Слайд 1

Сегодня я расскажу о том, как надо строить приложение. Первое с чего нужно начать, это выбрать среду разработки и язык программирования. Если с языком программирования всё понятно – язык Java, то при выборе среды разработки было два основных варианта – Android Studio и IntelliJ IDEA. При установке IntelliJ IDEA возникли проблемы, поэтому был установлен Android Studio.

Слайд 2

Android Studio является официальной интегрированной средой разработки (IDE) для разработки приложений для Android. Она предоставляет широкий набор инструментов, таких как визуальный редактор макетов, поддержка профилирования и отладки, а также интеграцию с системой сборки Gradle. Кроме того, Android Studio предлагает эмуляторы устройств, которые позволяют тестировать приложения на различных конфигурациях устройств, а также интегрируется с Git для управления версиями.

В целом, Android Studio предоставляет все необходимые инструменты для разработки высококачественных приложений для Android, что делает ее отличным выбором для разработчиков.

Слайд 3

Следующие в чём стоит определиться – это выбора архитектурного паттерна мобильного приложения. Существует несколько MV паттернов: MVC, MVVP, MVP. Сначала рассмотрим первый подход MVC (Model Viev Controller).

Слайд 4

Основная суть, которые заложили авторы – это разделение бизнес логики от интерфейса. Под бизнес логикой подразумевается функционал, который приносит конкретную пользу вашему проекту (например регистрация нового пользователя, в котором есть правила хранения пароля, сохранения пользователя в базу данных и т.д.). При этом под интерфейсом подразумевается непосредственно тот графический интерфейс, с которым взаимодействует пользователь (формы, кнопки, текст и т.д.).

Слайд 5

Давайте начнем с проблематики и поймём почему объединение интерфейса (UI) и логики это плохо? Сейчас на слайде вы можете заметить, что всё перемешано.

Слайд 6

Представим, что мы разработчики калькулятора, он выглядит как-то вот так. И нужно сделать вот такой калькулятор, с новым отображением, но при этом много старых пользователей, которые к текущему интерфейсу уже привыкли и мы хотим дать им возможность выбрать самим (например добавить какую-то галочку), а новым пользователям всегда будем показывать новое изображение. То есть проблема тут в том, что мы не можем взять старый калькулятор, изменить у него стиль изображения, чтобы стал новый. Нам надо поддерживать и предыдущее и новое состояние калькулятора

Слайд 7

И вот на этом этапе мы сталкиваемся с первыми сложностями, по сколько UI и логику у нас связаны, мы не можем взять и легко разделить их одно от другого

Слайд 8

В идеале это должно было быть так UI отдельно, логика отдельно, при этом внутри UI мы с помощью кнопок, форм как-то взаимодействуем с логикой. Причём мы не делаем это напрямую, между UI и логикой всегда есть некое связующее звено – контроллер. За счёт этого контроллер мы не получаем сильной связанности между UI и логикой, мы всегда можем их подменять.

Слайд 9

В контексте паттерна MVC интерфейс называется VIEW, контроллер так и называется – controller, бизнес логика называется model. Model View Controller. То есть ещё раз какой у нас поток данных, у нас есть отображение, например кнопка с формой сохранить, пользователь заполняет форму, нажимает сохранить и при нажатие на эту кнопку у нас дёргается контроллер, внутри контроллер мы можем например залогировать (запись логов (в лог записывается инфа об ошибках)) события, провалидировать (проверка данных по критериям корректности) даннные и если всё хорошо, данные, которые мы получили из формы мы передаём в модель. Модель в свою очередь содержит явные бизнес правила, например сохранение пароля. Контроллер при этом не должен содержать логики, максимум какие то проверки.

Слайд 10

Так же существует архитектурный паттерн Model View Presenter (MVP). Особенностью паттерна MVP является то, что Presenter, это тот компонент, который раньше именовался контроллером. Является промежуточным компонентом между моделью и представлением. То есть напрямую модель и представление больше не взаимодействуют.

А в паттерне Model View ViewModel, вместо контроллер или презентора используется компонент, который называет ViewModel и отличие заключается в том, что у ViewModel нет ссылки на представление. А представление (View) подписывается на изменения ViewModel.

Слайд 11

При разработке мобильного приложения важно учитывать различия в характеристиках и разрешениях экранов различных устройств Android. Это позволит вашему приложению выглядеть и работать оптимально на широком спектре устройств, от смартфонов до планшетов.

Масштабируемый дизайн:

- Создание масштабируемого дизайна позволяет вашему приложению адаптироваться к различным размерам экранов. Чтобы создать масштабируемый дизайн, при определении размеров и полей в макетах следует использовать независимые от плотности пиксели (dp) вместо абсолютных пикселей (px). Таким образом, ваши элементы пользовательского интерфейса будут отображаться с одинаковым визуальным размером на разных устройствах с разной плотностью экрана.

Масштабируемый функционал:

- Предоставление масштабируемого функционала означает, что ваше приложение должно корректно работать на устройствах с разными спецификациями. Это включает в себя адаптацию к разным версиям Android, поддержку различных разрешений экранов и взаимодействие с разными версиями библиотек и API.

Чтобы ваше приложение было удобным для пользователей на как можно большем количестве устройств, важно оптимизировать его для разных устройств.

Cлайд 12

Так же я попытался для каждого найти подходящие библиотеки, для создания своего проекта

Для работы с модулем GPS можно использовать следующие библиотеки:

Google Play Services Location Services: Это официальная библиотека от Google, которая предоставляет доступ к GPS и другим местоположениям на устройстве.

Android Fused Location Provider: Это библиотека, которая объединяет данные с различных поставщиков местоположения (GPS, Wi-Fi, сеть) и предоставляет более точное и эффективное местоположение.

Android Location Manager: Это встроенная библиотека Android, которая предоставляет доступ к GPS и другим местоположениям на устройстве.

Для работы с модулем ГЛОНАСС я практически ничего не нашёл)))))), есть вот такая библиотека с использованием глобальной навигационной спутниковой системы (GNSS) Глобальная навигационная спутниковая система (от англ. – Global Navigation Satellite System) GNSS - это совокупность группировок спутников (GPS, ГЛОНАСС, Galileo, Beidou)

Android GNSS Logger: это образец приложения, предоставленный Google, который демонстрирует, как регистрировать данные GNSS. Вы можете использовать эту библиотеку для регистрации данных GNSS в целях отладки и анализа.

Для модуля гироскопа я нашёл статью, в которой рекомендовали использовать Классы SensorManager и Sensor в Android. Они используются для взаимодействия с различными датчиками, доступными на устройстве, включая гироскоп.

В контексте мобильного приложения, использующего модуль гироскопа, классы SensorManager и Sensor используются для:

Регистрации прослушивателя событий гироскопа, чтобы ваше приложение могло получать обновления при изменении данных гироскопа.

Получите текущее состояние гироскопа, включая его ориентацию и скорость вращения.

Установите режим отчетности и задержку для гироскопа, чтобы оптимизировать производительность и расход заряда батареи.

Что касается библиотек, есть несколько доступных, которые могут помочь в обработке и визуализации данных гироскопа, например Processing и OpenCV. Однако эти библиотеки могут оказаться излишними.

Для использования модуля акселерометра в проекте существует несколько вариантов:

Google's Sensor Library: эта библиотека предоставляет простой и стандартизированный способ доступа к данным датчиков в Android. Он включает поддержку акселерометра и других датчиков.

Accelerometer API от Senstive: эта библиотека предлагает простой и удобный в использовании API для данных акселерометра. Он также включает в себя дополнительные функции, такие как обнаружение движения и обработка событий.

AndSensor от NanoHttpd: эта библиотека обеспечивает простой и эффективный способ доступа к данным датчиков, включая акселерометр. Он также поддерживает другие датчики и включает в себя такие функции, как обработка и фильтрация событий.

Для магнитрона я вообще ничего не нашёл;(

Слайд 13

Так же стоить обратить внимание на вот эти аспекты, для создания более правильной версии приложения, но я считаю, что это можно оставить на потом