ITcpServer

SocketEvent

virtual void handleNewConnection() = 0;

virtual ITcpClient *createNewClient() = 0;
virtual void setSocketEvent(ITcpServer::SocketEvent *listener) = 0;
virtual void listen() = 0;
virtual void deleteClient() = 0;

inherit

inheritinherit

TcpServer

void listen() final;
void stop();
void accept();
void setSocketEvent(ITcpServer::SocketEvent *listener) final;
TcpClient *createNewClient() final;
void deleteClient() final;
private:
bool _running;
asio::io_service _service;
asio::signal_set _sig;
asio::ip::tcp::acceptor *_acceptor;
std::list<asio::ip::tcp::socket *> _sockets;
ITcpServer::SocketEvent *_listener;
uint16_t _port;

Server

• bool run();
• void handleNewConnection() final;

• ItcpServer _tcpServer;
• std::shared_ptr<ArraySharedPtrT<IUser>> _users;
• std::shared_ptr<Babel::CommandManager> _commandManager;
• std::shared_ptr<Babel::Server::Database> _database;

Database

public:
explicit Database(std::string const &dbName);
~Database();
using RequestT = std::tuple<int, std::vector<std::string>, std::vector<std::string>>;
void addUser(std::string const &username, std::string const &password, std::string const &email);
void addMessage(std::size_t const from_id, std::size_t const to_id, std::string const &time_created, std::string const &message);
void addFriendRequest(std::size_t const from_id, std::size_t const to_id);
void removeFriendRequest(std::size_t const from_id, std::size_t const to_id);
void addFriend(std::size_t const from_id, std::size_t const to_id);
char checkIfAccountAlreadyExist(std::string const &username, std::string const &email = "");
bool isAccountExistByUsername(std::string const &username);
bool isAccountExistById(size_t const id);
bool isFriend(std::size_t const first_id, std::size_t const second_id);
bool isFriendRequestAlreadySend(std::size_t const first_id, std::size_t const second_id);
user_t *getUserinfosByUsername(std::string const &username);
user_t *getUserinfosByEmail(std::string const &email);
user_t *getUserInfos(size_t const user_id);
std::vector<std::string> *getFriends(size_t const user);
std::vector<std::string> *getFriendsRequests(size_t const user);
std::vector<message_t *> *getMessage(size_t const firstUser_id, size_t const secondUser_id);
private:
void init(const std::string &dbName);
RequestT exec(std::string const &request);
sqlite3 *_db;

InheritInheritInherit

IUser

virtual ~IUser() {};
virtual void initializeConnection(Babel::Request const &) = 0;
virtual void signUp(Babel::Request const &) = 0;
virtual void signIn(Babel::Request const &) = 0;
virtual void sendFriendRequest(Babel::Request const &) = 0;
virtual void handleFriendRequestResponse(Babel::Request const &) = 0;
virtual void askFriendCalling(Babel::Request const &) = 0;
virtual void terminateCall(Babel::Request const &) = 0;
virtual void sendMessage(Babel::Request const &) = 0;
virtual void askMessageHistory(Babel::Request const &) = 0;
virtual void askFriendRequest(Babel::Request const &r) = 0;
virtual void askFriendList(Babel::Request const &) = 0;
virtual void handleCallResponse(Babel::Request const &) = 0;
virtual std::size_t getId() const = 0;
virtual void setId(std::size_t const id) = 0;
virtual void send(unsigned char const ins, void *body = nullptr, std::size_t const bodySize = 0) = 0;
virtual void sendToAnotherWithUsername(std::string const &username, unsigned char const ins, void *body = nullptr, std::size_t const bodySize = 0) = 0;
virtual void sendToAnother(size_t const id, unsigned char const ins, void *body = nullptr, std::size_t const bodySize = 0) = 0;
virtual std::string getIp() const = 0;

inherit

User

*explicit User(ITcpClient *tcpClient_, std::shared_ptr<Babel::CommandManager> commandManager_, std::shared_ptr<Babel::Server::Database> database_, std::shared_ptr<ArraySharedPtrT<IUser>> users_);*
~User() override;
void handleRcvBuffer() final;
void close() final;
void initializeConnection(Babel::Request const &) final;
void signUp(Babel::Request const &) final;
void signIn(Babel::Request const &) final;
void sendFriendRequest(Babel::Request const &) final;
void handleFriendRequestResponse(Babel::Request const &) final;
void askFriendCalling(Babel::Request const &) final;
void handleCallResponse(Babel::Request const &) final;
void terminateCall(Babel::Request const &) final;
void sendMessage(Babel::Request const &) final;
void askMessageHistory(Babel::Request const &) final;
void askFriendRequest(Babel::Request const &r) final;
void askFriendList(Babel::Request const &) final;
*void send(uint8_t const ins, void *body = nullptr, std::size_t const bodySize = 0);*
*void sendToAnotherWithUsername(std::string const &username, unsigned char const ins, void *body = nullptr, std::size_t const bodySize = 0) final;*
*void sendToAnother(size_t const id, unsigned char const ins, void *body = nullptr, std::size_t const bodySize = 0) final;*
std::size_t getId() const final;
void setId(std::size_t const id) final;
std::string getIp() const final;
private:
std::size_t _id;
*ITcpClient *_socket;*
std::shared_ptr<Babel::CommandManager> _commandManager;
std::shared_ptr<Babel::Server::Database> _database;
std::shared_ptr<ArraySharedPtrT<IUser>> _users;

ITcpClient

SocketEvent

explicit SocketEvent() = default;
~SocketEvent() = default;
virtual void handleRcvBuffer() = 0;
virtual void close() = 0;

virtual ~ITcpClient() {};
virtual void setSocketEvent(ITcpClient::SocketEvent *listener) = 0;
virtual Babel::Request const &getRequest() = 0;
virtual void addToMessageQueue(Babel::Message const &msg) = 0;
virtual std::string const getIp() const = 0;

inherit

TcpClient

explicit TcpClient(asio::ip::tcp::socket *socket_);
~TcpClient() override;
void setSocketEvent(ITcpClient::SocketEvent *listener_) final;
Babel::Request const &getRequest() final;
void addToMessageQueue(Babel::Message const &msg) final;
std::string const getIp() const final;
private:
void startToReceive();
void close() noexcept;
void sendMessageInQueue(std::error_code const &err, std::size_t const bytesTransfered);
asio::ip::tcp::socket *_socket;
Babel::Request _request;
ITcpClient::SocketEvent *_listener;
std::queue<Babel::Message> _msgs;
std::mutex _m;