uChat Project:

Complete Technical Project Encyclopedia

Version: 2.1

Technology stack: C# .NET 8, WinUI 3, SQLite, TCP/IP.

By Alexey

Contents

Part 1: Architecture and Protocol (Core)

This is the foundation of the system. The files are identical on the client and the server.

1. Transfer Protocol (TCP Layer)

A custom protocol is used on top of TCP to solve the problem of packet sticking in the TCP stream.

- Packet format: [Header: 4 bytes (Length)] + [Body: JSON Bytes].
- Reading logic:
    1. Read the first 4 bytes -> convert them to int (this is the message size).
    2. Read exactly as many bytes as the size specifies.
    3. Deserialize the bytes into a ProtocolMessage object.

2. Data Structure (ProtocolMessage.cs)

The main container class for data exchange.

- Type (MessageType enum): The command type.
    - Examples: Login, Register, SendMessage, SendImage, CreateGroup, Heartbeat.
- Data (string?): The main payload.
    - Text: "Hi, how are you?"
    - File/Image: a Base64 string.
    - Lists: a JSON array (for example, a list of users [{"Id":1, "Username":"Alex"}]).
- Parameters (Dictionary<string, string>?): Metadata.
    - Keys: sender, targetUsername, messageId, filename, timestamp, groupId, isPrivate.

Part 2: Server Side (uchat_server)

1. Program.cs & Server.cs

- Program.cs: Entry point. Reads the port (1337). Starts the main server and the background scheduler thread.
- Server.cs:
    - _clients: List<ClientHandler> - the list of active connections.
    - TcpListener: Listens on the port. When a new connection appears, it creates a ClientHandler and runs it in Task.Run.
    - BroadcastToAll: Serializes the message once and sends it to everyone in the _clients list.
    - CheckAndSendScheduledMessagesAsync: Background process (loop). Every 10 seconds it checks the DB for messages whose time has arrived.

2. ClientHandler.cs (Processing Logic)

One instance for each connected user.

- HandleClientAsync(): An infinite loop that reads the stream.
- ProcessMessageAsync(message): The main switch that routes the logic (Login -> HandleLoginAsync, SendMessage -> HandleSendMessageAsync).

3. DatabaseContext.cs (Data)

Work with SQLite (uchat.db). Uses WAL (Write-Ahead Logging) mode for performance.

- Main tables:
    - Users: Profiles (Id, Username, PasswordHash, DisplayName, AvatarData).
    - Messages: Global chat history.
    - DirectMessages: Private messages (SenderId, RecipientId).
    - Groups / GroupMembers / GroupMessages: Group structure.
    - ScheduledMessages: Scheduled messages.
    - FileBlobs: Stores heavy data (files) separately to speed up chat history loading.
- DeleteUser(userId): Implements soft deletion (wipes the data, keeps the ID) and runs VACUUM to compress the database file.

4. AuthenticationService.cs

- HashPassword: Hashing through SHA256.
- Authenticate: Checks the login/hash pair.

Part 3: Client Side (uchat)

1. MainPage.xaml.cs (UI Controller)

Controls the display and state.

- State variables:
  - _currentChatTarget: string? (null = Global chat, otherwise = username of the contact).
  - _currentUsername: Login of the current user.
  - _currentDisplayName: Display name.
- Collections (Data Binding):
  - _messages: ObservableCollection<MessageViewModel> - the list of messages on the screen.
  - _usersList: Contact list in the sidebar.
  - _allUsersCache: Full cached list of all users (for search).

2. NetworkClient.cs

- ConnectAsync: Initializes TcpClient, establishes the connection, and starts two background processes: one for reading incoming data and one for processing the outgoing queue.
- ReceiveMessagesAsync: Background reading loop. First reads 4 header bytes (packet size), then reads the body, deserializes it, and calls the MessageReceived event.
- SendMessageAsync / ProcessSendQueueAsync: Implements safe sending through a queue (ConcurrentQueue). Messages are not written directly to the stream. They are added to a queue that is processed sequentially by a separate thread (ProcessSendQueueAsync) to avoid conflicts and socket overload.
- ReconnectAsync: Closes the current resources and automatically tries to establish a new connection.

3. Models and Services

- MessageViewModel: Contains UI logic (IsVoice, IsImage, IsOwnMessage). Controls element visibility in XAML.
- VoiceMessageManager: Records audio through MediaCapture and returns Base64.
- SettingsService: Stores JSON with settings (theme, download path).

Part 4: Detailed Function Description (Input -> Logic -> Output)

This section describes how specific functions work, which variables they receive, and what they do.

1. Authorization (Login)

- Function (Client): AttemptConnection(bool isLogin)
  - Input variables: UsernameTextBox.Text, PasswordBox.Password.
  - Logic:
    1. Creates a connection through _networkClient.ConnectAsync.
    2. Builds a packet: Type = Login, Parameters = { username, password }.
    3. Sends it to the server.
- Function (Server): HandleLoginAsync
  - Input data: message.Parameters["username"], message.Parameters["password"].
  - Logic:
    4. _authService.Authenticate(...) hashes the password and searches in the DB.
    5. If found: fills _currentUser.
  - Output: LoginResponse packet with profile data (userId, displayName, avatar).

2. Sending a Message (Send)

- Function (Client): SendMessage()
  - Input variables: MessageTextBox.Text, _currentChatTarget (state).
  - Logic:
    1. Creates a local MessageViewModel and adds it to _messages (so the user sees it instantly).
    2. Checks _currentChatTarget:
       - If null (Global) -> Type = SendMessage.
       - If group -> Type = SendGroupMessage, adds groupId.
       - If user -> Type = SendPrivateMessage, adds targetUsername.
    3. Sends the packet.
- Function (Server): HandleSendMessageAsync / HandleSendPrivateMessageAsync
  - Input data: message.Data (text), _currentUser.
  - Logic:

1. _db.CreateMessage(...): Writes to the Messages or DirectMessages table.

- Output:
  1. Global: _server.BroadcastToAll.
  2. Private: Finds the recipient ClientHandler and calls SendMessageAsync only for that user.

3. Sending Files (Media)

- Function (Client): AttachButton_Click
  - Input data: The selected file through FileOpenPicker.
  - Logic:
    1. Reads the file into a stream -> converts the byte array into a Base64 string.
    2. Builds a SendFile (or SendImage) packet. Body (Data) = Base64.
    3. Adds the filename parameter.
- Function (Server): HandleSendMediaAsync
  - Logic:
    1. _db.SaveFileBlob(base64) -> stores the heavy data in FileBlobs and returns int BlobId (for example, 55).
    2. Builds a text reference (Ref): <FILE_REF:55|doc.pdf|1024>.
    3. Saves this reference as the message text.
    4. Sends the reference to clients (the file itself is not sent, so the channel is not overloaded).
- Function (Receiving Client): AddMessageToView
  - Input data: Text <FILE_REF:55...>, BlobId from the parameter.
  - Logic:
    1. Sees the tag. Understands that this is a file.
    2. If it is an image -> automatically sends GetFileContent(55).
    3. If it is a file -> shows the "Download" button.

4. Groups (Group Chat)

- Function (Client): CreateGroup_Click
  - Input data: List of selected userId values from the CreateGroupDialog.
  - Logic: Sends a CreateGroup packet where Data = JSON list of IDs.
- Function (Server): HandleCreateGroupAsync
  - Logic: _db.CreateGroup (writes to the Groups and GroupMembers tables).
- Function (Server): SendToGroupMembersAsync
  - Use: During SendGroupMessage.

- Logic:
    1. _db.GetGroupMemberIds(groupId) -> receives the list of member IDs.
    2. Filters the active _clients list.
    3. Sends the message only to users who are in the group.

5. Search

- Function (Client): UserSearchBox_TextChanged
    - Input data: Search text.
    - Logic (Hybrid):
        1. First filters the local _allUsersCache.
        2. If there are too few local results -> starts a timer (debounce) and sends SearchUsers to the server.
- Function (Server): HandleSearchUsersAsync
    - Logic: SQL SELECT ... WHERE Username LIKE %query%. Returns the found list.

6. Notifications

- Function (Client): ShowNotification
    - Input data: title (sender name), content (text), isPrivate.
    - Logic:
        1. Checks SettingsService settings (All, Mentions Only, None).
        2. Checks window focus through the Win32 API (IsAppInForeground). If the user is currently looking at this chat, the notification is blocked.
        3. Trims long text (>100 characters).
        4. Uses ToastContentBuilder to show a Windows system Toast notification.

7. Replying to a Message (Reply)

- Function (Client - UI): ReplyContext_Click -> EnterReplyMode
    - Logic: Sets the _replyingToMessage variable. Shows the ReplyPanel above the input field with the author name and the original message text.
- Function (Client - Network): SendMessage
    - Logic: If _replyingToMessage != null, adds the "replyToId" parameter to ProtocolMessage.
- Function (Server - DB): GetMessages / GetGroupMessages
    - Logic: Performs a SQL LEFT JOIN of the messages table with itself to get ReplySender and ReplyContent for the message that is a reply. This lets the client display context without extra requests.

8. Security and Password Change

Protection is implemented through hashing. When you register with the password 12345, the database does not store 12345.

1. The server takes 12345.
2. Runs it through the SHA256 algorithm.
3. Receives something like a8f5f167f44f4964e6c998dee827110c.
4. Stores this set of characters.

When you change the password:

1. You enter the old password.
2. The server hashes it and compares it with what is in the database. If it matches, it means it is you.
3. Then it hashes the new password and overwrites the old database record.

9. Database (System Skeleton)

Your SQLite database is a set of tables connected to each other:

- Users: Contains logins, password hashes (encrypted), and links to avatars.
- Messages: Contains the entire Global Chat history.
- DirectMessages: Contains private conversation history (who wrote to whom).
- FileBlobs: The storage area. It contains the files themselves (images, documents) so the message tables are not polluted.
- ScheduledMessages: The notebook. It contains messages that must be sent in the future. The server constantly checks this table.

10. File Auto-Save

- Function (Client): TryAutoSaveFileAsync
    - Logic:

1. Checks SettingsService.IsAutoDownloadEnabled.
2. Uses StorageApplicationPermissions.FutureAccessList to access the folder without opening the dialog again (even after restarting the application).
3. If the token is valid -> saves the file.
4. If not -> ignores it (the user must press the "Download" button manually).

11. Theme and UI Change

- Function (Client): ThemeToggleButton_Click
  - Logic: Toggles _isLightTheme. Saves it in SettingsService.
  - Application: Calls SetAppTheme, which changes the RequestedTheme of the root element (ElementTheme.Dark/Light) and updates color resources (Brushes) on the fly.
- Function (Client - Scaling): OnZoom (Ctrl + Wheel)
  - Logic: Intercepts the scroll event. If Ctrl is pressed, changes the _currentZoom coefficient (from 0.7 to 1.2).
  - Result: Changes the Width and Height of the root RootGrid, which emulates scaling of the whole interface.

12. Profile and Account

- Profile editing:
  - Client: SaveProfile_Click. Sends an UpdateProfile packet with the new name, bio, and Base64 avatar.
  - Server: Updates the Users table. Broadcasts ProfileUpdated to all clients so they update their contact cache.
- Account deletion:
  - Client: DeleteAccountButton_Click -> DeleteAccount packet.
  - Server: HandleDeleteAccountAsync -> _db.SoftDeleteUser.
    - Replaces Username with deleted_GUID.
    - Clears the password and avatar.
    - Runs VACUUM to clean up space.
  - Client: Receives confirmation, logs out, and clears the local cache.

13. Stickers and GIF

- Loading stickers:
  - Client: When opening the menu, sends GetStickerPacks.
  - Server: Scans the Stickers folder and returns the list of packs.
  - Client: When selecting a pack, sends GetStickerPackContent. After receiving the file list, it displays them. The images themselves are loaded lazily through GetSticker and cached in _stickerCache.
- Sending (Mechanism):

- A sticker is sent as text: <STICKER:PackName|FileName>.
- A GIF is sent as text: <GIF:FileName>.
- The server does not store the image itself in the message DB, only this tag.

14. Search

Chat/user search (UserSearchBox):

- Works in hybrid mode. First filters the local _allUsersCache list. After 500 ms (debounce), sends SearchUsers to the server to search global users.

- Message search (ChatSearchBox):
  - Client: Sends a SearchHistory packet with the query text and targetUsername (or groupId).
  - Server: Runs SQL LIKE %query% in the message tables. Returns the list of found messages.
  - Client: When a result is clicked, uses GetHistoryAroundId (Jump to message) to load the message context.

15. Chat Management (Deletion and Leaving)

- Deleting a chat (Private):
  - Client: DeleteChat_Click. Parameter forEveryone (checkbox).
  - Server (HandleDeleteChatAsync):
    - If forEveryone = true: Deletes records from DirectMessages.
    - If false: Updates the ChatSettings table, setting LastClearedMessageId to the ID of the last message. This hides old messages for this user but leaves them in the database.
- Leaving a group:
  - Client: If the user is not an admin, the "Leave Group" button appears. Sends a LeaveGroup packet.
  - Server: Deletes the record from the GroupMembers table.

Deleting a group (Admin):

- Client: If the user is an admin -> DeleteGroup.
- Server: Deletes the group from Groups, all members from GroupMembers, and the history from GroupMessages. Broadcasts a GroupDeleted notification.

16. Voice Messages

- Function (Client - Recording): VoiceButton_Click

    - Logic: Uses VoiceMessageManager and MediaCapture. Records audio into memory -> converts it to Base64 -> sends a message with the tag <VOICE:base64_data>.
- Function (Client - Playback): PlayVoiceMessage_Click.
    - Logic: Uses one global MediaPlayer. Decodes Base64 back into a stream. A timer updates the progress bar.

17. Replying to a Message (Reply)

- UI: ReplyContext_Click activates reply mode (_replyingToMessage).
- Network: When sending, the replyToId parameter is added.
- Display: The server fills the ReplyToSender and ReplyContent fields (through SQL Join), so the client immediately shows what the reply refers to without extra requests.

Part 5: Special Tag Reference (Rich Content)

The system recognizes the content type by a text tag at the beginning of the message.

Type	Format in DB / Protocol	How the Client Handles It (AddMessageToView)
Voice	<VOICE:base64_data>	Decodes Base64, shows a Slider and a Play button.
GIF	<GIF:filename>	Shows the animation. If it is not in cache -> requests GetGif.
Sticker	<STICKER:pack|name>	Shows the image from the sticker pack.
Image (Ref)	<IMG_REF:blobId|name>	Shows a placeholder. Loads GetFileContent in the background.
File (Ref)	<FILE_REF:blobId|name|size>	Shows the file icon, name, and size. Click -> Download.
Image (Legacy)	<IMAGE:base64>	Old format. The image is embedded into the text.

Part 6: Important Technical Nuances

1. **Threading UI:**

All events from NetworkClient arrive from a background thread. In MainPage.xaml.cs, _dispatcherQueue.TryEnqueue(() => { ... }) must be used for any UI update (adding to ObservableCollection, changing text); otherwise the program will crash.

2. **Memory Cleanup (DB Vacuum):**

When messages or users are deleted (DeleteChat, DeleteAccount), SQLite does not automatically free disk space. Therefore, DatabaseContext calls the VACUUM; command after deletion. It rebuilds the database file and reduces its size.

3. **Password Security:**

Passwords are never stored in plain text after registration (except for the moment of login transmission). Only the hash is stored in the database (AuthenticationService). When changing the password, the server first requires confirmation of the old password (checks its hash).

Part 7. Short File Glossary

- Program.cs: The Start button for the server.
- Server.cs: The dispatcher. It accepts calls (connections) and distributes them to operators (ClientHandler).
- ClientHandler.cs: The operator. Works with a specific client. Knows how to respond to Login, SendFile, and so on.
- DatabaseContext.cs: The librarian. Only it knows how to read and write to the book (the uchat.db file).
- MainPage.xaml.cs: The face. Controls everything you see: buttons, lists, animations.
- NetworkClient.cs: The courier. Its job is simply to take a packet and deliver it to the server, or bring a packet from the server. It does not think about what is inside.