

# TRENT MESSENGER CHAT APP

A Messaging Chat App built using .NET Framework

## ABSTRACT

This Module of the Trent Messenger App was built to provide basic chat functionality using excellently designed UML Design Models, implemented on .NET using AWS Data-tier Provisioning.

Tim Olatunji (N0887674)

SOFTWARE ENGINEERING 2

## TABLE OF CONTENTS

SYSTEM DESCRIPTION.....	2
Defining Trent Messenger Application Requirements.....	2
Opportunities for Extensions and Improvements .....	3
COHESION, COUPLING AND EVENT-DRIVEN DESIGN IMPLEMENTATIONS .....	3
Code Organization .....	3
CLASS DIAGRAM .....	<b>Error! Bookmark not defined.</b>
SEQUENCE DIAGRAM .....	<b>Error! Bookmark not defined.</b>
STATE DIGRAM .....	<b>Error! Bookmark not defined.</b>
COMPONENT DIAGRAM .....	<b>Error! Bookmark not defined.</b>
DEPLOYMENT DIAGRAMS.....	<b>Error! Bookmark not defined.</b>
DATABASE IMPLEMENTATIONS ( <i>HIGHEST DIFFICULTY</i> ) .....	5
Selecting a Relational Database Set-Up and Service .....	5
Internal Structures.....	7
Implemented SQL Algorithm .....	<b>Error! Bookmark not defined.</b>
The Table Structure of Database .....	9
TRENTS AUTOS MESSENGER DOCUMENTATION.....	10
Login .....	10
Account registration .....	11
General Settings: .....	12
Add Organization Contacts .....	13
Account Setting: Setting a new Password, Changing Security Questions and Responses.....	14
About This App .....	14
Delete Account .....	14
Application Settings .....	14
Chats.....	<b>Error! Bookmark not defined.</b>

## SYSTEM DESCRIPTION

In Building this App I explored diverse Implementational Approaches and styles to building a Server-Multiclient Chat System, I researched and considered strategically the following developmental frameworks:

1. A Web Chat App with SignalR and ReactJS or ASP.NET MVC Architecture.
2. LAN-Networked Chat App using Serverless UDP infrastructure, where the client device serves as the Unique ID.
3. Socket Programming using TCP leveraging on the Client-Server deployment (with the socket, bind, listen, accept, receive, send and close Implementations), directly implementing the TCP Client and IPAddress Class Libraries.
4. I favoured using Windows Forms Application using a more recent Standard SQL Connection Library, which I found very convenient to implementation, proving to be operationally efficient for a Server-based TCP/IP Connection using a connection string.

My choice for Implementation Framework was based on the time-efficiency of implementation and Functional Requirements of this Project. The Web Chat using either ASP.NET MVC Architecture reinforced with Web Services will consume significant Logic and Time. Additionally, the TCP Client-Server Implementation Design will require a lot of Asynchronous Programming – Background Worker or Task Based Asynchronous Pattern – and Multi-threading Programming. Furthermore, LAN-Networked UDP has severe usability concerns and restraints, namely the insecure serverless architecture and its intrinsic poor information management downside.

---

## DEFINING TRENT MESSENGER APPLICATION REQUIREMENTS

1. The Chat App must allow User Authentication.
2. The Chat App must allow the registration of the following user details: User Profile Image, Username, User Password, User Phone Number, Security Question, Security Responses.
3. Chat App must allow User to View and Add Employee Contacts from the Company Employee Contact Book.
4. Chat App must allow Chats with Contacts: Audio and Smiley Chats.
5. User must be able to adjust Application Settings Colour, namely Sender and Receiver Text Colour, Chat Bubble Appearance, Text Box Appearance, etc.
6. Chat App must allow User to modify previously created account details.
7. Chat App must display the state of message sent, read or unread.
8. Chat App must display "Last Seen Online" time.
9. Chat App must display a count of all messages sent by a peer, involved in a chat.
10. Chat App must allow the Deletion of Account.
11. Must Allow Recovery of User Authentication Credentials.
12. Chat App must notify User of the status of task initiated above
13. Must be built using C# .NET Framework, running on Windows OS.
14. Chat App must be Connected to Amazon Web Services Relational Database Services.
15. Chat App must use TCP/IP using SQL-Client Connection.
16. Chat App must connect to the SQL Server with a non-owner security role.
17. Chat App must allow user to Search Company Contact List, looking up strings such as Username, Phone Numbers and Position Description of the User.

18. Chat App must be built using Event-Driven Frameworks namely Windows Forms Application and Event Driven Design Pattern.

---

## OPPORTUNITIES FOR EXTENSIONS AND IMPROVEMENTS

This App has immense potential for improvements and extensions. The following extensibility considerations are possible in view of the Usability requirements of an Enterprise Instant Messaging (EIM) Application:

- a) Add Admin Specializations and Extensions to the User Class with advanced privileges such as investigating all chats.
- b) Top notch Permissions Systems for Admin and Senior Level Officers to assign roles and privileges to staff based on Staff Role, Projects and Region.
- c) Institute Data Deletion, Group Creation and Membership Modification Permissions.
- d) Add Image, Video, Audio Conferencing and Chat Functionalities.
- e) Edit Sent Messages and View All Chat Files.
- f) Add a Plug-ins and APIs for Managing External Apps.
- g) Create Activity Feeds for Project Room, Project Charts, Timelines and other workload functionalities.
- h) Additional Security Rules can be written to strengthen the security of User Information and Database Interactions with the Client Application using Windows Authentication, Amazon IAM (Identity and Authentication Management) and other Best Practices for SQL Server Database Implementations.
- i) Implement a robust Hash Encryption for sensitive user Data like Password and Payment Details.

## COHESION, COUPLING AND EVENT-DRIVEN DESIGN IMPLEMENTATIONS

---

### CODE ORGANIZATION

Using the Event-driven Windows Forms Application, I organized my code into Forms, Controls and Classes, making the three Packages this project is made up of. The Classes Package used for the definition of the objects used across the codes. It is cohesively organized to ensure that changes to the character of each Class is made only from a single place in the code.

**The Connection Class** is a static (and without a constructor) class used to handle only SQL Connection Implementations. I achieved functional Cohesion with this class by having all its methods and members perform only SQL Connection and Database Query. Similarly, a functional cohesion design was attained with the **AppSettings**, **Accounts** and **Emoticons** Classes as the naming indicate house the properties and methods related to only to the intent of the Classes.

**The UserControls Package:** I have this package to manage all user-initiated and control functions and UI elements. The Package consists of the 7 Controls: **ChatUC** containing the Control Design for Chats and its properties including mouse events related to the ChatUC. In the same fashion, **AccountUC** (*used for other second party account users different from the first-party Static Account Class mentioned above*), **ClockUC**, **MessageUC**, **ContactUC**, **EmojiUC** are implemented to contain the design and logic for the properties related to the name assigned to it. Here also, I achieved functionally cohesive class design. However, the **TabUC** was needfully designed to operate with Logical Cohesion has the Tab Control manages the Message Control(**MessageUC**) and Contacts Control (**ContactsUC**)—both are in the Tab Control Component but as you would observe from the implementations, they point to different functions.

**The Forms Package:** All UI Forms are placed into this Package composed of a total of 10 Forms (with each Form having, and referring to, a Winform-unique combination of Design UI, a corresponding auto-generated *Design.cs* files and a linked .Cs Logic File).

The AccountProfile Form cohesively, in a functional manner, points to only the Account Profile Task. Homogeneously operating on a specific Contact, executing the *DownloadAccountData()* Method and handling the Contact Deletion Event using the *RemoveButton\_Click()* Event. The PairChat Class functionally contains all the essential implementation for a chat Activity. It uses the ContactID to instantiate its object, while member methods *DownloadContactData()* and *DownloadMessages()* uses the ContactID instance parameter to perform its task. The DownloadMessages() also couples setting the values of the *MessageUC* Properties. Other methods that work to perform the Chat Activity and other related Messaging Computations are *button\_send\_Click()*—working on the *MessageUC* to set the properties of the sent message, *ChatShown()*—displays all chats in the *panelMain.Control* Collections, *OnEmoticon\_Click()*—to handle emoticon messages. The Login Class utilizes a justifiable Temporal Cohesion being there are requisite tasks linked to the Login instance—namely, User Authentication blocks of Code and a following snippets for recalling User-specific App Settings. The Settings Class (Form and Logic) Logically contains settings related components *AboutButton\_Click()* to manage About App events, while other methods logically related but operationally different Deletion of User's Account and Account Settings. AppSettings and ContactsUpdate Class had a relatively easy implementation of a functionally cohesive design.

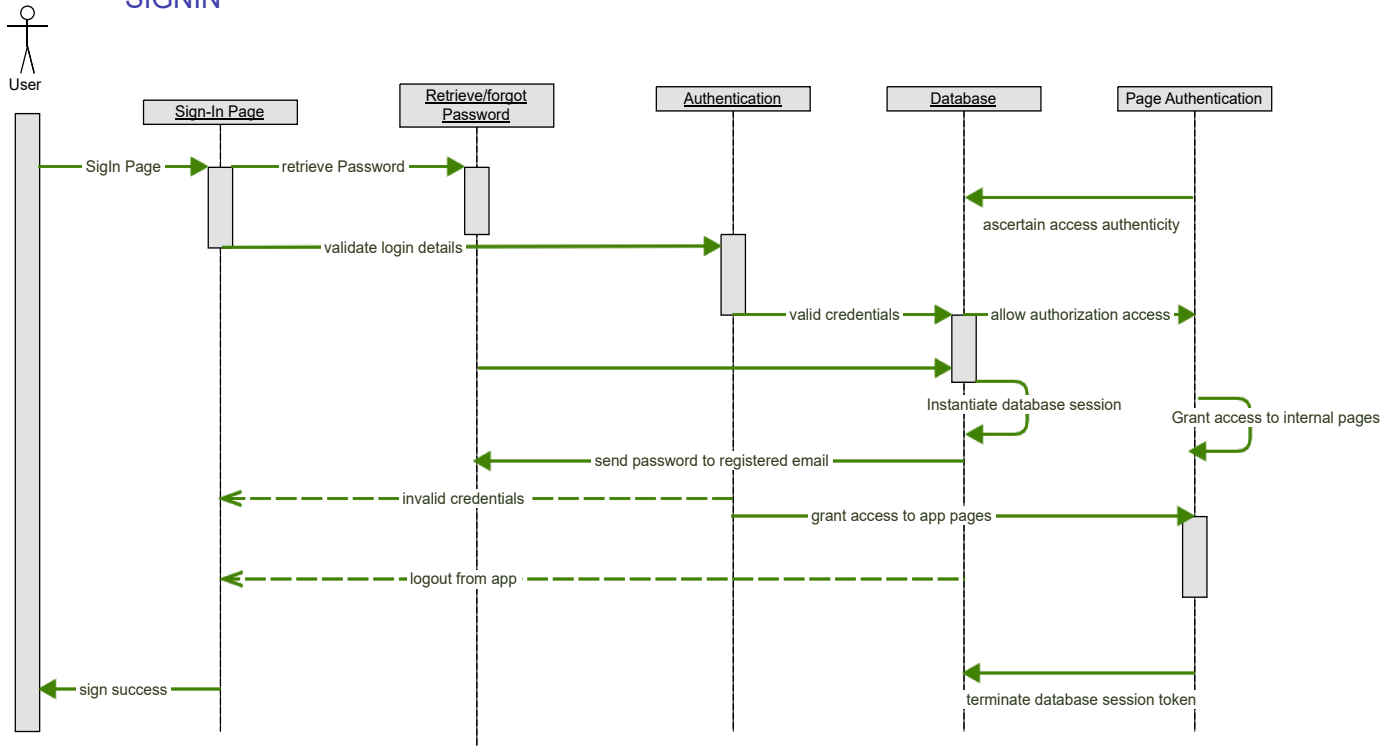
The Trent Messenger App uses extensively Stamp Coupling as you would notice in the relationships between the Forms and the UserControls. The Forms generally have access to all the properties of the Controls it has a Is-Part Composite or Has-a Aggregation Relationship with even if it might not always need them. I think this is an intrinsic nature of the WinForms Set up. Overall, a good system design was implemented using Event-Driven Design Pattern available in the WinForms Architecture.

The Trent Messenger being a simple application does not exhaustively utilize the features of a mature event-driven architecture, however, while providing opportunity for scalability and API integration into a larger heterogenous system that the Trent Autos might develop in the future, it does utilize simple event processing that implements the Event Emitters- Consumers and the TCP Event Channels model; this was established through the events fired by the Form's and Control event handlers. For every User Control feature, the WinForms Library furnishes me with a suite of very impressive Events and Properties that I can manipulate to achieve a desired UI-Event Activity and System Functionality. With this I moved from UI Design Mock-up to actual design Implementation using the WinForms Designer.

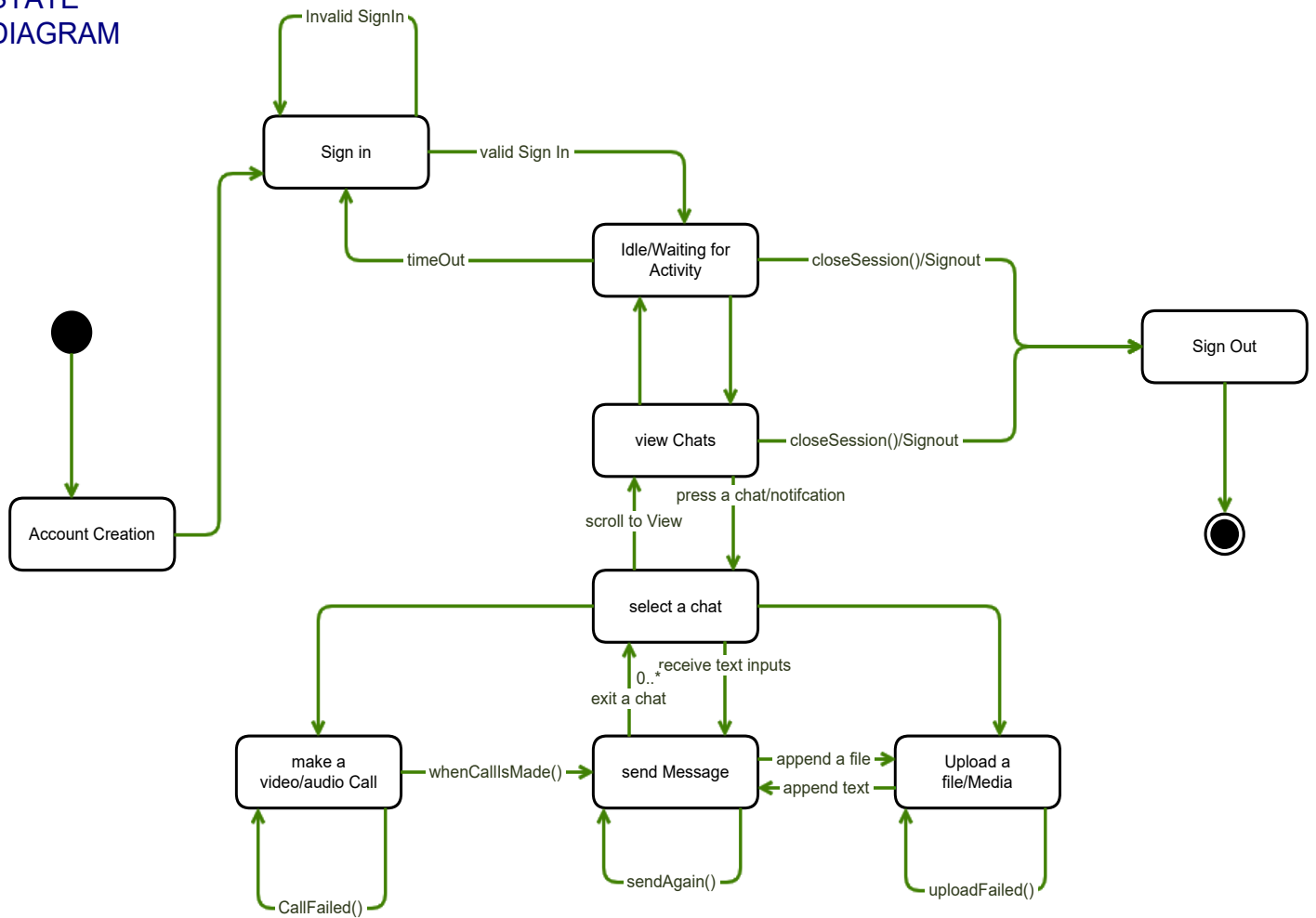
DIAGRAM(WHOLE SYSTEM )



## SEQUENCE DIAGRAM: SIGNIN

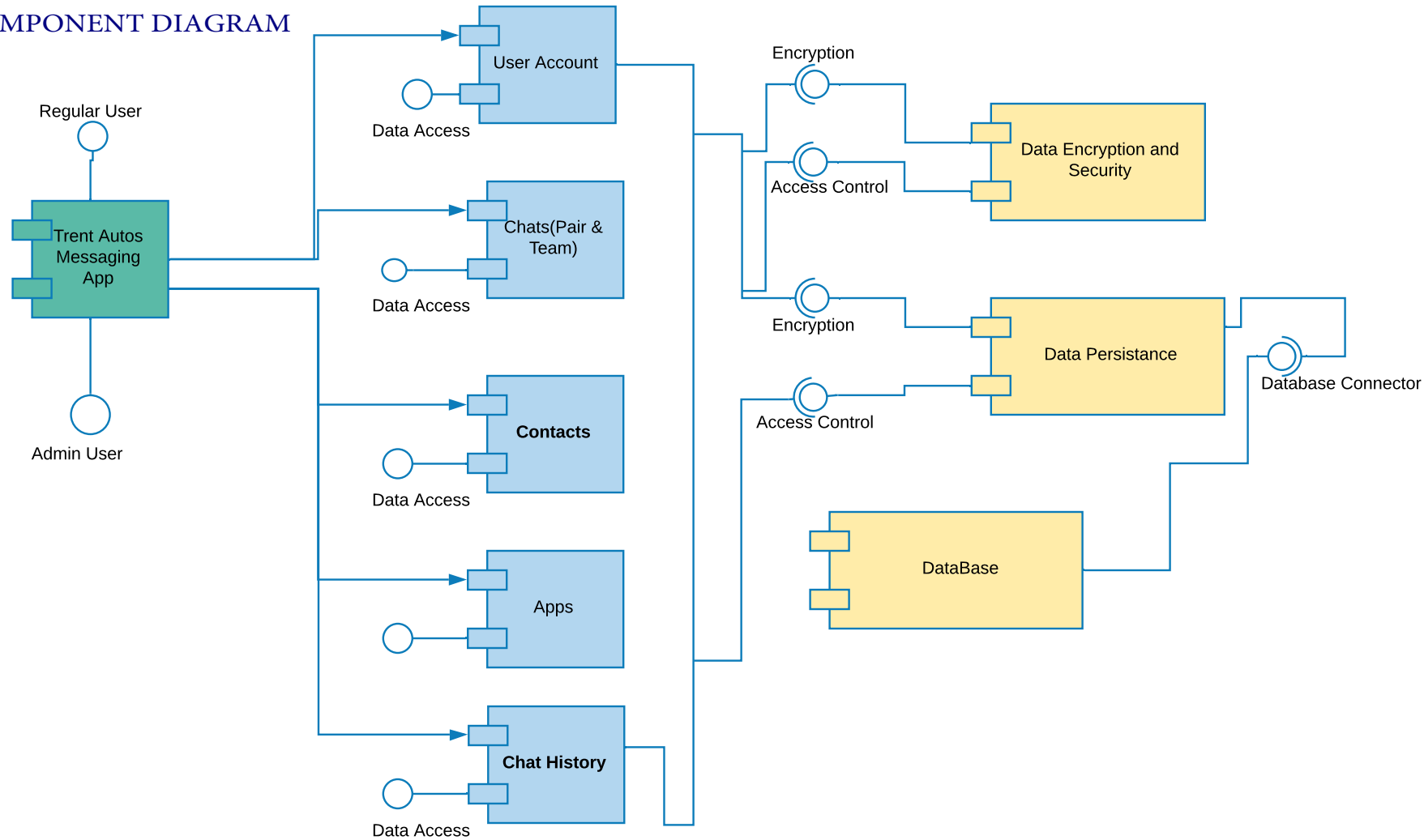


# STATE DIAGRAM

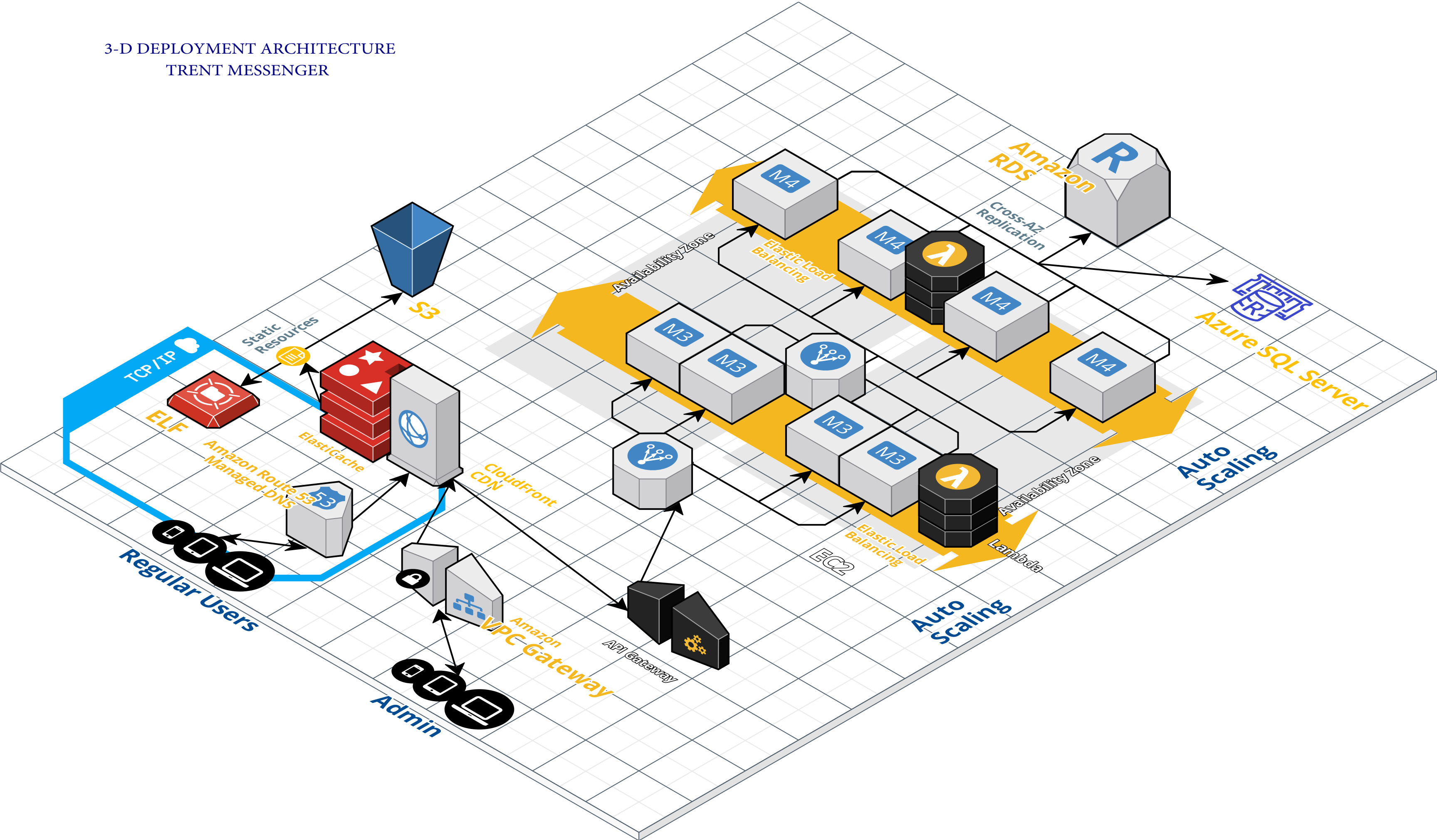


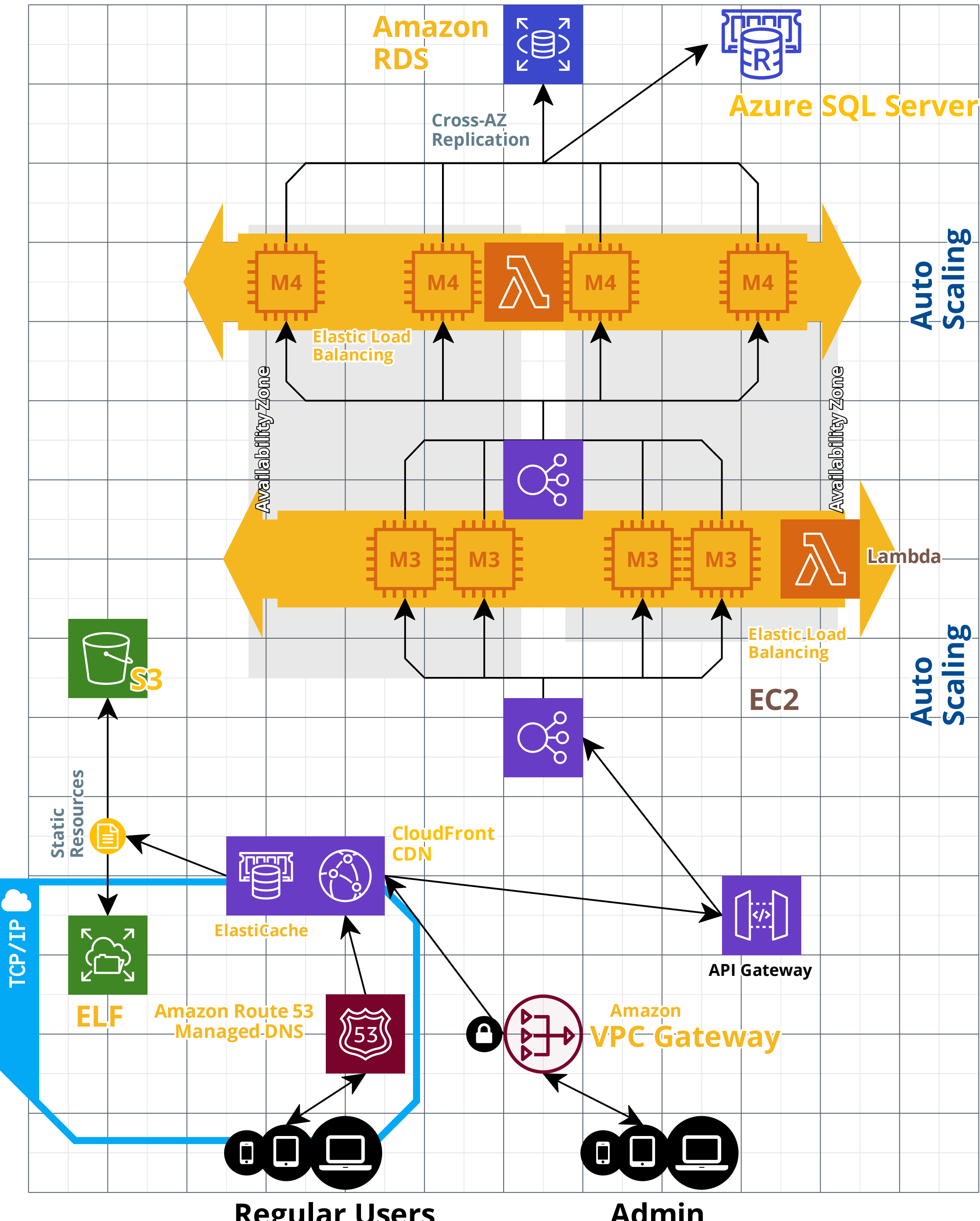


COMPONENT DIAGRAM



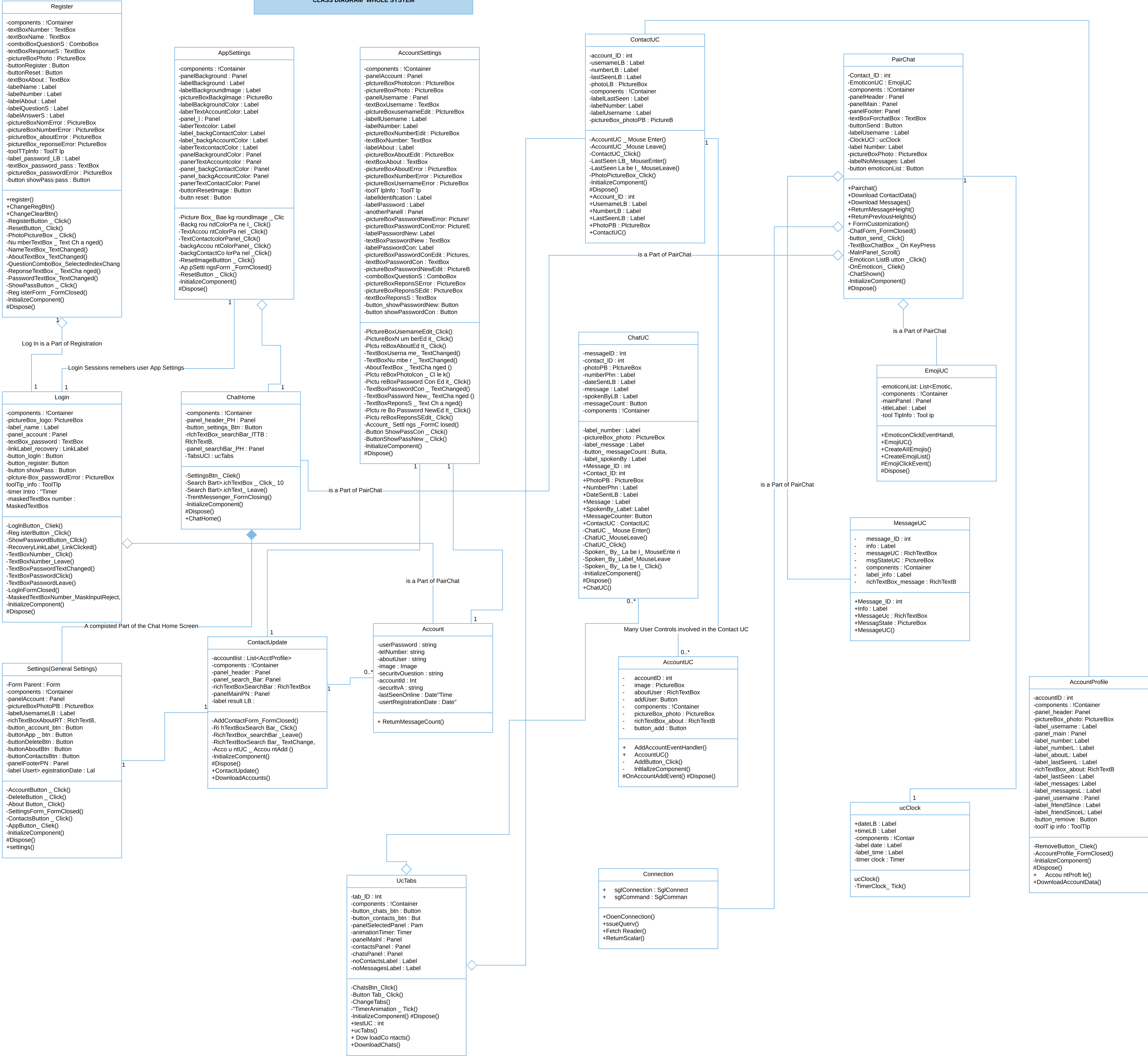
3-D DEPLOYMENT ARCHITECTURE  
TRENT MESSENGER







CLASS DIAGRAM WHOLE SYSTEM



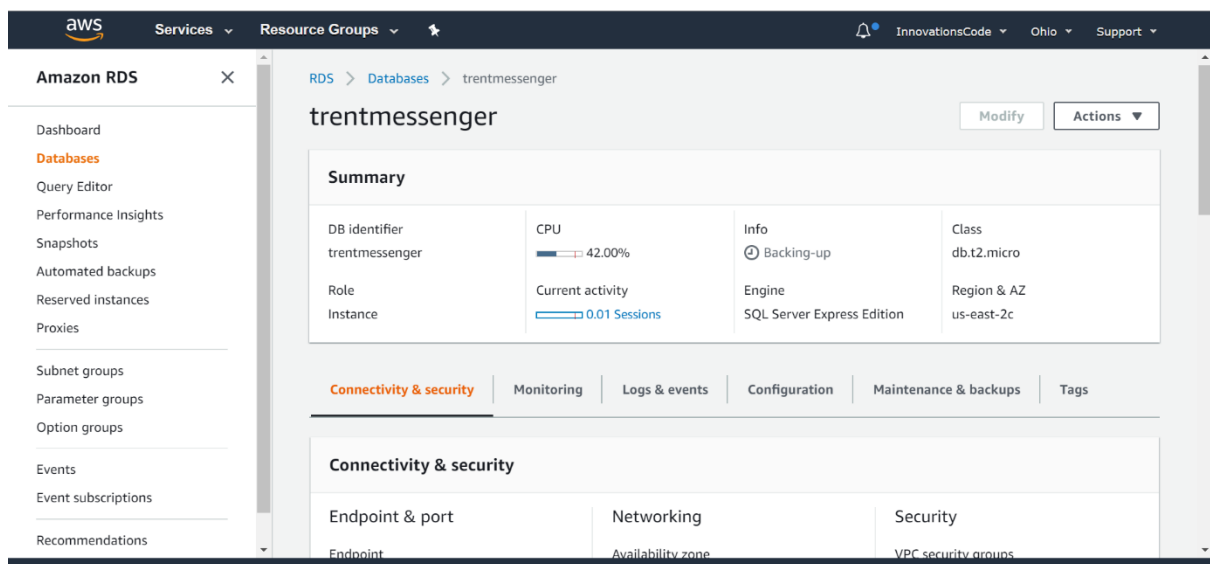
## DATABASE IMPLEMENTATIONS (HIGHEST DIFFICULTY)

### SELECTING A RELATIONAL DATABASE SET-UP AND SERVICE

With an eye for Database Management Excellence for the TrentMessenger Chat App, I explored a number of Relational Database Services from Amazon Web Services, Microsoft Azure and Google Cloud Services.

In my set-up process with Microsoft Azure, I had issues provisioning the Database Instance to a Geo-location as all weren't available. I wanted to use Azure because they have a family of products geared for the .NET Technologies.

However, Amazon Web Services were very efficient with Provisioning Process.



The above image shows a snapshot of the trentmessenger Database Instance provisioned by Amazon Web Services. I designed the following the Configuration Specifications and Networking Parameters:

#### System Spec:

1. Storage :20 GiB
2. Storage autoscaling: Enabled
3. Maximum storage threshold 1000 GiB
4. Instance class db.t2.micro
5. vCPU RAM 1 GB

#### Endpoint & port

6. Endpoint: trentmessenger.cr02x0gbm6yi.us-east-2.rds.amazonaws.com
7. Port: 1433
8. Networking
9. Availability zone: us-east-2c
10. VPC: vpc-020bdf69
11. Subnet group: default-vpc-020bdf69

### **Subnets**

- 12. subnet-bd81abc7
- 13. subnet-5f9d0013
- 14. subnet-bb7c80d0

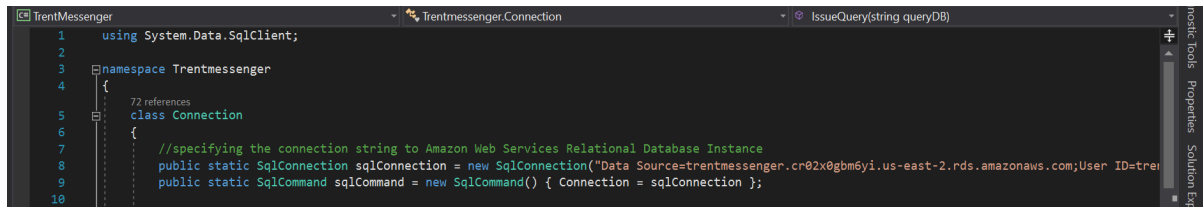
### **Security**

- 15. VPC security groups: default (sg-12b31a6b)( active )
- 16. Public accessibility: Yes
- 17. Certificate authority: rds-ca-2019
- 18. Certificate authority date: Aug 22nd, 2024

With fully automated analytics and adjustable maintenance controls, I had a great experience building part of the Trent Messenger deployment infrastructure with Amazon Web Services. The full System Deployment is created with Elastic Load Balancing (ELB), Elastic Compute (EC2), S3 Bucket and an impressive Multi A-Z Cross Replication using Redis Database.

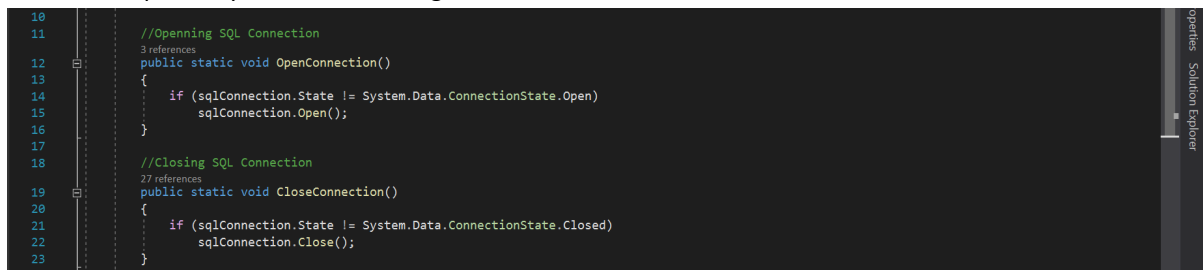
## INTERNAL STRUCTURES

I used an instance of the SqlConnection Class, passing the connection string I retrieved from the Database Properties of the Server Object Explorer Menu, to connect to the Server. This was done after linking the AWS Provisioned trentmessenger Database to the TrentMessenger App using Server Name/ DNS through Visual Studio.



```
1 using System.Data.SqlClient;
2
3 namespace Trentmessenger
4 {
5     72 references
6     class Connection
7     {
8         //specifying the connection string to Amazon Web Services Relational Database Instance
9         public static SqlConnection sqlConnection = new SqlConnection("Data Source=trentmessenger.cr02x8gbm6yi.us-east-2.rds.amazonaws.com;User ID=trentmessenger;Password=;");
10        public static SqlCommand sqlCommand = new SqlCommand() { Connection = sqlConnection };
11    }
12 }
```

I wrapped the connection opening and closing procedures in OpenConnection() and CloseConnection() Methods respectively after ascertaining the state of the connection with an If-condition statement.



```
10
11 //Opening SQL Connection
12 3 references
13 public static void OpenConnection()
14 {
15     if (sqlConnection.State != System.Data.ConnectionState.Open)
16         sqlConnection.Open();
17 }
18
19 //Closing SQL Connection
20 27 references
21 public static void CloseConnection()
22 {
23     if (sqlConnection.State != System.Data.ConnectionState.Closed)
24         sqlConnection.Close();
25 }
```

I created an IssueQuery(string queryDB) Method that receives the Query String for making changes to the Database namely INSERT, DELETE AND UPDATE. This Method calls the ExecuteNonQuery() Method which will execute the Transact SQL Statements passed as the parameter for the IssueQuery(queryDb) Method and returns the corresponding number of trentMessenger DB rows affected by such a query. Similarly, a FetchReader(string queryDB) of type SqlDataReader was executed to fetch objects and values from the trentmessenger Database.



```
24
25 //Taking a Query
26 19 references
27 public static int IssueQuery(string queryDB)
28 {
29     int rowDB = 0;
30     sqlCommand.CommandText = queryDB;
31     OpenConnection();
32     rowDB = sqlCommand.ExecuteNonQuery();
33     CloseConnection();
34     return rowDB;
35 }
36
37
38 //Fetching Data
39 14 references
40 public static SqlDataReader FetchReader(string queryDB)
41 {
42     sqlCommand.CommandText = queryDB;
43     OpenConnection();
44     return sqlCommand.ExecuteReader();
45 }
46 }
```

## Example of Usage of the Above-Mentioned DB-querying Methods

```
Regex phoneNum = new Regex(@"^(07)\d{9}$");
if (!phoneNum.IsMatch(textBoxNumber.Text)) throw new Exception("You entered an invalid phone number!");
if (textBoxAbout.Text.Length > 100) throw new Exception("About User must not exceed 100 characters!");
if (textBox_password_pass.Text.Length > 30) throw new Exception("UserPassword must not exceed 30 characters!");
if (textBoxResponseS.Text.Length > 40) throw new Exception("Security Answer must not exceed 40 characters!");
if (Connection.FetchReader("USE trentMessengerDB3; SELECT AccountID FROM Accounts WHERE TelNumber = '" + textBoxNumber.Text + "';").HasRows) throw n

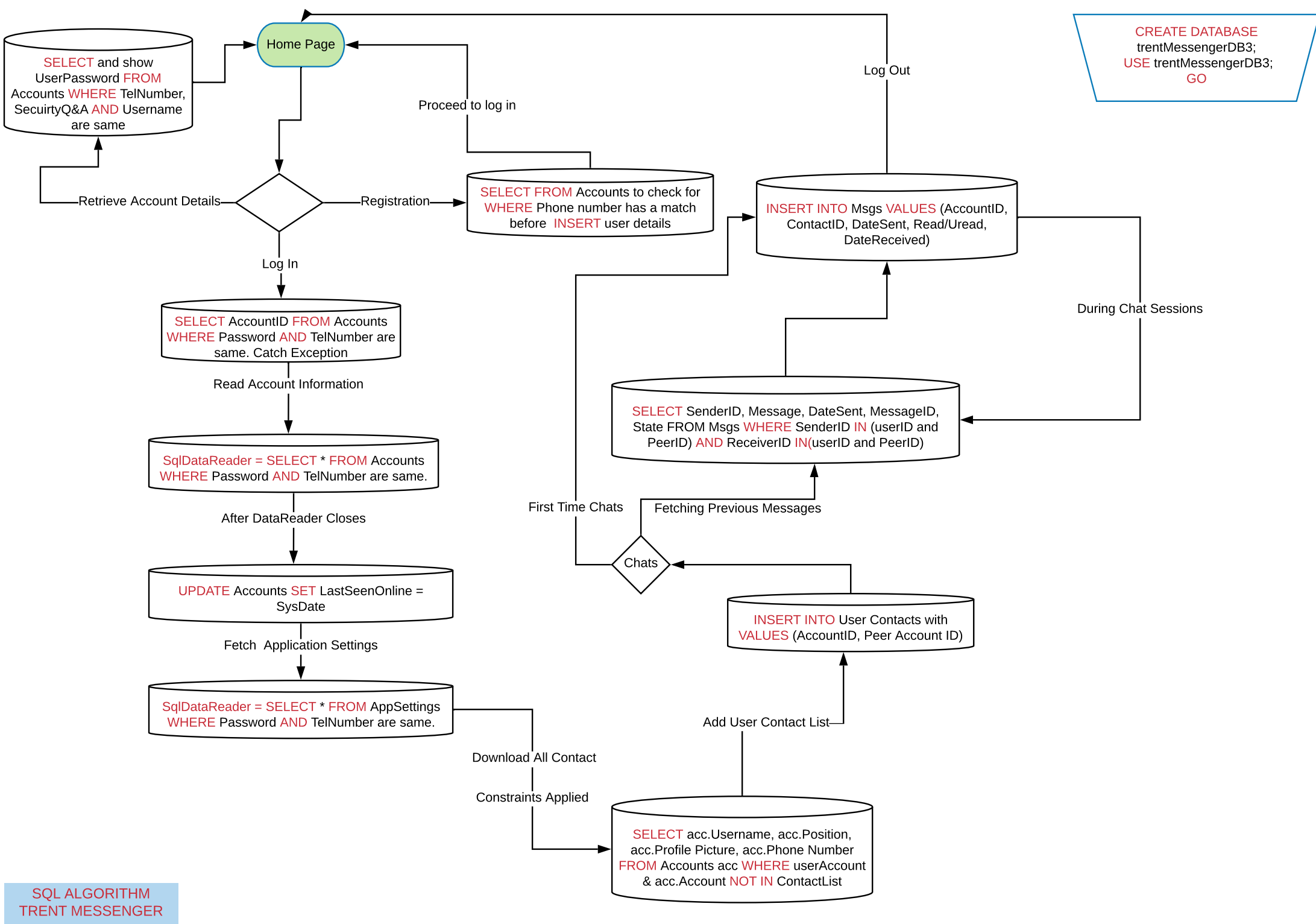
Connection.CloseConnection();
MemoryStream memoryStream = new MemoryStream();
pictureBoxPhoto.Image.Save(memoryStream, pictureBoxPhoto.Image.RawFormat);
byte[] picture = memoryStream.GetBuffer();
memoryStream.Close();
Connection.sqlCommand.Parameters.Clear();
Connection.sqlCommand.Parameters.AddWithValue("@photo", picture);
Connection.sqlCommand.Parameters.AddWithValue("@about", textBoxAbout.Text);
ConnectionIssueQuery(" USE trentMessengerDB3; INSERT INTO Accounts(Username, UserPassword, TelNumber, AboutUser, Image, SecurityQuestion, Security
MessageBox.Show("Account successfully registered", "Notice", MessageBoxButtons.OK, MessageBoxIcon.Information);
this.Close();
//Exception handling
```

Here is a block of snippet from my Register Class designed to handle the Logic for the User Registration of Trent Messenger. I used a regular expression constructor to handle user error input for the phone numbers and other Boolean checks to see that the characters of the forms are of the right stipulated length.

Thereafter, I ran a check calling the FetchReader(queryDB) to see if the Phone Number already exists in the Database, after which either exception is thrown or the connection for the reader is closed and the Insertion Processes of the registration is continued.

If the latter is true, then memory stream for the image is instantiated and closed after the Buffer is gotten. Thereafter, I appended the Picture to the Query String by calling the AddWithValue("@photo, picture)—this is observable from the code block above—after which the Insertion Query was issued. I designed a similar implementation procedure for the INSERTION, DELETE AND UPDATE operations of other activities like Login, App Settings, Messaging Activities, Contacts and Account Details Recovery.





## THE TABLE STRUCTURE OF DATABASE

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Object Explorer displays the database structure for 'trentmessengerDB3'. The 'Accounts' table is selected. The main pane shows the SQL query: `SELECT TOP (1000) [AccountID], [Username_], [UserPassword], [TelNumber], [AboutUser], [Image], [SecurityQuestion], [SecurityR], [LastSeenOnline], [RegistratingDate] FROM [trentmessengerDB3].[dbo].[Accounts]`. The results pane shows the first 10 rows of the table, including columns like AccountID, Username\_, UserPassword, TelNumber, AboutUser, Image, SecurityQuestion, SecurityR, LastSeenOnline, and RegistratingDate.

AccountID	Username_	UserPassword	TelNumber	AboutUser	Image	SecurityQuestion
102	DavidAdama	England123#	07123456799	I'm David Adama	0xFFD8FFE000104A46494600010101004800480000FFFE20C...	Your hobby
103	Yahaya	England123#	07123456778	I'm Yahyah	0xFFD8FFE000104A46494600010101004800480000FFFE20C...	Your hobby
104	Scarlet	England123#	07123456777	I'm Scarlet	0xFFD8FFE000104A46494600010101004800480000FFFE20C...	Your favorite :
105	MichaelOnye	England123#	07123456777	I'm Michael	0xFFD8FFE000104A46494600010101004800480000FFFE20C...	Your hobby
106	JamesPark	England123#	07111234566	I'm James Park	0xFFD8FFE000104A46494600010101004800480000FFFE20C...	Your hobby
107	ArnoldClark	England123#	07122345678	CEO, Trent Autos Limited	0xFFD8FFE000104A46494600010101004800480000FFFE20C...	Your favorite :
108	Clifton Trump	England123#	07123334567	Sales Manager, East-Midlands, Trent Autos Limited	0xFFD8FFE000104A46494600010101004800480000FFFE20C...	Your hobby
109	AdamKaleigh	England123#	07123458964	Adam Blue, Finance Lead, Trent Autos	0xFFD8FFE000104A46494600010101004800480000FFFE2021...	Your favorite color
110	SarahWills	England123#	07345678901	Sales Auditor, London, Trent Autos	0xFFD8FFE000104A46494600010101004800480000FFFE2021...	Your hobby
111	JolinaJolie	England123#	07123456780	Assistant Sales Lead, Nottingham	0xFFD8FFE000104A46494600010101004800480000FFFE2021...	Your favorite :

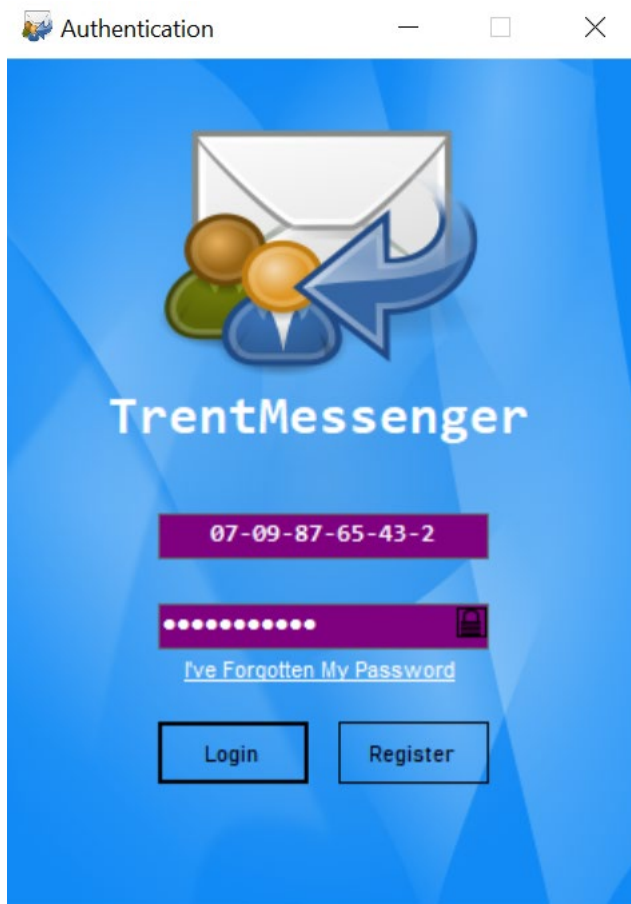
Here is a snapshot of the Accounts Table in trentmessengerDB3 using the SQL Server Management Studio (SSMS). In line with Best Practices, I created a new User Account for the trentmessenger DB for Application Usage while the Master Account is reserved for the role of the “Owner”. I also had to create a new Database for this chat application within the Server to avoid the pre-built restrictions that comes with the rdsadmin and model databases. Visual Studio also has an impressive functionality for relating with the Database.

While the above table was populated with pre-release user testing, I also employed the services of [www.Mockaroo.com](https://www.mockaroo.com) to generate Test Data using my previously designed Database properties.

The screenshot shows the Mockaroo website interface for generating test data. The 'Field Name' column lists the fields: AccountID, Username\_, UserPassword, TelNumber, AboutUser, Image, SecurityQuestion, SecurityR, LastSeenOnline, and RegistratingDate. The 'Type' column shows the data types: Row Number, Username, Password, Number, Words, Base64 Image URL, Words, Words, Date, and Date. The 'Options' column shows the configuration for each field, including constraints like min, max, decimals, blank, and format.

Field Name	Type	Options
AccountID	Row Number	blank: 0 % fx
Username_	Username	blank: 0 % fx
UserPassword	Password	blank: 0 % fx
TelNumber	Number	min: 11 max: 11 decimals: 0 blank: 0 % fx
AboutUser	Words	at least 15 but no more than 20 blank: 0 % fx
Image	Base64 Image URL	blank: 0 % fx
SecurityQuestion	Words	at least 10 but no more than 20 blank: 0 % fx
SecurityR	Words	at least 10 but no more than 20 blank: 0 % fx
LastSeenOnline	Date	4/8/2019 to 4/8/2020 in SQL datetime blank: 0 % fx
RegistratingDate	Date	4/8/2019 to 4/8/2020 in yyyy-mm-dd blank: 0 % fx

## LOGIN



The screenshot shows a web browser window titled "Authentication". The main content area has a blue background with a graphic of an envelope and three colored circles (brown, orange, green) with a blue arrow pointing right. Below the graphic, the text "TrentMessenger" is displayed. There are two input fields: the first is a phone number field containing "07-09-87-65-43-2", and the second is a password field with masked characters ".....". Below the password field is a link that says "I've Forgotten My Password". At the bottom, there are two buttons: "Login" and "Register".

*Welcome to The Trent Messenger Home Page.*

The Home Page is the Sign-In Portal where the User Authentication Credentials are collected and verified. This is a requisite step before Messaging and core app functionality can be unboxed.

To Sign in enter your registered phone number. You would notice that the input is masked to allow an 11- Digit UK mobile number, starting with a mandatory 07 number.

If You have forgotten your Password, a retrieval can be made via the "I've forgotten My Password" link. At the Retrieval Page fill in your Security Question from the Combo Box Drop-down, where a selection of pre-defined Security Questions can be made. Thereafter, fill in your Security Answer and hit the retrieve Password Button.

If you don't have an account, press the "Register" Button to Register a Trent Messenger Account.

## ACCOUNT REGISTRATION



Registration

Username  
PauTimothy

Phone Number  
07123456789

Position

Password

Security Question

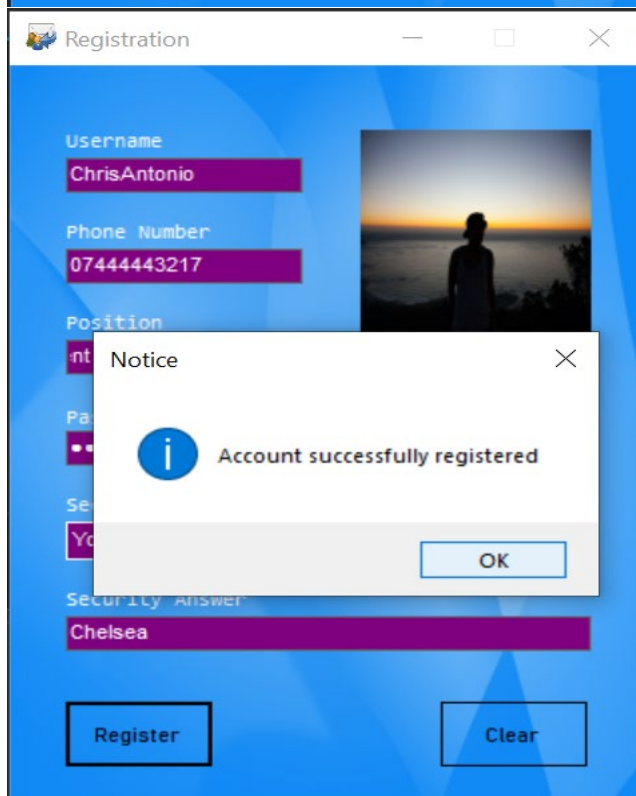
Security Answer

Register Clear

This is the Registration Page. Here all the fields are mandatory for the Profile Image. Also ensure that inputs are correctly entered. If you enter a phone number that exceeds the specified 11-Digit, an error warning will appear from Trent Messenger.

Upload a profile picture, fill in all the text boxes, press clear to start all over again or register to complete the process.

On a Successful Registration, Trent Messenger will send "Account Successfully Registered".



Registration

Username  
ChrisAntonio

Phone Number  
07444443217

Position

Password

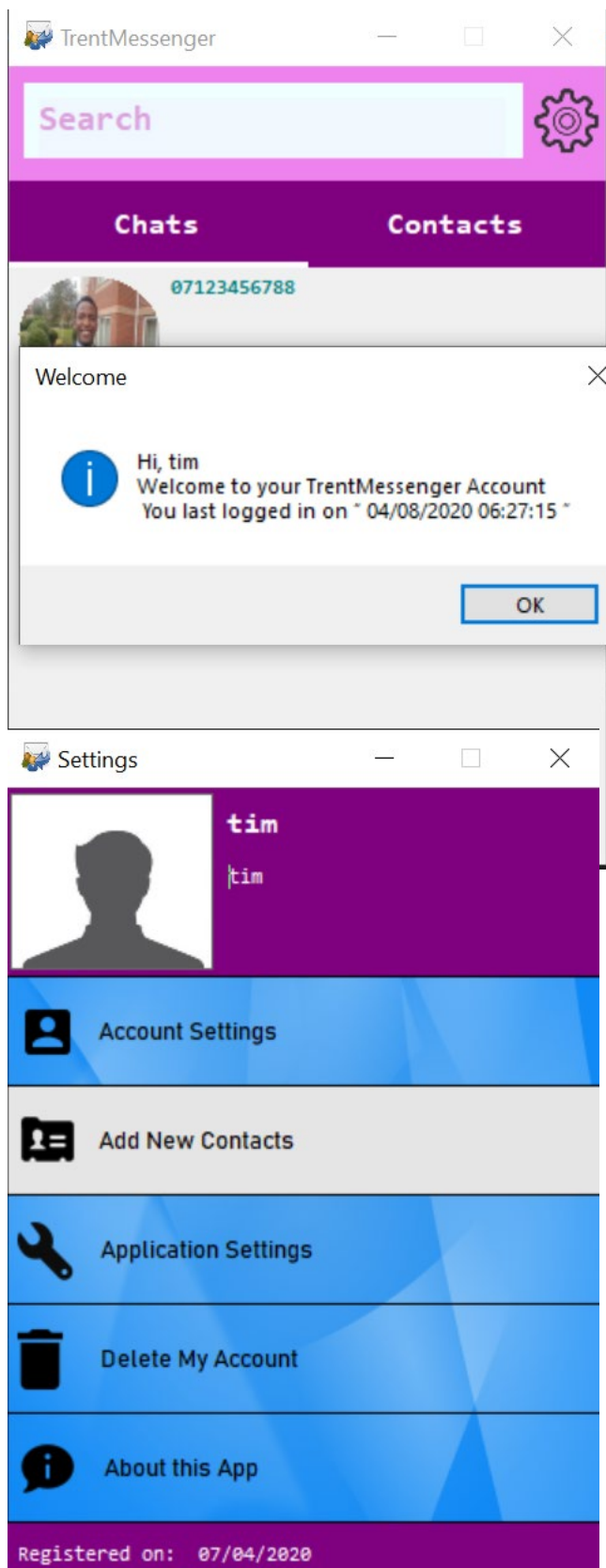
Security Question

Security Answer  
Chelsea

Register Clear

Notice  
Account successfully registered  
OK

## GENERAL SETTINGS:



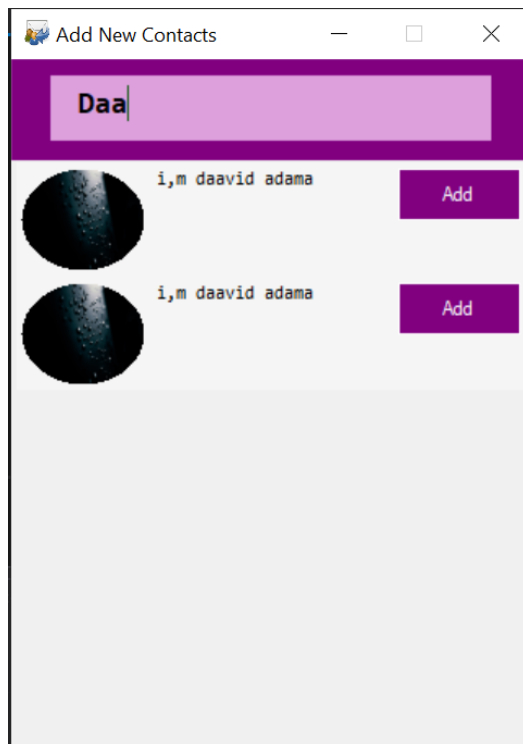
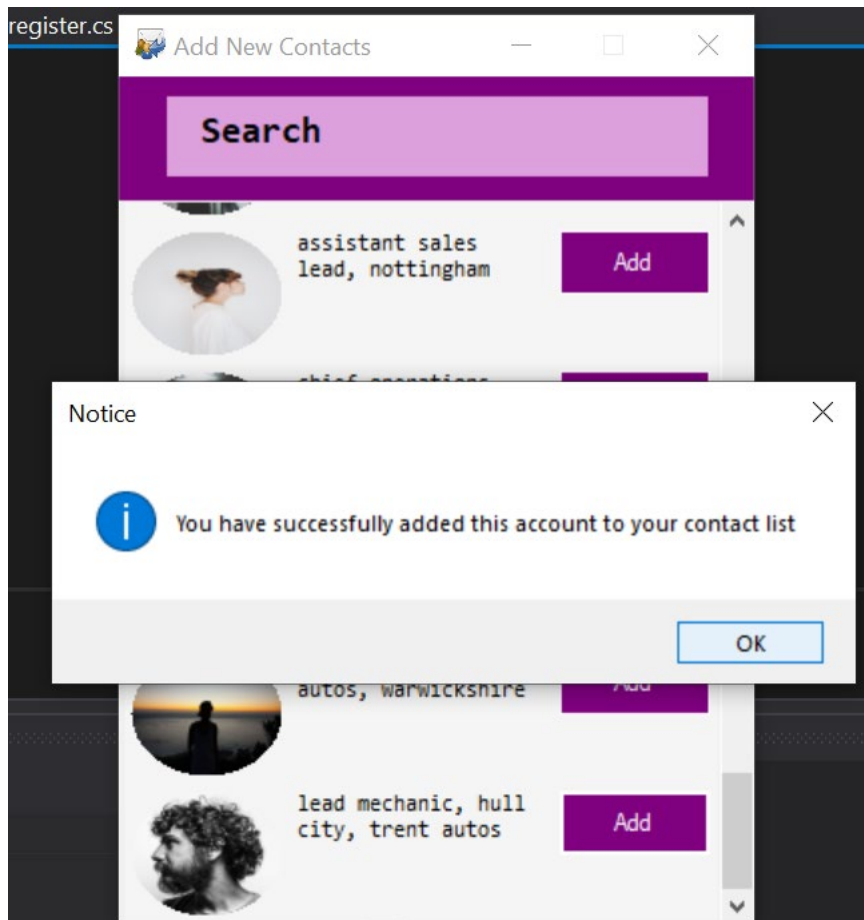
After a Successful Log in. This Page unfolds with the Search Bar, Chats and Contacts Tabs, Chats and Contacts Controls.

From here, Navigations can be made to either search existing chats and added contacts, Press Chats or Contacts Tab or Press and existing chats or contacts control to resume or start a chat respectively.

If your Trent Messenger Account is new, you would have no active chats or contacts. You therefore need to go to the settings button to load contacts from the Organization Chat Lists.

On pressing the Settings Icon, the Settings Page unfolds, press the "Add Contacts" Button.

## ADD ORGANIZATION CONTACTS

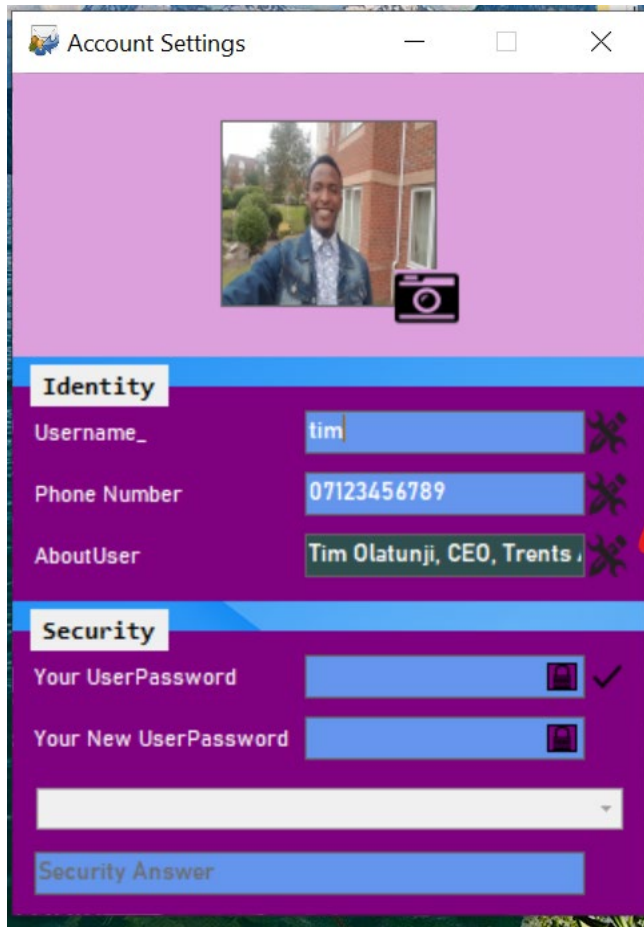


After pressing the “Add New Contacts” Button, the UI shifts to this page, loading all the Contacts of the Organization, here you can Search and Add relevant contacts to your Account Contact Lists. A message is sent by Trent Messenger immediately notifying of the success of the add operation.

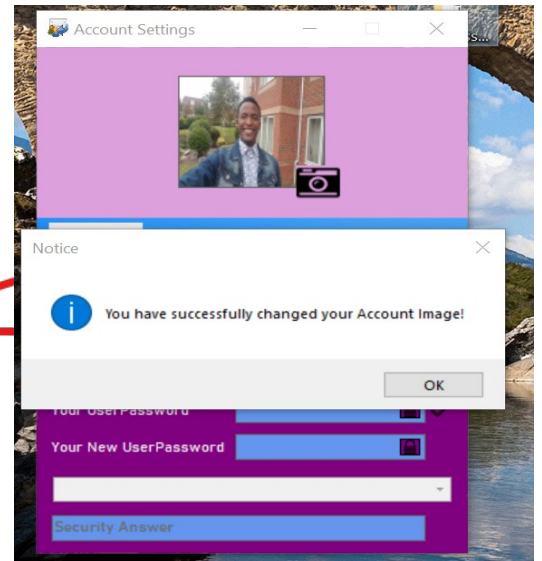
Other Functionality accessible from the General Settings Page are as follows:

---

#### ACCOUNT SETTING: SETTING A NEW PASSWORD, CHANGING SECURITY QUESTIONS AND RESPONSES



Changes can be made to previously registered account parameters through the Account Settings



---

#### ABOUT THIS APP

You can even get to know more about the rent Messenger App from the About Pop up window.

---

#### DELETE ACCOUNT

Account can also be permanently deleted.

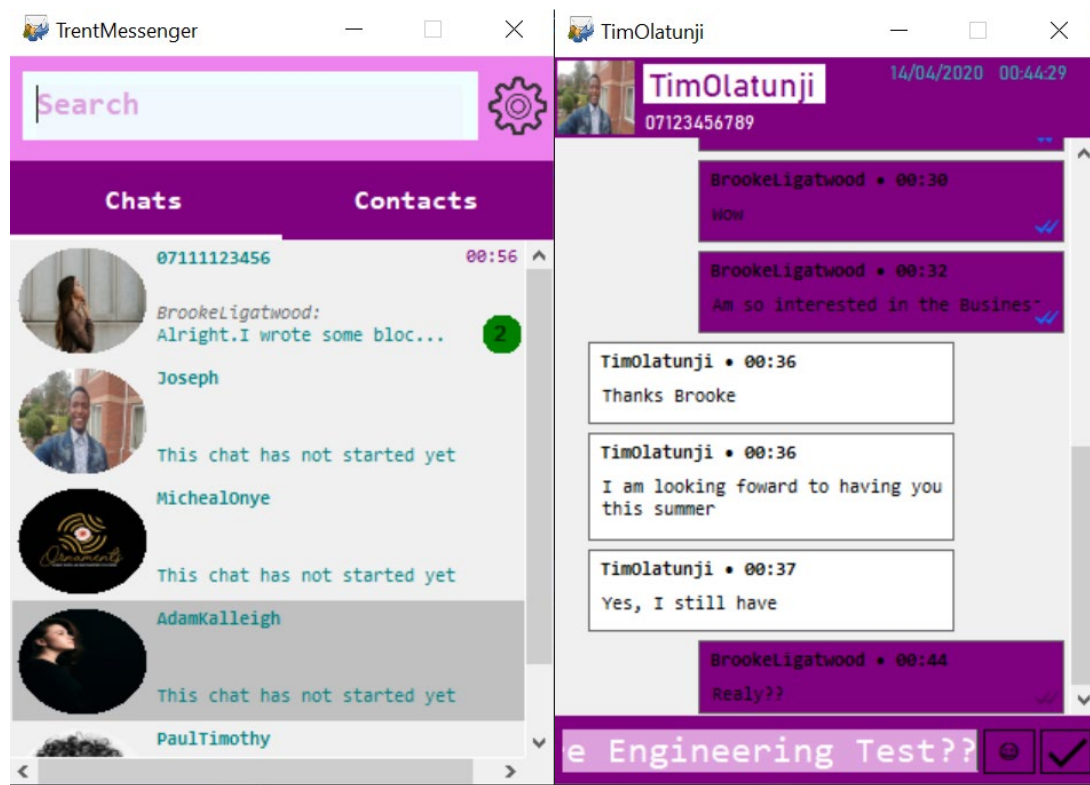
---

#### APPLICATION SETTINGS

Other useful features from this page is the App Settings, where user can customize the look and feel of this app by changing certain UI Properties. Changes made are saved directly to the server. This ensures that Trent Messenger remembers App Settings linked to account from any device.



## CHATS



To start using the TrentMessenger App to reach out to co-workers and friends, all you need to do is to either press a new chat, chat showing a received message, or search using the search bar after pressing the chat tab. You would also notice the New Message Notifier – a green round button that tells you the amount of unread message you still have from a chat. On Click of the Horizontal Chat Channel, you can see the Chat Panel unfold. Here you would notice atop the Panel there is a sub-panel that shows you the username, phone number of the user you are chatting with.

To send a message, press the horizontal plum-coloured box; as you type, your text will appear in white as illustrated above. Click the emoticon logo to append emojis or the tick icon to send your message.

Every chat bubble has the Sender Name and the Receiver Name: You wouldn't have to wonder who sent what. You can also personalize and customize the features of the chat bubbles (for both sender and receiver) at the Application Setting page at the General Settings Page described above. The highly impressive usability engineering of Trent Messenger will ensure that your settings are remembered regardless of the device you login in from.

WELCOME TO THE TRENT MESSENGER APP BUILT BY TIM OLATUNJI FOR TRENT AUTOS UK LIMITED