MÔ TẢ KỸ THUẬT VÀ NGHIỆP VỤ

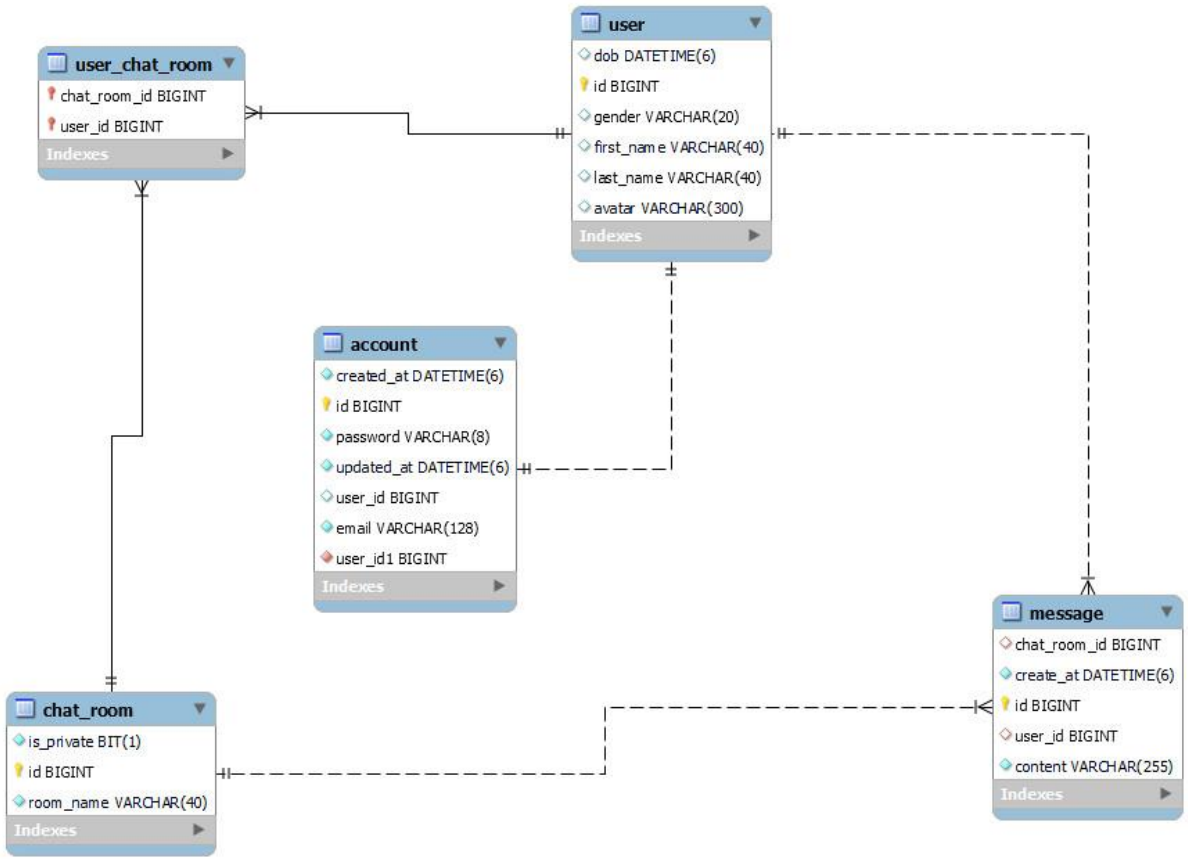
Môn: Lập trình trên thiết bị di động

Đề tài: Tạo ứng dụng nhắn tin trực tuyến

Nhóm: 7

1. Tổ chức cơ sở dữ liệu

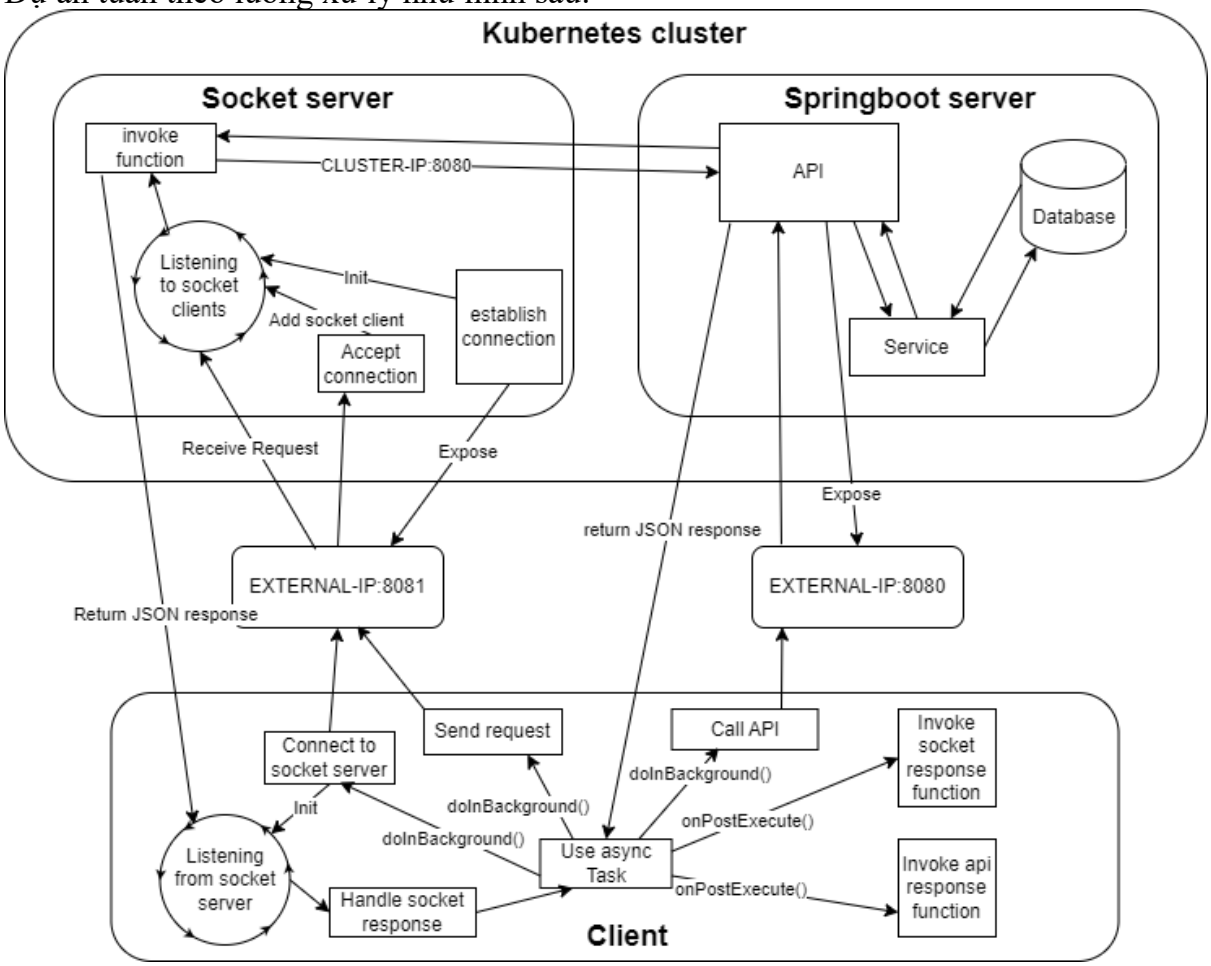
Dự án sử dụng lược đồ cơ sở dữ liệu quan hệ gồm các bảng:



- **Bảng account**: lưu trữ thông tin để đăng ký, đăng nhập như email, password,… có mối quan hệ 1-1 với bảng user vì 1 account chỉ đại diện cho 1 thông tin user.
- **Bảng user**: lưu trữ thông tin cá nhân của người dùng như first_name, last_name,… có mối quan hệ 1-1 với bảng account vì 1 user chỉ đại diện cho 1 account và quan hệ 1 nhiều với bảng message vì 1 user có nhiều message.
- **Bảng chat_room**: lưu trữ thông tin phòng chat có mối quan hệ 1 nhiều với message vì trong chat_room có nhiều message.
- **Bảng message**: lưu trữ thông tin của message và có mối quan hệ 1-1 với chat_room vì 1 message chỉ thuộc 1 chat_room và quan hệ 1-1 với user vì 1 message chỉ thuộc 1 user.
- **Bảng user_chat_room**: đây là bảng trung gian giữa chat_room và user vì 1 user có nhiều chat_room và 1 chat_room có nhiều user.

2. Luồng xử lý của hệ thống

Dự án tuân theo luồng xử lý như hình sau:



Đầu tiên khi chạy Springboot server, server sẽ expose các API ra EXTERNAL-IP để bên ngoài có thể gọi vào và CLUSTER-IP để bên trong kubernetes có thể giao tiếp với port 8080.

Sau đó khi chạy Socket server, server sẽ thiết lập 1 server socket ra cổng 8081 và expose EXTERNAP-IP cho phía client có thể thiết lập kết nối.

```
public class ServerController {
    public static void handleConnect() throws IOException, InterruptedException {
        ServerService.handleConnect(new ServerSocket(port:8081));
    }
}
```

Sau đó socket server sẽ khởi tạo 1 vòng lặp cho tới khi server bị tắt, khi client kết nối vào socket server này, socket server sẽ accept() và thêm vào danh sách các clients và mỗi client sẽ là 1 Thread riêng để lắng nghe request từ socket client đó và xử lý.

```java
public static void handleConnect(ServerSocket serverSocket) throws IOException {
    System.out.println(x:"Server started!");
    while (!serverSocket.isClosed()) {
        Socket clientSocket = serverSocket.accept();
        ClientHandleService clientHandleService = new ClientHandleService(clientSocket);
        clientHandlers.add(clientHandleService);
        Thread thread = new Thread(clientHandleService);
        thread.start();
    }
}
```

Phía client khi chạy ứng dụng, async task có tên ConnectTask sẽ được gọi để kết nối socket tới phía server.

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_load);
    preferencesCheck = getSharedPreferences( name: "check", MODE_PRIVATE);
    preferencesAccount = getSharedPreferences( name: "account", MODE_PRIVATE);
    currentContext = this;
    IP = Utils.getIpV4( mainActivity: this,  fileName: "my_ipv4.json");
    apiUrl = String.format("http://%s:8080/api/", IP);
      IP = "34.101.218.243";
      apiUrl = String.format("http://%s:8080/api/", "34.128.104.216");
    init();
    progressBar.setVisibility(View.VISIBLE);
    ConnectTask connectTask = new ConnectTask( activity: this);
    connectTask.execute();
}
```

Hàm kết nối trong ConnectTask:

```java
@Override
protected Void doInBackground(Void... voids) {
    try {
        clientFd = new Socket(LoadActivity.IP, LoadActivity.PORT);
        dOut = new DataOutputStream(clientFd.getOutputStream());
        dIn = new DataInputStream(clientFd.getInputStream());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    return null;
}
```

Sau khi kết nối thành công phía client sẽ khởi tạo 1 Thread luôn bắt tín hiệu từ server cho đến khi client bị đóng.

```
@Override
protected void onPostExecute(Void unused) {
    super.onPostExecute(unused);
    PlaIsMe *
    new Thread(new Runnable() {
        PlaIsMe *
        @Override
        public void run() {
            while (clientFd.isConnected()) {
                try {
                    socketResponse = dIn.readUTF();
                    ((LoadActivity) activity).handleSocketResponse(socketResponse);
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
        }
    }).start();
```

Khi client gửi tín hiệu socket tới server, sử dụng async Task có tên SendTask với dữ liệu gửi đi là 1 JSON với requestFunction và requestParam.

```
@Override
protected Void doInBackground(String... params) {
    String request;
    try {
        request = new JSONObject()
                .put( name: "requestFunction", params[0])
                .put( name: "requestParam", params[1])
                .toString();
    } catch (JSONException e) {
        throw new RuntimeException(e);
    }
```

Sau đó phía server Thread đang đảm nhiệm lắng nghe client vừa gửi sẽ nhận được request, hàm lắng nghe từ Thread của client:

```java
@Override
public void run() {
    while (clientSocket.isConnected()) {
        String buffer = ServerService.socketReceive(this);
        if (buffer == null) {
            ServerService.removeClient(this);
            break;
        } else {
            try {
                ServerService.handleRequest(this, buffer);
            } catch (NoSuchMethodException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SecurityException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
    // ServerService.realTimeActiveStatus(this, false);
    ServerService.removeClient(this);
}
```

```java
public static String socketReceive(ClientHandleService clientHandleService) {
    try {
        String bufferIn = clientHandleService.getdIn().readUTF();
        System.out.printf(format:"RECEIVED: %s\n", bufferIn);
        return bufferIn;
    } catch (EOFException e) {
        clientHandleService.closeClientSocket();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Nếu dữ liệu từ client không phải là rỗng, phía server sẽ invoke function với tên function từ requestFunction và param truyền vào từ requestParam của json.

```
public static void handleRequest(ClientHandleService clientHandleService, String jsonRequest)
 throws NoSuchMethodException, SecurityException, IllegalAccessException, InvocationTargetException {
    Gson gson = new Gson();
    Request request = gson.fromJson(jsonRequest, classOfT:Request.class);
    Method method = ServerService.class.getMethod(request.getRequestFunction(), ...parameterTypes:ClientHandleService.class ,String.class);
    method.invoke(obj:null, clientHandleService, request.getRequestParam());
}
```

Các function được invoke có thể gọi API từ Springboot server thông qua CLUSTER-IP của springboot sever tại port 8080, để gọi API, bên socketserver có class request service gồm các phương thức get post put delete để tương tác với springboot server khi cần, khi gọi api trong springboot server rest controller, service tương ứng trong controller đó sẽ được gọi và thực thi các hàm trong lớp repository tương tác với database sau đó trả về response dạng json cho socket server, sau đó hàm được invoke trong socketserver sẽ thực hiện các đoạn mã logic khác và trả về response dạng json cho client gồm responseFunction và responseParam.

```
public static void socketSend(ClientHandleService clientHandleService, String responseName, String responseParam) {
    String message;
    try {
        message = new JSONObject()
                .put("responseFunction", responseName)
                .put("responseParam", responseParam)
                .toString();
        // System.out.printf("DEBUG MESSAGE %s\n", message);
    } catch (JSONException e) {
        throw new RuntimeException(e);
    }
    try {
        System.out.printf(format:"SENT: %s\n", message);
        clientHandleService.getdOut().writeUTF(message);
        clientHandleService.getdOut().flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
    // System.out.println("DEBUG SEND DONE ");
}
```

Sau đó client nhận response từ server và sau đó invoke function từ responseFunction và truyền param từ responseParam, các function được invoke sẽ được viết sẵn trong lớp SocketResponse.

```
1 usage   PhatsMe
public void handleSocketResponse(String jsonResponse) {
    Gson gson = new Gson();
    Response response = gson.fromJson(jsonResponse, Response.class);
    try {
        Log.d( tag: "debugCurrentContext", currentContext.toString());
        Method responseMethod = SocketResponse.class.getDeclaredMethod(response.getResponseFunction(), String.class);
        responseMethod.invoke(new SocketResponse(currentContext), response.getResponseParam());
    } catch (InvocationTargetException e) {
        throw new RuntimeException(e);
    } catch (IllegalAccessException e) {
        throw new RuntimeException(e);
    } catch (NoSuchMethodException e) {
        throw new RuntimeException(e);
    }
}
```

Tương tự như socket server, client có thể gọi API từ Springboot server thông qua EXTERNAL-IP từ kubernetes cluster của springboot server service tại cổng 8080. Việc

gọi API phải thông qua các asycn task gồm: GetRequestTask, PostRequestTask, PutRequestTask, PatchRequestTask, sau khi gọi api ở hàm doInBackground() của asycn task, invoke function được định nghĩa khi gọi async task(lấy từ parameters) ở hàm onPostExecute(), các function này được viết trong lớp HttpResponse.

3. Các chức năng của hệ thống

Chức năng đăng nhập:

Người dùng đã có tài khoản có thể đăng nhập và sử dụng ứng dụng bằng các cung cấp thông tin về email và mật khẩu đã đăng ký trước. Khi đăng nhập thành công, người dùng sẽ được chuyển tới trang chủ của ứng dụng.

Chức năng đăng ký:

Chức năng đăng ký tài khoản cho phép người dùng tạo tài khoản mới để truy cập vào ứng dụng. Khi đăng ký, người dùng cần cung cấp các thông tin bao gồm … Sau khi xác nhận thông tin hợp lệ, hệ thống sẽ lưu trữ tài khoản của người dùng và người dùng sẽ được điều hướng tới trang đăng nhập của hệ thống.

Chức năng nhắn tin:

Chức năng này cho phép gửi và nhận tin nhắn giữa các người dùng trong thời gian thực. Người dùng lần đầu có thể tìm kiếm các người dùng khác ở trong cửa sổ Contact và bắt đầu nhắn tin bằng cách nhấn vào người dùng đó. Những cuộc trò chuyện với người dùng khác sẽ được lưu trữ và hiển thị trong cửa sổ Chats, người dùng có thể nhấp vào và tiếp tục cuộc trò chuyện. Khi có tin nhắn mới, hệ thống sẽ gửi thông báo tới thiết bị của người dùng, và khi người dùng nhấp vào thông báo sẽ được chuyển trực tiếp về cửa sổ nhắn tin.