# 2207-BSE

# Implementation – SRS Document

# CS106.1

*Alexander Craig, Oliver Anders Grönkrans, Alexander Legner, Liam Konise*

## Document Outline

### Table of Contents

## Table of Figures

## Table of Tables

Intentionally Blank

## SRS Document

### Introduction

#### Purpose

The purpose of this document is to present a detailed explanation of the ticketing system. The document will clarify the system, what the system does, how the system is supposed to work, the interfaces of the system, the constraints for the system to work as intended and how the system reacts to unintended incitation, and the purpose plus features of the system. This document is intended for stakeholders, developers of the system, and potential clients of the proposed system. The document will be part of the proposed plan to the stakeholders for approval.

#### Scope

This software system will be a helpdesk ticketing program, used to submit and handle tickets. Everything will be hosted locally as a proof of concept, and thus will not be accessible through the web at this stage. The program shall be able to handle login and signup requests, ticket submissions and edits, as well as basic error handling. The inputs required are to be kept at a minimum to increase the ease of use, to prepare for a real-world application. Users should be able to send in new tickets, see their current ones and add comments to their existing tickets. Admins should be able to update the ticket status, category, and priority, as well as change user details and delete accounts.

#### Overview of document

The first section of the document specifies the expected technical deployment of features, using a mix of technical terminology and more understandable English, while the second part of the document visualizes different use cases for the types of users, with brief explanations for each use case.

Implementation

## Overview

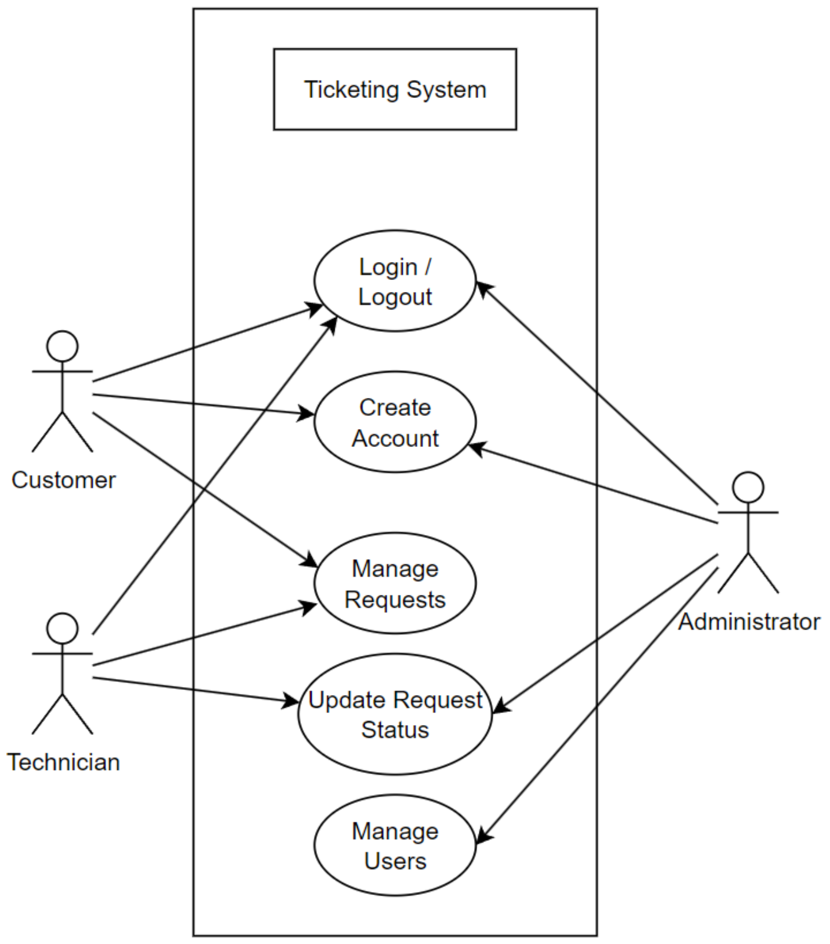

*Figure 1: System Environment Overview*

## System Features and Requirements:

### User Requirements

We have three types of end users: Administrators, Technicians and Customers.

Administrator User Requirements:

- Login/Logout
- Create Accounts
- Manage Accounts/Users
- Edit Tickets

Technician User Requirements:

- Login/Logout
- Update Tickets
- Manage Tickets

Customer User Requirements:

- Login/Logout
- Create an account
- Manage Tickets

### User Functional Requirements:

All users must be able to:

- ~~Create an account~~
- **REVISION** ACCOUNT CREATION MOVED TO ADMIN PRIVILEGES
- Login to their account
- Create a ticket
- Submit a ticket
- Overview tickets
- View selected assigned ticket
- Add comment to assigned ticket

Technicians must also be able to:

- Set ticket status
- Set ticket priority

Admins must also be able to:

- Change account details
- Delete account
- Change account IDs
- Assign ticket to a user
- ~~Disassociate ticket from a user~~
- ~~Delete ticket~~
- **REVISION** REMOVED AS IT IS UNNECESSARY
- **REVISION** ADDED ACCOUNT CREATION

## Functional requirements

### *Account creation*

*Table 1: Account Creation*

| | |
|---|---|
| Description | When the user has entered the application and chosen to create an account, they are assigned an ID, shown their user ID, and automatically logged in to the generated account. |
| Trigger | The user has clicked generate account on the landing page. |
| Processing | The system generates a pseudorandom string of numbers and characters, checks it against the user database to see if the ID is already in use. If it is unoccupied, it adds a line in the user database with the generated ID, and then delegates a request to the login system with the generated ID and logs the user in, as well as displays a message for the user showing their new account ID. |
| Output | User is directed to landing page for logged in users and shown their ID. |
| Error handling | ~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

## *Login*

*Table 2: Login*

| | |
|---|---|
| Description | When the user has entered the application and chosen to login to an existing account, and filled in their account ID, the system checks that the account exists and if it does it logs them in. |
| Trigger | User has entered login page, input their ID in the input field, and pressed login. |
| Processing | The value in the input field is compared against each line of the database until a match is found. If none is found in the database it outputs an error message to the user, letting them know that no account was found. If an account is found, it will load the landing page for logged in users. |
| Output | User is directed to landing page for logged in users and shown a success message if account was found. If not, they are shown an error message. |
| Error handling | ~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>If no user has been found with the given ID, output an error message on screen for the user.<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

## *Create ticket*

*Table 3: Create Ticket*

| | |
|---|---|
| Description | When the user has selected to create a ticket, overlay menu for ticket creation. |
| Trigger | User has chosen to create a new ticket. |
| Processing | An overlay menu for ticket creation is displayed, and a gradient on the background is applied. |
| Output | The user is shown a menu where they can enter ticket details. |
| Error handling | N/A |

### Submit ticket

*Table 4: Submit Ticket*

| | |
|---|---|
| Description | The user has filled out the ticket creation menu and chosen to submit their ticket. The system checks if all fields are filled out and gives appropriate success or error messages. |
| Trigger | The user has pressed submit ticket in the ticket creation menu. |
| Processing | Before handing the ticket details off to the ticket database, the system checks whether all obligatory fields contain information, and if not, outputs an error message, cancelling the handoff. If all fields are filled out, it establishes a link to the database and writes the ticket details to a new object in the database. |
| Output | The user is shown a message if the ticket was not successfully filled out, and a success message is displayed if the ticket was successfully written to the database or not. |
| Error handling | Check if all obligatory fields are filled out.<br><br>~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### Overview tickets

*Table 5: Overview Tickets*

| | |
|---|---|
| Description | The user has chosen to overview all tickets they are involved in (either posted or handling) and is then shown a list of previews. |
| Trigger | The user has pressed overview tickets. |
| Processing | The program establishes a link to the user database, indexes the current users assigned tickets, and then uses those ticket IDs to establish a link to the ticket database, index the user's tickets and display ticket ID and subject line as a preview. |
| Output | The user is shown a menu where they can see previews of all tickets that they are involved in. |
| Error handling | ~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### *Viewing a ticket*

*Table 6: Viewing A Ticket*

| | |
|---|---|
| Description | The user has chosen a ticket they would like to view in full detail, and the system then opens an overlay showing all details. |
| Trigger | User has pressed a specific ticket. |
| Processing | The program establishes a link to the ticket database and indexes the desired ticket. It then feeds each respective field into the menu overlay fields. |
| Output | The program shows a menu with all the respective details for the desired ticket. |
| Error handling | ~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### Add comment to ticket

*Table 7: Add Comment to Ticket*

| | |
|---|---|
| Description | The user has filled out the comment field and pressed add, leading to the program pushing the change to the ticket database so others involved can see it as well. |
| Trigger | User has pressed send on comment field. |
| Processing | The program establishes a link to the ticket database and indexes the desired ticket. It then pushes a new comment to the corresponding section of the ticket instance. |
| Output | ~~The program displays a message of whether the comment was successfully posted or not and shows the posted comment in the corresponding field.~~ **REVISION** The comment is simply displayed in the comment section if it is successfully pushed |
| Error handling | ~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### *Set ticket status*

*Table 8: Set ticket status*

| | |
|---|---|
| Description | The user has clicked the dropdown menu for ticket status and chosen a new status. The system then updates the records to reflect the change. |
| Trigger | User has chosen an option in the ticket status dropdown menu. |
| Processing | The program will compare the chosen alternative against the current selection to see if a change has been made. It then establishes a link to the ticket database and pushes the change of status to the indexed ticket. |
| Output | The ticket shows the new status. |
| Error handling | Check if any changes has been made.<br><br>~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### *Set ticket priority*

*Table 9: Set ticket priority*

| | |
|---|---|
| Description | The user has clicked the dropdown menu for ticket priority and chosen a new priority. The system then updates the records to reflect the change. |
| Trigger | User has chosen an option in the ticket priority dropdown menu. |
| Processing | The program will compare the chosen alternative against the current selection to see if a change has been made. It then establishes a link to the ticket database and pushes the change of priority to the indexed ticket. |
| Output | The ticket shows the new priority. |
| Error handling | Check if any changes has been made.<br><br>~~Check whether a link to the database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### *Change account details*

*Table 10: Change account details*

| | |
|---|---|
| Description | The user (admin) has chosen to update an account's details. They then get shown a menu overlay with potential changes they can make, such as change ID, assign ticket, and delete account. |
| Trigger | The user (admin) has chosen to update an account's details and input an account ID. |
| Processing | An overlay menu for account management is displayed, and a gradient on the background is applied. |
| Output | The user is shown a menu where they can change account details. |
| Error handling | ~~Check if a link to the user database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### *Delete account*

*Table 11: Delete account*

| | |
|---|---|
| Description | The user (admin) has chosen to delete an account from the account management menu. This account is subsequently deleted from the user database, and the ID removed from any tickets. |
| Trigger | The user (admin) has pressed delete account in the account management menu. |
| Processing | The program establishes a link to the user database and indexes the desired user to be deleted. It then establishes a link to the ticket database and indexes all tickets the user is linked to and removes the account ID from each respective ticket. It then removes the user from the user database. |
| Output | The user is removed from the user database and all related tickets. |
| Error handling | ~~Check if a link to the user database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~Check if a link to the ticket database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### ~~*Change account ID*~~

~~*Table 12: Change account ID*~~

| | |
|---|---|
| ~~Description~~ | ~~The user (admin) has chosen to change an account's ID from the account management menu. This account is subsequently updated in the user database, and the ID updated in any tickets.~~ |
| ~~Trigger~~ | ~~The user (admin) has entered a new account ID in the account management menu.~~ |
| ~~Processing~~ | ~~The program establishes a link to the user database and indexes the desired user to be updated. It then establishes a link to the ticket database and indexes all tickets the user is linked to and updates the account ID in each respective ticket. It then updates the user in the user database.~~ |
| ~~Output~~ | ~~The account ID is updated in the user database and all related tickets.~~ |
| ~~Error handling~~ | ~~Check if a link to the user database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~Check if a link to the ticket database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~**REVISION**~~<br>~~Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success.~~ |

**REVISION** REMOVED AS IT WAS DEEMED TO DANGEROUS FOR SYSTEM FUNCTIONALITY

### *Assign ticket to user*

*Table 13: Assign ticket to user*

| | |
|---|---|
| Description | ~~The user (admin) has chosen to assign a ticket to an account from the account management menu. This account subsequently gets the ticket added in the user database, and the account ID is added in affected ticket.~~ **REVISION** THE USER IS ASSIGNED AT TICKET CREATION INSTEAD |
| Trigger | The admin has entered a ticket ID in the corresponding field and pressed assign |
| Processing | The program establishes a link to the user database and indexes the desired user to be updated. It then establishes a link to the ticket database and indexes the ticket the user is to be linked to and adds the account ID in the ticket. It then adds the ticket to the user in the user database. |
| Output | The account ID is added to the ticker in the ticket database and the ticket ID is added to the account in the user database. |
| Error handling | ~~Check if a link to the user database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~Check if a link to the ticket database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>Check if a value for the ticket is entered.<br><br>Check if the value for ticket is valid.<br><br>Check if the user is not already assigned to the ticket.<br><br>**REVISION**<br>Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success. |

### ~~*Disassociate ticket from user*~~

~~*Table 14: Disassociate ticket from user*~~

| | |
|---|---|
| ~~Description~~ | ~~The user (admin) has chosen to disassociate a ticket from an account from the account management menu. This account subsequently gets the ticket removed in the user database, and the account ID is removed in affected ticket.~~ |
| ~~Trigger~~ | ~~The admin has clicked remove on the preview list of the user's assigned tickets.~~ |
| ~~Processing~~ | ~~The program establishes a link to the user database and indexes the desired user to be updated. It then establishes a link to the ticket database and indexes the ticket the user is to be removed from and removes the account ID in the ticket. It then removes the ticket ID from the user in the user database.~~ |
| ~~Output~~ | ~~The account ID is removed from the ticker in the ticket database and the ticket ID is removed from the account in the user database.~~ |
| ~~Error handling~~ | ~~Check if a link to the user database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~Check if a link to the ticket database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~Check if a value for the ticket is entered.~~<br><br>~~Check if the user is assigned to the ticket before trying to remove.~~<br><br>~~**REVISION**~~<br>~~Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success.~~ |

**REVISION** REMOVED AS IT WAS DEEMED UNNECESSARY

### ~~*Delete ticket*~~

~~*Table 15: Delete Ticket*~~

| | |
|---|---|
| ~~Description~~ | ~~The user (admin) has chosen to delete a ticket from the ticket management menu. This ticket is subsequently removed from any associated users and deleted from the ticket database.~~ |
| ~~Trigger~~ | ~~The admin has clicked delete in the ticket view.~~ |
| ~~Processing~~ | ~~The program establishes a link to the ticket database and to the user database, indexes all users in the user database that are associated with the ticket, removes the ticket reference from them, and then deletes the ticket from the ticket database.~~ |
| ~~Output~~ | ~~The ticket is deleted and all references to it removed.~~ |
| ~~Error handling~~ | ~~Check if a link to the user database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~Check if a link to the ticket database has been established, if not, try again, and if it fails after 3 requests, output an error message on screen for the user.~~<br><br>~~**REVISION**~~<br>~~Changed to simply display an error message as the loop functionality would simply add unnecessary complexity, where most of the errors caused would be that the database is inaccessible and multiple attempts back to back would unlikely result in success.~~ |

**REVISION** REMOVED AS IT WAS DEEMED UNNECESSARY AND COULD BREAK FINANCIAL ARCHIVING LAWS DEPENDING ON WHICH COUNTRIES IT WOULD BE DEPLOYED IN

## Non-functional requirements

The database holding all data related to the program will on a server with high-speed internet functionality. The speed of a user's connection will be dependent on the hardware and internet connection of the user's system rather than the systems design.

The administrator will use a machine on the database server and will contain access to the database. Access is already installed on the administrator system and is a Windows operating system. The ticketing system will run on technicians and users' computers and is using a Windows operating system.

Users must be presented with a clear, uncluttered UI so that they are not overwhelmed with information.

Text must use a readable font, with contrasting colours to the background so that it is easy to view, and preferably increase readability for users with impaired vision.

Given that users will not be aware of the background workings of the software, they must be presented with clear, concise information when they encounter errors. They must also get clear visual and potentially audio cues to infer if their requested action was successful or if an error occurred.

## Detailed Non-Functional Requirements

Logical structure of the data to be stored in the internal ticketing system database is shown below.

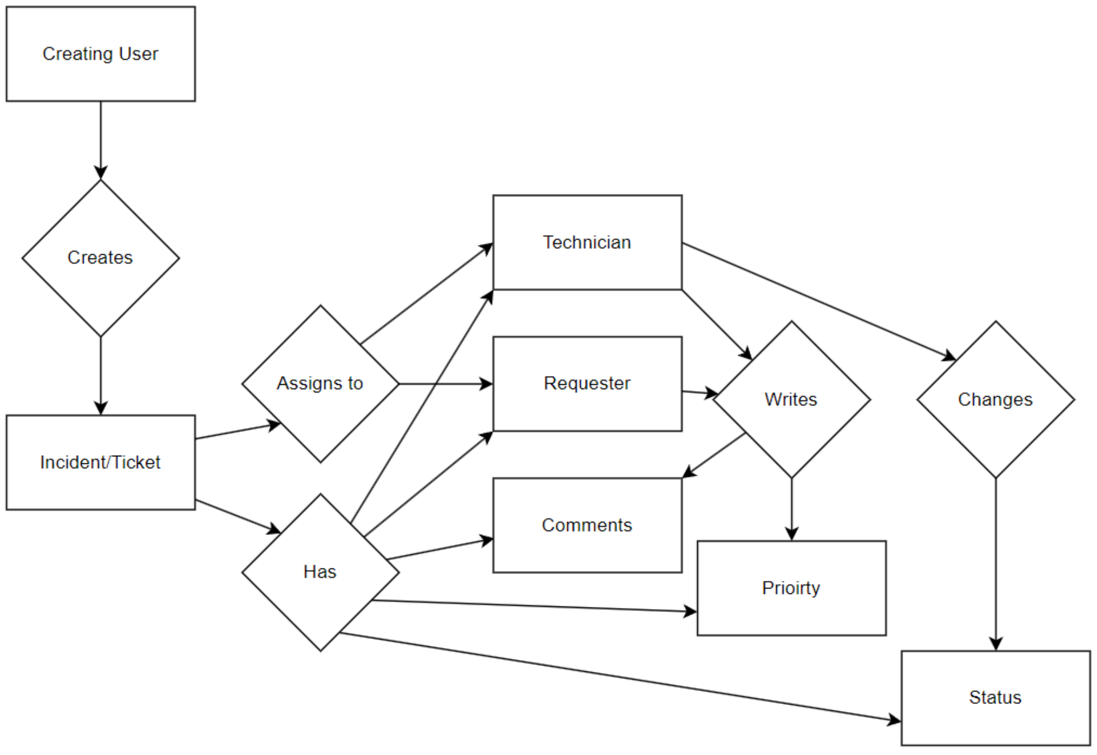

*Figure 2: Logical Structure of Data Stored on Internal Database*

The logical structure of all the data which will be stored on the internal database for the ticketing system is as follow:

### Login Details Data Entity

*Table 16: Login Details Data Entity*

| Data Item | Type | Description | Comment |
|---|---|---|---|
| Unique User ID | Text | The user id unique to a user used to login | |
| Full Name | Text | The user's full name | |
| Password | Encrypted Text | The user's password | |
| User Type | Integer | The type of user | |

### Incident/Ticket Data Entity

*Table 17: Incident/Ticket Data Entity*

| Data Item | Type | Description | Comment |
|---|---|---|---|
| Creating User ID | Text | The user id unique to the user that created the ticket | |
| Requester User ID | Text | The user id unique to the user that requested the ticket | This is the user the ticket is for |
| ~~Technician User ID~~ | ~~Text~~ | ~~The user id unique to a technician the ticket has been assigned to.~~ | **REVISION** REMOVED AS ALL TECHNICIANS CAN ACCESS ANY GIVEN TICKET INSTEAD |
| Comments | Text | A comment added to the incident with user id and time | May be several |
| Priority | Integer | The priority of the ticket | |
| Status | Integer | The status of the ticket | |

### All Incidents Data Entity

*Table 18: All Incidents Data Entity*

| Data Item | Type | Description | Comment |
|---|---|---|---|
| Assigned Incidents | List of pointers to tickets | An ever-changing list of currently active tickets with an assigned technician. | |
| Closed Incidents | List of pointers to tickets | An ever-changing list of currently closed or completed tickets | |
| Unassigned Incidents | List of pointers to tickets | An ever-changing list of tickets that are yet to be assigned to a technician | |

Security – the server which the ticketing system is on will have its own security measures in place to prevent unauthorized write/delete access. Only administrators logged into the internal system will have unrestricted access to the data except for user's passwords which are stored as encrypted data.

## Prioritisation / negotiation

The core functionality that must be prioritized includes, but are not limited to -

1. Creating tickets
2. Viewing tickets
3. Login system
4. Signup system

Secondary features include, but are not limited to -

1. Functional UI
2. Updating ticket status
3. Updating ticket priority
4. Adding comment to ticket

Tertiary features include, but are not limited to -

1. User-tested UI
2. Deleting accounts
3. ~~Changing account IDs~~
   **REVISION** Superceded by name changes

As we cannot have a helpdesk ticket application without ticket creation and viewing, these must be prioritized. Dynamic ticket handling is important, but not strictly necessary for the software. The UI must be able to do at the very least the basic actions of viewing tickets and creating tickets, but preferably in a manner that is tested and planned, making for a readable experience. However, we must focus on core functionality over aesthetics.

<u>There will be two times the project features and requirements can be negotiated.</u>

- Once on review and approval of the SRS Requirements by the client, where the client can confirm all the requirements, they were expecting are present. Any added or new requirements that the client wants to add on top of this will require more time to complete the project and therefore are subject to rejected by the developers if an agreement to any new features and new resource allocation cannot be agreed upon at the time. This negotiation period is up to the client to decide if they want to add more time to the deadline in this case as we the developers are on a strict deadline to complete the project as outlined by the client.

- Lastly on review of the prototype the client will get test the prototype to ensure all features specified in the SRS are present. All new feature requests by the client are subject to rejection by the developers if an agreement to any new features and new resource allocation cannot be agreed upon at the time. This is because the requirements outlined in the SRS are what are agreed upon by both the client and the developer after negotiation the first time. This negotiation period is up to the client to decide if they want to add more time to the deadline in this case as we the developers are on a strict deadline to complete the project as outlined by the client. The client is welcome to request changes if they can allocate an agreed amount of time with the developers. There will be no changes after this stage of the project as changing the plan while developing the product after the prototype has be approved by the client will take more added time to the project.

### System features

The software is run on a local machine, running Windows 10, with the database accessible from the same machine. In the real-world deployment it runs on a server running Windows, queried over the internet.

### Interface requirements

The external links include the user database, to load the correct user details, and the ticket database, to load the corresponding tickets linked to the current user. For the user database, the software looks at account ID and assigned tickets. Regarding the ticket database, the software looks at assigned users/accounts, subject, description, priority, status, and comments.

**REVISION**

### System architecture

The software is built on a WPF/.NET frame, in which instances of objects such as tickets and users are stored in local variables, but these load variables from a SQL server on the local machine by querying the corresponding keywords.

A user instance is created by querying the id of the user (and in cases such as login comparing password as well), loading the fields into the corresponding variable, such as FirstName goes into firstName in the application.

The main deviation from this structure is comments, which are stored with a limit character between each individual comment of a ticket, and an example of a stored comment thread would be

```
♦User¦One¦2023/05/20-19:10¦Heres My Comment string♦User¦Two¦2023/05/20-
20:19¦This is another comment
```

Passwords are hashed before being sent to the SLQ database, and thus all login attempts hashes the users input password, against the stored password hashes. For ticket indexing, checking if a ticket belongs to a user, it simply compares caller and creator ID's against the logged in user's ID.

The databases are divided into two SQL servers, Tickets.mdf (which contains the table AllTickets), and Users.mdf (which contains the table Users). The user database is queried when handling logins, account creation, updating names, emails, passwords, etc, while the ticket database is queried when adding or editing tickets.

## Use Case Diagrams

### Overview

To start we will create an overview of all the use case diagrams put together to show the big picture of how the product should be used. Then we will go into detail for each use case.

**Diagram:**

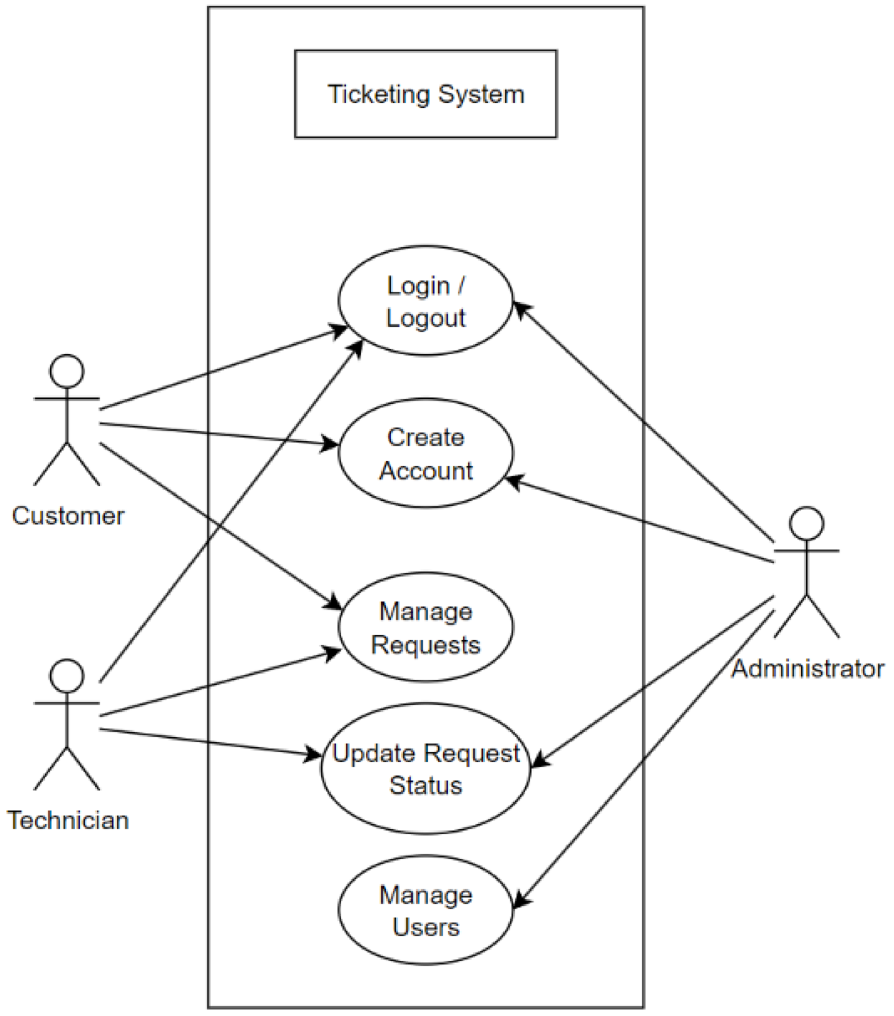

*Figure 3: Use Case Diagram Overview*

## Administrator Use Cases

**Diagram:**

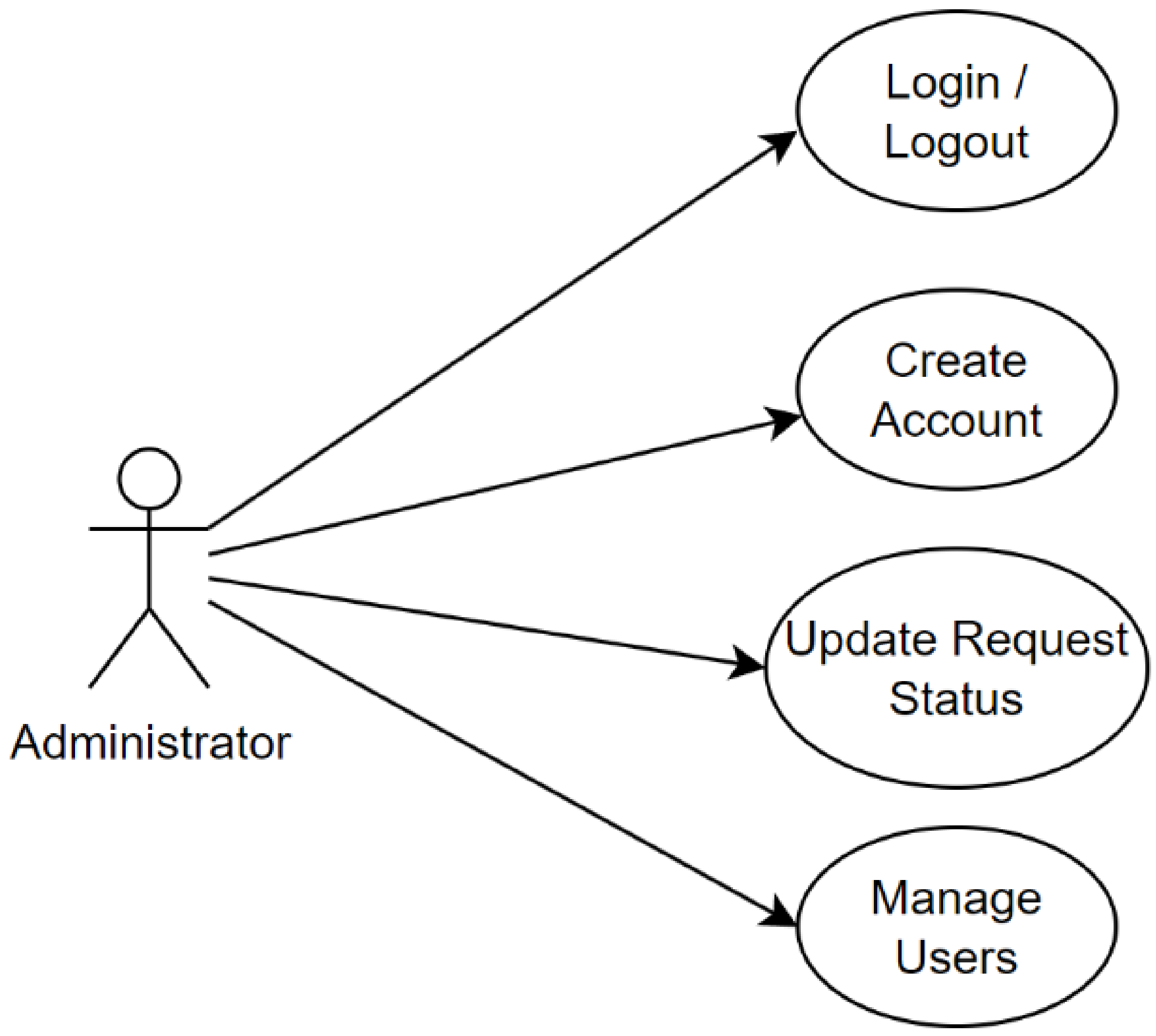

*Figure 4: Use Case Diagram Administrator*

The administrator can login and log out, create an account, update tickets, and manage users. Their overall role is system oversight, making sure account details are correct, and correcting any incorrect information.

### *Use case: Update Request Status*

**Diagram:**

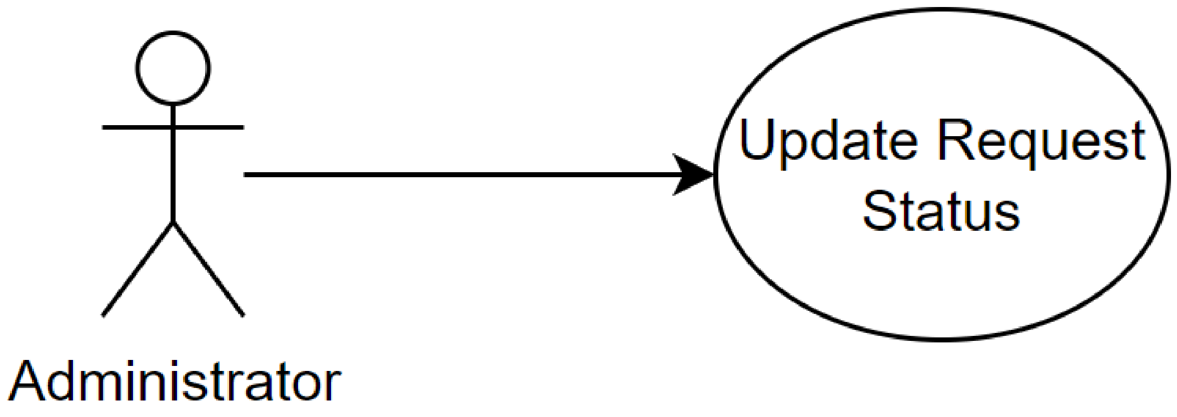

*Figure 5: Use Case Administrator Update Status Request*

**Brief Description**

The Administrator can update existing request statuses. Inputting and updating data within each individual request for clients and employees to see. Making sure the details within a request are correct and ensuring all relevant information is provided.

**Initial Step-By-Step Description**

Before this use case can be initiated, the Administrator has already logged in on an administrator computer to the program.

1. The admin has searched for a ticket ID.
2. The system displays the ticket details in the ticket view menu.
3. The admin selected a field.
4. The admin input a new value or chose a different option in the field.
5. The admin presses confirm changes.
6. The system pushes the changes to the ticket database.

**Xref:** Table 5 – Overview Tickets, Table 6 – Viewing a Ticket, Table 7 – Add a comment to Ticket, Table 8 – Set Ticket Status, Table 9 – Set Ticket Priority.

### *Use case: Create Account*

**Diagram:**

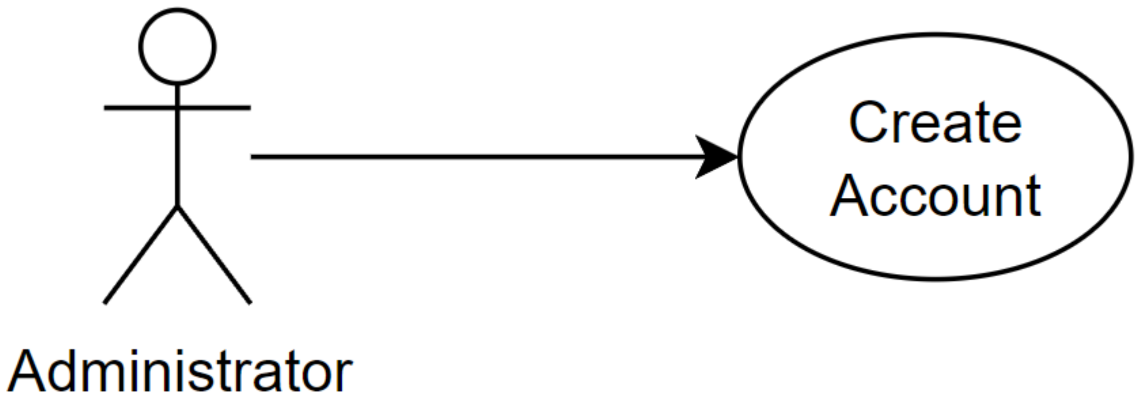

*Figure 6: Use Case Administrator Create Account*

**Brief Description**

The Administrator can create an account for other Customers or Technicians.

**Initial Step-By-Step Description**

Before this use case can be initiated, the Administrator has already logged in on an administrator computer to the program.

1. The administrator logs into the system.
2. The administrator selects Create Account.
3. The Administrator inputs a username and password for the account.
4. The administrator assigns a role to the account e.g., Technician, Client, Administrator

**Xref:** Table 1 – Account Creation

### *Use case: Manage Users*

**Diagram:**

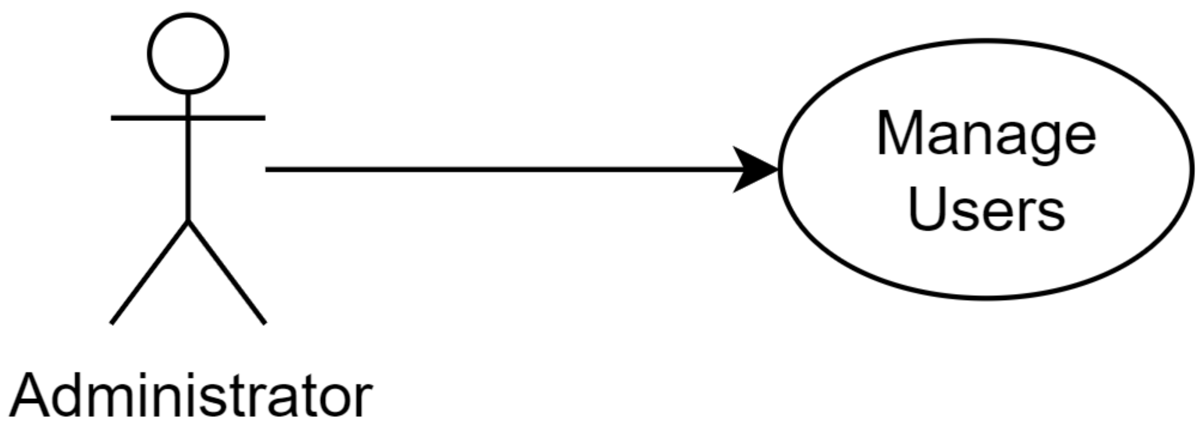

*Figure 7: Use Case Administrator Manage Users*

**Brief Description**

The Administrator can manage users, i.e., update account IDs, delete accounts, assign a user to a ticket, dissociate users from a ticket, and deleting tickets.

**Initial Step-By-Step Description**

Before this use case can be initiated, the Administrator has already logged in on an administrator computer to the program.

1. The admin has pressed user management.
2. The system displays the user management screen, displaying all available users.
3. The admin has selected a user from the user overview.
4. The admin has pressed delete account, change account ID, assign ticket, or dissociate from ticket.
5. The admin has pressed confirm changes.
6. The system pushes the changes to the ticket database.

**Xref:** Table 1 – Account Creation, Table 10 – Change Account Details, Table 11 – Delete Account, Table 12 – Change Account ID.

*Use case: Login/Logout*

**Diagram:**

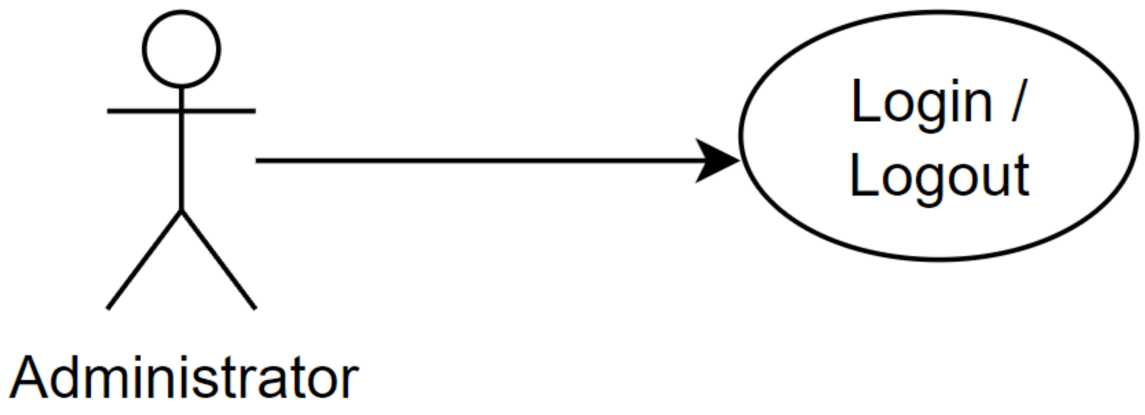

*Figure 8: Use Case Administrator Login/Logout*

**Brief Description**

The Administrator is capable of logging in and logging out.

**Initial Step-By-Step Description**

The Administrator must use an administrator computer connected to the server to login with administrator privileges.

1. The Administrator opens the system.
2. The Administrator selects Log in/out.
3. The Administrator inputs their assigned username and password.
4. The Administrator is presented with Update request status, manage users, create account, and log out.

**Xref:** Table 2 – Login

## Technician Use Cases

**Diagram:**

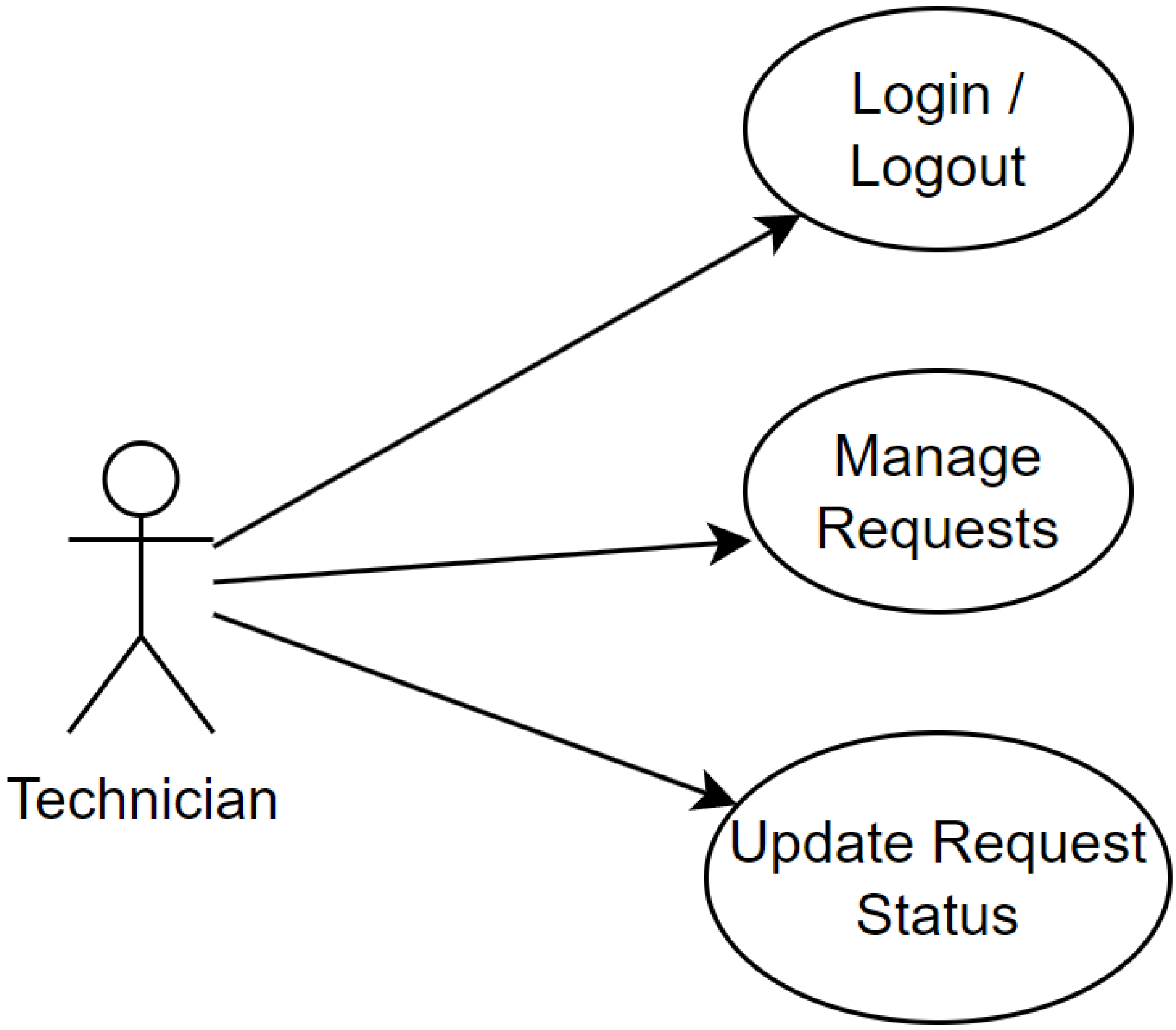

*Figure 9: Use Case Diagram Technician*

The Technician is capable of logging in/out, managing requests, and updating request status. Their overall role is to complete requests and update them with the valid information.

### *Use case: Manage Requests*

**Diagram:**

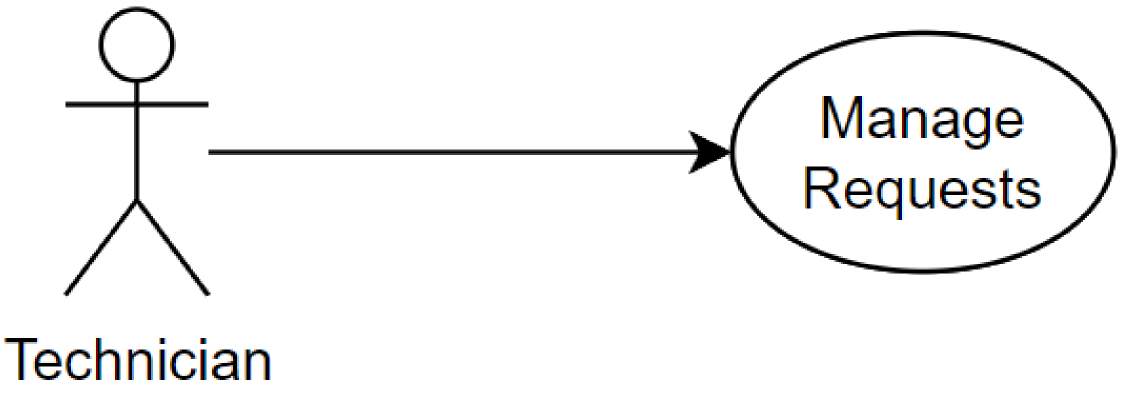

*Figure 10: Use Case Technician Manage Requests*

**Brief Description**

The Technician after having logged in can view all current requests with details retaining to each individual request, approve/deny requests, assign request to a team member if they have any.

**Initial Step-By-Step Description**

Before this use case the Technician has already logged in.

*Open a request

1. The Technician views all requests allocated to them in the Program.
2. The Technician selects the request they want to manage.
3. The Program retrieves the request from the Server.
4. The Program displays the request to the Technician.

*Resolve request status

Before this step the Technician has opened a request.

1. The Technician clicks on the resolve button.
2. The Program provides a list of reasons as to why the request is resolved.
3. The Technician clicks on the respective reason.
4. The Program sends data to the Server updating the status of the request.

~~*Delete a request~~

~~Before this step the Technician has opened a request.~~

~~1. The Technician clicks on the delete button.~~
~~2. The Program provides a list of reasons as to why the request is going to be deleted.~~
~~3. The Technician clicks on the respective reason.~~
~~4. The Program sends data to the Server to delete the request.~~
~~5. The Program informs the Technician that the request is deleted.~~

**REVISION** DELETED TO FULFILL INTERNATIONAL FINANCIAL ARCHIVING LAWS

*Set priority of request

Before this step the Technician has opened a request.

1. The Technician clicks on the priority button.
2. The Program provides a list of priorities to the Technician.
3. The Technician clicks on the respective priority.
4. The Program sends data to the Server to update the request.

*Add comment to the request

Before this step the Technician has opened a request.

1. The Technician types into the comment box of the request and enters a comment.
2. The Program sends the comment to the Server to update the respective request.

**Xref:** Table 3 – Create Ticket, Table 4 – Submit Ticket, Table 5 – Overview Tickets, Table 6 – Viewing a Ticket, Table 7 – Add a comment to Ticket, Table 8 – Set Ticket Status, Table 9 – Set Ticket Priority. Table 14 – Disassociate Ticket from User, Table 15 – Delete Ticket.

### *Use case: Update Request Status*

**Diagram:**

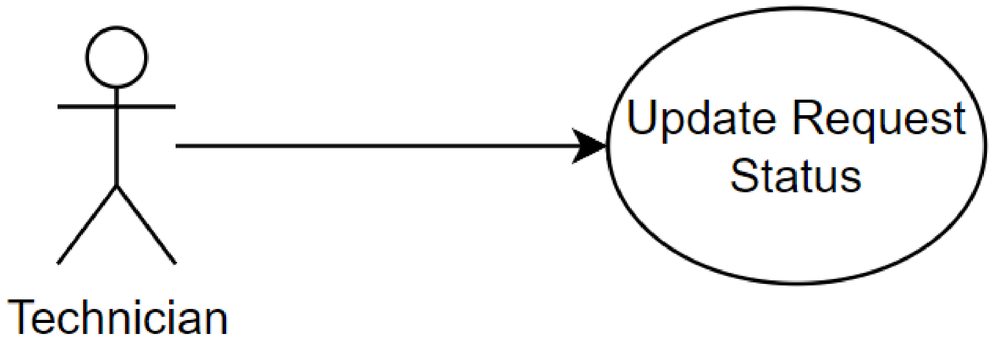

*Figure 11: Use Case Technician Update Request Status*

**Brief Description**

The technician has selected a ticket and entered a new value in any of the fields and pressed the confirm changes button.

**Initial Step-By-Step Description**

1. The technician has selected a ticket they are assigned to.
2. The system displays a ticket view menu, displaying the ticket details.
3. The technician has entered a new value or chosen a different option in any field.
4. The technician has pressed confirm changes.
5. The system pushes the changes to the ticket database.

**Xref:** Table 6 – Viewing a Ticket, Table 8 – Set Ticket Status, Table 9 – Set Ticket Priority.

### *Use case: Login/Logout*

**Diagram:**

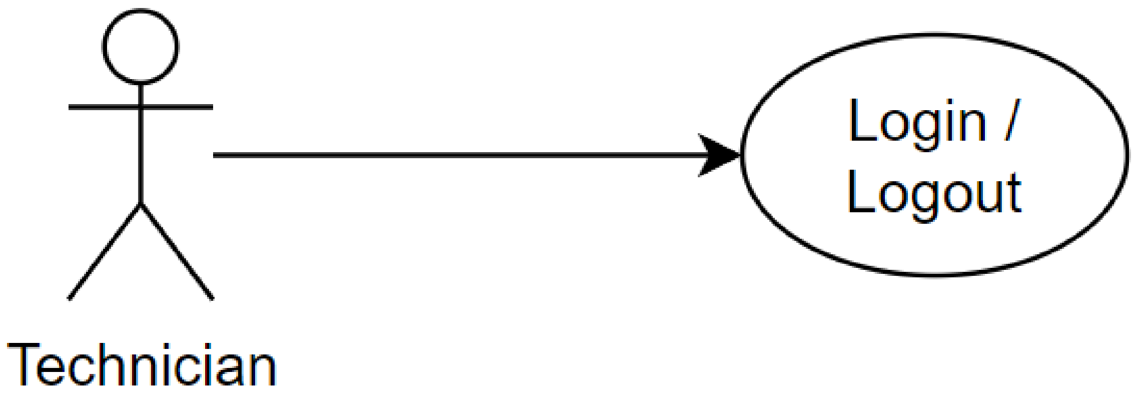

*Figure 12: Use Case Technician Login/Logout*

**Brief Description**

The technician has selected to either login with their account ID or logout from the main menu.

**Initial Step-By-Step Description**

*Login

1. The Technician launches the Program which is initially not logged in.
2. The Program displays a login page for the Technician.
3. The Customer types in the login details and logs in.
4. The Program encrypts the login data and sends it to the server.
5. The Server compares the encrypted data from the Program with the list of stored encrypted login details and sends an appropriate response back to the Program.
6. The Program will check the response from the Server and inform the user if the login was successful or unsuccessful.

**Xref:** Table 2 - Login

*Logout

1. The Technician selects their profile from within the Program and clicks the logout button.
2. The Program gives the user a warning and prompts the Technician again if they are sure they want to log out.
3. The Technician clicks the confirm button.
4. The Program logouts and removes any stored memory, encrypted login details and any other data from the Program.

## Customer Use Cases

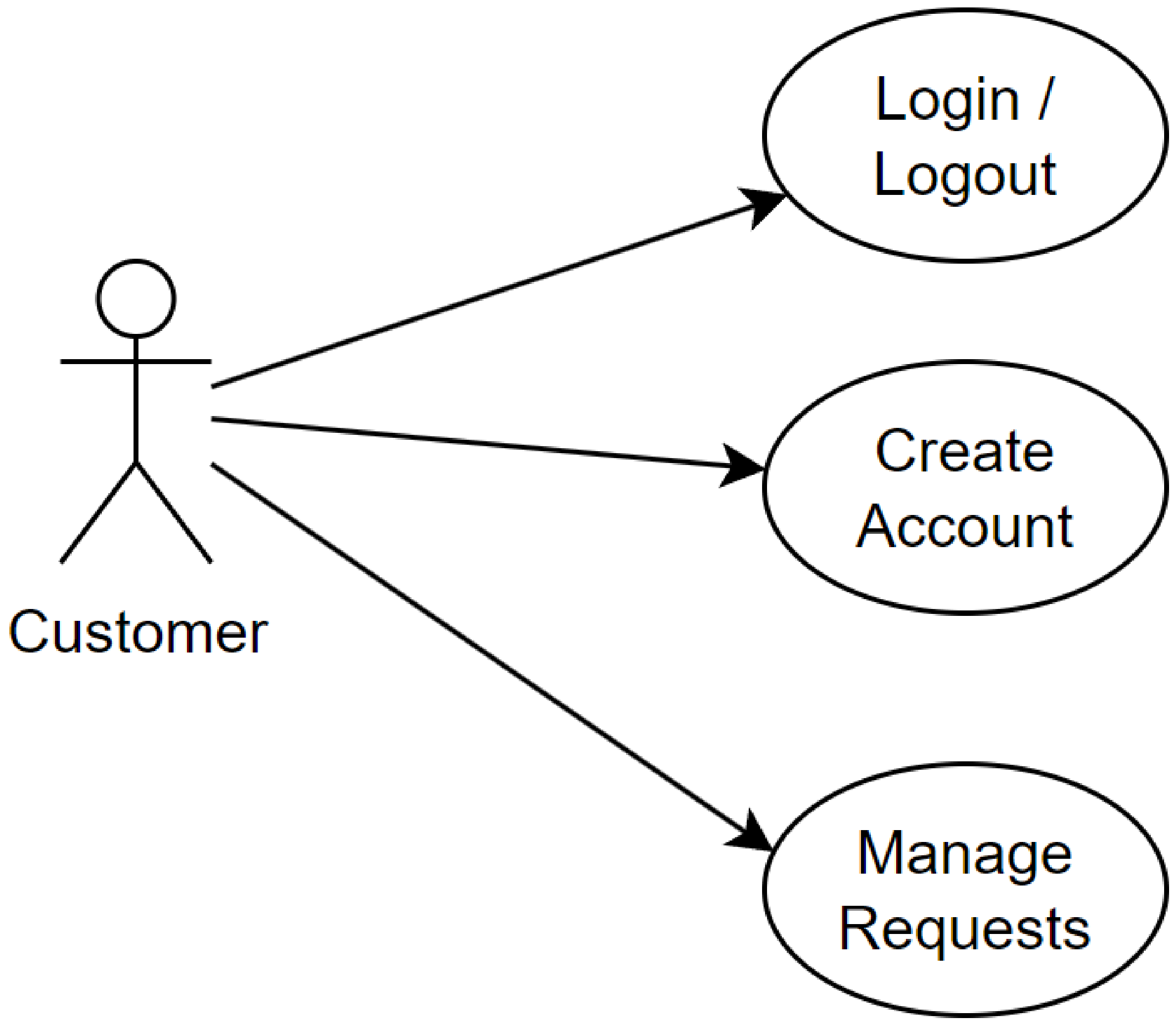

*Figure 13: Use Case Diagram Customer*

The customer after accessing the system can log in/out, create account, manage requests. Their overall role is to create requests pertaining to their encountered problems.

*Use case: Login/Logout*

**Diagram:**

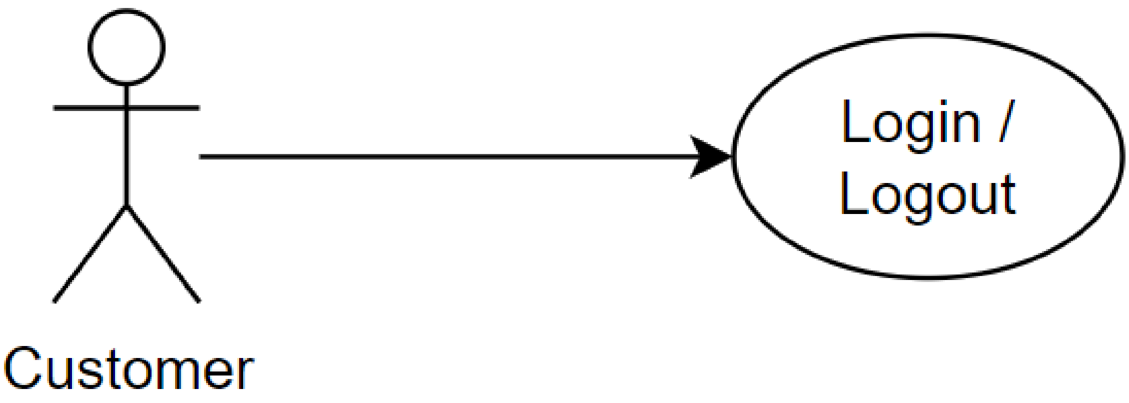

*Figure 14: Use Case Customer Login/Logout*

**Brief Description**

The Customer logs in or out of the program using their account.

**Initial Step-By-Step Description**

*Login

7. The Customer launches the program which is initially not logged in.
8. The Program displays a login page for the Customer.
9. The Customer types in the login details and logs in.
10. The Program encrypts the login data and sends it to the server.
11. The Server compares the encrypted data from the program with the list of stored encrypted login details and sends an appropriate response back to the Program.
12. The program will check the response from the Server and inform the user if the login was successful or unsuccessful.

**Xref:** Table 2 - Login

*Logout

5. The Customer selects their profile from within the Program and clicks the logout button.
6. The Program gives the user a warning and prompts the Customer again if they are sure they want to log out.
7. The Customer clicks the confirm button.
8. The Program logouts and removes any stored memory, encrypted login details and any other data from the Program.

*Use case: Create Account*

**Diagram:**

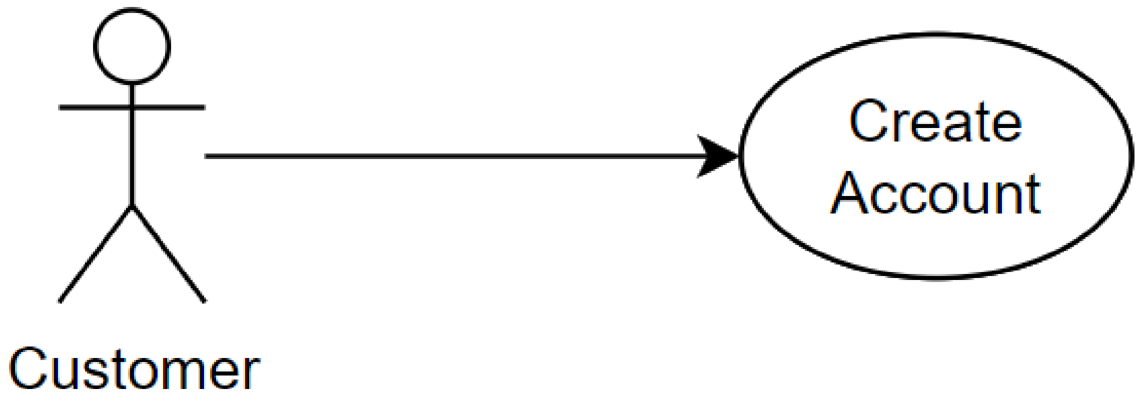

*Figure 15: Use Case Customer Create Account*

~~**Brief Description**~~

~~The Customer creates and account to get access to the Program.~~

~~**Initial Step-By-Step Description**~~

1. ~~The Customer accesses the Program.~~
2. ~~The Customer selects Create Account.~~
3. ~~The Customer inputs all required data including password and username.~~
4. ~~The Program sends the encrypted data to the Server to create an account.~~
5. ~~The Server sends a success message back to the Program.~~
6. ~~The Program displays respective outcome of the message from the Server.~~
7. ~~If the server message was a success, then the Program will login with the newly created account.~~

~~**Xref:** Table 1 – Account Creation, Table 2 - Login~~

**REVISION** Account creation locked to admin to avoid malicious bulk registrations.

### *Use case: Manage Requests*

**Diagram:**

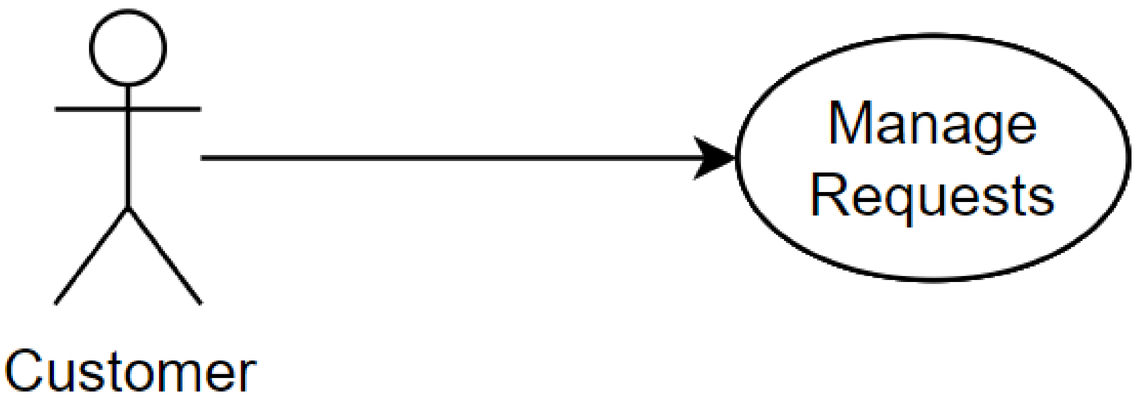

*Figure 16: Use Case Customer Manage Requests*

**Brief Description**

The customer has chosen to manage their requests, overviewing their tickets.

**Initial Step-By-Step Description**

1. The Customer views their requests.
2. The Program displays the ticket overview menu, displaying previews of all their tickets.
3. The Customer can create a new ticket or open an existing ticket.

*Create
Before this step can proceed, the Customer has opened the overview of all their requests.

1. The Customer clicks on the create request button.
2. The Customer enters all the details of the request that are required.
3. The Customer clicks on the submits button.
4. The Program sends the request data to the Server.
5. The Program opens the newly created request.

*Open
Before this step can proceed, the Customer has opened the overview of all their requests.

1. The customer clicks on one of their requests that they want to view.
2. The Program retrieves the request data from the Server.
3. The Program displays the request.

*Add Comment
Before this step can proceed, the Customer has opened a request of theirs.

1. The Customer clicks on the comment box and enters a comment.
2. The Program sends the comment to the server to update the ticket.

**Xref:** Table 3 – Create Ticket, Table 4 – Submit Ticket, Table 5 – Overview Tickets, Table 6 – Viewing a Ticket, Table 7 – Add a comment to Ticket.

## Use case: Finn’s Scenario

** Finn’s Scenario can be found in the Research Document

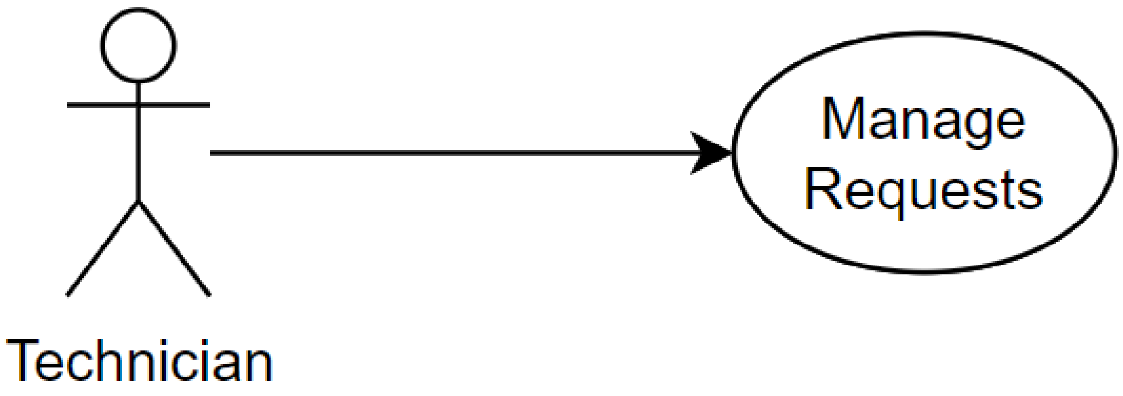

*Figure 17: Finns Scenario*

**Brief Description**

The technician (in this case Finn) needs to reopen a ticket to further resolve an issue.

**Initial Step-By-Step Description**

1. The Technician opens page with all the closed tickets that he recently closed.
2. The Program displays all the tickets that recently were closed by the Technician that is logged in.
3. The Technician click on the reopen button for the correct ticket.
4. The Program gives the Technician a textbox to type in to add a reason the ticket has been reopened.
5. The Program sends data to the Server to Reopen the ticket.
6. The Program opens the ticket.

**Xref:** Table 5 – Overview Tickets, Table 6 – Viewing a Ticket, Table 7 – Add a comment to Ticket. Research Document – Finn’s Scenario

## References

Burak, A. (2023). Your 2023 Guide to Writing a Software Requirements Specification (SRS) Document. *Relevant Software*. https://relevant.software/blog/software-requirements-specification-srs-document/

Engineers, I. O. E. a. E. (1998). *IEEE Recommended Practice for Software Requirements Specifications*.

Krüger, G. (2023). How to Write a Software Requirements Specification (SRS Document). *Perforce Software*. https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document