# Title Page

2207-BSE

Implementation – Evidence Document

CS106.2

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

# Document Outline

## Table of Contents

[Title Page 1](#_Toc138420950)

[Document Outline 2](#_Toc138420951)

[Table of Contents 2](#_Toc138420952)

[Table of Figures 2](#_Toc138420953)

[Table of Tables 3](#_Toc138420954)

[Changes 5](#_Toc138420955)

[HiFi 5](#_Toc138420956)

[System Architecture 6](#_Toc138420957)

[Activity Diagram 6](#_Toc138420958)

[Class Diagram 7](#_Toc138420959)

[Justification 8](#_Toc138420960)

[Key Functionality 9](#_Toc138420961)

[Function Testing 16](#_Toc138420962)

[Black Box Testing 16](#_Toc138420963)

[User Documentation 26](#_Toc138420964)

[Installation Guide 26](#_Toc138420965)

[User Guide 27](#_Toc138420966)

## Table of Figures

[Figure 1: Hifi Changes 5](#_Toc138344184)

[Figure 2: Activity Diagram 6](https://myacg-my.sharepoint.com/personal/270168960_yoobeestudent_ac_nz/Documents/Desktop/Implementation.docx#_Toc138344185)

[Figure 3: Class Diagram 7](#_Toc138344186)

[Figure 4: Key Functionality #1 9](#_Toc138344187)

[Figure 5: Key Functionality #2 9](#_Toc138344188)

[Figure 6: Key Functionality #3 10](#_Toc138344189)

[Figure 7: Key Functionality #4 11](#_Toc138344190)

[Figure 8: Key Functionality #5 11](#_Toc138344191)

[Figure 9: Key Functionality #6 12](#_Toc138344192)

[Figure 10: Key Functionality #7 13](#_Toc138344193)

[Figure 11: Key Functionality #8 13](#_Toc138344194)

[Figure 12: Key Functionality #9 14](#_Toc138344195)

[Figure 13: Key Functionality #10 15](#_Toc138344196)

[Figure 14: Functional Testing #1 16](#_Toc138344197)

[Figure 15: Functional Testing #2 17](#_Toc138344198)

[Figure 16: Functional Testing #3 18](#_Toc138344199)

[Figure 17: Functional Testing #4 19](#_Toc138344200)

[Figure 18: Functional Testing #5 20](#_Toc138344201)

[Figure 19: Functional Testing #6 21](#_Toc138344202)

[Figure 20: Functional Testing #7 22](#_Toc138344203)

[Figure 21: Functional Testing #8 22](#_Toc138344204)

[Figure 22: Functional Testing #9 23](#_Toc138344205)

[Figure 23: Functional Testing #10 24](#_Toc138344206)

[Figure 24: Functional Testing #11 25](#_Toc138344207)

[Figure 25: Installation guide 26](#_Toc138344208)

[Figure 26: User Guide #1 27](#_Toc138344209)

[Figure 27: User Guide #2 28](#_Toc138344210)

[Figure 28: User Guide #3 29](#_Toc138344211)

[Figure 29: User Guide #4 30](#_Toc138344212)

[Figure 30: User Guide #5 31](#_Toc138344213)

[Figure 31: User Guide #6 32](#_Toc138344214)

[Figure 32: User Guide #7 33](#_Toc138344215)

## Table of Tables

**No table of figures entries found.**

Intentionally Blank

# Changes

## HiFi

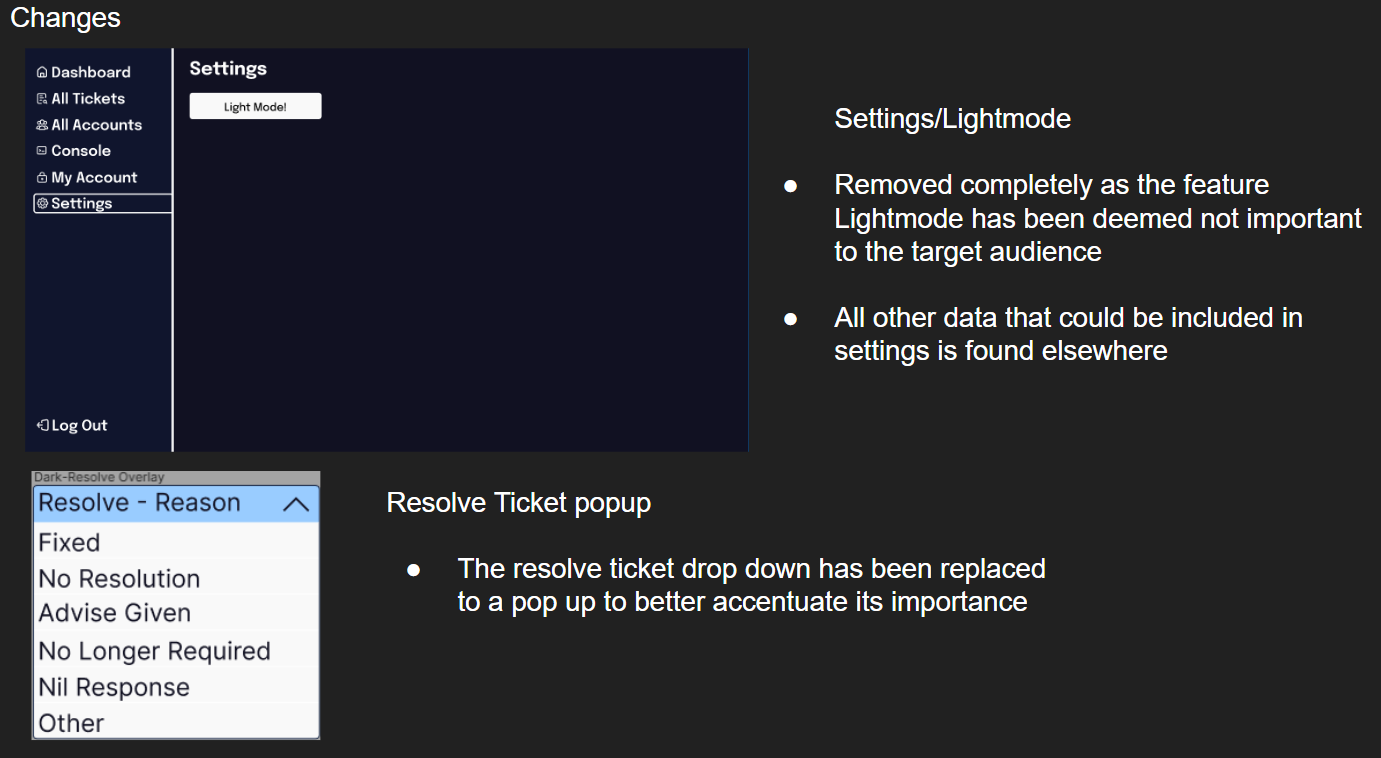


Figure 1: Hifi Changes

# System Architecture

## Activity Diagram

Figure 2: Activity Diagram

## Class Diagram

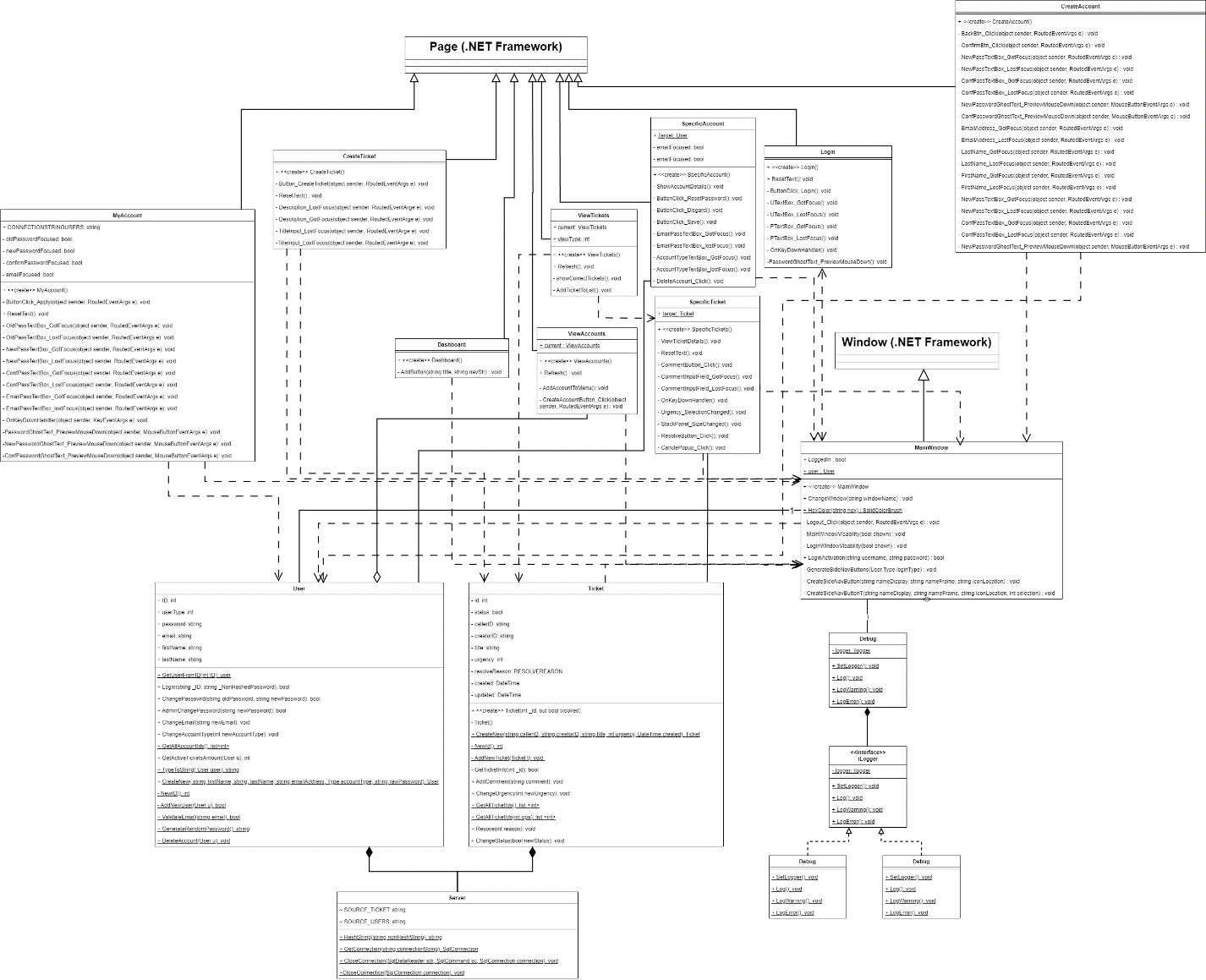


Figure 3: Class Diagram

## 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#:

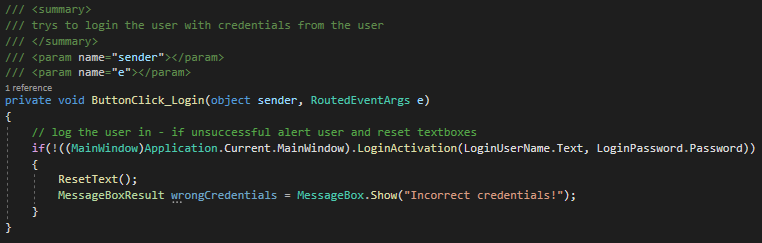


Figure 4: Key Functionality #1

System:

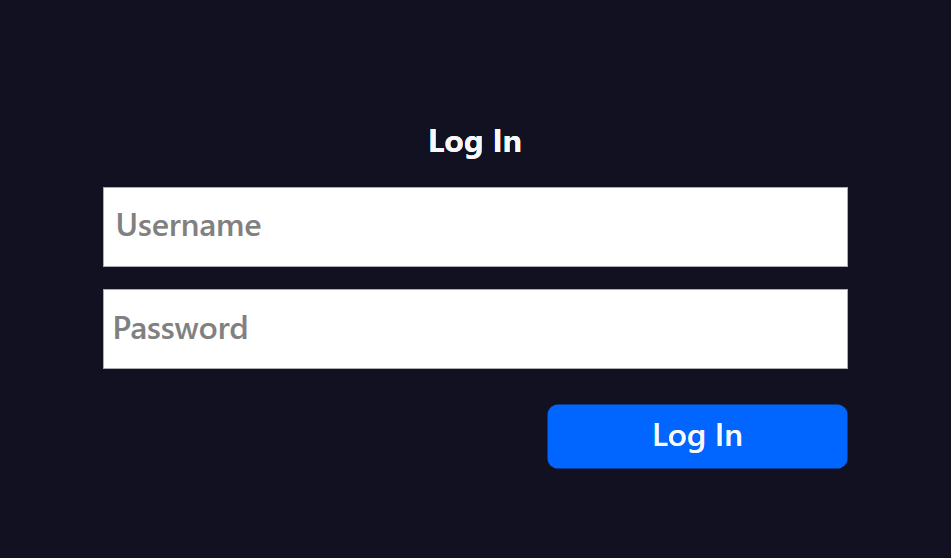


Figure 5: Key Functionality #2

**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#:

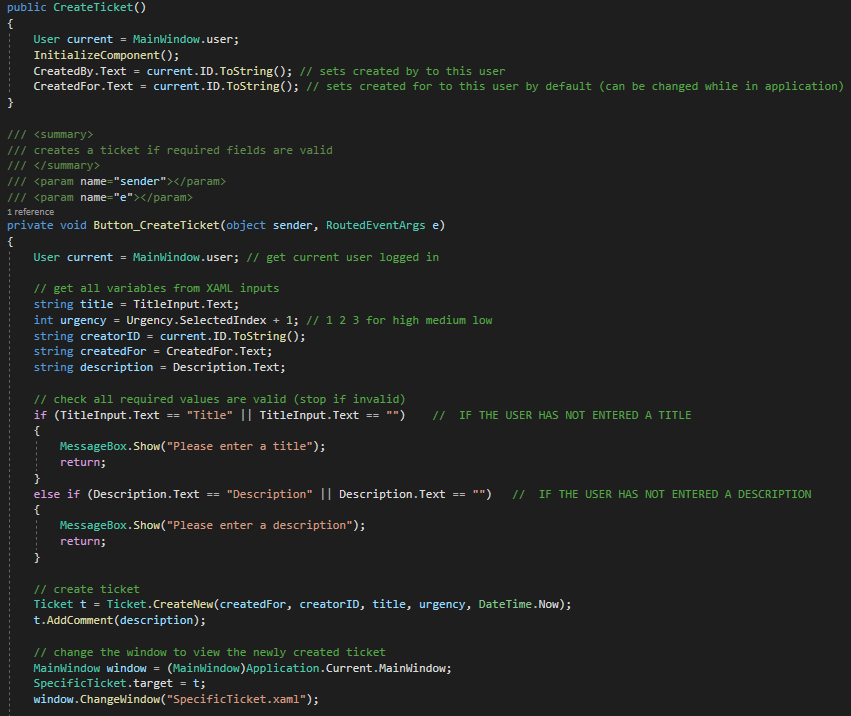


Figure 6: Key Functionality #3

System:

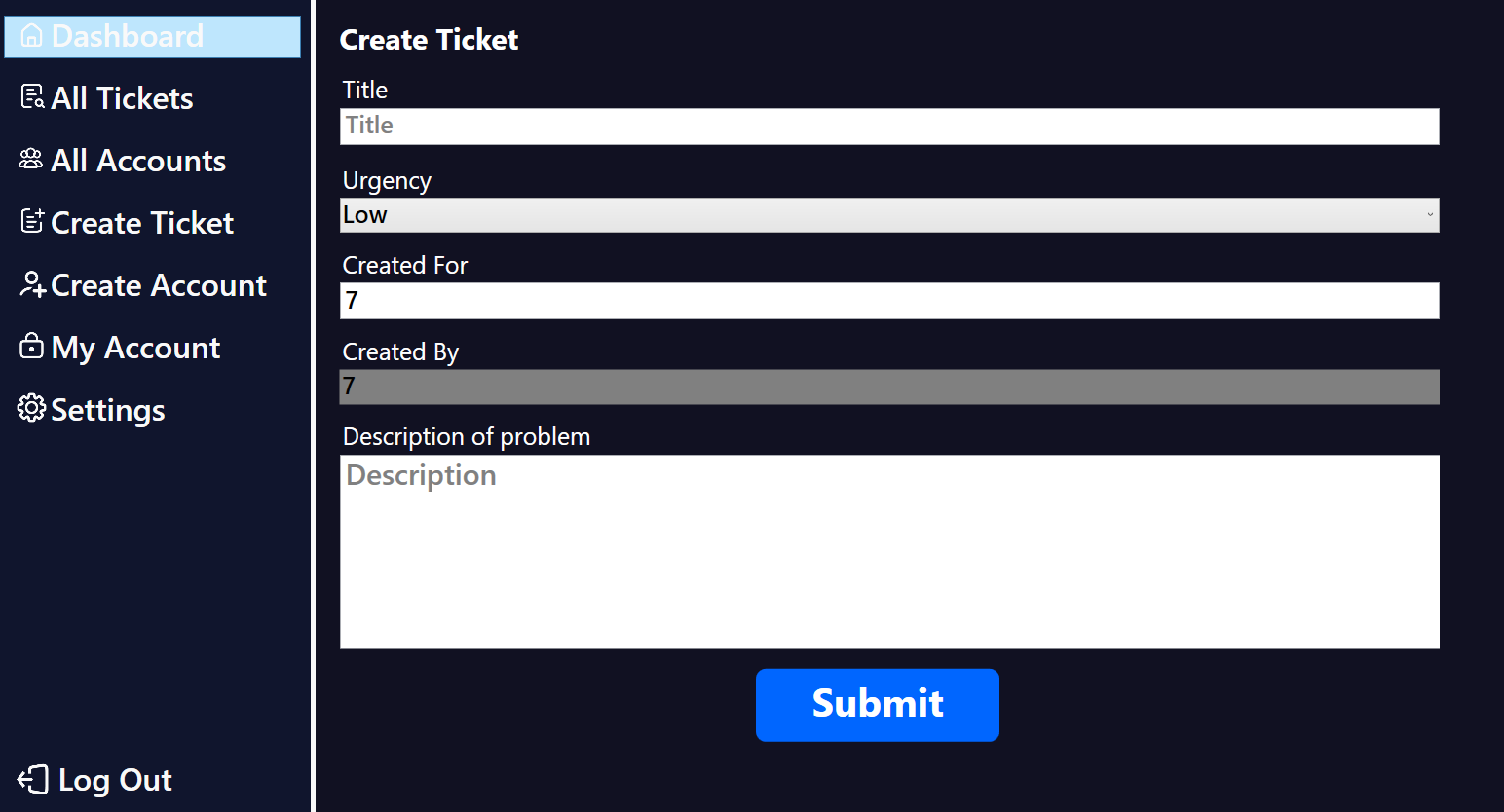


Figure 7: Key Functionality #4

**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#:

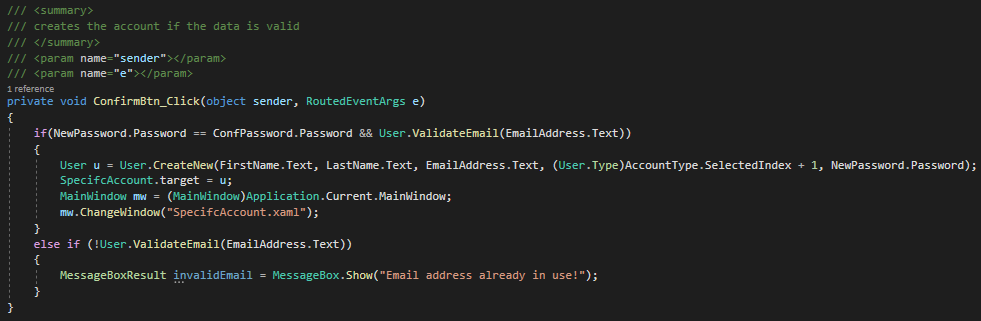


Figure 8: Key Functionality #5

System:

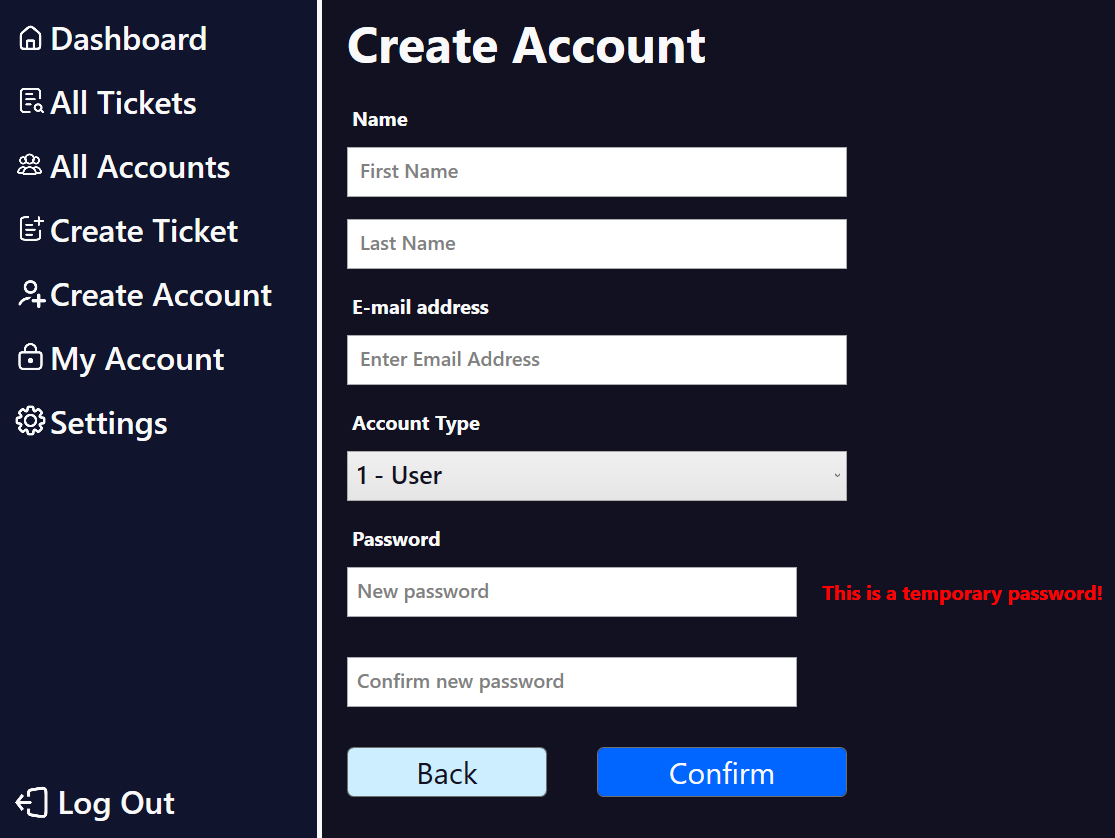


Figure 9: Key Functionality #6

**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#:

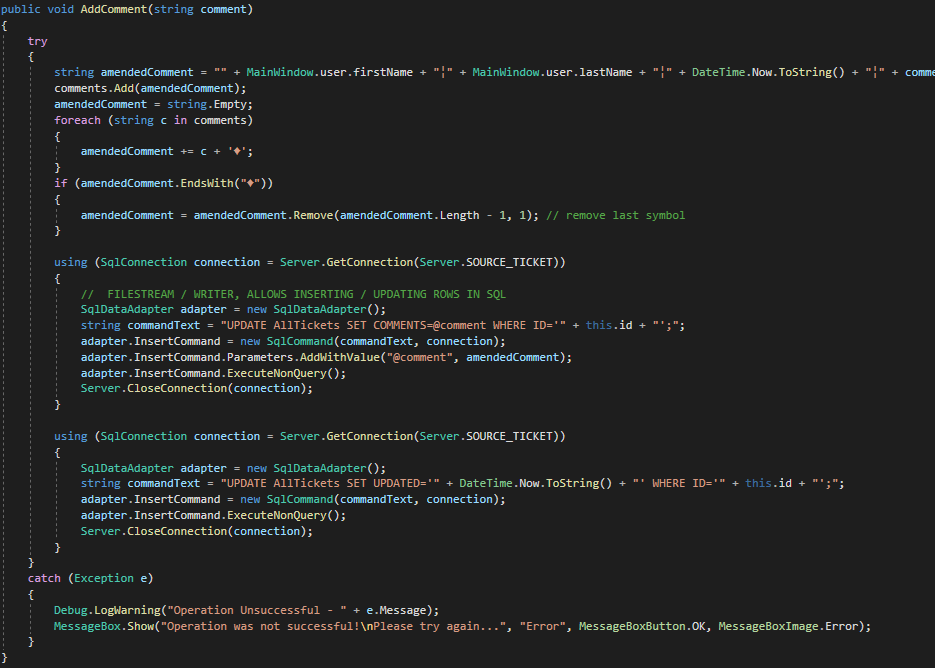


Figure 10: Key Functionality #7

System:

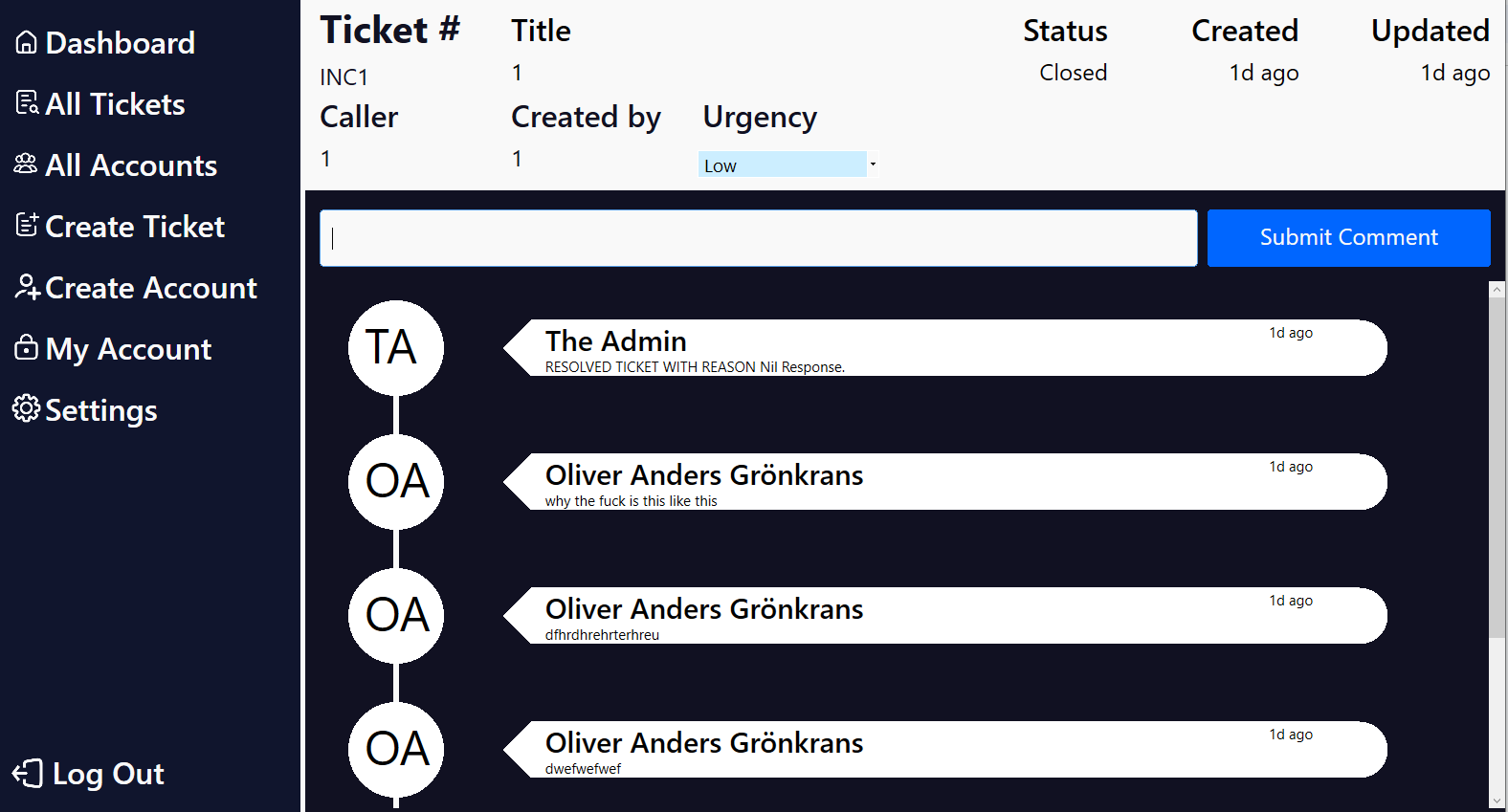


Figure 11: Key Functionality #8

**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#:

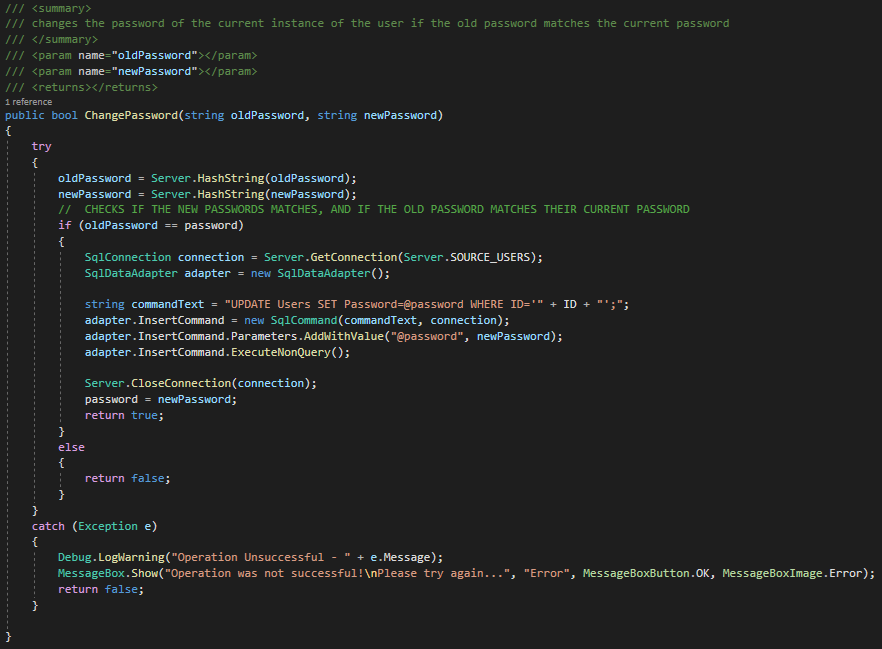


Figure 12: Key Functionality #9

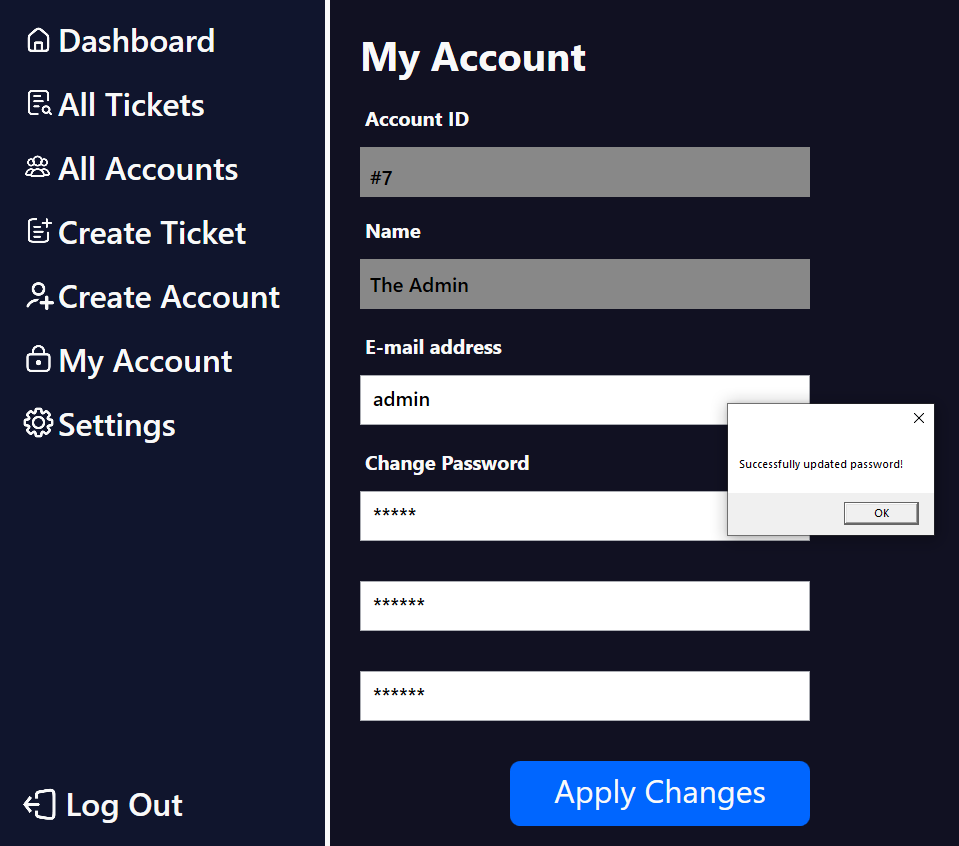
System:

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**

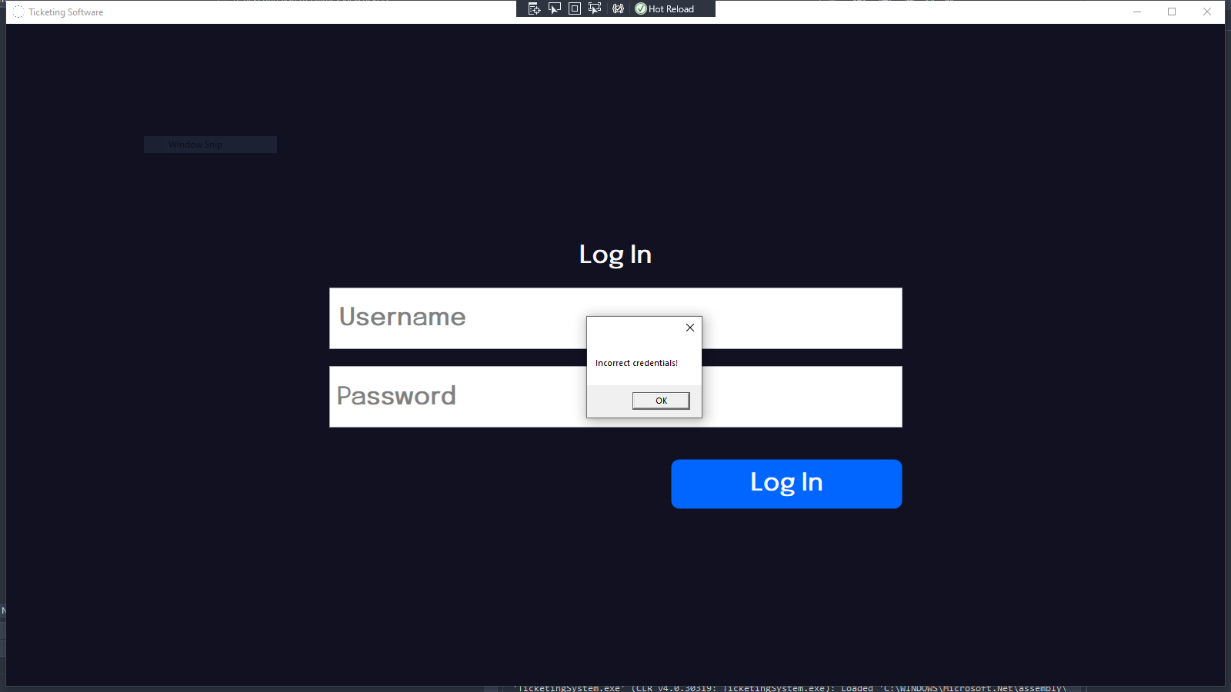


Figure 14: Functional Testing #1

**Result**

Passed

1. **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**

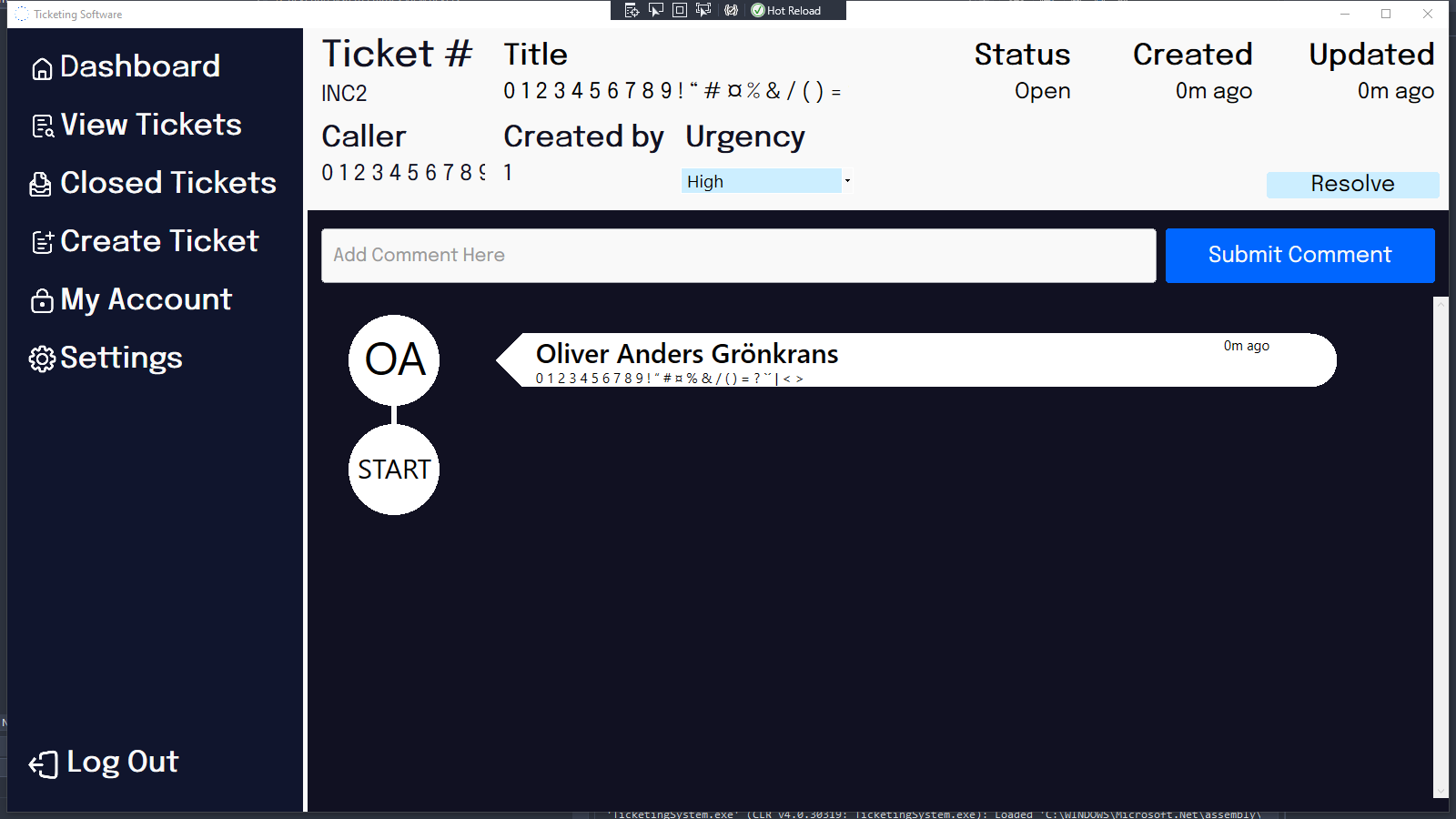


Figure 15: Functional Testing #2

**Result**

Passed

1. **Resolve ticket**

**Expected output**

Changes documented in comment field without issue.

**Used parameters**

Resolve status - Fixed

**Received output**

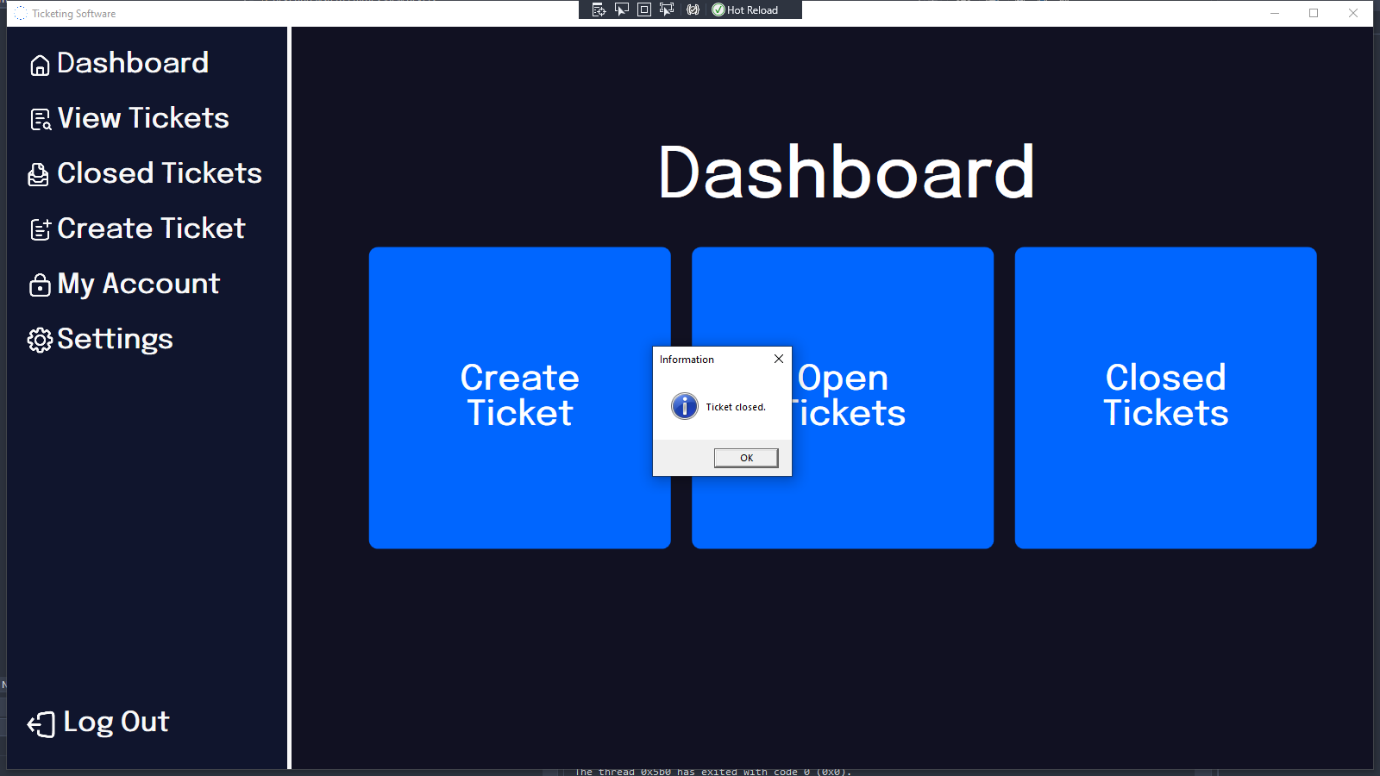


Figure 16: Functional Testing #3

**Result**

Passed

1. **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**

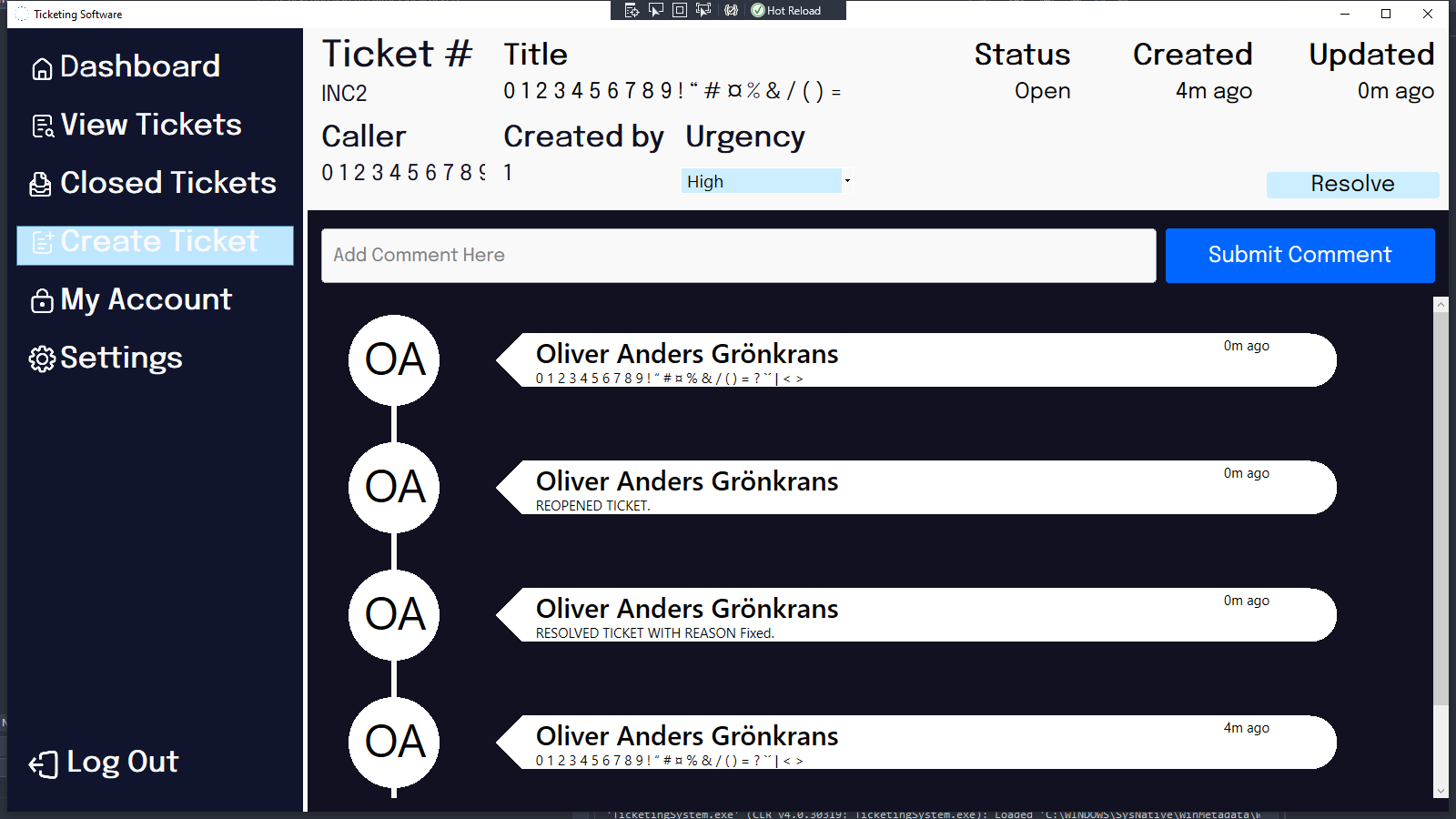


Figure 17: Functional Testing #4

**Result**

Passed

1. **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](mailto:270045020@yoobeestudent.ac.nz) (used by user 1)

**Received output**

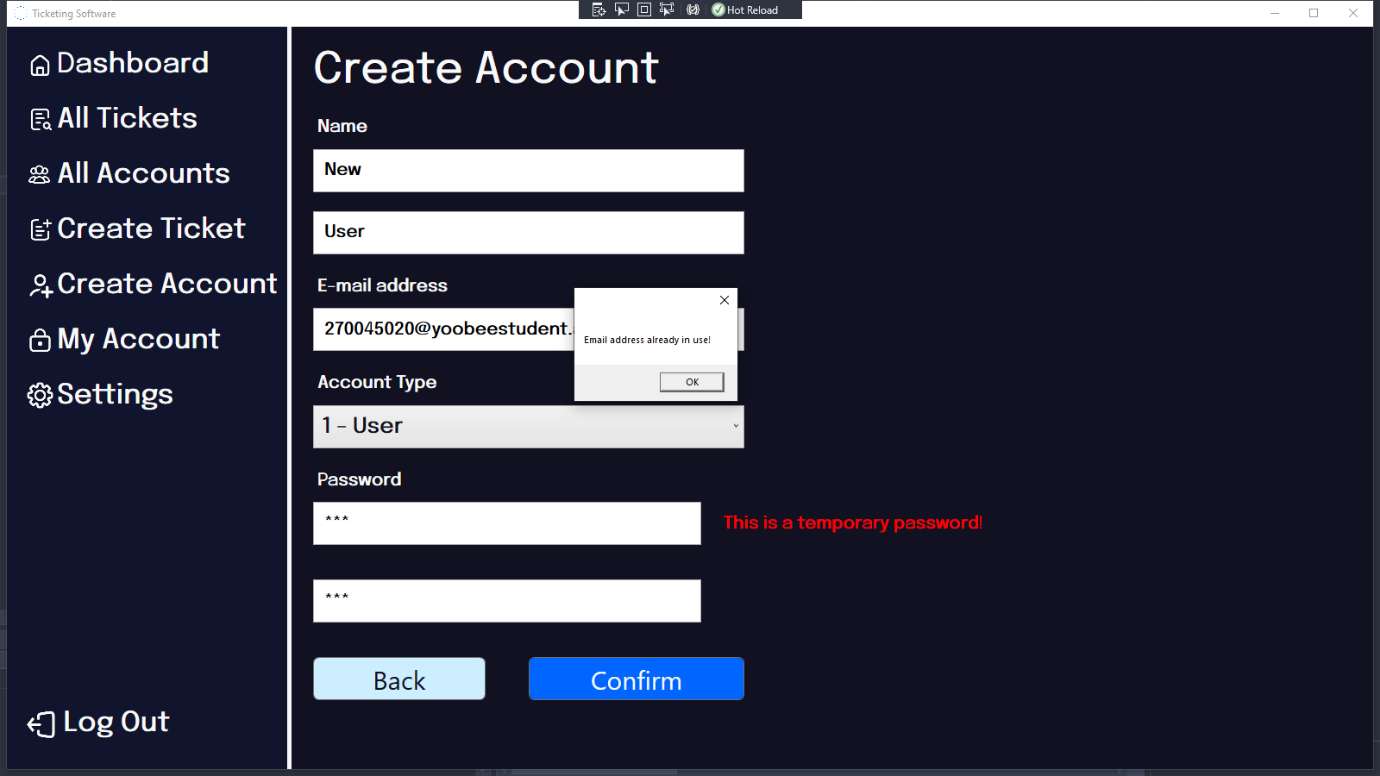


Figure 18: Functional Testing #5

**Result**

Passed

1. **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**

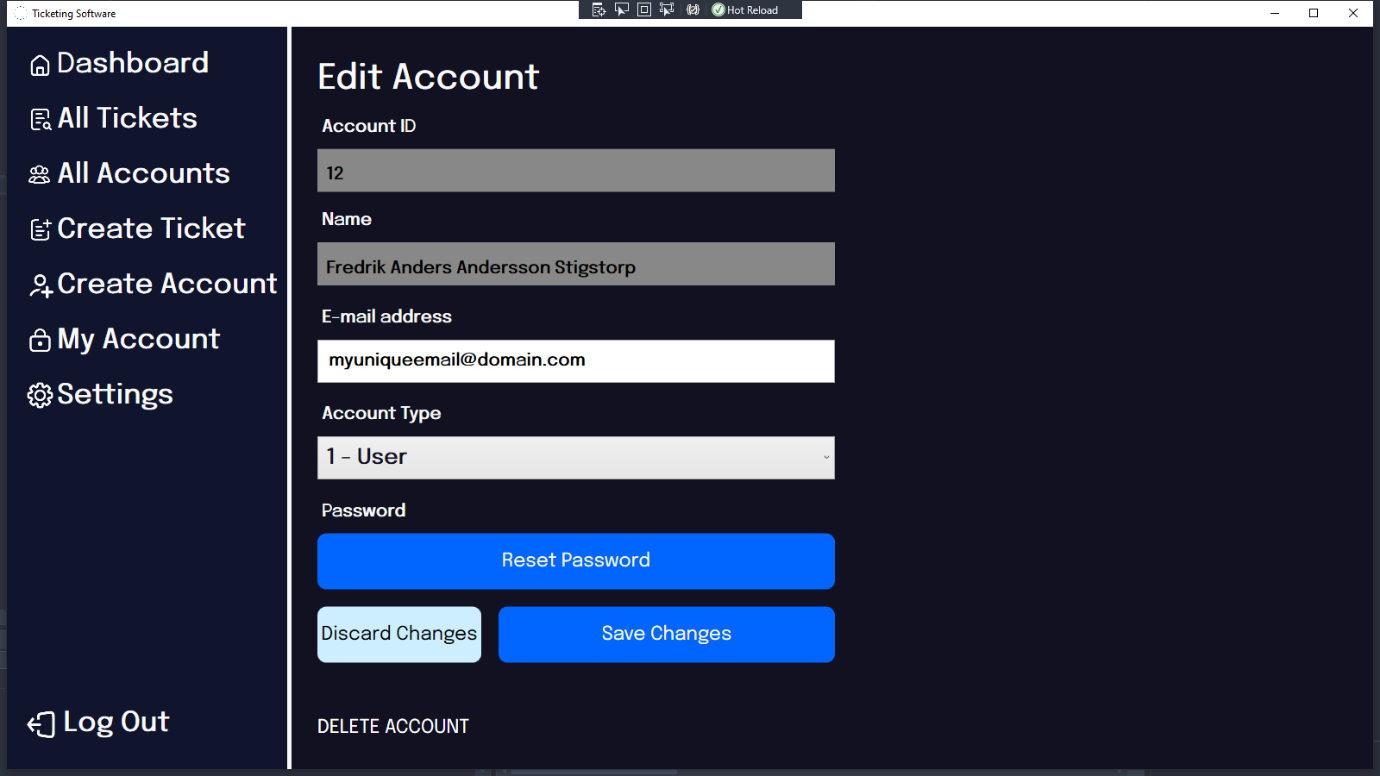


Figure 19: Functional Testing #6

**Result**

Passed

1. **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**

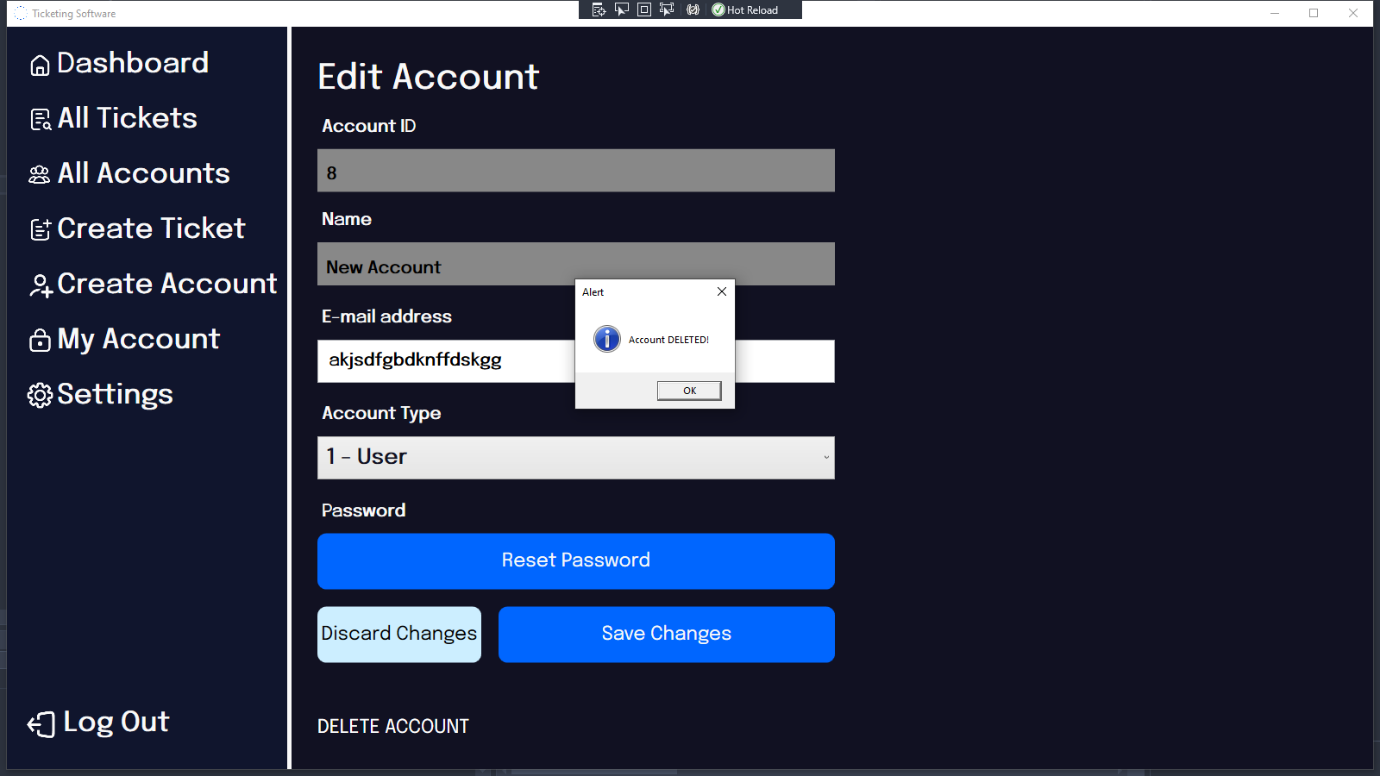


Figure 20: Functional Testing #7

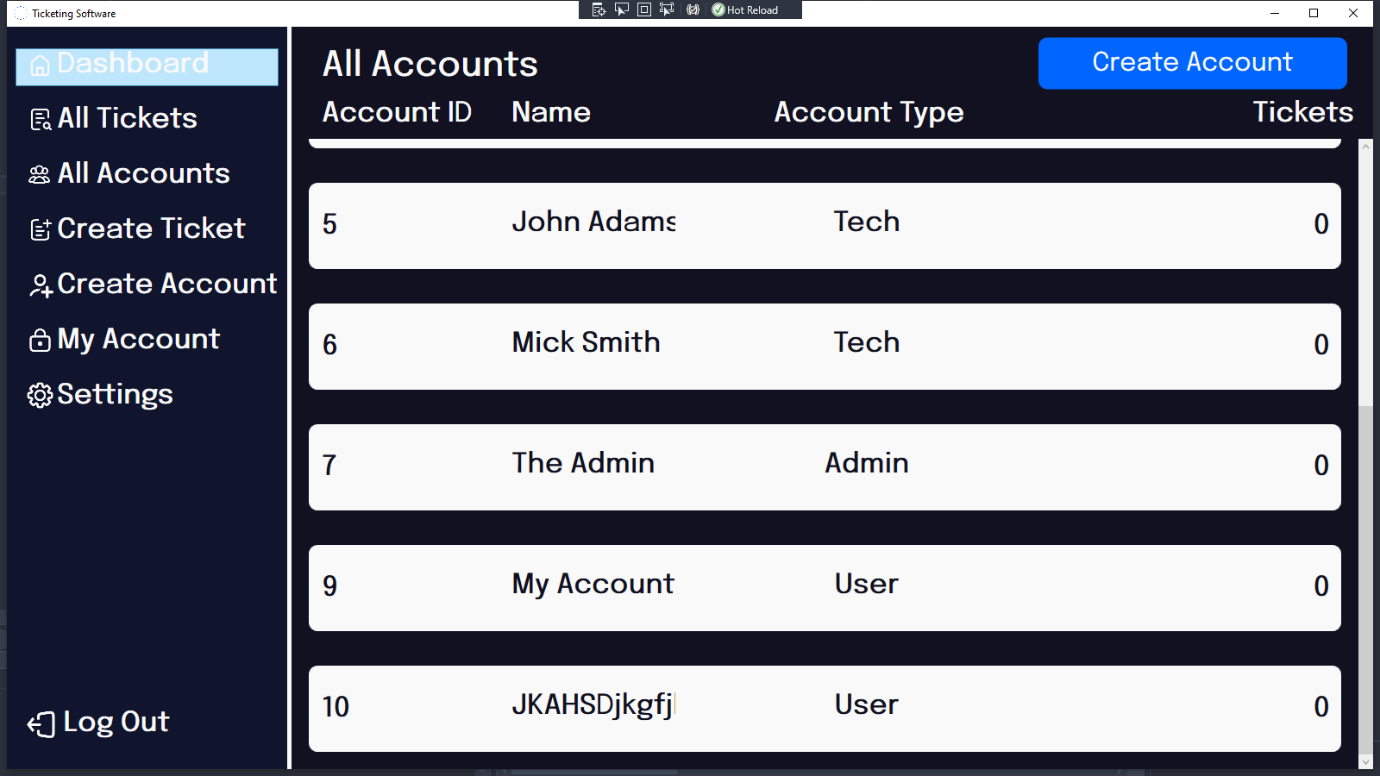


Figure 21: Functional Testing #8

**Result**

Passed

1. **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](mailto:lisbeth@olsson.se)

Account type: 1 (user)

Password: 123

**Received output**

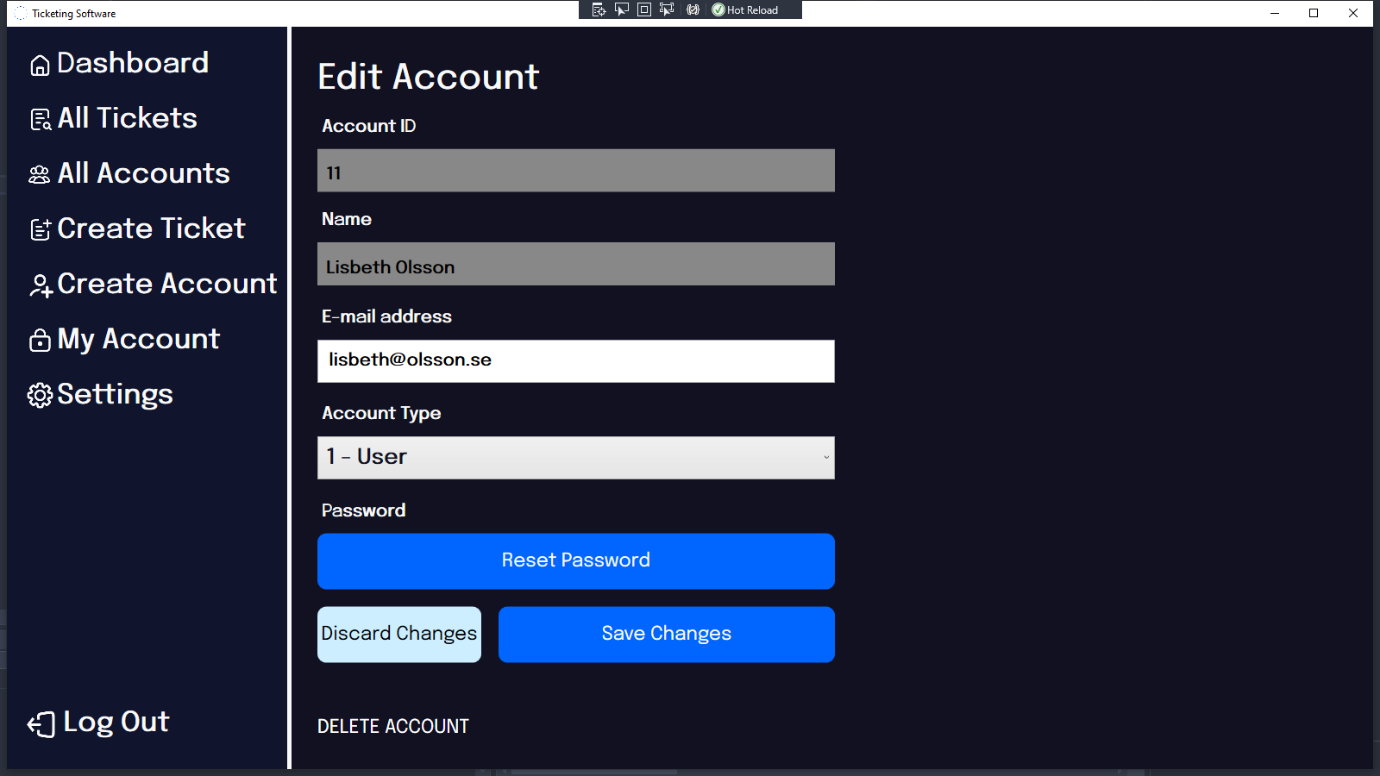


Figure 22: Functional Testing #9

**Result**

Passed

1. **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**

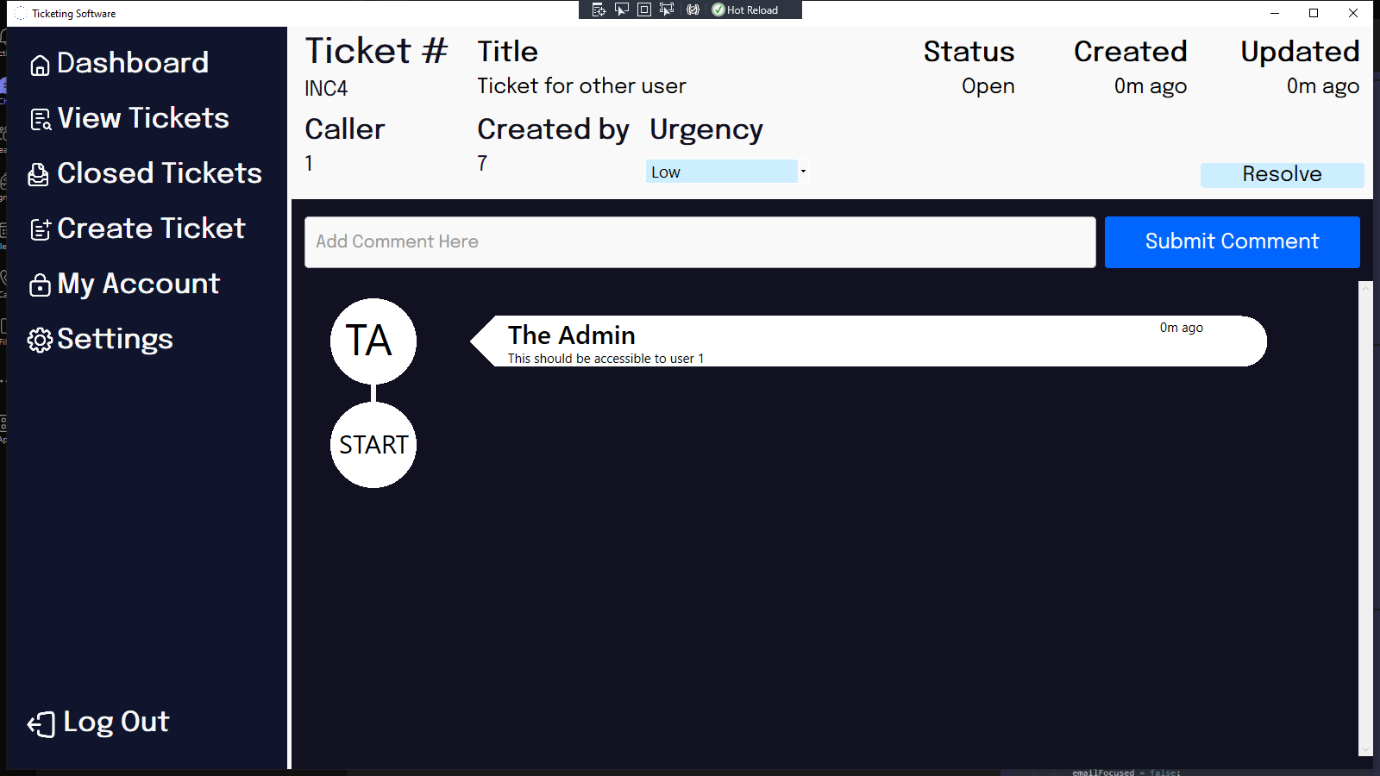


Figure 23: Functional Testing #10

**Result**

Passed

1. **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**

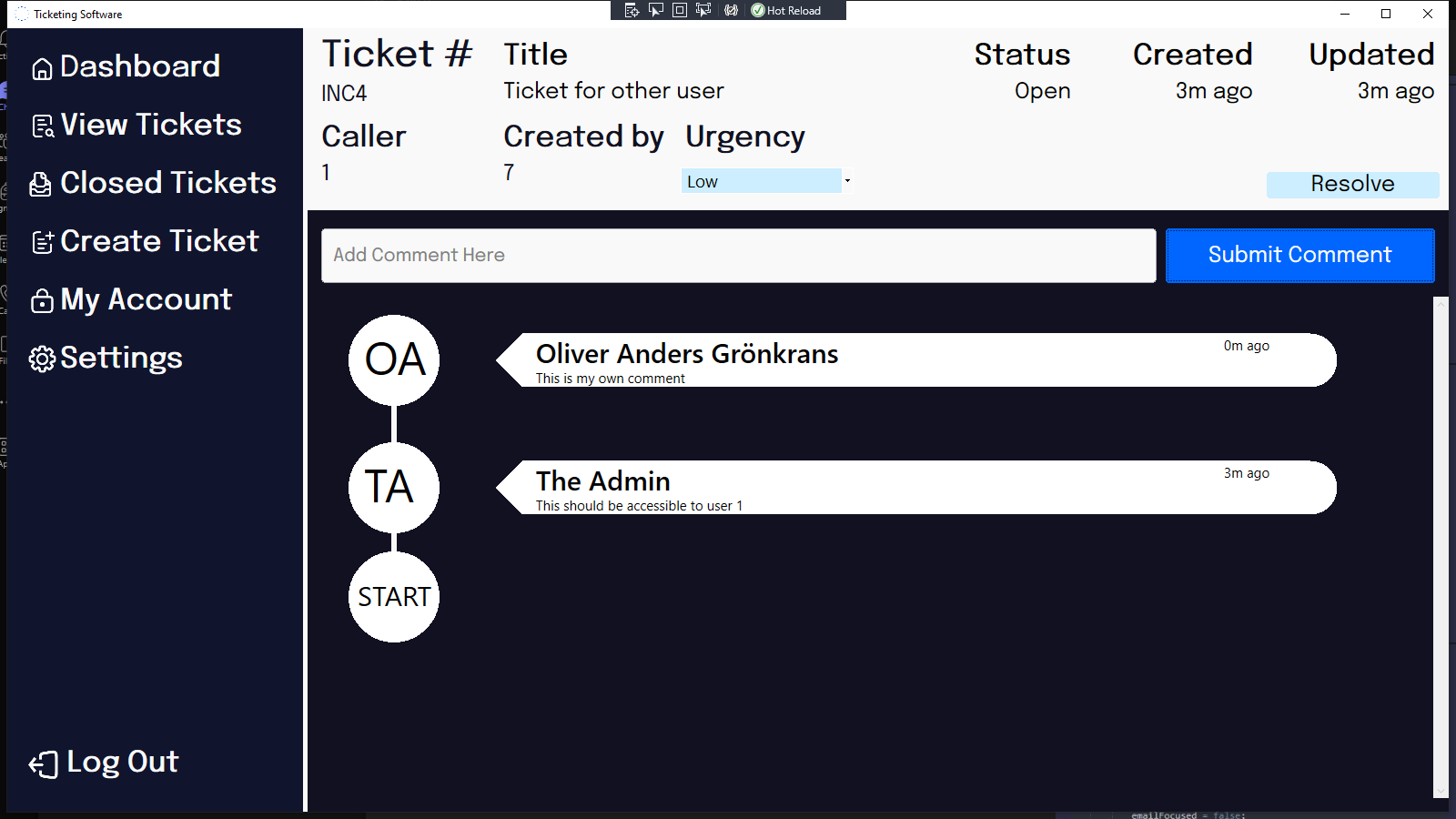


Figure 24: Functional Testing #11

**Result**

Passed

# User Documentation

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

## Installation Guide

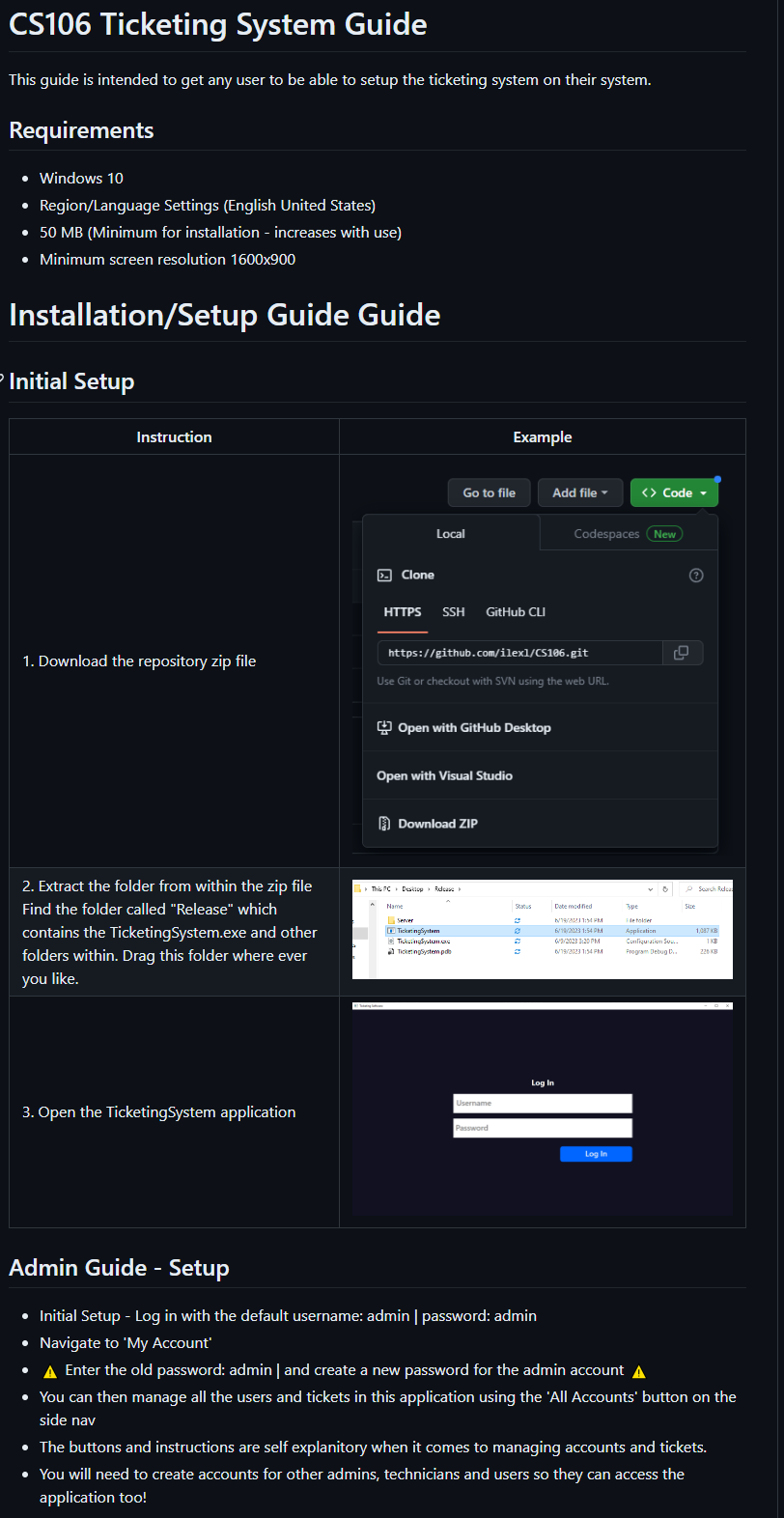


Figure 25: Installation guide

## User Guide

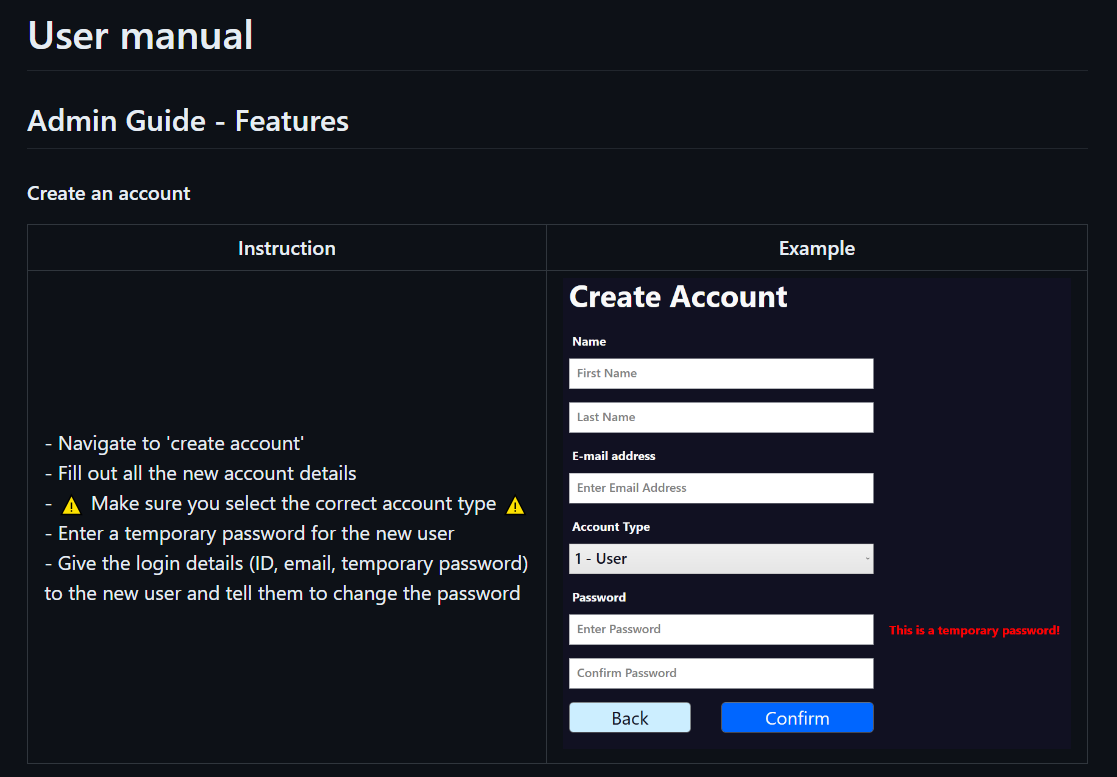


Figure 26: User Guide #1

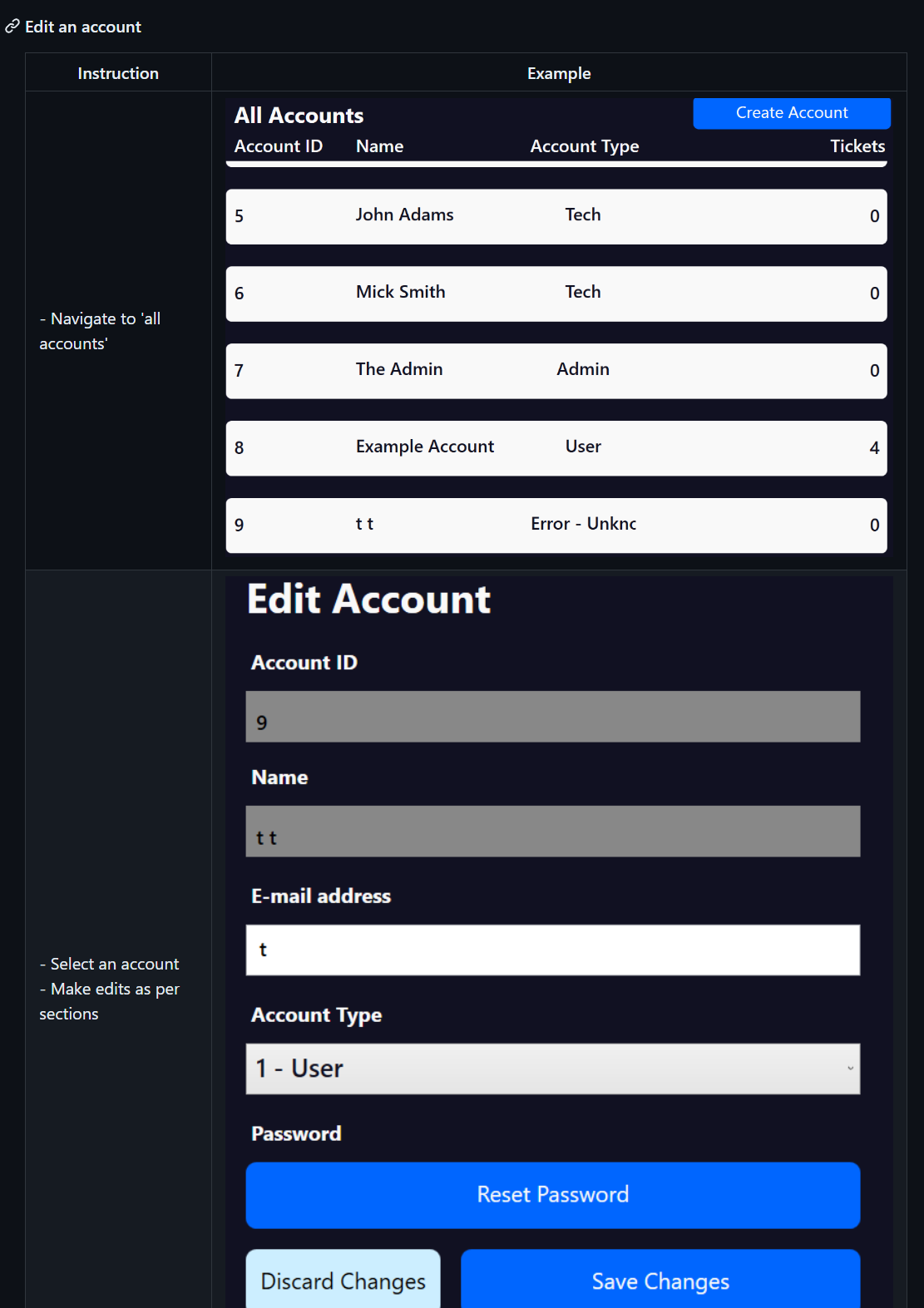


Figure 27: User Guide #2

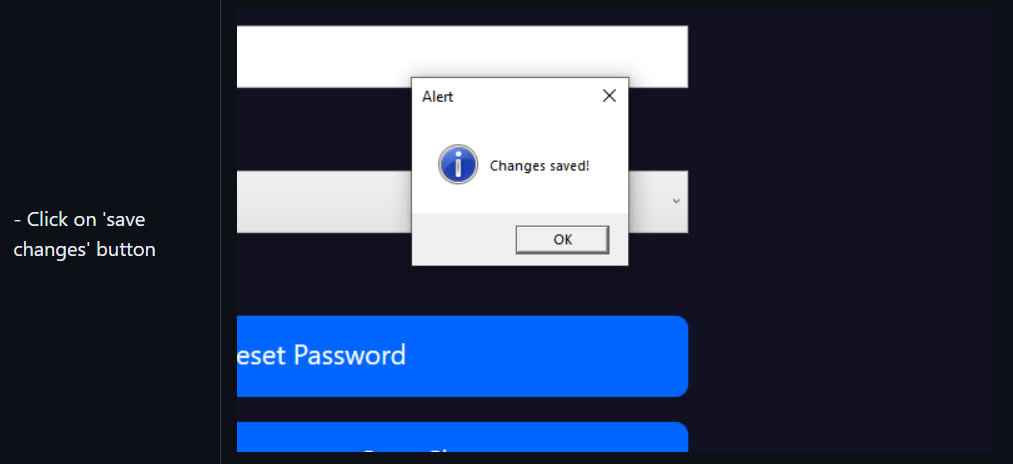


Figure 28: User Guide #3

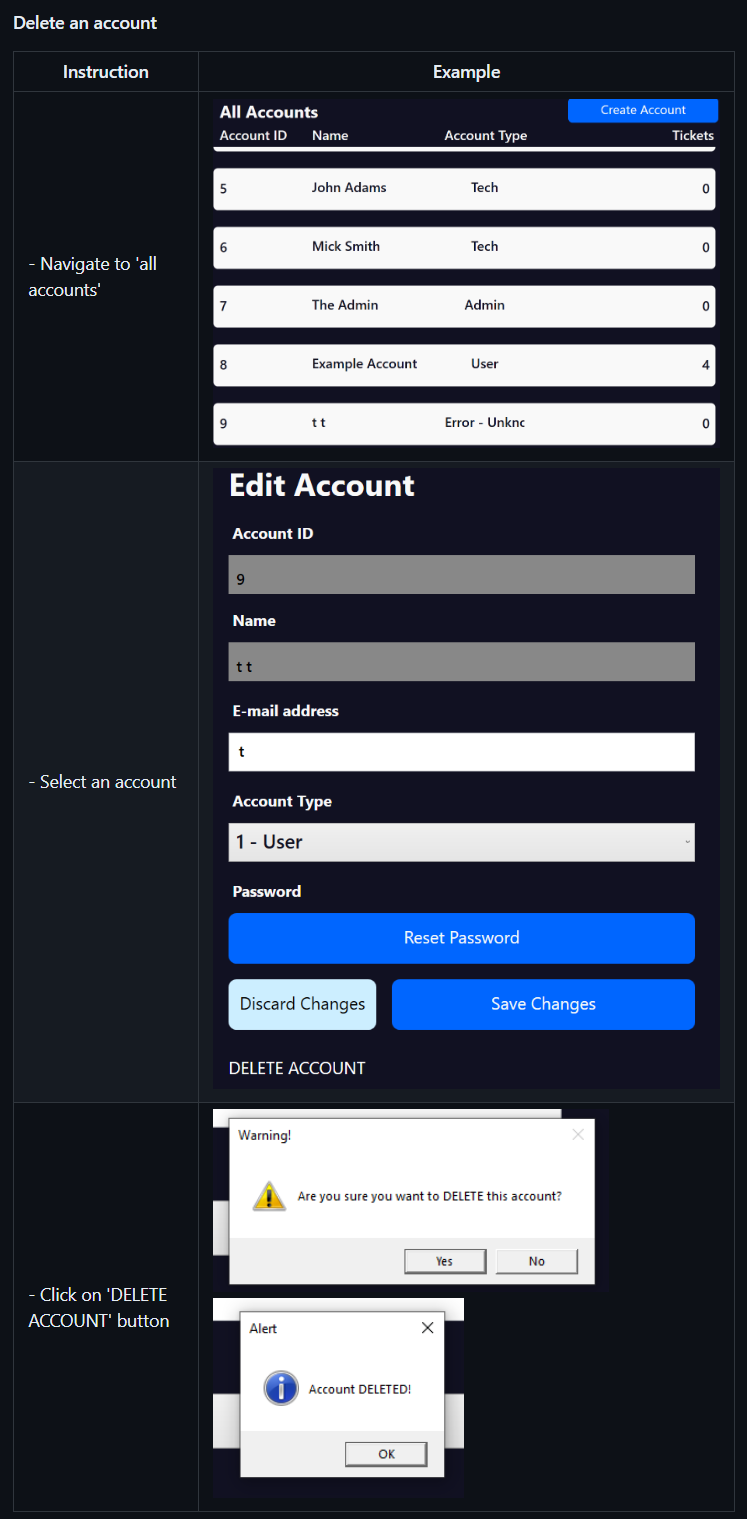


Figure 29: User Guide #4

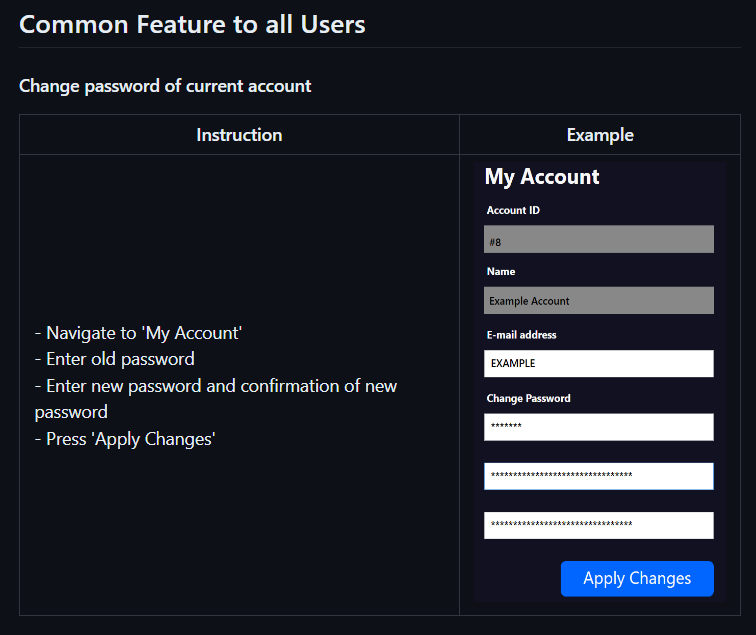


Figure 30: User Guide #5

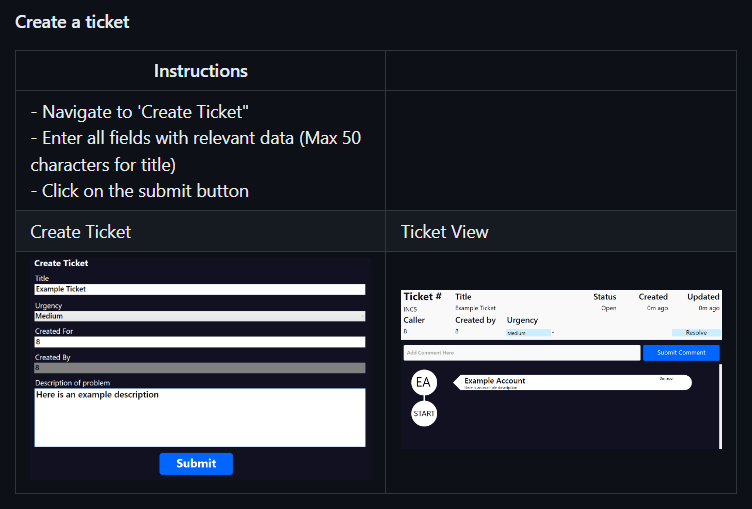


Figure 31: User Guide #6

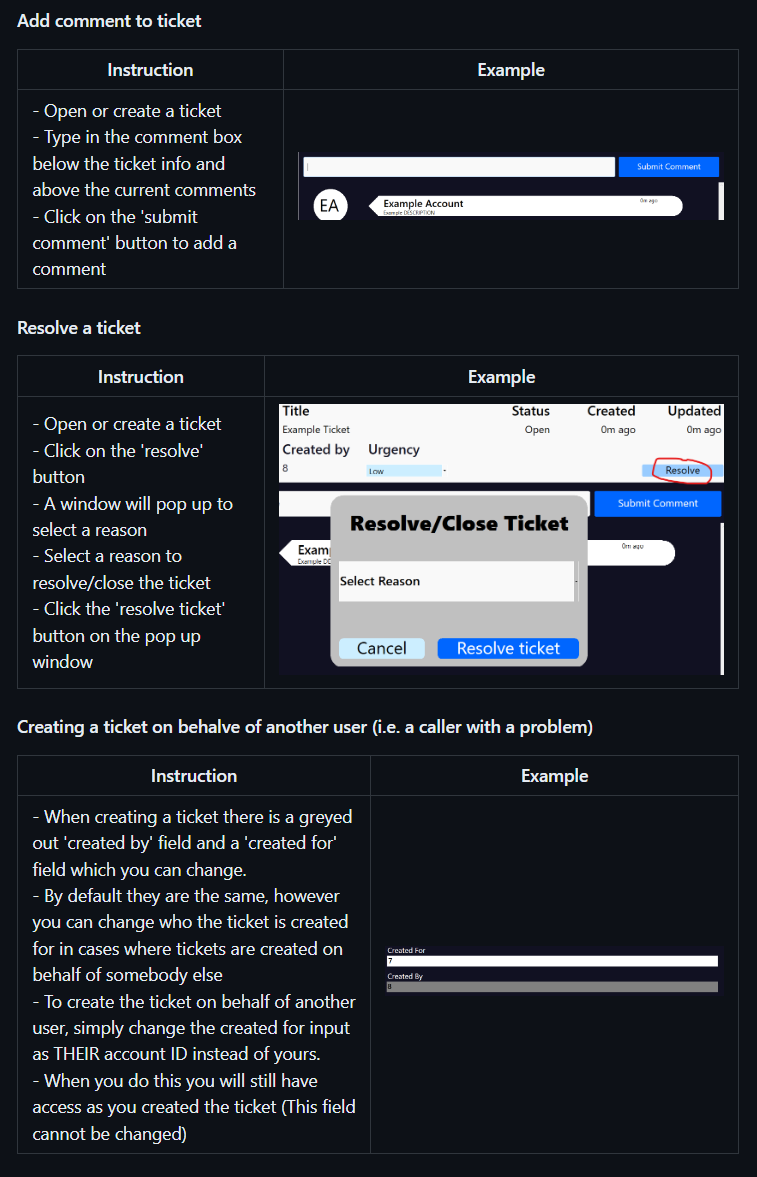


Figure 32: User Guide #7