ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки 09.04.02 «Информационные системы и технологии», профиль «Мобильные технологии»

КУРСОВОЙ ПРОЕКТ

Дисциплина: Разработка приложений для IOS

Выполнил: студент группы 244-332

Курносова А.В.

Дата, подпись __06.06.25_____ _____

(Дата) (Подпись)

Проверила: к.т.н., ст. преп. Алпатова М.В._____

(Оценка)

Дата, подпись _____ _____

(Дата) (Подпись)

Замечания:

_____

_____

Москва

2024

Оглавление

Введение

Корректная работа мобильного приложения складывается из множества факторов, но одним из ключевых показателей плавности и корректности работы мобильного приложения, является показатель FPS. На данный показатель может влиять как корректность работы программного кода приложения, так и его внешняя оболочка. Так или иначе, важным аспектом при работе мобильного приложения, влияющим на показатель FPS является стабильная, корректная обработка различных состояний мобильного приложения.

Помимо состояния стабильной работы, приложение может переходить в пустое состояние, когда информация для страницы отсутствует, состояние ошибки или состояние загрузки. Пустое состояние и состояние ошибки может быть напрямую обработано в коде приложения, при работе с определёнными данными. Последнее же состояние, состояние загрузки, представляется наиболее интересным для подробного рассмотрения и исследования, поскольку данное состояние может быть связано не только с работой определённых функций приложения, но и быть задействовано при стандартных функциях навигации приложения.

В данной работе исследуется состояние загрузки мобильного приложения и влияние данного состояния на стабильную и плавную работу приложения. Проведено исследование влияния внедрения состояния загрузки в приложение на плавность его работы. Проведена оценка по показателю FPS качества работы приложения при переводе его в состояние загрузки.

Ссылка на проект: https://disk.yandex.ru/d/KUCxy7YkkUjCSA

ГЛАВА 1. Аналитическая разработка

В первой главе проводится аналитический анализ вопроса введение приложения в состояние загрузки и влияние данного состояния на стабильную и плавную работу приложения.

Цель

Исследование влияние внедрения системы перехода в состояние загрузки (loading) в моб ильное приложение на базе IOS Flutter.

Задачи

Разработка приложения на базе IOS и исследование влияние состояния загрузки на работу приложения, может быть разбита на следующие задачи:

1. Подготовка среды разработки приложения на IOS;
2. Разработка приложения в программной среде;
3. Оценка качества работы приложения (показателя FPS) на начальном этапе работы приложения;
4. Внедрение в приложения системы введения в состояние загрузки при различных взаимодействиях с интерфейсом;
5. Оценка качественных изменений приложения, после внедрения системы перехода приложения в состояние загрузки;
6. Проведение сравнительного анализа, оценка полученного результата.

Предметная область

Основным показателем качества работы мобильного приложения является показатель FPS – количество кадров приложения в секунду. По данному показателю возможно определить, на сколько корректно и плавно работает приложение при тех или иных событиях.

Если FPS падает ниже 60 кадров в секунду – считается, что происходит просадка по количеству кадров и «заикание» приложения. Это говорит о том, что плавность работы приложения страдает и приложение нуждается в оптимизации.

Так же, немаловажным фактором при работе приложения, влияющим на показатель FPS является корректная обработка различных состояний приложения. Среди состояний работы приложения могут быть:

- «Пустое состояние» (empty) - когда отсутствует информация для вывода на странице приложения. При корректной работе, приложение должно отображать специальную страницу «пустого» состояния – страницу, оповещающую пользователя о корректной работе приложения, но отсутствии информации на странице;
- Состояние ошибки (error) – состояние, при котором происходит ошибка на том или ином этапе работы приложения (загрузки данных, получения данных со сторонних ресурсов, ошибка отображения и т.д.). При корректной работе мобильного приложения, состояние ошибки также обрабатывается, и пользователь получает оповещение о произошедшей ошибке. В случае, если ошибки не обрабатываются самим приложением – происходит остановка работы приложения и некорректное его закрытие;
- Состояние загрузки (loading) – состояние, при котором происходит обработка команды, получение данных, процесс отображения данных и т.д. На выполнение программного кода приложением для выполнения данных действий, требуется время. В случае, если состояние загрузки не предусмотрено, выполнение кода происходит в реальном времени, а пользователь не получает оповещений о происходящих процессах. В этом случае, кадры приложения могут обрабатываться слишком долго и может происходить заикание приложения.

Последнее состояние представляется наиболее интересным для исследования. Его внедрение может положительно влиять на плавность работы приложения, но при этом, данное состояние требует грамотного внедрения – в противном случае, будут затрачены избыточные ресурсы приложения на выполнение ненужных команд.

Состояние загрузки – это специальное состояние пользовательского интерфейса, которое отображается во время проведения асинхронных операций. Примером асинхронных операций может являться загрузка данных из сети или со сторонних файлов. Данное состояние оповещает пользователя о течении процесса и предотвращает появление ощущения «зависания».

Данное состояние так же может способствовать более корректному взаимодействию пользователя с приложением, так как возникновение ощущения «зависания» у пользователя, может приводить к некорректным, поспешным действиям со стороны пользователя по отношению к приложению. Это, в свою очередь, может привести к некорректной работе приложению, его реальному зависанию и не корректному прекращению работы приложения.

С технической точки зрения, состояние загрузки, обычно, реализуется с помощью управления состояниями виджетами. Так же, реализация может быть осуществлена при помощи использования менеджеров состояния. При изменении состояния приложения из состоянии загрузки, происходит автоматическое обновление UI и корректное отображение пользовательских элементов.

При теоретическом обзоре влияния внедрения состояния загрузки в приложение, можно выделить следующие, немаловажные аспекты:

- Улучшение UX через информирование пользователя

Состояние загрузки может быть внедрено при асинхронной загрузке данных в приложении. Наличие интерфейса, отображающего состояние «получение данных», позволяет оповестить пользователя о корректной работе приложения, отсутствии ошибок в работе приложения и побуждает к корректному взаимодействию с приложением. В случае, если пользователь не получает отклика от приложения о происходящем получении данных, появляется ощущение «зависания» - в действительности, приложение продолжает корректную работу, но ему требуется время на выполнение программного кода. Для пользователя процесс выполнения кода остаётся невидимым.

- Реактивное обновление UI

Поскольку в мобильном приложении, разработанном на базе IOS, с использованием Flutter, используется реактивное управление состоянием, внедрение состояния загрузки позволяет более корректно обрабатывать и перестраивать элементы UI. При изменении состояния загрузки, UI автоматически перестраивается, после чего происходит отображение страницы приложения в том или ином состоянии: при стандартном состоянии работы – отображение пользовательского интерфейса; при наличии ошибок или иных загрузок – отображение элементов интерфейса, отражающих данное состояние приложения.

- Поддержка асинхронных операций

Для отображения состояния загрузки, используются специальные виджеты и классы. Данные структуры позволяют обрабатывать асинхронные данные и изменять состояние приложения, исходя из результатов обработки. В противном случае, обработка асинхронных процессов происходит в реальном времени, вместе с отображением пользовательского интерфейса. Это приводит к выпадению кадров (снижению FPS) и «заиканиям приложения».

- Повышение стабильности и предсказуемости

Корректная обработка асинхронных данных, в том числе и состояний загрузки, позволяет снизить риск рассинхронизации данных и возникновения ошибок. В приложении могут быть как предусмотренные приложением ошибки (например, ошибки при загрузке данных из сети), так и непредусмотренные (в случае, если ошибка возникает на этапе, относящемуся к стандартному состоянию работы приложения). Ошибки, связанные с рассинхронизацией данных, являются непредусмотренными, так как данное состояние не должно быть достигнуто при корректной работе приложения. В случае возникновения не предусмотренных ошибок приложения, происходит некорректное завершение работы. Таким образом, внедрение состояния загрузки позволяет избежать фатальных ошибок приложения и сделать работу общей архитектуры более стабильной и предсказуемой.

Поскольку оценка качества работы приложения происходит на основании показателя FPS, вопрос влияния состояния загрузки на количество кадров в секунду рассмотрен отдельно. При рассмотрении данного вопроса, в основе лежит структура приложения на IOS, построенного при помощи Flutter.

При смене состояния приложения, происходит перерисовка компонентов UI. Если состояние загрузки отсутствует в приложении, перерисовка компонентов происходит в реальном времени относительно пользователя. Если состояние загрузки внедрено, Flutter вызывает перестроение виджетов, что требует ресурсов CPU и GPU. Если операция достаточно сложная, это может привести к снижению FPS. Если загрузка данных выполняется асинхронно и не блокирует главный поток, FPS снижаться не будет. Но если загрузка блокирует UI-поток, может происходить «зависание» и падение показателя уровня кадров.

В свою очередь, внедрение состояния загрузки, так же требует отрисовки пользовательского отображения состояния (графические изображения загрузки). Это, в свою очередь, увеличивает нагрузку на рендеринг, что может негативно сказаться на показателе FPS. Особенно сильное влияние на показатель FPS может оказать сложная анимация загрузки. Чем проще анимация пользовательского интерфейса загрузки – тем меньшее влияние на сложность рендеринга она оказывает.

Использование эффективных паттернов управления состоянием (например Provider, BLoC, Riverpod) помогает минимизировать лишние перестроение UI снижая нагрузку и позволяя сохранить стабильное число кадров.

Из этого следует, что при внедрении состояния загрузки, важно оценить необходимость внедрения и оценить сложность обработки состояния. При корректном внедрении специального состояния в приложение, возможно достижение стабильного показателя FPS и плавной работы приложения. В противном случае, возможно достижение обратного эффекта – увеличения нагрузки на CPU и GPU, усложнение рендеринга и проседания FPS.

В Flutter существует несколько методов добавления состояния загрузки в приложение:

- StatefulWidget

Этот способ является самым простым и завязан на добавлении переменной типа bool. Данный метод используется при наличии асинхронной загрузки. В таком случае, в качестве переменной типа bool, устанавливается переменная _isLoading. При начале загрузки, переменной присваивается значение true, а в пользовательском интерфейсе отображается иконка загрузки. При завершении процесса загрузки, значению переменной передаётся значение false, и обновляется UI через setState().

- Менеджеры состояний

Менеджеры состояний являются специализированными компонентами и применяются для более крупных приложений. Данные менеджеры применяются для обработки различных состояний приложения. Основой работы менеджеров является отделение логики приложения от UI и отдельная их обработка.

- Библиотеки и пакеты

Данные методы применимы для специфических задач, например, для создания загрузки приложения с баром загрузки. Для этих целей могут быть применены как индивидуальные пакеты, так и готовые решения. Примером готового решения может служить flutter_rustore_update. В данном пакете предоставлены встроенные индикаторы и инструменты для управления загрузкой и другими состояниями приложения.

При выборе инструмента создания загрузки, необходима оценка сложности приложения и функции, для которой применим перевод приложения в данное состояние. Использование излишне сложных инструментов для перевода приложения в состояние загрузки, может привести к ухудшению оптимизации приложения и снижению FPS.

Предметом исследования в данном случае является простое приложение с навигацией страниц через AppBar. В основ приложения лежит следующая иерархия:

- Главный файл (main)

Файл содержит общую оболочку приложения: общую систему навигации приложения (AppBar и меню «гамбургер»), общие стили приложения и логику переходов между экранами;

- Экраны приложения

Каждый экран приложения вынесен в отдельный файл. Таким образом, при добавлении новых страниц, изменении имеющихся страниц или удалении страниц из структуры приложения, изменение всей структуры не требуется. Это влияет на масштабируемость приложения, но требует отдельной обработки переходов между экранами;

- Служебные файлы приложения

К служебным файлам приложения относятся файлы, содержащие стили приложения (на которые ссылается файл main), стили текстов, изменяемые файлы, хранящие информацию, получаемую приложением. Данные файлы служат для более простого и быстрого изменения стиля приложения и его наполнения, без изменения общей структуры. Таким образом, сохраняется стабильная работа приложения, вне зависимости от его информационного наполнения.

Особое внимание для изучения внедрения системы загрузки заслуживают:

- Система поиска в приложении

Среди экранов приложения, добавлена страница, содержащая возможность поиска в сети Интернет. Страница содержит поле ввода темы поиска и кнопку отправки поискового запроса. После отправки поискового запроса, происходит поиск по сети Интернет сайтов, удовлетворяющей тем запроса. После осуществления поиска, происходит вывод найденных ссылок в виде списка. Пользователь может осуществить переход по ссылке и перейти на сайт.

В данной системе, при осуществлении поиска, происходит асинхронная обработка данных. При отсутствии системы перехода приложения в состояние загрузки, возможно зависание приложения или заметная потеря кадров при загрузке ссылок.

- Страница новостей

В приложении добавлена страница новостей, на которой происходит загрузка данных из отдельного файла формата JSON. При этом, так же, как и в случае с поисковой системой, происходит асинхронная загрузка данных. Аналогично, при отсутствии системы перехода приложения в состояние загрузки, возможна заметная потеря кадров.

- Система навигации

При рассмотрении вопроса внедрения системы загрузки в приложение, возможно исследование влияния перехода в состояние загрузки приложением, при переходе между различными экранами. Внедрение данной системы, предположительно, может иметь как положительный результат, так и отрицательный. Поскольку в данном случае, работа навигации завязана на едином файле навигации приложения, асинхронных процессов не происходит. Но при этом, переход приложения в состояние загрузки, может качественно повлиять на скорость и плавность отображения пользовательского интерфейса.

ГЛАВА 2. Технологическая разработка

Во второй главе рассматривается разработка приложения и исследование влияние внедрения системы перехода в состояние загрузки на качественные характеристики приложения, с точки зрения показателя FPS.

Выбор технических инструментов

Разработка приложения производится на базе IOS. Для разработки приложения применены инструменты:

- Android Studio – на базе приложения, задействованы эмуляторы мобильных устройств, для запуска и тестирования приложения. Эмуляторы подключены к программной среде разработки при помощи настройки JDK и SDK файлов, а так же изменения системных переменных на устройстве;
- Visual Studio Code – программная среда разработки. На базе приложения происходит написание кода проверка на наличие ошибок и билд;
- Flutter – фреймворк, позволяющий производить разработку мобильного приложения на базе IOS, при отсутствии программной среды IOS на устройстве, на котором производится разработка приложения.

Для корректной работы всех программ и настройки видимости, необходимо произвести настройку версий программ, настройку изменения среды переменных на устройстве, установить необходимые версии JDK и SDK файлов и проверить совместимость всех пакетов.

По итогу настройки среды, необходимо получить комплексную работу программ и фреймворка, при которой пакеты не вступают в конфликт, видимы друг для друга и позволяют в полной мере производить разработку и тестирование приложения.

Структура приложения

Для разработки приложения, была определена структура приложения, необходимый функционал и архитектура.

Исследование будет проведено на базе приложения «интернет-магазин», со свойственной, стандартной структурой подобных приложений, и дополнениями в виде страницы встроенного поиска по сети Интернет и новостной страницей.

С точки зрения структуры экранов, приложение содержит:

- Экран авторизации;
- Экран регистрации;
- Главный экран;
- Страницу каталога;
- Страницу корзины;
- Страницу поиска;
- Страницу новостей.

С точки зрения архитектуры приложения, данные страницы представляются отдельными элементами (файлами) в структуре приложения. Данное решение позволяет создавать масштабируемое приложение, внесение изменений в которое в меньшей степени влияет на работу всего приложения в целом – ели ошибка происходит на одной странице, остальные страницы могут продолжать функционировать безошибочно.

Так же, отдельным файлом создана общая оболочка для всего приложения. Данная оболочка позволяет применить ко всем страницам одинаковую логику навигации и общую структуру – AppBar, меню «гамбургер», систему выхода из приложения и перехода к странице авторизации. Данная система не только решает вопрос с общим стилем для всех страниц, но и так же позволяет применять оболочку к новым или изменяемым страницам, таким образом, снижая риск возникновения критических ошибок в приложении, полностью нарушающих работу всего приложения.

Исходя из определённой структуры приложения, возможно определение мест для дальнейшего исследования внедрения процессов перехода приложения в состояние загрузки:

Состояние загрузки может быть вызвано на страницах, где происходит асинхронная загрузка – страница новостей и страница поиска по сети;

Состояние загрузки так же может быть вызвано при переходе между страницами, несмотря на то что навигация страниц определена в главном файле и не является асинхронными процессами. При переходе между страницами происходит значительное изменение в графике и пользовательском интерфейсе, что приводит к необходимости обработки существенного объёма информации. Это, в свою очередь, может служить поводом для перехода приложения в состояние загрузки и обработки информации без параллельного вывода обработанной информации.

Разработка страниц приложения

При рассмотрении вопроса разработки приложения, отдельное внимание будет уделено разработке страниц, на основании которых, в дальнейшем, планируется исследование влияния перехода в состояние загрузки на показатель FPS.

Так, в первую очередь, необходимо рассмотреть структуру файла, хранящего в себе общую оболочку приложения. Основой для дальнейшего изменения послужит система переходов между страницами. Изначально система переходов не предусматривает перевод приложения в состояние загрузки, при совершении перехода между экранами приложения. Переход реализован напрямую (листинг 1):

Листинг 1 – пример перехода на экран приложения

```
ListTile(
            leading: Icon(Icons.search, color: Colors.white),
            title: Text('Поиск', style: TextStyle(color: Colors.white)),
            onTap: () {
              Navigator.pop(context);
              _selectPage(Pages.search);
            },
          ),
```

В данном случае, рассмотрен пример перехода на страницу поиска в приложении. При переходе, функция работает напрямую – происходит прямая переадресация на экран приложения, заданный как Page.search.

Сама навигация между экранами осуществляется через меню «гамбургер» приложения и также не предусматривает перехода в специальное состояние.

Для страницы поиска, задействованы дополнительные ресурсы, позволяющие осуществлять поиск по сети Интернет, на основе поисковой системы Google. Для этого задействован ресурс программируемой поисковой системы от Google, который позволяет создавать собственный поисковые системы на базе Google и внедрять их в структуру приложения. Для внедрения такой поисковой системы в приложение требуется получение API адреса созданного поисковика.

Поисковой запрос имеет следующую структуру (листинг 2):

Листинг 2 – Пример поискового запроса на странице поиска

```
final url =
        'https://www.googleapis.com/customsearch/v1?key=$apiKey&cx=$cx&q=${Uri.enco
deQueryComponent(query)}&num=4';
```

- key — API ключ Google.
- cx — идентификатор поисковой системы.
- q — поисковый запрос пользователя.
- num=4 — ограничение на 4 результата.

Далее в функции происходит отправка HTTP GET-запроса, по этому URL, с помощью пакета http.

Приложение осуществляет поиск по сети Интернет, после чего выводит на экран пользователя первые 4 ссылки по заданной теме. При исследовании вопроса влияния перехода в состояние загрузки, количество ссылок, выводимых на экран для пользователя, может быть увеличено, для повышения наглядности полученного результата. В случае, когда в приложении не требуется обработка большого количество данных и отрисовка графических элементов, переход в состояние загрузки может быть не на столько значительным. В данном же случае, вывод большего числа ссылок может быть закономерным и логичным, поскольку для увеличения точности найденной информации и расширения её спектра,

необходим больший вывод найденных ресурсов. Этот функционал может быть полезен, поэтому его внедрение не будет считаться инородным для приложения.

С точки зрения отображения ссылок, вывод происходит при помощи следующей функции (листинг 3):

Листинг 3 – Вывод найденных ссылок на экран

```
Future<void> _openLink(String url) async {
    final uri = Uri.parse(url);
    if (await canLaunchUrl(uri)) {
      await launchUrl(uri);
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Не удалось открыть ссылку')),
      );
    }
  }
```

Функция содержит проверку на возможность открытия ссылки. Если ссылка не может быть открыта – выводится оповещение об ошибке. Если ссылка рабочая – происходит открытие ссылки и переход на веб-ресурс. При наличии на устройстве нескольких способов открытия веб-ресурса, пользователю предлагается выбор браузера, в котором будет открыта ссылка.

С точки зрения внедрения состояния загрузки, оно может быть предусмотрено на этапе, когда приложением осуществляется поиск ссылок по теме запроса пользователя и отрисовка графических элементов для возможности перехода по данным ссылкам. Несмотря на несложный способ отображения ссылок и отсутствие сложных графических элементов при отрисовке интерфейса, увеличение числа выводимых ссылок, может негативно сказаться на времени обработки информации и привески к «зависаниям» приложения.

Для работы новостной страницы, был создан отдельный файл формата JSON.

Для передачи данных из файла JSON в приложение, необходимо создать структуру передачи данных. Структура создана в отдельном файле, в разделе assets/models/news_item. В файле хранится разметка для передачи данных (листинг 4).

Листинг 4 – Код разметки страницы для вывода данных файла JSON

```
class NewsItem {
  final int id;
  final String title;
  final String date;
  final String image;
  final String content;

  NewsItem({
    required this.id,
    required this.title,
    required this.date,
    required this.image,
    required this.content,
  });

  factory NewsItem.fromJson(Map<String, dynamic> json) {
    return NewsItem(
      id: json['id'],
      title: json['title'],
      date: json['date'],
      image: json['image'],
      content: json['content'],
    );
  }
}
```

В новости содержится идентификационный номер для данной новости, после чего указываются заголовок, дата, URL изображения и краткое описание новости.

Файл JSON размещён в папке assets/files приложения.

Далее, необходимо осуществить передачу данных в приложение. Для этого была подготовлена страница news_page, на которой располагается вся передающаяся информация из JSON-фала. Так же, был настроен переход на данную страницу через меню страницы main. Для этого страница была добавлена в общий список страниц и для неё был указан корректный переход.

Для тела страницы news_page необходимо определить функцию для загрузки данных файла JSON (листинг 5).

Листинг 5 – Функция загрузки данный из файла

```
  final jsonString = await rootBundle.loadString('assets/files/news.json');
```

- Метод rootBundle.loadString асинхронно загружает содержимое файла news.json, который находится в папке assets вашего проекта.
- Файл должен быть предварительно зарегистрирован в pubspec.yaml в разделе assets, чтобы Flutter мог его найти. Для этого необходимо зарегистрировать его в качестве ассета и определить при помощи функции в терминале *flutter pub get.*

Далее, необходимо парсировать данные из файла в JSON в Dart. Для этого используется следующий метод (листинг 6):

Листинг 6 – Парсинг данных из файла

```
final List<dynamic> jsonResponse = json.decode(jsonString);
  return jsonResponse.map((item) => NewsItem.fromJson(item)).toList();
```

- Функция json.decode из пакета dart:convert преобразует JSON-строку в структуру Dart — в данном случае, в список динамических объектов (List<dynamic>);
- Каждый элемент списка — это Map с ключами и значениями, соответствующими полям новости;
- Метод map преобразует каждый Map в объект модели NewsItem через фабричный конструктор fromJson, что упрощает работу с данными в коде

Далее, при помощи разметки (news_item), происходит асинхронное получение и отображение данных на странице приложения при помощи FutureBuilder.

Таким образом, для отображения новостей, в приложении используется течение асинхронных процессов. С точки зрения внедрения системы загрузки, наиболее рациональным является приведение приложения в состояние загрузки при осуществлении парсинга данных из JSON-файла в приложение.

Внедрение состояния загрузки

Перед внедрением перехода приложения в состояние загрузки, необходимо зафиксировать получаемые результаты по показателю FPS при работе приложения

без данной системы. Для этого начальный код был сохранён и, в дальнейшем, запущен для проведения сравнительного анализа.

При внедрении перехода в состояние загрузки, необходимо оценить системы, применимые для перехода в данное состояние.

В случае, когда вопрос касается навигации между экранами приложения, использование сложных систем перехода в специальное состояние не является оправданным. Поскольку приложение не является сложным, а его графические элементы являются стандартными для структуры мобильного приложения, предпочтительным будет использование стандартных способов перехода в состояние «loading».

Для внедрения системы загрузки в общую оболочку приложения, добавлена переменная типа bool, отображающая переход приложения в состояние загрузки. Переменная вынесена в отдельную функцию загрузки, которая срабатывает при любом переходе между экранами приложения (листинг 7).

Листинг 7 – Функция перехода в состояние загрузки

```
Future<void> _selectPage(Pages page) async {
    setState(() {
      _isLoading = true;
    });

    // Имитация задержки загрузки
    await Future.delayed(Duration(seconds: 1));

    setState(() {
      _selectedPage = page;
      _isLoading = false;
    });
  }
```

При осуществлении каждого перехода между экранами приложения, происходит вызов функции приведения приложения в состояние загрузки. При этом, значение переменной типа bool изменяется на true и приложение переходит к этапу загрузки данных. С точки зрения пользователя, данное состояние отражается как бар загрузки, отображаемый по центру экрана при осуществлении подгрузки данных нового экрана (рис.1).

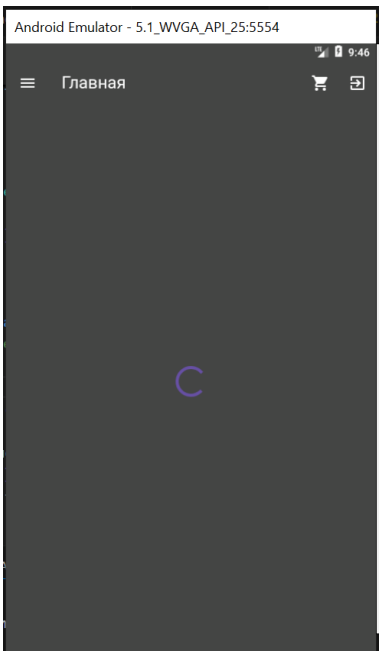


Рис.1 – Отображение бара загрузки при переходе между экранами

Для страницы новостей и страницы поиска, внедрение системы перехода в состояние загрузки имеет тот же принцип. Для данных страниц выбраны простые системы загрузки, так как в данном случае, переход в специальное состояние требуется вследствие течения асинхронных процессов. При поиске данных в сети Интернет и при загрузке новостной страницы с использованием стороннего файла формата JSON, используются близкие по структуре процессы отрисовки пользовательского интерфейса.

Для страницы поиска, система перехода в состояние загрузки, была внедрена на этапе получения и вывода полученных ссылок в пользовательском интерфейсе (листинг 8).

Листинг 8 – Добавление состояния загрузки на странице поиска

```
child:
                  _isLoading
                      ? const CircularProgressIndicator(
                        color: Color.fromARGB(255, 144, 144, 144),
                      )
                      : const Text(
                        'Поиск',
                        style: TextStyle(color: Colors.white),
                      ),
```

При переходе в данное состояние, аналогично с переходом в состояние загрузки при переходе между экранами, для пользователя отображается бар загрузки. Для пользовательского интерфейса задействован наиболее простой, встроенный бар загрузки, для избежания дополнительной нагрузки на обработку графики приложения (рис.2).

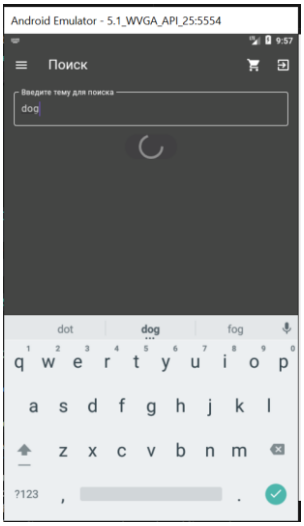


Рис.2 – Отображение бара загрузки при поиске ссылок

Аналогичные инструменты задействованы и для страницы новостей. В случае с новостной страницей, переход в состояние загрузки добавлен при осуществлении парсинга данных из JSON-файла в приложение.

В отличие от поиска по сети Интернет, когда состояние осуществление поиска является более ожидаемым со стороны пользователя, состояние получения данных со стороннего файла для новостной страницы является менее ожидаемым. В данном случае, оповещение пользователя о течении загрузки, является наиболее выраженным. В данном случае, оповещение пользователя о переходе приложения в специальное состояние, в более значительной степени снижает риск некорректного взаимодействия пользователя с элементами приложения (листинг 9).

Листинг 9 – Состояние загрузки на странице новостей

```
return Center(child: CircularProgressIndicator());
```

Таким образом, переходы в состояние загрузки были внедрены во все этапы работы приложения, где данные состояние могло бы быть необходимо. Далее, для оценки качественных изменений, необходимо проведение сравнительного анализа на основании показателя FPS.

Сравнительный анализ на основании показателя FPS

Для оценки показателя FPS при работе приложения до и после внедрения системы перехода в состояние загрузки, использован инструмент performance overlay. Данный инструмент позволяет выводить на экран два графика:

верхний — время работы GPU (растеризация),

нижний — время работы UI-потока.

Исходя из показателей графиков, возможна оценка FPS приложения. Зеленые и красные вертикальные полосы показывают, укладывается ли приложение в 16 мс на кадр (60 FPS). Если график выходит за пределы 16 мс, значит FPS падает, и появляется красная полоса — индикатор джиттеров (заиканий).

Для обоих вариантов приложения, было настроено отображение данного инструмента. Для корректной работы с инструментом, были отключены другие отображения performance overlay в программном коде самого приложения. Код запущен в режиме отладки (flutter run --debug), после чего было включено отображение инструмента.

Инструмент отображается поверх экрана приложения и позволяет отслеживать показатель FPS на каждом этапе работы и взаимодействия с приложением.

Первоначально было исследовано приложение без внедренной системы перехода в специальное состояние загрузки. Были зафиксированы показатели FPS приложения при переходе между страницами, осуществления поиска на странице поиска и загрузке новостной страницы. Были получены следующие результаты (рис. 3, 4, 5):

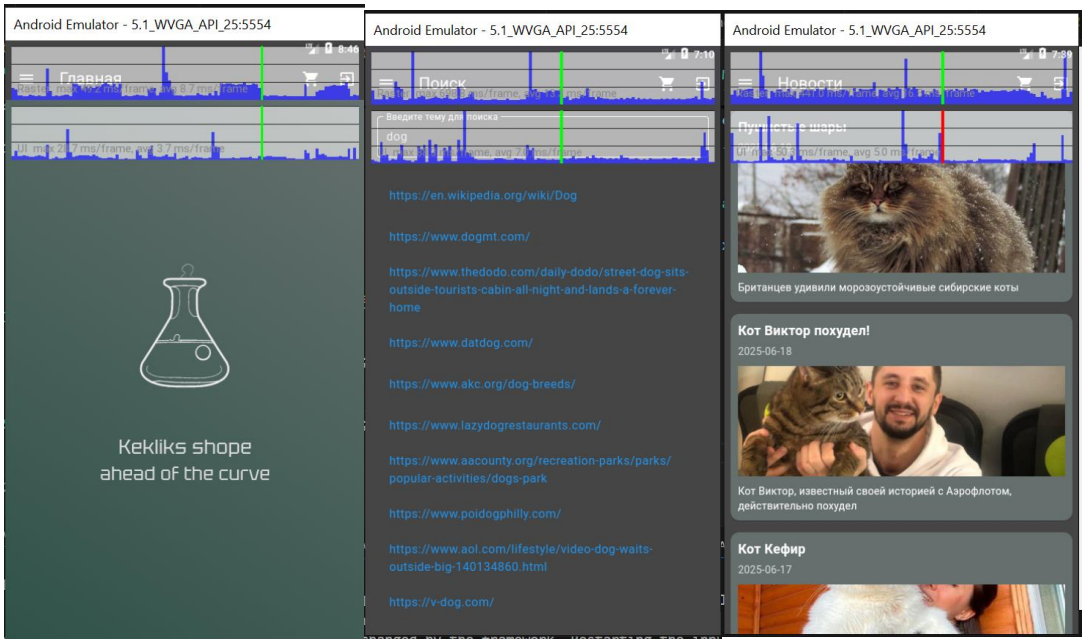


Рис.3 – Значения графиков «до» при переходе между страницами

Рис.4 – Значение графиков «до» при осуществлении поиска в сети Интернет

Рис.5 – Значения графиков «до» при осуществлении парсинга данных JSON

При анализе полученных данных, важно обратить внимание на возникновение зелёных и красных полос на графиках, а также на высоту столбцов. Во всех трёх случая, возникает потеря кадров и снижение FPS до значения, ниже 60 кадров в секунду. Данное значение не является удовлетворительным. Особенно заметно отсутствие качественной оптимизации будет при масштабировании подобного приложения.

Наиболее плохие результаты получены на странице новостей, при парсинге данных с файла JSON. На данной странице, проседание кадров происходит наиболее ощутимо.

Далее, необходимо произвести тестирование приложения, при внедрении перехода приложения в состояние загрузки. Для этого произведён запуск кода с внедрённой системой, в том же режиме, с аналогичным отображением графиков. При этом получены следующие результаты (рис. 6, 7, 8):

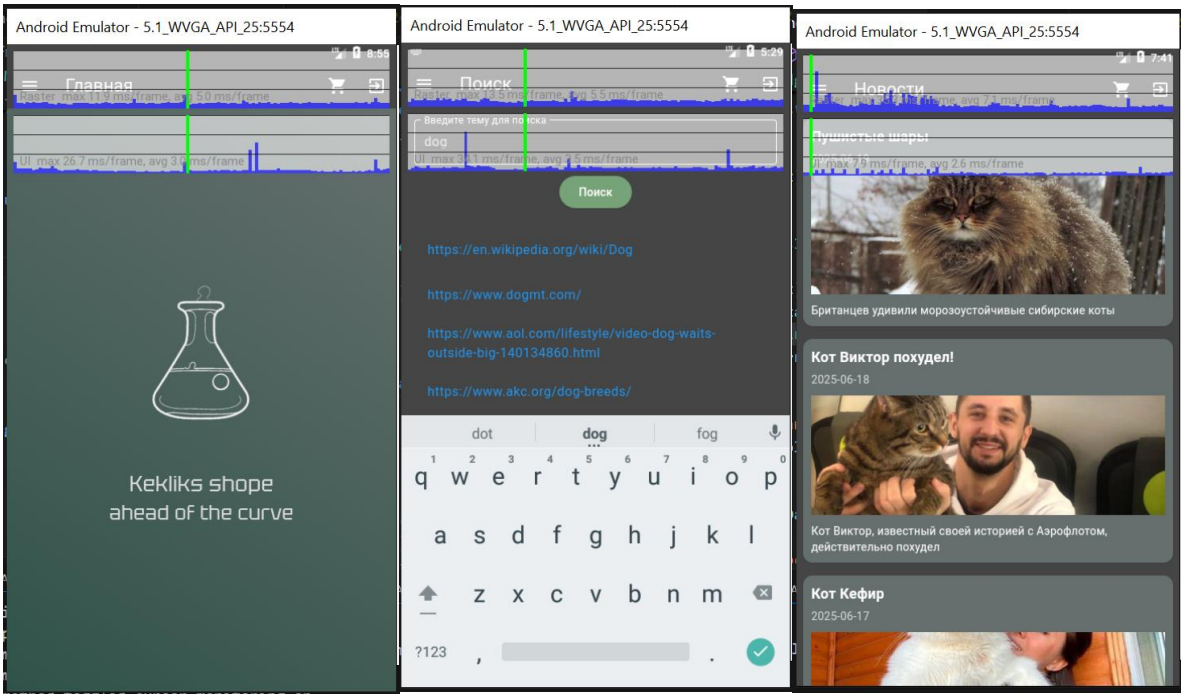


Рис.6 – Значения графиков «после» при переходе между страницами

Рис.7 – Значение графиков «после» при осуществлении поиска в сети Интернет

Рис.8 – Значения графиков «после» при осуществлении парсинга данных JSON

При анализе полученных данных, можно заметить, что при внедрении системы загрузки, заметно снижены показатели FPS во всех трёх случаях.

В случае осуществления переходов между страницами приложения, FPS стабильно сохраняет высокие значения и просадок не возникает, даже в случае многократного повторения перехода между страницами. В случае осуществления поиска информации в сети Интернет, наименьший показатель FPS получен при непосредственном отображении данных на экране пользователя. При этом, показатель FPS так же сохраняется в рамках допустимых значений. В случае парсинга данных на странице новостей, наименьшее значение FPS получено при непосредственном отображении данных. Так же, более низкие показатели FPS получены и при процессе парсинга. В этот момент, на экране пользователя происходит графическое отображение процесса загрузки, поэтому более низнике показатели кадров не влияют на качественные характеристики отображаемой графической оболочки. Несмотря на то, что фактические показатели кадров

достаточно низкие, данный факт практически не сказывается на восприятии пользовательского интерфейса приложения.

Анализ полученных данных

Для анализа полученных данных, были отдельно рассмотрены полученные графики и составлена сравнительная таблица (табл.1).

Таблица1 – Полученные значения FPS

	переходы	поиск	новости
loading_off GPU	49,2	69,8	441
loading_off UI	28,7	38,3	50,3
loading_on GPU	11,9	13,5	35,3
loading_on UI	26,7	34,1	7,9

Из таблицы заметно, что во всех случаях применения на страницах методов перехода приложения в специальное состояние загрузки, полученные значения ниже, а FPS – выше и стабильнее. Это говорит о положительном влиянии использования перехода в специальное состояние приложением, при необходимости обработки асинхронных процессов обработки данных или при значительных изменениях графических элементов на экране приложения.

Для большей наглядности, на основании полученных данных, так же построены графики (график 1, 2, 3).

График 1 – Изменение FPS при переходах между страницами

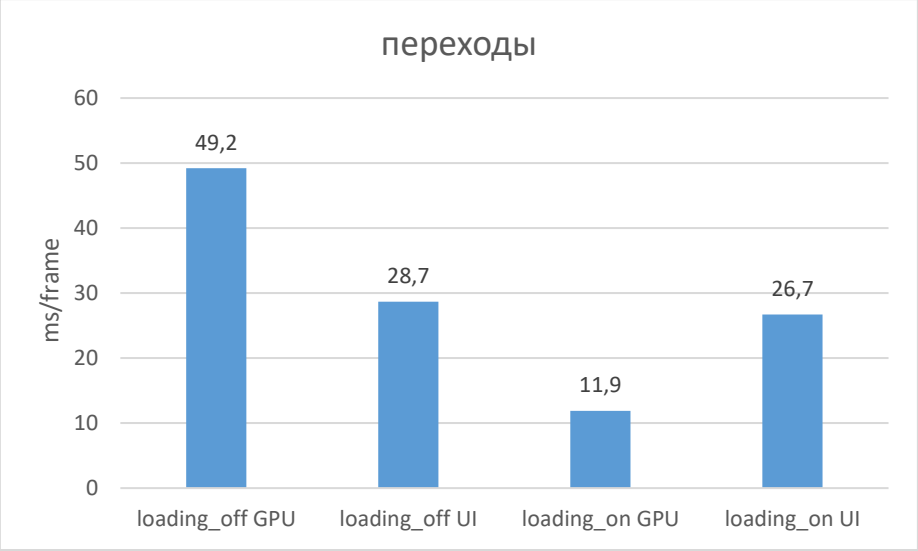


График 2 – Изменение FPS при поиске по сети Интернет

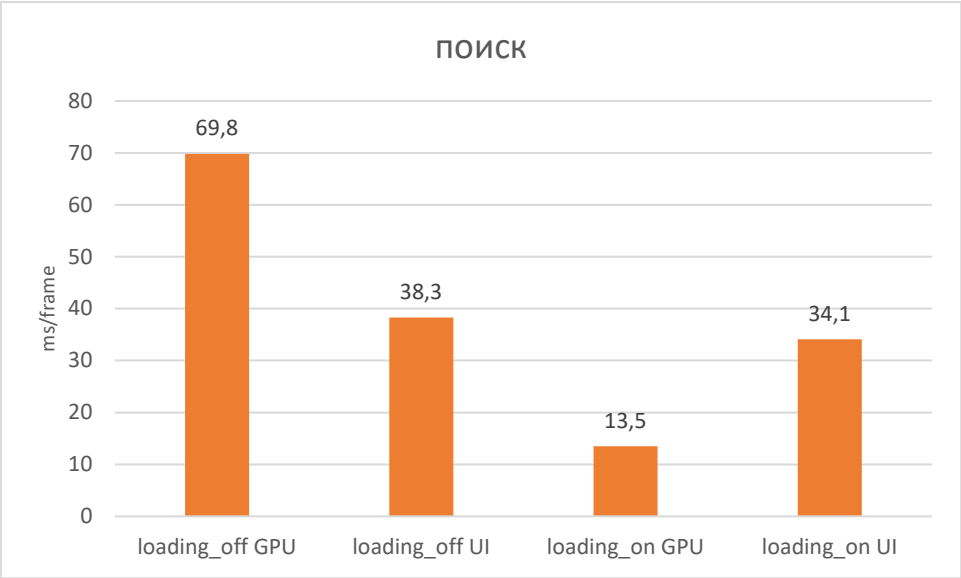
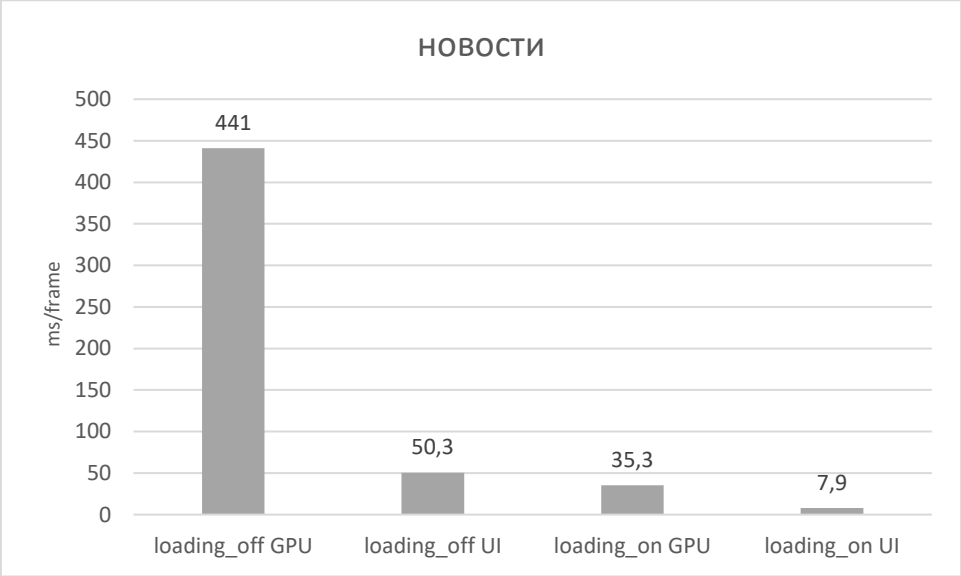


График 2 – Изменение FPS при асинхронном парсинге



Наиболее заметные результаты изменений по показателю FPS были получены при добавлении перехода приложения в состояние загрузки на странице новостей. Из этого может быть получен вывод о высокой степени необходимости добавления системы перехода в состояние «loading» при наличии в приложении объёмных асинхронных процессов.

Так же, важно заметить, что во всех случаях внедрения системы загрузки, большее значение данная система оказывает на показатель нагрузки на GPU.

Значительное изменение в лучшую сторону данного показателя, может говорить об оптимизации процессов рендеринга. Шейдеры, отрисовка которых происходила в режиме реального времени (с точки зрения пользователя), теперь компилируются и отрисовываются заранее. Это минимизирует накладную нагрузку на GPU и снижает его затраты на обработку графических операций.

Также, при добавлении системы загрузки в приложение, может происходить перераспределение ресурсов – часть нагрузки с GPU может быть перераспределена на CPU, что оптимизирует процесс обработки информации. При этом, важно заметить, что распределение нагрузки положительно влияет на общую производительность и плавность работы приложения.

Исходя из полученных результатов, можно говорить о положительном влиянии внедрения системы загрузки в приложение в тех случаях, когда требуется обработка большого количества графических данных или выполнение асинхронных процессов.

Заключение

В ходе работы выполнены задачи по разработке приложения на базе IOS с использованием фреймворка Flutter и проведено исследование влияния внедрения специального состояния загрузки.

В ходе разработки приложения, создано приложение, содержащее главную страницу, страницу каталога и корзины, страницы авторизации и регистрации, а также страницу поиска по сети Интернет и страницу новостей. Для всех страниц приложения создана единая оболочка, реализующая общую систему навигации в приложении.

На основании разработанного приложения, проведено исследование влияния внедрения системы перехода приложения в специальное состояние «loading». Данная внедрена и исследована на базе страниц поиска по сети Интернет и новостной страницы приложения. Так же, рассмотрено влияние внедрения данной системы на осуществление переходов между экранами приложения.

Исследование влияния системы загрузки в приложении проведено на основании показателя FPS и оценки распределения нагрузки на GPU и CPU при работе приложения.

В результате исследования получены результаты, указывающие на положительное влияние внедрения системы загрузки и перехода приложения в состояние загрузки, при осуществлении в приложении задач, связанных с сложной обработкой графической информации или течении асинхронных процессов.

При исследовании зафиксирован более стабильный и высокий показатель FPS при наличии в приложении системы перехода в состояние загрузки при осуществлении задач, связанных с обработкой графической информации или асинхронными процессами. Так же, зафиксировано снижение нагрузки на GPU при наличии системы перехода приложения в состояние «loading» и осуществлении обработки графической информации в скомпилированном виде. Отмечено

положительное влияние предварительной компоновки и отрисовки графических элементов приложения.

Сделаны выводы о возможной области применения перевода приложения в состояние загрузки и о качественном влиянии на стабильность и плавность работы приложения. Выводы основаны на анализе полученных при проведении исследования данных и построенных графиках, отражающих характер и степень влияния внедрения состояния «loading» в систему работы приложения.

Список литературы

1. Гамзаев, А. Игры со временем: ускоряем приложение на уровне восприятия. Habr. URL: https://habr.com/ru/companies/sravni/articles/732594/
2. Гиниятуллина, Э. И. Оценка качества мобильного программного обеспечения. CyberLeninka. 2021. URL: https://cyberleninka.ru/article/n/otsenka-kachestva-mobilnogo-programmnogo-obespecheniya
3. Набиев, Н. Ускорьте работу своего мобильного приложения: практические советы. Lebara Blog. 2024. URL: https://blog.lebara.com/ru/how-to-speed-up-your-mobile-app-practical-tips/
4. Савчук, В. С. Оптимизация производительности приложений Android. AppMaster. 2024. URL: https://appmaster.io/ru/blog/optimizatsiya-proizvoditelnosti-android-prilozhenii
5. Сидоренко, А. Оптимизация производительности мобильных приложений: стратегии и лучшие практики. CyberLeninka. 2023. URL: https://cyberleninka.ru/article/n/optimizatsiya-proizvoditelnosti-mobilnyh-prilozheniy-strategii-i-luchshie-praktiki
6. Фролов, С. А., Ермакова, Т. А. Разработка мобильного приложения на Android Studio. Учебный материал. 2021. URL: https://infourok.ru/razrabotka-mobilnogo-prilozheniya-na-android-studio-4876798.html
7. Щербаков, И. Риски мобильных приложений: безопасность и архитектура. Safe-Surf. 2023. URL: https://safe-surf.ru/articles/mobilnye-prilozheniya-bezopasnost-i-arhitektura-riski-razrabotka-ispolzovanie.html
8. Якушев, В. 9 главных способов продвижения мобильных приложений в 2025 году. AppBooster. 2024. URL: https://appbooster.com/blog/kak-prodvigat-mobilnye-prilozheniya/