Mobile Project Dashboard

Bachelorthesis

Fakultät für Informatik

Studiengang Software Engineering

Hochschule Heilbronn

Benjamin Richter
Matrikelnummer: 172155

Erstprüfer: Prof. Dr.-Ing. Gerald Permantier
Zweitprüfer: Dipl.-Ing. Jürgen Kraus

August 15, 2014

Contents

List of Figures

1. Introduction

1.1 Project description

Usually the members of a project communicate via phone or e-mail. This ways of communication are not always very reliable or effective. The beap lightweight user bulletin board (blubb) offers a dashboard where all project participants can talk about the current state of a project. The messages are sorted by subject to a certain thread.

Blubb is a database based software that has been developed to improve the communication of project teams. It can be integrated in an existing web app, that is based on beap or work as a standalone application.

Goal of the Mobile Project Dashboard is to support blubb with a mobile application for Android. In this way mobile project members can stay up to date and participate in discussions about a project.

1.2 General goals

The following requirements where given for the mobile blubb app, see appendix A:

- Login with username and password.
- Overview of all threads of the project.
  Threads with new messages are recognizable and it is possible to open a new thread. The selection of a thread opens the Thread View.
- The Thread View is a list of all messages of the selected thread in reverse chronological order (newest first) in a short form. Every list entry shows
  - username,
  - user type (admin, project leader or regular user),
  - date and time and

  - subject of the message.

  The different user types of the message creator are recognizable by a visual component. A distinction between read and new messages would be desirable but is not mandatory. At the Thread View a user can select or write messages.

- The Message View shows a selected message in addition to the short form with the message content.
- Navigating to the next or previous message is possible by a swipe up / down.
- Swipe-Left should lead back to the Thread View. From this view it is possible to write a reply.
- Post View for writing new messages. The structure for messages is predefined and contains mandatory and optional fields.
- Update and Notifications.
  The application sends requests for new messages to the server in definable intervals. Through the Notification framework of Android, the user becomes notified about new messages (similar to SMS, e-mail,...). An indication of the number of new messages is sufficient.
- Settings.
  In the settings it is possible to turn on or off the auto login or the update function. Further it the frequency for message requests can be adjusted.

The start date for the project was April 15, 2014 and August 15, 2014 is the delivery date of the implemented app.

2. Implementation of the project

2.1 Product backlog

To implement the blubb Android app the agile software development framework scrum was used. In scrum requirements commonly are written in the user story format. The product backlog is the collection of all required user stories. From the given requirements of section 1.2 the following User stories have been extracted:

- As a User, I want to decide whether I stay logged in or not.
- As a User, I want to have an overview of all the threads.
- As a User, I want to open a new thread to talk with others about a specific topic.
- As a User, I want to add descriptions to my threads.
- As a User, I want a list of messages of a thread in reverse chronological order (newest first). Each list item contains information about who wrote a message, what kind of user role this person had (admin, project leader or user), when the message was written and what the subject of the message is.
- As a User, I want to read the content of a message.
- As a User, I want to swipe to through a message list.
- As a User, I want to write a text message for a specific thread.
- As a User, I want to reply to a message I'm reading.
- As a User, I want to have some kind of link to the origin message of a reply so that I can read immediately what has been replied to.
- As a User, I want to get a Notification on my mobile about new messages.

- As a User, I want to switch off notifications on my mobile.
- As a User, I want to change how often my mobile pulls new messages.

As requirements like the login are not a goal of a user they are just implicitly included in the backlog.

2.2 Current state

The current version of the blubb Android app is version 0.4. At this state it meets all requirements with few modifications due to a better usability.

One modification is that instead of a list with messages and a detailed view for single messages, the details are also displayed in the list. Because of this change, there is no need to realize the the extra view for single messages and the navigation by swiping.

For messages it is not possible to enter a subject. This makes it much easier to write messages and the subject has already been set by the thread. With this deviations from the original requirements the app is more similar to most chat apps. This makes the handling of the app easier for users since they can read and write messages in ways they already know.

In addition to the requirements some features were implemented. So it is possible to initialize and reset the password in the app and to perform a logout. To improve the overall view of the threads list, the thread description is not shown by default. Instead, the view for every thread can be switched to show it and the list of messages contains a header with the thread details. Furthermore the title, description and status of a thread can be modified.

A notification for a single message shows its author and content. Notifications for a collection of messages show details, too. Both types of notifications are also implemented for threads, so the user also gets notified about new threads.

3. Technical basics

3.1 Beap

3.1.1 Beap API

Beap is an application platform for medium and large scale desktop applications. It uses the flexibility of HTML and CSS for the user interface in combination with C++ compiled libraries and native system resources on the back end. It allows consistent, OS independent system access while ensuring security. Beap is server-client based and gives access to its system resources through a unified API, using well defined HTTP GET/POST requests and structured JSON responses. (Gross, 2014b)

3.1.2 BeapDB

BeapDB is an object oriented database running under the node.js JavaScript engine as an in-memory database with a focus on robustness, exetensibility/scriptability and data versioning.

Its main purpose is to serve as application specific data storage and to allow a variation of inline data operations while enforcing CRUD-, ACID- and type safeness.

All data entries are fully indexed through an object oriented and structure based meta representation. This allows quick queries and programmatically/scriptable access to individual entries or entire data sets.

Every database operations are done via functions or property setters/getters in JavaScript, which is the query language of beapDB. Queries and requests to beapDB are executed in their own, encapsulated environment. Transactions are session, permission, structure and type checked before applied to the core execution context.

Access to the beapDB is provided directly by the BeapAPI cycle (HTTP(S) / JSON). Therefore the beapDB and BeapSession are firmly tied together. On successful login, a session based JavaScript execution environment is created and remains active until logout or session timeout.

BeapDB responses follow the same rules and conditions as any other beapAPI-conform module (Gross, 2014a).

3.2 Android

Android is an operating system mainly designed for mobile devices like smartphones and tablet computers. With specialized user interfaces it also runs on some TVs (Android TV), on smartwatches and other wearable devices (Android Wear) and in cars (Android Auto).

Among mobile devices Android is the most installed and most used OS. At the Google I/O 2014, Sundar Pichai (Senior vice president at Google) announced that it has more than 1 billion 30-Day active users. With that it has far the biggest smartphone OS market share.

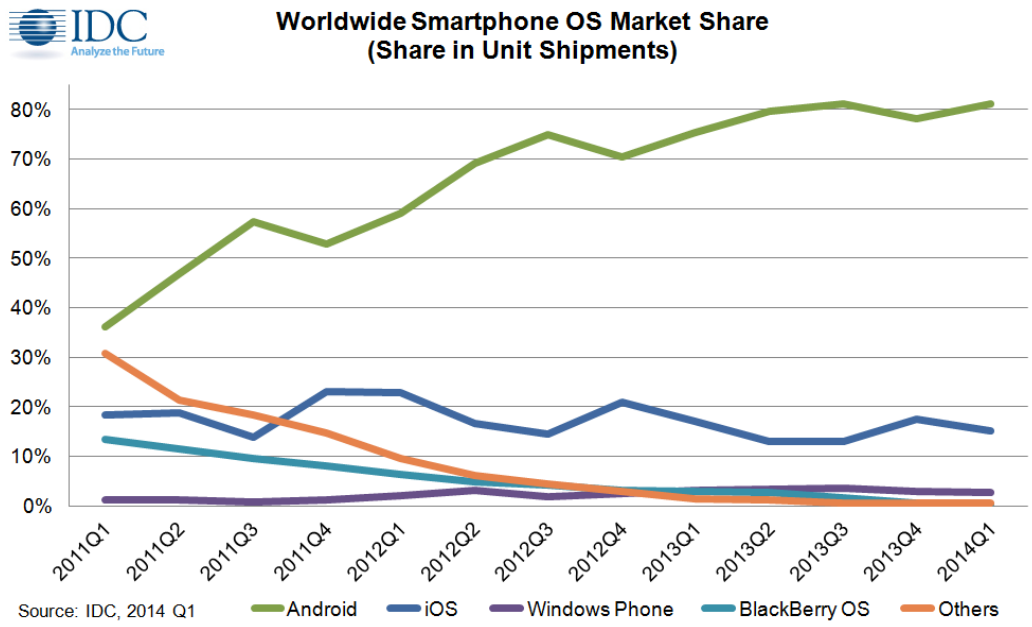

Figure 3.1: Worldwide Smartphone OS Market Share

Source: (Corporation', 2014)

With app stores user can download and install apps easily on their devices. The Google Play Store is the most popular with over 1 million apps. Besides many app stores offer the distribution of Android apps, e.g. the Amazon Appstore for Android.

Android is based on a Linux kernel. Every Android application runs in its own process, with its own instance of the Dalvik or ART virtual machine (Gandhewar & Sheikh, 2010).

The user interface is designed for touch input but also features voice input.

Apps are usually developed in the Java programming language using the Android Development Kit. But other toolsets are available too. Android NDK for development in languages like C and C++ (AndroidDeveloper, 2014c) and the Scripting Layer for Android (SL4A) for development in Python, JavaScript and other languages (Matthews, 2011).

4. Blubb Android app

4.1 Architecture overview

Beap lightweight user bulletin board is an application to manage the communication within a project. A mobile front end improves the availability of the project members. They become more reachable with their smartphones and tablets. Unlike a mobile website a native app offers the possibility that the user can read messages even offline. Additionally it can make use of notifications to grasp the attention of users and improve their availability.
In order to realize this task an Android application (blubb) has been implemented. This technical documentation describes the way blubb is operating. blubb can be separated in three main parts:

- Data storage,
- View (Front end, user interface) and
- Managers or controllers.

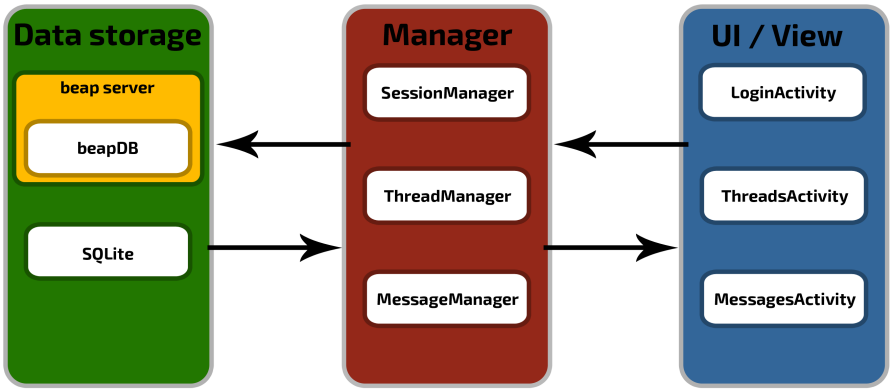

Figure 4.1: Architecture overview

The data storage itself consist of two nearly redundant databases. A beapDB running on a beap server is the central database for all user. It works as the exchange server for all data. The second is a local SQLite database mirroring a representation of the beapDB. This is needed to provide the offline access and to add some user specific fields to the data sets, e.g. a message has already been read.
Activity classes implement the user interface (UI). At the current state of the project there are four different activities:

- ThreadsActivity
- MessagesActivity
- LoginActivity
- SettingsActivity

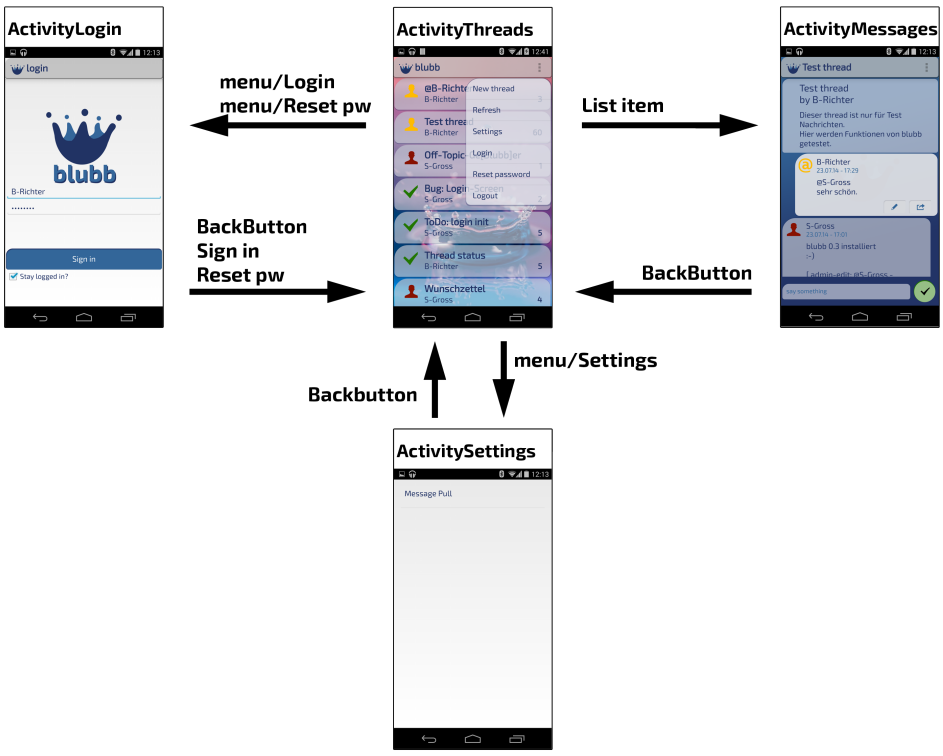

Figure 4.2: Activities navigation

The main purpose of the ThreadsActivity is to show the list with available threads but it is the center of the applications user interface, too.
A user can access all the other activities from there.

In response to a click on the list ithem, MessageActivity is started. This shows a list of BlubbMessages that belong to the clicked thread.
The LoginActivity and SettingsActivity will start when the corresponding item in the menu of ThreadsActivity has been clicked. In the LoginActivity the user can either log in or perform a password reset. In the SettingsActivity only the frequency of the PullService can be set. For details about this service see section 4.5.5.

The ThreadsActivity can be accessed from all other Activities using the BackButton.

The connection between the Activities and the data storage is made by three manager classes:

- SessionManager
- ThreadManager
- MessageManager

The SessionManager holds an object of the type SessionInfo and manages all matters about the session, like performing a login, auto login when possible, refreshes the session if necessary and provides the session ID, that is needed for all interactions with the beap server. Additionally it makes a quick check with the beapDB, to see whether new messages or threads are available.

Managing everything regarding threads is the task of the ThreadManager. The tasks are to get threads from beapDB, get threads from the SQLite database, compare the threads from beap and SQLite and mark new threads as 'new', add a thread to the data storage and modify an existing thread. The MessageManager has the corresponding tasks for messages.

4.2 Threads and messages

4.2.1 Purpose

blubb is an application to manage the communication for small scale projects. Communication is the act of sending a message from a sender to a receiver. In case of blubb the sender is always a single person but the receiver are all participants of the project. Even in a very small project a lot of messages

must be sent to maintain a proper communication. Therefore it is important to organize all messages in subjects, otherwise it would end in a big mess.

To implement this model three objects are needed. User representing the sender and receiver, messages that can be sent and threads to organize the messages to subjects. A user is represented by the user ID. That is also the unique user name within the application. Messages are implemented by the BlubbMessage class and threads by the BlubbThread class.

4.2.2 Messages

The class BlubbMessage

An object of the class BlubbMessage represents a message within the blubb application. Such an object holds all information about one message, plus it can create a MessageView for the visual representation in an Activity.

To hold all the information of the message a BlubbMessage has many fields, mostly of the type string:

- `String mID`
  Unique ID of the message.

- `String mTitle`
  Title of the message. For text messages this is optional but must be sent to the beapDB as "UNDEFINED".

- `MContent mContent`
  In the current version only TextContent is available, which is implemented by the class TextContent. For further content types the interface MContent must be implemented.

- `String mCreator`
  The unique ID/username of the user who wrote the message.

- `String mCreatorRole`
  This is mainly used to highlight special kinds of users like admins or project leader, since their content might be more important.

- `String mType`
  Through the message type it will be more easy to identify the content of the message.

- `String mThread`
  The unique ID of the thread the message belongs to.

- `String mLink`
  If the message is a reply to another message the link points to this message and contains the ID of it.

- `Date mDate`
  The date of creation of the message.

- `boolean isNew`
  The field isNew indicates whether the user of the app has read the message, yet. It will be set to 'true' when loading it the first time from the beapDB and to 'false' when the user closes the MessagesActivity for the messages thread (in `MessagesActivity.onStop()`).

A BlubbMessage can be constructed in two ways. Either with a proper formatted JavaScript Object Notation (JSON) object, like in listing 4.1 or by providing all of the fields above.

Listing 4.1: JSON object received form beap server

```
{
        "mId": "m2014-07-16_170322_S-Gross",
        "mType": "Message",
        "mCreator": "S-Gross",
        "mCreatorRole": "admin",
        "mDate": "2014-07-16T17:03:22.000+0200",
        "mThread": [
            "t2014-07-03_165629_B-Richter"
        ],
        "mTitle": "Thread bearbeiten",
        "mContent": "@B-Richter\nwerde ich checken ...",
        "mLink": "m2014-07-16_150532_B-Richter"
}
```

The MessageManager builds BlubbMessages when receiving messages from the beap server. For that it uses the JSON object in the response. The field 'isNew' will then automatically set to true.

The SQLite database uses the constructor where all fields must be set manually.

MessageView

A MessageView represents a message in the UI and is created by a BlubbMessage object. It is shown as an item in the ListView of the MessagesActivity.

How a MessageView looks depends on three factors. A message from the thread creator will have a lighter background, messages from the current user appear on the right side of the screen the others on the left side and new messages have a red background. Combining these, results in eight different appearances of messages. The basic layout for all is the `message_layout.xml`. The views contained in the MessageView are five TextViews, two Buttons and one custom ContentView. The TextViews are:

- `message_layout_icon_tv`
  Used for the icon of the message. It will show either the shape of a head or '@' if the message is a reply to another message. This is done via a custom font (BeapIconic.otf). According to the role of the creator the color will be red for admins, yellow for project leader and blue for regular user.

- `message_layout_creator_role_tv`
  Shows the creator's role as text. Currently it is set 'INVISIBLE'.

- `message_layout_creator_tv`
  Shows the user name of the message creator.

- `message_layout_title_tv`
  This TextView displays the title of a message. The current text messages are entered without title, so this will be mostly empty but nevertheless important for further message types.

- `message_layout_date_tv`
  The date when the message was created will be displayed in this TextView. In a BlubbMessage object the date is represented by a Date object. A SimpleDateFormat object from the BlubbMessage formats the string representation of the date to a specified pattern. This is defined in the constant `BlubbMessage.DATE_PATTERN`.

The figure 4.3 shows exemplary a message of a regular project participant and a reply of the app user, who is a project leader.

4.2.3 Threads

The class BlubbThread

In a project, there are different subjects to discuss. A thread contains the messages for one subject, e.g. bugs in a software development project. This gives the communication in a project a certain structure.

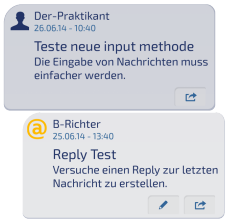

Figure 4.3: MessageView example

The class BlubbThread represents a thread in blubb. Like the BlubbMessage it has many fields to hold all information:

- `String tId`
  Unique ID for the thread.
- `String tTitle`
  The title or name for a thread.
- `String tDesc`
  Description for the thread, that should briefly explain its purpose.
- `String tCreator`
  ID of the creator.
- `String tCreatorRole`
  Like in a message, the creator role is mainly used to highlight special kinds of users like admins or project leader, since their content might be more important.
- `ThreadStatus tStatus`
  The status of a thread can either be OPEN, CANCELED or SOLVED. Is it CANCELED or SOLVED it is not possible to post any new messages. SOLVED indicates that the discussion has come to a successful end, while CANCELED shows that it has failed.
- `Date tDate`
  The date of creation of the thread.
- `int tMsgCount`
  The number of messages, belonging to this thread.
- `boolean isNew`
  This field tells whether the thread is new or not.

- `boolean hasNewMsgs`
  When the thread has new messages this will be 'true' otherwise it will be 'false'.

Like the BlubbMessage a BlubbThread has two constructors. One requires every fields described above as parameter the other just a proper formatted JSON object. The formatting of it is shown in listing 4.2

Listing 4.2: Thread JSON object

```
{
        "tId": "t2014-07-03_165629_B-Richter",
        "tType": "Thread",
        "tCreator": "B-Richter",
        "tCreatorRole": "PL",
        "tPath": [
            "Thread"
        ],
        "tDate": "2014-07-03T16:56:29.000+0200",
        "tTitle": "Bugs",
        "tDescr": "Bitte hier unerwartetes oder fehlerhaftes
            Verhalten der App eintragen. ",
        "tMsgCount": 10
    }
```

The ThreadManager builds BlubbThreads according to the JSON object, when receiving threads from the beapDB. Since the fields 'isNew' and 'hasNewMsgs' are not included in the response they will be set to 'true'. This is because only new threads are not contained in the SQLite database and will be added instead of updated.

The SQLite database uses the other constructor to create BlubbThread objects.

ThreadView

ThreadViews are created by BlubbThread objects. In fact there are a small and a big View to represent a thread in the ListView of the ThreadsActivity. The small shows the most important information about the thread like the title, creator, status and the message counter. Additionally the big view displays the creator role, date, description and a button for editing the thread. The user can switch these views with a long click on the current ThreadView.

The file `thread_head_layout.xml` defines the layout for the part both views use:

- `thread_status_tv`
  Shows the status of the thread. The font for this TextView is set to BeapIconic to show the icons indicating the status. A head or letter 'U' stands for OPEN, a green check mark for SOLVED and a red restricted sign for CANCELED. The color of the head stands for the role of the creator. Red is for admins, yellow for project leaders and blue for regular users, see figure 4.4.

Figure 4.4: Statuses of threads.

- `thread_title_tv`
  TextView for the title.

- `thread_autor_tv`
  TextView for the creator.

- `thread_msg_count_tv`
  TextView for the message counter. If the thread has a new message the color of it will be red.

The small ThreadView is defined in the file `thread_small_layout.xml` and contains only the head layout. Custom layouts can be integrated in layouts with the 'include' tag, see an example in listing 4.6.1.

Listing 4.3: Thread small layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/threadview_rc_transpgray_bg"
    android:orientation="horizontal"
    android:paddingLeft="20dp"
    android:paddingTop="10dp">

    <include layout="@layout/thread_head_layout" />
</LinearLayout>
```

In addition to the head layout the big view contains following items:

- `thread_info_tv`
  Shows the creator role and the date of creation for the thread.
- `thread_description_tv`
  TextView for the description of the thread.
- `thread_edit_button`
  This is the button to open the edit thread dialog. If the current user is not the creator of the thread it will be invisible. The font of it is the BeapIconic font and shows the letter 'E'.

The figure 4.5 shows an example of one small and one big ThreadView.

Figure 4.5: ThreadView example

4.3 Data storage

4.3.1 beapDB at the beap server

Communication with the beap server

Communication with the beap server is performed via http requests. Therefore it is necessary to declare the internet permission in the App Manifest. Also the network access can not work on the main thread. So it is important to start the request only from separate threads (in this case the java.lang.Thread). For details see section 4.6. The general sequence of events to perform a request is:

1. Build a request string with the RequestBuilder.
2. Execute the http request with an object of BlubbHttpRequest.
3. Receive the response string and build a BlubbResopnse object.
4. Pass this object to the calling method.

5. Extract the result object within the BlubbResponse according to what was requested. For example if a new tread has been created with the request the result will contain a JSON object upon which a BlubbThread can be build.

Building request strings

For the communication with the beap server two different kinds of request strings are needed. One kind for the beap server itself (all requests about the session). The others are queries at the beapDB. Therefore they have different beap IDs. A request string for session requests contains following parts:

- The url of the server, e.g. "http://blubb.traumtgerade.de:9980/".
- A question mark, indicating a request.
- The BeapId, "BeapSession".
- The desired action, either "login", "refresh", "logout", "check" or "setOwnPwd".
- The beap app version.
- Parameters describing the action:
  - Login: Username and password.
  - Check: BeapSession.
  - Refresh: BeapSession
  - Logout: BeapSession
  - ResetOwnPwd: Username, password, new password and the confirmation of the new password.

For example the request string for a login would look like this: http://blubb.traeumtgerade.de:9980/?BeapId=BeapSession&Action=login&appVers=1.5.0rc1&uName=Der-Praktikant&uPwd=test

The request for the beapDB is different and contains the following parts:

- The url of the server, e.g. "http://blubb.traumtgerade.de:9980/".
- A question mark, indicating a request.
- The BeapId, "BeapDB".

- The session ID, e.g. "50ae0c60282372467016c832840cd678".
- The action will always be "query".
- A query that will be executed at the beapDB, e.g. "tree.functions.getAllThreads(self)".

Parameters for the query are in braces. The first parameter is a reference to the session object. For blubb it is always its own session - "self". Following a list of available queries. They all have the prefix "tree.functions.":

- `getAllThreads(self)`
- `createThread(self, "Thread title", "Thread description")`
- `setThread(self, "Thread id", "Thread title"[opt.], "Thread description"[opt.], "Thread status"[opt.])`
- `getMsgsForThread(self, "Thread id")`
- `createMsg(self, "Thread id", "Message title"[opt.],MessageContent, "Message link"[opt.])`
- `setMsg(self, "Message id", "Message title"[opt.], "Message content"[opt.], "Message link"[opt.])`
- `quickCheck(self)`

Parsing parameter

Every parameter send to the beapDB like username, password, a text content of a message etc. needs to be properly escaped, URL-encoded and sent as UTF-8. The class BPC (Blubb Parameter Checker) uses a URLEncoder to encode parameter. Additionally special characters must be escaped like:

- newline "\n" to "\\n"
- tab "\t" to "\\t"
- "\\\""

When all newlines, tabs and quotation marks are escaped, the whole string is enclosed in quotation marks and then url encoded. Lets try a simple example. A user types a message like in listing 4.4

Listing 4.4: Example string

```
Always encode character like:
   "ü"
```

First the newlines, tabs and quotation marks must be escaped. The resulting string will look like in listing 4.5:

Listing 4.5: Escaped characters

```
"Always encode characters like:\n\t\"ü\""
```

After this the string is url encoded and will be send to the server like:

Listing 4.6: Url encoded string

```
"Always%20encode%20character%20like%3A%5Cn%5Ct%5C%22%C3%BC%5C%22"
```

The parameter send to the RequestManager class for all session request are parsed automatically. Parameter entered in a database query must be parsed with BPC by calling the static method `parseStringParameterToDB()`, like in listing 4.7.

Listing 4.7: Parse a string parameter

```
   ...
   String result = BPC.parseStringParameterToDB("\nübercool");
   ...
```

Response from beap

The response from beap will always be a JSON object with the following members, shown in listing 4.8.

Listing 4.8: JSON object for beap response

```
{
      BeapStatus : <int>,  /* Response status, e.g. 200 for OK.*/
      StatusDescr : <string>, /* A description for the status.*/
      Result : <var>,       /* A JSON object.*/
      sessInfo : {       /* Info object about the session.*/
         sessId : <MD5-string>,
         sessUser : <string>,
         sessRole: <string>,
         sessActive : <bool>,
         expires: <GMT-Date>
```

```
    }
  }
```

BeapStatus and StatusDescr describe the status of a request at beap, e.g. that the request was correct. The following are the most common statuses:

- 200 - OK
  The request could be processed properly and will contain the expected result.
- 203 - No content
  The request could also be processed properly but there is no result.
- 204 - session already deleted
  The session ID is not valid any more.
- 400 - request failure
  A response will have this status most likely if parameter are missing.
- 401 - login required
  A login is required before sending this request.
- 403 - permission denied
  This status will be send if a user has not the permission to request a certain action at the beapDB.
- 407 - connection error
  If there is no network connection or the server is offline the response object will have this status.
- 406 and 409 - parameter error
  In the request was something wrong with a parameter, e.g. a string has not been encoded properly.
- 418 - syntax or reference error
  The status for a error within a query string.

The result object will vary due to the request. Integer, JSON object or JSON array are the possible types of the result. For details see appendix A.

4.3.2 SQLite as a local database

The Android system offers different storage options (AndroidDeveloper, 2014k):

- shared Preferences
  key value pairs
- internal storage
  private data for app e.g in xml
- external storage
  public data on shared external storage
- SQLite databases
  Store structured data in a private database
- network connection
  Store data with own network server.

For blubb the SharedPreferences are used to store the username, password, settings of pull service and the number of threads and messages at beapDB to compare them and request if new are available.

BlubbThreads and BlubbMessages are stored with a SQLite database. The DatabaseHandler The database can be accessed through the DatabaseHandler. It implements the SQLiteOpenHelper, which manages the database creation and version management (AndroidDeveloper, 2014j).

The version number is a constant within the class and must be increased if any changes are made at the database, e.g. adding a new field to a table. If not done, the database will probably crash.

The names for the database, tables and column names are set with constants, too. They should be accessed only through these constants to avoid typos.

The database consists of two tables for threads and messages. They are created in `onCreate()`. This is called the first time the database is created. The primary key for the threads are the thread ID, for the messages the message ID, since they are already unique from the beapDB. The tables are connected via the thread ID which is given for every message. The tables are nearly the same as at the beapDBbut have columns for the boolean values 'isNew' in messages, 'isNew' and 'hasNewMsg' in threads. The boolean values are stored as ints and must be parsed. ¡ 1 equals true and 0 false.

In the listing 4.9 the plain SQL statement for the creation of threads and messages tables is shown:

Listing 4.9: SQL code for creating the table messages

```
CREATE TABLE messages(
  mId TEXT PRIMARY KEY,
  mTitle TEXT,
```

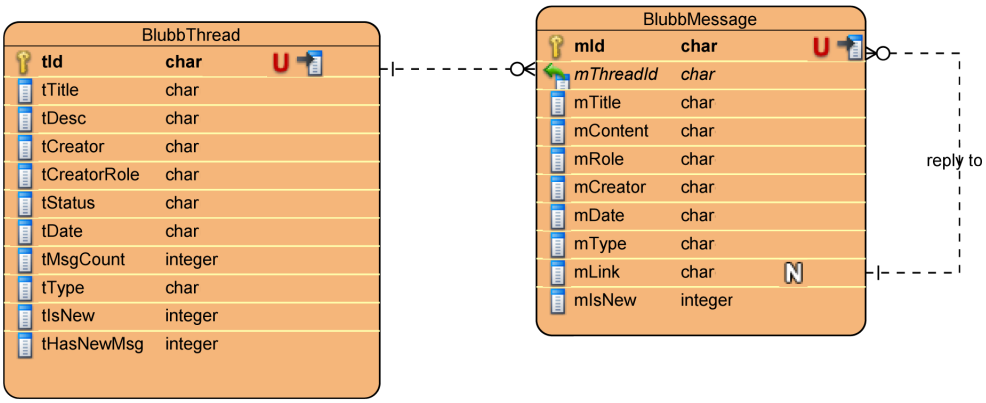

Figure 4.6: ERM of the SQLite database.

```
   mContent TEXT,
   mRole TEXT,
   mCreator TEXT,
   mDate TEXT,
   mType TEXT,
   mThreadId TEXT,
   mLink TEXT,
   mIsNew INTEGER)


CREATE TABLE threads (
   tId TEXT PRIMARY KEY,
   tTitle TEXT,
   tDesc TEXT,
   tCreator TEXT,
   tCreatorRole TEXT,
   tStatus TEXT,
   tDate TEXT,
   tMsgCount INTEGER,
   tType TEXT,
   tIsNew INTEGER,
   tHasNewMsg INTEGER)
```

The databaseHandler offers some methods to access the database directly:

- `addMessage(BlubbMessage message)`
  Adds a message to the database.

- `addThread(BlubbThread thread)`
  Adds a thread to the database.

- `getMessage(String mId)`
  Gets a message with the provided message id.

- `getThread(String tId)`
  Gets a thread from the database.

- `getAllMessages()`
  Gets all messages stored in the database. (not used yet)

- `getMessagesForThread(String tId)`
  Gets all messages for a thread.

- `getAllThreads()`
  Gets all threads stored in the database.

- `setMessageRead(String mId)`
  Sets the read flag of a message to true.

- `setThreadNewMsgs(String tId, boolean hasNewMsgs)`
  Sets the 'hasNewMsgs' flag of a thread.

- `updateMessage(BlubbMessage message)`
  Updates the values of a stored message, e.g. when the content has been changed.

- `updateMessageFromBeap(BlubbMessage message)`
  This updates without the isNew flag because this cannot be modified at the beapDB.

- `updateThread(BlubbThread thread)`
  Update the values of a stored BlubbThread, e.g. when the description has been changed.

- `updateThreadFromBeap(BlubbThread thread)`
  This updates a thread without changing the 'isNew' or 'hasNewMsgs'.

- `getMessageCount()`
  Gets the number of messages stored in the database.

- `getThreadCount()`
  Gets the number of threads stored in the database.

The database can easily be accessed from any class of the project by instantiating an object of the DatabaseHandler and calling the desired method, like in listing 4.10.

Listing 4.10: Accessing the SQLite database

```
...
BlubbMessage message = new BlubbMessage(result);
DatabaseHandler db = new DatabaseHandler(context);
db.addMessage(message);
...
```

In order to make an instance the applications context must be provided, due to that the SQLite database is only accessible for the blubb application.

4.4 Manager

4.4.1 Singleton pattern

All manager classes make use of the singleton pattern to provide global access to a single instance. The constructors of the SessionManager, ThreadManager and MessageManager have only private access. Each class holds one instance of itself as a private class variable. The static method `getInstance()` returns this instance and initializes it if necessary (Cooper, 2000), like in listing 4.11

Listing 4.11: Example for the singleton pattern

```
public class SessionManager {

   private static SessionManager instance;

   private SessionManager() {
   }

   public static SessionManager getInstance() {
       if (instance == null) {
         instance = new SessionManager();
       }
       return instance;
   }
}
```

The SessionManager mainly provides the session ID for interactions with the beapDB. With the singleton design pattern it can be accessed easily throughout the whole program.

In Android there is a little problem with the singleton pattern. The Android system can kill Activities that are not in the foreground in extreme low memory situations. Because of this it is possible that a singleton will not be available for other Activities, if it has been instantiated in the context of an Activity. The solution to this issue is to instantiate the singletons in the application context, that is available as long as the app is running. Therefore BlubbApplication extends the Application class. In its constructor it calls `getInstance()` on all three manager classes. In this way the ownership of all instances is transferred to the application's context and stay addressable for all contexts of blubb. To enable a custom Application class it must be declared in the manifest file like shown in listing 4.12.

Listing 4.12: Application tag of the manifest.xml

```
...
<application
        android:name=".blubbbasics.BlubbApplication"
        android:allowBackup="true"
        android:icon="@drawable/blubb_logo"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        ...
```

Special attention must be paid when subclassing a singleton. This can be difficult, since the subclass can work only if the base class has not been instantiated (Cooper, 2000, p. 46).

4.4.2 SessionManager

All queries at the beapDB need a valid session ID to authenticate the users identity and check whether the user has the permission to perform a certain query. After a login the beap server returns a response object. One part of it is always a SessionInfo object. The SessionManager holds this SessionInfo to provide access to it and the session ID within it.

If the user has allowed to saved the username and password, the login will be performed automatically. Otherwise the user need to log in manually. In both cases the username and password will be stored temporarily so the user must not be bothered again while the app is running. If the password is "init" the response status will be 'passwordInit', which means the user must

set an own password and should be asked to do so.

The SessionManager also provides a method to do a quickCheck at the beapDB and see whether there are any new threads or messages available. Beap will return a result array with just to integer. The first is the number of threads, the second the number of messages at the beapDB. If there are new threads or messages they will be requested and returned in a QuickCheck object containing two lists with only the new threads and messages.

Additionally the logout and password reset can be done through the SessionManager. Following a complete list of methods provided by the SessionManager:

- `getSessionID(Context context)`
  Returns a string with the session ID. If necessary and possible this will perform a auto login or refresh the session.

- `login(String username, String password)`
  For a manual login. After this the session ID will be available and the session will be refreshed automatically.

- `quickCheck(Context context)`
  Performs a quick check with beapDB and sees whether new threads or messages available. If so the new ones will be available in a QuickCheck object like:

  ```
  ...
  QuickCheck quickCheck = sessionManager.quickCheck(context);
  List<BlubbThread> threads = quickCheck.threads;
  List<BlubbMessage> messages = quickCheck.messages;
  ...
  ```

- `resetPassword(String username,`
  `String oldPassword, String newPassword, String confirmNewPassword)`
  Performs a tree.functions.resetPwd(...) at the beap server and returns 'true' if this action was successful.

- `getUserId(Context context)`
  Returns a string with the current users username.

- `logout(Context context)`
  Logs the user out of the beap server, the session will not be active any more and a formerly, at the SharedPreferences saved password will be deleted. After this the user needs to log in manually again.

4.4.3 ThreadManager

The Thread manager manages the access to all available threads. When it requests the threads from the beapDB it updates the SQLite database immediately. Following a complete list of public methods of the ThreadManager:

- `getAllThreads(Context context)`
  Gets first the threads from beapDB and updates the SQLite database. Finally it returns the list of threads stored in the SQLite database.

- `getNewThreads(Context context)`
  Returns a list of BlubbThreads with all threads that have the 'isNew' tag set to true.

- `updateAllThreadsFromBeap(Context context)`
  Updates all threads at the SQLite database from the beapDB. Returns 'true' if the task has been completed successfully.

- `getAllThreadsFromSqlite(Context context)`
  Returns a list of BlubbThreads with all the threads stored in the local SQLite database.

- `getThreadFromSqlite(Context context, String tId)`
  Returns a single thread from SQLite database as a BlubbThread object.

- `createThread(Context context, String tTitle, String tDescription)`
  Creates a new Thread at the beapDB. The thread will be added to the SQLite database and the new BlubbThread object will be returned.

- `readingThread(Context context, String threadId)`
  Sets the 'isNew' tag of a thread to 'false' and updates the SQLite database.

- `setThread(Context context, BlubbThread thread)`
  Updates the thread at beapDB and the SQLite database.

4.4.4 MessageManager

Like the ThreadManager for threads, the MessageManager manages all concerns about messages. It is the link between the data storage, local and at the beap server and the user interface. Through the following methods it provides this service:

- `getNewMessagesFromAllThreads(Context context)`
  Returns a list of BlubbMessages of all messages from all threads with the 'isNew' set to 'true'.

- `createMsg(Context context, String... messageParameter)`
  Creates a new message within the beapDB. If the task has been executed successful, the message will be added to the SQLite database as well.

- `getAllMessagesForThread(Context context, String tId)`
  Returns a list of all messages for one thread.

- `getAllMessagesForThreadFromBeap(Context context, String tId)`
  Returns a list of all messages for a thread from the beapDB.

- `putMessageToSqliteFromBeap(Context context, BlubbMessage message)`
  Puts a message to the SQLite database. If the database contains a message with the same message ID it will be updated otherwise it will be added.

- `getAllMessagesForThreadFromSqlite(Context context, String tId)`
  Returns a list of all messages for a thread, stored in the local SQLite database.

- `setMsg(Context context, BlubbMessage message)`
  Changes a message at the beapDB and the SQLite database.

4.5 View or Front end

4.5.1 Basics

In Android the user interface is realized by different Activities. They provide a screen with which users can interact in order to do something(AndroidDeveloper, 2014a).

Different Layout types provide different structures for the screen layout, e.g. a LinearLayout orders its children in a linear structure, either vertically or horizontally. A RelativeLayout will order its children relatively to each other, e.g. the second child on the right side of the first.

The children of these Layouts can be Views, ViewGroups or Layouts. Android provides a collection of both View and ViewGroup subclasses (AndroidDeveloper, 2014m), like TextViews, ImageViews, Buttons and many

more. The Views are Items at the screen the user actually sees and interacts with. ViewGroups hold other Views in order to define the layout of the interface. All Layouts inherit from ViewGroup.

Blubb's UI basically consists of TextViews, EditTexts and Buttons. TextViews only only display text in a 'readonly' manner. The user can not interact with the text. For this, a special kind of TextView, the the EditText has been created for this purpose. Buttons also are just a specialized TextView, mostly different in its appearance.

It is possible to define the user interface programmatically or in XML files. The Android framework provides the possibility to use either or both. For blubb mainly the XML files are used. When developing an application for Android it is important to consider the Activities life cycles. Figure 4.7 shows this life cycle.

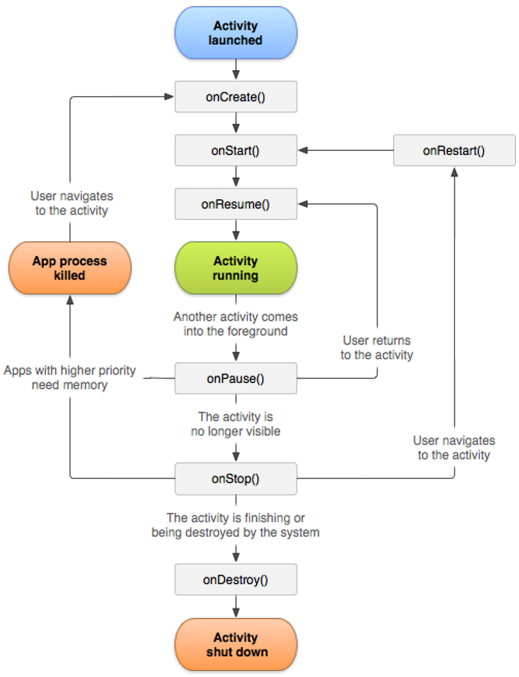

Figure 4.7: Application lifecycle.

Source: (AndroidDeveloper, 2014b)

The content view is usually set in `onCreate()`. That is, the UI layout is placed in the window of the Activity. Other actions, e.g. saving the current state of the application, should be considered at the appropriate life time events.

4.5.2 ThreadsActivity

At the start of the application the first Activity is the ThreadsActivity. Its main purpose is to display all available threads. But it also acts as the center of the application, since all navigation goes through it. At the ThreadsActivity the user can perform the following actions :

- Create a new thread.
  By clicking on 'New thread' in the menu the user opens a dialog where he can enter the title and description and create a new thread.

- Refresh the list of threads.
  This menu option will reload the threads from the beapDB, update the SQLite database and the thread list in the Activity.

- Go to the Settings screen.
  The SettingsActivity will start when the user clicked 'Settings' in the menu.

- Go to the Login screen.
  In case the user needs to log in manually he can start the LoginActivity with the LoginType LOGIN, by clicking 'Login' in the menu.

- Go to the reset password screen.
  To reset the password the user opens the menu and clicks 'Reset password'. This will start the LoginActivity with the LoginType RESET.

- Log out.
  If the user wants to log out of blubb he/she clicks 'Logout' in the menu. In case the password was stored in the SharedPreferences it will be deleted, the PullService will be stopped and the Application will be send to the background. Unfortunately it's not possible to close or kill an application. This is reserved to the Android system.

- Toggle the size of a thread view.
  A long click at one ThreadView will toggle its size between the big and the small view.

- Modify a thread.
  The user can modify his own threads by clicking at the 'EditButton', with the pencil icon. This will open a dialog window where the title, description and status can be changed.

- Go to the messages of a thread. A click on a thread within the list opens the MessagesActivity for this thread.
- Scroll through the list to see all threads.
  If the list has more than 7 items the user needs to scroll to see all entries.

Start of the Activity

When the ThreadsActivity is started it will first check whether it is possible to log in at the beap server. If the username and password are not stored at the SharedPreferences it will check whether threads are available at the local SQLite database. In case neither a login is possible nor any threads are stored the LoginActivity will be started. Otherwise the BlubbThreads are loaded form the SQLite database and displayed in the ListView of the activity. But if a automatic login is possible the BlubbThreads will be updated form beap and reload to the list.

Layout structure

The Layout for the ThreadsActivity is defined in `threads_activity_layout.xml`. It just defines the ListView for the threads, and a ProgressBar to show the loading process when a network action is performed.

For a thread within the ListView two different Views are available. The small one (`thread_small_layout.xml`) shows only an icon, the title of the thread, who created it and how much messages it contains. Additionally the big view (`thread_big_layout.xml`) shows a date when the thread has been created, the role of the creator and the description. If the current user is the creator of this thread, a button will be shown, too. With it the user can open the edit thread dialog.

Two dialog windows are also part of the ThreadsActivity. With the CreateThreadDialog (`create_thread_dialog.xml`) the user can open a thread for a new subject. For this he/she must enter the title and a description. To change this, e.g. because of a typo, the user can open the EditThreadDialog, defined in (`edit_thread_dialog.xml`) and change it. With this the status of a thread can be changed, too.

Figure 4.8 shows all layouts that are part of the ThreadsActivity.

Menu

Most actions a user can perform with the ThreadsActivity are accessed through the Menu. The Menu is a common UI component, to provide a

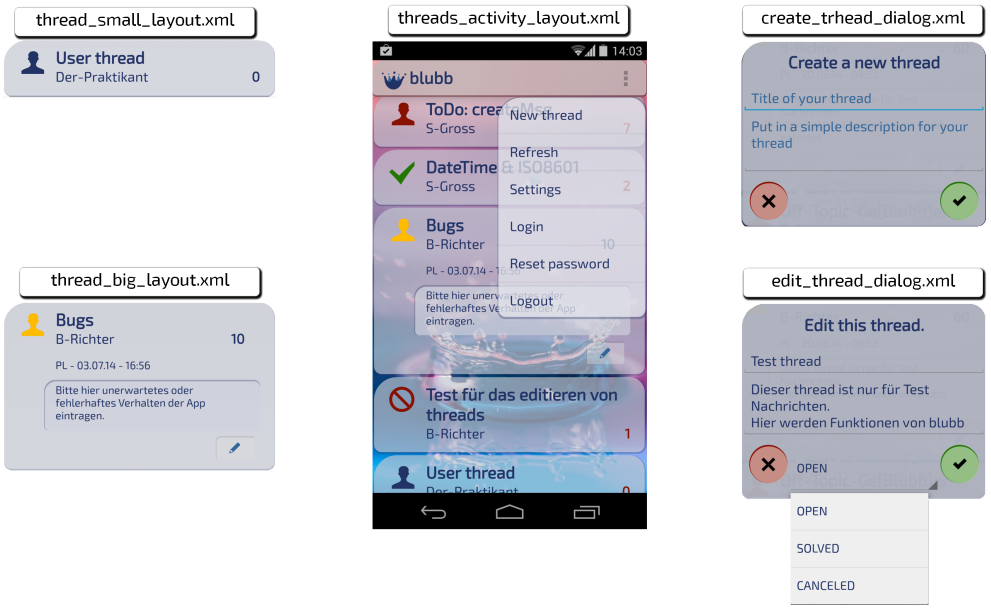

Figure 4.8: Layouts for the ActivityThrads.

familiar and consistent user experience. In order to present actions and other options in Activities the Menu API is used (AndroidDeveloper, 2014f). Some devices even have a hardware button to access it. The menu for ThreadsActivity is defined in `activity_threads_menu.xml`. The actions for the menu items are determined in the `ThreadsActivity.class` within the `onOptionsItemSelected()` method.

ListView, ArrayAdapter and ThreadViews

A ListView is a ViewGroup that displays a list of scrollable items(AndroidDeveloper, 2014g). The items are loaded to the ListView via a ListAdapter. In case of the ThreadsActivity, the ListAdapter is an ThreadArrayAdapter, inheriting a ArrayAdapter and containing a list of BlubbThread objects. The items for the ListView are provided by these objects. If an object on its list needs to be displayed, the ThreadArrayAdapter calls `getView()` on it. Depending on the current state of the thread it will return a small or a big View object.

Each thread has an OnClickListener and an OnLongClickListener, both given to it in the constructor of the ThreadArrayAdapter. A click on a thread view creates a new Intent. The Intent starts an MessagesActivity for the clicked thread. To load the right messages the MessagesActivity needs the thread ID as an Extra in the Intent. The name of that Extra is declared

in the constant `MessagesActivity.EXTRA_THREAD_ID`.

A long click on the view will change its size by switching its layout between `thread_small_layout.xml` and `thread_big_layout.xml`. A button on the big layout starts the EditThreadDialog.

Create thread and edit thread dialogues

To create a new thread the user has to enter the title and a description in a CreateThreadDialog. The layout for is defined in `create_thread_dialog.xml`. It consists of two LinearLayouts. The first contains a TextView for the dialog title, two EditTexts for entering the title and description for the thread. Two buttons are in the second LinearLayout. The green Button is to create a new thread with the entries of the two EditTexts. The red Button cancels the dialog window.

When the user clicks the green button a new AsyncNewThread will be created and executed. For details see 4.6.1. `newThreadDialog()` starts a CreateThreadDialog and must be called from within an ThreadsActivity object. The dialog to modify a thread is similar build like the dialog to create a new thread. Except that between the two buttons lays a Spinner, containing the different statuses for the thread. A Spinner is an Android equivalent for a drop-down list or combo box. The user can select one of three predefined items. For the status they are:

- Open
- Solved
- Canceled

If the user hits the green button on this dialog an AsyncSetThread will be executed. For details see section 4.6.1. The EditThreadDialog starts if `editThreadDialog()` is called from within an ThreadsActivity object.

4.5.3 MessagesActivity

The MessagesActivity will start when the user clicks on one thread in the ThreadsActivity. Its purpose is to show all messages belonging to the clicked thread. The appearance of a message depends on who has created it and whether it is new or has been loaded previously. The user can perform different actions in this activity:

- Write a new message.
  Clicking in the EditText on the bottom of the activity will open the soft keyboard. Alternatively this can also be achieved by clicking 'New message' in the menu of the activity. After entering a text, the message can be sent by clicking the green button on the right side of the EditText. An AsyncSendMessage uses the MessageManager to send the message.

- Refresh the list with messages.
  This action can be accessed through the Menu. It will reload the messages from the beapDB with an AsyncGetAllMessagesToThread and if new messages are available show them on top of the ListView.

- Reply to a message.
  Every message has a small button to reply to it. Like when writing a new message, this starts the soft keyboard and works the same way as writing a new message but will be send as a reply. The AsyncSendMessage is used to do this, too.

- Edit a message.
  The user can modify own messages by clicking the edit button besides the reply button. The current content of the message will be set to the EditText and be editable for the user. Clicking the green button, will send the content of the EditText with an AsyncSetMessage.

Start of the MessagesActivity

It is important to add a thread ID as an Extra to the Intent that starts the MessagesActivity. With this ID the messages can be requested form the SQLite database and the beapDB. On start of the activity it will first prepare the message ListView with a header containing the thread's title, creator and description. The layout for the header is defined in `messages_activity_lv_header.xml`. It will also start the InputView, so the user can enter messages. After that the messages are loaded from the SQLite database and then from the beapDB. In this way the user sees most messages immediately and the new just pop up when the used AsyncGetAllMessagesToThread has finished its task.

Layout structure

The file `activity_messages_layout.xml` defines the layout for MessagesActivity. Like the ThreadsActivity it contains a ListView and a ProgressBar.

The ListView shows the messages and the ProgressBar indicates that a background process is running. On the bottom of the activity window the InputView supplements the layout. Here the user enters and sends new messages. It consists of an EditText and a Button. For instance a new message can be entered to the EditText and be sent by clicking the Button. The InputView is used to write a new, modify or reply to an existing message. For this it has different modes that are set up by the methods `startInputView()`, `replyToMessage()` and `changeMessage()` in the MessagesActivity. From the Menu the user can refresh the messages and start the creating of a new one.

The ListView of the ActivityMessage is populated with MessageViews through a MessageArrayAdapter inheriting from the ArrayAdapter class. In its constructor it expects a list of BlubbMessages. To get the View for a message the MessageArrayAdapter just invokes `getView()` on the corresponding object of this list.

Menu

The menu at the ActionBar of a MessagesActivity has only two items. One to sets up the InputView for creating a new message, the other just to reloads the messages from the beap server. Clicking the item 'Create a new message' will simply call `startInputView()`, which sets up the InputView properly. 'Refresh' executes an AsyncGetAllMessagesToThread, for details see section 4.6.2.

4.5.4 LoginActivity

Only registered user can access the beapDB. For this they get a username and a password. With these can the user perform a login in the LoginActivity. This Activity has two different modes. One for a regular login the other to reset the password. The mode is triggered by an Extra contained in the starting Intent. This Extra is a LoginType and can be accessed like shown in listing 4.13. The constant `EXTRA_LOGIN_TYPE` holds the name needed to access the extra in the Intent.

Listing 4.13: Accessing the LoginType.

```
Intent intent = getIntent();
 LoginType loginType =
     (LoginType)intent.getSerializableExtra(EXTRA_LOGIN_TYPE);
```

LOGIN is the standard LoginType. If the user enters a valid username and password and clicks the LoginButton it will perform a login and start the ThreadsActivity.

Was the AcitvityLogin started with the LoginType RESET, the user can enter the username, password, a new password and the new password again to detect typos early. Clicking the LoginButton will then reset the password and redirect to the LoginActivity for a regular login.

The very first login is a special case. At this point the user has not yet chosen an own password. 'init' is the standard password and must be reset. If the user sends a login with the init password a dialog window will pop up in which the password must be reset.

Layout structure

Compared to the ThreadsActivity or MessagesActivity the layout of LoginActivity is pretty simple. It contains a ImageView for the logo, four EditTexts for the username and passwords, a Button and a CheckBox.

- `login_activity_logo_iv`
  ImageView which shows the logo and name for the application.

- `login_activity_username_et`
  The EditText where the user enters the username.

- `login_activity_password_et`
  The EditText for the current password.

- `login_activity_password_reset_et`
  EditText for a new password, when resetting it. Only visible when the LoginType is RESET.

- `login_acitivity_password_reset_confirm_et`
  In this EditText the user must enter the same as in

  `login_activity_password_reset_et` to avoid typos. It will also be visible only when the LoginType is RESET.

- `login_activity_login_btn`
  The text on the LoginButton depends on the LoginType. For LOGIN it is 'Sign in', for RESET 'Reset password'. In LOGIN mode it will start an AsyncLogin, in RESET mode an AsyncResetPassword. See section 4.6.3 for details to this AsyncTasks.

- `login_activity_stayloggedin_cb`
  If this CheckBox is checked when the user clicks the LoginButton, the username and password will be stored in the SharedPreferences. This enables an automated login, so the user must not be bothered with the login process on every start of the application.

The file `login_activity_layout.xml` contains the layout for the LoginActivity.

The InitDialog is defined in the file `password_init_dialog.xml`. Like the LoginActivity it contains 4 EditTexts for username, password, new password and the confirmation of the new password. Since the user already has entered the username and the init password this fields will be filled in advance. The green Button starts a AsyncResetPassword, the red Button cancels the dialog.

4.5.5 Notifications

A notification is a message that can be displayed to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time (AndroidDeveloper, 2014h). For blubb the notifications are used to show new messages or new threads. A background service, the PullService, is responsible for publishing these notifications. It is started by an AlarmService provided form the Android system and only starts an AsyncQuickCheck. This performs a quickCheck at the SessionManager. If the returned QuickCheck object contains any results they will be published as notifications and will look like in figure 4.9

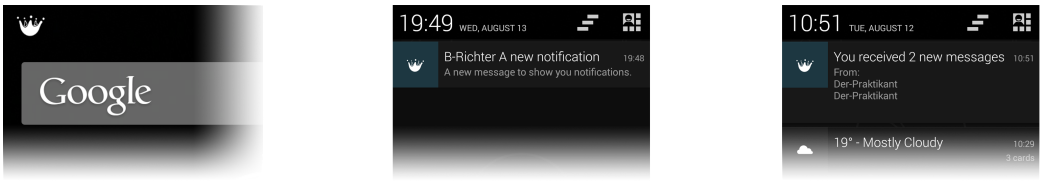

Figure 4.9: Notifications from blubb

A click on a notification in the notification drawer will open either the ThreadsActivity or the MessagesActivity. If there is one new message the MessagesActivity will open for the corresponding thread. For more than one message or a thread notification the ThreadsActivity will be started.

The PullService only can perform a quickCheck when the user has accepted to stay logged in in the LoginActivity, otherwise the mandatory login is not possible.

4.6 Asynchronous tasks

AsyncTasks

Tasks like network operations should not be executed on the UI thread (java.lang.Thread). They probably may take some seconds to finish and would block the user interface during their operation. So it is recommended to run long lasting jobs on a background thread. Android provides different mechanisms for running a task in the background. Services are meant for long lasting tasks and are not affected by the life cycle events of an activity. But they can't interact directly with the UI.
blubb is mostly considered to display content provided by the beapDB. Therefore many requests are sent by the UI. For instance the creation of a new thread must be send to beap. A service could execute the request but the update of the user interface would be tricky. The abstract class AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads or handlers(AndroidDeveloper, 2014e). There are two rules for background threads (AndroidDeveloper, 2014i):

1. Do not block the UI thread
2. Do not access the Android UI toolkit from outside the UI thread

With AsyncTask both can be met easily. Therefore all network operations at blubb run on them. All AsyncTasks are inner classes of an Activity. They extend the class AsyncTask and must implement the `doInBackgroud()` method, which will run on the background thread. The result of it is delivered to `onPostExecute()`. Since this runs on the UI thread, it is easy to show the results.

Exceptions in AsyncTasks

AsyncTasks lack a proper way of exception handling. There is no build-in mechanism to transfer exceptions form a background to the main thread. But this is important to show any message about a malfunction on the UI.

To address this issue the AsyncTask classes of blubb have a field for a BlubbException. All custom Exceptions inherit from BlubbException. In

case a Exception is thrown in `doInBackgroud`, it must be caught and set to this field. The handling of it can then take place in `onPostExecute`, see listing 4.14.

Listing 4.14: Exception handling in AsyncTasks

```
private class AsyncNewThread extends AsyncTask<String, Void,
    BlubbThread> {

        BlubbException blubbException;

        @Override
        protected BlubbThread doInBackground(String... parameter) {
            String title = parameter[0];
            String description = parameter[1];
            try {
               ThreadManager manager = ThreadManager.getInstance();
               BlubbThread thread = manager.createThread(
                       ThreadsActivity.this, title, description);
                return thread;
            } catch (BlubbException e) {
                this.blubbException = e;
            }
            return null;
        }

        @Override
        protected void onPostExecute(BlubbThread thread) {
            getApp().handleException(blubbException);
         ...
        }
   }
```

Usually the Exceptions are forwarded to the BlubbApplication. This will show a Toast message according to the type of Exception. The text for each message is defined in the file `res/values/strings.xml`.

4.6.1 AsyncTasks of the ThreadsActivity

AsyncUpdateThreads

If the threads need to be updated from the beapDB a AsyncUpdateThreads will call the ThreadManager' `updateAllThreadsFromBeap()`. The threads table at the SQLite database will then be updated with the threads from the

beapDB. When they have been updated the ListView in the ThreadsActivity gets all threads from the SQLite database to display them. Since the AsyncUpdateThreads needs no parameter it can simply been instantiated and executed, like shown in listing 4.15

Listing 4.15: AsyncUpdateThreads

```
...
AsyncUpdateThreads asyncUpdateThreads = new AsyncUpdateThreads();
asyncUpdateThreads.execute();
...
```

AsyncCheckLogin

When starting the application it is necessary to check whether it is possible to log in automatically. Therefore an AsyncCheckLogin is executed. It does not need any parameter since it tries to log in with the saved username and password in the SharedPreferences. The task is executed by calling `getSessionID(..)` at the SessionManager.

If it is possible to log in the threads will be updated and reload to the thread ListView. If not, the thread list is loaded immediately from the SQLite database. In case the thread list from the database is empty, this must be the first login of the user and the LoginActivity will be started.

AsyncNewThread

A new thread must have a title and a description. Therefore the AsyncNewThread receives two string arguments in its `execute()` method. Like shown in listing 4.16. The first parameter is the title, the second the description.

Listing 4.16: AsyncNewThread example

```
...
String title = "Bugs";
String desription = "Please report bugs in this thread.";
AsyncNewThread asyncNewThread = new AsyncNewThread();
asyncNewThread.execute(title, description);
...
```

The creation of the new thread will be made by the ThreadManager with the method `createThread(...)`.

After finishing the execution, the application will display a short message to inform the user about the successfully accomplished task. The ListView for the threads will be updated so the new thread is shown, too.

AsyncSetThread

To modify a thread a AsyncSetThread is needed. It receives a BlubbThread object, with the new title, description or status in its constructor, see listing 4.17.

Listing 4.17: AsyncSetThread example

```
...
BlubbThread thread = bugThread;
thread.setTitle("Bugs and complaints");
AsyncSetThread asyncSetThread = new AsyncSetThread(thread);
asyncSetThread.execute();
...
```

Like the creation of a thread the modification will be executed by the ThreadManager. The AsyncSetThread simply calls `setThread(..)` on the instance of it.

If the modification of the thread was successful a Toast informs the user about it and the currently displayed threads will be updated.

4.6.2 AsyncTasks of the AcitvityMessages

AsyncGetAllMessagesToThread

With a AsyncGetAllMessagesToThread the messages can be updated from the beapDB and shown on the UI in the MessagesActivity. In the background thread it simply calls `getAllMessagesForThread()` on the MessageManager. A string containing the thread ID is the only argument needed and is given to this AsyncTask in the constructor. Listing 4.18 shows an example, how to use an AsyncGetAllMessagesToThread.

Listing 4.18: AsyncGetAllMessagesToThread example

```
...
String threadID = "t2014-07-03_165629_B-Richter";
AsyncGetAllMessagesToThread asyncTask = new
    AsyncGetAllMessagesToThread(threadID);
asyncTask.execute();
...
```

After executing the task it will update the MessageArrayAdapter in the MessagesActivity.

AsyncSendMessage

An AsyncSendMessage is used to create new messages. The arguments for this are Strings with the thread ID, the message title, the content and if it is a reply, a message ID. An example how to use AsyncSendMessage is shown in listing 4.19.

Listing 4.19: AsyncSendMessage example

```
...
AsyncSendMessage asyncTask = new AsyncSendMessage();
String tID = "t2014-07-03_165629_B-Richter";
String title = "A new bug";
String content = "I have found a new bug!";
String link = "m2014-07-03_165817_B-Richter";

asyncTask.execute(tID, title, content, link);
...
```

When the new message has been created the AsyncSendMessage will update the ListView in the MessagesActivity through the MessageArrayAdapter and show the new message on top of the list.

AsyncSetMessage

The AsyncSetMessage will update an existing message at the beapDB as well as at the SQLite database. It works pretty much like the AsyncSendMessage, except that the only argument is the modified BlubbMessage object.

4.6.3 AsyncTasks of the LoginActivity

AsyncLogin

An AsyncLogin will perform a login at the beap server and store the SessionInfo in the SessionManager, so all requests to the beapDB can use the session ID to verify the identity of the user. To execute a login a string with the username and one with the password is needed as arguments in the `execute()` method. After a successful login the ThreadsActivity will be started and if the user has checked the CheckBox the username and password will be stored in the SharedPreferences. If a PasswordInitException has been caught, the

password init dialog will be started by calling `showPasswordInitDialog()` of the current LoginActivity.

AsyncResetPassword

With a AsyncResetPassword object the password of a user can be reset. It needs four strings as arguments in the `execute()` method, the username, old password, new password and the new password again to confirm it. After it has finished its task, it will start the LoginActivity with the LoginType LOGIN. So if the user has entered a wrong new password this will be recognized immediately. Has a init password dialog started the AsyncResetPassword, it will be closed.

4.7 Android Manifest

Every application must have an `AndroidManifest.xml` file (with precisely that name) in its root directory. The manifest file presents essential information about an app to the Android system (AndroidDeveloper, 2014d). In blubb the manifest declares the ActivityThread as the main activity and the activities LoginActivity, MessagesActivity and SettingsActivity. The PullService is also defined in the manifest. Since the app uses a custom Application class this is stated within the application tag.

To make use of protected features of an Android device, `<uses-permission>` tags declaring the permissions must included in the `AndroidManifest.xml` (AndroidDeveloper, 2014l).

The permissions needed for the app are the 'internet' and 'vibrate'. Of course is a network access mandatory to communicate with others and the vibration of the Android device is used to inform the user about new messages.

5. User documentation

5.1 Basics

5.1.1 What is blubb?

Blubb stands for beap lightweight user bulletin board. It is meant to be an application to manage the communication within small projects. In the moment it consists only of an Android app but a web app will follow soon. With the app you can write messages in different threads and create new threads about a subject. You can use it if you have an Android device running at least Android 4 or Ice Cream Sandwich.

5.1.2 How to install the blubb app on your Android device.

The blubb is not available in an appstore. But it is easy to install it manually:

1. Copy the blubb.apk file to the internal or external storage of your device.

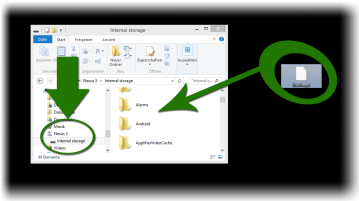

Figure 5.1: Copy blubb.apk file to a device

2. Open it with a file manager software on your device.
3. Click on the 'Install' button on the bottom of your screen. Now blubb will be installed.
4. After the app has been installed you can start it by clicking 'Open'.

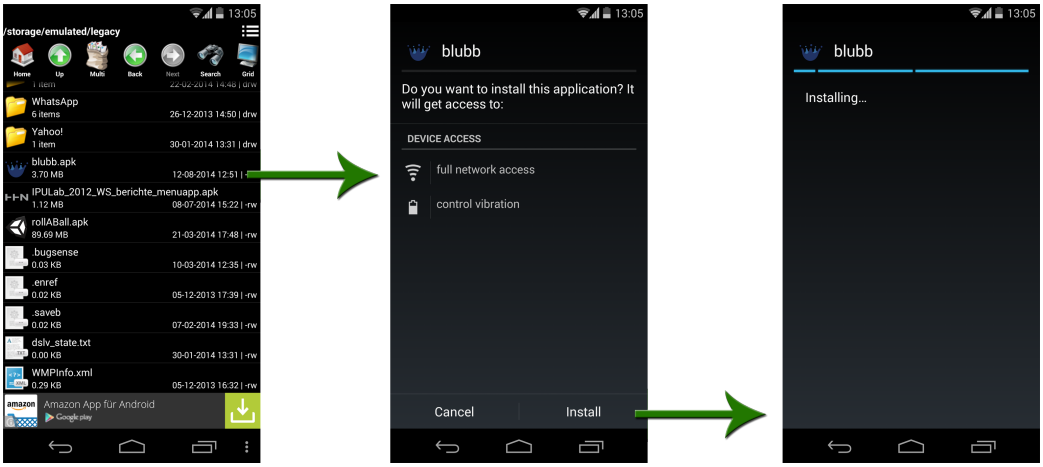

Figure 5.2: Install app on device

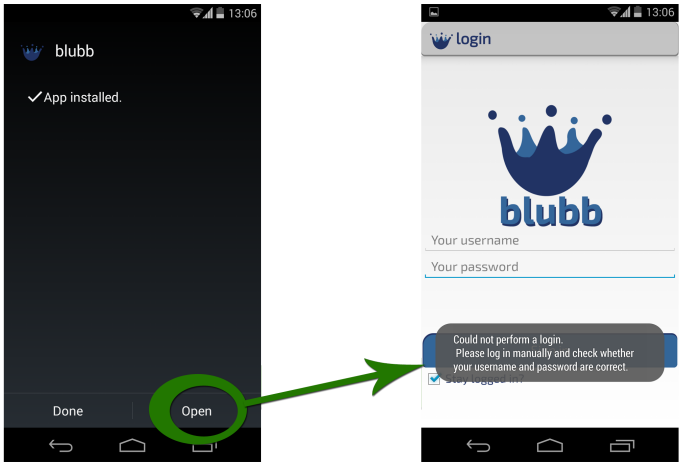

Figure 5.3: Open app after installation

5.2 Login

5.2.1 Why is the login needed?

The login is needed to identify the user of the blubb app. Otherwise it would not be possible to know who wrote a message or opened a thread. In this way you can recognize the author of a message.

5.2.2 How to login.

If you have received an username from your admin you can use the blubb app. If this is the first time you use it, you must initialize your own password

first. For details to the first login see section 5.2.4.

Now just enter your username and password and click the 'Sign in' button. If the username and password were correct the thread overview will start and show you all available threads. Otherwise you will see a message telling you that something was wrong and you need to sign in again.

Figure 5.4: Login

5.2.3 Why stay logged in?

At the login screen you will see a check box 'Stay signed in'. You can disable this option by clicking on the check mark. Then you have to log in every time you start the app. All threads and messages you have loaded previously will stay on your device, so you can read them even without login. But you can not see any new messages or threads and can not receive notifications.

It is recommended to leave this option checked. The username and password will be stored secure and only accessible for the blubb app.

5.2.4 The first login - How to initialize the password

When you log in the first time, you will not have set your password. The standard password is 'init'. After a login with this password a dialog window will open and ask you to set your own password. The password must be at least 6 character long. You have to enter it twice to avoid typos. The green button stays disabled till the two new passwords are the same. If you have already set your password and want to cancel the dialog just click the red button.

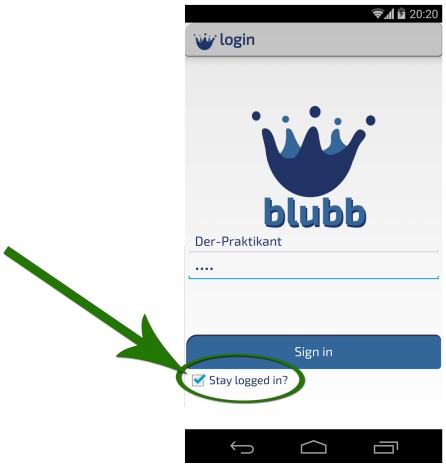

Figure 5.5: Stay logged in

Figure 5.6: Initialize the password

5.2.5 How to reset a password

If you want to reset your password click on the 'Reset password' button at the menu of the thread overview. This will open a new screen where you can reset your password. Like at the first login you have to enter the new password twice. Now click the 'Reset password' button and the password will be reset. You will see a short message at the screen when the password has been reset. After this you must log in again with your new password. So if you had a typo in your password you will recognize it immediately.

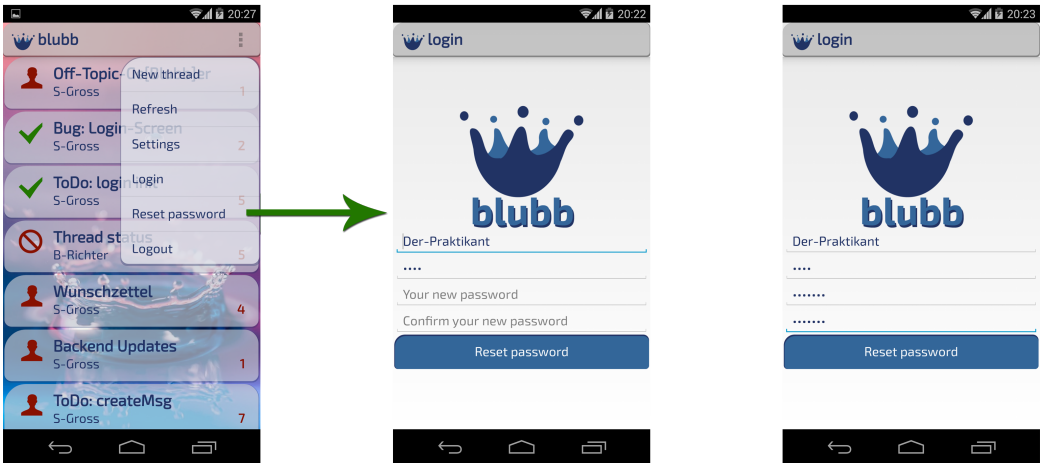

Figure 5.7: Reset the password

5.3 Threads

5.3.1 What is a thread at blubb?

Threads are a way to group messages. In blubb you will start a thread about a specific topic. All messages about this topic should be posted within its thread.

The threads in the list of the thread screen show information like the title how many messages the thread contains, the username of the creator and the status. If new messages are available the number of messages will be red instead of blue.

The icon of the status shows whether it is open or already closed. If open, the icon will be the shade of a head. The color of it depends on the role of creator in the project. Is it red the creator is an admin, yellow stands for project leaders and blue for regular user.

A check mark indicates that the thread has been closed and the discussion was successful. For example, if the subject was the planing of a meeting and the meeting is over, the status of the thread can be set to SOLVED. If the meeting could not take place, e.g. the participants could not agree on a date, CANCELED would be the right status for this thread.
In a detailed view of a thread additionally the date of creation and the description for the thread will be shown. And if you are the creator of this thread a button to access the edit dialog to it.

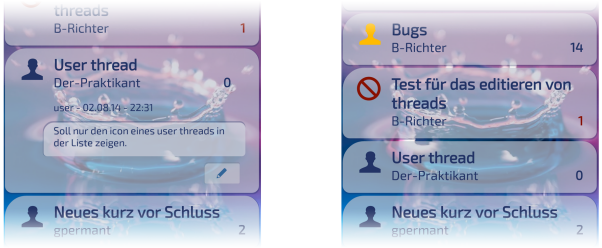

Figure 5.8: Different views of threads

5.3.2 What is the thread screen?

At the thread overview you can see all threads you have access to. They are in a list on the screen. Maybe you need to scroll through the list to see all threads. If you need a thread and you can not see it, you do not have the permission to view it. Tell your admin to give you access to this thread, too.

The thread overview is also the center of the application. All other screens can be accessed from it and all screens go back to it.

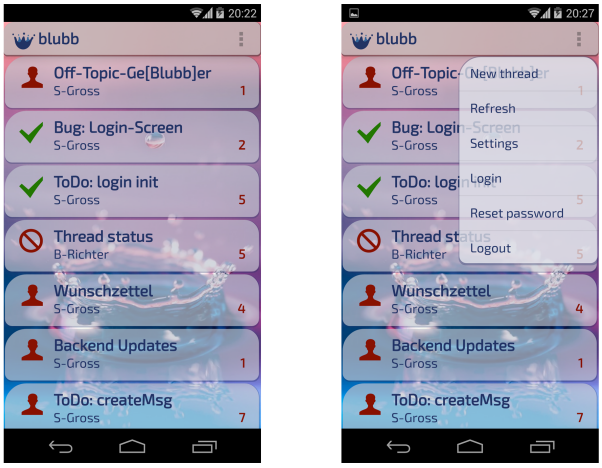

Figure 5.9: The thread screen

5.3.3 How to view details about a thread

If you want to see details about a thread do a long click on it. This will open a detailed view of the thread. To close the detailed view just perform a long click on the thread again.

You also will see the more details like the description when you open the thread by a regular click.

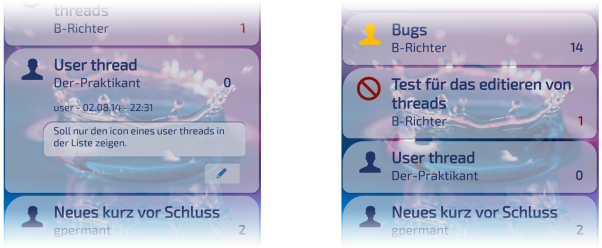

Figure 5.10: Switch the view of a thread

5.3.4 How to create a new thread

To create a new thread, open the menu of the thread view and click 'New thread'. A dialog window will open. Here you can enter the title/name of your thread and a short description about its subject. When you have finished this click the green button and the thread will be created. A short message at the screen will confirm that the thread has been created. New threads pop up at the bottom of the thread list. When you can not see your new thread scroll to the end of the list.

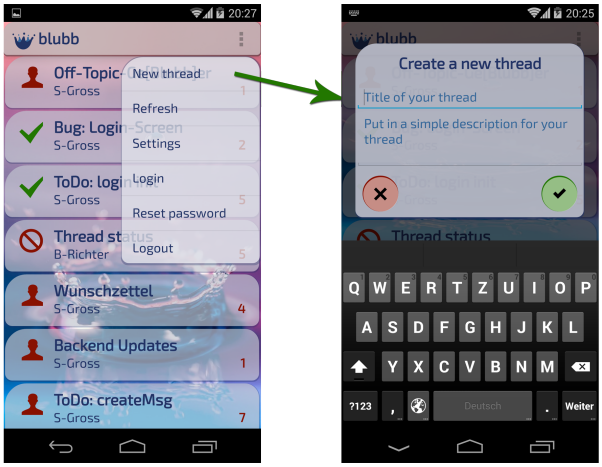

Figure 5.11: Create a new thread

5.3.5 How to change a thread title, description or status

If you need to change the thread title of description, e.g. because of a typo, you can do this by clicking the edit button at the detailed view of the thread. Only if you have created the thread you will see this button, so you can not change the threads of others. After clicking this button a dialog window will

open. It looks much like the dialog window to create a new thread, except that between the to buttons on the bottom you can change the status of the thread, too.

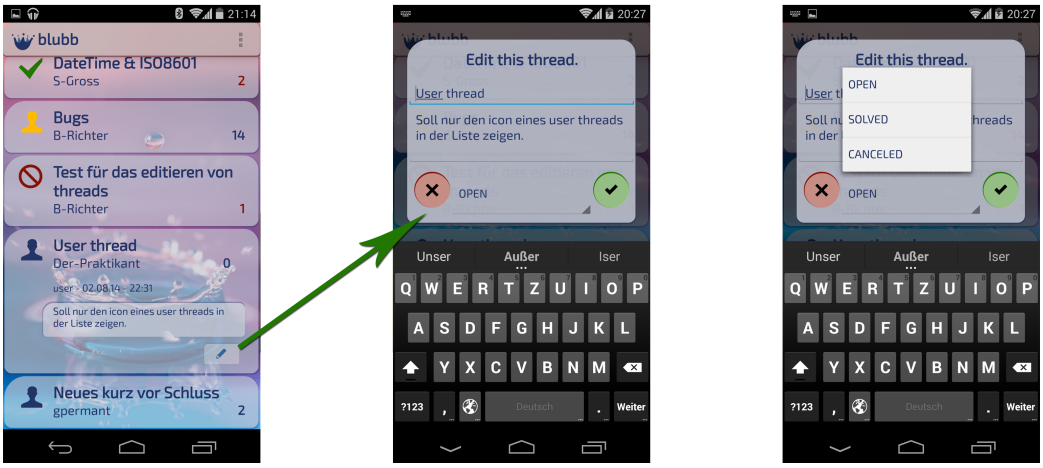

Figure 5.12: Edit a thread

Clicking the green button will confirm your entries and modify the thread.

5.3.6 How to refresh the thread list

If you want to check whether a new thread has been opened, click the 'Refresh' button in the menu of the thread screen. You can close and reopen the app to get the same result.

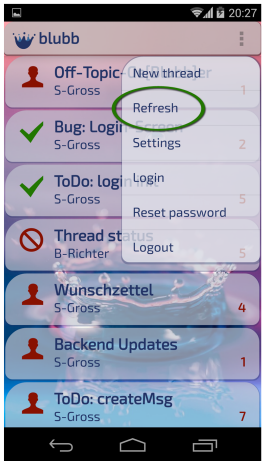

Figure 5.13: Refresh the thread list

5.3.7 How to show the messages of a thread

The messages of a thread can be accessed by clicking on the thread view in the thread overview. A screen, which shows all messages belonging to it, will be shown then.

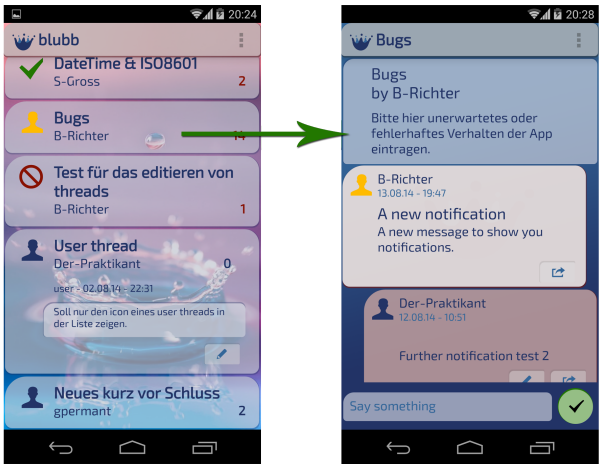

Figure 5.14: Show the messages for a thread

5.4 Messages

5.4.1 What is a message in blubb?

A message is a single post within a thread. It can link to a previous post, then it is a reply.

Like the thread views message views show an icon, looking like the shade of a head. This represents the role of the message author. Red is for admins, yellow for project leaders and blue for regular user. This will help you to identify more important messages. In case the message is a reply, the head will be replaced by an '@'. Additionally the message view shows the username of its author, the date of creation of the message and the message content. At the bottom right corner is the reply button. If a message is your own you can edit it by clicking the edit button (with the pen) besides the reply button.

Message views can look different, depending on who created them, whether it is your own message or whether it is a new message. New messages are all messages of a thread that were posted since the last time you opened a thread. A normal message has a white background. New messages have

red backgrounds. In both cases the background is slightly transparent. If the message is from the thread creator, the background is less transparent. Your own messages will be aligned at the right side of the screen. All other messages are aligned at the left side.

5.4.2 What is the message screen?

At the message screen you can see all messages of a thread. At the bottom of it is the InputView where you can enter new messages.

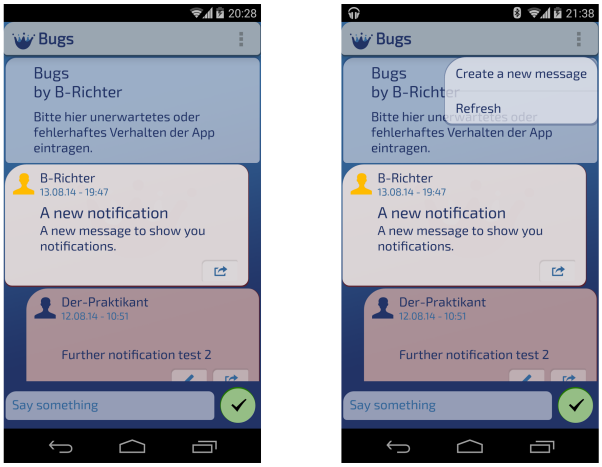

Figure 5.15: The message screen

5.4.3 How to write a new message

To write a new message follow these steps:

1. Open the desired thread.
2. Click in the InputView at the bottom of the screen or open the menu and click 'Create a new message'.
3. Type your message.
4. Click the green button with the check mark.
5. Your new message will be at the top of the message list.

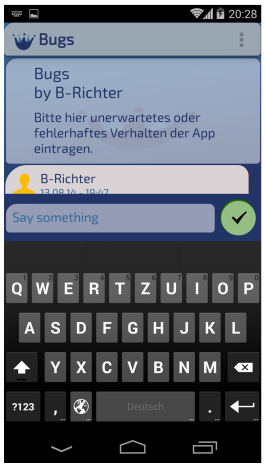

Figure 5.16: Write a message

5.4.4 How to change a message

Like threads you only can change your own messages. If you want to change the content of a message, e.g. because of a typo, just click the edit button at the bottom of the message view. The text will be set to the InputView and you can change it. Click the green button with the check mark when you have made the changes.

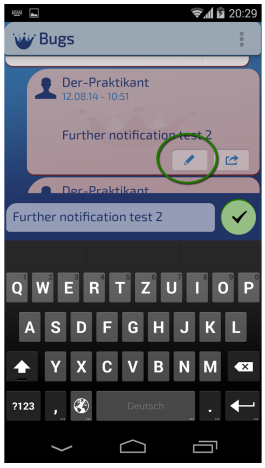

Figure 5.17: Edit a message

5.4.5 How to reply to a message

If you want to reply to a message, click the reply button on the bottom of the message view. The InputView will open and show the hint 'Reply to ...'.

When you have entered your message click the green button to send it.

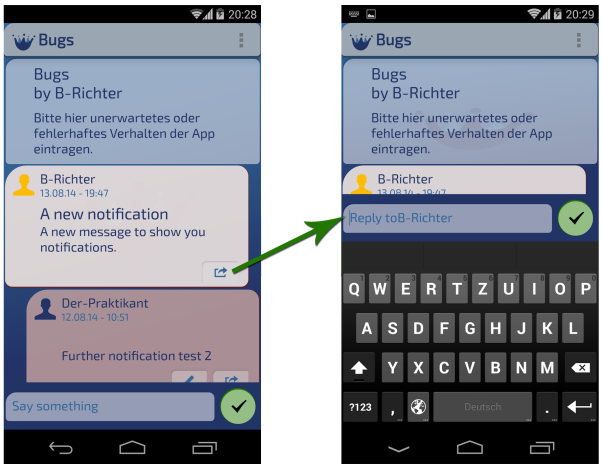

Figure 5.18: Reply to a message

5.4.6 How to send a new message when stuck in reply or change mode

If you have not finished a reply or the changing of a message the InputView will stay in this mode. To write a new message when stuck in this modes open the menu of the message screen and click 'Create new message'. Alternatively you can go back to the thread screen and reopen the thread.

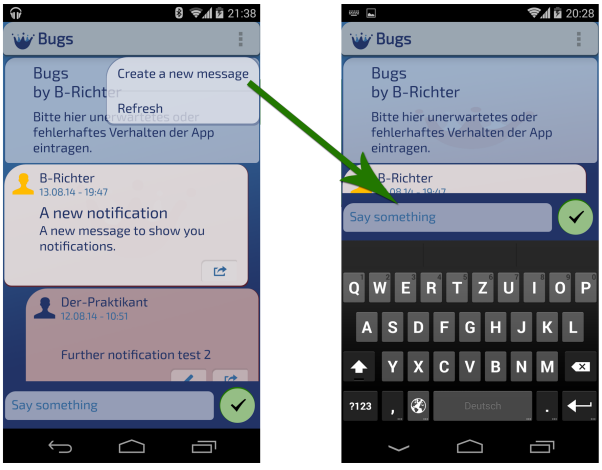

Figure 5.19: Write a new message when stuck in another input mode

5.4.7 How to refresh the message list

To refresh the message list open the menu of the message screen and click 'Refresh'. All new messages will be shown on the top of the message list.

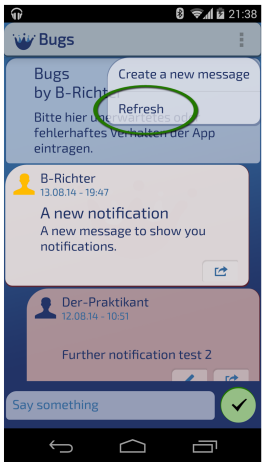

Figure 5.20: Refresh the message list

5.4.8 How to view the message a reply was written to

When you read a reply message you probably want to read the message this is the reply to. To view it click on the '@' icon of the reply. The message list will scroll to the message and the message view will squeeze.

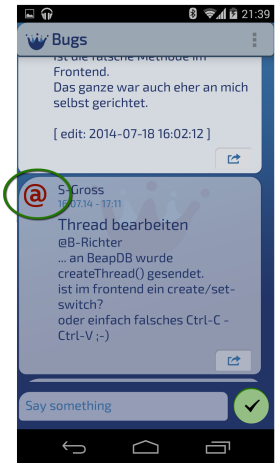

Figure 5.21: Show a message a reply was written to

5.5 Notifications

5.5.1 What is a notification?

A notification informs you about new messages or threads. If you enable notifications you can stay up to date without currently running the blubb app.

A small blubb icon in the notification area will tell you that you have received a new message or thread. If you open the notification drawer you will see the details to the new message or thread. More than one new item will be shown in a list.

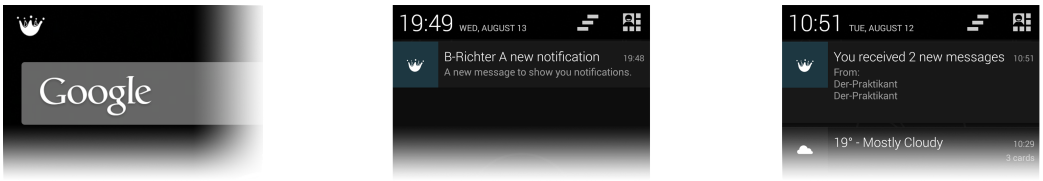

Figure 5.22: Notifications

Touching a notification opens the blubb app. If it is a message the message screen for thread of the message opens. A touch on a thread notification or a notification for more than one message opens the thread screen.

A background service requests frequently whether new messages or threads are available at the beap server. This may increase your data usage. To change the frequency of requests or disable the notifications, see section 5.5.2.

5.5.2 How to change the notifications settings

Touching the 'Settings' button in the menu of the thread screen opens the settings screen. Touch the option 'Notification frequency'. In the 'Notifications' dialog you can change the frequency of requests to the beap server or disable the notifications.

Figure 5.23: Open the notifications settings

6. Conclusion

6.1 Evaluation of the project

With blubb a user can view a list of all threads and lists with messages for each thread. It is possible to write, edit and reply to messages. New threads can be created and existing threads can be modified. For new messages or threads the user can receive notifications. The frequency of requests to the beap server can be set in settings or turn off. The user can log in, log out, initialize or reset the password. Furthermore the whole app runs reliable and without any known bugs.

With all this features it is possible to manage the communication of a project. The user can write messages and discuss a certain topic. But not only discussions are possible. For example can a project leader open a thread for tasks. His messages in this thread will then represent the tasks. The other users reply to this tasks and e.g. report that has been executed.

Polls can be executed in a similar way. The project leader opens a thread. The thread title contains a question and the other user post their answers.

Due to the modular architecture, it is easy to extend the application. Furthermore the parts responsible for the communication with the beap server can be reused for other apps.

A project can be measured by its scope, schedule and cost. The cost is not further considered because for blubb no cost limit was set. Since blubb meets all set requirements and has no delay in delivery the project for its implementation has been finished successfully.

6.2 Prospect of blubb

With blubb it is possible to manage project communication. But the modular structure of it makes it possible to extend the application easily. Following some ideas for features that could extend it:

- User profiles.
  A profile for each user would meet the demand of people to present themselves.
- Pictures, audio or video files in messages.
- Further thread types, e.g. a poll type with automatic evaluation.
- Communication to the beap server via HTTPS to improve the security of the application.

A. Projcet request from Siggi Gross

Sehr geehrter Herr Richter,

vielen Dank für Ihr Interesse an unserem Projekt.
Nach unserem heutigen Gespräch sende ich Ihnen anbei eine Zusammenfassung der Details.

Android-App zur firmeninternen Projekt-Kommunikation

Zur internen Kommunkation innerhalb eines Projektes wird eine Datenbankbasierte Software erstellt, die ähnliche Funktionen wie die eines klassisches Forums erfüllen soll. Alternativ könnte man diese auch als internes "Mini-Facebook" oder "Project Dashboard" auffassen.

Die Software soll im Verbund mit und auf Basis eines bereits bestehenden Backends und Frameworks verwendet werden. Dieses verwendet eine auf node.js basierende Datenbank, auf die über HTTP und eine bereits existierende API zugegriffen werden kann. Datenaustausch-Format ist JSON.

Zielsetzung

Im Umfeld kleinerer oder mittelständische Unternehmen ist Projekt-Management oft ein "vernachlässigtes Kind" oder so gut wie nicht vorhanden. Aus diesem Grund haben wir in den letzen Jahren ein webbasiertes Software-Paket entwickelt, daß die Erfassung, Bearbeitung und Auswertung über eine vereinheitlichtes Backend und Fronent-Framework ermöglicht.

In diesem Zusammenhang wird die Kommunktation der Mitarbeiter des Projektteams untereinander ein immer wichtiger Faktor und die "klassische" Kommunkation über Telefon oder E-Mail ist nicht immer sehr effektiv oder zuverlässig. Daher ist es nur logisch, die projektinterne Kommunikation an der gleichen Stelle anzusiedeln, wo die praktische Arbeit erfolgt.

Alle Mitarbeiter eines Projektes sollen sich über das gemeinsame Dashboard über den aktuellen Stand der einzelnen Teilaspekte des Projektes informieren bzw. über diese mittels eigenen Kurznachrichten aktualisieren können. Am Schreibtisch (PC/Laptop) erfolgt dies über eine browserbasierte Anwendung, die in die Gesamt-Software integriert ist.

Die Android-App soll diese Funktionalität ergänzen und Mitarbeitern und Projektleitern, die viel unterwegs sind eine Möglichkeit geben, dies auch von ihren Mobilfunkgeräten aus komfortabel zu tun und immer auf dem neuesten Stand zu bleiben.
Eine native Android-Anwendung wäre daher wünschenswert, da diese erfahrungsgemäß besser und flüssiger zu bedienen ist als eine Webanwendung in den gängigen mobilen Browsern.

Des weiteren sollte die Android-App die nativen Benachrichtigungs-Funktionen verwenden, um die Benutzer auf neue Nachrichten hinzuweisen, ähnlich denen, die bei neue eintreffenden SMS, Mails oder Facebook- Nachrichten verwendet wird.

Funktionsumfang und Komponenten

Login
: über Benutzername und Passwort. Es wird zwischen drei verschiedenen Typen von Benutzer-Accounts unterschieden: [Reguläre Benutzer | Projektleiter | Administratoren]
Das Session-Handling erfolgt über eine vom Server erzeugte, eindeutige Session-ID, die für alle weiteren Anfragen zwingend erforderlich ist.

Gesamt-Übersicht
: aller Teilaspekte/Threads des Projektes.
Threads mit neuen Nachrichten sollten entsprechend markiert sein.
Erlaubt die Auswahl oder Erstellung eines Neuen Threads und öffnet die ...

Thread-Ansicht
: Diese listet alle Nachrichten des gewählten Threads in umgekehrter chronologischer Reihenfolge (newest first)
in ihrer Kurzform. Jeder Listeneintrag enthält Benutzer-Name und -typ (Siehe Benutzer-Accounts), Datum/Uhrzeit und Betreff der jeweiligen Nachricht. Dabei sollten je nach Benutzer-Typ des Erstellers eine optische oder farbliche Unterscheidung möglich sein.
Eine Unterscheidung zwischen gelesenen und neuen Nachrichten wäre wünschenswert, ist aber nicht zwingend erforderlich. Die Ansicht erlaubt die Auswahl einer einzelnen Nachricht, bzw die Verfassung einer Neuen.

Nachrichten-Ansicht
: zeigt die jeweils ausgewählte Nachricht mit allen Details an:
Benutzer-Name und -Typ, Datum/Uhrzeit, Betreff und (mehrteiligen) Nachrichten-Inhalt.
Eine Navigation zur jeweils nächsten bzw vorherigen Nachricht sollte über Swipe-Up/Down möglich sein. Swipe-Left sollte zurück zur Thread-Ansicht führen. Aus dieser Ansicht sollte ebenso die Verfassung einer Antwort möglich sein.

Verfassen-Ansicht
: zur Erstellung einer neuen Nachricht. Deren Aufbau ist vordefiniert und enthält sowohl verpflichtende als auch optionale Felder. Weitere Datenfelder (Benutzer/Ersteller, Datum/Uhrzeit ...) sollten automatisch generiert werden.

Update und Benachrichtigungen
: Die App sollte beim Server in definierbaren Abständen nach neuen Nachrichten fragen und über die Android Notification Funktionen darauf hinweisen (ähnlich wie SMS, Mail, Facebook Apps). Ein Hinweis auf die Anzahl der neuen Nachrichten ist dabei ausreichend.

Einstellungen
: Auto-Login Ein/Ausschalten, Benachrichtigungen Ein/Ausschalten, Update-Zeitraum (1,5,10,30,60 min)

Kto: 100 820 075
BLZ: 733 699 02
Raiffeisenbank Kempten
St. - Nr. : 65 119 / 36 004
siggi@xl-programmierung.de
tel: +49 170 / 3 220 220

Umsetzung und Infrastruktur

Zur Umsetzung wird Ihnen ein Administrations-Zugang zu einem funktionsfähigen Test-Server mit Datenbank und BackEnd zur Verfügung gestellt,
Dokumentation sowie eine detaillierte Beschreibung der API sind vorhanden.

In der Gestaltung und technischen Umsetzung der Software haben Sie, abgesehen von der API zum Serverzugriff und den vordefinierten JSON-Formaten völlig freie Hand. Wir unterstützen Sie dabei gerne beratend in jeder Phase von Konzeption zur Implementierung.
Da Sie diese Software im Rahmen Ihrer Bachelor-Thesis umsetzen wollen, sind diesbezüglich vielleicht noch einige Absprachen nötig.

Copyright und Rechte

Sie behalten an der entstehenden Android-App aller Rechte und veröffentlichen diese unter Ihrem eigenen Namen bzw. einer Lizenz Ihrer Wahl.

Wir entwickeln die Gesamt-Software (BackEnd, Datenbank, Webanwendung) unsererseits im Moment ohne konkreten Kundenauftrag. Diese werden wir unter unserem Namen und entsprechendem Lizenzmodell veröffentlichen und unseren Kunden zur Integration in bestehende bzw. in Arbeit befindliche Projekte anbieten.

In diesem Zuge werden wir, Ihr Einverständnis vorausgesetzt, die von Ihnen entwickelte App ebenfalls anbieten. Die Erlöse aus Verkauf/Nutzung Ihrer App werden wird entsprechend berücksichtigen und Ihnen gutschreiben.

Sollten Sie für weitere Informationen oder Referenz-Material benötigen, bitte wenden Sie sich an uns.

Über eine Zugsage Ihrerseits würden wir uns sehr freuen,
und sehen einer erfolgreichen Zusammenarbeit hoffnungvoll entgegen.

Mit freundlichen Grüßen

Siggi Gross

Kto: 100 820 075
BLZ: 733 699 02

Raiffeisenbank Kempten
St. - Nr. : 65 119 / 36 004

siggi@xl-programmierung.de
tel: +49 170 / 3 220 220

B. Requests and responses from beap

B.1 Session requests

To try an url please change parameter to valid values, e.g. the username and the password or the session id.

Login

BeapId=BeapSession
Action=login
appVers=1.5.0 rc1
uName="username"
uPwd="password"
http://blubb.traeumtgerade.de:9980/?BeapId=BeapSession&Action=login&appVers=1.5.0rc1&uName=Der-Praktikant&uPwd=xxxx

```
{ /* ReturnOkObj */
    "BeapStatus" : 200,
    "StatusDescr" : "OK",
    "sessInfo" : {
        "sessId" : "5a379bd91cc16ca1dc01198f54fbb55d",
        "sessUser" : "Der-Praktikant",
        "sessRole" : "user",
        "sessActive" : true,
        "expires" : "2014-07-31T22:28:54.000+0200"
    }
}
```

Check session

BeapId=BeapSession
Action=check
sessId="session id" http://blubb.traeumtgerade.de:9980/?BeapId=BeapSession&
Action=check&sessId=5a379bd91cc16ca1dc01198f54fbb55d

```
{ /* ReturnOkObj */
    "BeapStatus" : 200,
    "StatusDescr" : "OK",
    "Result" : 6,
    "sessInfo" : {
        "sessId" : "5a379bd91cc16ca1dc01198f54fbb55d",
        "sessUser" : "Der-Praktikant",
        "sessRole" : "user",
        "sessActive" : true,
        "expires" : "2014-07-31T22:32:21.000+0200"
    }
}
```

Session refresh

BeapId=BeapSession
Action=refresh
sessId="session id" http://blubb.traeumtgerade.de:9980/?BeapId=BeapSession&
Action=refresh&sessId=5a379bd91cc16ca1dc01198f54fbb55d

```
{ /* ReturnOkObj */
    "BeapStatus" : 200,
    "StatusDescr" : "OK",
    "Result" : 5,
    "sessInfo" : {
        "sessId" : "5a379bd91cc16ca1dc01198f54fbb55d",
        "sessUser" : "Der-Praktikant",
        "sessRole" : "user",
        "sessActive" : true,
        "expires" : "2014-07-31T22:32:21.000+0200"
    }
}
```

Logout

BeapId=BeapSession
Action=logout
sessId=”session id” http://blubb.traeumtgerade.de:9980/?BeapId=BeapSession&
Action=logout&sessId=5a379bd91cc16ca1dc01198f54fbb55d

```
{ /* ReturnOkObj */
    "BeapStatus" : 200,
    "StatusDescr" : "OK",
    "sessInfo" : {
        "sessId" : "",
        "sessUser" : "",
        "sessRole" : "",
        "sessActive" : false,
        "expires" : {

        }
    }
}
```

Reset Password

BeapId=BeapSession
Action=setOwnPwd
uName=”username”
uPwd=”password”
newPwd1=”new password”
newPwd2=”new password again”
http://blubb.traeumtgerade.de:9980/?BeapId=BeapSession&Action=setOwnPwd&
uName=”Der-Praktikant”&uPwd=”xxxx”&newPwd1=”xxxx”&newPwd2=
”xxxx”

```
{ /* ReturnOkObj */
    "BeapStatus" : 200,
    "StatusDescr" : "OK"
}
```

B.2 Database requests

General request

The queries to the beapDB have all the same request structure but different query strings and result objects. The following sections will focus on these queries and their results.
BeapId=BeapDB
sessId=”session id”
Action=query
queryStr=”the query string”
http://blubb.traeumtgerade.de:9980/?BeapId=BeapDB&sessId=5a379bd91cc16ca1dc01198f54fb
Action=query&queryStr=tree.functions.getAllThreads(self)

```
{ /* ReturnOkObj */
    "BeapStatus" : 200,
    "StatusDescr" : "OK",
    "Result" : [
        {
            "tId" : "t2014-06-18_115628_S-Gross",
            "tType" : "Thread",
            "tCreator" : "S-Gross",
            "tCreatorRole" : "admin",
            "tPath" : [
                "Thread"
            ],
            "tDate" : "2014-06-18T11:56:28.313Z",
            "tTitle" : "Off-Topic-Ge[Blubb]er",
            "tDescr" : "einfach nur zum blubbern.",
            "tStatus" : "open",
            "tMsgCount" : 1
        },
        ...
    ],
    "sessInfo" : {
        "sessId" : "5a379bd91cc16ca1dc01198f54fbb55d",
        "sessUser" : "Der-Praktikant",
        "sessRole" : "user",
        "sessActive" : true,
        "expires" : "2014-07-31T23:00:25.000+0200"
    }
}
```

Get all threads

Query=tree.functions.getAllThreads(self)
The result is an array of threads.

```
"Result" : [
    {
        "tId" : "t2014-06-18_115628_S-Gross",
        "tType" : "Thread",
        "tCreator" : "S-Gross",
        "tCreatorRole" : "admin",
        "tPath" : [
            "Thread"
        ],
        "tDate" : "2014-06-18T11:56:28.313Z",
        "tTitle" : "Off-Topic-Ge[Blubb]er",
        "tDescr" : "einfach nur zum blubbern.",
        "tStatus" : "open",
        "tMsgCount" : 1
    },
    {
        "tId" : "t2014-06-20_145439_S-Gross",
        "tType" : "Thread",
        "tCreator" : "S-Gross",
        "tCreatorRole" : "admin",
        "tPath" : [
            "Thread"
        ],
        "tStatus" : "solved",
        "tDate" : "2014-06-20T14:54:39.775Z",
        "tTitle" : "Bug: Login-Screen",
        "tDescr" : "...",
        "tMsgCount" : 2
    },
    ...

]
```

Create a thread

tree.functions.createThread(self,"tTitle","tDescription")
The result is the created thread.

```
"Result" :
  {
        "tId" : "t2014-06-18_115628_S-Gross",
        "tType" : "Thread",
        "tCreator" : "S-Gross",
        "tCreatorRole" : "admin",
        "tPath" : [
            "Thread"
        ],
        "tDate" : "2014-06-18T11:56:28.313Z",
        "tTitle" : "Off-Topic-Ge[Blubb]er",
        "tDescr" : "einfach nur zum blubbern.",
        "tStatus" : "open",
        "tMsgCount" : 1
    }
```

Modify a thread

Query=tree.functions.setThread(self,"tId","tTitle","tDescription","tStatus")
The result is the modified thread.

```
"Result" :
  {
        "tId" : "t2014-06-18_115628_S-Gross",
        "tType" : "Thread",
        "tCreator" : "S-Gross",
        "tCreatorRole" : "admin",
        "tPath" : [
            "Thread"
        ],
        "tDate" : "2014-06-18T11:56:28.313Z",
        "tTitle" : "Off-Topic-Ge[Blubb]er",
        "tDescr" : "einfach nur zum blubbern.",
        "tStatus" : "open",
        "tMsgCount" : 1
    }
```

Quick check

Query=tree.functions.quickCheck(self)
The result will be a simple array of integer. The first integer represents the number of threads at the database and the second the number of messages.

```
"Result" : [
    17,
    102
]
```

Get messages for a thread

Query=tree.functions.getMsgsForThread(self,"tId")
The result will be an array of messages.

```
"Result" : [
    {
        "mId" : "m2014-06-18_122643_S-Gross",
        "mType" : "Message",
        "mCreator" : "S-Gross",
        "mCreatorRole" : "admin",
        "mDate" : "2014-06-18T12:26:43.422Z",
        "mThread" : [
            "t2014-06-18_115628_S-Gross"
        ],
        "mTitle" : " ",
        "mContent" : "Bitte Alle ToDo-Threads mit ..."
    },
    ...
]
```

Create a message

Query=tree.functions.createMsg(self,"tId","mTitle","mContent","mLink")
The result is the created message.

```
"Result" :
    {
        "mId" : "m2014-06-18_122643_S-Gross",
        "mType" : "Message",
        "mCreator" : "S-Gross",
        "mCreatorRole" : "admin",
```

```
        "mDate" : "2014-06-18T12:26:43.422Z",
        "mThread" : [
            "t2014-06-18_115628_S-Gross"
        ],
        "mTitle" : " ",
        "mContent" : "Bitte Alle ToDo-Threads mit ..."
    }
```

Modify a message

Query=tree.functions.setMsg(self,"mId","mTitle","mContent","mLink")
The result is the modified message.

```
"Result" :
    {
        "mId" : "m2014-06-18_122643_S-Gross",
        "mType" : "Message",
        "mCreator" : "S-Gross",
        "mCreatorRole" : "admin",
        "mDate" : "2014-06-18T12:26:43.422Z",
        "mThread" : [
            "t2014-06-18_115628_S-Gross"
        ],
        "mTitle" : " ",
        "mContent" : "Bitte Alle ToDo-Threads mit ..."
    }
```

Bibliography

AndroidDeveloper. (2014a). Activities. Retrieved August 2, 2014, from http://developer.android.com/guide/components/activities.html

AndroidDeveloper. (2014b). Activity lifecycle. Retrieved August 7, 2014, from http://developer.android.com/images/activity_lifecycle.png

AndroidDeveloper. (2014c). Android ndk. Retrieved August 14, 2014, from http://developer.android.com/tools/sdk/ndk/index.html

AndroidDeveloper. (2014d). App manifest. Retrieved August 7, 2014, from http://developer.android.com/guide/topics/manifest/manifest-intro.html

AndroidDeveloper. (2014e). Asynctask. Retrieved August 7, 2014, from https://developer.android.com/reference/android/os/AsyncTask.html

AndroidDeveloper. (2014f). Listview. Retrieved August 7, 2014, from http://developer.android.com/guide/topics/ui/menus.html

AndroidDeveloper. (2014g). Listview. Retrieved August 2, 2014, from http://developer.android.com/guide/topics/ui/layout/listview.html

AndroidDeveloper. (2014h). Notifications. Retrieved August 7, 2014, from http://developer.android.com/guide/topics/ui/notifiers/notifications.html

AndroidDeveloper. (2014i). Processes and threads. Retrieved August 7, 2014, from http://developer.android.com/guide/components/processes-and-threads.html

AndroidDeveloper. (2014j). Sqlhelper. Retrieved July 30, 2014, from http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

AndroidDeveloper. (2014k). Storage options. Retrieved July 30, 2014, from http://developer.android.com/guide/topics/data/data-storage.html

AndroidDeveloper. (2014l). System permissions. Retrieved August 7, 2014, from http://developer.android.com/guide/topics/security/permissions.html

AndroidDeveloper. (2014m). Ui overview. Retrieved August 7, 2014, from http://developer.android.com/guide/topics/ui/overview.html

Cooper, J. W. (2000). *Java design patterns: a tutorial*. Addison-Wesley Professional.

Corporation', '. D. (2014). Smartphone os market share, q1 2014. Retrieved August 14, 2014, from http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Gandhewar, N. & Sheikh, R. (2010). Google android: an emerging software platform for mobile devices. *International Journal on Computer Science and Engineering*, *1*(1), 12–17.

Gross, S. (2014a, September). private interview.

Gross, S. (2014b). Beap. Retrieved August 14, 2014, from http://www.beap-code.de/index.html

Matthews, R. (2011). Android-scripting. Retrieved August 14, 2014, from https://code.google.com/p/android-scripting/

Eidesstattliche Versicherung:

Hiermit erkläre ich eidesstattlich,

dass die vorliegende Arbeit von mir selbständig und ohne unerlaubte Hilfe angefertigt worden ist und dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind und alle Informationen, die aus interviewähnlichen Gesprächen gewonnen wurden, als Zitate gekennzeichnet habe.

Ort, Datum _____ Unterschrift _____