UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Kristofer Käosaar

Analysis of Common User Flows in Open-Source Android Applications

Master's Thesis (30 ECTS)

Supervisor: Kristiina Rahkema, MSc

Tartu 2023

Analysis of Common User Flows in Open-Source Android Applications

Abstract:

Many project templates for Android applications can be found on GitHub, but few contain implementations of whole user flows. This thesis focuses on extracting representations of user flows in the form of activity diagrams from existing open-source Android applications, with the goal to develop a project template featuring user flows such as onboarding, user creation and signing in. The activity diagrams of each user flow were combined with the intention of creating a representation of an average user flow that could be applied to most use cases. The project template was developed based on those combined activity diagrams in Android Studio using Kotlin and following a Model-View-ViewModel architecture. The project template was evaluated by recreating user flows of six open-source Android applications and measuring how much of the codebase needed to be changed to achieve a similar user interface. The result indicates that systematically analysing the user flows of existing applications is a useful starting point for creating easily adaptable project templates.

Keywords:

Android, Kotlin, template, MVVM, activity diagram, user flow

CERCS: P170 (Computer science, numerical analysis, systems, control)

Taaskasutatud Kasutajavoogude Analüüs Avatud Lähtekoodiga Androidi Rakendustes

Lühikokkuvõte:

GitHubist on leida mitmeid Androidi rakenduste projektimalle, kuid vähesed sisaldavad tervete kasutajavoogude implementatsioone. See lõputöö keskendub avatud lähtekoodiga Androidi rakendustest kasutajavoogude eraldamisele tegevusdiagrammide kujul, eesmärgiga arendada projektimall, mis sisaldab kasutajavooge nagu rakenduse tutvustus, kasutaja loomine ja sisselogimine. Kasutajavoogude tegevusskeemid kombineeriti eesmärgiga luua kasutajavood, mida saaks rakendada enamiku kasutusjuhtude puhul. Projekti mall arendati kombineeritud tegevusskeemide põhjal Android Studios, kasutades Kotlinit ja järgides Model-View-ViewModel arhitektuuri. Projekti malli hinnati kuue avatud lähtekoodiga Androidi rakenduse kasutajavoogude uuesti loomisega ja mõõdeti, kui suurt osa koodibaasist on vaja muuta sarnase kasutajaliidese saavutamiseks. Tulemus näitab, et olemasolevate rakenduste kasutajavoogude süstemaatiline analüüsimine on kasulik lähtepunkt kergesti kohandatavate projektimallide loomisel.

Võtmesõnad:

Android, Kotlin, mall, MVVM, tegevusskeem, kasutajavoog

CERCS: P170 (Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine)

Table of Contents

1 Introduction

Mobile applications follow similar design patterns, due to design trends and the limitations of similarly sized touch screens on phones [1]. Many applications contain components such as a user onboarding flow, where users are shown an introduction to the functions of the application, a sign-in or sign-up screen, or a screen which displays an enumerated list of items [2]. Getting started with writing a new application from scratch can be challenging, especially for new developers. In such cases, application templates can serve as a useful starting point and help establish good architecture patterns.

The main goal of this thesis is to find a way to combine activity diagrams of user flows, which will be extracted from existing applications, and to test this approach analytically. Based on the result of combining these activity diagrams, at least a single user flow of an Android application will be developed as an open-source template that can be used as a guideline or starting point for other developers. The specific user flows will be chosen after using code search tools such as GitHub code search to find examples of the most commonly implemented user flows. The goals of this thesis will be achieved when some commonly reused user flows are found, an approach has been found for combining their activity diagrams, and a template project has been created based on the combined activity diagrams of the chosen user flows. Development of the template project will follow Clean Architecture and SOLID principles[1] and the project itself will be published in an open-source GitHub repository. This approach to creating a project template from combined activity diagrams will be evaluated by recreating the UI of some of the applications found in the code search part of the thesis and comparing how much code needed to be changed to achieve the same visual result.

The Android platform was chosen as the field of study for this thesis due to it being the most used mobile operating system at the time of writing and having considerably more available open-source projects than the iOS platform [3]. This thesis is being written under the assumption that project templates containing pre-made user flows, such as the ones that will be created in the process of this thesis, are sought after by Android developers, as common user flows are often programmed over and over again, specifically in companies that are service-based and develop large amounts of custom software projects for other companies. At the time of writing, such tools and templates exist, but there are none that could be found,

[1] https://www.baeldung.com/solid-principles

which are based on a study of existing projects and their common user flows. Additionally, existing templates might not always be kept up to date with the newest updates to libraries or new toolkits such as Jetpack Compose[2].

This thesis consists of seven sections. Section 2 discusses the basics of Android applications as well as covers related work and the basics of mobile application design. Section 3 describes the method of the study, detailing the code search, creation of activity diagrams, specification of requirements, development of the project template and evaluation. Section 4 details the results of the code search and development. Section 5 covers the discussion of the results and finally, Section 6 summarises the work.

[2] https://developer.android.com/jetpack/compose

2 Background

This section covers related projects and research papers, design patterns and common user flows in mobile applications and concepts related to developing Android applications. Additionally, this section deals with fundamental components of Android applications and what tools can be used to find open-source repositories.

2.1 Related work

Deka proposes a semi-automated approach, using interaction mining, to analyse and map the UI and user flows of mobile applications for data-driven design purposes [4]. They describe how designing mobile applications requires a large team of dedicated workers and how a data-driven approach could simplify this process. Their approach to analysing the user flows of mobile applications coincides somewhat with the goals of this thesis project, as having a semi-automated process to map existing applications user flows could allow the activity diagram based approach proposed for this thesis project to be applied to a much larger scale. If such an approach were to be used, there would also have to be a way to automatically generate activity diagrams from their image-based interaction graphs. Having an application template based on an average of a large subset of user flows would provide a fairly universal starting point for most projects.

Akopian et al. investigate how using a template to teach Android application development could be beneficial to motivate students and speed up learning in certain situations [5]. They focus on the first stage of mobile programming, i.e. introducing students to the field, with template-based learning, which provides experience with professional tools in relatively short timeframes. While their work is not directly related to providing reusable components or user flows for professional developers, it does suggest that having a baseline to work from is helpful for both less experienced developers and people looking to get into the field of mobile development.

There are many open-source repositories that offer examples for various different user flows. Tekombo and Marajanovic's OnboardingViewPagerSamples[3] provide an example of how to introduce users to an application with animated transitions. Google's Android Conditional Navigation with Login Codelab[4] teaches readers how to lock out parts of an application to a

[3] https://github.com/gabriel-TheCode/OnboardingViewPagerExamples
[4] https://developer.android.com/codelabs/advanced-android-kotlin-training-login-navigation#0

user that is not logged in. Some tools such as Bloco's Android project template[5] can be used as a starting point to develop an Android application using modern components like Jetpack Compose. Lampropoulos' study details that while code reuse is very useful, repositories containing components and user flows, like the ones mentioned previously, become quickly outdated, since providers such as Google continuously update their tools and guidelines. He states that reusing code is generally helpful under most circumstances, but projects older than a year might already be outdated [6].

Barnett et al. created a tool that generates the scaffolding of a data-driven mobile application based on a high-level description [7]. They describe how modern mobile app IDEs do not offer much in code generation for starting the development of a new app. While their article was written in 2015, this problem persists at the time of writing, as Android Studio offers a project with a bottom navigation view as the most complex option for generating a new project.

2.2 Mobile application design

A user flow is a description of a set of tasks that a user must do to complete a process, which can branch out in different directions. Even though mobile applications can have different purposes, they sometimes share similar components and user flows [1]. If there is a need to tell the user what the application's main functions are, then a user onboarding flow is used to provide such information [2]. If the application needs the user to have an account to use specific functionalities, there must be log in and sign-up user flows where such actions can be made. Similarly, there are often lists of information that are displayed to users. In a weather application, there could be a weather report for each hour of the day, which would be displayed in a vertical list, as shown in an example in Figure 1.

[5] https://github.com/blocoio/android-template

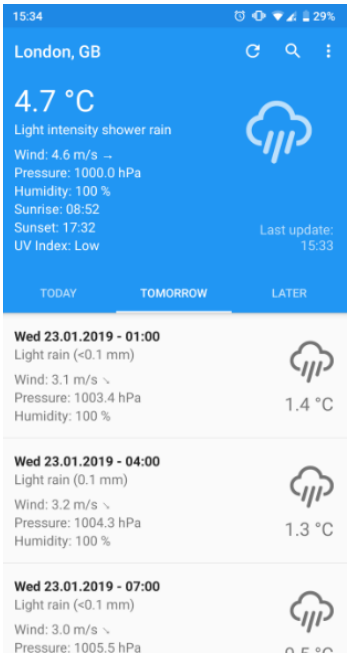


Figure 1. Weather report in the Forecastle application[6]

Erikkson's and Parflo's dissertation shows that mobile applications benefit from onboarding user flows where users are introduced to the application's functions, as perceived usefulness and attitude towards use are improved, although there was no significant difference between different onboarding patterns [8].

While many functionalities remain the same, design trends in mobile applications change over time. Eisfeld's thesis describes how Dark Mode became a standard option around 2019 in both Android and iOS devices to improve user experience in certain circumstances such as low-light environments [9].

2.3 Android application development

This subsection focuses on the fundamentals of how Android applications are structured and the best practices for developing Android applications. It also covers how user interface components are used and how they can be written in both Android's XML notation or Kotlin code.

2.3.1 Fundamentals

Android applications are most often developed using Kotlin. Kotlin was declared as the recommended programming language for Android Applications in May 2019 [10]. It has since been compared to Java in multiple studies and has been generally considered superior.

[6] https://github.com/martykan/forecastie

Reasons include its conciseness compared to Java, greater security due to supporting non-nullable types, and compatibility with Java, which allows it to be integrated into existing Java projects [11]. Google suggests Android Studio as the main integrated development environment (IDE) for working on Android applications, although alternative IDEs that use fewer resources can be used since Android applications can be compiled and built from a command line using corresponding tools and packages.

The most crucial component of an Android application is the Activity class[12]. It provides the window in which the application draws its user interface (UI). It can fill the whole screen of a phone or be overlaid on other applications. It is also from where other activities or services are launched. A service is a component of an Android application that does not have its own UI, but can perform long-running operations in the background[7]. A Fragment is a reusable portion of an application's UI[8] and must be hosted by an activity or another fragment. Google has also developed a navigation component which relies on a single-activity architecture[9], where there is only one activity and the rest of the application's UI is created using fragments, as managing the details of an applications UI is better left to a smaller, reusable part of the UI [13].

Activities and Fragments have their own lifecycle[10], which has states related to the usage of the application. There are different states for when an application is in the background, is paused, or has been resumed, as pictured in Figure 2. Due to this, writing code related to business logic into Activities or Fragments is generally discouraged[11]. The main focus of this thesis will mostly be the UI layer as user flows are created there.

[7] https://developer.android.com/guide/components/services
[8] https://developer.android.com/guide/fragments
[9] https://developer.android.com/guide/navigation/navigation-migrate
[10] https://developer.android.com/guide/components/activities/activity-lifecycle
[11] https://developer.android.com/topic/architecture/ui-layer

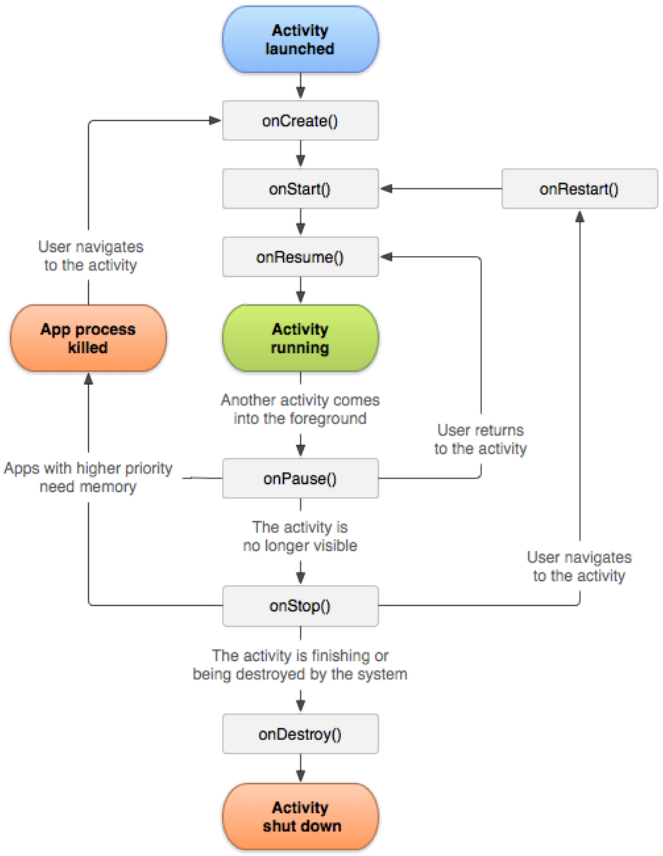


Figure 2. Android Activity Lifecycle[12]

Common pitfalls of not accounting for the lifecycles of view components might be state handling errors, such as having a variable be null upon rotating the screen, as an activity might be restarted when the orientation of a phone is changed. Google has developed multiple lifecycle-aware components to deal with such issues and has issued multiple best practices, such as having business or data logic be contained in a ViewModel, a component which will be described in the next subsection in greater detail [14].

2.3.2 Best practices & Clean Architecture

Android applications are commonly developed using architectural patterns which split the code into a UI layer, a domain layer and a data layer. This provides a valuable separation of concerns, which increases testability and quality of code [15][16]. Classes are also usually arranged into separate packages within the codebase, which signify which part of the application they are related to. For example, code related to the UI of a login user flow could be in a package named ‘appname.ui.login’.

[12] https://developer.android.com/guide/components/activities/activity-lifecycle

Clean Architecture is a set of design principles which separates parts of an application into a domain, presentation and model layer. This is not to be confused with the application architecture itself, which does not have to match Clean Architecture but can be combined with it. This is sometimes done with the Model-View-ViewModel architecture to lessen bloat in ViewModels of larger applications [17]. In such cases, some business logic for a specific action is held in a UseCase class, which is accessed from the ViewModel.

A ViewModel is a class designed to hold business logic or screen states which is commonly accessed from either activities or fragments[13]. Its main advantage is not having its lifecycle depend on changes in the UI such as the activity being rotated or being put to the background. This makes it ideal to hold UI states which depend on events happening in the data layer, such as loading. A loading state might disappear if held in a variable in an activity or a fragment, but if it is observed from a ViewModel, the UI can adjust accordingly without having to account for what is happening to the rest of the view. It is scoped to an activity or a fragment, and it is removed when an activity finishes or a fragment detaches.

2.3.3 User interface components

Structures for user interfaces for Android applications are usually defined using Layouts created in Android's XML vocabulary[14]. These layouts are then compiled into a View resource in a corresponding Activity or Fragment, by passing a reference to a specific layout to that activity or fragment. A .xml file containing a layout must contain a ViewGroup object, which acts as a container for Views, which draw specific elements that a user can interact with. Views can be accessed in a layout's corresponding activity or fragment where attributes can be modified or business logic can be created, i.e. having a ButtonView execute some code when it is clicked.

Jetpack Compose is Android's modern toolkit for building native UI. It bypasses Android's XML vocabulary by having UI components written as declarative functions in Kotlin code. It is described by Google as having less code and being more intuitive than traditional Android development using Layout files. It is compatible with existing code and applications and can be integrated with traditional layouts.

[13] https://developer.android.com/topic/libraries/architecture/viewmodel
[14] https://developer.android.com/develop/ui/views/layout/declaring-layout

2.4 Code search tools

To find the most commonly occurring components in open-source applications, platforms such as GitHub[15] and Searchcode[16] can be used to identify the previously mentioned components in repositories that have been made public. GitHub is an online hosting service for software development and version control using Git[17]. Github's code search can find code matching a user's query across many public repositories. Filters can be added for specific file types and results can be sorted by recently indexed, best match, etc. Searchcode is a simple open-source code search tool which integrates with multiple code hosting services, including Github, GitLab and others. Aside from the search functionality itself, it allows the user to filter by code hosting service or by programming language.

[15] https://github.com/
[16] https://searchcode.com/
[17] https://git-scm.com/

3 Method

The main goal of this thesis is to extract representations of user flows in the form of activity diagrams from existing open-source Android applications, with the goal to develop a project template featuring user flows such as onboarding, user creation and signing in. The thesis will focus on user flows that are commonly found at the start of an application, i.e. onboarding, signing in, and creating a user. Onboarding will be chosen as the first user flow to be analysed and it will be the first flow to be developed in the template project.

This section focuses on describing the methodology used to achieve the goals of the thesis and how code search tools will be used to find classes related to specific user flows in open-source Android applications. In addition, this section will detail how activity diagrams will be extracted and combined, how the requirements will be specified for the project, how they will be prioritised, what specific technologies will be used to develop the template project based on the chosen user flows and how the results will be evaluated. The structure of the method can be seen in Figure 3.

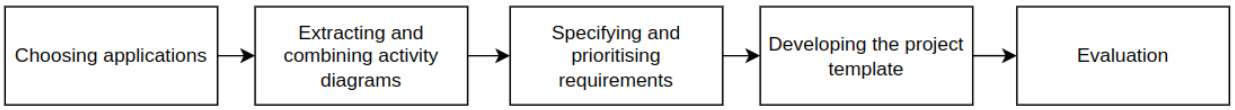


Figure 3. Illustration of the structure of the method

3.1 Code search

GitHub code search will be used to find applications containing onboarding, user creation, and sign-in user flows. The Searchcode website, which can search across multiple code hosting platforms, was also considered to obtain a greater variety of results across multiple source code hosting platforms. However, the results were limited compared to what was found on GitHub due to Searchcode being a private project with network and query limitations. To limit the search to applications that have been developed following some sort of architectural pattern, the queries will follow the package naming structure of Android applications [18]. The query will consist of an 'ui' package and the name of the user flow, such as 'ui.user_flow_name'. The search will be further limited to Kotlin files, as this is the main programming language used in native Android application development. As Kotlin has been the recommended programming language for Android applications since 2019, Java will

be filtered out as the focus is on more recently developed applications. If the objective is to find a user flow where the user logs in to their account, the search query would be ‘ui.login’. The queries and the limitations of the searches are detailed in Table 1.

Table 1. Queries to be made using code search tools

User flow	Query	Limitations or filters
Onboarding carousel	‘ui.onboarding’	only .kt files
Login	‘ui.login’ and ‘ui.signin’	only .kt files
Sign up	‘ui.signup’	only .kt files

Up to ten applications of each user flow will be compared following an algorithm to see if their implementation of the user flow is different and if these user flows can be combined. Since the code search tool used for this thesis does not offer many ways of filtering repositories, the search will be sorted by ‘Recently indexed’ on Github to find repositories that have been actively worked on within the last few days, weeks or months of writing. The applications will be chosen following the repository requirements specified in Table 2 by going through the search results manually.

Table 2. Requirements for repositories

Requirement ID	Description
RR1	Must have been updated within the last year
RR2	Has at least one tagged release
RR3	Has more than one contributor
RR4	Has some sort of licence (MIT, Apache)
RR5	Has a rating of more than 10 stars (Github) or a similar equivalent
RR6	Is available on the Google Play Store
RR7	Supports latest Android version

3.2 Activity Diagrams

Applications will be launched and the actions will be described with activity diagrams as pictured in Figure 3. Activity Diagrams were chosen because they are suitable for modelling how multiple events in a single use case relate to each other to represent business workflows, or in the case of this thesis, user flows [19]. In the context of this thesis, a node in an activity diagram represents an action or a decision by the user. A new node will be created in a given activity diagram when the user does an action, like clicking a button, or if the application prompts the user to perform an action, as shown in Figure 4. Due to the relatively low number of activity diagrams that will be created, the algorithm which they will be combined with will be used manually, instead of implementing an automated approach.

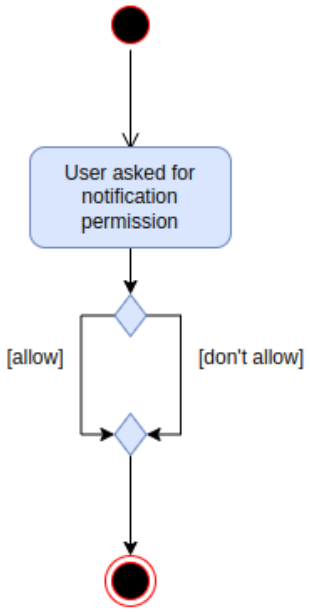


Figure 4. Example of an activity diagram

Activity diagrams of different applications will be compared manually following an algorithm to see how many nodes are identical or overlapping, with the purpose of combining them into one activity diagram that represents an average of a certain user flow. For example, if all chosen applications for a sign-up user flow implement each step of signing in in exactly the same way, then there would be a complete overlap of nodes in their activity diagrams and a template based on that user flow would be able to be a base for all of those applications. In the case that the implementation of the user flow differs heavily in each application, there would be no need to create a generic reusable tool for that user flow, as each application

would seem to need a unique approach. To account for this, a node will be classified as common if it is found in at least a third of all activity diagrams for a specific type of user flow. An algorithm for combining these activity diagrams can be found in Figure 5.

```
node = defines and describes a certain part of a user flow in an activity
diagram = sequential flow of nodes (tree)
chart = array of diagrams
chart.max_depth = maximum depth of chart (from the longest diagram)
chart.depth(i) = gets a list of nodes at chart depth of i (integer)
chart.diagrams_that_contain_node(node) = list of diagrams that contain the node given as an argument
diagram.depth(i) = node at diagram depth i (integer)
diagram.get_matching_node(node) = gets matching node given as an argument from diagram if it exists
connection = transition to next node
node.diagram.next_node = Next node in the diagram for the current node
node.type = Activity, Decision, Merge
diagram.flow_type = Onboarding, Login, etc
node.flow_type = Onboarding, Login, registration, terms & conditions, etc
```

```
fun main(chart):
    common_nodes = []
    combined_diagram = []
    determine_starter_diagram(chart)
    add_missing_common_nodes()
```

```
fun determine_starter_diagram(chart):
      FOR i in range(0, chart.max_depth)
          FOR node IN chart.depth(i):
               IF node NOT IN common_nodes
                   parse_node(node, i)
      combined_diagram = get_diagram_with_max_common_nodes(common_nodes)
```

```
fun parse_node(node, depth):
      FOR diagram in chart:
           IF diagram.depth(i) == node AND diagram.flow_type == node.flow_type:
                 count++
```

```
      IF count >= len(chart)/3:
           common_nodes.add(node)
```

```
fun get_diagram_with_max_common_nodes(array_of_nodes):
    max_count = 0
    most_matching_diagram = null

    FOR diagram IN chart:
       count = 0
       FOR node IN common_nodes
            IF diagram.contains(node)
            count++
       IF count > max_count:
            max_count = count
            most_matching_diagram = diagram
    return most_matching_diagram
```

```
fun add_missing_common_nodes():
    FOR node IN common_nodes:
        IF NOT combined_diagram.contains(node):
             position = get_best_pos(node)
             combined_diagram.insert(connection, position - 1)
             combined_diagram.insert(node, position)
             combined_diagram.insert(connection, position + 1)
```

```
fun get_best_pos(node):
    set_nodes_before = {} // no duplicate values
    set_nodes_after = {} // no duplicate values

    FOR diagram IN chart.diagrams_that_contain_node(node):
        node_pos = diagram.get_matching_node(node).position
        set_nodes_before.add(diagram.nodes[:node_pos])
        set_nodes_after.add(diagram.nodes[node_pos:])
    FOR index, node IN combined_diagram:
        IF node IN set_nodes_before:
            continue
```

```
        IF node IN set_nodes_after:
            return index
    return combined_diagram.length // add at the end
```

Figure 5. Pseudocode for combining Activity Diagrams

The algorithm goes through every node in the diagrams in the chart, parsing the whole chart in a row-wise traversal, while counting how often the current node being parsed occurs in other diagrams. If the node occurs in more than one-third of all diagrams, it will be put into a list of common nodes.

Next, the diagram with the most commonly occurring nodes will be chosen as a starting point for the combined diagram. This is done by iterating through each diagram and counting how many commonly occurring nodes are in each diagram and choosing the one with the maximum amount of common nodes as the starting point. Finally, the other common nodes which are not present in the starting diagram, will be added by determining their best possible position in the combined diagram, by iterating through other diagrams containing the current node being evaluated. All of the nodes that are found before and after the node under evaluation, in other diagrams containing said node, will be added to corresponding sets. Then, the node under evaluation will be added in a position in the combined diagram where the combined diagram starts having nodes that commonly follow the node under evaluation in other diagrams. If no such position is found, the node is added to the end of the combined diagram.

After following this algorithm, the person combining the diagrams should manually check if the flow makes logical sense. In the case that a node is placed in a location that does not make logical sense, it should be moved so that the user flow is usable. This is due to the algorithm not being able to differentiate between specific qualities in the actual user flow, as a node only contains a general description of what happens in it.

3.3 Requirements

After a specific user flow has been chosen and a new activity diagram has been created based on the overlapping features between different applications containing the chosen flow, the functional requirements that will be imposed on the first iteration of the template project will be specified from that activity diagram. Other requirements will be specified after the main functional requirements have been decided upon, based on qualities that would be useful for other developers using the template created in this thesis project. In the case of comparing activity diagrams for a user sign-in flow, if nine applications out of ten require a field for the user's email, then the email field will be included. Likewise, if only one application requires the user to input their date of birth, then that feature will not be included.

Requirements for the template will be prioritised using the MoSCoW method, where each requirement is marked with one of the following letters: M - must have, S - should have, C - could have, W - will not have [20]. While there are more detailed ways to prioritise requirements, the MoSCoW method will be used due to its simplicity. All of the functional requirements specified from the newly created activity diagram will be marked as must have. An example of a requirement in the 'should have' category would be the following - to simplify the process of launching the project for developers, it should be published on the Google Play store. A 'could have' requirement might be developing a second version of the project using Jetpack Compose, to provide support for developers using that UI toolkit.

3.4 Development

This section describes what technology and application architecture will be used to develop the template. The template will be developed following best practices such as SOLID principles and Clean Architecture to simplify writing tests and make the template scalable for developers. As the focus of this thesis is developing a template based on a specific user flow, there will be less focus on parts of the template concerning business logic, thus no unit tests will be written either.

3.4.1 Choice of technology

Android UI components have mostly been developed by describing the views in Android's XML vocabulary[18], and then writing the logic for the view in a Kotlin or Java file. From 2021, Jetpack Compose has been promoted by Google as a means to write the entire view in

[18] https://developer.android.com/develop/ui/views/layout/declaring-layout

Kotlin, with a focus on simplifying and accelerating UI development [21]. Due to the widespread usage of the standard development style using both XML and Java or Kotlin files, the project will initially not be developed using Compose. In the case that both code search and development will be finished before expected, a Compose variant of the project will also be created, to provide users of the template with greater flexibility in development.

Due to the popularity of Kotlin, its endorsement by Google, and its numerous advantages over Java, it was chosen as the development language for this thesis project.

3.4.2 Application architecture

As the result of this thesis project will be a template for other developers to use, it should also be a working application so that developers can interact with it. Developing an Android library, which would be a dependency that could be added to any project, instead of an application, would allow developers to use the project more easily, but doing so would detract from customizability, as users would not be able to change classes and layout files freely.

Android applications are usually developed following specific architectural patterns such as Model-View-Controller (MVC), or Model-View-Viewmodel (MVVM), which help separate business logic and user interface components [22]. This is also important in this thesis because the choice of architecture will change how the UI layer, which is the focus of this thesis project, will be developed. Since Google's guidelines suggest using ViewModels in Android applications[19], the template project will be developed using the MVVM architecture.

3.4.3 Version control

The template created in the course of this thesis project will be hosted on a public GitHub repository. During the initial development of the template, code will be committed and pushed to the repository in an unstructured format. In the case that additional features are added after the initial development of the project, they will be created in separate feature branches and merged into the main branch after development is complete.

When the project has been deemed complete, it will be given a release tag and a README file will be written to provide a tutorial for any developer wanting to download and run the project. The README file will contain a general description of the finished project, an overview of the architecture of the application, the dependencies used in the project, and a licence.

[19] https://developer.android.com/topic/libraries/architecture/viewmodel

3.5 Evaluation

Evaluation will be done in three iterations by recreating the UI of some applications found in the code search, using the template project, and seeing how much code needs to be changed to end up with an application with a similar-looking UI. Functionality and business logic will not be recreated as the focus of this thesis project is on user flows and providing a UI template for developers. In the first iteration, three applications with an onboarding user flow will be recreated, after which the template project will be improved by marking down any problems and missing features that came up during the initial evaluation. The second iteration will repeat the process using onboarding flows and see whether the improvements to the template project returned better results. The third iteration will feature recreating the sign-in and sign-up flows from two applications to see if the approach taken for onboarding is applicable for other flows as well.

4 Results

This section covers the results of the code search, choice of the user flow, code analysis, the development of the chosen user flow template and multiple iterations of evaluation of the created template.

4.1 Code search

Due to the presumed high occurrence of onboarding user flows in mobile applications, the onboarding user flow was chosen as the first topic of interest.

With the ‘ui.onboarding’ query run in Github, with limitations to only Kotlin files, there were 3500 results. Due to the lack of search options for code in Github, the results were sorted by recently indexed files. The purpose of this was to find code that had been either recently created or updated, assuming that finding high-quality open-source projects was easier with this sorting filter. It could also be true that user flows such as onboarding aren’t updated often, so sorting by recently indexed files might not have had a positive effect on the search. After that, the results were parsed manually to find repositories that matched the requirements detailed in Table 2. Ten matching repositories were found and the results of this search can be found in Appendix 1.

Repositories containing code for log-in and sign-up flows were searched for in a similar manner to onboarding flows. The first search with the queries ‘ui.login’ and ‘ui.signin’ combined returned 9300 results, which were also sorted by recently indexed. The search was combined with a query for ‘ui.signup’, under the assumption that an application that has login functionality also has the option to create an account for the user. The results of the query were gone through manually, trying to match found repositories with the requirements detailed in Table 2. Eight matching repositories were found and the results of this search can be found in Appendix 2.

4.2 Activity diagrams

This subsection describes the creation of activity diagrams based on the applications found after the search described in the previous section. The applications were installed through the Google Play store and activity diagrams were created after manually going through every screen of the specific user flow that was under investigation. If the user flow contained a path that was inaccessible, it was left out of the activity diagram for that user flow. An example of this would be finishing account creation in the Entourage Reseau Solidaire application, where a French or Belgian phone number was needed to finish user registration. Similarly, finishing account creation in WooCommerce needed the user to pay a fee, so that part of the user flow was also left out of the activity diagram. After all of the activity diagrams for each specific user flow were created, they were combined into one activity diagram that contained all of the common features between these user flows.

4.2.1 Onboarding

The onboarding flows of ten applications were translated into activity diagrams. While there were some applications which had a completely unique onboarding flow, most of them followed a format where users were given between three and five screens which introduced the application to the user. These screens often featured being able to either swipe between screens or press a button to continue to the next one. The Tangem application also featured a timer that switches between screens automatically after around five seconds. Another common quality between these onboarding flows was asking the user for either notification or location permission. This was most likely done since the application needed to use these at some point, as they were unrelated to the onboarding flow itself. While this isn't the recommended approach by Google, these were also added to the combined activity diagram, found in Figure 6. All of the activity diagrams for onboarding can be seen in Appendix 3.

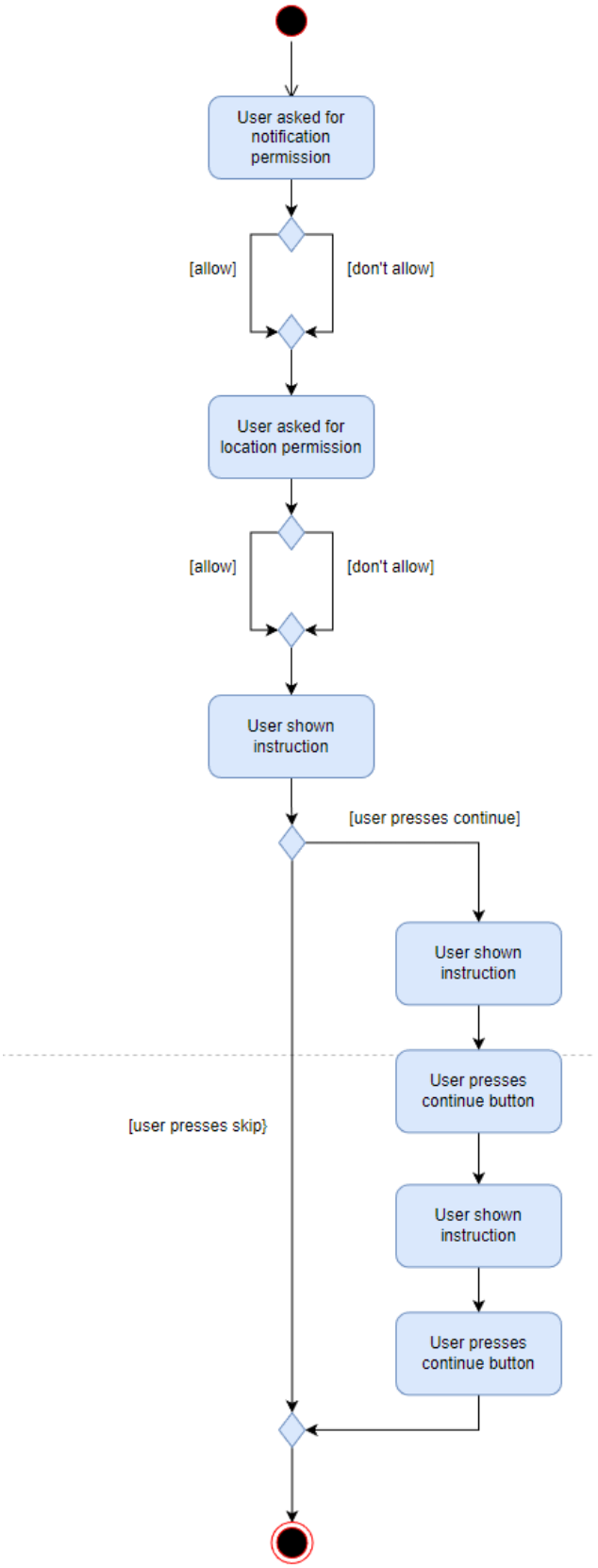


Figure 6. Combined onboarding activity diagram for onboarding

4.2.2 Sign-in & Sign-up

Due to applications which contained a sign-in user flow also often featuring a sign-up flow, these two user flows were combined into one search and two activity diagrams were created for each application when possible. One for logging in and one for creating a user. Six applications were chosen for their sign-in and sign-up user flows. The complete log-in user flow of the WooCommerce application was not able to be translated into an activity diagram, since the user was required to pay at one point to set up their commerce account. A common feature between some of these applications was the ability to use a Google account to log in without having to create a custom account for a specific application. For signing up, most applications require just a username or an email with a password which would have to be entered twice for confirmation. There was slightly more variety in the case of sign-in and sign-up user flows compared to onboarding user flows. These diagrams can be seen in Appendix 4 and the combined diagram can be seen in Figure 7.

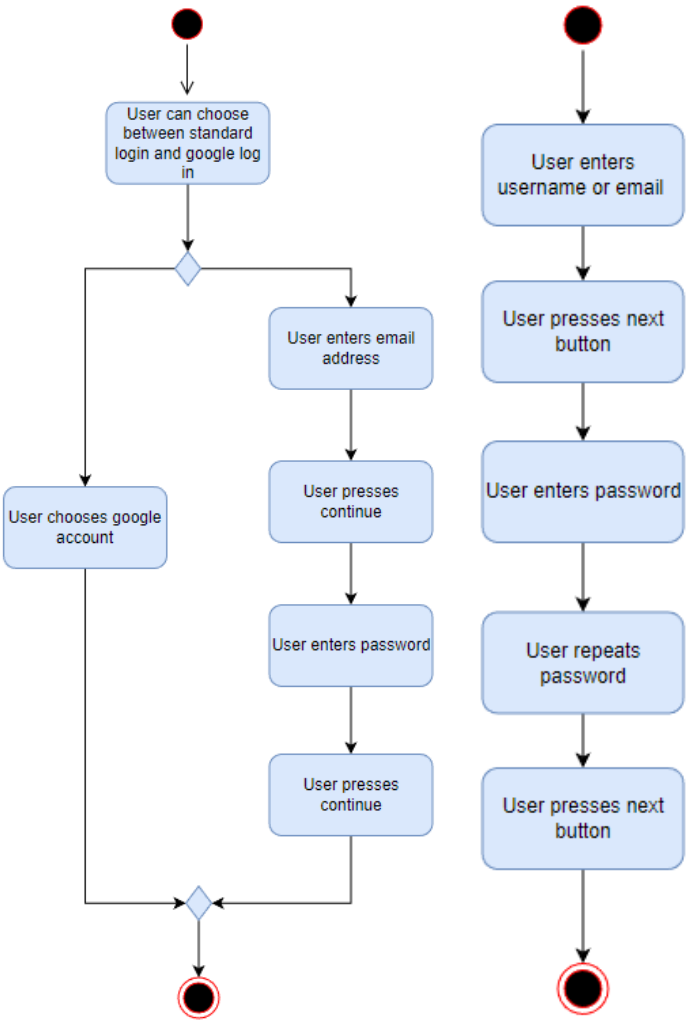


Figure 7. Combined onboarding activity diagram for sign in & sign up user flows

4.3 Requirements

This subsection details how requirements were specified and prioritised for the development of the template project.

4.3.1 Specification

In order to start with the development process, functional requirements were determined from the combined activity diagram. Some additional requirements were taken from analysing the onboarding screens, for example, the number of text views shown to the user. All of the functional requirements can be seen in Table 4.

Table 4. Functional requirements

Requirement name	**Description**
FR1	The onboarding user flow should contain a variable number of screens
FR2	The user should be able to swipe between screens
FR3	The user should be able to skip the onboarding flow entirely
FR4	The user should be able to press a continue button to go to the next screen
FR5	A screen of the onboarding user flow should contain a title
FR6	A screen of the onboarding user flow should contain a description
FR7	A screen of the onboarding user flow should contain an image
FR8	The application should be able to specify if the user is asked for notification permission
FR9	The application should be able to specify if the user is asked for location permission
FR10	The template will be developed using Android's XML notation
FR11	The template will be developed using Jetpack Compose

Nonfunctional requirements for the user interface were derived from Material design guidelines[20] while requirements for the code itself were defined from Clean Architecture guidelines, with the focus being on scalability and modifiability.

Table 5. Nonfunctional requirements

Requirement name	**Description**
NR1	The application should support dark and light mode
NR2	The application should support both portrait and landscape orientations
NR3	The application should follow Material Design guidelines
NR4	Swiping between screens should be smooth
NR5	A developer should be able to easily change the content of each screen
NR6	A developer should be able to easily able to change the number of screens

4.3.2 Prioritisation

Priorities for the previously described requirements are shown in Table 6. Requirements were assigned one of four priorities. Must have - M, Should have - S, Could have - C, Will not have - W. Requirements marked as Could Have will not initially be added to the template project in order to comply with time constraints, however, if there is time left over after initial development and writing, they will be added as well. Requirements marked as Will Not Have will not be added to the project. The prioritised requirements can be seen in Table 6.

Table 6. Requirements prioritisation

Requirement name	**Priority**
FR1	M
FR2	M
FR3	M

[20] https://m2.material.io/design/communication/onboarding.html#specs

FR4	M
FR5	M
FR6	M
FR7	M
FR8	C
FR9	C
FR10	M
FR11	W
NR1	S
NR2	S
NR3	S
NR4	M
NR5	M
NR6	M

4.4 Development - Phase 1

The template project was developed in Kotlin between February and March of 2023 using Android Studio.

4.4.1 UI Design

The user interface was designed following the guidelines from the best practices section of the previously mentioned Material Design system. Since the focus of this project was to give other developers a baseline project on which to develop their own applications upon, it was important to focus more on customizability. Due to this, the focus was mostly on the quality of code as opposed to a visually pleasing user interface. Additionally, some creative liberty was taken by the author due to the expectation that the UI of the project would be reconfigured by developers using it anyway. The onboarding screens consist of an ImageView and two TextViews, where one TextView is used for a title and the other for a more detailed description. The Material View Pager Dots Indicator project[21] by Tommy Buonomo was used to show which page the user is currently on. Additionally, the screens feature a continue button in case a user is not able to or does not want to swipe between screens. A skip button is also featured at the top right of every page, except the last one as the functionality is then the same as the continue button. The screens can be seen in Figure 8.

[21] https://github.com/tommybuonomo/dotsindicator

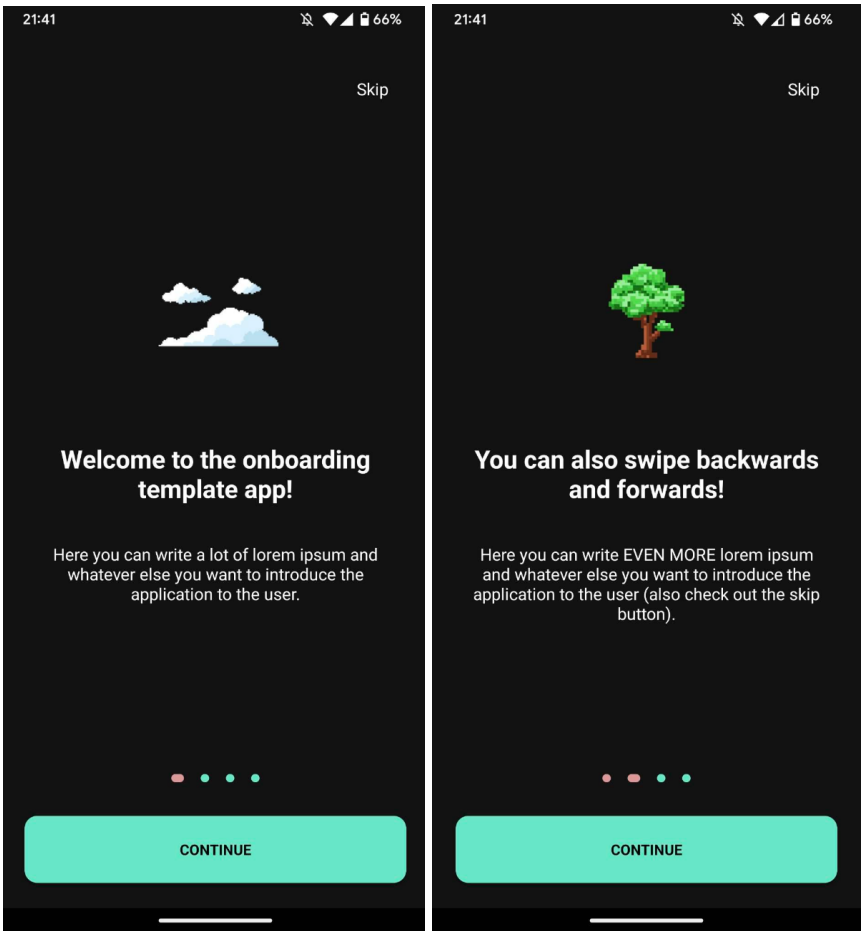


Figure 8. Two screens of the onboarding flow in the template application

4.4.2 Application architecture

The template project was developed following the Model-View-ViewModel architecture, in which business logic is separated from the View and held in a class called a ViewModel, which bridges the View and Model layers. While the onboarding flow itself does not have any business logic, button presses are still triggered through the ViewModel to give developers an example of correct the implementation of this architecture.

The Hilt Dependency Injection library[22] was used to reduce boilerplate code for future developments and to make the code more reusable. This library allows classes to be injected into other classes instead of having to instantiate them each time they're used, as shown in Figure 9.

[22] https://developer.android.com/training/dependency-injection/hilt-android

```kotlin
package ee.ksr.template.ui.onboarding

import androidx.lifecycle.ViewModel
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject

 Kristofer K
@HiltViewModel
class OnboardingViewModel @Inject constructor() : ViewModel() {
}
```

Figure 9. Dependency injection using Hilt in the Onboarding Fragment

4.4.3 Configurability

In the case that a developer chooses to simply use the onboarding user flow as is, instead of modifying any of the source code, the content of the screens can be changed in the arrays.xml file, as seen in Figure 10. The amount of screens however must be changed in the OnboardingContainerFragment.kt file, using the variable NUM_SCREENS, which is set to 4 by default. The number held in that variable must match the amount of items in the arrays holding the content of the screens.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="onboarding_title_array">
        <item>@string/onboarding_title_1</item>
        <item>@string/onboarding_title_2</item>
        <item>@string/onboarding_title_3</item>
        <item>@string/onboarding_title_4</item>
    </string-array>

    <string-array name="onboarding_body_array">
        <item>@string/onboarding_body_1</item>
        <item>@string/onboarding_body_2</item>
        <item>@string/onboarding_body_3</item>
        <item>@string/onboarding_body_4</item>
    </string-array>

    <integer-array name="onboarding_image_array">
        <item>@drawable/clouds</item>
        <item>@drawable/tree</item>
        <item>@drawable/cactus</item>
        <item>@drawable/cow</item>
    </integer-array>
</resources>
```

Figure 10. Arrays containing the content for the onboarding screens

4.4.4 Version Control

The source code of the template project is held in a public GitHub repository. The repository features a simple README file that describes which dependencies were used in the development of the project and a guide to configuring the content of the included onboarding screens. The link to the repository can be found below.

https://github.com/ksr-dev/onboarding-mvvm-template

4.5 Evaluation - Phase 1

This section describes evaluation of the template project by using it to recreate onboarding flows of three applications - WooCommerce and Entourage Réseau Solidaire, which were chosen due to their similarity to the onboarding user flow recreated in the template project, and the Tangem Crypto Wallet application, which had significant differences to the other applications and thus the template project as well.

4.5.1 WooCommerce

A rough draft of the WooCommerce application's onboarding user flow was recreated using the template project. An example comparing the first pages of both the original WooCommerce application's onboarding user flow and the recreated version in the template project can be seen in Figure 11. Certain design elements like the stylized title and images were not brought over from the WooCommerce application, as the purpose of this recreation was to show the customizability of the template project and not to create a perfect copy of the original.

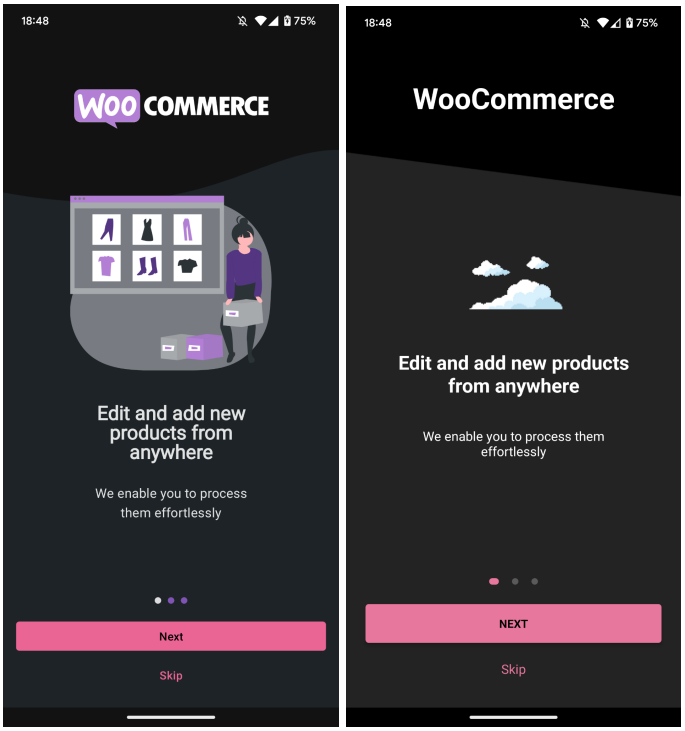


Figure 11. The first screen of the WooCommerce application's onboarding user flow (left) recreated in the template project (right)

4.5.1.1 Codebase

In order to recreate the application in the template project, 12 files were changed, of which 11 were .xml files. One Kotlin file, OnboardingContainerFragment.kt was changed, in which two lines were removed and one variable was changed. The purpose of the removal of two lines of code was to not hide the skip button in the last page of the onboarding screen. This was the default behaviour of the skip button in the template project, but not in the WooCommerce application. The variable which was changed was NUM_PAGES, since the WooCommerce application's onboarding flow only had 3 pages, instead of 4, which was the default for the template project. The changes in the .xml files were to recreate the appearance of the WooCommerce application's onboarding user flow, change the content of the text, and add new constants for dimensions, colours, etc.

The general architecture of the onboarding flow in the WooCommerce codebase is very similar to the template project. While the arrays for the contents of the screens are held in a Kotlin class, instead of a .xml file, the way the content of the screens is shown is the same. This might be a slightly more convenient approach for developers, as there is no separate NUM_PAGES variable which has to be changed when the length of the arrays is changed, rather the amount of screens is dictated by the length of the array. A similar dot indicator library was used for showing pages, but one seemingly created by the WooCommerce team itself, instead of something external. Additionally, there is handling for portrait and landscape views, which is not present in the template project.

4.5.2 Entourage Réseau Solidaire

As the next step, a rough draft of the Entourage Réseau Solidaire application was recreated using the template project. The implementation of the sliding functionality of the screen seemed to differ in the Entourage Réseau Solidaire application, only the images moved with a sliding motion and the text simply changed after the button was pressed. This was not changed in the template project, as the objective of recreating the application was to see if the template project was compatible to recreate the application under analysis as simply as possible. The original and recreated third screen of the application can be seen in Figure 12.

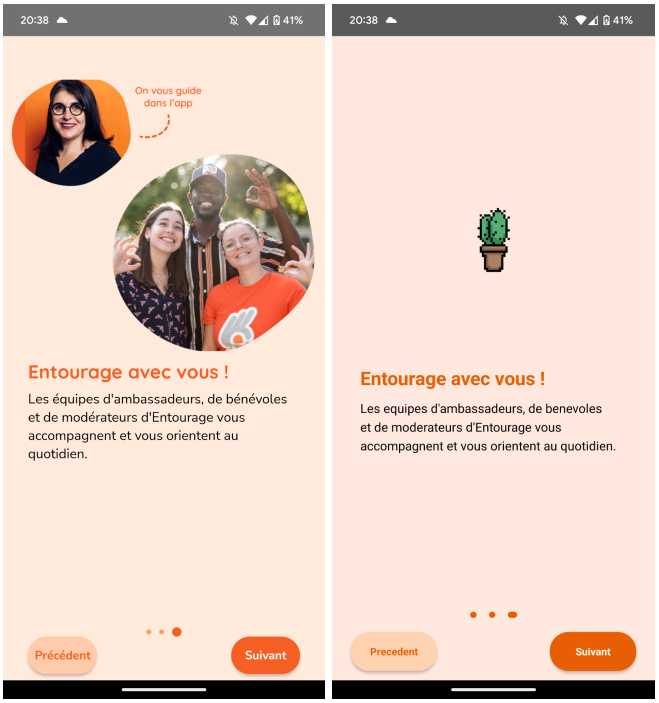


Figure 12. Last screen of the Entourage Réseau Solidaire application's onboarding user flow (left) recreated in the template project (right).

4.5.2.1 Codebase

In order to recreate the application in the template project, 13 files were changed, 12 of which were .xml files. In the case of this application, a back button had to be added to the template project, which required 23 new lines of code in OnboardingContainerFragment.kt, as a new method had to be added to handle the functionality to go back a page. Additionally, swiping was removed by disabling user input on the ViewPager2 component.

Unlike the template project and the WooCommerce application, no ViewPager was used to show separate fragments to the user in the codebase of Entourage Réseau Solidaire. Instead, a RecyclerView, in which items took up the entire width of the screen, was used to show the images to the user. Upon clicking the "next" button, the smoothScrollToPosition method was called on the RecyclerView with the next position given as an argument. The text shown to the user was changed manually upon the RecyclerView position changing and was not swipeable. The dot indicator was manually created and shown by using an ArrayList of ImageViews and iterating through them every time the position of the RecyclerView changed. While no external libraries were used for this, which reduces application size and in some cases improves maintainability, some of the approaches used in this codebase seemed needlessly complicated and not efficient performance-wise.

An argument could be made that since the entire onboarding flow was written in one Activity with less than 150 lines of code, it was relatively concise and easy to follow. However, using Fragments instead of Activities is considered a better practice by Google and can be more performance efficient in certain cases. Additionally, separating the page and container fragments of the onboarding flow might be considered good separation of concerns, as one could be modified without affecting the other directly.

4.5.3 Tangem - Crypto Wallet

Finally, the Tangem Crypto Wallet application was attempted to be recreated in the template project. This proved to be more difficult than the previous two applications and some features were left out, as they required heavy modification of the template project. The main goal of the template project was to allow the user to create an onboarding flow with minor modifications, which may include changing or creating a few methods. The Tangem Crypto Wallet application featured multiple qualities which would not be as simple to recreate as just a back button, which was added for the previous application used for evaluation, including videos playing on every page instead of just plain text, complex animations and also a system which resembled Instagram Reels, in which the current page would change automatically after a set time which would be shown on a progress bar at the top of the page. A basic version of the application with the complex features redacted was still recreated for comparison purposes and can be seen in Figure 13.

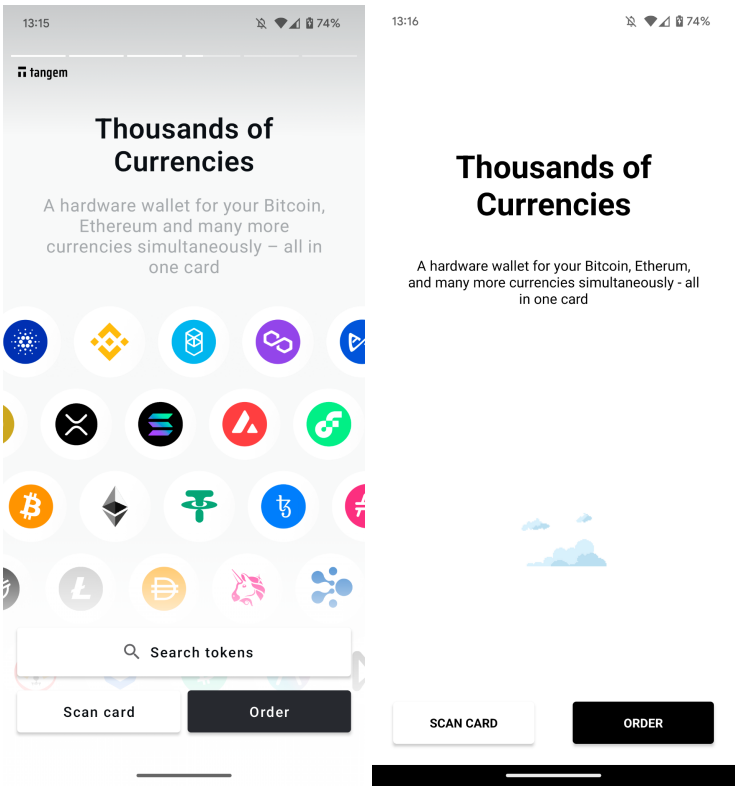


Figure 13. The third screen of the Tangem Crypto Wallet application's onboarding user flow (left) recreated in the template project (right).

4.5.3.1 Codebase

The Tangem Crypto Wallet's user interface, unlike the other applications analysed in this section, was written in Jetpack Compose. The content of the screens, previously thought to be videos, were in fact custom-made animations created using Jetpack Compose and different types of animation functions. Due to the author's unfamiliarity with Jetpack Compose, the quality of the code could not be evaluated to the degree that the previous applications were, although even though it did differ heavily from the previous two applications, it seemed to also follow good coding practices from the SOLID principles. The use of Jetpack Compose seemed to offer more configurability to view components compared to standard .xml layouts.

4.5.4 Improvements on the template project

From the three applications that were recreated, it was clear that the option to have a back button should be added to the template project, along with an option to hide it and other buttons as well.

To make the template project easier to customise without writing extra code, it should be possible to enable or disable this button and others by modifying some base parameters. For example, in the case of the Entourage Réseau Solidaire application, swiping between screens was also not possible and navigation only happened by pressing the forward or back buttons. This was something that could also be easily configured using a boolean, so it was also added to the improvements of the template project. Enabling or disabling these booleans was done by changing parameters which were added to OnboardingContainerFragment.kt in the application's navigation graph, as seen in Figure 14.

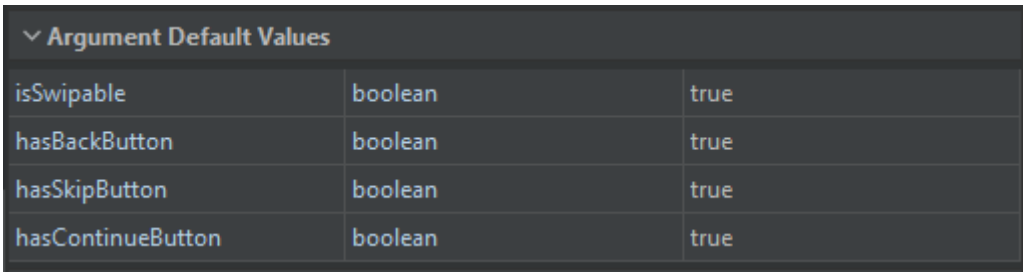


Figure 14. OnboardingContainerFragment.kt arguments

4.5.5 Evaluation after improvements - Repeatcard

Repeatcard's onboarding flow was recreated after improvements to the template project had been made after the previous round of evaluations. Improvements to the template project proved to be useful as both a back button and the ability to hide the skip button were needed. The recreated UI can be seen in Figure 15.

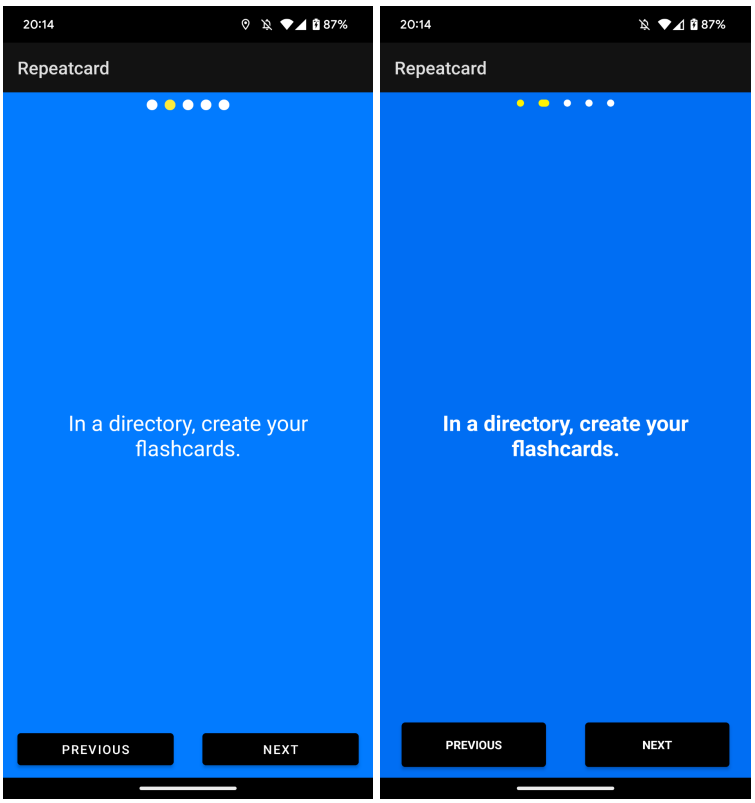


Figure 15. The second screen of the Repeatcard application's onboarding user flow (left) recreated in the template project (right).

4.5.5.1 Codebase

In order to recreate the UI of the Repeatcard application's onboarding user flow, 11 files were changed, of which 2 were Kotlin files and the rest were .xml files. Repeatcard featured no images but the background of each screen was of a different colour, so some adjustments needed to be made to the creation of the onboarding page fragment. No new methods were added, as existing methods could be modified to provide this functionality. Due to the lack of a second text view and no skip button, more code was removed than added. The original codebase for Repeatcard resembled the template project as there was a similar amount of configurability. The string array for the content of the pages also came from a .xml array and a ViewPager was used to show the pages. Additionally, the naming conventions used for methods and separation of concerns were very similar in both projects.

4.6 Development - Phase 2

Following the evaluation of the onboarding user flow created in the template project based on the combined activity diagram of other applications flows, it was clear that it could be used to recreate the onboarding flows of other applications with minimal Kotlin code needing to be written. In order to further prove that a combined activity diagram based on existing applications could be used to create a template, sign-in and sign-up flows were also added to the template project's codebase. Functional requirements were specified from the combined activity diagram and configurability lessons learned from the evaluation of the onboarding user flow. All of the requirements will be marked as Must Have and will be added to the project template. The requirements can be seen in Table 7.

Table 7. Functional requirements

Requirement name	**Description**
FR12	The user should be able to sign in with a Google account
FR13	The user should be able to sign in with a user identifier and password
FR14	The user should be able to create an account
FR15	The user should be able to access account creation, sign in regularly, and sign in with Google from the first screen.
FR16	The user needs to confirm their password in another text field when signing up.
FR17	A developer should be able to configure whether the user is asked for an email or a username
FR18	A developer should be able to configure which of the buttons on the first screen are visible.

Due to more business-related functionality, such as handling user input, being needed in the sign-up and sign-in processes, there was an opportunity to add some placeholder business logic between the ViewModel and the fragments. Similarly, UseCases classes were used to bridge the gap between the data layer and the ViewModel and ensure that a developer using

this project as a template could easily switch out the functionality in one layer without interfering with the other.

When creating a user or signing in, the repository layer returns dummy data to the ViewModel layer after a small delay, which then sends an event to the view to navigate the user to the home screen to indicate a successful user creation or signing in. The Sign In With Google button, seen in Figure 16, is fully functional, although a developer using this project as a template needs to configure a Google API Console project to use this in their application.

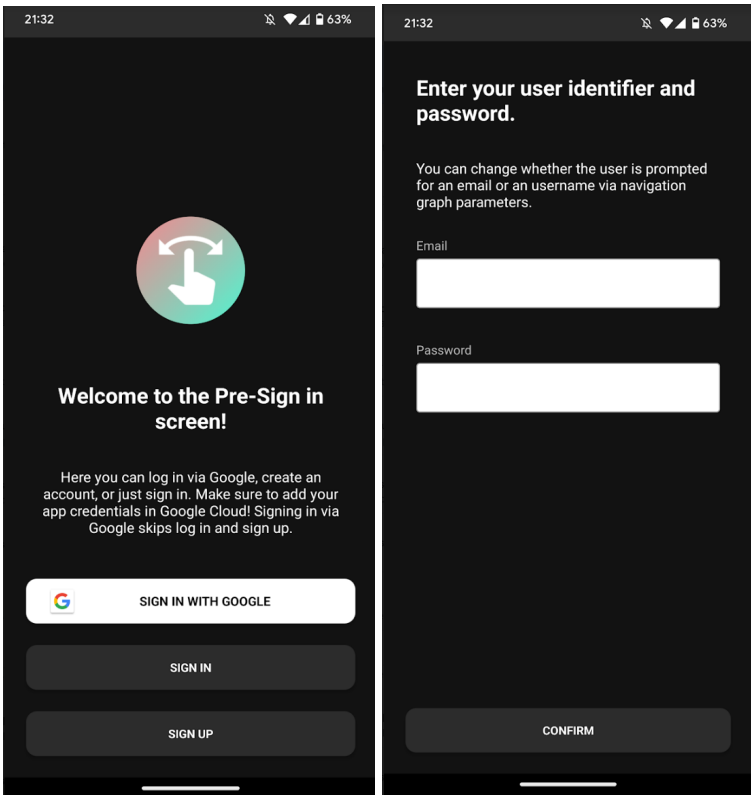


Figure 16. The Pre-Sign in screen and Sign in screen in the template project

As seen in Figure 17, the sign-up user flow consists of two screens. The first one asks the user for their identifier, which can be an email or a username, and the second screen asks them to insert their password twice. Having the user confirm their password was a common trend in applications that were found and also was featured in the combined activity diagram for sign-up. The screens also include a title and description which can be easily changed by editing the strings.xml file in the template project.

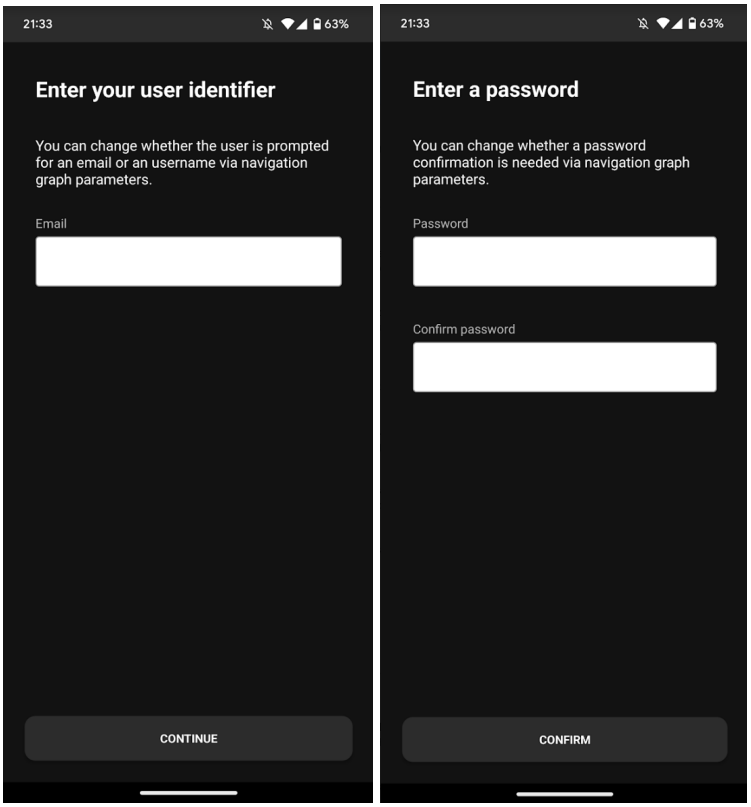


Figure 17. The sign-up screens in the template project

For these user flows, the configurability is mostly present in the Pre-Sign in screen, where any of the buttons can be disabled and hidden by changing the default parameters in the application's navigation graph. Additionally, in the user identifier screen seen in Figure 15, a parameter can be changed which determines whether the user is asked for an email or a username. Changing this parameter only changes the title shown above the text field, but a developer using the template project could further modify this logic.

4.7 Evaluation - Phase 2

This section describes the evaluation of the template project by using it to recreate the sign-in and sign-up flows of two applications - Posture Reminder and ProtonMail. Posture Reminder was chosen since it contained a simpler user flow with fewer screens. ProtonMail was chosen due to its complexity compared to other applications.

4.7.1 Posture Reminder

Posture Reminder was chosen as the first application with a sign-up or sign-in flow to recreate. The application had a simple sign-in screen, as having a user was not a mandatory part of using the application, however as it featured a button to log in using Google authentication, it was still usable for evaluation. Recreating the UI of this application proved to be fairly simple as the user creation, password and sign-in screens did not need to be modified. The original and recreated screens can be seen in Figure 18.

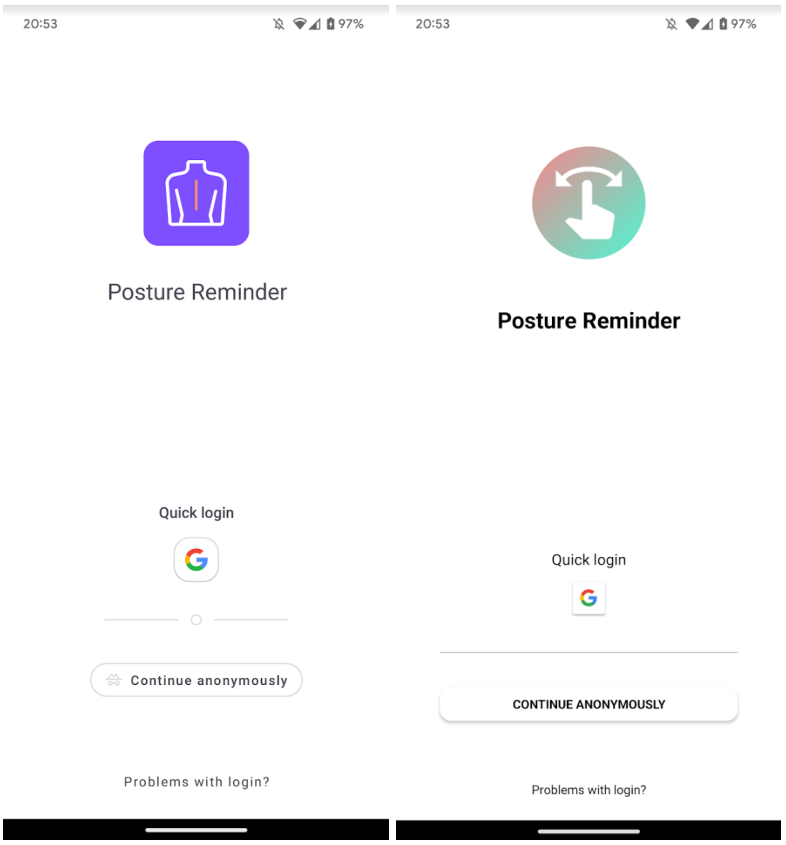


Figure 18. The Posture Reminder pre-sign in screen (left) recreated in the Template project (right)

4.7.1.1 Codebase

Recreating the sign-in screen of Posture Reminder required changing 4 .xml files. The small number of changed files can be explained by the lessons learned by improving the onboarding user flow part of the template project and the simple structure of the application's sign-in screen. Comparing the codebases of the template project and the Posture Reminder application, it is clear that the approaches taken for development were quite similar, although there is better separation of concerns in the codebase of Posture Reminder, as code related to the Google sign-in functionality is held in a separate class. Overall, the implementation of this screen in the Posture Reminder application is very concise, as the screen is also quite simple. The rest of the application is built in a similar manner to the template project, utilising Hilt for dependency injection and an MVVM architecture using Fragments for displaying views.

4.7.2 ProtonMail

The drop-down menu for choosing an email domain in the ProtonMail sign-up flow was left out of the recreated UI in the template project, as this would be considered a larger functionality in the scope of recreating these applications. Adding this would have required adding a few Kotlin methods and possibly an extra utility class for a better separation of concerns. The original and recreated screens can be seen in Figure 19 and Figure 20.

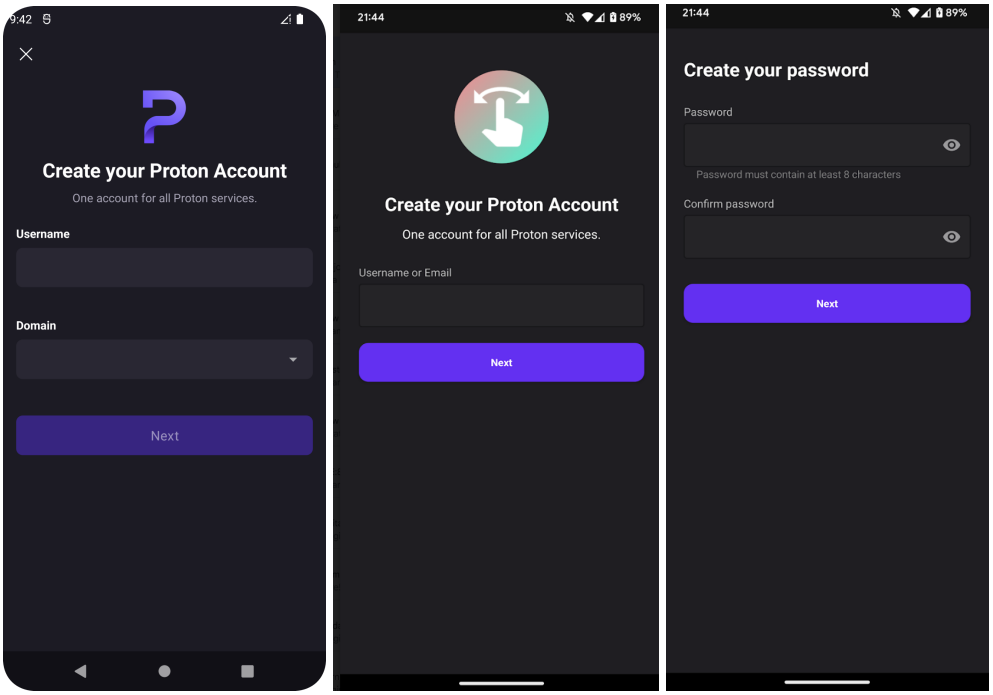


Figure 19. The ProtonMail sign-up screen (left recreated in the Template project (right)

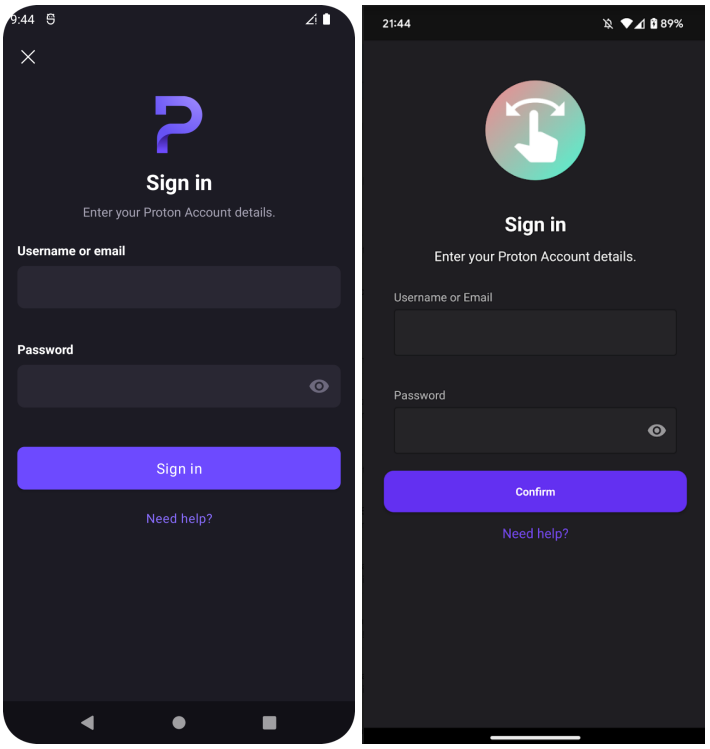


Figure 20. The ProtonMail sign-in screen (left) recreated in the template project (right)

4.7.2.1 Codebase

To recreate the sign-up and sign-in flow for ProtonMail, 13 .xml files and no Kotlin files were changed. Due to the ProtonMail codebase being large and containing multiple modules, it could not be evaluated in its entirety, however from the implementation of the sign-up and sign-in flow, it is visible that an architecture similar to MVVM is used, with Activities used as the displaying of the UI instead of fragments. Similarly to the template project, Hilt was used for dependency injection. The codebase of ProtonMail is generally of very high quality, with differences to the template project mostly being in view architecture and complexity, as ProtonMail is an older and larger project.

5 Discussion

This section covers a comparison with similar project templates and tools, potential shortfalls of this thesis project, threats to validity, and additional work that might be done in the future.

Combining activity diagrams to find an average user onboarding flow proved to be relatively useful. The structure for each application used for evaluation proved to be similar enough that the template project, which was based on the combined activity diagram, could be used to recreate their onboarding user flows without very large modifications. The template project might benefit from simpler UI configurability, however, since most UI configuration is done in .xml files due to the project not using Jetpack Compose, having some UI configuration in the Kotlin code might make some parts of the template project harder to use or configure.

5.1 Comparing results to existing templates & tools

Blocoio's Android app starter template[23] provides a high-quality base for an application. It follows Clean Architecture to an even more detailed degree than the template project by splitting the project into modules for data, domain logic and UI. This project also uses compose to build its views, features unit tests, application tests and activity tests. From a technical standpoint, this could serve as a guideline for what to do to improve the template project in the future. It does not however feature any user flows and can be described more like an empty slate from which to build upon, whereas the template project created in the course of this thesis already features multiple user flows and can speed up development in that regard.

NimbleHQs Android templates[24] are separated into compose and XML variants but strongly resemble Blocoio's template due to both of them following best practices set by Google. Like the previously mentioned project, these templates also do not feature any user flows. Unlike Blocoio's template, this project features an executable script which renames the project and its packages to whatever is needed and lets the developer pick which variant they wish to use. This could also be implemented in the template project to speed up the setup process of starting a new project.

A lot of project templates exist on GitHub, but they often only contain one Activity or Fragment with setup done for the included dependencies, such as ones related to navigation, dependency injection, storage, etc. Another example is Codelight Studio's Android Smart

[23] https://github.com/blocoio/android-template
[24] https://github.com/nimblehq/android-templates

Login library[25], which does contain some functionality for logging a user in through various methods, but is outdated and does not contain any recognizable application architecture. The benefit of the template project created for this thesis project is having the basic setup for an application done, along with a real-life example of a user flow which is commonly found in many applications. This not only provides functionality but also thorough examples of how to use fragments, Jetpack navigation and ViewModels. Furthermore, the template project is based on user flows found in applications that are used and published on the Google Play Store, which makes it more reliable as a starting point for a project, as similar user flows are used in real-world applications.

5.2 Threats to validity

Since this thesis project analysed 30 applications in total, and a third of that for creating a combined activity diagram for each user flow under analysis, repeating the process with another set of applications might have returned a different result. This could be avoided by using a larger amount of applications, but due to the activity diagrams being created manually, this approach would increase the workload considerably. The approach to finding these open-source applications for this thesis project could be improved as well, as the code search functionality of GitHub proved to be less than ideal. Due to the lack of filtering for ratings, releases and popularity, repositories matching the requirements defined for the code search had to be filtered manually after entering the query for a specific user flow. This included opening each repository and looking at the ratings and amount of releases. Furthermore, the search was capped at 100 pages, even though there were more results shown in GitHub's sidebar.

While the activity diagrams could have been created using closed-source applications, the rest of the analysis would be less detailed, as the source code of the template project would not be able to be compared to the source code of the applications used for evaluation. The comparison of source codes helped both verify and improve the quality of the template project. For the same reason, evaluation was done on projects found during the initial code search. Evaluating the project template on different applications might have returned a result that is less biased to applications found during the code search, as all of the applications used for evaluation were used to create the combined activity diagram the user flows in the project template were based on.

[25] https://github.com/CodelightStudios/Android-Smart-Login

The use of activity diagrams proved useful for analysing user flows in applications, but determining common UI elements or design queues could not be done with the help of the activity diagrams, as these qualities were not reflected in them. Extracting activity diagrams from applications might also be considered somewhat subjective as there are no specific guides for wording the nodes of the activity diagrams or what the amount of required detail is.

5.3 Potential shortfalls for the template

The project template does not feature any sort of unit or integration testing.. Reasons for this include the lack of business logic and inevitable changes made by developers using the template as a base for a new project. Adding automated UI tests using frameworks such as Espresso[26] would ensure users do not encounter unexpected results, however they were not added due to time constraints. The architecture chosen for the project template provides a good separation of concerns and accounts for testability. Some improvements can be made, such as delegating handling of the result of the Google sign-in feature to a helper class to further increase separation of concerns.

Developers using the project template must account for performance related limitations of the view components used in the template. In the case that the images used in the onboarding screens are of a very large size, some performance issues occur in the ViewPager component and the transition between two screens becomes slower. This might be fixed by using vector-based images or just keeping the resolution of the image below 640x640.

If a developer were to use this project template as the base for a larger project, it might lack some capabilities such as a local database and more in-depth examples of how to properly use ViewModels and UseCases. Additionally, while functionality can be configured through navigation parameters, the design, positioning of the buttons, and colours used in the project have to be changed manually. This might not be a problem for a developer looking to add an onboarding flow to their project but might prove challenging to a beginner using this template project as a base for their first project.

[26] https://developer.android.com/training/testing/espresso

5.4 Future work

Development of the project template template will continue in the future. Due to the rising popularity of Jetpack Compose and Google starting to recommend it more often, the UI of the application will also be recreated using Compose in a separate branch hosted in GitHub [21]. If more feedback is received from developers using the created project template, it will be taken into consideration and potentially implemented into the project template. Additionally, the project will be periodically updated to keep it up to date with the latest updates to third-party libraries and other components.

At the time of writing, the project template is missing a network layer, which would be used for querying data and other types of communication with a server. Additional improvements may also include integrating a local database, e.g. Room, since many applications use a local database to cache data for longer-term usage, instead of constantly querying new information.

6 Conclusion

This thesis project gives an overview of whether using activity diagrams to describe and combine user flows of existing applications is a good approach and whether a usable project template can be developed from the results. The evaluation was done by recreating six user flows from existing applications and seeing how much additional development was needed in the template project to achieve a similar user interface. From the results of the evaluation, it is clear that the user flows used for comparison could be recreated in the template project without significant changes to the codebase, although significant changes to the Android Layout files were needed as most of the user interface configuration was done there. This indicates that systematically analyzing the user flows of existing applications is a useful starting point for creating easily adaptable project templates. The template project was published in a public GitHub repository after development was completed.

While this thesis project focused on flows used at the start of an application, specifically onboarding, sign-in and sign-up, this approach might be applied on other flows as well, although further evaluation is needed. Due to changing design trends and security requirements for mobile applications, the process of combining activity diagrams of existing applications could be repeated in the future to update the template project with newer features.

7 References

[1] Punchoojit, L., & Hongwarittorrn, N. (2017). Usability studies on mobile user interface design patterns: a systematic literature review. Advances in Human-Computer Interaction, 2017.

[2] Strahm, B., Gray, C. M., & Vorvoreanu, M. (2018, June). Generating mobile application onboarding insights through minimalist instruction. In Proceedings of the 2018 Designing Interactive Systems Conference (pp. 361-372).

[3] Taylor, P. (2023, February 21). Global Mobile OS Market Share 2022. Statista. Retrieved from https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (Last visited: 03.12.2022).

[4] Deka, B. (2016, October). Data-driven mobile app design. In Adjunct Proceedings of the 29th Annual ACM Symposium on User Interface Software and Technology (pp. 21-24).

[5] Akopian, D., Melkonyan, A., Golgani, S. C., Yuen, T. T., & Saygin, C. (2013). A template-based short course concept on android application development. Journal of Information Technology Education. Innovations in Practice, 12, 13.

[6] Lampropoulos, A. (2018). Mobile Application Development with reusable code.

[7] Barnett, S., Vasa, R., & Grundy, J. (2015, May). Bootstrapping mobile app development. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 657-660). IEEE.

[8] Eriksson, H., & Parflo, E. (2019). Mobile application onboarding processes effect on user attitude towards continued use of applications.

[9] Eisfeld, H., & Kristallovich, F. (2020). The rise of dark mode: A qualitative study of an emerging user interface design trend.

[10] Lardinois, F. (2019, May 7). Kotlin is now Google's preferred language for Android App Development. TechCrunch. Retrieved from https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/?guccounter=1 (Last visited: 03.04.2023).

[11] Flauzino, M., Veríssimo, J., Terra, R., Cirilo, E., Durelli, V. H., & Durelli, R. S. (2018, September). Are you still smelling it? A comparative study between Java and Kotlin

language. In Proceedings of the VII Brazilian symposium on software components, architectures, and reuse (pp. 23-32).

[12] Google. (2022). Introduction to activities. Retrieved from https://developer.android.com/guide/components/activities/intro-activities (Last visited: 07.05.2022).

[13] Android Developers. (2018). Single activity: Why, when, and how. YouTube. Retrieved from https://www.youtube.com/watch?v=2k8x8V77CrU (Last visited: 07.05.2022).

[14] Handling lifecycles with lifecycle-aware components. Google. (2023). Retrieved from https://developer.android.com/topic/libraries/architecture/lifecycle (Last visited: 07.05.2022).

[15] Guide to app architecture. Google. (2022). Retrieved from https://developer.android.com/topic/architecture (Last visited: 31.10.2022).

[16] van Gurp, Jilles & Bosch, Jan. (2002). Separation of Concerns: A Case Study.

[17] Tyagi, A. (2018, December 6). Better Android apps using MVVM with Clean Architecture. Toptal Engineering Blog. Retrieved from https://www.toptal.com/android/android-apps-mvvm-with-clean-architecture (Last visited: 11.03.2023).

[18] Packages Structure. Android Handbook. Infinum. (2020). Retrieved from https://infinum.com/handbook/android/project-structure/package-structure (Last visited: 05.12.2022).

[19] Nikiforova, A. Interaction Modelling: Activity Diagrams. Software Modelling Lecture 8. University of Tartu. (2022). Retrieved from https://drive.google.com/file/d/1uFxAfSRIIbCFHOTjUaU80wwUAZXaWw2B/view (Last visited: 03.04.2023).

[20] Vestola, M. (2010). A comparison of nine basic techniques for requirements prioritization. Helsinki University of Technology, 1-8.

[21] Why adopt compose. Google. (2022). Retrieved from https://developer.android.com/jetpack/compose/why-adopt (Last visited: 03.04.2022).

[22] Verdecchia, R., Malavolta, I., & Lago, P. (2019, March). Guidelines for architecting android apps: A mixed-method empirical study. In 2019 IEEE International Conference on Software Architecture (ICSA) (pp. 141-150). IEEE.

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Kristofer Käosaar**,

(*author's name*)

1. grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis
   Detection of Commonly Reused Components in Open-Source Android Applications,
   (title of thesis)

   supervised by Kristiina Rahkema.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kristofer Käosaar
09.05.2023

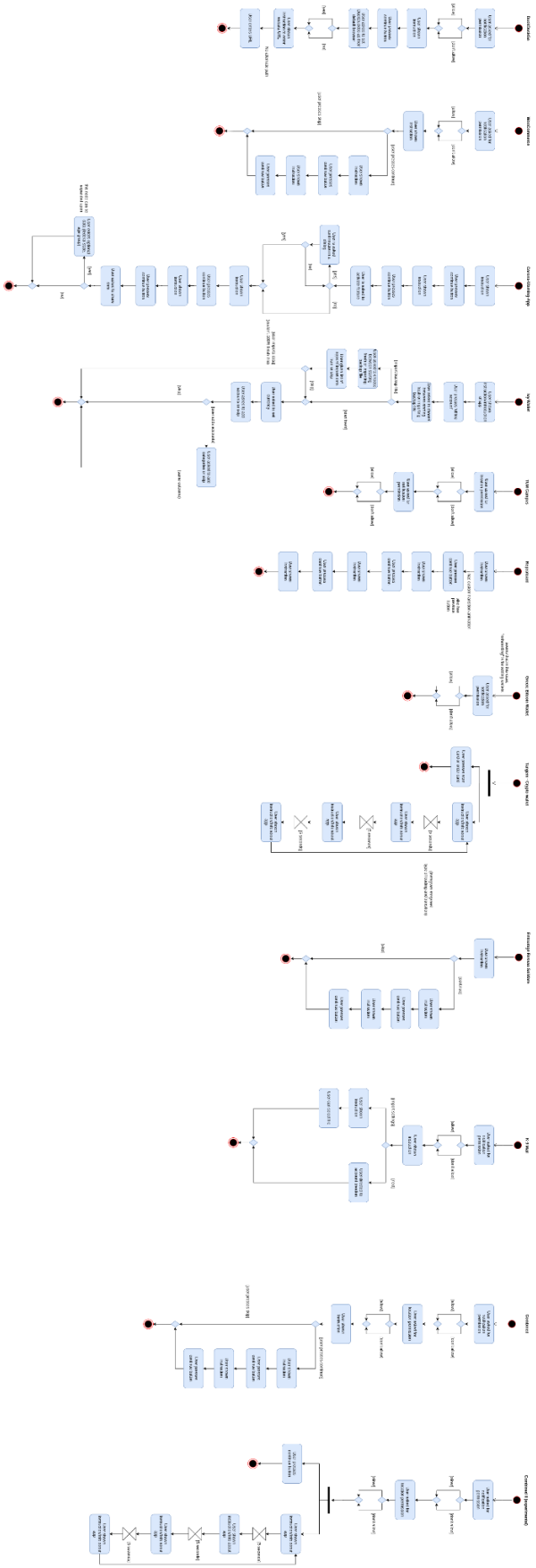
Appendix 1 - Applications for the onboarding flow

Project name & Description	Link to repository
DuckDuckGo - Web browser	https://github.com/duckduckgo/Android
CWA - Corona warning application	https://github.com/corona-warn-app/cwa-app-android
WooCommerce - eCommerce platform application	https://github.com/woocommerce/woocommerce-android
Ivy Wallet - Wallet/Money tracker	https://github.com/Ivy-Apps/ivy-wallet
TUM Campus Android - University campus application	https://github.com/TUM-Dev/Campus-Android
Repeatcard - Application to create and memorize Flashcards	https://github.com/ilkeraslan/repeatcard
Green Android - Cryptocurrency wallet	https://github.com/Blockstream/green_android
Tangem - Crypto Wallet / Blockchain application	https://github.com/tangem/tangem-app-android
Entourage Reseau Solidaire - Social network app for homeless people	https://github.com/ReseauEntourage/entourage-android
K-9 Mail - email client	https://github.com/thundernest/k-9

Appendix 2 - Applications for the sign up and log in flow

Project name & Description	Link to repository
ProtonMail - Email client	https://github.com/ProtonMail/protoncore_android
Muun - Apollo - Crypto wallet	https://github.com/muun/apollo
xxMessenger - Messaging application	https://github.com/xxfoundation/elixxir-xx-Messenger-Android
ProtonVPN - VPN Client for Android	https://github.com/ProtonVPN/android-app
Posture reminder - application to remind the user to keep a straight posture	https://github.com/puntogris/posture-reminder
WooCommerce - Commerce application	https://github.com/woocommerce/woocommerce-android
Taz - Newspaper reader	https://github.com/die-tageszeitung/taz-neo
Entourage Reseau Solidaire - Social network app for homeless people	https://github.com/ReseauEntourage/entourage-android

Appendix 3 - Onboarding activity diagrams



Appendix 4 - Sign-in & sign-up activity diagrams