**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ**

**ГОУВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**ФАКУЛЬТЕТ** Компьютерных наук и технологий\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** Программной инженерии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**НАПРАВЛЕНИЕ**  Программная инженерия\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ПРОФИЛЬ** Инженерия программного обеспечения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка проекта: \_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Подпись расшифровка подписи

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Подпись расшифровка подписи

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

ППодпись расшифровка подписи

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту по дисциплине**

**«Архитектура и проектирование программного обеспечения»**

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

должность, звание подпись расшифровка подписи

Нормоконтролер

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

должность, звание подпись расшифровка подписи

Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

номер группы подпись расшифровка подписи

Донецк – 2018

**РЕФЕРАТ**

Тема курсового проекта: Разработка мобильного приложения «Новостной портал» на платформе Android.

В состав проекта входят:

пояснительная записка 50 страниц, 20 рисунков, 10 таблиц,

5 библиографических названий, 2 приложений.

В курсовом проекте рассмотрен процесс проектирования программного обеспечения для системы «Мобильное приложение новостного портала».

Проведен анализ предметной области и сформированы требования к разрабатываемой программной системе. Исходя из требований (атрибутов качества), выбрана архитектура системы с применением системных паттернов. Сделан обзор методологий разработки программного обеспечения, языков описания архитектур программных систем и применяемых видов. Выполнены этапы проектирования системы, включая разработку в CASE-среде различных моделей на языке UML.

С учётом типа приложения выбраны инструментальные средства разработки программного обеспечения: среда разработки Microsoft Visual Studio, среда разработки Android Studio, языки программирования Java и C#. При разложении системы на объекты использованы паттерны проектирования. Создано программное обеспечение, обладающее следующими функциональными характеристиками: (*указать функциональные возможности реализованного программного кода*).

Область применения системы: (*указать возможные области применения полученных результатов*).

ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ НОВОСТНОГО ПОРТАЛА, ПРОГРАММНАЯ СИСТЕМА, АРХИТЕКТУРА, МОДЕЛИ UML, ПАТЕРНЫ ПРОЕКТИРОВАНИЯ, ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММ

Содержание

[Введение 4](#_Toc533271502)

[1 Анализ объекта автоматизации 5](#_Toc533271503)

[1.1 Назначение системы и задачи автоматизации 5](#_Toc533271504)

[1.2 Формирование и анализ требований 5](#_Toc533271505)

[1.3 Задачи архитектуры программного обеспечения 5](#_Toc533271506)

[2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СИСТЕМЫ 6](#_Toc533271507)

[2.1 Анализ ключевых вопросов проектирования 6](#_Toc533271508)

[2.2 Стратегии и методы проектирования ПО 6](#_Toc533271509)

[2.3 Определение целей архитектуры 6](#_Toc533271510)

[2.4 Основные сценарии 6](#_Toc533271511)

[2.5 Тип приложения 6](#_Toc533271512)

[2.6 Определение ограничений развертывания 6](#_Toc533271513)

[2.7 Выбор архитектурного стиля проектирования 6](#_Toc533271514)

[2.8 Графическое представление архитектуры на языке UML 6](#_Toc533271515)

[2.9 Детализация программ с помощью паттернов проектирования 6](#_Toc533271516)

[2.10 Проектирование пользовательского интерфейса 6](#_Toc533271517)

[2.11 Выбор инструментальных средств разработки системы 6](#_Toc533271518)

[2.12 Программная реализация базовых модулей системы (код?) 7](#_Toc533271519)

[2.13 Анализ качества и оценка программного дизайна (КАК? Метриками?) 7](#_Toc533271520)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МОДУЛЯ ПРОЕКТА 8](#_Toc533271521)

[3.1 Описание используемых паттернов 8](#_Toc533271522)

[3.2 Структура используемых паттернов 8](#_Toc533271523)

[3.3 Программный код паттернов 8](#_Toc533271524)

[Заключение 9](#_Toc533271525)

[Список литературы 10](#_Toc533271526)

[Приложение 1 11](#_Toc533271527)

[Приложение 2 12](#_Toc533271528)

[Приложение 3 13](#_Toc533271529)

# Введение

Хорошая архитектура - это прежде всего выгодная архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным. Программу с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать. То есть, на самом деле можно сформулировать список вполне разумных и универсальных критериев:

1. Эффективность системы. В первую очередь программа, конечно же, должна решать поставленные задачи и хорошо выполнять свои функции, причем в различных условиях. Сюда можно отнести такие характеристики, как надежность, безопасность, производительность, способность справляться с увеличением нагрузки (масштабируемость) и т.п.
2. Гибкость системы. Любое приложение приходится менять со временем — изменяются требования, добавляются новые. Чем быстрее и удобнее можно внести изменения в существующий функционал, чем меньше проблем и ошибок это вызовет — тем гибче и конкурентоспособнее система. Поэтому в процессе разработки нужно стараться оценивать то, что получается, на предмет того, как это потом, возможно, придется менять. Изменение одного фрагмента системы не должно влиять на ее другие фрагменты. По возможности, архитектурные решения не должны быть монолитными, и последствия архитектурных ошибок должны быть в разумной степени ограничены. "Хорошая архитектура позволяет откладывать принятие ключевых решений" (Боб Мартин) и минимизирует «цену» ошибок.
3. Расширяемость системы. Возможность добавлять в систему новые сущности и функции, не нарушая ее основной структуры. На начальном этапе в систему имеет смысл закладывать лишь основной и самый необходимый функционал (принцип YAGNI — you ain’t gonna need it, «Вам это не понадобится») Но при этом архитектура должна позволять легко наращивать дополнительный функционал по мере необходимости. Причем так, чтобы внесение наиболее вероятных изменений требовало наименьших усилии.

Это означает, что приложение следует проектировать так, чтобы изменение его поведения и добавление новой функциональности достигалось бы за счет написания нового кода (расширения), и при этом не приходилось бы менять уже существующий код. В таком случае появление новых требований не повлечет за собой модификацию существующей логики, а сможет быть реализовано прежде всего за счет ее расширения.

# 1 Анализ объекта автоматизации

## Назначение системы и задачи автоматизации

Еще пару десятков лет назад люди узнавали новости из газет, которые нужно было покупать в киосках, затем с появлением телевидения — ждать выпуск новостей. С развитием компьютерных технологий новости стали доступны 24 часа в сутки 7 дней в неделю. Но прогресс на этом не остановился. Сегодня, чтобы быть в курсе событий, которые происходят в мире, в стране или в одном населенном пункте, достаточно мобильного телефона с установленным на нем новостным приложением, транслирующим новости в постоянном режиме.

В новостном приложении публикуются новости о событиях, происходящих в городе или в стране. Люди всегда хотят быть в курсе событий, а любую свободную минуту современный человек проводит в различных приложениях, установленных на его смартфон, поэтому новостное приложение будет популярным и крайне востребованным у пользователей.

Формируются такие задачи:

1. Необходим мгновенный доступ к новым событиям;
2. Нужен быстрый доступ к новостям в интересной категории;
3. Нужна возможность быстро просмотреть выбранную новость.

На данный момент существуют новостные сайты, новостные приложения, новости также показывают по телевизору и передают по радио. Основная проблема – быстрый доступ к информации, причем именно к той, которая нужна прямо сейчас. Несмотря на то, что люди привыкли к газетам и телевидению, эти способы не дают позволяют информацию мгновенно: выпуска газеты нужно дождаться, а затем еще и купить в киоске, телевизионная передача также передается по расписанию. Кроме того, проблему доступа к выбранной информации телевидение и газеты не могут решить, т.к. в газете не может быть много материала, а телевизионную программу нельзя промотать до нужного момента.

С появлением Интернета появились и новостные сайты, теперь люди получают информацию еще быстрее, могут выбирать интересные категории, подписываться на новости. Пользователь также может подписаться на новостной канал, и получать письма-рассылки на электронную почту. Тем не менее, для получения доступа к любой новости нужен постоянный доступ в Интернет, а это не всегда возможно. Не позволяет также и сравнить информацию из разных источников, т.к. новостной сайт – это один источник.

Все эти проблемы успешно решают мобильные приложения-клиенты, которые работают по принципу «спроси у пользователя, что он хочет, и дай ему это, сразу же». Преимущества мобильный приложений перед новостными сайтами:

1. Приложение позволяет выбрать источник информации;
2. Приложение позволяет использовать сохраненные данные, что дает возможность просмотра новостей даже без подключения к Интернету;
3. Приложение позволяет за один раз загружать меньше данных, чем вся страница сайта, что позволяет ускорить доступ к новостям.

Единственный, незначительный, недостаток мобильного приложения – это еще один уровень между данными и пользователем, в теории это уменьшает надежность системы. Тем не менее, такое происходит крайне редко, к тому же не убирает доступность других источников, например новостных сайтов.

## Формирование и анализ требований

Целевой аудитория приложения крайне широка, охватывает почти всех людей, т.к. новости читают вне зависимости от пола и возраста.

На основании опросов потенциальных пользователей и заказчика, а также проанализировав успешные аналоги, сформированы следующие требования.

### Требования к дизайну приложения

При разработке должны быть использованы преимущественно светлые и контрастные цветовые решения.

Оформление должно быть разработано в достаточно консервативном ключе.

На страницах не должно быть большого объема текста.

В дизайне приложения не должны присутствовать:

* мелькающие баннеры;
* много сливающегося текста;
* резкая анимация.

### Функциональные требования

Пользователь должен иметь возможность быстро получить доступ к любой странице новостного сайта.

Пользователь должен иметь возможность просмотреть уже загруженную новость без подключения к интернету.

### Требования к информационному обеспечению

Требования к хранению данных:

Все данные сайта должны храниться в структурированном виде под управлением реляционной СУБД. Исключения составляют файлы данных, предназначенные для просмотра и скачивания (изображения, видео, документы и т.п.). Такие файлы сохраняются в файловой системе, а в БД размещаются ссылки на них.

### Требования к представлению приложения

Меню должно быть боковое, появляющееся по слайду пальцем, а также верхнее, где пунты меню также сменяются по слайду пальцем.

### Анализ требований

Требованиие: Пользователь должен иметь возможность быстро получить доступ к любой странице новостного сайта.

Изменение: Пользователь должен иметь возможность получить доступ к любой странице новостного сайта в течение одоной секунды.

Требование: При разработке должны быть использованы преимущественно светлые и контрастные цветовые решения.

Дополнение:

Пример:

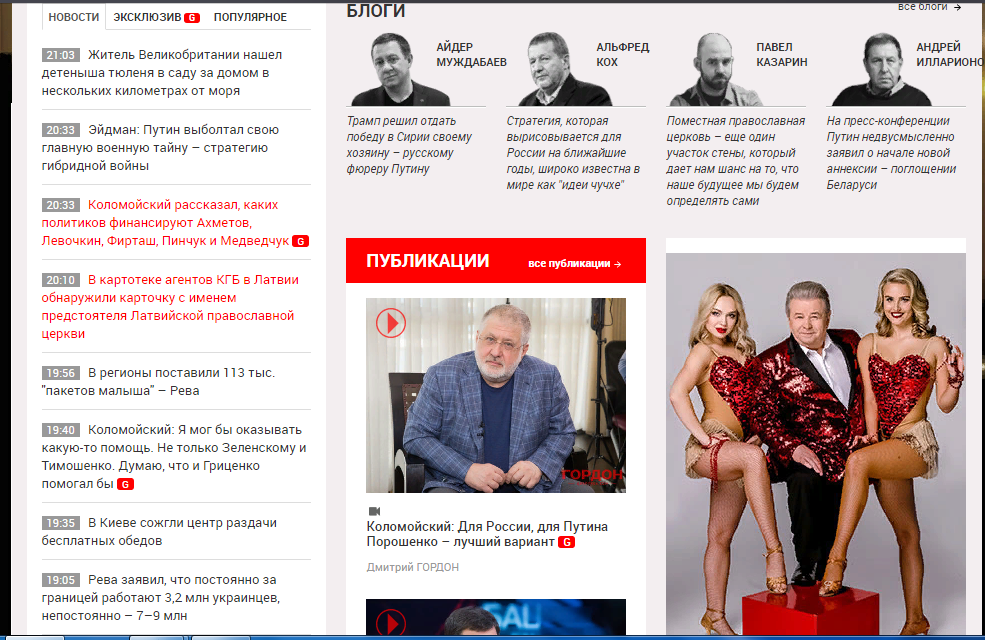


Рисунок 1 – пример цветового оформления

Требование: На странице не должно быть большого объема текста.

Дополнение:

Пример:



Рисунок 2- пример оформления текста

## Задачи архитектуры программного обеспечения

Разрабатываемая архитектура должна реализовывать следующие поставленные заказчиком задачи:

* *Улучшение и повышение продуктивности процессов получения информации*. Это уменьшение времени на просмотр новости по сравнению с переходом по ссылке, ускорение времени обновления новостной ленты по сравнению с полной загрузкой страницы браузера.
* *Небольшая стоимость поддержки жизненного цикла приложения*. Это подразумевает легкость добавления новых разделов в приложение и удаление или изменение старых.

# 2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ СИСТЕМЫ

## Анализ ключевых вопросов проектирования

* *Параллелизм*. Параллельные вычисления в чистом виде использоваться не будут, чтобы не перегружать приложение фоновыми процессами.
* *Асинхронные процессы*. Приложение будет использовать асинхронные задачи, чтобы обновлять интерфейс без блокировки самого интерфейса, а также для выполнения запросов к серверу.
* *Обеспечение отказоустойчивости*. Для критических ошибок, напирмер, отсутствие интернета при запросе обновления, будут использоваться исключения. Для некритических ошибок, например, не удалось получить новость, будут использоваться коды ошибок, т.к. некритические ошибки не должны замедлять работу программы.

## Стратегия и методы проектирования программы

Почти все стратегии проектирования программ преследуют одну цель – «разделяй и властвуй». Целью проектирования является определение внутренних свойств системы и детализации её внешних (видимых) свойств на основе выданных заказчиком требований к ПО (исходные условия задачи). Эти требования подвергаются анализу. Первоначально программа рассматривается как «чёрный ящик». Ход процесса проектирования и его результаты зависят не только от состава требований, но и выбранной модели процесса, опыта проектировщика. В данной работе используется структурное проектирование «сверху вниз», т.к. позволяет постепенно уточнять архитектуру, при этом не изменяя ее, другими словами, постепенно раскрывая этот «черный ящик».

## Определение целей архитектуры

* *Начальное определение задач архитектуры*. В рамках этой работы создается прототип архитектуры приложения, который, несомненно, требует доработок, уточнений и изменений по ходу реализации.
* *Определение потребителей архитектуры*. Разрабатываемая архитекутра, предположительно, должна использоваться другими архитекторами для доработки, а также разработчиками и их руководителями для реализации общих концепций.

## Основные сценарии

Заинтересованные лица:

1. Пользователь – любой человек, который интересуется новостями. Любого пола, национальности и цвета кожи. Из пользователей состоит вся аудитория приложения.
2. Заказчик – организация/владелец сайта/новостного портала, для которого создается приложение.

Действующие лица:

1. Пользователь – пользователь приложения
2. Новостное приложение (приложение) – проектируемое приложение.

Сценарии использования:

|  |  |  |
| --- | --- | --- |
| **Номер** | **Пользователь** | **Приложение** |
| 1 | Открывает приложение | Открывает последнюю просмотренную пользователем категорию |
| 2 | Листает вниз | Прокручивает список новостей |
| 3 | Листает вниз до конца списка новостей | Загружает список новостей из интернета |

Таблица 2.1 – сценарий использования

|  |  |  |
| --- | --- | --- |
| **Номер** | **Пользователь** | **Приложение** |
| 1 | Открывает приложение | Открывает последнюю просмотренную пользователем категорию |
| 2 | Обновляет категорию | Загружает последние новости в этой категории из Интернета |
| 3 | Отключается от Интернета | Выдает уведомление «работа в оффлайне» |

Таблица 2.2 – сценарий использования

|  |  |  |
| --- | --- | --- |
| **Номер** | **Пользователь** | **Приложение** |
| 1 | Открывает приложение | Открывает последнюю просмотренную пользователем категорию |
| 3 | Выбирает новость | Показывает выбранную новость |
| 5 | Отключается от Интернета | Выдает уведомление «работа в оффлайне» |
| 7 | Возвращается к ленте новостей | Отображает сохраненный список новостей |
| 8 | Открывает недавно просмотренную новость | Открывает новость из данных кэша |

Таблица 2.3 – сценарий использования

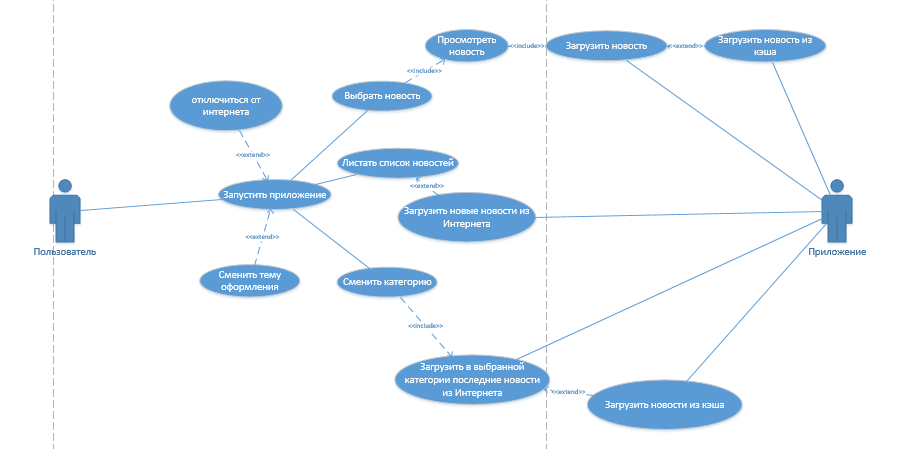


Рисунок 2.1- диаграмма сценариев использования

## Тип приложения

В соответствии с выданным заданием, приложение будет мобильным приложением для системы Android. Особенность таких приложений – небольшое разрешение экрана, небольшой размер оперативной памяти, сравнительно небольшая вычислительная мощность, а также ввод данных преимущественно с помощью касаний и жестов. Меню и другая информация представляется в основном в виде списков на всю ширину экрана.

## Определение ограничений развертывания

В общем, приложение должно поддерживать широкий круг устройств и программного обеспечения, не только самые современные.

Ограничений по безопасности нет, т.к. новости – свободно распространяемая информация, и приложение не должно передавать никаких конфиденциальных данных.

Ограничений собственно по развертыванию нет, выбор средств оставлен разработчикам.

### Ограничения по аппаратным требованиям

* **Процессор уровня 1.5GHz Qualcomm Snapdragon 2012 года выпуска;**
* **Диагональ экрана от 4.3 дюйма;**
* **Оперативная память от 512 мб.**

### Ограничения по программным требованиям

* ОС устройства - Android от 4.4 Kitcat и позднее;
* Необходимо создать удобное и быстрое API для доступа к материалам сайта.

## Выбор архитектурного стиля проектирования

Приложение представляет собой классическое клиенткое приложение. В связи с этим, было решено выбрать паттерн Клиент-сервер.

Приложение как клиент подключается к серверу баз данных сайта, берет оттуда информацию и выводит ее на экран.

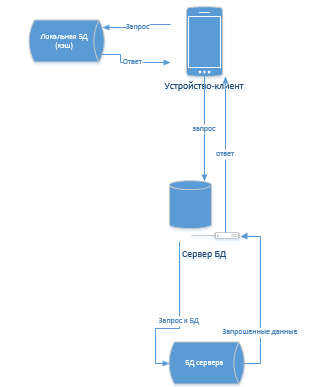


Рисунок 2.2 – схема арзитектуры приложения

## Графическое представление архитектуры на языке UML

## Детализация программ с помощью паттернов проектирования

Для разделения слоя представления и слоя доступа к данным было решено использовать архитектурный паттертн MVC (Model View Controller), который преобразован в MVP (Model-View-Presenter).

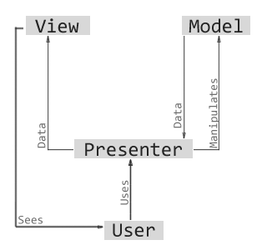


Рисунок 2.3 – схема MVP

Для реализации «подписки» на изменение локальной БД и обновления интерфейса взаимодействие Presenter и Model), использован поведенческий паттерн Observer(Наблюдатель), где Presenter «подписывался» на изменение модели, и в случае изменений, обновлял интерфейс.

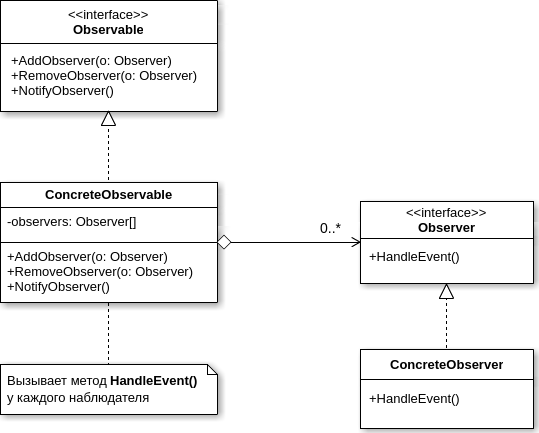


Рисунок 2.4 – схема паттерна Observer

## Проектирование пользовательского интерфейса

Было решено за основу взять приложение издательства «Гордон», интерфейс которого удовлетворяет всем требованиям к интерфейсу проетируемого приложения.

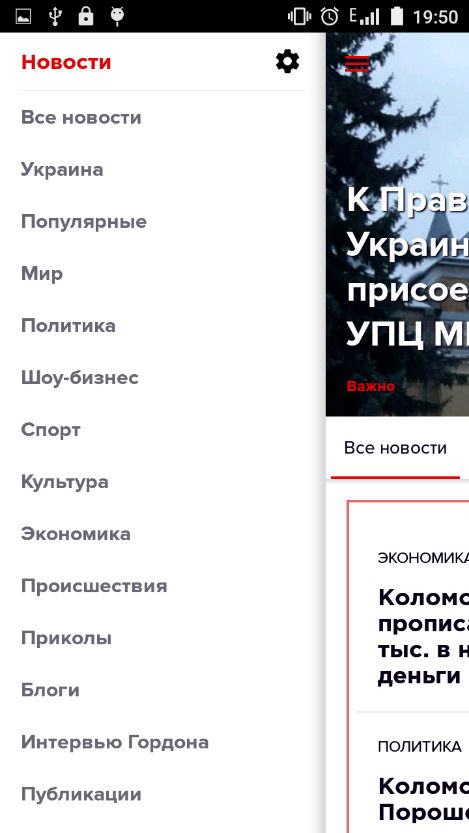


Рисунок 2.5 – боковое меню

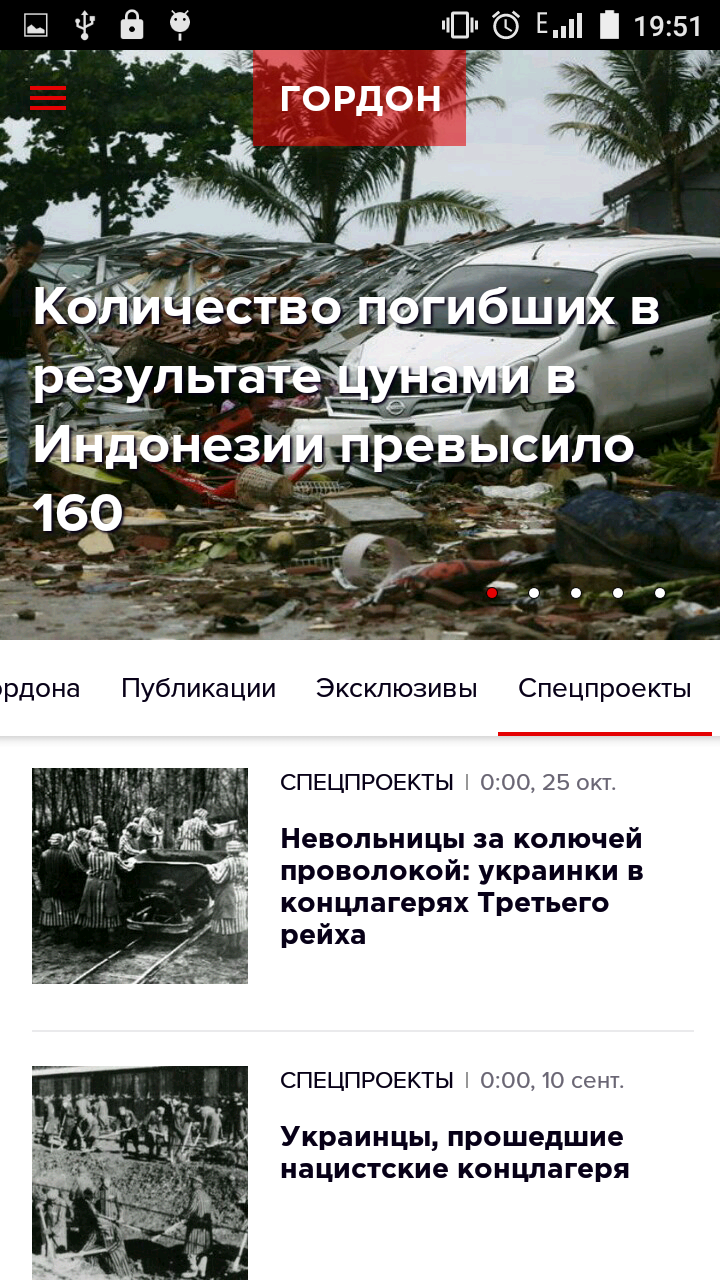


Рисунок 2.6 – основной список новостей

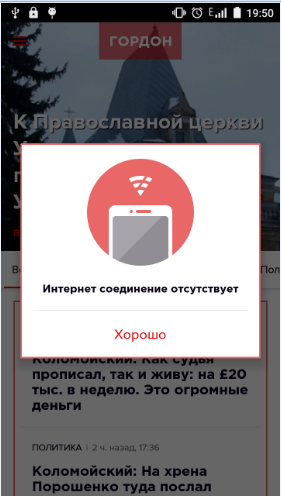


Рисунок 2.7 – уведомление о пропаже интернета

## Выбор инструментальных средств разработки системы

Для разработки и тестирования отдельных модулей системы была выбрана IDE Visual Studio 2017, за простоту в освоении и сравнительно невысокую нетребовательность к системным ресурсам компьютера.

Для разработки основных модулей системы было решено использовать среду разработки Android Studio 3.2 из-за специализации этой IDE на разработку мобильных Android приложений и крайнюю удобность в этом.

Для проектирования и описания архитектуры продукта решено использовать Microsoft Visio из-за удобности и современности этого продукта.

Язык UML был выбран как универсальный язык описания архитектуры и жизненного цикла ПО. Несмотря на универсальность этого языка, в корпорациях используются и другие языки описания архитектуры ПО:

* [**Райт**](http://www.cs.cmu.edu/afs/cs/project/able/www/wright/) позволяет архитекторам указывать временные протоколы связи и проверять такие свойства, как свобода взаимоблокировки;
* [**UniCon**](http://www.cs.cmu.edu/afs/cs/project/vit/www/unicon/) - это язык архитектурного описания, основное внимание в котором уделяется поддержке разнообразия архитектурных деталей и стилей, а также построения систем из их архитектурного описания;
* [**Acme**](http://www.cs.cmu.edu/~acme/) - это простой, общий язык описания архитектуры программного обеспечения (ADL), который может использоваться как общий формат обмена для инструменты проектирования архитектуры и/или как основа для разработки новых инструменты архитектурного проектирования и анализа;
* [**Aesop**](http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/aesop_home.html) предоставляет общий набор инструментов и инфраструктуру связи которые пользователи могут настраивать с помощью описаний архитектурного стиля и набор инструментов, которые они хотели бы использовать для архитектурного анализа;
* [**C2**](ftp://ftp.ics.uci.edu/pub/arch/c2.html) - это общий архитектурный стиль, основанный на компонентах и ​​сообщениях, который хорошо подходит для крупномасштабных, гетерогенных и распределенных приложениях. C2 SADL - это ADL для выражения архитектур в стиль C2;
* [**Darwin**](http://www-dse.doc.ic.ac.uk/Software/) - показывает системную архитектуру с точки зрения компонентов и их взаимосвязи;
* [**MetaH**](http://www.honeywell.com/sites/docs/DKRHVMR134XK8DQRHFA5JVOVUZQ23EN0K.pdf) определяет, как разрабатываются программные модули в различных стилях и как они вместе соответствуют полной архитектурной системе.

## Программная реализация базовых модулей системы (код?)

Было решено программно реализовать модуль получения и отображения данных БД.

Для этого использовался паттерн Наблюдатель.

В реализованном модуле в роли подписчика выступает класс RepositiriesPresenter, а в роли Издателя – стандартный интерфейс Java Observable.

Методы этих классов, задействованные в реализации паттерна Наблюдатель:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Параметры** | **Возвращаемое значение** | **Описание** |
| dispatchCreated | - | - | Подписывает объект себя на изменение кэшированных данных и отображение их. |
| refresh | - | - | Отображает данные из Интернета |

Таблица 2.4 – методы класса RepositoriesPresenter, задействованные в реализации паттерна «наблюдатель»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Параметры** | **Возвращаемое значение** | **Описание** |
| subscribeOn | Scheduler scheduler | Observable<T> | Определяет, какое действие нужно выполнить при начале синхронизации данных. |
| doOnTerminate | Action onTerminate | Observable<T> | Определяет, какое действие нужно выполнить при завершении синхронизации данных. |

Таблица 2.5 – методы интерфейса Observable , задействованные в реализации паттерна «наблюдатель»

## Анализ качества и оценка программного дизайна (Откуда брать цифры?)

# 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МОДУЛЯ получения данных

## Описание используемых паттернов

Наблюдатель (англ. Observer) — поведенческий шаблон проектирования. Также известен как «подчинённые» (Dependents). Реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов, и тем самым наблюдать за ними. Также называется «Издатель-подписчик».

В шаблоне учавствуют две сущности: Субъект (издатель) и Наблюдатель (подписчик).

Подписчики отправляют Издателю запрос о подписке, и он их заносит в свой список. При изменении данных Издателя, он уведомляет каждого подписчика об этом.

## Структура используемых паттернов

Обобщенная диаграмма классов может выглядеть так:

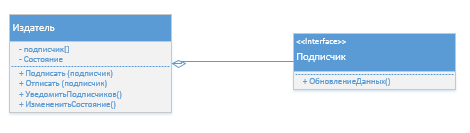


Рисунок 2.8 – диаграмма классов шаблона Наблюдатель

Структура шаблона выглядит так:

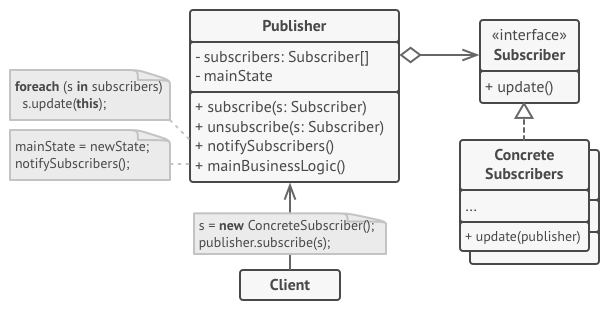


Рисунок 2.9 – структура шаблона Наблюдатель

1. Издатель владеет внутренним состоянием, изменение которого интересно отслеживать подписчикам. Издатель содержит механизм подписки: список подписчиков и методы подписки/отписки.
2. Подписчики регистрируют себя на обновления издателя.
3. Когда внутреннее состояние издателя меняется, он оповещает своих подписчиков. Для этого издатель проходит по списку подписчиков и вызывает их метод оповещения, заданный в общем интерфейсе подписчиков.
4. Подписчик определяет интерфейс, которым пользуется издатель для отправки оповещения. В большинстве случаев для этого достаточно единственного метода.
5. Конкретные подписчики выполняют что-то в ответ на оповещение, пришедшее от издателя. Эти классы должны следовать общему интерфейсу подписчиков, чтобы издатель не зависел от конкретных классов подписчиков.
6. По приходу оповещения подписчику нужно получить обновлённое состояние издателя. Издатель может передать это состояние через параметры метода оповещения. Более гибкий вариант — передавать через параметры весь объект издателя, чтобы подписчик мог сам получить требуемые данные. Как вариант, подписчик может постоянно хранить ссылку на объект издателя, переданный ему в конструкторе.

## Программный код паттернов

Паттерн Наблюдатель.

Код абстрактного метода Subscribe объекта Подписчик, реализованного в виде класса RepositoriesPresenter:

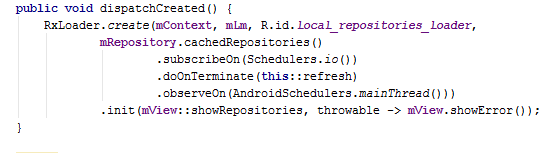


Рисунок 2.10 – программный код метода dispatchCreated объекта Подписчик паттерна Наблюдатель

Как видно, используется вариация паттерна Наблюдатель, при которой подписчик уже при подписке передает метод, который должен быть вызван при изменении данных.

Т.к. подписчик может быть подписан на несколько издателей (что и происходит в моем коде), то имеется еще одна реализация метода Subscribe:

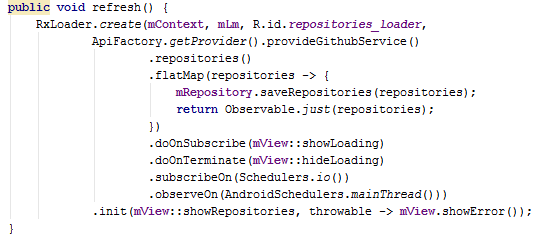


Рисунок 2.11 – программный код метода refresh объекта Подписчик паттерна Наблюдатель

Объект Издатель.

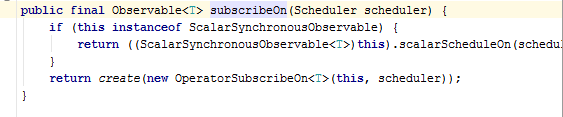


Рисунок 2.12 – программный код метода subscribeOn объекта Издатель паттерна Наблюдатель

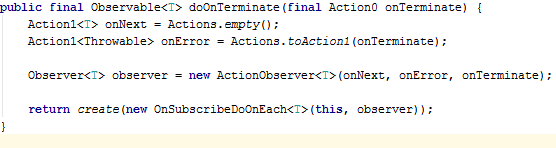


Рисунок 2.13 - программный код метода doOnTerminate объекта Издатель паттерна Наблюдатель

# Заключение

В результате проделанной работы были проанализированы требования к приложению для просмотра новостей, спроектирована архитектура приложения, дан пример пользовательского интерфейса приложения, реализован модуль получения данных из БД.

В дальнейшем, эту работу и архитектуру можно использовать как прототип для дальнейшего развития приложения. Пользовательский интерфейс можно доработать и стилизовать под стиль собственного приложения, прототип архитектуры может помочь понять схему работы клиент-серверных приложений.

# Список литературы

1. [Электронный ресурс] Режим доступа: <https://habr.com/company/e-Legion/blog/265405/>
2. [Электронный ресурс] Режим доступа: http://app-global.ru/blog/kak-sozdat-novostnoe-prilozhenie-dlya-android-v-servise-appglobal/
3. [Электронный ресурс] Режим доступа: https://bigpicture.ru/?p=1063964 [Электронный ресурс] Режим доступа: https://habr.com/company/e-Legion/blog/265405/
4. [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller
5. [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Архитектура\_программного\_обеспечения
6. [Электронный ресурс] Режим доступа: https://habr.com/post/276593/

# Приложение 1

Техническое задание

# Приложение 2

Функциональная модель

# Приложение 3

Фрагменты листинга программы