DOI: 10.2352/ISSN.2470-1173.2016.7MOBMU-303

MultiMo: A Multimodal University Evaluation Software Designed for High Response Rates and Usability

Sebastian Müller, Max Gregor, Raoul van Rüschen, Rico Wildenhein, Reiner Creutzburg, Martin Christof Kindsmüller
Brandenburg University of Applied Sciences, Department of Informatics and Media, P. O. Box 2132, D-14737 Brandenburg, Germany

Abstract

This paper presents a software solution of the evaluation problem faced by universities. The software consists of a REST - server application and to date an Android client. The software features automated processing of given answers, independency of location, a flexible data model supporting multiple questionnaires and an multimodal interface designed to be used effectively upon first encounter. While our approach aims to support a high participation rate of the evaluations it is currently tested in a live environment to see if it reaches this goal. Findings of these tests will be presented at the conference.

Motivation

In Germany universities are bound by law to evaluate their courses at the end of each semester. The evaluation is based of the opinions of participating students. At our university two departments (Engineering and Economics) had introduced web based solutions, but that had let to a decrease in participation in comparison to the former paper based solution. That is the reason our department (Computer Science and Media) still relies on the paper based approach which is conducted within one of the last lectures of a course and analyzed manually afterwards. In this contribution we present a software solution that aims to automate the analyzis process while maintaining a high participation rate.

Objective

The objective of this project is to create a system for evaluating university teaching that does not fall behind a paper based solution but instead makes the process more efficient. Therefore it has to meet six main prerequisites:

1. The anonymity of the participants must be guaranteed.
2. The risk of compromising data or information theft through an attacker has to be minimal.
3. The software must be designed to be easily usable by students as well as tutors.
4. The speed with which the students can vote must be as fast as the paper based approach allows or faster.
5. The quality and the quantity of the answers must be as high as the paper based approach or higher.
6. The analysis of an evaluation must be fully automated.

To meet the third requirement it is not only necessary to craft an accessible design but also to support multiple kinds of clients for platforms that are used by today's students.

Workflow Description

We developed a prototype of two clients for Android and windows phone and a backend server software. The Android client has reached production stage and will be further discussed within this contribution. Additionally, we have developed a web interface for the tutor(s) of a course which can be used to manage evaluations. Furthermore we developed strategies to ensure anonymity and security of the data and implemented them into the prototypes.

Our goal was to design the clients in such a way that the workflow resembles the paper based approach. We guaranteed that it is possible to combine the old paper based questionnaire with our new automated approach. So that students without a supported smart phone or unwilling to use the software solution can participate the old way. The workflow of the different clients differs only regarding system specific design guidelines. Thus the following workflow description applies to both clients.

First the tutor opens the aforementioned web interface in his browser and uses his university account information to log in and verify his identity. He or she then has the option to create a new evaluation. In order to do so the tutor first has to enter some basic information. The most important one is the number of participating students since it determines how many tickets are created in the following step. Figure 4 shows the frontend page to create a new evaluation in its mobile version. With these information the backend creates a PDF file which contains all tickets for this evaluation. The tickets are stored within QR codes. The QR code contains the address of the backend-server as well. This is to ensure that the clients are independent of the server location. Therefore each department can host their own server, so that sensitive evaluation data stays within the the boundaries of the infrastructure of the department. The tutor can now hand out these QR codes to the students. The students use the client app applicable for their smart phone architecture to scan the QR code (as seen in figure 2). The app automatically tries to connect to the backend server. It sends the ticket and a unique session ID generated beforehand. This ID is generated after the app was started and is discarded upon finishing the evaluation, scanning in another QR code or closing the app. Upon retrieving the ticket and the unique ID the server stores them together in the database. Is the same ticket send with another unique ID the server knows that another user tries to use the same QR code and stops him or her from using this particular code. As soon as the student sends his evaluation information back to the server the unique ID is erased from the database. Upon closing the evaluation the server automatically deletes all remaining entries.

If the ticket sent to the server is valid, the server sends all information needed for this evaluation, including all questions and answers, to the client. Upon retrieving all information the student is asked to determine his study path. In the next step the student enters the voting phase in which one is able to answer all questions previously send. These questions are displayed in a

DOI: 10.2352/ISSN.2470-1173.2016.7MOBMU-303

predefined order. The students can go through the questionnaire following this predefined order or use the menu to jump to any question they want. Upon sending the vote the server once more verifies the validity of the ticket and, if the ticket was valid, stores the answers in the database. At this point the ticket can only be invalid if the evaluation was closed before the ticket was send back to the server. The exchange of the unique ID at the beginning ensures that a scenario in which a student must repeat answering the questionnaire because his or her QR code was already used cannot occur. After sending the data back to the server the app deletes said data including taken images and closes itself. The web interface on the backend server shows the number of completed questionnaires (see figure 3). This enables the tutor to end the evaluation as soon as all students have voted.

After all students have send their votes, the tutor can close the evaluation within the web frontend. The server then generates an excel file containing all answers of every student. If there were students not using our software based solution the tutor can man-

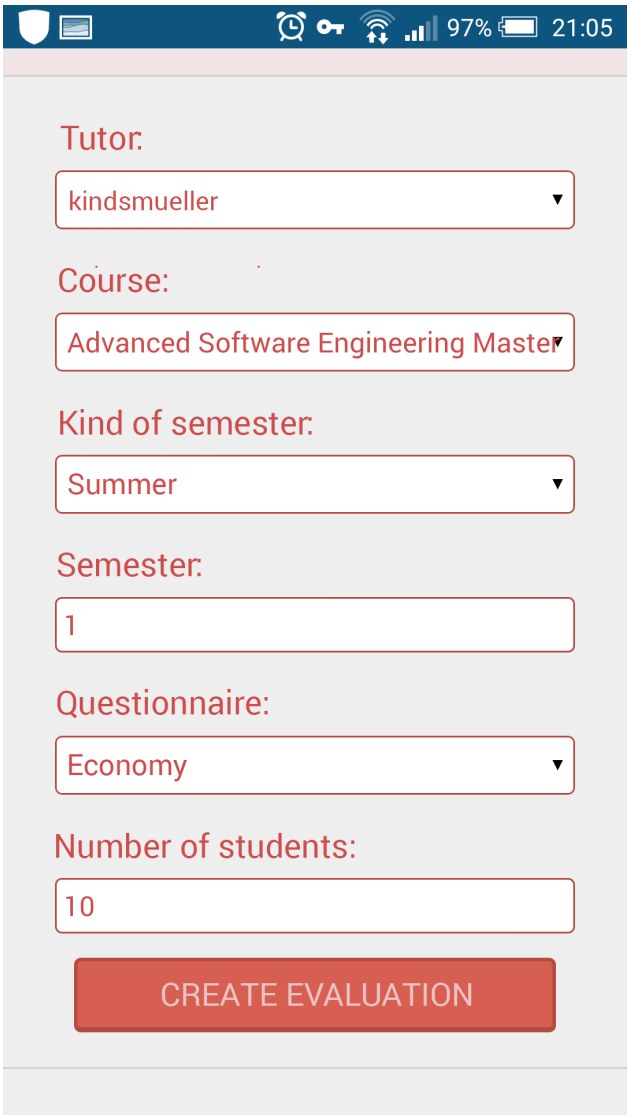

Figure 1. *Frontend interface for creating a new evaluation*

Figure 2. *First screen of the app: Scan a QR code ticket to proceed*

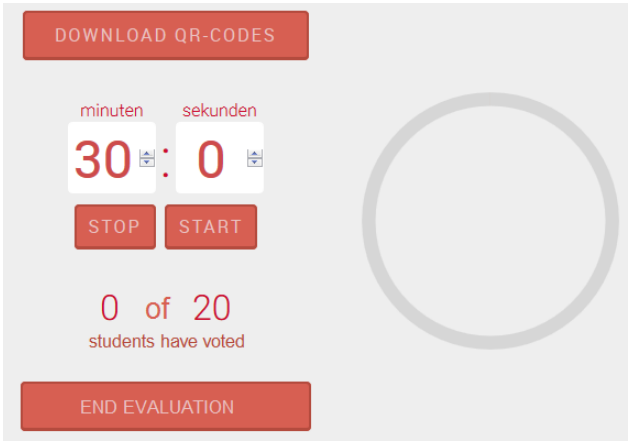

Figure 3. *Frontend in mid-evaluation*

DOI: 10.2352/ISSN.2470-1173.2016.7MOBMU-303

ually add their information to the excel file (like it was done in the paper evaluation in the last 10 years). Note that the student is at no point forced to enter login data or other information that could lead to the discovery of their identity. At no point does the client send user or smart phone specific data. Therefore there is no data processed or stored in the system, that allows for revealing the identity of the student.

Design Description

The main goal when designing the app was to ensure easy accessibility. Under these circumstances the app had to be designed in a way that users can immediately start to work with it and do not need to think about the purpose of some interface elements. In fact the app should be as easy to use as a paper questionnaire, i.e. without consulting a manual or training (walk up and use situation [18]). Therefore we refrained from using exotic or custom interface elements and instead only used elements that we expected the user to be accustomed to through the usage of other apps on their smart phones.

Figure 4 shows the Android app in a mid-evaluation state. Currently one of the three supported question types is displayed. In the next section the elements of the interface of a generic single choice question are explained. The standard way of navigating to the next or to the previous question is by applying horizontal swipe gestures. The user can discover this by a self revealing gesture [3]. Touching the screen reveals the possibility to swipe. The affordance [2] ”swipe me” is additionally supported by showing the other questions left and right as signifiers. The *toolbar* at the top of the screen shows a ”burger menu” which on touch opens up a navigation list to jump directly to other questions. Below the *toolbar* a *pager tab strip* is used to show the number of the currently displayed question and the number of questions overall. Below that the text of the question is placed followed by a number of buttons representing the possible options for this question. The app supports questions from two to seven options whereas ”not applicable” is an option always included and placed vertically at the right edge of the screen. If a button is pressed the app automatically navigates to the next question visually marking the chosen answer in the process so that when a user returns to this question he or she sees which answer was chosen before. As seen in figure 5 the app also supports questions with bipolar answering schemes.

We use a traffic light inspired scheme to colour the buttons to enable the user to better and faster judge in which directions the Likert Scales [4] are poled.

Besides single choice questions our app also supports open ended questions. Figure 7 shows the interface for this kind of question. Below the question a text input box was added to support long texts with multiple lines. As we strived to make our app as easily accessible as possible we encountered the problem that typing answers can be perceived as cumbersome. While it is possible with our app to use Google Nows speech recognition to transform speech into text it compromises the privacy of the user if someone were to eavesdrop their spoken answer. As a solution to this problem we implemented a camera functionality to give the users the possibility to write down their answer and take a photo of it instead.

As can be seen in figure 7 a taken picture is placed below the edit text field. A press upon it enlarges the picture so that the student is able to confirm if the text is readable (figure 6). Upon entering such a question a camera symbol appears in the toolbar. Touching it opens the native Android camera app. The third supported question type is a numerical text question. Internally it is a text question with constraints. Namely that only numbers can be typed into the edit text field. For this kind of question the camera functionality gets deactivated.

Implementation

The core of our solution is a *REST*[1] based backend server application with a *SQL* database. It uses *Spring Boot*[15] as a framework and multiple libraries such as *Lombok*[8] and *Joda Time*[12] to ease the development process. The following section describes which libraries are used and what features were realised with them. *ZXing*[9] is used to generate QR codes which are then stored in a PDF document created with Apache *PDFBox*[10]. *Thymeleave*[11] is used to design the frontend of the

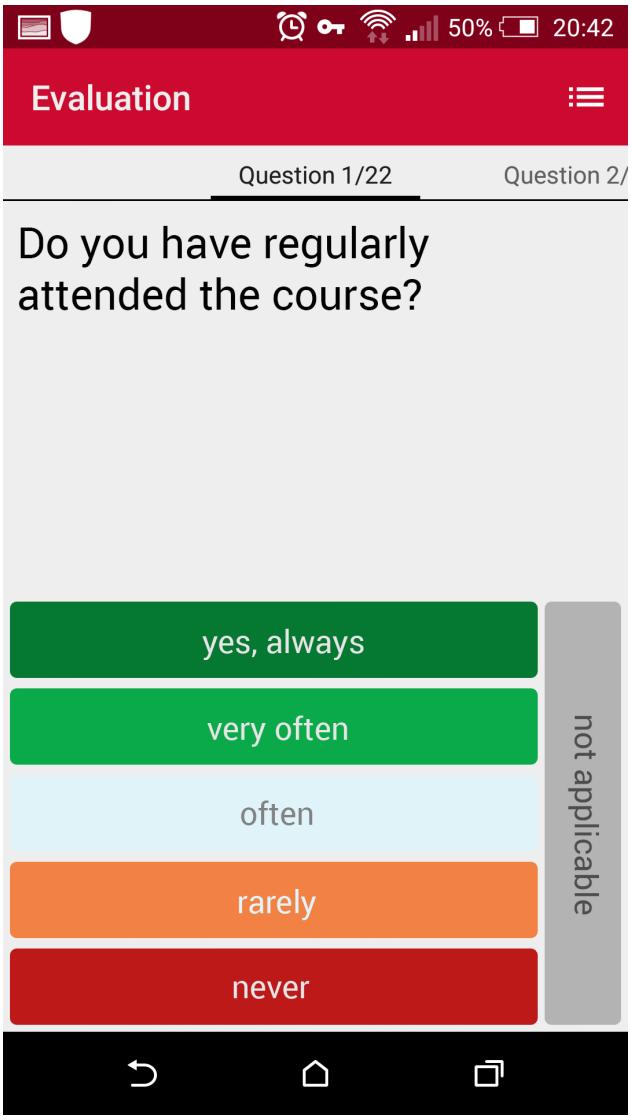

Figure 4. *Android app in mid-evaluation. Showing interface with Likert Scale rating question*

DOI: 10.2352/ISSN.2470-1173.2016.7MOBMU-303

server side and *Apache POI*[17] is used to create the Excel document after an evaluation has ended.

Currently the Android client implementation has reached production stage. The following section describes how the features mentioned in the chapter Design Description are implemented. The Android client uses multiple libraries to enhance the development process of certain features. The network communication with the backend server is realised with *Retrofit*[14]. *ZXing* is used on the client side as well, here to read the generated QR codes. *Jackson 2*[13] is used to transform Java objects into instances of *String* which are representing the encoded objects as JSON thus making it possible to easily transmit those objects over the network.

One of the more complex problems to solve was the implementation of image processing that does not block the main thread, does not clutter the storage of the smart phone with unneeded images or causing an out of memory exception effectively crashing the app. For this to understand it is necessary to know that Android uses the bitmap format to store images in RAM. If the user takes a photo with the built-in camera and an app tries to display it as it is it can cause an out of memory exception since the camera takes images in an resolution that is most of the times far greater than the RAM allocated for one app can handle. To reduce the size of the image in memory it must be scaled down.

In the app there are three variants needed for each picture taken. The first variant, visible in figure 7, is a thumbnail displayed directly below the answer. The second variant is needed when touching the thumbnail. As already mentioned in Design Description an enlarged version will be displayed. This image is scaled so it fits the whole screen. The last variant is the image that is send over the network. Besides being rescaled it is also transformed into a *JPEG* image to reduce network load. The creation of all three image variants must be done on a different thread than the main thread. The CPU heavy operations would otherwise render the user interface unresponsive. To solve the issue of transforming the image into a *JPEG* file in a background thread

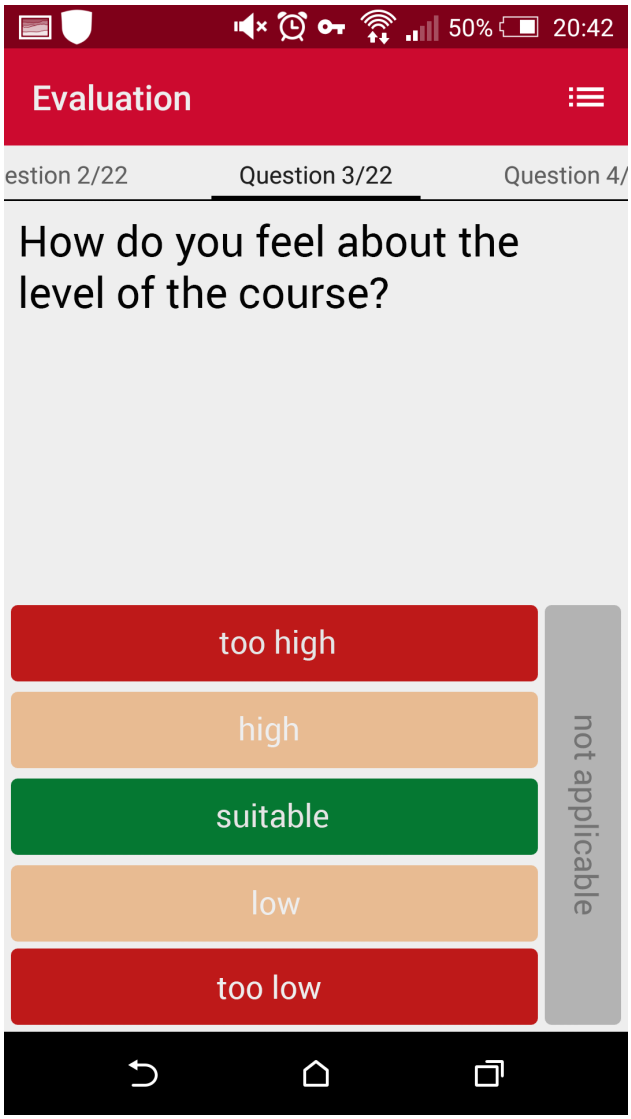

Figure 5. *The most positive answer is placed in the middle of all options*

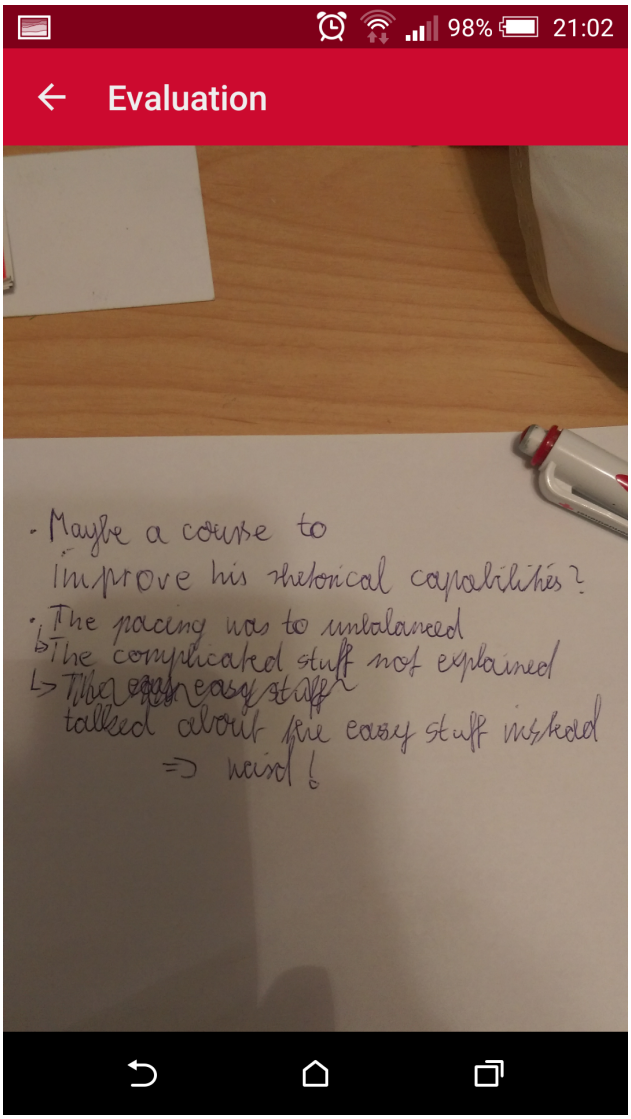

Figure 6. *Figure shows the app after a touch on a thumbnail*

DOI: 10.2352/ISSN.2470-1173.2016.7MOBMU-303

the library *RXJava* [6] is used in conjunction with *RXAndroid*[7]. Those libraries introduce a multithreadable observer pattern into the app environment. The library *Picasso* [5] is used to display the thumbnail and the enlarged version. This library manages caching and prevents the aforementioned out of memory exception. Lastly to prevent cluttering of the storage an Android Service is used. Such a service runs in a background thread and can, if correctly configured, run independently from the app it was started from. The service implemented in our project has no other function than to listen when the app is closed. When Android notifies this listener the service loops through all stored images related to our app and deletes them sequentially. Afterwards it stops itself. This approach is needed since Android itself defines when an app is closed and removed from memory. The development tools used for the different clients and for the backend are Visual Studio for the Windows Phone app, Android Studio for the Android app and IntelliJ for the development of the backend software.

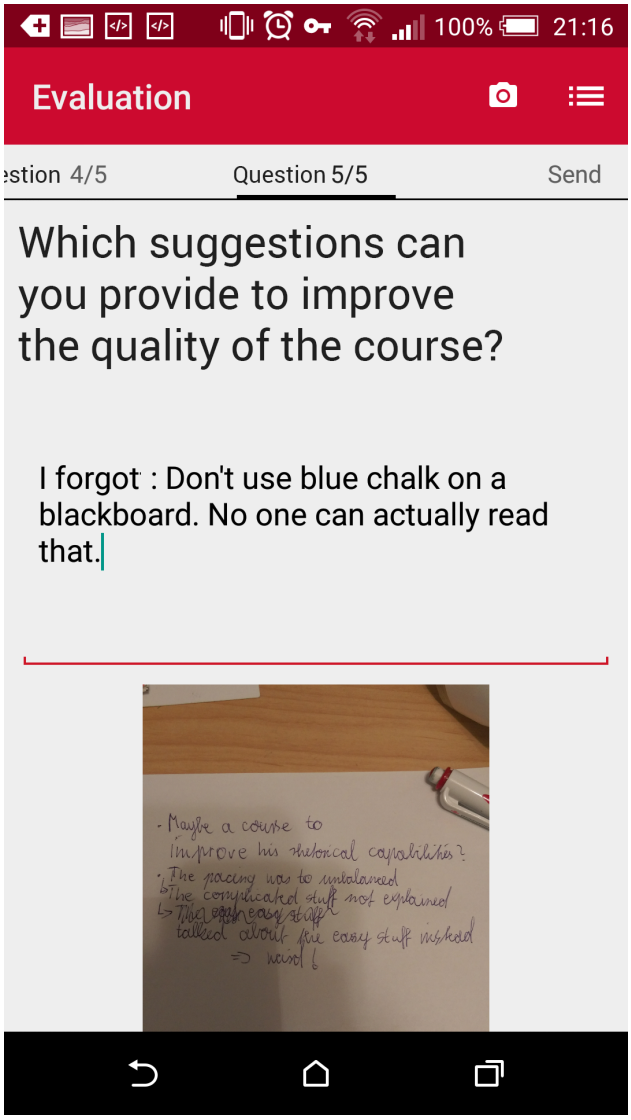

Figure 7. *Interface for text based question. Photo thumbnail displayed below text field.*

Preliminary Result

Up until now three courses were evaluated using the Android app. In total 61 students evaluated their course with the app successfully. Some students of these courses had to use the paper version due to a lack of an Android smart phone or were unwilling to use the new app. 6 students tried to use the client but were unable to due to an exception that ended the app. 3 students were unable to install the app probably due to the fact that their smart phones used a different language than is currently supported by the app. This needs further investigation. The following section describes the findings of the first evaluated course. The data of the other two is currently in the process of further analysis. The course had 16 attendants. Of those 16 all used the MultiMo app. Of those 16 students one was not able to install the app but was able to vote with another smart phone. One student was not able to vote the first time because the app crashed before he could send the data. The high participation was supported by 5 pre-configured smart phones that were lent to the students without a suitable Android smart phone. It was also observed that students shared their smart phones once they were done. Other observations were:

1. Text answers were slightly shorter compared to the paper versions over the last years.
2. Nobody used Google Now (speech recognition).
3. Nobody took a picture.
4. Nobody used the paper version instead.
5. Execution time was 20% increased compared to the paper version. This was due to the time needed downloading the app and due to the lack of experience the users had with the new software. The core of the evaluation was faster than with the paper version. We expect that in the long run the time needed will be equal for both approaches.
6. The tutor rated the app as innovative and work load reducing. He was pleased when he learned that no manual creation of excel files was needed anymore.

In addition to these observations a heuristic evaluation [19] with 16 usablity experts revealed some further problems with the beta version: Many students complained that it is not possible to delete a picture in the app. Currently a picture can only be replaced when another one is shot. That means once the picture feature is used for one question it is not possible to not send a picture to the server for this particular question. Another requested feature was that the enter button of the keyboard, displayed when answering text answers, leads to the next question as soon as touched. As a button touch also leads to the next question this would improve the behavioral consistency of the app. On certain smart phones the pictures taken were not displayed. That is most certainly related to a bug occurring in specific Android versions provided by specific vendors. Those versions do not properly store the taken image in the storage of the smart phone. As it is not guaranteed that these vendors fix these bugs in the future a generic workaround for this issue must be found. A severe bug causing the app to crash occurred seldomly in all three courses. This bug is caused when the app is pushed into the background by the user. Apps waiting in the background are exposed to Androids self-optimization mechanics. If Android decides that the RAM allocated for a background app is needed for a different process this allocated RAM is cleared and reallocated once the user decides to return to the app. For MultiMo this behaviour has

DOI: 10.2352/ISSN.2470-1173.2016.7MOBMU-303

fatal consequences since all questions and answer as well as the ticket and the generated unique ID are stored in RAM. To circumvent this issue a library was implemented prior to the tests in the courses saving the data to the smart phone storage every time it is changed. Obviously there are still some flaws that need to be fixed.

Outlook

MultiMo is currently under revision to get the stability and usability issues fixed. The next step will be a more comprehensive field test. At this point we have some preliminary evidence that our approach is able to maintain the high response rate of the paper based solution. But further investigation is necessary before we can make this claim. It is counterproductive to high participation if some students cannot use the software solution due to a scarcity of supported devices. Therefore the main goal for the future must be to create clients for all smart phone architectures that are used by todays students. Namely the Windows Phone app must be developed further until it reaches the level of the Android app. It is also planned to develop an iOS app and a web-based app following the Responsive Web Design principle introduced by Marcott [16]. Our app currently only supports three question types. It is also planned to further the functionality of the frontend. Currently all questionnaires need to be hard coded into a specific backend class. It would be better if the frontend supported a way to add a new questionnaire without the need to shut the server down and recompile the backend. Our first ideas in this direction are to allow either a file upload or to create a form in whose fields all information must be entered. Implementing the file upload solution would need an implementation of a parser on the backend side which processes the uploaded files. In order to enable our parser to add the questionnaire supposedly contained in the uploaded file to the database the file has to be formatted in a specific way. Another feature that will be implemented in the coming versions are ad hoc questions which can be added by any tutor to the predefined questionnaire in order to get information about course specific aspects. The security specialists of our university have volunteered to put the security measurements we have taken so far under stress, trying to find ways to improve those measurements in future releases.

References

[1] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. (Diss., University of California, Irvine, CA).

[2] Norman, D. A. (2013). The Design of Everyday Things. New York, NY: Basic Books.

[3] Wigdor & Wixon (2011). Brave NUI World: Designing Natural User Interfaces for Touch and Gesture. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[4] Likert, Rensis (1932). "A Technique for the Measurement of Attitudes". Archives of Psychology 140: 1–55.

[5] Square (2013): A powerful image downloading and caching library for Android. Available online: http://square.github.io/picasso/ Last visited: 2016-01-08

[6] ReactiveX (2012): Reactive Extensions for the JVM. Available online: https://github.com/ReactiveX/RxJava Last visited: 2016-01-08

[7] ReactiveX (2013): Reactive Extensions for Android. Available online: https://github.com/ReactiveX/RxAndroid Last visited: 2016-01-08

[8] Reinier Zwitserloot (2009): Very spicy additions to the Java programming language. Available online: https://github.com/rzwitserloot/lombok Last visited: 2016-01-08

[9] zxing (2013): Official ZXing ("Zebra Crossing") project home. Available online: https://github.com/zxing/zxing Last visited: 2016-01-08

[10] Apache (2002): Apache PDFBox - A Java PDF Library. Available online: https://pdfbox.apache.org/ Last visited: 2016-01-08

[11] THE THYMELEAF Team: thymeleaf. Available online: http://www.thymeleaf.org/index.html Last visited: 2016-01-08

[12] JodaOrg (2002): Joda-Time is the widely used replacement for the Java date and time classes. Available online: https://github.com/JodaOrg/joda-time Last visited: 2016-01-08

[13] FasterXML (2009): Jackson Release: 2.0 Available online: http://wiki.fasterxml.com/JacksonRelease20 Last visited: 2016-01-08

[14] Square (2010): A type-safe HTTP client for Android and Java Available online: http://square.github.io/retrofit/ Last visited: 2016-01-08

[15] Pivotal Software (2012): Spring Boot. Available online: https://github.com/spring-projects/spring-boot Last visited: 2016-01-08

[16] Ethan Marcotte, Responsive Web Design, A Book Apart, New York, NY, 2011

[17] Apache (2002): Apache POI - the Java API for Microsoft Documents. Available online: https://poi.apache.org/ Last visited: 2016-14-01

[18] DIN ISO 20282-1:2008-10 (E), Ease of operation of everyday products - Part 1: Design requirements for context of use and user characteristics (ISO 20282-1:2006). Geneva, Switzerland: International Organization for Standardization

[19] Nielsen, J. (1994). Heuristic evaluation. In J. Nielsen & R.L. Mack (Eds.), Usability Inspection Methods. New York, NY: John Wiley & Sons.

Author Biography

Sebastian Müller, Max Gregor, Raoul van Rüschen and Rico Wildenhein received their Bachelor of Science in 2014 and are now in the process of acquiring their respective Master of Science at Brandenburg University of Applied Sciences.

Martin Christof Kindsmüller studied computer science, psychology and media arts and sciences at Karlsruhe Institute of Technology (KIT), Germany, Berlin University of Technology (TUB), Germany and Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. He holds Master degrees in computer science and psychology from TUB and a PhD in natural sciences from Humboldt University Berlin. He is now professor for Human Computer Interaction and Mobile Computing at Brandenburg University of Applied Sciences (THB).

Reiner Creutzburg studied Math and Physics at Rostock University and took his Ph.D. in Math in 1985. His research interests include, digital signal and image processing, image compression, computer security and forensics, Internet of things and mobile systems security. He is a Professor of Applied Computer Science since 1992 at Brandenburg University of Applied Sciences in Brandenburg, Germany.