Desktop Messenger Application

FOR DOGE FINANCIAL

Aleksy RUSZALA @1902370

Contents

Introduction 2
Propose 2
Scope 2
Research 2
Specific requirements 3
Functional requirements 3
Non-functional requirements 4
Security 5
Patterns 6
Development methodology 7
Project Timeline 8
UML Diagrams 9
User Guide 16
Testing 17
Resources ..... **Error! Bookmark not defined.**
Evaluation (and Improvements) 19

Introduction

Propose

The purpose of this document is to provide a detailed specification of details about creating project. This specification is intended to be presented and approved by the client and as a reference point for the developers to create applications to meet customer requirements.

Scope

The project is to create an application for DOGE FINANCIAL, used for communication between company users. The program is to be characterized by a high standard of security, as the data transmitted is sensitive and requires special treatment and protection against third parties. Also, the program is to send data between users as soon as possible - in real time. The only place to store data is to be the database that is hosted on the company's server. Only verified employees can use the program, they will can access with the appropriate login and password.

Research

Mainly research was focus to all application security issues available for windowed applications also real-time connection between programs. As a search result, I decided to use the SignalR framework to implement the connection in real time and implement rest-API. In order to secure the application, I chose client verification by token. Separating application layers into client-server-databases to isolate the database from direct requests. For data security, HTTPS protocol and various types of encryption were used.

Specific requirements

Functional requirements

Login to the application
The application should be available only to users who have a user account. Such a user must be able to log into the program with a username and password.

Sending messages
The user should be able to send messages to selected users. The message should be sent only to the selected user.

Receiving messages
The user should be able to receive messages sent by other users.

Starting a conversation
The user has the option to choose a user from the list with whom he wants to contact and start a conversation with him in a new window. The user must be able to talk to each other user individually.

Viewing the user list
The program should show a list of all available users in the system.

Message encryption
All messages should be secure and encrypted to prevent other users from accessing confidential information.

Minimal latency
Messages should arrive as quickly as possible.

Microsoft Windows 7 compatibility
The software should be compatible with Windows 7 and run smoothly.

Copy of messages
The program cannot store any information on the machine from which the goat user. They cannot be stored on disk.

Message sound
When receiving a new message, the program should communicate this by playing a sound or inform the user visually.

Remove old messages
Messages older than six months should be permanently deleted from the system.

Non-functional requirements

Offloading the database
The program should not unnecessarily call the database with queries. Should use it only when it downloads data for the first time, or in case the data has changed.

Connection security
The connection should be secure and preferably encrypted. The program should only exchange information in this way. This reduces the chances of receiving valuable information by third parties.

Professional application template
The application should look professional and reflect the best possible user experience.

User management
The system administrator should be able to manage users through the program.

Work in a remote network
As the company grows, users should be able to communicate between branches that are away from each other and that use other networks.

Preventing multi-logging
The user should be able to log in only once at the same time, otherwise the previous login session should be deleted, and the program should log out user on last PC

Session
Users should be automatically logged out when the server shuts down, this also applies to blocking users who want to resume sessions after restarting the server. The user should be forced to log in again.

Security

Password encryption

The user's password is encrypted to prevent intercept by third party. The password is already encrypted at the stage of sending the request to the server. The password in the database itself is also not stored in a pure form, so that anyone hacking into the database could not steal it or restrict administrators from reading passwords. Password checking is based on a comparison of the encrypted password in the database with that in the client request.

Message encryption

Like the password, messages are also encrypted to protect them from being read by unauthorized persons. Compared to the password encoding used, the messages are hashed with a more advanced script that uses salt. In this case, messages received can be decoded by the customer who has the appropriate salt. Messages are also stored in coded version. To encode messages AES helper class has been used. (Adriancs, 2019)

Tokens

The server uses external tokens to verify connections. The token is generated by the server when the user logs in and his lifetime lasts 5 minutes to increase security. Along with the token, the client also receives a key to refresh the token, which after it expires will generate a new token and a new key. Each connection requires a valid key to be provided by the client, then the server checks its correct and allows it to use authorized queries. If the token is not valid, the server will reject the connection.

Session

The server monitors the issued keys during the session if the program does not have a key in memory. Rejects the call. The same happens with SignalR connections. All session memory is reset when the server is restarted, which prevents the use of old keys or previously established connections.

Hold data in memory

The client does not write data to disk but only stores current data in memory during the window's life. When the window closes, all data downloaded from the server is lost. This requires downloading fresh data from the server when reopening the message window or the main window.

Application Layers

Dividing the program into layers allowed to increase security e.g. prevent direct queries to the database that is stored locally.

Patterns

Dependency injection
using this pattern will make the code less associated with other elements, and thus be more open to extensions.

MVC/MVP
although WinForms is not designed for this pattern, its use a benefit here. Separation of application responsibility between three units; model, view and controller. These three units work together to support the entire system and its logic. The main advantage is the fact that each of the units can operate independently of the other which gives modularity and reusability.

Repository pattern + Entity Framework
Using this pattern following benefits can be mentioned: decouples an application from the persistent framework and promotes testability.

Singleton
This this program provide access to the same instance of class for example in case need to access to same set of data globally.

Real-Time
Not directly but using the SignalR framework server and client have the option of real-time invoke methods with each other

Application Layers
The main advantage of layered architecture is the division of logic between components. Specific layers apply only to the logic that applies to this layer. It also increases safety. Access to the database is behind server layer, that means no one from outside the device (in network) can access it. (Hamedani, 2019)

Others
There are also several other patterns and approaches to be found in the project.

Development methodology

The project is created using the Agile methodology, which will allow us to implement subsequent application modules and collect early feedback from the client. The whole process was broken down by me into smaller tasks and into weekly sprints which, depending on the priority, were implemented into the project. When creating the application, I was in constant contact with the client, which is why details could be agreed on a regular basis. The task was clearly defined and in case of problems I could freely navigate them to get next working versions of prototypes, which in turn could be tested regularly and potential errors or future failures can be detected. Thanks to the comments from the client, it was possible to bring the application to the right track and provide subsequent versions in line with the client's expectations, meeting further requirements. Currently, most requirements are met, and applications run independently without problems. This does not mean that the applications cannot be further developed to meet all expectations. Using this methodology, there was a risk that the product would not work well enough to allow use, although the risk was quite low due to well defined tasks in the backlog. Despite the changes in requirements during the project, the changes were implemented by modifying several tasks and refactoring several previous lines of code created. With each iteration, the program more and more reminded the target program to as a result give its first full working version.

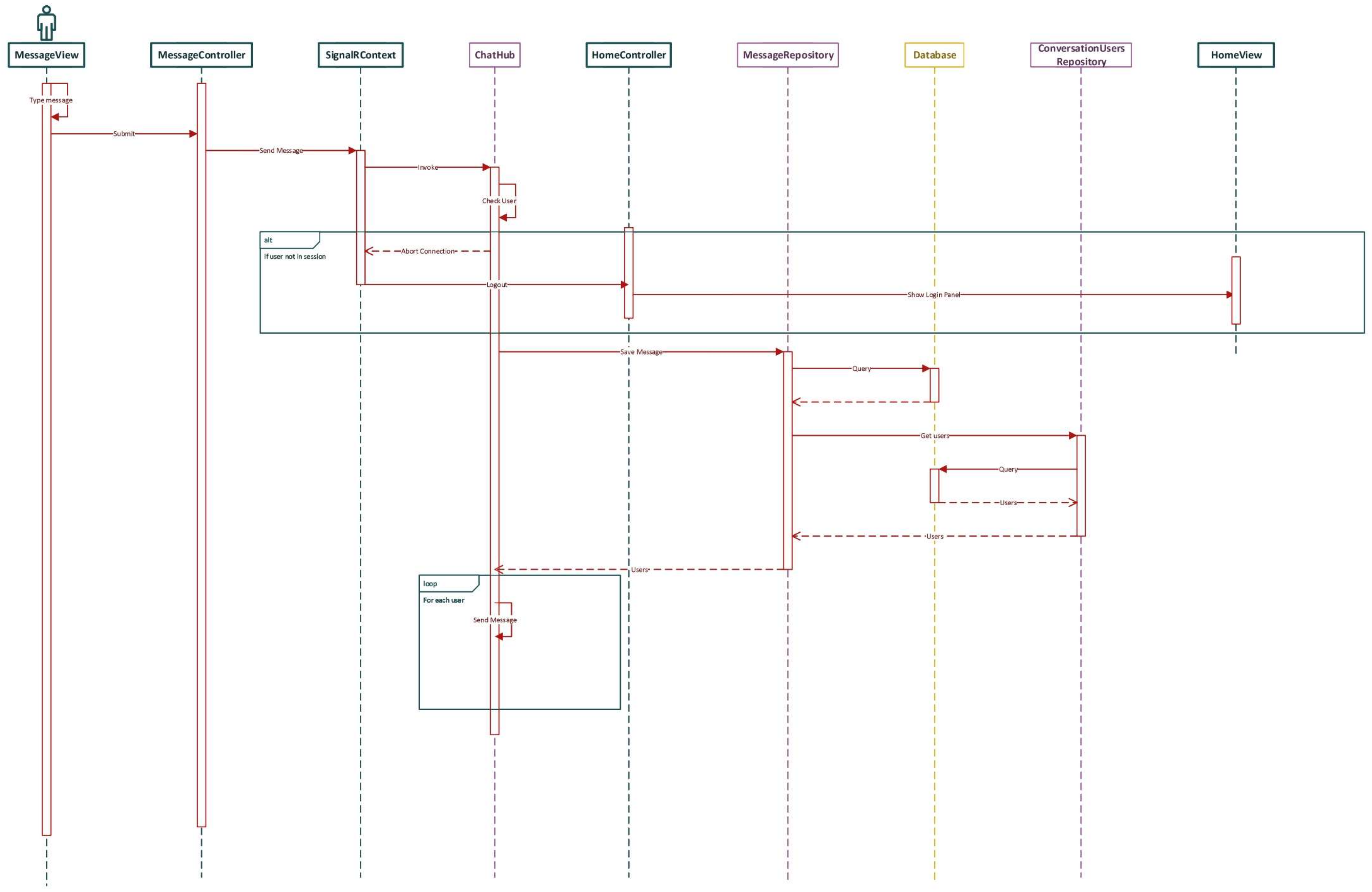
Project Timeline

Week	Dates		Task
1	8/9/2019	14/9/2019	Research and design application architecture
2	15/9/2019	21/9/2019	Create VS Project
			Create Git Repo
			Create projects for each layer
			Implement MVP and Unity for DI
3	22/9/2019	28/9/2019	Create MVP structure and implementation
			Discussion about details with the client
			Create early domain models
4	29/9/2019	5/9/2019	Init Server project
			Add repositories
			Config SignalR and WebAPI
			Create new chat hub and login controller
			Create Test projects
5	6/10/2019	12/10/2019	Create database + Test LINQ
			Create and init EF database context and test connection
			Connection established between Server <-> Client (tested)
			Test SignalR and WebApi
			Re design database and sample user
			Change database project to .netFramework instead .netCore
			Create home panel view
6	13/10/2019	19/10/2019	Sent user from db via webApi
			Refactor projects to netcoreapp3.0
			Clean unused code and fix bugs
			Create in-memory data class and test DI in SignalR
			Create Message controller and view and test lifecycle for unity
7	20/10/2019	26/10/2019	Add repository patterns
			Add hashed / salted password
			Add signalR and rest client contexts
			Add global event, Track lifetime, implement basic forms
			Create login and message form layout
			Create users api
			Login and logout done. Open messages windows done.
			Add Dispose. Release memory. Garbage colector
			Add to domain new classes
8	27/10/2019	2/11/2019	Create Message repo and controller
			Create Message Api
			Display messages into view from server
			Implement Token
9	3/11/2019	9/11/2019	Create server session
			Validate tokens based on session
			Set classes Authorize
			Send Messages to server
			Implement new message sound
			Init SingalR connection
			Check users connections based on token and session
			Add repositories for conversations
			Redesign Database
			Send confimation to server - message recieved
			Add Task to delete old messages
			Fix bugs
			Implement auto refresh tokens
10	10/11/2019	16/11/2019	Kick off logged users after server down
			Kick off users already loged on
			Create UML Diagrams
11	17/11/2019	23/11/2019	Write documentation
			complete the comments in code
			Create user guide
			Complete log book
			Create extra uni tests
12	24/11/2019	30/11/2019	Improve application layout
			Check for other potential errors
			Fix manditory bugs

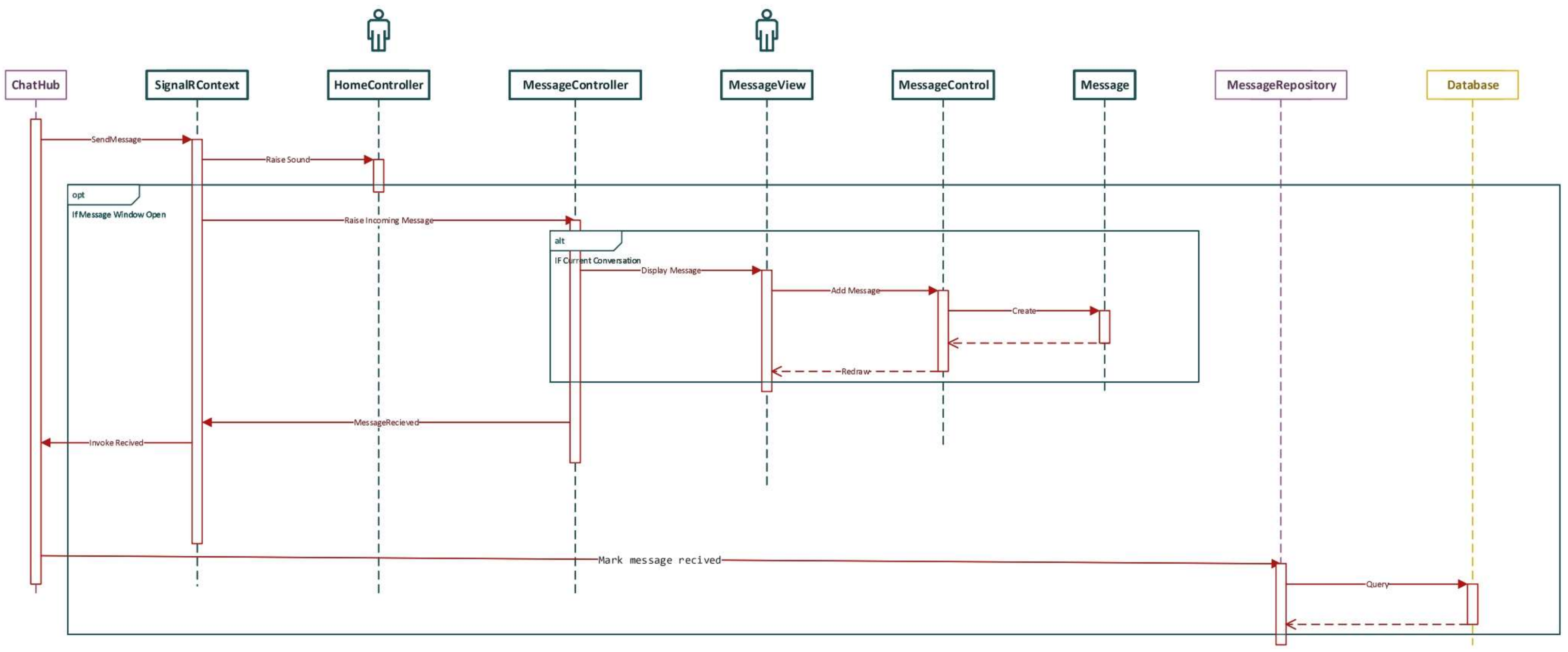
UML Diagrams

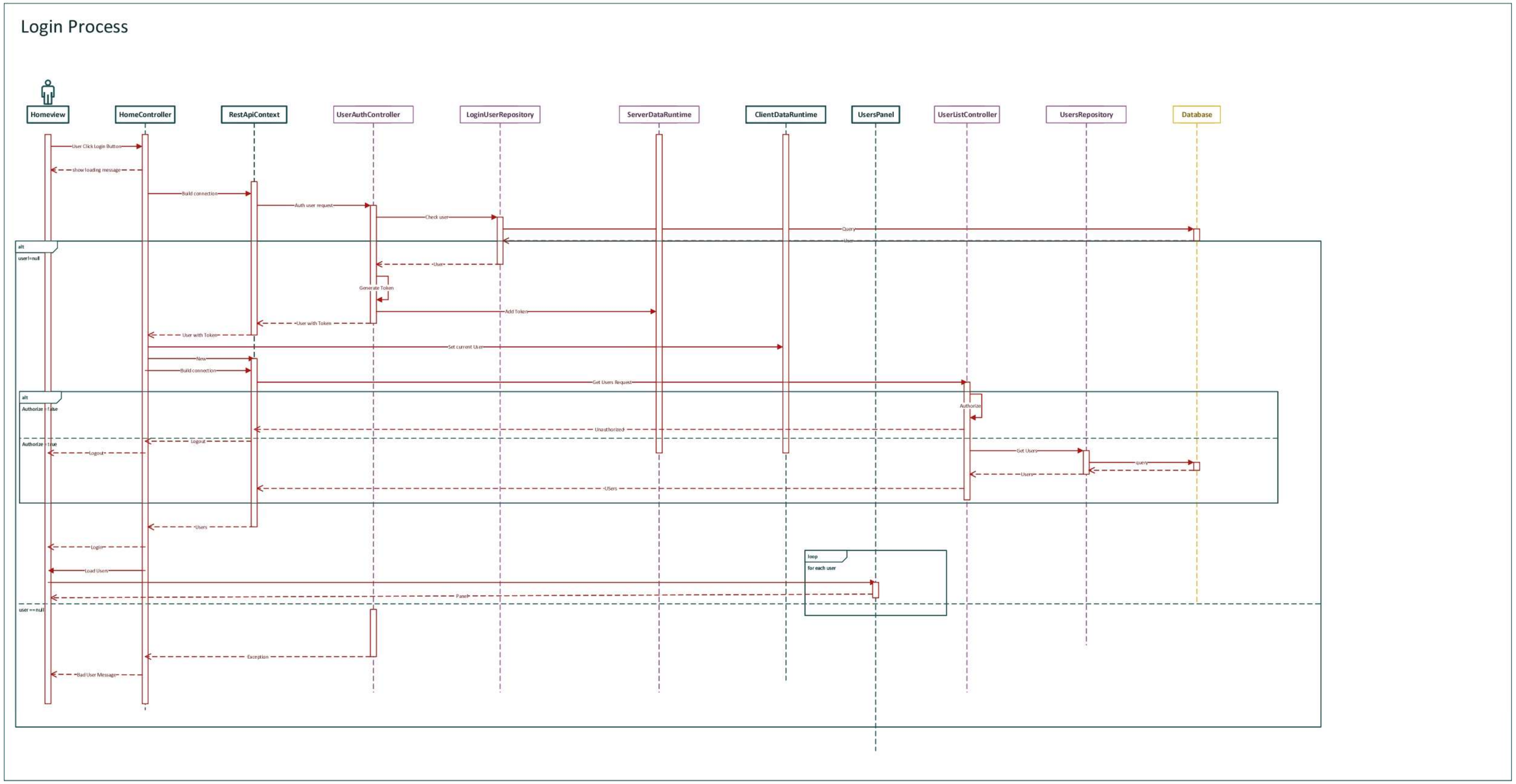
Labels
Server
Client
Database

Send Message Process

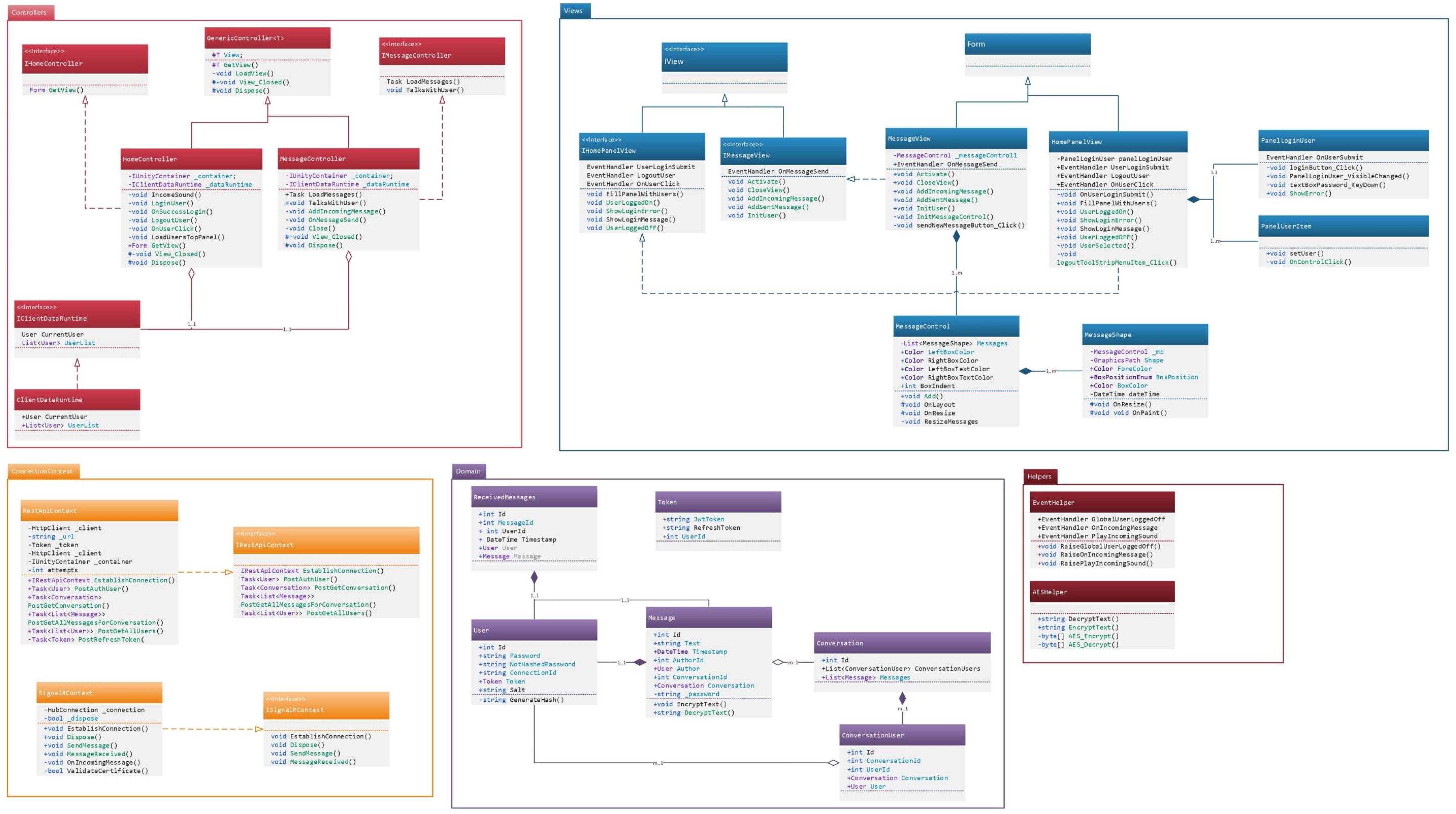


Receive Message Process

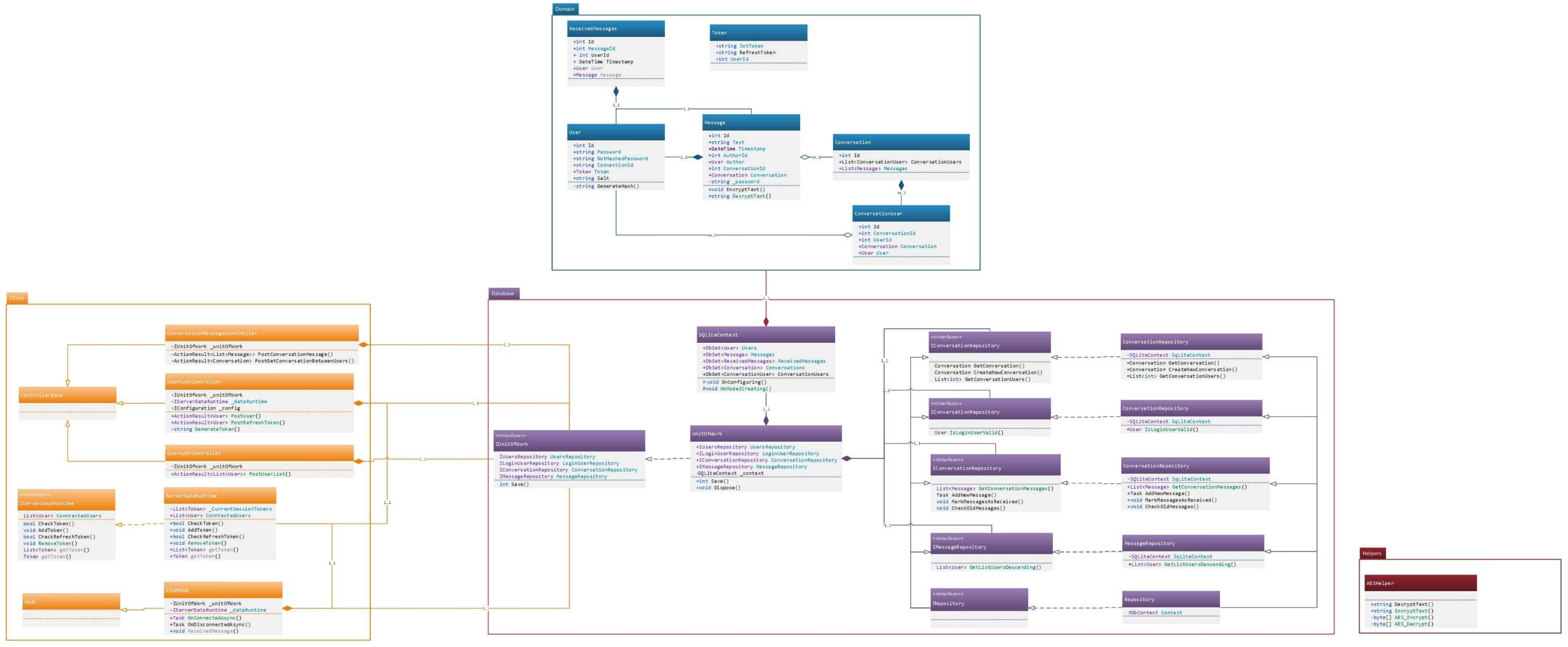



Login Process
Homeview
HomeController
RestApiContext
UserAuthController
LoginUserRepository
ServerDataRuntime
ClientDataRuntime
UsersPanel
UserListController
UsersRepository
Database
User Click Login Button
show loading message
Build connection
Auth user request
Check user
Query
User
alt
user!=null
User
Generate Token
Add Token
User with Token
User with Token
Set current User
New
Build connection
Get Users Request
alt
Authorize = false
Authorize
Unauthorized
Authorize = true
Logout
Logout
Get Users
query
Users
USers
Users
Login
Load Users
loop
for each user
Panel
user == null
Exception
Bad User Message

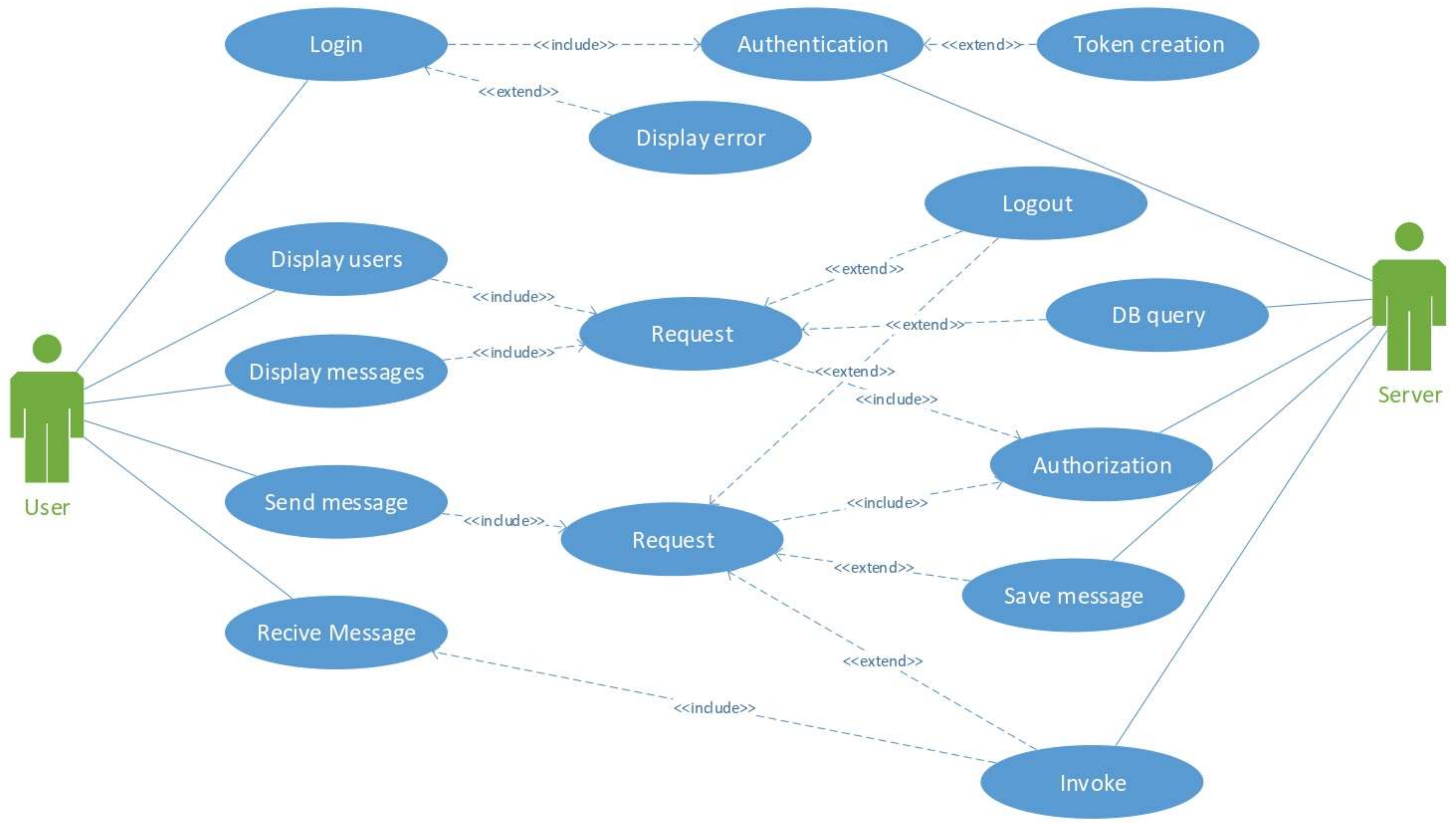
Client class diagram



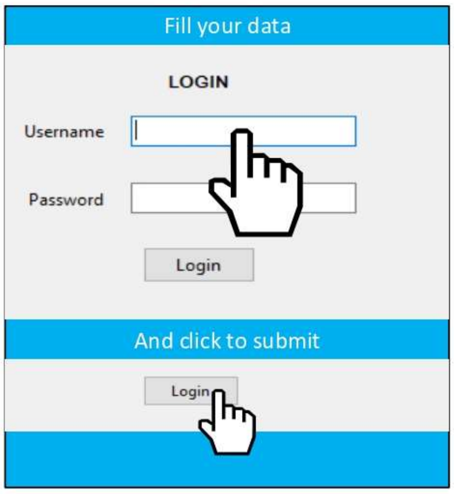
[illegible]

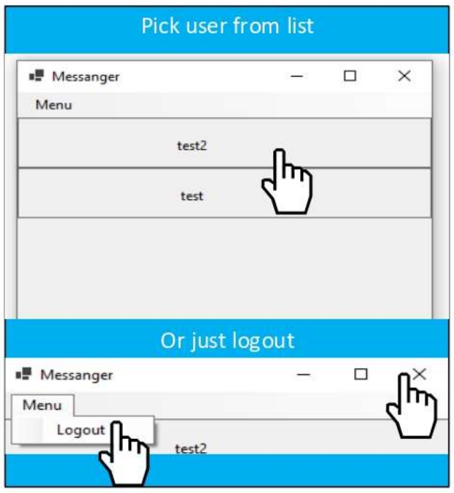

Domain
ReceivedMessages
+int Id
+int MessageId
+ int UserId
+ DateTime Timestamp
+User User
+Message Message
Token
+string JwtToken
+string RefreshToken
+int UserId
User
+int Id
+string Password
+string NotHashedPassword
+string ConnectionId
+Token Token
+string Salt
-string GenerateHash()
Message
+int Id
+string Text
+DateTime Timestamp
+int AuthorId
+User Author
+int ConversationId
+Conversation Conversation
-string _password
+void EncryptText()
+string DecryptText()
Conversation
+int Id
+List<ConversationUser> ConversationUsers
+List<Message> Messages
ConversationUser
+int Id
+int ConversationId
+int UserId
+Conversation Conversation
+User User
1..1
1..1
1..1
m..1
m..1
m..1
Clinet
ConversationMessagesController
-IUnitOfWork _unitOfWork
-ActionResult<List<Message>> PostConversationMessage()
-ActionResult<Conversation> PostGetConversationBetweenUsers()
ControllerBase
UserAuthController
-IUnitOfWork _unitOfWork
-IServerDataRuntime _dataRuntime
-IConfiguration _config
+ActionResult<User> PostUser()
+ActionResult<User> PostRefreshToken()
-string GenerateToken()
UserAuthController
-IUnitOfWork _unitOfWork
+ActionResult<List<User>> PostUserList()
<<Interface>>
IServerDataRuntime
List<User> ConntectedUsers
bool CheckToken()
void AddToken()
bool CheckRefreshToken()
void RemoveToken()
List<Token> getToken()
Token getToken()
ServerDataRuntime
-List<Token> _CurrentSessionTokens
+List<User> ConntectedUsers
+bool CheckToken()
+void AddToken()
+bool CheckRefreshToken()
+void RemoveToken()
+List<Token> getToken()
+Token getToken()
Hub
ChatHub
-IUnitOfWork _unitOfWork
-IServerDataRuntime _dataRuntime
+Task OnConnectedAsync()
+Task OnDisconnectedAsync()
+void ReceivedMessage()
1..1
1..1
1..1
1..1
1..1
Database
SQLiteContext
+DbSet<User> Users
+DbSet<Message> Messages
+DbSet<ReceivedMessages> ReceivedMessages
+DbSet<Conversation> Conversations
+DbSet<ConversationUser> ConversationUsers
#void OnConfiguring()
#void OnModelCreating()
<<Interface>>
IUnitOfWork
IUsersRepository UsersRepository
ILoginUserRepository LoginUserRepository
IConversationRepository ConversationRepository
IMessageRepository MessageRepository
int Save()
UnitOfWork
+IUsersRepository UsersRepository
+ILoginUserRepository LoginUserRepository
+IConversationRepository ConversationRepository
+IMessageRepository MessageRepository
-SQLiteContext _context
+int Save()
+void Dispose()
<<Interface>>
IConversationRepository
Conversation GetConversation()
Conversation CreateNewConversation()
List<int> GetConversationUsers()
ConversationRepository
-SQLiteContext SqLiteContext
+Conversation GetConversation()
+Conversation CreateNewConversation()
+List<int> GetConversationUsers()
<<Interface>>
IConversationRepository
User IsLoginUserValid()
ConversationRepository
-SQLiteContext SqLiteContext
+User IsLoginUserValid()
<<Interface>>
IConversationRepository
List<Message> GetConversationMessages()
Task AddNewMessage()
void MarkMessagesAsReceived()
void CheckOldMessages()
ConversationRepository
-SQLiteContext SqLiteContext
+List<Message> GetConversationMessages()
+Task AddNewMessage()
+void MarkMessagesAsReceived()
+void CheckOldMessages()
<<Interface>>
IMessageRepository
List<User> GetListUsersDescending()
MessageRepository
-SQLiteContext SqLiteContext
+List<User> GetListUsersDescending()
<<Interface>>
IRepository
Repository
#DbContext Context
1..1
1..1
1..1
1..1
1..1
1..1
Helpers
AESHelper
+string DecryptText()
+string EncryptText()
-byte[] AES_Encrypt()
-byte[] AES_Decrypt()

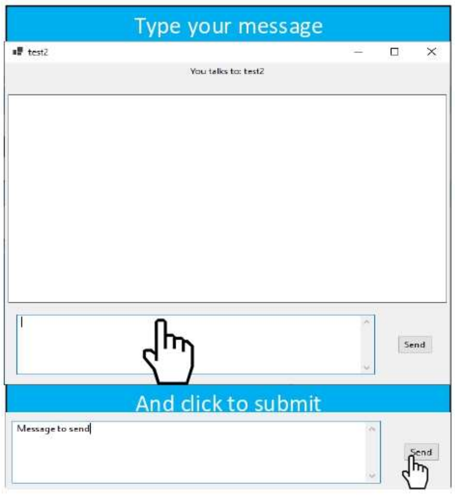
Use case diagram



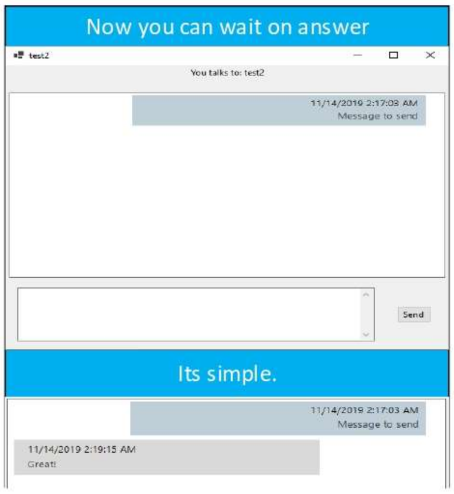
User Guide







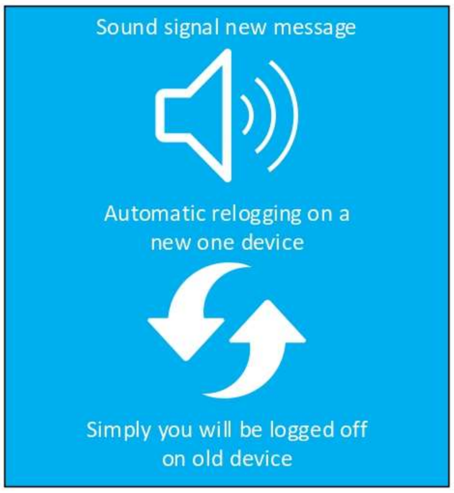
Message to send



11/14/2019 2:17:03 AM
Message to send

11/14/2019 2:17:03 AM
Message to send

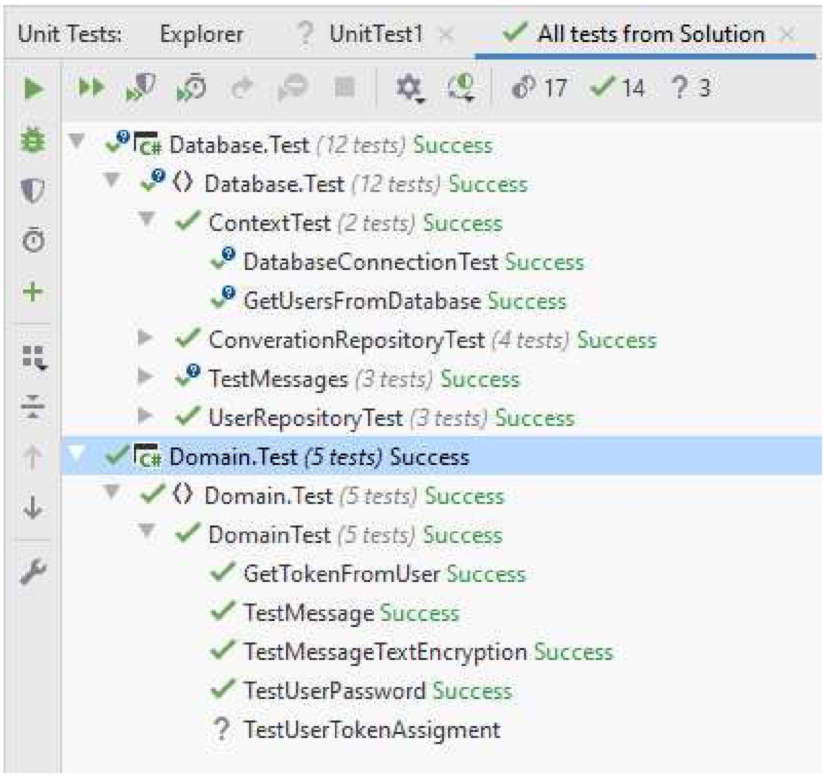
11/14/2019 2:19:15 AM
Great!


Sound signal new message
Automatic relogging on a new one device
Simply you will be logged off on old device

The data is never stored on your device
Only one conversation per user

Testing

On the one hand, the test object was the application code, on the other hand the application itself on the user's side. To this end, techniques called white and black box tests were used. Testing the white box allowed to test the program based on the internal structure of the application. To achieve this, applications were tested using unit tests where they could be used.



```
Unit Tests:   Explorer   ? UnitTest1 ×   ✓ All tests from Solution ×
17  ✓ 14  ? 3
▼ Database.Test (12 tests) Success
  ▼ Database.Test (12 tests) Success
    ▼ ✓ ContextTest (2 tests) Success
          DatabaseConnectionTest Success
          GetUsersFromDatabase Success
    ▶ ✓ ConverationRepositoryTest (4 tests) Success
    ▶ TestMessages (3 tests) Success
    ▶ ✓ UserRepositoryTest (3 tests) Success
▼ ✓ Domain.Test (5 tests) Success
  ▼ ✓ Domain.Test (5 tests) Success
    ▼ ✓ DomainTest (5 tests) Success
        ✓ GetTokenFromUser Success
        ✓ TestMessage Success
        ✓ TestMessageTextEncryption Success
        ✓ TestUserPassword Success
        ? TestUserTokenAssigment
```

Because the application has a complex structure, it uses several frameworks and the logic has been divided between the server and the client. Therefore, the program was also tested as a product. For this purpose, we will use a technique called Black Box. Testing consisted of testing individual functions of the application by following specific steps and using various types of input data (normal, exceptional and extreme); making sure that the data received during the tests are as expected.

Description	Data	Expected Result	Result
Login using correct user and password	admin password	User login into system	Ok
Login using wrong user	Wrong password	Display error above inputs	Ok
Login using wrong password	admin password2	Display error above inputs	Ok
Login using user with upper case	Admin password	Display error above inputs	OK
Fill user as long text	<<Long String>>	Display error above inputs	Ok
Login and see users in home window	admin password	User login into system and users are displayed	Ok
Login and see users in home window	admin password	User login into system and users are displayed	Ok
Logout user by Menu->Logout	Click Menu->Logout	User logged out	OK
Login again to same account	admin password	User login into system and users are displayed	Ok
Close program and lunch again	Close program	User logged out	Ok
Login on one device then login on same account using second device	admin password	User login into system and users are displayed, first program has been logged out	Ok

Description	Data	Expected Result	Result
Open new conversation with user test	Click user test	Window open with user conversation, window title is user name, header display correct user	Ok
Open new conversation with user test and send message	Click user test Write message: test Submit	Window opens, then message has been sent and displayed above in message box	Fail; Message not displayed, no record in database
Close window with test conversation	Close window button	Windows has been closed	Ok
Open again conversation with test	Click user test	Window opens with user conversation, window title is user name, header display correct user, messages has displayed in message box	OK
Type new message in reopen window	Click user test Write message: test2 Submit	Window opens, then message has been sent and displayed above in message box	OK
Send message from different user to admin	Open new instance of program Login into test Click user admin Write message: test2 Submit	Window opens with user conversation, window title is user name, header display correct user, messages has displayed in message box, message has been sent, Sounds has been played on admin program and new message displayed on both programs	OK

Evaluation

The program meets all functional requirements and most non-functional ones. The program has many security features that will prevent attacks and attempts to steal sensitive data. The architecture of the program allows for further extension and improvement of the program in the long term, also to easier maintain application. Currently, the application can operate on the local network and on the Internet after appropriate configuration. Using the appropriate frameworks, it was possible to relieve the database of unnecessary queries (pulling). The main assumption was to focus more on the security and structure of the application than over its appearance because the view of application is independent of the entire application, therefore the application template needs improvement or purchase a ready-made template. Now, the template is rendering rather slowly and is noticeable to the user. Messages in conversations are currently not copiable for security reasons, which should be noted (requirements) when upgrading the template.

Some of the tests operate on the base database, therefore they should not be run after application release. Those tests should be converted to work on in-memory database. The application currently has several known and detected bugs inside the server code. These errors do not significantly affect the operation of the program and their repair is not quite complicated and can be implemented at the next iteration. One of them is the error that appears after the newly added user that user want to contact. The error appears because there is a problem creating a conversation when for first open conversation the window. Currently, to continue the conversation, all is need that reopening window.

The customer also has the option of implementing the connection certificate validation in the appropriate place in the program. This has not been configured yet. Configuration also requires the appropriate setting of ports and IP addresses for the proper connections of the application. Correct server-side firewall configuration will also be required.

Currently, users can enjoy shared communication in a secure environment after accessing the program. It is also important that login data should be manually set by administrator and there is no tool to do it.

References

Adriancs. (2019, 10 154). *Code project*. Retrieved from C# AES 256 bits Encryption Library with Salt: https://www.codeproject.com/Articles/769741/Csharp-AES-bits-Encryption-Library-with-Salt

Hamedani, M. (2019, 10 1). *Programming with Mosh*. Retrieved from Layered Architecture in ASP.NET Core Applications: https://programmingwithmosh.com/asp-net/layered-architecture/