FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Android-based mobile client for LinkedPipes ETL
Student:	David Paleček
Supervisor:	RNDr. Jakub Klímek, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2020/2021

Instructions

The student will get to know the Linked Data principles [1][3] and the RDF data model and serializations [2][4].
The student will study the architecture of the LinkedPipes ETL tool to get to know its API [5].
The student will design, implement, document and evaluate an Android-based mobile application serving as an alternative client to the current LinkedPipes ETL frontend.
The application will provide pipeline and execution management and notification capabilities for multiple LinkedPipes ETL instances.

–

[1] Christian Bizer, Anja Jentzsch. State of the LOD Cloud. http://www4.wiwiss.fu-berlin.de/lodcloud/state/
[2] W3C. RDF Primer. http://www.w3.org/TR/rdf-primer/
[3] Linked Data. http://linkeddata.org/
[4] SPARQL Query Language for RDF. W3C Recommendation 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/
[5] LinkedPipes ETL, https://etl.linkedpipes.com

Electronically approved by Ing. Michal Valenta, Ph.D. on 12 February 2020 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Android-based mobile client for LinkedPipes ETL

David Paleček

Department of Software Engineering
Supervisor: RNDr. Jakub Klímek, Ph.D.

June 26, 2021

Acknowledgements

I would like to thank my supervisor RNDr. Jakub Klímek, Ph.D. for his never ending patience and willingness to guide me. Also, I would like to thank my parents for supporting me through my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In V Praze on June 26, 2021 …………………

Citation of this thesis

Paleček, David. *Android-based mobile client for LinkedPipes ETL*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

LinkedPipes ETL je systém pro práci s linked data, který má pouze webové rozhraní. Tato práce se zabývá tvorbou mobilního klienta LinkedPipes ETL pro platformu Android. Obsahuje všechny stádia vývoje této aplikace, což jsou sepsání požadavků pro aplikaci, procházení exiistujících řešení, navrhování aplikace, implementace, testování a tvorba dokumentace.

Klíčová slova tvorba klienta, Android, Kotlin, LinkedPipes, ETL

Abstract

LinkedPipes ETL is a system for working with linked data, which has only a web interface. This work deals with the creation of the LinkedPipes ETL mobile client for the Android platform. It contains all stages of the development of this application, which are writing down the application requirements, the browsing of existing solutions, the designing of application, implementation, testing and the creation of documentation.

Keywords client creation, Android, Kotlin, LinkedPipes, ETL

Contents

List of Figures

List of Tables

Introduction

Linked data is data that links to other data using URIs. The URIs identify not only other data objects, but also relations, types or other concepts. [1]

LinkedPipes ETL is a system that consumes, transforms and produces linked data. It was created in 2016 by students and researchers from Prague computer science universities [2]. It is an open source project [3] with MIT license, so anyone can use it. The system is accessible via web front-end. The front-end has support for mobile phones, but it has some drawbacks that will be discussed later, with the main drawback being that it is unable to manage multiple server instances from one place.

In this thesis a solution to both problems is presented by making an Android application, where all of the user's LinkedPipes ETL server instances can be managed at the same time. The Android application will provide a nice and smooth mobile experience to LinkedPipes ETL users, especially to those, who are working with multiple server instances.

The rest of this thesis is structured as follows. Requirements for the application will be written down in chapter 1. The written requirements will be compared to existing solutions in chapter 2. The application's parts that can be seen and users can interact with will be designed in chapter 3. Parts that can not be seen, such as the architecture, will be designed in chapter 4. At the end of the chapter 4, everything needed for the implementation should be known. In the chapter 5, important libraries and some inconspicuous algorithms will be described. Creation of documentation will be described in chapter 6. Testing will be described in chapter 7.

Work goal

The goal of this thesis is to pitch an Android client for the LinkedPipes ETL system as an alternative to opening a web page in a mobile browser.

In order to do that, the following steps must be fulfilled.

- A list of requirements for the application will be put together in the analysis chapter.
- Existence of no current solution will be verified in the existing solutions chapter.
- Application will be designed, including appearance and code layering, in the design chapters.
- Result of the implementation chapter will be the application.
- Documentation will be created after the application is created and described in the documentation chapter.
- Application tests will be described in the test chapter.

Glossary

History History of pipeline executions

List A group of objects

Managing Adding, editing, deleting

Pipeline A set of tasks defined in LinkedPipes ETL server instance

Server instance LinkedPipes ETL server instance

User Common user of an Android smartphone

Chapter **1**

Requirements engineering

Features expected from the application will be written down in this chapter. All information needed was gathered in an interview with the thesis supervisor, an experienced LinkedPipes ETL user.

1.1 Requirements

In this section, granular requirements are gathered and described.

F-1.1: Settings screen

Application must have a separate screen for settings.

F-2.1: View server instance

List of server instances will be visible from the settings screen.

F-2.2: Add server instance

User must be able to add server instances.

F-2.3: Edit server instance

User must be able to edit already added server instances.

F-2.4: Delete server instance

User must be able to delete already added server instances. It will be possible to undo the action.

F-2.5: Deactivate server instance

User can deactivate server instances in settings instead of deleting it, so it is possible to activate it again later easily. App will not communicate with deactivated server instances.

F-2.6: Ping server

User can test if the server address is correct.

F-2.7: Load server instance info from QR code

User can load server instance URLs from QR code.

F-3.1: Notification after pipeline execution finish

Application shall create a notification on pipeline execution finish.

F-3.2: Notifications in settings

It will be possible to toggle notifications in settings.

F-4.1: Pipeline list screen

Application must have a separate screen for working with pipelines. Which pipelines will be visible there depends on F-4.8.

F-4.2: View pipelines

List of pipelines will be visible from pipeline list screen. Which pipelines will be visible depends on F-4.8.

F-4.3: Edit pipeline screen

Application must have a screen for editing pipelines.

F-4.4: Create pipelines

User must be able to start an empty edit pipeline screen (F-4.3) from the pipeline list screen (F-4.1).

F-4.5: Edit existing pipelines

User must be able to edit pipelines by starting the edit pipeline screen (F-4.3) with the selected pipeline loaded.

F-4.6: Delete pipelines

User must be able to delete a pipeline of his choice.

F-4.7: Execute pipeline

User must be able to execute pipelines.

F-4.8: Source for visible pipelines

User must be able to choose, if he wants to see pipelines from all instances, or just a specific one.

F-5.1: Execution history screen

Application must have a separate screen for execution history. History of which server instance will be visible depends on F-5.5

F-5.2: View execution history

List of executions will be visible from the execution history screen. History of which server instance will be visible depends on F-5.5

F-5.3: Delete execution from history

User must be able to delete a specific execution from the execution history. It will be possible to undo the action.

F-5.4: Re-execute pipelines from history

There must be an option to re-execute pipeline from the execution history screen. This action will also make a new record in execution history.

F-5.5: Source of visible history

User must be able to choose, if he wants to see the history of all instances, or just a specific one.

F-6.1: Night mode

User can have an option in settings to use a light or a dark theme, or use the system default theme (Android 10 and newer).

1.2 Use cases and scenarios

In this section, use cases, representing reasons why users want to use our application, will be described and complemented by scenarios, describing how to achieve the goals of those use cases. This section is supplemented by a diagram concluding use cases. See Figure 1.1

UC-1: Get overview of executions in particular server instance

Enables user to see what pipelines were executed in chronological order from a specific server instance.

- **SC-1.1: Get overview of executions in particular server instance** User opens the execution history screen (F-5.1, F-5.2) and selects what server instance's executions he wants to see (F-5.5).

UC-2: Execute specific pipeline

Enables user to execute a pipeline of his choice from a specific server instance.

- **SC-2.1: Execute specific pipeline** User opens pipeline list screen (F-4.1, F-4.2). He then finds the desired pipeline and executes it (F-4.7). Optional: After opening the pipeline screen, user can filter pipelines by the server instance (F-4.8).

UC-3: Manage registered server instances

Enables user to register server instances in the application. Application will check, if IP is already registered or if name of the new server instance is already in use, in order to warn user about duplication or name collision that could cause chaos. It also enables user to change the IP address of already registered server instances due to type error or network changes. User can also remove registered server instances.

- **SC-3.1: Change IP address or name of registered server instance** User opens settings screen (F-1.1), selects the desired server instance (F-2.1) for editing (F-2.3). He then changes the IP address and saves the changes.
- **SC-3.2: Register server instance** User opens settings screen (F-1.1), tells the application he wants to register a new server instance and proceeds to enter server instance's information (F-2.2) and saves it.
- **SC-3.3: Delete registered server instance** User opens the settings screen (F-1.1), views registered server instances (F-2.1) and tells the application what server instance he wants to delete. It will be possible to undo the action (F-2.4).

UC-4: Manage pipelines

Enables user to manage pipelines in desired server instances.

- **SC-4.1: Create pipeline** User opens pipeline list screen (F-4.1). Then he tells the application he wants to create a new pipeline (F-4.4). He chooses a server instance to which the pipeline will be saved and the screen for editing pipeline will be launched (F-4.3) and the user can design a new pipeline here. When he is finished, he will save the pipeline.

- **SC-4.2: Edit pipeline** User opens pipeline list screen (F-4.1, F-4.2). He then finds the desired pipeline and tells the application he wants to edit it (F-4.5). The screen for editing pipeline will be launched (F-4.3) with the selected pipeline loaded so the user can make and save changes here. Optional: After opening the pipeline screen, user can filter pipelines by the server instance (F-4.8).

- **SC-4.3: Delete pipeline** User opens pipeline list screen (F-4.1, F-4.2). He then finds the desired pipeline and tells the application he wants to delete it (F-4.6). It will be possible to undo the action. Optional: After opening the pipeline screen, user can filter pipelines by the server instance (F-4.8).

UC-5: Re-execute pipeline from history

Enables user to quickly execute the pipeline he sees while viewing history.

- **SC-5.1: Re-execute pipeline from history** User opens the execution history screen (F-5.1, F-5.2). He finds a pipeline and realizes he wants to execute it now, so he tells that to the application (F-5.4). Optional: After opening the execution history screen, user can select what server instance executions he wants to see (F-5.5).

UC-6: Delete history

Enables user to delete items from history.

- **SC-6.1: Delete history** User opens the execution history screen (F-5.1, F-5.2). He finds a record and realizes that he does not want this record in history anymore, so he tells that to the application (F-5.3). Optional: After opening the execution history screen, user can select what server instance executions he wants to see (F-5.5).

UC-7: View execution history

Enables user to view the execution history of all the instances at the same time.

- **SC-7.1: Get overview of executions** User opens the execution history screen (F-5.1, F-5.2).

UC-8: View pipelines

Enables user to view pipelines from all the server instances.

- **SC-8.1: Get overview of pipelines** User opens pipeline list screen (F-4.1, F-4.2).

UC-9: Be notified on execution finish

User has the option to be notified about execution completion.

- **SC-9.1: Be notified** User executes a specific pipeline, just like in SC-2.1. Application will notify user about the execution completion (F-3.1). Notifying will happen only if it is allowed in settings (F-3.2).

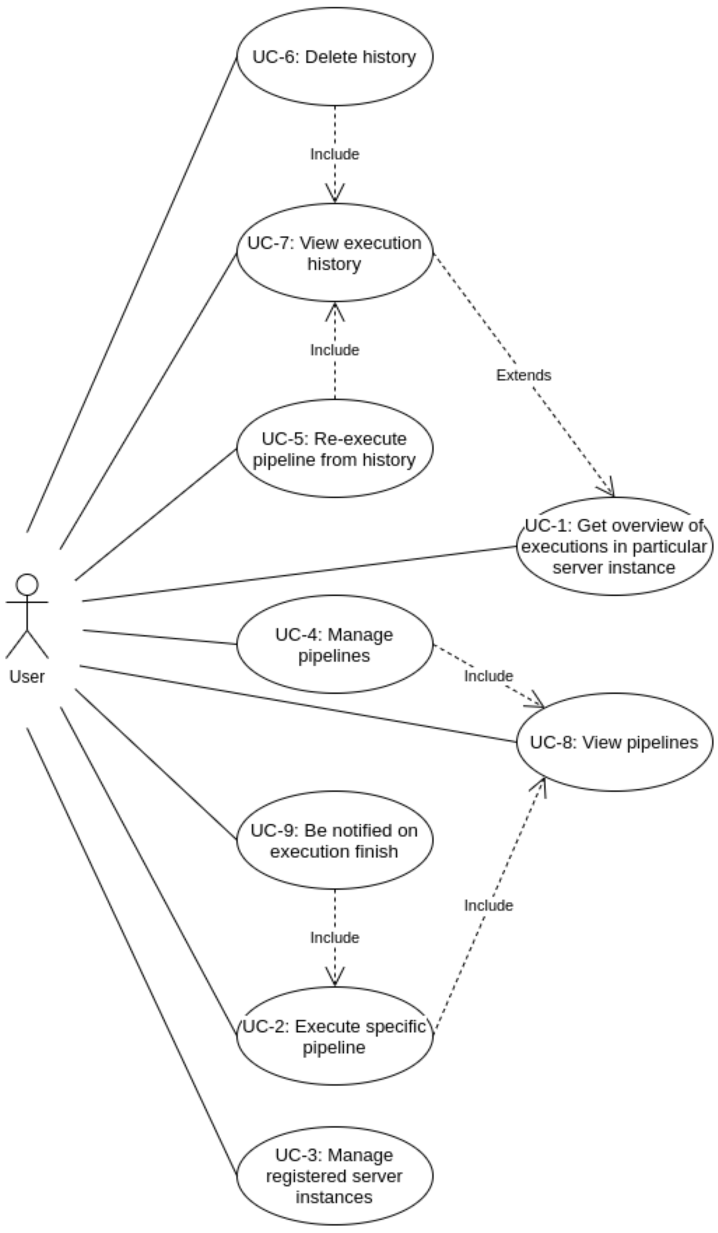

UC-6: Delete history
Include
UC-7: View execution history
Include
Extends
UC-5: Re-execute pipeline from history
UC-1: Get overview of executions in particular server instance
User
UC-4: Manage pipelines
Include
UC-8: View pipelines
UC-9: Be notified on execution finish
Include
Include
UC-2: Execute specific pipeline
UC-3: Manage registered server instances

Figure 1.1: Diagram consisting of use cases

Chapter 2

Existing solutions

In this section, an overview of possible existing solutions can be found. There will be a table with comparisons of those solutions by the end of this chapter.

2.1 Responsive web app

The web front-end can be used by any device possessing a web browser. Users are not obligated to download any application, which also means they do not have to update anything. The responsive web app only works with one server instance. On an Android device, the web app responds slower to screen rotation and animations often lag. Users have to be online, even just for browsing execution history or viewing a pipeline list. The responsive web app is also browser dependent.

2.2 Summary

In this section, existing solutions are being compared with each other (see Table 2.1), resulting in a decision if there is a need to create a new application.

Table 2.1: Features of existing solutions

Feature	**Android App**	**Web App**
Can work with multiple server instances	+	-
Works on any device	-	+
Does not need to be downloaded	-	+
Smooth UI	+	-
Can view stuff while offline	+	-

The comparison indicates that the web application is missing at least one critical feature, that is not being able to work with multiple server instances, thus there is a need for the creation of a new application.

Chapter 3

UI Design

The appearance of the UI will be described in this chapter.

3.1 Design language and UI framework

Because the application should look decent, some UI guidelines have to be chosen and followed. These guidelines cover information about colors, shapes and individual components, including their layout. A set of these guidelines is called design language. Following design languages are suitable for the Android platform due to the existence of frameworks for this platform, containing themes and components of those languages.

Bootstrap [4] is a framework for designing web pages, but there also exists a third party library [5] for the Android platform. Both Microsoft Fluent Design System [6] and Material Design [7] have their own official libraries available from their representative web pages.

The Bootstrap Android library has not been updated since December 2016 and considering that UI design is always changing and evolving, this library is out of question. Both Microsoft Fluent Design System and Material Design are being kept up-to-date and are backed by big international companies, which should ensure their stability. Because our application will be available on the Android platform, which is Google's domain and most Android phones come with several Google applications pre-installed, Android users are already used to Material Design.

That is why Material Design will be used by our application.

3.2 Main screens

Based on the analysis of the user requirements in chapter 1, three screens which cover the functionality of displaying execution history, pipeline list and settings have to be designed.

3.3 Main navigation

On the Android platform, there are multiple navigation designs and they will be described in this section.

3.3.1 Navigation drawer

The hamburger icon at the top left and sliding menu from left to right is what the navigation drawer looks like. This navigation is suitable for five or more top level screens, or some sort of hierarchical menu [8].

3.3.2 Tabs

Slidable tabs on top of the screen. Users can click on tab names or just slide left or right in order to navigate between the screens.

3.3.3 Bottom navigation

Bottom navigation consists of icons, usually with text, located at the bottom of the screen.

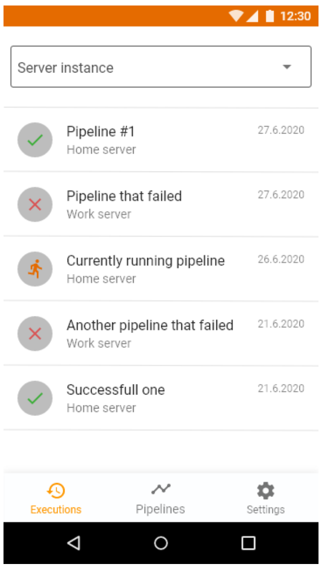
3.4 Conclusion about the main navigation

The navigation drawer will not be used, because our application does not require five or more main screens nor a hierarchical menu. Also, with the increasing sizes of mobile phones and most people being right-handed, it is hard to reach the hamburger menu with the right thumb. There will be lists of items displayed on each of the three main screens. Those items will be swipeable and having swipeable items on top of swipeable navigation would cause confusion. Material Design states that the recommended number of links in the bottom navigation is three to five [9]. The bottom navigation satisfies our needs and will be used for the main navigation. The three main screens, containing the bottom navigation, can be seen in Figure 3.1, Figure 3.2 and Figure 3.3.

3.5 Lists

Each of the three main screens will display some sort of list. For the execution screen it is a list of executions, for the pipeline screen it is a list of pipelines and for the settings screen it is a list of server instances.

All of those lists will have one thing in common and that being the swipe gesture. When users swipe an item to the left or to the right, the item will be deleted. This can be seen in Figure 3.4. Users will have the ability to undo



Pipeline #1 — Home server — 27.6.2020

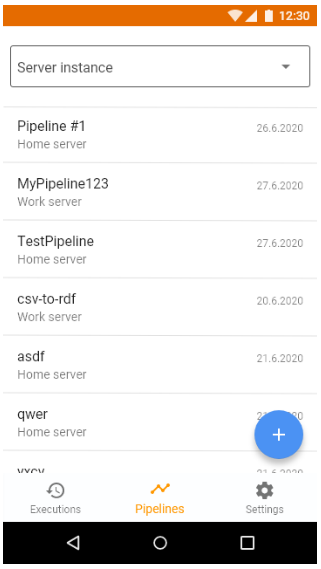
Pipeline that failed — Work server — 27.6.2020

Currently running pipeline — Home server — 26.6.2020

Another pipeline that failed — Work server — 21.6.2020

Successfull one — Home server — 21.6.2020

Figure 3.1: History screen design



Pipeline #1 — Home server — 26.6.2020

MyPipeline123 — Work server — 27.6.2020

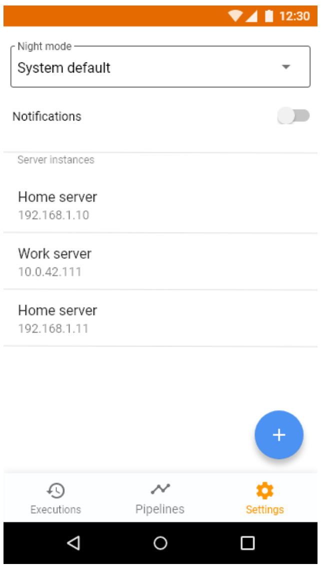
TestPipeline — Home server — 27.6.2020

csv-to-rdf — Work server — 20.6.2020

asdf — Home server — 21.6.2020

qwer — Home server

Figure 3.2: Pipelines screen design



Night mode: System default

Notifications

Server instances

Home server — 192.168.1.10

Work server — 10.0.42.111

Home server — 192.168.1.11

Figure 3.3: Settings screen design

this operation for a short period of time. The undo option can be seen in Figure 3.5.

Tapping on an item from the pipeline screen will open the edit pipeline screen. Long click on item from execution screen or from pipeline screen will launch the pipeline.

3.6 Edit server instance screen

While registering a new server instance or editing an already registered one, the application needs the address for communication and some name for labeling and better organising. Users will be able to add a description of the instance, so that there is no pressure to store every information about the instance in the server name. There could also be an option to ping the server (F-2.6, section 1.1) to verify the address and a way to cancel the registration/edit. Because of this, another screen, just for registering/editing server instances, will be added and can be seen in Figure 3.6.

3.7 Edit pipeline screen

According to the F-4.3 requirement, described in section 1.1, there has to be a screen for editing pipelines. This screen will be displaying pipeline components and drawing links between them. The preliminary design of this screen can be seen in Figure 3.7

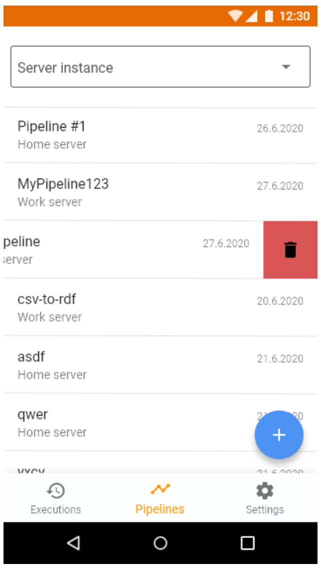
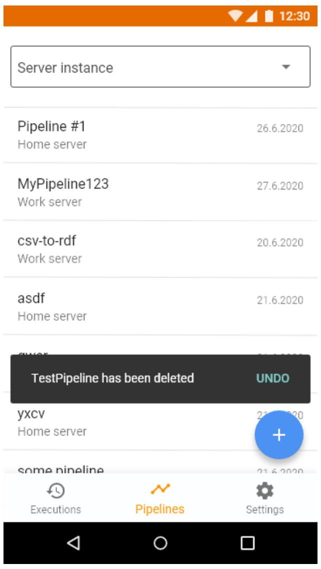
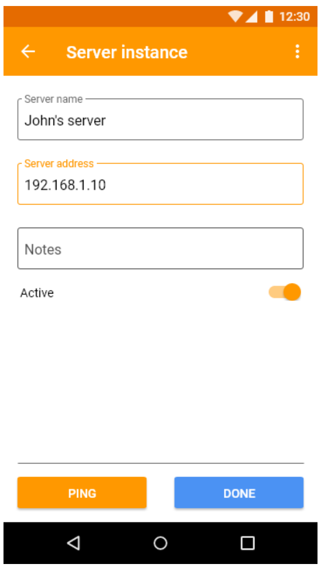


Figure 3.4: Deleting pipeline design



TestPipeline has been deleted UNDO

Figure 3.5: Undo option design



Server name: John's server

Server address: 192.168.1.10

Notes

Active

Figure 3.6: Edit server instance screen design

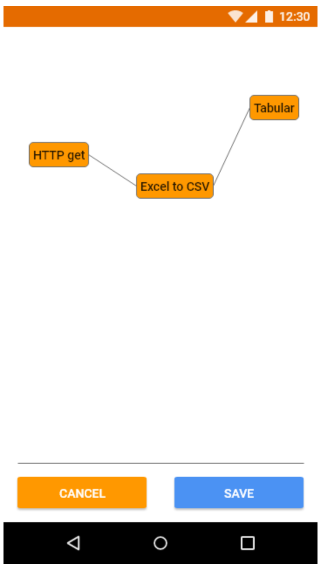
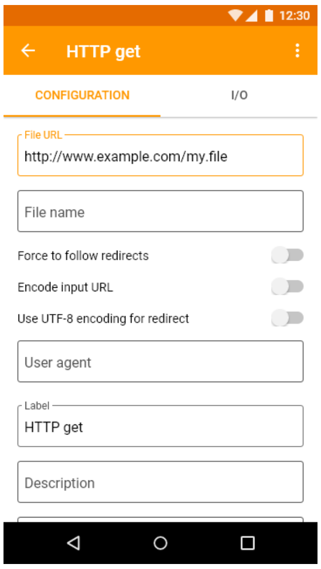


Figure 3.7: Edit pipeline screen design



File URL: http://www.example.com/my.file

File name

Force to follow redirects

Encode input URL

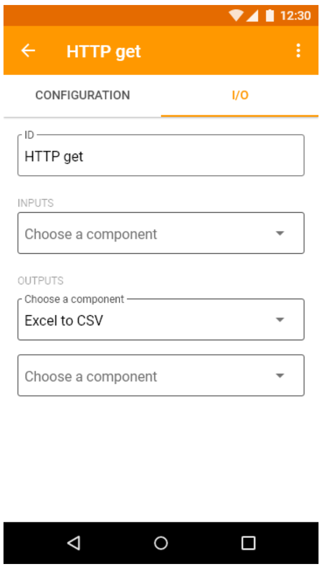
Use UTF-8 encoding for redirect

User agent

Label: HTTP get

Description

Figure 3.8: Edit component screen 1 design



ID: HTTP get

INPUTS

Choose a component

OUTPUTS

Choose a component: Excel to CSV

Choose a component

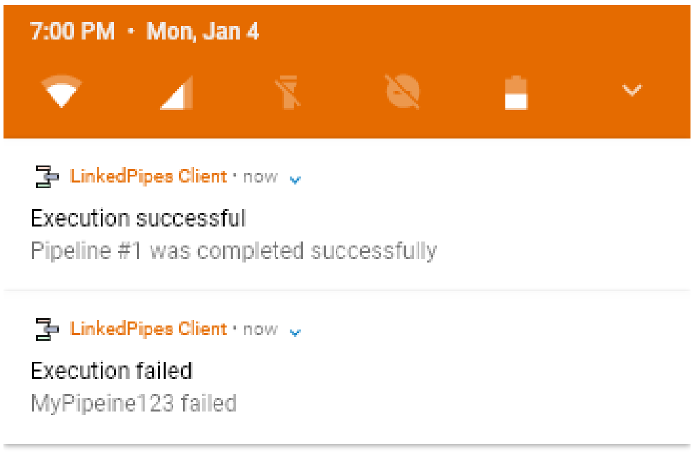
Figure 3.9: Edit component screen 2 design

3.8 Edit component screen

This screen has to be created, because each pipeline's component has its own settings. In Figure 3.8 and Figure 3.9 is the preliminary design of this screen.

3.9 Notifications

According to the F-3.1 requirement, described in section 1.1, notifications need to be implemented. The preview of notifications can be seen in Figure 3.10.



7:00 PM • Mon, Jan 4

LinkedPipes Client • now

Execution successful

Pipeline #1 was completed successfully

LinkedPipes Client • now

Execution failed

MyPipeine123 failed

Figure 3.10: Notification design

Chapter 4

Architecture Design

The architecture of the application will be described in this chapter.

4.1 Software architecture patterns

Every non trivial software project should follow some architecture design so it always remains maintainable and expandable as much easily as possible. The project should be modularized, so that any part could be replaced without the need of rewriting the entire project. And for this reason, there exist architecture patterns.

4.1.1 Basic patterns

The three basic software architecture patterns will be described here. The conclusion on which one to pick for our application will be made afterwards.

All of these patterns have one thing in common, that being structuring code into three main layers. The names of those layers are present in these pattern's names.

4.1.1.1 MVC

Model View Controller. View is responsible for displaying data, controller is responsible for getting the user input and model for storing and serving data. The controller takes user input, it updates the model and then tells the view to update itself. [10]

On the Android platform, the part of application responsible for displaying data is also responsible for dealing with user input. The pattern can be modified the way that the view gets the user input, sends it to the controller, the controller updates the model and then informs the view to update itself, based on the data from the model. [10]

The MVC and it's modified version can be seen in Figure 4.1.

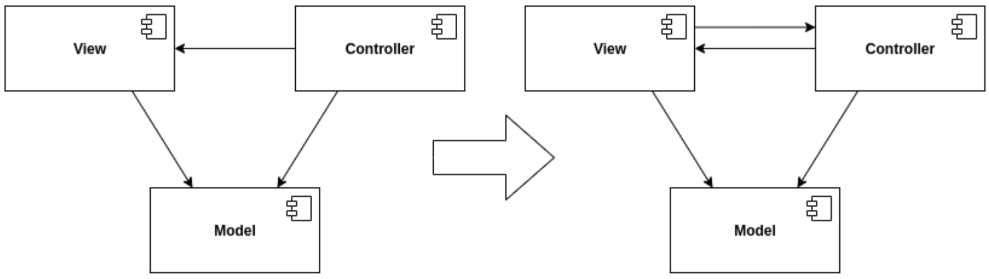


Figure 4.1: Model View Controller

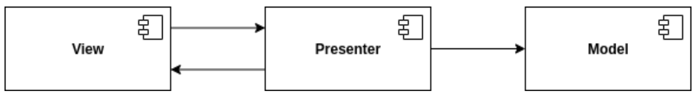


Figure 4.2: Model View Presenter

So both view and controller know the model, the view knows the controller and the controller knows the view. That is high consistency and that is a thing to be avoided, especially in the Android world, where forgetting to remove a link can and will cause memory leaks.

And what if the data should be somehow transformed for presentation? What part should do this presentation transformation, also known as UI logic? This logic should not be pushed to the view, because it's purpose is only to display the given data. The controller and the model also can not possess it, because the controller does not supply any data to the view and the model is responsible for data storing and serving and there is no reason why it should know how to display data.

4.1.1.2 MVP

Following the MVC and the UI logic problem. What if the view does not communicate with the model, but only communicates with the controller and the controller was also responsible for taking data from the model and supplying them to the view. The UI logic could be put in this new controller. This new controller will be called presenter, instead of new controller. And that is what **Model View Presenter** is (see Figure 4.2). [10]

But that means the presenter still holds a link to the view and some repetitive code would have to be written because of this. [11]

Also, the presenter should not know what parts of the view should be updated after the data changes, because displaying data is not his job.

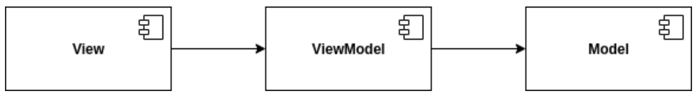


Figure 4.3: Model View ViewModel

4.1.1.3 MVVM

Model View Viewmodel. When following the MVP, the presenter knows the view and is executing the view's methods when the view needs to be updated. The presenter will no longer know it's view. Instead it will provide some kind of stream and the view will be able to observe the stream, so it can display the data whenever the data changes. This new presenter will be called viewmodel.

The view now knows the viewmodel, can call it's methods on user input and observe it's data streams. The viewmodel knows the model, observes it's data streams and omits them to the view. The model does not know the view or the viewmodel. This architecture is displayed in Figure 4.3.

4.1.2 Conclusion about picking the pattern

MVVM suits applications for the Android platform the best. Google even made some libraries to support MVVM.

4.2 Main layers and libraries

The three main layers of the MVVM will be described here, including a new fourth layer separated from the model. Some libraries that will be used inside of those layers will also be described here. The diagram of the architecture can be seen in Figure 4.4.

4.2.1 View

As stated before, the view is the layer responsible for displaying data. The way it is implemented is through a combination of standard Kotlin code and XML. The XML is mostly used to form the display structure and Kotlin is mostly used to determine what data the view should display, how to update itself and what to do with user input.

In order to work this way, XML and Kotlin have to be somehow linked together. A library called Data Binding will be used for this purpose. Compared to other solutions, like Kotlin synthetics, Data Binding offers more features and has a more robust outlook.

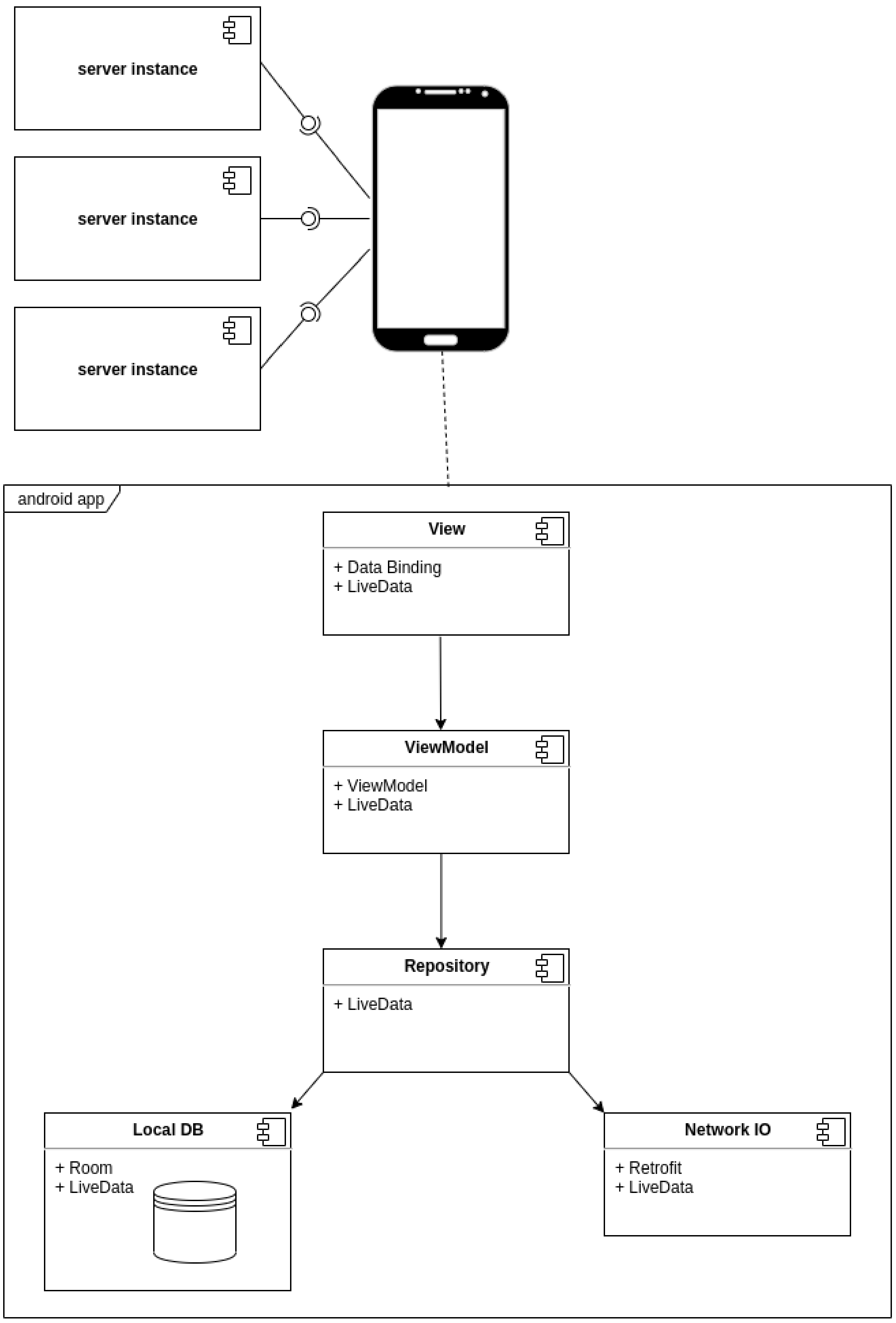


Figure 4.4: Diagram consisting of MVVM and server instances

4.2.2 Viewmodel

Google has made some architecture components and one of them is exactly for viewmodel. There is no better way to represent this layer, so this component will be used in our application.

4.2.3 Repository

Repository will represent the part of the model responsible for caching and maintaining data stored in the application's local database through communication with the following layer.

4.2.4 DAO and network IO

This will be the fourth layer responsible for storing and loading data and communicating over the internet.

Storing and loading data is a common problem with already existing solutions. These solutions are called persistence libraries. One persistence library will be chosen for our application, so the data maintenance does not have to be implemented again. The three popular persistence libraries for the Android platform are Room [12], Realm [13] and ObjectBox [14]. Out of these three, only Room offers the usage of custom written SQL commands, thus providing more flexibility and will be used for our application.

Communicating over the internet is also a common problem, so an existing library will be chosen. Between commonly used HTTP libraries, not specialised for images, belong Volley [15] and Retrofit [16]. Volley is more of a HTTP client and Retrofit makes it very easy to adapt to REST APIs. Our application will be communicating with the LinkedPipes ETL's REST API and Retrofit is a more fitting solution.

4.2.5 Observable data structure

Previously in subsubsection 4.1.1.3, "some kind of stream" was mentioned. Libraries suitable for this "some kind of stream" are LiveData [17] or RxJava [18]. Compared to RxJava, LiveData misses some features like working on a background thread and is a bit more complex to use, but LiveData is lifecycle-aware. Views have something called a lifecycle. Once they are not on screen, all links to them need to be removed, otherwise memory leaks occur. LiveData objects respect these lifecycles, so no additional repetitive code has to be written. Also, LiveData objects are not really streams, because they do nothing when there is no observer. If content changes multiple times between the screen refresh rate, the view gets only the latest change. The "some kind of stream" will be represented by LiveData.

4.3 Conclusion about software architecture

Repository will communicate with the DAO and Network layer in order to download, store and load data. Viewmodel will communicate with the repository and offer data ready to display to the view layer. View layer will display data and react to user input by forwarding it to the viewmodel. Passing data to the view layer will be handled using LiveData. The summary can be seen in Figure 4.4

CHAPTER **5**

Implementation

In this chapter, the most important libraries used while implementing our application, will be described. Besides libraries, the undo algorithm will be also described at the end of this chapter.

It is advantageous to use libraries when programming, because some parts that are needed to be implemented in the application have already been implemented by someone else and may be available in the form of libraries. Using these solutions not only saves time, but it also prevents creating bugs while trying to implement these parts of code.

5.1 Room

Room is a persistence library that was decided to be used in our application in subsection 4.2.4.

Room makes it easier to work with an application's internal database. For each entity stored in the database, an entity class is defined, which is just a standard Kotlin class with some annotations. The simplest entity class looks as follows. The whole class is annotated with `@Entity` and primary key is annotated with `@PrimaryKey`. In case there is a need for multiple primary keys, they can be listed in the `@Entity` annotation. DAO classes are annotated with `@Dao` and are used to store and load objects to and from the database. Creating DAO classes is done by declaring abstract methods with annotations, optionally accompanied by SQL syntax. Those methods are being implemented by Room. Entities, lists of entities or LiveData instances of either of those can be returned by the DAO's methods.

5.2 LiveData

LiveData is an observable data structure, that was decided to be used in our application in subsection 4.2.5

View components can observe this observable structure by registering instances of `Observer`. When registering an `Observer`, the view have to pass an instance of it's lifecycle, so the LiveData instance can detect when to remove this `Observer`, so no memory leaks are created and so the performance stays the same. Previously mentioned library Room can also return LiveData instances, so observers can react to database changes directly, without any other unnecessary code.

5.3 Retrofit

Retrofit is a HTTP library specialized for implementing REST API clients, that has been decided to be used in our application in subsection 4.2.4.

An abstract method is declared for each API call, complemented with an annotation with HTTP request details. Retrofit also uses the OkHttp [19] library, where the basic authentication can be arranged.

5.4 Coroutines

Coroutines are Kotlin's way to improve multithreaded programming. Compared to Java, Kotlin introduces a new keyword `suspend` that can be written in front of methods. Methods with the `suspend` keyword can not be called the normal way, but can be called from other `suspend` methods or launched via higher order methods of `CoroutineScope`. Although `suspend` is a Kotlin's keyword, dependency for a library is needed in order to work with coroutines and that is why Coroutines are described here, between libraries.

5.5 Scheduling of asynchronous tasks

Service is a part of an application that can run in background and can be started even when the application is not running. Our application needs services for checking execution statuses. Scheduling services across multiple Android versions can be tricky, so using a library for his purpose is a good option. Popular solutions for scheduling services are Android-Job [20] and WorkManager [21]. The readme file of Android-Job states that it is deprecated and that WorkManager should be used instead [20]. Because of that, WorkManager will be used for scheduling services.

5.6 QR code scanner

There is a need to implement a QR code scanner, because of section 1.1. ZXing library [22], Mobile Vision [23], or Varvet's QR code scanner [24] can be used.

When using the ZXing library, a third party application needs to be downloaded to the phone, which can be unpleasant. Mobile Vision is a new library from Google that allows scanning QR codes without the need of a third party application. Varvet's QR code scanner is a library based on Google's Mobile Vision. For the purpose of reading QR codes, the Varvet's QR code scanner is easier to implement than the Mobile Vision library.

Varvet's QR code scanner will be used for scanning QR codes in our application.

5.7 Draggable Views

Pipeline's components should be freely movable across the canvas while editing the pipeline. That can be achieved by making some regular view (button or image) movable. This is often achieved by a lot of unnecessary code and that is why a library will be used for this purpose.

The only library suited for this purpose, that has been found, is DraggableView by hyuwah [25]. It allows programmers to turn any view into a draggable view, which means it can be moved with a finger.

5.8 Undo operations

In chapter 1 are some undo options in planning. The user should be able to undo the deletions of pipelines and executions. It can not be done just by scheduling the sending of the delete request. The application will create and store a mark for every deleted item alongside with scheduling the delete request (see Figure 5.1). Because of that, when the application is killed for any reason before the delete request is sent, the application can check all of those marks at the application launch and send the delete requests then (see Figure 5.2). Marks can also be used for filtering pipelines and executions that will be shown to the user. If the user undo the deletion, the mark is deleted and the scheduling of the delete request is cancelled (see Figure 5.3).

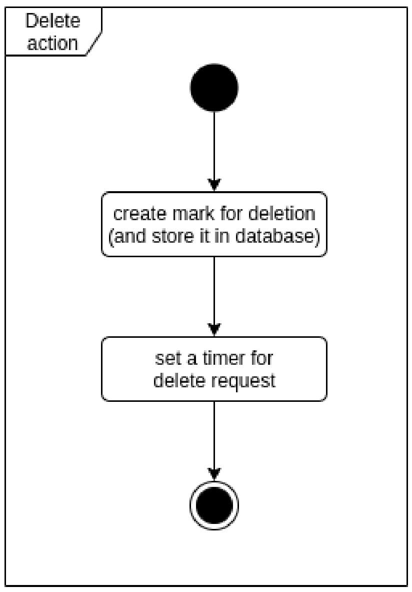

Delete action
create mark for deletion (and store it in database)
set a timer for delete request

Figure 5.1: Delete action

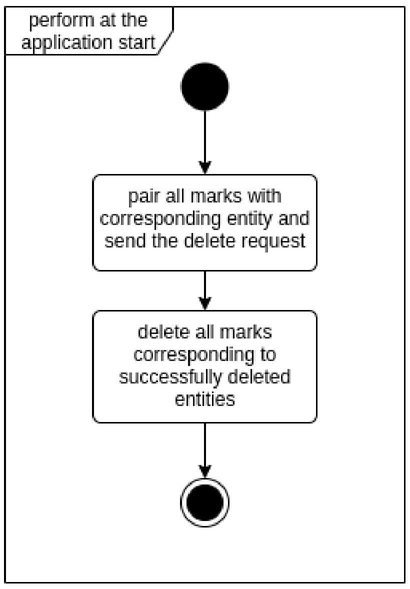

perform at the application start
pair all marks with corresponding entity and send the delete request
delete all marks corresponding to successfully deleted entities

Figure 5.2: Finish interrupted delete actions

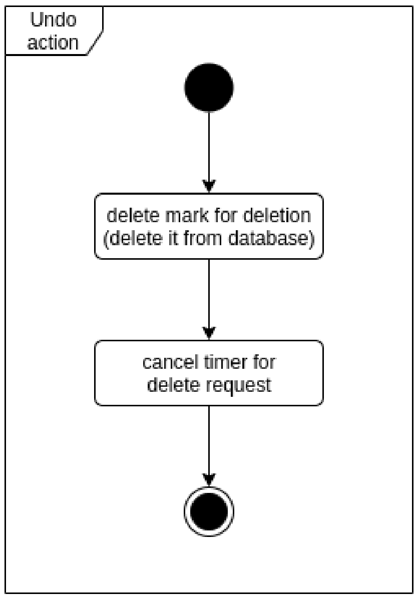

Undo action
delete mark for deletion (delete it from database)
cancel timer for delete request

Figure 5.3: Undo action

Chapter 6

Documentation

This section describes user documentation, developer documentation, system requirements and deployment.

6.1 UI Documentation

A documentation containing screenshots with descriptions (see Figure 6.1) has been created and is available at the front page of the application's github repository, alongside with a link to video tutorials going through previously written use cases. The project can be found at `https://github.com/Palda97/LinkedPipesAndroidClient`.

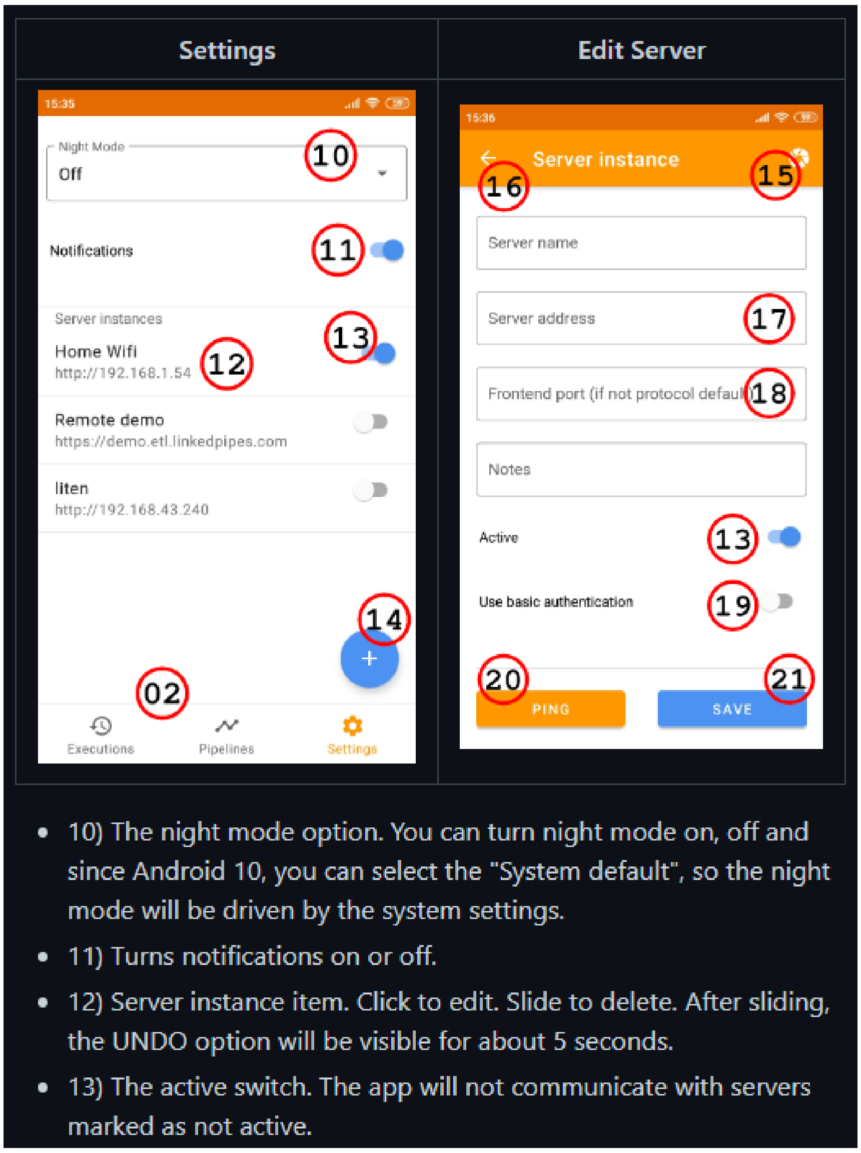
6.2 Developer documentation

In Kotlin, there is a documenting type of comment called KDoc (see Figure 6.2). With a plugin called Dokka [26], it is possible to construct a website based on the KDoc comments, documenting the application's code. An example of a part of a class, documented via javadoc can be seen in Figure 6.3.

The second part of developer documentation has been written by hand, explaining the internal operations from a greater distance, so other developers can have a more bearable understanding of the application when they follow up on development.

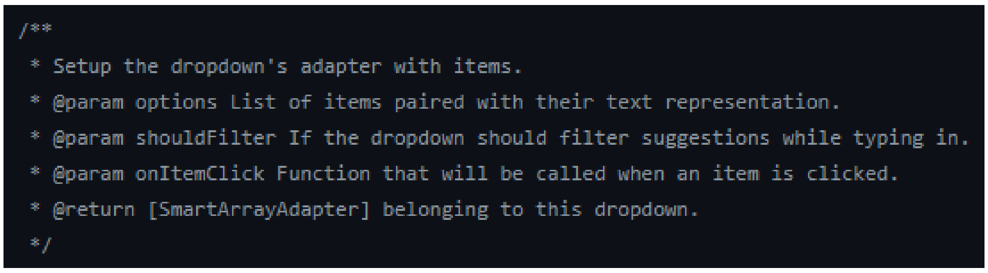
Section "Technologies" describes technology needed for the application development, such as IDE and SDK. Section "Server instances" contains links to an online demo server and to the LinkedPipes ETL github page. In the "Basic code overview", MVVM is mentioned and briefly described alongside two important source code files. Then there are the headings "model" "viewmodel" and "view", each with its own explanation, especially the "model", because it is the most extensive one.

Part of the handwritten developer documentation can be seen in Figure 6.4



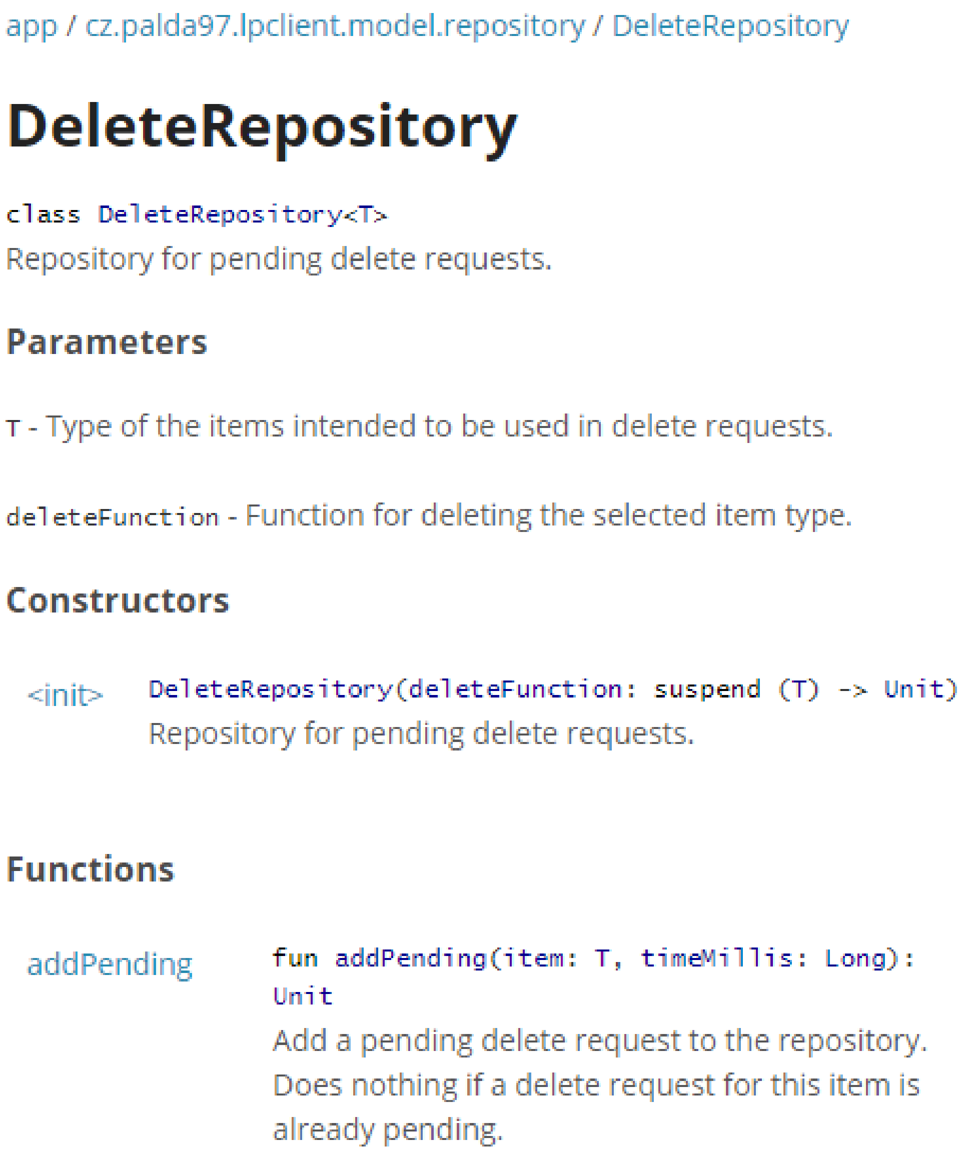
- 10) The night mode option. You can turn night mode on, off and since Android 10, you can select the "System default", so the night mode will be driven by the system settings.
- 11) Turns notifications on or off.
- 12) Server instance item. Click to edit. Slide to delete. After sliding, the UNDO option will be visible for about 5 seconds.
- 13) The active switch. The app will not communicate with servers marked as not active.

Figure 6.1: UI documentation



```
/**
 * Setup the dropdown's adapter with items.
 * @param options List of items paired with their text representation.
 * @param shouldFilter If the dropdown should filter suggestions while typing in.
 * @param onItemClick Function that will be called when an item is clicked.
 * @return [SmartArrayAdapter] belonging to this dropdown.
 */
```

Figure 6.2: KDoc comment



app / cz.palda97.lpclient.model.repository / DeleteRepository

DeleteRepository

`class DeleteRepository<T>`

Repository for pending delete requests.

Parameters

`T` - Type of the items intended to be used in delete requests.

`deleteFunction` - Function for deleting the selected item type.

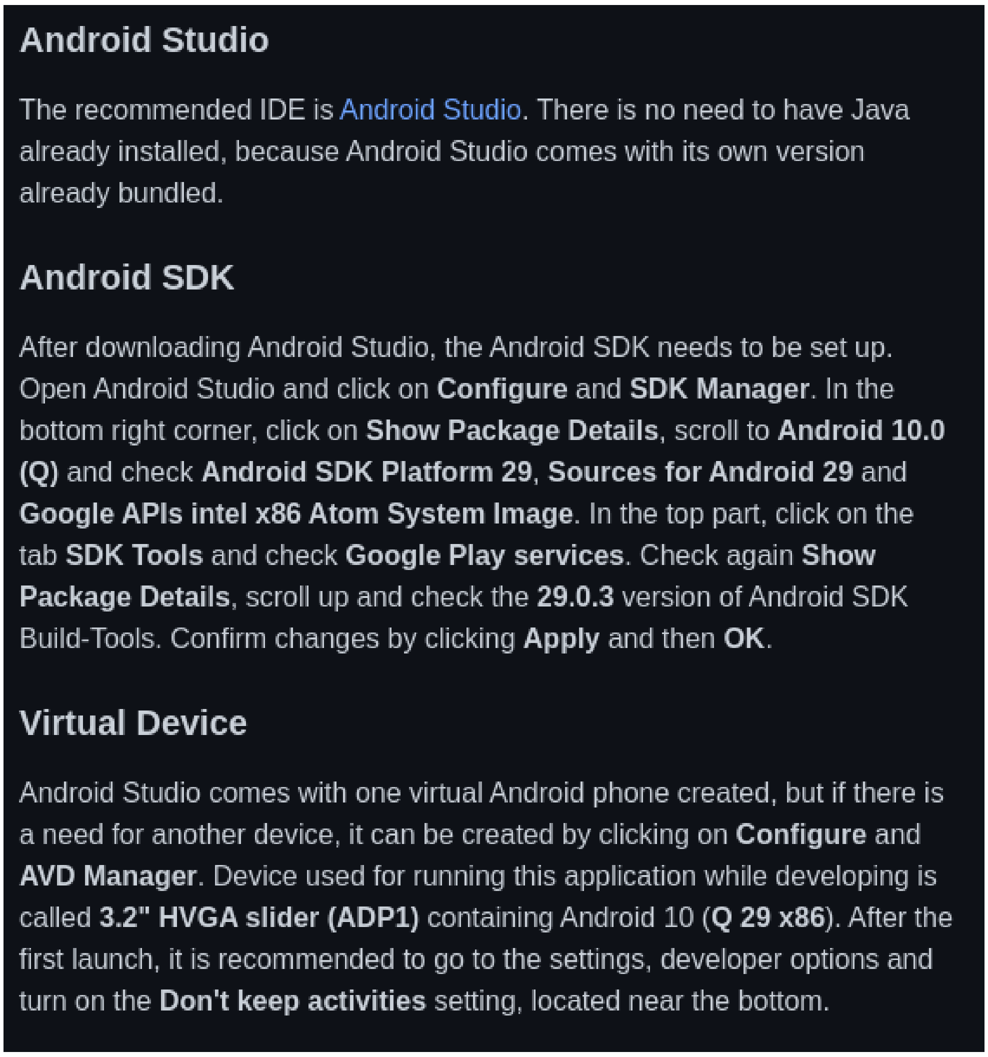
Constructors

<init> `DeleteRepository(deleteFunction: suspend (T) -> Unit)`
Repository for pending delete requests.

Functions

addPending `fun addPending(item: T, timeMillis: Long): Unit`
Add a pending delete request to the repository. Does nothing if a delete request for this item is already pending.

Figure 6.3: Example of part of a javadoc page



Android Studio

The recommended IDE is Android Studio. There is no need to have Java already installed, because Android Studio comes with its own version already bundled.

Android SDK

After downloading Android Studio, the Android SDK needs to be set up. Open Android Studio and click on **Configure** and **SDK Manager**. In the bottom right corner, click on **Show Package Details**, scroll to **Android 10.0 (Q)** and check **Android SDK Platform 29**, **Sources for Android 29** and **Google APIs intel x86 Atom System Image**. In the top part, click on the tab **SDK Tools** and check **Google Play services**. Check again **Show Package Details**, scroll up and check the **29.0.3** version of Android SDK Build-Tools. Confirm changes by clicking **Apply** and then **OK**.

Virtual Device

Android Studio comes with one virtual Android phone created, but if there is a need for another device, it can be created by clicking on **Configure** and **AVD Manager**. Device used for running this application while developing is called **3.2" HVGA slider (ADP1)** containing Android 10 (**Q 29 x86**). After the first launch, it is recommended to go to the settings, developer options and turn on the **Don't keep activities** setting, located near the bottom.

Figure 6.4: Part of the handwritten developer documentation

Link to the developer documentation is accessible from the application's github page.

6.3 System requirements

The application requires Android 5.0 or greater. It needs at least 40 MB of space. An internet connection is needed in order to communicate with servers, but it is not needed for communication with servers on the local network. Camera is needed for loading server information via QR code.

The system requirements are also available from the front page of the application's github page.

6.4 Deployment

Google Play is used for the deployment of this application. It is an Android application store that is already installed on most Android phones. Using Google Play is far better than just downloading and installing the app from some storage. Some of the advantages are simplicity of updating and the option to leave reviews. The application can be downloaded at `https://play.google.com/store/apps/details?id=cz.palda97.lpclient`.

Chapter 7

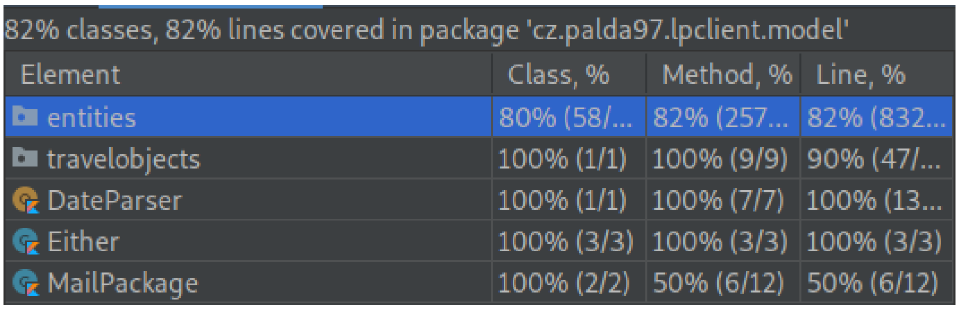
Tests

This chapter will cover types of tests that were used in this project. It is a good practise to write special code that can check whether parts of the application work as intended. This code is simply called tests. This is not only useful to test the application parts when they are being written, but these tests could be run in future to ensure that possible code changes did not break the functionality of previously written parts.

7.1 Local unit tests

These are tests that require only JVM. They do not need any part of the Android framework, so there is no need to run them on an Android device, which makes them fast. Parsing objects from JSON and back can and is tested here.

The model layer is tested here, but since the application's database is not accessible here, repository and DAOs are excluded from these tests, alongside with some generated code, like from the Retrofit library. The coverage, as displayed in Figure 7.1, is 82 % for classes and 82 % for lines of code. Coverage of 70 % to 80 % is commonly considered good coverage.



82% classes, 82% lines covered in package 'cz.palda97.lpclient.model'

Element	Class, %	Method, %	Line, %
entities	80% (58/...	82% (257...	82% (832...
travelobjects	100% (1/1)	100% (9/9)	90% (47/...
DateParser	100% (1/1)	100% (7/7)	100% (13...
Either	100% (3/3)	100% (3/3)	100% (3/3)
MailPackage	100% (2/2)	50% (6/12)	50% (6/12)

Figure 7.1: Picture with local test coverage

7.2 Instrumented unit tests

Instrumented tests require some part of the Android framework, so they run in background on an Android device. Previously mentioned library Room (in subsection 4.2.4) requires a part of Android framework, so it can be used here. Repositories are tested here too.

Unfortunately, the coverage could not be put into operation here. After researching how coverage can be done on instrumented tests, most of the tutorials are bound to various forks of the JaCoCo library [27], which caused some errors during commissioning attempts.

Conclusion

The goals were a list of requirements, a look at existing solutions, design of the application, the application itself, documentation and testing.

- Use cases, scenarios and requirements were put together in the analysis chapter.
- The existence of no current solution for our requirements was verified in the existing solutions chapter.
- The UI was designed in the UI design chapter and the software architecture was designed in the architecture design chapter.
- The application was created during the implementation chapter.
- The documentations were created and described in the documentation chapter.
- The application tests were described in the test chapter.

In the future, the application may be improved with a better UI. Some components are currently not supported and in the future, it could be solved by a cooperation of the application developer(s) and the server developers.

Bibliography

[1] Berners-Lee, T. Linked Data - Design Issues. [online], [cit. 2020-12-06]. Available from: `https://www.w3.org/DesignIssues/LinkedData.html`

[2] Klímek, J.; Skoda, P. LinkedPipes ETL in use: practical publication and consumption of linked data. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2017, Salzburg, Austria, December 4-6, 2017*, edited by M. Indrawan-Santiago; M. Steinbauer; I. L. Salvadori; I. Khalil; G. Anderst-Kotsis, ACM, 2017, pp. 441–445, doi: 10.1145/3151759.3151809. Available from: `https://doi.org/10.1145/3151759.3151809`

[3] Škoda, P.; Klímek, J.; et al. LinkedPipes ETL REST API. In: Github [online], GitHub Inc., Oct. 2017, [cit. 2020-12-06]. Available from: `https://github.com/linkedpipes/etl/wiki/LinkedPipes-ETL-REST-API`

[4] Twitter, Inc. Bootstrap · The most popular HTML, CSS, and JS library in the world. [online], [cit. 2020-12-06]. Available from: `https://getbootstrap.com/`

[5] Bearded, H. Android-Bootstrap. In: Github [online], GitHub Inc., Dec. 2016, [cit. 2020-12-06]. Available from: `https://github.com/Bearded-Hen/Android-Bootstrap`

[6] Microsoft Corporation. Microsoft Design - Android. [online], [cit. 2020-12-06]. Available from: `https://www.microsoft.com/design/fluent/#/android`

[7] Google LLC. Develop - Android - Material Design. [online], [cit. 2020-12-06]. Available from: `https://material.io/develop/android`

[8] Google LLC. Navigation drawer - Material Design. [online], [cit. 2021-04-10]. Available from: https://material.io/components/navigation-drawer#usage

[9] Google LLC. Bottom navigation - Material Design. [online], [cit. 2021-04-10]. Available from: https://material.io/components/bottom-navigation#usage

[10] Muntenescu, F.; Touchlab. A Journey Through MV Wonderland (updated). Youtube [video], Youtube LLC, Nov. 2016, [cit. 2020-12-06]. Available from: https://youtu.be/QrbhPcbZv0I

[11] Leijdekkers, P.; ANZ Coders. Android MVP vs MVVM and the winner is. Youtube [video], Youtube LLC, Feb. 2018, [cit. 2020-12-06]. Available from: https://youtu.be/ugpC98LcNqA

[12] Google LLC. Save data in a local database using Room — Android Developers. [online], [cit. 2020-12-06]. Available from: https://developer.android.com/training/data-storage/room

[13] MongoDB, Inc. Home — Realm.io. [online], [cit. 2020-12-06]. Available from: https://realm.io

[14] ObjectBox Ltd. ObjectBox - Edge Database for Mobile and IoT. [online], [cit. 2020-12-06]. Available from: https://objectbox.io/

[15] Google LLC. Volley overview — Android Developers. [online], [cit. 2020-12-06]. Available from: https://developer.android.com/training/volley

[16] Square, Inc. Retrofit. [online], [cit. 2021-04-10]. Available from: https://square.github.io/retrofit

[17] Google LLC. LiveData Overview — Android Developers. [online], [cit. 2021-04-10]. Available from: https://developer.android.com/topic/libraries/architecture/livedata

[18] RxJava Contributors. RxJava: Reactive Extensions for the JVM. In: Github [online], GitHub Inc., [cit. 2020-12-06]. Available from: https://github.com/ReactiveX/RxJava

[19] Square, Inc. OkHttp. [online], [cit. 2020-12-06]. Available from: https://square.github.io/okhttp

[20] Evernote Corporation. Android-Job. In: Github [online], GitHub Inc., Oct. 2019, [cit. 2020-12-06]. Available from: https://github.com/evernote/android-job

[21] Google LLC. Schedule tasks with WorkManager — Android Developers. [online], [cit. 2020-12-06]. Available from: `https://developer.android.com/topic/libraries/architecture/workmanager`

[22] ZXing authors. ZXing. In: Github [online], GitHub Inc., Sept. 2020, [cit. 2020-12-06]. Available from: `https://github.com/zxing/zxing`

[23] Google LLC. Mobile Vision — Google Developers. [online], [cit. 2020-12-06]. Available from: `https://developers.google.com/vision`

[24] Varvet. Android QR Code Reader Made Easy — Varvet. [online], [cit. 2021-04-10]. Available from: `https://www.varvet.com/blog/android-qr-code-reader-made-easy`

[25] Wahyudin, M. DraggableView. In: Github [online], GitHub Inc., May 2020, [cit. 2021-04-10]. Available from: `https://github.com/hyuwah/DraggableView`

[26] JetBrains s.r.o. and Dokka project contributors. Kotlin/dokka: Documentation Engine for Kotlin. In: Github [online], GitHub Inc., May 2018, [cit. 2021-04-10]. Available from: `https://github.com/Kotlin/dokka`

[27] Gradle Inc. The JaCoCo Plugin. [online], [cit. 2021-04-10]. Available from: `https://docs.gradle.org/current/userguide/jacoco_plugin.html`

Appendix A

List of abbreviations used

API	Application Programming Interface
DAO	Data access object
GUI	Graphical user interface
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JVM	Java virtual machine
MVC	Model View Controller
MVP	Model View Presenter
MVVM	Model View Viewmodel
REST	Representational State Transfer
SDK	Software development kit
UI	User interface
URI	Uniform Resource Identifier
XML	Extensible markup language

Appendix B

Contents of the enclosed CD

```
│ devDoc.pdf ...................... handwritten developer documentation
├── doc/ ............................. developer documentation repository
│   ├── index.html ..................... developer documentation (javadoc)
│   ├── README.md ............. handwritten developer documentation source
│   └── videos.html ................................ video tutorials online
├── ETLClient.apk .................................... application package
├── LinkedPipesAndroidClient/ ................. application's source codes
│   └── README.md ............ UI documentation and OS requirements source
├── readme.txt ................. brief description of the contents of the CD
├── thesis/ ............................ LaTeX source codes of this thesis
├── thesis.pdf ...................................... this thesis as a PDF
├── uiDoc.pdf .................... UI documentation and OS requirements
└── videos/ ............................................. video tutorials
```