System design document for Edda

Version: 1.0.0

Date: 9/5/2016

Author: Group 11

This version overrides all previous versions.

1 Introduction

1.1 Design goals

Vi kommer sträva efter en så löskopplad design som möjligt.

1.2 Definitions, acronyms and abbreviations

- *Android*, ett operativsystem för mobiltelefoner.
- *Gradle*, ett verktyg som hanterar systemets beroende av andra bibliotek.
- *Java*, ett programmeringsspråk som är plattformsoberoende.
- *GUI*, grafiskt användargränssnitt
- *MVC*, ett sätt att bygga ett program med GUI där man delar upp GUI-koden, programkoden och all data i strikta delar.
- *Feed*, appens flöde av innehåll.
- *Post*, ett inlägg i appens flöde.
- *Chat*, en konversation mellan två personer.

2 System design

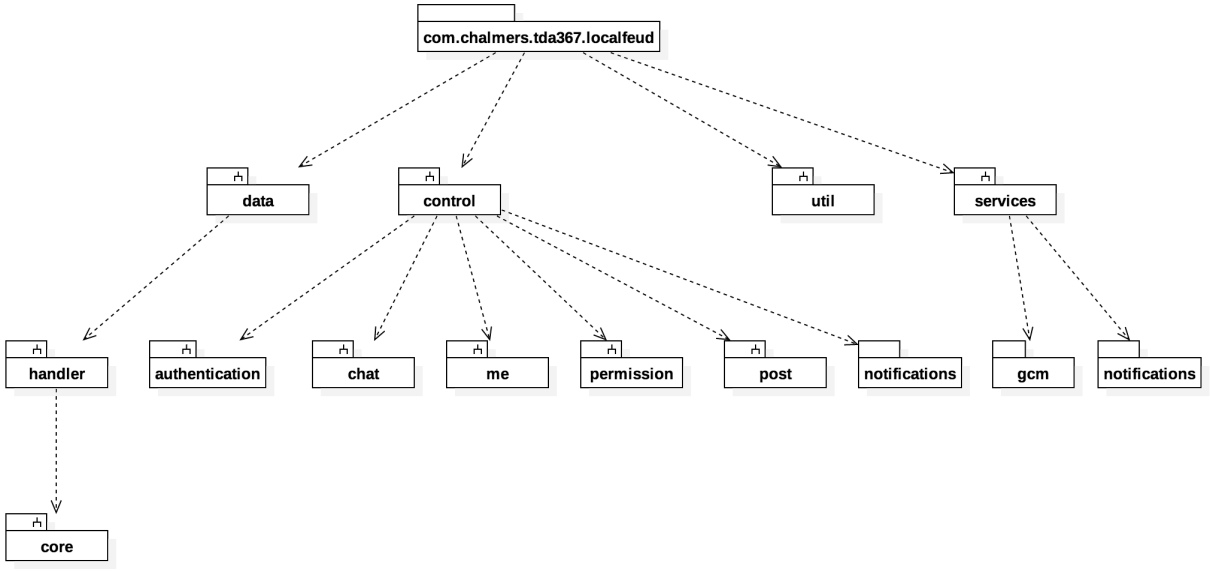
2.1 Overview

Applikationen kommer att använda sig av MVC. Vyerna är uppdelade i olika aktiviteter, exempelvis flödesaktivitet och skapa inläggsaktivitet. Vissa aktivteter kan används till flera olika saker. Dessa saker, exempelvis inläggsflödet och chattflödet, är uppdelade i fragment, som sätts till aktiviteterna. Små detaljer och objekt som upprepas flera gånger, exempelvis ett enskiljt inlägg, är uppdelade i adaptrar som läggs in i fragmenten och aktiviteterna.

2.2 Software decomposition

2.2.1 General

Applikationens paket är uppdelade på följande sätt:



- **com.chalmers.tda367.localfeud** är applikationens huvudpaket.
  - **data** innehåller all typ av data som kommunicerar med servern.
    - **handler** innehåller de klasser som sköter datans kommunikation med servern.
      - **interfaces** innehåller de interfaces som implementeras av handler-klasserna.
  - **control** är det paket där kontrollen med vyerna finns.
    - **authentication** innehåller de kontrolldelarna som har med autentiseringen att göra.
    - **chat** innehåller de kontrolldelarna som har med chatten att göra.
    - **me** innehåller de kontrolldelarna som är personliga för användaren, så som inställningar och notiser.
    - **permission** innehåller de nödvändiga kontrolldelarna som behövs för att få behörigheter av användaren.
    - **post** innehåller de kontrolldelarna som berör applikationens flöde och inlägg.
  - **services** är det paket som sköter nätverksdelen med servern.
  - **util** innehåller de övriga verktyg som behövs för applikationen.

2.2.2 Decomposition into subsystems

Ett interface vi har skapat är **DataResponseListener**, som håller reda på huruvida datan lyckades/misslyckades hämtas från antingen servern eller ett lokalt system.

```
public interface DataResponseListener<D> {

    void onSuccess( D data );

    void onFailure(DataResponseError error, String errormessage );

    Type getType();

}
```

Vi har även ett interface som heter **IChatDataHandler**, som sköter de allmänna funktionerna för *alla* chatter, såsom om man vill skapa en ny chat med någon eller få en lista på de man redan chattar med.

```
public interface IChatDataHandler {

    void sendRequest(Post post, int userID, DataResponseListener<Chat> listener);

    void getList(DataResponseListener<List<Chat>> listener);


    void addChangeListener( DataChangeListener<Chat> listener );

    void triggerChange( Chat oldValue, Chat newValue );

}
```

IChatMessageDataHandler är det interface som har hand om de funktionerna för *en specifik* chat. Här finns möjlighet att få listan med meddelanden samt skicka ett nytt meddelande.

```
public interface IChatMessageDataHandler {

    void getList(Chat chat, DataResponseListener<List<ChatMessage>> listener);

    void send(Chat chat, ChatMessage message, DataResponseListener<ChatMessage> listener );

}
```

ICommentDataHandler sköter kommentarerna på en post, både hämta, skapa och ta bort.

```
public interface ICommentDataHandler {

    void getList(Post post, DataResponseListener<List<Comment>> listener );

    void getSingle( int id, DataResponseListener<Comment> listener);

    void delete( Comment comment, DataResponseListener<Void> listener);

    void create( Post post, Comment comment, DataResponseListener<Comment> listener);

}
```

ILikeDataHandler fungerar precis som *ICommentDataHandler*, fast med gillningar på en post istället för kommentarer.

```
public interface ILikeDataHandler {

    void getList( Post post, DataResponseListener<List<Like>> listener);

    void create( Post post, DataResponseListener<Like> listener);

    void delete( Post post, DataResponseListener<Void> listener);

}
```

IMeDataHandler sköter information angående användaren.

```
public interface IMeDataHandler {

    void get(DataResponseListener<Me> listener);

    void setMe( Me me );

    Me getMe() throws NullPointerException;

}
```

IPostDataHandler innehåller funktioner som har med posts att göra.

```
public interface IPostDataHandler {

    void getList( Position pos, DataResponseListener<List<Post>> listener );

    void getSingle( int id, DataResponseListener<Post> listener );

    void create( Post post, DataResponseListener<Post> listener );

    void delete( Post post, DataResponseListener<Void> listener );

    void addChangeListener( DataChangeListener<Post> listener );

    void triggerChange( Post oldValue, Post newValue );

}
```

DataChangeListener är ett interface som används för att få olika vyer att prata med varandra. Till exempel, om man ändrar

något i en post i en viss vy, hjälper DataChangeListener till så att även andra vyer uppdateras.

```
public interface DataChangeListener<T> {

    void onChange( T oldValue, T newValue );

}
```

IAuthentication används för att tracka huruvida användaren är inloggad eller inte. Det finns även ett inre interface som heter **IAuthenticationListener** som hanterar när användaren loggar in och ut.

```
public interface IAuthentication {

    void startTracking(Context context, IAuthenticationListener listener );

    HashMap getRequestHeaders();

    boolean isLoggedIn();


    interface IAuthenticationListener {
            void onLogInSuccessful();
            void onLoginFailed( AuthenticationError err);
            void onLogOut();
    }


    enum AuthenticationError {
        DECLINED_PERMISSIONS
    }

}
```

IResponseAction är ett interface som används vid all typ av kontakt med servern. Den håller reda på om anslutningen lyckades som det skulle eller ej.

```
public interface IResponseAction {

    void onSuccess( String responseBody );

    void onFailure( int statusCode, String responseBody );

}
```

IMessageHandler tar emot alla notiser och sorterar ut dessa till rätt komponent av applikationen.

```
public interface IMessageHandler {

    void removeMessageListener(String type, IMessageListener listener);

    void removeMessageListener(String type, @Nullable MapEntry<String, Object> data, IMessageListener listener);

    void addMessageListener(String type, IMessageListener listener);

    void addMessageListener(String type, @Nullable MapEntry<String, Object> data, IMessageListener listener);
    void handleMessage(String type, JSONObject data);
}
```

IMessageListener lyssnar efter nya inkommande meddelanden.

```
public interface IMessageListener {

    void onMessageReceived(Map<String, Object> data);

}
```

ILocationHandler används när man vill spåra efter användarens position.

```
public interface ILocation {

    void startTracking(Context context) throws LocationPermissionError;

    void stopTracking() throws LocationPermissionError;

    boolean isTracking();

    android.location.Location getLocation();

}
```

IRestClient sköter kontakten med servern.

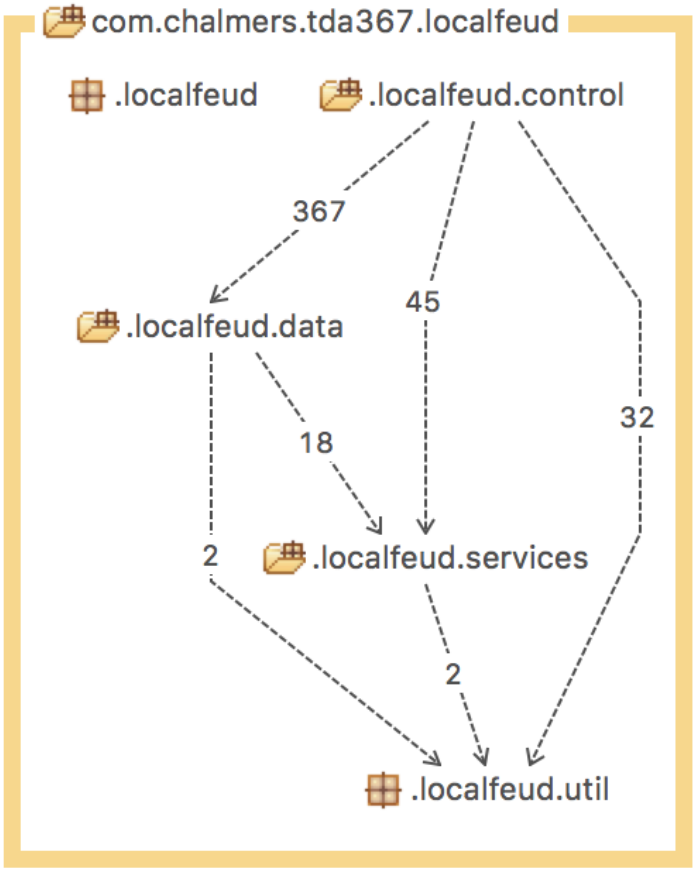
```
public interface IRestClient {

    void get(String url, Map<String,String> paramsMap, IResponseAction action);

    void get(String url, IResponseAction action);

    void post(String url, IResponseAction action);

    void post(String url, Map<String,String> paramsMap, IResponseAction action);

    void put(String url, IResponseAction action);

    void put(String url, Map<String,String> paramsMap, IResponseAction action);

    void delete(String url, IResponseAction action);

    void delete(String url, Map<String,String> paramsMap, IResponseAction action);

}
```

2.2.3 Layering

N/A

2.2.4 Dependency analysis

De olika beroenden mellan paketen visas här. Det finns inga cirkulära beroenden mellan paketen.



2.3 Concurrency issues

N/A

2.4 Persistent data management

All data tillhörande applikationen sparas i en relationsdatabas på en extern server. Det enda som sparas på klienten är uppgifter om enheten som används för underlättad kommunikation med notiser.

2.5 Access control and security

För att kunna komma åt innehållet i applikationen måste användaren logga in. Inloggningen sker mot Facebook vilket gör processen säker, då vi inte lagrar några privata uppgifter om användaren.

2.6 Boundary conditions

Applikationen körs endast på enheter med Android som operativsystem. Versionen på operativsystemet måste vara minst *API 21 (5.0 Lollipop)*.