Contents

[Task 1: Design Document 1](#_Toc144212953)

[Overview: 1](#_Toc144212954)

[Use Case Diagram: 2](#_Toc144212955)

[UML Class Diagram 3](#_Toc144212956)

[Data Store Specification 3](#_Toc144212957)

[Design Patterns 3](#_Toc144212958)

[Application Architecture 4](#_Toc144212959)

[Appendices 4](#_Toc144212960)

[Appendix A – UML Class Diagram: 4](#_Toc144212961)

# Task 1: Design Document

## Overview:

This is a Command-Line Interface application that allows a software project team to track and manage tickets. Users can display a list of all active tickets filtered by priority, view completed or deleted tickets and search the database for tickets that match their query, they can add create new tickets, update & delete existing tickets & log time to tickets.

## Use Case Diagram:

A diagram of a person's work flow

Description automatically generated

Figure Use Case Diagram

## UML Class Diagram

A diagram of text and words

Description automatically generated with medium confidence

Figure UML Class Diagram - See Appendix A for screenshots of smaller pieces.

## Data Store Specification

## Design Patterns

This application makes use of a few Object-Oriented Design Patterns.

Singleton Pattern:

Ensures that throughout the lifecycle of the application, ‘Singleton’ classes can only be instantiated once. This is useful for ensuring you re-use the same instance of your database connection for multiple users of the application, whilst not showing it’s strengths in a CLI application, this is commonly used in Backend Website-Development to implement the ACID principles.

Generic Interface & Object Relational Mapper pattern:

The application is built in an easily extendable way. The Repositories & Services are not dependent on the specific datatype (Database Table) that they are working with, new tables can easily be integrated because our infrastructure is ***generic***.

Dependency Injection & Compositional patterns:

All logic is broken up into classes with single responsibilities, such as the Repository that handles all communication with the database and maps data to Python Objects or an InputValidator that prompts the user for inputs and ensures no invalid data can be passed into the application. These small building blocks are then *composed* within a class that manages the control flow of the application, our Controller.

## Application Architecture

This application uses a modular, N-tier architecture based on the dependency injection pattern. The application is made up of individual modules, each having a single responsibility and designed to be generic for easy and safe extension.

At the heart is the TicketController, which interfaces with the user and invokes the appropriate services and functions based on user needs. It includes an Output & Prompt Manager, responsible for print, read, and input validation operations, as well as a Formatter module for human-readable data presentation.

The TicketController utilizes a TicketService for data manipulation and database operations, including filtering, sorting, auto-generation of timestamps, and ticket building. To ensure the integrity of data, there's a separate TicketValidationService that handles the generation and validation of ticket fields based on user input.

The TicketService encapsulates a TicketRepository that performs database actions, assured of data integrity due to the upper-layer validators and services.

Both the TicketService and TicketRepository inherit from BaseService and BaseRepository respectively, which are generic base classes accommodating any Data Model. This provides common functionalities like GetById, DeleteById, etc., to all future Services and Repositories. They are designed to work with generics, simplifying data management by accepting Table Name at runtime with no further modification needed.

## Navigation

### Main Context Menu

A screenshot of a computer program

Description automatically generated

Prints a list of all possible options and awaits input.

### View Active Records

A screenshot of a computer program

Description automatically generated

Displays a list of active (not completed or deleted) tickets, ordered by priority descending (lower is greater priority), then remaining time ascending.

### View Record by ID

A screenshot of a computer program

Description automatically generated

Lists a detailed report of the ticket, allows a user to log time.

### Add a new Record

A screen shot of a computer

Description automatically generated

Runs through multiple prompts to generate a new ticket from user input.

### Update by ID

A screenshot of a computer program

Description automatically generated

Allows a user to select a property of an existing ticket to change.

### Delete by ID

A screen shot of a computer

Description automatically generated

Mark a ticket for soft deletion.

### Search for Records

A screenshot of a computer program

Description automatically generated

User enters a string, tickets with any matching fields are displayed.

### Show completed records.

A screen shot of a computer

Description automatically generated

Prints a list of completed records.

### Show deleted records.

A screen shot of a computer

Description automatically generated

Prints a list of deleted records.

### Show all records Raw & Unfiltered

A screenshot of a computer program

Description automatically generated

Prints the entire ticket table from the database.

### Quit

A black screen with white text

Description automatically generated

Allows a user to exit the application.

# Appendices

## Appendix A – UML Class Diagram:

A screenshot of a computer program

Description automatically generated

Bottom Right Section from UML Class Diagram.

A diagram of a server

Description automatically generated

Center-Right section from UML Class Diagram.

A screenshot of a computer

Description automatically generated

Center-Left Section from UML Class Diagram.

A screenshot of a computer

Description automatically generated

Bottom-Left Section from UML Class Diagram.

A screenshot of a computer

Description automatically generated

Left Section from UML Class Diagram.

A diagram of a program

Description automatically generated

Top-Left & Top-Middle section from UML Class Diagram.