2207-BSE

Implementation – Evidence Document

CS106.2

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

Document Outline

Table of Contents

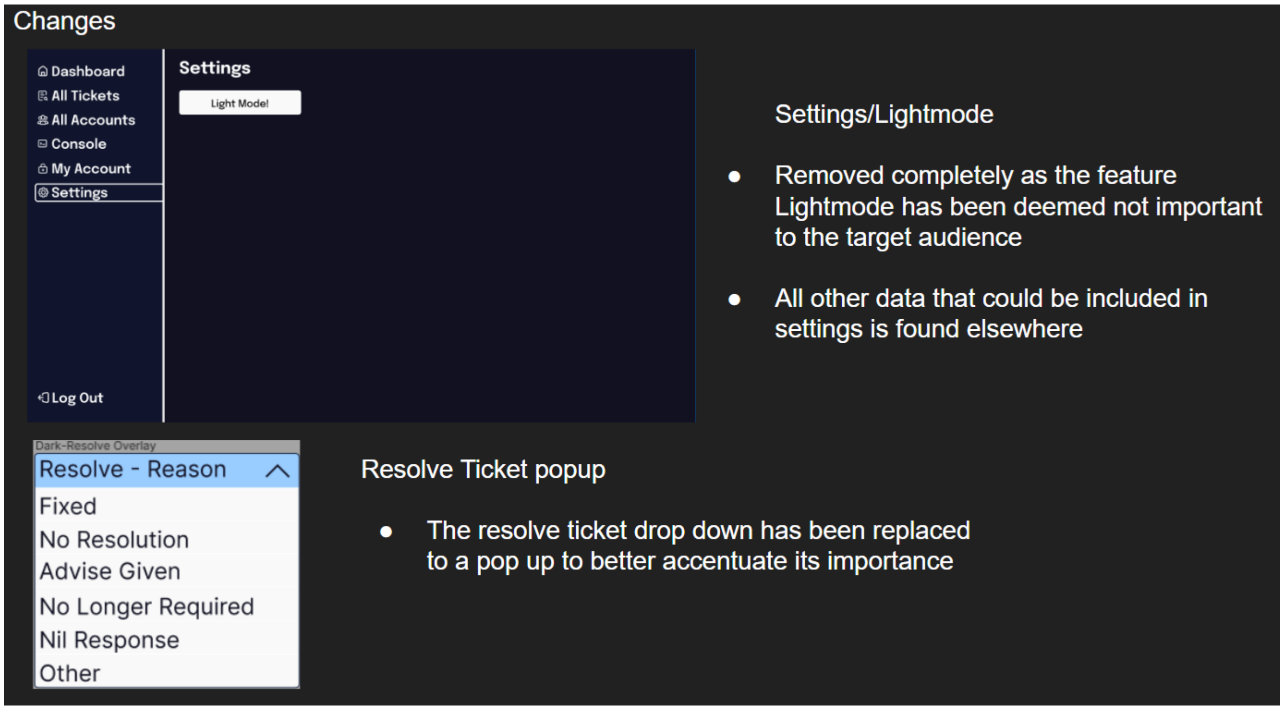
Table of Figures

Table of Tables

Intentionally Blank

Changes

HiFi



Changes

Settings/Lightmode

- Removed completely as the feature Lightmode has been deemed not important to the target audience
- All other data that could be included in settings is found elsewhere

Resolve Ticket popup

- The resolve ticket drop down has been replaced to a pop up to better accentuate its importance

Figure 1: Hifi Changes

System Architecture

Activity Diagram

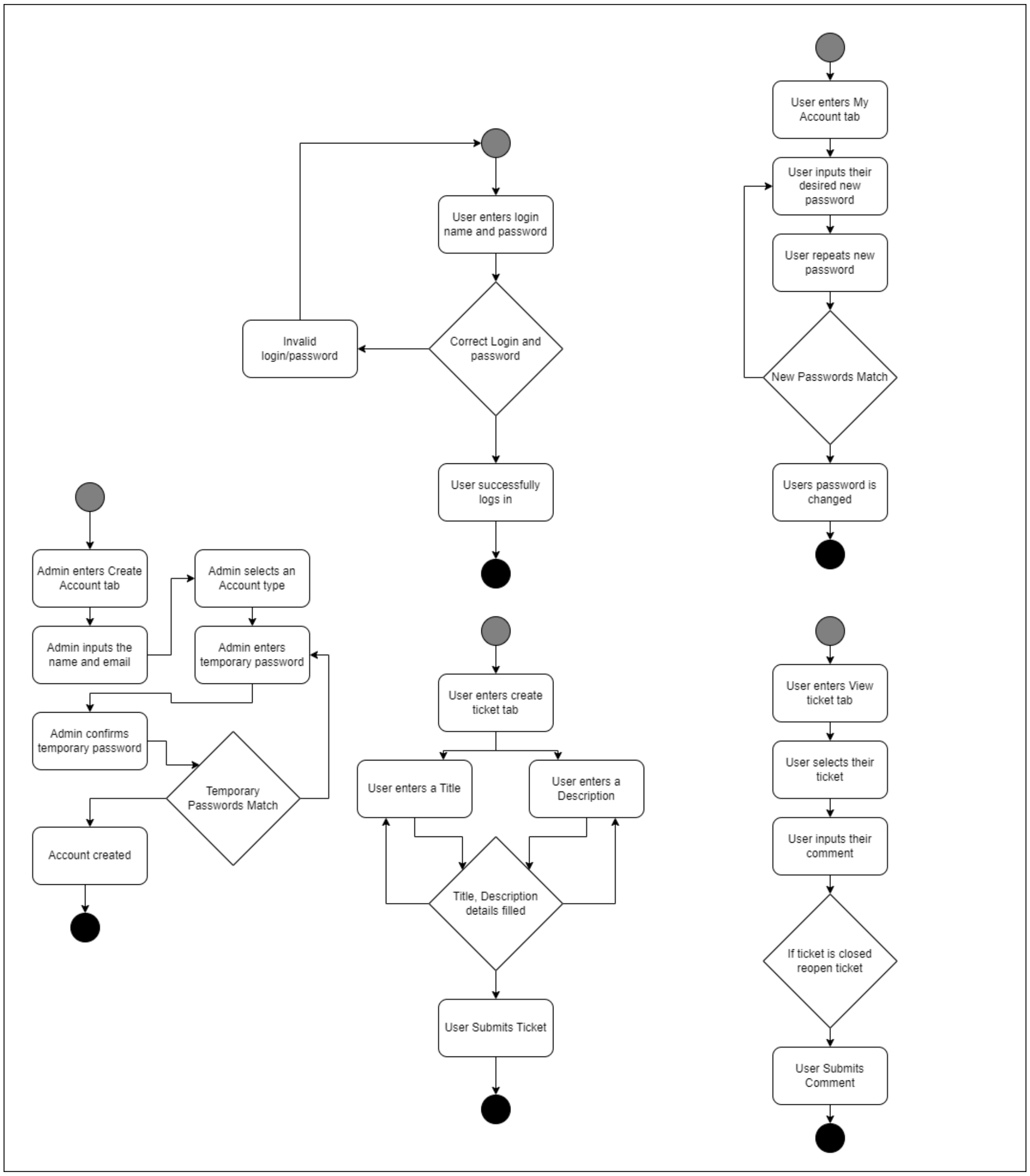

User enters login name and password
Correct Login and password
Invalid login/password
User successfully logs in
User enters My Account tab
User inputs their desired new password
User repeats new password
New Passwords Match
Users password is changed
Admin enters Create Account tab
Admin inputs the name and email
Admin selects an Account type
Admin enters temporary password
Admin confirms temporary password
Temporary Passwords Match
Account created
User enters create ticket tab
User enters a Title
User enters a Description
Title, Description details filled
User Submits Ticket
User enters View ticket tab
User selects their ticket
User inputs their comment
If ticket is closed reopen ticket
User Submits Comment

Figure 2: Activity Diagram

Class Diagram

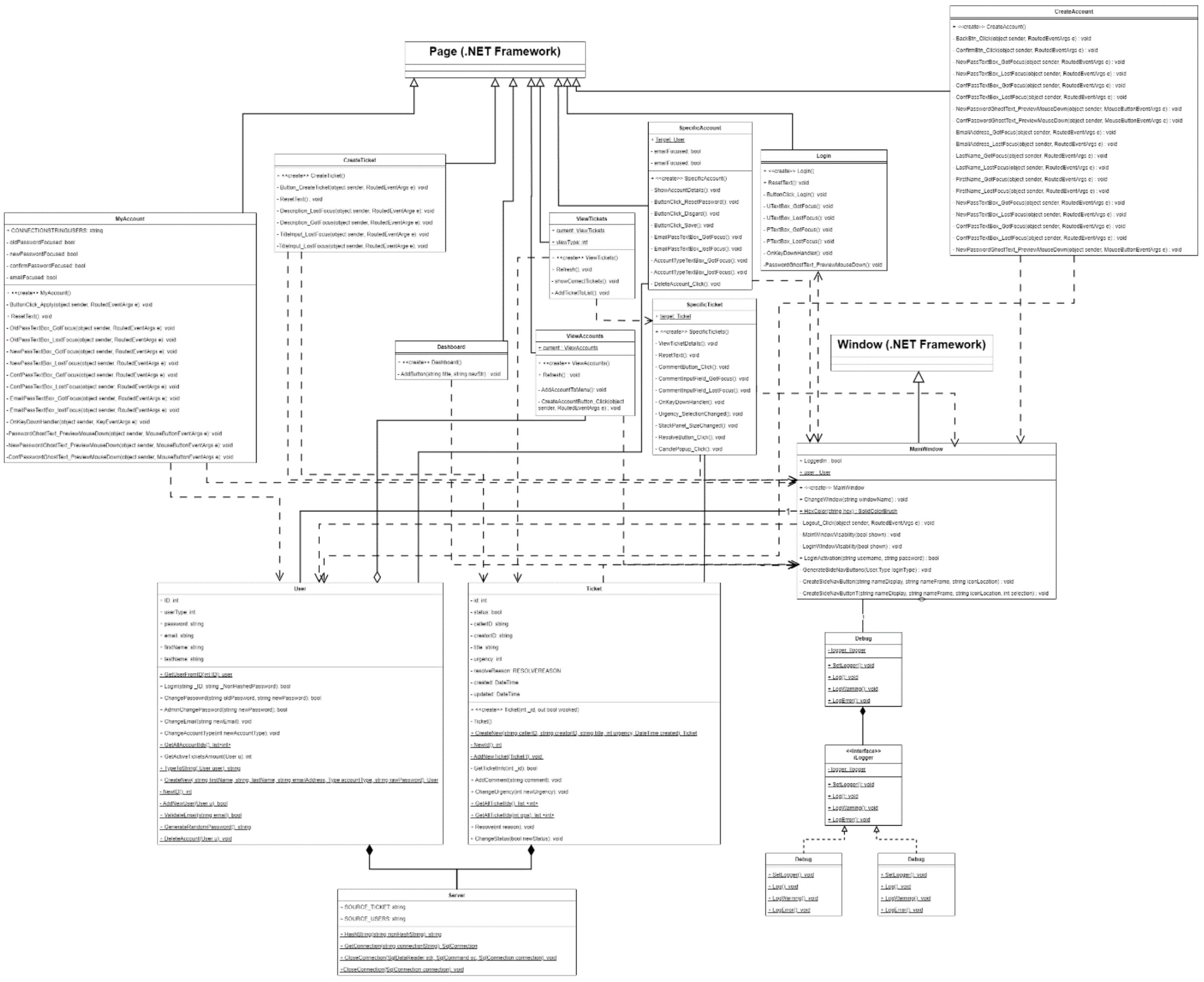


Figure 3: Class Diagram

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.

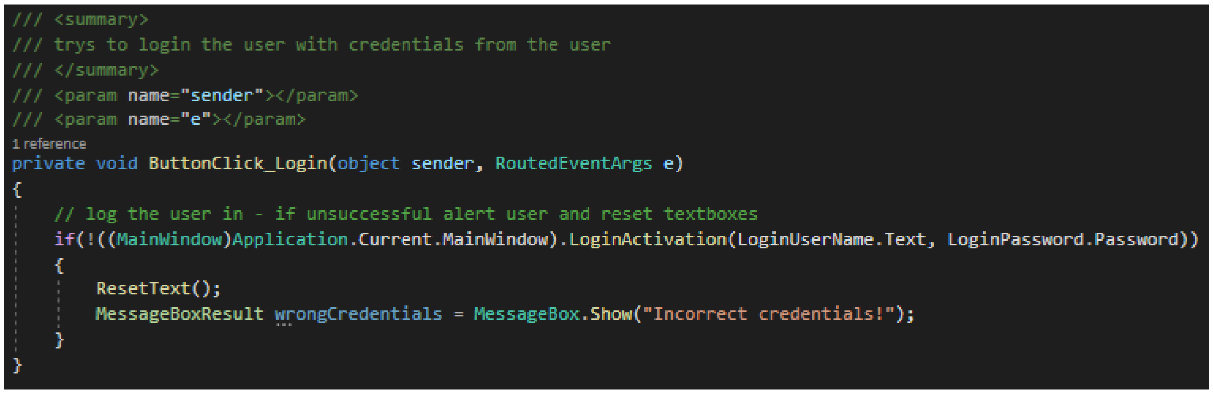
Key Functionality

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:

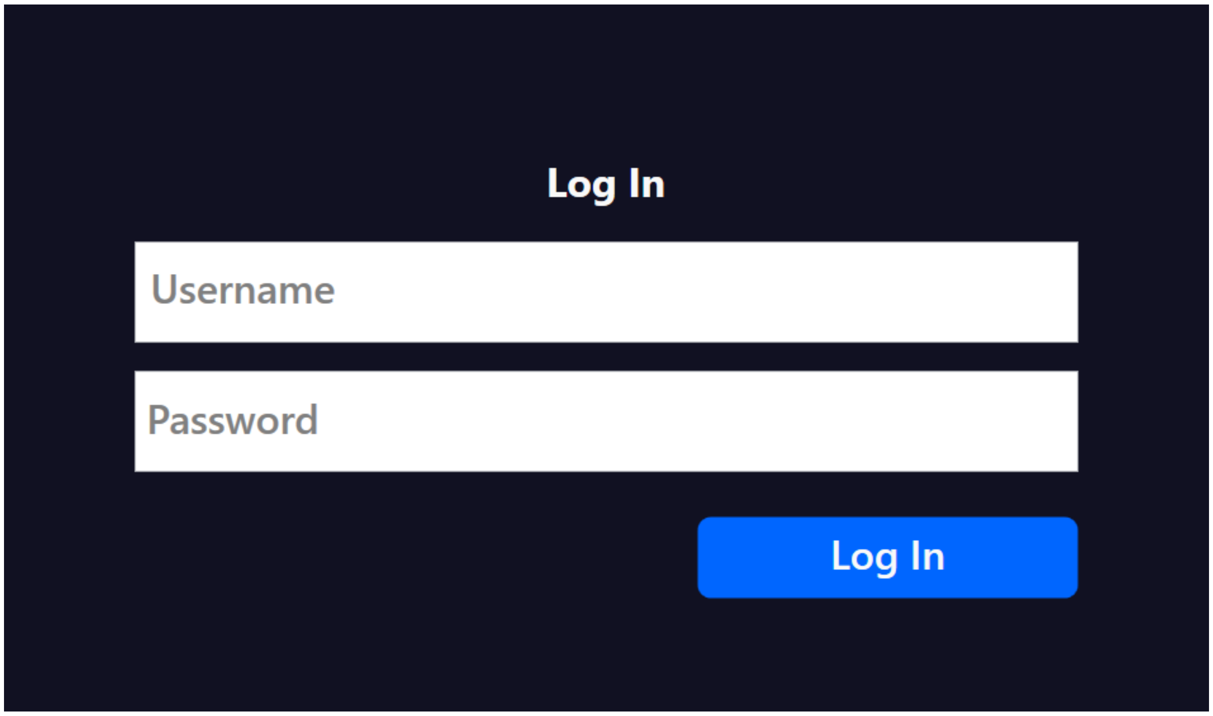
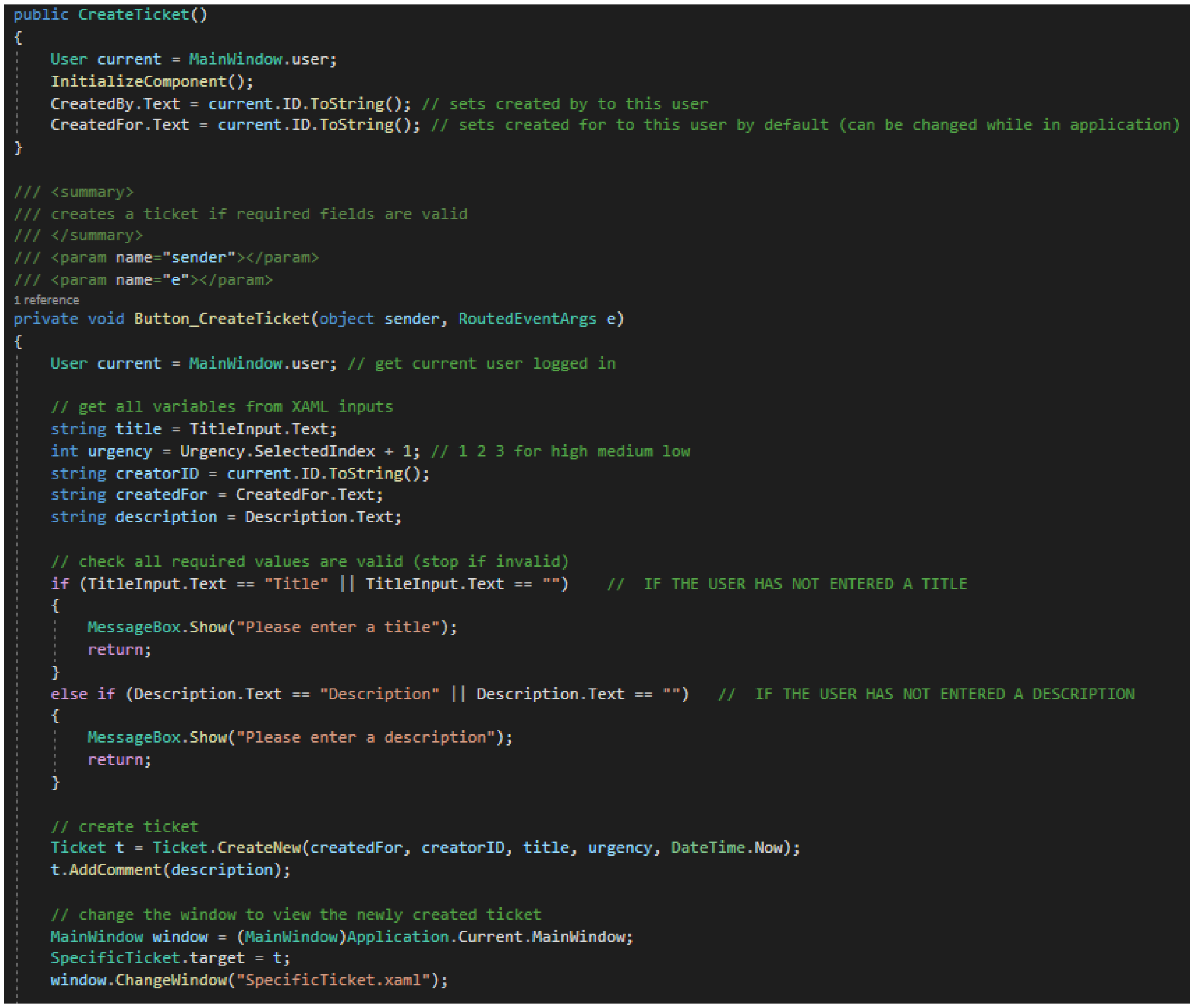


Figure 5: Key Functionality #2

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.

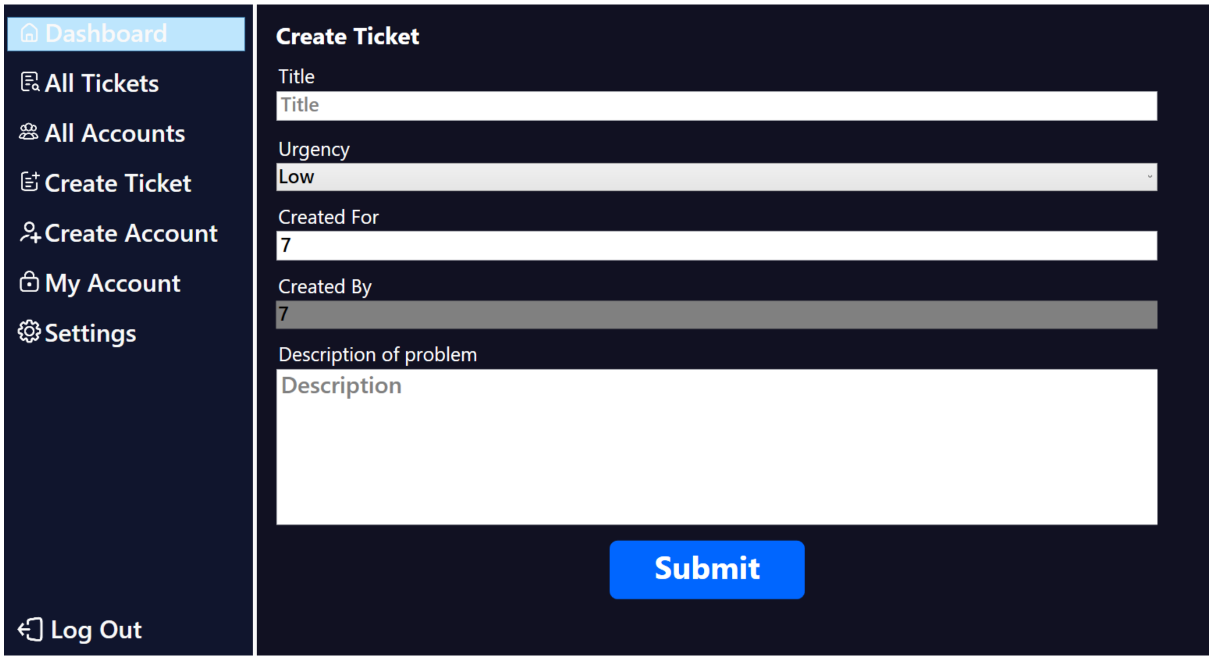
C#:



```
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:



Create Ticket

Title: Title

Urgency: Low

Created For: 7

Created By: 7

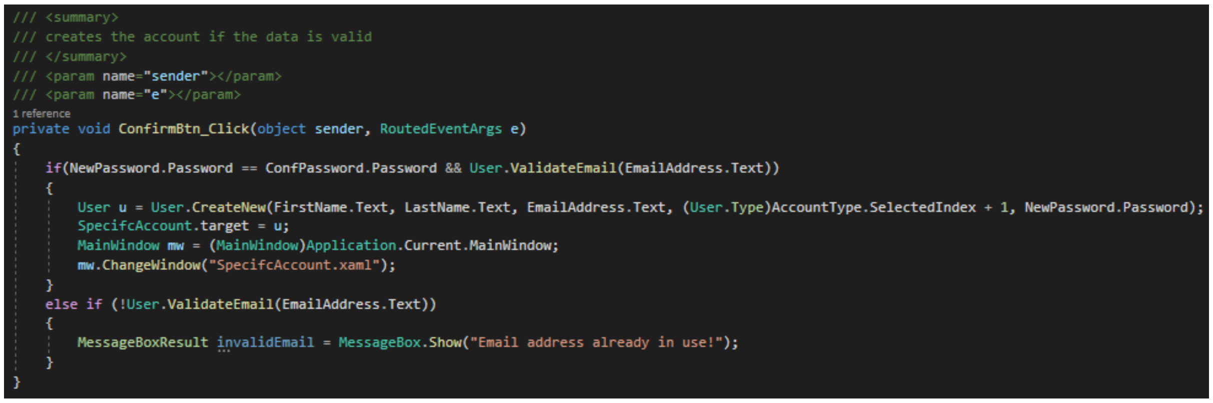
Description of problem: Description

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.

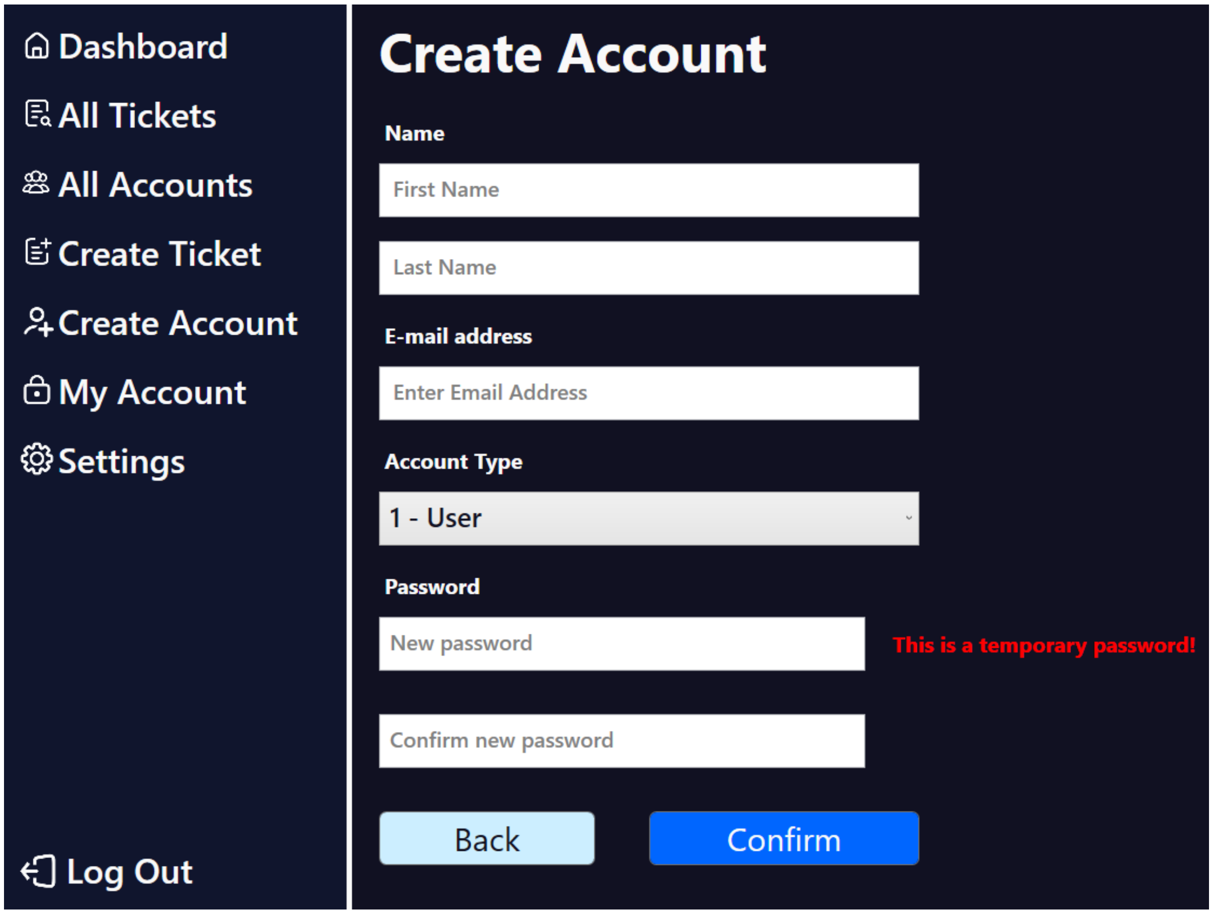
C#:



```
/// <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:



Create Account

Name

First Name

Last Name

E-mail address

Enter Email Address

Account Type

1 - User

Password

New password

This is a temporary password!

Confirm new password

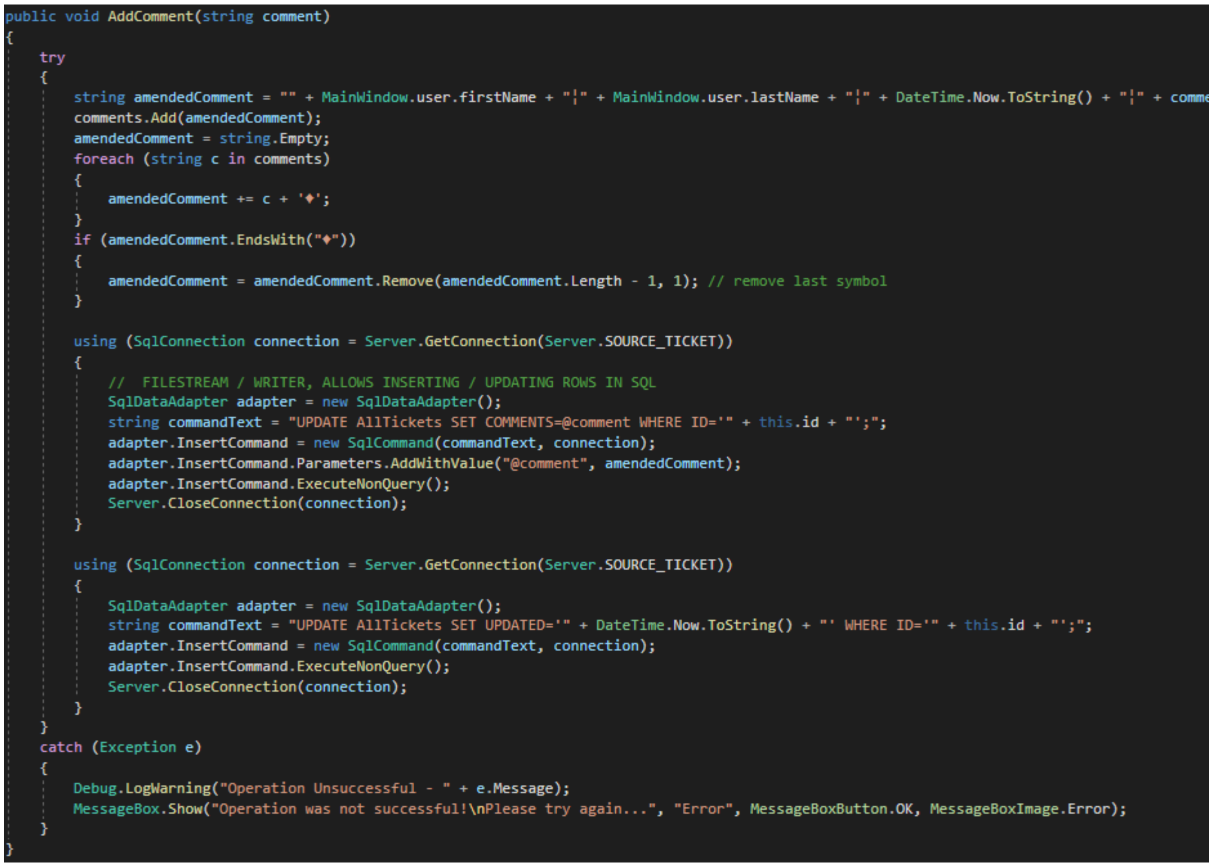
Figure 9: Key Functionality #6

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#:

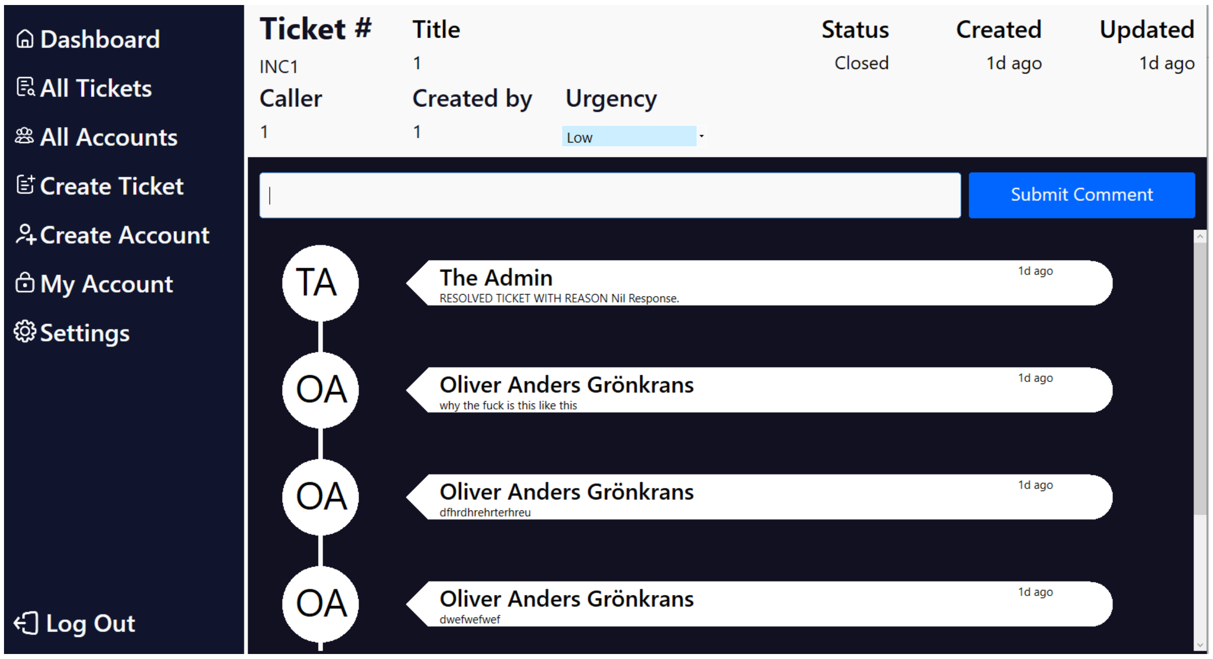
Implementation



```
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:



Ticket #	Title	Status	Created	Updated
INC1	1	Closed	1d ago	1d ago

Caller	Created by	Urgency
1	1	Low

TA — The Admin — 1d ago
RESOLVED TICKET WITH REASON Nil Response.

OA — Oliver Anders Grönkrans — 1d ago
why the fuck is this like this

OA — Oliver Anders Grönkrans — 1d ago
dfhrdhrehrterhreu

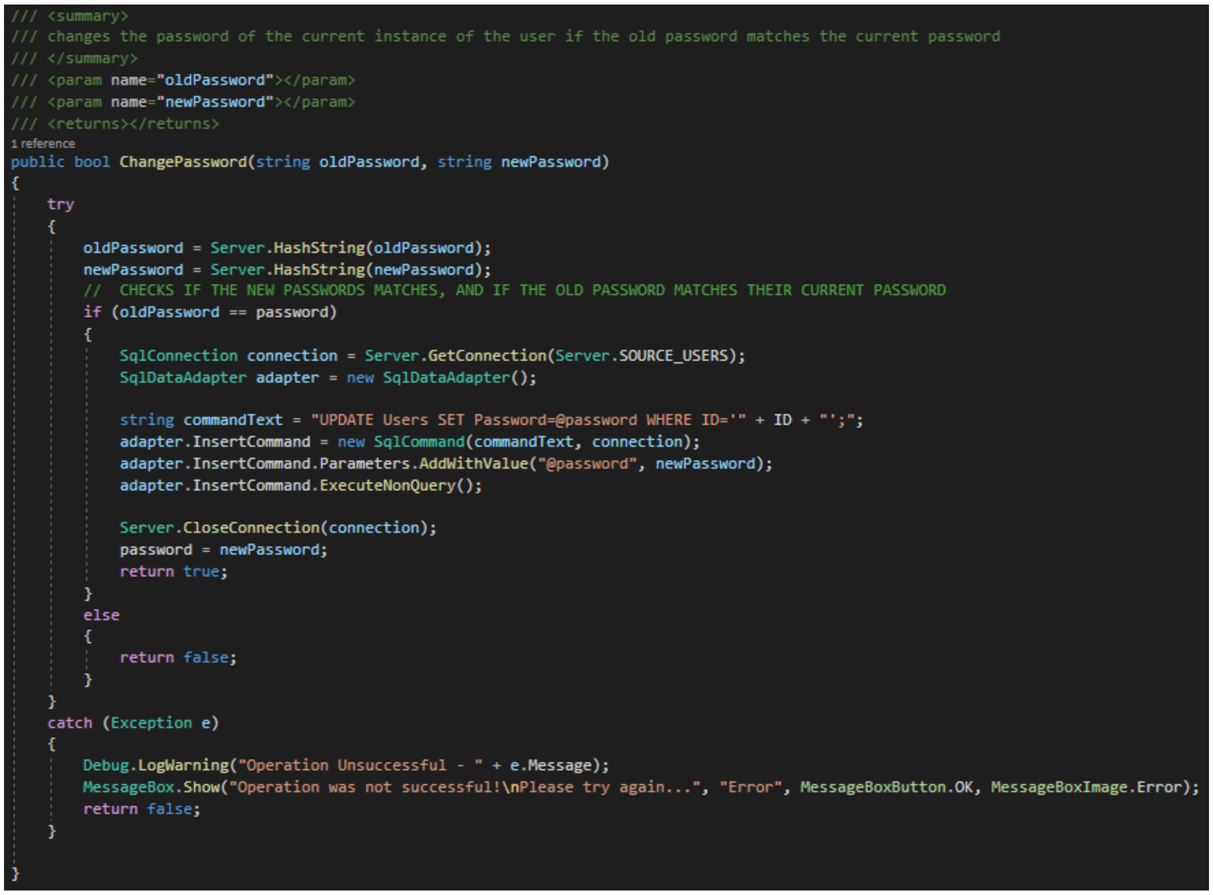
OA — Oliver Anders Grönkrans — 1d ago
dwefwefwef

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”).

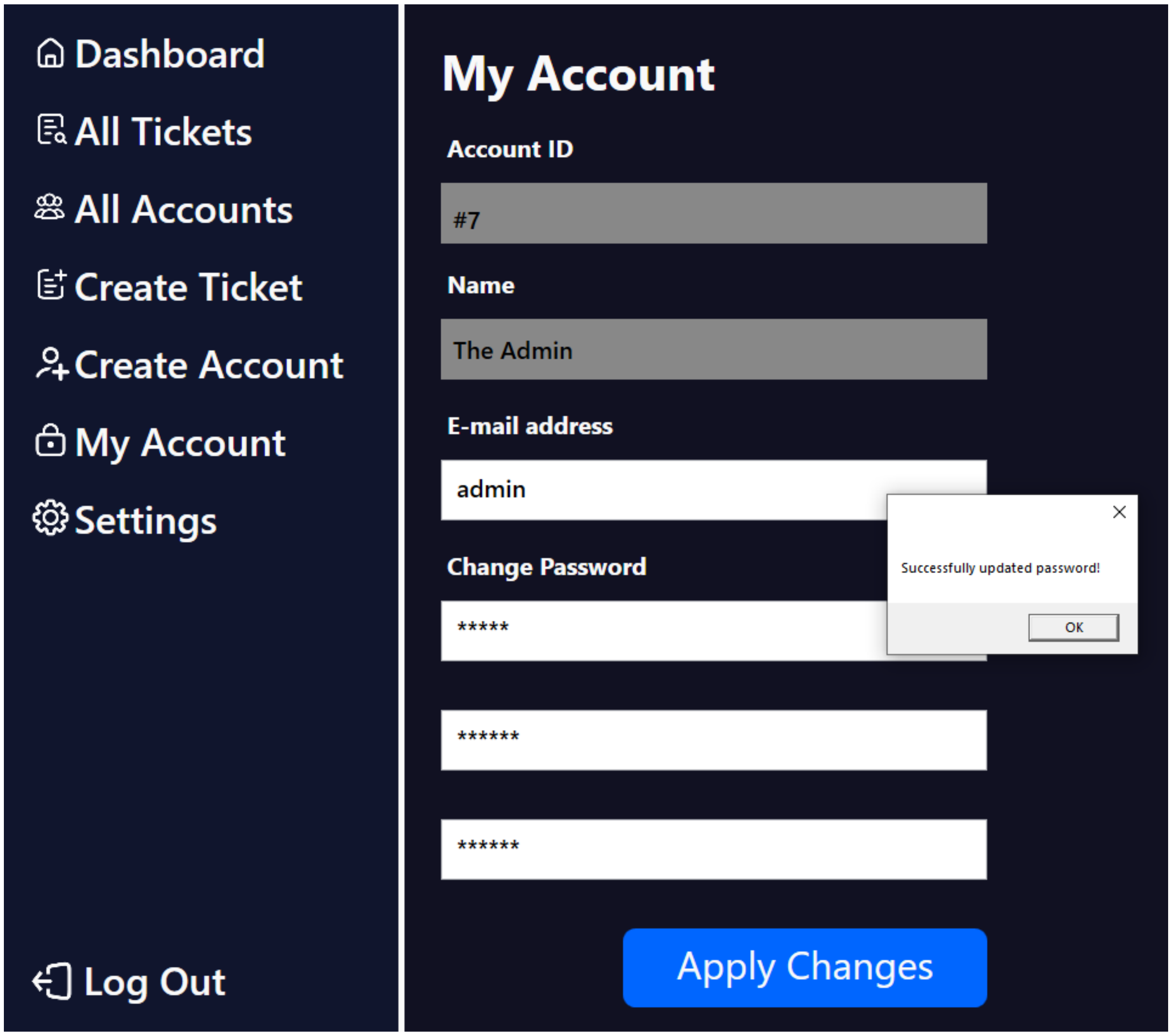
C#:



```
/// <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:



My Account

Account ID: #7

Name: The Admin

E-mail address: admin

Change Password: *****

******

******

Successfully updated password!

Figure 13: Key Functionality #10

Function Testing

Black Box Testing

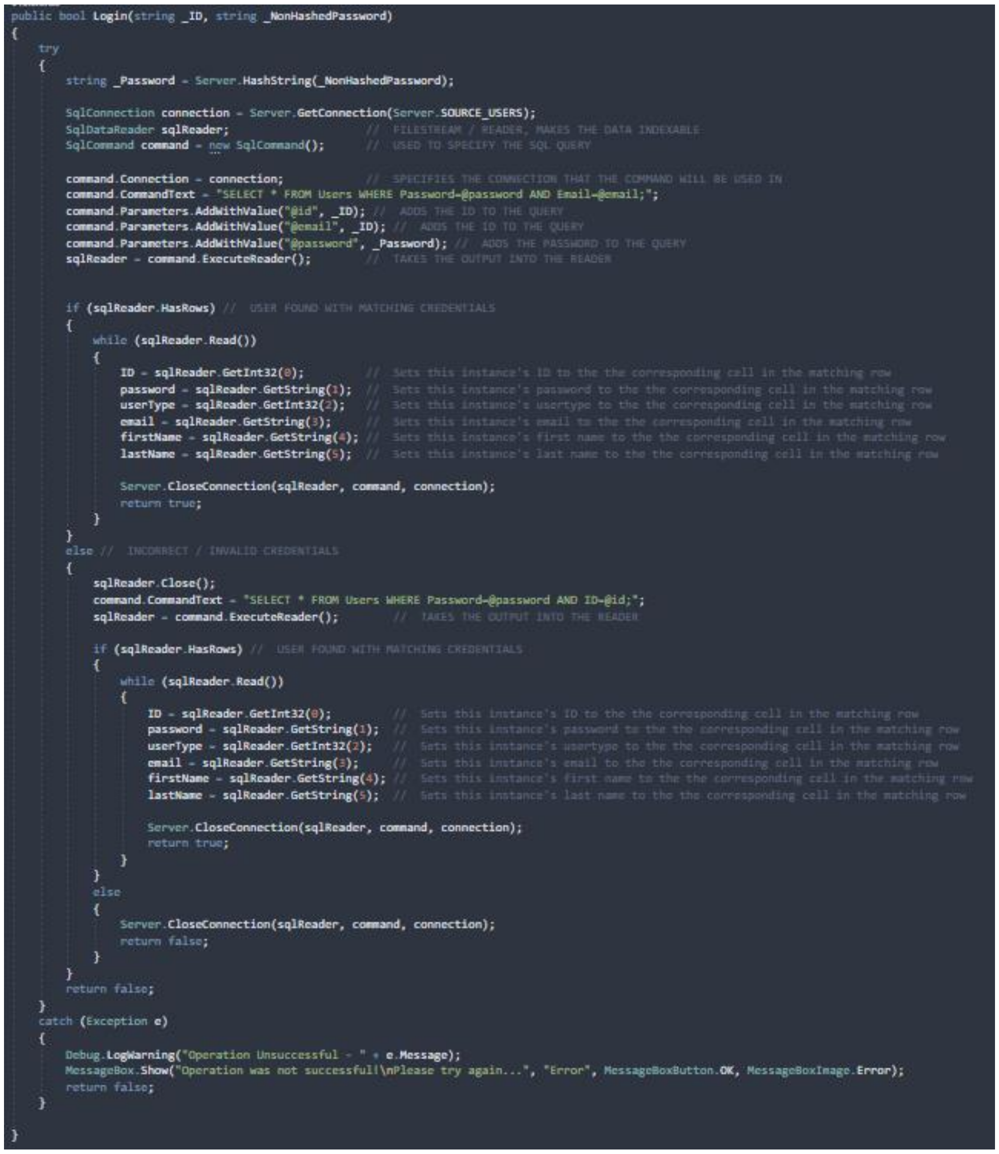
Test cases

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

Expected output

Login attempt fails as with any other invalid credentials.

Code snippet



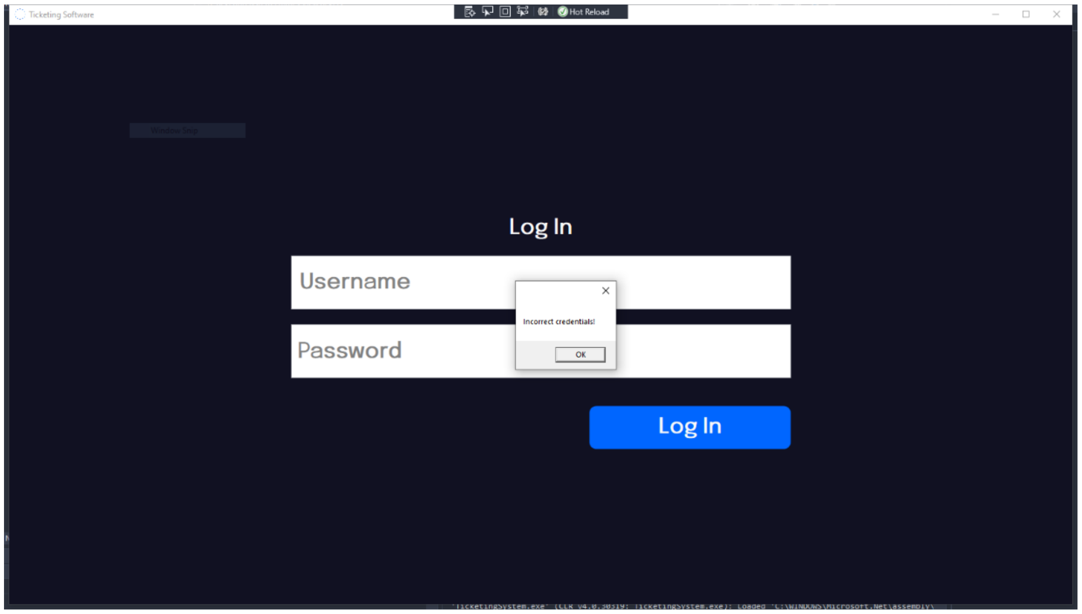
```
public bool Login(string _ID, string _NonHashedPassword)
{
    try
    {
        string _Password = Server.HashString(_NonHashedPassword);

        SqlConnection connection = Server.GetConnection(Server.SOURCE_USERS);
        SqlDataReader sqlReader;                    //  FILESTREAM / READER, MAKES THE DATA INDEXABLE
        SqlCommand command = new SqlCommand();      //  USED TO SPECIFY THE SQL QUERY

        command.Connection = connection;            //  SPECIFIES THE CONNECTION THAT THE COMMAND WILL BE USED IN
        command.CommandText = "SELECT * FROM Users WHERE Password=@password AND Email=@email;";
        command.Parameters.AddWithValue("@id", _ID); //  ADDS THE ID TO THE QUERY
        command.Parameters.AddWithValue("@email", _ID); //  ADDS THE ID TO THE QUERY
        command.Parameters.AddWithValue("@password", _Password); //  ADDS THE PASSWORD TO THE QUERY
        sqlReader = command.ExecuteReader();        //  TAKES THE OUTPUT INTO THE READER


        if (sqlReader.HasRows) //  USER FOUND WITH MATCHING CREDENTIALS
        {
            while (sqlReader.Read())
            {
                ID = sqlReader.GetInt32(0);         //  Sets this instance's ID to the the corresponding cell in the matching row
                password = sqlReader.GetString(1);  //  Sets this instance's password to the the corresponding cell in the matching row
                userType = sqlReader.GetInt32(2);   //  Sets this instance's usertype to the the corresponding cell in the matching row
                email = sqlReader.GetString(3);     //  Sets this instance's email to the the corresponding cell in the matching row
                firstName = sqlReader.GetString(4); //  Sets this instance's first name to the the corresponding cell in the matching row
                lastName = sqlReader.GetString(5);  //  Sets this instance's last name to the the corresponding cell in the matching row

                Server.CloseConnection(sqlReader, command, connection);
                return true;
            }
        }
        else //  INCORRECT / INVALID CREDENTIALS
        {
            sqlReader.Close();
            command.CommandText = "SELECT * FROM Users WHERE Password=@password AND ID=@id;";
            sqlReader = command.ExecuteReader();        //  TAKES THE OUTPUT INTO THE READER

            if (sqlReader.HasRows) //  USER FOUND WITH MATCHING CREDENTIALS
            {
                while (sqlReader.Read())
                {
                    ID = sqlReader.GetInt32(0);         //  Sets this instance's ID to the the corresponding cell in the matching row
                    password = sqlReader.GetString(1);  //  Sets this instance's password to the the corresponding cell in the matching row
                    userType = sqlReader.GetInt32(2);   //  Sets this instance's usertype to the the corresponding cell in the matching row
                    email = sqlReader.GetString(3);     //  Sets this instance's email to the the corresponding cell in the matching row
                    firstName = sqlReader.GetString(4); //  Sets this instance's first name to the the corresponding cell in the matching row
                    lastName = sqlReader.GetString(5);  //  Sets this instance's last name to the the corresponding cell in the matching row

                    Server.CloseConnection(sqlReader, command, connection);
                    return true;
                }
            }
            else
            {
                Server.CloseConnection(sqlReader, command, connection);
                return false;
            }
        }
        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 14: Login function in the user class, which is called when logging in

Used parameter

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

Received output



Incorrect credentials!

Figure 15: Functional Testing #1

Result

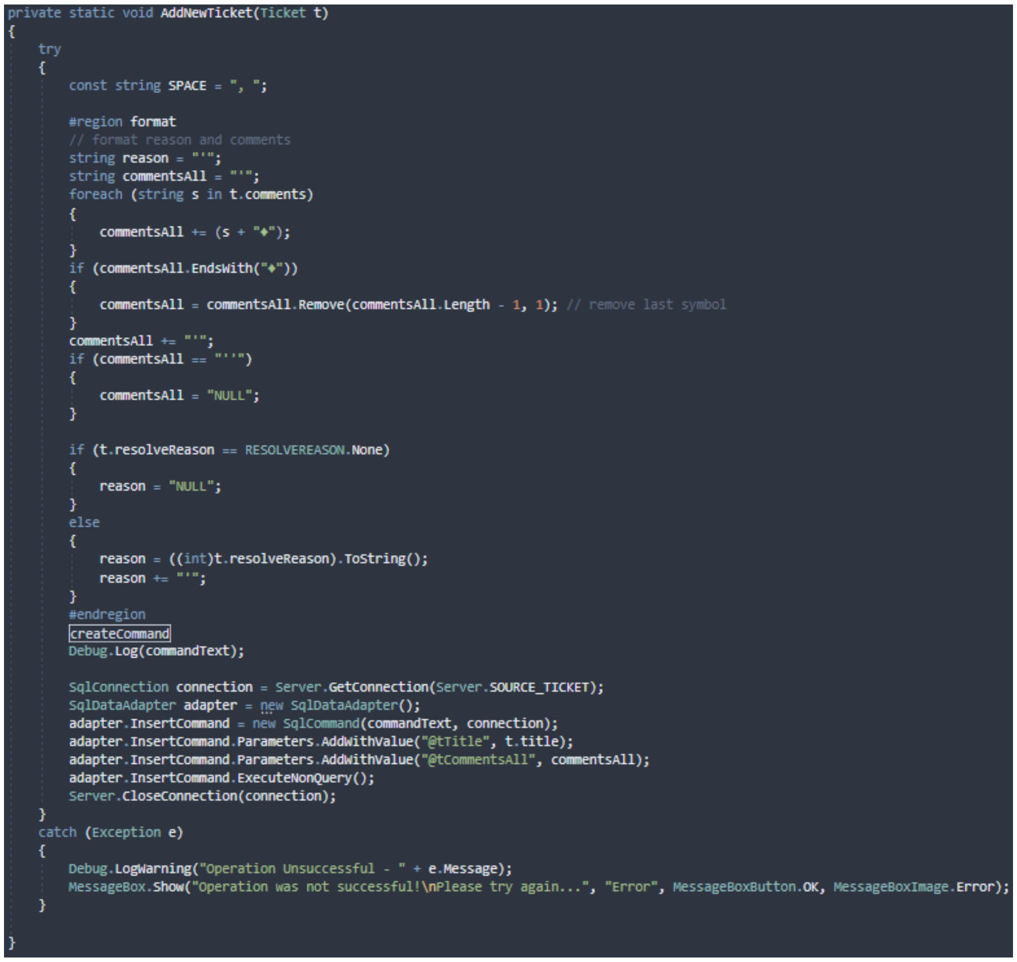
Passed

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.

Code snippet



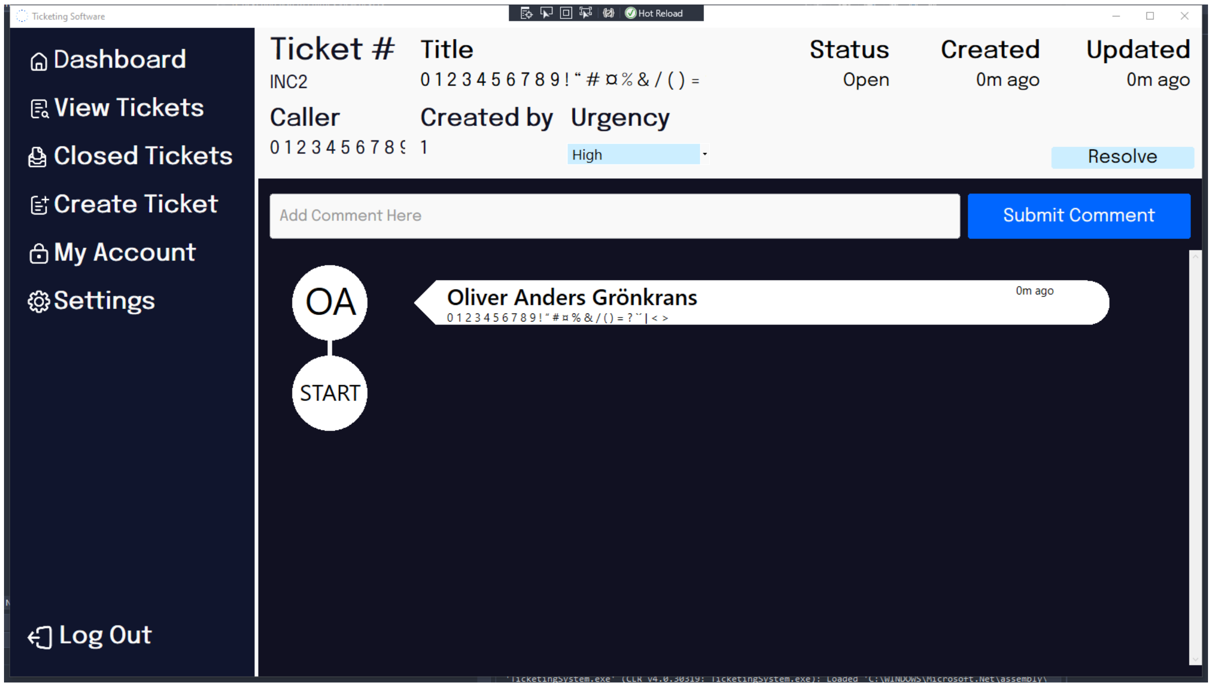
```
private static void AddNewTicket(Ticket t)
{
    try
    {
        const string SPACE = ", ";

        #region format
        // format reason and comments
        string reason = "'";
        string commentsAll = "'";
        foreach (string s in t.comments)
        {
            commentsAll += (s + "♦");
        }
        if (commentsAll.EndsWith("♦"))
        {
            commentsAll = commentsAll.Remove(commentsAll.Length - 1, 1); // remove last symbol
        }
        commentsAll += "'";
        if (commentsAll == "''")
        {
            commentsAll = "NULL";
        }

        if (t.resolveReason == RESOLVEREASON.None)
        {
            reason = "NULL";
        }
        else
        {
            reason = ((int)t.resolveReason).ToString();
            reason += "'";
        }
        #endregion
        createCommand
        Debug.Log(commandText);

        SqlConnection connection = Server.GetConnection(Server.SOURCE_TICKET);
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.InsertCommand = new SqlCommand(commandText, connection);
        adapter.InsertCommand.Parameters.AddWithValue("@tTitle", t.title);
        adapter.InsertCommand.Parameters.AddWithValue("@tCommentsAll", commentsAll);
        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 16: AddNewTicket function, used to push a ticket to the ticket database

Used parameters

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

Received output



Ticket # INC2
Title 0 1 2 3 4 5 6 7 8 9 ! " # ¤ % & / ( ) =
Status Open
Created 0m ago
Updated 0m ago
Caller 0 1 2 3 4 5 6 7 8 9
Created by 1
Urgency High

Oliver Anders Grönkrans 0m ago
0 1 2 3 4 5 6 7 8 9 ! " # ¤ % & / ( ) = ? ¨ | < >

Figure 17: Functional Testing #2

Result

Passed

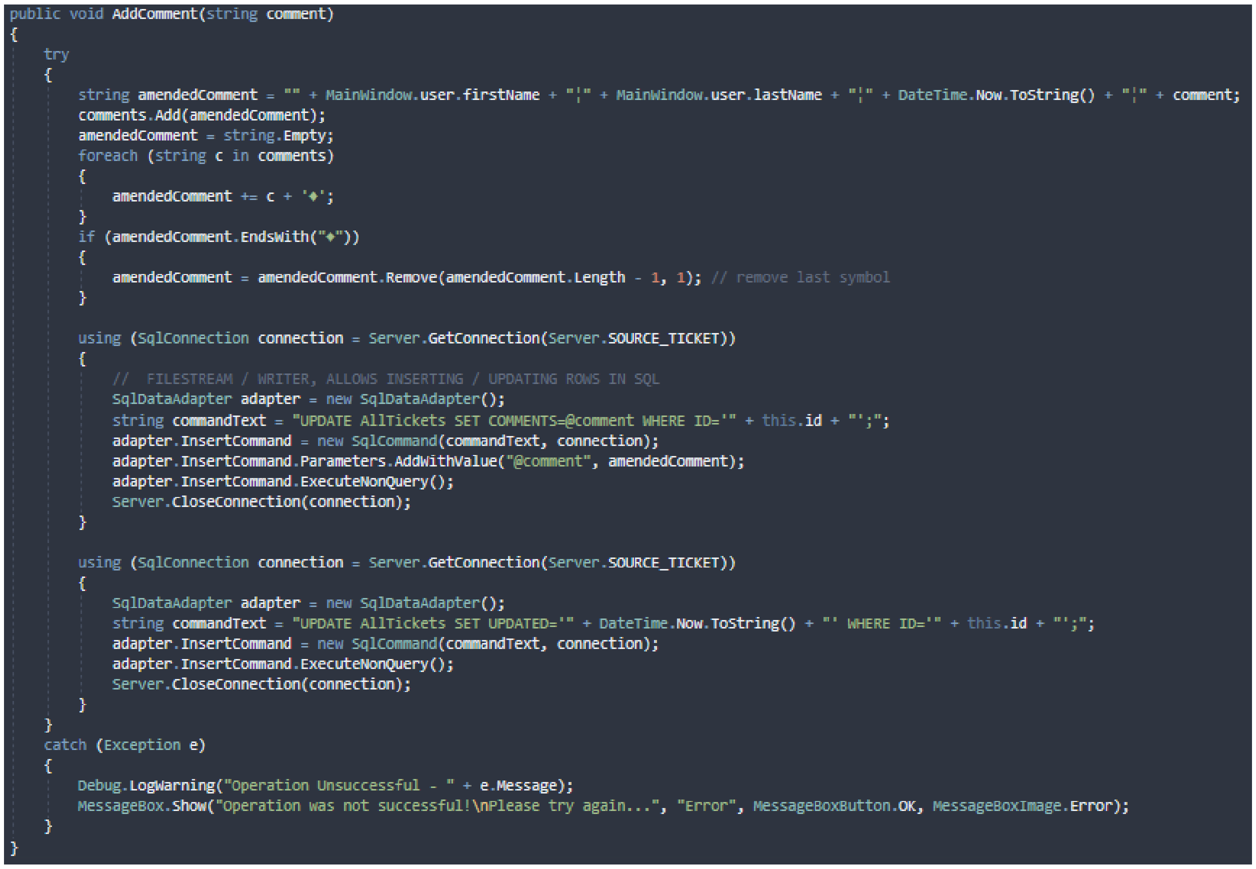
3. Resolve ticket

Expected output

Changes documented in comment field without issue.

Code snippet

Implementation



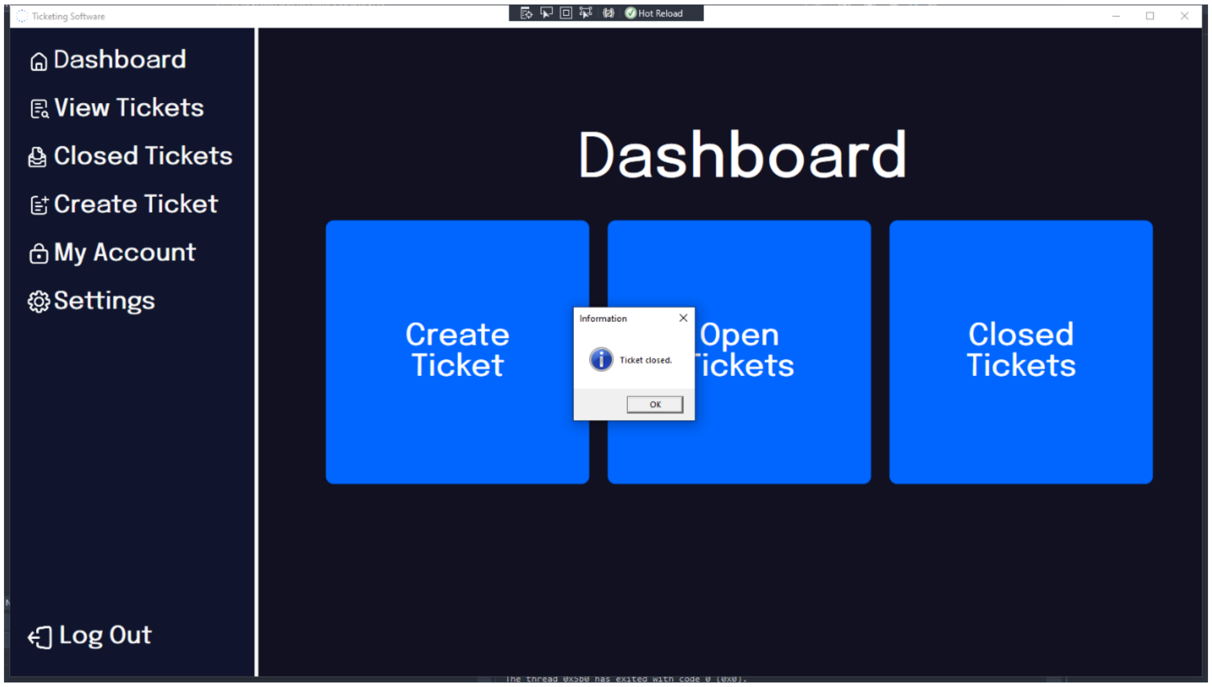
```
public void AddComment(string comment)
{
    try
    {
        string amendedComment = "" + MainWindow.user.firstName + "¦" + MainWindow.user.lastName + "¦" + DateTime.Now.ToString() + "¦" + comment;
        comments.Add(amendedComment);
        amendedComment = string.Empty;
        foreach (string 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 18: AddComment function in ticket, used to push new comments

Used parameters

Resolve status - Fixed

Received output



Ticket closed.

Figure 19: Functional Testing #3

Result

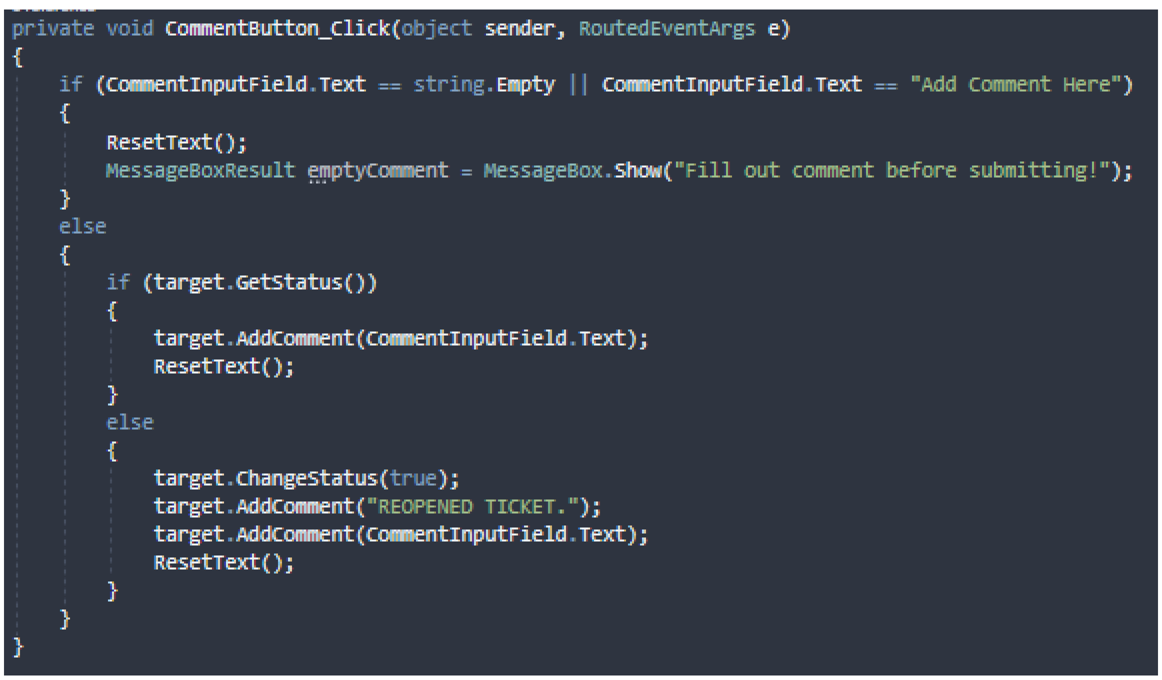
Passed

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.

Code snippet



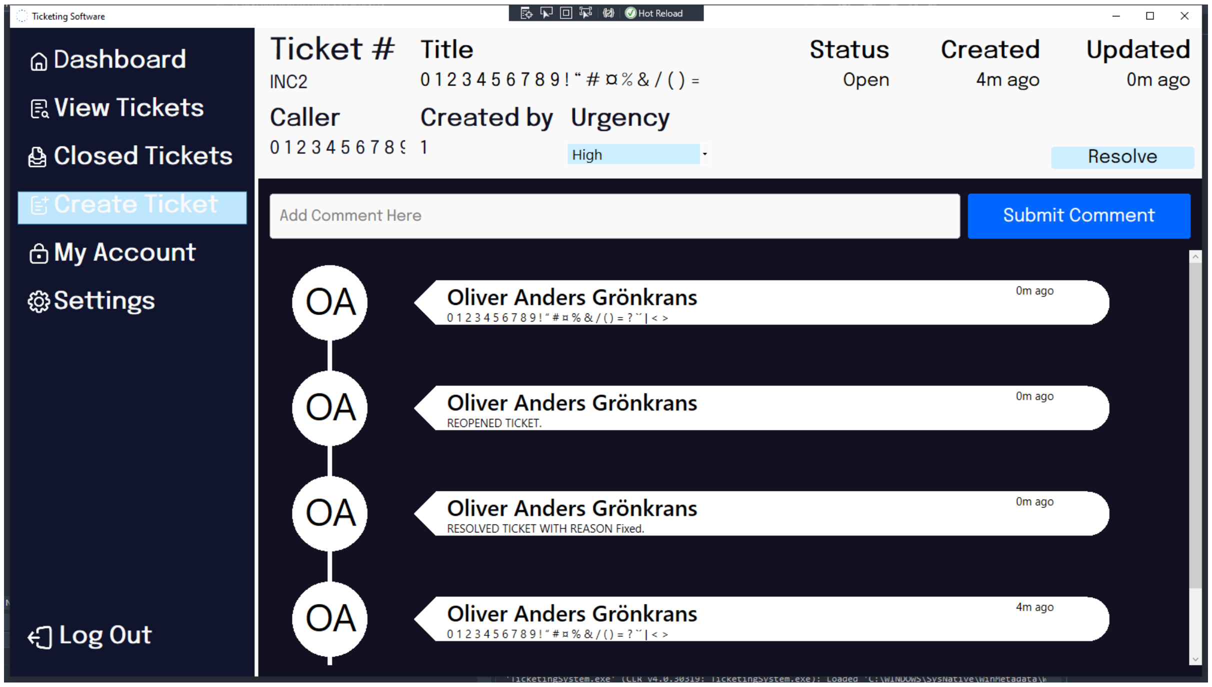
```
private void CommentButton_Click(object sender, RoutedEventArgs e)
{
    if (CommentInputField.Text == string.Empty || CommentInputField.Text == "Add Comment Here")
    {
        ResetText();
        MessageBoxResult emptyComment = MessageBox.Show("Fill out comment before submitting!");
    }
    else
    {
        if (target.GetStatus())
        {
            target.AddComment(CommentInputField.Text);
            ResetText();
        }
        else
        {
            target.ChangeStatus(true);
            target.AddComment("REOPENED TICKET.");
            target.AddComment(CommentInputField.Text);
            ResetText();
        }
    }
}
```

Figure 20: EventHandler, called when pressing the submit comment button

Used parameters

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

Received output



Ticketing Software

Ticket # INC2 | Title 0 1 2 3 4 5 6 7 8 9 ! " # ¤ % & / ( ) = | Status Open | Created 4m ago | Updated 0m ago

Caller 0 1 2 3 4 5 6 7 8 9 | Created by 1 | Urgency High

Oliver Anders Grönkrans — 0m ago
0 1 2 3 4 5 6 7 8 9 ! " # ¤ % & / ( ) = ? ` | < >

Oliver Anders Grönkrans — 0m ago
REOPENED TICKET.

Oliver Anders Grönkrans — 0m ago
RESOLVED TICKET WITH REASON Fixed.

Oliver Anders Grönkrans — 4m ago
0 1 2 3 4 5 6 7 8 9 ! " # ¤ % & / ( ) = ? ` | < >

Figure 21: Functional Testing #4

Result

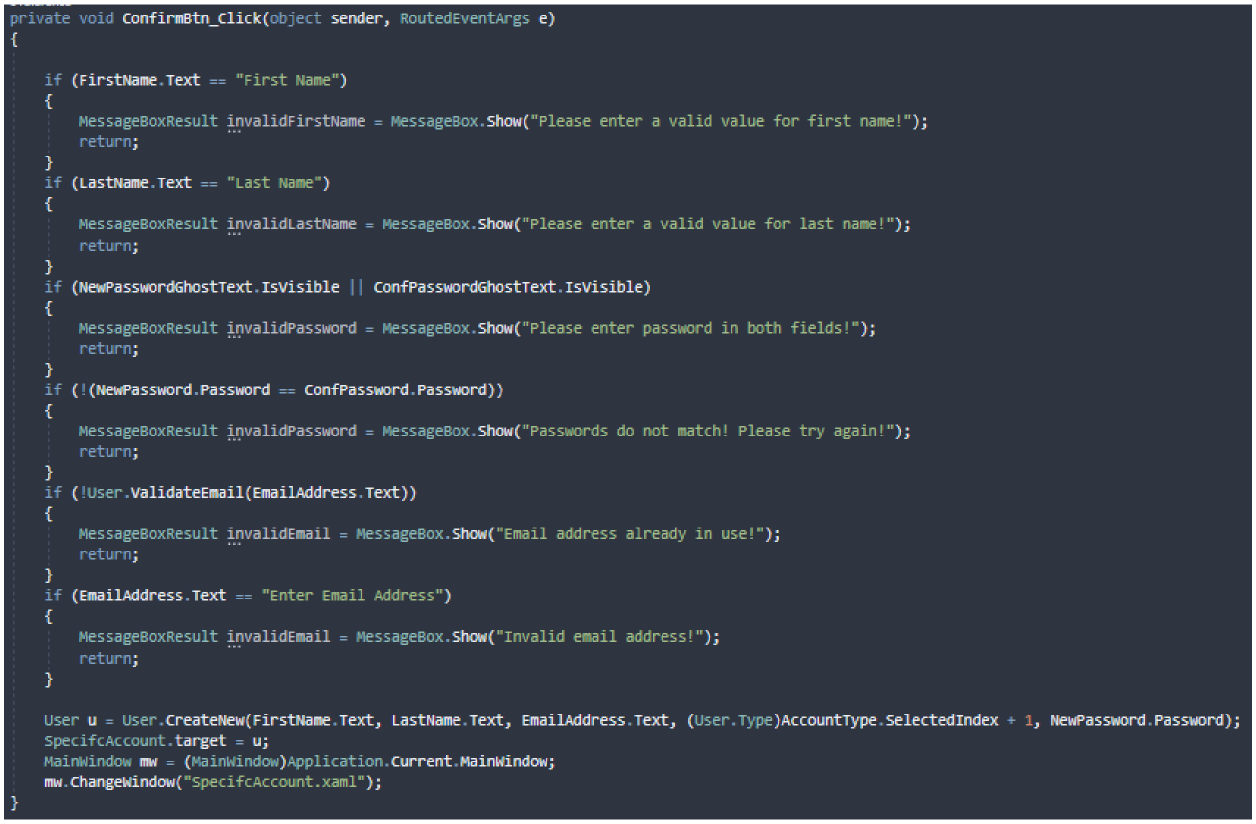
Passed

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.

Code snippet



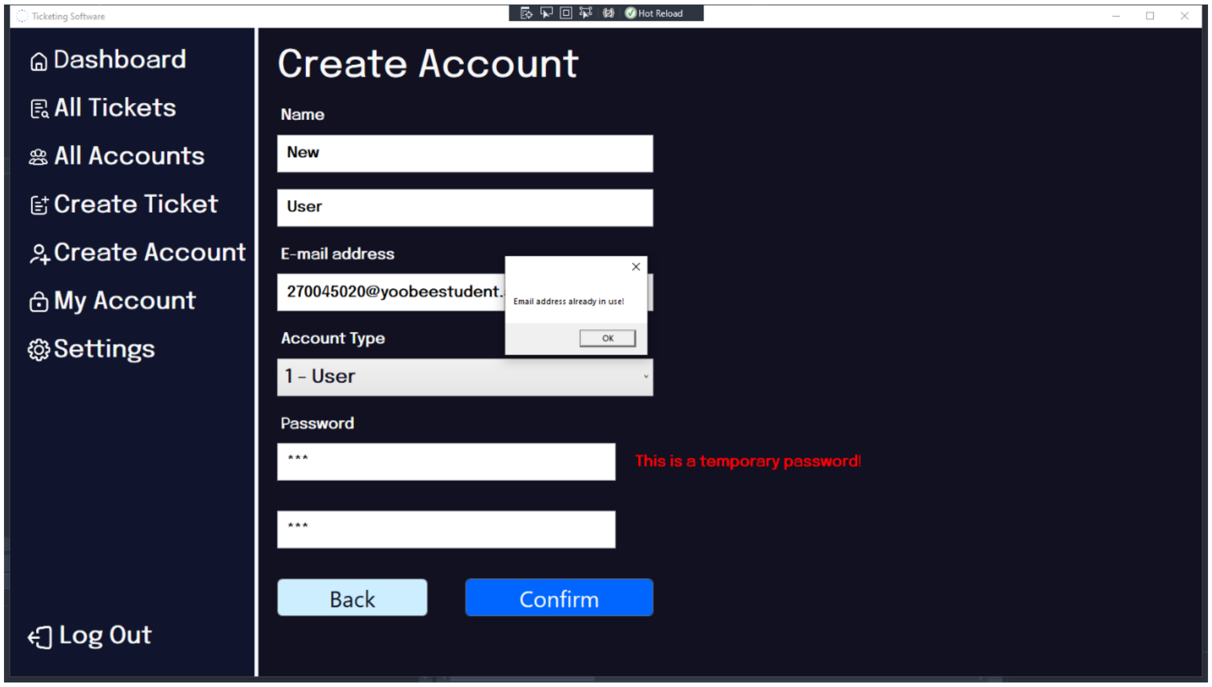
```
private void ConfirmBtn_Click(object sender, RoutedEventArgs e)
{

    if (FirstName.Text == "First Name")
    {
        MessageBoxResult invalidFirstName = MessageBox.Show("Please enter a valid value for first name!");
        return;
    }
    if (LastName.Text == "Last Name")
    {
        MessageBoxResult invalidLastName = MessageBox.Show("Please enter a valid value for last name!");
        return;
    }
    if (NewPasswordGhostText.IsVisible || ConfPasswordGhostText.IsVisible)
    {
        MessageBoxResult invalidPassword = MessageBox.Show("Please enter password in both fields!");
        return;
    }
    if (!(NewPassword.Password == ConfPassword.Password))
    {
        MessageBoxResult invalidPassword = MessageBox.Show("Passwords do not match! Please try again!");
        return;
    }
    if (!User.ValidateEmail(EmailAddress.Text))
    {
        MessageBoxResult invalidEmail = MessageBox.Show("Email address already in use!");
        return;
    }
    if (EmailAddress.Text == "Enter Email Address")
    {
        MessageBoxResult invalidEmail = MessageBox.Show("Invalid email address!");
        return;
    }

    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");
}
```

Figure 22: EventHandler, called when pressing the create account button

Used parameters

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

Received output



Create Account

Name

New

User

E-mail address

270045020@yoobeestudent.

Email address already in use!

Account Type

1 – User

Password

***

This is a temporary password!

***

Figure 23: Black Box Testing

Result

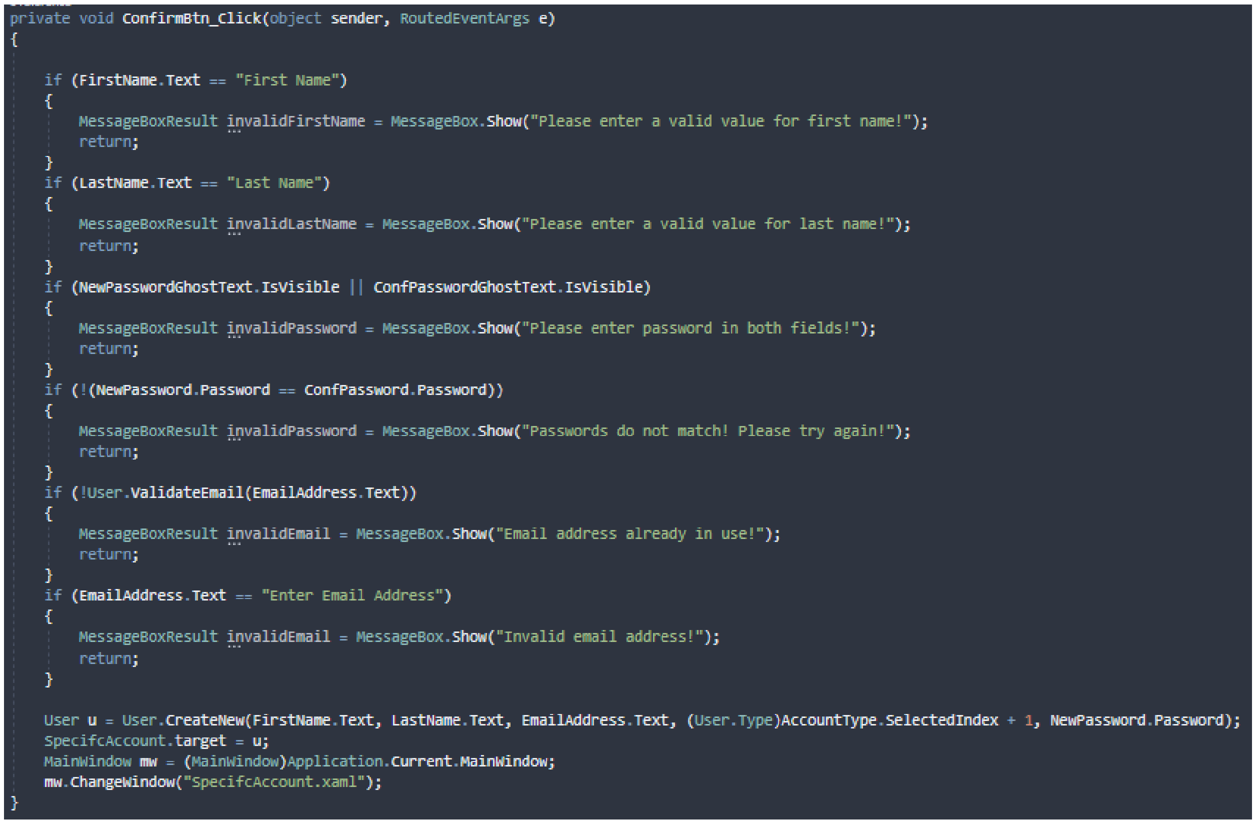
Passed

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

Expected output

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

Code snippet



```
private void ConfirmBtn_Click(object sender, RoutedEventArgs e)
{

    if (FirstName.Text == "First Name")
    {
        MessageBoxResult invalidFirstName = MessageBox.Show("Please enter a valid value for first name!");
        return;
    }
    if (LastName.Text == "Last Name")
    {
        MessageBoxResult invalidLastName = MessageBox.Show("Please enter a valid value for last name!");
        return;
    }
    if (NewPasswordGhostText.IsVisible || ConfPasswordGhostText.IsVisible)
    {
        MessageBoxResult invalidPassword = MessageBox.Show("Please enter password in both fields!");
        return;
    }
    if (!(NewPassword.Password == ConfPassword.Password))
    {
        MessageBoxResult invalidPassword = MessageBox.Show("Passwords do not match! Please try again!");
        return;
    }
    if (!User.ValidateEmail(EmailAddress.Text))
    {
        MessageBoxResult invalidEmail = MessageBox.Show("Email address already in use!");
        return;
    }
    if (EmailAddress.Text == "Enter Email Address")
    {
        MessageBoxResult invalidEmail = MessageBox.Show("Invalid email address!");
        return;
    }

    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");
}
```

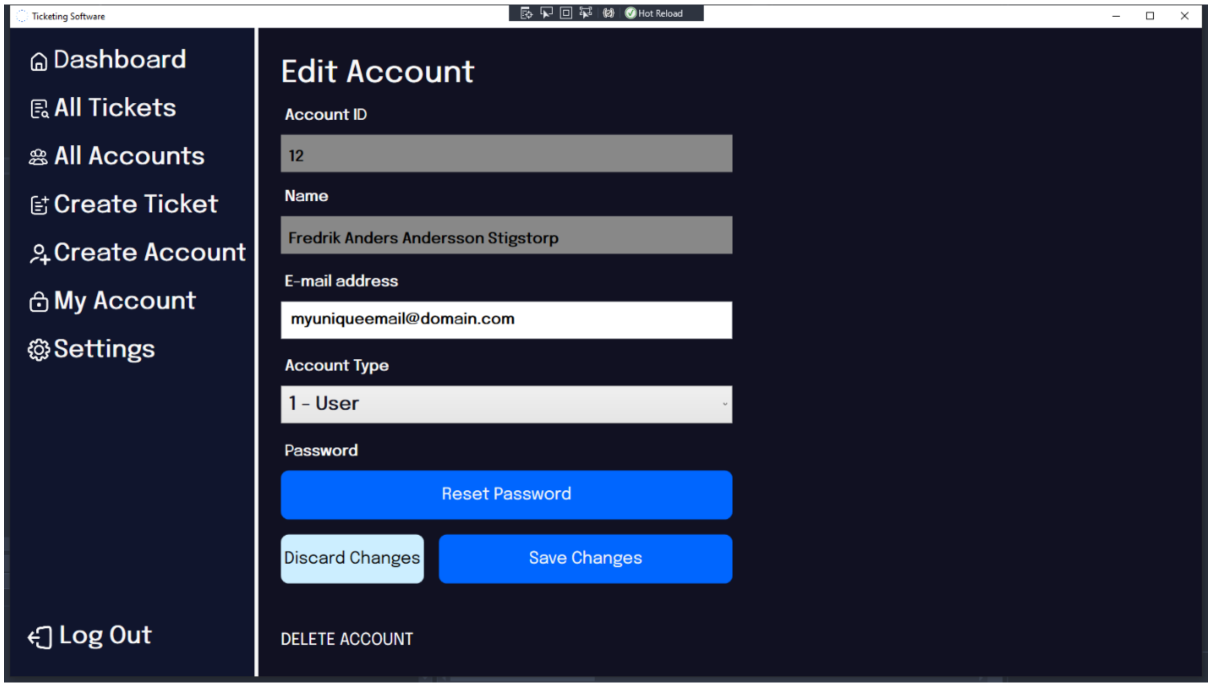
Figure 24: EventHandler, called when pressing the create account button

Used parameters

First name: Fredrik Anders

Last name: Andersson Stigstorp

Received output



Edit Account

Account ID: 12

Name: Fredrik Anders Andersson Stigstorp

E-mail address: myuniqueemail@domain.com

Account Type: 1 – User

Password

Figure 25: Functional Testing #6

Result

Passed

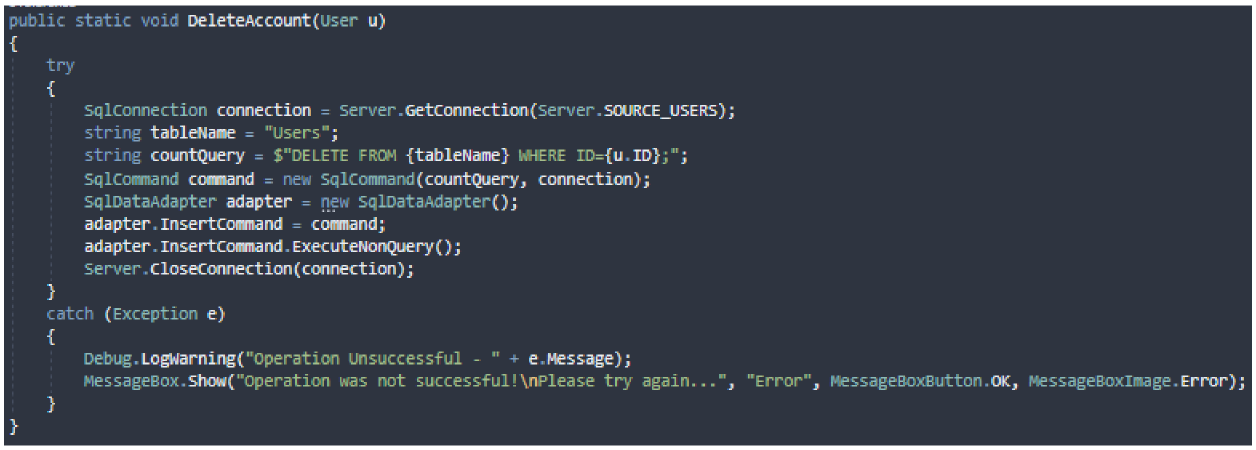
7. Delete account with ID 8

Expected output

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

Code snippet

Implementation



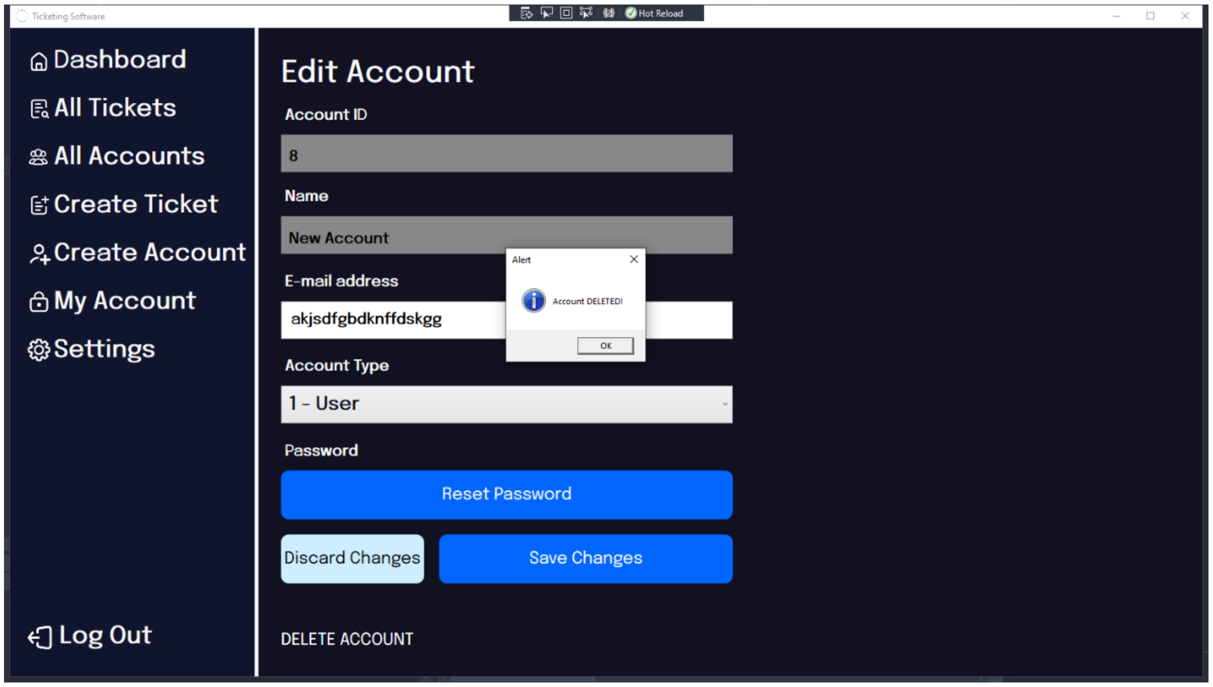
```
public static void DeleteAccount(User u)
{
    try
    {
        SqlConnection connection = Server.GetConnection(Server.SOURCE_USERS);
        string tableName = "Users";
        string countQuery = $"DELETE FROM {tableName} WHERE ID={u.ID};";
        SqlCommand command = new SqlCommand(countQuery, connection);
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.InsertCommand = command;
        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 26: DeleteAccount function, which takes a user input and removes it from the database

Used parameter

User: ID 8 out of 10

Received outputs



Edit Account

Account ID: 8

Name: New Account

E-mail address: akjsdfgbdknffdskgg

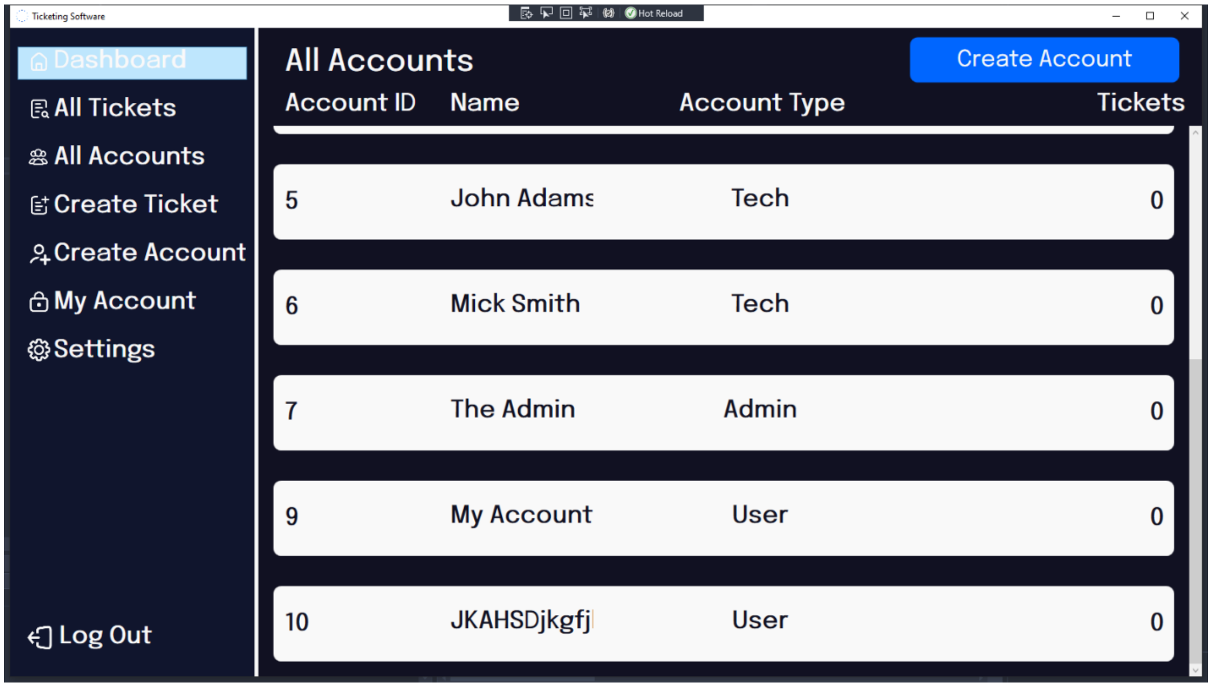
Account Type: 1 - User

Password

Alert

Account DELETED!

Figure 27: Functional Testing #7



All Accounts

Account ID	Name	Account Type	Tickets
5	John Adams	Tech	0
6	Mick Smith	Tech	0
7	The Admin	Admin	0
9	My Account	User	0
10	JKAHSDjkgfj	User	0

Figure 28: Functional Testing #8

Result

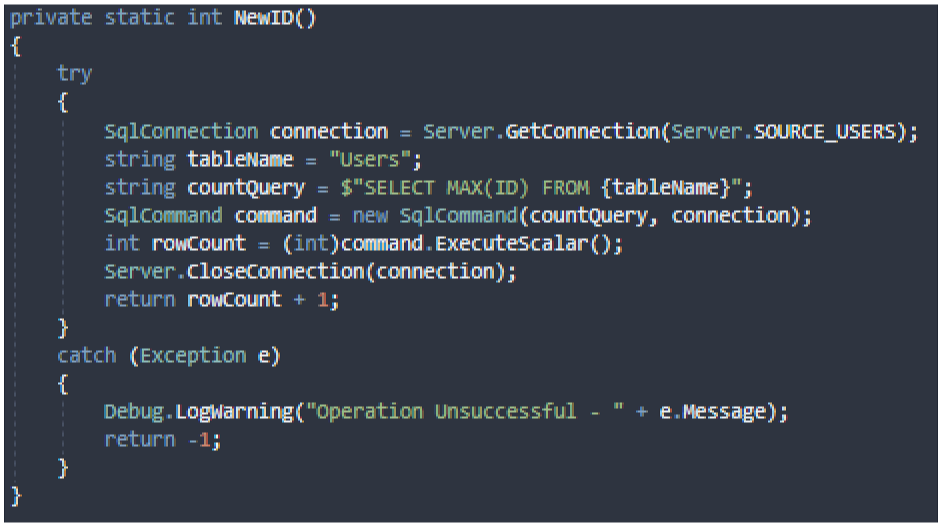
Passed

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.

Code snippet



```
private static int NewID()
{
    try
    {
        SqlConnection connection = Server.GetConnection(Server.SOURCE_USERS);
        string tableName = "Users";
        string countQuery = $"SELECT MAX(ID) FROM {tableName}";
        SqlCommand command = new SqlCommand(countQuery, connection);
        int rowCount = (int)command.ExecuteScalar();
        Server.CloseConnection(connection);
        return rowCount + 1;
    }
    catch (Exception e)
    {
        Debug.LogWarning("Operation Unsuccessful - " + e.Message);
        return -1;
    }
}
```

Figure 29: NewID function, used to generate a new, valid, ID for users

Used parameters

Database: Contains users 1-7, 9-10

First name: Lisbeth

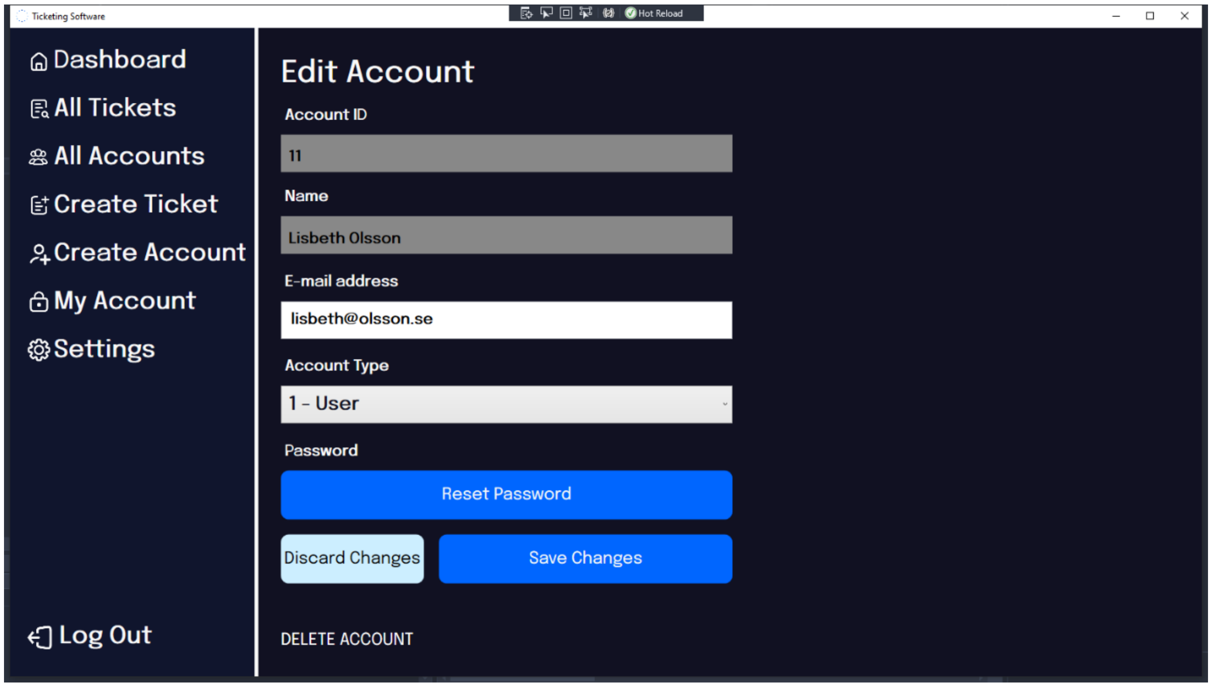
Last name: Olsson

E-mail: lisbeth@olsson.se

Account type: 1 (user)

Password: 123

Received output



Edit Account

Account ID: 11

Name: Lisbeth Olsson

E-mail address: lisbeth@olsson.se

Account Type: 1 - User

Password

Figure 30: Functional Testing #9

Result

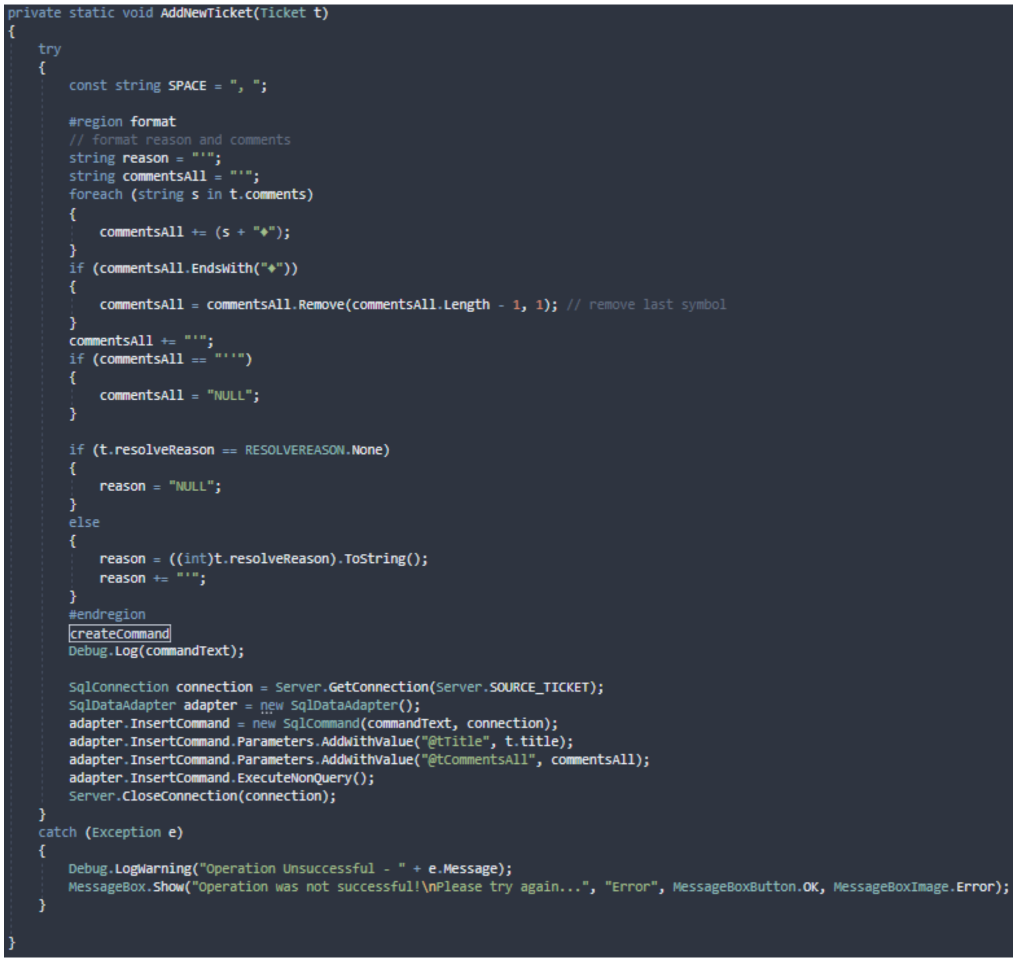
Passed

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.

Code snippet



```csharp
private static void AddNewTicket(Ticket t)
{
    try
    {
        const string SPACE = ", ";

        #region format
        // format reason and comments
        string reason = "'";
        string commentsAll = "'";
        foreach (string s in t.comments)
        {
            commentsAll += (s + "♦");
        }
        if (commentsAll.EndsWith("♦"))
        {
            commentsAll = commentsAll.Remove(commentsAll.Length - 1, 1); // remove last symbol
        }
        commentsAll += "'";
        if (commentsAll == "''")
        {
            commentsAll = "NULL";
        }

        if (t.resolveReason == RESOLVEREASON.None)
        {
            reason = "NULL";
        }
        else
        {
            reason = ((int)t.resolveReason).ToString();
            reason += "'";
        }
        #endregion
        createCommand
        Debug.Log(commandText);

        SqlConnection connection = Server.GetConnection(Server.SOURCE_TICKET);
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.InsertCommand = new SqlCommand(commandText, connection);
        adapter.InsertCommand.Parameters.AddWithValue("@tTitle", t.title);
        adapter.InsertCommand.Parameters.AddWithValue("@tCommentsAll", commentsAll);
        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 31: AddNewTicket function, used as final step to push a ticket to the database

Used parameters

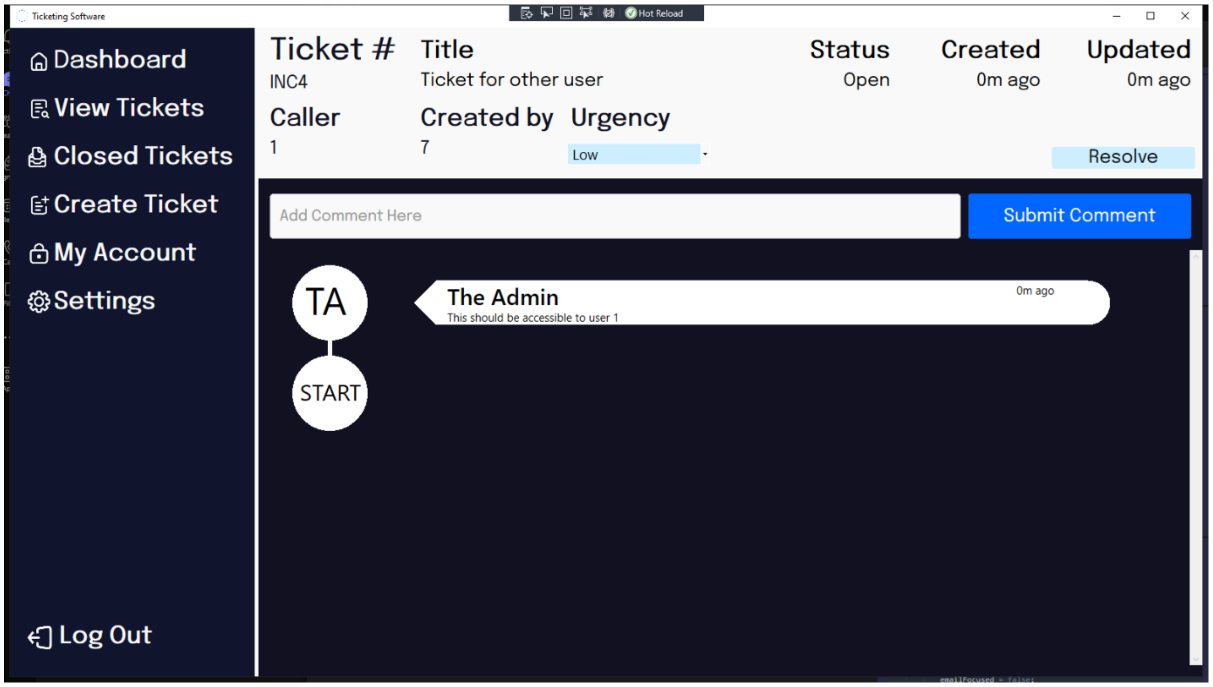
Creator: User with ID 7

Caller: User with ID 1

Ticket title: Ticket for other user

Ticket description: This should be accessible to user 1

Received output


Ticketing Software

Hot Reload

Dashboard
View Tickets
Closed Tickets
Create Ticket
My Account
Settings
Log Out

Ticket #	Title	Status	Created	Updated
INC4	Ticket for other user	Open	0m ago	0m ago

Caller	Created by	Urgency
1	7	Low

The Admin
0m ago
This should be accessible to user 1

Figure 32: Functional Testing #10

Result

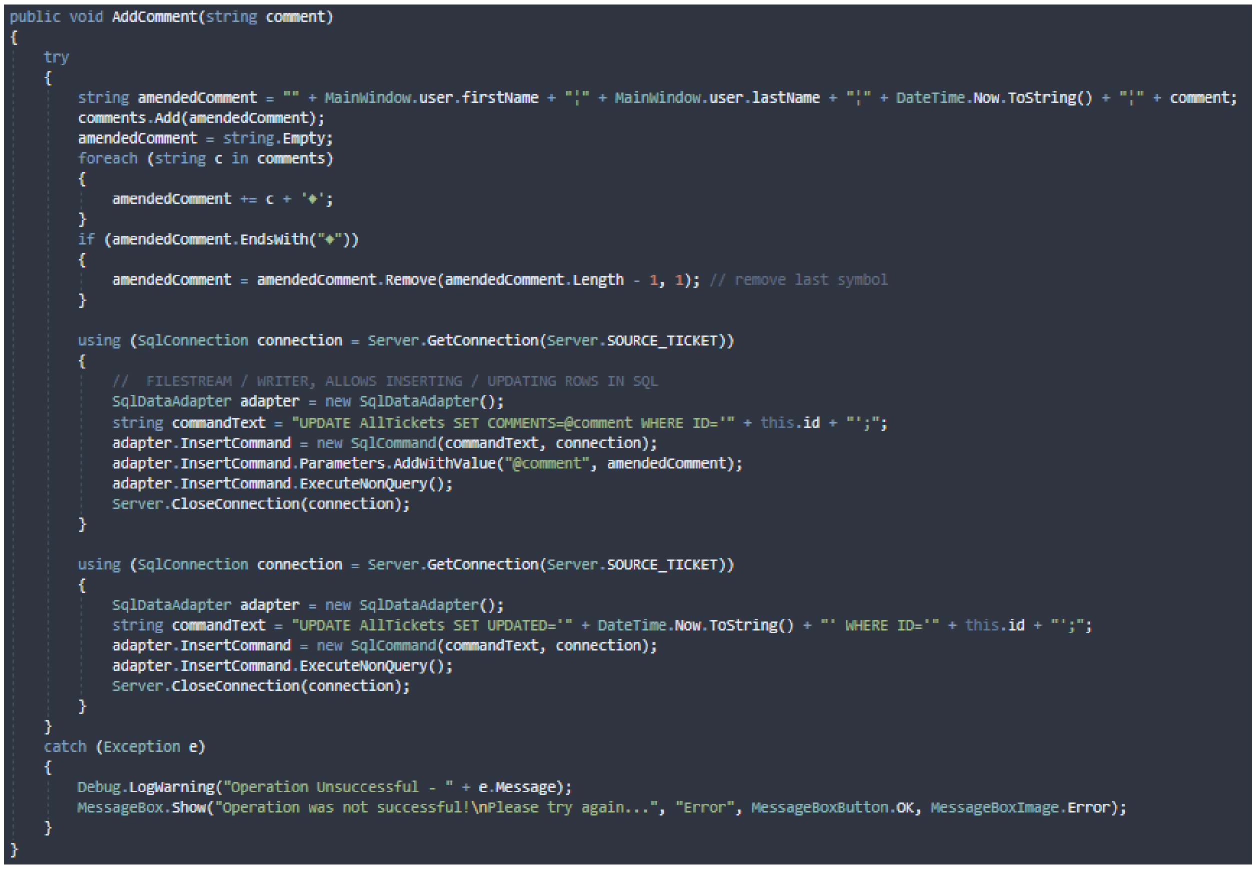
Passed

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.

Code snippet



```
public void AddComment(string comment)
{
    try
    {
        string amendedComment = "" + MainWindow.user.firstName + "¦" + MainWindow.user.lastName + "¦" + DateTime.Now.ToString() + "¦" + comment;
        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 33: AddComment function, used to push a comment to the database

User parameters

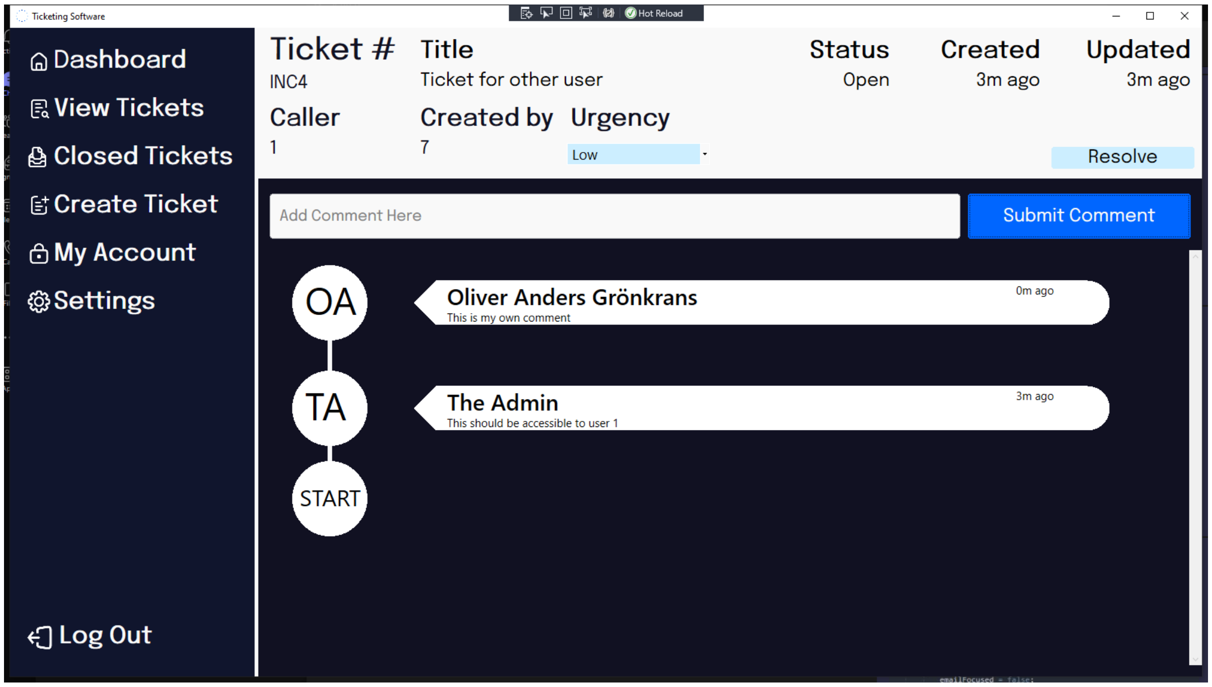
Ticket: INC4 (Ticket from previous test)

Creator: User with ID 7

Caller/Commenter: User with ID 1

Comment text: This is my own comment

Received output


Ticketing Software

Ticket #
INC4

Title
Ticket for other user

Status
Open

Created
3m ago

Updated
3m ago

Caller
1

Created by
7

Urgency
Low

Oliver Anders Grönkrans
This is my own comment
0m ago

The Admin
This should be accessible to user 1
3m ago

Figure 34: Functional Testing #11

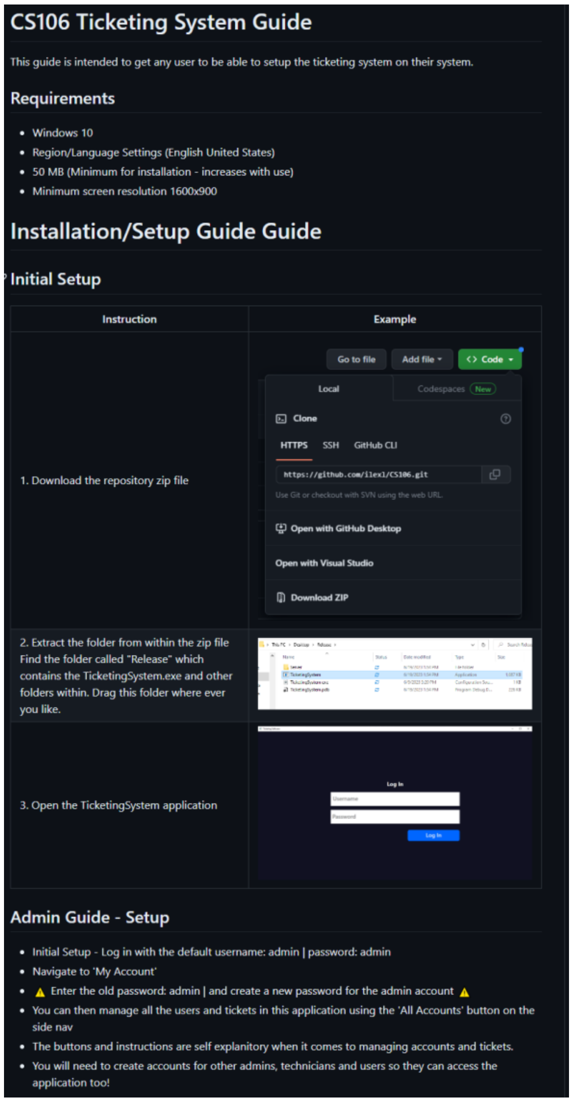
Result

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

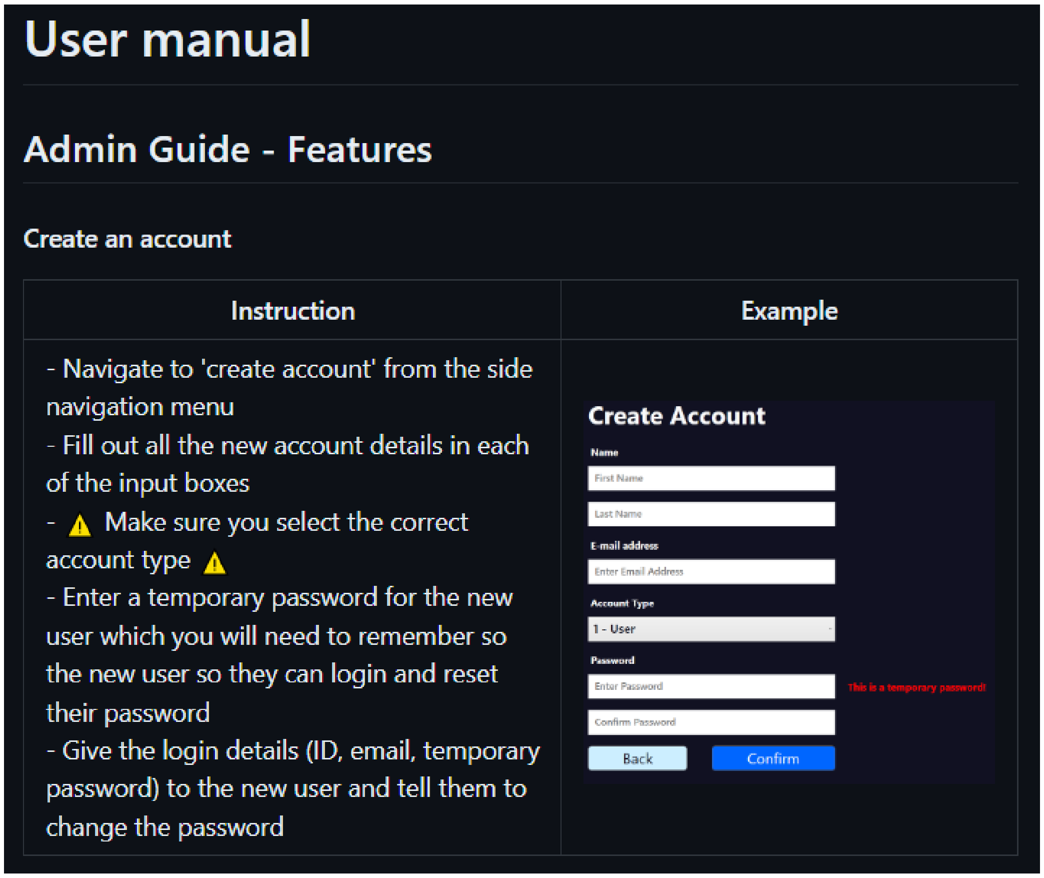
Instruction	Example
1. Download the repository zip file	Go to file Add file Code Local Codespaces New Clone HTTPS SSH GitHub CLI https://github.com/ilexl/CS106.git Use Git or checkout with SVN using the web URL. Open with GitHub Desktop Open with Visual Studio Download ZIP
2. Extract the folder from 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 35: Installation guide

User Guide



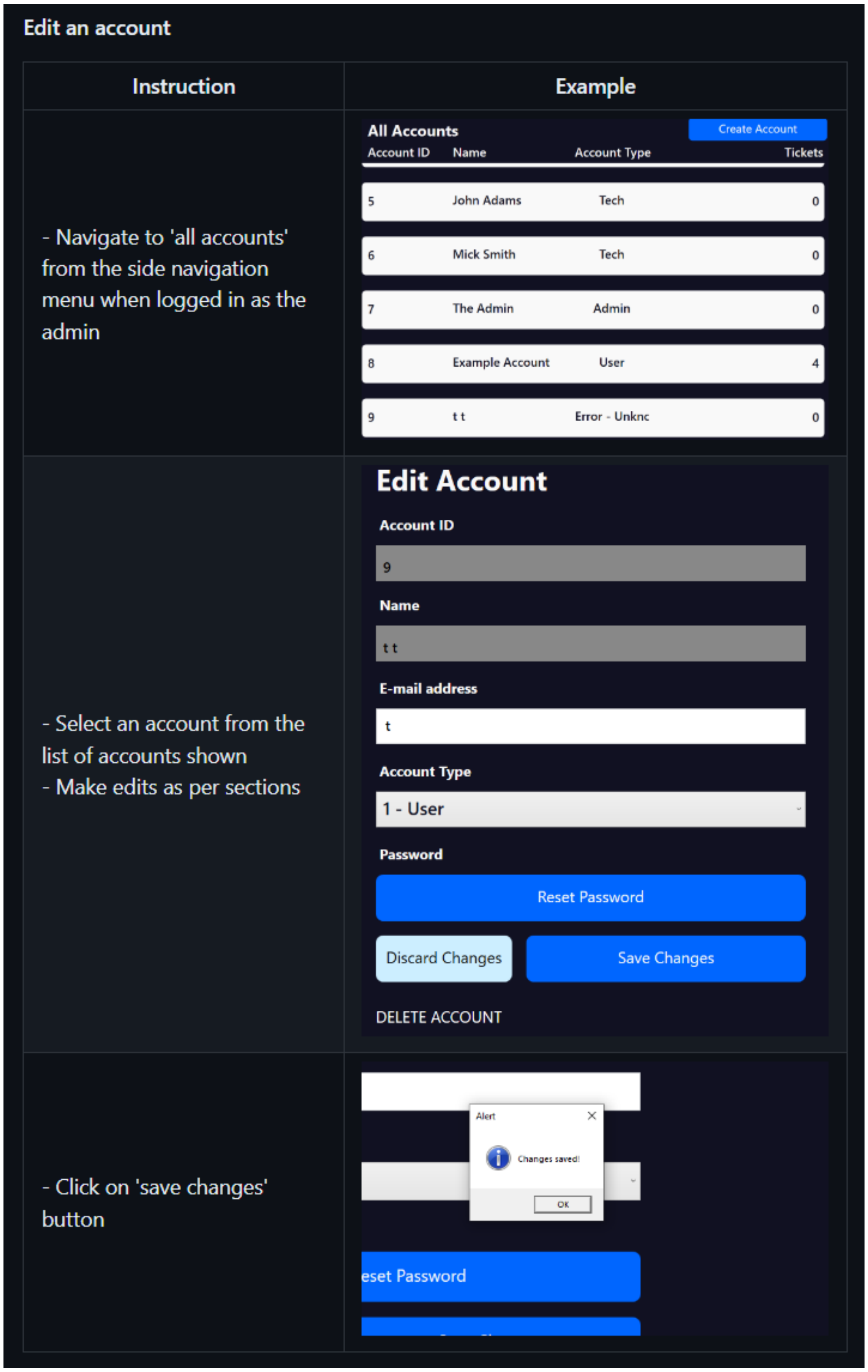
User manual

Admin Guide - Features

Create an account

Instruction	Example
- Navigate to 'create account' from the side navigation menu - Fill out all the new account details in each of the input boxes - ⚠ Make sure you select the correct account type ⚠ - Enter a temporary password for the new user which you will need to remember so the new user so they can login and reset their password - Give the login details (ID, email, temporary password) to the new user and tell them to change the password	**Create Account** Name First Name Last Name E-mail address Enter Email Address Account Type 1 - User Password Enter Password — This is a temporary password! Confirm Password Back Confirm

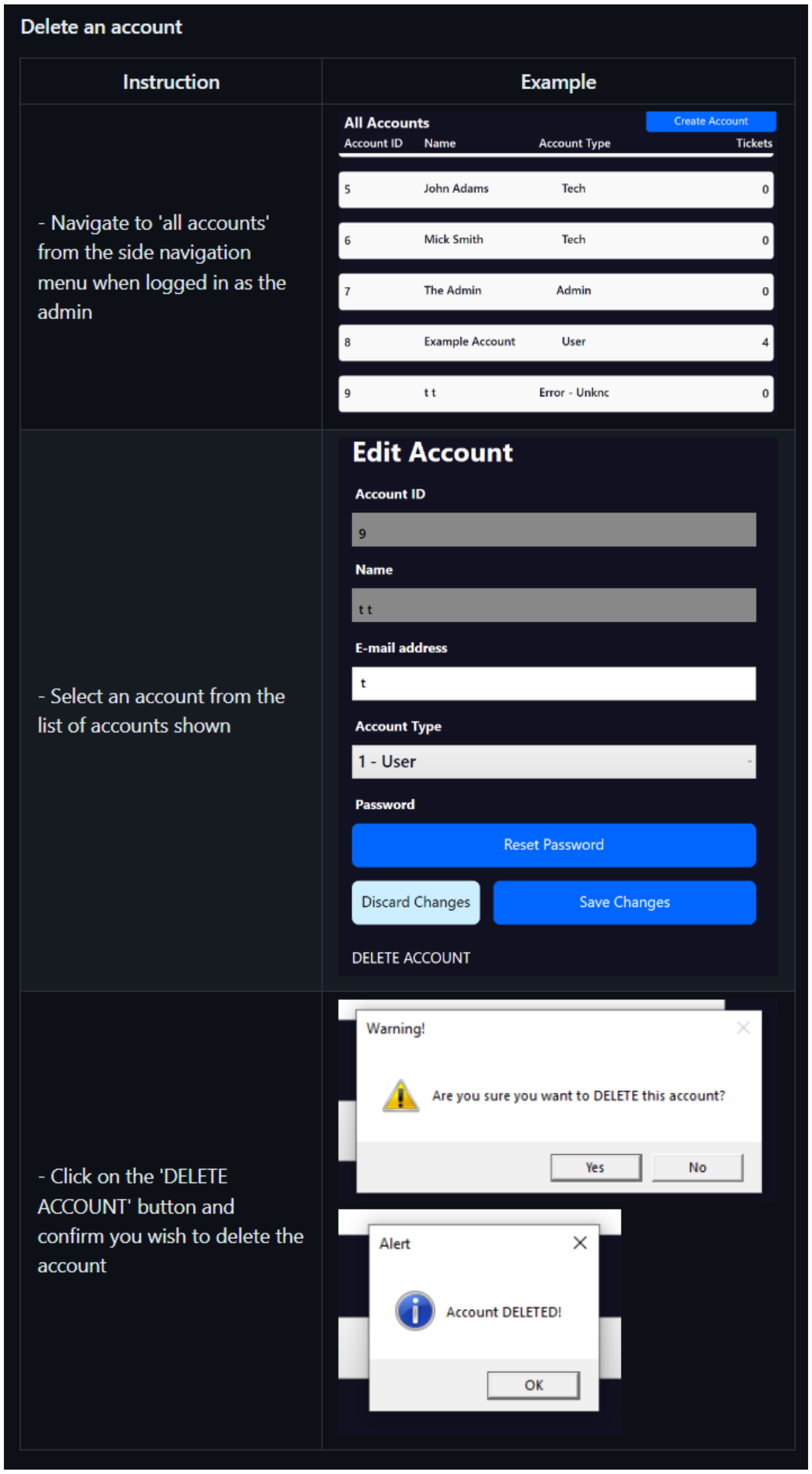
Figure 36: User Guide #1



Edit an account

Instruction	Example
- Navigate to 'all accounts' from the side navigation menu when logged in as the admin	**All Accounts** · Create Account Account ID · Name · Account Type · Tickets 5 · John Adams · Tech · 0 6 · Mick Smith · Tech · 0 7 · The Admin · Admin · 0 8 · Example Account · User · 4 9 · t t · Error - Unknc · 0
- Select an account from the list of accounts shown - Make edits as per sections	**Edit Account** **Account ID**: 9 **Name**: t t **E-mail address**: t **Account Type**: 1 - User **Password**: Reset Password Discard Changes · Save Changes DELETE ACCOUNT
- Click on 'save changes' button	Alert Changes saved! OK eset Password

Figure 37: User Guide #2



Delete an account

Instruction	Example
- Navigate to 'all accounts' from the side navigation menu when logged in as the admin	All Accounts (Create Account)
- Select an account from the list of accounts shown	Edit Account
- Click on the 'DELETE ACCOUNT' button and confirm you wish to delete the account	Warning! Are you sure you want to DELETE this account? Yes / No Alert: Account DELETED! OK

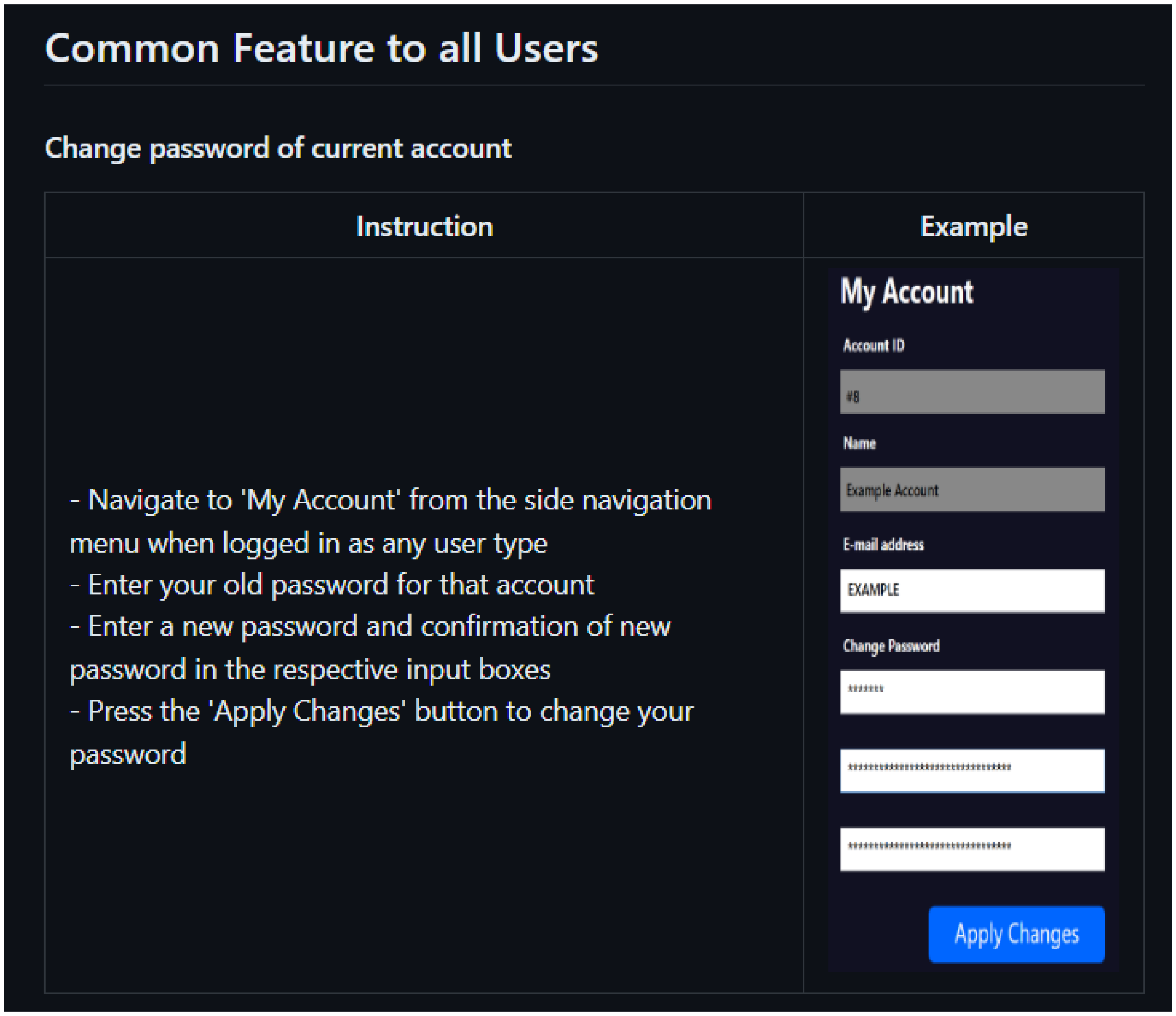
All Accounts

Account ID	Name	Account Type	Tickets
5	John Adams	Tech	0
6	Mick Smith	Tech	0
7	The Admin	Admin	0
8	Example Account	User	4
9	t t	Error - Unknc	0

Edit Account

Account ID	9
Name	t t
E-mail address	t
Account Type	1 - User
Password	Reset Password

Figure 38: User Guide #3

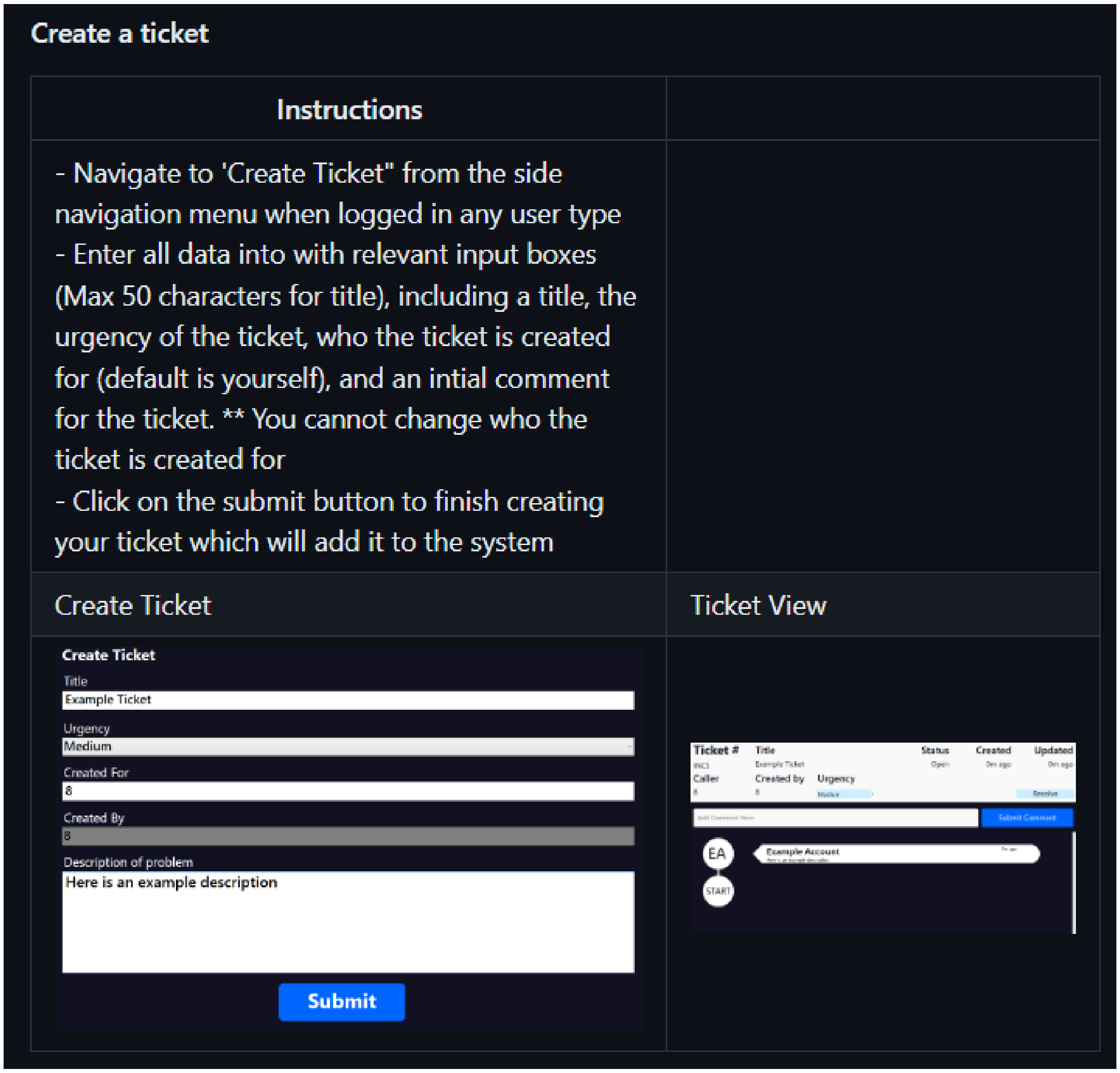


Common Feature to all Users

Change password of current account

Instruction	Example
- Navigate to 'My Account' from the side navigation menu when logged in as any user type - Enter your old password for that account - Enter a new password and confirmation of new password in the respective input boxes - Press the 'Apply Changes' button to change your password	**My Account** Account ID: #8 Name: Example Account E-mail address: EXAMPLE Change Password: ******* ********************************** ********************************** Apply Changes

Figure 39: User Guide #4

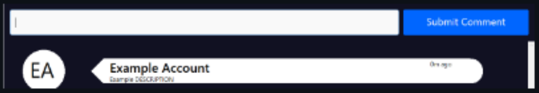


Create a ticket

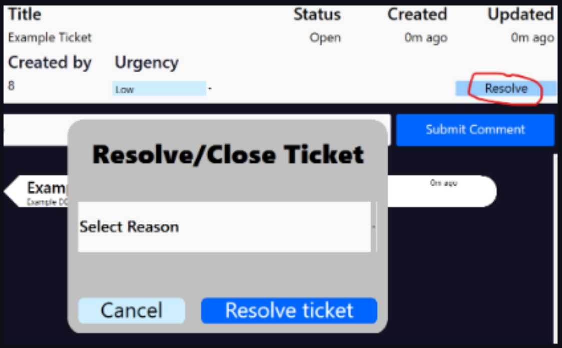
Instructions	
- Navigate to 'Create Ticket" from the side navigation menu when logged in any user type - Enter all data into with relevant input boxes (Max 50 characters for title), including a title, the urgency of the ticket, who the ticket is created for (default is yourself), and an intial comment for the ticket. ** You cannot change who the ticket is created for - Click on the submit button to finish creating your ticket which will add it to the system	
Create Ticket	Ticket View
Create Ticket Title: Example Ticket Urgency: Medium Created For: 8 Created By: 8 Description of problem: Here is an example description **Submit**	Ticket # INC5 · Title Example Ticket · Status Open · Created 0m ago · Updated 0m ago Caller 8 · Created by 8 · Urgency Medium · Resolve Add Comment Here · Submit Comment EA · Example Account · START

Figure 40: User Guide #5

Add comment to ticket

Instruction	Example
- Open a ticket from the list of tickets or create a ticket - Type in the comment box below the ticket info and above the current comments - Click on the 'submit comment' button to add the comment	

Resolve a ticket

Instruction	Example
- Open or create a ticket - Click on the 'resolve' button - A window will pop up to select a reason - Select a reason to resolve/close the ticket - Click the 'resolve ticket' button on the pop up window	 Title Status Created Updated Example Ticket Open 0m ago 0m ago Created by Urgency 8 Low Resolve Submit Comment Resolve/Close Ticket Select Reason Cancel Resolve ticket

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

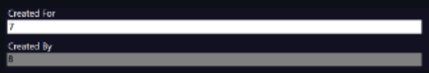
Instruction	Example
- When creating a ticket there is a greyed out 'created by' field and a 'created for' field which you can change. - By default they are the same, however you can change who the ticket is created for in cases where tickets are created on behalf of somebody else - To create the ticket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

Figure 41: User Guide #6