b
UNIVERSITÄT
BERN

Using RSS Feeds to Support Second Language Acquisition

Bachelor Thesis

Linus Schwab
from
Langenthal BE, Switzerland

Faculty of Science
University of Bern

June 30, 2016

Prof. Dr. Oscar Nierstrasz
Dr. Mircea Lungu

Software Composition Group
Institute of Computer Science
University of Bern, Switzerland

Acknowledgements

First I would like to thank *Prof. Dr. Oscar Nierstrasz* for giving me the opportunity to do my bachelor project at the Software Composition Group of the University of Bern.
My special thanks to *Dr. Mircea Lungu* for his invaluable support and advice during the the project, which continued in the same way after he switched to the University of Groningen in the Netherlands and allowed me to successfully complete the project!
Additionally, I would like to thank the other members of the Zeeguu team, namely *Pascal Giehl*, *Karan Sethi* and *Ada Lungu*, who helped me with suggestions, code sharing and extending the Zeeguu project with new features.
Finally, thanks a lot to all the people who had the time and patience to take part in the usability tests of the application and provided suggestions to improve it.

Abstract

Zeeguu is a language learning platform that allows people to extend their vocabulary and language skills by reading any text they like. Additionally, it provides feedback on the progress and exercises to further improve the aquired knowledge.
This thesis introduces the Zeeguu Reader, a mobile application for Android that uses RSS feeds to automatically provide the user with the newest articles from the news sources or blogs he subscribes to, while supporting the user with translations and keeping track of his reading progress.
Its features include an article recommender that tries to automatically select the articles that fit best to the user's current language level. For the RSS part, the application uses the Feedly Cloud[1], which synchronizes the subscriptions and reading progress between different devices and allows the user to continue reading where he left on any computer in the Feedly web interface and keep learning the language with the Zeeguu Chrome plugin.

[1] `https://feedly.com`

Contents

1 Introduction

There are several ways to learn a new language nowadays: Besides traditional options like language courses and textbooks with exercises, different mobile apps and online services are available to support people on their way. However, most of these have one thing in common: You have to take your time every day specifically to learn the language. Zeeguu[1] wants to use a different approach: Instead of adding a new task to our already busy life, it allows people to keep reading their favorite news or other texts that they would read anyway, while supporting them with translations and collecting information for personalized exercises at the same time.
This approach is called extensive reading [1] (or free reading) and allows the reader to choose the texts according to his personal preferences and interests. The idea is that the reader encounters unknown words in specific contexts, which allows him to infer and therefore learn the meaning of the words. The main benefit is that the experience is focused more on the text rather than the language, which in return increases the reader's motivation.
One possible way to regularly learn a new language through extensive reading is to read news, blogs or magazines, which is conveniently possible with an RSS feed reader. This thesis introduces such an application with Zeeguu integration for the Android operating system. It allows the reader to instantly translate any unknown word in the text he is reading, and bookmark it together with the context to learn and repeat it later. Additionally, the app provides recommendations for articles to read based on the subscriptions of the user that are as close as possible to his current language level.

[1]`https://zeeguu.unibe.ch/`

2
Related Work

There are already plenty of language services available that provide language learning applications for multiple platforms, including web-based services. This chapter provides a basic overview of the currently existing solutions and will mostly focus on the most popular ones. Additionally, there will be a short description of the existing parts of the Zeeguu platform.

2.1 Other Language Learning Services

If you search for ways to learn a new language with a search engine or in the mobile app stores for Android or on iOS, there is a huge number of results: For example, searching for the term "learn english online" on Google produces about 61.2 million results.[1] Obviously those results are not all different language learning platforms, but it shows that there are already a lot of existing services available. This section introduces two popular services that also provide mobile apps to learn the language anywhere: Duolingo[2] and Babbel.[3]

[1]Checked on Google Switzerland in March 2016.
[2]`https://www.duolingo.com`
[3]`https://www.babbel.com`

2.1.1 Duolingo

One of the currently bigger language learning services with mobile applications is Duolingo. They provide apps for Android and iOS and a web platform.
For the language learning, they almost exclusively use exercises organized by topic and difficulty, however it looks like they only cover the basic language level. The exercises use a combination of text, voice and images and are solved by writing text, selecting words in the correct order, choosing the correct image or by speaking with voice recognition to make them more varying.

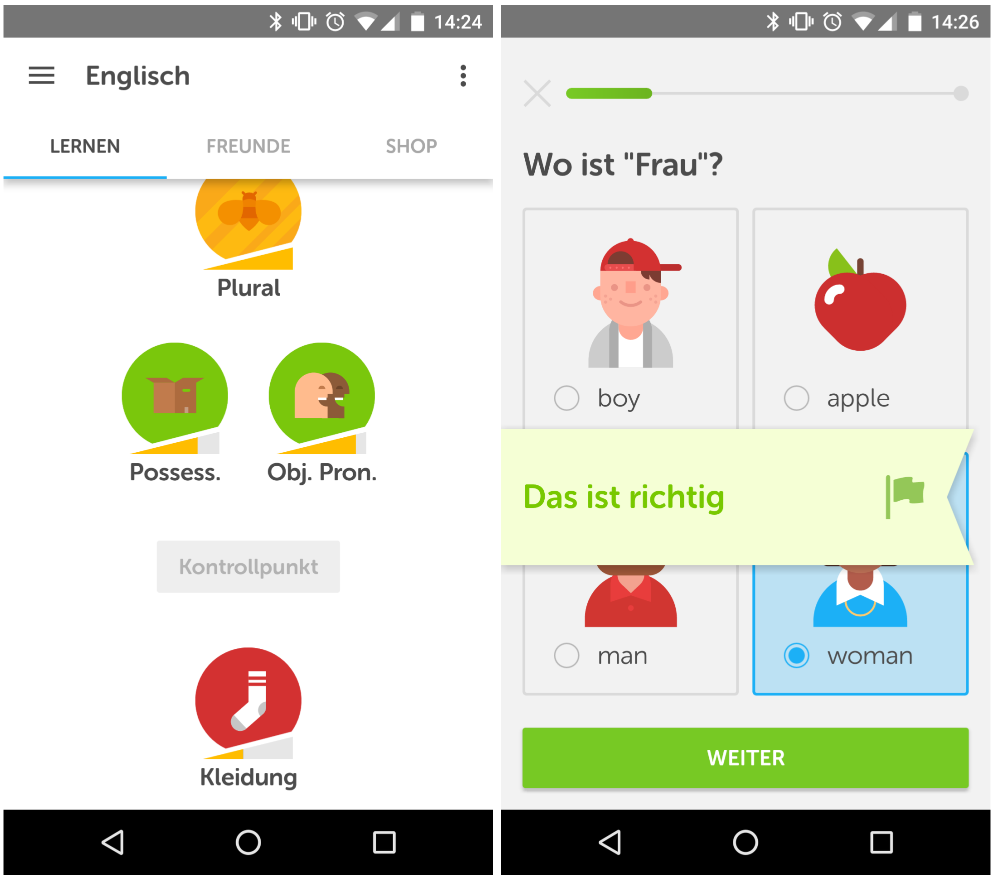
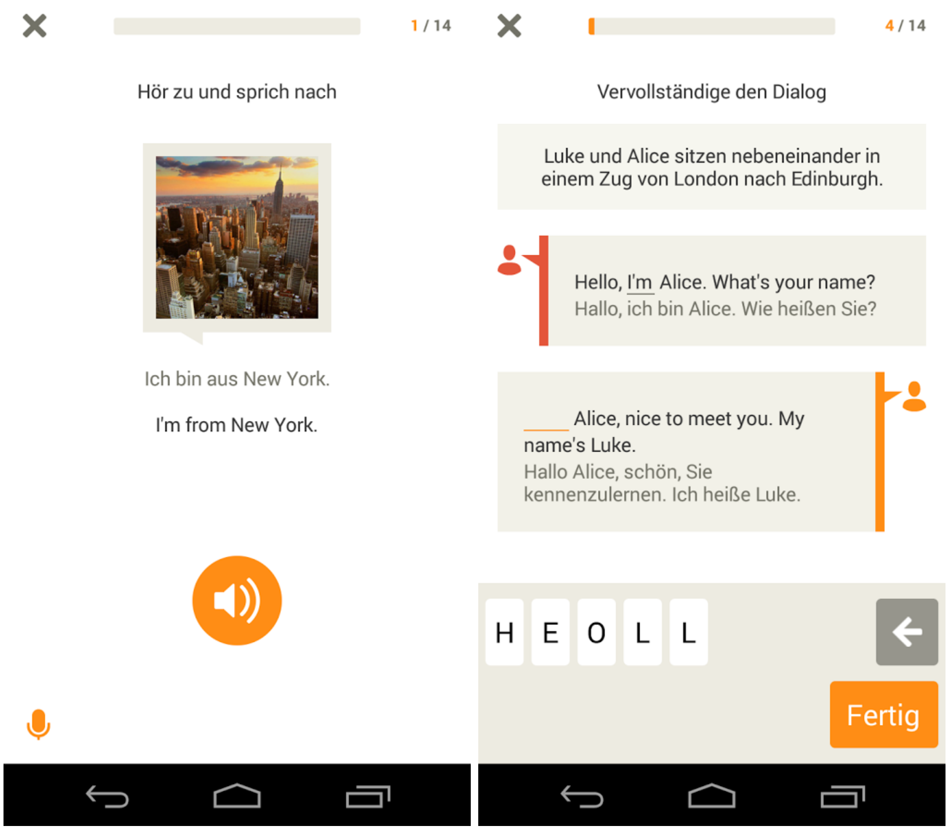


Figure 2.1: The Duolingo Android application

They also use some gamification elements: During an exercise, a progress bar is displayed that grows with each correct answer (or shrinks with each mistake). After the exercise is completed, experience points are awarded that count towards a daily goal and a user level. It then counts the number of days where the goal was met in a row and displays that value. Additionaly, some statistics like the percentage of the current language proficiency are shown. Duolingo is free with some optional premium content.

2.1.2 Babbel

Another well-known language learning platform is Babbel. They also provide a mobile app for iOS and Android in addition to a web-based application. Babbel uses an exercice-based approach too and seems to target new language learners mostly.
The exercises are organized by topic and are less in a traditional question-answer form. A more interactive style like simulated dialogues, with a lot of explanations in between and almost always supported with a corresponding image is used instead. Newly learned words or phrases occur more than once in the exercise and are directly repeated at some exercise steps.



Ich bin aus New York.

I'm from New York.

Luke und Alice sitzen nebeneinander in einem Zug von London nach Edinburgh.

Hello, I'm Alice. What's your name?
Hallo, ich bin Alice. Wie heißen Sie?

____ Alice, nice to meet you. My name's Luke.
Hallo Alice, schön, Sie kennenzulernen. Ich heiße Luke.

Figure 2.2: The Babbel Android application

Babbel uses recorded voice in combination with the written text so that the user can hear the word too instead of just reading it. There is a rating after each exercise (number of correct answers) with a short statement. There is much less gamification in this sense. It offers feedback to the user by automatically creating a vocabulary list with the words from the exercises, including a rating how well the user knows the word. Babbel is a paid service; only the first exercise of each topic is available for free.

2.2 The Zeeguu Platform

Zeeguu was initially created in 2013 by *Simon Marti* [3] and has since then been extended by various people. It now features a web platform with a vocabulary list, exercices and advanced statistics about the learning progress and language knowledge of the user. Aditionally, there is a Chrome plugin and Android applications. An iOS app is currently being developed in another bachelor's project.

2.2.1 Chrome Plugin

There is a Zeeguu plugin for the Google Chrome browser available. It provides direct translations on any webpage and allows the user to bookmark words, which are then added to his word list to consolidate in exercises later.

2.2.2 Zeeguu Quantifier

The Zeeguu Quantifier is the result of the bachelor thesis of *Karan Sethi* [5]. It quantifies the user's knowledge about the language that is currently being learned and gives them a metric about their progress.

2.2.3 Zeeguu Translate

The Zeeguu Translate app for Android was developed by *Pascal Giehl* [4] and provides a translator with the ability to add the translated words to the personal word list, allows one to display the word list, displays the web-based exercises and includes a browser.

2.2.4 Zeeguu Reader for iOS

Currently a Zeeguu Reader app is also being developed for iOS as the bachelor project of *Jorrit Oosterhof*. It will follow a different approach than this app, it is planned to hide the fact that it is an RSS reader from the user.

3 A Different Approach to Language Learning

As described in the last chapter, there is already a wealth of language learning platforms available. Most of the popular ones also provide mobile applications for iOS and/or Android, so why do we need another one? Most of those applications only feature a limited number of exercises created by the developers that in most cases only cover the basic language levels and are not really personalized. Zeeguu promotes a different approach than the commonly used exercise-based one: It is based on extensive reading [1] (or free reading), which allows the reader to choose texts in the language he wants to learn according to his personal preferences and interests and learn the language while reading something he enjoys.
Compared to simply reading a text without any help, Zeeguu supports the user with translations while reading and automatically collects information for personalized exercises and advanced language knowledge statistics at the same time. The idea is that the reader encounters unknown words in specific contexts, which allows him to infer and therefore learn the meaning of the words. The main benefit is that the experience is focused more on the text rather than the language, which in return increases the reader's motivation.
Other language learning applications like Duolingo try to use gamification to keep the user motivated, with things like experience points, user levels, awards or the number of days learned in a row or concepts like user points in the case of Babbel. This seems to be necessary, because even with a variety of exercise types, they might get boring fast otherwise.
The Zeeguu Reader extends the supported extensive reading approach to mobile devices.

Before this application was developed, there was no way to directly read texts with Zeeguu on a mobile device.[1] While reading any RSS feed the user wants, the application directly provides translations and makes it possible to bookmark any unknown word at the tap of a fingertip, without having to switch the app first, and simultaneously extracts the context in the background.
Zeeguu is currently mostly oriented towards advanced learners, because the reader already needs to know the basic vocabulary to be able to read and understand texts that are not specifically tailored for beginners.

[1]The browser fragment which is integrated in the Zeeguu Translate app was developed as part of this project (based on an extended version of the Android WebView)

4 Application Architecture

The following chapter provides an overview and explains the different parts of the application's architecture. The full source code is open source and available on Github.[1]

4.1 Architecture Overview

Figure 4.1 gives an oververview of the application architecture. The yellow box on the left side represents the back end; the user interface is shown on the right side. The upper box provides an overview of the fragment navigation in the MainActivity; the lower part is about the SettingsActivity. This overview has been simplified to stay clean and readable. A more detailed description will follow in the next sections.

4.2 The User Interface

On Android, the user interface is built with activities and optionally fragments, which contain View (and ViewGroup) objects that draw something on the screen that the user can interact with. Activities are the main application components and provide the window for the user interface.[2] Fragments are basically a reusable portion of user interface and are dynamically replaced by the activity depending on the context. There are two possibilities to display the user interface: The activity can either directly draw the UI or only provide

[1]`https://github.com/linusschwab/zeeguu-feed-reader`

[2]`https://developer.android.com/guide/components/activities.html`

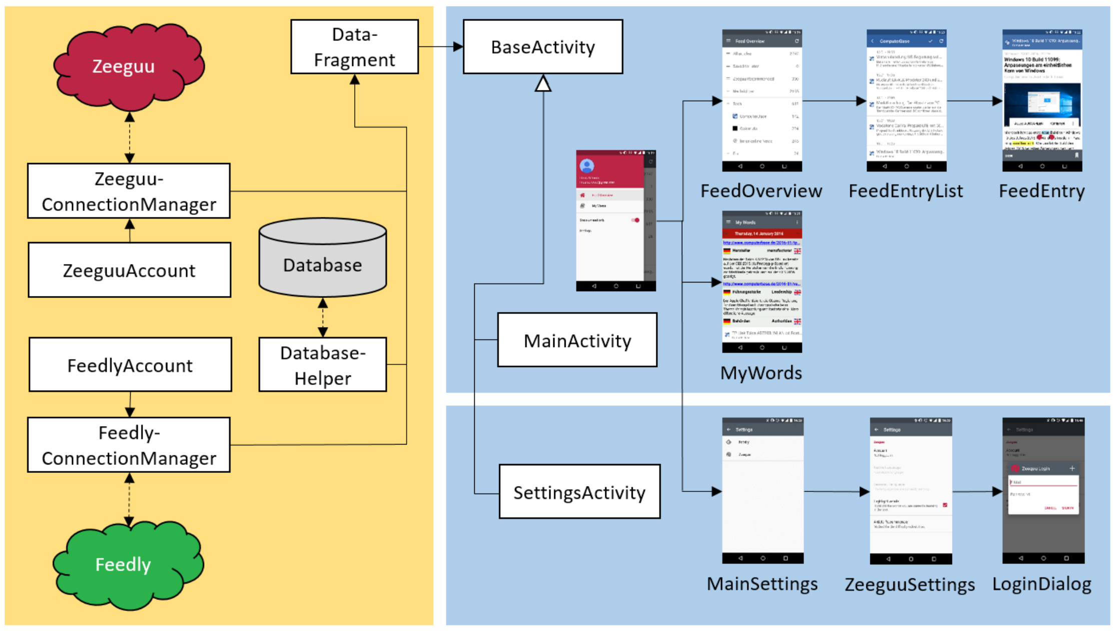


Figure 4.1: Simplified overview of the application architecture

one or multiple container views for fragments. For this application, the decision was made to use fragments whenever possible due to the reusability and reduced coupling. With the use of fragments, the activities mainly provide the app navigation and the window structure, additionally they handle the communication between the different fragments. Both activities and fragments are not purely responsible for the UI; they also contain some logic, as they both follow a lifecycle and handle input events.

4.2.1 Activities

An application usually consists of multiple activities, at least one of which is necessary. At first, the plan was to only use one activity for the Zeeguu Reader to keep the UI as fast as possible, but during the development the choice was made to use a separate activity for the settings in addition to the main one. This way, the settings fragments are separate from the main fragments which keeps the code clearer, while still having a fast UI with minimal transition time between different screens for the main part of the app. Therefore, the application uses two activities in total, which both extend from an abstract base activity.

Base Activity (Abstract)

A common base activity was introduced to prevent duplicated code and to keep the structure clean and organized. It contains the dialog fragments and helper functions used by both activities, for example to manage the fragments, backstack and current title. Additionally it makes sure that there is only one instance of the Zeeguu and Feedly API connection manager classes and the database helper class. This is achieved with the help of a static data fragment that manages the references to those objects.[3]

Main Activity

The main activity is responsible for the main UI and contains the action bar, the drawer menu for the app navigation and the sliding panel to display the current feed entry. By default, the Feed Overview fragment is displayed, which then instructs the activity using a callback interface to switch to the next fragment, the Feed Entry List.
It was important to use only one activity for this part to keep everything fast without visible transitions or delays due to increased overhead from the creation of additional activities.

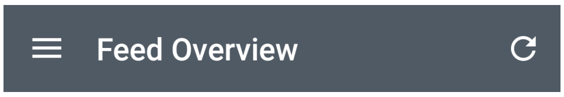


Figure 4.2: The action bar

Settings Activity

The settings activity manages the setting fragments and makes it possible to easily implement different preference screens for the setting categories. It is organized into a overview screen, which links to the different settings categories. For this, the Android Preference API was used.[4]

4.2.2 Fragments

While the activities were responsible for the basic structure of the user interface and the app navigation, fragments are used to display the actual content.
A fragment is basically a reusable portion of user interface that is (and needs to be) embedded into an activity. Fragments have their own lifecycle, but are also dependent on the lifecycle of the hosting activity. However, fragments do not necessarly need to

[3]The data fragment will be explained in the “Data Fragment” paragraph in section 4.2.2
[4]`https://developer.android.com/guide/topics/ui/settings.html`

contain a UI and can for example also be used to store data by keeping references to the objects they represent.[5]
The main design philosophy behind fragments is that they allow a more dynamic and flexible UI. It is possible to replace fragments at runtime and to display a different layout within the same activity depending on the screen size and orientation.
The following fragments are the most important ones:

Feed Overview

The feed overview is the main fragment of the Zeeguu Reader application. As the name implies, it provides an overview of all the feeds the user is subscribed to. The feeds are organized into the categories that were defined on Feedly and are expandable, so that it is possible to read each feed separately. Additionally there are default categories displayed: “All articles”, “Saved for later” (contains the entries the user marked as favorite) and “Zeeguu Recommended” (from the article recommender) – those are not expandable. Feeds that are not assigned to a category on Feedly are displayed in the “Uncategorized” category.

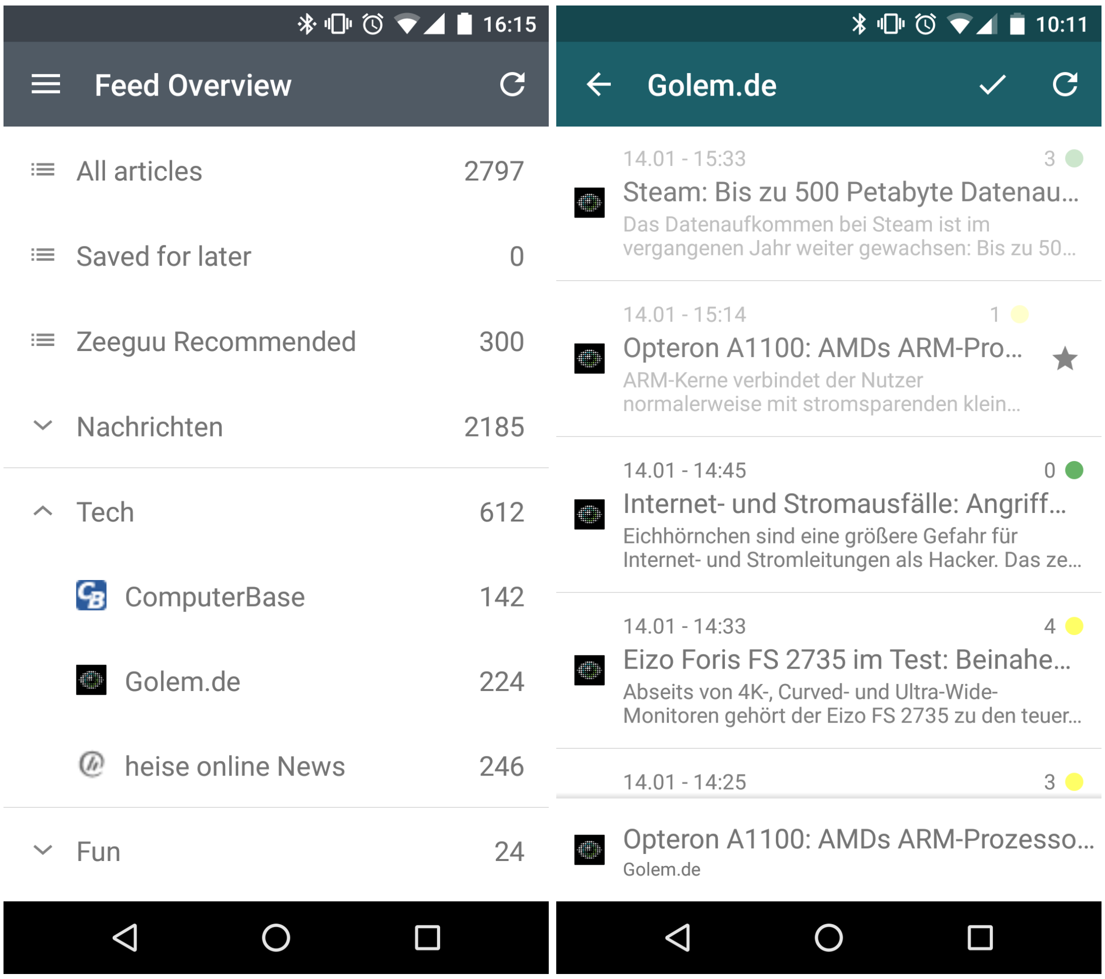
Feed Entry List

After a feed or category is selected, its entries are displayed in a ListView inside the FeedEntryList fragment. For each entry, the title, a short summary and the publication date are displayed, along with the number of words that are currently being learned and are present within the article. The difficulty score from the article recommender, which will be described in detail in chapter 5, is also displayed.

Feed Entry

Each feed entry is displayed in an instance of the FeedEntryFragment class. They are managed by a ViewPager that contains all FeedEntry fragment instances of the current feed or category and allows one to switch to the previous and next entry by swiping left or right. It is based on the ZeeguuWebViewFragment, which provides an extended Android WebView to allow the translation and bookmarking of words in the feed entry content itself and also on any webpage by injecting JavaScript code. It will be described in detail in section 4.3.1.3.

[5] `https://developer.android.com/guide/components/fragments.html#AddingWithoutUI`


16:15

Feed Overview

All articles	2797
Saved for later	0
Zeeguu Recommended	300
Nachrichten	2185
Tech	612
ComputerBase	142
Golem.de	224
heise online News	246
Fun	24

10:11

Golem.de

14.01 - 15:33 3
Steam: Bis zu 500 Petabyte Datenau...
Das Datenaufkommen bei Steam ist im vergangenen Jahr weiter gewachsen: Bis zu 50...

14.01 - 15:14 1
Opteron A1100: AMDs ARM-Pro...
ARM-Kerne verbindet der Nutzer normalerweise mit stromsparenden klein...

14.01 - 14:45 0
Internet- und Stromausfälle: Angriff...
Eichhörnchen sind eine größere Gefahr für Internet- und Stromleitungen als Hacker. Das ze...

14.01 - 14:33 4
Eizo Foris FS 2735 im Test: Beinahe...
Abseits von 4K-, Curved- und Ultra-Wide-Monitoren gehört der Eizo FS 2735 zu den teuer...

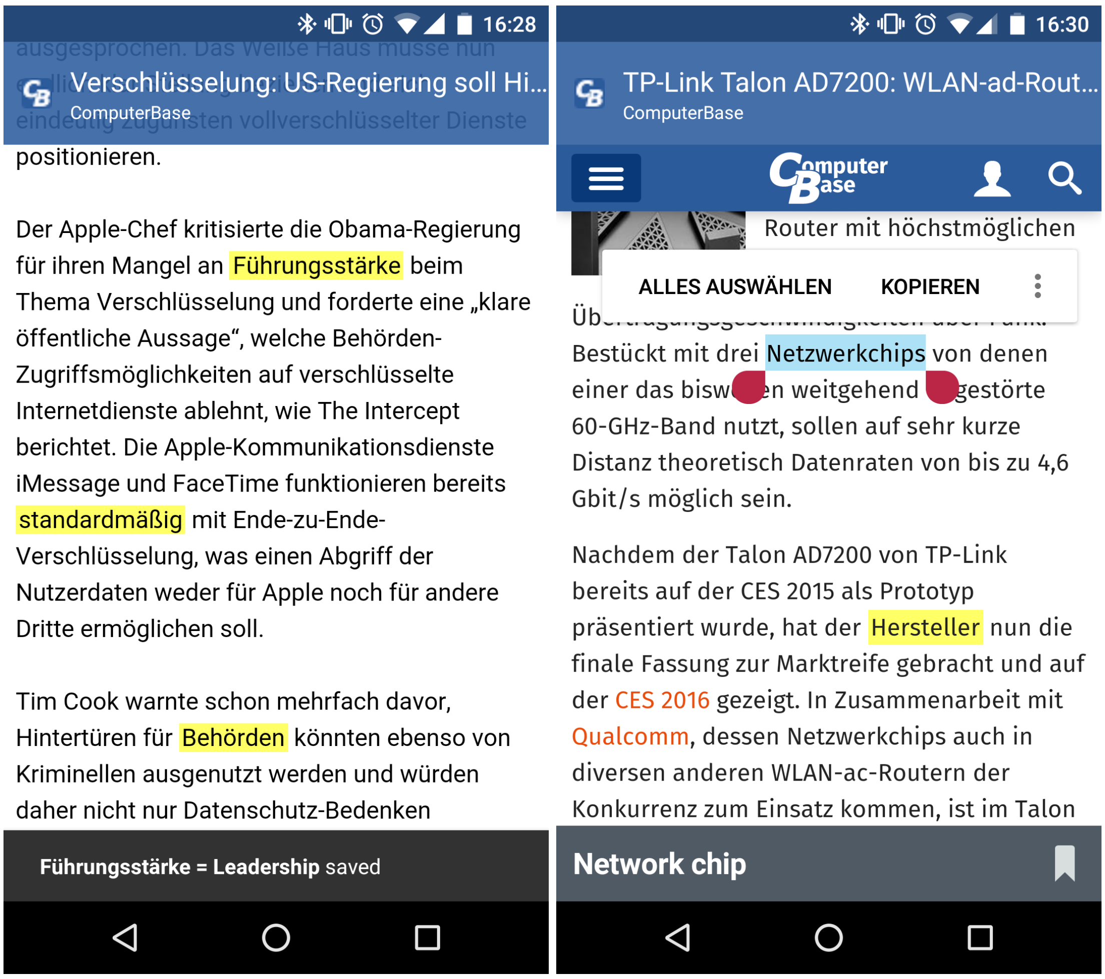
14.01 - 14:25 3

Opteron A1100: AMDs ARM-Prozesso...
Golem.de

Figure 4.3: The Feed Overview and the Feed Entry List fragments

Data Fragment

As opposed to the other fragments in the application, the DataFragment does not contain a user interface. Instead, its purpose is to store the references to important Java objects like the classes to connect with an API and the database, where only one instance can exist at the same time. This approach makes sure that these objects are stored independently of the activities, which by default get destroyed on screen rotation, instead it is managed by the FragmentManager.


16:28
Verschlüsselung: US-Regierung soll Hi...
ComputerBase
16:30
TP-Link Talon AD7200: WLAN-ad-Rout...
ComputerBase
ALLES AUSWÄHLEN
KOPIEREN

ausgesprochen. Das Weiße Haus müsse nun eindeutig zugunsten vollverschlüsselter Dienste positionieren.

Der Apple-Chef kritisierte die Obama-Regierung für ihren Mangel an Führungsstärke beim Thema Verschlüsselung und forderte eine „klare öffentliche Aussage", welche Behörden-Zugriffsmöglichkeiten auf verschlüsselte Internetdienste ablehnt, wie The Intercept berichtet. Die Apple-Kommunikationsdienste iMessage und FaceTime funktionieren bereits standardmäßig mit Ende-zu-Ende-Verschlüsselung, was einen Abgriff der Nutzerdaten weder für Apple noch für andere Dritte ermöglichen soll.

Tim Cook warnte schon mehrfach davor, Hintertüren für Behörden könnten ebenso von Kriminellen ausgenutzt werden und würden daher nicht nur Datenschutz-Bedenken

Führungsstärke = Leadership saved

Router mit höchstmöglichen Übertragungsgeschwindigkeiten über Funk. Bestückt mit drei Netzwerkchips von denen einer das bisw en weitgehend gestörte 60-GHz-Band nutzt, sollen auf sehr kurze Distanz theoretisch Datenraten von bis zu 4,6 Gbit/s möglich sein.

Nachdem der Talon AD7200 von TP-Link bereits auf der CES 2015 als Prototyp präsentiert wurde, hat der Hersteller nun die finale Fassung zur Marktreife gebracht und auf der CES 2016 gezeigt. In Zusammenarbeit mit Qualcomm, dessen Netzwerkchips auch in diversen anderen WLAN-ac-Routern der Konkurrenz zum Einsatz kommen, ist im Talon

Network chip

Figure 4.4: The Feed Entry fragment (with article website on the right side)

4.2.3 App Navigation

In addition to the activities and fragments, the user interface of the Zeeguu Reader consists of other elements, which allow a more comfortable navigation. Those are mostly basic Android UI patterns, which provides the benefit that the users are most likely already familiar with it from other applications. They are described in the following section.

The Action Bar

The top part of every activity is called the action bar, which is one of the most important design elements of an Android application. It includes the main navigation buttons like the back arrow and the button to open the navigation drawer, is responsible for displaying the title of the current location in the app and provides shortcut buttons for important actions like the synchronization.

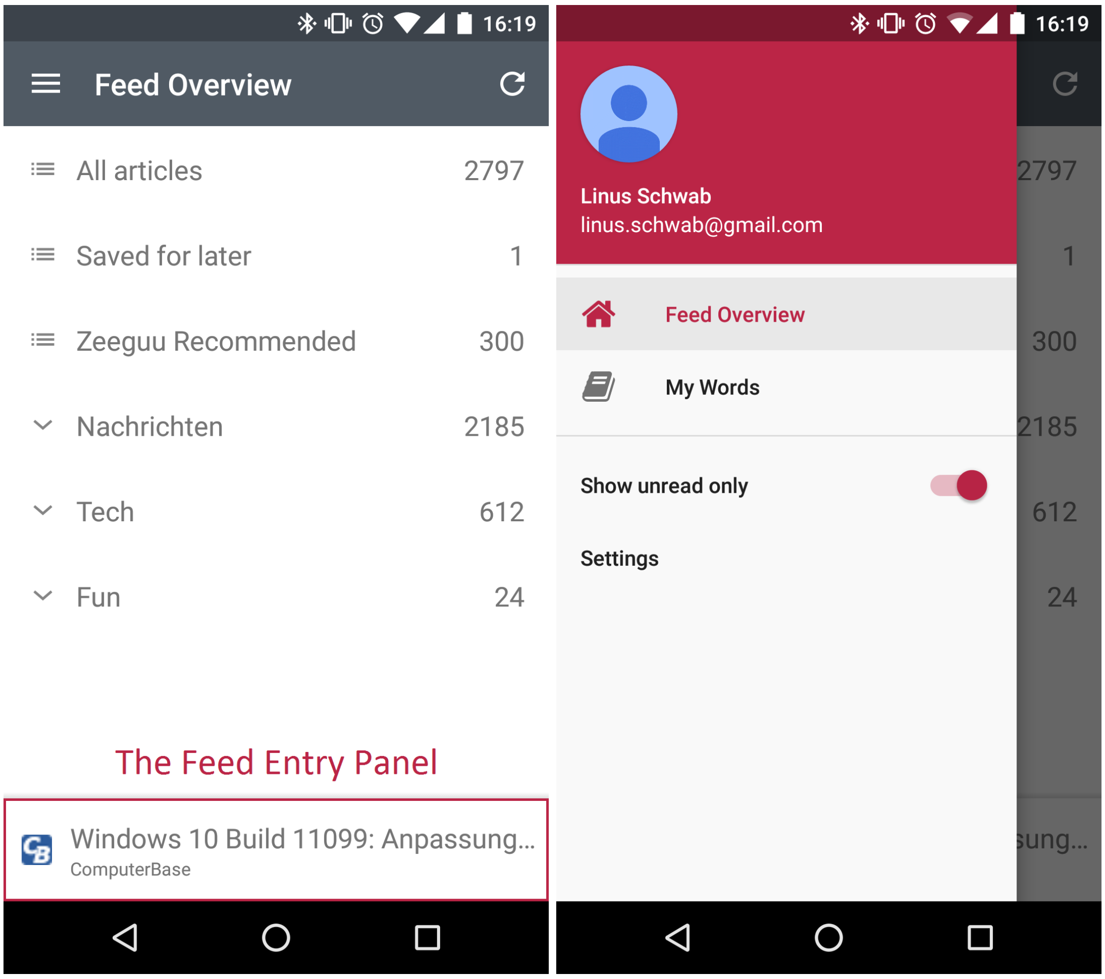


Figure 4.5: The action bar with the feed entry panel and the extended drawer menu

The Drawer Menu

Another basic Android navigation pattern is the drawer menu. The navigation drawer slides from the left side on top of the content and can be opened either with a touch gesture or by pressing the corresponding button in the action bar. It provides the navigation between the different parts of the app, displays the information of the currently logged in user like the name, email address and profile picture and gives access to quick settings.

It is accessible from almost anywhere in the application, so that the user always knows where he is.

The Feed Entry Panel

After the user selects a feed entry to read, it is displayed in a sliding feed entry panel. The panel has two states: While it is minimized, it keeps the most important information about the entry the user is currently reading visible and allows him to continue reading later from anywhere in the main activity. In the expanded state the content of the feed entry is displayed.
To make it possible to quickly browse through the entries, it contains a ViewPager. The ViewPager itself holds multiple FeedEntry fragments from the currently selected feed or category. By swiping left or right it is possible to directly display the previous or next entry.
Unlike the action bar or the navigation drawer, the sliding panel is not a default Android design pattern. A third party library was used for its implementation.[6]

[6]`https://github.com/umano/AndroidSlidingUpPanel`

4.3 Back-End Architecture

4.3.1 Zeeguu integration

4.3.1.1 API Overview

Zeeguu provides a REST API, which allows applications to communicate with it by using HTTP requests. This section provides an overview of the endpoints, which were used for the Zeeguu Reader application.[7] Except for the first two endpoints used for the authentication, all API calls must include the session id of the user.

Endpoint	**Description**
/add_user	Creates a new Zeeguu user account on the server
/session	Log in for existing users, returns session id
/translate	Translates a word or text to a specified language
/bookmark_with_context	Bookmarks a word, with its context and translation
/learned_and_native_language	Gets the native and learning language of the user
/native_language	Changes the users native language on the server
/learned_language	Changes the users learning language on the server
/get_difficulty_for_text	Difficulty calculation for the article recommender
/get_learnability_for_text	Learnability calculation for the article recommender
/get_content_from_url	Extracts the full article content for a URL

Table 4.1: The Zeeguu API endpoints used in this application

4.3.1.2 Implementation

The communication with the Zeeguu API is implemented in the ZeeguuConnectionManager class that is part of the Zeeguu Android Library. It contains the reference to the ZeeguuAccount class that manages the user data and stores them in the SharedPreferences that Android provides.
For the actual communication with the REST API, the Volley[8] library from Google is used. Volley provides benefits like automatic scheduling of requests, a request queue, asynchronous communication and includes simple response and error listeners. Therefore, it is possible to focus on the specific API requirements instead of the network communication.

[7] A full list of the available endpoints including their method, parameters and return values is available on Github at `https://github.com/mircealungu/Zeeguu-Web/blob/master/zeeguu/api/endpoints.py`

[8] `https://developer.android.com/training/volley/index.html`

Authentication

Before it is possible to use the Zeeguu API, the user must either register a new account or log into his existing one. For this, an email adress as username and a password are required, which can be exchanged for a session id. This session id is a mandatory parameter for each API call, it does not expire until the user logs out.

4.3.1.3 The Zeeguu WebView

The goal of the app is to allow the direct translation and bookmarking of unknown words in any web page. To enable this, we used an extended version of the Android WebViewClient: The Zeeguu Webview. The Webview is integrated in the ZeeguuWebviewFragment[9] that provides the UI for the translating and bookmarking. The ZeeguuWebviewFragment is used as the basis for the FeedEntryFragment in this application, as well as the BrowserFragment in the Zeeguu Translate app and will also be used in the Zeeguu Books app, which is currently in development.

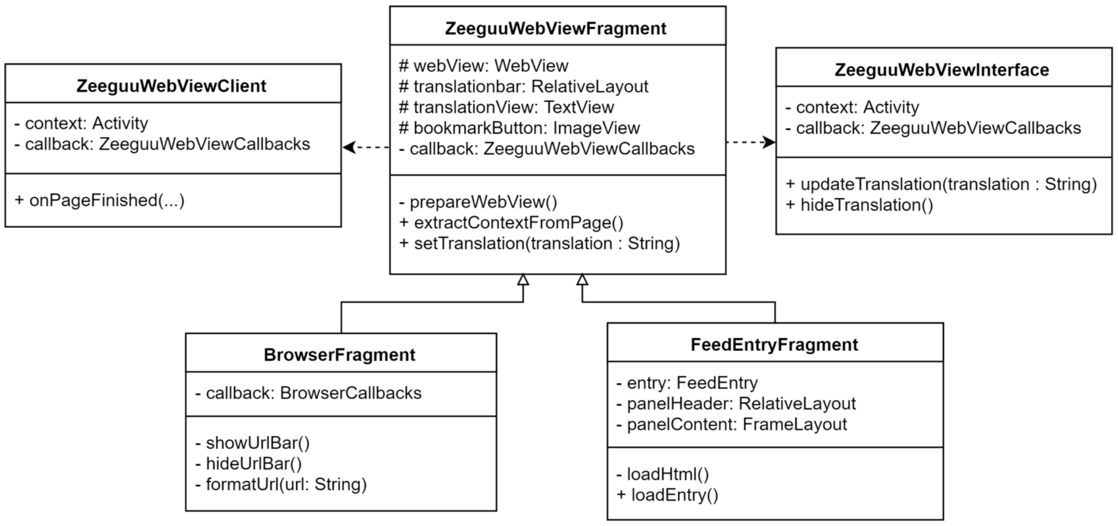

ZeeguuWebViewFragment
# webView: WebView
# translationbar: RelativeLayout
# translationView: TextView
# bookmarkButton: ImageView
- callback: ZeeguuWebViewCallbacks
- prepareWebView()
+ extractContextFromPage()
+ setTranslation(translation : String)
ZeeguuWebViewClient
- context: Activity
- callback: ZeeguuWebViewCallbacks
+ onPageFinished(...)
ZeeguuWebViewInterface
- context: Activity
- callback: ZeeguuWebViewCallbacks
+ updateTranslation(translation : String)
+ hideTranslation()
BrowserFragment
- callback: BrowserCallbacks
- showUrlBar()
- hideUrlBar()
- formatUrl(url: String)
FeedEntryFragment
- entry: FeedEntry
- panelHeader: RelativeLayout
- panelContent: FrameLayout
- loadHtml()
+ loadEntry()

Figure 4.6: UML diagram of the Zeeguu WebView[10]

The functionality included in the Zeeguu WebView was implemented using JavaScript, which is injected in every page and communicates with the fragment containing the

[9]See section 4.2.2 for additional information about the front end part included in the fragment. This section only covers the Zeeguu WebView itself.

[10]To keep the diagram readable, only the most important methods and class variables are shown. Additionally, all Android methods like onCreateView() are not listed.

WebView using a JavaScript to Java interface.[11] The process works in the following steps:

1. The user selects a word or part of a word
2. A selection listener extends any partially selected words[12] by getting the position of both selection markers in the root node of the HTML document.
   The selection expansion is done with the following function[13]:

```
function textOffsetOfSelection(node, offset) {
    var nodePosition;

    while (node.parentNode.parentNode != null) {
        if (node.parentNode.childNodes.length > 1) {
            nodePosition = findSelectedNodeInParent(node);
            offset = selectedNodeTextOffset(node, ..., offset);
        }
        node = node.parentNode;
    }

    return {
        "node": node,
        "offset": offset
    };
}

function findSelectedNodeInParent(node) {
    for (i = 0; i < node.parentNode.childNodes.length; i++) {
        if (node.isSameNode(node.parentNode.childNodes[i]))
            return i;
    }
}

function selectedNodeTextOffset(node, nodePosition, offset) {
    for (i = 0; i < nodePosition; i++) {
        var childNode = node.parentNode.childNodes[i]
        offset += $(childNode).text().length;
    }

    return offset;
}
```

[11] `https://developer.android.com/guide/webapps/webview.html#BindingJavaScript`

[12] The reason for this is to make it possible to directly translate the word once it is selected, otherwise each time another letter from the word is selected a new API call would be sent. Additionally it makes sure that the user does not accidentally translate (and bookmark) partial words.

[13] Most of the JavaScript code was moved to a CommonJS Github repository, to be able to also use it in the Zeeguu Chrome plugin: `https://github.com/mircealungu/Zeeguu-CommonJS`

With the exact position of the selection in the text content of the root node (the "offset" return value), it is easy to extend the selection in both directions until a word border is hit.

3. The word is then submitted to the fragment using the JavaScript interface. The ZeeguuWebviewFragment fragment submits the word to Zeeguu for the translation[14] and displays the translation bar with the translation.

4. If the user bookmarks the word, the context is extracted by extending the selected text to the full phrase in a similar way to the word extension. Finally, the collected information is submitted to the Zeeguu server and the bookmarked word is highlighted anywhere in the text of the page using a regex and injected CSS.

4.3.2 Feedly integration

4.3.2.1 Why use Feedly?

There were two options which were considered for the implementation of the RSS part of the app: Either to directly subscribe to the different RSS feeds, or to use a feed aggregator service like Feedly which offers some additional features.
The main reason for the decision to use Feedly was that it already has a large user base and became one of the bigger feed aggregator services after the shutdown of Google Reader.[15] It therefore allows existing Feedly users to directly import their subscriptions. It allows the synchronization of the user's subscription between different devices and provides a web interface, which makes it possible to use the already existing Zeeguu Chrome plugin to continue reading and learn the language on a computer.
Additionally, Feedly also offers features like the normalization of RSS and ATOM feeds into one JSON stream and supports the organization of feeds into categories.

4.3.2.2 API Overview

Feedly provides a REST JSON API to access the Feedly cloud, which uses HTTP requests for the communication between the server and the application. This section gives a short overview of the functions which were used for the development of the application.

[14]More specifically, the fragment uses a callback interface to communicate with the Activity and calls the translation function from the ZeeguuConnectionManager class, which returns the result back to the Activity and then back to the fragment.

[15]Feedly blog announcement

Endpoint	Description
/v3/auth/auth	Gets an authorization code using OAuth
/v3/auth/token	Exchanging the authorization code for a refresh token and an access token
/v3/profile	Get the Feedly profile of the user
/v3/categories	Get a list of all user categories
/v3/subscriptions	Get a list of all feeds the user subscribed to
/v3/streams/contents	Get a defined amount of feed entries
/v3/markers	Marks entries as read/unread on the feedly server
/v3/markers/reads	Returns a list of entries marked as read and unread
/v3/markers/tags	Returns a list of tagged and favorited entries

Table 4.2: The Feedly API endpoints used in this application

4.3.2.3 Implementation

The communication with the Feedly API is implemented in the FeedlyConnectionManager class. Similar to the implementation of the ZeeguuConnectionManager, Volley is used for the network communication with the Feedly server. Apart from that, the FeedlyConnectionManager contains the FeedlyAccount class, which is responsible for the management of the user data and provides the interface to the database and the shared preferences.[16] Additionally, the FeedlyResponseParser class is used to convert the JSON responses from the Feedly API into Java objects.
The FeedlyConnectionManager is managed by the BaseActivity mentioned previously in the user interface section and uses the FeedlyCallbacks interface to communicate with the activity, for example to update the UI with the new data after the synchronization is complete.

Authentication

At the first start of the application the user has to authenticate to connect with Feedly.[17] Feedly uses the OAuth 2.0 standard[18] for the authentication, which allows the user to use an already existing third-party account for the login process. Instead of an username and password, an access token issued by the authorization server (like Google or Facebook) is used.
The authentication starts with an authorization request displayed in a WebView, where the user authenticates with one of the available authorization servers. If the user granted

[16]The FeedlyAccount class will be explained in more detail in the database section

[17]It would also be possible to use the basic Feedly API features like getting all feed entries for a specific feed without authorization, however this was not implemented in this application.

[18]`http://oauth.net/2/`, `https://tools.ietf.org/html/rfc6749`

permission, Feedly returns an authorization code, which can be exchanged for the actual access token and an additional refresh token using the Feedly API. For security reasons the access token is only valid for a limited time; the refresh token is necessary to request a new one after it has expired.
For every API endpoint that requires authentication, the access token then needs to be submitted in the header of the request as authorization parameter.

Synchronization

After the authentication is complete, the download of new and updating of existing content is started. The content is synchronized in five steps:

1. Insure that all the necessary user information is complete and that the access token is not expired, otherwise a new one is requested using the refresh token.

2. Then the feed categories of the user are loaded from the Feedly server and are synchronized with the existing ones using the FeedlyAccount class. Basically this is done by checking for each category that exists in the Feedly cloud if it is already existing locally, update the data if it changed and delete categories that are stored locally but do not exist on the Feedly server anymore.

3. The next step is to download the feeds the user is subscribed to, which are saved as subscriptions on the Feedly server and contain the category ids. Therefore, the feed and category objects are linked during this step and the feeds are synchronized in a similar way to the categories.

4. After this is completed, the favicons and the latest feed entries are downloaded and assigned to the feeds.

5. The last step is to synchronize the read and marked entries with Feedly.

4.3.3 Database with Object-relational Mapping

To cache the local feed data efficiently, a database with object-relational mapping was used. There were several different frameworks available which specifically supported Android, for example OrmLite, SugarORM, GreenDAO or Active Android, to name a few. After spending some time evaluating the different frameworks, OrmLite seemed to be the best solution. The main reasons for this decision were that OrmLite was lightweight, allowed to minimize coupling by using data access objects (instead of having to extend from a database class for every object stored in the database and performing most operations directly on the object) and the framework was already a couple of years in development.

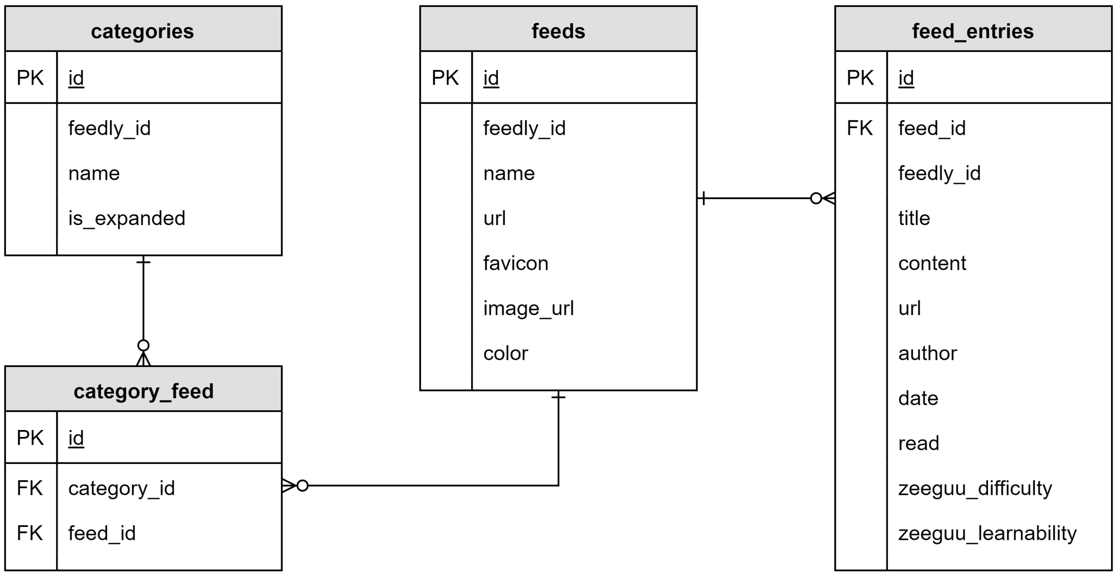


Figure 4.7: The database schema[19]

Database Schema

There are three main object types, which needed to be stored in the database: The categories, the feeds and the feed entries. The relation between these objects is that feeds can have any number of entries (1:N), and the feeds itself are organized into one or multiple categories. The categories can also have one or multiple feeds each, so here we have a N:M relation.

Unfortunately OrmLite does not directly support N:M relations, a intermediary object was necessary to make this possible. The CategoryFeed object is responsible for the relation between categories and feeds.

Integration

The integration of the database into the app consists of the following parts:

Stored objects: To decide which objects and which fields are stored in the database, annotations are used. Each persisted object needs an @DatabaseTable annotation with parameters like the table name, variables are annotated with the @Database-Field annotation. This means it is possible to decide which variables are saved and how. Nothing else needs to be changed or added to the class; the coupling is mini-

[19]In case you are wondering about the additional primary key in the category_feed association table, this is necessary due to an OrmLite constraint - it does not support multiple primary keys.

mal. OrmLite can also generate a unique id and set it on the object automatically (by using the SQLite autoincrement feature) with a simple annotation parameter.

DatabaseHelper: The DatabaseHelper class is mainly responsible for creating the database and providing the data access object (DAO) for each object type stored in the database. The reference to the DatabaseHelper object is stored in the BaseActivity class.

QueryHelper: This class contains prepared database queries which are used by the FeedlyAccount class.

FeedlyAccount: The FeedlyAccount is the bridge between the database and the Java objects. It keeps references to all the user data like the Category and Feed objects and the login data (stored in the Android preferences) and provides methods for all other classes to save and load the objects. This way the actual implementation of the database is hidden and the coupling is reduced. It is also responsible for the synchronization of the existing data with new data.

4.4 The Zeeguu Android Library

As the development of the Zeeguu Reader took place at the same time as the Zeeguu Translate application and certain basic features like the API communication needed to be implemented in both apps, we moved those parts to a common library.

Having a shared library for multiple Zeeguu applications on the android platform minimizes duplicated and redundant code. The library will also be used for the Zeeguu Books app, which is currently in development as another bachelors project.

Features

The Zeeguu library contains frontend and backend code necessary to implement the core Zeeguu functions in an application, which include the following:

- ZeeguuConnectionManager to connect to the Zeeguu API
- ZeeguuAccount to store the user information
- ZeeguuLoginDialog and ZeeguuLogoutDialog fragments to allow the user login/logout and account creation
- The MyWordsFragment to display the bookmarked words
- The ZeeguuWebViewFragment, which is used as parent class of webview fragments used in the app (for example the BrowserFragment and the FeedEntryFragment are subclasses)

5 Article Recommender

To help the user to find suitable articles to read and enjoy the reading as much as possible, the Zeeguu Reader application features an article recommender. The selected articles are presented in the “Zeeguu Recommended” category, sorted by difficulty in increasing order.
The goal is to make it easier for the user to learn and understand the meaning of unknown words. The better the user understands the words in the context, the easier it should be to infer the meaning of the unknown word.
For each article, the difficulty is visualized with a traffic light system – a green circle for a low difficulty, yellow for medium and red for hard to understand entries. Additionally, the number of words the user is currently learning that occur in the text is displayed.

5.1 Idea

The main idea for the article recommender is to analyze a text on a word-based level by estimating the difficulty for each word. Two metrics are used to estimate the difficulty of a word:

- Zeeguu estimates the difficulty of the words that a user has encountered in the past. The estimation is based on the ways in which the user has interacted with these words in the context of Zeeguu apps: if a user has encountered a word many times and never looked it up, it is more likely that the user knows the word and therefore it is easier to understand for him. The thesis of Karan Sethi provides more details about how this is done [5].

- The difficulty of words that the user has not encountered in the past in the context of Zeeguu apps is estimated based on word frequency lists, which Zeeguu stores for each language. The more common a word is in a language, the more likely it is that the user already encountered and learned it.

Based on the difficulty score for each word, the difficulty of the text is then calculated as the average or median of the individual word difficulties.

5.2 Algorithm

The algorithm used for the article recommender is divided in two components:

1. **Difficulty estimator**: the difficulty calculation of the text
2. **Learnability estimator**: the learnability, which is the percentage and number of words that the user is currently learning that occur in the text

Although initially intended as part of the Android application, both of these calculations were written in Python and implemented as REST API endpoints on the Zeeguu web platform.[1] The reasons for moving this functionality out of the Android app were:

1. To increase the reusability of the difficulty estimation components
2. And to be able to directly access the information from the Zeeguu database - the language knowledge information about the user, and the word frequency lists, which are stored as ranked words

As a lot of feed providers shorten their feed content for advertising reasons, it was necessary to fetch the full text content from the article url to be able to get comprehensive results. To achieve this, another endpoint which uses the Goose[2] content extractor library was implemented.

5.2.1 Difficulty Estimator

The following code is the algorithm used for the difficulty estimator. It basically works as explained in the idea section:

[1]https://github.com/mircealungu/Zeeguu-Web/blob/master/zeeguu/api/endpoints.py
[2]https://github.com/grangier/python-goose

```python
def text_difficulty(text, language, known_probabilities,
                    rank_boundary = REFERENCE_VOCABULARY_SIZE):

    word_difficulties = []

    # Calculate difficulty for each word
    words = split_words_from_text(text)

    for word in words:
        ranked_word = RankedWord.find_cache(word, language)
        difficulty = word_difficulty(known_probabilities,
                        True, rank_boundary, ranked_word, word)
        word_difficulties.append(difficulty)

    # Average difficulty for text
    difficulty_average = sum(word_difficulties)
                             / float(len(word_difficulties))

    # Median difficulty
    word_difficulties.sort()
    center = int(round(len(word_difficulties) / 2, 0))
    difficulty_median = word_difficulties[center]

    difficulty_scores = dict(
        score_median=difficulty_median,
        score_average=difficulty_average

    return difficulty_scores


def word_difficulty(known_probabilities, personalized,
                    rank_boundary, ranked_word, word):

    # Assume word is difficult and unknown
    estimated_difficulty = 1.0

    if not ranked_word:
        return estimated_difficulty

    # Check if the user knows the word
    try:
        # Value between 0 (unknown) and 1 (known)
        known_probability = known_probabilities[word]
    except KeyError:
        known_probability = None

    if personalized and known_probability is not None:
        estimated_difficulty -= float(known_probability)
    elif ranked_word.rank <= rank_boundary:
        # Value between 0 (rare) and 1 (frequent)
        word_frequency = (rank_boundary - (
            ranked_word.rank - 1)) / rank_boundary
        estimated_difficulty -= word_frequency
    return estimated_difficulty
```

The difficulty value for a text can theoretically go from 0 (easy) to 1 (hard). In reality however, values between about 0.2 and 0.5 were observed for normal texts with the average method.[3] As is visible in the code above, the value for each word is either calculated by subtracting the known word probability or by subtracting the normalized word frequency. If a word is unknown and is not in the word frequency list, it receives the highest difficulty value, a 1.0.

5.2.2 Learnability Estimator

To calculate the learnability of a text, the learnability estimator first caches the bookmarked words of the user that the user is currently learning, and then checks for each word in the text if it is currently being learned. Then, both the count and percentage of learned words in the text are returned:

```
def words_being_learned(self, language):
   # Get the words the user is currently learning
   words_learning = {}
   bookmarks = Bookmark.find_by_specific_user(self.user)
   for bookmark in bookmarks:
      learning = not bookmark.check_is_latest_outcome_too_easy()
      user_word = bookmark.origin
      if learning and user_word.language == language:
         words_learning[user_word.word] = user_word.word
   return words_learning

def text_learnability(text, words_learning):
    # Calculate learnability
    words = split_words_from_text(text['content'])
    words_learnability = []
    for word in words:
        if word in words_learning:
            words_learnability.append(word)
    count = len(words_learnability)
    learnability = count / float(len(words))
    return count, learnability
```

[3]For a rating of 0.0, the text would have to consist only of the most common word of a language, and for a 1.0 it could only contain the least common word or words that do not occur in the list of the 10’000 most frequent ones.

5.3 Performance

Initially, all necessary data was directly accessed from the database for every word inside the loop that iterates over the whole text. The calculation of both the difficulty and the learnability values took too long this way. Especially since in some situations it was necessary to analyze about 500 texts per API call, we had to drastically improve the performance for both algorithms.
The following table displays the execution time (in seconds) for both the initial and improved version of the difficulty and learnability calculation with different numbers of analyzed texts. The analysis was performed with 10 randomly selected articles from the newspaper “Der Bund”[4] with an average of 411.3 words and used a copy of the Zeeguu database with the real data of the most active Zeeguu user to get as realistic results as possible.[5] The measurements with a star were extrapolated, as the runtime behaviour of the initial versions was clearly linear.

Number of texts	**Difficulty (initial / improved)**		**Learnability (initial / improved)**	
10	21.35	1.03	2.74	2.25
100	213.5*	1.25	27.4*	2.31
1000	2135*	3.33	274*	2.53

Table 5.1: Performance measurements for difficulty and learnability, time in seconds

The improvements were achieved by caching all necessary data in Python dictionaries, a hashtable based data structure, and only performing key checks instead of accessing the actual data whenever possible.
For the difficulty algorithm, the ranked words (frequency lists) for all languages are cached on server startup and always kept in the memory. Additionally, the known word probabilities for the specific user are cached at the beginning of the API call, which takes about 1.01 seconds for the test user mentioned before. The actual analysis of the text inside the loop only takes 0.002 seconds on average per text for the used articles.
The learnability algorithm now caches the words the user is currently learning at the beginning of the API call in a dictionary. This takes roughly 2.20 seconds in this case, the loop for the text analysis taking 0.0002 seconds per text on average.

[4] `http://www.derbund.ch/`

[5] The analysis was performed on a Windows computer with an Intel Core i5-4200U processor. XAMPP with Apache was used to run the Zeeguu server

5.4 Limitations

The current version of the article recommender has several limitations, which could be improved in future versions to get better recommendations:

- Names, Brands, Countries, Cities etc. always receive the highest difficulty score. Currently we assume that they are about equally distributed across different articles. A possible solution for this would be to use a database for each type and ignore them, or the other way around, create a database of valid words for each language.
- In some articles there are quotes or some words in another language, which results in a higher difficulty score. At least for entire paragraphs, it should not be hard to detect and ignore them.
- Currently, the article recommender only evaluates single words and does not consider metrics like the average phrase or word length, singular or plural forms and other grammar properties.
- At the moment, we do not stem words to discover inflexions and variations.

6 Evaluating the Usability of the Application

To evaluate the application, especially the user interface, multiple usability tests were conducted at different points in the development cycle. This chapter summarizes the results and explains the methodology of the tests.[1]

6.1 Usability Tests

First Usability Tests

The first usability tests were conducted in an early stage of the app development, together with the Zeeguu Translate application and the Chrome plugin. At this time, only the Zeeguu WebView was ready to use. The application was presented to the participants as a simple web browser. For this, it was only necessary to add a simple URL bar and a menu with some basic navigation buttons.

The participants received detailed instructions about the actions they needed to perform. If they already had a Zeeguu account, they could use their own, otherwise a testing account was provided. The instructions for each participant were the following:

1. Log into your Zeeguu account (or use the testing account provided)

[1]The detailed notes about the usability tests are available in appendix B.

2. Search for a word in the browser
3. Browse to a site of your choice in the language you are learning
4. Translate a couple of words
5. Bookmark a word
6. Log out or explore the app a bit longer

Overall, the users did not have many problems using the app and they provided valuable feedback and suggestions such as adding a loading bar to the WebView, a better visible notification after a word has been bookmarked, and fixing a bug in the word highlighting regex. Based on this, several improvements were done.

Another suggestion was to merge both apps into one, as the Zeeguu Reader only consisted of a single browser fragment at this time. As a result, the browser fragment that was initially only planned to use for testing purposes was included into the Zeeguu Android Library and is used in the Zeeguu Translate application.

Second Usability Tests

For the second usability test the users did not get a lot of small goals they needed to achieve. Based on the experience from the first round as browser, the better alternative seemed to be one large goal instead and some small hints if necessary.
The following task description was used for the second tests:

You are currently learning a new language and want to combine this with your daily activities by reading news. For that, you have the possibility to test the new Android app Zeeguu Reader. First, you will need to log in to your Feedly account, after that you can read your favorite news sources and translate & bookmark the words you don't know yet.

Before the users participated in the usability test, they received a short description of Zeeguu. As participants users with an existing Feedly account were chosen to keep everything as simple as possible, and due to the non-final state of the app they needed to transfer their subscriptions to the development version of Feedly before the start. They received help and detailed instructions for this step, as it will not be necessary anymore in the final app.
After this step was completed, the app was installed on the phone of the user or if they did not have an Android phone, they were provided with a Google Nexus 5 for the test. Neither of the test users had a Zeeguu account yet, so they needed to create it during the

usability test directly in the app.

The second iteration of the usability tests mainly helped to discover a couple of bugs in the application and led to some small usability fixes. The participants did not have many problems in using the app in general. However, there was a problem with the Zeeguu bookmarking API during the tests, which resulted in not being able to save words in some language combinations. The problem was reported and has been fixed.

7 Evaluating the Article Recommender

To evaluate if the article recommender can actually provide the user with helpful recommendations, a qualitative case study was conducted. As the Zeeguu platform is still in development and therefore does not have many active users with data available about their language knowledge, it was decided to focus on one participant instead of a quantitative study with multiple participants.

7.1 Case Study

7.1.1 About the User

The case study was performed with the most active Zeeguu user, *Dr. Mircea Lungu*. A couple of years ago, he started learning German and is using Zeeguu on a regular basis since June 2014 to support this process. During this time, new features were constantly added, including the Zeeguu Quantifier on the 20th August 2015, which introduced the known word probability used for the personalization of the recommended articles.

7.1.2 Evaluating the Article Recommender

7.1.2.1 Execution of the Study

Before the beginning of the experiment, there were 9 articles from different sites and difficulty levels selected for the user to read. This was done to make sure that the participant did not know the supposed difficulty level of the text before reading them, as

this could unknowingly have an influence on the perceived difficulty of the text.
For the selection of the articles, the exported feeds from the Feedly account of *Dr. Mircea Lungu* were used, which included a variety of topics like different newspapers from Switzerland and Germany and technology websites, in combination with his Zeeguu account. The articles were then manually selected (as randomly as possible) from the "Zeeguu Recommended" category, which was modified for the experiment to additionaly include the more difficult articles. 3 articles from each difficulty level were chosen, with different learnability values.
After the articles were selected, the article urls were sent in a random order by email to the participant. To be able to record the screen and his voice, the reading was done using the Zeeguu Chrome plugin. He was instructed to use the "think aloud"-method [7], which allows us to recognize the perceived difficulty and how well the text is understood in general.

7.1.2.2 Analysis

The first part of the analysis consisted of collecting the text characteristics: the total number of words and the number of characters (without spaces). Next, the video recordings were analyzed, during this metrics like the number of looked up, bookmarked words and the number of words that were not understood after the translation were collected. This led to the following criteria, which were used to assess the actual difficulty of the text for the user:

- **Understanding**: The perceived understanding based on the comments and reactions of the participant during the reading of the text. It was measured in numbers from 1 (hard) to 5 (easy to understand).
- **Time per character**: The reading speed. This was measured in seconds per character instead of per word, because the average word length was different for the texts that were used.
- **Percentage of words looked up**: The percentage of words that the user did not understand or needed to confirm if he understood them correctly.
- **Percentage of words bookmarked**: The percentage of words that were bookmarked after they were looked up and were therefore new to the user.

7.1.3 Results

The following table 7.1 presents the detailed results for all 9 articles.[1] The first column lists the difficulty values calculated by the article recommender (using the average method).[2] The other 4 columns contain the metrics to assess the actual difficulty for the reader, which were introduced in the previous section.

	Difficulty	**Understanding**	**Time per char**	**Looked up**	**Bookmarked**
1	0.22	4.50	0.24 s	7.77 %	6.25 %
2	0.26	4.50	0.17 s	4.52 %	3.87 %
3	0.23	4.50	0.23 s	7.27 %	5.45 %
4	0.30	1.50	0.23 s	8.86 %	7.91 %
5	0.32	4.00	0.19 s	4.73 %	4.14 %
6	0.34	4.50	0.26 s	9.65 %	8.38 %
7	0.43	1.00	0.31 s	19.38 %	13.24 %
8	0.41	4.00	0.18 s	5.78 %	5.53 %
9	0.49	3.00	0.33 s	8.21 %	5.00 %

Table 7.1: The difficulty score compared to the actual difficulty of the articles

In most cases, the difficulty value calculated by the article recommender and the actual difficulty are approximately correlated. However, there are two articles that seem to be in the wrong difficulty category:
Article number 4 was noticeably harder to read for the user than indicated by the difficulty score. The reason for this was not clear. Another wrong attribution was article number 8, which happened because of a mistake in the word separation regex that the Zeeguu platform used – numbers were recognized as words. Because this article contained an unusually high amount of numbers that were each evaluated as words with the score 1.0, the text received a much higher difficulty score than it should.[3]

[1]The sources of the articles are listed in appendix B.2.1, the detailed article characteristics can be found in section B.2.2.

[2]For more information about the algorithm, see section 5.2

[3]Apart from this case, this did not have an influence on the order of the difficulty scores for the articles, as the amount of numbers is usually much lower and seems to be more or less equally distributed between the articles.

As the articles were divided in three difficulty groups[4], the results are aggregated per group in the next table 7.2 to get a clearer picture:

Difficulty	Understanding	Time per char	Looked up	Bookmarked
0.24	4.50	0.21 s	6.52 %	5.19 %
0.32	3.33	0.23 s	7.75 %	6.81 %
0.44	2.66	0.28 s	11.13 %	7.92 %

Table 7.2: The average values per difficulty group

Here, the average difficulty scores per group are clearly correlated for the average of all 4 metrics to measure the actual difficulty.

7.1.3.1 Summary

In general, the difficulty score roughly matches with the actual difficulty for the user, and the aggregated table shows a clear correlation per difficulty group. This indicates that the recommendations should be useful for the user. This was also the general impression of Lungu when he read the recommended articles outside of the case study. However, this case study does not prove that the article recommender works, especially since it involved only one participant. A larger study will be required for this.

[4]Articles number 1-3 were rated easy, 4-6 medium and 7-9 hard

7.1.4 Additional Insights

7.1.4.1 Personalization Influence

The following graphs compare the difficulty scores for the articles from the case study with and without enabled personalization, which refers to the KnownWordProbability as described in chapter 5. As visible, it only has a small influence on the score and does not really change the order of the difficulty scores for those articles. This is most likely due to the fact that the KnownWordProbability is only available for a small percentage of the words from a text, even for the most active Zeeguu user.

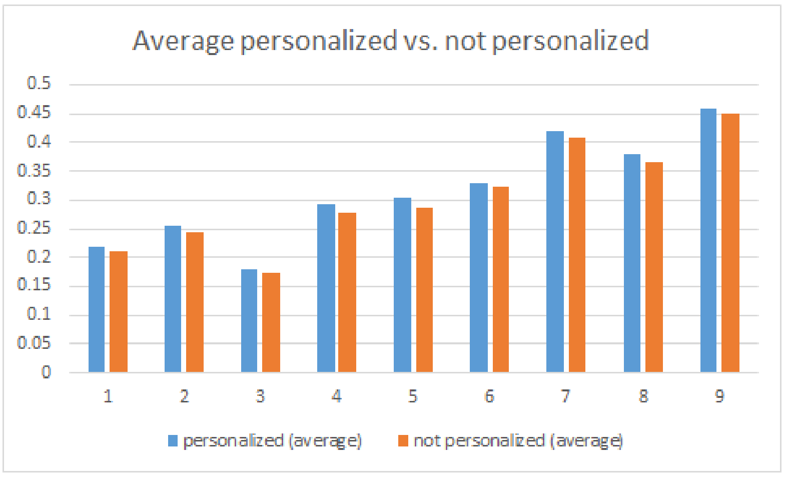


Figure 7.1: Personalization influence (average)

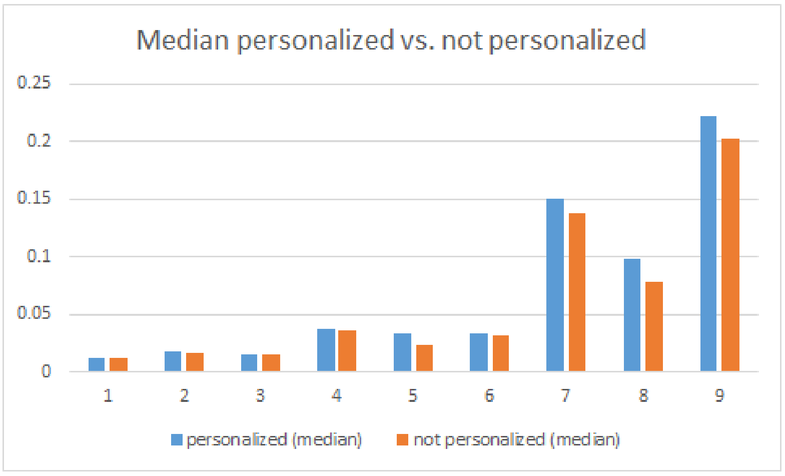


Figure 7.2: Personalization influence (median)

7.1.4.2 Difficulty Score Distribution

This section provides a histogram of the average difficulty score distribution for the words of the texts from the case study. The intervals are always 0.1 score points wide, the number in the figure indicates the starting point.
It is clearly visible that almost 60% of the words receive a score between 0.0 and 0.1, which indicates that they are in the 1000 most common words of the word frequency list that was used. Most of the remaining words are in the interval between 0.9 and 1.0, as all words for which no KnownWordProbability is available and that are not in the word frequency list (which contains the 10'000 most frequent words) receive the maximum score of 1.0.

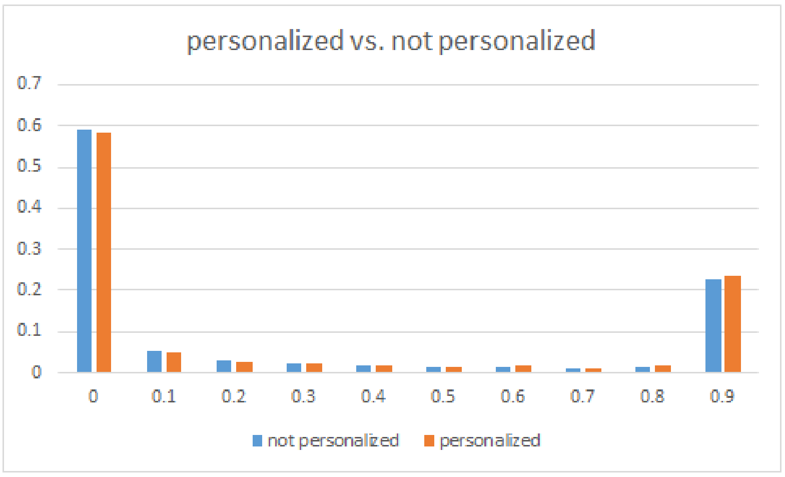


Figure 7.3: Histogram of the difficulty score distribution

8 Conclusion and Future Work

8.1 Conclusion

The Zeeguu Reader makes it possible to learn a new language in a comfortable way on Android devices. By using RSS feeds, it allows one to read texts according to the personal preferences, and in the process helps to improve the language knowledge with the extensive reading [1] approach.

The main contributions of this thesis are the following:

1. An application which faciliates the reading of texts in a foreign language by providing instant translations and bookmarking unknown words. With the included Feedly synchronization, it is even possible to continue reading exactly where the user left off on any computer in the Feedly web interface in combination with the already existing Zeeguu Chrome plugin.
2. An article recommender that tries to assess the difficulty of the feed entries in order to provide suitable recommendations.
   The algorithms for the article recommender are implemented in the Zeeguu server as API endpoints, which will enable future applications in the Zeeguu ecossystem to use them.
3. A series of usability studies, which show that the Zeeguu Reader provides all the planned features, even though it still offers much room for extensions and improvements, as will be mentioned in the Future Work section.

4. A case study in which the article recommender algorithm is evaluated based on one Zeeguu user. The evaluation showed that the difficulty score roughly matched with the actual difficulty for the user, with a clear correlation per difficulty group. The results indicate that the recommendations should be useful for the user.

8.2 Personal Lessons Learned

During the project, there were a lot of new things that I learned. I consider the following points especially important:

- An ORM is a comfortable way to implement a database: During the development of this application it was the first time for me that I implemented an ORM. The initial setup was easier than expected, altough because of the fact that OrmLite does not directly support M:N relationships, some additional work and planning were required. In the first Android application that we developed as part of the "Introduction to Software Engineering" course, it was much more work to update the SQL queries etc. needed for the database with every change.
- Prioritize planned features: Even though all required features that were planned at the beginning of the project were successfully included, my personal plan was to include even more. Those include features that would have improved the usability of the app, as well as the points that are listed in the Future Work section.
- Gained experience in new programming languages: During the project, it was necessary to use JavaScript to implement the Zeeguu features into the Android WebView, as the provided interface methods did not offer access to all the needed data. At first this was a bit of a challenge, as I did not know enough about the possibilities JavaScript offers, especially that there was a JavaScript to Java interface available on Android. Additionally, I decided to implement the article recommender algorithm on the Zeeguu server (instead of in the application), which required the use of Python.
- Performance optimization experience: For me, during this project it was the first time that I really had to improve the performance of an algorithm. This was necessary for the article recommender, as it needs to analyze a high amount of texts in a short amount of time. The exact procedure and results of the performance optimization are explained in section 5.3.

8.3 Future Work

Option without a Feedly Account

Currently, it is necessary to use a Feedly account in order to use the application. Even though it is easy to create a new one directly in the app with an existing Google, Facebook, Twitter or Microsoft account, it would be more user friendly to make this optional. A possible solution for this would be to allow the user to select the topics he is interested in at the first start of the app and then provide default feeds for the chosen language.

Manage Subscriptions directly in App

Initially, it was planned to be able to directly manage the subscriptions in the app. As it is possible to do this in the Feedly web interface, it was decided to not include this feature in this version of the application.
Instead of including a half-baked version of the subscription management, which would have required to implement additional Feedly API endpoints and synchronizing methods, the focus was put on the article recommender instead.

Extend the Application Settings

The settings section of the application is currently limited. It is only possible to manage the Feedly and Zeeguu account and some basic settings which are related to them. The SettingsActivity was specifically designed so that it is easy to add new setting categories, while keeping the code clean and organized. For example it would be possible to allow the user to customize the font size, define the order of the entries (newest or oldest first) or similar customizations.

Improving the Article Recommender

Even though the article recommender already delivers usable and helpful recommendations in most cases according to the case study, there are still a lot of ways in which it could be improved, as was already described in the limitations section (see 5.4).
This would entail more testing and experimenting, which could for example be done by including a user feedback button to rate the recommendations and then analyzing the gathered data. Another possibility would be to consult a linguist for potential improvements.

Bibliography

[1] Rob Waring. "The inescapable case for extensive reading". `http://www.robwaring.org/er/what_and_why/er_is_vital.htm`

[2] Stephen Krashen. "We Acquire Vocabulary and Spelling by Reading: Additional Evidence for the Input Hypothesis". The Modern Language Journal Vol 73 Issue 4, 1989.

[3] Simon Marti. "A Platform for Second Language Acquisition Through Free Reading and Repetition". University of Berne, 2013.

[4] Pascal Giehl. "The Zeeguu Translate Application". University of Berne, 2015.

[5] Karan Sethi. "Modelling the Acquisition of Natural Language". University of Berne, 2015.

[6] Stephen Krashen. "False Claims About Phonemic Awareness, Phonics, Skills vs. Whole Language, and Recreational Reading". `http://www.nochildleft.com/2003/may03reading.html`, 2003.

[7] Clayton Lewis. "Using the "Thinking Aloud" Method In Cognitive Interface Design". IBM, 1982.

A
Anleitung zu wissenschaftlichen Arbeiten

This chapter describes how to implement a database for an Android application with object-relational mapping. Android uses SQLite as database technology, where the data is normally stored with manually created SQL queries in the relational database tables, as explained in the official tutorial.[1] An ORM provides an easy way to persist objects in the database, mostly without the need for complex SQL query strings.
This tutorial explains how to set up and adapt the ORM used for the development of the Zeeguu Reader application, OrmLite.[2]

A.1 Introduction to OrmLite

OrmLite is a lightweight ORM system developed for Java projects that supports various database types and has been adapted for Android. Its goal is to provide "simple, lightweight functionality for persisting Java objects to SQL databases while avoiding the complexity and overhead of more standard ORM packages".[3]
While it is easy to implement simple databases with OrmLite, it does have some limitations. Those include the missing direct support for many-to-many relations. It is necessary to manually create a Java class for the association table, as will be explained later.

[1] `https://developer.android.com/training/basics/data-storage/databases.html`
[2] `http://ormlite.com/`
[3] `http://ormlite.com/simple_orm_java.shtml`

A.2 Mark Classes to be Persisted

OrmLite uses simple Annotations to mark the classes, whose objects need to be stored in the database. Basically, it is only necessary to add an *@DatabaseTable* annotation with the table name as parameter to the class, and an *@DatabaseField* annotation to each variable that needs to be persisted with optional parameters like the column name or if the value can be null. The primary key is defined with the “id = true” parameter. With the “generatedId = true” option it is also possible to let OrmLite automatically generate an incremental number to be used primary key. A full list of the available parameters can be found in the official documentation.[4]
The following example demonstrates the basic structure necessary for a persisted class:

```
@DatabaseTable(tableName = "categories")
public class Category {

    // Id is generated by the database and set on the object
    @DatabaseField(generatedId = true)
    private int id;

    @DatabaseField(columnName = "name", canBeNull = false)
    private String name;

    public Category() {
        // All persisted classes must define a no-arg constructor
    }

    ...
}
```

It is not necessary to inherit from a given class or use generated code for the persisted classes, as it is done in some other ORM libraries for Android.

A.3 Define a Database Schema

After the classes to be persisted are chosen, it is necessary to define the relations between them. This is also done with annotations:

[4]Adding OrmLite Annotations: `http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite_2.html#Local-Annotations`

A.3.1 One-to-one Relations

For one-to-one relations, it is only necessary to create a class variable for the referenced object in both classes of the relation and add an *@DatabaseField* annotation with the parameter “foreign = true”. OrmLite then automatically stores the id of the referenced object in this database field, therefore it is recommended to indicate this with an appropriate column name using the annotation parameter “columnName”.
The foreign object is automatically loaded when the object is retrieved from the database if the “foreignAutoRefresh = true” parameter is set, otherwise only the primary key is set, with default values for the other fields.
In the following example, one side of the one-to-one relation is demonstrated. Obviously it is necessary to do the same in the referenced object:[5]

```
@DatabaseTable(tableName = "feed_entries")
public class FeedEntry {

    // Id is generated by the database and set on the object
    @DatabaseField(generatedId = true)
    private int id;

    @DatabaseField(foreign = true, foreignAutoRefresh = true,
                   columnName = "feed_id")
    private Feed feed;

    ...
}
```

A.3.2 One-to-many Relations

One-to-many relations are supported with the “ForeignCollection” class that needs to be defined as a class variable with an *@ForeignCollectionField* annotation in the persisted class on the many side of the relation. OrmLite then automatically loads all the referenced objects as soon as the collection is accessed by default. It is also possible to directly load the foreign objects as soon as the object is loaded from the database, by setting the “eager” annotation parameter to false.
For the one side of the relation, the same *@DatabaseField* annotation with the parameter “foreign = true” as in the one-to-one relation is used.
The many side of the one-to-many relation is shown in the next example:

[5]In this case, the Feed object would also require a FeedEntry variable with a similar *@DatabaseField* annotation. Please note that in the actual project of this thesis the real relation type between the Feed and FeedEntry objects actually is one-to-many, however as the one-side is identical in both cases, it was still used to keep the examples familiar.

```
@DatabaseTable(tableName = "feeds")
public class Feed {

    // Id is generated by the database and set on the object
    @DatabaseField(generatedId = true)
    private int id;

    @DatabaseField(columnName = "name")
    private String name;


    // If eager is set to false, the collection is considered "lazy"
    @ForeignCollectionField(eager = false,
       orderColumnName = "date", orderAscending = false)
    private ForeignCollection<FeedEntry> entries;

    ...
}
```

A.3.3 Many-to-many Relations

Many-to-many relations require more work to implement, as they are not directly supported by OrmLite. It is therefore necessary to manually create a separate class for the necessary association table, which needs to contain foreign references to both the linked objects, annotated with the *@DatabaseField* annotation and the parameter “foreign = true”. As OrmLite does not support multiple primary keys, an additional unique primary key is required. It is not necessary to add any additional annotations to the classes of the objects involved in the many-to-many relation directly.
The example below demonstrates the structure of the object used for the association table:

```
@DatabaseTable(tableName = "category_feed")
public class CategoryFeed {

    // Generated id as PK, ormlite does not support multiple PKs
    @DatabaseField(generatedId = true)
    private int id;

    // Foreign object, just stores the id from the Category object
    @DatabaseField(foreign = true, columnName = "category_id")
    private Category category;

    // Foreign object, just stores the id from the Feed object
    @DatabaseField(foreign = true, columnName = "feed_id")
    private Feed feed;

    ...
}
```

In order to save a relation between two objects of the many-to-many relation, an object from the association table with the two objects to be linked needs to be instantiated and saved to the database. After this is done, the helper object is not needed anymore and can get garbage collected.
To load the objects from the many-to-many relation, it is necessary to manually query them. For this, it is possible to create a prepared query that can be accessed with a helper method. Either this is done directly after the objects are loaded from the database and then the foreign objects are manually assigned to the objects from the relation, or the helper method can be called as soon as the foreign objects are needed.

A.4 Manage Persisted Objects

OrmLite uses the Data Access Pattern (DAO) to access and update the objects stored in the database. For each class, whose objects are persisted in the database, there is one DAO that manages the objects from that class.
In addition to creating, updating and deleting objects in the database table, the DAOs make it possible to get all objects from the class stored in the database (*dao.queryForAll()*) and provide query helper methods that can for example match by id or a field value.[6]

```
public void saveFeedEntry(FeedEntry entry) {
    try {
        if (entry.getId() == 0)
            feedEntryDao.create(entry);
        else {
            feedEntryDao.update(entry);
        }
    }
    catch (SQLException e) {
        ...
    }
}
```

The DAO pattern has the advantage that the coupling between the ORM and the rest of the application is as minimal as possible, as the database code is isolated in the DAOs. This makes it possible to switch to a different database or ORM solution if necessary.

[6]A full list of the methods provided by the DAO classes is available here: `http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite_5.html#DAO-Methods`

Additionally, this allows the classes to have their own hierarchy, as it is not necessary to extend an ORM class. However, it is also possible to implement another ORM pattern, where the objects perform the database operations on themselves instead of using separate DAO objects. This requires to inherit from the *BaseDaoEnabled* class.[7]

A.5 Creating the Database Helper

To set up the database and create the database tables, it is necessary to create a database helper class, which needs to extend an OrmLite class called *OrmLiteSqliteOpenHelper*. Additionally, it provides and manages the data access objects.
Basically, the following is absolutely necessary:[8]

- A constructor that calls the parent implementation, which needs a database name String, a database version integer and the Android application context as parameters
- A custom implementation of the *onCreate()* method, where it is necessary to create the database table for each persisted class
- Getter methods that provide the DAOs for each object type
- A custom implementation of the *close()* method that needs to call the parent implementation and clear any cached DAOs

A.5.1 Updating the Database Schema

The database helper parent class additionally provides a method to handle schema updates. The *onUpgrade(..., int oldVersion, int newVersion)* method gets automatically called if the version number of the database changes and makes it easy to define migration code for each update by overriding the default implementation.

A.6 Performing SQL Queries

Instead of having to use SQL query strings, ORMLite makes it possible to perform simple queries directly with DAO helper methods, and additionally provides a QueryHelper class for more complex queries. The following example demonstrates how to perform a custom query on a DAO object:

[7] More details about this pattern can be found here: `http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite_2.html#DAO-Enabled-Objects`

[8] A full example of a database helper implementation is available on the “ormlite-examples” Github repository: `https://github.com/j256/ormlite-examples/blob/master/android/HelloAndroidNoBase/src/com/example/hellonobase/DatabaseHelper.java`

```
public List<FeedEntry> getRecommendedEntries(float maxDifficulty) {
    try {
        return feedEntryDao.queryBuilder()
            .orderBy("zeeguu_difficulty", true) // ascending
            .where().between("zeeguu_difficulty", 0, maxDifficulty)
            .query();
    }
    catch (SQLException e) {
        ...
    }
}
```

This query returns all feed entries, ordered by the estimated difficulty from the Article Recommender, that are below a certain difficulty level.

B
Validation Notes

This chapter contains the detailed notes from the observations during the usability tests and the case study that were done to evaluate the application. The details of the procedure and the implications are described in chapters 6 and 7.

B.1 Usability Tests

B.1.1 First Usability Tests

Haidar Osman (29.04.2015)

- Instructions (about what to do with the app) were a bit unclear sometimes
- Translation etc. was more or less clear to him, most likely because its similar to the Chrome plugin he already used
- Do not save all the words the user translates here

Thomas Steinmann (29.04.2015)

- Word not highlighted when a special sign (”) was in front
- Bookmark a word: remove and add highlight again (toggle)
- Clear feedback that a word has been bookmarked
- Logout button not found
- Look that the design is about the same in both apps
- Back button in settings not working
- One app instead of two

B.1.2 Second Usability Tests

Andreas Hohler (22.10.15)

No Android phone, used the Nexus 5

- The user had no problems logging into his Feedly account
- He understood that it is syncing, but he didnt know how far the progress was
- Tried to access all articles while there was no content yet
- He realized by mistake that it is possible to swipe to the next article
- After that, when translating a word by selecting it, he almost instantly understood how to create the necessary Zeeguu account
- Wrong language was set (currently there is no possibility to set the language during account creation over the API), went to the settings to change his native/learning language and set both correctly
- He accidentally bookmarked a word and tried to delete it by bookmarking it again
- Shortly after, he found the word list and was able to delete it there
- He found the unread switch
- He had no problems to understand how the panel works
- It was not perfectly clear for him which articles were already read at the beginning (currently they are in a lighter font color), but after some time he understood it
- Bookmarking didnt work anymore after the user deleted his words (seems to be an API error, the request and response are correct, but the word is not bookmarked on the server and therefore also not visible in the word list in the app, after the language was switched it worked again maybe because a word was bookmarked twice)

Guillaume Corsini (22.10.15)

Owned an Android phone

- The user had no problems logging into his Feedly account
- Synchronization was clear
- Password should be typed two times because it is hidden (to avoid mistakes)
- He had no problem creating his Zeeguu account
- He tried to translate a word, however the language was not set correctly
- He did not realize that the entry is displayed in a panel which he can slide down to exit first, then he found the back button (it needs to be clear to the user that the article is used in a panel)
- After that he found the settings without any problems and was able to set his languages
- He tried to translate a word in the title, it was not possible to select it
- Note: Font is a bit small on smaller screens (but seems to be the case in all apps on his phone, for the user the size was ok - maybe add font size to settings?)
- Maybe display drawer in ViewPager too?
- Implement refresh pulldown in feed entry list (consistent design)

B.2 Case Study

B.2.1 Article Source and Reading Order

	Source	Title (URL)	Order
1	Der Bund	Paris ist nicht ihre Stadt	6
2	Golem	Wenn der Roboter die Einkäufe nach Hause bringt	4
3	Caschys Blog	Ausgeknipst: Samsung stellt Kamera-Geschäft in Deutschland ein	1
4	Computerbase	Anbieter äussern sich kritisch über Valves Betriebssystem	8
5	Spiegel	Nigeria: Viele Tote bei zwei Anschlägen im Norden des Landes	5
6	Der Bund	Jung, ehrgeizig und blochertreu	3
7	NZZ	Was vom Tage übrig bleibt	9
8	Heise	ShareRoller: Tragbarer Motor macht jedes Fahrrad zum E-Bike	7
9	Golem	Crytek verteilt VR-Dino-Demo kostenlos	2

Table B.1: The source of the articles and the reading order

B.2.2 Detailed Article Statistics

	Difficulty (Average/Median)		**Learnability (Count/Percentage)**	
1	0.22	0.01	33	2.72 %
2	0.26	0.02	28	3.67 %
3	0.23	0.02	5	2.42 %
4	0.30	0.04	26	4.15 %
5	0.32	0.04	9	3.01 %
6	0.34	0.04	5	1.35 %
7	0.43	0.16	15	3.26 %
8	0.41	0.19	15	3.72 %
9	0.49	0.42	7	2.41 %

Table B.2: The detailed difficulty scores and learnability values

	Characters	**Words**	**Time**	**Looked up**	**Bookmarked**
1	3783	656	900 s	51	41
2	4411	775	754 s	35	30
3	993	165	226 s	12	9
4	3875	632	899 s	56	50
5	2062	338	382 s	16	14
6	3465	549	900 s	53	46
7	2829	423	900 s	82	56
8	2521	398	462 s	23	22
9	1651	280	549 s	23	14

Table B.3: The article characteristics[1]

[1]Due to a video recording error in the articles number 1, 6 and 7, they were only evaluated for the first 15 minutes. Therefore, the characteristics in this table only include the part of the text that the participant read in that amount of time