

SPRINT #2

SCRUM GANG

GOALS

Branding

Storing Data in the Database

Separate Admin and User Pages

Remember Me Functionality

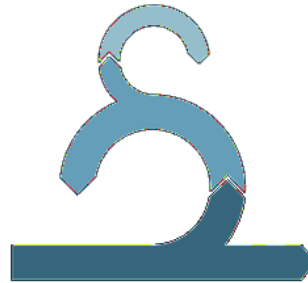
Documentation (Naming Conventions)

ACHIEVED

FRONT END

Branding

- Documented with Trello
- Fonts:
 - Headers - Microsoft Tai Le (Bold / Oblique)
 - Form text - Default Microsoft Font
- Colors:
 - Dark Background - Steel Blue
 - Form - Light Steel Blue



Forgot Password

Enter Email:

CLEAR ON INITIALIZE

Confirm

Question

CLEAR ON INITIALIZE

Submit

New Password:

Confirm

Already have an account?

SCRUM INSURANCE

*Please log in using your
SCRUM INSURANCE username and password*

Username:

Password:

Log In

[Forgot Password?](#) [Create Account](#)

Adding Accounts to the Database

Create Account

Back

Username

Password

Email

Security Question

Answer

Create Account

```
1 reference
public bool addAccount(string username, string password, string email, string question, string answer, string role)
{
    string[] args = new string[6];
    args[0] = username;
    args[1] = password;
    args[2] = email;
    args[3] = question;
    args[4] = answer;
    args[5] = role;

    //will verify that it went through. if it doesnt go through, will return false.
    if (myConnection_.insertQuery("login", args))
    {
        return true;
    }
    else
    {
        return false;
    }
}

private DatabaseConnection myConnection_;
private string username_;
private string password_;
}
```

Landing Account Support Submit Request Settings keegan Log Out

Welcome, keegan

What would you like to do?

Make a claim

Recieve you funds

Contact your claim manager

Query functions

Insert Query

- Abstract (can pass in any amount of args)
- Passes in using parameters
- Returns true / false based on if the item was inserted

Select Query

- Still in process of making abstract
- Returns a query result

```
public bool insertQuery(string tableName, string[] args)
{
    // Open SQL connection for queries
    if (!openConnection())
    {
        // Return false when connection fails
        return false;
    }

    // Initialize command to query database
    Command = Connection.CreateCommand();
    // INSERT INTO login VALUES("admin", "admin1234");
    // Assign query to command and insert args as parameters
    Command.CommandText = "INSERT INTO " + tableName + " VALUES( ";
    for (int i = 0; i < args.Length; i++)
    {
        //set up this way to make it as abstract as possible.
        Command.CommandText = Command.CommandText + "@" + i; // we have to do @i, because text can contain @'s (for emails)
        // and to pass it in as a string we need to add it via the parameter function.

        if (i < args.Length - 1)
        {
            Command.CommandText = Command.CommandText + ", ";
        }
    }
    Command.CommandText = Command.CommandText + ")";

    for (int i = 0; i < args.Length; i++)
    {
        string paramNum = "@" + i;
        Command.Parameters.AddWithValue(paramNum, args[i]);
    }
}
```

Separate Admin and User Pages

```
1 reference
private void btnLogin_Click(object sender, EventArgs e)
{
    Console.WriteLine("User (" + txtUsername.Text + ") Pass (" + txtPassword.Text + ")");
    // If username and password text is found in database (valid login)
    // if it is not found null was returned, not a string array
    string[] usernameAndRole = session_.getDbController().validateLogin(txtUsername.Text, txtPassword.Text);
    if (usernameAndRole != null)
    {
        session_.Username = usernameAndRole[0];
        session_.Role = usernameAndRole[1];
        // Load landing page, admins go to admin page, clients go to new client page, decided by role column in database
        if (session_.Role.Equals("admin"))
        {
            this.swapControl(new adminLanding(session_));
        }
        else
        {
            this.swapControl(new ctrlLandingClient(session_));
        }
    }
    // Error if account with correct username & password isn't found
    else
    {
        lblLoginError.Text = "Incorrect username or password";
    }
}
}
```

Client Landing Page

[Landing](#)[Account](#)[Support](#)[Submit Request](#)[Settings](#)[testing](#)[Log Out](#)

Welcome,

What would you like to do?

Make a claim

Recieve your funds

Contact your claim manager

Admin Landing Page

Enter user to display:

Enter

	Username	Password	Email	Question
*				

Functionality: Can look up all users and display information

Session Class

```
namespace ScrumInsurance
{
    9 references
    public class Session
    {
        private List<Account> accounts_;
        private DatabaseController dbController_;

        5 references
        public DatabaseController getDbController() { return dbController_; }

        //Eventually, session should save an account, with all the current user's info from database
        4 references
        public string Username { get; set; }
        2 references
        public string Role { get; set; }

        0 references
        public void addAccount(string username, string password, string email, string securityQuestion, string securityQuestionAnswer)
        {
            accounts_.Add(new Account(username, password, email, securityQuestion, securityQuestionAnswer));
        }

        //Returns index of first account in list of accounts with specified username + password, returns -1 otherwise
        0 references
        public int findAccount(string username, string password)
        {
            for (int i = 0; i < accounts_.Count; i++)
            {
                if (accounts_[i].validCreds(username, password))
                {
                    return i;
                }
            }
            return -1;
        }
    }
}
```

```
//Returns index of first account in list of accounts with specified email, returns -1 otherwise
0 references
public int findAccount(string email)
{
    for (int i = 0; i < accounts_.Count; i++)
    {
        if (accounts_[i].validCreds(email))
        {
            return i;
        }
    }
    return -1;
}

//Returns security question of first account with specified email, returns null otherwise
0 references
public string findQuestion(string email)
{
    for (int i = 0; i < accounts_.Count; i++)
    {
        if (accounts_[i].validCreds(email))
        {
            return accounts_[i].SecurityQuestion;
        }
    }
    return null;
}

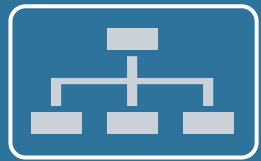
0 references
public Account getAccount(int index)
{
    return accounts_[index];
}
}
```

BACK END

ICMS provides services for four kinds of users:



Clients



Administrator



Client/Claim Manager



Finance Manager

10.3.3 Use Case Diagram

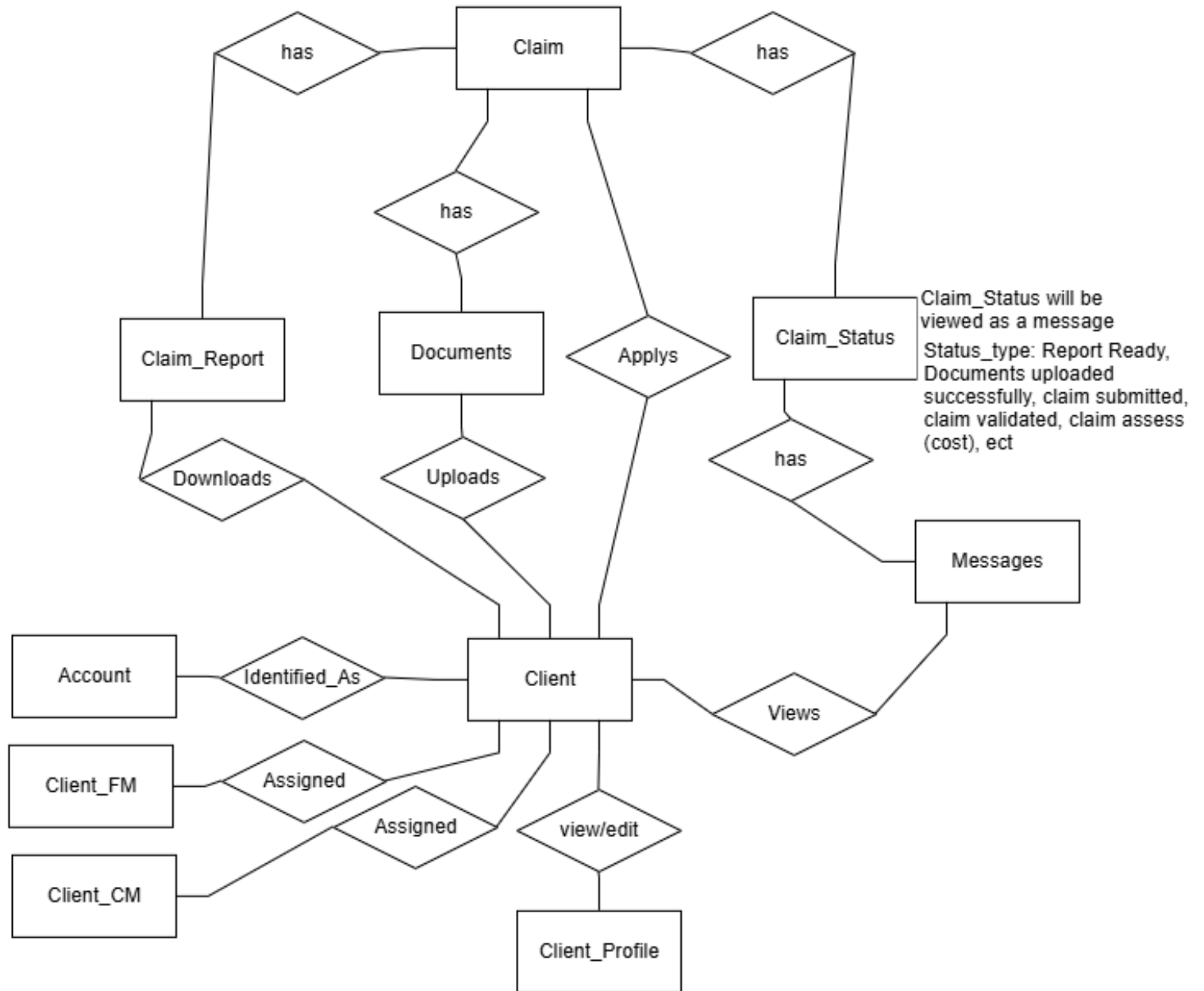
Primary Use Case Diagram

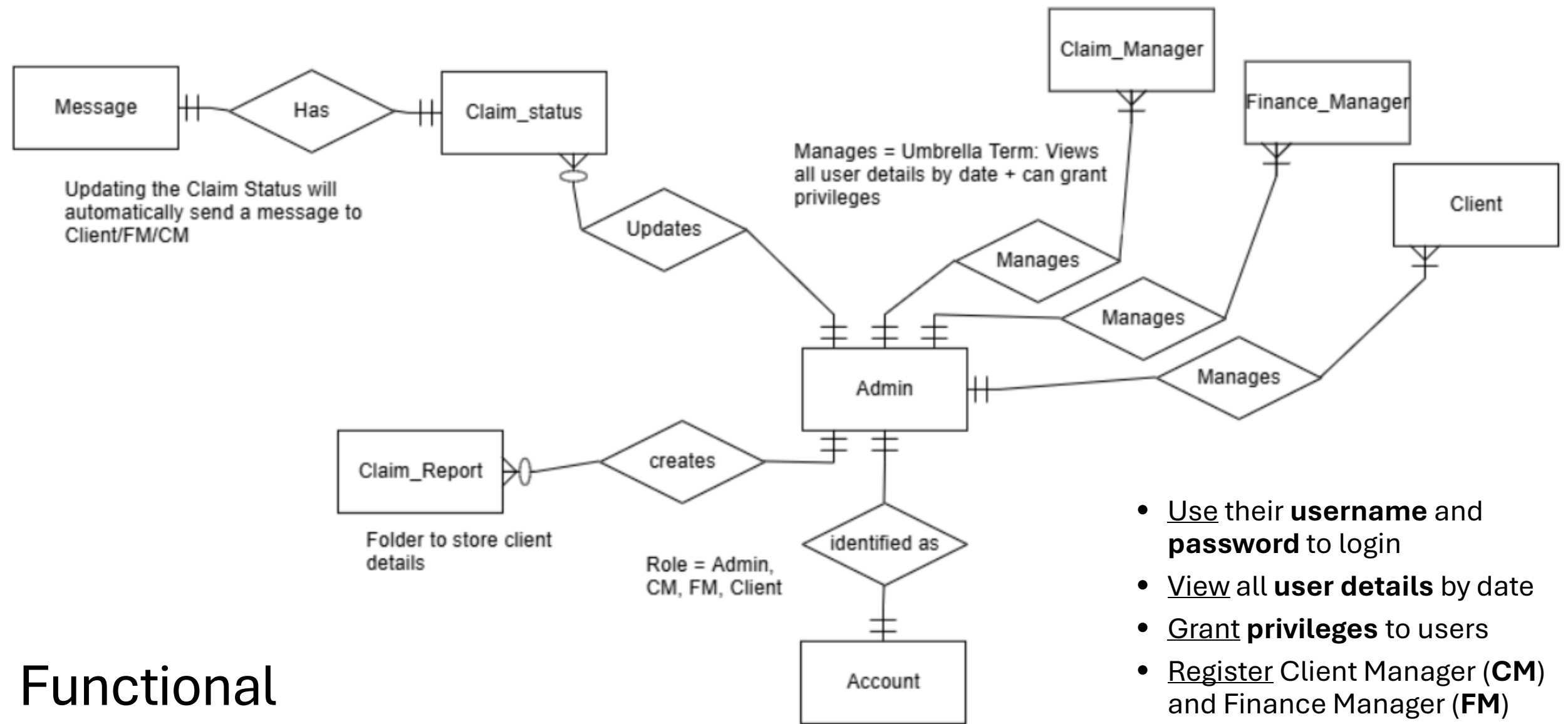


Fig. 10.37 Insurance Claim Management–Primary use case diagram

Functional Requirements: **Client**

- Apply for a **claim**
- Use their **username** and **password** to login
- View the **messages** sent by CM, FM, Admin
- Upload scanned **documents**
- View/edit their **profiles**
- View current **status**
- Download their claim **reports**

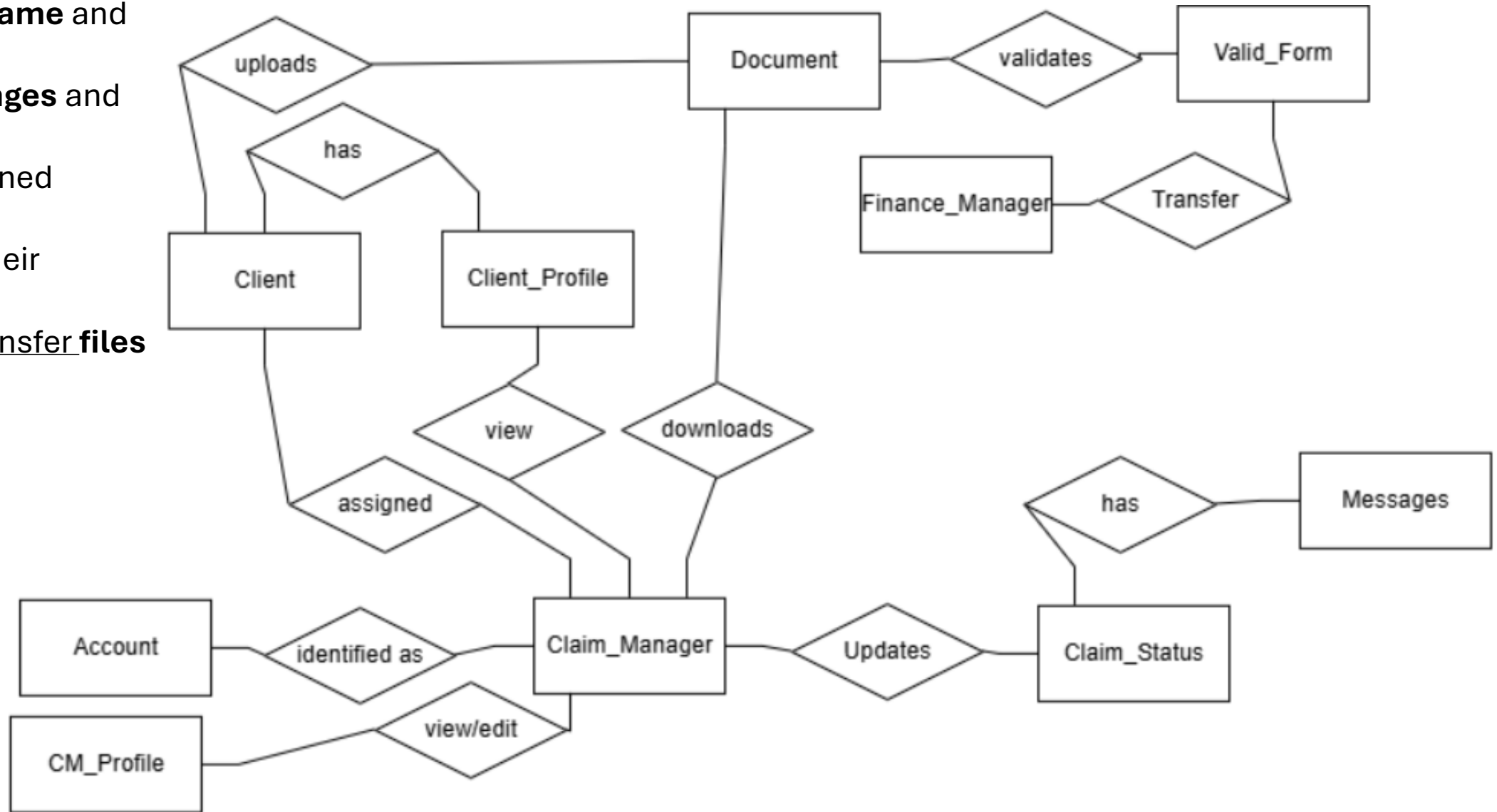




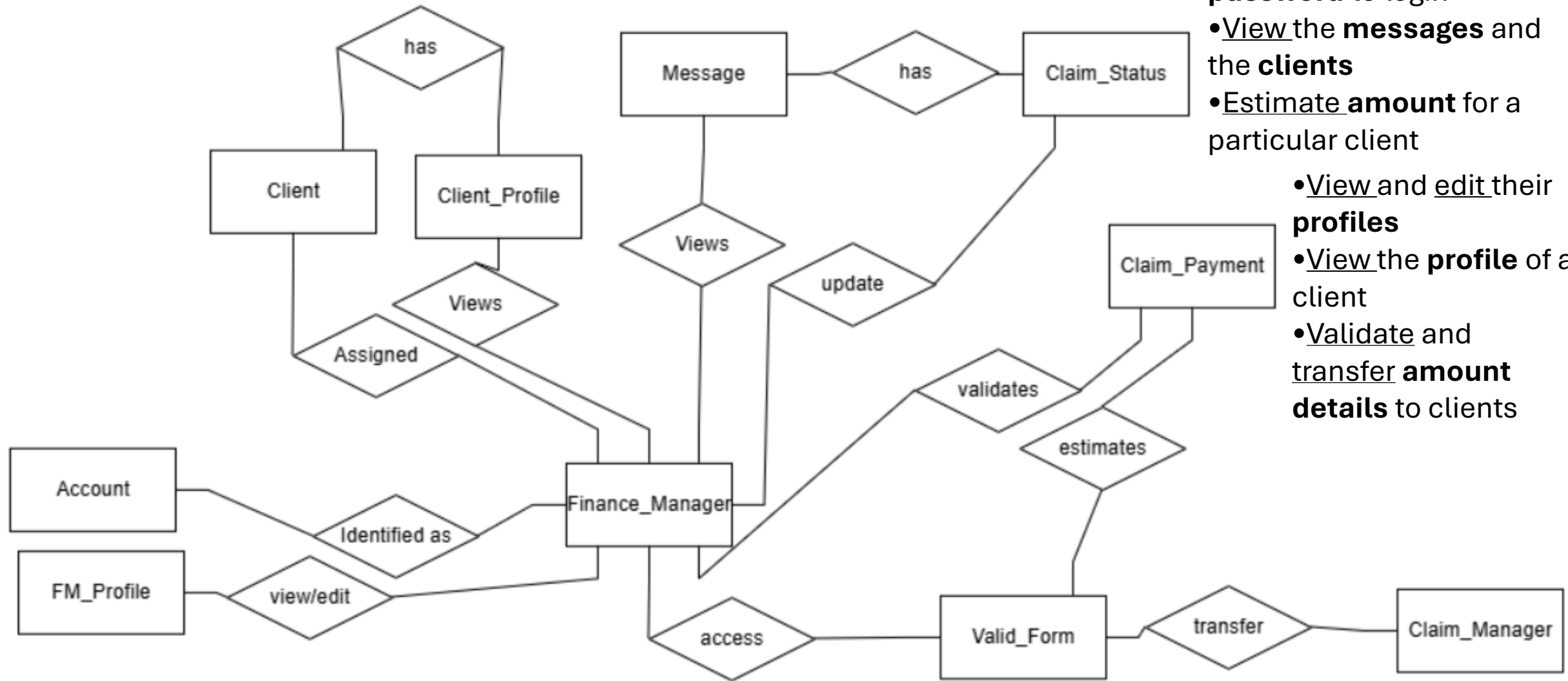
- Use their **username** and **password** to login
- View all **user details** by date
- Grant **privileges** to users
- Register Client Manager (**CM**) and Finance Manager (**FM**)
- Create a **Folder** for each client to store their details

Functional Requirements: Admin

- Use their **username** and **password** login
- View the **messages** and the **clients**
- Download scanned **documents**
- View and edit their **profiles**
- Validate and transfer **files** to FM

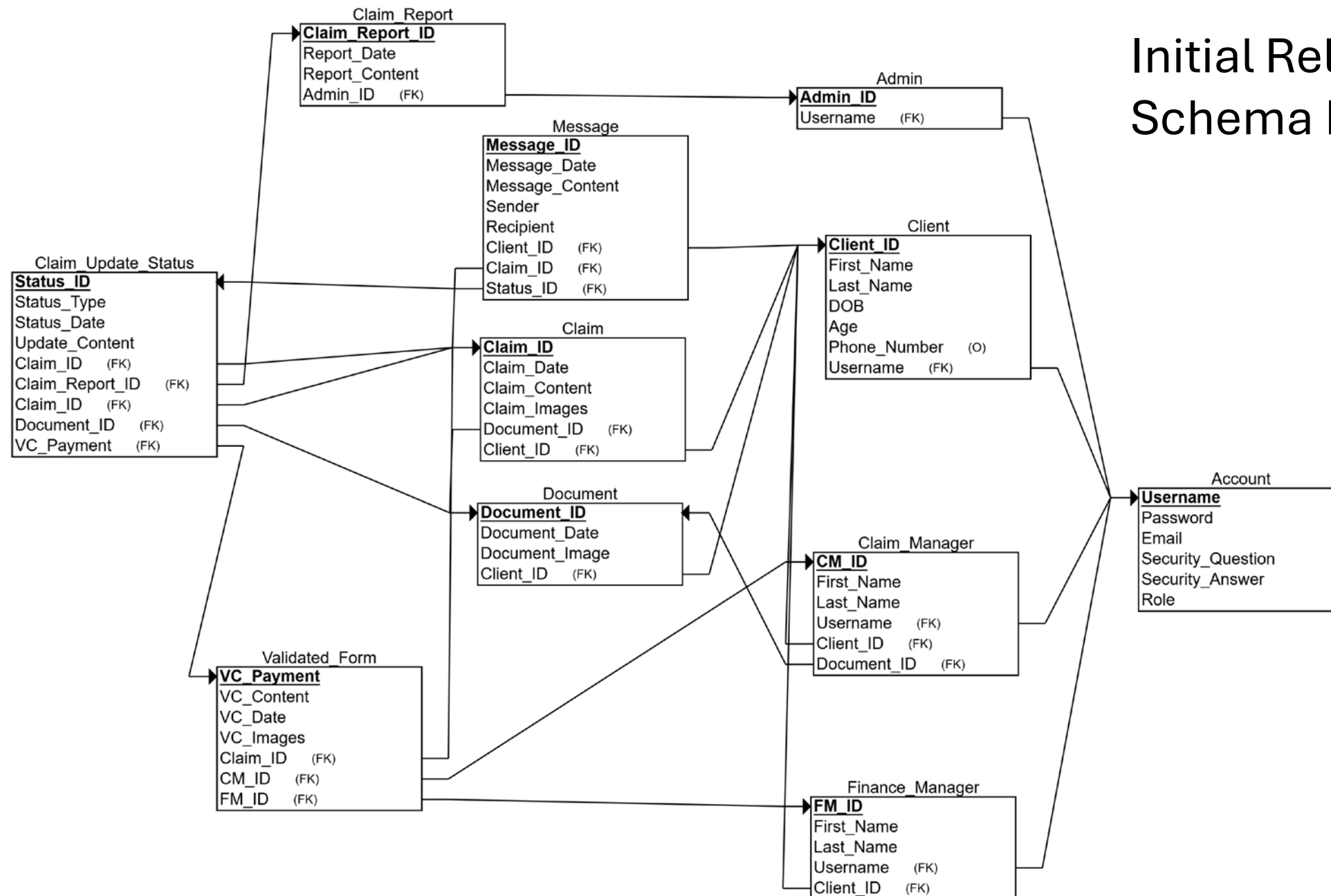


Functional Requirements: **Claim Manager (CM)**



Functional Requirements: **Finance Manager (FM)**

Initial Relational Schema Design



Database Connection

```
1 reference
public bool insertQuery(string tableName, string[] args)
{
    // Open SQL connection for queries
    if (!openConnection())
    {
        // Return false when connection fails
        return false;
    }

    // Initialize command to query database
    Command = Connection.CreateCommand();
    // INSERT INTO login VALUES("admin", "admin1234");
    // Assign query to command and insert args as parameters
    Command.CommandText = "INSERT INTO " + tableName + " VALUES( " ;
    for (int i = 0; i < args.Length; i++)
    {
        //set up this way to make it as abstract as possible.
        Command.CommandText = Command.CommandText + "@" + i; // we have to do @i, because text can contain @'s (for emails)
        // and to pass it in as a string we need to add it via the parameter function.
        if (i < args.Length - 1)
        {
            Command.CommandText = Command.CommandText + ", " ;
        }
    }
    Command.CommandText = Command.CommandText + ")";

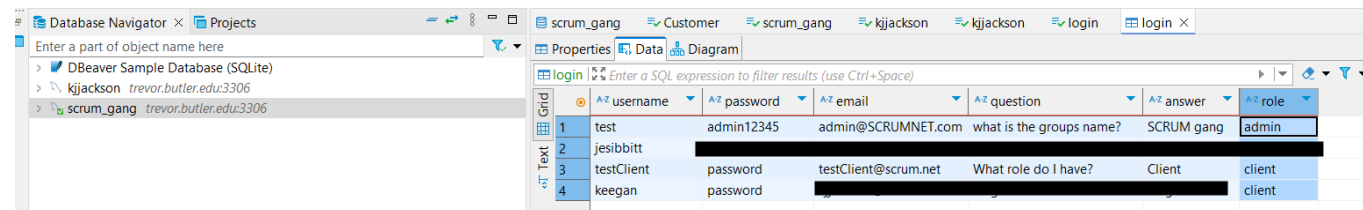
    for (int i = 0; i < args.Length; i++)
    {
        string paramNum = "@" + i;
        Command.Parameters.AddWithValue(paramNum, args[i]);
    }
}
```

```
Console.WriteLine(Command.CommandText);

// Execute command and store returned data
Reader = Command.ExecuteReader();

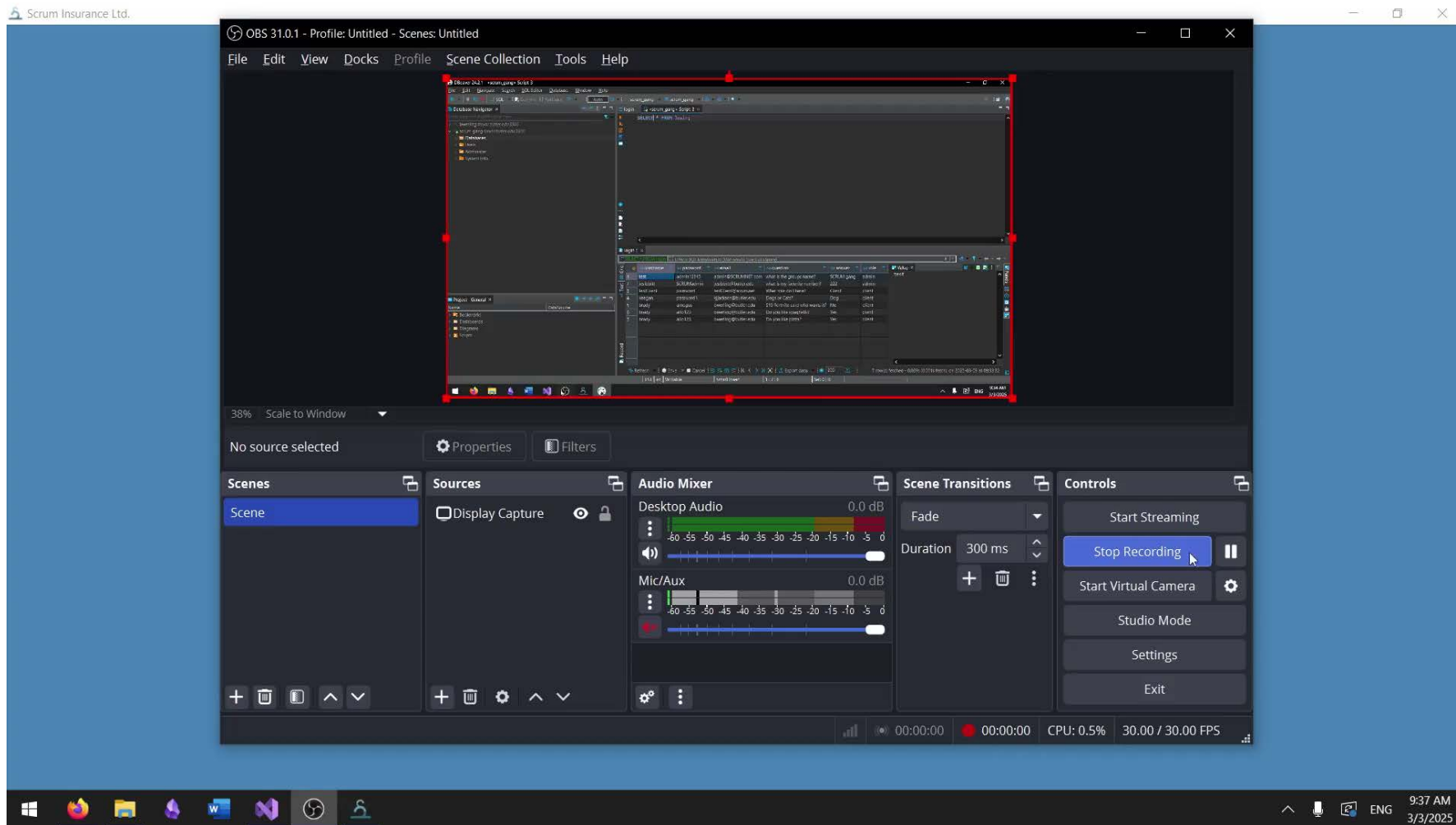
// If reader has data (matching user and password were found)
if (Reader.HasRows)
{
    //Reads every matching row
    while (Reader.Read())
    {
        //Returns username (0th column) and role (5th column) to be used by login and session
        //Uses password (1st column) temporarily find into correct account
        string[] usernamePasswordRole = new string[] { Reader.GetString(0), Reader.GetString(1), Reader.GetString(5) };
        if (usernamePasswordRole[0].Equals(args[1]) && usernamePasswordRole[1].Equals(args[3]))
        {
            printData(Reader);
            closeConnection();
            return new string[] { usernamePasswordRole[0], usernamePasswordRole[2] };
        }
    }
    Console.WriteLine("Reader has no matching rows, this should never happen");
    closeConnection();
    return null;
}

// Else no matching data was found (invalid username or password)
else
{
    Console.WriteLine("No matching data");
    closeConnection();
    return null;
}
```



	username	password	email	question	answer	role
1	test	admin12345	admin@SCRUMNET.com	what is the groups name?	SCRUM gang	admin
2	jesibbitt					
3	testClient	password	testClient@scrum.net	What role do I have?	Client	client
4	keegan	password				client

Demo



OBSTACLES

Designing the
Database
Structure

Centering
Layout on
forms (UI)

FUTURE PLANS

Finalizing
Database
Organization

Adding more
queries

Quality of life
updates

Remember
Me

Code
Cleanup



QUESTIONS?