目次

1. 概要-------------------------------------------------2
2. 機能要求事項------------------------------------------2
3. 技術的要求事項---------------------------------------3
4. 開発環境要件------------------------------------4
5. システムアーキテクチャ--------------------------4
6. 通信プロトコル--------------------------------------5
7. データ構造--------------------------------------------6
8. セキュリティおよび認証--------------------------6
9. パフォーマンス要求事項--------------------------7
10. テスト計画----------------------7
11. 配布---------------------------------------------------8
12. メンテナンス --------------------------------------8
13. DB 設定 --------------------------------------------------10
14. SQLクエリ-------------------------------------------------11

1.概要

この仕様書はTCP/IP プロトコルを使用するチャット プログラムの要求事項を説明します。このチャット プログラムは、クライアント サーバー モデルに基づいて機能します。ユーザーはクライアント アプリケーションを介してサーバーに接続し、他のユーザーとチャットできます。 Mas

2.機能要求事項  
  
ユーザー登録: 新しいユーザーは、ユーザー名とパスワードを提供して会員登録する必要があります。  
ログインおよびログアウト: 登録されたユーザーは、自分のアカウントにログインしてログアウトできる必要があります。  
オンライン ユーザー リスト: サーバーは、現在オンラインで接続されているユーザーのリストを保持し、クライアントにそれを提供する必要があります。  
プライベート メッセージ: ユーザーは、特定のユーザーにプライベート メッセージを送信できる必要があります。  
グループチャット: ユーザーは、複数のユーザーが参加するグループチャットルームに参加できる必要があります。  
メッセージの送信と受信: ユーザーはメッセージを送信し、他のユーザーからメッセージを受信できる必要があります。  
絵文字とファイルの転送: ユーザーは絵文字を送信するか、ファイルを共有できる必要があります。  
接続履歴: サーバーはユーザーの接続記録を保持する必要があります。

スタート -> クライアントアプリケーションの実行  
  
クライアントアプリケーション:  
- ログインまたは会員登録  
- ユーザー認証要請(TCP接続)  
- ユーザー認証結果確認  
- ログイン成功  
- 会員加入成功  
- 失敗時のエラー処理  
  
- オンラインユーザーリストのリクエスト(TCP接続)  
- オンラインユーザーリストの受信  
  
- プライベートメッセージ送信  
- 受信者選択  
- メッセージ作成  
- メッセージ送信リクエスト(TCP接続)  
- メッセージ送信結果の確認  
- 転送 成功   
- 失敗時のエラー処理  
  
- グループチャット参加  
- グループチャットルーム選択  
- グループチャット参加要請(TCP接続)  
- 参加結果確認  
- 参加成功  
- 失敗時のエラー処理  
  
- メッセージ受信待ち  
- 新しいメッセージ受信(TCP接続)  
- メッセージ出力  
  
- 絵文字転送  
- 絵文字選択  
- 絵文字転送リクエスト(TCP接続)  
- 転送結果確認  
- 転送 成功  
- 失敗時のエラー処理  
  
- ファイル転送  
- ファイル選択  
- ファイル転送リクエスト(TCP接続)  
- 転送結果確認  
- 転送 成功  
- 失敗時のエラー処理  
  
- ログアウト  
- ログアウト要求(TCP接続)  
- ログアウト結果の確認  
- ログアウト成功  
- 失敗時のエラー処理  
  
クライアントアプリケーション終了→終了

3.技術的要求事項  
  
TCP/IP プロトコル: チャット プログラムは、TCP/IP プロトコルを使用してクライアントとサーバ間の通信を実行する必要があります。  
マルチクライアント サポート: サーバーは同時に複数のクライアントとの接続を管理できる必要があります。  
ユーザー認証および権限管理: サーバーはユーザーの認証を行い、適切な権限を付与してユーザー間の適切な相互作用を保証する必要があります。  
データ暗号化: ユーザー間の通信は暗号化され、データの安全な転送を保証する必要があります。  
エラー処理: プログラムはエラーに対する適切な処理を行い、例外状況に対する安定性を維持する必要があります。そのためには、例外処理およびエラー メッセージを提供する必要があります。  
  
オフラインメッセージ処理:ユーザーがオフライン状態の時、他のユーザーから受け取ったメッセージはサーバーが保管し、ユーザーが再びオンライン状態になるとそのメッセージを伝達する必要があります。  
ロギングおよび監視: サーバーはログ ファイルを作成して、システムの状態とユーザー アクティビティを記録する必要があります。 また、システム監視により、パフォーマンスの問題を特定し、措置する必要があります。  
ユーザーインターフェイス: クライアント アプリケーションは、ユーザーが便利にチャットを実行できる直感的なインターフェイスを提供する必要があります。 これには、メッセージ入力、ユーザー リストの表示、メッセージ ヒストリーの表示などを含める必要があります。  
スケーラビリティ: システムは、必要に応じてサーバー クラスタリングやロード バランシングなどのスケーラビリティ機能をサポートする必要があります。  
安全性: チャット プログラムは、適切なセキュリティ対策を適用して、ユーザー データの安全性を保証する必要があります。 暗号化、脆弱性の補完、認証および権限管理などのセキュリティ機能が必要です。

4.開発環境要件  
  
言語: 選択された言語に基づいてチャット プログラムを開発する必要があります。 例えば、Java、C++、Pythonなどを使用することができます。  
フレームワークおよびライブラリ: 必要に応じて、適切なフレームワークまたはライブラリを使用して開発を支援する必要があります。 例えば、Spring、Django、Socket.IOなどを使用することができます。  
オペレーティング システム: サーバーとクライアント アプリケーションの両方でサポートされているオペレーティング システムを選択する必要があります。 例えば、Windows、Linux、macOSなどをサポートできます

5.システムアーキテクチャ  
  
クライアント - サーバー モデルに基づいています。 クライアントはユーザーインターフェイスを提供し、サーバーはクライアント間の通信を調整します。  
クライアント アプリケーションは、ユーザーがサーバーに接続し、チャット機能を実行できる機能を提供しますする必要があります。そのためには、例外処理およびエラー メッセージを提供する必要があります。する必要があります。そのためには、例外処理およびエラー メッセージを提供する必要があります。

6.通信プロトコル  
  
TCP/IP プロトコルを使用してクライアントとサーバー間の通信を行います。  
クライアントは、サーバーに接続するためにサーバーのIPアドレスとポート番号を使用します。  
クライアントとサーバー間のデータ交換は、TCP接続により安定的かつ信頼性の高い方法で行われます。

7.データ構造  
  
ユーザー情報: ユーザー名、パスワード、接続状態などの情報を含むユーザー オブジェクトを使用します。  
メッセージ形式: メッセージは、送信者、受信者、内容、タイムスタンプなどのフィールドで構成されます。  
オンライン ユーザー リスト: サーバーは、現在オンラインで接続されているユーザーのリストを維持するためのデータ構造を使用します。

8.セキュリティおよび認証  
  
ユーザー認証: ユーザーは、会員登録時に作成されたアカウントにログインして認証を受ける必要があります。  
パスワードのセキュリティ: パスワードは暗号化されて保存する必要があり、安全なハッシュアルゴリズムを使用する必要があります。  
データ暗号化: ユーザー間の通信は、SSL/TLSなどの暗号化プロトコルを使用して保護する必要があります。  
脆弱性の補完: プログラムは、セキュリティの脆弱性に対する強力な防御機能を持っている必要があります。 入力検証、SQLインジェクション、クロスサイトスクリプティング(XSS)などの攻撃を防止する必要があります。

9.パフォーマンス要求事項  
  
遅延時間: メッセージの送信と受信の待ち時間は最小限に抑える必要があります。  
同時性処理: サーバーは同時に複数のクライアント要求を処理できる必要があり、スレッドまたは非同期処理を活用できます。  
スケーラビリティ: システムは、数多くの同時ユーザーと大規模なトラフィックに対して拡張可能である必要があります。 サーバー クラスタリング、ロード バランシング、キャッシングなどのテクノロジーを使用できます。

10. テスト計画

単位テスト: 各モジュールと関数に対する単位テストを行う必要があります

パフォーマンステスト: システムが予想される負荷に対してどれほど効率的に動作するかを確認するためのパフォーマンステストを行う必要があります。これには、同時接続者数、メッセージ処理速度などを測定することが含まれます。  
- ユーザーインターフェーステスト: クライアントアプリケーションのユーザーインターフェースをテストして、ユーザーがチャット機能を簡単に利用できるかどうかを確認する必要があります。  
- セキュリティテスト: システムのセキュリティ機能をテストして、暗号化、認証、脆弱性の防止などが正しく機能していることを確認する必要があります。  
- エラーおよび例外処理テスト: 様々なエラー条件と例外状況に対するテストを行い、プログラムが安定的に動作し、適切にエラー処理されていることを確認する必要があります。

11.配布  
  
サーバー環境の構成: チャット プログラムをホストするサーバー環境を構成する必要があります。 これには、オペレーティング システム、Web サーバー、データベースなどのインストールと構成が含まれます。  
クライアント配布: ユーザーがクライアント アプリケーションをダウンロードしてインストールできる配布ファイルを提供する必要があります。 これは、実行ファイルまたはモバイルアプリで提供できます。  
サーバー配布: サーバー アプリケーションを展開して、ユーザーが接続できる形で提供する必要があります。 これには、サーバーの設定、ポート フォワーディングなどのタスクが含まれます。

12.メンテナンス  
  
バグの修正: プログラムの運用中に発生するバグに対する修正作業を行う必要があります。  
機能の改善: ユーザー フィードバックと要件に基づいて機能を改善し、追加する必要があります.. stronger  
セキュリティ更新: システムのセキュリティホールを監視し、新しいセキュリティパッチと更新を適用してシステムの安全性を維持する必要があります

13.DB 設定

ユーザー(User)テーブル:  
  
UserID : INT(Primary Key)、ユーザーID  
Username: VARCHAR,ユーザー名またはニックネーム  
Password: VARCHAR、暗号化されたパスワード  
Email: VARCHAR, ユーザーのメールアドレス  
Registration Date : DATETIME、ユーザー登録日  
Status:ENUM(「Online」、「Offline」)、ユーザ接続状態  
メッセージ(Message)テーブル:  
  
MessageID:INT(Primary Key)、メッセージID  
Sender ID: INT(Foreign Key)、発信者のユーザーID  
Receiver ID: INT(Foreign Key)、受信者のユーザーID  
Content:TEXT、メッセージ内容  
Timestamp:DATETIME、メッセージ送信日時  
グループ(Group)テーブル:  
  
GroupID: INT(Primary Key)、グループID  
Group Name: VARCHAR、グループ名  
CreatorID: INT(Foreign Key)、グループ作成者のユーザーID  
グループ参加者(Group Member)テーブル:  
  
会員ID:INT(Primary Key)、参加者ID  
GroupID: INT(Foreign Key)、参加したグループのID  
UserID: INT(Foreign Key)、グループに参加したユーザーのID

オンライン ユーザー テーブル (Online User)  
online user  
ユーザーID(UserID):Primary Keyで使用される固有の識別子  
ログイン時間(LoginTime):ユーザーのログイン時間  
絵文字 テーブル (Emoticon)  
  
絵文字ID(Emoticon ID):Primary Keyとして使用される固有の識別子  
絵文字ファイル名(Filename):絵文字ファイルのパスまたは名前  
ファイルテーブル (File)  
  
ファイルID(FileID):Primary Keyとして使用される固有の識別子  
ファイル名 (Filename): アップロードされたファイルの名前  
ファイルパス(FilePath):アップロードされたファイルのパス  
個人メッセージテーブル (Personal Message)  
  
メッセージID(Message ID):Primary Keyとして使用される固有の識別子  
発信者(Sender ID):メッセージを送信したユーザーのID  
受信者(Receiver ID):メッセージを受信したユーザーのID  
内容(Content):メッセージの内容  
転送日時(Timestamp): メッセージの送信時間  
グループメッセージテーブル (GroupMessage)  
  
メッセージID(Message ID):Primary Keyとして使用される固有の識別子  
グループID:メッセージが属するグループのID  
発信者(Sender ID):メッセージを送信したユーザーのID  
内容(Content):メッセージの内容  
転送日時(Timestamp): メッセージの送信時間

DBテーブル間の関係:  
  
ユーザテーブルとOnline UserテーブルはユーザID(UserID)を基準にマンツーマン関係を持ち、Online UserテーブルのユーザIDはユーザテーブルのユーザIDを外来キー(Foreign Key)として参照します。  
GroupテーブルとGroupMemberテーブルはグループIDを基準に一対多関係を持ち、GroupMemberテーブルのGroupIDはGroupテーブルのGroupIDを外来キーとして参照します。  
MessageテーブルとUserテーブルは発信者(SenderID)と受信者(ReceiverID)を基準に一大関係を持ちます。

PersonalMessageテーブルのSenderIDとReceiverIDは、ユーザテーブルのユーザIDを外来キーとして参照します。  
  
PersonalMessageテーブルのSenderIDは、ユーザテーブルのユーザIDを参照して、そのメッセージを送信したユーザを識別します。

PersonalMessageテーブルのReceiverIDは、ユーザテーブルのユーザIDを参照して、そのメッセージを受信したユーザを識別します。

GroupMessageテーブルのGroupIDとSenderIDは、それぞれGroupテーブルのGroupIDとUserテーブルのUserIDを外来キーとして参照します。  
  
GroupMessageテーブルのGroupIDは、GroupテーブルのGroupIDを参照して、そのメッセージが属するグループを識別します。

GroupMessageテーブルのSenderIDは、ユーザテーブルのユーザIDを参照して、そのメッセージを送信したユーザを識別します。

14.SQL クエリ

using System;

using System.Data.SQLite;

namespace ChatProgram

{

//DB와의 연결

public class ChatDB

{

private SQLiteConnection \_connection;

public ChatDB(string dbFile)

{

\_connection = new SQLiteConnection($"Data Source={dbFile};Version=3;");

\_connection.Open();

CreateTables();

}

// 유저 테이블 생성

public void CreateTables()

{

string createUserTableQuery = @"CREATE TABLE IF NOT EXISTS User (

UserID INTEGER PRIMARY KEY,

Username TEXT NOT NULL,

Password TEXT NOT NULL,

Email TEXT NOT NULL,

RegistrationDate DATETIME DEFAULT CURRENT\_TIMESTAMP,

Status TEXT DEFAULT 'Offline'

)";

//메시지 테이블 생성

string createMessageTableQuery = @"CREATE TABLE IF NOT EXISTS Message (

MessageID INTEGER PRIMARY KEY,

SenderID INTEGER NOT NULL,

ReceiverID INTEGER NOT NULL,

Content TEXT NOT NULL,

Timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (SenderID) REFERENCES User(UserID),

FOREIGN KEY (ReceiverID) REFERENCES User(UserID)

)";

//그룹 테이블 테이블의 user id를 참조

string createGroupTableQuery = @"CREATE TABLE IF NOT EXISTS Group (

GroupID INTEGER PRIMARY KEY,

GroupName TEXT NOT NULL,

CreatorID INTEGER NOT NULL,

FOREIGN KEY (CreatorID) REFERENCES User(UserID)

)";

// 식별을 위해 그룹 테이블을 참조

string createGroupMemberTableQuery = @"CREATE TABLE IF NOT EXISTS GroupMember (

MemberID INTEGER PRIMARY KEY,

GroupID INTEGER NOT NULL,

UserID INTEGER NOT NULL,

FOREIGN KEY (GroupID) REFERENCES Group(GroupID),

FOREIGN KEY (UserID) REFERENCES User(UserID)

)";

//현재 접속한 유저

string createOnlineUserTableQuery = @"CREATE TABLE IF NOT EXISTS OnlineUser (

UserID INTEGER PRIMARY KEY,

LoginTime DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (UserID) REFERENCES User(UserID)

)";

// 감정을 표현하는 테이블

string createEmoticonTableQuery = @"CREATE TABLE IF NOT EXISTS Emoticon (

EmoticonID INTEGER PRIMARY KEY,

Filename TEXT NOT NULL

)";

// 파일 테이블을 생성

string createFileTableQuery = @"CREATE TABLE IF NOT EXISTS File (

FileID INTEGER PRIMARY KEY,

Filename TEXT NOT NULL,

FilePath TEXT NOT NULL

)";

using (SQLiteCommand command = new SQLiteCommand(\_connection))

{

//사용자 테이블 쿼리 생성 실행

command.CommandText = createUserTableQuery;

command.ExecuteNonQuery();

//메시지 테이블 쿼리생성실행

command.CommandText = createMessageTableQuery;

command.ExecuteNonQuery();

//그룹 테이블 생성쿼리

command.CommandText = createGroupTableQuery;

command.ExecuteNonQuery();

//그룹멤버테이블 생성쿼리 실행

command.CommandText = createGroupMemberTableQuery;

command.ExecuteNonQuery();

//온라인 사용자 테이블쿼리생성실행

command.CommandText = createOnlineUserTableQuery;

command.ExecuteNonQuery();

//감정표현 테이블 생성 쿼리 실행

command.CommandText = createEmoticonTableQuery;

command.ExecuteNonQuery();

// 파일테이블 생성쿼리 실행

command.CommandText = createFileTableQuery;

command.ExecuteNonQuery();

}

}

public void CloseConnection()

{

\_connection.Close();

}

}

}