# 在线聊天系统实验报告

一、背景描述

设计在线聊天系统可以学习理解基本的网络通信原理和技术，如 socket 编程、TCP/IP 协议等。掌握在网络环境下实现客户端和服务器端通信的方法和技巧。学习如何在服务器端有效地管理多个客户端的连接和消息交互。学习实现并发处理机制，避免由于一个客户端的阻塞而影响其他客户端的正常通信。设计良好的系统架构和模块化组件，以便于扩展和维护，所以我们合作完成了这个系统来巩固和应用学过的理论知识。

二、总体设计

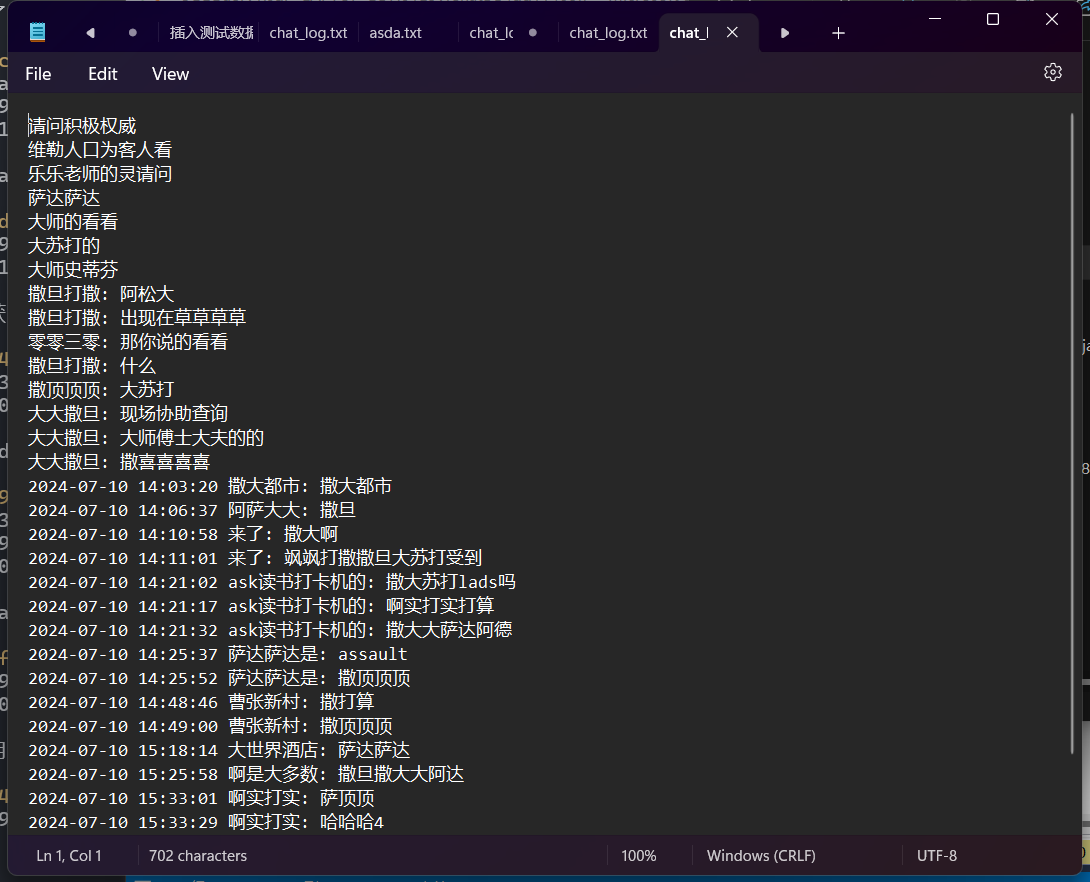
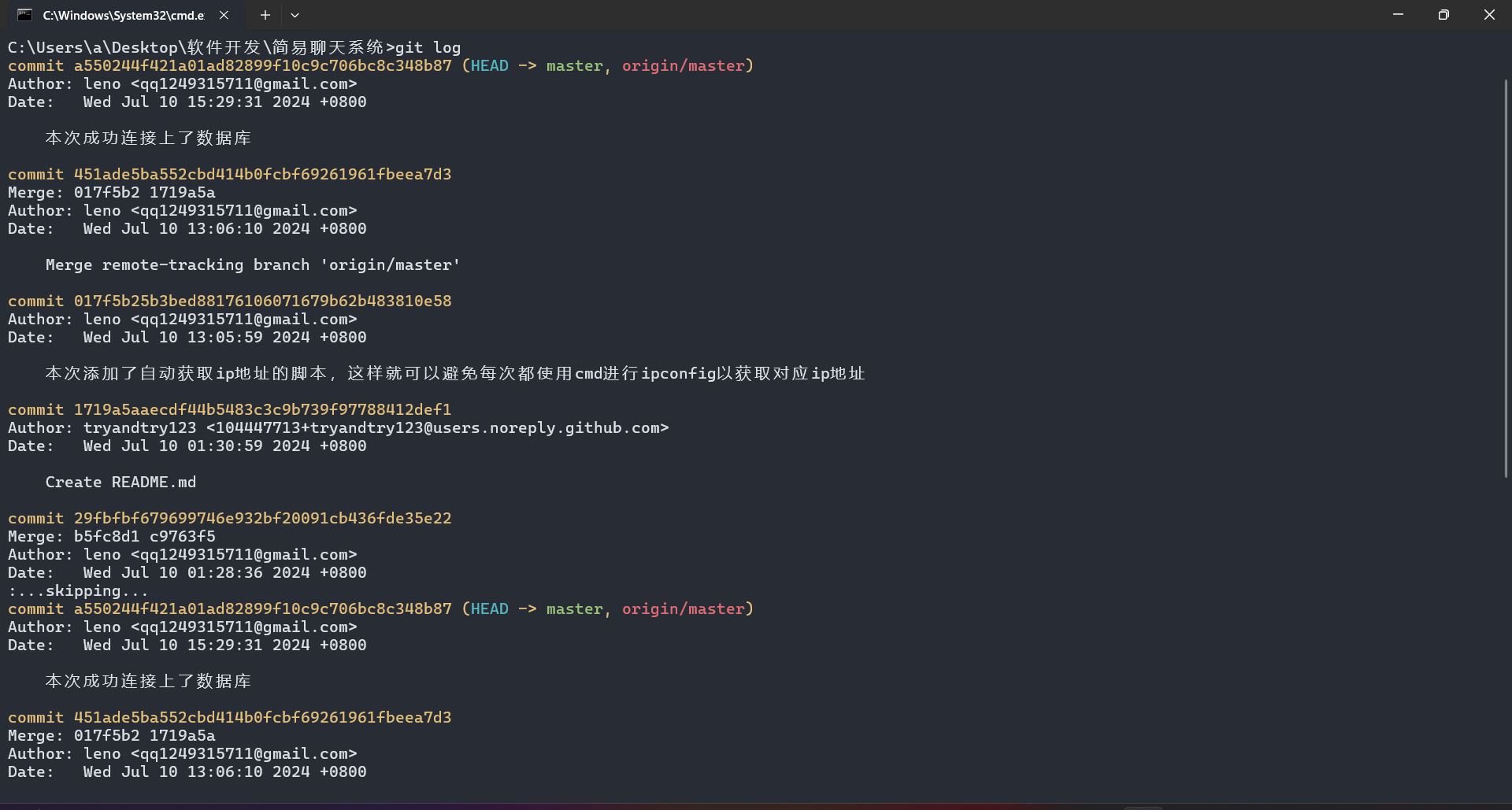
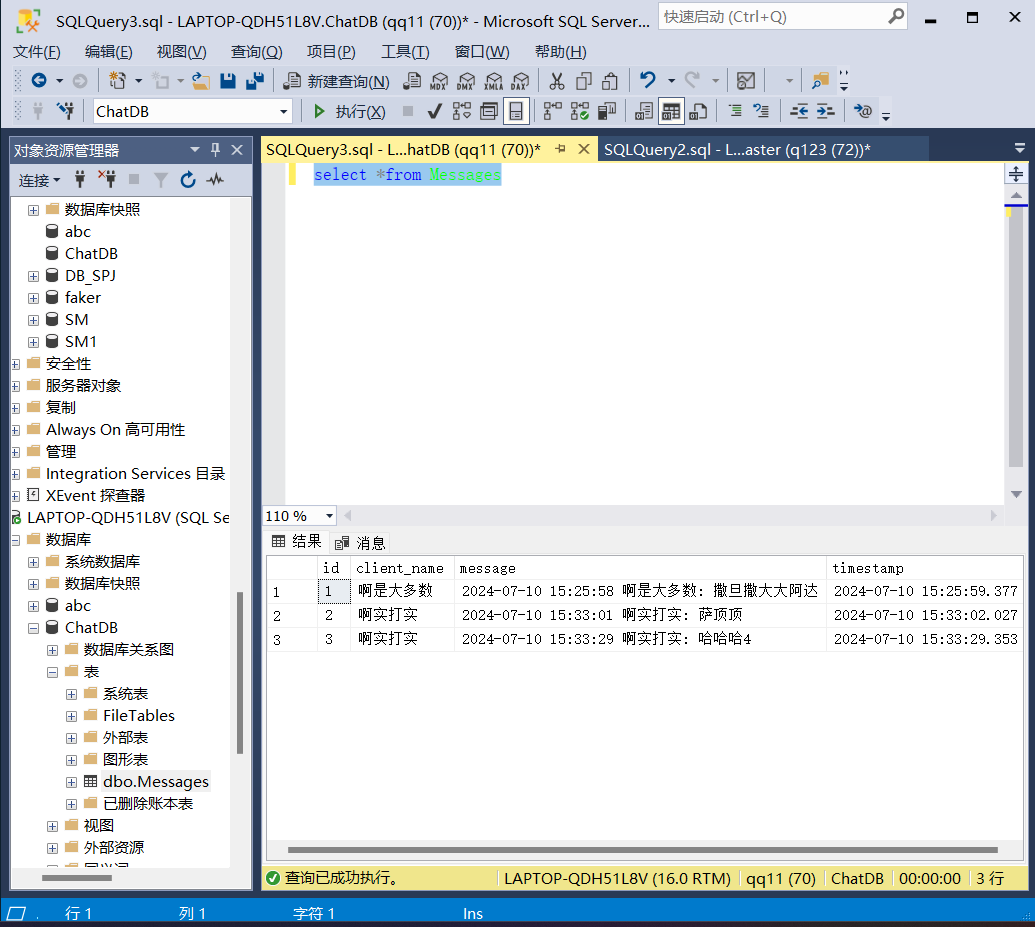
本实验设计了一个简单的聊天系统，该系统包括客户端和服务器端，能够实现基本的消息发送和接收功能。系统采用C/S架构，客户端和服务器端通过Socket进行通信。系统主要包括以下几个部分：

1. ChatClient：客户端程序，负责与服务器端建立连接并发送消息。

2. ChatClientLauncher：客户端启动器，用于启动客户端程序。

3. ChatServer：服务器端程序，负责接收客户端发送的消息并进行广播。

三、数据库设计



四、代码中用到的高级Java技术

1. 集合：

- 在`ChatServer`类中使用了`List`集合来存储连接的客户端。

java

private List<PrintWriter> clientOutputStreams;

2. 多线程：

在`ChatServer`类中，每个客户端连接都会启动一个新的线程来处理消息接收。

java

new Thread(new ClientHandler(clientSocket)).start();

3. 网络编程：

在`ChatClient`和`ChatServer`类中都使用了Socket进行通信。

java

Socket socket = new Socket(serverAddress, serverPort);

ServerSocket serverSocket = new ServerSocket(PORT);

4. 序列化：

在消息传输过程中使用了对象流来实现消息的序列化和反序列化。

java

ObjectOutputStreamout=new ObjectOutputStream(socket.getOutputStream());

ObjectInputStreamin=new ObjectInputStream(socket.getInputStream());

```

五、系统代码

以下是各个主要类的代码实现：

1. ChatClient.java：

java

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class ChatClient {

private String serverAddress;

private int serverPort;

public ChatClient(String serverAddress, int serverPort) {

this.serverAddress = serverAddress;

this.serverPort = serverPort;

}

public void start() {

try (Socket socket = new Socket(serverAddress, serverPort);

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in= new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

Scanner scanner = new Scanner(System.in);

System.out.println("Connected to the chat server");

new Thread(new ServerListener(in)).start();

while (scanner.hasNextLine()) {

out.println(scanner.nextLine());

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static class ServerListener implements Runnable {

private BufferedReader in;

public ServerListener(BufferedReader in) {

this.in = in;

}

@Override

public void run() {

try {

String message;

while ((message = in.readLine()) != null) {

System.out.println(message);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

```

2. ChatClientLauncher.java：

```java

public class ChatClientLauncher {

public static void main(String[] args) {

String serverAddress = "localhost";

int serverPort = 12345;

ChatClient client = new ChatClient(serverAddress, serverPort);

client.start();

}

}

3. ChatServer.java：

java

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class ChatServer {

private static final int PORT = 12345;

private List<PrintWriter> clientOutputStreams;

public ChatServer() {

clientOutputStreams = Collections.synchronizedList(new ArrayList<>());

}

public static void main(String[] args) {

new ChatServer().start();

}

public void start() {

System.out.println("Chat server started on port " + PORT);

try (ServerSocket serverSocket = new ServerSocket(PORT)) {

while (true) {

Socket clientSocket = serverSocket.accept();

PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);

clientOutputStreams.add(writer);

new Thread(new ClientHandler(clientSocket)).start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

private void broadcastMessage(String message) {

synchronized (clientOutputStreams) {

for (PrintWriter writer : clientOutputStreams) {

writer.println(message);

}

}

}

private class ClientHandler implements Runnable {

private Socket socket;

private BufferedReader in;

public ClientHandler(Socket socket) {

this.socket = socket;

try {

this.in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

try {

String message;

while ((message = in.readLine()) != null) {

System.out.println("Received: " + message);

broadcastMessage(message);

}

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

六、Git团队代码管理

团队代码管理部分主要通过Git进行版本控制，每个成员负责不同的模块，并定期进行代码合并和冲突解决。具体操作如下：

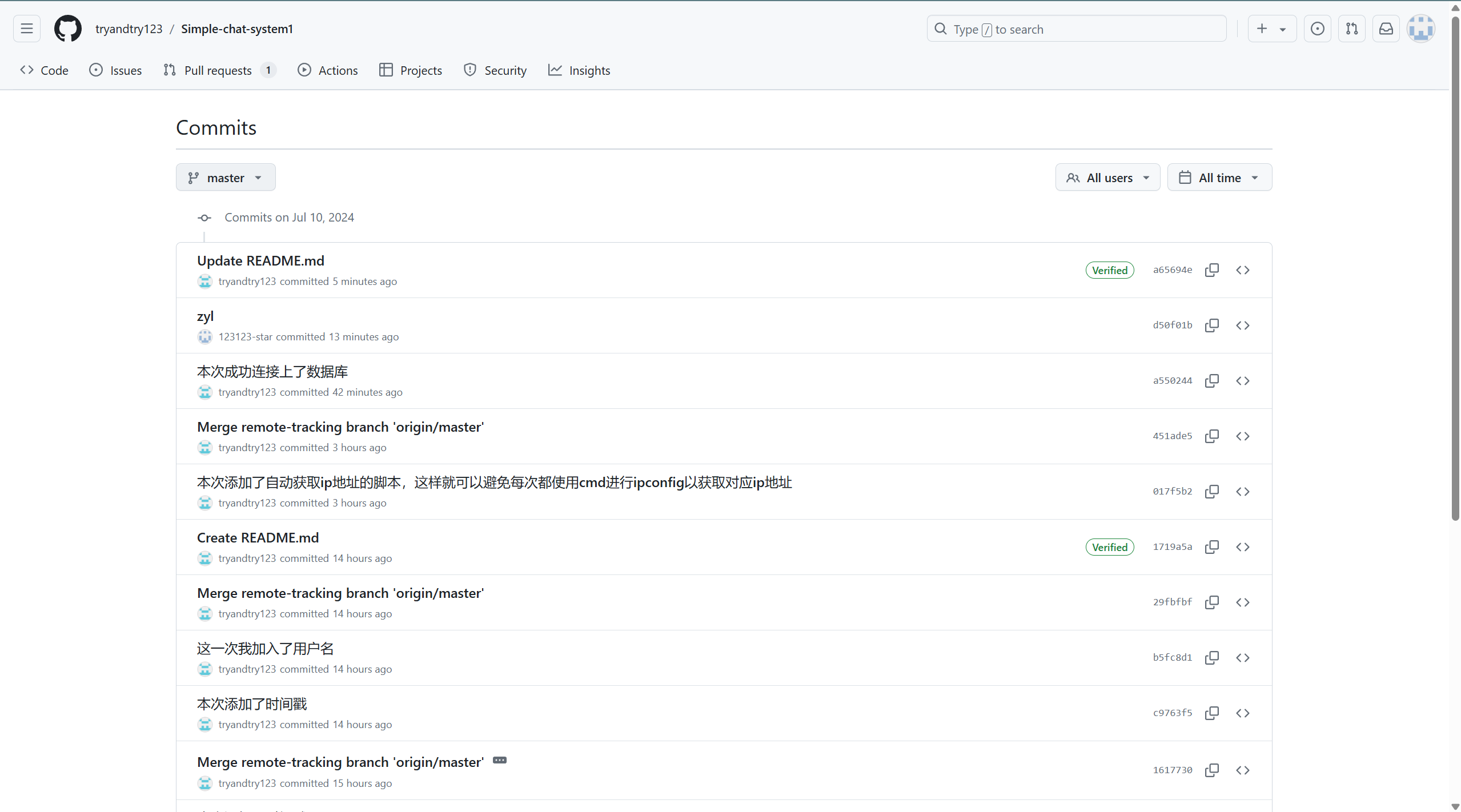
1. 创建Git仓库并初始化项目。

2. 各个成员克隆仓库并在本地进行开发。

3. 定期进行代码提交并推送到远程仓库。

4. 使用Pull Request的方式进行代码审核和合并。

5. 解决代码冲突并保持代码库的整洁和一致。



七、实验小结

以上是本次Java高级程序设计大作业的实验报告。通过本次实验，学习到了以下知识点：

**网络编程原理**：理解网络通信的基本原理，如TCP/IP协议栈、socket编程等，能够实现客户端和服务器之间的通信。

**并发编程**：处理多个客户端同时连接的情况，涉及线程管理、同步与互斥、事件驱动等技术，确保系统能够有效处理并发请求。

**用户认证与权限控制**：设计用户注册、登录、身份验证和权限管理机制，保障聊天系统的安全性和可靠性。

**实时通信**：使用实时通信技术（如WebSocket）实现即时消息的传输，包括消息发送、接收和显示等功能。

**数据存储与管理**：选择合适的数据库存储方案，如关系型数据库（SQL Server）或非关系型数据库（Redis），管理用户信息、聊天记录等数据。

**系统稳定性与扩展性**：设计高可用性系统架构，考虑负载均衡、容错处理、系统监控等，支持系统随用户增长而扩展。

**实验与调试能力**：通过实际的设计与开发过程，提升问题排查、调试和优化的能力，锻炼解决复杂技术问题的能力。

总体而言，设计在线聊天系统是一个综合性强、涵盖多个技术领域的项目，可以帮助学习者深入理解和掌握现代软件开发中的网络通信、并发处理、性能优化等重要概念和技能。