Need4Feed:
Software Specification
by BIT

Brian Pohl
Student ID: 9100920124
enzothebaker8@gmail.com

Iker Trun
Student ID: 9101920127
iktrun@gmail.com

Thomas Ingvarsson
Student ID: 9093920122
ingvarsson.thomas@sillys.se

December 8, 2012

Revision History

Version	Date	Description	Author
0.1	24-Oct-2012	First draft, an empty skeleton.	T. Ingvarsson
0.2	24-Oct-2012	Added initial section structure.	T. Ingvarsson
0.10	28-Oct-2012	Added entity and class diagrams along with initial descriptions. Also added the introduction content.	T. Ingvarsson
0.11	28-Oct-2012	Added content to Development Environment.	I. Trun
0.12	30-Oct-2012	Added User Interfaces.	B. Pohl
0.13	30-Oct-2012	Added Task distribution. Corrected and cleaned up Development Environment, included references for tools as well as cost estimation.	T. Ingvarsson
0.20	20-Nov-2012	Removed the user interfaces, as to prepare for them to be along with their respective system feature. Added the required subsections for the design phase; Overall Architecture, Directory Organization and Class Description. Also started filling the subsections.	T. Ingvarsson
0.21	21-Nov-2012	Finished class descriptions.	T. Ingvarsson
0.22	26-Nov-2012	Updated and simplified architecture diagram. Added system feature; View Statistics.	T. Ingvarsson
0.23	28-Nov-2012	Updated Specification section with some small changes, also added use cases including screenshots for the system features.	T. Ingvarsson

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to specify a software for Android users enabling the functionality to gather, organize and read feeds, regardless of if it is a RSS[1], ATOM[2], Podcasts or social medium feed. Furthermore, this specification will act as base for the design document. This will be the first release of the mentioned software which is completely standalone in the sense that it is not part of a larger system.

1.2 Intended Audience and Reading Suggestions

The intended audience for this document are those with an engineering background interested in this project, whether it be the actual teacher assigning this project to the project group, one of the group members or an actual user wanting to know more about the product or continue development. The document is divided into two major sections. Section 2 is about the development environment in which the product will be developed and section 3 describes the classes and their connection as well as mock-ups of the user interface.

1.3 Definitions, acronyms and abbreviations

Below is a list of definitions, acronyms and abbreviations referenced in this document, sorted after appearance.

API
: Application Programming Interface

1.4 References

In this section is a list of all documents referenced throughout the rest of this document.

[1] RSS Advisory Board. Rss 2.0 specification (version 2.0.11). `http://www.rssboard.org/rss-specification`, October 2012.

[2] R. Sayre M. Nottingham. The atom syndication format. `http://www.ietf.org/rfc/rfc4287.txt`, October 2012.

[3] Google Inc. Adt plugin — android developers. `http://developer.android.com/tools/sdk/eclipse-adt.html`, October 2012.

[4] Google Inc. Exploring the sdk — android developers. `http://developer.android.com/sdk/exploring.html`, October 2012.

[5] Google Inc. Android ndk — android developers. `http://developer.android.com/tools/sdk/ndk/index.html`, October 2012.

[6] The Eclipse Foundation. Eclipse public license - v 1.0. http://www.eclipse.org/legal/epl-v10.html, October 2012.

[7] Steve Mezak. The cost of global software development. http://www.stevemezak.com/?p=61, October 2012.

[8] Avram Piltch. Average price of laptop now just $8 more than ipad. http://blog.laptopmag.com/the-average-pc-laptop-cost-507-in-march, April 2012.

[9] Numbeo. Cost of living in seoul, south korea. http://www.numbeo.com/cost-of-living/city_result.jsp?country=South+Korea&city=Seoul&displayCurrency=USD, October 2012.

2 Development Environment

2.1 Platform

2.1.1 Target Platform

Smartphone users mostly choose Android OS compared to other OS such as Windows phone and iPhone OS. Major manufacturing giants like Samsung , LG, Motorola, HTC choose Android OS for their smart phones. Smartphone users also choose Android OS ahead of its rivals as it is the best selling smart phone platform in the world. Here are some of the reasons to choose Android. First of all is that Android is an open source software stack for mobile devices under Google Inc. The maintenance and further development of Android open source code is led by Google. As Android is an open source no manufacturer can control the innovations of another. The mistake of one manufacturer in Android open source code gets corrected by another and there will be no central point of failure. Furthermore Android applications are easily downloaded from either Google Play, the Android application market, or from a third party. Android applications are easily obtained for various purposes as it is an open source software and there are over 150,000 applications from a large community of developers. Android applications are available in both free as well as paid versions of applications, which is beneficial to both developers as well as the users.

2.1.2 Development Platform

The environment for the development of Android applications is provided by Google and it's open source which means that any developer can download it from the Android web page. All these tools are gathered in Android Development Tools(ADT)[3], in which most common used tool for developers is the Software Development Kit(SDK)[4]. Android SDK is a tool for Android applications developers which provides the developer all necessary tools such as the API libraries and developer tools necessary to build, test, and debug apps for Android. But also is needed a programming environment because the SDK must be supported from an Integrated Development Environment(IDE) as Eclipse. Eclipse is one of the most common IDEs used by developers. In Eclipse it is possible to program in many different languages such as C++, Java, Python, PHP or Perl. This multi-language versatility is due to the possibility to install different plug-in system. The Android ADT is such a plug-in specific for Android. The most common language to develop Android applications in is Java, but it is also possible to develop in C or C++ with the Android Native Development Kit(NDK)[5]. Development in C or C++ does however require knowledge of low level Android systems and is not very common compared to Java. It is also possible, but even more uncommon, to develop applications in Scala, Python, Lua or Perl.

2.1.3 Cost

Item	Average Estimation	Actual Cost
Development Environment		
Android ADT	Free	0$
Eclipse	Eclipse Public License[6]	0$
Google Code	Free	0$
Developers		
Group Members	25$/h[7]	0$
Open-Source Community	Free	0$
Computers	500$/person[8]	0$
Internet	20$/month[9]	0$
Total:	**10580$**	**0$**

Table 1: Costs of Development

The cost of development is divided into a set of areas; development environment, developers, computers and Internet. For the actual environment we have the IDE, Eclipse, combined the the Android ADT plug-in which are both free to use for open-source projects such as this so there is no cost. The project is hosted, including revision handling and issue tracker, on Google Code which is also free for open-source projects bringing the total cost of the development environment to nothing. Developers in this case can be divided into the group members and any volunteer from the open-source community. If the group members had been paid developers an average cost estimation of 25$/h could be used for each developer. Combine that with 40/5 hours per week, standard work week divided by amount of courses, and 15 weeks and it would result in 3000$/developer of the project time frame. However, as the project group are students and are doing this as a school project there is no cost. For the equipment, computers and Internet, standard rates has been used for the average estimation. However, similar to the developer cost the students already have computers and Internet access as they are located now, even if they didn't have it themselves they would have had access at school. The total estimated cost would then result in 10580$ while the actual cost for this project is 0$.

2.2 Task Distribution

Brian Pohl

- Demonstration
- Modelling
- Design

Iker Trun

- Programming: Database
- Development Environment
- Testing

Thomas Ingvarsson

- Programming
- Programming: User Interfaces
- Programming: Feed Access
- Documentation
- Version Handling
- Issue Tracker

3 Specifications

The structure behind the product is described in section 3.1, where the required entities are modelled along with there relationships. The different layers of the application architecture is also described along with a navigation map of the different user interfaces. Each system feature will, in section 3.2, be examined and described. All the required user interfaces are described along with their respective system feature. In section 3.3 is the physical file structure of the project described. In the last section, 3.4, are all classes listed along with a description, what classes are imported and their location in the project.

3.1 Overall Architecture

The overall architecture of the project is built as an *Android* application; the user interfaces are built up by so called *Activities* while any background tasks are run as *Services*. All background functionality is built up by standard *Java* classes though some of them inherit from *Android* classes. To allow the user

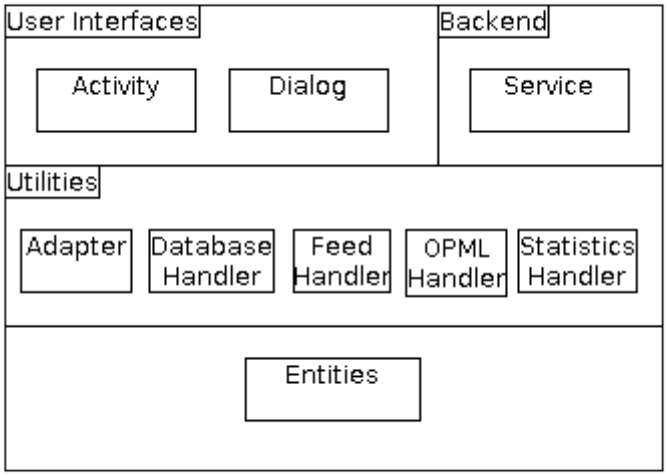

Figure 1: Layers of the Application Architecture

to input information such as category names or feed links *Dialogs* are used, based on the *Android* class *DialogFragment*. The complete architecture of the application is shown in figure 1 divided into different layers. The two top layers are the actual application, *User Interfaces* and *Backend*, and utilizes the middle layer *Utilities*. These give fully access and control over the lower layer containing the *Entities*. In the current architecture all the *Utilities* are connected to all the types of *Entities*. The *Entities* of the application are *Category*, *Feed* and *Post* and are shown in figure 2. Here all sub-entities are included and their

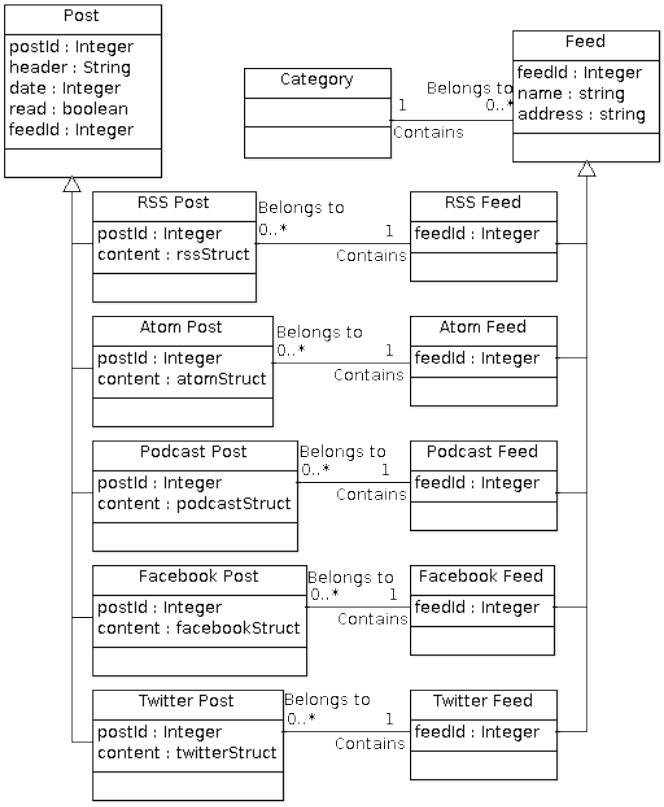

Figure 2: Entity Class Diagram of the Product

relationship to each other.

3.2 System Features

In this section are all the included system features examined and described with the help of their respective class diagram.

3.2.1 Share Post

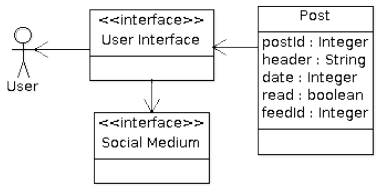

Figure 3: Class Diagram of system feature Share Post

The ability to share a post requires two things, a post to share and a target to share it to. The relationship required to perform this feature is visualized in figure 3. Here the user has the ability through the user interface to share a post to a social medium of its choice. The social medium is accessed through an interface to its API.

1. Access the post view, shown in figure 4.
2. Display all actions by pressing the *Menu* button on the device.
3. Press *Share Post*.
4. In the displayed dialog enter the required credentials to be able to share the post the chosen medium.

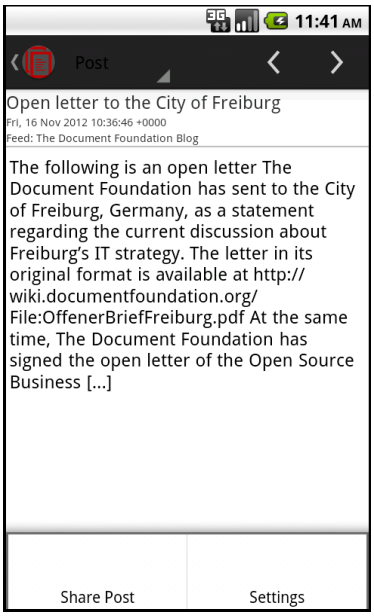

Figure 4: Screenshot of the post view

3.2.2 Count Un-read Post(s)

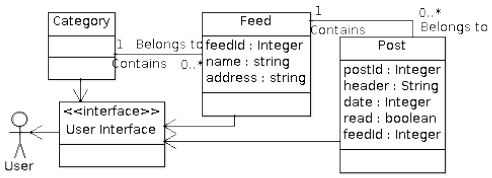

Figure 5: Class Diagram of system feature Count Un-read Post(s)

To count un-read post(s) access to not only the posts but also the feeds and the feed list are required so that posts of all feeds are accounted for. The relationship required to perform this feature is visualized in figure 5. Here the user accesses the number of un-read post(s) through the user interface which in turn is connected to all the required classes to determine the total un-read post(s) count.

1. Access the feed view, shown in figure 6.
2. Unread posts are shown as bold in the list, while read posts are not bold.

The total amount of unread posts can be accessed internally through a database query, and implemented to be shown where it fits.

Figure 6: Screenshot of the feed view

3.2.3 Read Post(s)

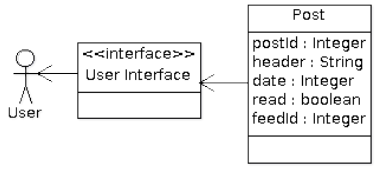

Figure 7: Class Diagram of system feature Read Post(s)

For users to access and read post(s) an user interface is required that is connected to the actual post(s). The relationship required to perform this feature is visualized in figure 7. The user interface will determines which post(s) based on previous state, e.g. which post the user has selected to read coming from a previous view.

1. Access the post view, shown in figure 8.
2. Be notified by the interface that the post has been marked as read.
3. Read title, publication date and source feed in the header.
4. Read the description, that may include pictures, below the header.
5. To access the actually post through a web browser click the title in the header.

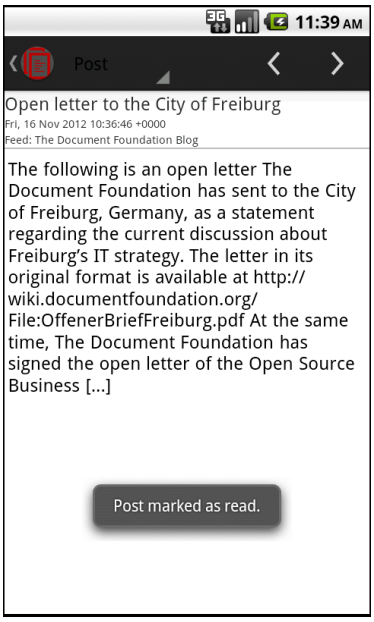

Figure 8: Screenshot of the post view

3.2.4 View Feed(s)

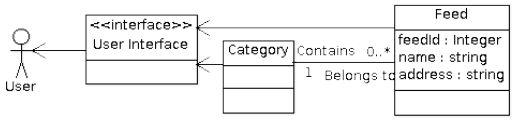

Figure 9: Class Diagram of system feature View Feed(s)

Users access and view feed(s) through an user interface that is connected to the feed(s). The relationship required to perform this feature is visualized in figure 9. The user interface determines which feed(s) based on previous state, e.g. which category of feeds the user has selected to view coming from a previous view.

1. Access the feed view, shown in figure 10.
2. The 30 latest posts are shown in a scrollable list.
3. Press a post to access the post view for that specific post.

Figure 10: Screenshot of the feed view

3.2.5 View Statistics

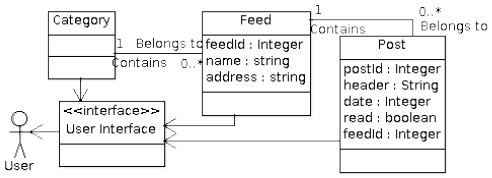

Figure 11: Class Diagram of system feature View Statistics

To view statistics access to not only the posts but also the feeds and the feed list are required so that all statistics on all levels are collected. The relationship required to perform this feature is visualized in figure 5. Here the user accesses the statistics through the user interface which in turn is connected to all the required classes to collect all interesting data.

1. Access any view of feed level or higher, shown in figure 12.
2. Display all actions by pressing the *Menu* button on the device.
3. Press *Statistics*.
4. In the displayed dialog statistics are shown for the current level.

Figure 12: Screenshot of the feed view

3.2.6 Import/Export Feed(s)

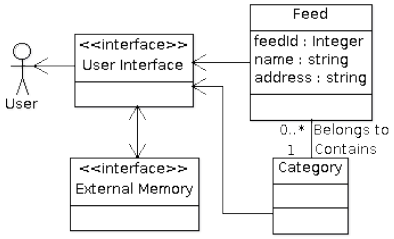

Figure 13: Class Diagram of system feature Import/Export Feed(s)

By an interface to a storage area, external memory in the case of Android, the user has the possible to either import a set of feed(s) or export. The relationship required to perform this feature is visualized in figure 13. Both actions are triggered through the user interface. For importing the user chooses a file on external memory, with a standardized file format for feed lists, and the product performs a mass add of the feeds in the file. An export action from the user will create a feed list file, with the same standardized file format as mentioned earlier.

1. Access the main view, shown in figure 14.
2. Display all actions by pressing the *Menu* button on the device.
3. Press *Import/Export Feeds*.
4. In the displayed dialog chose if you want to import or export feeds.
5. In the following dialog chose either which file to import or what file name to export to.

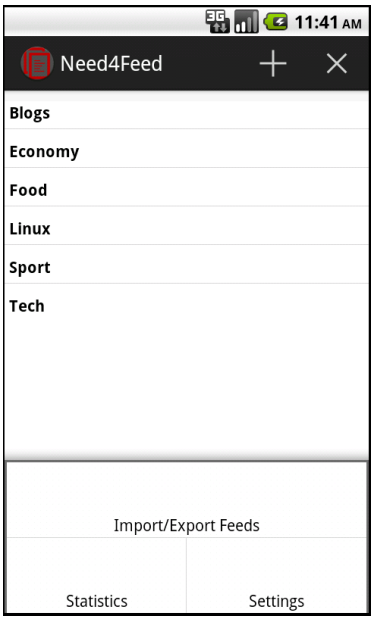

Figure 14: Screenshot of the main view

3.2.7 Find Similar Feed(s)

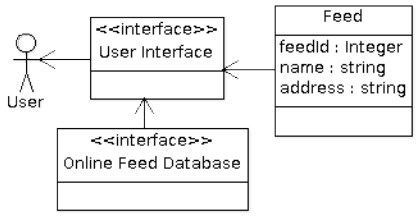

Figure 15: Class Diagram of system feature Find Similar Feed(s)

When a user has a feed and wants to find similar feeds he can through the user interface trigger a search. The relationship required to perform this feature is visualized in figure 15. The search will look in online feed databases, through their web-based API's, based on data from the feed in form of name and address.

1. Access the feed view, shown in figure 16.
2. Press *Find Similar Feeds*, shown as the first icon in the action bar visualized as a globe.
3. In the displayed dialog chose which similar feed you like to add.

Figure 16: Screenshot of the feed view

3.2.8 Manage Feed(s)

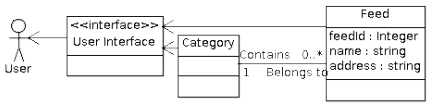

Figure 17: Class Diagram of system feature Manage Feed(s)

For a user, or administrator to be exact, to manage feeds he has to do so through the user interface in which there are options and functions to perform all required sub-features. The relationship required to perform this feature is visualized in figure 17. In this case the user interface is directly connected to the feed list and feed(s).

1. Access the main view, shown in figure 18.
2. Press *Add Category*, shown as the first icon in the action bar visualized as a plus sign.
3. In the displayed dialog chose a name for the category.

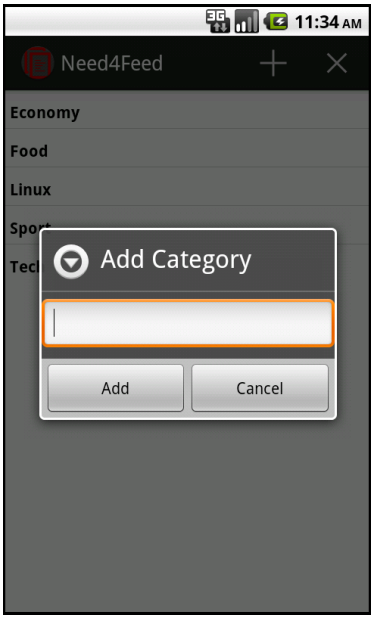

Figure 18: Screenshot of the add category dialog

1. Access the category view, shown in figure 19.
2. Press *Add Feed*, shown as the first icon in the action bar visualized as a plus sign.
3. In the displayed dialog enter the url address to the feed to be added.

Figure 19: Screenshot of the add feed dialog

3.3 Directory Organization

The structure of the project is the default structure of an *Eclipse* project for *Android* using the *ADT*. The source file are located in */src/* with a substructure based on the *Java* packages they are located in, something that is mandatory for *Android projects*. The used packages has to be completely unique to latter on be accepted into *Google Play*, the online application market for *Android*. Beyond source files, seen in table 2, an *Android* project also contains *XML*

Directory	**File name**
/src/bit/app/need4feed/	MainApplication.java
/src/bit/app/need4feed/	MainActivity.java
/src/bit/app/need4feed/	CategoryActivity.java
/src/bit/app/need4feed/	FeedActivity.java
/src/bit/app/need4feed/	PostActivity.java
/src/bit/app/need4feed/	RssService.java
/src/bit/app/need4feed/	AddCategoryDialog.java
/src/bit/app/need4feed/	RemoveCategoryDialog.java
/src/bit/app/need4feed/	AddFeedDialog.java
/src/bit/app/need4feed/	RemoveFeedDialog.java
/src/bit/app/need4feed/util/	DatabaseHandler.java
/src/bit/app/need4feed/util/	RssHandler.java
/src/bit/app/need4feed/util/	OpmlHandler.java
/src/bit/app/need4feed/type/	Category.java
/src/bit/app/need4feed/type/	CategoryAdapter.java
/src/bit/app/need4feed/type/	Feed.java
/src/bit/app/need4feed/type/	FeedAdapter.java
/src/bit/app/need4feed/type/	Post.java
/src/bit/app/need4feed/type/	PostAdapter.java

Table 2: Directory Organization of Android Source Files

files used for layouts for activities, dialogs and menus. *XML* files are also used for strings used graphically throughout the program. All of these *XML* files are located under */res/layout/*, */res/menu/* and */res/strings/*. In addition to *XML* files */res/* also includes image files used for icons and symbols, these are located in their respective */res/drawable/* directory. All files under */res/* are resources that are used dynamically during runtime, this design enables *Android* to change the its look while running based on feature such as size of phone or as simple as the current orientation of the screen. The directory organization of the resource files are shown in table 3.

Directory	File name
/res/layout/	*.xml
/res/menu/	*.xml
/res/strings/	*.xml
/res/drawable-hdpi/	*.png
/res/drawable-ldpi/	*.png
/res/drawable-mdpi/	*.png
/res/drawable-xhdpi/	*.png

Table 3: Directory Organization of Android Resource Files

3.4 Class Description

3.4.1 MainApplication

MainApplication belongs to the *Backend* layer of the architecture but is however not a *Service*. This class extends the *Android* class *Application* and has the purpose to create a structure with a different life-cycle than of the activities, that can pause or terminate at any point and later on be re-created if returned to. With this feature, of being non-terminated until the application is terminated, *MainApplication* gives the functionality to hold truly global variables and instances accessible to all activities through the *Android* virtual machine.

Location: */src/bit/app/need4feed/MainApplication.java*

Class Components:

- databaseHandler: DatabaseHandler
- getDatabaseHandler(): DatabaseHandler

3.4.2 MainActivity

MainActivity is the main user interface and the entry screen of the application, it extends *Activity* and belongs to the *User Interfaces* layer of the architecture. The purpose of the class is to create the first view, holding a list of all categories. It also takes the user to the next view when a category has been clicked. *MainActivity* gives the functionality, beyond displaying categories and managing the transition to *CategoryActivity*, to add and remove categories.

Location: */src/bit/app/need4feed/MainActivity.java*

Class Components:

- actionBar: ActionBar
- categoryListView: ListView
- categoryAdapter: CategoryAdapter

- databaseHandler: DatabaseHandler
- onFinishAddCategoryDialog(): void
- onFinishRemoveCategoryDialog(): void

3.4.3 CategoryActivity

CategoryActivity is the second activity the user gets in contact with, it extends *Activity* and belongs to the *User Interfaces* layer of the architecture. The purpose of the class is to create a list view of feeds for a certain category, thereby the name *CategoryActivity*. It gives the functionality, except to list all feeds of a category, to take the user to a certain feed if clicked on as well as add and remove feeds to that specific category.

Location: */src/bit/app/need4feed/CategoryActivity.java*

Class Components:

- actionBar: ActionBar
- feedListView: ListView
- feedAdapter: FeedAdapter
- databaseHandler: DatabaseHandler
- categoryId: long
- onFinishAddFeedDialog(): void
- onFinishRemoveFeedDialog(): void

3.4.4 FeedActivity

FeedActivity is the third activity the user gets in contact with, it extends *Activity* and belongs to the *User Interfaces* layer of the architecture. The purpose of the class is to create a list view of posts for a certain feed, thereby the name *FeedActivity*. It gives the functionality, except to list all posts of a feed, to take the user to a certain post if clicked on.

Location: */src/bit/app/need4feed/FeedActivity.java*

Class Components:

- actionBar: ActionBar
- postListView: ListView
- postAdapter: PostAdapter
- databaseHandler: DatabaseHandler
- feedId: long

3.4.5 PostActivity

PostActivity is the fourth, and last, activity the user gets in contact with, it extends *Activity* and belongs to the *User Interfaces* layer of the architecture. The purpose of the class is to display a specific post and its content, thereby the name *PostActivity*. It gives the functionality, except to display a post, to take the user to the web page of the post through a click on the post heading.

Location: */src/bit/app/need4feed/PostActivity.java*

Class Components:

- actionBar: ActionBar
- titleTextView: TextView
- feedTextView: TextView
- contentWebView: WebView
- sourceFeed: Feed
- displayedPost: Post
- databaseHandler: DatabaseHandler
- postId: long

3.4.6 RssService

RssService extends *Intent Service* and belongs to the *Backend* layer of the architecture. The purpose of this class is to serve as a background task, periodically fetching the latest posts. Its functionality is built up out off two parts; database access to both fetch all existing feeds and then store all new posts as well as access the actual *RSS* feeds.

Location: */src/bit/app/need4feed/RssService.java*

Class Components:

- databaseHandler: DatabaseHandler
- rssHandler: RssHandler

3.4.7 AddCategoryDialog

AddCategoryDialog is a *Dialog* and belongs to the *User Interface* layer of the architecture. The class extends *DialogFragment* to enable all required *Dialog* functionality. The purpose of this class is to create the user interface that enables the user to input a category name and create a category. Furthermore the functionality includes adding the new category to the database and signal the associated *Activity* through *addCategoryDialogListener* which is connected to *MainActivity*.

Location: */src/bit/app/need4feed/AddCategoryDialog.java*

Class Components:

- databaseHandler: DatabaseHandler
- addCategoryDialogListener: AddCategoryDialogListener

3.4.8 RemoveCategoryDialog

RemoveCategoryDialog is a *Dialog* and belongs to the *User Interface* layer of the architecture. The class extends *DialogFragment* to enable all required *Dialog* functionality. The purpose of this class is to create the user interface that enables the user to input which category to remove. Furthermore the functionality includes removing the specific category from the database and signal the associated *Activity* through *removeCategoryDialogListener* which is connected to *MainActivity*.

Location: */src/bit/app/need4feed/RemoveCategoryDialog.java*

Class Components:

- databaseHandler: DatabaseHandler
- removeCategoryDialogListener: RemoveCategoryDialogListener

3.4.9 AddFeedDialog

AddFeedDialog is a *Dialog* and belongs to the *User Interface* layer of the architecture. The class extends *DialogFragment* to enable all required *Dialog* functionality. The purpose of this class is to create the user interface that enables the user to input a feed link and create a feed. Furthermore the functionality includes adding the new feed to the database, fetch the latest posts for it, and signal the associated *Activity* through *addFeedDialogListener* which is connected to *CategoryActivity*.

Location: */src/bit/app/need4feed/AddFeedDialog.java*

Class Components:

- databaseHandler: DatabaseHandler
- addFeedDialogListener: AddFeedDialogListener

3.4.10 RemoveFeedDialog

RemoveFeedDialog is a *Dialog* and belongs to the *User Interface* layer of the architecture. The class extends *DialogFragment* to enable all required *Dialog* functionality. The purpose of this class is to create the user interface that enables the user to input which feed to remove. Furthermore the functionality includes removing the specific feed from the database and signal the associated *Activity* through *removeFeedDialogListener* which is connected to *CategoryActivity*.

Location: */src/bit/app/need4feed/RemoveFeedDialog.java*

Class Components:

- databaseHandler: DatabaseHandler
- removeFeedDialogListener: RemoveFeedDialogListener

3.4.11 DatabaseHandler

DatabaseHandler is part of the *Utilities* layer of the architecture and works as a *Singleton* class as it is only initiated within the *MainApplication*. To enable *SQLite* functionality it extends *SQLiteOpenHelper*. The purpose of this class is to handle all queries to and from the database containing all entities.

Location: */src/bit/app/need4feed/util/DatabaseHandler.java*

Class Components:

- db: SQLiteDatabase
- addCategory(Category): void
- addFeed(Feed): void
- addPost(Post): void
- updateCategory(Category): void
- updateFeed(Feed): void
- updatePost(Post): void
- deleteCategory(long): void
- deleteFeed(long): void
- deletePostOfFeed(long): void
- deletePost(long): void
- getCategories(): List<Category>
- getCategoryNames(): String[]
- getAllFeeds(): List<Feed>
- getFeeds(long): List<Feed>
- getFeedNames(long): String[]
- getFeed(long): Feed
- getPosts(long): List<Post>
- getPost(long): Post
- getPostCount(long) int

3.4.12 RssHandler

RssHandler is part of the *Utilities* layer of the architecture and extends *DefaultHandler* to be able to process *XML* files. The class imports *SAXParser* to help parse tags. The functionality of *RssHandler* is to access the *Internet* and fetch *RSS XML* files and process them, either to fetch information about a feed or its posts. The class then stores the information in the database.

Location: */src/bit/app/need4feed/util/RssHandler.java*

Class Components:

- db: DatabaseHandler
- currentFeed: Feed
- currentPost: Post
- postList: List<Post>
- verifyFeed(String): String
- getFeed(String): Feed
- getAllLatestPosts(): void
- getLatestPosts(Feed): void

3.4.13 OpmlHandler

OpmlHandler is part of the *Utilities* layer of the architecture and extends *DefaultHandler* to be able to process *XML* files. The class imports *SAXParser* to help parse tags. The functionality of *OpmlHandler* is to process feed lists written in *OPML* file format, that expends *XML*. It enables both import and export of feed lists.

Location: */src/bit/app/need4feed/util/OpmlHandler.java*

Class Components:

- db: DatabaseHandler
- ImportFeeds(File file): void
- ExportFeeds(void): File

3.4.14 Category

Category, the entity class, of the *Entities* layer of the architecture. The purpose of this class is to symbolize an entity and manage its variables. Also included is a *compareTo* method to deal with sorting of the entity, for categories the sorting is done alphabetically.

Location: */src/bit/app/need4feed/type/Category.java*

Class Components:

- id: long
- name: String
- getId(): long
- setId(long): void
- getName(): String
- setName(String): void
- compareTo(Category): int

3.4.15 CategoryAdapter

CategoryAdapter is utility tight connected to its type, *Category*, and is part of the *Utilities* layer of the architecture. Its purpose is to handle lists of categories and present them in *ListViews*.

Location: */src/bit/app/need4feed/type/CategoryAdapter.java*

Class Components:

- categoryList: List<Category>
- holder: ViewHolder
- setCategoryList(List<Category>): void

3.4.16 Feed

Feed, the entity class, of the *Entities* layer of the architecture. The purpose of this class is to symbolize an entity and manage its variables. Also included is a *compareTo* method to deal with sorting of the entity, for feeds the sorting is done alphabetically.

Location: */src/bit/app/need4feed/type/Feed.java*

Class Components:

- id: long
- categoryId: long
- title: String
- link: String
- feedLink: String
- description: String
- getId(): long
- setId(long): void
- getCategoryId(): long
- setCategoryId(long): void

- getTitle(): String
- setTitle(String): void
- getLink(): String
- setLink(String): void
- getFeedLink(): String
- setFeedLink(String): void
- getDescription(): String
- setDescription(String): void
- compareTo(Feed): int

3.4.17 FeedAdapter

FeedAdapter is utility tight connected to its type, *Feed*, and is part of the *Utilities* layer of the architecture. Its purpose is to handle lists of categories and present them in *ListViews*.

Location: */src/bit/app/need4feed/type/FeedAdapter.java*

Class Components:

- feedList: List<Feed>
- holder: ViewHolder
- setFeedList(List<Feed>): void

3.4.18 Post

Post, the entity class, of the *Entities* layer of the architecture. The purpose of this class is to symbolize an entity and manage its variables. Also included is a *compareTo* method to deal with sorting of the entity, for posts the sorting is done based on the date which is stored in *pubDate*.

Location: */src/bit/app/need4feed/type/Post.java*

Class Components:

- id: long
- feedId: long
- title: String
- link: String
- description: String
- pubDate: String
- thumbnail: String
- getId(): long

- setId(long): void
- getFeedId(): long
- setFeedId(long): void
- getTitle(): String
- setTitle(String): void
- getLink(): String
- setLink(String): void
- getDescription(): String
- setDescription(String): void
- getPubDate(): String
- setPubDate(String): void
- getThumbnail(): String
- setThumbnail(String): void
- compareTo(Post): int

3.4.19 PostAdapter

PostAdapter is utility tight connected to its type, *Post*, and is part of the *Utilities* layer of the architecture. Its purpose is to handle lists of categories and present them in *ListViews*.

Location: */src/bit/app/need4feed/type/PostAdapter.java*

Class Components:

- postList: List<Post>
- holder: ViewHolder
- setPostList(List<Post>): void