Assignment 3

MOBILE SOFTWARE

Network Application Framework

Contents

1. Learning the basics 1
Hello World 1
Using the network 2
Using UI elements 2
Parsing simple data 2
2. Helsinki Region Transport App 3
REST API 3
Using phone's context 3
Accessing server-side data 4
3. Twitter client 4
4. Working time 5
5. References 5

1. Learning the basics

The next mobile app tries to show the implementation of several android features in a simple way. These features are from the management of activities and fragments to the use of the network to retrieve data from an URL. I have already worked with Android to the Mobile Systems Programming course. However, in this assignment I have been able to improve my Android knowledge about Fragments and parsing data.

I have chosen Android because as an owner of an Android device I wished to be able to develop my own apps and because the Android is one of the Mobile OS which is most increasing the number of users.

The next features has been developed in the mobile app LearningTheBasics. Each fragment of the app is one of the next features and the user can navigate between the fragments using the tabs at the top of the screen. This means that the layout of each feature is designed in a separate file, while the logic of all the features resides in the MainActivity. These Fragments are run by the MainActivity and can receive their own input events and start new activities.

Hello World

This feature has consisted on following the "Hello world" tutorial of the Android Training website (http://developer.android.com/training/basics/firstapp/index.html). Firstly, I have designed the layout for the Fragment, which consists of an EditText to introduce input text and a button. The logic of this fragment consists of where the user click the button a new activity called DisplayMessageActivity is started to show the string of the EditText. For that purpose, we have to indicate in the layout XML

which function will be called when we click the button using the attribute *android:onClick*. The layout of the DisplayMessageActivity is just a TextView to show the content of the EditText of the fragment.

Using the network

In this Fragment I have learned how to use an HTTP Client to connect with the network. I have chosen *HttpURLConnection* as a client, due to it is the HTTP Client recommended in the Android Developer website (http://developer.android.com/training/basics/network-ops/connecting.html). The layout of this fragment consists of a button and two TextViews to show the date of modification and the content of the retrieved document.

The logic of this application consists of when the user click the button "Retrieve data" the function called *retrieveData* is called. This function first checks whether the device has access to the network via Wifi or mobile data. If it does not have access, a dialog is showed to alert the user.

Secondly, if the device has access to the network the activity executes the task DownloadDocumentTask. This task is an AsyncTask, which is one of the simplest ways in Android to fire off a new task from an activity. These kinds of tasks can do its work in the background without stopping the user interface.

This kind of tasks has three main functions:

- *doInBackground*: when we write the code or the functions that will be called during the task
- onProgress: when we can write code to show to the user for example the progress of the task.
- onPostExecute: the code inside this functions will be run when the code of doInBackground finalizes and returns a result.

Based on that the *doInBackground* for this task consists of using the HTTP client to download the document and extract the content and the date of modification from it. Later, the onPostExecute function will set the information in the views of the screen.

Using UI elements

In this section I have learned how to get media file from the network and how to show it on the device screen. The layout of this fragment consists of an EditText to introduce the URL of the media file, a button to start the download and show process of the media file and an ImageView to show the picture.

The way of downloading the media file is similar to the last section. In this case we do not need to extract extra information from the response. The AsyncTask DownloadImageTask performs the request with the indicated URL and retrieve the response. Finally it calls the function *readContentAsImage* that get the *InputStream* and decode it as a bitmap that it shows in the *ImageView* of the screen.

Parsing simple data

This feature consists of performing a request to the Yahoo Weather API to retrieve the weather forecast of a location identified by a WOEID. The response consists of an XML file that contains the weather forecast and has to be parsed to extract the data we want to use.

About the layouts, there is an editText to introduce the WOEID of the location we want to know the weather and a set of TextViews to show the weather forecast. The first part of the logic of this fragment is the same of the two previous sections and consists of download the data from the Yahoo Weather API as an InputStream.

Once the data has been downloaded we need to parse the XML of the response. This XML file contains weather forecast for the input WOEID. For parsing the XML I have used the

ExpatPullParser. The parser can be found in the file *WeatherXMLParser.java*. It has two main parts, the instantiation of the parser and the read of the feed from the input stream.

We are interested in reading the location (city and country), the weather forecast for today and the weather forecast for the four next days. Then, in the parser we need to define which tags we want to read. After parsing the XML file the retrieved information is set in the TextViews of the screen.

2. Helsinki Region Transport App

This app has been developed as an easy solution for all the users of the Helsinki Region Transport region. With this mobile app everyone can consult the routes between two locations of the Helsinki Region using the public transport. Moreover, a user system has been implemented to let store personalized data that make easier the search, for example the possibility of save journeys to repeat the search in a future faster.

REST API

The server REST API has been developed in Django. The server-side programming can be divided in two parts: the models and the views.

The models I have designed to manage the data in the database are ExtendedUser and Journey. ExtendedUser is an extension of the Django default User model where I add the access token as an attribute of the model. Using the default User model most of the attributes I need are already implemented and the encryption of the password is already implemented. Moreover, I have designed the model Journey to save a journey behalf a user. A journey consists of a string to identify the journey and two string to represent the origin (from) and the destination (to) of the journey. These models can be found in the file *models.py*.

The views represent the API functions that our REST service offers, these views can be found in the file *views.py*. Each view performs one of the REST API functions. The following functions have been designed:

- *register_user*: Register a user in the system.
- *login*: Log in using user's credentials username and password.
- *search_routes*: Search routes from an origin to a destination using the Helsinki Region Public transport services.
- *save_journey*: Save a journey for a user.
- *retrieve_saved_journeys*: Retrieve the save journeys of a user.

I have designed a welcome page for the Server REST API where the services of the API are explained in detail, specially the format of the request parameters and responses. This page can be found in http://group36.naf.cs.hut.fi/transport_website/transport_route/.

Using phone's context

Setting the address from where someone wants to depart is slower and sometimes the user of the app may not know where they are, for example a tourist, I have implemented the feature of getting the localization from the device to set it as "from" point.

Using the Location Service requires the device has installed the Google Play Services APK, which must be linked to the Android App project as a library. For that reason, the app checks that the Google Play Services are available before retrieving the current location of the device. Because we want to offer an accurate location, we set the permissions of the application to use the fine location (*ACCESS_FINE_LOCATION*). We can also define the callbacks to attend the response of the location service. The location feature was developed following the tutorial in the next link of the Android developer website https://developer.android.com/training/location/retrieve-current.html.

Once the app has the location of the device as an object Location it passes the latitude and longitude of the location with the parameter *from_coord* in the request to the server. In the server we need to treat this coordinates in order to be able to use them with the routing service of the Reittiopas API.

The coordinates cannot be passed directly to the Reittiopas API in the parameters of the request. Firstly, we need to use the Google Geocoding API to retrieve the address of the coordinates. Once the server has the address of the coordinates this is used to retrieve coordinates that are understandable for the Reittiopas API using its Geocoding service. These two steps has to be done due to the Reittiopas API cannot use the device coordinates directly.

Finally, when the server has the coordinates from the Reittiopas Geocoding service, then it performs the request to the Routing service in the same way as when the user type the *from* address manually.

Accessing server-side data

Because of the users usually have transport journeys that they repeat almost diary, I considered necessary to introduce a feature which lets users save and retrieve journeys to speed the consult of a route. A journey is saved in the server database with four values: the identifier of the journey, the *from* address, the *to* address and the user who save the journey.

Firstly, I designed an activity that lets the app user register on the server. For that purpose the user provides its name, the username and a password. Then a request to the server is done using the Register user API function. The request to the server can return two responses, one if the user has been registered successfully and another if the user cannot be registered due to its username already exists in the server database.

Once the user is registered in the server he can log in using his username and password. The request to server is done using the Login API function. When the server receives the request, firstly, it checks if the user exists, it the user does not exists it returns a response to inform about the error. Secondly, the server checks if the password provided in the request matches with the password of the user in the database, if the password do not match then it returns a response to let the app know the error.

Finally, if the passwords match the server generate an Universally unique identifier (UUID) of 21 characters length using the *uuid* module of Python that will be used as access token for the user [3]. This access token is saved with the user in the database, in that way the user just needs to provide the access token in the requests to save and retrieve journeys from the server. Then, the server responses to the request attaching the access token, which is saved in the app to use it in the next requests.

If the user has logged in previously he will see enabled a button on the search route screen of the app called “Saved journeys”. If the user press the button he will see a screen with a spinner to select the saved journeys and information about the select journey (identifier and from and to addresses). Then the user can select one of the journeys and the information will be set on the Search route screen. Finally the user just need to select the date and time of the journey to search routes.

The other feature is the possibility of saving journeys. Once the user has made a route search he will find a button on the screen where the routes are shown to save the journey. In the screen to save the journey the user can indicate the identifier of the journey. For example a student user would like to save a journey from his home to the university with the identifier “University”. Later, the user will be able to use this saved journey in the way I have explained above.

3. Twitter client

In this app called TwitterSimpleClient I developed a proof-of-concepts mobile app that retrieves the top tweets for a query. This app use the Twitter REST API, concretely the function search tweets.

Firstly, I needed to declare my app in the Twitter developer site in order to get an API key and API secret that identify my application in the next requests. The request of the search of the tweets containing a word cannot be done directly, the app need to retrieve first and access token that let it perform the search request.

The app first does a request to the Twitter OAuth service where it send the API key and API secret of the app using a Base64 codifications. In response, it gets the access token that will be attach as a parameter when it performs the request to the search function.

After getting the access token, the app executes the AsyncTask *TwitterSearchAPITask* to retrieve the tweets that contain the word typed by the user. In the request the access token is attach as a header while the word to search is set in the parameters using the field *q*.

Finally, the app receives as a response the tweets that contain the word of the query as a JSON object. Then, this JSON object is parsed to retrieve the information of the author, content of the tweet and date from the fields: user, text and created_at. After that, the tweets are show in the screen of the app.

4. Working time

The whole assignment has taken around 50 hours to complete it.

5. References

[1] http://developer.android.com/training/index.html Android Training. Google. Website. 4th of April of 2014.

[2] https://www.youtube.com/watch?v=JJeeZHFgLfc Tutorials to develop scrollable tabs on Android. Author: slidenerd. Youtube. Video. 6th of April of 2014.

[3] http://stackoverflow.com/questions/621649/python-and-random-keys-of-21-char-max Create a random key. Stack Overflow. Forum Website. 15th of April of 2014.

[4] https://developers.google.com/maps/documentation/geocoding/#ReverseGeocoding ReverseGeocoding. Google Geocoding API. Website. 15th of April of 2014.

[5] https://dev.twitter.com/docs/api/1.1/get/search/tweets Twitter Search Tweets Documentation. Twitter. Website. 6th of April of 2014.

[6] https://dev.twitter.com/docs/auth/application-only-auth Twitter Application Only-auth. Twitter. Website. 8th of April of 2014.

[7] https://developer.apple.com/iad/ iAd Developer Site. Apple. Website. 20th of April of 2014.

[8] http://www.google.com/ads/admob/ Google admob. Google. Website. 20th of April of 2014.

[9] http://www.smaato.com/ Smatto. Website. 20th of April of 2014.

[10] http://advertising.microsoft.com/en-us/home Microsoft advertising. Website. 20th of April of 2014.