Laborprotokoll

DEZSYS-12: Mobile Chat

Systemtechnik
5BHITT 2015/16

Rene Hollander

Version 1.0

Note:

Betreuer: Borko

Begonnen am 31. März 2016

Beendet am 15. April 2016

Inhaltsverzeichnis

Einführung

Diese Übung soll eine Vertiefung des Wissens für mobile Anwendungen darstellen.

1 Ziele

Das Ziel dieser Übung ist die Entwicklung eines serverbasierten Gruppenchats. Die Applikation soll auf den Entwicklungen der letzten beiden Übungen aufbauen und diese um folgende Funktionalität erweitern:

- Anbindung mit Hilfe eines RESTful Webservice
- Zur Teilnahme beim Gruppenchat ist eine Anmeldung erforderlich (Username/Passwort)
- Abmeldung des Users am Ende der Chatsession
- Verwendung des Observer-Pattern

2 Voraussetzungen

- Grundlagen Java und XML
- Grundlegendes Verständnis über Entwicklungs- und Simulationsumgebungen
- Verständnis von RESTful Webservices
- Grundlegendes Wissen zum Design Pattern "Observer" und dessen Umsetzung

3 Aufgabenstellung

Es ist eine mobile Anwendung zu implementieren, die sich mit Hilfe der Übung DezSysLabor-11 "Mobile Access to Web Services" bei einem Gruppenchat anmeldet. Nach erfolgreicher Anmeldung bekommt der Benutzer alle Meldungen, die in diesem Chat eingehen. Der Benutzer hat ebenso die Moeglichkeit Nachrichten in diesem Chat zu erstellen. Diese Nachricht wird in weiterer Folge an alle Teilnehmer versendet und in der mobilen Anwendung angezeigt. Am Ende muss sich der Benutzer vom Gruppenchat abmelden.

Es ist freigestellt, welche mobile Implementierungsumgebung dafür gewählt wird. Empfohlen wird aber eine Implementierung auf Android.

4 Quellen

"Android Restful Webservice Tutorial – How to call RESTful webservice in Android – Part 3"; Posted By Android Guru on May 27, 2014; online: http://programmerguru.com/android-tutorial/android-restful-webservice-tutorial-how-to-call-restful-webservice-in-android-part-3/

"Referenzimplementierung von DezSys09"; Paul Kalauner; online: https://github.com/pkalauner-tgm/dezsys09-java-webservices

"Android Asynchronous Http Client"; James Smith; online: http://loopj.com/android-async-http/

"Android Asynchronous Networking and Image Loading"; Koushik Dutta; online: https://github.com/koush/ion

"AndroidAsync"; Koushik Dutta; online: https://github.com/koush/AndroidAsync

5 Bewertung: 16 Punkte

- Anmeldung/Abmeldung Gruppenchat (2 Punkte)
- Empfangen von Nachrichten des Gruppenchats (4 Punkte)
- Erstellen von Nachrichten im Gruppenchat (4 Punkte)
- Korrekte Umsetzung des Observer Pattern (2 Punkte)
- Simulation bzw. Deployment auf mobilem Gerät (2 Punkte)
- Protokoll (2 Punkte)

Zeitschätzung

2 Stunden Chat GUI implementieren
2 Stunden Chat Backend implementieren
1 Stunde Protokoll
→ 5 Stunden Zeitaufwand

Tatsächlich:
3 Stunden Chat GUI implementieren
6 Stunden Chat Backend implementieren
1 Stunde Protokoll
→ 4.5 Stunden

Ergebnisse

Aufbauend auf der Übung DezSys11 Mobile Access to Web Services wurde die Android Applikation implementiert und ein Backend für Websockets basierend auf SocketIO eingerichtet.

1 SocketIO Server

Für den SocketIO Server wurde die netty-socketio Bibliothek genutzt. [4]

Mithilfe von Annotationen und Reflection wurde eine Spring Configuration Klasse erstellt, die automatisch nach zu erstellenden SocketIO Server sucht gebaut:

```
@Configuration
public class SocketIOConfiguration {
    private static final Logger LOG =
LoggerFactory.getLogger(SocketIOConfiguration.class);
    @Autowired
    private ApplicationContext context;
    @Bean(name = "socketIOServers")
    public List<SocketIOServer> socketIOServers() {
        // Use beans to find all instances of a @SocketIO annotated class to create a
server for
        Map<String, Object> beans = context.getBeansWithAnnotation(SocketIO.class);
        List<SocketIOServer> servers = new ArrayList<>();
        beans.forEach((name, o) -> {
            SocketIO annotationConfig = o.getClass().getAnnotation(SocketIO.class);
            LOG.info("Starting SocketIO server for bean " + name + " on port " +
annotationConfig.port());
            servers.add(createFrom(annotationConfig, o));
        });
        return servers;
    }
    @SuppressWarnings("unchecked")
    public SocketIOServer createFrom(SocketIO annotationConfig, Object object) {
        // Setup SocketIO configuration
        com.corundumstudio.socketio.Configuration config = new
com.corundumstudio.socketio.Configuration();
        config.setPort(annotationConfig.port());
        config.getSocketConfig().setReuseAddress(true);
        // Create a new SocketIO server instance with the given config
        SocketIOServer server = new SocketIOServer(config);
        // If the Target class implements the given listener, add it to the server
        if (object instanceof ConnectListener)
            server.addConnectListener((ConnectListener) object);
        if (object instanceof DisconnectListener)
            server.addDisconnectListener((DisconnectListener) object);
        // Got through all methods in the class and check if it it has the @Event
annotation
        for (Method method : object.getClass().getDeclaredMethods()) {
            Event event = method.getAnnotation(Event.class);
            // if the event annotation is existing, create a handler stub for socketio
server and
            // forward it to the method via reflection
            if (event != null) {
                if (event.value().isEmpty()) {
```

```
                    throw new IllegalArgumentException("Invalid event name for method
" + stringifyMethod(method));
                }
                if (method.getParameterCount() == 2) {
                    // Uses a default Ack Request implementation that throws an error
                    checkParameterType(method, 0, SocketIOClient.class);
                    LOG.info("Mapped event \"" + event.value() + "\" onto " +
stringifyMethod(method));
                    server.addEventListener(event.value(), method.getParameterTypes()
[1], (DataListener) (client, data, ackSender) -> {
                        if (ackSender.isAckRequested())
                            throw new IllegalStateException("Event " + event.value()
+ " wants an acknowledgement, but it's not implemented in method " +
stringifyMethod(method));
                        method.invoke(object, client, data);
                    });
                } else if (method.getParameterCount() == 3) {
                    // Used if an Ack Request should be handled
                    checkParameterType(method, 0, SocketIOClient.class);
                    checkParameterType(method, 2, AckRequest.class);
                    LOG.info("Mapped event \"" + event.value() + "\" onto " +
stringifyMethod(method));
                    server.addEventListener(event.value(), method.getParameterTypes()
[1], (DataListener) (client, data, ackSender) -> {
                        method.invoke(object, client, data, ackSender);
                    });
                } else {
                    throw new IllegalArgumentException("Invalid signature for method "
+ stringifyMethod(method));
                }
            }
        }
        // Loop through all fields to inject the SocketIOServer instance if needed
        for (Field field : object.getClass().getDeclaredFields()) {
            if (field.getType() == SocketIOServer.class) {
                boolean accessible = field.isAccessible();
                field.setAccessible(true);
                try {
                    field.set(object, server);
                } catch (IllegalAccessException e) {
                    throw new RuntimeException(e);
                }
                field.setAccessible(accessible);
            }
        }
        // Try to initialize the Handler via the init() Method, do nothing if it doesnt
exist
        try {
            Method initMethod = object.getClass().getDeclaredMethod("init");
            boolean accessible = initMethod.isAccessible();
            initMethod.setAccessible(true);
            initMethod.invoke(object);
            initMethod.setAccessible(accessible);
        } catch (NoSuchMethodException ignored) {
        } catch (InvocationTargetException | IllegalAccessException e) {
            throw new RuntimeException(e);
        }
        // Start the server
        server.start();
        return server;
    }
    private static String stringifyMethod(Method method) {
        return method.getDeclaringClass() + "." + method.getName() + "()";
    }
```

```
    private static void checkParameterType(Method method, int idx, Class<?> expected) {
        Class<?>[] paramTypes = method.getParameterTypes();
        if (idx > paramTypes.length)
            throw new IllegalArgumentException("Parameter " + idx + " of method " +
stringifyMethod(method) + " is not " + expected);
        if (paramTypes[idx] != expected)
            throw new IllegalArgumentException("Parameter " + idx + " of method " +
stringifyMethod(method) + " is not " + expected);
    }
}
```

Diese automatische Initialisierung des Servers kann nun wie folgt genutzt werden:

```
@SocketIO(port = 8081)
public class ChatHandler implements ConnectListener, DisconnectListener {
    private static final Logger LOG = LoggerFactory.getLogger(ChatHandler.class);
    private SocketIOServer server;
    private Map<String, Map<String, Object>> chatrooms = new HashMap<>();
    @Override
    public void onConnect(SocketIOClient client) {
        LOG.info("Client " + client + " connected!");
    }
    @Event("message")
    @SuppressWarnings("unchecked")
    public void onChat(SocketIOClient client, Map<String, Object> data) {
        // Get room id from request
        String roomid = (String) data.get("room");
        List<Object> messages;
        // if a room with the given id exists, append message,otherwise create new room
        if (!chatrooms.containsKey(roomid)) {
            // creates a new room
            messages = new ArrayList<>();
            Map<String, Object> roomObj = new HashMap<>();
            roomObj.put("name", roomid);
            roomObj.put("messages", messages);
            chatrooms.put("room", roomObj);
        } else {
            messages = (List<Object>) chatrooms.get(roomid).get("messages");
        }
        // append message and inform clients
        messages.add(data);
        server.getBroadcastOperations().sendEvent("message", data);
    }
    @Event("get_rooms")
    public void onGetRooms(SocketIOClient client, Object ignore, AckRequest ackSender) {
        // send all chatrooms to the client
        ackSender.sendAckData(chatrooms);
    }
    @Event("new_room")
    public void onNewRoom(SocketIOClient client, Map<String, Object> data, AckRequest
ackSender) {
        // create a new room with the given name and inform clients
        chatrooms.put((String) data.get("room"), Maps.of("name", data.get("room"),
"messages", new ArrayList<>()));
        server.getBroadcastOperations().sendEvent("new_room", data);
    }
    @Override
    public void onDisconnect(SocketIOClient client) {
        LOG.info("Client " + client + " disconnected!");
    }
}
```

2 Support für Benutzname

Damit ein Benutzer einen Namen im Chat hat, wurde ein entsprechendes Feld zur User Entity hinzugefügt:

```
@Column(unique = true)
@NotNull
private String username;
```

Dementsprechend wurde auch der Konstruktor angepasst und eine Getter Methode erstellt.

Im Zuge dieser Änderung musste auch der Login und Register Endpoint angepasst werden, um Usernamen zu unterstützen:

Login:

```
@POST
public Response post(User user) {
    User other = userRepository.findOne(user.getEmail());
    return Response.status(user.equals(other) ? 200 : 403).entity(Maps.of("success",
user.equals(other), "token", userSessionStore.createSession(other), "username",
other.getUsername())).build();
}
```

Register:

```
@POST
public Response post(User user) {
    if (userRepository.exists(user.getEmail()) ||
userRepository.findByUsername(user.getUsername()) != null) {
        return Response.status(400).entity(Maps.of("success", false)).build();
    } else {
        userRepository.save(user);
        return Response.status(201).entity(Maps.of("success", true)).build();
    }
}
```

3 Anpassungen in der Android App

Bei der App mussten umfangreiche Änderungen durchgeführt werden. In erster Linie wurden alle bestehenden Activities in Fragments umgewandelt, damit leichter zwischen Verschiedenen Dialogen gewechselt werden kann. Auf diese Änderung wird hier nicht näher eingegangen.

Design

Für das Design wurde in der ChatRoom Liste eine ListView verwendet:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical">
```

```
    <ListView
        android:id="@+id/listView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/add_room"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end|bottom"
        android:layout_margin="10dp"
        android:src="@drawable/ic_add_light"
        app:backgroundTint="@color/colorPrimaryDark" />
</FrameLayout>
```

Für die einzelnen Zeilen gibt es auch ein eigenes Layout, dass ein großes Header TextView für den Namen des Raumes und ein kleines TextView wo die letzte Nachricht angezeigt wird bereitstellt:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="8dp"
    android:paddingLeft="8dp"
    android:paddingRight="8dp"
    android:paddingTop="8dp">
    <TextView
        android:id="@+id/header"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="22sp"/>
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Für den eigentlichen ChatRoom wurde auch ein ListView genutzt. Diese ListView wurde mit verschiedenen Änderungen angepasst. Mit dem `android:stackFromBottom="true"` Attribut wird die Liste von unten nach oben gefüllt. Auch wurde ein EditText und ein ImageButton mit einem entsprechendem Icon für den zu sendeten Text erstellt:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginBottom="48dip"
        android:background="@color/messages_background"
        android:divider="@null"
        android:listSelector="@android:color/transparent"
        android:showDividers="none"
        android:stackFromBottom="true"
        android:transcriptMode="alwaysScroll" />
```

```
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="48dip"
        android:layout_alignParentBottom="true"
        android:layout_margin="0dp"
        android:background="#FFFFFF"
        android:orientation="horizontal"
        android:padding="0dp">
        <EditText
            android:id="@+id/newmsg"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_weight="10"
            android:hint="@string/send_a_message" />
        <ImageButton
            android:id="@+id/newmsgsend"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:layout_weight="1"
            android:background="#FFFFFF"
            android:src="@drawable/ic_send_dark" />
    </LinearLayout>
</RelativeLayout>
```

Für die einzelnen Nachrichten wurden verschiedene Layouts für gesendete und Empfangene Nachrichten erstellt, die dann verschieden aussehen. Zuerst wurden dafür zwei Shapes erstellt, eine Shape für gesendete Nachrichten und eines für emfangene:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/message_sent_color" />
    <corners android:radius="@dimen/rounded_corner_radius" />
</shape>
```

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="@color/message_recieved_color" />
    <corners android:radius="@dimen/rounded_corner_radius" />
</shape>
```

Dann wurden zwei Layouts für gesendete und empfangene Nachrichten erstellt:

Empfangen:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="left"
    android:orientation="vertical"
    android:padding="4dp">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/message_recieved_rounded_corner"
        android:orientation="vertical"
```

```
        android:paddingBottom="4dp"
        android:paddingLeft="8dp"
        android:paddingRight="8dp"
        android:paddingTop="4dp">
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal">
            <TextView
                android:id="@+id/username"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textStyle="bold" />
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text=", " />
            <TextView
                android:id="@+id/date"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </LinearLayout>
        <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
</LinearLayout>
```

Gesendet:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="right"
    android:orientation="vertical"
    android:padding="4dp">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/message_sent_rounded_corner"
        android:orientation="vertical"
        android:paddingBottom="4dp"
        android:paddingLeft="8dp"
        android:paddingRight="8dp"
        android:paddingTop="4dp">
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal">
            <TextView
                android:id="@+id/username"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textStyle="bold" />
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text=", " />
            <TextView
                android:id="@+id/date"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
```

```
        </LinearLayout>
        <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
</LinearLayout>
```

ChatRoom und Message Klasse

Damit die Nachrichten besser verwaltet werden können, wurde eine Message Klasse erstellt:

ChatRoom:

Der ChatRoom verwendet eine ObservableArrayList, damit die GUI von empfangenen Nachrichten informiert werden kann:

```
public class ChatRoom {
    private String name;
    private ObservableArrayList<Message> messages;
    public ChatRoom(String name, ObservableArrayList<Message> messages) {
        this.name = name;
        this.messages = messages;
    }
    public String getName() {
        return name;
    }
    public ObservableArrayList<Message> getMessages() {
        return messages;
    }
    public Message getLast() {
        if (getMessages().size() == 0) return null;
        else return getMessages().get(getMessages().size() - 1);
    }
    @Override
    public String toString() {
        return "ChatRoom{" +
                "name='" + name + '\'' +
                ", messages=" + messages +
                '}';
    }
}
```

Message:

Die Message Klasse hat eine Flag, die angibt ob die Nachricht gesendet oder Empfangen wurde:

```
package at.renehollander.chat.model;
import java.util.Date;
public class Message {
    public static enum Flag {
        SENT, RECIEVED;
    }
    private Flag flag;
    private String room;
    private String user;
    private Date date;
```

```
    private String text;
    public Message(Flag flag, String room, String user, Date date, String text) {
        this.flag = flag;
        this.room = room;
        this.user = user;
        this.date = date;
        this.text = text;
    }
    public Flag getFlag() {
        return flag;
    }
    public String getRoom() {
        return room;
    }
    public String getUser() {
        return user;
    }
    public Date getDate() {
        return date;
    }
    public String getText() {
        return text;
    }
    @Override
    public String toString() {
        return "Message{" +
                "flag=" + flag +
                ", room='" + room + '\'' +
                ", user='" + user + '\'' +
                ", date=" + date +
                ", text='" + text + '\'' +
                '}';
    }
}
```

List Adapter

Damit die im Design erstellten ListViews genutzt werden können, mussten Adapter geschrieben werden.

ChatRoomListAdapter:

Dieser Adapter kümmert sich um die Darstellung der ChatRoom Liste. Sobald ein neuer ChatRoom oder eine neue Nachricht in einem ChatRoom erhalten wird, wird die GUI upgedated:

```
public class ChatRoomListListAdapter extends BaseAdapter {
    private final Activity activity;
    private ObservableArrayList<ChatRoom> chatrooms;
    private LayoutInflater inflater;
    private ObservableListChangeListener changeListener;
    public ChatRoomListListAdapter(Activity activity, ObservableArrayList<ChatRoom>
chatrooms) {
        this.activity = activity;
        this.chatrooms = chatrooms;
        changeListener = new ObservableListChangeListener(this.activity, this);
        for (ChatRoom room : this.chatrooms) {
            room.getMessages().addOnListChangedCallback(changeListener);
        }
        this.chatrooms.addOnListChangedCallback(changeListener);
```

```
        this.inflater = (LayoutInflater)
activity.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }
    @Override
    public int getCount() {
        return chatrooms.size();
    }
    @Override
    public ChatRoom getItem(int i) {
        return chatrooms.get(i);
    }
    @Override
    public long getItemId(int position) {
        return position;
    }
    @Override
    public View getView(int position, View view, ViewGroup viewGroup) {
        if (view == null) {
            view = inflater.inflate(R.layout.chatroomlist_row, null);
        }
        TextView header = (TextView) view.findViewById(R.id.header);
        TextView text = (TextView) view.findViewById(R.id.text);
        header.setText(getItem(position).getName());
        Message last = getItem(position).getLast();
        if (last != null)
            text.setText(last.getUser() + ": " + last.getText());
        else text.setText("");
        return view;
    }
    private class ChangeListener extends
android.databinding.ObservableList.OnListChangedCallback<ObservableList<ChatRoom>> {
        @Override
        public void onChanged(ObservableList<ChatRoom> sender) {
            System.out.println("here");
            for (ChatRoom room : sender) {
                room.getMessages().addOnListChangedCallback(changeListener);
            }
            activity.runOnUiThread(ChatRoomListListAdapter.this::notifyDataSetChanged);
        }
        @Override
        public void onItemRangeChanged(ObservableList<ChatRoom> sender, int
positionStart, int itemCount) {
            activity.runOnUiThread(ChatRoomListListAdapter.this::notifyDataSetChanged);
            for (int i = positionStart; i < positionStart + itemCount; i++) {
                sender.get(i).getMessages().addOnListChangedCallback(changeListener);
            }
        }
        @Override
        public void onItemRangeInserted(ObservableList<ChatRoom> sender, int
positionStart, int itemCount) {
            for (int i = positionStart; i < positionStart + itemCount; i++) {
                sender.get(i).getMessages().addOnListChangedCallback(changeListener);
            }
            activity.runOnUiThread(ChatRoomListListAdapter.this::notifyDataSetChanged);
        }
        @Override
        public void onItemRangeMoved(ObservableList<ChatRoom> sender, int fromPosition,
int toPosition, int itemCount) {
            activity.runOnUiThread(ChatRoomListListAdapter.this::notifyDataSetChanged);
        }
        @Override
        public void onItemRangeRemoved(ObservableList<ChatRoom> sender, int
positionStart, int itemCount) {
            for (int i = positionStart; i < positionStart + itemCount; i++) {
```

```
sender.get(i).getMessages().removeOnListChangedCallback(changeListener);
            }
            activity.runOnUiThread(ChatRoomListListAdapter.this::notifyDataSetChanged);
        }
    }
}
```

MessageAdapter:

Der Message Adapter kümmert sich um die Listen in den einzelnen ChatRooms. Wird eine Nachricht erhalten, wird eine Notification an die Liste gegeben, und ein update der Liste ausgefürt.

```
public class MessageAdapter extends BaseAdapter {
    private ChatRoom room;
    private LayoutInflater inflater;
    public MessageAdapter(Activity activity, ChatRoom room) {
        this.room = room;
        this.room.getMessages().addOnListChangedCallback(new
ObservableListChangeListener(activity, this));
        this.inflater = (LayoutInflater)
activity.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }
    @Override
    public int getCount() {
        return room.getMessages().size();
    }
    @Override
    public Message getItem(int i) {
        return room.getMessages().get(i);
    }
    @Override
    public long getItemId(int position) {
        return position;
    }
    @Override
    public View getView(int position, View view, ViewGroup viewGroup) {
        Message message = getItem(position);
        if (view == null) {
            if (message.getFlag() == Message.Flag.RECIEVED) {
                view = inflater.inflate(R.layout.message_recieved, null);
            } else if (message.getFlag() == Message.Flag.SENT) {
                view = inflater.inflate(R.layout.message_sent, null);
            } else {
                throw new RuntimeException("unknown message type");
            }
        } else {
            if (view.getId() != R.layout.message_recieved && message.getFlag() ==
Message.Flag.RECIEVED) {
                view = inflater.inflate(R.layout.message_recieved, null);
            } else if (view.getId() != R.layout.message_sent && message.getFlag() ==
Message.Flag.SENT) {
                view = inflater.inflate(R.layout.message_sent, null);
            }
        }
        TextView text = (TextView) view.findViewById(R.id.text);
        TextView date = (TextView) view.findViewById(R.id.date);
        TextView username = (TextView) view.findViewById(R.id.username);
        username.setText(message.getUser());
        date.setText(SDF.format(message.getDate()));
```

```
        text.setText(message.getText());
        return view;
    }
    private SimpleDateFormat SDF = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
}
```

SocketIO Client

Der SocketIO Client kümmert sich um die Kommunikation mit dem Server. Dazu wurde die socket.io-client-java Library genutzt. [5]

```
public void connectToChat(String username) {
    // Connect to SocketIO server at the given URL
    this.username = username;
    Log.d("chat", "connecting");
    IO.Options opts = new IO.Options();
    try {
        this.socket = IO.socket(SOCKETIO_URL, opts);
    } catch (URISyntaxException e) {
        throw new RuntimeException(e);
    }
    // Add Listeners for the events
    socket.on(Socket.EVENT_CONNECT, this::onConnect);
    socket.on(Socket.EVENT_DISCONNECT, this::onDisconnect);
    socket.on("message", this::onMessage);
    socket.on("new_room", this::onNewRoom);
    socket.connect();
}
```

Die Verschiedenen Events die vom Server ausgelöst werden können sind hier implementiert:

```
private void onNewRoom(Object[] objects) {
    // Trigger update of chatroom list
    Log.d("chat", String.valueOf(objects[0]));
    JSONObject obj = (JSONObject) objects[0];
    try {
        String roomName = obj.getString("room");
        if (!chatRoomMap.containsKey(roomName)) {
            ChatRoom room = new ChatRoom(roomName, new ObservableArrayList<>());
            chatRoomMap.put(room.getName(), room);
            chatrooms.add(room);
        }
    } catch (JSONException e) {
        throw new RuntimeException(e);
    }
}
private void onMessage(Object[] objects) {
    // Trigger update on message list
    Log.d("chat", String.valueOf(objects[0]));
    JSONObject obj = (JSONObject) objects[0];
    try {
        Message msg = decode(username, obj);
        if (!chatRoomMap.containsKey(msg.getRoom())) {
            ChatRoom room = new ChatRoom(msg.getRoom(), new ObservableArrayList<>());
            chatRoomMap.put(room.getName(), room);
            chatrooms.add(room);
            room.getMessages().add(msg);
        } else {
            chatRoomMap.get(msg.getRoom()).getMessages().add(msg);
```

```
            }
        } catch (JSONException e) {
            throw new RuntimeException(e);
        }
    }
    private void onDisconnect(Object[] objects) {
        Log.d("chat", "disconnected");
    }
    private void onConnect(Object[] objects) {
        // Retrieves the current chatroom list from the server
        Log.d("chat", "connected");
        this.socket.emit("get_rooms", null, args -> {
            JSONObject obj = (JSONObject) args[0];
            Log.d("chat", String.valueOf(obj));
            // TODO fix crash on reconnect
            for (ChatRoom room : chatrooms) {
                room.getMessages().clear();
            }
            chatrooms.clear();
            chatRoomMap.clear();
            for (String key : iterable(obj.keys())) {
                try {
                    ChatRoom room = decodeRoom(username, obj.getJSONObject(key));
                    chatRoomMap.put(room.getName(), room);
                    chatrooms.add(room);
                } catch (JSONException e) {
                    throw new RuntimeException(e);
                }
            }
        });
    }
```

Zum Senden einer Nachricht oder eines ChatRooms wird ein Event mit den benötigten Daten an den Server gesendet:

```
    public void sendMessage(String room, String msg) {
        // send a message to the given chatroom
        try {
            JSONObject obj = new JSONObject();
            obj.put("date", new Date().getTime());
            obj.put("room", room);
            obj.put("user", username);
            obj.put("text", msg);
            if (this.socket != null) {
                this.socket.emit("message", obj);
            }
        } catch (JSONException e) {
            throw new RuntimeException(e);
        }
    }
    public void createRoom(String room) {
        // create a chatroom with the given name
        try {
            JSONObject obj = new JSONObject();
            obj.put("room", room);
            if (this.socket != null) {
                this.socket.emit("new_room", obj);
            }
        } catch (JSONException e) {
            throw new RuntimeException(e);
        }
    }
```

Um die JSON Objekte in die vorher erstellten Klassen zu deserialisieren wurde folgendes Implementiert:

```
private static ChatRoom decodeRoom(String thisUser, JSONObject object) throws
JSONException {
    // Decode the JSON of a Room
    String name = object.getString("name");
    JSONArray messages = object.getJSONArray("messages");
    ObservableArrayList<Message> messageList = new ObservableArrayList<>();
    for (int i = 0; i < messages.length(); i++) {
        messageList.add(decode(thisUser, messages.getJSONObject(i)));
    }
    return new ChatRoom(name, messageList);
}
private static Message decode(String thisUser, JSONObject object) throws JSONException {
    // Decode a message object
    Date date = new Date(object.getLong("date"));
    String room = object.getString("room");
    String user = object.getString("user");
    String text = object.getString("text");
    Message.Flag flag = user.equals(thisUser) ? Message.Flag.SENT :
Message.Flag.RECIEVED;
    return new Message(flag, room, user, date, text);
}
```

Heroku

Da man bei Heroku nur einen vorgegebenen Port zur Verfügung hat, kann kein SocketIO Server auf einem anderen Port gestartet werden. Somit ist das Deployment auf Heroku dieser Applikation nicht möglich.Source Code

Der Code ist Verfügbar unter: https://github.com/ReneHollander/dezsys11-mobileapp

Kompilieren und Ausführen

Die Webapplication kann wie beschrieben in Dezsys09 [1] lokal gestartet werden. Um die Lokale Installation der Webapplication zu nutzen, müssen die IPs in der Application Class der App angepasst werden:

```
private static final String API_URL = "http://10.0.105.191:8080";
private static final String SOCKETIO_URL = "http://10.0.105.191:8081";
```

Quellen

[1] DEZSYS09-Webservices, Rene Hollander
Abrufbar unter: https://github.com/ReneHollander/dezsys09-webservices
zuletzt abgerufen: 19.02.2016

[2] Android Asynchronous Http Client
Abrufbar unter: http://loopj.com/android-async-http/
zuletzt abgerufen: 10.03.2016

[3] Retrolambda Gradle Github
Abrufbar unter: https://github.com/evant/gradle-retrolambda
zuletzt abgerufen: 10.03.2016

[4] netty-socketio Github
Abrufbar unter: https://github.com/mrniko/netty-socketio
zuletzt abgerufen: 14.04.2016

[5] socket.io-client-java Github
Abrufbar unter: https://github.com/socketio/socket.io-client-java
zuletzt abgerufen: 14.04.2016