Heinrich-Heine-Universität Düsseldorf  
Fachbereich „Angewandte Informationswissenschaft“  
Projektseminar: Angewandte Informationswissenschaft  
Leitung: Maria Henkel  
Sommersemester 2016

**Bromi Dokumentation**

23.09.2016

Autor: Marco Kluin (4. Semester)  
Bachelorstudiengang Informationswissenschaft und Sprachtechnologie  
Matrikelnummer: 2181477  
E-Mail: [Marco.Kluin@hhu.de](mailto:Marco.Kluin@hhu.de)  
Github: xayfuu

**Inhaltsverzeichnis**

1. Einleitung . . . . . . . 4

2. Prämissen und Vorarbeiten . . . . 5

3. App-Aufbau und Erläuterung . . . . 6

3.1 Package: Activities . . . . . 6

3.1.1 MainActivity.java . . . . 8

3.1.2 StartScreenActivity.java . . . 8

3.1.3 CreateProfileInfoActivity.java . . 9

3.1.4 CreateProfile.java . . . . 10

3.1.5 LogInActivity.java . . . . 13

3.1.6 MainMenuActivity.java . . . 14

3.1.7 LanguageSelectActivity.java . . 15

3.1.8 PracticeLevelSelectActivity.java . . 16

3.1.9 PracticeLevelActivity.java . . . 17

3.1.10 PracticeLevelResultActivity.java . 20

3.1.11 UserProfileActivity.java. . . . 22

3.2 Package: Custom . . . . . 23

3.2.1 ExperienceBar.java . . . . 23

3.3 Package: db . . . . . . 24

3.3.1 LanguageLevelData.java . . . 24

3.3.2 LanguageLevelDbHelper.java . . 25

3.4 Package: util . . . . . . 25

3.4.1 constants.java . . . . 25

3.4.2 methods.java . . . . 25

4. Fazit . . . . . . . . 26

5. Literaturverzeichnis . . . . . . 27

6. Anhang . . . . . . . 27

**1 Einleitung**

Im Rahmen des Projektseminars I4: „Angewandte Informationswissenschaft“ sollte ein eigenständiges Projekt von der Idee, über die Planung, bis zur Ausführung, erarbeitet werden. Hierbei sollten eigene Interessen in das Projekt einfließen, mit dem Sinn, dem Studenten eine Herausforderung zu geben, welche es in dieser Form noch nicht im Studiengang gab. Trotzdem musste die Relevanz zu den behandelten Themen im Studiengang gegeben sein.

Mein Projekt war es, eine Android App zu programmieren, welches im Kern zum Vokabeltraining für verschiedene Sprachen dienen sollte. Vereint mit Spielelementen soll die Motivation und das Lernergebnis des Nutzers gefördert werden (*Gamification*). Die App trägt den Namen *Bromi*. Zu dieser Idee kam ich durch die Hilfe einer der hilfsbereiten Tutoren im Projektseminar. Der Einfluss kam danach durch die bereits herunterladbare App *Memrise*, welches das Konzept einer gamifizierten Sprach- und Vokabel-Lern-App für Mobilgeräte bereits seit Jahren erfolgreich durchführt. Ich selbst habe diese App bereits verwendet, u.a. für meinen Chinesisch-Kurs an der HHU. Neben der Verwirklichung dieser App war das Hauptziel und damit die Herausforderung dieses Projekts das Erlernen der Android-App-Programmierung, um eine zusätzliche Qualifikation für einen möglichen Beruf zu erlernen.

Im Folgenden wird der Werdegang der App vorgestellt. Zuerst wird auf Prämissen eingegangen, die vor dem eigentlichen Programmieren der App vorhanden sein mussten. Hierzu gehören zum Beispiel Tools und das Basiswissen über die Android-Programmierung. Im Anschluss wird die App im Detail vorgestellt, so wie es eine Entwicklerdokumentation vorsieht. Um zu zeigen, wie sich die App entwickelt hat, werden die Fortschritte und Probleme in der Reihenfolge vorgestellt, wie sie in der Entwicklung vorgekommen sind.

**2 Prämissen und Vorarbeiten**

Nach der kurzen Woche, in welcher wir Zeit hatten, uns für ein Projekt zu entscheiden, war es notwendig, einen Lernplan zu erstellen. Android kannte ich lediglich durch das Verwenden meines eigenen Smartphones, also musste ich einen Weg finden, dies zu ändern. Dementsprechend wurde mit ein paar Suchen im Internet nach Tutorials gesucht, mit welchen man sich kurzerhand die absoluten Kernelemente der Android-Programmierung aneignen kann. Glücklicherweise hat Google, der Entwickler von Android, eine Kollektiv-Webseite zur Android-App-Programmierung erstellt. Dort gibt es Tutorials über alle Grundkonzepte mit Programmbeispielen, sowie die gesamte Dokumentation über die Bibliotheken, die für die App-Entwicklung zur Verfügung gestellt wurden. Die ersten 4 bis 5 Tage habe ich somit nur auf dieser Webseite verbracht und mir alle Punkte angeschaut, die zur Entwicklung von *Bromi* wichtig werden könnten. Zwischenzeitlich habe ich auch nach weiteren Tutorials und Programmbeispielen gesucht. Da zur Programmierung von Apps Java verwendet wird, musste ich mir zusätzlich einige Java Code-Snippets ansehen, da es auch einige Zeit her war, dass ich mit Java gearbeitet habe. Hierzu habe ich das Projekt meiner Gruppe aus dem Programmierpraktikum und meine Notizen aus der Vorlesung *Informatik 1* hinzugezogen. Als Letztes galt es, mich wieder mit *Github* vertraut zu machen. Durch die Präsentation von Git und dem kurzem Training auf *try.github.io* im Seminar wurde dies einfacher.

Als ich mich bereit fühlte, mit dem Programmieren anzufangen, habe ich begonnen, meine Entwicklungsumgebung vorzubereiten. Zunächst musste ich Java (JDK und JRE 8) installieren und auf Windows zum Laufen bringen. Danach habe ich mir *Android Studio* heruntergeladen, welches von Google gratis eigens für die Entwicklung von Apps zur Verfügung gestellt wird. Es basiert auf *IntelliJ*, wodurch ich mich sehr schnell zurechtfand, da ich dieses IDE bereits im Programmierpraktikum verwendet habe. Als Zusatz sind viele Funktionen hinzugefügt worden, die für die App-Entwicklung sehr nützlich sind. So gibt es, neben der kompletten Android-Library, Virtual Machines zur Emulation von Apps auf einem der vielen Android OS Versionen, die es inzwischen gibt. Außerdem mitenthalten, ist ein eingebauter Layout-Editor, was die Entwicklung eines UI’s einfacher macht. Ich konnte nun mit der Verwirklichung des Projekts beginnen.

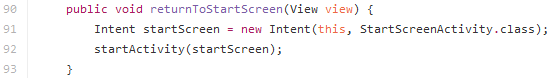
**3 App-Aufbau und Erläuterung**

Es folgt nun die Vorstellung des Aufbaus der gesamten App. Hierfür wird jede einzelne erstellte Java-Klasse präsentiert und erläutert. Es werden ebenfalls Probleme angesprochen, welche beim Programmieren auftraten, sowie Abweichungen vom anfänglichen Projektplan versucht zu begründen. Hierbei muss ich persönlich um Verständnis bitten, dass ich durch die Größe des Projekts nicht jede einzelne Methode und Funktion einer Klasse erklären kann, da die Dokumentation sonst weit über dem gewollten Limit von ca. 3500 Wörtern hinausgeht. Dementsprechend werden nur die wichtigsten Kernmethoden und deren Funktion im Zusammenhang mit dem Gesamtzweck einer Java-Klasse angesprochen.

**3.1 Package: Activities**

Jede Android-App besteht aus sogenannten *Activities*, welche quasi aktiven Fenstern auf dem Computer entspricht. Jede Java-Klasse, die eine Activity repräsentiert, muss per Vererbung die Grundeigenschaften einer Android-Activity besitzen. Dazu gehört mit großer Wichtigkeit der „Lebenszyklus“ einer Activity (Google and Open Handset Alliance n.d. Android API Guide, 2016). onCreate() kreiert die Activity beim Aufruf und muss bei jeder Activity-Klasse mitenthalten sein. Nun gibt es noch weitere Methoden, wie onStart(), onResume(), onPause(), onRestart(), onStop() und onDestroy(), welche auch vererbt werden, aber nicht implementiert werden müssen, da diese von Android selbst zu einem großen Teil gemanagt werden. Der nächste wichtige Punkt ist es, die Activity zu zeigen. Mit setContentView() wird aus dem Android-Ressourcen-Verzeichnis res/layout/… das Layout zur Activity geladen und gezeigt. Jede Activity besitzt ein Layout mit Buttons, Grafiken, usw. und werden zur Interaktion mit dem Nutzer verwendet. Layouts werden in Android mit XML erstellt und vom OS auf dem Smartphone fast wie Magie zu eleganten UI’s zusammengestrickt. In diesem Projekt wurden die XML Layout-Dateien automatisch von Android Studio nach ihren zugehörigen Java-Klassen benannt, wie zum Beispiel MainActivity.java + activity\_main.xml.

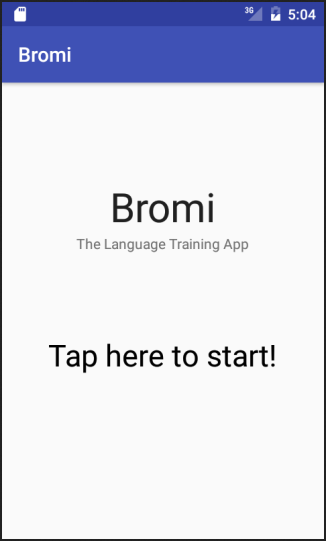
Um von einer Activity zur nächsten zu gelangen werden sogenannte Intents verwendet. Ein Intent ist in Android „eine abstrakte Beschreibung einer Operation, die ausgeführt werden soll“ (Google and Open Handset Alliance n. d. Android API Guide, 2016) und kann dazu verwendet werden, neue Activities zu starten. Im Grundlegenden sieht dies so aus (entnommen aus MainMenuActivity.java):



Activities können alle nach eigenem Willen gestaltet und verwendet werden, wie man in den folgenden Punkten sehen wird, aber es muss immer eine sogenannte MainActivity.java geben. Das ist die erste Activity, die vom OS aufgerufen wird. Sie agiert, meiner Ansicht nach, wie eine Landing-Page zu einer Webseite. Genauso muss es ein sogenanntes Android-Manifest geben (AndroidManifest.xml), in welchem das System, neben weiteren Dingen, gesagt bekommt, welche Activities es überhaupt gibt.

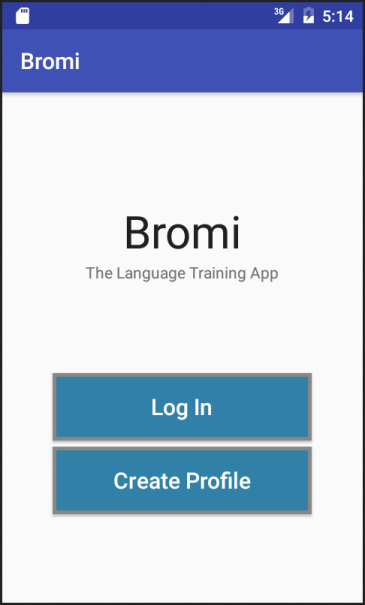
Im Folgenden werden alle Activity-Klassen vorgestellt und beschrieben, was sie von der Logik her machen. Die Layouts werden nicht genauer erläutert, weil diese durch kurzes Reinschauen in die jeweilige XML Datei einer Activity selbsterklärend sind. Jedoch werde ich, genau wie im Projektplan, Screenshots vom Aussehen der Activity zeigen und ein paar Worte über die Verwirklichung des Layouts sagen.

**3.1.1 MainActivity.java**



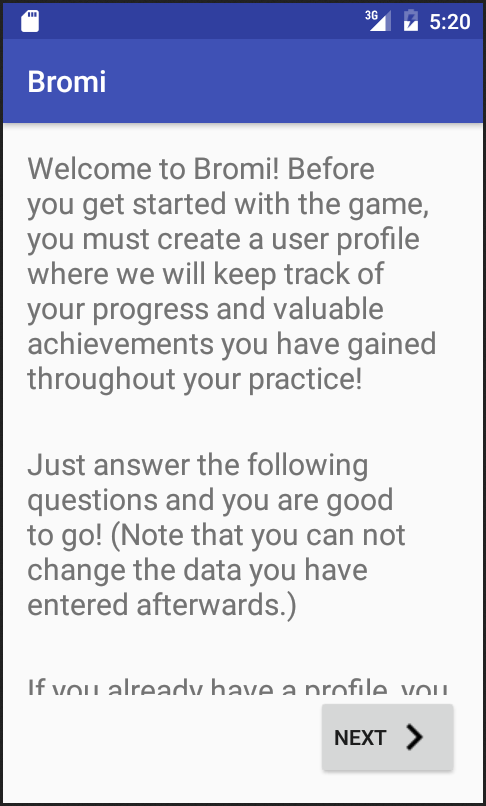
Hier zu sehen ist die erste Activity, die beim Öffnen der App angezeigt wird. Man sieht den Namen der App, sowie einen klassischen „Startbutton“. In der Klasse selbst passiert nichts Erwähnenswertes, außer das beim Tappen auf den Text „Tap here to start!“ die nächste Activity aufgerufen wird. Das Interessante hierbei ist, dass es sich hier nicht um einen Button handelt, sondern um einen sogenannten TextView, welcher durch einen onClickListener klickbar gemacht wurde.

**3.1.2 StartScreenActivity.java**



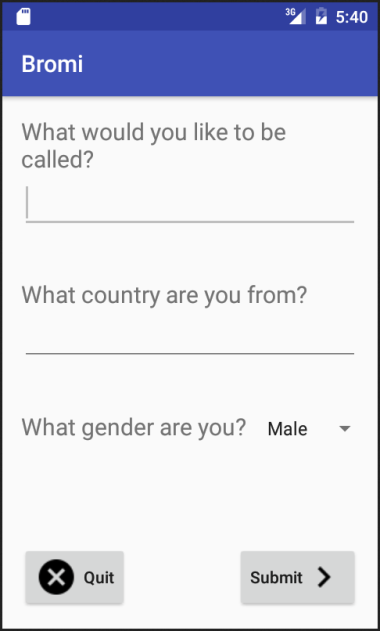
Nun befinden wir uns im eigentlichen Startmenü der App. Der Nutzer hat zwei Buttons zur Verfügung, mit welchen er sich entweder direkt einloggen kann, oder ein Profil erstellen kann. Beide Buttons führen zu ihren jeweiligen nachfolgenden Activities LogInActivity.java oder CreateProfileInfo.java. Da die Dokumentation sich nach der Reihenfolge richtet, in der die Klassen erstellt wurden, wird zunächst die programmatische Vorgehensweise einer Profilerstellung vorgestellt.

**3.1.3 CreateProfileInfoActivity.java**

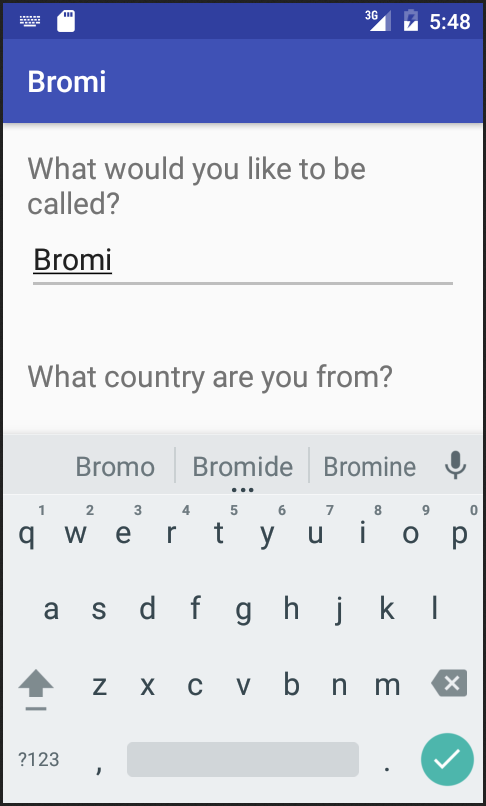


Hierbei handelt es sich um eine kurze Informationsanzeige über die App. Der TextView wurde in einem Vertikalen ScrollView eingebettet, um keine Probleme mit kleineren Smartphones zu verursachen. Der Text selbst ist in der strings.xml Ressourcendatei zu finden. Ein Klick auf den „Next“-Button ruft die Methode initProfileQuestions(View view) auf und leitet den Nutzer zur nächsten Activity CreateProfileActivity.java. An dieser Stelle sollte erwähnt werden, dass alle Buttons eine sogenannte onClick()-Methode wie hier benötigen, weil der Button ansonsten nichts macht. Diese Methoden müssen die Struktur public void onClickMethod(View view) besitzen, da sie sonst von der XML Datei aus (via android:onClick=“onClickMethod“) nicht erkannt werden. Button-onClick() kann auch programmatisch, wie im Falle des „Tap here to start!“ Buttons, mit einem onClickListener erstellt werden, aber dies ist laut der Android Dokumentation von Google nur dann zu empfehlen, wenn der Button während der Laufzeit sein Verhalten ändert oder erstellt wird. Mit der XML Variante spart man nebenher auch einige Zeilen Code.

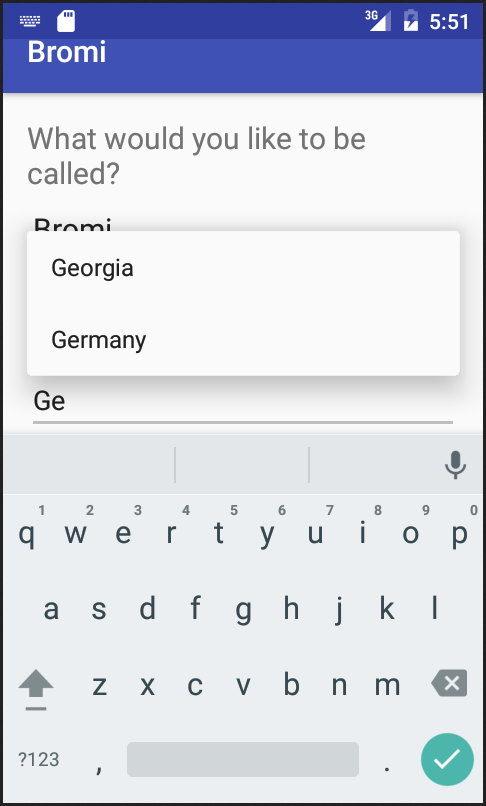
**3.1.4 CreateProfileActivity.java**



Nun kommen wir zu ersten Activity, wo mehr als nur Button-Klicks passieren. Es handelt sich hierbei um die Profilerstellung. Der Nutzer wird hier nach einem Namen gefragt, sowie das Land, aus welchem er kommt. Die Frage nach dem Geschlecht war aus experimentellen Gründen erstellt worden, da ich mir einen sogenannten Spinner anschauen wollte. Bei der Frage nach dem Namen kann sich der Nutzer ein beliebiges Pseudonym ausdenken, nach dem er benannt werden möchte. Ein Klick auf das EditText-Widget initiiert die Tastatur des Smartphones, mit dem der Nutzer einen Namen eingeben kann, wie man in der folgenden Abbildung sehen kann.



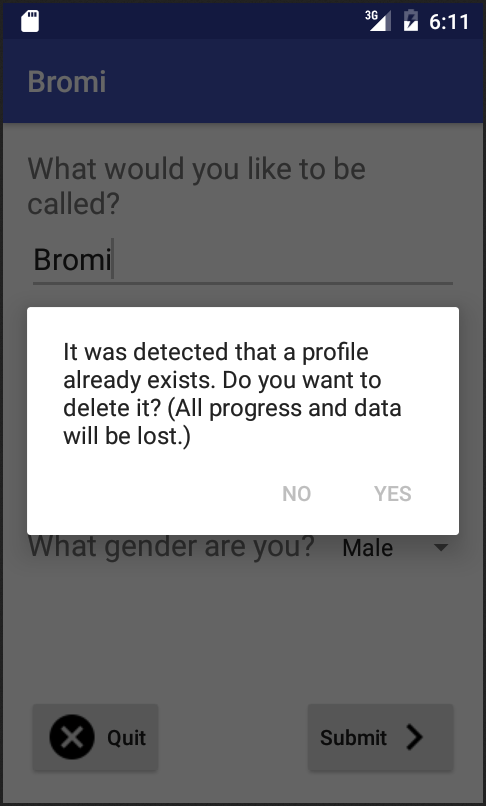
Dasselbe gilt auch für die Frage nach dem Land, aus dem der Nutzer kommt. Der Unterschied hierbei ist, dass hier eine Art Auto-Finish-System von der Google Doc übernommen wurde, die automatisch bei laufender Eingabe eines Ländernamen Vorschläge zum Antappen gibt.



Die Ländernamen sind alle in strings.xml gespeichert und werden durch einen ArrayAdapter in der initAutoCountryAdapter() Methode an das EditText-Widget angefügt. Den Rest erledigt das OS. Bei der Angabe eines Geschlechts wurde ein Spinner verwendet, welches ein Dropdown-Menü mit ein paar Auswahloptionen darstellt.

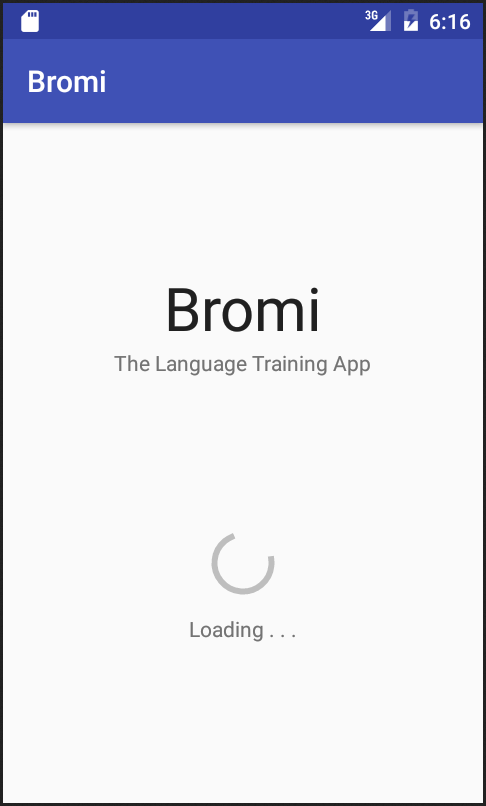
Sowohl das Name-Widget, als auch das Länder-Widget wurden mit TextWatchern ausgestattet. Dies ermöglicht die Validierung der eingegeben Daten während der Laufzeit - genauer gesagt, wenn der Nutzer fertig mit einer Eingabe ist und das EditText-Widget verlässt, was durch die TextWatcher-Methode afterTextChanged() impliziert wird. Beim Namen wird kontrolliert, dass dieser mindestens 3 Zeichen lang ist. Beim Land wird lediglich sichergestellt, dass es sich um ein legitimes Land handelt. Wenn etwas nicht in Ordnung ist, wird ein sogenanntes Toast, also eine kurze Meldung, angezeigt, um den Nutzer zu informieren.

Wenn alle Daten eingegeben wurden, kann der Nutzer auf den „Submit“-Button klicken. Dies ruft die Methode submitProfile Data(View view) auf. Hier werden die Daten mit Gettern entnommen, so wie erneut sichergestellt, dass die Daten alle valide sind. Wenn nicht, sollen die entsprechenden Toasts angezeigt werden und die Methode abgebrochen werden. Wenn alles in Ordnung ist, wird mit der createProfile-Methode eine JSON-Datei erstellt, welches das Profil repräsentiert (siehe createProfileJSONObject()). Das JSON-Objekt wird unter den Namen PROFILE in den internen Speicher geschrieben (openFileOutput und fos.write()). Im Falle, dass der Nutzer bereits ein Profil besitzt, wird in dieser Methode ein AlertDialog angezeigt, in welcher der Nutzer gefragt wird, ob wirklich ein neues Profil erstellt werden soll. Profile können nur auf diese Weise gelöscht werden.



Wenn die Profilerstellung abgeschlossen ist, wird der Nutzer zur LogInActivity.java weitergeleitet und anschließend ins Hauptmenü der App. Wenn der Nutzer die Erstellung abbrechen möchte, kann er jederzeit auf den „Quit“-Button drücken, um zu StartScreenActivity.java zurückzukehren.

**3.1.5 LogInActivity.java**

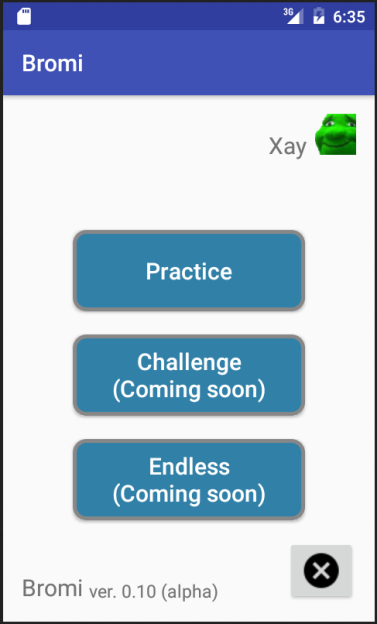


In meinem Projektplan habe ich einen Log-In-Bildschirm vorgestellt, was genau diese Activity darstellen soll. Beim Ausführen der App wird dieser Ladebildschirm jedoch übersprungen. Den Grund dafür konnte ich nicht herausfinden. Da das Programm so funktioniert, wie es soll, habe ich mich damit nicht weiter beschäftigt, sondern weiter die Pflichtpunkte des Projekts programmiert.

In dieser Activity werden alle benötigten Ressourcen für die App geladen. Hierzu gehören bisher das JSON-Objekt des Profils und die SQLite-Datenbank mit den Levels und Wörtern des später vorgestellten Practice-Modus. checkProfile() schaut nach, ob ein Profil existiert. Wenn ja, dann werden die JSON-Daten aus dem internen Speicher (openFileInput) herausgelesen und in einer HashMap<String, String> gespeichert. Wenn kein Profil existiert, wird der Nutzer eine entsprechende Meldung bekommen und zu CreateProfileInfoActivity.java weitergeleitet. Für die Datenbank wird in der loadLanguageLevelDb() Methode kurzerhand insertLevelData() aufgerufen, welches im Package „db“ bei der LanguageLevelDbHelper Klasse die Erstellung einer SQLite Datenbank initiiert, sofern noch keine existiert. Mehr Informationen zur Datenbank gibt es später unter Punkt 3.3.

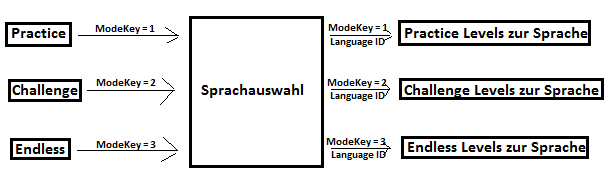
Anstatt die JSON-Datei jedes Mal zu laden, wenn Daten zum Profil benötigt werden, wird die Profil-HashMap nun durch den gesamten weiteren Verlauf der App „getragen“. Das heißt, die HashMap wird von einer Activity zur nächsten übernommen. Schlüssel hierfür ist die beim Intent verwendete Methode putExtra(Key, Data). Hierdurch wird ein Bundle erstellt, welches bei der nächsten Activity mit Gettern für bestimmte Datentypen „ausgepackt“ werden kann und somit die übernommene Data mit dem jeweiligen Key in der neuen Klasse als Variable abspeichern kann. Im Verlauf der App wird dieses äußerst nützliche Feature häufiger zur Verwendung kommen, wie man später sehen wird.

**3.1.6 MainMenuActivity.java**



Hier zu sehen ist das Hauptmenü der App. Es gibt einen „Quit“-Button, welcher zurück zu StartScreenActivity.java leitet, einen „Practice“-Button, sowie das Profil, welches mittels Klick auf das Profilbild angezeigt wird (mehr zum Profil später unter 3.1.11). Challenge- und Endless-Button funktionieren derzeit noch nicht, sind aber trotzdem vorhanden, da es beim Projektplan genauso gemacht wurde. Das Profilbild, welches momentan verwendet wird, ist ein Sub-Emote des Twitch-Partners ZFG1, welches durch seine Zustimmung verwendet werden darf (siehe Anhang).

Durch einen Klick auf den „Practice“-Button wird der Nutzer zur Sprachauswahl weitergeleitet. Weiterhin wird erneut per putExtra() die Modus-ID weitergetragen, damit das Programm eindeutig zwischen Practice-, Challenge- und Endless-Modus unterscheiden kann und nach der erfolgten Sprachauswahl den richtigen Modus lädt. Siehe hierzu noch einmal die Folgende Abbildung aus meinem Projektplan:



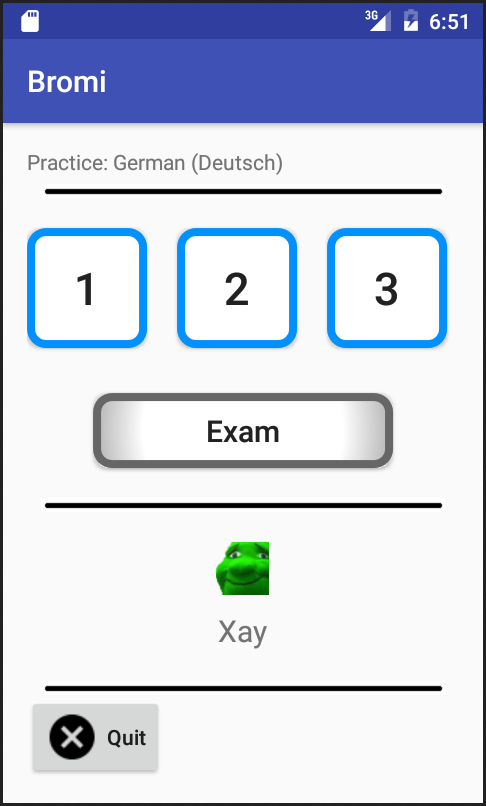
**3.1.7 LanguageSelectActivity.java**



Hier zu sehen ist nun der Sprachauswahlbildschirm. Hier kann der Nutzer die Sprache auswählen, zu welcher er gerne den ausgewählten Modus spielen möchte. Momentan gibt es aus Sparzwecken nur Training für die Deutsche Sprache. Die Buttons zu den anderen Sprachen sollten dennoch vorhanden sein, was mit dem Tutor abgesprochen wurde.

Das Wichtige bei dieser Activity ist das Hinzukommen der LanguageID Variable. Sie wird, genauso wie ModeId und das Profil, nun mit putExtra() durch den weiteren Verlauf der App getragen. Ein Klick auf den „Quit“-Button führt zurück zum Hauptmenü.

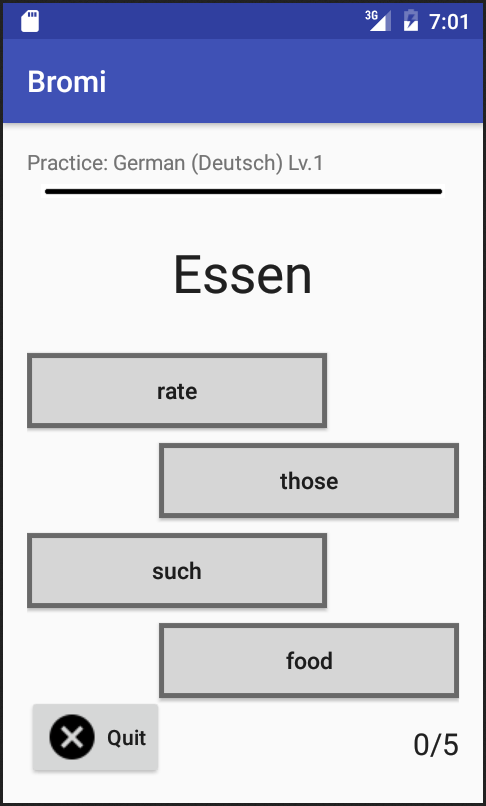
**3.1.8 PracticeLevelSelectActivity.java**



Nun kommen wir zur ersten Activity, die vom Design her vom Projektplan stark abweicht. Am Anfang wurde die gekachelte Level-Auswahl erfolgreich übernommen. Da dies jedoch mit kleinen Smartphone-Screens Probleme verursachte, wurde die Level-Auswahl in einem horizontalen ScrollView eingebettet. Sie passt sich automatisch an verschiedenen Bildschirmgrößen an, ohne dabei die Buttons zu überlappen. Weiterhin wurde statt der Progress-Bar im unteren Bereich des Layouts ein Link zum Profil eingefügt, auf welchem alle Statistiken, inklusive von Fortschrittsanzeigen, angezeigt werden sollen. Dies ist natürlich momentan nicht vorhanden, da das nicht eine Pflicht für dieses Projekt ist. Ebenso nicht vorhanden ist eine Art Fortschrittsanzeige um die Level-Buttons herum. Hierfür habe ich stattdessen verschiedene Rahmenfarben geplant (blau = ungespielt, rot = failed, grün = passed, gold = 100% correct). Neu hinzugekommen ist ein „Exam“-Button, welcher derzeit aber nicht funktioniert und deshalb vergraut ist, da auch dies kein Pflichtpunkt für das Projekt ist.

Das Programm selbst soll hauptsächlich dazu dienen, die Level-Buttons klickbar zu machen und das angeklickte Level mit Hilfe der startLevelClicked(View view) Methode zu identifizieren. Hier kommt für das putExtra-Bundle eine LevelId hinzu. Im Anschluss wird PracticeLevelActivity.java gestartet und das eigentliche Spiel beginnt.

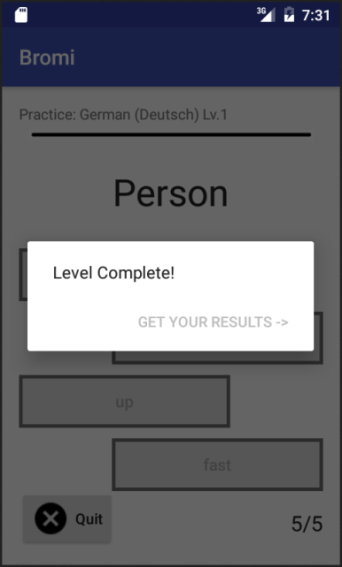
**3.1.9 PracticeLevelActivity.java**



Auch hier gibt es einige Abweichungen vom originalen Plan. Die Antworten sind nun in einer Weise angeordnet, dass sie sich bei verschiedenen Smartphone-Größen automatisch anpassen. Weiterhin wurde der Vokabelzähler nach unten rechts verschoben. Bei dieser Activity handelt es sich um das eigentliche Spiel der App. Die Klasse ist dementsprechend auch sehr groß geworden. Ich werde nun grob erklären, wie das Level durchgespielt wird und was zwischen jeder Vokabel passiert. Für weitere Informationen sollten die Kommentare zu den einzelnen Methoden ausreichen.

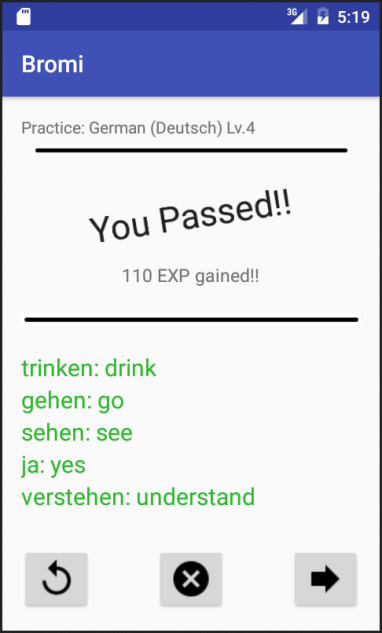
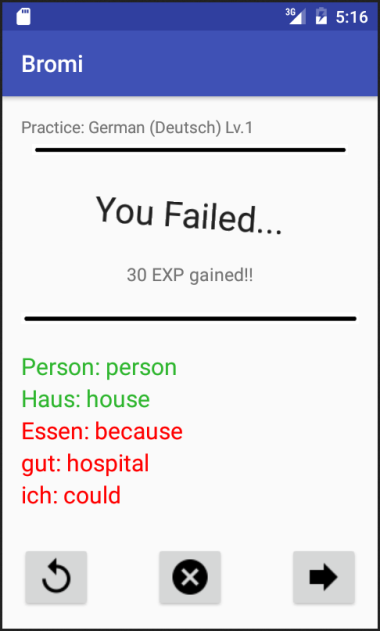
Als erstes wird mit Hilfe der bis hierher transportierten ID’s das benötigte Level aus der Datenbank geladen (loadLevelData()) und als HashMap unter der Klassenvariable currentLevel gespeichert. Hierbei handelt es sich um Wortpaare, bei denen Key das Fremdwort ist (im Bezug zur oberen Abbildung das Wort „Essen“) und Value die richtige Antwort ist („food“). Wenn alle weiteren Variablen durch das Bundle instanziiert wurden und die setText() Methoden ausgeführt wurden, wird runLevel() aufgerufen, welches die Schaltzentrale des Spiels repräsentiert. In dieser Methode werden zwei mögliche Spielzustände unterschieden. Einmal der Zustand isNewLevel = true, bei welchem das Level von PracticeLevelSelectActivity.java aus gestartet wurde und als komplett neu initiiertes Level angesehen wird. Als Zweites gibt es den Zustand isNewLevel = false, bei welchem das Level vom später vorgestellten Ergebnis-Screen „restarted“ wurde. Der Kernunterschied zwischen diesen beiden Zuständen ist, dass bei isNewLevel = true die Vokabelfolge erst generiert werden muss und bei isNewLevel = false diese bereits vorhanden ist und beibehalten werden soll.

Bei isNewLevel = true wird demnach durch die Methode getVocabulary() eine Vokabel aus der currentLevel HashMap zufällig ausgewählt und dem Nutzer angezeigt. Durch die generateAnswerButtonCaptionOrder() Methode werden aus der Klasse LanguageLevelData.java im Package „db“ aus dem word-Array zufällig vier verschiedene Wörter ausgewählt, welche als Falschantworten dienen sollen. Eine davon wird durch die richtige Antwort ersetzt und bildet somit das Answer-Caption-Set, die später auf die Antwort-Buttons geschrieben werden (answerX.setAnswerText()). Sowohl getVocabulary() als auch generateAnswerButtonCaptionOrder() sind so programmiert, das jegliche Doppelungen von Vokabeln und Errors nicht passieren sollten. Dies wird solange wiederholt, bis alle Vokabeln aus currentLevel abgefragt wurden. Die Schleife hierfür ist in den rekursiven Aufrufen von runLevel() in sich selbst oder durch das Drücken eines Antwortbuttons in der Methode submitAnswer() versteckt. Die Antwort wird in einer ArrayList mit dem Namen answersGiven gespeichert. Ob die Antwort richtig oder falsch ist, wird über die compareAnswers()-Methode kontrolliert und in Form von String-Bools (true oder false) einer weiteren ArrayList hinzugefügt, die den Namen correctAnswersGiven trägt. Wenn alle Vokabeln abgefragt wurden, wird eine provisorische Meldung angezeigt und der Ergebnis-Bildschirm initiiert.

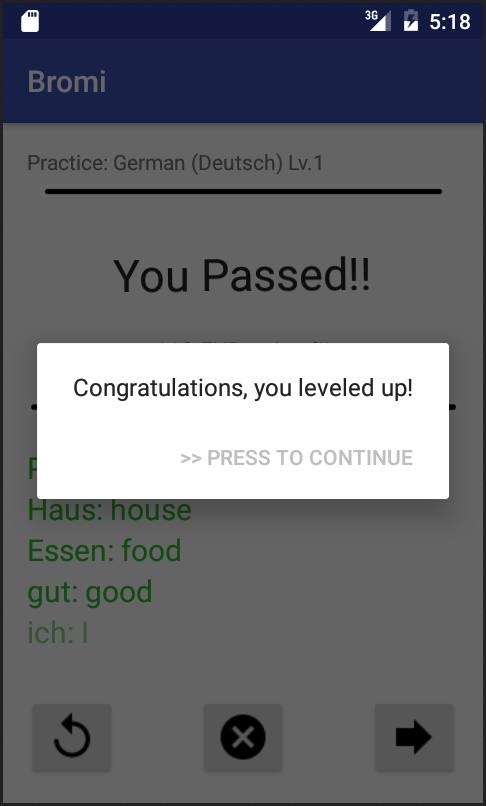


Im Zustand isNewLevel = false ist dieser Vorgang identisch, nur dass die getVocabulary() Methode übersprungen wird und stattdessen die Vokabel-Daten vom bereits gespielten Level wiederverwendet werden. Als letzte Funktion ist zu erwähnen, dass die Profilstatistiken geupdated werden, wenn der Nutzer während des Levels auf den „Quit“-Button drückt (siehe setProfileStats()). Alle erstellten ArrayLists (currentLevel, answersGiven, vocabularyUsed und correctAnswersGiven) werden in der Methode initResultScreen() via putExtra() beigefügt.

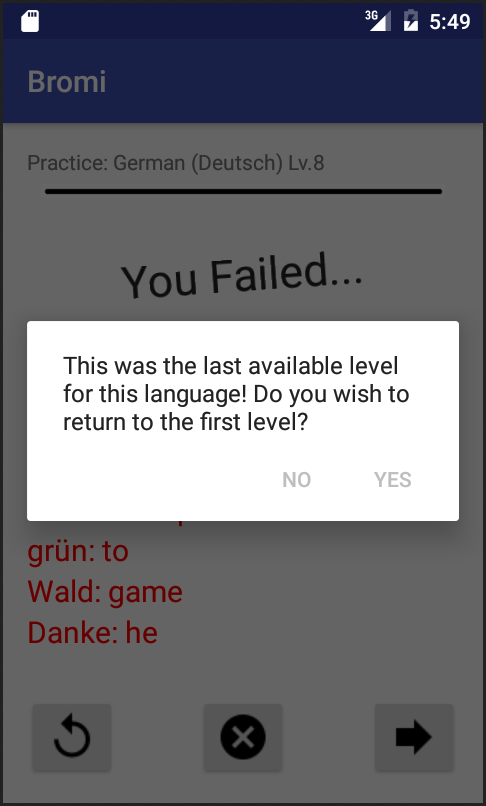
**3.1.10 PracticeLevelResultActivity.java**



Diese Activity ist für die Generierung der Erfahrungspunkte, sowie der Bewertung der Nutzerleistung im gespielten Level verantwortlich. Als Erstes wird in der Methode computeAndShowResults(), mit Hilfe der correctAnswersGiven ArrayList, ermittelt, wie viele Vokabeln der Nutzer richtig übersetzt hat (int correctAmt). Zu jeder Vokabel, die kontrolliert wird, wird ebenso die Methode createVocabularyTextView() aufgerufen, welche programmatisch ein TextView-Widget an einem LinearLayout Objekt anhängt, wobei der Kernunterschied nur die Farbe des Textes ist (richtig = grün, falsch = rot). Hierdurch kann der Nutzer später, wenn das Layout angezeigt wird, sehen, welche Vokabeln er richtig hat und welche nicht. Als nächstes kann, aufgrund der gezählten richtigen Vokabeln, in der Methode showGrade() der Ergebnistext entweder auf „You Failed…“ oder „You Passed!!“ eingestellt werden. Die Bedingung, um ein Level zu bestehen, ist lediglich 50% der Vokabeln richtig zu beantworten. Das heißt, man muss im momentanen Zustand der App drei von fünf Vokabeln richtig haben, um zu bestehen. Danach wird die Methode computeExperience() initiiert, welches durch die gezählten richtigen Vokabeln die Erfahrungspunkte (EXP) errechnen kann, die der Nutzer durch seine Performance im Level bekommen kann. Generell ist die Balance der EXP, die man erhält, sehr ausgeglichen, selbst wenn man besteht oder nicht besteht. Für das Fertigspielen des Levels gibt es garantiert 10 EXP. Es gibt keine Strafpunkte wenn man nicht bestanden hat, aber genauso gibt es keinen Bonus wenn man „nur“ bestanden hat. Für jede richtige Vokabel erhält man 10 EXP. Das heißt also, dass der Unterschied der Anzahl EXP, die man beim Bestehen und Nichtbestehen erhält, gerade mal 10 EXP groß sein kann. Dafür wird ein großer Bonus vergeben, wenn der Nutzer alle Vokabeln richtig beantwortet hat. Man erhält so 10 EXP für das Fertigspielen des Levels, plus 50 EXP für alle 5 richtigen Vokabeln, plus 50 EXP Bonus für das Meistern des Levels. Somit beträgt sich das Maximum der erhältlichen EXP auf 110. Ich erhoffte mir mit diesem Balancing der EXP, dass die Motivation des Spielers ein Level zu meistern erhöht wird. Im Falle eines Level-Ups (bei 500 EXP), erhält der Spieler auch ein kleines Pop-Up:

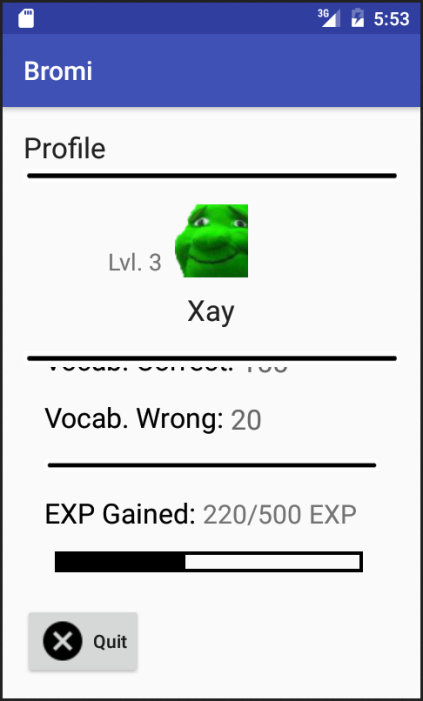
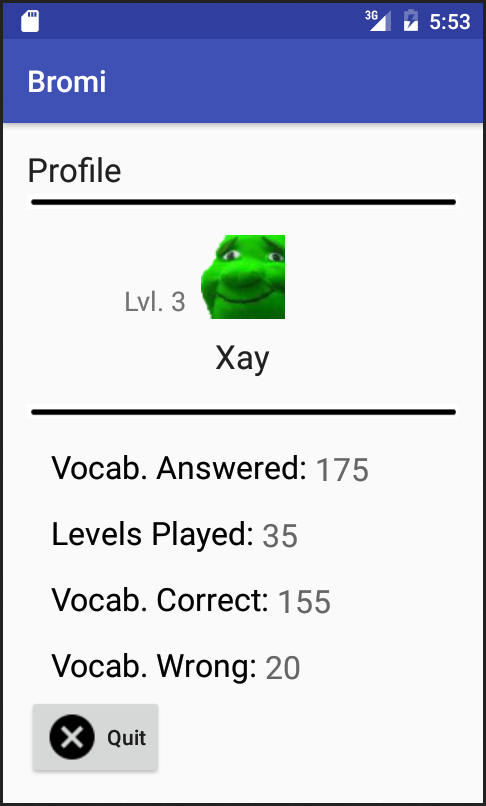


Nachdem die Ergebnisse erstellt und animiert wurden, muss der Nutzer einen von den drei Buttons im unteren Rand der Activity drücken. Der runde Pfeil bedeutet, dass man das Level noch einmal spielen kann (redoLevel()). Hierbei wird die Reihenfolge der generierten Vokabeln beibehalten und in PracticeLevelActivity.java wiederverwendet. Mit dem großen „X“-Button in der Mitte kann man zurück zur Level-Auswahl gelangen (returnToLevelSelectScreen()). Mit dem dicken Pfeil nach rechts kann man zum nächsten Level gelangen (nextLevel()). Wenn der Nutzer bereits beim letzten Level angekommen ist, erhält der Spieler eine Meldung, dass es nicht mehr weiter geht und kann entweder zum ersten Level zurückkehren oder zurück zur Level-Auswahl gehen.



Alle drei Buttons haben gemeinsam, dass das Profil mit saveProfileToJSON() im internen Speicher des Handys gespeichert wird, sodass der Fortschritt nicht verloren geht.

**3.1.11 UserProfileActivity.java**



Die letzte Activity-Klasse der App, die erstellt wurde, ist das Nutzerprofil. Dieses soll lediglich dazu da sein, eine Reihe Statistiken darzustellen, wie man in den Abbildungen gut sehen kann. Das Besondere ist die EXP-Bar, welche den Fortschritt bis zum nächsten Level relativ zu den bereits erhaltenen Erfahrungspunkten anzeigt. Weiterhin ist es wichtig zu wissen, dass das User-Profil von verschiedenen Activities aus aufgerufen werden kann (momentan MainMenuActivity.java und PracticeLevelSelect.java). Dementsprechend gibt es für den Quit-Button einen Identifier openedFrom, der zur richtigen Activity zurückführen soll.

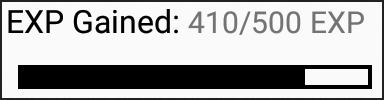
**3.2 Package: Custom**

Bei diesem Package handelt es sich lediglich um einen Speicherort für selbsterstellte Elemente, die für die App entwickelt wurden.

**3.2.1 ExperienceBar.java**

Diese Klasse ist eigens für das Zeichnen eines Rechtecks erstellt worden, die den Erfahrungspunktefortschritt bis zum nächsten Level up darstellen soll. Da sie eine Unterklasse von View ist, kann sie in der XML-Layout-Datei wie jedes andere View-Objekt verwendet werden.

Die EXP-Bar besteht aus zwei Rechtecken xpRect und baseRect. Letzteres ist ein leeres Rechteck und repräsentiert die Gesamtlänge der EXP-Bar. xpRect wird auf das baseRect als schwarzgefülltes Rechteck gemalt, sodass der Fortschritt angezeigt wird. Die Länge des xpRect wird in der drawXpRectangle() in der Klasse UserProfile Activity.java errechnet.

http://i.imgur.com/9Rv2gRy.png

**3.3 Package: db**

Bei diesem Package handelt es sich um eine SQLite-Datenbank, welches Level-Daten zu den verschiedenen Sprachen speichern soll. Hierzu muss gesagt werden, dass ich noch nie mit Datenbanken oder SQL gearbeitet habe, was bedeutet, dass ich mir SQL zuerst etwas anschauen und lernen musste, bevor ich Queries schreiben konnte. Der Grund, warum ich anstelle von JSON, wie es im Projektplan dargestellt wurde, nun eine Datenbank verwendet habe, ist wegen dem Interesse, der mein Tutor bei mir geweckt hat. Wenn das Projekt tatsächlich weiterentwickelt und größer wird, so habe ich mit der Datenbank weiterhin ein solideres, erweiterbareres und schnelleres Zugriffssystem für Level-Daten, als mit JSON.

**3.3.1 LanguageLevelData.java**

Trotzdem mussten die Level-Daten irgendwo herkommen. Ich habe es nicht geschafft, die Level-Daten wie im Projektplan vorgestellt in einem Directory-Baum für verschiedene Sprachen mit .txt Dateien zu speichern, weil Android eben doch anders ist als Windows. Dementsprechend habe ich mich dazu entschieden, die Daten als Arrays in einer Java-Klasse zu speichern. Eigentlich hätte ich die Datenbank komplett weglassen und die Levels von den Arrays aus aufrufen können, aber meine Motivation, eine Datenbank zu erstellen, war zu hoch. Außerdem kann man nicht wissen, was für weitere Daten man zu einer Column hinzufügen möchte, vor allem im Hinblick auf Achievements. Hierzu gehört zum Beispiel, wie oft eine bestimmte Vokabel beantwortet wurde oder ob alle Vokabeln richtig beantwortet wurden oder nicht usw.

**3.3.2 LanguageLevelDbHelper.java**

Hierbei handelt es sich nun um die SQLite-Datenbank. Es gibt die wichtige insertLevelData() Methode, welche die Datenbank mit Inhalt füllt, und die getLevel() Methode, welche die benötigten Level-Daten anhand der levelId aus der Datenbank rausnimmt. onCreate() erstellt die Datenbank und wird aufgerufen sobald ein Objekt dieser Klasse instanziiert wird, wie z. B. in LogInActivity.java. Mit der Methode readWordPairsAsMap() kann man die in getLevel() gelesenen Daten direkt zu einer HashMap konvertieren.

**3.4 Package: util**

In diesem Package sind Konstanten und nützliche Methoden erhalten, die im gesamten Programm Verwendung finden aber in keiner Klasse wirklich „reinpassen“.

**3.4.1 constants.java**

In dieser Klasse wurden verschiedene statische Konstanten gespeichert, die in der gesamten App Verwendung finden. Dies soll dazu dienen, dass man die verschiedenen Werte nicht alle einzeln in jeder Klasse, wo sie verwendet werden, ändern muss.

**3.4.2 methods.java**

In dieser Klasse findet man verschiedene statische Helfermethoden, welche immer wieder in den Klassen Verwendung finden (könnten). Ein paar Methoden sind von Stackoverflow übernommen worden, die entsprechenden Quellen wurden angegeben. Weiterhin gibt es eine Reihe Identifier-Methoden, wie getLanguageFromId(), getImageResourceId() und getIntentFromId(). Ebenso enthalten ist eine showToast() Methode, welche ein kleines Meldungsfenster auf der angegeben Activity kreiert.

**4 Fazit**

Nachdem das Projekt nun vorgestellt und erklärt wurde, bleibt nur noch die Bewertung. Mir persönlich hat das Projekt, trotz des großen Arbeitsanteils und der überzogenen Dokumentation, viel Spaß gemacht, was hoffentlich durch den Umfang der App ersichtlich wird. Durch das Wechseln von Designarbeit und der Programmierung gab es viel Abwechslung, es wurde nie langweilig. Dank des Projektplans hatte ich auch immer vor Augen, was als nächstes zu tun war.

Google hat mit Android Studio eine sehr komfortable Entwicklerumgebung zur Verfügung gestellt und mit der detaillierten Dokumentation und den Tutorials konnte ich mich sehr schnell in die Android-App-Programmierung zurechtfinden. Auch die Erstellung einer .apk hat sich als sehr einfach erwiesen und ich konnte die App auf meinem eigenen Gerät testen, sowie Freunden und Familie zeigen. Die Reaktionen waren allesamt mit positiven Reaktionen geschmückt, sodass das Erfolgsgefühl bei mir entsprechend groß war (eine Liste der Handys, auf welchem die App getestet wurde, ist im Anhang zu finden).

Mit diesem Projekt habe ich es geschafft, mir eine Zusatzqualifikation anzueignen, die ich mit einem möglichen Beruf verbinden kann, worüber ich sehr erfreut bin. Wenn ich mich mit Datenbanken im nächsten Semester auseinandergesetzt habe, kann ich mir sehr gut vorstellen, nach einem Praktikum und vielleicht sogar einem Beruf, welche in die Richtung Android-App-Programmierung und SQL/PHP gehen, Ausschau zu halten. Dieses Seminar hat meinen Horizont am meisten von allen Projektseminaren erweitert. Großes Lob gilt an dieser Stelle auch den Tutoren, die das Seminar auf diese Weise gestaltet haben.

**5 Literaturverzeichnis**

1. Google and Open Handset Alliance (n. d.) “*Android API Guide”*. http://developer.android.com/guide/index.html. Accessed between August 24th and September 14th, 2016.

**6 Anhang**

Liste der Handys, auf welchem die App (.apk) gespielt wurde:

* Samsung Galaxy S2
* Samsung Galaxy S4
* Samsung Galaxy S5
* Samsung Galaxy Alpha
* Samsung Galaxy A5
* Huawei Ascend P6
* HTC ONE
* Nexus 5X

Zustimmung von ZFG1 sein Sub-Emote „zfgRunOgre“ in der App als User-Avatar verwenden zu dürfen:

