1. Операционна система Android. Архитектура на операционната среда Android. Софтуерни слоеве.

Общи сведения:

През 2005 г. Google Inc. закупува операционната система Android от Android Inc. и започва нейното развитие. Пускането и на пазара на 5 ноември 2007 г

За развитието на Android се грижат голям брой софтуерни разработчици, които създават така наречените "apps" - малки приложения, които разширяват функционалността на системата. Приложенията могат да бъдат сваляни от различни сайтове в Интернет или от големи он-лайн магазини като Android Market (ново име Google Play)

Архитектура:

Основата на Android е ядрото на Linux (версия 2.6). То е отговорно за управлението на паметта и процесите, както и за мрежовите връзки.

- Тук са разположени и драйвърите.

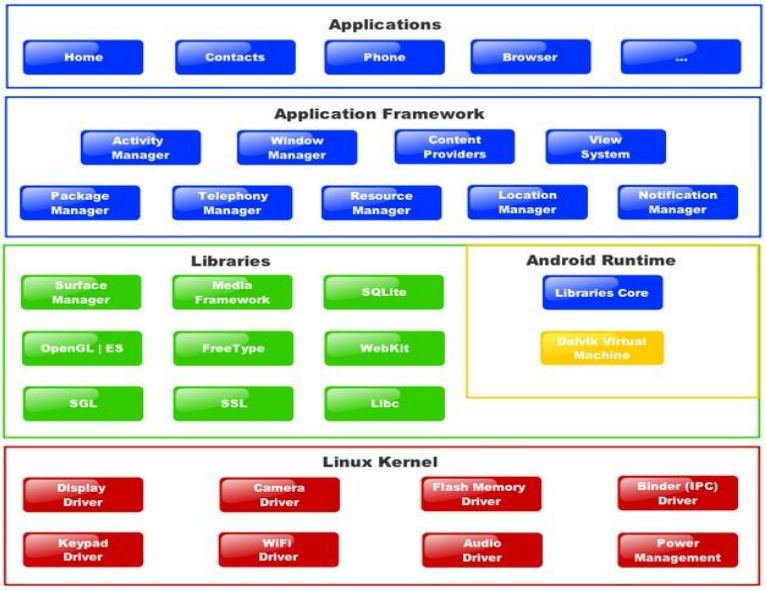
Директно над ядрото се намира т. нар. "Runtime Environment".

- Тя съдържа най-важните библиотеки по време на изпълнение и най-важната функционалност на езика Java.
- Тук се съдържа и виртуалната машина Dalvik Virtual Machine (DVM).
- Тя се различава от класическите виртуални машини на Java (Java Virtual Machine, JVM) по това, че е оптимизирана за мобилни устройства с малко памет.
- Оптимизацията позволява и едновременното изпълнение на няколко виртуални машини на същото устройство.

Android съдържа няколко библиотеки на C/C++:

- Surface Manager (управлява достъпа до дисплея)
- OpenGL ES (приложно-програмен интерфейс за 3D компютърна графика), SGL (приложно-програмен интерфейс за 2D компютърна графика)
- Media Framework (управление на мултимедийно съдържание, на основата на OpenCORE, поддържа формати като MPEG4, H.264, MP3, AAC, AMR, JPG PNG и др.)
- FreeType (библиотека за рендъринг на пикселни и векторни шрифтове)
- SSL (криптиране)
- SQLite (бази данни)
- WebKit (рендъринг на HTML)
- Libc (версия на стандартната C-библиотека за Android)

Приложният фреймуърк (Application Framework) ползва библиотеките на C/C++ и предлага стандартизиран приложно-програмен интерфейс за програмистите на приложения (Apps). Android се доставя с няколко приложения, сред които са комуникационните приложения за телефониране, електронна поща, SMS и браузър, както и Google Maps, календар и приложение за управление на контактите.



2. Компоненти на приложение в Android. Комуникации между компонентите.

Ето и някои от по важните компоненти за едно Андроид приложение.

- Activity
  - Представя тип слой от приложението например екран, който потребителя ще вижда.
  - Едно приложение може да има няколко слоя активности и може да превключва между тях по време на работа на проложението.
  - Слоевете view се управляват от ViewGroups.
- Services
  - Услугите ви осигуряват фонови задачи без наличието на интерфейс.
  - Те могат да известяват потребителя посредством notification framework в Андроид.
- Content Provider
  - Доставя данни от приложението.
  - Благодарение на този компонент можете да споделяте данни с други приложения.
  - Андроид съдържа SQLite база данни, която може да служи като доставчик на тези данни.
- Intents
  - Това са асинхронни съобщения, които позволяват приложението да иска функционалност от други услуги или дейности.
  - Приложението може директно да извика услуга или дейност за тези си намерения (implicit intents).
  - Например приложението може да извика чрез Intent приложението за контакти.
  - Компонентът Intent е много мощен инструмент, благодарение на който можете да създавате свободно свързани приложения.
- Broadcast Receiver
  - Получава системни съобщения, както и асинхронни съобщения implicit intents.
  - Може да се използва за реакция при променящи се условия в системата.
  - Приложението може да се регистрира като приемник (Broadcast Receiver) за определени събития, и може да се стартира при наличието на такова (събитие).

3. Манифест на приложение. Android SDK и процес на разработка. Публикуване на приложение.

The Manifest File:
Всички компоненти трябва да бъдат декларирани в manifest файл, наречен AndroidManifest.xml, който се намира в главната директория на приложението.
Този файл работи като интерфейс между Android и приложението ни.
Ако не декларираме компонента си в този файл, той няма да бъде намерен от ОС.
Това са следните тагове, които ще използваме в нашия manifest файл, за да посочим различните компоненти на дадено Android приложение:
<activity> - елементи за activities;
<service> - елементи за services ;
<receiver> - елементи за broadcast receivers;
<provider> - елементи за content providers.

Android SDK и Процес на разработка:
Android SDK е набор от инструменти за разработка, използвани за разработване на приложения за Android платформата.
Android SDK съдържа следните компоненти – необходими библиотеки, дебъгер, емулатор, документациая за различни API и др.
Една често използвана IDE за разработка на Android приложения през последните години е Eclipse. Можете да получите достъп до необходимите инструменти за Eclipse чрез ADT (Android Development Tools) – плъгин за Eclipse IDE, чрез който се предоставя мощна интегрирана среда за създаване на Android приложения.
Освен Eclipse, съществува друга мощна и по-модерна среда за разработка – Android Studio.
Тя е официалната IDE за разработка на Android приложения, базирана на IntelliJ IDEA.
Много от разработчиците, работещи преди това на Eclipse, са мигрирали към Android Studio заради по-добрата поддръжка.
В нея са включени много нови функционалности и Google я поддържа с постоянни ъпдейти.
Основните стъпки за разработване на приложения с или без Eclipse са едни и същи:
1.Настройте Android Virtual Devices или хардуерни устройства.
Трябва да създадете AndroidVirtual Devices(AVD) или да се свържете хардуерни устройства, на които ще инсталирате приложения.
Един Android Virtua lDevice(AVD) е емулатор конфигурация, която ви позволява да симулирате действително устройство чрез определяне на хардуерни и софтуерни опции да бъдат емулирани от Emulator Android.
2. Създаване на проекта за Android.
Един проект за Android съдържа всичкия сорс код и ресурсни файлове за вашето приложение.
Той е построен в apk пакет, които можете да инсталирате на устройства с Android.
3. Компилирайте и стартирайте вашето приложение.
4. Debug на вашето приложение с SDK debugging инструменти. Инструментите са осигурени с Android SDK. Eclipse вече идва в комплект със съвместим дебъгер.
5. Тествайте вашето приложение.

Публикуване на приложение:
След пълно тестване на приложението ни, можем да го продаваме или разпространяваме в Google Play магазина или директно да го споделяме с нашите потребители, като например го качим на нашия уеб сайт. Обаче, за да го публикуваме в Google Play, трябва да сме изпълнили няколко изисквания като тестване на различни устройства за съвместимост, възрастови ограничения, размер на приложението, цена, използвано SDK и съвместимост с дисплея, описание, снимки от потребителски интерфейс и др. Приложението трябва да бъде запазено в APK формат преди да бъде качено в Google Play магазина. След това трябва да си направите Google Play developer акаунт (който е платен - 25 $) и вече можете да качите вашите приложения и да чакате за одобрение и официално публикуване в Google Play магазина.

4. Android приложение. Файлова структура. Създаване и изпълнение. R клас. Примери.

Android API- то включва повече от 70 пакета, с повече от 400 класа
Първо приложение: Hello,World

- Като разработчик, вие знаете, че първото впечатление за development frameworka-а е колко лесно еда се напише"Hello, World".
- Е,на Android, това е доста лесно.
- Това е особено лесно, ако използвате Eclipse IDE + ADT или Android Studio.

Създаване в Eclipse IDE:

1.Вие ще стартирате вашето приложение в Emulator Android.
- Преди да можете да стартирате емулатора, трябва да създадете Android Virtual Device (AVD).
- AVD определя системен образ и настройките на устройството, използвани от емулатора
2. Създаване на нов проект Android:
- След като сте създали AVD можете да преминете към следващата стъпка и да започне нов проект Android в Eclipse.
- Вашият Android проект вече е готов.
- Той трябва да се вижда в пакет Explorer в ляво.
- Отворете файла HelloAndroid.java, разположени вътре HelloAndroid→SRCcom.example.helloandroid

Тя трябва да изглежда така:

```
package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {
/** Called when the activity is first created. */
        @Override public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
        }
}
```

Забележете, че този клас се базира на Activity класа.
Activity е един обект от приложението, който се използва за извършване на действия.
Приложението може да има много отделни дейности, но потребителят взаимодейства с тях един по един.
OnCreate() метод се извиква от системата Android, когато вашето Activity започва- това е мястото, където трябва да се изпълни цялата инициализация и UI настройка.

R клас:

Файлът R.java е индекс на всички ресурси, определени във файла.
Можете да използвате този клас в изходен код като един по-кратък начин да реферира на ресурсите, които сте включили във вашия проект.
Това е автоматично генериран файл и не трябва да се модифицира съдържанието на R.java файла.

5. Организация на ресурси. Достъп до ресурси.

Организация:
Всеки тип ресурс трябва да се поставя в специфична поддиректория от res/ директорията на проекта.
- anim/ xml файловете дефинират пропъртита на анимации. Те се пазят в папката res/anim и се достъпват чрез R.anim
- color/ xml файловете дефинират списък от цветове. Те се пазят в res/color и достъпват чрез R.color класа
- drawable/ Image файловете като .png, .jpg, .gif, или XML файлове. Те се компилират до bitmaps, state списъци, форми, animation drawables. Те се съхраняват в res/drawable/ и се достъпват чрез R.layout класа
- menu/ xml файловете дефинират менюта, като Options Menu, Context Menu, Sub Menu. Те се съхраняват в res/menu/ и се достъпват чрез R.menu класа.
- raw/ Arbitrary файловете, запазват смоята "raw" сурова форма. За да се отвори такъв файл, трябва да се извика callResources.openRawResource() с ID на ресурса, което е R.raw.filename.
- xml/ Arbitrary xml файловете, могат да бъдат четени по време на изпълнение на програмата, чрез извикване на Resources.getXML(). Тук могат да се съхраняват различни конфигурационни файлове, които да се използват по време на изпълнение
- values/ xml файловете, съдържат прости стойности, като string, integer, colors. Например, това са някои конвенции за името на файл на ресурси, които могат да се създадат в тази директория:
  - arrays.xml за масиви, достъпвани чрез R.array клас
  - integers.xml за integer ресурси, достъпвани чрез R.integer клас
  - bools.xml за boolean ресурси, достъпвани чрез R.bool клас
  - colors.xml за color стойности, достъпвани чрез R.color клас
  - dimens.xml за dimension стойности, достъпвани чрез R.dimen клас
  - strings.xml за string стойности, достъпвани чрез R.string клас
  - styles.xml за style ресурси, достъпвани чрез R.style клас

Достъп:
Когато Android приложението се компилира, се генерира R клас, който съдържа ID на ресурси за всички ресурси, налични в res/ директорията.
Можем да използваме R класа, за да достъпваме конкретен ресурс, използвайки поддиректория и името на ресурса или директно чрез ID на ресурса.

6. Компонент Activity в Android приложение. Създаване на Activity. Стек на Activity.

Activities — Презентационния слой на приложението. UI на приложението се изгражда около 1 или повече разширения на Activity класа. Activities използват Fragments и Views за оформяне и визуализиране на информацията и за реагиране на действията на потребителя. В сравнение с desktop разработката, Activities са еквивалент на Forms.

- всяко Activity представлява екран, който приложението показва на своите потребители
- колкото по-комлексно е приложението, толкова повече екрани са необходими
- за придвижване между екраните се стартира ново Activity (или се връщаме от някое).
- повечето Activities са проектирани да заемат целия дисплей, но също така могат да се създават полупрозрачни или плаващи (floating) Activities.

Създаване: (skeleton code)

```
package com.paad.activities;

import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity {

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

Activity Stack:
Състоянието на всяко Activity се определя от неговата позиция върху Activity stack, стек колекция от всички текущо изпълняващи се Activities.
Когато ново Activity се стартира, то става активно и се премества на върха на стека.
Ако потребителят се върне назад чрез Back бутона или Activity във foreground (на преден план) бъде затворено, следващото Activity надолу в стека се качва нагоре и става активно.

Activity състояния:
Active – Когато едно Activity е на върха на стека, това е видимото Activity, на фокус, това на преден план, което получава входни данни от потребителя. Когато друго Activity стане активно, текущото ще бъде оставено на пауза.
Paused – В някои случаи вашето Activity ще бъде видимо, но няма да бъде на фокус; в такъв случай то е на пауза. До това състояние се достига, ако прозрачно или не напълно прозрачно Activity е активно пред него. Когато е паузирано, едно Activity се третира все едно е било активно, но не може да получава събития за въвеждане на данни от потребителя.
Stopped – Когато едно Activity не е видимо, то е спряно.. Това Activity ще остане в паметта, запазвайки цялата информация за състояние; обаче става кандидат за терминиране, когато системата има нужда от памет някъде другаде. Когато едно Activity е в спряно състояние, е важно да се запазят данните и настоящето UI състояние, и да се спрат всички некритични операции.
Inactive – След като едно Activity е било унищожено и преди да бъде стартирано, то е неактивно. Неактивните Activities са били премахнати от Activity стека и трябва да бъдат рестартирани преди да могат да се визуализират и използват.

7. Изграждане на графически потребителски интерфейс. Основни елементи. Работа с Layout. Фрагменти. Контроли.

Фундаментален Android UI дизайн:
Views — Views са базовият клас за всички елементи на графичния интерфейс (по-познати като контроли или уиджети). Всички UI контроли, включително и layout класовете, са наследници на View.
View Groups — View Groups са разширения на View класа, които могат да съдържат множество child Views. Разширете ViewGroup класа, за да създадете смесени контроли, създадени от взаимносвързани дъщерни Views. ViewGroup класът също е разширен, за да осигурява помощ на Layout мениджърите, когато нареждаме контроли в своите Activities.
Fragments — Fragments, въведени в Android 3.0, се използват за капсулация на части от нашия UI.

-Тази капсулация прави фрагментите особено полезни при оптимизациране на UI layouts за различни големини на екрана и създаване на преизползваеми UI елементи.
- Всеки фрагмент съдържа свой собствен UI layout и получава съответните входни събития, но е тясно обвързан към Activity, в което всеки трябва да бъде вграден.

Activities — Activities представят прозореца, екрана, които са показани. Activities са Android еквивалента на формите в традиционната desktop Windows разработка. За да визуализираме UI, ние задаваме View (обикновено layout или Fragment) към едно Activity.

Layout:
Определя как елементите са позиционирани относително един спрямо друг (един до друг, един под друг, в таблица, мрежа и т.н.)
Може да има различни layouts за всяка ViewGroup.

Layout Managers:
Могат да са контейнери за други View.
Имплементират стратегии за управление на размер, позиция на техните дъщерни елементи.
Layout managers, използвани в Android:

◦ **Linear Layout:** Подрежда всички дъщерни елементи в една посока (хоризонтално или вертикално); Дъщерните елементи се нареждат един след друг. Можем да влагаме множество LinearLayouts

◦ **Relative Layout:** Показва дъщерни View в относителни позиции; Можем да зададем позиция в зависимост от родителския елемент или другите дъщерни елементи на едно View; Премахва нуждата от вложени View; Много вложени LinearLayouts могат да бъдат конвертирани в един Relative Layout.

◦ **Table Layout:** Пази всички дъщерни View в таблица; В Table Layout, TableRow представлява един ред; Всички дъщерни елементи в TableRow са колони; Полезен за визуализиране на данни в редове и колони; Не е полезен за проектиране на завършен потребителски интерфейс.

◦ **Grid Layout:** Пази всички свои дъщерни Views в правоъгълна мрежа; По подразбиране можем да дефинираме брой редове и колони и всички дъщерни View в Grid Layout са като в матрица. Можем ръчно да дефинираме към кой ред/колона даден обект принадлежи, използвайки пропъртита layout_row и layout_column; Полезен за визуализиране на галерии от изображения, данни в мрежа и т.н.

Fragments**:**

Фрагментите дават възможност за разделяне на Activities в напълно капсулирани преизползваеми компоненти, всеки със свой жизнен цикъл и потребителски интерфейс.
Всеки фрагмент е независим модул, който е тясно обвързан с това Activity, в което е поставен.
Фрагментите могат да бъдат преизползвани в множество Activities.
Фрагментите предоставят начин за представяне на стабилен UI, оптимизиран за широк обхват от Android устройства, размер и плътност на екрана
Всяко Activity включва Fragment Manager, за да управлява фрагментите, които съдържа.
Трансакции върху фрагменти могат да бъдат използвани за добавяне, премахване и замяна на фрагменти, в рамките на едно Activity при стартиране.
Всяка трансакция върху фрагмент може да включва всяка комбинация от поддържани действия, включително добавяне, премахване или замяна на фрагменти.

Основни контроли за вход**:**

Контролите за вход се използват за получаване на данни от потребителя.
Най-често използваните контроли в Android са:

- Текстови полета
- TextView
- Бутони (Button, ImageButton, RadioButton, ToggleButton)
- Checkboxes
- Spinners
- ImageView

8. Управление на събитията в Android. Пример.

Обработка на събития при взаимодействие с потребителя:
За да бъде едно ново View интерактивно, то трябва да отговаря на инициирани от потребителя събития като задържане на бутон, докосване на екрана и кликване върху бутон.
В Android се използват няколко виртуални обработчици на събития, които можете да използвате за реагиране на вход от потребителя:

- onKeyDown — Извиква се, когато някой бутон е натиснат; включва клавиатура, обаждане, връщане назад, камера бутони и др.
- onKeyUp — Извиква се, когато потребителят освободи натиснат бутон
- onTouchEvent — Извиква се, когато екранът е натиснат или освободен, или когато засече движение

Обработка на събития:
- Определете кои Widgets кои събития да обработят
- Дефинирайте event listener и го регистрирайте с View.
  - View.OnClickListener (за обработка на кликване във View),
  - View.OnTouchListener (за обработка на докосване на екрана във View),
  - View.OnKeyListener (за обработка на натискане на бутон във View)

Пример:
Стъпка 1: Добавяне на бутон
Стъпка 2: Регистрирайте обработчик на събитие (Event Handler)
+ ДВЕ ОПЦИИ – отделни класове за обработка на събитие(я) ИЛИ предоставяне на Activity, съдържащо бутона, да направи обработката на събитието
Стъпка 3: Имплементиране на обработчика на събитие - за бутон това означава да се имплементира View.OnClickListener интерфейса

Тук кодът за обработка е в самото Activity:

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        // Стъпка 1
        Button button = (Button)findViewById(R.id.corky);
        // Стъпка 2
        button.setOnClickListener(this);
    }
    // Имплементация на OnClickListener
    // Стъпка 3
    public void onClick(View v) {
        // направи нещо, когато бутонът е кликнат
    }
    …
}
```

9. Работа с компонента Intents. Неявно и явно стартиране на Activity в Android приложение. Broadcast събития.

Intents:
Intents се използват като механизъм за предаване на съобщения, който работи както във вашето приложение така и между различни приложения.
Изрично стартиране на определено Service или Activity чрез неговото име на клас.
Стартиране на Activity или Service за изпълнение на действие със (или върху) определена част данни.
Предаване(Broadcast), че дадено събитие е възникнало.

Използване на Intents за стартиране на Activities:
За създаване и визуализиране на Activity, извикайте startActivity, като подадете Intent, както следва:

```
startActivity(myIntent);
```

startActivity методът намира и стартира това Activity, което най-добре съвпада с вашия Intent.
Изрично стартиране на нови Activities:

```
Intent intent = new Intent(MyActivity.this, MyOtherActivity.class);
startActivity(intent);
```

Всеки път, когато извикате startActivity, ново Activity ще бъде добавено към стека.
Натискане на бутона за връщане назад или прекратяване ще премахне всяко от тези Activities.
Едно Activity, стартирано чрез startActivity, е независимо от своя родител и няма да предостави никаква обратна връзка, след като се затвори.

Implicit (неявни/скрити) Intents:
Механизъм, позволяващ анонимни заявки (anonymous application components service action requests).
Можете да поискате от системата да стартира Activity, което да изпълни действие без да се знае кое приложение или Activity ще бъде стартирано.
Набиране на номер:

```
Intent intent = new Intent (Intent.ACTION_DIAL, Uri.parse("tel:93675359"));
startActivity(intent);
```

Отваряне на уебсайт:

```
Intent intent = new Intent (Intent.ACaTION_VIEW, Uri.parse("http://codeandroid.org"));
startActivity(intent);
```

Използване на Intents за предаване на събития:
Също така можем да използваме Intents за предаване на съобщения анонимно между компонентите чрез метода sendBroadcast.
Като механизъм за предаване на съобщения на системно ниво, Intents са способни да изпращат структурирани съобщения извън рамките на процес.
Като резултат, можем да имплементираме Broadcast Receivers за слушане и отговаряне на тези Broadcast Intents в нашето приложение.
Broadcast Intents се използват за уведомяване на системните приложения или събития в приложение, event-driven programming модела за разработка между приложенията.

10. Провайдер на съдържание и споделяне на данни. Достъп да данни в Android приложение. Пример.

Провайдер на съдържание:
Провайдерът предлага капсулация на данни, базирани на URI.
Всяко URI, което започва с "content://", сочи към ресурси, които могат да бъдат достъпени чрез провайдер.
URI за ресурс може да да позволява извършването на базовите CRUD операции (Create, Read, Update, Delete) върху ресурса чрез провайдера на съдържание.
Провайдерът позволява на приложенията да достъпват данни.
Провайдерите на съдържание управляват достъпа до структуриран набор от данни.
Те капсулират данните и осигуряват механизми за дефиниране на сигурността на данните.
Провайдерите на съдържание са стандартният интерфейс, който свързва данните в един процес с кода, работещ в друг процес.
Когато искаме да достъпим данни в провайдер на съдържание, използваме ContentResolver обекта в контекста на нашето приожение, за да комуникираме с провайдера като с клиент.
ContentResolver обектът комуникира с провайдер обекта, инстанция на клас, който имплементира ContentProvider.
Провайдер обектът получава заявки за данни от клиентите, изпълнява поисканото действие и връща резултатите.
Няма нужда да разработваме свои собствен провайдер, ако не планираме да споделяме нашите данни с други приложения.
Обаче, се нуждаем от свой собствен провайдер, за да осигуряваме персонализирани предложения за търсене в нашето приложение.
Също така се нуждаем от свой собствен провайдер, ако искаме да копираме и поставяме комплексни данни или файлове от нашето приложение в други приложения.

Android съдържа провайдери на съдържание, които управляват данни като аудио, видео, изображения и персонална информация за контакти.
Можем да видим някои от тях, изброени в документацията за android.provider пакета.
С някои ограничения, тези провайдери са достъпни от всяко Android приложение.
Провайдерът на съдържание управлява достъпа до централно хранилище с данни.
Провайдерът е част от Android приложение, което често предостаява свой UI за работа с данните.
Провайдерите на съдържание са главно предназначени да бъдат използвани от други приложения, които достъпват провайдера, използвайки провайдер клиент обект.
Заедно, провайдерите и провайдер клиентите предлагат стабилен, стандартен интерфейс към данните, които поддържат междупроцесна комуникация и сигурен достъп до данни.
Провайдерът на съдържание представя данните на външни приложения като 1 или повече таблици, които са подобни с таблиците в релационните бази данни.
Един ред представлява инстанция на някакъв тип данни, които провайдерът съхранява, и всяка колона от реда представлява индивидуална част данни, съхранявани за инстанцията.

Достъп до данни:
Едно приложение достъпва данните от провайдер на съдържание чрез ContentResolver клиент обект.
Този обект има методи, които извикват идентично именувани методи в провайдер обекта, който е инстанция на един от подкласовете на ContentProvider.
ContentResolver обектът в клиент-процеса на приложението и ContentProvider обектът в приложението, който притежава провайдера, автоматично обработват междупроцесната комуникация.

ContentProvider също така действа като слой на абстракция между своето хранилище с данни и външното представяне на данните като таблици.

Пример:
- За да вземем списък с думите и техните места от UserDictionary Provider, можем да извикаме:
  ContentResolver.query();
- Методът query() извиква ContentProvider.query() метода, дефиниран от UserDictionary Provider
- Следните редове код показват извикване на ContentResolver.query():

```
// Заявки към UserDictionary и връщане на резултати
mCursor = getContentResolver().query(UserDictionary.Words.CONTENT_URI,
 // URI за съдържанието на таблицата с думи
mProjection, // Колоните, които да се върнат за всеки ред
mSelectionClause, // Критерий за селекция
mSelectionArgs, // Критерий за селекция
mSortOrder); // Сортираната подредба за върнатите редове
```

11. AdapterView и Adapters. Работа с адаптери. Примери.

AdapterView:
AdapterView е ViewGroup подклас, чиито дъщерни View се определеят от Adapter, който свързва AdapterView обект към данни от някакъв тип.
Обикновено се използват подкласове на AdapterView класа, вместо използването му директно.
Примерни подкласове на AdapterView класа:

- ListView
- Spinner
- Gallery

2 главни отговорности на AdapterView:

- Запълване на layout с данни (с помощта на Adapter)
- Обработка на потребителски действия – когато потребителят избере елемент, изпълни някакво действие:
  - + AdapterView.OnItemClickListener
  - + AdapterView.OnItemLongClickListener
  - + AdapterView.OnItemSelectedListener

Adapter:
Можем да напълним AdapterView като ListView или GridView чрез свързване на AdapterView инстанция към Adapter, който извлича данни от външен източник и създава View, което представя всеки запис данни.
Adapter осигурява достъп до елементите с данни.
Adapter е също така отговорен за създаване на View за всеки елемент в колекцията от данни.
Видове Adapter – имплементират ListAdapter интерфейса:

- ArrayAdapter
- CursorAdapter
- и други

12. SQLite база данни. Класове с работа с SQLite в Android приложение. Основни операции.

SQLite база данни:
Embedded RDBMS
Размер – около 257 KB
Архитектурата не е клиент/сървър. Достъпва се чрез извиквания на функции от приложението.
Записването (insert, update, delete) заключва базата данни, заявките могат да се изпълняват паралелно.
Android осигурява някои полезни помощни класове: SQLiteDatabase, Cursors, ContentValues и др.
Datastore(съхраняване на данни) – единичен, кросплатформен файл (Definitions, Tables, Indexes, Data).
Доста различна в сравнение с обикновените SQL типове данни.

android.database.sqlite:
Съдържа SQLite класове за управление на базата данни, които приложението може да използва, за да управлява своята лична база данни.
Съдържа класове и интерфейси за търсене/обработка на данните, върнати от провайдера на съдържание.
Съдържа методите за: създаване, отваряне, затваряне, вкарване, ъпдейтване, изтриване и quering в SQLite база данни.
Тези методи са подобни на тези в JDBC, но по-метод ориентирани в сравнение с тези в JDBC.
Главната функционалност, която може да се използва тук, е Cursor интерфейсът, който служи за извличане на данните от resultset, който се връща от заявка.
Класове: SQLiteCloseable, SQLiteCursor, SQLiteDatabase, SQLiteProgram, SQLiteQuery, SQLiteQueryBuilder, SQLiteStatement;

Основни операции:
openOrCreateDatabase(): Този метод ще отвори съществуваща база данни или ще създаде нова в областта за данни на приложението.
long insert(String table, String nullColumnHack, ContentValues values);
int update(String table, ContentValues values, String whereClause, String[ ] whereArgs);
int delete(String table, String whereClause, String[] whereArgs);
query(заявка): Метод от SQLiteDatabase класа, който изпълнява заявки върху базата данни и връща резултатите в Cursor обект:
Cursor c = mdb.query(p1,p2,p3,p4,p5,p6,p7), където:
◦ p1 : Име на таблицата (String)
◦ p2 : Колони, които да бъдат върнати (масив от String)
◦ p3 : WHERE клауза (null – за всички, ?s – за аргументи за избор)
◦ p4 : избор на стойности за аргументи за ?s от WHERE клаузата
◦ p5 : GROUP BY (null – за нищо) (String)
◦ p6 : HAVING (null, освен ако GROUP BY не изисква нещо) (String)
◦ p7 : ORDER BY (null – за подредба по подразбиране)(String)
◦ p8 : LIMIT (null – без лимит) (String)

13. Въведение в идеите на крос-платформеното програмиране. Обзор на най-популярни инструменти.

Прогнозата за крос-платформеното програмиране на пазара достига $7.5 милиарда до 2018 година.

Очаква се повече от 20 млн. приложения да бъдат разработени. Повече от 80% от потребителите смятат, че крос – платформените приложения са по – бързи за разработка и са финансово изгоден подход. Статистиката показва, че потенциала на крос платформените приложения расте всяка изминала година. Най – високият процент на използването им е в Европа 38%. Двете големи предимства на крос-платформеното програмиране са: финансова ефективност и спестяващи време. Някои от най – добрите крос – платформени мобилни инструменти за разработка са: Xamarin, PhoneGap, Sencha, appcelerator, iFactr, kony, alphaSoftware, redhat, React Native.

- **Xamarin** – Към момента съществуват повече от 1.4 млн разработчици и 15000 компании използващи Xamarin. Могат да се разработват native iOS, Windows, Android приложения, използвайки една обща C# основа. В C# може да се направи всичко, което се прави и в Objective – C, Swift, Java. Позволява използването на същите APIs, IDE и език, навсякъде. Също така предоставя инструмент за интерфейс разработка и онлайн обучение в Xamarin университетска програма. Теста се на различни устройства, като се използват cloud услугите на комапнията.
- **PhoneGap** – това е open source решение, базирано на проект на Apache Cordova и притежавано от Adobe. Абсолютно безплатно ползване. Помага на разработчиците да конвертират код, написан на HTML5, CSS, JavaScript до NativeOS. Поддържа няколко платформи като iOS, Android, Windows, WebOS с една обща база код. Предоставя разнообразие от plugins, от които може да се избере, позволяващи на разработчиците да използват своите умения. Съхранява софтуерни пакети за разработка, за всяка една от платформите създадени от разработчиците на приложение.
- **Sencha** – предлага разнообразие от инструменти за крос – платформена разработка на приложения, като Sencha Architect, Sencha Animator и други. Главният продукт за потребителите е Ext JS 6, който позволява на разработчиците да създават HTML5 приложения, които могат в последствие да се конвертират в native приложения с PhoneGap. Позволява приложенията да се подкарват на различни браузъри, както и на най

  – новите устройства базирани на докосване. Привлича клиенти като Google, CNN и Samsung. Платформата е оценена на $3.225 на година за до 5 разработчика и до $12.495 за година за до 20 разработчика.
- **ReactNative** – базирана на React, JavaScript библиотеката на Facebook за построяване на потребителски интерфейси, тази платформа таргетира мобилните платформи вместо браузър. Използва се за писане на реални, native iOS и Android приложения. Просто трябва да се напише 1 ReactNative компонент и той работи на множество платформи. Хиляди разработчици използват ReactNative. Може да се каже, че това е бъдещето на хибридните приложения.

14. Основни етапи при разработката на мултиплатформени мобилни приложения и особености на архитектурата им. Потребителски опит (User Experience). Прототипиране.

Основните етапи при разработката на мултиплатформени приложения, още наречено цикъл на живот, са следните:

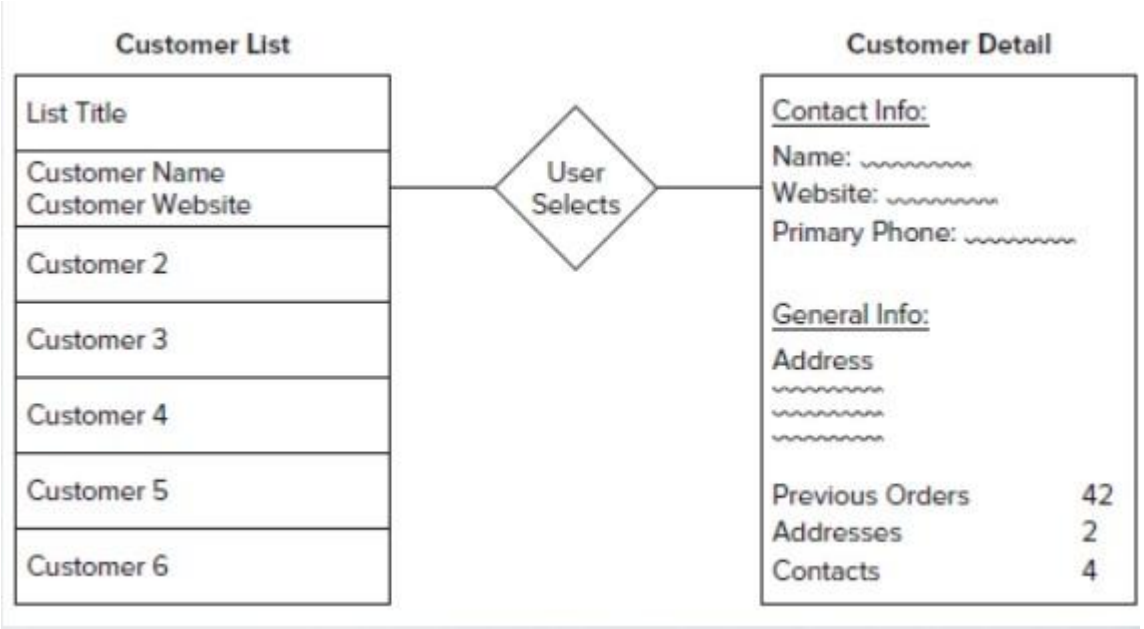
1) Дефиниция
2) Планиране
3) Дизайн
4) Разработка
5) Тестване
6) Deploy
7) Управление

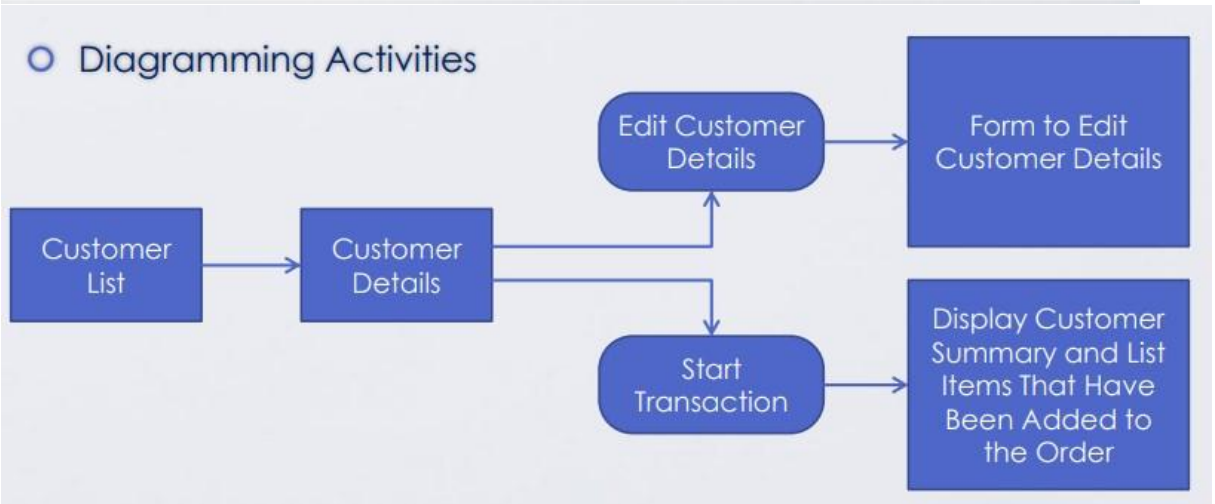
Когато говорим за архитектурата, трябва да се спрем на следните 3 неща: достъп на устройството, свързаност, използваемост.

- Относно свързването с мрежата съществуват опции за безжично свързване, като изолиран и защитен WiFi hotspot, или „извън" мрежата, което е по – малко надеждно. Идеята е да може да се работи от всякъде. Съществуват и offline схеми за кеширане на данни.
- Има ограничения във връзка със съхраняването и процесора. Ограничения в: service – ориентирани архитектури, системи за релационни бази данни и високо нормализирани данни. Множество извикване на услуги. Може да доведе до прекомерно процесорно зареждане на устройството, което се отразява на ефективността и използваемостта. Структурата на данните – XML, SOAP. Мобилните устройства лесно могат да се пренатоварят, особено, когато се работи с голям сет от информация. За да се намали риска, трябва да се внимава с количеството информация, което се предоставя на устройството и да се предоставя, само когато е нужно.
- Защитата на данните на устройството се осъществява чрез автентикация и оторизация (LDAP, SSO), криптиране на данните (SSL, VPN), унищожаване на данните (BYOD).
- Построяване на мащабируеми приложения – scaling up: това не е опция за мобилните приложения. Scaling out: добавяне на допълнителни нодове към група. Потребителят има единствено устройство, нито повече, нито по – малко. Мащабируемостта е построена на страната сървъра.
- Планиране на deployment – удобството на deploy на уеб приложенията се изразява в централния deploy и управлението на уеб приложението, лесна мащабируемост, ниска цена. Деплой на мобилните приложения – MDM (Mobile Device Management), Stores, Apple моделът за деплой.
- Писане на разширяеми модули – създаване на преизползваеми компоненти. Абстракцията на слоевете на устройствата и хардуера може да предостави поддръжка за множество функции и периферни устройства.
- Поддръжка на кода – ООП, енкапсулация, ОС, Storage. Трябва да се минимизира нуждата от поддръжка на множество алгоритми в общия код, като това което се отнася специфично за платформата, трябва да се компилира специално за нея.

User experience – приложението е използваемо, платформата е отделена от дизайна, прототипиране.

- Използваемост – приложенията трябва да са прости. Първо се създава главния поток на работа. Дизайнът се прави за по – малки резолюции на екрана, приложението се разбива на функционални части, идентифицира се информацията, която ще се използва, използва се по – едър шрифт за основната информация. Стандартите на платформата – използват се native контроли и конвенции в UI и интуитивен интерфейс.
- Разделяне на платформата от дизайна – преди да контролерът, е важно да се обмисли работния поток. Дизайнът се фокусира върху данните и задачите, лимитира се показваната информация.
- Прототипиране – бяла дъска (whiteboarding), решана се каква информация да се показва (диаграма на действията помага да се опише основната функционалност и изискваните данни), диаграма на действията, използване на функционални прототипи, обратна връзка от клиента, използване на agile итерации
- Whiteboarding





15. Видове архитектура на крос-платформеното приложение. Предимства и недостатъци. Сравнение с Native приложенията

Изборът на подходяща архитектура е свързан с редица фактори, като например тип на ОС.

За програмисти на C# и .NET, съществува MonoTouch и Mono за Андроид, както и уеб приложение, използващо HTML 5 и ASP.NET. Разработчиците имат и нов избор – Hybrid приложения.

Native приложения

- Предимства
  - Пълен достъп до гъвкавостта и пълнотата на потребителските интерфейс APIs
  - Пълен достъп до всички уникални функции на устройството (услуги за геолоккация, жироскоп, акселерометър, камера, видео запис, аудио запис, компас и други)
  - Работа в несвързан режим (самолетен)
- Недостатъци
  - Деплой посредством вътрешен app store, предоставен от MDM или build from scratch
  - Управление на провизиите на нови устройства
  - Увеличена цена на собственост

Web приложения

- Предимства
  - Работят на по - нови устройства
  - Form-factor не е проблем
  - Деплой е управляван централно
  - Крос-платформени възможности на HTML 5 – data-driven контроли, услуги за геолокация, HTML 5 кеш манифест стандарт
- Недостатъци
  - Липса н достъп до функции на устройства
  - Зависимост на връзката – офлайн информацията е само за употреба на ресурси
  - Няма storage или threading
  - Не поддържа богатото потребителско изживяване

Hybrid приложения

- Предимства
  - Доставяне на порции от приложението, използвайки уеб технологии и други порции, използвайки native контроли и APIs
  - Native “wrapper” приложение, което доставя съдържание посредством уеб
  - Достъпва чувствителна информация само чрез съществуващи уеб политики за защита

    - Гъвкавост, предимства от native и web приложенията
- Недостатъци
    - Изисква деплой на устройството както при native приложенията
    - Увеличава цената на управлението
    - Понижава ефективността поради навигация между native и web компоненти
    - Уеб съдържанието не отговаря на native user experience

16. От Mono до Xamarin: създаване и развитие. Възможности. Принцип на работа.

От Mono до Xamarin

Платформата Mono – Mono е open source имплементация на .NET Framework на Microsoft на ECMA стандартите за C# и Common Language Runtime. Върви на различни платформи - Linux, OS X, BSD, iPhone, Android, PlayStation 3, Wii, Xbox 360, и Microsoft Windows, включително x86, x86-64, ARM, s390, PowerPC, SPARC, IA64, MIPS и много други. Поддържа различни езици - C# 5.0, VB 8, Java, Python, Ruby, Eiffel, F#, Oxygen и други. Miguel de Icaza е мексикански програмист, известен със стартирането на проектите – GNOME и Mono. Той е лидер на GNOME, Ximian CTO, вице президент на Novell, Xamarin CTO. Mono се появява 2001 година с Beta версия (2004 – Mono 1.0, 2008 – 2.0,
2012 – 3.0, 2015 – 4.0). През 2009 – MonoTouch 1.0, 1.2 (C#, .NET Framework BCL, XIB CodeBehind и iOS native API, MonoDevelop с Interface Builder, GC, Soft дебъгер). 2011 – MonoMac 1.0, Mono за Android 1.0.
Xamarin стартира май, 2011. Xamarin е компанията, спонсорираща разработването и поддръжката на Mono.

Xamarin възожности

Xamarin има:

- почти пълна поддръжка на платформените SDKs на iOS и Android
- Objective-C, Java, C, and C++ Interop – директно изпълнение на библиотеките
- Modern Language Constructs – възможност за използване на Lambdas, LINQ , Parallel Programming,Generics
- Base Class Library (BCL) – използва .NET BLC, голям набор от класове (напр. XML, Database, Serialization, IO, String и Networking)
- Modern Integrated Development Environment (IDE) - Visual Studio for Mac, Visual Studio on Windows
- Mobile Cross Platform Support – споделяне на до 90% от кода. Библиотеката Xamarin.Mobile унифицира достъпа до ресурси

Принцип на работа

Как работи Xamarin-два комерсиални продукта: Xamarin.iOS и Xamarin.Android:

- iOS – AOT (Ahead Of Time) компилаторът компилира Xamarin.iOS приложения директно до ntive ARM код; .app файл; MonoTouch.dll
- Android - Xamarin компилаторът превежда до Intermediate Language (IL), който при стартиране на приложението с Just-in-Time (JIT) се компилира до native assembly; .apk file; Mono.Android.dll

17. Разработване на приложения с Xamarin. Софтуерни шаблони и слоеве. Механизми за споделяне на код.

Разработване на приложения с Xamarin

Разработката на приложения с Xamarin е свързана с:

- Езикът C# - познат синтаксис, сложни конструкции като Generics, LINQ и Parallel Task Library
- Mono .NET framework - cross-platform функционалности от Microsoft's .NET
- Compiler – компилатора създава нативно приложение (iOS) или runtime (Android)
- IDE tools

Построяване на приложението:

- Използване на MVC или MVVM шаблон за дизайн ("Core" и "UserInterface")
- Построяване на native Uis - OS-специфичен потребителски-интерфейс слой (UIKit APIs – iOS, Android.Views – Android
- Xamarin.Forms – XAML

Общи услуги и компоненти

- SQLite-net за локално съхраняване на SQL
- Xamarin Plugins за достъп до специфични възможности на устройството като камера, котгакти, геолоация
- NuGet пакети, които са съвместими с Xamarin проекти, като Json.NE
- Използване на .NET framework свойства за свързване с мрежата, уеб услуги, IO и други

Слоеве в приложенията

- Data Layer – неволетилно запазване на данните (SQLite database, XML files или др.)
- Data Access Layer – Wrapper около Data Layer, който предоставя Create, Read, Update, Delete (CRUD) достъп до данните
- Business Layer – (Business Logic Layer or BLL) съдържа entity дефинициите (Model) и бизнес логиката.
- Service Access Layer – използва се за достъп до услуги в облака: от сложни web services (REST, JSON, WCF) до просто извличане на данни и изображения от отдалечени сървъри. Капсулира мрежовите операции и предоставя просто API което да се консумира от приложението или UI слоя
- Application Layer – Код, който обикновено е платформено зависим (не се споделя и преизползва). За да се прецени дали кодът трябва да е в Application Layer а не в UI Layer
  - (a) да се определи дали класа има някакви визулани елементи
  - (b) или дали може да бъде споделен между различни екрани или устройства (напр. iPhone и iPad).
- User Interface (UI) Layer – съдържа екрани, уиджети и контролерите, които ги управляват.

Софтуерни шаблони

- Model, View, ViewModel (MVVM) – framework, поддържащ data-binding.
- Model, View, Controller (MVC) – най-често се използва за конструирането на User Interfaces позволява разделяне между дефиницията на UI екрана (View) и механизма отзад, който контролира взаимодействието с него (Controller) и данните, които се зареждат (Model). MVC е популярен подход за iOS приложенията.
- Singleton – шаблонът предоставя подход при който само една инстанция на определен обект може да съществува. Например, ако се използва SQLite и е необходимо да има само една негова инстанция, това лесно може да се осигури със Singleton.
- Async – (да не се бърка с ключовата дума Async) Шаблонът се използва при дълго- изпълняващи се задачи, които не трябва да задържат UI. В най- простата си форма Async шаблонът просто описва това че задачите трябва да бъдат изнесени в друга нишка (или Task) докато основната нишка продължи да обработва и слуша за отговор от background процеси, след което опрецнява UI.

Механизми за споделяне на код

- Shared projects – използване на Shared Asset Project тип за организиране на сорс кода, използване на #if компилаторни директиви, за управление на платформено специфичните изисквания. Предимствата са, че позволява споделянето на код сред множество платформи. Споделеният код може да се отдели на бранч. Проектите могат да включват платформено специфични референции, които споделеният код да използва. Недостатъци
  – противно на повечето типове проекти, Shared Project няма output assembly. По време на компилация, файловете са приемани като част от реферираща проект и се компилират в това асембли. Рефакториране, което афектира код в “inactive” компилаторски директиви, няма да ъпдейтне кода.
- Portable Class Libraries – създаване на PCL, като се таргетирт платформите, които искаме да поддържа. Използват се интерфейси, за да се предостави платформено специфична функционалност. Предимствата са, че позволява споделянето на код и че рефакториращите операции винаги ъпдейтват кода. Недостатъцита са, че не могат да се ползват компилаторски директиви и, че само подсет на .NET framework е достъпен за употреба.
- .NET Standard Libraries - .NET стандартните проекти, работят подобно на PCL, като изискват използването на интерфейси, за да инжектират платформено специфична функционалност. Предимствата са, че позволява споделянето на код, рефакториращите операции ъпдейтват кода и, че е достъпна по – голяма част от .NET Base Class Library отколкото при PCL. Недостатъкът, е че не може да използва компилаторски директиви.

18. Връзка между данни и визуален интерфейс в Xamarin приложения. Data binding. Примери. Жестове.

Връзка между данни и визуален интерфейс в Xamarin приложения. Data binding

Data Binding е техника за свързване на пропъртита на 2 обекта, така че промяна в едно пропърти, се отразява в другото. Data Binding е част от MVVM архитектурата.

Пример:

```
<StackLayout Padding="10, 0">
        <Label x:Name="label"
                Text="TEXT"
                FontSize="48"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand"
                BindingContext="{x:Reference Name=slider}"
                Rotation="{Binding Path=Value}"/>
        <Slider x:Name="slider"
                Maximum="360"
                VerticalOptions="CenterAndExpand" />
</StackLayout>
```

Binding режим – режимът се определя с няколко BindingMode енумерации:

- Default – специфично за всяко отделно свойство
- TwoWay – данните се предават двупосочно между източника и целта
  - Date (DatePicker), Text (Editor, Entry, SearchBar, EntryCell), Value (Slider, Stepper)
- OneWay – данните се предават от източника към целта
- OneWayToSource – данните идват от целта към източника
- SelectedItem property на ListView

Жестове

- Tap – възможността да се натисне върху елемент на екрана (не само бутони)
- Pinch – дава възможност за zoom на конкретен елемент
- Pan – представлява възможността даден елемент да бъде „влачен“ на екрана

Примери

- Tap C#

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) => {
// handle the tap
};
image.GestureRecognizers.Add(tapGestureRecognizer);
// double-tap
tapGestureRecognizer.NumberOfTapsRequired = 2;
```

- Tap XAML

```
<Image Source="tapped.jpg">
        <Image.GestureRecognizers>
                <TapGestureRecognizer
                        Tapped="OnTapGestureRecognizerTapped"
                        NumberOfTapsRequired="2" />
        </Image.GestureRecognizers>
</Image>
void OnTapGestureRecognizerTapped(object sender, EventArgs args)
{
        tapCount++;
        var imageSender = (Image)sender;
        // watch the monkey go from color to black&white!
        if (tapCount % 2 == 0) imageSender.Source = "tapped.jpg";
        else imageSender.Source = "tapped_bw.jpg";
}
```

- Tap ICommand

```
<Image Source="tapped.jpg">
        <Image.GestureRecognizers>
                <TapGestureRecognizer
                        Command="{Binding TapCommand}"
                        CommandParameter="Image1" />
        </Image.GestureRecognizers>
</Image>
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.SetBinding(TapGestureRecognizer.CommandProperty,"TapCommand);
image.GestureRecognizers.Add(tapGestureRecognizer);
```

- Pinch

```
public class PinchToZoomContainer : ContentView
{
        public PinchToZoomContainer()
        {
```

```
                var pinchGesture = new PinchGestureRecognizer();
                pinchGesture.PinchUpdated += OnPinchUpdated;
                GestureRecognizers.Add(pinchGesture);
        }
    void OnPinchUpdated(object sender, PinchGestureUpdatedEventArgs e)
    { . . . }
    <local:PinchToZoomContainer>
        <local:PinchToZoomContainer.Content>
            <Image x:Name="image1" Source="xamarin.png" />
        </local:PinchToZoomContainer.Content>
    </local:PinchToZoomContainer>
```

- Pan

```
    public class PanContainer : ContentView
    {
        double x, y;
        public PanContainer()
        {
            // Set PanGestureRecognizer.TouchPoints to control the
            // number of touch points needed to pan
            var panGesture = new PanGestureRecognizer();
            panGesture.PanUpdated += OnPanUpdated;
            GestureRecognizers.Add(panGesture);
        }
        void OnPanUpdated(object sender, PanUpdatedEventArgs e)
        {
            ...
        }
    }
    <ContentPage.Content>
        <AbsoluteLayout>
            <local:PanContainer>
                <Image Source="MonoMonkey.jpg“
                    WidthRequest="1024"
                    HeightRequest="768" />
            </local:PanContainer>
        </AbsoluteLayout>
    </ContentPage.Content>
```

19. Навигация между страници в Xamarin приложения. Видове навигация. Навигационен стек. Предаване на данни при навигация. Примери.

NavigationPage класът предоставя йерархично навигационно изживяване, където потребителят

може да навигира между страниците, напред и назад, както пожелае. Класът имплементира навигацията като LIFO стек от Page обекти. В Android и iOS, навигационната лента се намира в горната част на страницата където се показва заглавието й. В нея има Back бутон за връщане към предишна страница. Предоставени страници: ContentPage, MasterDetailPage, NavigationPage, TabbedPage (състои се от табове и по – голяма зона за дейтли, като всеки таб зарежда съдържание в таи зона), CarouselPage (потребителите могат да я суайпват от страна до страна, за да навигират между страниците, като галерия), ModalPages (потребителите трябва да завършат или откажат определено нещо преди да могат да навигират), Pop-ups (alert, action sheet).a

Създаване на Root страница

```
public App ()
{
	MainPage = new NavigationPage(new WelcomePage());
}
```

Добавяне на страници в навигационния стек

```
async void OnNextPageButtonClicked(object sender, EventArgs e)
{
	await Navigation.PushAsync(new Page2Xaml());
}
```

Събития при PushAsync – редът им е платформено зависим

- Страницата викаща PushAsync има собствен OnDisappearing override, който се вика
- Страницата, до коятос е навигира има собствен OnAppearing override, който се вика
- PushAsync задачата привършва

Изваждане на страници от навигационния стек

```
async void OnPrevPageButtonClicked(object sender, EventArgs e)
{
	await Navigation.PopAsync();
	// await Navigation.PopToRootAsync();
}
```

Събития при PopAsync – редът им е платформено зависим

- Страницата викаща PopAsync има собствен OnDisappearing override, който се вика
- Страницата, до която се навигира има собствен OnAppearing override, който се вика
- PopAsync задачата привършва

Анимиране на преходи между страници

```
async void OnNextPageButtonClicked(object sender, EventArgs e)
{
	// Page appearance not animated
	await Navigation.PushAsync(new Page2Xaml(), false);
}
async void OnPreviousPageButtonClicked(object sender, EventArgs e)
{
	// Page appearance not animated
	await Navigation.PopAsync(false);
}
async void OnRootPageButtonClicked(object sender, EventArgs e)
{
	// Page appearance not animated
```

```
        await Navigation.PopToRootAsync(false);
}
```

Предаване на данни при навигация

- Чрез параметри на конструктора

```
        public App()
        {
                MainPage = new NavigationPage(
                new MainPage(DateTime.Now.ToString("u")));
        }
        public MainPage(string date)
        {
                InitializeComponent();
                dateLabel.Text = date;
        }
```

- Чрез BindingContext

```
        async void OnNavigateButtonClicked(object sender, EventArgs e)
        {
                var contact = new Contact { ... };
                var secondPage = new SecondPage();
                secondPage.BindingContext = contact;
                await Navigation.PushAsync(secondPage);
        }
```

Манипулиране на навигационния стек

```
async void OnLoginButtonClicked(object sender, EventArgs e)
{
        ...
        var isValid = AreCredentialsCorrect(user);
        if (isValid)
        {
                App.IsUserLoggedIn = true;
                Navigation.InsertPageBefore(new MainPage(), this);
                await Navigation.PopAsync();
        }
        else
        { /* Login failed */ }
}
```

20. Локализация и интернационализация Проблеми и приложение. Подходи за постигане на многоезичност.

Локализация и интернационализация. Проблеми и приложение

Локализация е част от процес, който позволява пускането на приложението на различни пазари.

Пазарите за приложения са глобални, каквото трябва да бъде и приложението. Локализацията е процесът на адаптация на приложението от един на много езици. Отделени стрингове и други ресурси като изображения, видео, аудио в приложението, така че да могат да се преведат на различни езици, които ще се поддържат.

Интернационализация – процес на адаптиране на приложението, за да работи в различни региони и държави с една и съща база на кода. Например: форматиране на числа – знак за десетични числа (, .), запис на дати – 12/05/2018 или May 12th, 2018.

Language String се предоставя на приложението от ОС. Повечето мобилни поддържат сет от държави и езици:

- С iOS, Apple поддържа повече от 30 езика за ОС, но потребителят може да избере който език и регион пожелае.
- Езиците поддържани от устройство варират. По – новите версии на ОС обикновено добавят повече езици.
- Някои продавачи лимитират поддържаните езици в региона си.

Има няколко неща, които трябва да се съобразят допълнително:

- Разлики в обема на потребление на различни продукти:
  - Например потребителите в Китай нямат достъп до Facebook и Twitter. На тяхно място се използват WeChat и Weibo. (>300 милиона потребители)
- Колко езика да се поддържат?
  - Договорни задължения
  - Големи компании като Facebook поддържат повече от 70 езика
  - За малки екипи може да се изберат езици в зависимост от броя хора които ги говорят

Подходи за постигане на многоезичност

Многоезичност – при превода трябва да се отчетат следните неща:

- Отделяне на текста и други елементи на приложението (изображения, видео, аудио) от кода на програмата
- Предимства на отделянето на text strings в ресурсен файл:
  - Приложенито избира подходящия език по ввреме на изпълнение
  - Възможност за предоставяне само на текстовите ресурси на преводач
- Особености на езика
  - Превод на цели изречения и фрази
  - Родове и числа, съгласуване
  - Right-To-Left поддръжка (Арабски). За да се активира RTL поддръжка.
- Layout особености
  - Позициониране на label-ите НАД свързаните с тях полета -без проблеми при RTL, без проблеми при различна дължина при превод на текста
    - Xamarin.Forms -> the StackLayout control
    - Xamarin.iOS -> the vertical UIStackView
    - Xamarin.Android -> the LinearLayout
- Контекст - Думите и фразите могат да имат различно значение в зависимост от контекста, но само едно от тях да е правилното при конкретната употреба. Добре е да се предоставят обяснения на текста за превод и/или Screenshots стартирано приложение в оригиналния език. Не всичко е задължително да се преведе буквално. Възможно е предлагането на други термини, за да се запази приблизително дължината на показвания текст.
- Форматиране на числата - .NET Framework поддържа глобалните числови формати

- Валути
- Дати и време
- Главни и малки букви – различните езици използват главни букви на различни места. Например Английският и Немският капитализират дните от седмицата, месеците, докато други езици, не.
- Сортиране – същия набор от букви може да се сортира по един начин на един език, и по съвсем друг, на друг език.
- Цветове и изображения - Избягвайте ползването на национални флагове за отбелязване на култура. Трябва да се внимава с цветовите палитри. Видео и аудио и възможност за субтитри.
- Валидация на входни данни – трябва да се избягва валидационно правило, което изисква минимум брой букви. В някои езици имената могат да са само с 1 буква.

21. Работа с ресурсни файлове. Примерна структура и съдържание. Местоположение и именуване на файловете в различните ОС.

Файлови формати:

- Приложения написани с .NET Framework използват RESX формат
- Xamarin.Android приложенията могат също да използват native Androif strings.xml файл
- Приложения написани с Xamarin.iOS мога да ползват native LocalizedString файлове

RESX – XML базиран. Освен .resx файл към проекта се добавя и code-behind file със същото име но разширение .designer.cs. Когато се компилира приложението инструмента resgen.exe конвертира ресурсите до клас в приложението. Този клас позволява реферирането на ресурсни стрингове като .NET свойства.

Пример:

```
public static string EyeColor
{
	get
	{
		return ResourceManager.GetString("EyeColor", resourceCulture);
	}
}
```

```
<TextBlock Text="{x:Static MyText.EyeColor}" />
```

```
eyeLabel.Text = MyText.EyeColor;
```

Търси се локализиран стринг подобен на Eye color.

Android XML – по конвенция, файлът се казва strings.xml и се съхранява в res/values папката. Xamarin Android приложенията ползват същите файлове, но папката и Resources/values.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
	<string name="Name">Name</string>
	<string name="Age">Age</string>
	<string name="EyeColor">Eye Color</string>
</resources>
```

Resources/values-es папка

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
	<string name="Name">Nombre</string>
	<string name="Age">Años</string>
	<string name="EyeColor">Color de los ojos</string>
</resources>
```

Взимане на стринг в Андроид: в кода се използва GetString() метод

```
<Button
	android:id="@+id/myButton"
	android:layout_width="fill_parent"
	android:layout_height="wrap_content"
	android:text="@string/EyeColor"
/>
```

```
eyeLabel.Text = context.Resources.GetString(Resource.String.EyeColor);
```

Apple iOS String речник – с iOS, Apple ползва различен механизъм за локализиране на стринг ресурси. Вместо XML – базиран формат за съхранние, Apple използва ключ – стойност двойки, и дефолтното има на файла е Localizable.strings, но може да се използва всякакво валидно име. Локацията е в xx.lproj папка, където xx е името на езика.

String extension – добавя t() метод към всички стрингове.

22. Apache Cordova (Phonegap): създаване и развитие. Възможности. Принцип на работа. Архитектура.

Apache Cordova (Phonegap): създаване и развитие

Посредством крос – платформено мобилно програмиране с Apache Cordova се създават мобилни приложение за различни платформи.

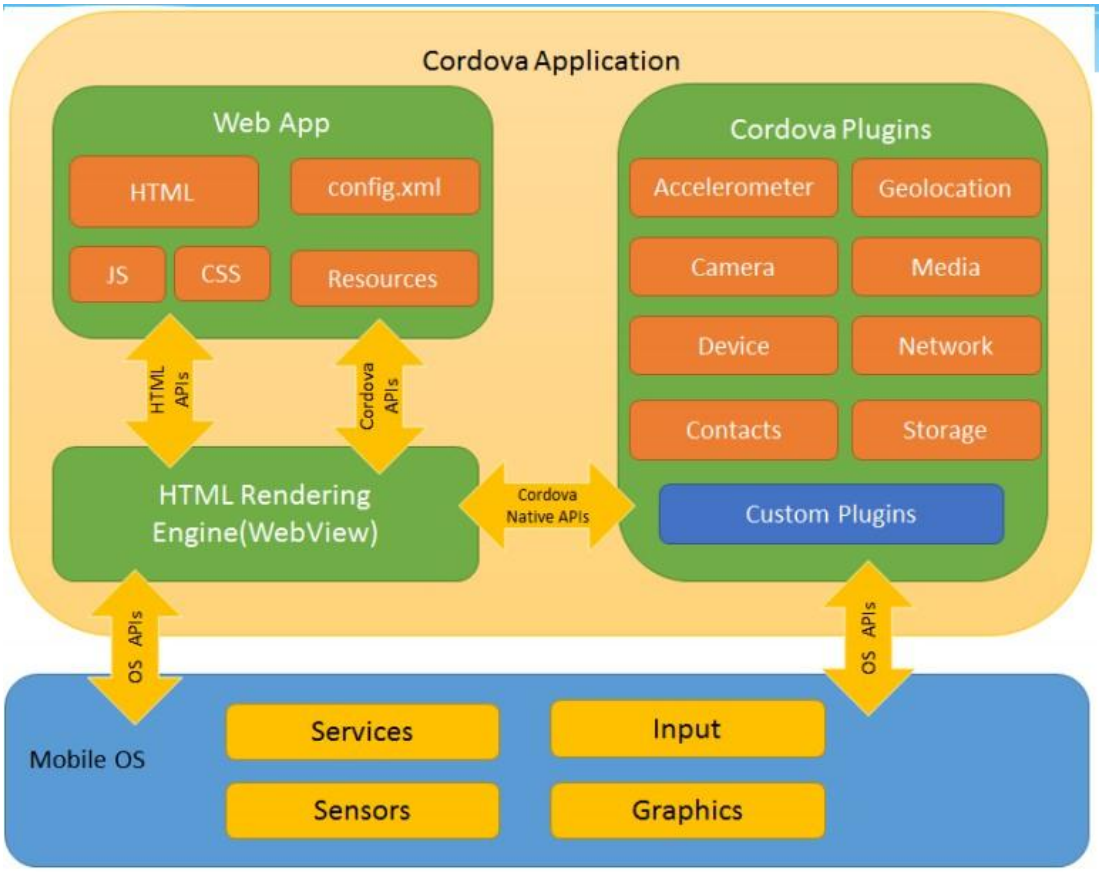
PhoneGap е open-source мобилен framework за разработка, създаден от Nitobi на iPhoneDevCamp през 2009 година. След това Adobe купуват Nitobi през 2011. Apache Cordova е бил оригинално именуван Phonegap. Open-source и безплатен софтуер в началото, сега има Apache лиценз. Nitobi дарява PhoneGAP базата на кода на Apache Software Corporation. PhoneGap все още е продукт на Adobe. Apache Cordova е двигателят, задвижващ PhoneGap.

Възможности

Използва стандартни уеб технологии – HTML 5, JavaScript за крос – платформена разработка. Приложенията се изпълняват в wrapper-и таргетирани към всяка платформа. Разчита на стандарти, отговарящи на API bindings, за да достъпва всички възможности на устройствата, като сензори, данни, статус на мрежа.

Използва се, когато разработчик иска да разшири приложение отвъд повече платформи, без да се наложи да го преимплементира с всеки език на платформите и спрямо различните им инструменти. Също се използва, когато разработчик иска да деплойне уеб приложение, което е пакетирано за дистрибуция в различни app store портали или, когато се смесват компоненти на native приложение и WebView, което може да достъпи APIs.

Архитектура



23. Основни файлове и структура на Apache Cordova приложение. Плъгини. Пример за достъп до геолокацията.

Основни файлове и структура на Apache Cordova приложение

Структура на приложение: PhoneGap приложенията са хибридни. Те не са нито native, нито напълно web based. Рендерирането на layout се осъществява посредством web view.

Web View – Cordova WebView може да предостави целия интерфейс на приложението или компонент от хибридно приложение, който смесва WebView с компоненти на native приложение. UI слоят е уеб браузър view, което заема 100% от височината и ширината на устройството. Web view – то, използвано от приложението е същото web view, използвано от native ОС.

Web App – това е частта, където стои кода на приложението. Приложението, само по себе си е имплементирано като уеб страница. По подразбиране има локален файл index.html, който реферира CSS, JavaScript изображения, медийни файлове или други ресурси. Приложението се изпълнява в WebView в рамките на native wrapper на приложението, което се дистрибутира на app store. Този контейнер има много важен файл – config.xml, който предоставя информация за приложението и специфицила параметри, които афектират начина, по който работи, като например дали отговаря на промяна на ориентацията.

Плъгини

Plugins – интерфейс за Cordova и native компоненти за взаимна комуникация и свързване към стандартни APIs на устройство. Позволява извикването на native код от JavaScript. Apache Cordova проектът, съдържа сет от плъгини, наричани Core Plugins (батерия, камера, контакти). Други плъгини, които предоставят допълнителни bindings към features, не винаги са достъпи да всички платформи. Могат да се разработват и собствени плъгини. Когато се създава Cordova проект, няма налични плъгини. Това е ново дефолтно поведение. Всякакви нужну плъгини, дори core, трябва да се добавят.

Пример за достъп до геолокацията

```
<!DOCTYPE html>
<html>
	<head>
		<title>Device Properties Example</title>
		<script type="text/javascript" charset="utf-8" src="cordova-2.0.0.js"></script>
		<script type="text/javascript" charset="utf-8">
		// Wait for Cordova to load
		document.addEventListener("deviceready", onDeviceReady, false);
		// Cordova is ready
		function onDeviceReady() {
		navigator.geolocation.getCurrentPosition(onSuccess, onError);
		}
		// onSuccess Geolocation
		function onSuccess(position) {
			var element = document.getElementById('geolocation');
			element.innerHTML = 'Latitude: ' + position.coords.latitude + '<br />' +
					'Longitude: ' + position.coords.longitude + '<br />' +
					'Altitude: ' + position.coords.altitude + '<br />' +
					'Accuracy: ' + position.coords.accuracy + '<br />' +
					'Altitude Accuracy: ' + position.coords.altitudeAccuracy
				+ '<br />' +
		}
	// onError Callback receives a PositionError object
	function onError(error) {
		alert('code: ' + error.code + '\n' + message: ' + error.message + '\n');
	}
	</script>
</head>
<body>
	<p id="geolocation">Finding geolocation...</p>
</body>
</html>
```