's details and events		
As a/n	Engineer			
I want	Delete a ticket			
So I can	Delete a ticket that was made in error			
Given	Support Tracker is open	And there are tickets that can be deleted		
When	A ticket is selected	And the `Delete` button has been pressed		
Then	The `Are you sure you want to delete` popup is launched			
Given	The `Are you sure you want to cancel` popup is open			
When	The `Yes` button has been pressed			
Then	The selected ticket is marked as deleted in data	And the `deleted` audit action is logged	And the ticket is removed from the view	And a popup is launched confirming the deletion
As a/n	Engineer, Developer			
I want	Add (1n) events to a ticket			
So I can	Track their specific event/s to a ticket-issue			
Given	Support Tracker is open	And a, new or existing, ticket view has been launched		
When	The `Add Event` button has been pressed			
Then	Launch the `new` event view			
Given	The `new` event view is open			
When	The `Add` button has been pressed	And the required details have been asserted		
Then	The `new` event view is closed	And the event is saved and assigned to the ticket	And the `created` audit action is logged	And the ticket's/event's view is refreshed
Given	The `new` event view is open			
When	The `Add` button has been pressed	And the required details have NOT been asserted		
Then	An error popup is launched, stating what is incorrect or missing			
Given	The `new` event view is open			
When	The `Cancel` button has been pressed			
Then	The `Are you sure you want to cancel` popup is launched			
Given	The `Are you sure you want to cancel` popup is open			
When	The `Yes` button has been pressed			
Then	The `new` event view is closed			



As a/n	Developer, Engineer			
I want	To link Mantis references to a ticket or event			
So I can	Track and associate development with support issues			
Given	Support Tracker is open	And either a `Ticket` or `Event` view is launched		
When	The `Link Mantis Reference` button has been pressed			
Then	The `Link Mantis References` view is launched			
Given	The `Link Mantis References` view is open			
When	The `Link` button has been pressed	And the required details have been asserted		
Then	The `Link Mantis References` view is closed	And the mantis reference is saved and assigned to the ticket/event	And the ticket's/event's view is refreshed	
Given	The `Link Mantis References` view is open			
When	The `Link` button has been pressed	And the required details have NOT been asserted		
Then	An error popup is launched, stating what is incorrect or missing			
Given	The `Link Mantis References` event view is open			
When	The `Cancel` button has been pressed			
Then	The `Are you sure you want to cancel` popup is launched			
Given	The `Are you sure you want to cancel` popup is open			
When	The `Yes` button has been pressed			
Then	The `Link Mantis References` view is closed			

As a/n	Engineer			
I want	Add file (document or image) to a ticket or event			
So I can	Track all events per ticket-issue			
Given	Support Tracker is open	And either a `Ticket` or `Event` view is launched		
When	The `Add File` button has been pressed			
Then	The `Add File` view is launched			
Given	The `Add File` view is open			
When	The `Add` button has been pressed	And the required details have been asserted		
Then	The `Add File` view is closed	And the file is copied to the appropriate directory	And assigned to the ticket/event	And the ticket's/event's view is refreshed
Given	The `Add File` view is open			
When	The `Add` button has been pressed	And the required details have NOT been asserted		
Then	An error popup is launched, stating what is incorrect or missing			
Given	The `Add File` event view is open			
When	The `Cancel` button has been pressed			
Then	The `Are you sure you want to cancel` popup is launched			
Given	The `Are you sure you want to cancel` popup is open			
When	The `Yes` button has been pressed			
Then	The `Add File` view is closed			

As a/n	Engineer		
I want	Add details to a ticket (such as notes)		
So I can	Track all details per ticket-issue		
Given	Support Tracker is open	And a `Ticket` view is launched	
When	The `Apply` button has been pressed	And the required details have been asserted	
Then	The `Ticket` view is closed	And the details are saved to the ticket	And the main view is refreshed
Given	Support Tracker is open	And a `Ticket` view is launched	
When	The `Apply` button has been pressed	And the required details have NOT been asserted	
Then	An error popup is launched, stating what is incorrect or missing		

As a/n	Engineer		
I want	Add details to a ticket's event (such as notes)		
So I can	Track all details per ticket-issue events		
Given	Support Tracker is open	And a ticket's `Event` view is launched	
When	The `Apply` button has been pressed	And the required details have been asserted	
Then	The `Event` view is closed	And the details are saved to the event	And the ticket view is refreshed
Given	Support Tracker is open	And an `Event` view is launched	
When	The `Apply` button has been pressed	And the required details have NOT been asserted	
Then	An error popup is launched, stating what is incorrect or missing		

4.2 Front-End Views

Support Tracker utilises a single windows form as its primary interface, with the intention of maintaining a consistent window for users even as they navigate through various views. This central windows form, termed the Main View, encompasses multiple nested user controls to present different functionalities and data sets related to the support tickets. This design choice ensures that while users can access a multitude of views and operations, they remain within a single primary window for the majority of their tasks, ensuring a consolidated and straightforward user experience.

In addition to the main user workflow presented within this primary window, there are a few dedicated windows forms that act as sub-forms for specific functionalities, such as viewing linked files or Mantis references.

The structure is hierarchical:

- **Main View (Windows Form):**
  The central form, containing the primary navigation and housekeeping features.
    - Contains a list of user controls it can display, and flags indicating if there are pending changes.
    - Has various controls including the contents panel, and buttons for back, apply, and exit functionalities.
    - Displays the following user controls:
        - **Tickets View**
        - **Ticket View**
        - **Event View**

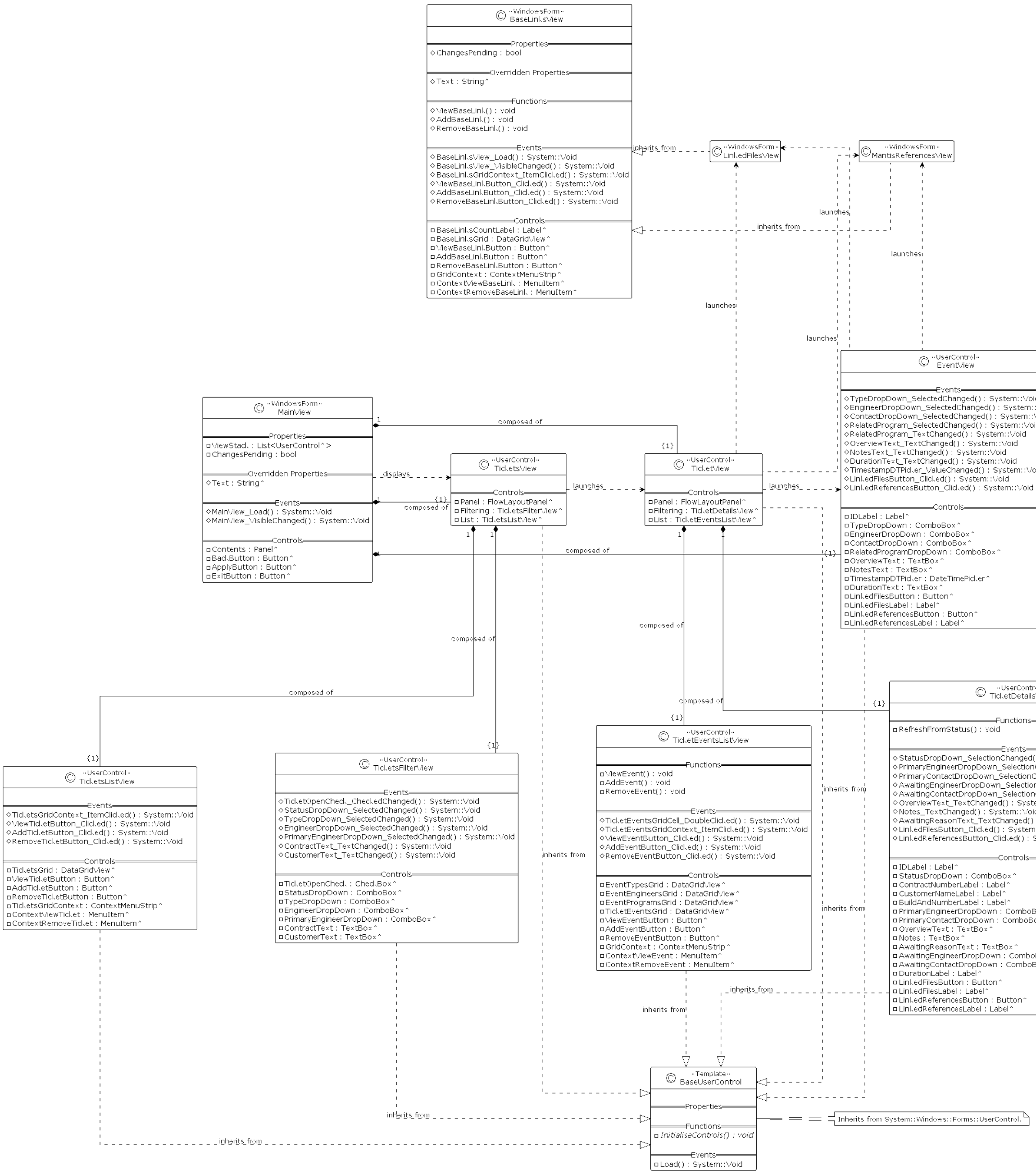
- **Tickets View (Derived User Control):**
  Provides an overview of all the support tickets.
    - Contains a panel and two nested user controls:
        - **Tickets Filter View:**
          Allows users to filter tickets based on multiple criteria such as open status, type, engineer, primary engineer, contract, and customer.
        - **Tickets List View:**
          Offers a list of all tickets with options to view, add, or remove a ticket through its grid and related controls.

- **Ticket View (Derived User Control):**
  Displays detailed information about a specific ticket.
  - Contains a panel and three nested user controls:
    - **Ticket Details View:**
      Provides detailed information about a ticket including status, engineer details, contact details, and more.
    - **Ticket Events List View:**
      Lists all events related to a ticket with options to view, add, or remove events.
- **Event View (Derived User Control):**
  Provides a detailed overview of specific events or interactions related to a ticket.
  - Details about the event, including type, engineer, contact, program, and more.
- **Base User Control (Template User Control):**
  Serves as a foundational template for various user controls.
  - Contains basic properties and methods, including a method for initializing controls.
  - Many of the user controls inherit from this base class to gain core properties and functionalities.
- **Linked Files View and Mantis References View (Derived Windows Forms):**
  Display files linked to tickets and references, respectively.
  - Both inherit from the Base Links View, which offers base functionalities for managing links.
- **Base Links View (Template User Control):**
  A foundational class for managing links.
  - Contains properties, functions, events, and controls related to viewing, adding, or removing links.

Relationship Dynamics:

- The `*Main View*` incorporates `*Tickets View*`, `*Ticket View*`, and `*Event View*`.
- `*Tickets View*` integrates both `*Tickets Filter View*` and `*Tickets List View*`.
- `*Ticket View*` collaborates with `*Ticket Details View*` and `*Ticket Events List View*`.
- `*Linked Files View*` and `*Mantis References View*` inherit from `*Base Links View*`.
- Many of the user controls inherit from the `*Base User Control*` template.

Support Tracker - User Interface View Classes



«WindowsForm» BaseLinksView
Properties
◇ChangesPending : bool
Overridden Properties
◇Text : String^
Functions
◇ViewBaseLink() : void
◇AddBaseLink() : void
◇RemoveBaseLink() : void
Events
◇BaseLinksView_Load() : System::Void
◇BaseLinksView_VisibleChanged() : System::Void
◇BaseLinksGridContext_ItemClicked() : System::Void
◇ViewBaseLinkButton_Clicked() : System::Void
◇AddBaseLinkButton_Clicked() : System::Void
◇RemoveBaseLinkButton_Clicked() : System::Void
Controls
□BaseLinksCountLabel : Label^
□BaseLinksGrid : DataGridView^
□ViewBaseLinkButton : Button^
□AddBaseLinkButton : Button^
□RemoveBaseLinkButton : Button^
□GridContext : ContextMenuStrip^
□ContextViewBaseLink : MenuItem^
□ContextRemoveBaseLink : MenuItem^
inherits_from
«WindowsForm» LinkedFilesView
«WindowsForm» MantisReferencesView
inherits_from
launches
launches
launches
launches
«UserControl» EventView
Events
◇TypeDropDown_SelectedChanged() : System::Void
◇EngineerDropDown_SelectedChanged() : System::Void
◇ContactDropDown_SelectedChanged() : System::Void
◇RelatedProgram_SelectedChanged() : System::Void
◇RelatedProgram_TextChanged() : System::Void
◇OverviewText_TextChanged() : System::Void
◇NotesText_TextChanged() : System::Void
◇DurationText_TextChanged() : System::Void
◇TimestampDTPicker_ValueChanged() : System::Void
◇LinkedFilesButton_Clicked() : System::Void
◇LinkedReferencesButton_Clicked() : System::Void
Controls
□IDLabel : Label^
□TypeDropDown : ComboBox^
□EngineerDropDown : ComboBox^
□ContactDropDown : ComboBox^
□RelatedProgramDropDown : ComboBox^
□OverviewText : TextBox^
□NotesText : TextBox^
□TimestampDTPicker : DateTimePicker^
□DurationText : TextBox^
□LinkedFilesButton : Button^
□LinkedFilesLabel : Label^
□LinkedReferencesButton : Button^
□LinkedReferencesLabel : Label^
«WindowsForm» MainView
Properties
□ViewStack : List<UserControl^>
□ChangesPending : bool
Overridden Properties
◇Text : String^
Events
◇MainView_Load() : System::Void
◇MainView_VisibleChanged() : System::Void
Controls
□Contents : Panel^
□BackButton : Button^
□ApplyButton : Button^
□ExitButton : Button^
1
composed of
{1}
displays
1
{1}
composed of
1
composed of
{1}
«UserControl» TicketsView
Controls
□Panel : FlowLayoutPanel^
□Filtering : TicketsFilterView^
□List : TicketsListView^
launches
«UserControl» TicketView
Controls
□Panel : FlowLayoutPanel^
□Filtering : TicketDetailsView^
□List : TicketEventsListView^
launches
1
1
1
1
composed of
composed of
composed of
composed of
composed of
{1}
{1}
{1}
{1}
«UserControl» TicketDetailsView
Functions
□RefreshFromStatus() : void
Events
◇StatusDropDown_SelectionChanged() : System::Void
◇PrimaryEngineerDropDown_SelectionChanged() : System::Void
◇PrimaryContactDropDown_SelectionChanged() : System::Void
◇AwaitingEngineerDropDown_SelectionChanged() : System::Void
◇AwaitingContactDropDown_SelectionChanged() : System::Void
◇OverviewText_TextChanged() : System::Void
◇Notes_TextChanged() : System::Void
◇AwaitingReasonText_TextChanged() : System::Void
◇LinkedFilesButton_Clicked() : System::Void
◇LinkedReferencesButton_Clicked() : System::Void
Controls
□IDLabel : Label^
□StatusDropDown : ComboBox^
□ContractNumberLabel : Label^
□CustomerNameLabel : Label^
□BuildAndNumberLabel : Label^
□PrimaryEngineerDropDown : ComboBox^
□PrimaryContactDropDown : ComboBox^
□OverviewText : TextBox^
□Notes : TextBox^
□AwaitingReasonText : TextBox^
□AwaitingEngineerDropDown : ComboBox^
□AwaitingContactDropDown : ComboBox^
□DurationLabel : Label^
□LinkedFilesButton : Button^
□LinkedFilesLabel : Label^
□LinkedReferencesButton : Button^
□LinkedReferencesLabel : Label^
«UserControl» TicketsListView
Events
◇TicketsGridContext_ItemClicked() : System::Void
◇ViewTicketButton_Clicked() : System::Void
◇AddTicketButton_Clicked() : System::Void
◇RemoveTicketButton_Clicked() : System::Void
Controls
□TicketsGrid : DataGridView^
□ViewTicketButton : Button^
□AddTicketButton : Button^
□RemoveTicketButton : Button^
□TicketsGridContext : ContextMenuStrip^
□ContextViewTicket : MenuItem^
□ContextRemoveTicket : MenuItem^
«UserControl» TicketsFilterView
Events
◇TicketOpenCheck_CheckedChanged() : System::Void
◇StatusDropDown_SelectedChanged() : System::Void
◇TypeDropDown_SelectedChanged() : System::Void
◇EngineerDropDown_SelectedChanged() : System::Void
◇PrimaryEngineerDropDown_SelectedChanged() : System::Void
◇ContractText_TextChanged() : System::Void
◇CustomerText_TextChanged() : System::Void
Controls
□TicketOpenCheck : CheckBox^
□StatusDropDown : ComboBox^
□TypeDropDown : ComboBox^
□EngineerDropDown : ComboBox^
□PrimaryEngineerDropDown : ComboBox^
□ContractText : TextBox^
□CustomerText : TextBox^
«UserControl» TicketEventsListView
Functions
□ViewEvent() : void
□AddEvent() : void
□RemoveEvent() : void
Events
◇TicketEventsGridCell_DoubleClicked() : System::Void
◇TicketEventsGridContext_ItemClicked() : System::Void
◇ViewEventButton_Clicked() : System::Void
◇AddEventButton_Clicked() : System::Void
◇RemoveEventButton_Clicked() : System::Void
Controls
□EventTypesGrid : DataGridView^
□EventEngineersGrid : DataGridView^
□EventProgramsGrid : DataGridView^
□TicketEventsGrid : DataGridView^
□ViewEventButton : Button^
□AddEventButton : Button^
□RemoveEventButton : Button^
□GridContext : ContextMenuStrip^
□ContextViewEvent : MenuItem^
□ContextRemoveEvent : MenuItem^
inherits from
inherits from
inherits from
inherits from
inherits_from
inherits_from
inherits_from
«Template» BaseUserControl
Properties
Functions
□InitialiseControls() : void
Events
□Load() : System::Void
Inherits from System::Windows::Forms::UserControl.

4.2.1 Tickets View

The `Tickets View` page details a key user interface component of the `Support Tracker`. This view plays a pivotal role in the system, offering users an interactive way to peruse and manage support tickets.

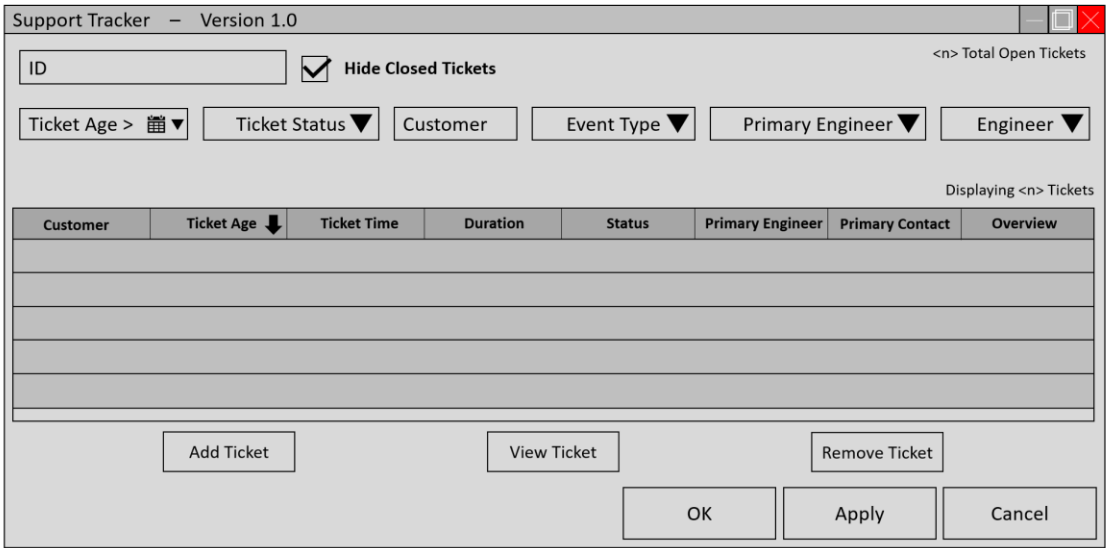
`Tickets View` is a user control that essentially comprises two integral parts:

- **Tickets Filter View:**
  This component facilitates the filtering of tickets based on various criteria, ensuring users can quickly pinpoint specific tickets. Filtering capabilities encompass checks for ticket status, type, associated engineers, primary engineers, contract details, and customer-related information.

- **Tickets List View:**
  This is the crux of the ticket management experience. It's a data grid view where individual tickets are listed. Within this view tickets can be viewed, added, or removed. These actions can be instigated from buttons or with an associated context menu.

It's worth noting that all these controls have their foundation in the `Base User Control`, inheriting basic functionalities and properties from it.

Whether you're navigating to a particular ticket or filtering through a multitude of them, the `Tickets View` is the heart of the ticketing operations in `Support Tracker`.



Support Tracker – Version 1.0

ID

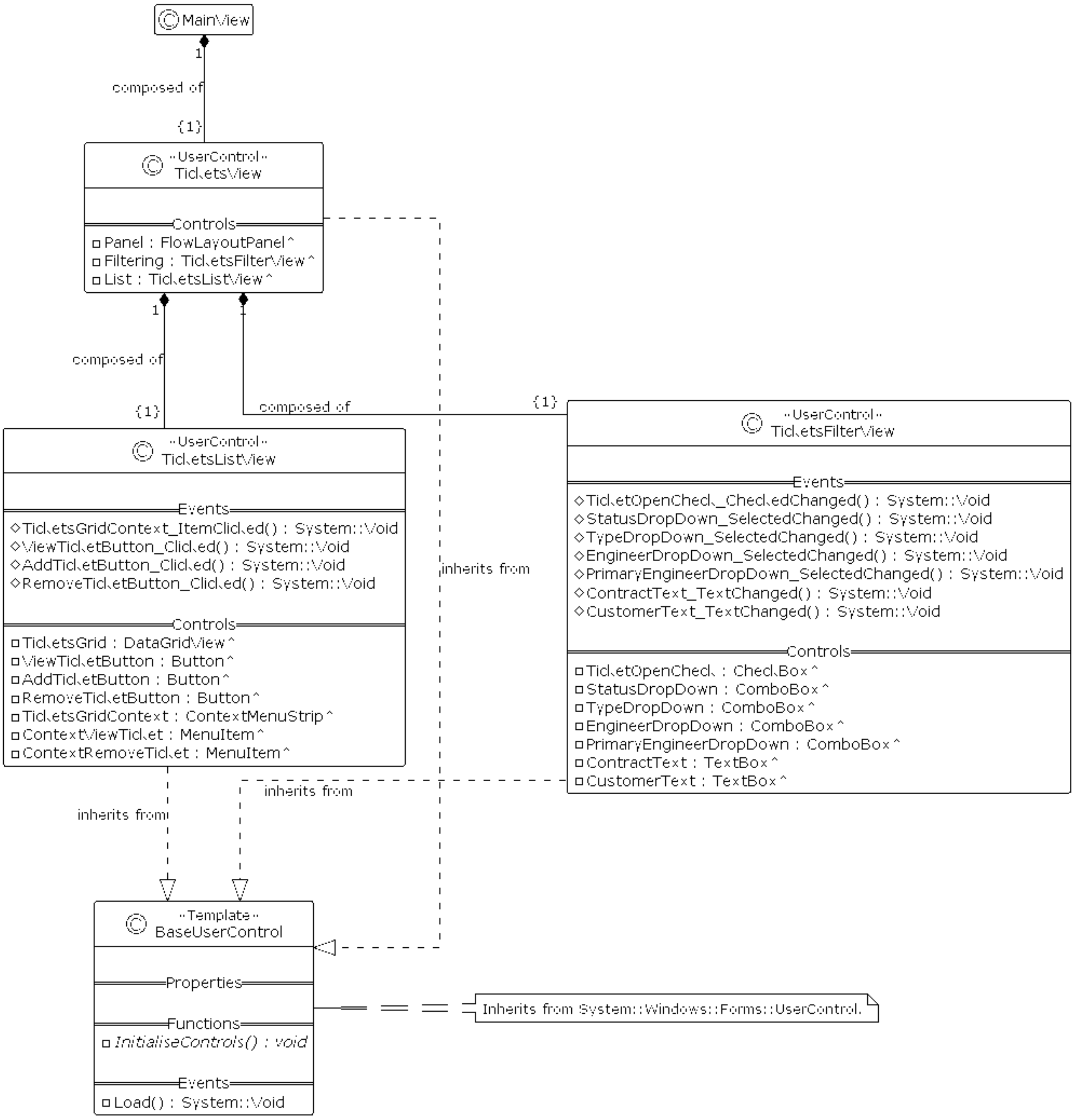
☑ Hide Closed Tickets

<n> Total Open Tickets

Displaying <n> Tickets

Customer	Ticket Age ⬇	Ticket Time	Duration	Status	Primary Engineer	Primary Contact	Overview

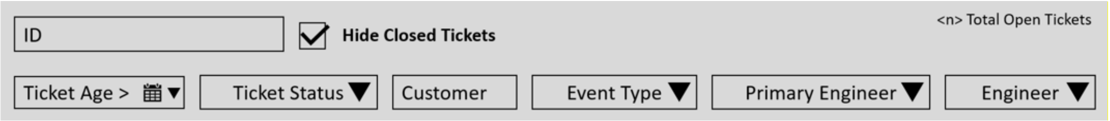
Support Tracker - User Interface - Tickets View



MainView
1
composed of
{1}
«UserControl»
TicketsView
Controls
□ Panel : FlowLayoutPanel ^
□ Filtering : TicketsFilterView ^
□ List : TicketsListView ^
1
1
composed of
{1}
composed of
{1}
«UserControl»
TicketsListView
Events
◇TicketsGridContext_ItemClicked() : System::Void
◇ViewTicketButton_Clicked() : System::Void
◇AddTicketButton_Clicked() : System::Void
◇RemoveTicketButton_Clicked() : System::Void
Controls
□ TicketsGrid : DataGridView ^
□ ViewTicketButton : Button ^
□ AddTicketButton : Button ^
□ RemoveTicketButton : Button ^
□ TicketsGridContext : ContextMenuStrip ^
□ ContextViewTicket : MenuItem ^
□ ContextRemoveTicket : MenuItem ^
«UserControl»
TicketsFilterView
Events
◇TicketOpenCheck_CheckedChanged() : System::Void
◇StatusDropDown_SelectedChanged() : System::Void
◇TypeDropDown_SelectedChanged() : System::Void
◇EngineerDropDown_SelectedChanged() : System::Void
◇PrimaryEngineerDropDown_SelectedChanged() : System::Void
◇ContractText_TextChanged() : System::Void
◇CustomerText_TextChanged() : System::Void
Controls
□ TicketOpenCheck : CheckBox ^
□ StatusDropDown : ComboBox ^
□ TypeDropDown : ComboBox ^
□ EngineerDropDown : ComboBox ^
□ PrimaryEngineerDropDown : ComboBox ^
□ ContractText : TextBox ^
□ CustomerText : TextBox ^
inherits from
inherits from
inherits from
«Template»
BaseUserControl
Properties
Functions
□ *InitialiseControls() : void*
Events
□ Load() : System::Void
Inherits from System::Windows::Forms::UserControl.

4.2.1.1 Tickets-Filter View

The `Tickets Filter View` is a user control that offers a suite of filtering options to narrow down the list of displayed support tickets. Through various controls like dropdown menus and text boxes, users can filter tickets based on status, type, assigned engineer, primary engineer, contract number, and customer name.



When a user opens the application, they should be greeted with a form that provides a list of all tickets. This ticket list will have filtering functionality, some of which will be selected by default:

Default Filtering
Open Tickets *(Any Ticket where their status is not closed)*
Tickets where events include the engineer opening the application.
Tickets where the engineer opening the application is listed as the primary engineer.

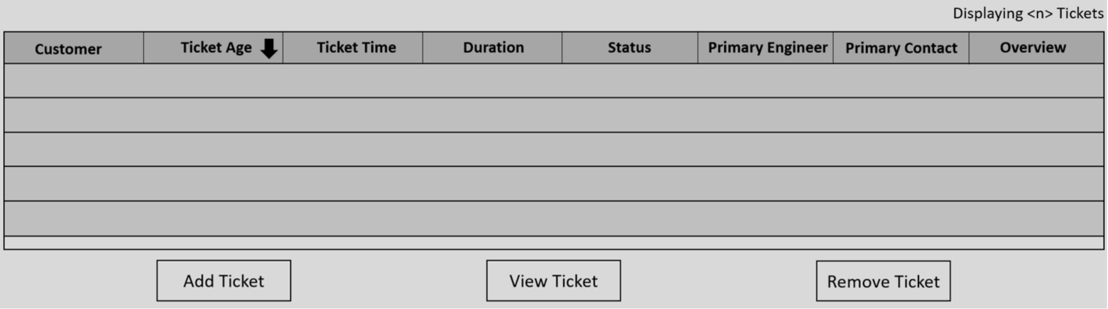
The user will be able to modify any of the following filtering:

Available Filtering
Ticket's age
Ticket's event status *<Can (Inclusive) Select Multiple>*
Ticket's customer
Ticket's event types *<Can (Inclusive) Select Multiple>*
Tickets where the primary engineer is: *<Select Engineer>*
Tickets events where the engineer is: *<Select Engineer>*

TODO: <Insert User Stories of Tickets-Filter View.>

4.2.1.2 Tickets-List View

The `Tickets List View` is a user control designed to present support tickets in a grid format. It provides functionalities for displaying tickets, allowing users to view details of an individual ticket, as well as add, and remove tickets. It also incorporates a context menu for direct actions on the listed tickets.



Customer	Ticket Age ⬇	Ticket Time	Duration	Status	Primary Engineer	Primary Contact	Overview

This view will also have buttons to manage selected tickets, as well as, a context menu to perform the same actions:

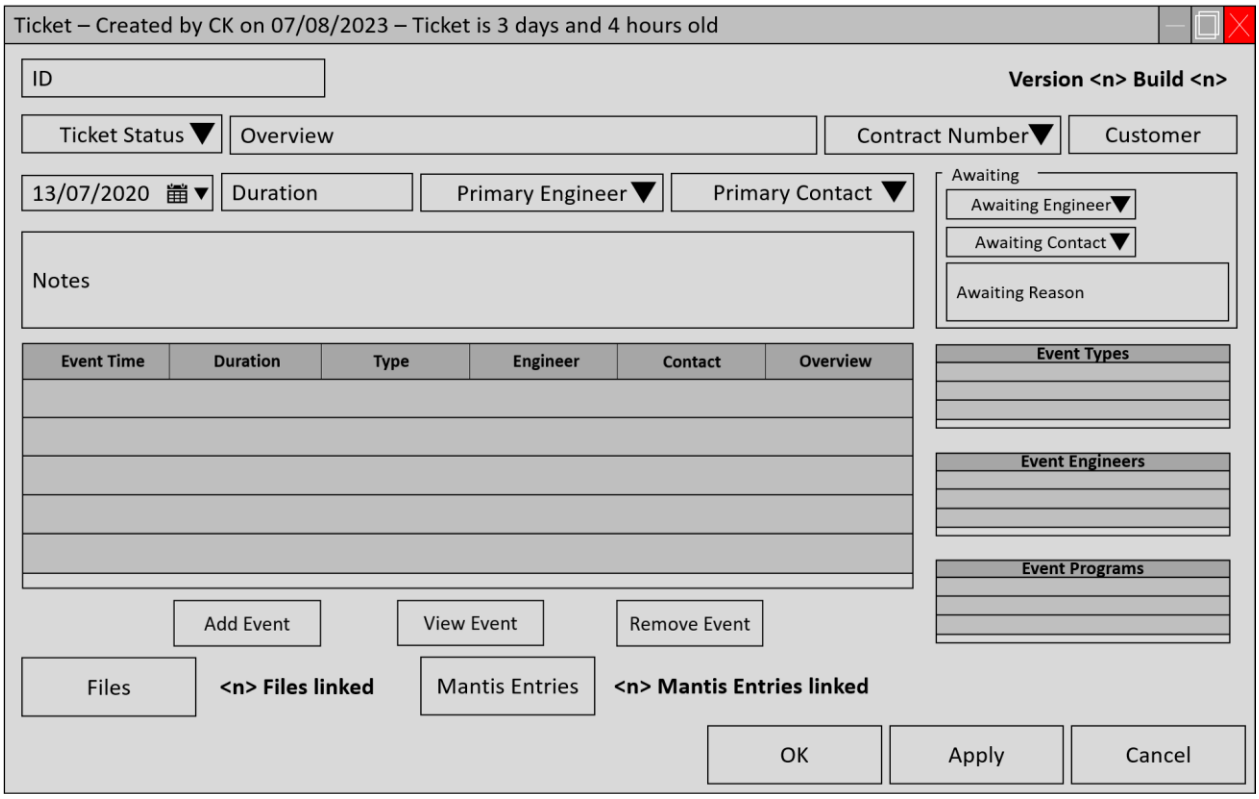
Tickets-List View Actions		
Action	**Button**	**Context Menu**
Add	Yes	No
View	Yes	Yes
Remove	Yes	Yes

TODO: <Insert User Stories of Tickets-List View.>

4.2.2 Ticket View

When a user in the ‘Ticket List View’ either views a selected ticket or creates a ticket, the ‘Ticket View’ will appear in place of the ‘Tickets View’.

The ‘Ticket View’ exists to display and manage all information regarding a ticket. This includes ticket-specific information, as well as, all of a ticket’s events and their overview information.



Ticket – Created by CK on 07/08/2023 – Ticket is 3 days and 4 hours old

The ticket information that will be accessible in this view will be the following:

Accessible Ticket Information			
Information	**Required**	**Provided by Default**	**Mutable**
Ticket ID	~ *<Calculated>*	Yes	No
Ticket Status	Yes	Yes	Yes
Ticket-Event Types	Yes	No	Yes ~ *(Mutated by Event Types)*
Contract Number	Yes	No	Yes ~ *(Mutates Customer)*
Customer	Yes	No	Yes ~ *(Mutates Contract Number)*
Build and Number	~	Yes ~ *(Provided by the selected customer or contract number)*	No
Primary Engineer	Yes	Yes	Yes
Primary Contact	Yes	No	Yes
Related Programs	No	No	Yes ~ *(Mutated by Event's Programs)*
Overview	Yes	No	Yes
Notes	No	No	Yes
Awaiting Reason	No	No	Yes
Awaiting Engineer	No	No	Yes
Awaiting Contact	No	No	Yes
Ticket Creation Date	~ *<Calculated>*	Yes	No
Ticket Closed Date	~ *<Calculated>*	Yes ~ *(Only If Ticket is Closed)*	No
Ticket Creation Engineer	~ *<Calculated>*	Yes	No
Ticket Closed Engineer	~ *<Calculated>*	Yes ~ *(Only If Ticket is Closed)*	No
Time Since Ticket Creation	~ *<Calculated>*	Yes	No
Ticket Duration	~ *<Calculated>*	Yes	No
Linked Files	No	No	Yes ~ *(Also mutated by last Ticket Event)*
Linked Mantis References	No	No	Yes ~ *(Also mutated by last Ticket Event)*

This view will not only display information specific to a ticket, but also a list of all of the events related to a ticket. Each ticket event displayed in the list will appear with an overview of their information and all will be ordered in the list by timestamp.

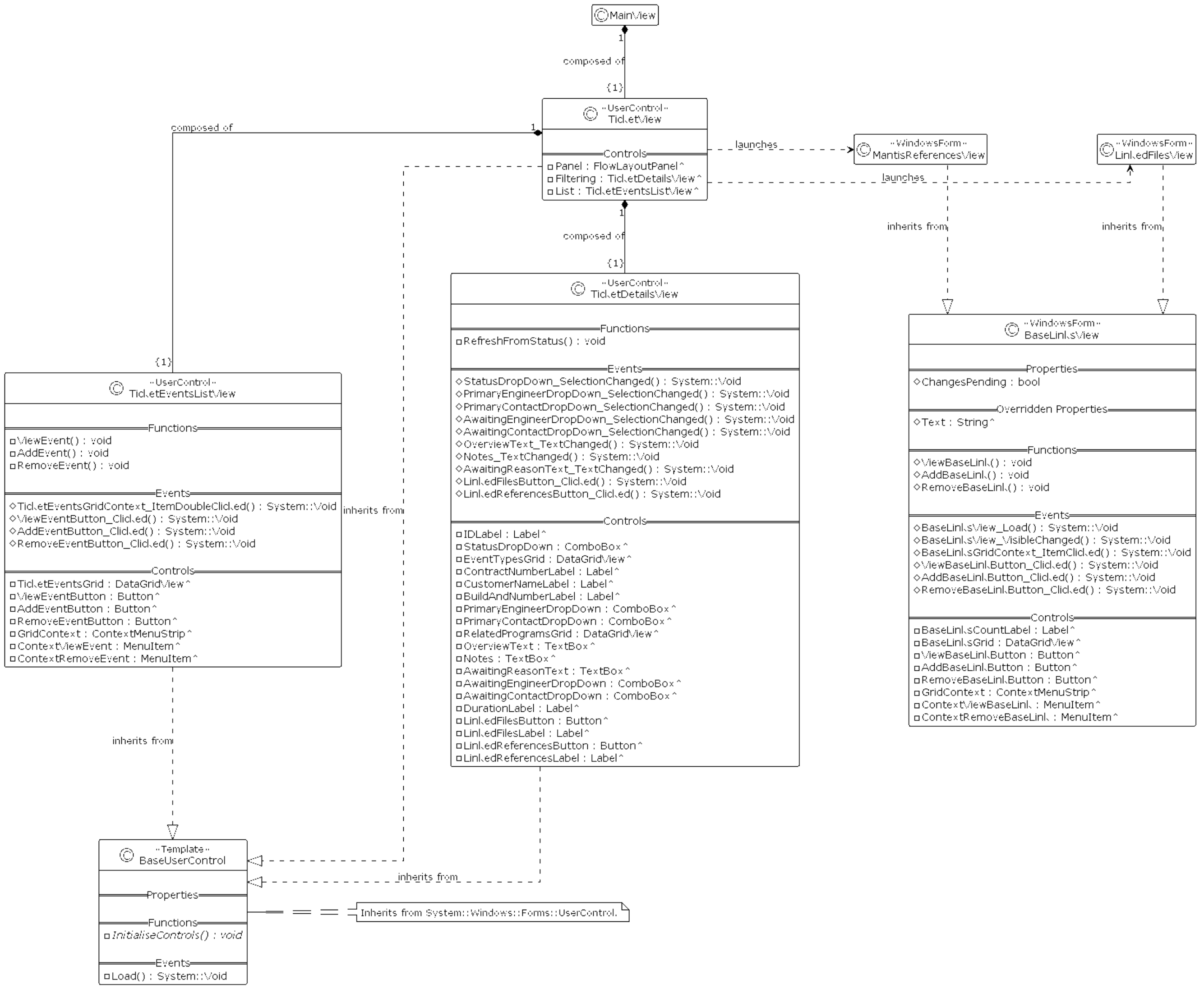
The user has the ability to assign files to the ticket and the ticket's events. All files assigned to any ticket events will also be assigned to the event's ticket.

The user also has the ability to assign Mantis references to the ticket and the ticket's events. Like assigned files, all Mantis references assigned to any ticket events will also be assigned to the event's ticket.

This form will also have buttons to manage selected ticket events and do the following:

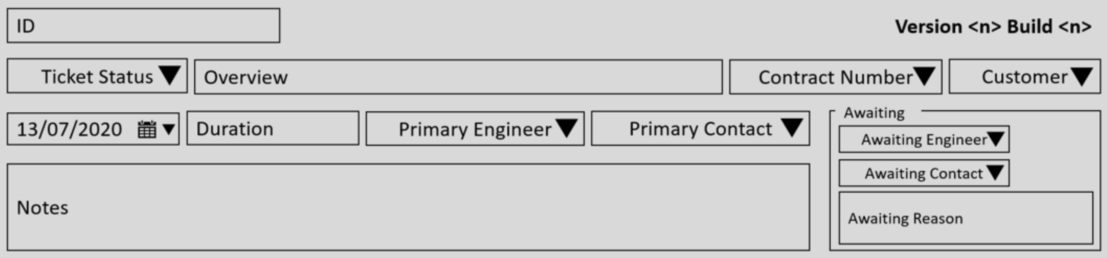
- View Ticket Event
- Add Ticket Event
- Remove Ticket Event

Support Tracker - User Interface - Ticket View


MainView
1
composed of
{1}
«UserControl»
TicketView
Controls
□Panel : FlowLayoutPanel^
□Filtering : TicketDetailsView^
□List : TicketEventsListView^
launches
launches
«WindowsForm»
MantisReferencesView
«WindowsForm»
LinkedFilesView
composed of
1
1
composed of
{1}
inherits from
inherits from
«UserControl»
TicketDetailsView
Functions
□RefreshFromStatus() : void
Events
◇StatusDropDown_SelectionChanged() : System::Void
◇PrimaryEngineerDropDown_SelectionChanged() : System::Void
◇PrimaryContactDropDown_SelectionChanged() : System::Void
◇AwaitingEngineerDropDown_SelectionChanged() : System::Void
◇AwaitingContactDropDown_SelectionChanged() : System::Void
◇OverviewText_TextChanged() : System::Void
◇Notes_TextChanged() : System::Void
◇AwaitingReasonText_TextChanged() : System::Void
◇LinkedFilesButton_Clicked() : System::Void
◇LinkedReferencesButton_Clicked() : System::Void
Controls
□IDLabel : Label^
□StatusDropDown : ComboBox^
□EventTypesGrid : DataGridView^
□ContractNumberLabel : Label^
□CustomerNameLabel : Label^
□BuildAndNumberLabel : Label^
□PrimaryEngineerDropDown : ComboBox^
□PrimaryContactDropDown : ComboBox^
□RelatedProgramsGrid : DataGridView^
□OverviewText : TextBox^
□Notes : TextBox^
□AwaitingReasonText : TextBox^
□AwaitingEngineerDropDown : ComboBox^
□AwaitingContactDropDown : ComboBox^
□DurationLabel : Label^
□LinkedFilesButton : Button^
□LinkedFilesLabel : Label^
□LinkedReferencesButton : Button^
□LinkedReferencesLabel : Label^
«WindowsForm»
BaseLinksView
Properties
◇ChangesPending : bool
Overridden Properties
◇Text : String^
Functions
◇ViewBaseLink() : void
◇AddBaseLink() : void
◇RemoveBaseLink() : void
Events
◇BaseLinksView_Load() : System::Void
◇BaseLinksView_VisibleChanged() : System::Void
◇BaseLinksGridContext_ItemClicked() : System::Void
◇ViewBaseLinkButton_Clicked() : System::Void
◇AddBaseLinkButton_Clicked() : System::Void
◇RemoveBaseLinkButton_Clicked() : System::Void
Controls
□BaseLinksCountLabel : Label^
□BaseLinksGrid : DataGridView^
□ViewBaseLinkButton : Button^
□AddBaseLinkButton : Button^
□RemoveBaseLinkButton : Button^
□GridContext : ContextMenuStrip^
□ContextViewBaseLink : MenuItem^
□ContextRemoveBaseLink : MenuItem^
{1}
«UserControl»
TicketEventsListView
Functions
□ViewEvent() : void
□AddEvent() : void
□RemoveEvent() : void
Events
◇TicketEventsGridContext_ItemDoubleClicked() : System::Void
◇ViewEventButton_Clicked() : System::Void
◇AddEventButton_Clicked() : System::Void
◇RemoveEventButton_Clicked() : System::Void
Controls
□TicketEventsGrid : DataGridView^
□ViewEventButton : Button^
□AddEventButton : Button^
□RemoveEventButton : Button^
□GridContext : ContextMenuStrip^
□ContextViewEvent : MenuItem^
□ContextRemoveEvent : MenuItem^
inherits from
inherits from
inherits from
«Template»
BaseUserControl
Properties
Functions
□*InitialiseControls() : void*
Events
□Load() : System::Void
Inherits from System::Windows::Forms::UserControl.

4.2.2.1 Ticket-Details View

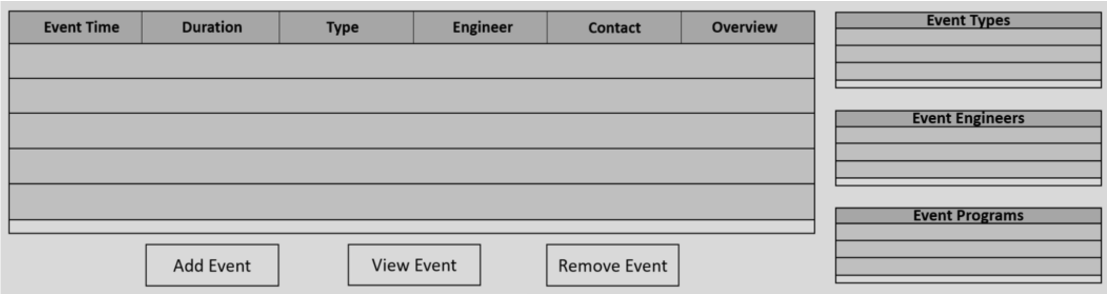
The `Ticket Details View` user control presents core information about a specific support ticket. It encompasses elements that display ticket identifiers, customer details, and status indicators. Key functionalities include status changes, noting the primary engineer and contact associated, and input fields for an overview and additional notes.



TODO: <Insert User Stories of Ticket-Details View.>

4.2.2.2 Ticket-Events-List View

The `Ticket Events List View` user control chronicles the sequence of events or actions associated with a particular support ticket. Presented in a grid format, it offers users the capability to view, add, or remove specific events. Interactive elements include buttons and context menus for streamlined event management. It also provides lists of all event types, engineers, and programs encompassed by the ticket's events.

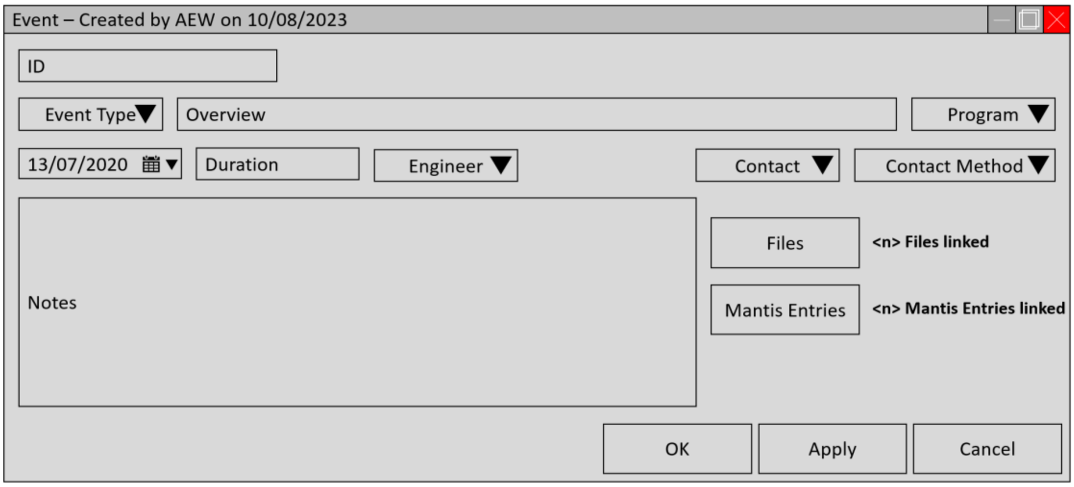


TODO: <Insert User Stories of Ticket-Events-List View.>

4.2.3 Event View

When a user in the ‘Ticket View’ either views a selected ticket event or creates a ticket event, the ‘Event View’ will appear in place of the ‘Ticket View’.

The ‘Event View’ exists to display and manage all information regarding a ticket event. This includes event-specific information.



Event – Created by AEW on 10/08/2023

13/07/2020

The event information that will be accessible in this view will be the following:

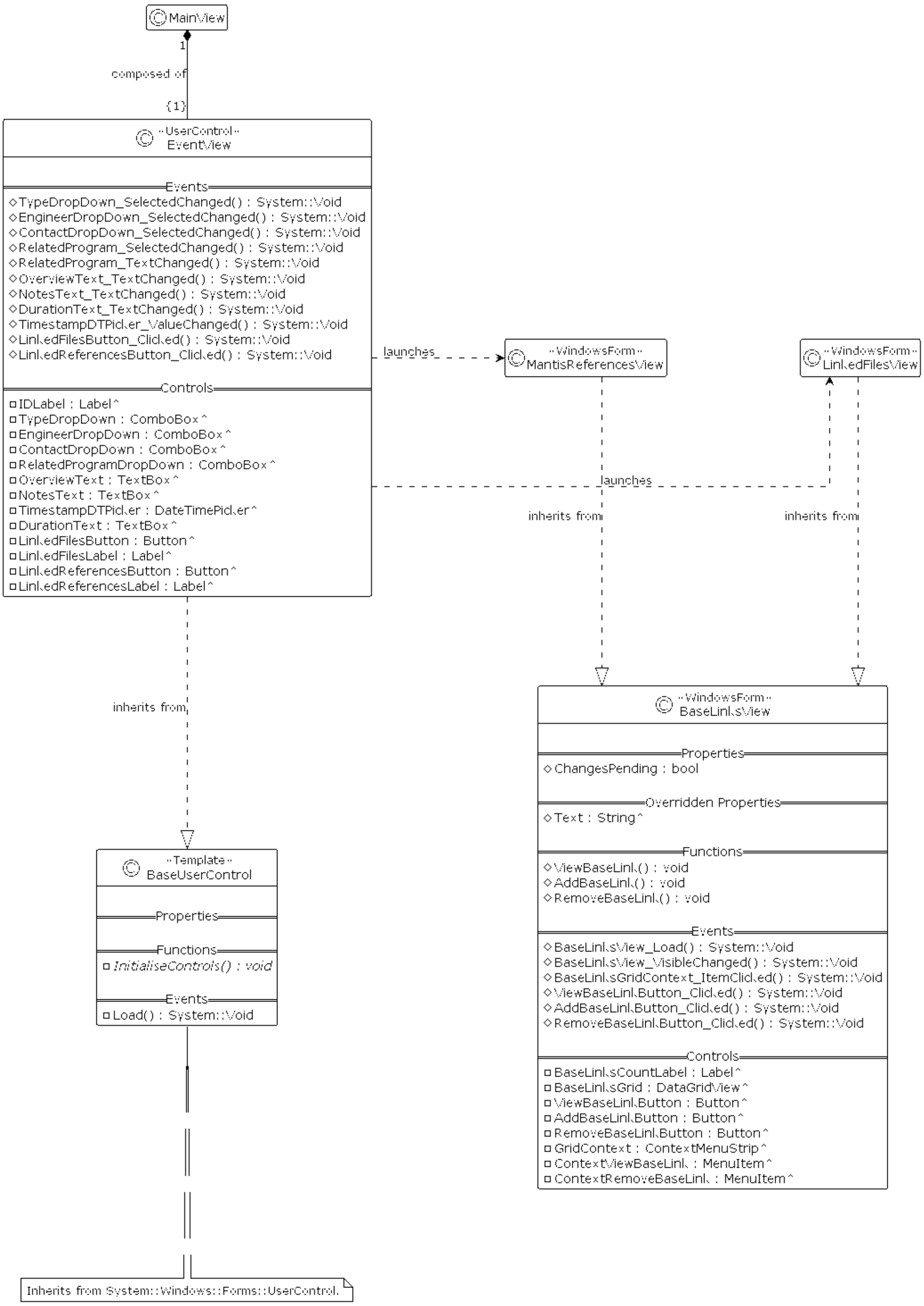
Accessible Event Information			
Information	**Required**	**Provided by Default**	**Mutable**
Event ID	~ *<Calculated>*	Yes	No
Event Types	Yes	No	Yes
Event Engineer	Yes	Yes	Yes
Event Contact	Yes	No	Yes
Related Program	No	No	Yes
Overview	Yes	No	Yes
Notes	No	No	Yes
Event Creation Date	~ *<Calculated>*	Yes	No
Event Creation Engineer	~ *<Calculated>*	Yes	No
Event Timestamp	Yes	Yes	Yes
Event Duration	Yes	Yes	Yes
Linked Files	No	No	Yes
Linked Mantis Reference	No	No	Yes

The user has the ability to assign files to the event, in doing this the file is also assigned to the ticket.

The user also has the ability to assign Mantis references to the event, like assigned files, all Mantis references assigned to any events will also be assigned to the event's ticket.

TODO: <Insert User Stories of Event View.>

Support Tracker - User Interface - Event View



MainView
1
composed of
{1}
«UserControl»
EventView
Events
◇TypeDropDown_SelectedChanged() : System::Void
◇EngineerDropDown_SelectedChanged() : System::Void
◇ContactDropDown_SelectedChanged() : System::Void
◇RelatedProgram_SelectedChanged() : System::Void
◇RelatedProgram_TextChanged() : System::Void
◇OverviewText_TextChanged() : System::Void
◇NotesText_TextChanged() : System::Void
◇DurationText_TextChanged() : System::Void
◇TimestampDTPicker_ValueChanged() : System::Void
◇LinkedFilesButton_Clicked() : System::Void
◇LinkedReferencesButton_Clicked() : System::Void
Controls
□IDLabel : Label^
□TypeDropDown : ComboBox^
□EngineerDropDown : ComboBox^
□ContactDropDown : ComboBox^
□RelatedProgramDropDown : ComboBox^
□OverviewText : TextBox^
□NotesText : TextBox^
□TimestampDTPicker : DateTimePicker^
□DurationText : TextBox^
□LinkedFilesButton : Button^
□LinkedFilesLabel : Label^
□LinkedReferencesButton : Button^
□LinkedReferencesLabel : Label^
launches
«WindowsForm»
MantisReferencesView
«WindowsForm»
LinkedFilesView
launches
inherits from
inherits from
inherits from
«WindowsForm»
BaseLinksView
Properties
◇ChangesPending : bool
Overridden Properties
◇Text : String^
Functions
◇ViewBaseLink() : void
◇AddBaseLink() : void
◇RemoveBaseLink() : void
Events
◇BaseLinksView_Load() : System::Void
◇BaseLinksView_VisibleChanged() : System::Void
◇BaseLinksGridContext_ItemClicked() : System::Void
◇ViewBaseLinkButton_Clicked() : System::Void
◇AddBaseLinkButton_Clicked() : System::Void
◇RemoveBaseLinkButton_Clicked() : System::Void
Controls
□BaseLinksCountLabel : Label^
□BaseLinksGrid : DataGridView^
□ViewBaseLinkButton : Button^
□AddBaseLinkButton : Button^
□RemoveBaseLinkButton : Button^
□GridContext : ContextMenuStrip^
□ContextViewBaseLink : MenuItem^
□ContextRemoveBaseLink : MenuItem^
«Template»
BaseUserControl
Properties
Functions
□*InitialiseControls() : void*
Events
□Load() : System::Void
Inherits from System::Windows::Forms::UserControl.

5. Domain Model

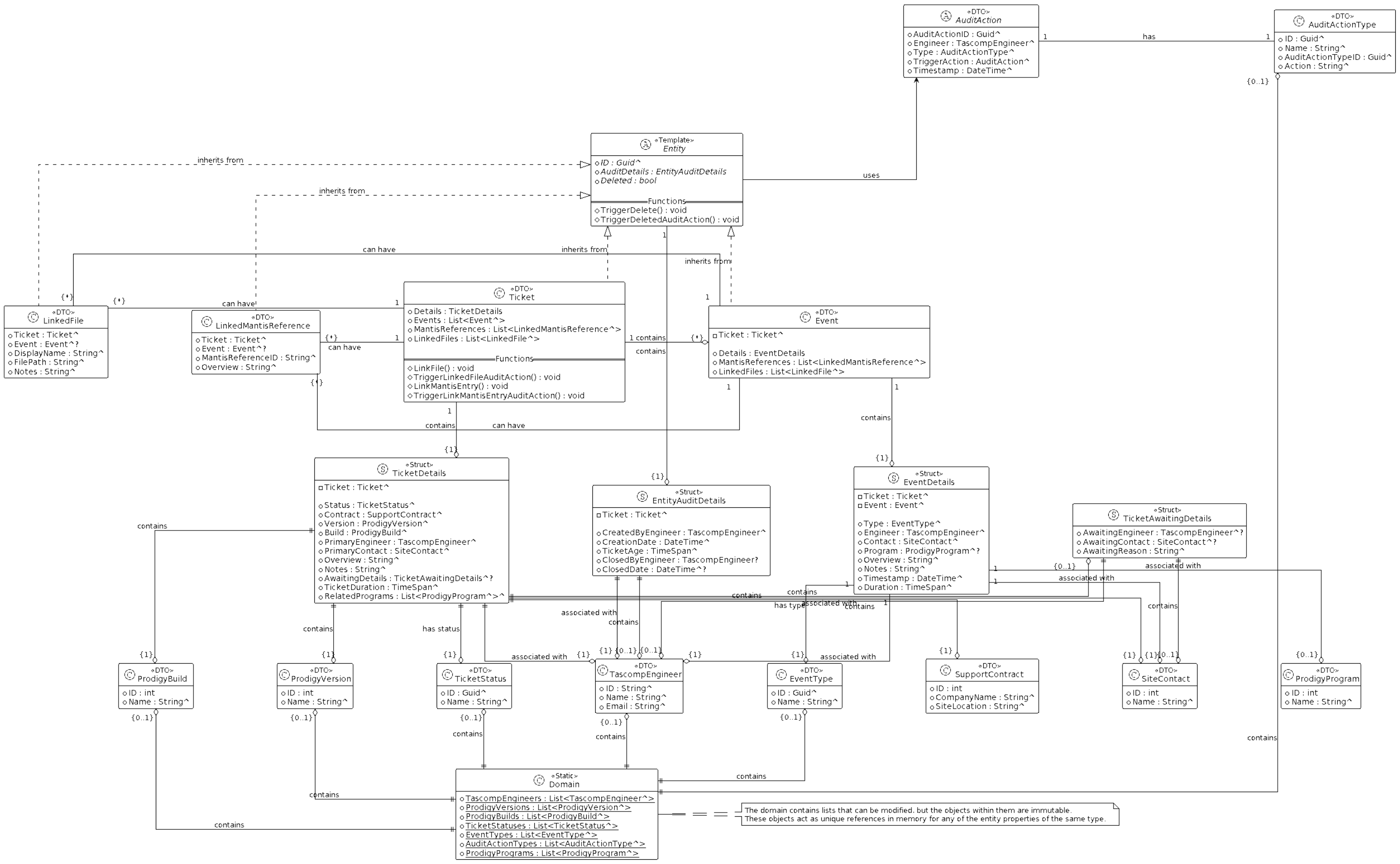
The domain model lays out the foundation of how the system's components interrelate and function cohesively. Presented in a UML diagram, it delineates the key classes and structures that constitute the system's architecture, highlighting both data and behaviours.

- **Core Entities:**
  - **Ticket:**
    Central to the system, a ticket captures details of a particular support case, including its status, associated events, linked files, and Mantis references.
  - **Event:**
    Each ticket can have multiple events, capturing specific actions or updates related to that ticket.
  - **Linked Mantis Reference:**
    This denotes the references from Mantis entries associated with either tickets or events.
  - **Linked File:**
    Files that are attached or linked with tickets and events, providing auxiliary information or evidence.
- **Reference Data:**
  The system maintains several DTO classes that act more as dictionaries or lists:
  - **Prodigy Build and Prodigy Version:**
    Represent various versions and builds of the Prodigy software.
  - **Support Contract:**
    Details the contracts with various companies, highlighting the site location.
  - **Ticket Status, Event Type, and Audit Action Type:**
    Enumerate the possible statuses of a ticket, types of events, and types of audit actions, respectively.
  - **Prodigy Program:**
    Denotes different software programmes under the Prodigy brand.
  - **Tascomp Engineer:**
    Engineers from Tascomp, who interact with the support tickets.
    This could be anyone from Applications, Sales, or Software.
  - **Site Contact:**
    Contacts from the client side, this would be whomever is the contact

- **Audit Mechanism:**
  - The system prioritises tracking and accountability. At its heart is the `AuditAction` which logs key actions, linking them back to engineers and the type of action.
  - All major entities like `Ticket`, `Event`, `LinkedMantisReference`, and `LinkedFile` are derivatives of a generic Entity class, ensuring that core attributes and audit functionalities are consistently maintained.
- **Entity Details:**
  Several structs detail the various attributes and relationships of the core entities:
  - **Ticket Details:**
    Encompasses all the intricate details of a ticket.
  - **Event Details:**
    Highlights what an event is about, its type, the engineer and contact involved, amongst other details.
  - **Entity Audit Details:**
    Encapsulates auditing details for any entity.
- **Domain Class:**
  - Acts as a repository, containing lists of various static data entities like engineers, Prodigy versions, ticket statuses, etc. It’s noted that while the lists in the domain can be modified, the objects within these lists remain immutable, acting as unique memory references. This ensures that there are no present duplicates of reference data, and all core entities that use reference data are ultimately pointing to their respective places in the domain.

This domain model, mapped out in the UML, offers a robust and comprehensive overview of how the “Support Tracker” system is structured, guiding both its development and subsequent utilisation. It’s meticulously designed to not just store data but to capture the very essence and workflow of the support process.

Support Tracker - Domain Model Classes



The domain contains lists that can be modified. but the objects within them are immutable.
These objects act as unique references in memory for any of the entity properties of the same type.

6. Database

This element promises to act as a crucial module within our architecture, meticulously archiving pertinent information within an orderly, functional framework, thus enabling proficient data storage, retrieval, and seamless consolidation with other in-house utilities.

Given that the 'Support Tracker' will function as an internal tool, its database will be stationed in uniformity with our other in-house tools. The selected type for this database is the Microsoft SQL Server which necessitates specific credentials for access.

A tabular representation of the required credentials is provided as follows:

Microsoft SQL Server Credentials	
Type	Database Engine
Name	TSC-SERVER\SQLEXPRESS
Authentication	SQL Server Authentication
Login	Prodigy
Password	*<Password>*

6.1.1 Schema

The **Support Tracker** database structure employs a strategic combination of base tables, junction tables, and lookup tables to establish a comprehensive representation of data and its relationships. The following describes the entity relationships present:

1. **Base Tables:**

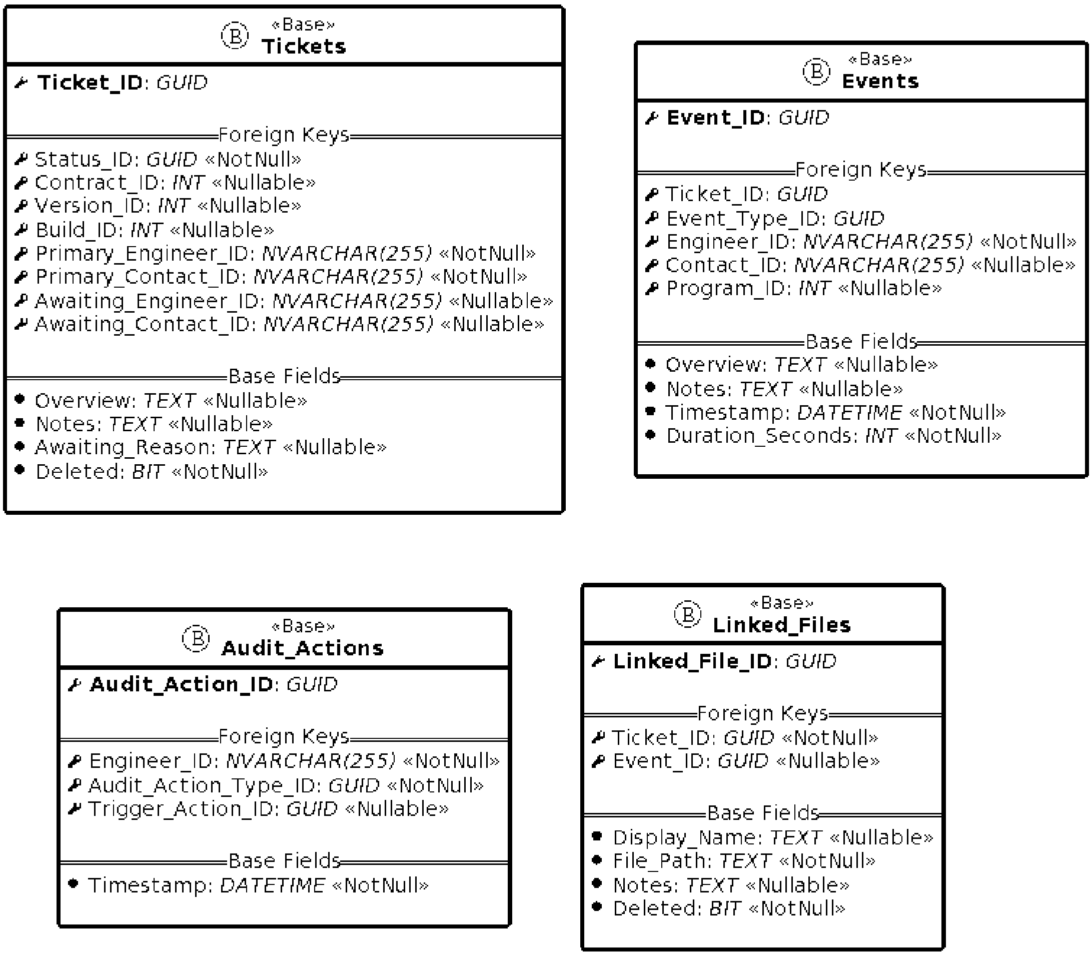
   - **Tickets:**
     A central table representing individual support tickets.

   - **Events:**
     Represents specific events or actions taken on tickets.

   - **Audit_Actions:**
     Captures audit trail actions.

   - **Linked_Files:**
     Contains links to files linked either to a ticket, and potentially an event.

Support Tracker - Database Schema - Base Tables



«Base»
Tickets

Ticket_ID: *GUID*

Foreign Keys
- Status_ID: *GUID* «NotNull»
- Contract_ID: *INT* «Nullable»
- Version_ID: *INT* «Nullable»
- Build_ID: *INT* «Nullable»
- Primary_Engineer_ID: *NVARCHAR(255)* «NotNull»
- Primary_Contact_ID: *NVARCHAR(255)* «NotNull»
- Awaiting_Engineer_ID: *NVARCHAR(255)* «Nullable»
- Awaiting_Contact_ID: *NVARCHAR(255)* «Nullable»

Base Fields
- Overview: *TEXT* «Nullable»
- Notes: *TEXT* «Nullable»
- Awaiting_Reason: *TEXT* «Nullable»
- Deleted: *BIT* «NotNull»

«Base»
Events

Event_ID: *GUID*

Foreign Keys
- Ticket_ID: *GUID*
- Event_Type_ID: *GUID*
- Engineer_ID: *NVARCHAR(255)* «NotNull»
- Contact_ID: *NVARCHAR(255)* «Nullable»
- Program_ID: *INT* «Nullable»

Base Fields
- Overview: *TEXT* «Nullable»
- Notes: *TEXT* «Nullable»
- Timestamp: *DATETIME* «NotNull»
- Duration_Seconds: *INT* «NotNull»

«Base»
Audit_Actions

Audit_Action_ID: *GUID*

Foreign Keys
- Engineer_ID: *NVARCHAR(255)* «NotNull»
- Audit_Action_Type_ID: *GUID* «NotNull»
- Trigger_Action_ID: *GUID* «Nullable»

Base Fields
- Timestamp: *DATETIME* «NotNull»

«Base»
Linked_Files

Linked_File_ID: *GUID*

Foreign Keys
- Ticket_ID: *GUID* «NotNull»
- Event_ID: *GUID* «Nullable»

Base Fields
- Display_Name: *TEXT* «Nullable»
- File_Path: *TEXT* «NotNull»
- Notes: *TEXT* «Nullable»
- Deleted: *BIT* «NotNull»

2. **Junction Tables:**
   These facilitate many-to-many relationships among different entities.

   - **Linked_Mantis_References:**
     Bridges the Mantis references to tickets and events.

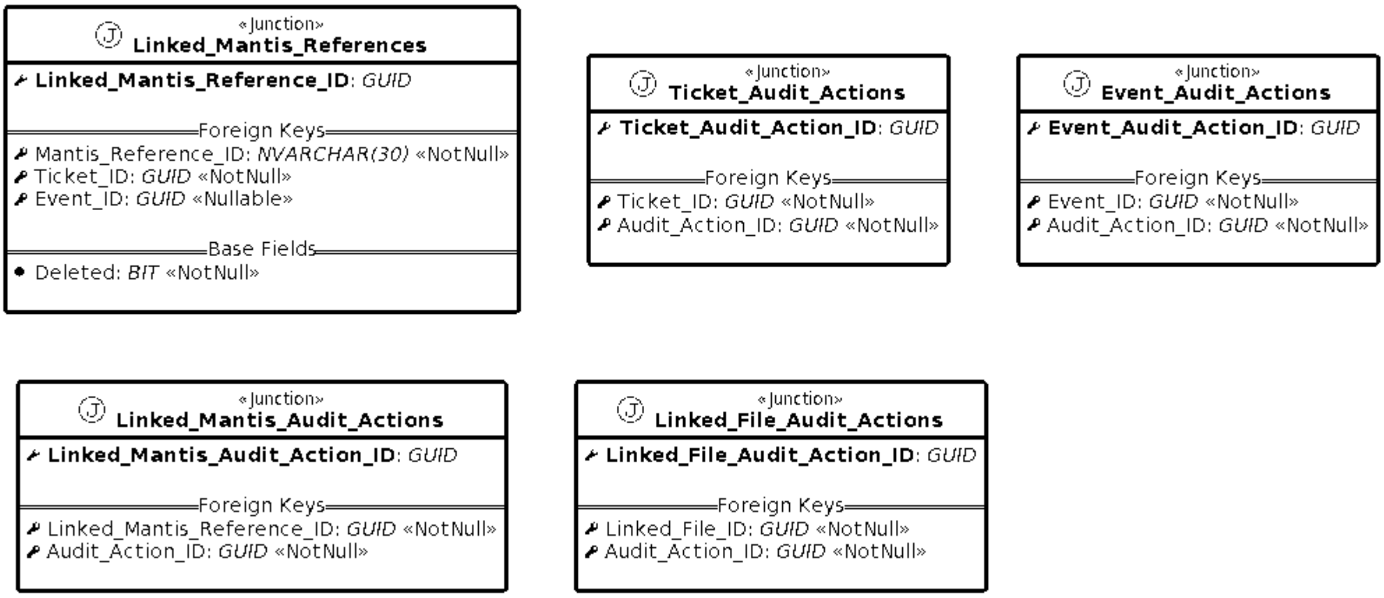
   - **Ticket_Audit_Actions:**
     Connects audit actions to their respective tickets.

   - **Event_Audit_Actions:**
     Relates audit actions to particular events.

   - **Linked_Mantis_Audit_Actions:**
     Associates Mantis references with audit actions.

   - **Linked_File_Audit_Actions:**
     Links file references with audit actions.

Support Tracker - Database Schema - Junction Tables



«Junction»
Linked_Mantis_References
Linked_Mantis_Reference_ID: GUID
Foreign Keys
Mantis_Reference_ID: NVARCHAR(30) «NotNull»
Ticket_ID: GUID «NotNull»
Event_ID: GUID «Nullable»
Base Fields
Deleted: BIT «NotNull»
«Junction»
Ticket_Audit_Actions
Ticket_Audit_Action_ID: GUID
Foreign Keys
Ticket_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»
«Junction»
Event_Audit_Actions
Event_Audit_Action_ID: GUID
Foreign Keys
Event_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»
«Junction»
Linked_Mantis_Audit_Actions
Linked_Mantis_Audit_Action_ID: GUID
Foreign Keys
Linked_Mantis_Reference_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»
«Junction»
Linked_File_Audit_Actions
Linked_File_Audit_Action_ID: GUID
Foreign Keys
Linked_File_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»

3. Lookup Tables:
Employed for static data lookup.

- **Ticket_Statuses:**
  Lists possible statuses for tickets.

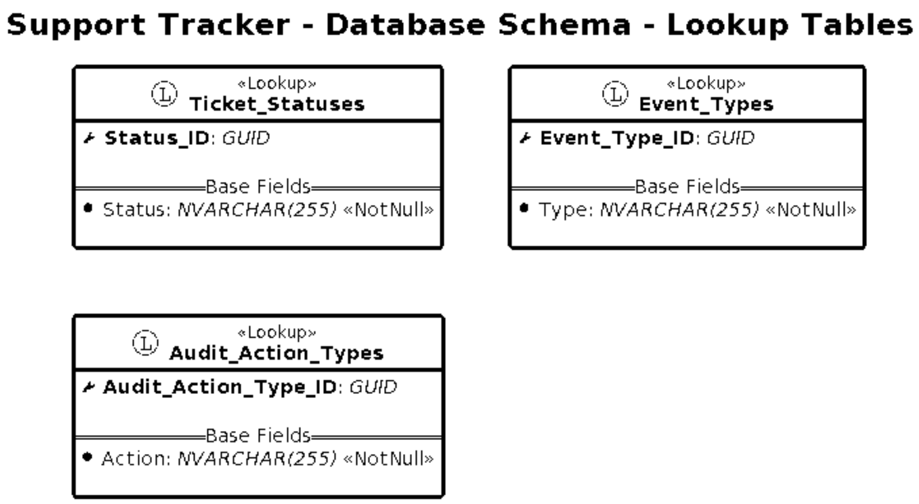
  Expected ticket statuses to be stored in the 'Ticket Statuses' lookup table would initially take after our existing 'Event Statuses' system:
  - Open
  - Customer To Respond
  - Tascomp To Respond
  - Closed

- **Event_Types:**
  Enumerates various event types.

  Expected event types are to be stored in the 'Event Types' lookup table to avoid duplication, and would initially take after our existing 'Support Event Statuses' system:
  - Configuration
  - Hardware
  - Paid Work
  - Prodigy Build
  - Question
  - User Error
  - Development

- **Audit_Action_Types:**
  Lists types of audit actions.

  Expected audit action types are to be stored in the 'Audit Actions Types' lookup table:
  - Create
  - Modify
  - Close
  - Delete



Support Tracker - Database Schema - Lookup Tables
«Lookup»
Ticket_Statuses
Status_ID: GUID
Base Fields
Status: NVARCHAR(255) «NotNull»
«Lookup»
Event_Types
Event_Type_ID: GUID
Base Fields
Type: NVARCHAR(255) «NotNull»
«Lookup»
Audit_Action_Types
Audit_Action_Type_ID: GUID
Base Fields
Action: NVARCHAR(255) «NotNull»

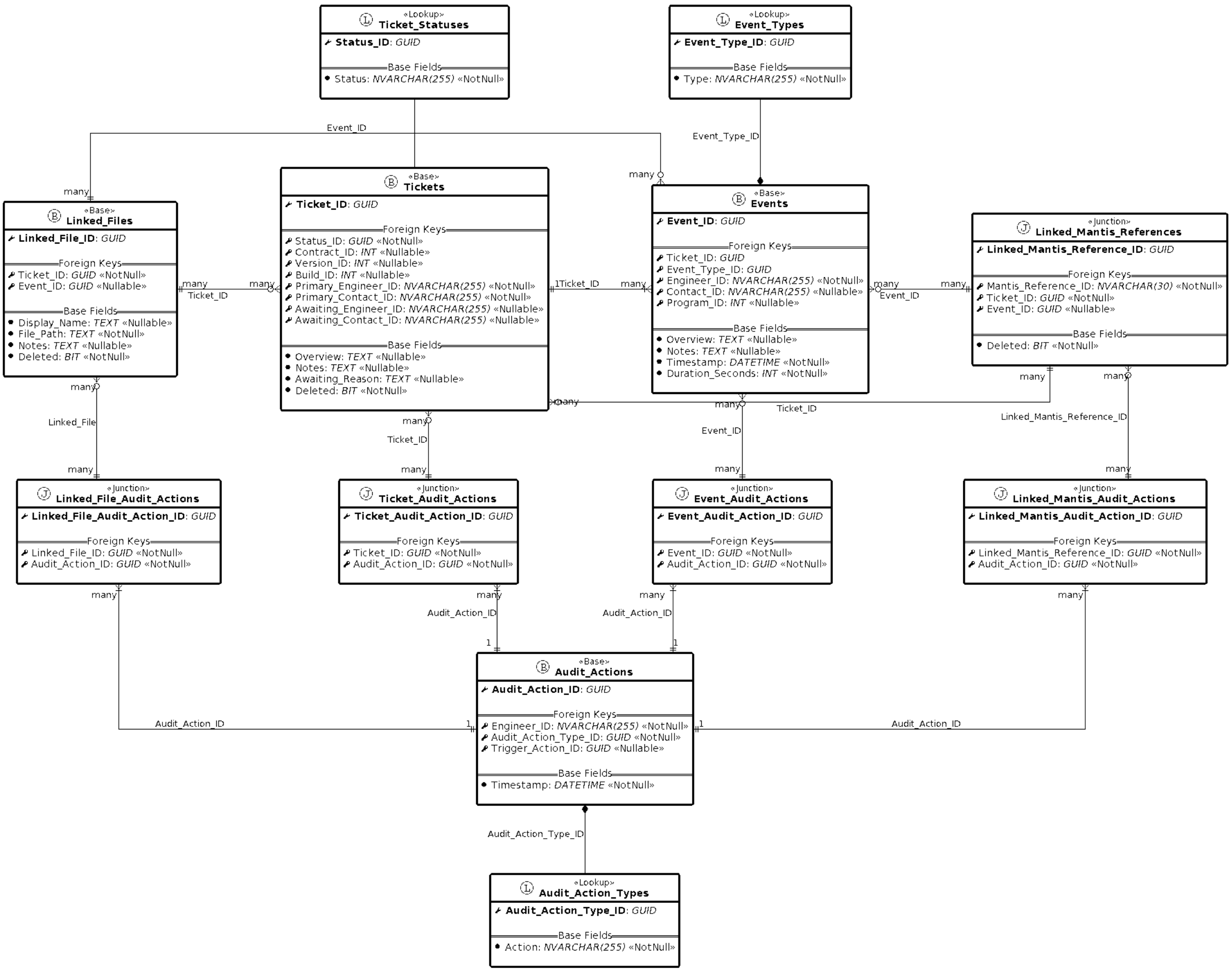
6.1.2 Entity Relationships

In any relational database system, the way data entities interrelate is of paramount importance. In **Support Tracker**, the entity relationships are structured to ensure a fluid flow of information, accurate data representation, and efficient retrieval. The relationships among tables are designed to be intuitive and representative of real-world interactions within the ticketing and support process.

Relationships:

- **Tickets and Events:**
  Every ticket can have multiple events associated with it.
- **Tickets and Linked_Files:**
  Multiple files can be linked to a particular ticket and vice-versa.
- **Events and Linked_Files:**
  Multiple files can be linked to a particular event and vice-versa.
- **Tickets and Linked_Mantis_References:**
  Every ticket can link to multiple Mantis references and vice-versa.
- **Events and Linked_Mantis_References:**
  Each event can link to multiple Mantis references and vice-versa.
- **Audit_Actions:**
  - Audit actions are tied to a specific type via the Audit_Action_Types lookup table.
  - Audit actions are associated with tickets, events, linked Mantis references, and linked files via the respective junction tables.

Support Tracker - Database Table Relationships



«Lookup»
Ticket_Statuses
Status_ID: GUID
Base Fields
Status: NVARCHAR(255) «NotNull»
«Lookup»
Event_Types
Event_Type_ID: GUID
Base Fields
Type: NVARCHAR(255) «NotNull»
Event_ID
Event_Type_ID
many
«Base»
Linked_Files
Linked_File_ID: GUID
Foreign Keys
Ticket_ID: GUID «NotNull»
Event_ID: GUID «Nullable»
Base Fields
Display_Name: TEXT «Nullable»
File_Path: TEXT «NotNull»
Notes: TEXT «Nullable»
Deleted: BIT «NotNull»
many
Ticket_ID
many
«Base»
Tickets
Ticket_ID: GUID
Foreign Keys
Status_ID: GUID «NotNull»
Contract_ID: INT «Nullable»
Version_ID: INT «Nullable»
Build_ID: INT «Nullable»
Primary_Engineer_ID: NVARCHAR(255) «NotNull»
Primary_Contact_ID: NVARCHAR(255) «NotNull»
Awaiting_Engineer_ID: NVARCHAR(255) «Nullable»
Awaiting_Contact_ID: NVARCHAR(255) «Nullable»
Base Fields
Overview: TEXT «Nullable»
Notes: TEXT «Nullable»
Awaiting_Reason: TEXT «Nullable»
Deleted: BIT «NotNull»
1Ticket_ID
many
many
«Base»
Events
Event_ID: GUID
Foreign Keys
Ticket_ID: GUID
Event_Type_ID: GUID
Engineer_ID: NVARCHAR(255) «NotNull»
Contact_ID: NVARCHAR(255) «Nullable»
Program_ID: INT «Nullable»
Base Fields
Overview: TEXT «Nullable»
Notes: TEXT «Nullable»
Timestamp: DATETIME «NotNull»
Duration_Seconds: INT «NotNull»
many
Event_ID
many
«Junction»
Linked_Mantis_References
Linked_Mantis_Reference_ID: GUID
Foreign Keys
Mantis_Reference_ID: NVARCHAR(30) «NotNull»
Ticket_ID: GUID «NotNull»
Event_ID: GUID «Nullable»
Base Fields
Deleted: BIT «NotNull»
many
many
many
Linked_File
many
many
Ticket_ID
many
many
Ticket_ID
Event_ID
many
many
Linked_Mantis_Reference_ID
many
«Junction»
Linked_File_Audit_Actions
Linked_File_Audit_Action_ID: GUID
Foreign Keys
Linked_File_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»
«Junction»
Ticket_Audit_Actions
Ticket_Audit_Action_ID: GUID
Foreign Keys
Ticket_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»
«Junction»
Event_Audit_Actions
Event_Audit_Action_ID: GUID
Foreign Keys
Event_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»
«Junction»
Linked_Mantis_Audit_Actions
Linked_Mantis_Audit_Action_ID: GUID
Foreign Keys
Linked_Mantis_Reference_ID: GUID «NotNull»
Audit_Action_ID: GUID «NotNull»
many
many
Audit_Action_ID
many
Audit_Action_ID
many
1
1
«Base»
Audit_Actions
Audit_Action_ID: GUID
Foreign Keys
Engineer_ID: NVARCHAR(255) «NotNull»
Audit_Action_Type_ID: GUID «NotNull»
Trigger_Action_ID: GUID «Nullable»
Base Fields
Timestamp: DATETIME «NotNull»
Audit_Action_ID
1
1
Audit_Action_ID
Audit_Action_Type_ID
«Lookup»
Audit_Action_Types
Audit_Action_Type_ID: GUID
Base Fields
Action: NVARCHAR(255) «NotNull»

6.1.3 External Databases

The table below outlines all inter-table connections, notably those linking to external databases of other internal tools.

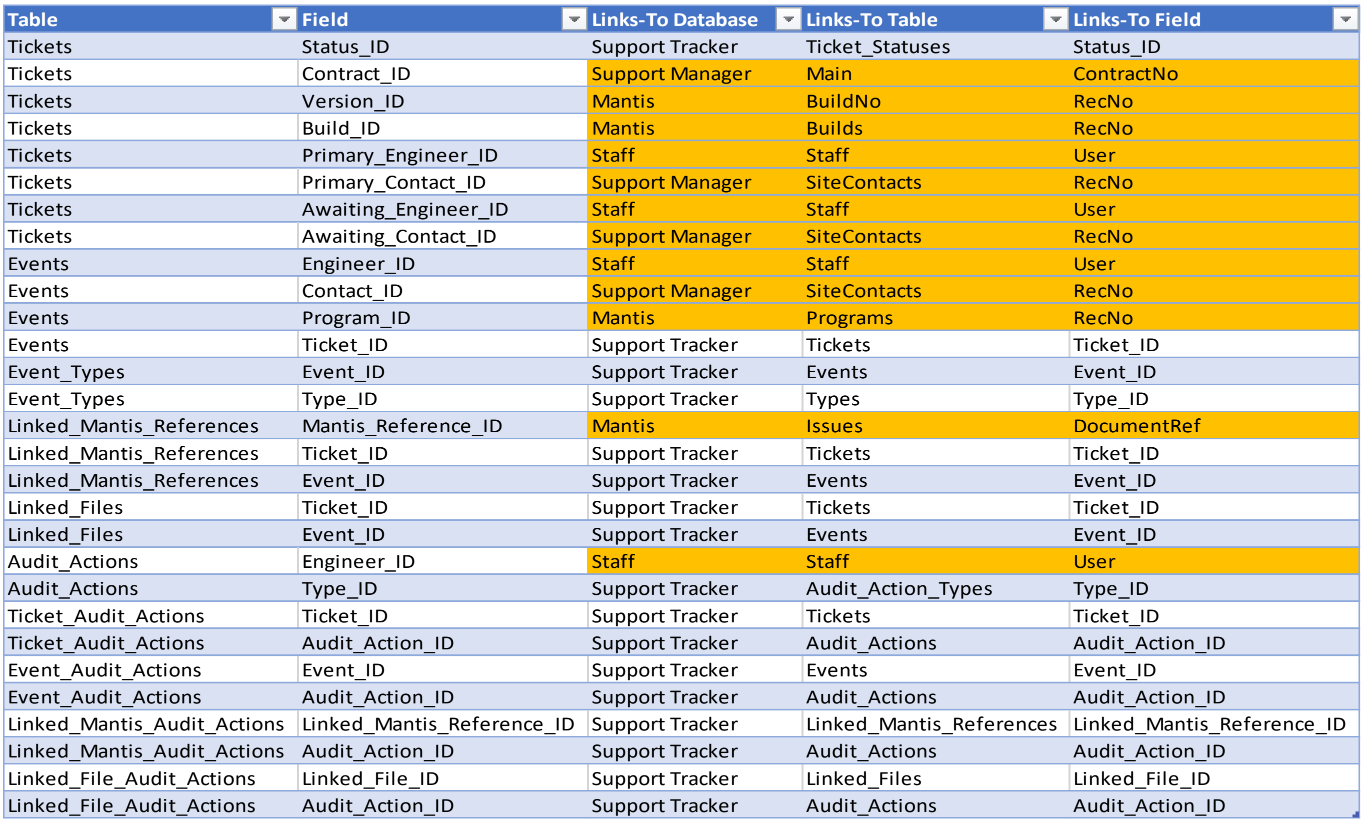


Table	Field	Links-To Database	Links-To Table	Links-To Field
Tickets	Status_ID	Support Tracker	Ticket_Statuses	Status_ID
Tickets	Contract_ID	Support Manager	Main	ContractNo
Tickets	Version_ID	Mantis	BuildNo	RecNo
Tickets	Build_ID	Mantis	Builds	RecNo
Tickets	Primary_Engineer_ID	Staff	Staff	User
Tickets	Primary_Contact_ID	Support Manager	SiteContacts	RecNo
Tickets	Awaiting_Engineer_ID	Staff	Staff	User
Tickets	Awaiting_Contact_ID	Support Manager	SiteContacts	RecNo
Events	Engineer_ID	Staff	Staff	User
Events	Contact_ID	Support Manager	SiteContacts	RecNo
Events	Program_ID	Mantis	Programs	RecNo
Events	Ticket_ID	Support Tracker	Tickets	Ticket_ID
Event_Types	Event_ID	Support Tracker	Events	Event_ID
Event_Types	Type_ID	Support Tracker	Types	Type_ID
Linked_Mantis_References	Mantis_Reference_ID	Mantis	Issues	DocumentRef
Linked_Mantis_References	Ticket_ID	Support Tracker	Tickets	Ticket_ID
Linked_Mantis_References	Event_ID	Support Tracker	Events	Event_ID
Linked_Files	Ticket_ID	Support Tracker	Tickets	Ticket_ID
Linked_Files	Event_ID	Support Tracker	Events	Event_ID
Audit_Actions	Engineer_ID	Staff	Staff	User
Audit_Actions	Type_ID	Support Tracker	Audit_Action_Types	Type_ID
Ticket_Audit_Actions	Ticket_ID	Support Tracker	Tickets	Ticket_ID
Ticket_Audit_Actions	Audit_Action_ID	Support Tracker	Audit_Actions	Audit_Action_ID
Event_Audit_Actions	Event_ID	Support Tracker	Events	Event_ID
Event_Audit_Actions	Audit_Action_ID	Support Tracker	Audit_Actions	Audit_Action_ID
Linked_Mantis_Audit_Actions	Linked_Mantis_Reference_ID	Support Tracker	Linked_Mantis_References	Linked_Mantis_Reference_ID
Linked_Mantis_Audit_Actions	Audit_Action_ID	Support Tracker	Audit_Actions	Audit_Action_ID
Linked_File_Audit_Actions	Linked_File_ID	Support Tracker	Linked_Files	Linked_File_ID
Linked_File_Audit_Actions	Audit_Action_ID	Support Tracker	Audit_Actions	Audit_Action_ID

7. Testing

Ensuring optimal performance, accuracy, and user satisfaction is paramount when developing any application, that doesn't change with Support Tracker. To achieve these goals, a robust and systematic testing strategy has been incorporated. Testing not only highlights areas for potential improvement but also provides stakeholders with the assurance that the system meets its specifications and delivers as promised.

- **Purpose of Testing:**
  - To ensure Support Tracker functions correctly, efficiently, and consistently.
  - To validate that all features align with the user requirements and business objectives.
  - To identify and rectify any defects, inconsistencies, or vulnerabilities before deployment.
- **Types of Testing:**
  - **Unit Testing**
  - **Black-Box Testing**
- **Benefits:**
  - **Quality Assurance:**
    Rigorous testing provides confidence in Support Tracker's reliability and readiness for deployment.
  - **Enhanced User Experience:**
    By identifying and rectifying surprises, users experience a smoother, more efficient system.
  - **Cost-Efficiency:**
    Early bug detection through testing can reduce costs overall, minimising potential rework.

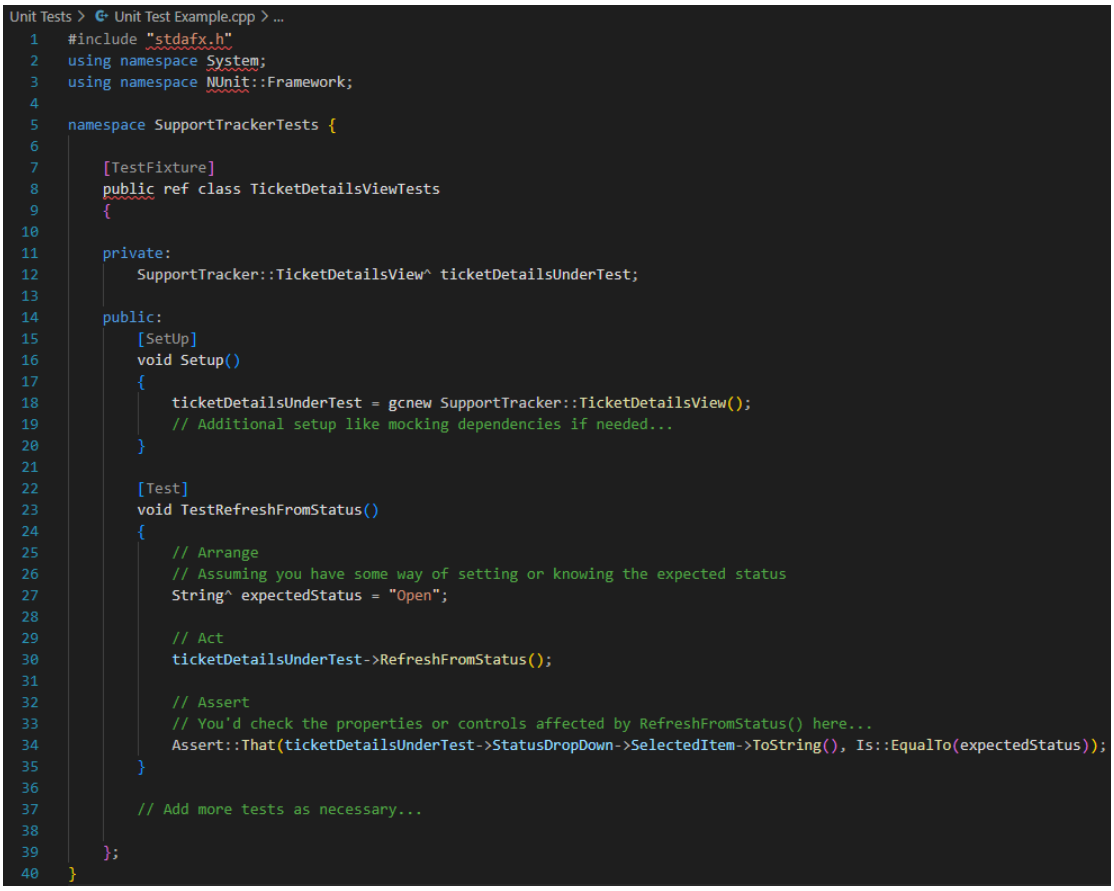
7.1 Unit Tests

Though unit tests are not currently standard within Tascomp’s development process, we may benefit from integrating them into development moving forward.

Unit tests, which focus on individual components like functions, classes, and methods, can help in early detection of errors, streamline debugging, and improve the overall quality of the codebase and increase reliability of builds.

By adopting unit tests, Tascomp could significantly enhance the robustness of Prodigy, since Support Tracker is a green-field project it could be used to set a standard for future projects.

The following is an example of how a unit test may appear:



```
#include "stdafx.h"
using namespace System;
using namespace NUnit::Framework;

namespace SupportTrackerTests {

    [TestFixture]
    public ref class TicketDetailsViewTests
    {

    private:
        SupportTracker::TicketDetailsView^ ticketDetailsUnderTest;

    public:
        [SetUp]
        void Setup()
        {
            ticketDetailsUnderTest = gcnew SupportTracker::TicketDetailsView();
            // Additional setup like mocking dependencies if needed...
        }

        [Test]
        void TestRefreshFromStatus()
        {
            // Arrange
            // Assuming you have some way of setting or knowing the expected status
            String^ expectedStatus = "Open";

            // Act
            ticketDetailsUnderTest->RefreshFromStatus();

            // Assert
            // You'd check the properties or controls affected by RefreshFromStatus() here...
            Assert::That(ticketDetailsUnderTest->StatusDropDown->SelectedItem->ToString(), Is::EqualTo(expectedStatus));
        }

        // Add more tests as necessary...

    };
}
```

7.2 Black-Box Tests

Without peering into the internal workings of the system, black-box tests evaluate the system’s external behaviour. Using Tascomp’s recently formalised excel template, user stories can be executed to confirm that the Support Tracker meets its functional requirements.

TODO: <Insert spreadsheet from testing document.>