# 2207-BSE

# Implementation – Evidence Document

# CS106.1

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

# Document Outline

## Table of Contents



## Table of Figures

## Table of Tables

**No table of figures entries found.**

# Intentionally Blank

# Changes

## HiFi

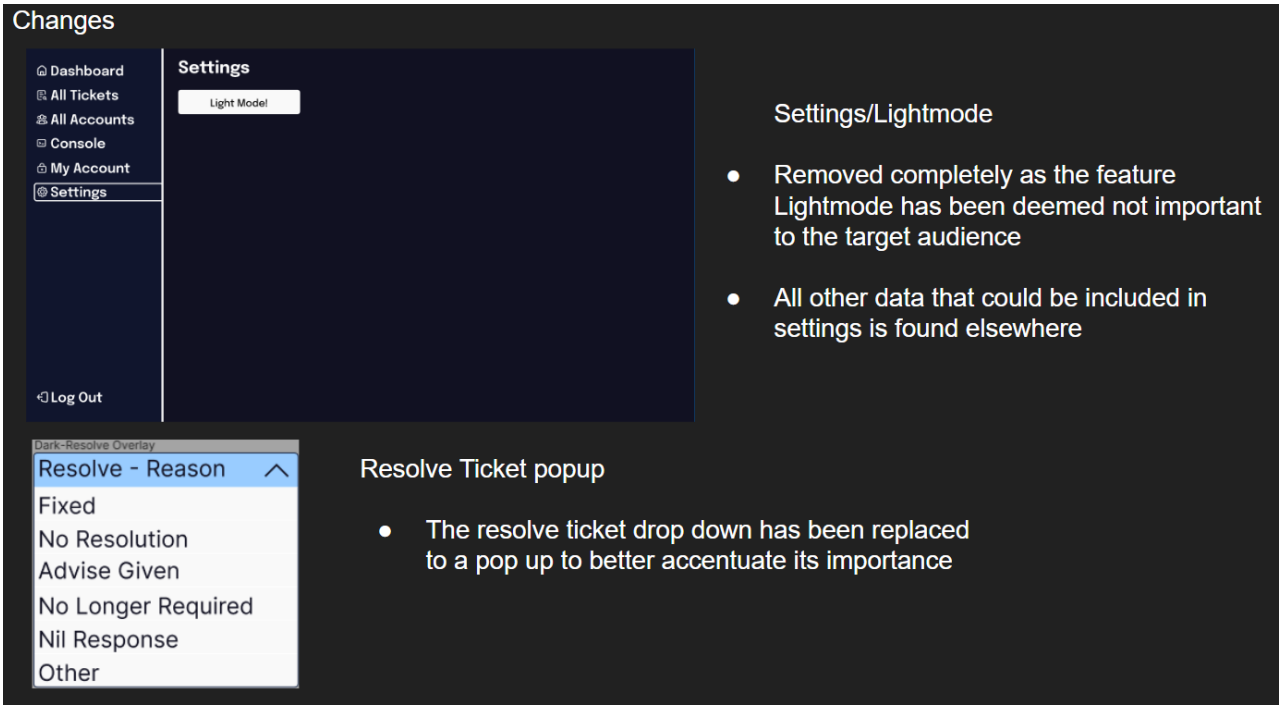



*Figure 1: Hifi Changes*

# System Architecture

## Activity Diagram

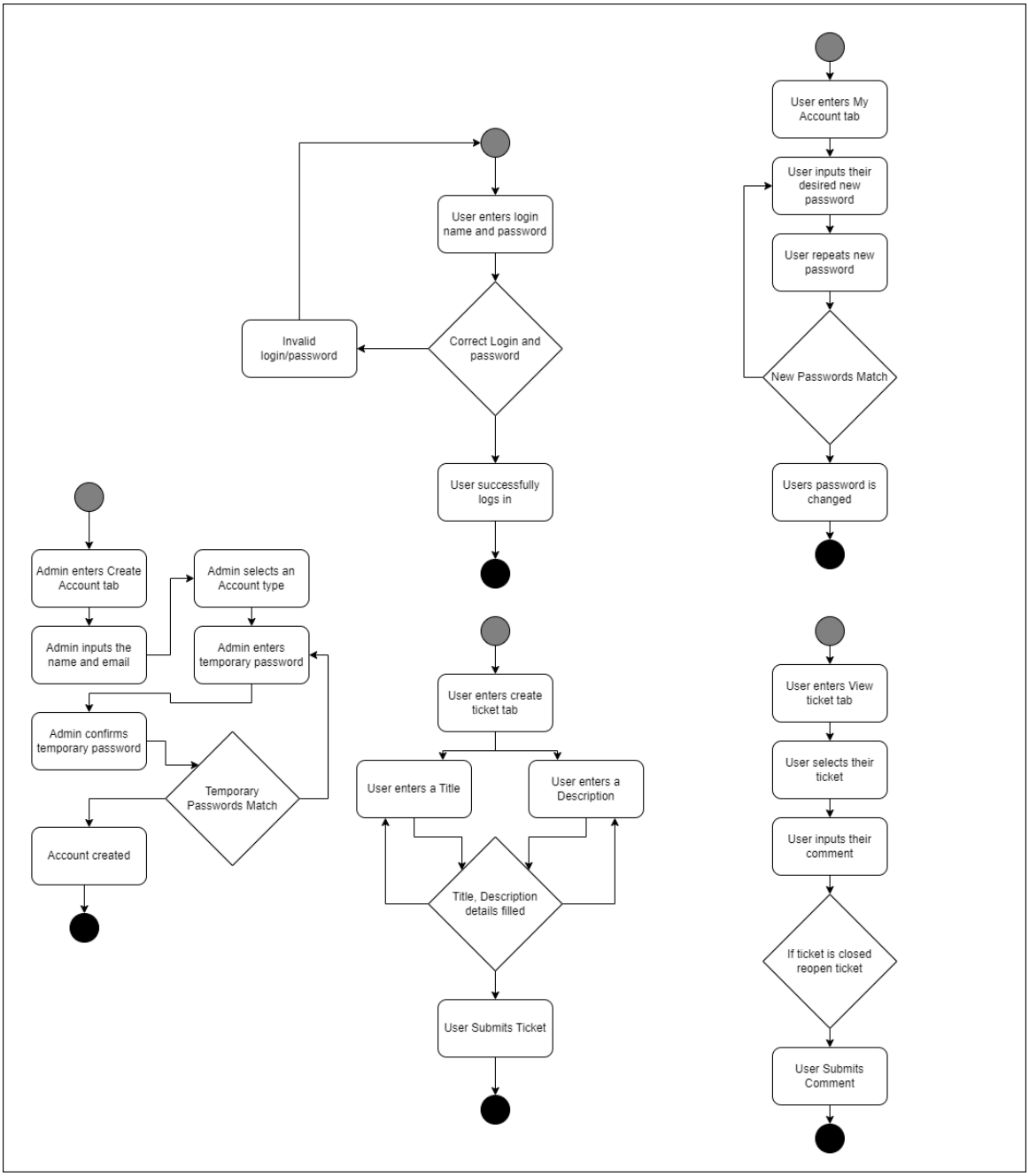



*Figure 2: Activity Diagram*

Implementation

## Class Diagram

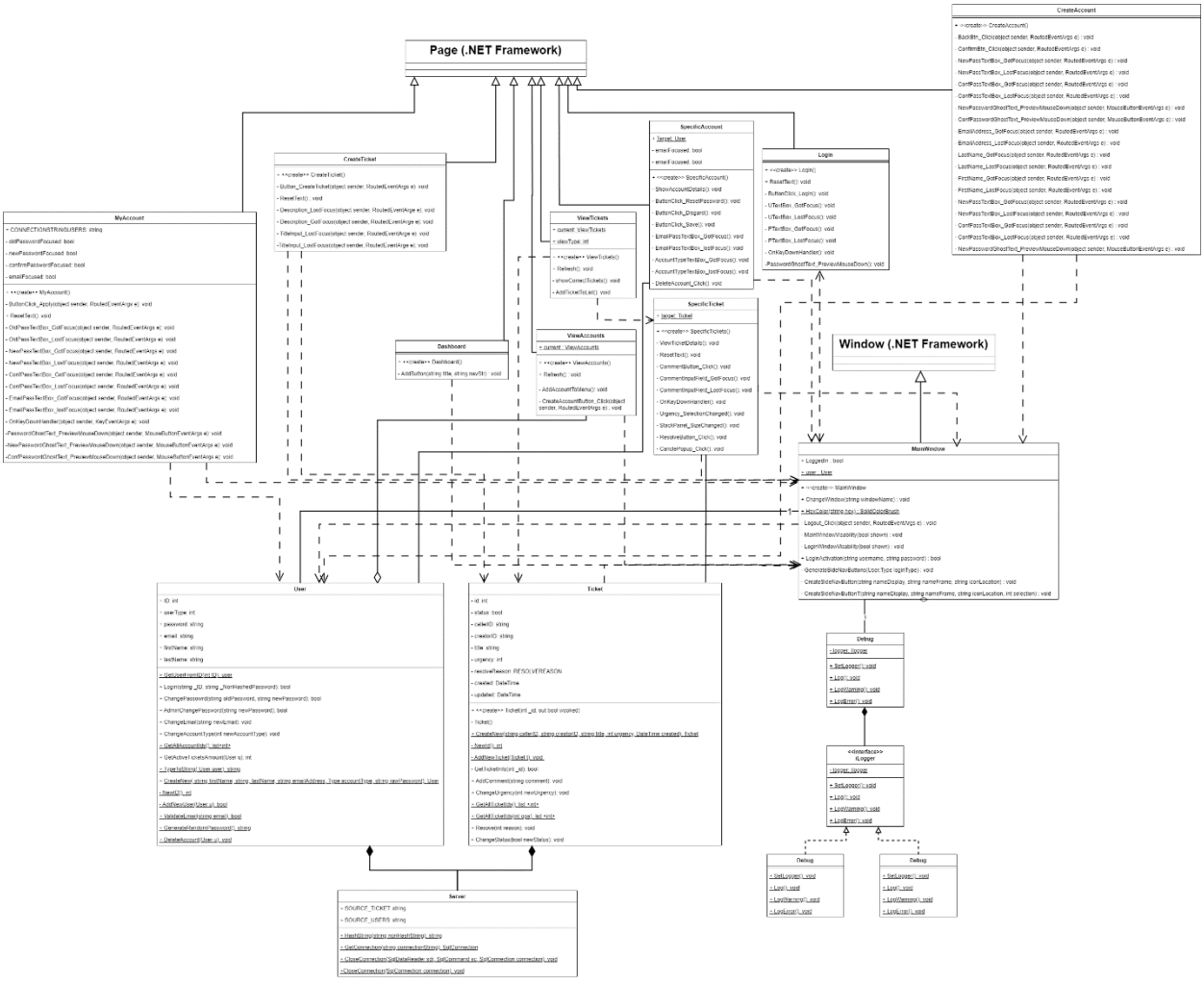



*Figure 3: Class Diagram*

## Justification

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.

## Key Functionality

Project Functionality Screenshots:

**Logging in:**

When the “LogIn” button is clicked in the system “ButtonClick_Login” runs  and checks if the password and username is correct and if not the “MessageBox Result” = “Incorrect Credentials”.

C#:

```
/// <summary>
/// trys to login the user with credentials from the user
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void ButtonClick_Login(object sender, RoutedEventArgs e)
{
    // log the user in - if unsuccessful alert user and reset textboxes
    if(!((MainWindow)Application.Current.MainWindow).LoginActivation(LoginUserName.Text, LoginPassword.Password))
    {
        ResetText();
        MessageBoxResult wrongCredentials = MessageBox.Show("Incorrect credentials!");
    }
}
```

*Figure 4: Key Functionality #1*

System:

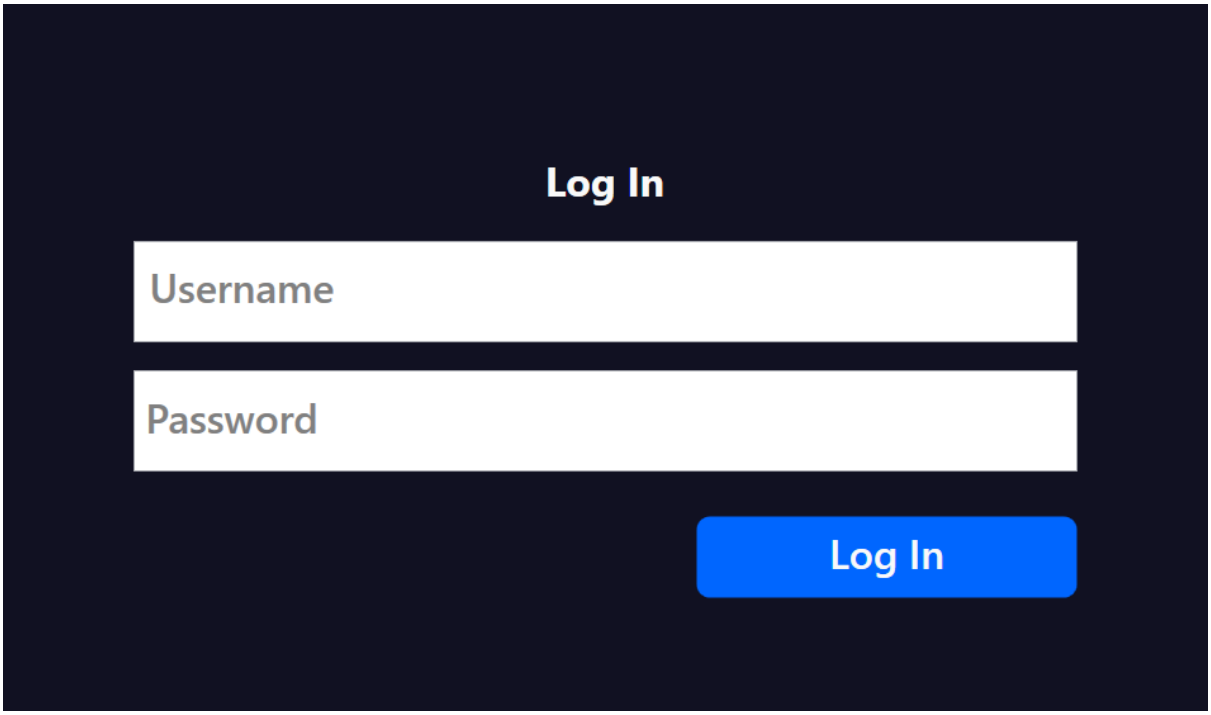



*Figure 5: Key Functionality #2*

**Create Ticket:**

When Submit is Selected, System checks Title (“TitleInput.text”) Urgency (Urgency.SelectedIndex”) Creator ID (“current.ID.ToString()”) who its created for (“CreatedFor.text”) and lastly the description (“Description.Text”) and saves the data to the database. If Description and Title are not filled in Users will be shown a text box via the “if” statement. The ticket is then created and the user's view is now replaced by the ticket they just created.

C#:

```
public CreateTicket()
{
    User current = MainWindow.user;
    InitializeComponent();
    CreatedBy.Text = current.ID.ToString(); // sets created by to this user
    CreatedFor.Text = current.ID.ToString(); // sets created for to this user by default (can be changed while in application)
}

/// <summary>
/// creates a ticket if required fields are valid
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void Button_CreateTicket(object sender, RoutedEventArgs e)
{
    User current = MainWindow.user; // get current user logged in

    // get all variables from XAML inputs
    string title = TitleInput.Text;
    int urgency = Urgency.SelectedIndex + 1; // 1 2 3 for high medium low
    string creatorID = current.ID.ToString();
    string createdFor = CreatedFor.Text;
    string description = Description.Text;

    // check all required values are valid (stop if invalid)
    if (TitleInput.Text == "Title" || TitleInput.Text == "")    //  IF THE USER HAS NOT ENTERED A TITLE
    {
        MessageBox.Show("Please enter a title");
        return;
    }
    else if (Description.Text == "Description" || Description.Text == "")   //  IF THE USER HAS NOT ENTERED A DESCRIPTION
    {
        MessageBox.Show("Please enter a description");
        return;
    }

    // create ticket
    Ticket t = Ticket.CreateNew(createdFor, creatorID, title, urgency, DateTime.Now);
    t.AddComment(description);

    // change the window to view the newly created ticket
    MainWindow window = (MainWindow)Application.Current.MainWindow;
    SpecificTicket.target = t;
    window.ChangeWindow("SpecificTicket.xaml");
```

*Figure 6: Key Functionality #3*

System:

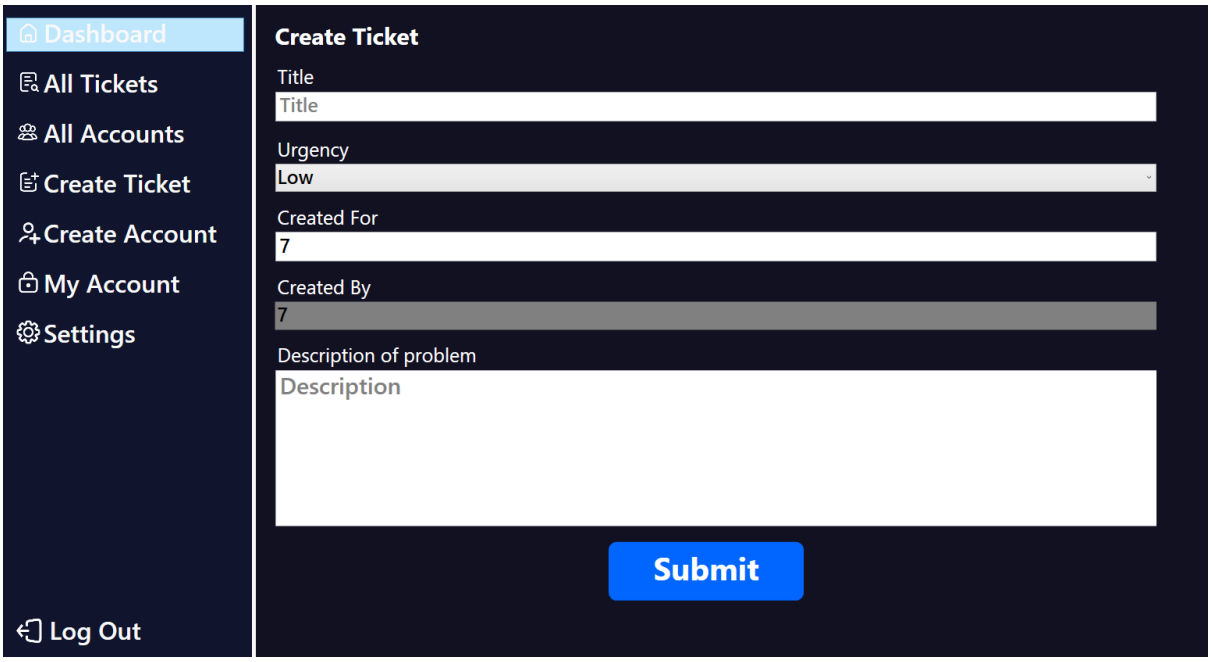



*Figure 7: Key Functionality #4*

**Creating Account:**

When creating an account and confirm is selected the system checks for Name ("firstName.Text", "lastName.Text") Email ("EmailAddress.Text") Account Type ("AccountType.SelectedIndex") and Password ("NewPassword.Password"). If these details are not filled in, the "else if" statement will trigger the respective "MessageBox.Show" to trigger.

C#:

```
/// <summary>
/// creates the account if the data is valid
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void ConfirmBtn_Click(object sender, RoutedEventArgs e)
{
    if(NewPassword.Password == ConfPassword.Password && User.ValidateEmail(EmailAddress.Text))
    {
        User u = User.CreateNew(FirstName.Text, LastName.Text, EmailAddress.Text, (User.Type)AccountType.SelectedIndex + 1, NewPassword.Password);
        SpecifcAccount.target = u;
        MainWindow mw = (MainWindow)Application.Current.MainWindow;
        mw.ChangeWindow("SpecifcAccount.xaml");
    }
    else if (!User.ValidateEmail(EmailAddress.Text))
    {
        MessageBoxResult invalidEmail = MessageBox.Show("Email address already in use!");
    }
}
```

*Figure 8: Key Functionality #5*

System:

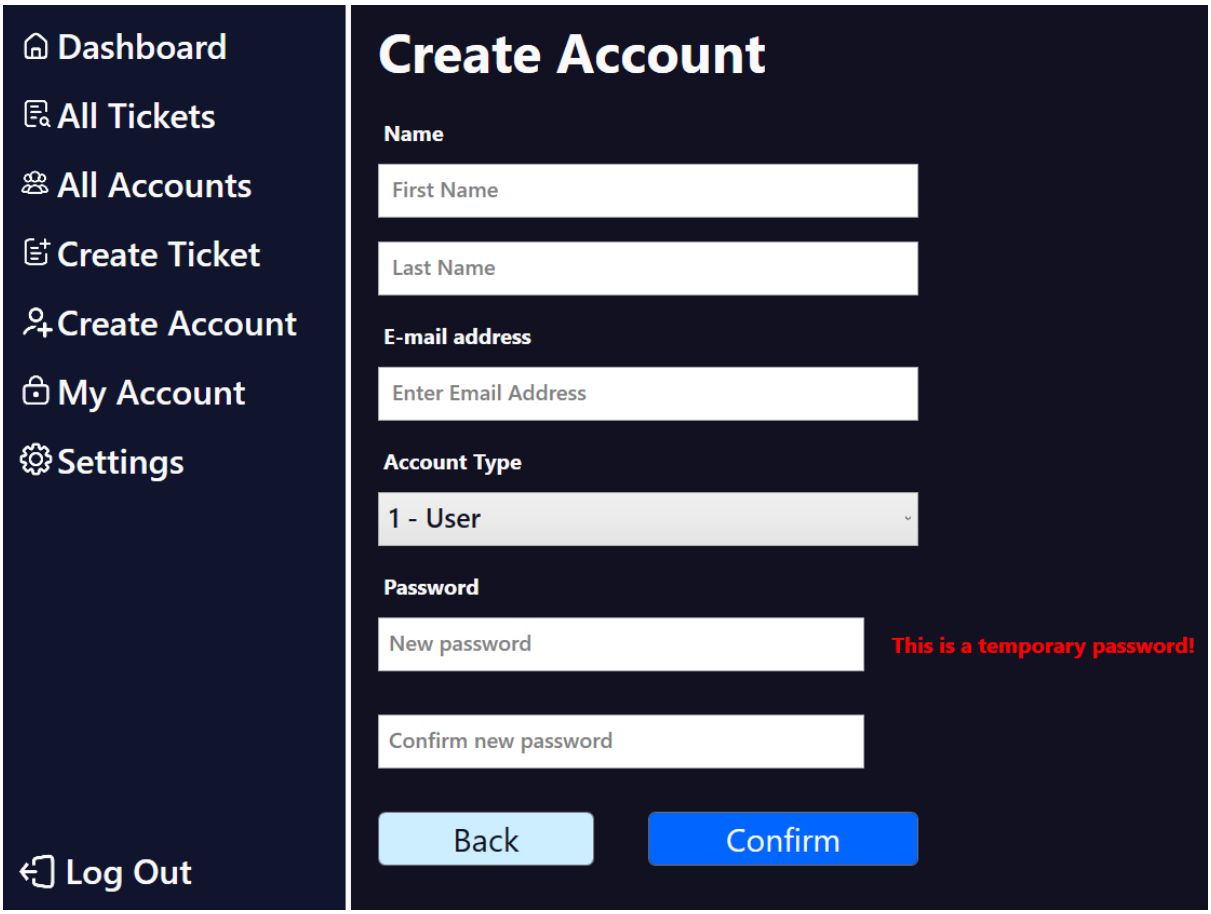



*Figure 9: Key Functionality #6*

**Add Comment:**

String “amended comment” allows user to input a comment in a ticket while saving and inserting their Name and Time. When the button “Submit Comment” is clicked the input comment will “Try” to add the comment to the ticket in the database and if it fails it alerts the user (“Catch”).

C#:

Implementation

```
public void AddComment(string comment)
{
    try
    {
        string amendedComment = "" + MainWindow.user.firstName + "¦" + MainWindow.user.lastName + "¦" + DateTime.Now.ToString() + "¦" + comme
        comments.Add(amendedComment);
        amendedComment = string.Empty;
        foreach (string c in comments)
        {
            amendedComment += c + '♦';
        }
        if (amendedComment.EndsWith("♦"))
        {
            amendedComment = amendedComment.Remove(amendedComment.Length - 1, 1); // remove last symbol
        }

        using (SqlConnection connection = Server.GetConnection(Server.SOURCE_TICKET))
        {
            //  FILESTREAM / WRITER, ALLOWS INSERTING / UPDATING ROWS IN SQL
            SqlDataAdapter adapter = new SqlDataAdapter();
            string commandText = "UPDATE AllTickets SET COMMENTS=@comment WHERE ID='" + this.id + "';";
            adapter.InsertCommand = new SqlCommand(commandText, connection);
            adapter.InsertCommand.Parameters.AddWithValue("@comment", amendedComment);
            adapter.InsertCommand.ExecuteNonQuery();
            Server.CloseConnection(connection);
        }

        using (SqlConnection connection = Server.GetConnection(Server.SOURCE_TICKET))
        {
            SqlDataAdapter adapter = new SqlDataAdapter();
            string commandText = "UPDATE AllTickets SET UPDATED='" + DateTime.Now.ToString() + "' WHERE ID='" + this.id + "';";
            adapter.InsertCommand = new SqlCommand(commandText, connection);
            adapter.InsertCommand.ExecuteNonQuery();
            Server.CloseConnection(connection);
        }
    }
    catch (Exception e)
    {
        Debug.LogWarning("Operation Unsuccessful - " + e.Message);
        MessageBox.Show("Operation was not successful!\nPlease try again...", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

*Figure 10: Key Functionality #7*

System:

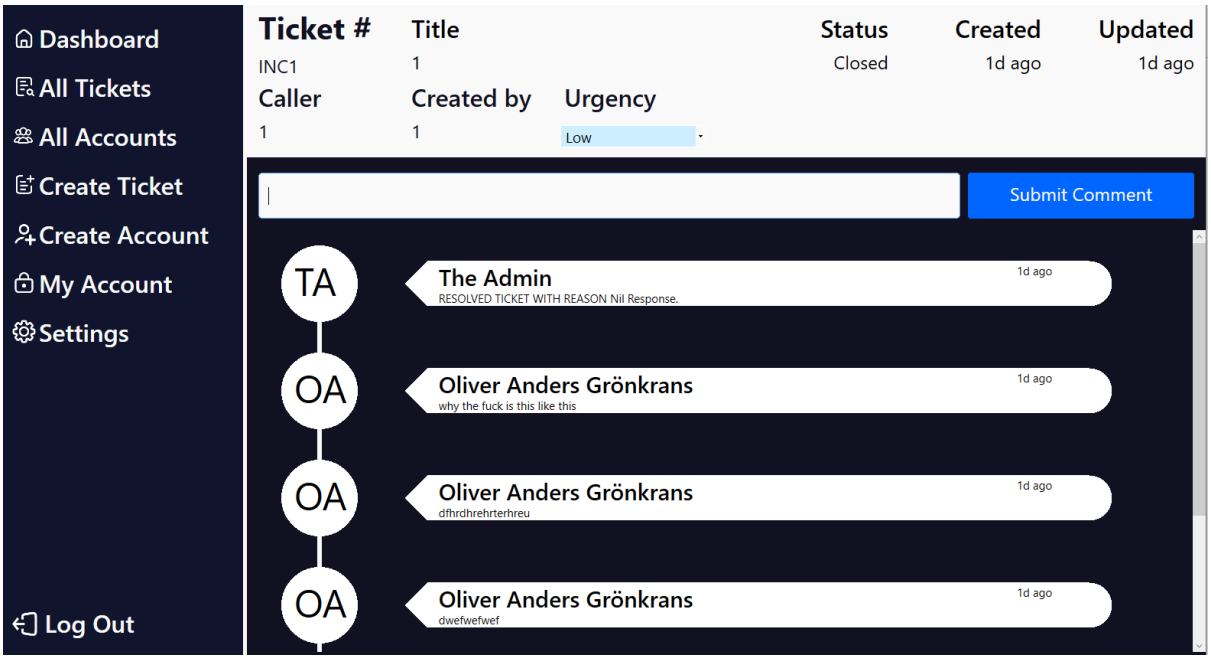



*Figure 11: Key Functionality #8*

**Change password:**

When changing password the system checks the database to see if your old password matches (“oldPassword = Server.HashString(oldPassword)”) and if your new password matches the confirmed password changes (“newPassword = Server.HashString(newPassword)”) if not the user is alerted (“Catch”).

C#:

```
/// <summary>
/// changes the password of the current instance of the user if the old password matches the current password
/// </summary>
/// <param name="oldPassword"></param>
/// <param name="newPassword"></param>
/// <returns></returns>
1 reference
public bool ChangePassword(string oldPassword, string newPassword)
{
    try
    {
        oldPassword = Server.HashString(oldPassword);
        newPassword = Server.HashString(newPassword);
        //  CHECKS IF THE NEW PASSWORDS MATCHES, AND IF THE OLD PASSWORD MATCHES THEIR CURRENT PASSWORD
        if (oldPassword == password)
        {
            SqlConnection connection = Server.GetConnection(Server.SOURCE_USERS);
            SqlDataAdapter adapter = new SqlDataAdapter();

            string commandText = "UPDATE Users SET Password=@password WHERE ID='" + ID + "';";
            adapter.InsertCommand = new SqlCommand(commandText, connection);
            adapter.InsertCommand.Parameters.AddWithValue("@password", newPassword);
            adapter.InsertCommand.ExecuteNonQuery();

            Server.CloseConnection(connection);
            password = newPassword;
            return true;
        }
        else
        {
            return false;
        }
    }
    catch (Exception e)
    {
        Debug.LogWarning("Operation Unsuccessful - " + e.Message);
        MessageBox.Show("Operation was not successful!\nPlease try again...", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        return false;
    }

}
```

*Figure 12: Key Functionality #9*

System:

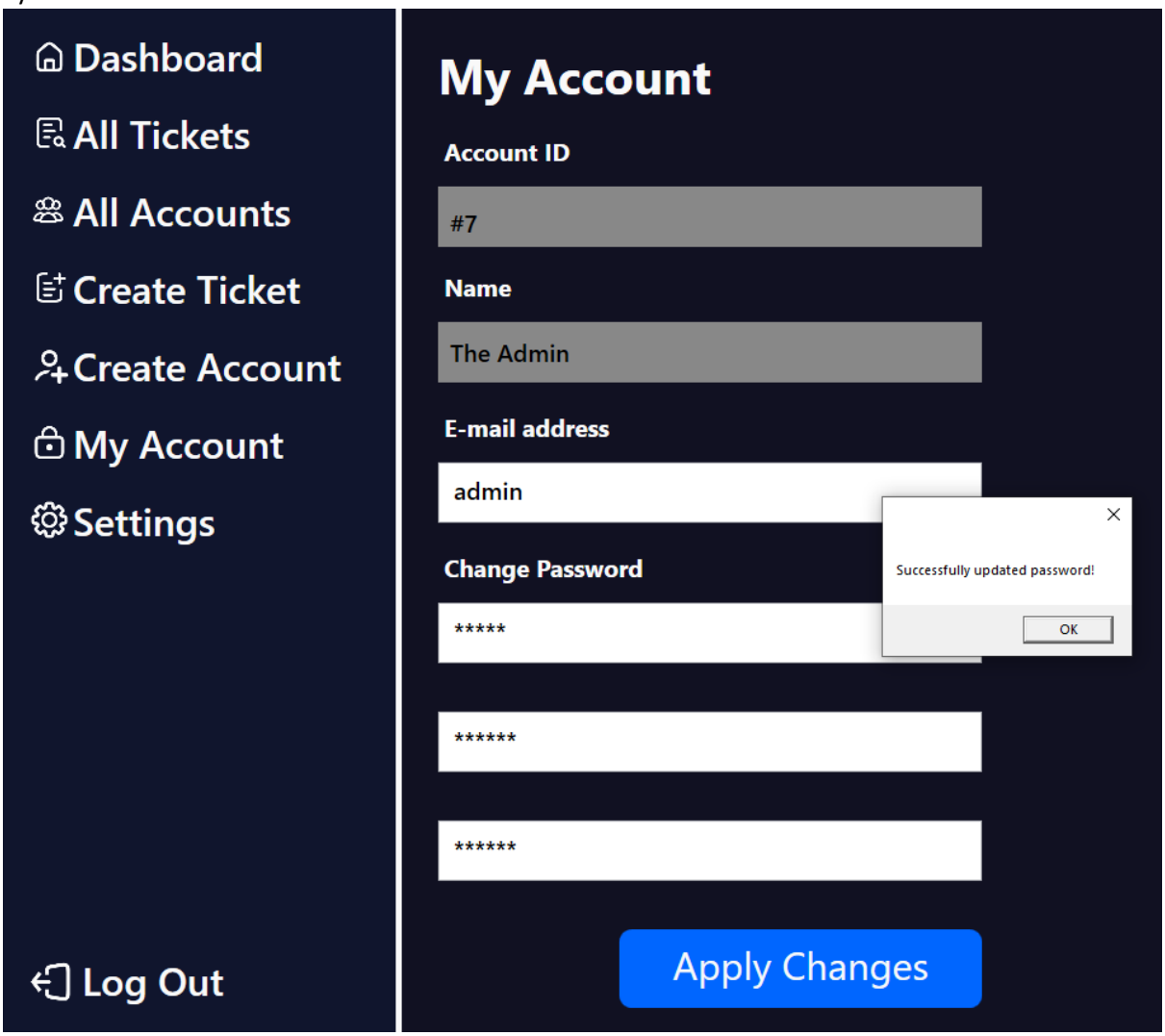



*Figure 13: Key Functionality #10*

# Function Testing

## Black Box Testing

**Test cases**

1. **Input “bad” input (spaces, special characters, etc) in login form**

**Expected output**

Login attempt fails as with any other invalid credentials.

**Used parameter**

Username, password: 0 1 2 3 4 5 6 7 8 9 ! “ # ¤ % & / ( ) = ? ` ´ | < >

**Received output**

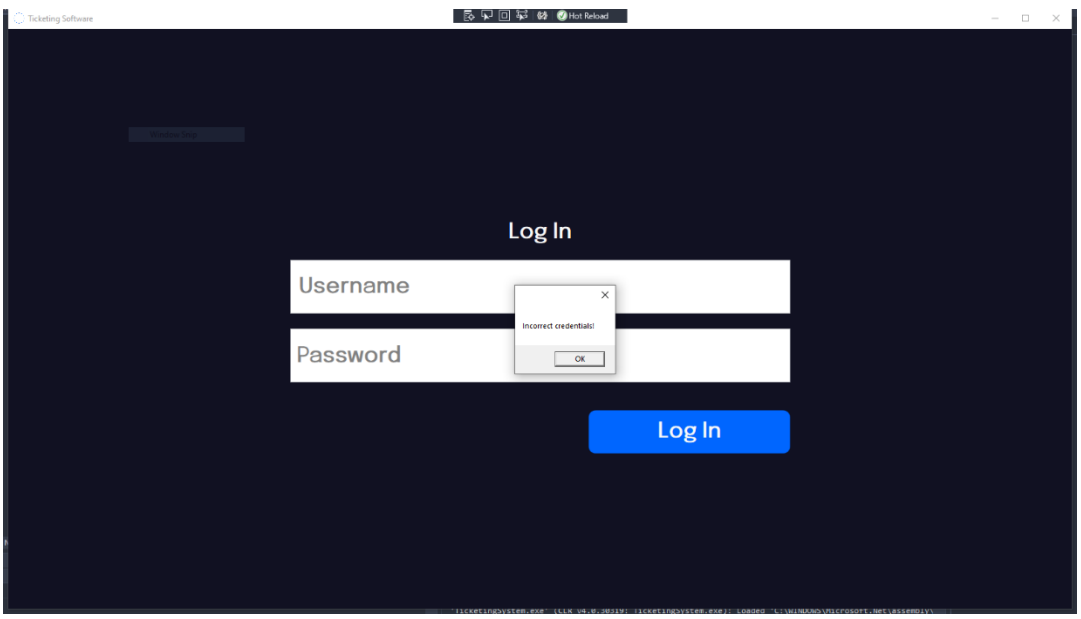



*Figure 14: Functional Testing #1*

**Result**

Passed

2. **Input spaces and special characters in new ticket**

**Expected output**

Ticket is created with corresponding data without issue, and the same data can be loaded in the ticket view page.

**Used parameters**

Title, Caller, Description: 0 1 2 3 4 5 6 7 8 9 ! “ # ¤ % & / ( ) = ? ` ´ | < >

**Received output**

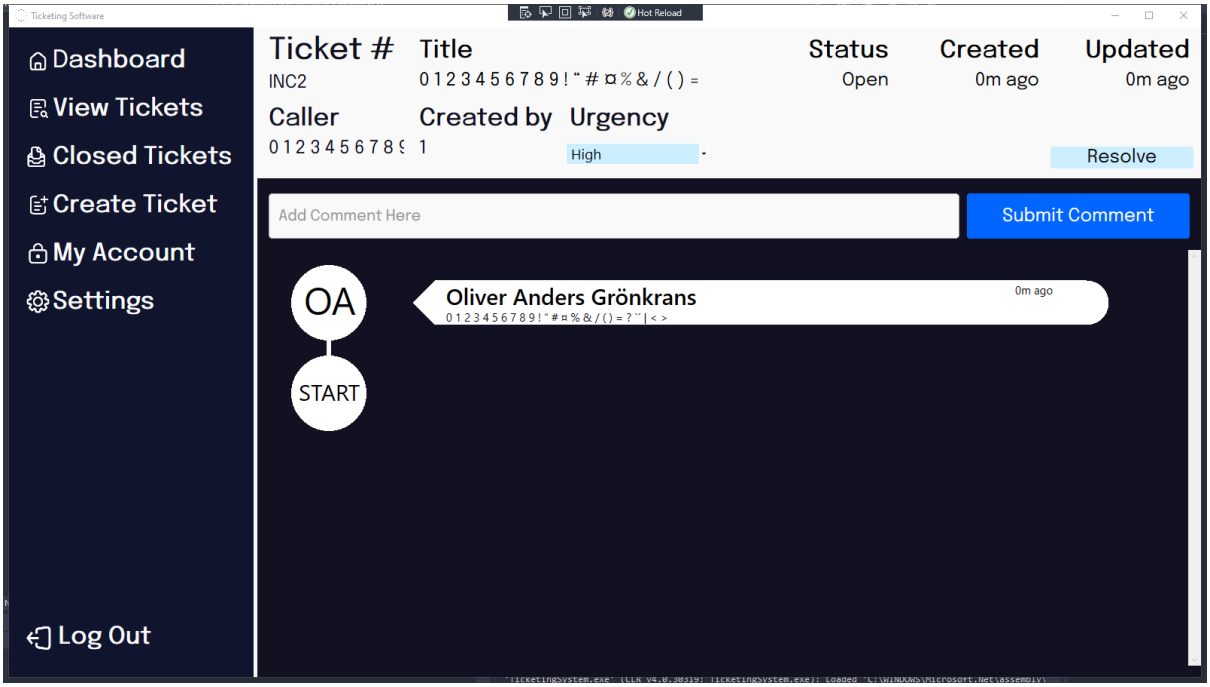



*Figure 15: Functional Testing #2*

**Result**

Passed

**3. Resolve ticket**

**Expected output**

Changes documented in comment field without issue.

**Used parameters**

Resolve status - Fixed

**Received output**

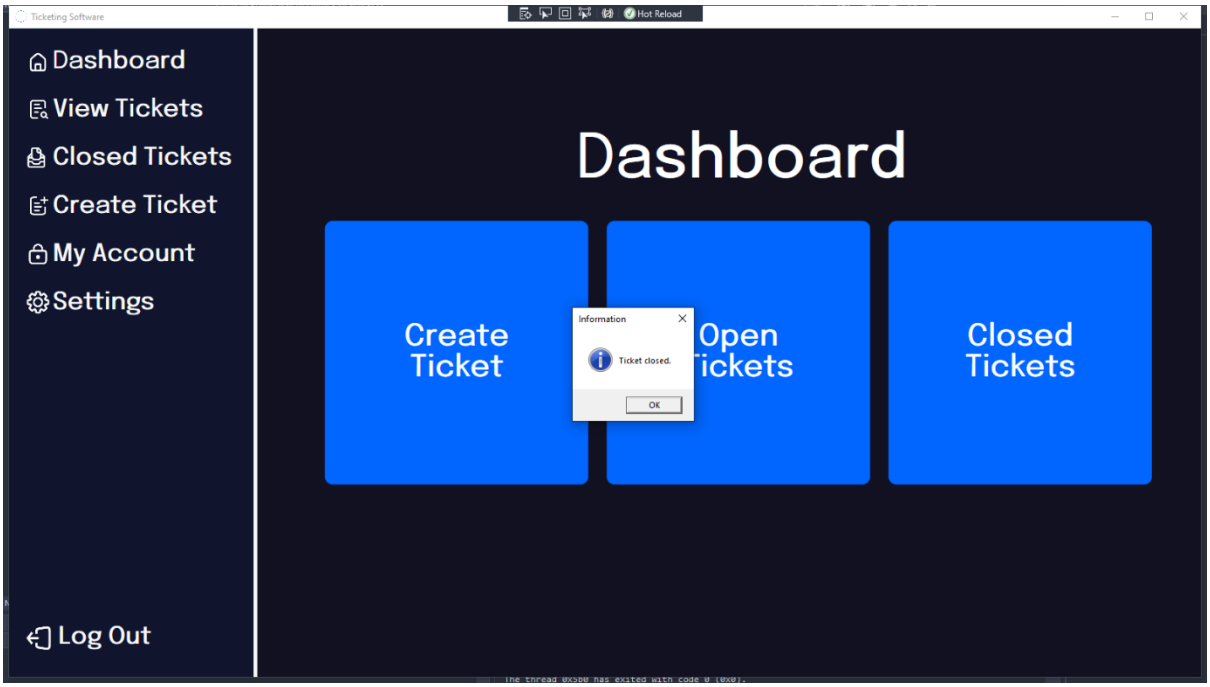



*Figure 16: Functional Testing #3*

**Result**

Passed

**4. Reopen ticket with comment containing spaces and special characters**

**Expected output**

Ticket is reopened, comment is added, and status update is added without issue.

**Used parameters**

Comment text: 0 1 2 3 4 5 6 7 8 9 ! “ # ¤ % & / ( ) = ? ` ´ | < >

**Received output**

Implementation

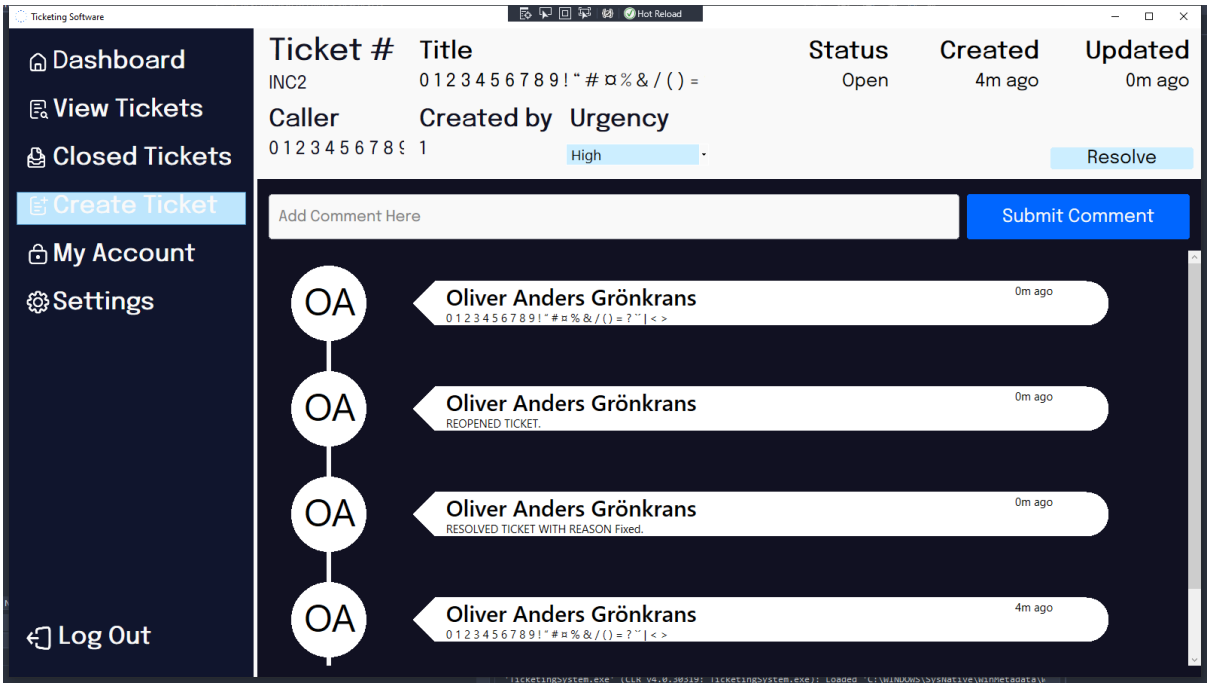



*Figure 17: Functional Testing #4*

**Result**

Passed

**5. Try to create account with an e-mail address which is already in use**

**Expected output**

Account creation is denied, with error message stating that the e-mail address already is in use.

**Used parameters**

Email: 270045020@yoobeestudent.ac.nz (used by user 1)

**Received output**

Implementation

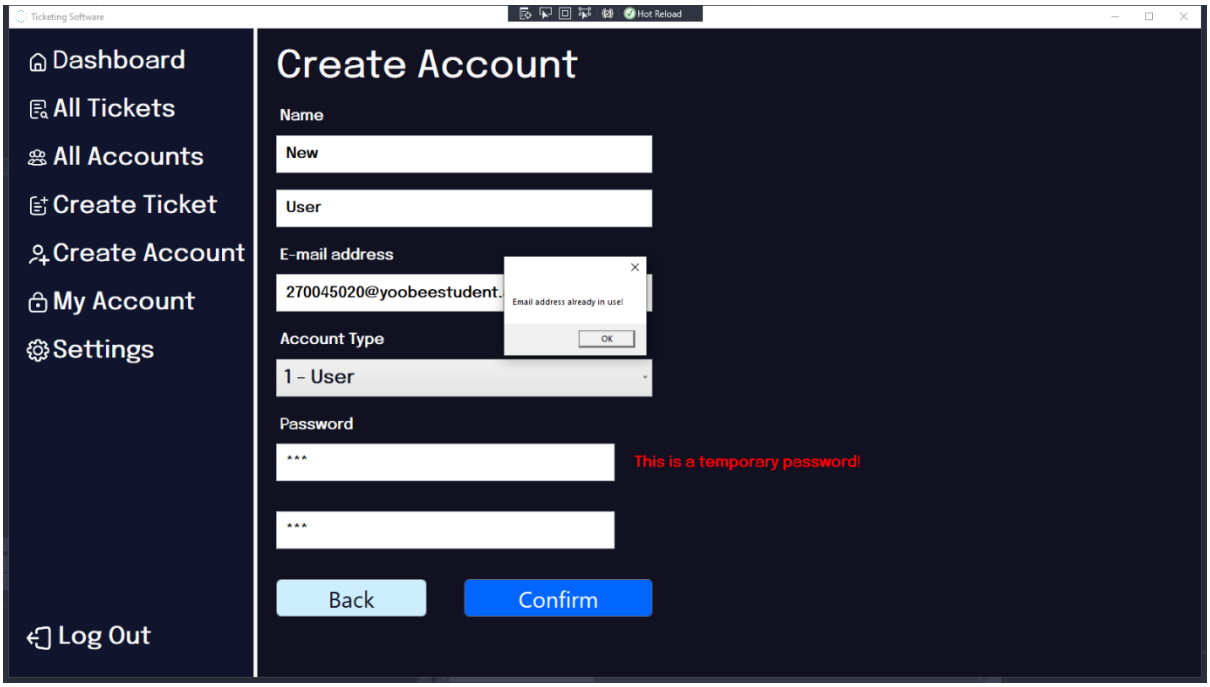



*Figure 18: Functional Testing #5*

**Result**

Passed

**6. Create account with two first names and two last names**

**Expected output**

Account is created without issue and the name is displayed correctly.

**Used parameters**

First name: Fredrik Anders

Last name: Andersson Stigstorp

**Received output**

Implementation

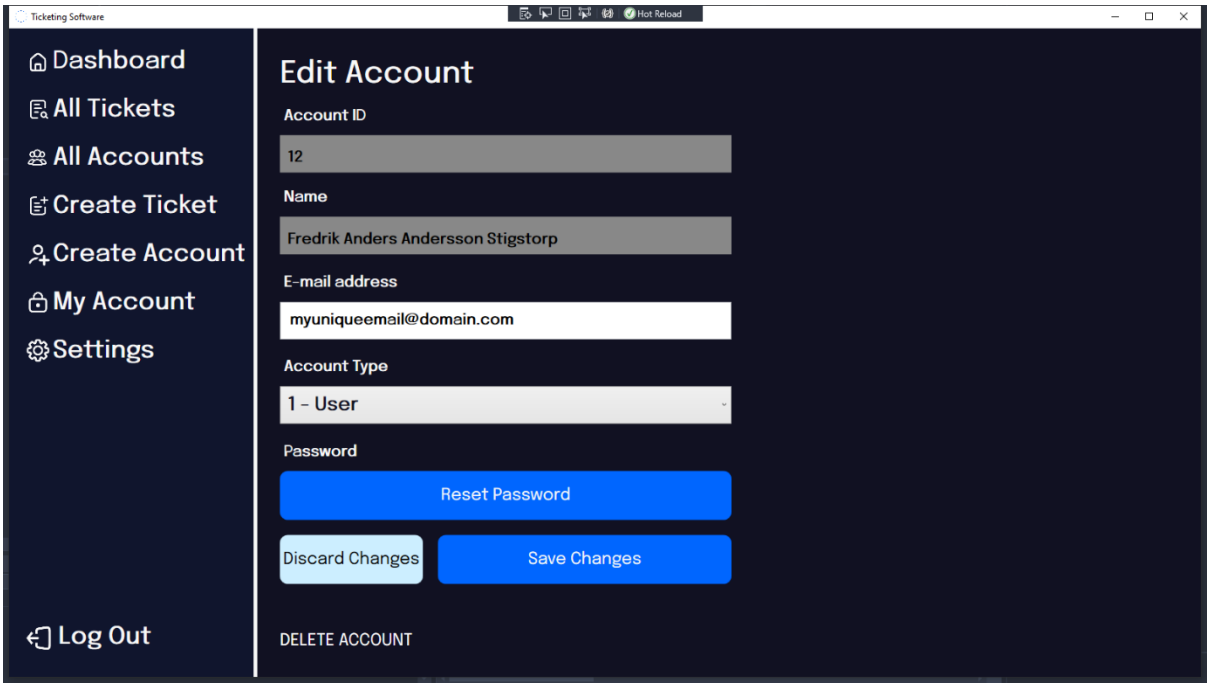



*Figure 19: Functional Testing #6*

**Result**

Passed

**7. Delete account with ID 8**

**Expected output**

The account with ID 8 (and no other account) is deleted without issue.

**Used parameter**

User: ID 8 out of 10

**Received outputs**

Implementation

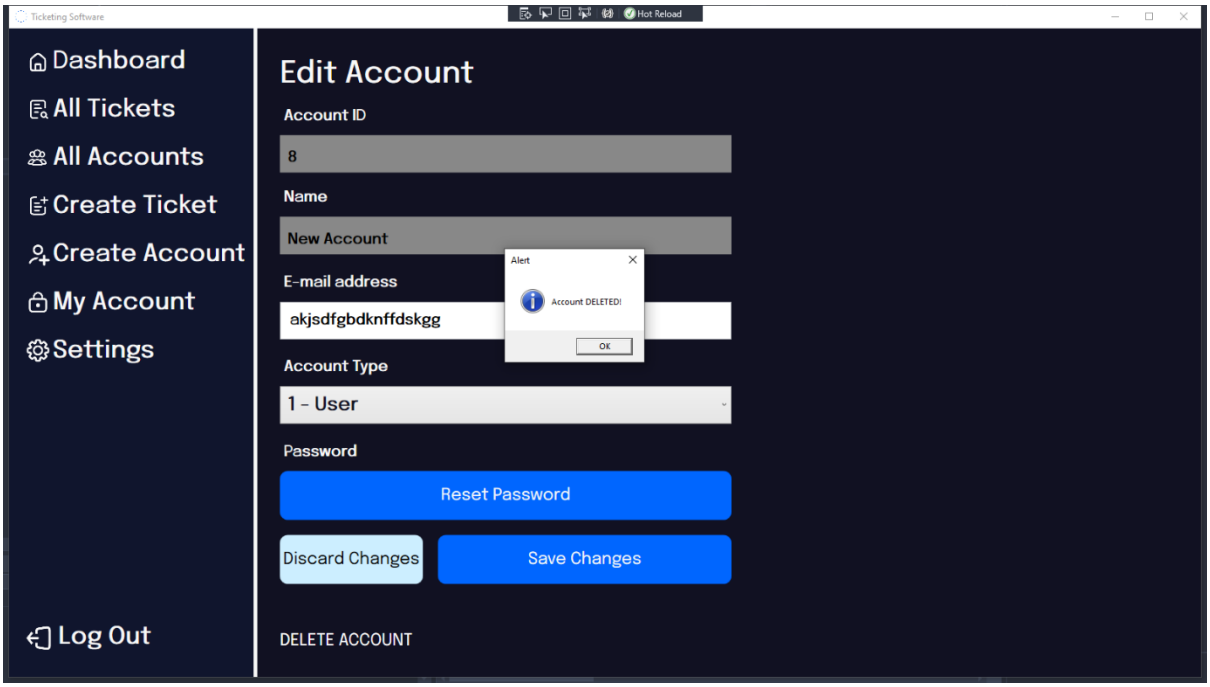



*Figure 20: Functional Testing #7*

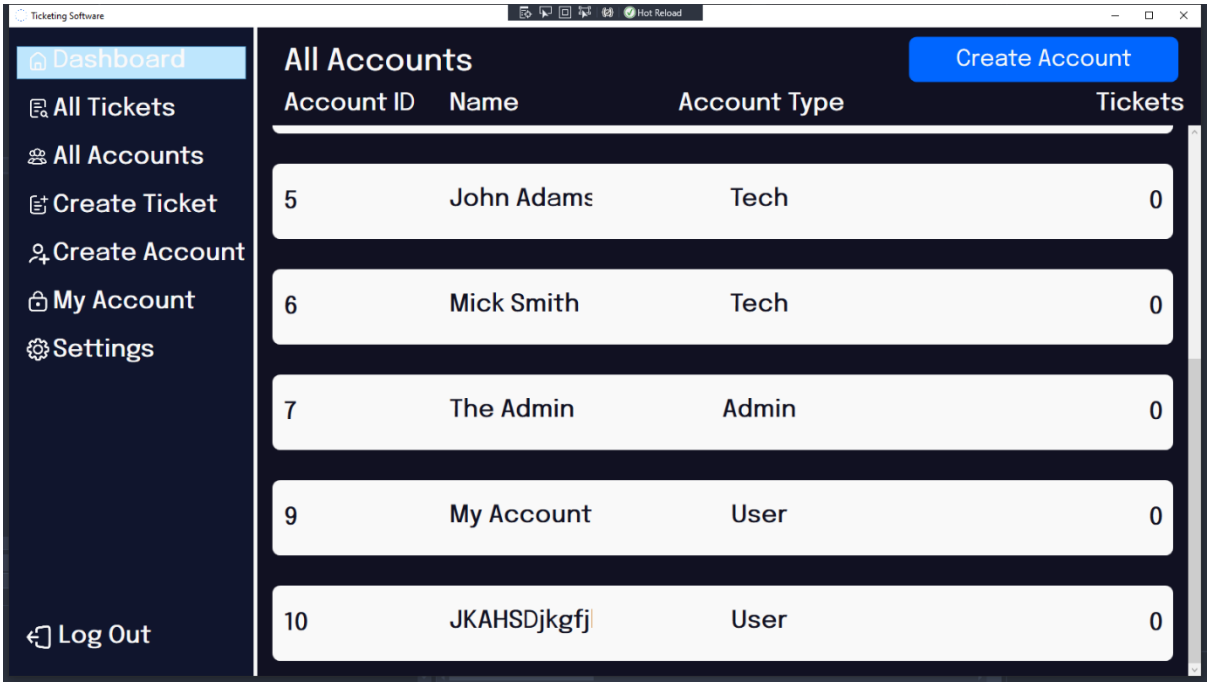



*Figure 21: Functional Testing #8*

**Result**

Passed

## 8. Create new account after deleting account with non-edge ID

**Expected output**

New account gets ID of one higher than the last account in the database, and not an ID generated of the length of the database which would result in overlapping ID's.

**Used parameters**

Database: Contains users 1-7, 9-10

First name: Lisbeth

Last name: Olsson

E-mail: lisbeth@olsson.se

Account type: 1 (user)

Password: 123

**Received output**

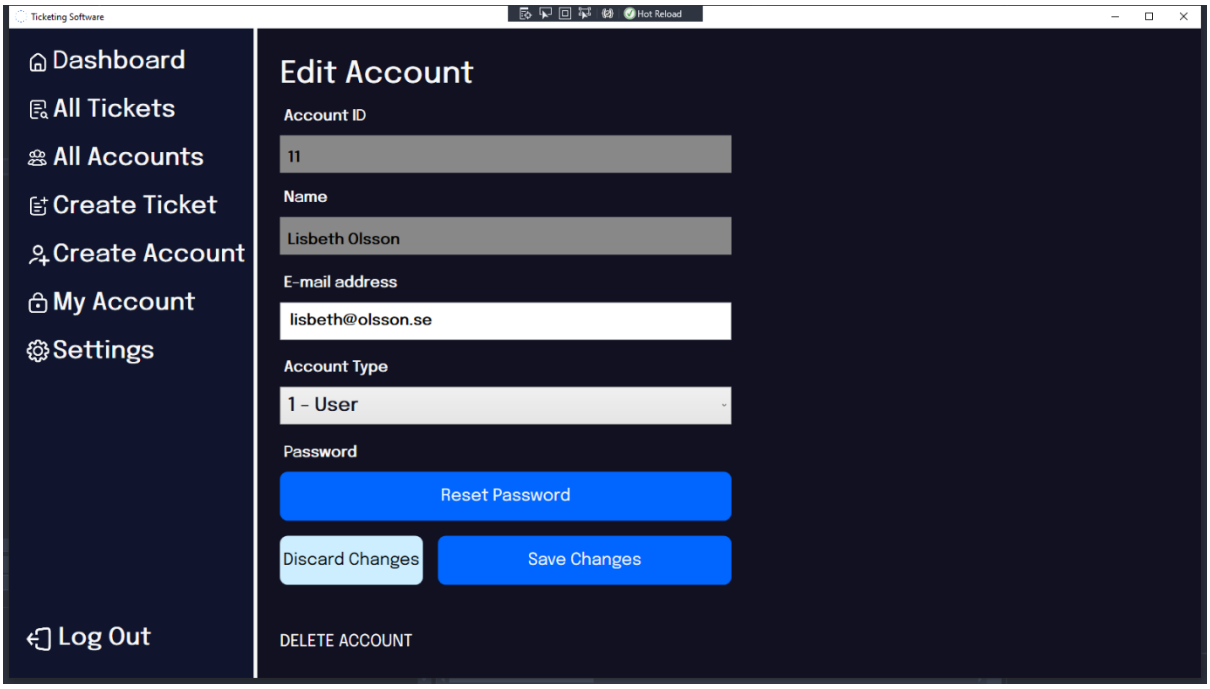



*Figure 22: Functional Testing #9*

**Result**

Passed

**9. Create ticket for other user (as in technician creates a ticket for a user)**

**Expected output**

Ticket is created and the user which it is created for can access it.

**Used parameters**

Creator: User with ID 7

Caller: User with ID 1

Ticket title: Ticket for other user

Ticket description: This should be accessible to user 1

**Received output**

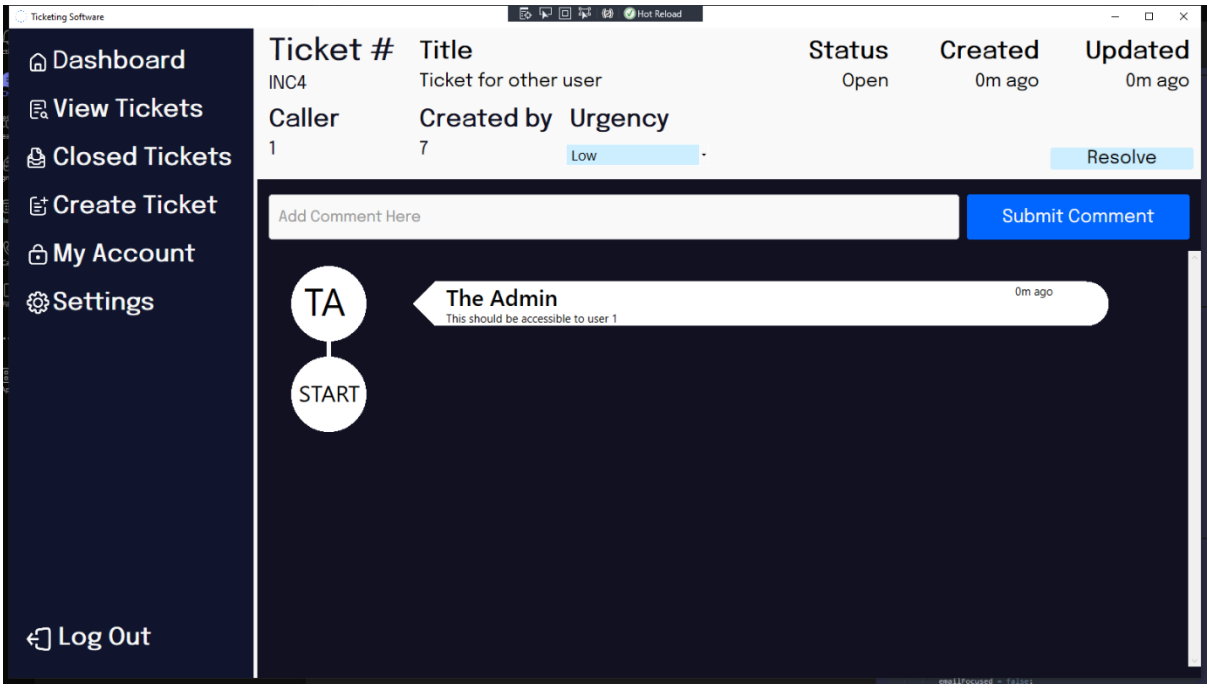



*Figure 23: Functional Testing #10*

**Result**

Passed

**10. Add comment as caller in a multi-user ticket**

**Expected output**

The created comment is added and displayed correctly, with the username of the user who is adding it.

**User parameters**

Ticket: INC4 (Ticket from previous test)

Creator: User with ID 7

Caller/Commenter: User with ID 1

Comment text: This is my own comment

**Received output**

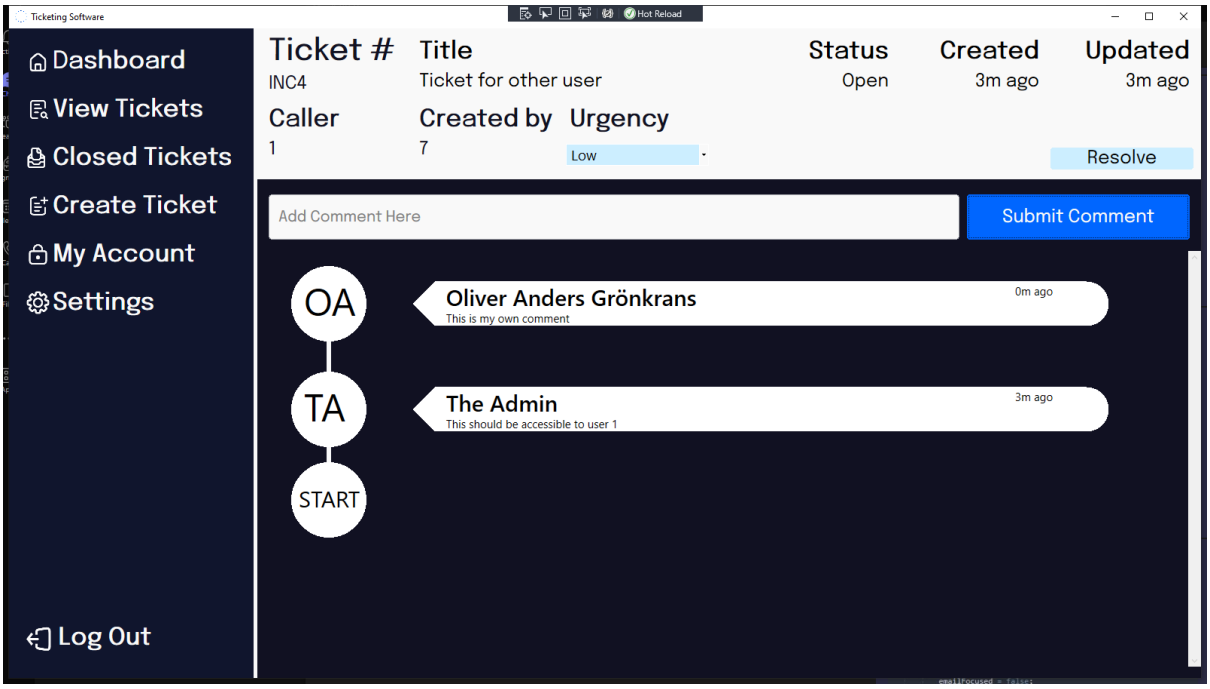



*Figure 24: Functional Testing #11*

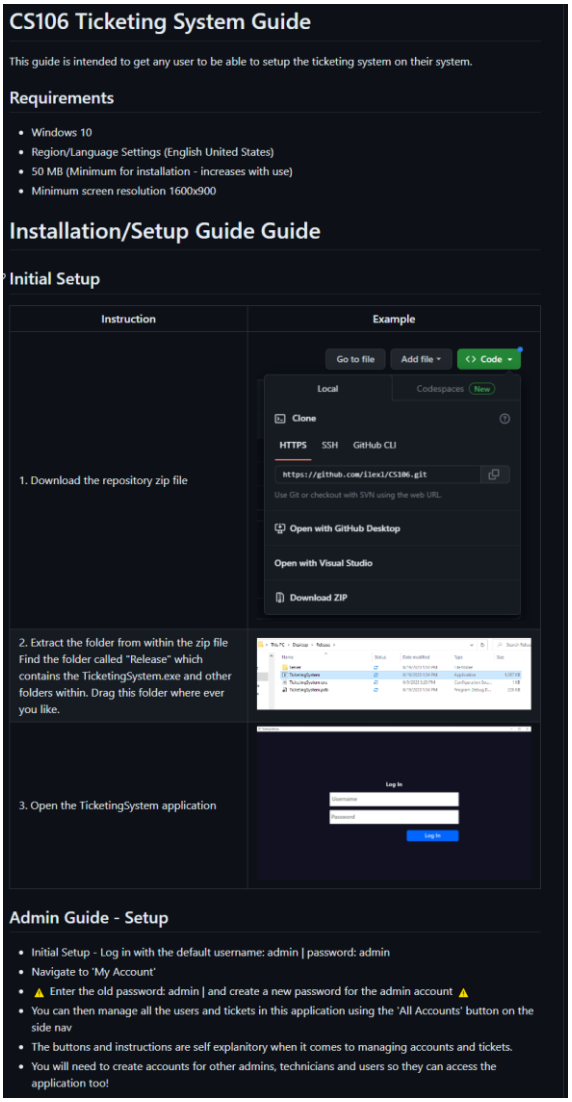
**Result**

Passed

# User Documentation

Easier to read version is on the GitHub repo: https://github.com/ilexl/CS106

## Installation Guide



### CS106 Ticketing System Guide

This guide is intended to get any user to be able to setup the ticketing system on their system.

#### Requirements

- Windows 10
- Region/Language Settings (English United States)
- 50 MB (Minimum for installation - increases with use)
- Minimum screen resolution 1600x900

### Installation/Setup Guide Guide

#### Initial Setup

| Instruction | Example |
|---|---|
| 1. Download the repository zip file | |
| 2. Extract the folder from within the zip file Find the folder called "Release" which contains the TicketingSystem.exe and other folders within. Drag this folder where ever you like. | |
| 3. Open the TicketingSystem application | |

#### Admin Guide - Setup

- Initial Setup - Log in with the default username: admin | password: admin
- Navigate to 'My Account'
- ⚠ Enter the old password: admin | and create a new password for the admin account ⚠
- You can then manage all the users and tickets in this application using the 'All Accounts' button on the side nav
- The buttons and instructions are self explanitory when it comes to managing accounts and tickets.
- You will need to create accounts for other admins, technicians and users so they can access the application too!

*Figure 25: Installation guide*

## User Guide

# User manual

## Admin Guide - Features

### Create an account

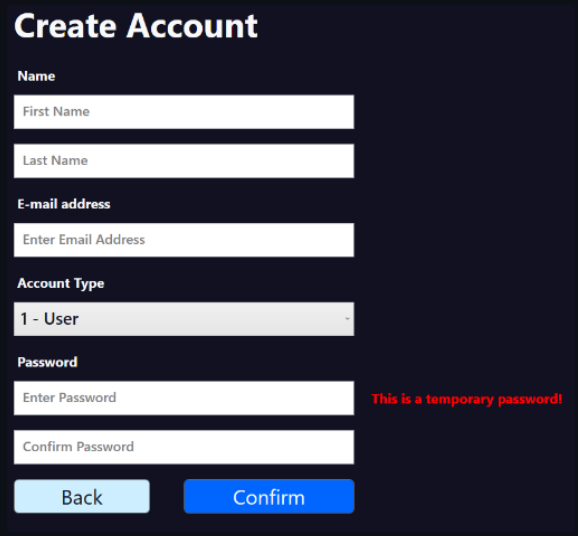
| Instruction | Example |
| --- | --- |
| - Navigate to 'create account'<br>- Fill out all the new account details<br>- ⚠ Make sure you select the correct account type ⚠<br>- Enter a temporary password for the new user<br>- Give the login details (ID, email, temporary password) to the new user and tell them to change the password |  |



*Figure 26: User Guide #1*

Edit an account

| Instruction | Example |
|---|---|
| - Navigate to 'all accounts' | All Accounts<br>Create Account<br>Account ID / Name / Account Type / Tickets<br>5 John Adams Tech 0<br>6 Mick Smith Tech 0<br>7 The Admin Admin 0<br>8 Example Account User 4<br>9 t t Error - Unknc 0 |
| - Select an account<br>- Make edits as per sections | Edit Account<br>Account ID: 9<br>Name: t t<br>E-mail address: t<br>Account Type: 1 - User<br>Password<br>Reset Password<br>Discard Changes / Save Changes |

*Figure 27: User Guide #2*

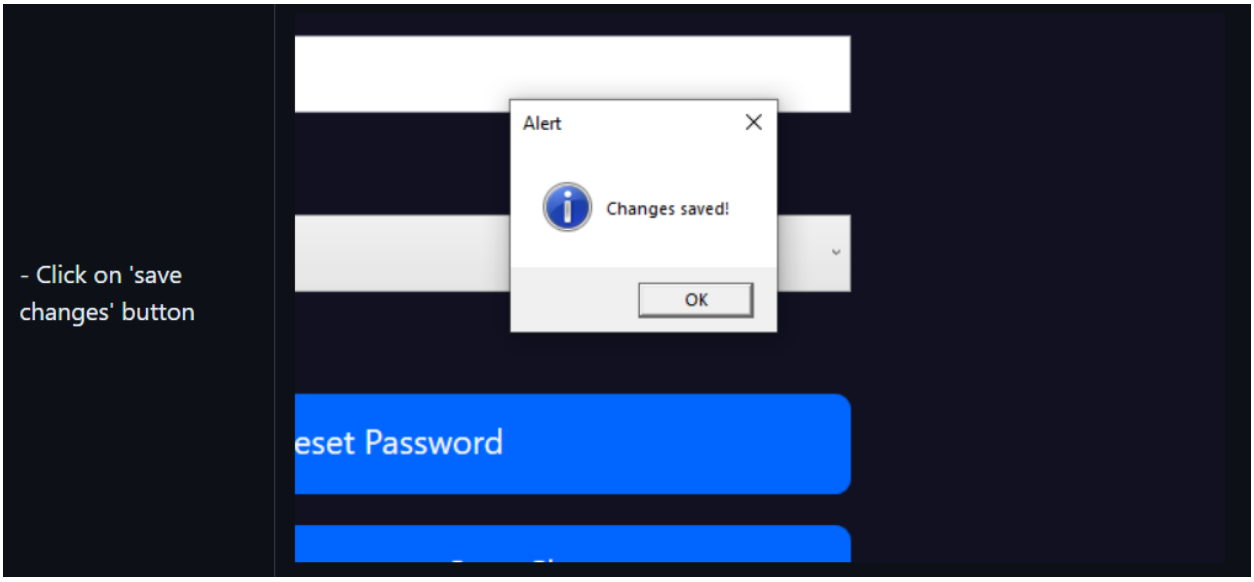



*Figure 28: User Guide #3*

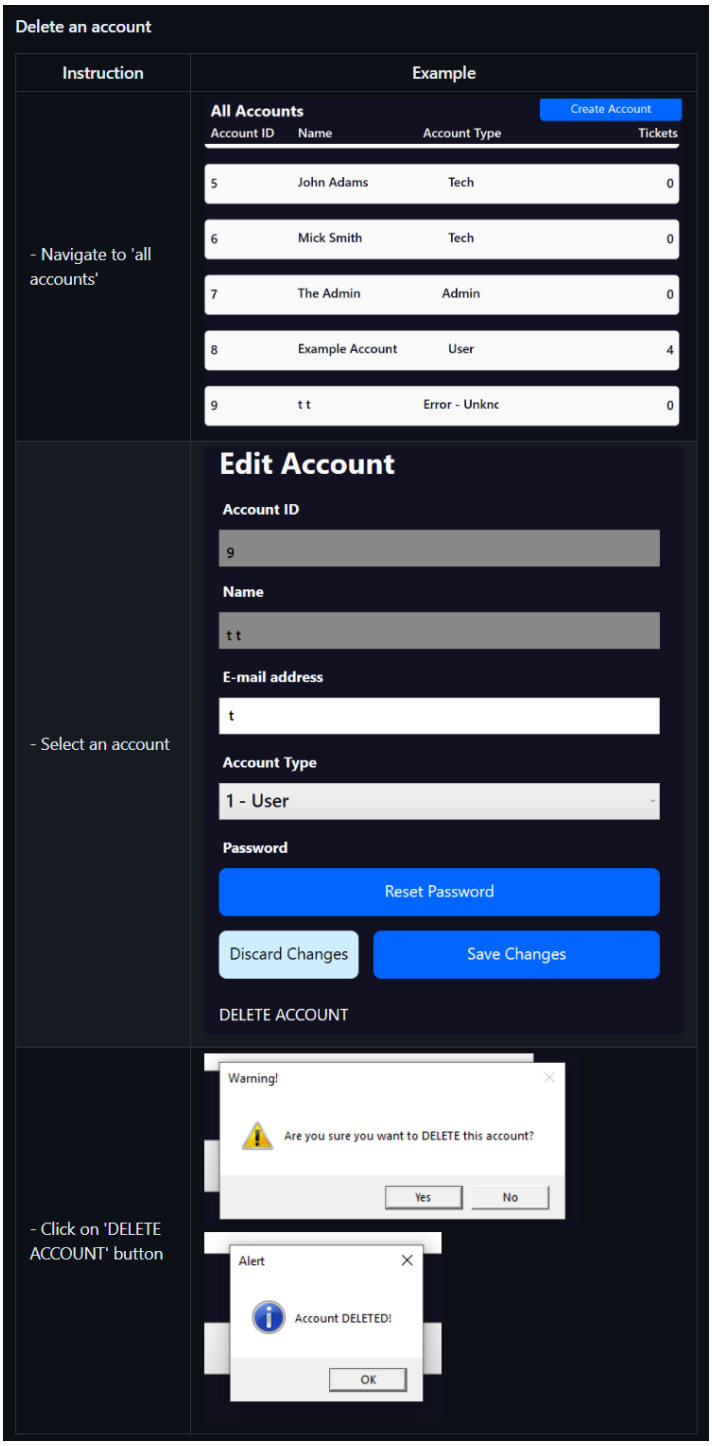



*Figure 29: User Guide #4*

## Common Feature to all Users

### Change password of current account

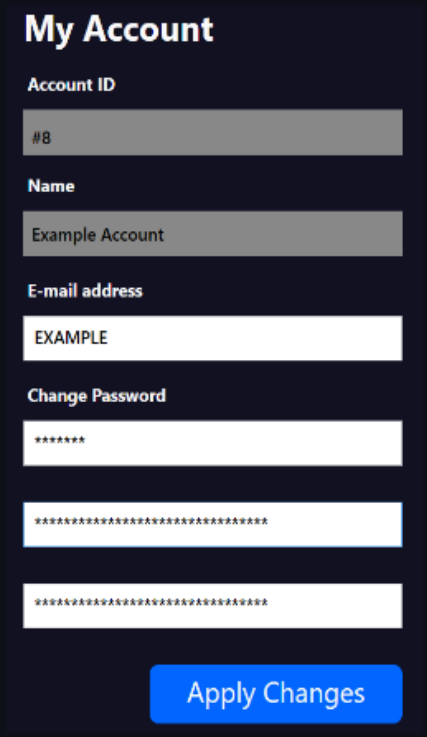
| Instruction | Example |
| --- | --- |
| - Navigate to 'My Account'<br>- Enter old password<br>- Enter new password and confirmation of new password<br>- Press 'Apply Changes' |  |



*Figure 30: User Guide #5*

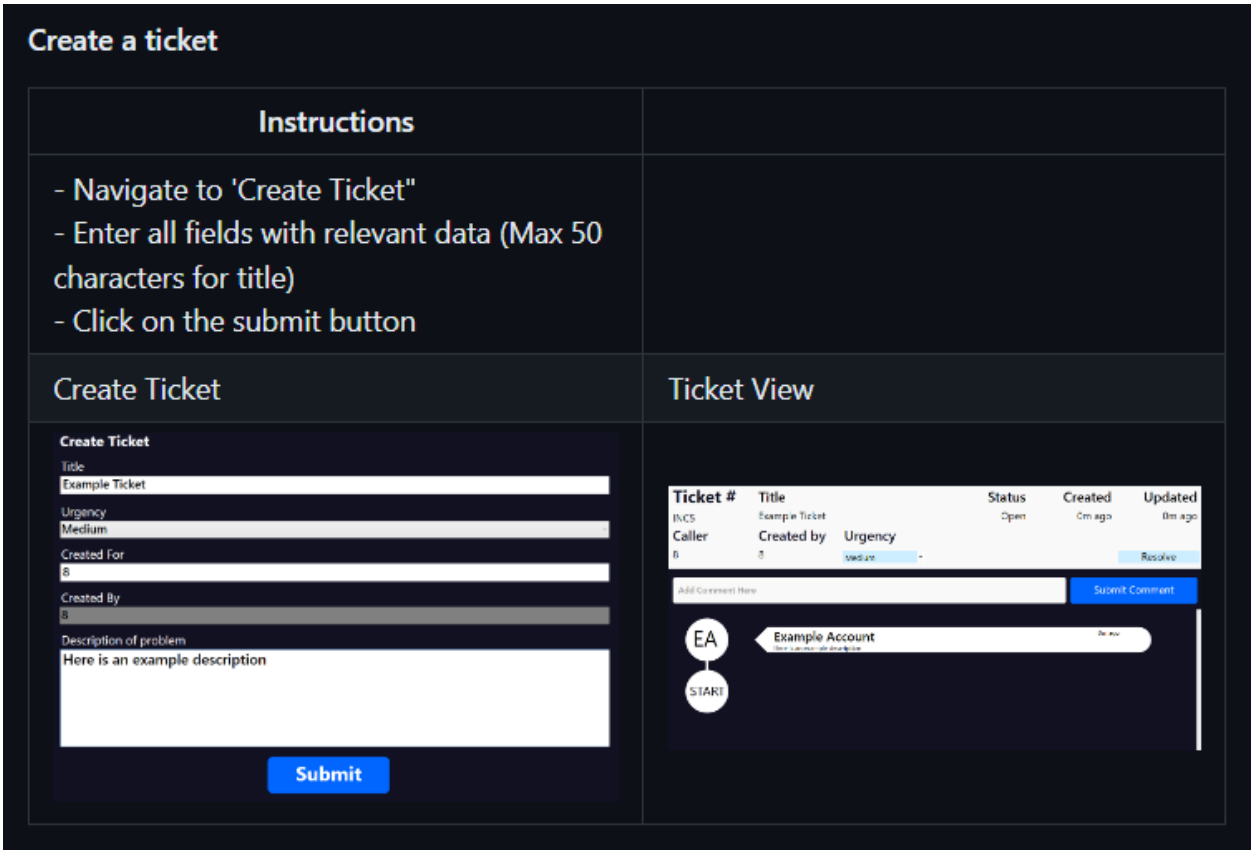
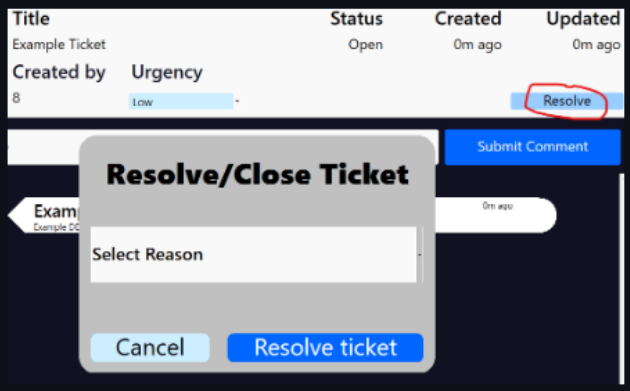



*Figure 31: User Guide #6*

**Add comment to ticket**

| Instruction | Example |
|---|---|
| - Open or create a ticket<br>- Type in the comment box below the ticket info and above the current comments<br>- Click on the 'submit comment' button to add a comment |  |

**Resolve a ticket**

| Instruction | Example |
|---|---|
| - Open or create a ticket<br>- Click on the 'resolve' button<br>- A window will pop up to select a reason<br>- Select a reason to resolve/close the ticket<br>- Click the 'resolve ticket' button on the pop up window |  |

**Creating a ticket on behalve of another user (i.e. a caller with a problem)**

| Instruction | Example |
|---|---|
| - When creating a ticket there is a greyed out 'created by' field and a 'created for' field which you can change.<br>- By default they are the same, however you can change who the ticket is created for in cases where tickets are created on behalf of somebody else<br>- To create the ticket on behalf of another user, simply change the created for input as THEIR account ID instead of yours.<br>- When you do this you will still have access as you created the ticket (This field cannot be changed) |  |

*Figure 32: User Guide #7*