МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

**Департамент математического
и компьютерного моделирования**

«Навигация в Android: сравнительный анализ библиотек и инструментов для эффективной навигации между экранами в приложении»

КУРСОВАЯ РАБОТА
по образовательной программе подготовки бакалавров
по направлению 01.03.02 – «Прикладная математика и информатика»

Выполнила студентка гр. Б9121-01.03.02пм
_____ Гетун А.Д.
(*подпись*) *(Ф.И.О.)*

Проверила ассистент

Чернышева Г. Ю._____
(подпись)

«21»___июня_____2024г.

г. Владивосток
2024

Содержание

Введение

Навигация между экранами в приложении — это процесс взаимодействия пользователя с мобильным сервисом, когда он открывает приложение, перемещается между экранами и фрагментами экранов.

Хорошая навигация в приложении улучшает пользовательский опыт, поскольку помогает пользователям ориентироваться в приложении и достигать целевых действий за минимальное количество шагов, обеспечивает понятное и удобное взаимодействие с мобильным сервисом, упрощает перемещение между экранами и выбор разделов меню, делая процесс интуитивно понятным и быстрым.

Правильно выбранный способ навигации улучшает жизнь не только пользователям, но и разработчикам. Инструменты и библиотеки навигации могут влиять на вес приложение, на размер кода и его читаемость.

Объектом исследования в данной работе являются библиотеки и инструменты, реализующие навигацию между экранами в Android-приложениях.

Цель работы — ознакомиться с библиотеками и инструментами, использующимися для реализации навигации в Android-приложениях, сравнить их, а именно:

1. Выбрать несколько библиотек и инструментов для исследования.
2. Сравнить исследуемые методы и выбрать наилучший.
3. Разработать приложение с использованием выбранной библиотеки.
4. Приобрести навыки представления итогов проделанной работы в виде отчета, оформленного в соответствии с имеющимися требованиями, с привлечением современных средств редактирования и печати.

Работа над курсовой работой предполагает выполнение следующих задач:

1. Дальнейшее углубление теоретических знаний и их систематизацию.
2. Получение и развитие прикладных умений и практических навыков.
3. Развитие навыков самостоятельной работы.
4. Развитие навыков обработки полученных результатов, анализа и осмысления их с учетом имеющихся литературных данных.
5. Повышение общей и профессиональной эрудиции.

Основные принципы навигации в Android приложениях

Навигация между различными экранами и приложениями является ключевой частью пользовательского опыта. Следующие принципы задают основу для согласованного и интуитивно понятного взаимодействия пользователя с приложениями [1].

Фиксированный пункт назначения

Каждое создаваемое приложение имеет фиксированный начальный пункт назначения. Это первый экран, который видит пользователь при открытии приложения из панели запуска. Этот пункт назначения также является последним экраном, который видит пользователь, когда возвращается к панели запуска после нажатия кнопки «Назад».

Состояние навигации представлено в виде стека пунктов назначения

При первом запуске приложения на устройстве создается новая задача, и приложение отображает стартовый пункт назначения. Он становится базовым пунктом назначения стека переходов назад, или бэкстека (back stack), который является основой навигации приложения. Вершина стека – это текущий экран, а предыдущие пункты назначения в стеке представляют историю посещенных экранов. Начальный пункт назначения приложения всегда находится в нижней части бэкстека.

Операции, изменяющие задний стек, всегда выполняются в верхней части стека, либо помещая новый пункт назначения на вершину стека, либо удаляя самое верхнее место назначения из стека.

Кнопки “Вверх” и “Назад” идентичны в рамках задачи приложения

Кнопка «Назад» появляется в системной навигационной панели внизу экрана и используется для навигации в обратном хронологическом порядке по истории экранов, с которыми работал пользователь. При нажатии кнопки «Назад» текущий пункт назначения выталкивается из верхней части бэкстека, а затем происходит переход к предыдущему пункту назначения.

Кнопка «Вверх» появляется в верхней части экрана. В рамках задач приложения кнопки «Вверх» и «Назад» ведут себя одинаково.

Кнопка «Вверх» никогда не выходит из приложения

Если пользователь находится в начальном пункте назначения приложения, кнопка «Вверх» не появляется, поскольку она не позволяет завершать работу приложения.

Когда ваше приложение запускается с использованием глубоких ссылок из другого приложения, кнопка «Вверх» проводит пользователей по вашему приложению через имитированный синтетический бэкстек, а не на приложение, которое вызвало глубокую ссылку. Однако кнопка «Назад» возвращает к другому приложению.

Сравнительный анализ инструментов для навигации

Вкладки

Рис 1: Панель вкладок

Панель вкладок (Рис 1) обычно располагается в верхней части экрана. Вкладки обеспечивают быструю навигацию между родственными представлениями внутри одного и того же родительского экрана. Они лежат в одной плоскости, что означает, что их можно пролистывать. Вкладки отлично подходят для фильтрации, сегментации или обеспечения глубины связанным фрагментам контента [2].

Достоинства

- возможна реализация скроллинга, что позволяет содержать неограниченное количество элементов.
- подходят для фильтрации, сегментации или обеспечения глубины связанным фрагментам контента.

Недостатки

- не подходят для несвязанных частей контента.
- обычно реализуются внутри одного экрана.

Нижняя навигация

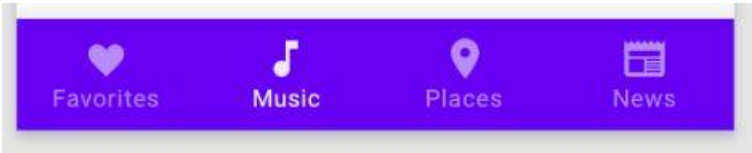

Рис 2: Нижняя навигация

Компонент нижней навигации (Рис 2) включает в себя от трех до пяти основных пунктов назначения. Нижняя навигация работает лучше всего, когда ваше приложение имеет ограниченное количество разрозненных пунктов назначения верхнего уровня, которые должны быть мгновенно доступны.

Достоинства

- мгновенный переход с дочернего экрана на несвязанный родительский экран без возврата назад к текущей родительской навигации.
- легкодоступна для однопальцевого взаимодействия, поскольку находится внизу экрана, в области, которую легко достать пальцем.

Недостатки

- должна содержать от трех до пяти элементов.
- не подходит для использования скроллинга.

Меню «гамбургер»

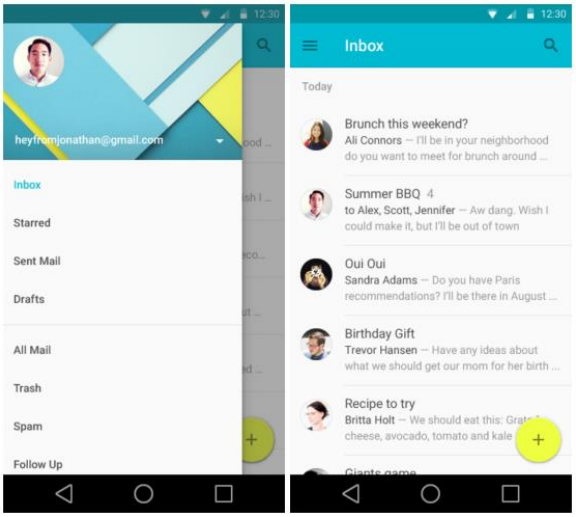

Рис 2: Меню "гамбургер"

В ограниченном пространстве мобильного интерфейса гамбургер-меню (Рис 3) скрывает сложную навигацию, чтобы избежать отвлекающих

элементов. Выдвижная панель скрывает элементы и активизируется только в том случае, если пользователь нажмет на иконку в левом верхнем углу.

Достоинства

- экономия места;
- чистый дизайн;
- возможность «отфильтровать» опции главного меню.

Недостатки

- неочевидность: когда меню спрятано, то вероятность его использования снижается.
- значок «гамбургера» скрывает контекст. Текущее положение пользователя не видно с первого взгляда. Информация появляется только после нажатия на иконку;
- обязательно совершить лишнее действие, чтобы выполнить желаемое действие. Для открытия определённой страницы требуется минимум два нажатия (один раз на значок меню, а другой — на целевой параметр).

Всплывающее меню

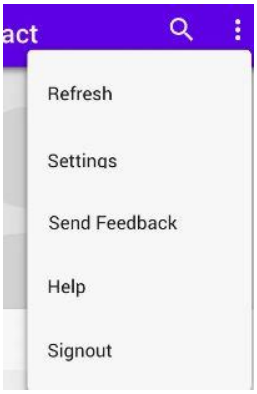

Рис 3: Всплывающее меню

Всплывающее меню (Рис 4) отображает элементы в вертикальном списке. Всплывающее меню обычно используется для дополнительных пунктов меню и может быть доступно через другие инструменты навигации.

Этот инструмент предназначен для предоставления пользователю быстрого доступа к дополнительным или мало используемым функциям, которые не должны быть доступны через более заметные пункты меню.

Достоинства

- может содержать всё, что не помещается в основном меню и на экране.
- занимает мало места на экране и отображается только при клике на него.

Недостатки

- неочевидность. Пользователь не узнает, что можно делать с помощью этого инструмента, пока не нажмет на него.
- совершение лишних действий для достижения нужного результата.

Контекстная навигация

Контекстная навигация состоит из любого навигационного взаимодействия вне инструментов, описанных выше. Это включает в себя такие вещи, как кнопки, плитки и карточки. Контекстная навигация обычно менее линейна, чем явная навигация. Взаимодействия могут переносить пользователя через иерархию, между различными шагами в дискретных иерархиях или к полному выходу из приложения.

Достоинства

- возможность переходить в любой пункт назначения приложения.
- может находиться в любой части экрана.

Недостатки

- не рекомендуется для приложений со сложной навигацией.
- занимают большую часть экрана.
- кроме самой навигации пользователь не видит любого другого контента.

Сравнение инструментов навигации

Сравним рассмотренные инструменты навигации.

Контекстную навигацию рекомендуется применять на мобильных устройствах с большим экраном и планшетах, поскольку кнопки, карточки и плитки занимают много места.

Панель вкладок и нижнюю навигацию можно применять всегда, потому что они не занимают много места, а также содержание их элементов, то есть иконки и названия, интуитивно понятны пользователю и направляют его в нужные пункты назначения.

Всплывающее меню и меню «гамбургер» привлекательны тем, что занимают минимум места на экране, однако их назначение может быть неочевидным для пользователя до взаимодействия с этими инструментами.

Сравнительный анализ библиотек для навигации

FragmentManager

FragmentManager – это класс в Android, ответственный за выполнение действий над фрагментами приложения [5]. Использование этого класса напрямую маловероятно, поскольку многие библиотеки, работающие с фрагментами, используют FragmentManager на различных уровнях.

FragmentManager управляет обратным стеком фрагментов. Во время выполнения он может выполнять операции обратного стека, такие как

добавление или удаление фрагментов. Каждый набор изменений фиксируется вместе как единый блок, называемый FragmentTransaction.

Поддержка нескольких задних стеков

В некоторых случаях приложению может потребоваться поддержка нескольких задних стеков. Например, приложение может использовать нижнюю панель навигации. FragmentManager позволяет поддерживать несколько обратных стеков с помощью методов savebackStack() и restoreBackStack(). Эти методы позволяют переключаться между задними стеками, сохраняя один и восстанавливая другой.

Команды FragmentManager

addToBackStack() – добавляет транзакцию на вершину стека. Транзакция может включать в себя любое количество операций, например добавление нескольких фрагментов или замену фрагментов в нескольких контейнерах.

popBackStack() - извлекает текущую транзакцию из заднего стека и переходит к предыдущей транзакции. Когда транзакция извлекается из стека, то все операции составляющие ее, меняются как одно атомарное действие. Если в стеке больше нет транзакций и вы не используете дочерние фрагменты, к действию подключается событие Back.

commit() – завершает транзакцию и выполняет операции транзакции.

Список транзакций

add() – помещает переданный фрагмент на вершину стека.

remove() – удаляет из стека переданный фрагмент.

replace() – заменяет фрагмент на переданный.

Итог

Преимущества использования FragmentManager:

- Класс работает с фрагментами, поэтому файл манифеста становится более чистым и удобным в сопровождении.
- Централизованное управление навигацией. Диспетчер FragmentManager внедрен в каждый компонент. Его можно использовать для сбора логов и управления стеком переходов, поэтому схемы навигации можно отделить от бизнес-логики и не разделять по реализациям разных экранов.
- Улучшенное взаимодействие между экранами. Между активностями могут передаваться только бандлы, а фрагменты могут получать друг от друга произвольные объекты.

Недостатки класса:

- Зависит от жизненного цикла фрагментов, различные неприятные ситуации необходимо обрабатывать самостоятельно.
- В коде приходится использовать много шаблонного кода, поскольку приходится постоянно описывать транзакции, из-за чего легко допустить ошибку.

Cicerone

Cicerone – это легкая библиотека, которая упрощает навигацию в приложении для Android [3].

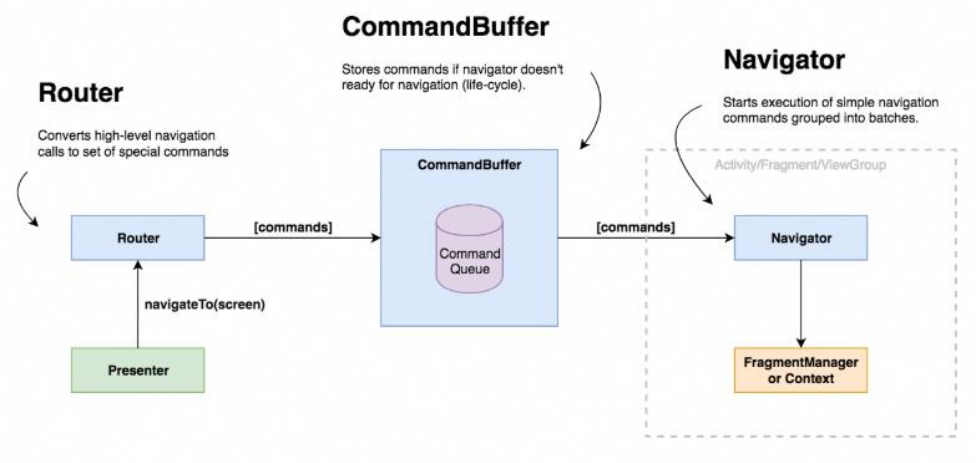

Рис 4: Навигация в Cicerone

Эта библиотека работает следующим образом:

Presenter вызывает метод навигации Router. Router преобразует вызов навигации в набор команд и отправляет их в CommandBuffer. CommandBuffer проверяет, есть ли активный Navigator:

- Если да, то он передает команды Navigator. Navigator обрабатывает их для достижения желаемого перехода.
- Если нет, то CommandBuffer сохраняет команды в очередь и применяет их, как только появится новый Navigator.

Navigator обрабатывает навигационные команды.

Опишем подробней сущности, использующиеся в навигации.

Presenter – связующее звено между представлением и моделью, которые не должны общаться напрямую.

Command – это простейшая команда перехода, которую выполняет Navigator.

Router – это класс, который превращает вызовы навигации в набор Command. Он занимается реализацией переходов при помощи комбинации команд. В

библиотеке есть готовый экземпляр, используемый по умолчанию, с самыми необходимыми переходами, но можно создать и свою реализацию.

Список переходов, реализованный в базовом экземпляре:

- navigateTo() – переход на новый экран.
- newScreenChain() – сброс цепочки до корневого экрана и открытие нового.
- newRootScreen() – сброс цепочки и замена корневого экрана.
- replaceScreen() – замена текущего экрана.
- backTo() – возврат на любой экран цепочки.
- exit() – выход с экрана.

Navigator – это интерфейс, который отвечает за переключение экранов внутри контейнера. В зависимости от задачи Navigator будет реализован по-разному, но он всегда будет там, где находится контейнер для переключаемых экранов. Поскольку в подавляющем большинстве андройд приложений навигация опирается на переключение фрагментов внутри активити, то в библиотеке уже есть готовый FragmentNavigator, реализующий представленные команды.

CommandBuffer – класс, который отвечает за доставку команд навигации Navigator. Если в CommandBuffer поступят команды, а в данный момент он не содержит Navigator, то они сохранятся в очереди и будут выполнены сразу при установке нового Navigator. Именно благодаря CommandBuffer удалось решить все проблемы жизненного цикла.

Команды навигации

Библиотека обладает малым числом команд навигации, однако они удовлетворяют потребности большинства приложений. Любую карту

переходов можно реализовать, используя четыре базовых перехода, комбинируя которые можно получить необходимое поведение.

1. Forward(Object screen, Object transitionData) – команда, которая осуществляет переход на новый экран, добавляя его в текущую цепочку экранов.
    1. screen – объект, реализующий интерфейс Screen.
    2. transitionData – данные, необходимые новому экрану.
2. Back() – команда, удаляющая последний активный экран из цепочки, и возвращающая на предыдущий. При вызове на корневом экране ожидается выход из приложения.
3. BackTo(Object screen) – команда, позволяющая вернутся на любой из экранов в цепочке. Если в цепочке два экрана с одинаковым ключом, то выбран будет тот, что ближе к концу стека. Если указанный экран не найден, или в параметр ключа передается null, то будет осуществлен переход на корневой экран.
4. Replace(Object screen, Object transitionData) – команда, заменяющая активный экран на новый. Этого же результата можно достичь, вызвав подряд команды Back и Forward, но тогда на корневом экране мы выйдем из приложения.

Итог

Библиотека Cicerone была создана для использования максимально стандартного подхода без различных фреймворков. По этой причине библиотека не содержит шаблоны инструментов навигации. Их необходимо реализовывать вручную или использовать другие библиотеки. Однако в контексте навигации такой подход позволит новым разработчикам быстро подключаться к проекту без необходимости изучения сторонних фреймворков, а также не придется полагаться на сложные сторонние решения, которые могут повлечь за собой затрудненную поддержку.

Библиотека Cicerone имеет следующие преимущества:

- Легкая библиотека
- Не зависит от жизненного цикла
- Предоставляет быстрые вызовы
- Легка в расширении

Недостатки Cicerone:

- Сторонний инструмент. Его использование не так популярно, как использование официальных библиотек, поэтому возможно прекращение поддержки.
- Нет возможности работать с диалоговыми и модальными окнами.

Decompose

Decompose – это многоплатформенная библиотека Kotlin для разбиения кода на древовидные компоненты бизнес-логики с учетом жизненного цикла с функцией маршрутизации [4]. Decompose разбивает код на небольшие и независимые компоненты и организует их в деревья, при этом каждый родительский компонент знает только о своих непосредственных дочерних компонентах.

Библиотека предоставляет различные способы навигации. Здесь навигация – это функция от старого состояния к новому. Состояние навигации полностью открыто – можно подключить любой пользовательский интерфейс.

Навигация

Навигация в Decompose устроена следующим образом: Decompose хранит два синхронизированных друг с другом стека – стек конфигураций и стек компонентов.

Конфигурация – это класс, который представляет дочерний компонент и содержит все его аргументы. На основе конфигураций создаются сами компоненты. Созданием компонентов из конфигураций управляет Decompose. Мы не можем изменять стек компонентов напрямую, мы лишь манипулируем стеком конфигураций, а Decompose автоматически меняет стек компонентов.

Причина создания двух стеков заключается в особенностях системы Android. Свернутое приложение может быть выгружено из памяти, а когда пользователь возвращается в приложение, стек экранов и данные на них должны быть восстановлены. В этот момент пригождаются конфигурации. Decompose сохраняет и восстанавливает стек конфигураций, а после создает сами компоненты.

В настоящее время данная библиотека предоставляет две предопределенные модели навигации:

- Дочерний стек
- Дочерний слот

Дочерний стек

Дочерний стек – это модель навигации для управления стеком компонентов.

Каждый компонент имеет свой собственный жизненный цикл. Каждый раз, когда новый компонент помещается в стек, текущий активный компонент останавливается. Когда компонент извлекается из стека, воспроизведение предыдущего компонента возобновляется. Это позволяет

выполнять бизнес-логику, пока компонент находится в заднем стеке. В компоненте может быть более одного дочернего стека, также поддерживаются вложенные стеки.

Навигация дочернего стека состоит из двух основных объектов:

- ChildStack – класс данных, хранящий стек компонентов и их конфигураций.
- StackNavigation – интерфейс, который принимает команды навигации и пересылает их всем подписанным наблюдателям.

Вся навигация осуществляется с помощью StackNavigator интерфейса, который расширяется интерфейсом StackNavigation. StackNavigator содержит navigate метод со следующими аргументами:

- transformer – преобразует текущий стек конфигураций в новый. Стек представлен как List, где последний элемент – это верх стек, а первый элемент – низ стека.
- onComplete – вызывается, когда навигация завершена.

В процессе навигации новый стек конфигураций сравнивается с предыдущим. Это гарантирует, что все удаленные компоненты будут уничтожены и что одновременно возобновляется только один компонент – верхний. Все компоненты в заднем стеке всегда либо останавливаются, либо уничтожаются.

Дочерний слот

Дочерний слот – это модель навигации, которая позволяет использовать только один дочерний компонент или ни одного. Каждый из дочерних слотов позволяет активировать дочерний компонент, заменить его другим дочерним компонентом или закрыть, когда он не нужен. В

компоненте может быть более одного слота, также поддерживаются вложенные слоты.

Наиболее распространенные варианты использования включают, помимо прочего, отображение диалоговых окон, ящиков, нижних листов и просто изменение видимости некоторых представлений. Это не обязательно что-то, что перекрывает родительский компонент.

Навигация состоит из двух основных объектов:

- ChildSlot – класс данных, содержащий активный в данный момент дочерний элемент, если таковой имеется.
- SlotNavigation – интерфейс, который принимает команды навигации и пересылает их всем подписанным наблюдателям.

Если в дочернем слоте одного компонента используется несколько компонентов, с каждым из них должен быть связан уникальный ключ. Ключи должны быть уникальными только в пределах родительского компонента, поэтому вполне допустимо, чтобы разные компоненты имели одинаковые ключи.

Команды переходов

Библиотека обладает большим списком команд навигации:

- push(configuration) – помещает предоставленную конфигурацию наверх стека. Библиотека выдаст исключение, если предоставленное значение конфигурации уже присутствует в стеке.
- pushNew(configuration) – помещает предоставленную конфигурацию наверх стека. Библиотека выдаст исключение, если предоставленное значение уже присутствует в заднем стеке, но не в верхней его части. Эта команда может быть полезна при нажатии кнопки, чтобы избежать

нажатия одного и того же компонента, если пользователь быстро нажимает одну и ту же кнопку несколько раз.

- pushToFront(configuration) – помещает предоставленную конфигурацию наверх стека, удаляя конфигурацию из заднего стека, если таковая имеется.
- pop – выталкивает последнюю конфигурацию наверх стека.
- popWhile(predicate) – отбрасывает конфигурации наверх стека, а предоставленный предикат возвращает true.
- popTo(index) – удаляет конфигурации на вершине стека, чтобы предоставленный индекс стал активным.
- replaceCurrent(configuration) – заменяет текущую конфигурацию в верхней части стека предоставленной конфигурацией.
- replaceAll(vararg configurations) – заменяет все конфигурации, находящиеся в настоящее время в стеке, предоставленными конфигурациями. Компоненты, оставшиеся в стеке, не воссоздаются, компоненты, которых больше нет в стеке, уничтожаются.
- bringToFront(configuration) – удаляет все компоненты с конфигурациями предоставленного класса и добавляет предоставленные конфигурации в начало стека.

Итог

Библиотека Decompose имеет следующие преимущества в навигации:

- Разбивает код на небольшие и независимые компоненты и организует их в деревья. Эта возможность позволяет бороться со сложностью экранов и навигации в больших приложениях.
- Подходит для создания функциональных блоков и структуры приложения.
- Проводит четкие границы между кодом пользовательского интерфейса и кодом, не связанным с пользовательским интерфейсом.

- Компоненты в заднем стеке не уничтожаются, они продолжают работать в фоновом режиме без пользовательского интерфейса.
- Навигация – это чистая функция от старого состояния к новому.

Библиотека имеет следующие недостатки:

- Сторонний инструмент. Его использование не так популярно, как использование официальных библиотек, поэтому возможно прекращение поддержки.
- Сложно интегрировать в существующий проект из-за иной системы навигации.
- Высокий порог входа.

Navigation Architecture Component

Библиотека Navigation Architecture Component упрощает осуществление навигации, а также помогает визуализировать навигационный поток приложения [6].

Чтобы организовать навигацию с помощью Navigation Architecture Component необходимо:

1. Создать граф
2. Добавить в него экраны
3. Добавить переходы между ними
4. Положить всё вышеперечисленное в NavHost
5. Указать переходы в коде

Компоненты навигации

1. Navigation graph – это ресурс XML, который содержит всю связанную с навигацией информацию в одном централизованном месте. Navigation graph представляет собой новый тип ресурса, который определяет все возможные пути, доступные пользователю в приложении. Он визуально показывает все пункты назначения, которые могут быть достигнуты из данного пункта назначения. Редактор навигации Android Studio отображает Navigation graph наглядно. Это граф, в котором вершины – экраны, а ребра – переходы между ними. Граф создается в отдельном xml-файле в папке res/navigation. Чтобы не вводить navController в заблуждение, необходимо в графе указать его стартовую точку.
2. Destination представляет UI-единицу на графе. Компонент навигации представляет концепцию пункта назначения. Пункт назначения – это любое место, в котором вы можете перемещаться в приложении, обычно это фрагмент или активити, но можно создать свои собственные типы назначения, если это необходимо.
3. Action обозначает переход между пунктами назначения графа. Может быть как направленным от одного экрана к другому, так и глобальным, то есть не иметь вершины, от которой происходит переход.
4. NavHost – контейнер, в котором будет происходить навигация. Это элемент пользовательского интерфейса, содержащий текущий пункт назначения навигации. Таким образом, когда пользователь перемещается по приложению, приложение меняет места назначения на хост навигации и обратно.
5. NavController – объект, который управляет навигацией приложения в NavHost. Контроллер предлагает методы навигации между пунктами назначения, обработки глубоких ссылок, управления обратным стеком и многое другое. Он содержит граф навигации и предоставляет методы, которые позволяют приложению перемещаться между пунктами назначения в графе. NavController содержит задний стек, состоящий из

пунктов назначения, которые посетил пользователь. Когда пользователь перемещается по экранам приложения, то NavController добавляет и удаляет пункты назначения в заднем стеке.

Команды навигации

- NavController.navigate() – помещает заданное место назначения на вершину стека.
- NavController.popBackStack() – извлекает текущий пункт назначения из заднего стека и переходит к предыдущему пункту назначения. С помощью этого метода также можно перейти к определенному пункту назначения, передав его идентификатор.
- NavController.navigateUp() – также как и popBackStack() извлекает текущий пункт назначения из заднего стека и переходит к предыдущему пункту назначения. Предполагаемое поведение этой команды отличается от popBackStack() в ситуации, когда пользователь не достиг текущего пункта назначения из собственной задачи приложения. Например, когда пользователь просматривает документ или ссылку в текущем приложении в действии, размещенном в задаче другого приложения, из которого пользователь перешел по ссылке.
- popUpTo(id) – удаляет пункты назначения из заднего стека до указанного пункта назначения как часть вызова NavController.navigate().

Безопасность типов при передаче информации во время навигации

Safe Args—это плагин, идущий отдельно от Navigation Component, но созданный специально для того, чтобы с библиотекой работалось легче. С ним нет необходимости указывать id destination и передавать параметры через Bundle—плагин генерирует для этого отдельные классы и имеет набор extension для работы с ними.

Вместе с плагином появился новый тег в xml: <argument>. Он применим и к action, и к destination—так можно передавать и принимать параметры в более удобном виде. Во-вторых, на основе экранов и переходов, указанных в графе, генерируются специальные классы, которые можно указывать в NavController-е вместо id action.

Поведение по умолчанию для анимации и переходов

Navigation Component позволяет добавлять к действиям анимацию. Библиотека содержит несколько анимаций по умолчанию, а также позволяет создать собственную. Библиотека предоставляет следующие типы анимации:

- Переход на новый пункт назначения.
- Переход к предыдущему пункту назначения.
- Переход на новый пункт назначения с помощью всплывающего действия - действия, которое извлекает дополнительные пункты назначения из бекстека при навигации.
- Переход к предыдущему пункту назначения с помощью всплывающего действия.

Итог

Библиотека Navigation Architecture Component предоставляет ряд преимуществ:

- Автоматическая обработка транзакция фрагментов.
- Корректная обработка кнопок «Вверх» и «Назад» по умолчанию.
- Поведение по умолчанию для анимации и переходов.
- Безопасность типов при передаче информации во время навигации.
- Инструменты Android Studio для визуализации и редактирования навигационного потока приложения.

- Поддержка таких шаблонов пользовательского интерфейса, как menus, drawer и bottom navigaion, с минимальной дополнительной работой.

Библиотека Navigation Architecture Component имеет следующие недостатки:

- Библиотека сильно завязана на жизненном цикле фрагментов. Если фрагмент находится в уничтоженном состоянии, то буферизацию команд навигации придется реализовывать самостоятельно.
- Зависимость от версии приложения Android Studio. Многие функции библиотеки могут не поддерживаться в старых версиях приложения.

Сравнение библиотек навигации

FragmentManager — это класс Android, ответственный за выполнение действий над фрагментами. Большинство библиотек для навигации опираются на этот класс. Поэтому FragmentManager будет использоваться в любом приложении с фрагментами. Существенным минусом этого класса, является то, что он зависит от жизненного цикла фрагментов. Поскольку команды класса работают с транзакциями, которые нужно всегда описывать, то в коде присутствует много шаблонности.

Cicerone — это библиотека, целью которой является только упрощение навигации. Она не содержит никаких шаблонов навигации, поэтому при выборе её вам необходимо самостоятельно реализовывать шаблоны или использовать для этого другие библиотеки. Это способствует увеличению кода и усложнению проекта. Однако в контексте навигации такой подход позволит новым разработчикам подключаться к проекту без изучения различным фреймворков. Существенным плюсом является независимость от жизненного цикла фрагментов. Cicerone является сторонним решением, и его используют не так часто как официальные решения. В связи с этим возможно прекращение поддержки. Еще одним недостатком библиотеки является отсутствие возможности работать с диалоговыми и модальными окнами.

Decompose мало подходит для приложений с множеством UI-элементов, поскольку в них нет бизнес-логики. Библиотека сложна для интеграций в существующие проекты из-за иной системы навигации. Однако она отлично подходит для создания функциональных блоков и структуры сложных приложений, соответственно из-за этого высок порог входа для разработчиков. Библиотека разработана таким образом, что компоненты заднего стека продолжают работать в фоновом режиме без пользовательского интерфейса, что положительно сказывается на пользовательском опыте. Decompose является сторонним решением, и его используют не так часто, как официальные решения. В связи с этим возможно прекращение поддержки.

Основными преимуществами библиотеки Navigation Architecture Component являются визуализация навигационного потока приложения и поддержка многих инструментов навигации. Библиотека подходит для быстрой реализации приложений с большим навигационным потоком, и к тому же уменьшает размер кода. В библиотеке реализованы автоматическая обработка транзакций и поведение по умолчанию для анимации и переходов.

Из серьезных минусов Navigation Architecture Component можно выделить зависимость от жизненного цикла фрагментов.

Использование выбранных библиотеки и инструментов

При сравнении библиотек для навигации было принято решение использовать Navigation Architecture Component. Эта библиотека позволяет быстро включиться в разработку и упрощает создание элементов, отвечающих за улучшение пользовательского опыта. Визуализация навигационного потока значительно упрощает понимание навигации в приложении и ускоряет процесс разработки.

В качестве инструментов навигации были выбраны нижняя навигация, всплывающее меню и контекстная навигация, представленная в виде кнопок и карточек. Использование меню “гамбургер” не понадобилось, поскольку приложение не очень большое.

Создано приложение “Телефонный справочник”, включающее в себя:

- список контактов, возможность добавления нового контакта
- экран с профилем пользователя, возможность изменения имени пользователя
- экран с дополнительными сервисами, такими как “Шаги” и “Новости”

Нижняя навигация (Рис 6) позволяет перемещаться между элементами “Профиль”, “Контакты”, “Сервисы”, поскольку эти элементы не связаны логически, они представляют отдельные сущности.

Рису 5: Реализация нижней навигации

Всплывающее меню (Рис 7) содержит элементы, не оказывающие значительного влияния на суть приложения, это “Информация о приложении”, “Изменить информацию”.

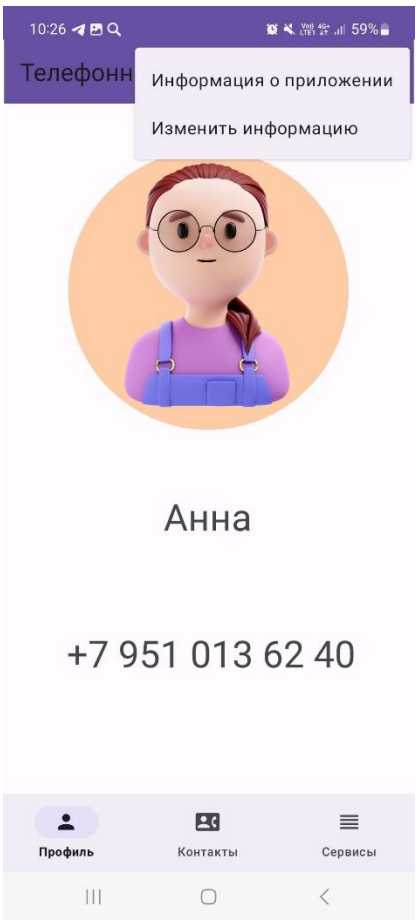

Рис 6: Реализация всплывающего меню

Контекстная навигация (Рис 8, 9) отвечает за переходы между списком и его элементами, а также за навигацию между разными компонентами.

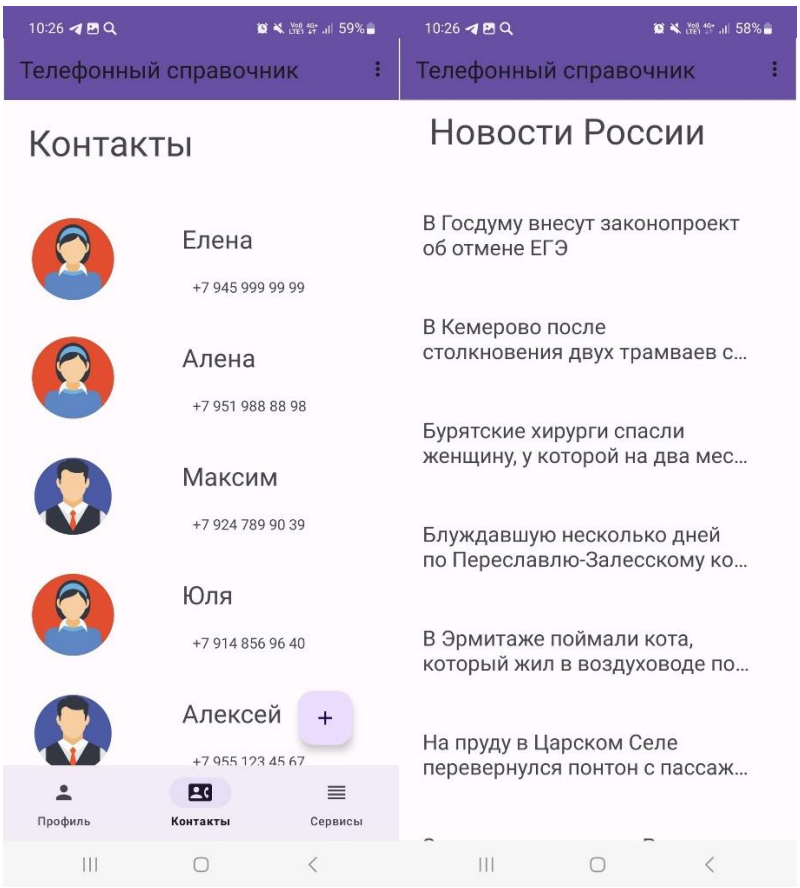

Рис 7: Переход по элементам списка

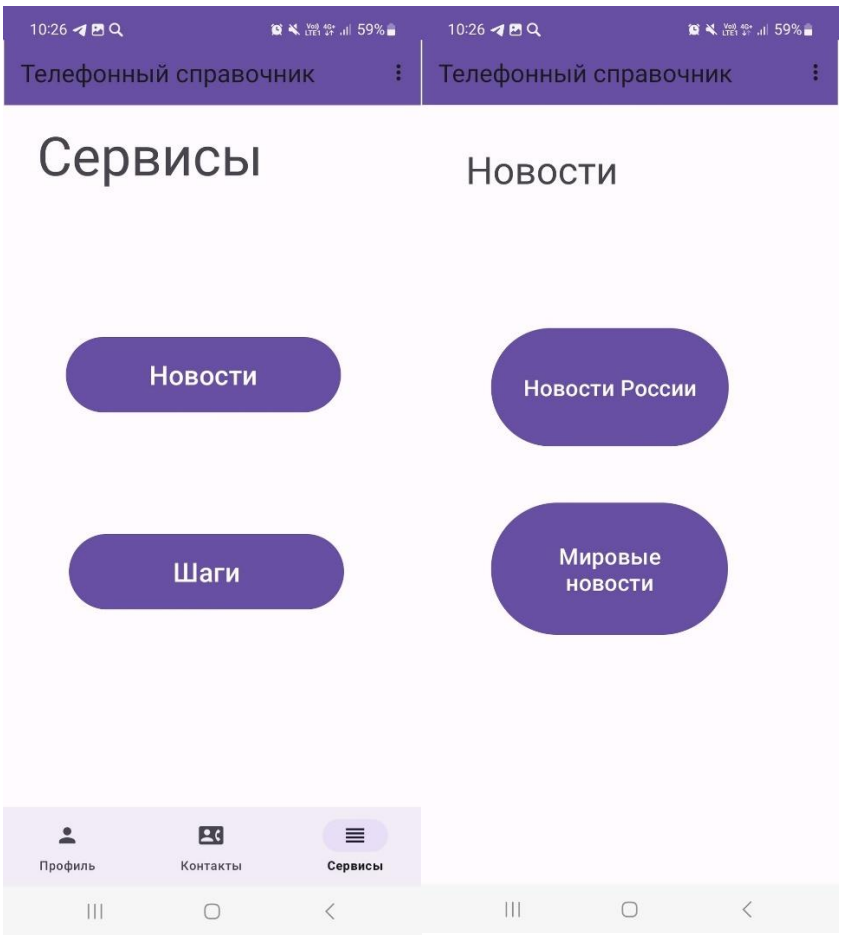

Рисунок 8 Переход по кнопкам

Визуализация навигационного потока (Рис 10) с помощью Navigation Architecture Component.

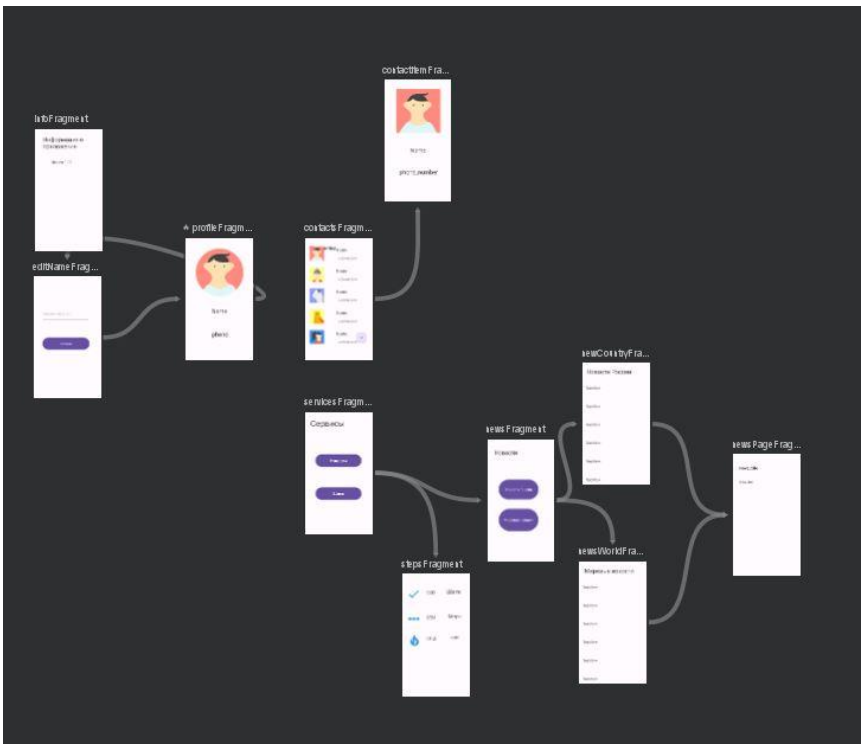

Рис 9: Навигационный поток приложения

Заключение

В результате исследовательской работы над курсовой работой были получены следующие результаты:

1. Проведено сравнение таких инструментов для навигации, как нижняя навигация, всплывающее меню, вкладки, всплывающая навигация и контекстная навигация.
2. Проведено сравнение библиотек для навигации: FragmentManager, Cicerone, Decompose, Navigation Architecture Component.
3. На основе выбранной библиотеки, Navigation Architecture Component, создано приложение «Телефонный справочник».

Список использованных источников

1. Principles of navigation / URL: https://developer.android.com/guide/navigation/principles / (дата обращения 15.06.2024)
2. Основы эффективной мобильной навигации / URL: https://askusers.ru/blog/osnovy-ux/6-primerov-mobilnoy-navigatsii-i-8-tipov-o-kotorykh-vam-sleduet-znat/ /(дата обращения 14.06.2024)
3. Cicerone / URL: https://github.com/terrakok/Cicerone / (дата обращения 7.04.2024)
4. Decompose / URL: https://github.com/arkivanov/Decompose / (дата обращения 5.05.2024)
5. Fragment manager / URL: https://developer.android.com/guide/fragments/fragmentmanager
6. / (дата обращения 20.05.2024)
7. Navigation / URL: https://developer.android.com/guide/navigation / (дата обращения 29.05.2024)