

высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ
НОВОСТНОГО ПОРТАЛА**

Студент Кири 28.05.24 Кирюткин Илья Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель А.Н. 28.05.24 Токарев А.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр А.С. 28.05.24 А.С. Бычков
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.24 г. № 11)

Заведующий кафедрой М.А. 28.05.24 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 04.06.24 № 5)

Секретарь ГЭК Е.И. 28.05.24 Е.И. Гурин
(подпись) (фамилия, инициалы)

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«18» 04 2024 г.

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Кириюткин Илья Алексеевич гр. 20ВВП1 факультета вычислительной техники
2. Тема работы Мобильное приложение для новостного портала

Тема утверждена приказом ПГУ № 405/о от "19" апреля 2024 г.

3. Руководитель бакалаврской работы Токарев А.Н., к.т.н., доцент кафедры ВТ

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать и отладить мобильное приложение, которое позволяет просматривать искать и сохранять новости

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ актуальности проекта, потребностей целевых пользователей, разрабатываемого приложения, аналогов, современных мобильных операционных систем, инструментов мобильной разработки (фреймворков, сред программирования).

1.	Анализ требований	8	20.04.2024	
2.	Анализ актуальности проекта	8	21.04.2024	
3.	Анализ потребностей пользователей	16	22.04.2024- 23.04.2024	
4.	Анализ аналогов	16	24.04.2024- 25.04.2024	
5.	Анализ современных мобильных операционных систем	16	26.04.2024- 27.04.2024	
6.	Выбор инструментов разработки	16	28.04.2024- 29.04.2024	
7.	Разработка структуры приложения	40	30.04.2024- 5.05.2024	
8.	Разработка мобильного приложения	116	5.05.2024- 20.05.2024	
9.	Тестирование работоспособности приложения	16	21.05.2024 - 22.05.2024	
10.	Оформление пояснительной записки	72	23.05.2024- 31.05.2024	

Дата выдачи задания "19" апреля 2024г.

Руководитель бакалаврской работы



19.04.24

(подпись, дата)

Токарев А.Н.

Задание к исполнению принял студент



19.04.24

(подпись, дата)

Кирюткин И.А.

Работу к защите допустить
Декан факультета



05.06.2024

(подпись, дата)

Фионова Л.Р.

Объектом разработки является мобильное приложение для новостного портала.

Цель работы – разработать мобильное приложение для просмотра новостей. Приложение должно обладать следующим функционалом: пользователь приложения должен иметь возможность просматривать список новостей, сохранять новости, осуществлять поиск по ключевым словам, датам публикации новости.

В процессе работы проводился анализ актуальности проекта, потребностей целевых пользователей, аналогов, мобильных операционных систем, способов разработки мобильного ПО.

В результате проведенной работы было разработано кроссплатформенное мобильное приложение для новостного портала.

Содержание

Перечень сокращений и обозначений.....	4
Введение.....	5
1 Мобильное приложение для новостного портала.....	6
1.1 Актуальность проекта.....	6
1.2 Анализ потребностей пользователей	7
1.3 Анализ аналогов	8
1.4 Анализ современных операционных систем	11
1.5 Постановка задачи	12
1.6 Выбор инструментов разработки	15
1.7 Структура приложения.....	19
1.8 Реализация	24
1.9 Тестирование	46
2 Охрана труда.....	50
Заключение	52
Список использованных источников	54
Приложение А – UML диаграммы	55
Приложение А.1 – Диаграмма прецедентов.....	56
Приложение А.2 – Диаграмма классов	57
Приложение А.3 – Диаграмма деятельности	57
Приложение А.4 – Диаграмма пакетов	58
Приложение Б – Листинг программы	58
Приложение В – Презентация.....	160

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ВЦИОМ – всероссийский центр изучения общественного мнения

ПК – персональный компьютер

СМИ – средства массовой информации

ОС – операционная система

API (ApplicationProgrammingInterface) – программный интерфейс взаимодействия с программой

BLoC (BuisnessLogicComponent) – архитектура представления

GPU (GraphicProcessingUnit) – графический процессор

HTTP (Hyper Text Transfer Protocol). — протокол передачи гипертекста

JSON (JavaScriptObjectNotation) – формат текстовых данных, который используется для обмена данными в современных веб – и мобильных приложениях

SDK (SoftwareDevelopmentKit.) – набор средств для разработки программного обеспечения под определенную платформу

SQL (StructuredQueryLanguage) – язык структурированных запросов

UI (User Interface) — пользовательский интерфейс

UML (UnifiedModelingLanguage) – унифицированный язык моделирования

URL (UniformResourceLocator) – единообразный локатор ресурса

Введение

В современном мире трудно найти человека, который не пользуется смартфоном. Это неудивительно, ведь современные технологии предоставляют массу возможностей. Так, в свою очередь, множество функций которые необходимо было выполнять на разных устройствах, объединены в одно устройство. Просмотр фильмов, прослушивание музыки, создание видео и фото – все это теперь является неотъемлемой частью жизни любого человека.

С каждым годом возможности смартфона увеличиваются. В рамках данной работы необходимо сделать мобильное приложение, предназначенное для просмотра новостей. Для этого нам потребуется:

- проанализировать актуальность проекта, потребности пользователей;
- произвести анализ аналогов;
- изучить современные мобильные операционные системы и инструменты разработки, чтобы выбрать наиболее подходящие для данного проекта;
- разработать структуру приложения, определив основные функциональные блоки и интерфейс;
- создать кодовую базу приложения, используя современные инструменты и библиотеки.

Приложение должно обладать удобным интерфейсом и широким функционалом, позволяя пользователям быстро и удобно получать актуальные новости прямо на своих мобильных устройствах.

1 Мобильное приложение для новостного портала

1.1 Актуальность проекта

В эпоху технологий люди имеют возможность собирать информацию с любой точки мира. Интернет, социальные сети, мобильные приложения – это источники которые упрощают жизнь человека и позволяют быть в курсе любых событий, не выходя из дома. Новости имеют огромное влияние на общество в современном мире. На данном этапе это не только источник информации, но и одно из средств общения и выражения мнения. Современный человек с помощью новостей может более подробно и целостей понимать, что именно ждет его в будущем.

Людям нужны приложения на смартфонах для просмотра новостей по нескольким причинам. Во–первых, смартфоны являются олицетворением нашего времени – невозможно представить современного человека без собственного смартфона. Это особенно ценно в сегодняшней стремительной жизни, где скорость реакции на события может иметь решающее значение. Во–вторых, приложения для смартфонов уделяют особое внимание адаптации интерфейса к мобильному устройству, обеспечивая максимальное удобство и эффективность использования. Они предлагают интуитивно понятные элементы управления и обтекаемый дизайн, делая просмотр новостей более приятным и интуитивно понятным. Кроме того, во многих случаях контент может быть адаптирован к интересам пользователя путем выбора соответствующих тем и источников новостей, что делает информацию более персонализированной и ценной. Это позволит пользователям быстро и эффективно получать информацию, оставаться в курсе событий. При этом процесс получения новостей станет более удобным и современным. По данным ВЦИОМ 77% россиян заходят на сайты онлайн–СМИ, 44% процентов из которых просматривают новости на смартфоне.

Исходя из этого, отдельное приложения для просмотра будет актуальным в наше время.

Мобильное приложение для новостного портала станет незаменимы инструментом для тех, кто постоянно интересуется происходящем в мире. Благодаря простому интерфейсу, мгновенным обновлениям и количеству информации, приложение сможет удовлетворить все необходимые потребности пользователей.

1.2 Анализ потребностей пользователей

В современном мире, иметь быстрый доступ к любой нужной информации является ключевым фактором. Но несмотря на большие возможности, не всегда удастся оперативно получать актуальные новости из-за ограниченного доступа. Именно эта проблема будет решена с помощью Мобильного приложения для новостного портала.

Одним из важнейших факторов является простой и удобный интерфейс. Зачастую, пользователи которые не разобрались как пользоваться приложением, удаляют его и ищут более удобное и понятное в использовании. Учитывая все факторы работы со смартфоном, а именно: небольшой экран в сравнении с другими устройствами, как компьютер или планшет. Был о создано приложение, которое позволяет эффективно использовать

Все эти факторы в совокупности определяют потребности пользователей в удобном поиске и просмотре новостей в мобильных приложениях.

Представление новостей в формате ленты, где каждая новость занимает минимальное количество места на экране. Это позволяет пользователям быстро просматривать множество заголовков и выбирать интересующие их статьи.

При нажатии на новость, пользователь сможет перейти на страницу с более полной информацией где будет представлен полный текст статьи, дополнительные материалы, такие как изображение или ссылки на связанные статьи.

Удобные элементы взаимодействия позволяют пользователю быстро переходить к просмотру полной статьи и следующей публикации при этом не совершая лишних усилий. Интуитивно понятный и эффективный механизм поиска, который позволит найти желанную статью по ключевым словам, категориям или датам публикации.

1.3 Анализ аналогов

В магазине приложений Googleplay и AppStore популярными приложения для просмотра новостей являются «Feedly» и «Inoreader». У этих приложений высокий рейтинг и больше 1 млн скачиваний что на ряду выше в сравнении с другими похожими приложениями.

Приложение «Feedly», имеет интуитивно понятный интерфейс и на Googleplay имеет свыше 5 млн. скачиваний и свыше 310 тыс. отзывов. Средняя оценка в этом магазине 3.5. Снимок экрана данного приложения представлен на рисунке 1. В этом приложении пользователи могут настроить свой список источников, и получать новости только из этих источников. Так же есть возможность сохранить новости на устройство, пользователь может в любое время без возможности выхода в интернет открыть сохраненную новость.

Пользователи могут сохранять статьи для чтения позже и просматривать их в автономном режиме, что удобно в случае отсутствия доступа к сети. «Feedly» интегрируется с различными сервисами и социальными сетями, что позволяет пользователям легко делиться интересным контентом с друзьями и коллегами.

Но у «Feedly» также есть некоторые недостатки. Некоторые функции доступны только в платной версии приложения, например, функции расширенного поиска. В бесплатной версии присутствуют рекламные материалы, что может раздражать пользователей. В большинстве отрицательных отзывов говорится о чрезмерном наличии рекламы, что в свою очередь мешает просмотру контента. Несмотря на возможность настройки списка источников, некоторым пользователям не совсем удобны инструменты поиска.

Кроме того, для пользователей из России есть ограничения доступа к некоторым функциям.

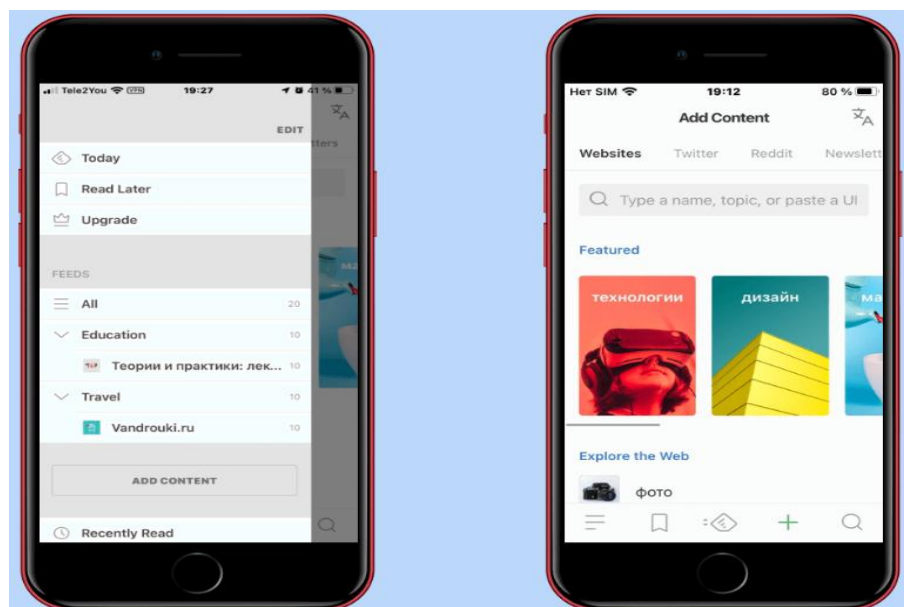


Рисунок 1 – приложение «Feedly»

Приложение «Inoreader» также является приложением для чтения новостей и управления контентом, аналогичным «Feedly». На Google play «Inoreader» имеет больше 500 тыс. скачиваний, свыше 9 тыс. отзывов и среднюю оценку 3.6. Снимок экрана данного приложения представлен на рисунке 2. Его достоинства и недостатки таковы; широкие возможности бесплатной версии «Inoreader» предоставляет значительное количество функций, пользователи могут настроить свой поток контента до самых

мелких деталей, что позволяет получать только ту информацию, которая действительно интересна, мощные инструменты фильтрации и поиска. Приложение предоставляет разнообразные инструменты для фильтрации и поиска контента. Но так же у этого приложения есть и недостатки. Для понимания интерфейса и многочисленных функций приложения может потребоваться некоторое время. Некоторые расширенные функции доступны только в платных тарифных планах, такие как например перевод текста, или сохранение статей.

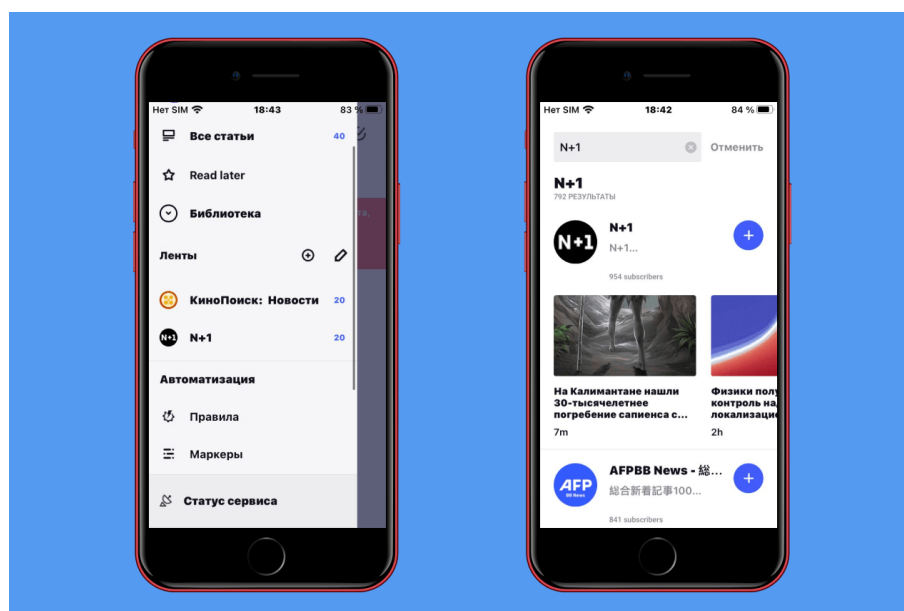


Рисунок 2— приложение «Inoreader»

После внимательного анализа преимуществ и недостатков двух приложений, можно прийти к выводу, что современному пользователю наиболее важным является простота использования и доступность функционала. Пользователи ценят удобство и быстроту получения нужной информации, поэтому идеальное приложение для просмотра новостей должно быть интуитивно понятным и легким в навигации.

Конечно, присутствие рекламы и платных дополнительных функций может раздражать и отвлекать пользователей от основной цели — чтения новостей. Поэтому важно, чтобы приложение предлагало базовый

функционал бесплатно и не навязывало дополнительные услуги за дополнительную плату.

Особенно актуально это для жителей России, у которых могут возникнуть проблемы с доступом к определенным функциям из-за географических ограничений или языковых барьеров. Поэтому важно, чтобы приложение было доступно без ограничений и предоставляло полный доступ к своим функциям для пользователей из всех стран, включая Россию.

Итак, идеальное приложение для просмотра новостей должно быть простым в использовании, лишенным рекламы и платных дополнительных функций, а также доступным без ограничений для всех пользователей, включая жителей России.

1.4 Анализ современных операционных систем

В современном мире мобильные приложения работают в основном на Android и iOS операционных системах. Согласно статистике, Android занимает наибольшую долю мирового рынка мобильных операционных систем – 71.09%

Одной из ключевых причин популярности Android является открытый характер этой ОС, что позволяет множеству производителей мобильных устройств устанавливать Android на свои гаджеты. Это способствует большему разнообразию устройств, среди которых пользователи могут выбирать в соответствии со своими потребностями и бюджетом. Благодаря открытости Android разработчики также имеют возможность создавать широкий спектр приложений.

Однако, помимо преимуществ, открытость Android может приводить к различным проблемам. Некоторые производители создают свои оболочки операционной системы, что может сказаться на производительности и стабильности работы приложения. То есть одно и то же приложение будет

работать по-разному работать на одной версии Android, но с разными оболочками. Кроме того, открытость Android также увеличивает риск воздействия вредоносных программ и вирусов.

iOS отличается более закрытым подходом к разработке. Этот подход также обеспечивает более высокий уровень безопасности, поскольку внешние разработчики не имеют доступа к ядру операционной.

Кроме того, App Store, официальный магазин приложений для iOS, известен своей строгой политикой контроля качества и безопасности. Все приложения проходят тщательную проверку перед публикацией, что снижает вероятность установки вредоносных программ на устройство.

Таким образом, как Android, так и iOS имеют свои уникальные преимущества и недостатки, и выбор между ними часто зависит от индивидуальных предпочтений и потребностей пользователя. Но зачастую все популярные мобильные приложения существуют как на

1.5 Постановка задачи

Целью этого проекта – разработка удобного приложения для просмотра новостей.

Все современные приложения существуют в версиях для устройств на платформе Android и iOS. Разработка нативных приложений является трудно затратной, так как конечным результатом будет два разных приложения, написанных на разных языках программирования и отличающихся друг от друга. Решением этой проблемы является кроссплатформенная разработка. В этом случае, приложения имеют одну кодовую базу, которая компилируется отдельно под iOS и Android. При такой разработке, приложения будут иметь минимальные различия, обусловленные особенностями фреймворка без необходимости дублировать изменения для каждой платформы. Ещё одним плюсом кроссплатформенной разработки является низкая цена на этапах

разработки и поддержки в сравнении с нативными приложениями.

Для Android, необходима выбрать версию SDK API Level 21 (Android 5.0 Lollipop) и выше. Версия 5.0 Lollipop является важным этапом в развитии Android, представляя новый дизайн Material Design, а также вводя ряд улучшений в производительности и функциональности. Выбор этой версии обусловлен тем, что она охватывает значительную часть рынка Android – устройств и включает в себя множество современных функций, таких как улучшенное управление уведомлениями и поддержка мультимедийных возможностей.

Для iOS, версия SDK iOS 14 и выше. iOS 14 была значительным релизом для Apple, вводя множество новых функций, в том числе новый виджетный экран, улучшенные функции сообщений, App Clips и многое другое. Выбор этой версии обоснован тем, что она представляет собой последнюю стабильную версию операционной системы iOS на момент разработки приложения, обеспечивая поддержку современных возможностей и функций, а также совместимость с новейшими устройствами от Apple.

Обе выбранные версии обеспечивают широкий охват рынка и включают в себя множество современных функций, которые могут быть использованы при разработке мобильного приложения просмотра новостей. Кроме того, они предоставляют разработчикам доступ к последним инструментам и API, необходимым для создания функционального и инновационного приложения.

Представленные аналоги имеют ряд преимуществ и недостатков, исходя из этого приложение должно сочетать лучшие стороны представленных приложений.

Приложение должно представлять удобный просмотр новостей, то есть необходимо реализовать, страницу со списком новостей, где элемент новости будет представлять лишь часть новости для более удобного просмотра

списка новостей и так же должны быть отдельная страница просмотра новости где будет представлен весь контент определенной новости. Такая реализация позволит пользователю более удобно просматривать список новостей не загромождая страницу со списком новостей полным контентом определенной новости, для этого необходимо определить размер виджета новости что этом виджете будет отображаться. Так же необходимо реализовать механизм перевод текста на русский язык.

Так же необходимо реализовать отдельную страницу просмотра картинки с возможностью увеличения изображения

Необходимо реализовать страницу поиска. Поиск должен включать поиск по ключевому слову. Поиск по диапазону дат. Для этого необходимо реализовать отдельную страницу поиска с возможностью использования настроек поиска.

Необходимо реализовать механизм сохранение новости в локальную базу данных, а также удаление новости из нее. Для потребуется реализовать отдельную страницу для просмотра сохраненных новостей. Исходя из этих требований необходимо реализовать 4 экранных форм:

- страница со списком новостей;
- страница отдельной новости;
- страница поиска новостей;
- страница со списком сохраненных новостей.

Разрабатываемое приложение должно соответствовать диаграмме вариантов использования, представленной на рисунке 3.

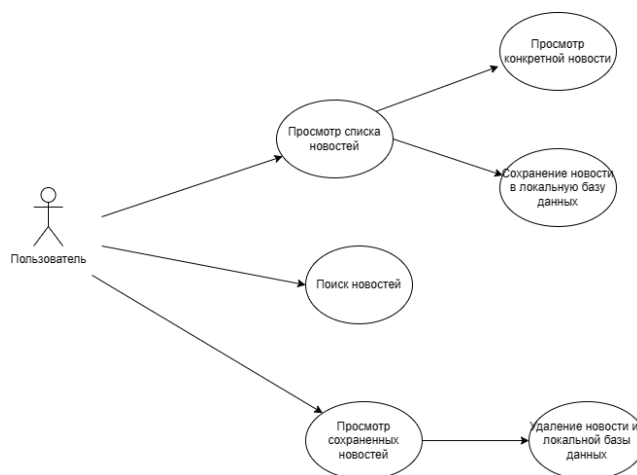


Рисунок 3 – UML - диаграмма вариантов использования

Данная диаграмма определяет различные сценарии использования приложения, включая основные действия пользователей и системы, а также взаимодействия между ними. Приложение будет разработано с учетом этих сценариев, обеспечивая пользователей интуитивно понятным и эффективным способом взаимодействия с функционалом приложения. Приложение необходимо будет реализовать с учетом описанных в диаграмме вариантов использования.

1.6 Выбор инструментов разработки

Приступая к разработке приложения нужно выбрать фреймворк кроссплатформенной разработки. Необходимо определить какой фреймворк лучше подойдет для реализации данной темы.

Flutter и React Native – это два популярных фреймворка для разработки мобильных приложений с использованием – JavaScript для React Native и Dart для Flutter. Flutter использует собственный движок рендеринга, что обеспечивает быстрый и отзывчивый интерфейс. Функция горячей перезагрузки позволяет мгновенно видеть изменения в реальном времени. Flutter предоставляет доступ к API нативных платформ, что позволяет реализовывать широкий спектр функциональности, включая работу с

камерой, геолокацией и другими аппаратными ресурсами. Приложения на Flutter имеют больший размер из-за включения движка Flutter внутри приложения [1].

React Native также имеет ряд недостатков и преимуществ. React Native имеет огромное сообщество разработчиков и обширное количество библиотек и компонентов. Приложения, разработанные на React Native, могут иметь меньший размер благодаря использованию JavaScript Core, которая является прослойкой между JavaScript и нативными методами. Из-за того, что React Native использует нативные компоненты интерфейса, может показаться, что приложения на RN должны работать быстрее, чем на Flutter, но результаты тестирования показали, что Flutter более производительный, чем RN [2].

В итоге, выбор между Flutter и React Native был сделан в пользу Flutter из-за его высокой производительности, активного развития и поддержки сообщества разработчиков, что делает его наиболее подходящим выбором для создания конкурентоспособных приложений.

После того, как выбран язык программирования и фреймворк, необходимо выбрать среду разработки. Главным критерием при выборе будет поддержка языка Dart и возможность работы с flutter.

Самыми популярными среди программистов, среды разработки, которые удовлетворяют этим условиям являются:

- IntelliJ IDEA;
- Android Studio;
- Visual Studio Code.

IntelliJ IDEA – среда разработки, созданная компанией JetBrains, поддерживает множество языков программирования. Эта среда поддерживает систему контроля версий, есть возможность скачивать и устанавливать плагины, которые облегчают разработку программы.

Например, подсветка скобок. IntelliJIDEA обладает и рядом недостатков, таких как платная подписка и высокие требования к ресурсам оперативной памяти компьютера.

AndroidStudio – среда разработки от компании Google, сделана на основе IntelliJIDEA. Эта среда предназначена для разработки мобильных приложений для операционную систему Android, Она включает в себя встроенный Android-эмулятор, с возможностью выбора не только версии Android, но и размеров экрана эмулятора, поддерживает рендер средствами GPU. AndroidStudio поддерживает также систему контроля версий. Среди недостатков, можно отметить высокие требования к ресурсам компьютера

Visual Studio Code – среда разработки от компании Microsoft, имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, есть возможность плагины для работы с Dart и Flutter, и подключение эмулятора, но изначально Visual Studio Code не разрабатывалась как среда для разработки под мобильные приложения. И многие возможности как в Android Studio просто отсутствуют.

Изучив все особенности и недостатки представленных сред разработки было принято решение использовать AndroidStudio. Эта среда разработки имеет встроенный эмулятор мобильных устройств. При создании эмулятора можно выбрать уже существующую версию смартфона, но только от компании Pixel так как Android Studio принадлежит Google. Но так же можно и самому создать эмулятор с нужными параметрами. Возможность работы с эмулятором позволит, не имея реального устройства, тестировать приложения на разных устройствах, что в свою очередь повлияет на качество разработки приложения [3].

Для улучшения функциональности приложения необходимо выбрать дополнительные библиотеки. На Flutter есть пакетный менеджер pub через

который можно устанавливать дополнительные библиотеки которые облегчают процесс создания приложения. Достаточно прописать названия библиотеки в файл pubspec.yaml и выполнить команду pubget, которая сама установит необходимые библиотеки.

Для данного проекта были выбраны следующие библиотеки:

- flutter_bloc. Библиотека реализует паттерн Business Logic Component. С помощью этого паттерна мы можем легко разделить бизнес-логику приложения от пользовательского интерфейса;
- get_it. Библиотека для управления зависимостями в приложении. Она позволяет создавать и управлять экземплярами объектов, которые могут быть доступны в разных частях приложения;
- http. (точки) Библиотека позволяет выполнять HTTP-запросы в приложении;
- cached_network_image. Библиотека предоставляет виджет для отображение картинки по url. Библиотека, облегчающая процесс сериализации данных в JSON и десериализации их в модель;
- serdes_json: Библиотека позволяет генерировать модели, которые облегчат работу с сервером;
- translator: Библиотека для перевода текста с одного языка на другой;
- flutter_date_pickers: Библиотек которые облегчает работу с типом DateTime, позволя форматировать дату в любой удобный формат;
- sqflite: Библиотека для работы с SQLite базами данных в Flutter. Она необходима для сохранения новостей на устройство;
- blurrycontainer: эта библиотека представляет возможность использовать уже реализованный виджет с размытым фоном.

Использование библиотек в проекте представляют больше возможностей, так как позволяют использовать уже готовые решения что сказывается на времени разработке.

1.7 Структура приложения

Структура приложения, будет состоять из трех частей:

- view;
- domain;
- gateway.

Часть View будет представлена классами, реализующие графический интерфейс приложения. Всего будет 5 таких классов, реализующие следующие страницы:

- экран со списком новостей;
- экран с поиском новостей;
- экран конкретной новости;
- экран просмотра картинки;
- экран со списком сохраненных новостей.

Во Flutter каждый графический элемент называется виджет, фреймворк представляет ряд виджетов которые можно использовать для формирования интерфейса приложения, например, виджет кнопки, текстового поля, так же есть композиционные виджеты для расположения других виджетов такие как `row` `column`. Оперируя этими виджетами можно создавать сложные макеты экранов. Код представляет из себя иерархическую структуру. Один виджет является родителем другого.

В Flutter каждый пользовательский виджет представляет собой класс, который либо наследуется от `StatelessWidget` (без состояния), либо от `StatefulWidget` (сохраняющий состояние). `StatelessWidget` являются статичными, они не изменяют своё состояние после инициализации, в то время как `StatefulWidget` способны обновлять своё состояние в ответ на внешние изменения или внутренние события [4].

Для эффективной организации кода каждый экран приложения должен иметь свой собственный класс-виджет, который определяет его внешний вид

и поведение. Повторно используемые компоненты могут быть вынесены в отдельные классы–виджеты, чтобы избежать дублирования кода и обеспечить модульность приложения.

Помимо этого, в разработке на Flutter широко применяется паттерн управления состоянием BLoC (Business Logic Component). BLoC представляет собой слой, ответственный за управление логикой приложения, в том числе за взаимодействие с сервером, обработку данных и обновление пользовательского интерфейса. Он основан на принципах реактивного программирования, используя два основных потока: поток событий (event stream) и поток состояний (state stream) [5].

При работе приложения события генерируются и передаются в поток событий BLoC, где они обрабатываются и преобразуются в соответствующие состояния. Эти состояния затем отправляются в поток состояний, который наблюдается графическим интерфейсом, обновляя его в соответствии с текущим состоянием приложения.

Для упрощения реализации этой архитектуры в проектах Flutter часто используется библиотека flutter_bloc, которая предоставляет удобные средства для создания, управления и подписки на потоки событий и состояний. Это позволяет разработчикам более эффективно управлять состоянием приложения и создавать более отзывчивые и масштабируемые интерфейсы.

Необходимо будет создать следующие классы:

- NewsBloc, который будет наследоваться от базового класса Bloc. Именно в нем будет описана логика представления;
- абстрактный класс NewsEvent. От него будут наследоваться классы событий. В рамках данного приложения будут необходимы следующие классы:

1) LoadNewsEvent. Событие загрузки новостей, по этому

событию будут загружаться новости.

2) LoadNewsOnNameEvent. Событие поиска новостей, по этому событию будут загружаться новости с учетом фильтра.

3) LoadNewsSaveEvent. Событие загрузки сохраненных новостей из локальной базы данных.

4) EmptySearchEvent. Событие сброса поиска, это событие будет вызываться, когда на странице поиска необходимо вернуться в исходное состояние. Это состояние будет очищать список найденных новостей и сбрасывать фильтры.

5) AddReactionNewsEvent. Событие добавление реакции к новости сохранение новости в локальную базу данных.

6) DeleteReactionNewsEvent. Событие удаления реакции к новости удаления новости из локальной базы данных.

– абстрактный класс NewsState. От него будут наследоваться классы, объекты которых будут являться состояниями [6]. В рамках данного приложения будут необходимы следующие классы:

1) LoadingNewsState. Объект этого класса будет являться начальным состоянием, то есть в момент загрузки новостей.

2) LoadedNewsState. Объект этого класса будет являться состоянием, которое хранит в себе загруженные новости.

3) EmptySearchState. Объект этого класса будет являться состоянием, которое будет сбрасывать страницу поиска в исходное состояние.

Часть Gateway содержит классы для работы с сервером, методы классов будут отправлять запросы к серверу и возвращать данные. NewsGatewayImpl будет унаследована от NewsGateway и содержать в себе методы для получения списка новостей:

– getNews: Получает список новостей. Генерирует HTTP GET запрос к

серверу и возвращает список новостей в виде объекта типа NewsResponse;

- getNewsOnName: Получает список новостей по заданным фильтрам.

В качестве фильтров будут такие параметры: диапазон дат, новости на русском, ключевое слово. Отправляет HTTP GET запрос к серверу и возвращает список новостей в виде объекта типа NewsResponse.

Слой бизнес–логики должен включать классы, которые осуществляют реализацию бизнес–логики приложения. Необходимо разработать интерфейс NewsService, который будет содержать методы для взаимодействия со слоем доступа к данным с целью получения информации из внешних источников. При необходимости данные должны быть преобразованы в соответствующий формат. Для этого предполагается создание интерфейса NewsFactory, методы которого будут принимать структуры данных в качестве параметров и преобразовывать их в формат, необходимый для представления данных слою представления. Реализацию этих интерфейсов будут осуществлять классы NewsServiceImpl и NewsFactoryImpl соответственно [7].

Методы, которые должен содержать класс NewsServiceImpl представлены ниже:

- getNews: Обращается к _newsGateway, чтобы получить список новостей, затем использует _newsModelsFactory для создания объекта News, содержащего информацию о новостях. Если происходит ошибка, выбрасывает исключение NewsException.

- searchNewsOnName: Поиск новостей по заданному названию. Обращается к _newsGateway, чтобы получить список новостей, затем использует _newsModelsFactory для создания объекта News, содержащего информацию о новостях. Если происходит ошибка, выбрасывает исключение NewsException. UML диаграммы классов, которые необходимо разработать, представлена на рисунке5.

Рисунок 6 — UML диаграмма пакетов

1.8 Реализация

Процесс разработки можно поделить на следующие пункты:

– разработка слоя presentation:

- 1) разработка графического интерфейса;
- 2) разработка логики представления.

– разработка слоя domain:

- 1) разработка классов– моделей;
- 2) разработка классов– сервисов, реализующих бизнес– логику приложения;

- 3) разработка классов для преобразования данных.

– разработка слоя gateway.

Разрабатываемое приложение будет разрабатываться с учетом принципов и потоков, представленных на диаграмме деятельности. Диаграмма деятельности представлена на рисунке 7.

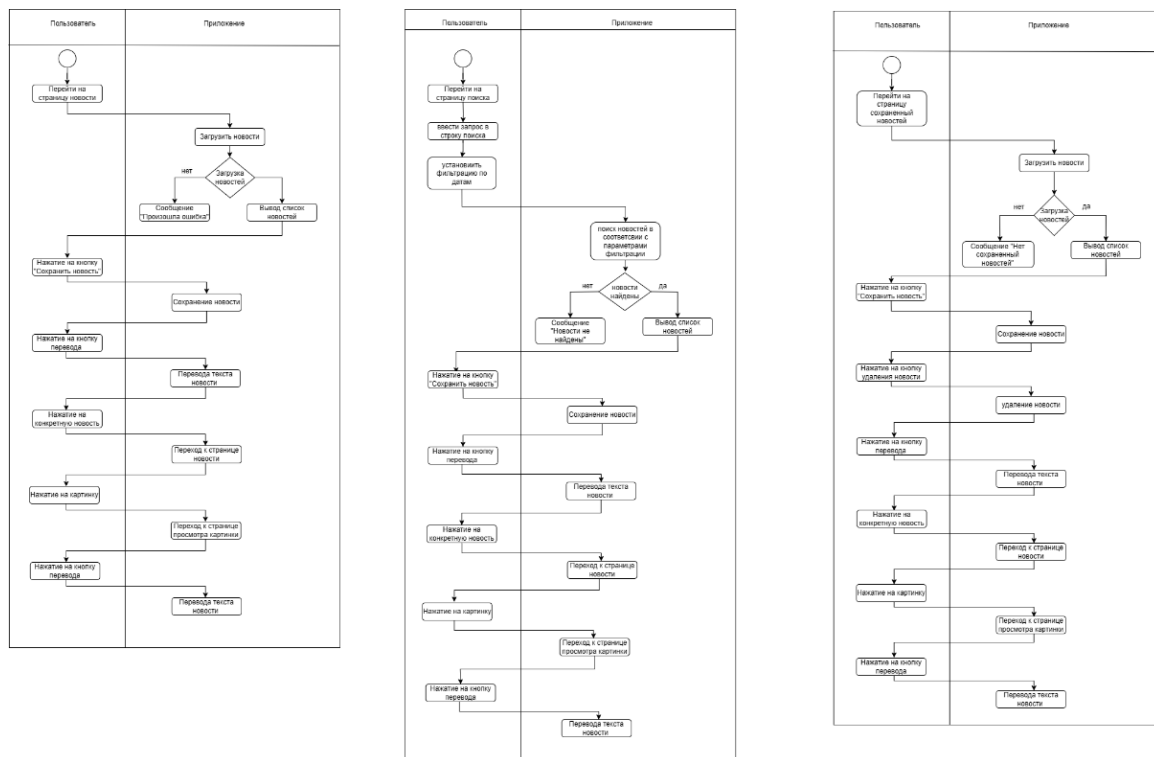


Рисунок 7 – UML – диаграмма деятельности

Данная диаграмма деятельности служит основой для проектирования логики приложения и определения последовательности действий, которые пользователь и приложения должны выполнить для достижения определенных целей. Таким образом, приложение должно реализовывать указанные на диаграмме деятельности процессы и процедуры, обеспечивая соответствие функциональных требований и ожиданиям пользователей.

Основные страницы данного приложения – страницы со списком новостей поиска и сохраненный новостей, на каждой из этих страниц может отображаться список новостей, виджет этой новости будет представлен отдельном классом. Исходя из предложенной диаграммы классов это виджет будет наследоваться от `StatefullWidget`, то есть виджет будет сохранять свое состояние. В этом виджете импортируются необходимые пакеты и библиотеки, пакеты для работы с изображениями, кэширования изображений, переводом текста. `NewsWidget`: Это `StatefulWidget`, который представляет собой виджет, отображающий одну новость. Он принимает параметры, такие как `newsItem` экземпляр блока новостей. `_NewsWidgetState` Это состояние для `NewsWidget`, которое реализует методы жизненного цикла и содержит логику отображения новости. `initState()`: Этот метод инициализирует состояние виджета при его создании. В данном случае он используется для получения данных о новости и установки начального состояния. `build()`: Этот метод строит UI для отображения новости. Он содержит различные виджеты, такие как заголовок новости, изображение, автор, дата публикации и кнопка перевода. `_headerContentNews()`: Этот метод создаёт виджет для отображения заголовка и информации о новости, такой как автор и дата публикации. `_imageContentNews()`: Этот метод создаёт виджет для отображения изображения новости. Он также обрабатывает случаи, когда изображение загружается с сервера или возникают ошибки при

загрузке. `_titleContentNews()`: Этот метод создаёт виджет для отображения заголовка и краткой информации о новости. `trans()`: Этот метод выполняет перевод заголовка и краткой информации о новости на другой язык. Ниже приведен фрагмент кода данного виджета.

```
class NewsWidget extends StatefulWidget {
  final NewsItem newsItem;
  final NewsBloc newsBloc;

  const NewsWidget({
    this.isFirst = false,
    required this.newsItem,
    required this.newsBloc,
  });

  @override
  State<NewsWidget> createState() => _NewsWidgetState();
}

class _NewsWidgetState extends State<NewsWidget>
  with AutomaticKeepAliveClientMixin {
  List<Emoji> emojiList = [];
  late String title;
  late String bottomTitle;
  bool isTrans = false;

  @override
  void initState() {
    super.initState();
    title = widget.newsItem.title;
    bottomTitle = widget.newsItem.bottomTitle;
    emojiList = widget.newsItem.emoji;
  }
}
```

```

GoogleTranslatortranslator = GoogleTranslator();
voidtrans() {
  if (!isTrans) {
    translator.translate(tittle, to: «ru»).then((value) {
      tittle = value.toString();
    });
    translator.translate(bottomTittle, to: «ru»).then((value) {
      setState(() {
        bottomTittle = value.toString();
        isTrans = !isTrans;
      });
    });
  } else {
    setState(() {
      tittle = widget.newsItem.title;
      bottomTittle = widget.newsItem.bottomTittle;
      isTrans = !isTrans;
    });
  }
}

@override
boolgetwantKeepAlive =>true;
lateboolisImage = false;

@override
Widgetbuild(BuildContextcontext) {
  super.build(context);
  returnContainer(
    margin: !widget.isFirst
      ? EdgeInsets.only(bottom: 8, top: 12)
      : EdgeInsets.only(
        bottom: 8,
      ),
  ),

```

```

decoration: BoxDecoration(
  color: Colors.white, //AppColors.white,
  borderRadius: !widget.isFirst
    ? BorderRadius.only(
      topRight: Radius.circular(12),
      topLeft: Radius.circular(12),
      bottomLeft: Radius.circular(12),
      bottomRight: Radius.circular(12),
    )
    : BorderRadius.only(
      bottomLeft: Radius.circular(12),
      bottomRight: Radius.circular(12),
    ),
  ),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      InkWell(
        onTap: () async {
          Navigator.of(context).push(
            CupertinoPageRoute(builder: (BuildContextcontext) {
              return NewsPage(
                newItem: widget.newsItem,
              );
            }
          ),
        ),
      ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          _headerContentNews(),
          _imageContentNews(),
        ],
      ),
    ],
  ),
)

```

```

        _tittleContentNews(),
      ],
    )),
  LikesWidget(
    tittle: widget.newsItem.title,
    emoji: emojiList,
    newsBloc: widget.newsBloc,
  ),
],
));
}

```

Этот виджет будет отображаться на трех страницах: списка, поиска, сохраненных новостей. Внешний вид этого виджета представлен на рисунке 8.



Рисунок 8 — Виджет новости

Страница поиска определяется двумя классами: `NewsSearchPage` и `NewsSearchPageFlow`, которые отвечают за отображение главного экрана

поиска новостей и его состояния соответственно. Также есть класс `SettingsModal`, который представляет собой модальное окно для настройки параметров поиска новостей. Виджеты используют `Bloc` для управления состоянием. Они подписываются на состояния блока и обновляют интерфейс в соответствии с полученными данными Приложение использует сервис для получения новостей и отправки запросов к API. В данном случае используется сервис `NewsService`, который передается в блок новостей при создании. Отображение списка новостей: Приложение отображает список новостей с помощью виджета `ListView.builder`. Этот список загружается с сервера при отправке запроса на поиск новостей. Настройка параметров поиска: Пользователь может настроить параметры поиска новостей, такие как диапазон дат, язык новостей и имя издателя. Это делается с помощью модального окна `SettingsModal`. Приложение обрабатывает различные состояния, такие как загрузка новостей, успешное получение новостей, пустой результат поиска и другие. Это позволяет пользователю видеть актуальную информацию о состоянии приложения. Ниже приведен фрагмент кода данной страницы.

```
class NewsSearchPage extends StatelessWidget {
  const NewsSearchPage();

  @override
  Widget build(BuildContext context) {
    return BlocProvider<NewsBloc>(
      create: (context) => NewsBloc(
        GetIt.instance<NewsService>()),
      child: NewsSearchPageFlow(),
    );
  }
}

class NewsSearchPageFlow extends StatefulWidget {
```

```

NewsSearchPageFlow();

@override
State<NewsSearchPageFlow> createState() => _NewsSearchPageFlow();
}

class _NewsSearchPageFlow extends State<NewsSearchPageFlow> {
  _NewsSearchPageFlow();

  List<NewsItem> items = [];

  @override
  void initState() {
    super.initState();
    _textController = TextEditingController();
    _textController.addListener(() {
      if (_textController.text.isNotEmpty) {
        setState(() {
          isHasText = true;
        });
      } else {
        setState(() {
          isHasText = false;
        });
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Container(
          child: Text('Поиск'),
        ),
      ),
    ),
  }
}

```

```

body: BlocConsumer<NewsBloc, NewsState>(listener: (context,
state) {
  if (state.isLoadingNewsState) {
    isLoading = false;
  }

  if (state.isLoadedNewsState) {
    if (state.news.items.isEmpty) {
      isEmptySearch = true;
    }

    items = state.news.items;
    isLoading = true;
  }

  if (state.isEmptySearchState) {
    isLoading = false;
    items = state.news.items;
    isEmptySearch = false;
    nameSearch = '';
    _textController.clear();
  }

  }, builder: (context, state) {
return Scaffold(
  appBar: AppBar(
    title: Container(
      child: widgetSearch(),
    ),
  body: Container(
    child: !isEmptySearch
      ? Container(
        color: Colors.black12,
        child: Column(children: [
          Expanded(
            child: Container(
              child:
                Column(

```

```

crossAxisAlignment: CrossAxisAlignment.center,
children: [
Expanded(
child: ListView.builder(
controller: _scrollController,
itemCount: items.length,
itemBuilder: (context, index) {
if (isLoading) {
returnNewsWidget(
newsItem: items[index],
newsBloc:

_blocNews(context),

);
} else {

returnSizedBox.shrink();

}},
),

```

Внешний вид данной страницы представлен на рисунке 9.

Изм.	Лист	№ докум.	Подпись	Дата

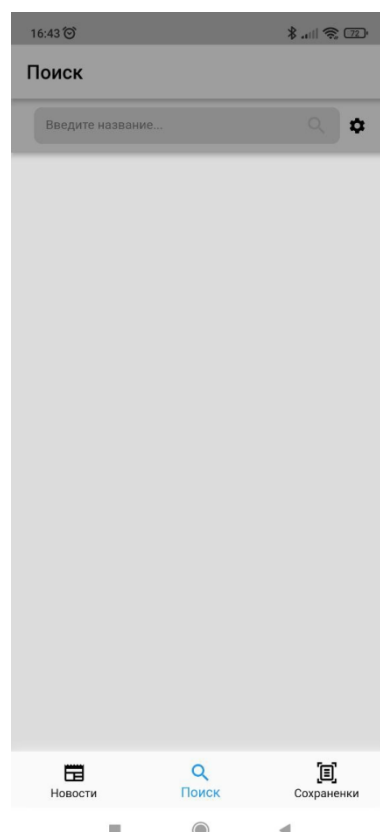


Рисунок 9 — Страница поиска

Страница отдельной новости представлена двумя классами `NewsPage` и `NewsPageFlow`. `NewsPage` представляет экран новости и принимает объект `NewsItem`, содержащий информацию о конкретной новости. `NewsPageFlow` является состоянием этого экрана. Экран новости использует `Flutter Bloc` для управления состоянием. Он предоставляет блок новостей и передает его вниз по виджетам. Код отвечает за отображение информации о новости, такой как заголовок, автор, дата публикации, изображение, ссылка на источник и другие детали. Эта информация извлекается из объекта `NewsItem`. Присутствует возможность перевода заголовка новости на русский язык с помощью библиотеки `translator`. Перевод выполняется при нажатии на соответствующую иконку. Пользователь может просмотреть изображение новости в полном размере и перейти по ссылке на источник новости. Обработка ошибок: В случае возникновения ошибок, например, при загрузке данных новости, экран

отображает соответствующее сообщение об ошибке. Внешний вид данной страницы представлен на рисунке 10.



Рисунок 10 — Страница новости

Страница сохраненных новостей содержит два класса: NewsSavePage и NewsSavePageFlow. NewsSavePage представляет экран «Сохраненные новости», который создается с помощью BlocProvider для предоставления блока новостей. NewsSavePageFlow является состоянием этого экрана. Экран «Сохраненные новости» использует FlutterBloc для управления состоянием. Он подписывается на состояния блока новостей и обновляет интерфейс в соответствии с полученными данными. При запуске экрана отправляется событие LoadNewsSaveEvent для загрузки сохраненных новостей. Полученные новости отображаются в списке. Код обрабатывает различные состояния, такие как загрузка, успешная загрузка новостей, ошибка загрузки и другие. В зависимости от состояния отображается соответствующий

контент на экране. Если сохраненные новости загружены успешно и список не пуст, они отображаются в виде списка с помощью ListView.builder. Для каждой новости создается виджет NewsWidget. В случае возникновения ошибок при загрузке новостей, на экране отображается сообщение об ошибке. Внешний вид данной страницы представлен на рисунке 11.

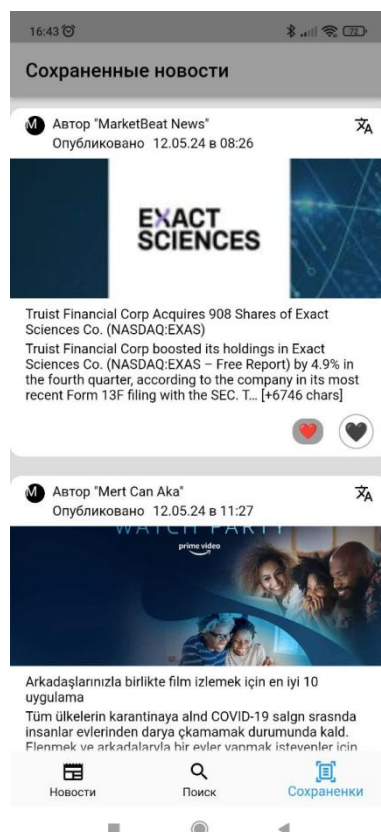


Рисунок 11 — Страница сохраненных новостей

Страница списка новостей представлена двумя классами NewsMainPage и NewsMainPageFlow. NewsMainPage представляет экран главной страницы новостей, который создается с помощью BlocProvider для предоставления блока новостей. NewsMainPageFlow является состоянием этого экрана. Экран главной страницы новостей использует FlutterBloc для управления состоянием. Он подписывается на состояния блока новостей и обновляет интерфейс в соответствии с полученными данными. При запуске экрана отправляется событие LoadNewsEvent для загрузки новостей. Полученные новости отображаются в списке. Код обрабатывает различные

Изм.	Лист	№ докум.	Подпись	Дата

состояния, такие как загрузка, успешная загрузка новостей, ошибка загрузки и другие. В зависимости от состояния отображается соответствующий контент на экране Пользователь может обновить список новостей, нажав на плавающую кнопку обновления, которая появляется на экране. Если новости успешно загружены и список не пуст, они отображаются в виде списка с помощью ListView.builder. Для каждой новости создается виджет NewsWidget. В случае возникновения ошибок при загрузке новостей, на экране отображается сообщение об ошибке.

Внешний вид данной страницы представлен на рисунке 12.

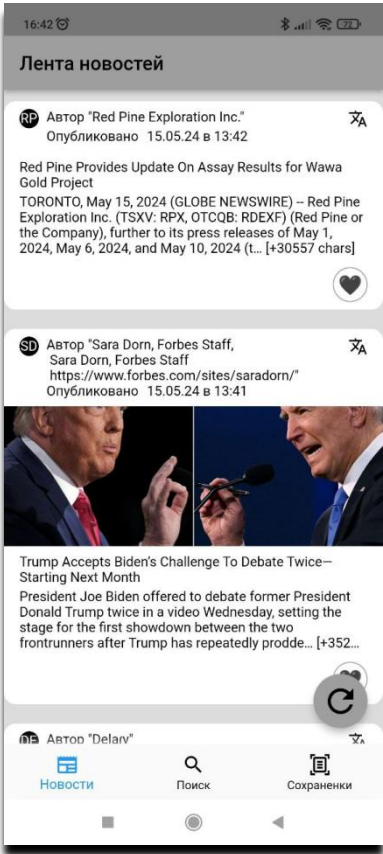


Рисунок 12 — Страница лента новостей

Страница просмотра картинки представлена виджетом ImageViewPage, который используется для отображения изображения по его URL. Давайте разберем основные моменты этого кода: StatefulWidget:ImageViewPageявляется StatefulWidget, что означает, что его

состояние может изменяться во время выполнения приложения. Атрибут imageUrl: Класс ImageViewPage принимает атрибут imageUrl, который представляет собой URL изображения, которое нужно отобразить. Состояние _isVisible: В классе _ImageViewPageState объявлена переменная _isVisible, которая управляет видимостью элементов пользовательского интерфейса. При нажатии на изображение она меняется, чтобы скрыть или показать верхнюю панель с кнопкой назад. Используется GestureDetector для обнаружения жестов, таких как нажатие. При нажатии на изображение меняется видимость верхней панели. AnimatedPositioned и AnimatedSwitcher: Эти виджеты обеспечивают анимированное перемещение и изменение видимости верхней панели при ее отображении или скрытии. Используется для возможности масштабирования и перемещения изображения с помощью жестов. WillPopScope обеспечивает возможность перехвата события нажатия на кнопку «Назад» на устройстве. В данном случае, при нажатии ничего не происходит, так как управление попыткой возврата обработано внутри GestureDetector. Обработка события нажатия на кнопку «Назад»: При нажатии на кнопку «Назад» происходит завершение активности с помощью метода Navigator.pop(context). Этот код создает полноэкранное окно с изображением, которое можно масштабировать и перемещать, а также отображать и скрывать верхнюю панель с кнопкой назад при нажатии на изображение. Внешний вид данной страницы представлен на рисунке 13. Пример кода приведен ниже.

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

class ImageViewPage extends StatefulWidget {
  ImageViewPage({
    required this.imageUrl,
  });
```

```

finalStringimageUrl;

    @override
    _ImageViewPageState createState() => _ImageViewPageState();
}

class _ImageViewPageState extends State<ImageViewPage> {
  late bool _isVisible = true;
  final navigatorKey = GlobalKey<NavigatorState>();

  @override
  void initState() {
    super.initState();
  }

  @override
  void dispose() {
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    MediaQueryData media = MediaQuery.of(context);
    return Scaffold(
      backgroundColor: Colors.white,
      body: WillPopScope(
        onWillPop: () async {
          return true;
        },
        child: GestureDetector(
          onTap: () {
            setState(() {
              _isVisible = !_isVisible;
            });
          }
        )
      )
    );
  }
}

```

```

        });
    },
    child: Stack(
      children: [
        AnimatedPositioned(
          duration: Duration(milliseconds: 333),
          curve: Curves.fastOutSlowIn,
          top: 0,
          bottom: 0,
          left: 0,
          right: 0,
          child: InteractiveViewer(
            panEnabled: true,
            minScale: 0.5,
            maxScale: 4,
            child: Image.network(
              widget.imageUrl,
            ),
          ),
        ),
      ],
    ),
    Positioned(
      top: 0,
      left: 0,
      right: 0,
      child: AnimatedSwitcher(
        duration: constDuration(milliseconds: 300),
        reverseDuration: constDuration(milliseconds: 300),
        child: !_isVisible
          ? constSizedBox.shrink()
          : Align(
              alignment: Alignment.topLeft,
              child: Container(
                height: media.padding.top + 54,
                width: media.size.width,

```

```

decoration: BoxDecoration(
  color: Colors.black.withOpacity(0.5),
),
child: Container(
  margin: EdgeInsets.only(
    top: media.padding.top),
  padding:
    EdgeInsets.symmetric(horizontal: 12),
  child: Stack(
    children: [
      Positioned(
        left: 0,
        top: 15,
        child: GestureDetector(
          onTap: () {
            Navigator.pop(context);
          },
        ),
        child: Icon(
          Icons.arrow_back_ios_new,
          color: Colors.black,
          size: 24,
        ),
      ),
    ],
  ),
),
);

```

```
}
}
```

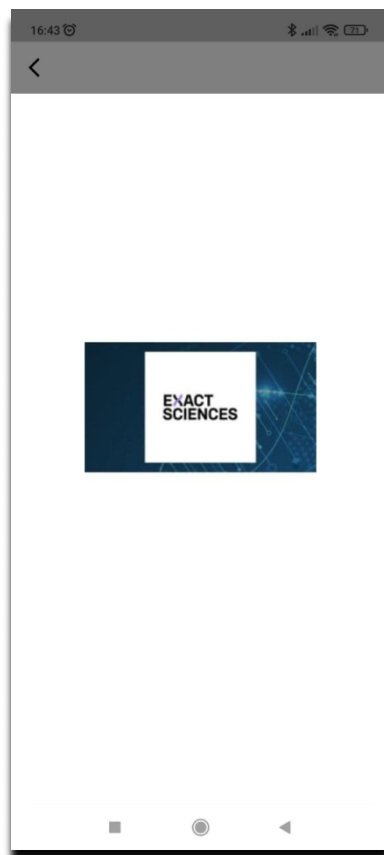


Рисунок 13 — Страница просмотра картинки

Для обеспечения функциональности приложения был создан класс `NewsServiceImpl`, который реализует интерфейс `NewsService`. Этот класс предоставляет методы для получения новостей, поиска новостей по различным параметрам.

Метод `getNews` позволяет получить список новостей с указанием номера страницы и количества новостей на странице. Он обращается к `NewsGateway`, чтобы получить данные о новостях, затем использует `NewsFactory`, чтобы преобразовать полученные данные в модель `News`.

Метод `searchNewsOnName` позволяет выполнить поиск новостей по имени с указанием дополнительных параметров, таких как диапазон дат, является ли новость российской и т.д. Этот метод также обращается к

NewsGateway для получения данных и использует NewsFactory для преобразования результатов в модель News.

Методы работы с базой данных.

Класс NewsServiceImpl также предоставляет методы для работы с базой данных. Методы addDatabaseItem, getDatabaseItems и deleteDatabaseItem позволяют добавлять, получать и удалять новости из БД соответственно.

Эти методы взаимодействуют с классом DatabaseProvider, который обеспечивает доступ к базе данных SQLite и реализует операции для работы с новостями. Пример кода приведен ниже.

```
import
'package:news_app/bloc/news/models/news_models.dart';

import 'package:news_app/factory/factory.dart';
import 'package:news_app/gateway/gateway.dart';
import 'package:news_app/models/news.dart';
import '../db.dart';
import 'news_service.dart';

class NewsServiceImpl implements NewsService {
  final NewsGateway _newsGateway;
  final NewsFactory _newsModelsFactory;

  NewsServiceImpl(
    this._newsGateway,
    this._newsModelsFactory,
  );

  @override
  Future<News> getNews(int page, int perPage) async {
    try {
      final NewsResponse response = await
        _newsGateway.getNews(page, perPage);
      return await
```

```

_newsModelsFactory.getNewsModel(response);
    } on ApiException catch (e) {
        throw NewsException(e.message);
    } on Exception catch (e) {
        throw NewsException('');
    }
}

@override
Future<News> searchNewsOnName(int page, int perPage,
String name, {
    String? firstDate,
    String? lastDate,
    bool? russianNews,
    String? publisherName,
    bool? searchDate
}) async {
    try {
        final NewsResponse response = await
_newsGateway.getNewsOnName(
            page,
            perPage,
            name,
            firstDate: firstDate,
            lastDate: lastDate,
            russianNews: russianNews,
            publisherName: publisherName,
            searchDate: searchDate,
        );
        return await
_newsModelsFactory.getNewsModel(response);
    } on ApiException catch (e) {
        throw NewsException(e.message);
    } on Exception catch (e) {
        throw NewsException('');
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

44

```

        }
    }
    @Override
    Future<void> addDatabaseItem(NewsItem newsItem) async {
        DatabaseProvider.db.addNewsItem(newsItem);
    }
    @Override
    Future<List<NewsItem>> getDatabaseItems() async {
        return await DatabaseProvider.db.getNewsItems();
    }
    @Override
    Future<void> deleteDatabaseItem(String title) async {
        DatabaseProvider.db.deleteNewsItem(title);
    }
}

```

Для взаимодействия с внешним API новостей был создан класс NewsGatewayImpl, который реализует интерфейс NewsGateway. Этот класс предоставляет методы для получения новостей и выполнения поиска новостей с различными параметрами.

Метод getNews используется для получения списка новостей с указанием номера страницы и количества новостей на странице. Он формирует запрос к внешнему API новостей и возвращает ответ в виде объекта NewsResponse, содержащего информацию о новостях.

Метод getNewsOnName позволяет выполнить поиск новостей по имени с дополнительными параметрами, такими как диапазон дат, является ли новость российской и т.д. Он формирует запрос к внешнему API с учетом указанных параметров и возвращает результат поиска в виде объекта NewsResponse.

Оба метода используют пакет http для отправки HTTP-запросов к внешнему API новостей и обработки полученных данных в формате JSON.

```

// Package imports:
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:news_app/models/news.dart';
import 'gateway.dart';

class NewsGatewayImpl implements NewsGateway {
  NewsGatewayImpl();

  @override
  Future<NewsResponse> getNews(int page, int perPage) async
{
    final uri =
Uri.parse('https://newsapi.org/v2/everything?q=covid&sortBy=publishedAt&apiKey=9cf2279e951d495a96f39cc876f11e1b');
    final response = await http.get(
      uri,
    );
    final response = await http.get(
      uri,
    );
    return
NewsResponse.fromJson(jsonDecode(response.body));
}

```

1.9 Тестирование

Целью этапа тестирования в разработке мобильного приложения для просмотра новостей является гарантирование высокого качества программного продукта. Функциональное тестирование является ключевым инструментом для обеспечения качества и надежности мобильного приложения для просмотра новостей, а также для удовлетворения потребностей и ожиданий пользователей. Функциональное тестирование

играет важную роль в процессе разработки программного обеспечения, в том числе и в мобильной разработке, по нескольким причинам

Проверка соответствия требованиям: Функциональное тестирование помогает убедиться, что разрабатываемое приложение соответствует заявленным требованиям и спецификациям. Это важно для того, чтобы обеспечить, что приложение выполняет свою основную задачу и ожидания пользователей. Выявление дефектов на ранних стадиях: Проведение функционального тестирования на ранних этапах разработки позволяет выявлять и устранять дефекты еще до того, как они окажутся в конечном продукте. Это значительно снижает затраты на исправление ошибок и позволяет сократить время разработки. Обеспечение качества:

- функциональное тестирование помогает обеспечить высокое качество разрабатываемого продукта. Проверка каждой функции и ее корректной работы способствует созданию надежного и стабильного приложения, которое не подводит пользователей;

- повышение удовлетворенности пользователей:

- 1) Корректная работа основных функций приложения, подтвержденная функциональным тестированием, способствует удовлетворенности пользователей. Пользователи ожидают, что приложение будет работать без сбоев и неполадок, и функциональное тестирование помогает обеспечить их ожидания.

- поддержание репутации компании: Качественное функциональное тестирование помогает предотвратить выпуск на рынок приложения с серьезными дефектами, что может негативно сказаться на репутации компании–разработчика. Стабильное и надежное приложение, прошедшее тщательное функциональное тестирование, способствует укреплению репутации компании как надежного разработчика.

Таким образом, функциональное тестирование является важной частью

процесса разработки программного обеспечения, которая помогает обеспечить качество, надежность и соответствие требованиям разрабатываемого приложения.

В рамках функционального тестирования были разработаны следующие тест кейсы представленные в следующих таблицах.

Таблица 1– Тест кейс. Сохранение новости

Предварительные действия	Пользователь открыл приложение, открыта страница новости, новости предварительно загружены, есть сохраненные новости			
Тестовый сценарий		Шаги	Ожидаемый результат	Фактический результат
	1	Пользователь нажал на кнопку лайк у определенной новости	Появилась иконка лайка в карточке новости	Совпадает с ожидаемым
	2	Пользователь удалил лайк	Иконка лайка пропала	Совпадает с ожидаемым
	3	Пользователь снова поставил лайка	Появилась иконка лайка в карточке новости	Совпадает с ожидаемым
	4	Пользователь перешел на страницу сохраненных новостей	Должна появиться только что сохраненная новость	Совпадает с ожидаемым

Таблица 2– Тест кейс. Работа приложения без интернета

Предварительные действия	Пользователь находится не в приложении			
Тестовый сценарий		Шаги	Ожидаемый результат	Фактический результат
	1	Пользователь оключил интернет, и вошел в приложение	Открылась страница просмотра новостей, после загрузки появилась сообщение «Произошла ошибка»	Совпадает с ожидаемым
	2	Пользователь перешел на странцу сохраненных новостей	Отобразились сохраненные новости	Совпадает с ожидаемым
	3	Пользователь перешел на страницу поиска, ввел в строку поиска произвольный текст,	Отобразилась надпись ничего не найдено	Совпадает с ожидаемым

		произвел поиск		
--	--	----------------	--	--

Таблица 3– Тест кейс. Поиск

Предварительные действия	Пользователь находится на странице поиска			
Тестовый сценарий		Шаги	Ожидаемый результат	Фактический результат
	1	Пользователь ввел с строку поиска некорректный текст	Отобразилась надпись ничего не найдено	Совпадает с ожидаемым
	2	Пользователь ввел с строку поиска корректный текст	Отобразились новости с учетом поиска	Совпадает с ожидаемым
	3	Пользователь в настройках поиска указал даты новости, отобразились, произвел поиск	Отобразилась соответствующие поиску новости	Совпадает с ожидаемым

В результате проведения функционального тестирования мобильного приложения для просмотра новостей можно с уверенностью заявить, что все тесты были выполнены успешно, и значимых дефектов или ошибок не было обнаружено. Это говорит о высоком уровне качества приложения и о его готовности к выпуску на рынок. Отсутствие обнаруженных багов свидетельствует о тщательной разработке и тестировании приложения. Это заключение подчеркивает готовность приложения к внедрению и использованию конечными пользователями без ощутимых проблем или неудобств.

2 Охрана труда

Несмотря на все преимущества современных технологий, не стоит забывать и о минусах. А именно о воздействии на физическое здоровье человека. Продолжительная работа с компьютером приводит к накопительному отрицательному воздействию на здоровье, т.е. ущерб долгое время может не проявляться явно, а постепенно и незаметно копится. Для того что бы минимизировать угрозы, необходимо проводить ряды профилактических мер. Ниже подробнее их рассмотрим.

Одну из важнейших ролей играет организации рабочего места. Необходимо соблюдать расстояние до монитора. Оно должно быть на расстоянии вытянутой руки. Самый оптимальный наклон экрана будет составлять 30 градусов, а клавиатура должна находиться на расстоянии не менее чем 20–30 сантиметров от края стола.

При работе за компьютером руки должны быть согнуты, а локти оставаться на столе. При таком расположении рук отсутствует напряжение в кистях. В дальнейшем при работе с клавиатурой, мышью расположение рук не должно сильно меняться.

Положение ног так же не менее важно. Не стоит загибать их под стул или кресло. Ноги должны быть выпрямлены вперед с упором в твердую поверхность, колени согнуты под углом 90 градусов. Стул или кресло должны обеспечивать прямую осанку, при котором спина немного упирается в спинку.

Так же необходимо ежечасно делать короткий перерыв в работе с компьютером и выполнять небольшую разминку для снятия напряжения в суставах и мышцах. Глаза так же подвержены воздействию, поэтому будет полезно сделать зарядку которая включает в себя круговые движения открытыми глазами, смена точки фокусировки взгляда с близкой до дальней.

Помимо рекомендаций о собственном здоровье, не стоит забывать и о

техники безопасности. Компьютер – это сложное устройство, работающее под напряжением, поэтому на него тоже распространяются техника безопасности к электрическим устройствам. Применимо к компьютеру соблюдайте основные правила:

- не работать с компьютером при наличии внешних повреждений корпуса или изоляции кабелей;
- не стоит включать ПК в розетки без заземления. сами розетки и вилки должны быть цельными и без повреждений;
- не включать ПК в помещении с высокой влажностью;
- провода и кабеля должны быть расположены так, чтобы исключить возможность наступить на них или поставить что-то тяжелое.

При соблюдении всех указанных мер безопасности и рекомендаций отрицательное воздействие на пользователей компьютеров сокращается до минимума.

При выполнении данной работы все выше перечисленные рекомендации были соблюдены.

Заключение

В рамках данной работы была выполнена разработка мобильного приложения для просмотра новостей, что включало в себя все этапы, от анализа требований и проектирования до реализации и тестирования. Проведенная работа позволила достигнуть следующих результатов: были изучены современные тенденции и требования пользователей к мобильным приложениям для новостных сервисов, проведен анализ существующих решений, что позволило выявить сильные и слабые стороны аналогов и определить ключевые функциональные особенности для нашего приложения, проектирование и архитектура приложения, разработана архитектура приложения, включающая основные модули и их взаимодействие, определены технологии и инструменты, используемые для реализации. были созданы пользовательские интерфейсы, ориентированные на удобство и интуитивность использования.

Приложение было реализовано с использованием фреймворка flutter, реализованы основные функции, такие как получение новостных данных из открытого источников, настроек поиска новостей, возможность сохранения избранных статей, удобный просмотр новости. Проведено функциональное тестирование, в результате которого проблем не было обнаружено.

В результате проведенного исследования, созданное мобильное приложение для просмотра новостей полностью соответствует поставленным задачам и требованиям. Оно обладает интуитивно понятным интерфейсом, высокой производительностью и широкими функциональными возможностями, что позволяет удовлетворить потребности современной аудитории в быстром и удобном доступе к актуальной информации.

Таким образом, данная работа демонстрирует возможность успешной реализации мобильного приложения для новостных сервисов, подтверждая важность и актуальность использования мобильных технологий в области

массовой информации. В дальнейшем планируется продолжать развитие приложения, внедряя новые функции и улучшая пользовательский опыт на основе полученных данных и отзывов.

Список использованных источников

1. FluttervsReactNativevsNative: DeepPerformanceComparison. — URL: <https://medium.com/swlh/flutter-vs-react-native-vs-native-deep-performance-comparison-990b90c11433> (датаобращения 07.04.2024).
2. Разработка программного комплекса для многоядерной реконфигурируемой вычислительной системы. Журнал "Современные наукоемкие технологии" № 7 2023, стр. 66-71
3. Медникс З. Программирование под Android. 2-е изд. — СПб.: «Питер», 2013. — С. 345–475.
4. Андрей Алеев. Быстрый старт Flutter-разработчика. — Екб.: «Издательские решения», 2019. — С. 73–102.
5. BLoC. — URL: <https://bloclibrary.dev/#/gettingstarted> (датаобращения 08.04.2024).
6. Гамма Э., Влиссидес Дж., Джонсон Р., Хелм Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования — СПб.: «Питер», 2019. — С. 267–312.
7. Крис Баккет. Dart в действии — Перевод с английского. — М.: «ДМК Пресс», 2016. — С. 135–309.

Приложение А

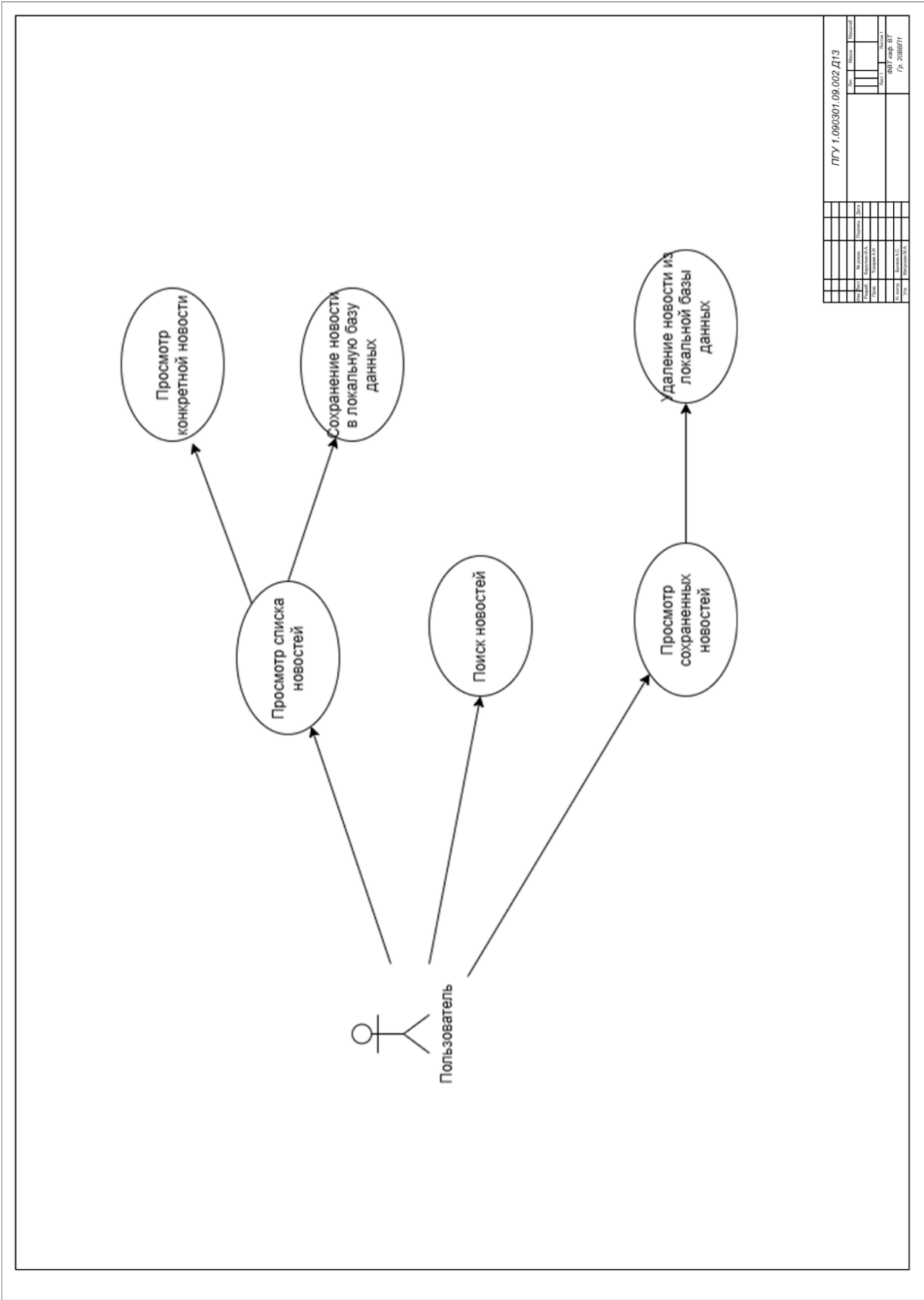
(обязательное)

Мобильное приложение для новостного портала

UML–диаграммы

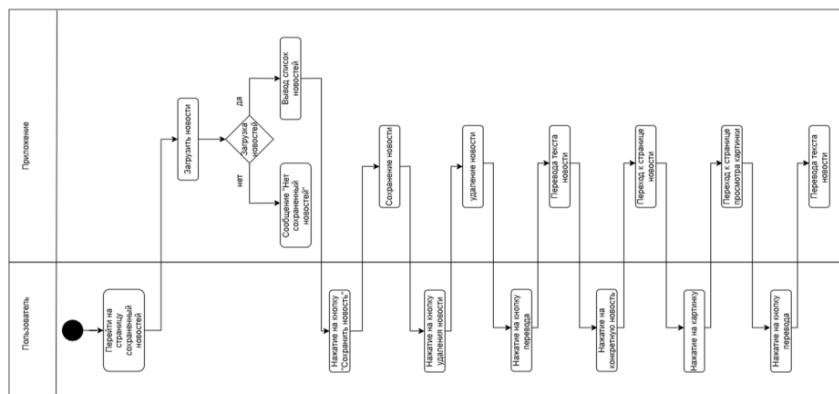
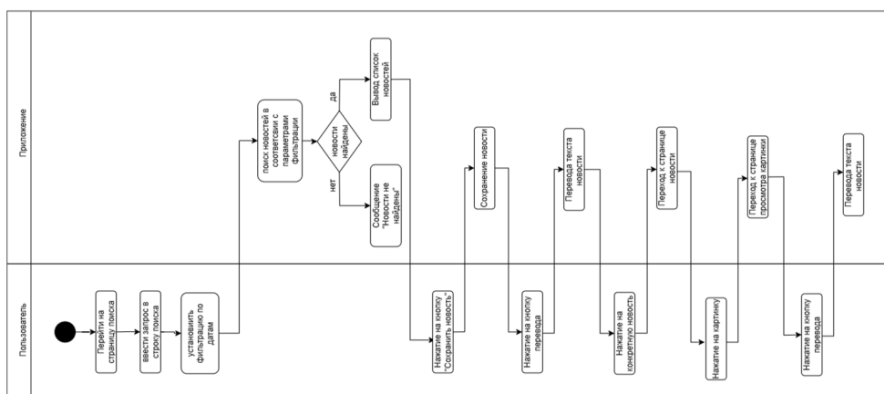
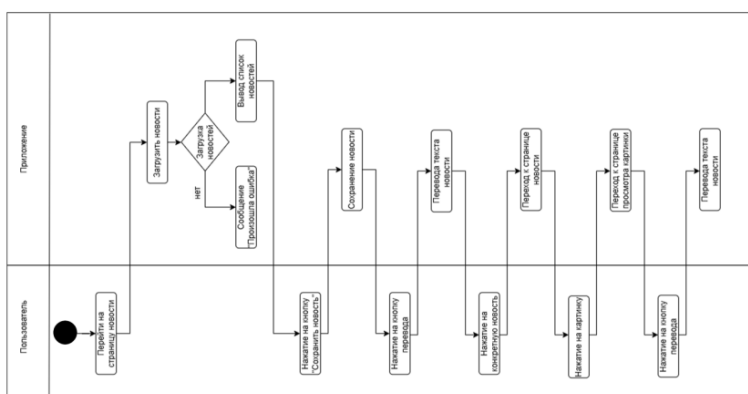
					ПГУ 1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		55

Приложение А.1 – Диаграмма прецедентов

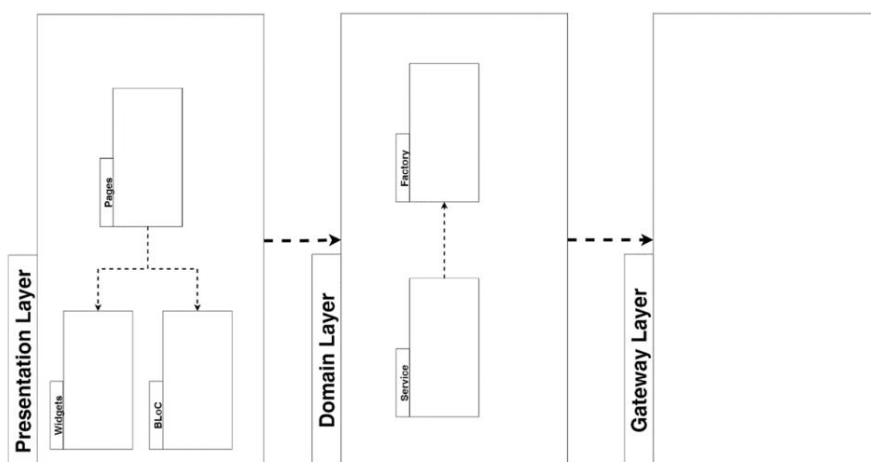


Приложение А.2 – Диаграмма классов

II

[illegible]

Приложение А.4 – Диаграмма пакетов

[illegible]

```

Файл: bottom_navigation.dart

import 'package:flutter/material.dart';

class BottomBarNavigate extends StatefulWidget {
  final int selectedSection;

  final bool isSectionSelected;

  const BottomBarNavigate(
    {super.key, this.isSectionSelected = true, required
    this.selectedSection,});

  @override
  _BottomBarNavigateState createState() =>
  _BottomBarNavigateState();
}

class _BottomBarNavigateState extends State<BottomBarNavigate> {
  @override
  Widget build(BuildContext context) {
    List<BottomNavigationBarItem> _listItems = [
      BottomNavigationBarItem(
        icon: Icon(Icons.newspaper),
        label: '',
      ),
      BottomNavigationBarItem(
        icon: Icon(Icons.search),
        label: '',
      ),
      BottomNavigationBarItem(
        icon: Icon(Icons.document_scanner_outlined),
        label: '',
      ),
    ],
  }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

59

```

];

return SafeArea(
  bottom: true,
  child: Container(
    margin: const EdgeInsets.only(top: 2.0),
    decoration: BoxDecoration(
      boxShadow: <BoxShadow>[
        BoxShadow(
          color: Colors.white,
          spreadRadius: 1,
          blurRadius: 10,
          offset: Offset(0, 0),
        ),
      ],
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.min,
      children: [
        BottomNavigationBar(
          elevation: 0.0,
          items: _listItems,
          type: BottomNavigationBarType.fixed,
          currentIndex: widget.selectedSection >= 0 ?
widget.selectedSection : 0,
          showSelectedLabels: true,
          showUnselectedLabels: true,
          selectedItemColor: widget.isSectionSelected

```

```

        ? Colors.blue
        : Colors.black,
        unselectedItemColor: Colors.black,
        onTap: (selectedSection) =>
            _onItemTapped(context, selectedSection)
    ),
],
),
),
);
}

void _onItemTapped(BuildContext context, int selectedSection)
async {
    if (widget.selectedSection != selectedSection ||
        widget.selectedSection == -1) {
        switch (selectedSection) {
            case 0:
                Navigator.pushNamedAndRemoveUntil(
                    context, '/newsMain', (route) => false);
                break;
            case 1:
                Navigator.pushNamedAndRemoveUntil(
                    context, '/search', (route) => false);
                break;
            case 2:
                Navigator.pushNamedAndRemoveUntil(
                    context, '/save', (route) => false);

```

```

        break;
    }
}
}
}

Файл: db.dart

import 'package:sqflite/sqflite.dart';
import 'dart:async';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';
import 'bloc/news/models/news_models.dart';

class DatabaseProvider {
    DatabaseProvider._();

    static final DatabaseProvider db = DatabaseProvider._();

    static Database? _database;

    Future<Database?> get database async {
        if (_database != null) {
            return _database;
        }

        _database = await initDB();

        return _database;
    }

    initDB() async {
        return await openDatabase(join(await getDatabasesPath(),
«news_app.db»),

        onCreate: (db, version) async {
            await db.execute('')

```

					ПГУ 1.090301.09.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

```

        CREATE TABLE news_items(

            id INTEGER PRIMARY KEY,

            title TEXT,

            bottomTitle TEXT,

            publishedAt TEXT,

            cover TEXT,

            urlToSource TEXT,

            author TEXT,

            reactions INTEGER DEFAULT 0

        )''');

    }, version: 1);

}

Future<void> addNewItem(NewItem newItem) async {

    final db = await database;

    final List<Map<String, dynamic>>? newsItems = await
db?.rawQuery(

        'SELECT id FROM news_items ORDER BY id DESC LIMIT 1',

    );

    int newId = 1;

    if (newsItems != null && newsItems.isNotEmpty) {

        newId = newsItems.first['id'] + 1;

    }

    NewItem newItemTemp = newItem.copyWith(id:newId);

    await db?.insert(

        «news_items»,

        newItemTemp.toMap(),

        conflictAlgorithm: ConflictAlgorithm.replace,

```

```

    );
}

Future<List<NewItem>> getNewsItems() async {

    final db = await database;

    var res = await db?.query(«news_items»);

    List<NewItem> newsItems = [];

    if (res != null && res.isNotEmpty) {

        newsItems = res.map((item) =>
NewItem.fromMap(item)).toList();

    }

    return newsItems;

}

Future<int?> deleteNewsItem(String title) async {

    final db = await database;

    return await db?.delete(«news_items», where: «title = ?»,
whereArgs: [title]);

}

Future<void> updateReactions(int newsItemId, int reactions)
async {

    final db = await database;

    await db?.rawUpdate('''

        UPDATE news_items

        SET reactions = ?

        WHERE id = ?

        ''', [reactions, newsItemId]);

}

}

Файл: depended.dart

```

```

import 'package:get_it/get_it.dart';
import 'package:news_app/service/service.dart';
import 'package:news_app/service/service_impl.dart';
import 'factory/factory.dart';
import 'factory/factory_impl.dart';
import 'gateway/gateway.dart';
import 'gateway/gateway_impl.dart';

setupDependencies() async {
    final serviceLocator = GetIt.instance;

    serviceLocator.registerFactory<NewsGateway>(() =>
NewsGatewayImpl());

    serviceLocator
        .registerFactory<NewsFactory>(() => NewsFactoryImpl());

    serviceLocator.registerFactory<ServiceNews>(() =>
NewsServiceImpl(

        GetIt.instance<NewsGateway>(),

        GetIt.instance<NewsFactory>(),

    ));
}

```

Файл: main.dart

```

import 'package:flutter/material.dart';
import 'package:news_app/pages/main_page.dart';
import 'package:news_app/pages/save_news.dart';
import 'package:news_app/pages/search_page.dart';
import 'db.dart';
import 'depended.dart';

void main() async{
    await setupDependencies();
}

```

```

WidgetsFlutterBinding.ensureInitialized();

runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: '',
      debugShowCheckedModeBanner: false,
      onGenerateRoute: RouteGenerator.generateRoute,
      theme: ThemeData(
        primarySwatch: Colors.grey,
      ),
      home: NewsMainPage(),
    );
  }
}

class RouteGenerator {
  static Route<dynamic> generateRoute(RouteSettings settings) {
    WidgetBuilder builder = (BuildContext context) =>
    Container();

    final args = settings.arguments as Map<String, dynamic>?;

    switch (settings.name) {
      case '/circular':
        builder = (BuildContext context) =>
        CircularProgressIndicator();

        break;
    }
  }
}

```

```

        case '/newsMain':
            builder = (BuildContext context) => NewsMainPage();
            break;
        case '/search':
            builder = (BuildContext context) => NewsSearchPage();
            break;
        case '/save':
            builder = (BuildContext context) => NewsSavePage();
            break;
    }

    return FadePageRoute(
        builder: builder,
        settings: settings,
    );
}

class FadePageRoute extends MaterialPageRoute {
    FadePageRoute(
        {required WidgetBuilder builder, required RouteSettings settings})
        : super(builder: builder, settings: settings);

    @override
    Widget buildTransitions(BuildContext context,
        Animation<double> animation,
        Animation<double> secondaryAnimation, Widget child) {
        return FadeTransition(opacity: animation, child: child);
    }
}

```

```

Файл: bloc.dart

export 'news_bloc.dart';
export 'news_event.dart';
export 'news_state.dart';
export 'models/news_models.dart';

Файл: news_bloc.dart

import 'dart:io';

import 'package:flutter_bloc/flutter_bloc.dart';

class NewsBloc extends Bloc<NewsEvent, NewsState> {

  final ServiceNews _newsService;

  int _currentNewsPage = 1;

  late LoadedState _loadedNewsState;

  NewsBloc(this._newsService) : super(InitialNewsState()) {

    on<LoadEvent>((event, emit) => _loadNews(event, emit));

    on<LoadNameEvent>((event, emit) => _loadNewsOnName(event,
emit));

    on<AddLikeEvent>((event, emit) => _addReaction(event,
emit));

    on<DeleteLikeevent>((event, emit) => _deleteReaction(event,
emit));

    on<EmptyEvent>((event, emit) => _emptySearch(event, emit));

    on<LoadNewsSaveEvent>((event, emit) => _getSave(event,
emit));

  }

  _loadNews(LoadEvent event, Emitter<NewsState> emit) async {

    try {

      emit>LoadingState());

      New news =

        await _newsService.getNews(_currentNewsPage,

```

```

event.perPage ?? 10);

    _loadedNewsState = LoadedState(
        news: news,
    );

    _currentNewsPage++;

    emit(_loadedNewsState);

} on NewsException catch (e) {
emit(ErrorState(message: e.message));
} on SocketException catch (e) {
    emit(ErrorState(message: e.message));
} catch (e) {
    emit(ErrorState());
}
}

_loadNewsOnName(LoadNameEvent event, Emitter<NewsState> emit)
async {
    try {
        late final New news;

        _currentNewsPage = 1;

        emit>LoadingState());

        news = await _newsService.searchNewsOnName(
            _currentNewsPage, 10, event.name,
            firstDate: event.firstDate,
            lastDate: event.lastDate,
            russianNews: event.russianNews,
            searchDate: event.searchDate,
            publisherName: event.publisherName);
    }
}

```

```

        _loadedNewsState = LoadedState(
            news: news,
        );
        _currentNewsPage++;
        emit(_loadedNewsState);
    } on NewsException catch (e) {
emit(ErrorState(message: e.message));
    } on SocketException catch (e) {
        emit(ErrorState(message: e.message));
    } catch (e) {
        emit(ErrorState());
    }
}

_addReaction(AddLikeEvent event, Emitter<NewsState> emit)
async {
    late LoadedState loadedOriginNewsState = _loadedNewsState;
    try {
        List<Emoji> emojiTemp = [];
        emojiTemp = List.from(event.emoji);
        bool isUpdate = false;
        if (emojiTemp.isNotEmpty) {
            isUpdate = true;
        }
        emojiTemp.sort((a, b) => a.id.compareTo(b.id));
       NewItem updatedNewsItem = _loadedNewsState.news.items
            .firstWhere((element) => element.title ==
event.tittle);
        updatedNewsItem =

```

```

        updatedNewsItem.copyWith(emoji: [Emoji(id:
event.idEmojiCurrent)]);

        List<NewItem> updatedItems =
_loadedNewsState.news.items.map((item) {

            if (item.title == event.tittle) {

                return updatedNewsItem;

            } else {

                return item;

            }

        }).toList();

        if (isUpdate)
_newsService.deleteDatabaseItem(updatedNewsItem.title);

        _newsService.addDatabaseItem(updatedNewsItem);

        New news = _loadedNewsState.news.copyWith(items:
updatedItems);

        _loadedNewsState = _loadedNewsState.copyWith(news: news);

        emit(_loadedNewsState);

    } on NewsException catch (e) {

        _loadedNewsState = loadedOriginNewsState;

        emit(_loadedNewsState);

    } on SocketException catch (e) {

        _loadedNewsState = loadedOriginNewsState;

        emit(_loadedNewsState);

    } catch (e) {

        _loadedNewsState = loadedOriginNewsState;

        emit(_loadedNewsState);

    }

}

_deleteReaction(

```

```

DeleteLikeevent event, Emitter<NewsState> emit) async {
late LoadedState loadedOriginNewsState = _loadedNewsState;

try {

    List<Emoji> emojiTemp = [];

    emojiTemp = List.from(event.emoji);

    emojiTemp.sort((a, b) => a.id.compareTo(b.id));

   NewItem updatedNewItem = _loadedNewsState.news.items

        .firstWhere((element)      =>      element.title      ==
event.title);

    updatedNewItem = updatedNewItem.copyWith(emoji: []);

    List<NewItem>                updatedItems                =
_loadedNewsState.news.items.map((item) {

        if (item.title == event.title) {

            return updatedNewItem;

        } else {

            return item;

        }

    }).toList();

    _newsService.deleteDatabaseItem(updatedNewItem.title);

    New    news    =    _loadedNewsState.news.copyWith(items:
updatedItems);

    _loadedNewsState = _loadedNewsState.copyWith(news: news);

    emit(_loadedNewsState);

} on NewsException catch (e) {

    _loadedNewsState = loadedOriginNewsState;

    emit(_loadedNewsState);

} on SocketException catch (e) {

    _loadedNewsState = loadedOriginNewsState;

```

```

        emit(_loadedNewsState);
    } catch (e) {
        _loadedNewsState = loadedOriginNewsState;
        emit(_loadedNewsState);
    }
}

_emptySearch(EmptyEvent event, Emitter<NewsState> emit) {
    New news = _loadedNewsState.news;
    _currentNewsPage = 1;
    news = news.copyWith(items: []);
    emit(EmptyState(news: news));
}

_getSave(LoadNewsSaveEvent event, Emitter<NewsState> emit)
async {
    try {
        emit>LoadingState();

        List<NewItem> newsItem = await
        _newsService.getDatabaseItems();

        _loadedNewsState = LoadedState(
            news: New(items: newsItem),
        );
        emit(_loadedNewsState);
    } on NewsErrorException catch (e) {}
}
}

Файл: news_event.dart

import 'package:equatable/equatable.dart';
import 'models/news_models.dart';

```

```

abstract class NewsEvent extends Equatable {}

class LoadEvent extends NewsEvent {
    final int? perPage;

    LoadEvent({this.perPage});

    @override
    List<Object> get props => [];
}

class LoadNewsSaveEvent extends NewsEvent {
    final int? perPage;

    LoadNewsSaveEvent({this.perPage});

    @override
    List<Object> get props => [];
}

class LoadNameEvent extends NewsEvent {
    final String name;
    final int? channelId;
    final bool isSearchMainNews;
    final String? firstDate;
    final String? lastDate;
    final bool? russianNews;
    final String? publisherName;
    final bool? searchDate;

    LoadNameEvent({
        required this.name,
        this.channelId,
        this.isSearchMainNews = false,
        this.firstDate,

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

74

```

        this.lastDate,

        this.russianNews,

        this.publisherName,

        this.searchDate,

    });

    @override

    List<Object?> get props => [name, channelId, isSearchMainNews,
firstDate, lastDate, russianNews, publisherName, searchDate]; //
firstDate, lastDate, russianNews, publisherName, searchDate
props

}

class AddLikeEvent extends NewsEvent {

    final bool isNewsPage;

    final String tittle;

    final int idEmojiCurrent;

    final List<Emoji> emoji;

    AddLikeEvent({required this.tittle,

        required this.idEmojiCurrent,

        required this.emoji,

        this.isNewsPage=false,

    });

    @override

    List<Object?> get props => [tittle,emoji];

}

class DeleteLikeevent extends NewsEvent {

    final String title;

    final int idEmoji;

    final List<Emoji> emoji;

```

```

DeleteLikeevent({required this.title,
  required this.emoji,
  required this.idEmoji,
});

@override
List<Object?> get props => [title, emoji];
}

class EmptyEvent extends NewsEvent {
  EmptyEvent();

  @override
  List<Object?> get props => [];
}

Файл: news_state.dart

import 'package:equatable/equatable.dart';
import 'models/news_models.dart';

abstract class NewsState extends Equatable {}

class InitialNewsState extends NewsState {
  @override
  List<Object?> get props => [];
}

class LoadingState extends NewsState {
  @override
  List<Object?> get props => [];
}

class LoadedState extends NewsState {
  final New news;

  LoadedState({

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        required this.news,
    });

    LoadedState copyWith({
        New? news,
        int? nextPage,
        bool? isLoadingNextPage,
    }) =>
        LoadedState(
            news: news ?? this.news,
        );

    @override
    List<Object?> get props => [
        news,
    ];
}

class ErrorState extends NewsState {
    final String message;

    ErrorState({
        this.message = '',
    });

    @override
    List<Object?> get props => [message];
}

class EmptyState extends NewsState {
    final New news;

    EmptyState({required this.news});

    @override

```

```

    List<Object?> get props => [];
}

Файл: news_models.dart

import 'package:equatable/equatable.dart';
import 'package:news_app/bloc/news/bloc.dart';

class New extends Equatable {
    final List<NewItem> items;

    New({
        required this.items,
    });

    @override
    List<Object> get props => [
        items,
    ];

    New copyWith({
        int? total,
        int? page,
        int? pageSize,
        int? pageCount,
        List<NewItem>? items,
    }) {
        return New(
            items: items ?? this.items,
        );
    }
}

class NewItem extends Equatable {

```

```

final int id;

final String title;

final String bottomTittle;

final DateTime publishedAt;

final String cover;

final String? urlToSource;

final String author;

final List<Emoji> emoji;

NewItem({
    required this.id,
    required this.title,
    required this.bottomTittle,
    required this.publishedAt,
    required this.cover,
    required this.author,
    required this.emoji,
    required this.urlToSource,
});

@override
List<Object> get props => [
    id,
    title,
    publishedAt,
    cover,
    emoji,
];

NewItem copyWith({

```

```

    int? id,

    String? title,

    String? bottomTittle,

    DateTime? publishedAt,

    String? cover,

    String? author,

    List<Emoji>? emoji,

    String ? urlToSource,

  }) {

    returnNewItem(

      id: id ?? this.id,

      title: title ?? this.title,

      bottomTittle: bottomTittle ?? this.bottomTittle,

      publishedAt: publishedAt ?? this.publishedAt,

      cover: cover ?? this.cover,

      author: author ?? this.author,

      emoji: emoji ?? this.emoji,

      urlToSource: urlToSource,

    );

  }

  Map<String, dynamic> toMap() {

    return {

      'id': id,

      'title': title,

      'bottomTitle': bottomTittle,

      'publishedAt': publishedAt.toIso8601String(),

      'cover': cover,

```

```

        'author': author,

        'urlToSource': urlToSource,

        'reactions':emoji.isNotEmpty? emoji.first.id:0
    };
}

factoryNewItem.fromMap(Map<String, dynamic> map) {
    returnNewItem(
        id: map['id'],
        title: map['title'],
        bottomTittle: map['bottomTitle'],
        publishedAt: DateTime.parse(map['publishedAt']),
        cover: map['cover'],
        author: map['author'],
        emoji:[Emoji(id: map['reactions'],)],
        urlToSource: map['urlToSource'],
    );
}

}

classEmoji extends Equatable {
    final int id;

    Emoji({
        required this.id,
    });

    @override
    List<Object> get props => [
        id,
    ];
}

```

```

Emoji copyWith({
  int? id,
  int? count,
  bool? reactedidUser,
}) {
  return Emoji(id: id??this.id,
  );}
}

```

Файл: factory.dart

```

import 'package:news_app/bloc/news/models/news_models.dart';
import 'package:news_app/models/news.dart';
abstract class NewsFactory {
  Future<New> getNewsModel(NewsResponse response);
}

```

Файл: factory_impl.dart

```

import 'package:news_app/bloc/news/models/news_models.dart';
import '../models/news.dart';
import 'factory.dart';
class NewsFactoryImpl implements NewsFactory {
  @override
  Future<New> getNewsModel(NewsResponse response) async {
    return New(
      items: response.articles.map((e) {
        returnNewItem(
          id:1,
          title: e.title != null ? removeHtmlTags(e.title!)
: '',

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

82

```

        bottomTittle:e.content!=null?
removeHtmlTags(e.content!) : '',

        publishedAt: e.publishedAt != null

            ? DateTime.parse(e.publishedAt!)

            : DateTime.now(),

        cover: e.urlToImage ?? '',

        author:e.author ?? '',

        emoji: [],

        urlToSource: e.url

    );

    })).toList());

}

}

```

Файл: gateway.dart

```

import 'package:news_app/models/news.dart';

abstract class NewsGateway {

    Future<NewsResponse> getNews(int page, int perPage);

    Future<NewsResponse>  getNewsOnName(int page, int perPage,
String name, {

    String? firstDate,

    String? lastDate,

    bool? russianNews,

    String? publisherName,bool? searchDate

    });

}

```

Файл: gateway_impl.dart

```

import 'dart:convert';

import 'package:http/http.dart' as http;

```

```

import 'package:news_app/models/news.dart';
import 'gateway.dart';

class NewsGatewayImpl implements NewsGateway {

  NewsGatewayImpl();

  @override

  Future<NewsResponse> getNews(int page, int perPage) async {

    final uri =
Uri.parse('https://newsapi.org/v2/everything?q=covid&sortBy=publishedAt&apiKey=9cf2279e951d495a96f39cc876f11e1b');

    final response = await http.get(

      uri,

    );

    return NewsResponse.fromJson(jsonDecode(response.body));
  }

  @override

  Future<NewsResponse> getNewsOnName(

    int page,

    int perPage,

    String name, {

      String? firstDate,

      String? lastDate,

      bool? russianNews,

      String? publisherName, bool? searchDate

    }

  ) async {

    String date='';

    if(firstDate!=null &&lastDate!=null &&firstDate== lastDate){

      date='from=$firstDate';

```

```

    }

    if (firstDate != null && lastDate != null && firstDate != lastDate) {
        date = 'from=${firstDate}to=${lastDate}';
    }

    final uri =
Uri.parse('https://newsapi.org/v2/everything?q=«${name}»${date.isNotEmpty?
'&$date':''}${russianNews != null && russianNews ?
'&language=ru':''}${publisherName != null && publisherName.isNotEmpty?
'&domains=$publisherName':''}&sortBy=publishedAt&apiKey=9cf2279e951d495a96f39cc876f11e1b');

    final response = await http.get(
        uri,
    );

    return NewsResponse.fromJson(jsonDecode(response.body));
}
}

```

Файл: news.dart

```

import 'package:serdes_json/serdes_json.dart';
part 'news.g.dart';

@SerdesJson(convertToSnakeCase: true)
class NewsResponseScheme {
    late String status;

    @SerdesJsonField(name: 'totalResults')
    late int totalResults;

    late List<NewsItemResponseScheme> articles;
}

@SerdesJson(convertToSnakeCase: true)
class NewsPaginationResponseScheme {
    late int total;
}

```

```

    late int page;

    late int pageSize;

    late int pageCount;
}

@SerdesJson(convertToSnakeCase: true)
class NewsItemResponseScheme {
    late String? author;

    @SerdesJsonField(name: 'publishedAt')
    late String? publishedAt;

    late String? url;

    @SerdesJsonField(name: 'urlToImage')
    late String? urlToImage;

    late String? title;

    late String? content;
}

@SerdesJson(convertToSnakeCase: true)
class EmojiNewsResponseScheme{
    late int id;

    late int count;

    late bool reactedByUser;
}

Файл: news.g.dart

part of 'news.dart';

class NewsResponse {
    final NewsResponseScheme? $scheme = null;

    final String status;

    final int totalResults;
}

```

```

final List<NewsItemResponse> articles;

NewsResponse({
    required this.status,
    required this.totalResults,
    required this.articles,
});

NewsResponse.fromJson(Map<String, dynamic> json)
    : status = getJsonValue<String>(json, 'status'),
      totalResults = getJsonValue<int>(json, 'totalResults'),
      articles = transformJsonListOfMap<NewsItemResponse,
dynamic>(
    json,
    'articles',
    (dynamic it) =>
        NewsItemResponse.fromJson(it as Map<String,
dynamic>)) {}

static NewsResponse fromStringJson(String json) =>
    NewsResponse.fromJson(jsonDecode(json) as Map<String,
dynamic>);

Map<String, dynamic> toJson() {
    final $result = <String, dynamic>{};

    $result['status'] = status;

    $result['totalResults'] = totalResults;

    $result['articles'] =
        articles.map<dynamic>((NewsItemResponse it) =>
it.toJson()).toList();

    return $result;
}

String toStringJson() => jsonEncode(toJson());

```

```

}

class NewsPaginationResponse {

    final NewsPaginationResponseScheme? $scheme = null;

    final int total;

    final int page;

    final int pageSize;

    final int pageCount;

    NewsPaginationResponse({

        required this.total,

        required this.page,

        required this.pageSize,

        required this.pageCount,

    });

    NewsPaginationResponse.fromJson(Map<String, dynamic> json)

        : total = getJsonValue<int>(json, 'total'),

          page = getJsonValue<int>(json, 'page'),

          pageSize = getJsonValue<int>(json, 'page_size'),

          pageCount = getJsonValue<int>(json, 'page_count') {}

    static NewsPaginationResponse fromStringJson(String json) =>

        NewsPaginationResponse.fromJson(jsonDecode(json)) as

Map<String, dynamic>);

    Map<String, dynamic> toJson() {

        final $result = <String, dynamic>{};

        $result['total'] = total;

        $result['page'] = page;

        $result['page_size'] = pageSize;

        $result['page_count'] = pageCount;

```

```

        return $result;
    }

    String toStringJson() => jsonEncode(toJson());
}

class NewsItemResponse {
    final NewsItemResponseScheme? $scheme = null;
    final String? author;
    final String? publishedAt;
    final String? url;
    final String? urlToImage;
    final String? title;
    final String? content;

    NewsItemResponse({
        this.author,
        this.publishedAt,
        this.url,
        this.urlToImage,
        this.title,
        this.content,
    });

    NewsItemResponse.fromJson(Map<String, dynamic> json)
        : author = getJsonValueOrNull<String>(json, 'author'),
          publishedAt = getJsonValueOrNull<String>(json, 'publishedAt'),
          url = getJsonValueOrNull<String>(json, 'url'),
          urlToImage = getJsonValueOrNull<String>(json, 'urlToImage'),
          title = getJsonValueOrNull<String>(json, 'title'),

```

```

        content = getJsonValueOrNull<String>(json, 'content') {}

    static NewsItemResponse fromStringJson(String json) =>
        NewsItemResponse.fromJson(jsonDecode(json) as Map<String,
dynamic>);

    Map<String, dynamic> toJson() {
        final $result = <String, dynamic>{};

        $result['author'] = author;

        $result['publishedAt'] = publishedAt;

        $result['url'] = url;

        $result['urlToImage'] = urlToImage;

        $result['title'] = title;

        $result['content'] = content;

        return $result;
    }

    String toStringJson() => jsonEncode(toJson());
}

class EmojiNewsResponse {
    final EmojiNewsResponseScheme? $scheme = null;

    final int id;

    final int count;

    final bool reactedByUser;

    EmojiNewsResponse({
        required this.id,

        required this.count,

        required this.reactedByUser,
    });

    EmojiNewsResponse.fromJson(Map<String, dynamic> json)

```

```

        : id = getJsonValue<int>(json, 'id'),
        count = getJsonValue<int>(json, 'count'),
        reactedByUser = getJsonValue<bool>(json,
'reacted_by_user') {}

    static EmojiNewsResponse fromStringJson(String json) =>
        EmojiNewsResponse.fromJson(jsonDecode(json) as Map<String,
dynamic>);

    Map<String, dynamic> toJson() {
        final $result = <String, dynamic>{};
        $result['id'] = id;
        $result['count'] = count;
        $result['reacted_by_user'] = reactedByUser;
        return $result;
    }

    String toStringJson() => jsonEncode(toJson());
}

```

Файл: image_view_page.dart

```

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

class ImageViewPage extends StatefulWidget {
    ImageViewPage({
        required this.imageUrl,
    });

    final String imageUrl;

    @override
    _ImageViewPageState createState() => _ImageViewPageState();
}

class _ImageViewPageState extends State<ImageViewPage> {

```

```

late bool _isView = true;

final navigatorKey = GlobalKey<NavigatorState>();

@override
void initState() {
    super.initState();
}

@override
void dispose() {
    super.dispose();
}

@override
Widget build(BuildContext context) {
    MediaQueryData media = MediaQuery.of(context);

    return Scaffold(
        backgroundColor: Colors.white,
        body: WillPopScope(
            onWillPop: () async {
                return true;
            },
            child: GestureDetector(
                onTap: () {
                    setState(() {
                        _isView = !_isView;
                    });
                },
                child: Stack(
                    children: [

```

```

        AnimatedPositioned(
          duration: Duration(milliseconds: 333),
          curve: Curves.fastOutSlowIn,
          top: 0,
          bottom: 0,
          left: 0,
          right: 0,
          child: InteractiveViewer(
            panEnabled: true,
            minScale: 0.5,
            maxScale: 4,
            child: Image.network(
              widget.imageUrl,
            ),
          ),
        ),
      Positioned(
        top: 0,
        left: 0,
        right: 0,
        child: AnimatedSwitcher(
          duration: const Duration(milliseconds:
300),
          reverseDuration: const
Duration(milliseconds: 300),
          child: !_isView
            ? const SizedBox.shrink()
            : Align(

```

```

alignment: Alignment.topLeft,
child: Container(
  height: media.padding.top +
54,
  width: media.size.width,
  decoration: BoxDecoration(
    color:
Colors.black.withOpacity(0.5),
  ),
  child: Container(
    margin: EdgeInsets.only(
      top:
media.padding.top),
    padding:
EdgeInsets.symmetric(horizontal: 12),
    child: Stack(
      children: [
        Positioned(
          left: 0,
          top: 15,
          child:
GestureDetector(
            onTap: () {
Navigator.pop(context);
            },
            child: Icon(
Icons.arrow_back_ios_new,

```

```

Colors.black,

color:

size: 24,

),

),

),

],

),

),

),

),

),

),

)),

],

)),

),

);

}

}

```

Файл: main_page.dart

```

import 'package:flutter/material.dart';

import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get_it/get_it.dart';
import 'package:news_app/bloc/news/models/news_models.dart';
import 'package:news_app/bloc/news/news_bloc.dart';
import 'package:news_app/bottom_navigation.dart';
import 'package:news_app/service/service.dart';

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

95

```

import 'package:news_app/widgets/item_widget.dart';
import '../bloc/news/news_event.dart';
import '../bloc/news/news_state.dart';
import '../db.dart';

NewsBloc _blocNews(BuildContext context) {
  return BlocProvider.of<NewsBloc>(context);
}

class NewsMainPage extends StatelessWidget {
  const NewsMainPage();

  @override
  Widget build(BuildContext context) {
    return BlocProvider<NewsBloc>(
      create: (context) =>
        NewsBloc(GetIt.instance<ServiceNews>())..add(LoadEvent()),
      child: NewsMainPageFlow());
  }
}

class NewsMainPageFlow extends StatefulWidget {
  const NewsMainPageFlow({super.key});

  @override
  State<NewsMainPageFlow> createState() => _NewsMainPageFlow();
}

class _NewsMainPageFlow extends State<NewsMainPageFlow> {
  bool isLoading = true;
  bool isLoaded = true;
  late New news;

```

```

@override

Widget build(BuildContext context) {

  return BlocConsumer<NewsBloc, NewsState>(listener: (context,
state) {

    if (state is LoadingState) {

      isLoading = true;

    }

    if (state is LoadedState) {

      isLoading = false;

      news = state.news;

      isLoading=true;

    }

    if (state is ErrorState) {

      print(state.message);

    }

  }, builder: (context, state) {

    return Scaffold(

      appBar: AppBar(title: Text(' '),),

      floatingActionButton: FloatingActionButton(

        onPressed: () {

          BlocProvider.of<NewsBloc>(context)

            .add(LoadEvent());

        },

        child: Icon(Icons.refresh,size: 40,color:
Colors.black,),

      ),

      body: Container(

        color:Colors.black12,

```

					ПГУ 1.090301.09.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		97

```

        child: Column(children: [
          if(state is LoadedState)
            Expanded(
              child: ListView.builder(
                itemCount: news.items.length,
                itemBuilder: (context, index) {
                  return Container(
                    key: Key('newsItem$index'),
                    child:
                      NewsWidget(
                        newItem: news.items[index],
                        newsBloc: _blocNews(context),
                      ),
                  );
                },
              ),
            ),
          if (state is LoadingState)
            Container(
              padding: EdgeInsets.all(12),
              margin: EdgeInsets.only(top: 8),
              height: 100,
              child:
                Center(child:
                  CircularProgressIndicator(color: Colors.black,)),
            ),
          if (state is ErrorState)
            Container(

```

```

        padding: EdgeInsets.all(12),
        margin: EdgeInsets.only(top: 8),
        height: 100,
        child: Center(child: Text(' ')),
    ),
  ],
),
bottomNavigationBar: BottomBarNavigate(
  selectedSection: 0,
  isSectionSelected: true,
),
);
});
}
}

```

Файл: news_page.dart

```

import 'dart:ui';
import 'package:flutter/material.dart';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_html/flutter_html.dart';
import 'package:get_it/get_it.dart';
import 'package:intl/intl.dart';
import 'package:translator/translator.dart';
import '../bloc/news/models/news_models.dart';
import '../bloc/news/news_bloc.dart';
import '../bloc/news/news_event.dart';

```

Изм.	Лист	№ докум.	Подпись	Дата

```

import '../bloc/news/news_state.dart';
import '../bottom_navigation.dart';
import '../db.dart';
import '../service/service.dart';
import 'image_view_page.dart';

class NewsPage extends StatelessWidget {
  finalNewItem newsItem;

  const NewsPage({
    required this.newsItem,
  });

  @override
  Widget build(BuildContext context) {
    return BlocProvider<NewsBloc>(
      create: (context) =>
NewsBloc(GetIt.instance<ServiceNews>()),

      child: NewsPageFlow(
        newsItem: newsItem,
      ));
  }
}

class NewsPageFlow extends StatefulWidget {
  finalNewItem newsItem;

  NewsPageFlow({required this.newsItem});

  @override
  State<NewsPageFlow> createState() => _NewsPageFlow(
    newsItem: newsItem,
  );
}

```

```

}

class _NewsPageFlow extends State<NewsPageFlow> {
  final NewItem newsItem;

  ScrollController _scrollController = ScrollController();

  _NewsPageFlow({
    required this.newsItem,
  });

  bool isLoading = true;

  late String tittle;

  late String bottomTittle;

  bool isTrans = false;

  @override
  void initState() {
    super.initState();

    tittle = widget.newsItem.title;

    bottomTittle = widget.newsItem.bottomTittle;
  }

  GoogleTranslator translator = GoogleTranslator();

  Future<void> _launchInBrowser(Uri url) async {
  }

  void trans() {
    if (!isTrans) {
      translator.translate(tittle, to: «ru»).then((value) {
        tittle = value.toString();
      });

      translator.translate(bottomTittle, to: «ru»).then((value)
{

```

```

        setState(() {
            bottomTittle = value.toString();
            isTrans = !isTrans;
        });
    });
} else {
    setState(() {
        tittle = widget.newsItem.title;
        bottomTittle = widget.newsItem.bottomTittle;
isTrans = !isTrans;
        });
    }
}

@override
void dispose() {
super.dispose();
}

@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            title: Container(
                child: Text(' '),
            )),
        body: BlocConsumer<NewsBloc, NewsState>(
            listener: (context, state) {},
            builder: (context, state) {

```



```

        )),
    ),
  ]),
)
: Center(child: CircularProgressIndicator());
}
}),
bottomNavigationBar: BottomBarNavigate(
  selectedSection: -1,
  isSectionSelected: false,
),
);
}
Widget _headerContentNews() {
  return Container(
    color: Colors.white,
    padding: EdgeInsets.only(left: 16, right: 16, bottom: 8),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.min,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        SizedBox(
          width: 8,
        ),
        Flexible(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.min,

```

```

        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.min,
            crossAxisAlignment:
CrossAxisAlignment.start,
            children: [
              Text(
                '\»${widget.newsItem.author}\»',
                style: TextStyle(color: Colors.black),
              ),
              Spacer(),
              InkWell(
                onTap: () {
                  setState(() {
                    trans();
                  });
                },
                child: Icon(
                  Icons.translate,
                  color: Colors.black,
                  size: 20,
                ),
              ),
            ],
          ),
          SizedBox(
            width: 2,
          ),
        ],

```

```

        Row(
            mainAxisAlignment: MainAxisAlignment.min,
            crossAxisAlignment:
CrossAxisAlignment.center,
            children: [
                Text(
                    '',
                    style: TextStyle(color: Colors.black),
                ),
                SizedBox(
                    width: 8,
                ),
                Text(
                    '${formatDateMonthsYearTime(widget.newsItem.publishedAt)}',
                    style: TextStyle(color: Colors.black),
                ),
                Spacer(),
            ],
        ),
    ],
),
),
),
]),
);
}

String formatViewsCount(int? viewsCount) {
    if (viewsCount == null) {

```

```

        return '';
    } else if (viewsCount == 1) {
        return '';
    } else if (viewsCount > 1 && viewsCount < 5) {
        return '';
    } else {
        return '';
    }
}

Widget _tittleContentNews() {
    return Container(
        decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.only(
                bottomLeft: Radius.circular(12.0),
                bottomRight: Radius.circular(12.0),
            ),
        ),
        padding: EdgeInsets.only(top: 8, bottom: 4),
        child: Column(children: [
            Container(
                padding: EdgeInsets.symmetric(horizontal: 16),
                child: Row(children: [
                    Flexible(
                        child: Text(
                            tittle,
                            style: TextStyle(color: Colors.black),

```

```

        ),
    ),
]),
),
SizedBox(
    height: 4,
),
SingleChildScrollView(
    child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
Container(
    color: Colors.white,
    padding:      EdgeInsets.symmetric(horizontal:
9),

    child: Column(
        children: [
            Text(
                bottomTittle,
                style: TextStyle(color: Colors.black),
            ),
        ],
    )),
    CachedNetworkImage(
        imageUrl: newsItem.cover,
        placeholder: (context, str) {

```

```

        return Container(
          child: Center(
            child: SizedBox(
              height: 20,
              child: CircularProgressIndicator(),
            ),
          ),
        ),
      );
    },
    imageBuilder:
      (BuildContext context, ImageProvider
imageProvider) {
      return InkWell(
        onTap: () {
          Navigator.of(context).push(
            MaterialPageRoute(builder:
(BuildContext context) {
              return ImageViewPage(
                imageUrl: newItem.cover,
              );
            }
          ),
        ),
        child: Container(
          margin: EdgeInsets.symmetric(vertical:
12),
          width:
MediaQuery.of(context).size.width,
          child: FittedBox(

```

```

        fit: BoxFit.fill,
        child: Image(image: imageProvider),
      ),
    ),
  );
},
errorWidget: (context, str, error) {
  return SizedBox.shrink();
},
),
SizedBox(
  height: 4,
),
newsItem.urlToSource != null
  ? Container(
    child: InkWell(
      onTap: () {
        _launchInBrowser(Uri.parse(newsItem.urlToSource!));
      },
      child: Row(
        mainAxisAlignment:
MainAxisAlignment.center,
        children: [
          Text(
            ' ',
            style:
Colors.blue),
          TextStyle(color:

```

```

        ),
        SizedBox(
          width: 8,
        ),
        Icon(
          Icons.link,
          size: 20,
          color: Colors.blue,
        ),
      ],
    ),
  ))
: SizedBox.shrink(),
  ])),
  ]));

}

Widget _buildError(BuildContext context, {String errorMessage
= ''}) {
  return Scaffold(
    body: Text(''),
  );
}

String formatDateMonthsYearTime(DateTime dateTime) {
  return DateFormat('dd.MM.yy HH:mm').format(dateTime);
}

Файл: save_news.dart

```

```

import 'package:flutter/material.dart';

import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get_it/get_it.dart';
import 'package:news_app/bloc/news/models/news_models.dart';
import 'package:news_app/bloc/news/news_bloc.dart';
import 'package:news_app/bottom_navigation.dart';
import 'package:news_app/service/service.dart';
import 'package:news_app/widgets/item_widget.dart';
import '../bloc/news/news_event.dart';
import '../bloc/news/news_state.dart';
import '../db.dart';

NewsBloc _blocNews(BuildContext context) {
  return BlocProvider.of<NewsBloc>(context);
}

class NewsSavePage extends StatelessWidget {
  const NewsSavePage();

  @override
  Widget build(BuildContext context) {
    return BlocProvider<NewsBloc>(
      create: (context) =>
        NewsBloc(GetIt.instance<ServiceNews>())..add(LoadNewsSaveEvent()),
      child: NewsSavePageFlow(),
    );
  }
}

class NewsSavePageFlow extends StatefulWidget {

```

```

const NewsSavePageFlow({super.key});

@override
State<NewsSavePageFlow> createState() => _NewsSavePageFlow();
}

class _NewsSavePageFlow extends State<NewsSavePageFlow> {
  bool isLoading = true;
  bool isLoading = true;
  late New news;
  final ScrollController _scrollController = ScrollController();
  @override
  Widget build(BuildContext context) {
    return BlocConsumer<NewsBloc, NewsState>(listener: (context,
state) {
      if (state is LoadingState) {
        isLoading = true;
      }
      if (state is LoadedState) {
        isLoading = false;
        news = state.news;
        isLoading=true;
      }
      if (state is ErrorState) {
        print(state.message);
      }
    }, builder: (context, state) {
      return Scaffold(
        appBar: AppBar(title: Text(' '),),

```

```

body: !isLoading && news.items.isNotEmpty ? Container(
  color:Colors.black12,
  child: Column(children: [
    if(state is LoadedState)
      Expanded(
        child: ListView.builder(
          controller: _scrollController,
          itemCount: news.items.length,
          itemBuilder: (context, index) {
            return Container(
              key: Key('newsItem$index'),
              child:
                NewsWidget(
                  newItem: news.items[index],
                  newsBloc: _blocNews(context),
                ),
            );
          },
        ),
      ),
    if (state is LoadingState)
      Container(
        padding: EdgeInsets.all(12),
        margin: EdgeInsets.only(top: 8),
        height: 100,
        child:
          Center(child:
CircularProgressIndicator(color: Colors.black,)),

```

```

        ),
        if (state is ErrorState)
          Container(
            padding: EdgeInsets.all(12),
            margin: EdgeInsets.only(top: 8),
            height: 100,
            child: Center(child: Text(' ')),
          ),
      ],
    ):Container(child: Center(child: Text(' ', style:
TextStyle(color: Colors.black),),),),
    bottomNavigationBar: BottomBarNavigate(
      selectedSection: 2,
      isSectionSelected: true,
    ),
  );
});
}
}

Файл: search_page.dart

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get_it/get_it.dart';
import 'package:intl/intl.dart';
import '../bloc/news/models/news_models.dart';
import '../bloc/news/news_bloc.dart';
import '../bloc/news/news_event.dart';

```

```

import '../bloc/news/news_state.dart';
import '../bottom_navigation.dart';
import '../service/service.dart';
import '../widgets/item_widget.dart';

NewsBloc _blocNews(BuildContext context) {
  return BlocProvider.of<NewsBloc>(context);
}

class NewsSearchPage extends StatelessWidget {
  const NewsSearchPage();

  @override
  Widget build(BuildContext context) {
    return BlocProvider<NewsBloc>(
      create: (context) => NewsBloc(
        GetIt.instance<ServiceNews>()),
      child: NewsSearchPageFlow());
  }
}

class NewsSearchPageFlow extends StatefulWidget {
  NewsSearchPageFlow();

  @override
  State<NewsSearchPageFlow> createState() =>
    _NewsSearchPageFlow();
}

class _NewsSearchPageFlow extends State<NewsSearchPageFlow> {
  _NewsSearchPageFlow();

  List<NewItem> items = [];

```

```

bool isLoading = false;

bool isEmptySearch = false;

late final TextEditingController _textController;

bool isHasText = false;

String nameSearch = '';

ScrollController _scrollController = ScrollController();

String? firstDate;

String? lastDate;

bool? russianNews;

String? publisherName;

bool? searchDate;

@override

void initState() {
    super.initState();

    _textController = TextEditingController();

    _textController.addListener(() {
        if (_textController.text.isNotEmpty) {
            setState(() {
                isHasText = true;
            });
        } else {
            setState(() {
                isHasText = false;
            });
        }
    });
}

```

```

@override

Widget build(BuildContext context) {

  return Scaffold(

    appBar: AppBar(

      title: Container(

        child: Text(''),

      )),

    body: BlocConsumer<NewsBloc, NewsState>(listener:
(context, state) {

      if (state is LoadingState) {

        isLoading = false;

      }

      if (state is LoadedState) {

        if (state.news.items.isEmpty) {

          isEmptySearch = true;

        }

        items = state.news.items;

        isLoading = true;

      }

      if (state is EmptyState) {

        isLoading = false;

        items = state.news.items;

        isEmptySearch = false;

        nameSearch = '';

        _textController.clear();

      }

    }, builder: (context, state) {

```

```

return Scaffold(
  appBar: AppBar(
    title: Container(
      child: widgetSearch(),
    ),
  body: Container(
    child: !isEmptySearch
      ? Container(
        color: Colors.black12,
        child: Column(children: [
          Expanded(
            child: Container(
              child:
                Column(
                  crossAxisAlignment:
CrossAxisAlignment.center,
                  children: [
                    Expanded(
                      child: ListView.builder(
                        controller:
                          _scrollController,
                        itemCount:
                          items.length,
                        itemBuilder: (context,
                          index) {
                            if (isLoading) {
                              return
                                NewsWidget(
                                  items[index],
                                  newsItem:

```

```

newsBloc:

_blocNews(context),

);

} else {

return

SizedBox.shrink();

))),

),

if (state is LoadingState)

Container(

padding:

margin:

child: Center(

child:

CircularProgressIndicator()),

),

]),

),

),

]),

)

: _buildError()),

bottomNavigationBar: BottomBarNavigate(

selectedSection: 1,

isSectionSelected: true,

),

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

120

```

        );

    }),

);

}

Widget widgetSearch() {
    return Container(
        height: 54,
        padding: EdgeInsets.all(8),
        child: Row(children: [
            Expanded(
                child: TextField(
                    controller: _textController,
                    textAlignVertical: TextAlignVertical.center,
                    maxLines: 1,
                    textInputAction: TextInputAction.search,
                    style: TextStyle(fontSize: 13, color:
Colors.white),
                    decoration: InputDecoration(
                        filled: true,
                        fillColor: Colors.black12,
                        contentPadding:
                            EdgeInsets.symmetric(horizontal: 12,
vertical: 0),
                        enabledBorder: OutlineInputBorder(
                            borderSide: BorderSide(color:
Colors.transparent),
                            borderRadius:
BorderRadius.circular(8)),

```

```

        focusedBorder: OutlineInputBorder(
          borderSide: BorderSide(color:
Colors.transparent),
          borderRadius:
BorderRadius.circular(8)),
        labelText: ' ...',
        floatingLabelBehavior:
FloatingLabelBehavior.never,
        labelStyle: TextStyle(color:
Colors.black45),
        suffixIcon: GestureDetector(
          child: isHasText
            ? Icon(
                Icons.close_rounded,
                color: Colors.black12,
              )
            : Icon(Icons.search, color:
Colors.black12),
          onTap: () {
            if (isHasText) {
              if (items.isNotEmpty) {
                items.clear();
              }
              BlocProvider.of<NewsBloc>(context)
                .add(EmptyEvent());
              isLoading = false;
              setState(() {});
              _textController.clear();
              FocusScopeNode currentFocus =

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

122

```

        FocusScope.of(context);

        if (!currentFocus.hasPrimaryFocus
&&
            currentFocus.focusedChild !=
null) {

            currentFocus.unfocus();

        }

    },

)),

cursorColor: Colors.black12,
onSubmitted: (searchText) {

    nameSearch = searchText;

    if (isEmptySearch) {

        isEmptySearch = false;

    }

    BlocProvider.of<NewsBloc>(context).add(

        LoadNameEvent(

            name: searchText,

            searchDate: searchDate,

            firstDate: firstDate,

            lastDate: lastDate,

            russianNews: russianNews,

            publisherName: publisherName));

    },

),

),

    SizedBox(

```

```

        width: 8,
    ),
    InkWell(
        onTap: () async {
            final Map<String, dynamic> settings = await
showDialog(
            context: context,
            builder: (BuildContext context) {
                return SettingsModal();
            },
        );
        if (settings != null) {
            firstDate = settings['firstDate'];
            lastDate = settings['lastDate'];
            russianNews = settings['russianNews'];
            publisherName = settings['publisherName'];
            searchDate = settings['searchDate'];
        }
    },
    child: Icon(
        Icons.settings,
        color: Colors.black,
        size: 20,
    ),
),
])

);

```

```

    }

    Widget _buildError() {
        return Center(child:Text('      '));
    }
}

class SettingsModal extends StatefulWidget {
    @override
    _SettingsModalState createState() => _SettingsModalState();
}

class _SettingsModalState extends State<SettingsModal> {
    bool _russianNews = false;
    bool _searchDate = false;
    String _publisherName = '';
    DateTime _startDate = DateTime.now();
    DateTime _endDate =
        DateTime.now().add(Duration(days: 7)); //
    Future<void> _selectDate(BuildContext context) async {
        final picked = await showDateRangePicker(
            context: context,
            firstDate: DateTime(2000),
            lastDate: DateTime(2101),
            initialDateRange: _startDate != null && _endDate != null
                ? DateTimeRange(start: _startDate, end: _endDate)
                : null,
        );
        if (picked != null) {
            setState(() {

```

```

        _startDate = picked.start;
        _endDate = picked.end;
    });
}

}

@override
Widget build(BuildContext context) {
    return AlertDialog(
        title: Text(' '),
        content: SingleChildScrollView(
            child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: <Widget>[
                    CheckboxListTile(
                        contentPadding: EdgeInsets.zero,
                        title: Text(' '),
                        value: _searchDate,
                        onChanged: (bool? value) {
                            setState(() {
                                _searchDate = value ?? false;
                            });
                        },
                    ),
                    InkWell(
                        onTap: () {
                            _selectDate(context);
                        },

```

```

        child: Row(
          children: [
            Text(' '),
            Icon(
              Icons.date_range,
              size: 20,
              color: Colors.black,
            )
          ],
        )),
      SizedBox(height: 10),
      Text(
        ' : ${_startDate != null ? DateFormat('yyyy-MM-dd').format(_startDate) : « «}',
      ),
      SizedBox(height: 5),
      Text(
        ' : ${_endDate != null ? DateFormat('yyyy-MM-dd').format(_endDate) : « «}',
      ),
      SizedBox(height: 10),
      CheckboxListTile(
        contentPadding: EdgeInsets.zero,
        title: Text(' '),
        value: _russianNews,
        onChanged: (bool? value) {
          setState(() {
            _russianNews = value ?? false;

```



```

        },
        child: Text(
            '',
            style: TextStyle(fontSize: 16), //
        ),
    ),
),
),
],
);
}
}

Файл: service.dart

import 'package:news_app/bloc/news/models/news_models.dart';

abstract class ServiceNews {

    Future<New> getNews(int page, int perPage);

    Future<New> searchNewsOnName(int page, int perPage, String
name, {

        String? firstDate,

        String? lastDate,

        bool? russianNews,

        String? publisherName, bool? searchDate

    });

    Future<void> addDatabaseItem(NewItem newItem);

    Future<List<NewItem>> getDatabaseItems();           Future<void>
deleteDatabaseItem(String title);

}

class NewsException implements ApiException {

```

```

    final String message;

    NewsException(this.message);
}

class NewsErrorException extends NewsException {

    final String message;

    NewsErrorException(this.message) : super('');
}

abstract class MessageException implements Exception {

    String get message;
}

class ApiException implements MessageException {

    final String message;

    ApiException(this.message);
}

Файл: service_impl.dart

import 'package:news_app/bloc/news/models/news_models.dart';
import 'package:news_app/factory/factory.dart';
import 'package:news_app/gateway/gateway.dart';
import 'package:news_app/models/news.dart';
import '../db.dart';
import 'service.dart';

class NewsServiceImpl implements ServiceNews {

    final NewsGateway _newsGateway;

    final NewsFactory _newsModelsFactory;

    NewsServiceImpl(

        this._newsGateway,

        this._newsModelsFactory,

```

```

);

@Override

Future<New> getNews(int page, int perPage) async {

    try {

        final NewsResponse response = await
_newsGateway.getNews(page, perPage);

        return await _newsModelsFactory.getNewsModel(response);

    } on ApiException catch (e) {

        throw NewsException(e.message);

    } on Exception catch (e) {

        throw NewsException('');

    }

}

@Override

Future<New> searchNewsOnName(int page, int perPage, String
name, {

    String? firstDate,

    String? lastDate,

    bool? russianNews,

    String? publisherName,

    bool? searchDate

}) async {

    try {

        final NewsResponse response = await
_newsGateway.getNewsOnName(

            page,

            perPage,

            name,

```

```

        firstDate: firstDate,

        lastDate: lastDate,

        russianNews: russianNews,

        publisherName: publisherName,

        searchDate:searchDate,

    );

    return await _newsModelsFactory.getNewsModel(response);
} on ApiException catch (e) {
    throw NewsException(e.message);
} on Exception catch (e) {
    throw NewsException('');
}
}

@override
Future<void> addDatabaseItem(NewItem newsItem) async {
    DatabaseProvider.db.addNewsItem(newsItem);
}

@override
Future<List<NewItem>> getDatabaseItems() async {
    return await DatabaseProvider.db.getNewsItems();
}

@override
Future<void> deleteDatabaseItem(String title) async{
    DatabaseProvider.db.deleteNewsItem(title);
}

}

```

```

Файл: item_widget.dart

import 'dart:ui';

import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:news_app/bloc/news/models/news_models.dart';
import 'package:translator/translator.dart';
import '../bloc/news/news_bloc.dart';
import '../pages/news_page.dart';
import 'likes_widget.dart';

class NewsWidget extends StatefulWidget {
  final bool isFirst;
  finalNewItem newsItem;
  final NewsBloc newsBloc;
  const NewsWidget({
    this.isFirst = false,
    required this.newsItem,
    required this.newsBloc,
  });
  @override
  State<NewsWidget> createState() => _NewsWidgetState();
}

class _NewsWidgetState extends State<NewsWidget>
  with AutomaticKeepAliveClientMixin {
  List<Emoji> emojiList = [];
  late String tittle;

```

```

late String bottomTittle;

bool isTrans = false;

@override

void initState() {

    super.initState();

    tittle = widget.newsItem.title;

    bottomTittle = widget.newsItem.bottomTittle;

    emojiList = widget.newsItem.emoji;

}

bool isViewImage=false;

GoogleTranslator translator = GoogleTranslator();

@override

void didUpdateWidget(covariant NewsWidget oldWidget) {

    super.didUpdateWidget(oldWidget);

    if (oldWidget.newsItem.emoji != widget.newsItem.emoji) {

        setState(() {

            emojiList = List.from(widget.newsItem.emoji);

        });

    }

}

void trans() {

    if (!isTrans) {

        translator.translate(tittle, to: «ru»).then((value) {

            tittle = value.toString();

        });

        translator.translate(bottomTittle, to: «ru»).then((value)
{

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

134

```

        setState(() {
            bottomTittle = value.toString();
            isTrans = !isTrans;
        });
    });
} else {
    setState(() {
        tittle = widget.newsItem.title;
        bottomTittle = widget.newsItem.bottomTittle;
        isTrans = !isTrans;
    });
}

@override
bool get wantKeepAlive => true;
late bool isImage = false;

@override
Widget build(BuildContext context) {
    super.build(context);
    return Container(
        margin: !widget.isFirst
            ? EdgeInsets.only(bottom: 8, top: 12)
            : EdgeInsets.only(
                bottom: 8,
            ),
        decoration: BoxDecoration(
            color: Colors.white, //AppColors.white,

```

```

borderRadius: !widget.isFirst
    ? BorderRadius.only(
        topRight: Radius.circular(12),
        topLeft: Radius.circular(12),
        bottomLeft: Radius.circular(12),
        bottomRight: Radius.circular(12),
    )
    : BorderRadius.only(
        bottomLeft: Radius.circular(12),
        bottomRight: Radius.circular(12),
    ),
),
child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
        InkWell(
            onTap: () async {
                Navigator.of(context).push(
                    CupertinoPageRoute(builder: (BuildContext
context) {
                        return NewsPage(
                            newItem: widget.newsItem,
                        );
                    }),
                );
            },
        ),
    ],
),
),

```

```

        child: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            _headerContentNews(),
            _imageContentNews(),
            _tittleContentNews(),
          ],
        )),
      LikesWidget(
        tittle: widget.newsItem.title,
        emoji: emojiList,
        newsBloc: widget.newsBloc,
      ),
    ],
  ));
}

```

```

Widget _headerContentNews() {
  return Container(
    padding: !widget.isFirst
      ? EdgeInsets.symmetric(horizontal: 16, vertical: 8)
      : EdgeInsets.only(left: 16, right: 16, bottom: 8),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.min,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Container(

```

```

        width: 20,
        height: 20,
        decoration: BoxDecoration(
          color: Colors.black,
borderRadius: BorderRadius.all(Radius.circular(10)),
),
        child: Center(
          child: Padding(
            padding: const EdgeInsets.only(bottom: 1),
            child: Text(
getChannelShortName(widget.newsItem.author),
              style: TextStyle(color: Colors.white),
            ),
          ),
        ),
      ),
    SizedBox(
      width: 8,
    ),
    Flexible(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.min,
            crossAxisAlignment:
CrossAxisAlignment.start,
            children: [

```

```

        Text(
          '\>${widget.newsItem.author}\>',
          style: TextStyle(color: Colors.black),
        ),
        Spacer(),
        InkWell(
          onTap: () {
            setState(() {
              trans();
            });
          },
          child: Icon(
            Icons.translate,
            color: Colors.black,
            size: 20,
          ),
        ),
      ]),
    SizedBox(
      width: 2,
    ),
    Row(
      mainAxisAlignment: MainAxisAlignment.min,
      crossAxisAlignment:
CrossAxisAlignment.center,
      children: [
        Text(

```

```

        '',
        style: TextStyle(color: Colors.black),
    ),
    SizedBox(
        width: 8,
    ),
    Text(
        '${formatDateMonthsYearTime(widget.newsItem.publishedAt)}',
        style: TextStyle(color: Colors.black),
    ),
    Spacer(),
],
)
],
),
),
]),
);
}

String getViewsCount(int viewsCount) {
    if (viewsCount == null) {
        return '';
    } else if (viewsCount == 1) {
        return '';
    } else if (viewsCount > 1 && viewsCount < 5) {
        return '';
    }
}

```

```

    } else {
        return '';
    }
}

Widget _imageContentNews() {
    return CachedNetworkImage(
        imageUrl: widget.newsItem.cover,
        placeholder: (context, str) {
            return Container(
                margin: EdgeInsets.all(20),
                child: Center(
                    child: SizedBox(
                        height: 20,
                        width: 20,
                        child: CircularProgressIndicator(),
                    ),
                ),
            );
        },
        imageBuilder: (BuildContext context, ImageProvider
imageProvider) {
            isImage = true;
            return GestureDetector(
                onLongPress:
                () {showBlurredImageOverlay(context,widget.newsItem.cover);},
                child: Container(
                    height: 145,
                    decoration: BoxDecoration(

```

```

        image: DecorationImage(
          image: imageProvider,
          fit: BoxFit.cover,
        ),
      ),
    ));
  },
  errorWidget: (context, str, error) {
    return SizedBox.shrink();
  },
);
}

Widget _tittleContentNews() {
  return Container(
    padding: EdgeInsets.only(left: 16, right: 16, top: 8,
bottom: 4),
    child: Column(children: [
      Row(children: [
        Flexible(
          child: Text(
            tittle,
            maxLines: 3,
            overflow: TextOverflow.ellipsis,
            style: TextStyle(color: Colors.black),
            softWrap: true,
          ),
        ),
      ],
    ),
  ),

```

```

    )),
    SizedBox(
      height: 4,
    ),
    Row(children: [
      Flexible(
        child: Text(
          bottomTittle,
          maxLines: 4,
          overflow: TextOverflow.ellipsis,
          style: TextStyle(color: Colors.black),
          softWrap: true,
        ),
      )
    ]),
  ]),
);
}

void showBlurredImageOverlay(BuildContext context,String url)
{
  OverlayEntry? overlayEntry;

  overlayEntry = OverlayEntry(builder: (context) {
    return Positioned.fill(
      child: GestureDetector(
        onTap: () {
          overlayEntry?.remove();
        },
      ),
    ),
  ),
}

```

```

child: Container(
  color: Colors.black.withOpacity(0.5),
  child: BackdropFilter(
    filter: ImageFilter.blur(sigmaX: 5, sigmaY: 5),
    child: Center(
      child: Container(
        child: ClipRRect(
          child: CachedNetworkImage(
            imageUrl: url,
            placeholder: (context, str) {
              return Container(
                margin: EdgeInsets.all(20),
                child: Center(
                  child: SizedBox(
                    height: 20,
                    width: 20,
                    child:
CircularProgressIndicator(),
                  ),
                ),
              );
            },
            imageBuilder:      (BuildContext context,
ImageProvider imageProvider) {
              return GestureDetector(
                child: Container(
                  decoration: BoxDecoration(
                    image: DecorationImage(

```

```

        image: imageProvider,
        fit: BoxFit.fitWidth,
    ),
),
));

},
errorWidget: (context, str, error) {
    return Text(' ');
},),
)
),
),
),
),
),
),
),
),
),
);
});

Overlay.of(context)?.insert(overlayEntry);
}
}

String getChannelShortName(String name) {
    if (name.isNotEmpty) {
        List<String> shortName = name.split(' ');
        if (shortName.length < 2) {
            if (shortName.first.length < 2) {
                return shortName.first.substring(0, 1).toUpperCase();
            } else {

```

```

        return shortName.first.substring(0, 2).toUpperCase();
    }
} else {
    return shortName[0].substring(0, 1).toUpperCase() +
        shortName[1].substring(0, 1).toUpperCase();
}
} else {
    return '';
}
}

class BlurredBackgroundImage extends StatelessWidget {
    final String imageUrl;
    BlurredBackgroundImage({required this.imageUrl});
    @override
    Widget build(BuildContext context) {
        return Stack(
            fit: StackFit.expand,
            children: [
                ImageFiltered(
                    imageFilter: ImageFilter.blur(sigmaX: 5, sigmaY: 5),
                    child: Image.network(
                        imageUrl,
                        fit: BoxFit.cover,
                    ),
                ),
                Center(
                    child: Container(

```

```

        width: MediaQuery.of(context).size.width * 0.8,
        height: MediaQuery.of(context).size.height * 0.8,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(15),
          boxShadow: [
            BoxShadow(
              color: Colors.black.withOpacity(0.2),
              blurRadius: 10,
              spreadRadius: 5,
            ),
          ],
        ),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(15),
          child: Image.network(
            imageUrl,
            fit: BoxFit.cover,
          ),
        ),
      ),
    ],
  );
}

Файл: likes_widget.dart

import 'dart:async';

```

```

import 'dart:ui';

import 'package:blurrycontainer/blurrycontainer.dart';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter/material.dart';
import 'package:collection/collection.dart';
import '../bloc/news/bloc.dart';

class LikesWidget extends StatefulWidget {

  final bool isNewsPage;

  final String tittle;

  final List<Emoji> emoji;

  final NewsBloc newsBloc;

  const LikesWidget({

    this.isNewsPage = false,

    required this.tittle,

    required this.emoji,

    required this.newsBloc,

  });

  @override

  State<LikesWidget> createState() => _NewsLikesWidgetFlow();

}

class _NewsLikesWidgetFlow extends State<LikesWidget>

  with AutomaticKeepAliveClientMixin {

  GlobalKey _buttonKey = GlobalKey();

  OverlayEntry? overlayEntry;

  late int? idReacted;

  late List<Emoji> emojiTemp = widget.emoji;

  late int countEmoji = 0;

```

```

bool isUpdated = false;

@override
void initState() {
    super.initState();
}

@override
void didUpdateWidget(covariant LikesWidget oldWidget) {
    super.didUpdateWidget(oldWidget);
    if (oldWidget.emoji != widget.emoji) {
        setState(() {
            emojiTemp = List.from(widget.emoji);
        });
    }
}

@override
void dispose() {
    overlayEntry?.remove();
    overlayEntry = null;
    super.dispose();
}

bool get wantKeepAlive => true;

@override
Widget build(BuildContext context) {
    super.build(context);
    return Container(
        padding: EdgeInsets.only(bottom: 8, left: 16, right: 16,
top: 8),

```

```

decoration: widget.isNewsPage

? BoxDecoration(

    color: Colors.white,

    boxShadow: [

        BoxShadow(

            color: Colors.black.withOpacity(0.08),

spreadRadius: 1,

            blurRadius: 3,

            offset: Offset(0, -1),

        ),

    ],

)

: null,

child: Row(children: [

    Spacer(),

    Row(

        children: List.generate(

            emojiTemp.length,

(i) => _likeWidget(

                idEmoji: emojiTemp[i].id,

            ),

        ),

    ),

    SizedBox(

        width: 8,

    ),

    GestureDetector(

```

```

onTap: () {
    if (widget.emoji.any((element) => element.id == 1))
{
    widget.newsBloc.add(DeleteLikeevent(
        title: widget.tittle,
        idEmoji: 1,
        emoji: emojiTemp,
    ));
} else {
    widget.newsBloc.add(AddLikeEvent(
        tittle: widget.tittle,
        idEmojiCurrent: 1,
        emoji: widget.emoji,
        isNewsPage: widget.isNewsPage,
    ));
}
},
onLongPress: () {
    showCustomTooltip(context, widget.emoji);
},

child: Container(
    width: 36,
    height: 36,
    decoration: BoxDecoration(
borderRadius: BorderRadius.circular(18.0),
        border: Border.all(
color: Colors.grey,

```

```

        width: 1.0,
      ),
      color: Colors.white,
    ),
    child: Center(
      child: Text(
        '',
        style: TextStyle(color: Colors.black, fontSize:
20),
        key: _buttonKey,
      ),
    ),
  ),
),
]),
);
}

Widget _likeWidget({
  required int idEmoji,
}) {
  return Container(
    margin: EdgeInsets.only(right: 8),
    padding: EdgeInsets.symmetric(vertical: 4, horizontal: 6),
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(12.0),
      color: Colors.grey,
    ),
  ),

```

```

        child: Row(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            InkWell(
              onTap: () {
                widget.newsBloc.add(DeleteLikeevent(
                  title: widget.tittle,
                  idEmoji: idEmoji,
                  emoji: emojiTemp,
                ));
              },
              child: Row(
                children: [
                  Text(
                    getImage(idEmoji),
                    style: TextStyle(fontSize: 16, color:
Colors.white),
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
    );
  }

  void removeCustomTooltip() {
    overlayEntry?.remove();
  }

```

```

        overlayEntry = null;
    }

    double calculateDigitWidth(
        String digit,
    ) {
        final Size txtSize = _textSize(digit);
        return txtSize.width;
    }

    Size _textSize(
        String text,
    ) {
        final TextPainter textPainter = TextPainter(
            text: TextSpan(text: text),
            maxLines: 1,
            textDirection: TextDirection.ltr)
            ..layout(minWidth: 0, maxWidth: double.infinity);
        return textPainter.size;
    }

    late double widthOfOneDigit;

    void showCustomTooltip(BuildContext context, List<Emoji>
emojiList) {
        RenderBox renderBox =
            _buttonKey.currentContext!.findRenderObject() as
RenderBox;

        Offset buttonPosition =
renderBox.localToGlobal(Offset.zero);

        overlayEntry = OverlayEntry(
            builder: (BuildContext context) => SizedBox(

```

```

width: MediaQuery.of(context).size.width,
height: MediaQuery.of(context).size.height,
child: Stack(
  children: [
    GestureDetector(
      onTap: () {
        removeCustomTooltip();
      },
      onVerticalDragUpdate: (details) {
        removeCustomTooltip();
      },
      onHorizontalDragUpdate: (details) {
        removeCustomTooltip();
      },
      child: ModalBarrier(
        onDismiss: () {
          removeCustomTooltip();
        },
      ),
    ),
    Positioned(
      top: buttonPosition.dy - renderBox.size.height -
170.0,
      left: buttonPosition.dx - 11,
      child: Material(
        color: Colors.transparent,
        child: Stack(children: [

```

```

        BlurryContainer(
            blur: 5,
            color: Colors.transparent,
            padding: EdgeInsets.zero,
            borderRadius:
BorderRadius.all(Radius.circular(18)),
            child: Container(
                decoration: BoxDecoration(
                    borderRadius:
BorderRadius.circular(18.0),
                ),
                height: 16 + 24 * 7,
                width: 48,
            ),
        ),
        Container(
            decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(18.0),
                border: Border.all(
                    color: Colors.grey,
                    width: 1.0,
                ),
            ),
            padding: EdgeInsets.all(8),
            child: Column(
                children: List.generate(
                    5,
                    (i) => Container(

```

const

```

padding: i < 5 - 1
      ? EdgeInsets.only(bottom: 8)
      : EdgeInsets.only(bottom:
40),

child: LiftedImageContainer(
  idEmoji: i + 1,
  onTap: () {

widget.newsBloc.add(AddLikeEvent(

      tittle: widget.tittle,
      idEmojiCurrent: i + 1,
      emoji: emojiList,
      isNewPage:
widget.isNewsPage,

      ));
    },
  )),
),
]),
),
),
],
),
),
);

Overlay.of(context)!.insert(overlayEntry!);
}
}

```

```

class LiftedImageContainer extends StatefulWidget {
    final int idEmoji;

    final Function onTap;

    LiftedImageContainer({
        required this.idEmoji,
        required this.onTap,
    });

    @override
    _LiftedImageContainerState createState() =>
    _LiftedImageContainerState();
}

class _LiftedImageContainerState extends
State<LiftedImageContainer> {
    bool isPressed = false;

    @override
    Widget build(BuildContext context) {
        return GestureDetector(
            onTapDown: (_) {
                setState(() {
                    isPressed = true;
                });
                widget.onTap.call();
            },
            onTapUp: (_) {
                Future.delayed(Duration(milliseconds: 250), () {
                    setState(() {
                        isPressed = false;
                    });
                });
            },
        );
    }
}

```

```

        });
    },
    onTapCancel: () {
        Future.delayed(Duration(milliseconds: 250), () {
            setState(() {
                isPressed = false;
            });
        });
    },
    child: AnimatedContainer(
        duration: Duration(milliseconds: 100),
        transform: Matrix4.translationValues(0.0, isPressed ? -
6.0 : 0.0, 0.0),
        child: Text(
            getImage(widget.idEmoji),
            style: TextStyle(fontSize: 24, color: Colors.white),
        ),
    ),
);
}
}

```

Приложение В

(обязательное)

Мобильное приложение для новостного портала

Презентация

					ПГУ 1.090301.09.001 ПЗ	Лист
						160
Изм.	Лист	№ докум.	Подпись	Дата		

Мобильное приложение для новостного портала

Автор работы: студент группы 20ВВП1 Кирюткин И.А.

Научный руководитель: к.т.н., доцент кафедры ВТ Токарев А.Н.

Актуальность проекта

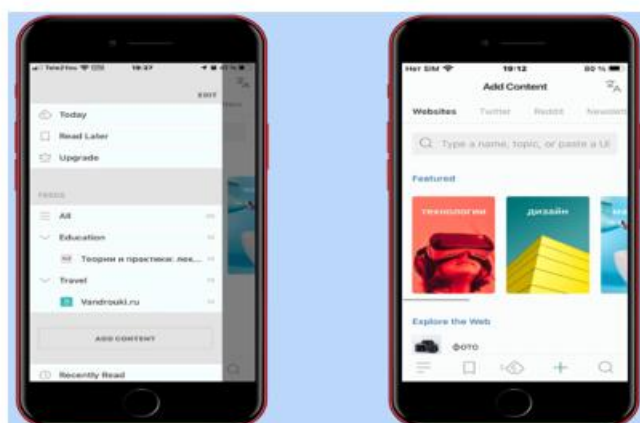
В эпоху технологий люди имеют возможность собирать информацию с любой точки мира. Интернет, социальные сети, мобильные приложения – это источники которые упрощают жизнь человека и позволяют быть в курсе любых событий, не выходя из дома. Новости имеют огромное влияние на общество в современном мире. На данном этапе это не только источник информации, но и одно из средств общения и выражения мнения. Современный человек с помощью новостей может более подробно и целостей понимать, что именно ждет его в будущем.

В эпоху технологий люди имеют возможность собирать информацию с любой точки мира. Интернет, социальные сети, мобильные приложения – это источники которые упрощают жизнь человека и позволяют быть в курсе любых событий, не выходя из дома. Новости имеют огромное влияние на общество в современном мире. На данном этапе это не только источник информации, но и одно из средств общения и выражения мнения. Современный человек с помощью новостей может более подробно и целостей понимать, что именно ждет его в будущем.

Цели работы

- изучение современных инструментов для создания мобильных приложений;
- изучение компонентов для создания мобильного ПО;
- разработка приложения, которое позволит просматривать новости.

Приложение “Feedly”



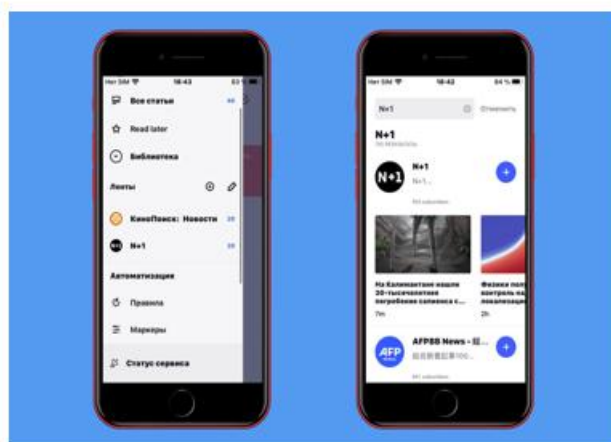
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

162

Приложение “Inoreader”



Постановка задачи

Приложение должно работать на операционных системах Android и IOS.

Обладать современным, интуитивно понятным интерфейсом.

Необходимый функционал: просмотр ленты новостей, просмотр конкретной новости, поиск, сохранение новостей на устройство, механизм перевод текста на русский язык.

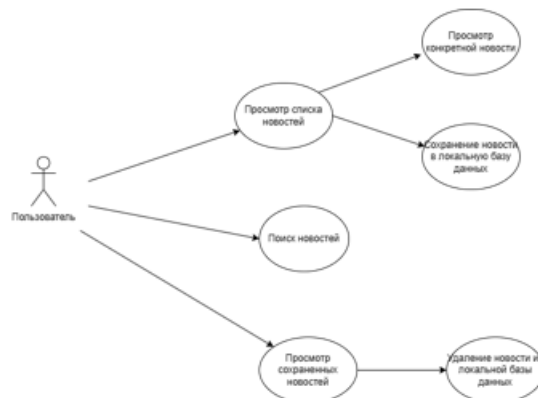
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

Лист

163

UML-диаграмма прецедентов



Flutter и React Native



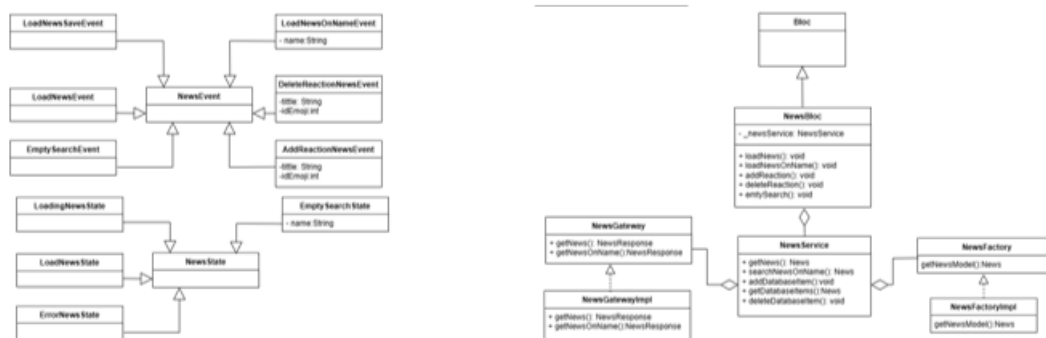
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.09.001 ПЗ

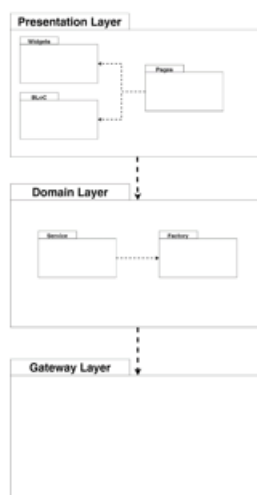
Лист

164

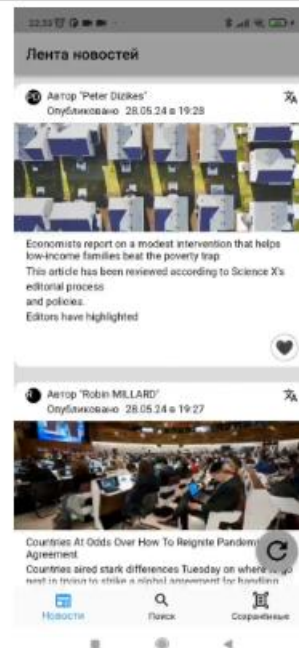
UML- диаграмма классов



UML- диаграмма пакетов



Работа приложения



UML- диаграммы деятельности

Диаграмма просмотр ленты



Диаграмма поиска



Диаграмма просмотр сохраненных новостей



Заключение

В результате проведенного исследования, созданное мобильное приложение для просмотра новостей полностью соответствует поставленным задачам и требованиям.

Оно обладает интуитивно понятным интерфейсом, высокой производительностью и широкими функциональными возможностями, что позволяет удовлетворить потребности современной аудитории в быстром и удобном доступе к актуальной информации.

Приложение было реализовано с использованием фреймворка *Flutter*, реализованы основные функции, такие как получение новостей из открытого источника, поиска, сохранения новостей, просмотр конкретной новости.

Проведено функциональное тестирование, в результате которого проблем не было обнаружено.

Таким образом, данная работа демонстрирует возможность успешной реализации мобильного приложения для новостных сервисов, подтверждая важность и актуальность использования мобильных технологий в области массовой информации. В дальнейшем планируется продолжать развитие приложения, внедряя новые функции и улучшая пользовательский опыт на основе полученных данных и отзывов.

Спасибо за внимание!