Билет 1

1. Основные платформы мобильных приложений Android и iOS: история развития, версии, отличия, архитектура.

Android:

* История: Создан Android Inc. (2003), куплен Google (2005). Первый телефон – HTC Dream (2008).
* Ключевая особенность: Открытый исходный код. Это значит, что разные производители могут менять систему под себя.
* Версии: У каждой версии название десерта
* Архитектура: Основан на ядре Linux.
* Плюсы: Более гибкий, много разных устройств, часто дешевле.

iOS:

* История: Разработан Apple и представлен вместе с iPhone в 2007.
* Ключевая особенность: Закрытый исходный код. Apple полностью контролирует систему.
* Версии: Просто iOS 14, iOS 15 и т.д.
* Архитектура: Основан на macOS.
* Плюсы: Более стабильный, единый стиль, хорошая оптимизация под "железо".

Отличия:

* Основное отличие между Android и iOS заключается в том, что Android - это открытая операционная система, в то время как iOS - это закрытая операционная система.
* Интерфейс Android более настраиваемый, пользователь может менять различные настройки, устанавливать сторонние приложения и дополнения. В то время как интерфейс iOS более ограничен и ориентирован на улучшенную работу только с приложениями из App Store.
* Android поддерживает различные производители устройств, включая Samsung, Huawei, Xiaomi и другие, в то время как iOS работает только на устройствах Apple - iPhone, iPad и iPod Touch.
* В Android используется язык программирования Java, Kotlin и C++ для разработки приложений, в то время как в iOS используется язык программирования Objective-C или Swift.

Архитектура:

Архитектура Android основана на ядре Linux. Она состоит из четырех основных слоев: ядра Linux, аппаратного слоя, фреймворка приложений и пользовательского интерфейса.

Архитектура iOS состоит из ядра операционной системы, фреймворков, аппаратного слоя и пользовательского интерфейса. Она также имеет свою файловую систему и систему безопасности.

**Билет 2**

1. Нативные, гибридные и кроссплатформенные приложения мобильных устройств: понятие, примеры, технология создания.

**Нативные приложения:**

Понятие: Нативные приложения разрабатываются специально для определенной платформы (например, iOS или Android), используя собственные языки программирования и инструменты разработки, предоставленные производителем платформы.

Технология создания: Для разработки нативных приложений на iOS используется язык программирования Swift или Objective-C, а для Android - Java или Kotlin. Примеры приложений: Pinterest и Twitter (Kotlin), WhatsApp и Instagram (Swift)

**Гибридные приложения:**

Понятие: Гибридные приложения создаются с использованием веб-технологий, таких как HTML, CSS и JavaScript, и запускаются внутри специальной обертки, которая позволяет им работать на разных платформах. Примеры: Airbnb, Uber.

Технология создания: Разработка гибридных приложений может осуществляться с использованием различных фреймворков, таких как Apache Cordova (PhoneGap), React Native и Ionic.

**Кроссплатформенные приложения:**

Понятие: Кроссплатформенные приложения разрабатываются с использованием одной базовой кодовой базы и могут работать на разных платформах без изменений в исходном коде. Примеры: Skype, Facebook (React Native), Google Play (Flutter),The World Bank, Alaska Airlines (Xamarin)

Технология создания: Для создания кроссплатформенных приложений широко используются фреймворки, такие как React Native, Flutter и Xamarin.

**Билет 3**

1. Веб-приложения мобильных устройств: понятие, технология создания веб-приложения, технология создания веб-сервиса.

Веб-приложение — это программное обеспечение, к которому можно получить доступ через веб-браузер на компьютере или мобильном устройстве. В отличие от нативных приложений, которые устанавливаются на устройство, веб-приложения работают на удаленном сервере и не требуют установки. Пользователи могут взаимодействовать с веб-приложениями, вводя данные и получая результаты.

Технология создания веб-приложений включает использование таких языков программирования, как HTML, CSS и JavaScript. HTML используется для создания структуры веб-страницы, CSS - для оформления и стилизации элементов страницы, а JavaScript - для создания интерактивности и динамического поведения веб-приложения.

Веб-сервис — это программная система, предназначенная для поддержки взаимодействия между различными машинами в сети. Веб-сервисы используют стандартизированные протоколы, такие как HTTP, SOAP и REST, для обмена данными между клиентом и сервером.

При создании веб-сервиса используются следующие технологии:

* Языки программирования: Java, Python, PHP и другие.
* Форматы данных: XML, JSON.
* Протоколы: HTTP, SOAP, REST.

Веб-сервисы могут предоставлять различные функциональные возможности, такие как доступ к базам данных, выполнение бизнес-логики и интеграцию с другими приложениями.

Основным отличием между веб-приложениями и веб-сервисами является то, что веб-приложения выполняются на стороне клиента (мобильного устройства), в то время как веб-сервисы выполняются на стороне сервера и предоставляют функциональность или данные, которые могут быть использованы клиентскими приложениями.

**Билет 4**

1. Публикация мобильного приложения: подготовка, компиляция, создание пакета.

**Подготовка:**

Проверить, что приложение готово к публикации и соответствует всем требованиям платформы, для которой оно разработано (например, iOS или Android).

Убедиться, что приложение прошло все необходимые тестирования и исправлены все выявленные ошибки и проблемы.

Подготовить все необходимые ресурсы и файлы, такие как иконки, изображения скриншотов и описания приложения.

**Компиляция:**

Если приложение разрабатывается для iOS, скомпилировать его в Xcode, используя подходящие настройки и профили разработчика. Убедиться, что приложение успешно компилируется без ошибок и предупреждений.

Если приложение разрабатывается для Android, скомпилировать его в Android Studio или другой среде разработки для Android. Убедиться, что компиляция проходит успешно и без ошибок.

**Создание пакета:**

Для публикации приложения в App Store (iOS) необходимо создать подписанный архив (.ipa файл) с использованием соответствующего профиля разработчика. Этот подписанный архив будет загружен в iTunes Connect для дальнейшей публикации в App Store.

Для публикации приложения в Google Play Store (Android) необходимо создать подписанный APK файл. Потребуется ключ для подписи приложения, который можно создать в Google Play Console. После создания подписанного APK файла его можно загрузить в Google Play Console для публикации в Play Store.

**Билет 5**

1. Основные языки для разработки мобильных приложений (Java, Objective-C, С#). Взаимодействие XAML и C#.

**Java:**

Один из самых популярных языков для разработки Android-приложений.

Обладает высокой универсальностью и поддерживает объектно-ориентированное программирование.

Преимущества включают богатую экосистему и большое количество библиотек, что облегчает разработку сложных приложений. Недостатком является более сложный синтаксис по сравнению с современными языками, такими как Kotlin[1](https://fortech.dev/blog/yazyki-programmirovaniya-dlya-mobilnyh-platform/)[3](https://firecode.ru/blog/na-chem-razrabatyvat-mobilnoe-prilozhenie).

**Objective-C:**

Основной язык для разработки приложений под iOS до появления Swift.

Поддерживает объектно-ориентированное программирование и имеет мощные возможности для работы с графикой и пользовательским интерфейсом.

Хотя Objective-C все еще используется, большинство новых приложений разрабатываются на Swift, который предлагает более современный синтаксис и улучшенную безопасность[2](https://sky.pro/wiki/javascript/na-kakom-yazyke-pisat-mobilnye-prilozheniya-sovety-i-primery/)[4](https://creazard.ru/mobilnye-prilozheniya/yazyki-i-tehnologii-razrabotki-prilozhenii/).

**C#:**

Используется в основном для разработки кроссплатформенных приложений с помощью фреймворка Xamarin.

Позволяет создавать приложения как для iOS, так и для Android с использованием одного кода.

Преимущества включают возможность использования существующих библиотек .NET и удобный синтаксис. Однако производительность может быть ниже, чем у нативных приложений[5](https://notissimus.com/16-luchshih-yazykov-programmirovaniya-dlya-razrabotki-mobilnyh-prilozhenij-2023/)[6](https://practicum.yandex.ru/blog/yazyki-programmirovaniya-dlya-mobilnoy-razrabotki/).

**Взаимодействие XAML и C#**

XAML используется для описания пользовательского интерфейса в приложениях, разработанных с использованием C#. Это позволяет разделить логику приложения (написанную на C#) от его визуального представления (определяемого в XAML).

**Преимущества взаимодействия:**

Упрощение разработки: разработчики могут сосредоточиться на логике приложения, в то время как дизайнеры могут работать над интерфейсом.

Поддержка привязки данных: XAML позволяет легко связывать данные с элементами интерфейса, что упрощает обновление UI при изменении данных в C#.

**Билет 6**

1. Инструмент разработки мобильных приложений: JDK, Android Studio, установка, компоненты, настройка, структура проекта, запуск приложения.

JDK (Java Development Kit) - это набор инструментов, необходимых для разработки Java-приложений, включая мобильные приложения на Android.

Android Studio - это интегрированная среда разработки (IDE), специально созданная для разработки приложений под Android. Она предоставляет удобные инструменты для создания, отладки и тестирования мобильных приложений Android.

**Установка JDK:**

Перейдите на официальный сайт Oracle и загрузить последнюю версию JDK для нужной ОС. Запустить инсталлятор и следуйте инструкциям по установке.

**Установка Android Studio:**

Перейдите на официальный сайт Android Studio и загрузить последнюю версию Android Studio для нужной ОС. Запустить инсталлятор и следуйте инструкциям по установке.

**Компоненты Android Studio:**

Android SDK - коллекция инструментов и библиотек для разработки приложений под Android.

Android Virtual Device - эмулятор Android-устройств для тестирования приложений.

Gradle - система сборки и управления зависимостями проекта.

Android Debugger - инструмент для отладки приложений под Android.

Android Profiler - инструмент для анализа производительности и использования ресурсов приложения.

**Компоненты JDK:**

JDK включает следующие компоненты: среда выполнения Java, компилятор Java и API Java.

**Настройка Android Studio:**

Запустить Android Studio и следовать инструкциям мастера установки.

Выбрать путь для установки Android SDK и установите необходимые платформы Android.

Создать или войти в учетную запись разработчика Google для доступа к дополнительным функциям и сервисам.

**Настройка JDK:**

* Откройте IntelliJ IDEA.
* Перейдите в «Файл» > «Структура проекта» > «SDKs».
* Добавьте новый JDK, указав путь к установленному JDK.
* Убедитесь, что JDK выбран для проекта.

**Структура проекта Android Studio:**

app/src/main/java - исходный код приложения.

app/src/main/res - ресурсы приложения(макеты, изображения, строки, стили)

app/src/main/AndroidManifest.xml - манифеста, содержащий информацию о приложении, его компонентах и разрешениях.

app/build.gradle - конфигурация проекта, в котором определяются зависимости, настройки версий и другие параметры.

**Структура проекта JDK**

Разделение исходного кода и тестового кода.

Следование стандартам именования пакетов. Это улучшает читаемость кода и облегчает навигацию в базе кода.

Использование Maven или Gradle для управления зависимостями.

Применение принципов модульного проектирования.

Документация кода.

**Запуск приложения Android studio:**

Подключите устройство или запустите AVD.

Выберите устройство в Android Studio и нажмите "Запустить".

**Запуск приложения JDK:**

Создайте новый проект, выбрав JDK.

Выберите шаблон "Command Line App".

Запустите проект через меню "Run".

**Билет 7**

1. Инструмент разработки мобильных приложений Visual Studio: создание и настройка проектов Xamarin Forms, Xamarin.Android, запуск приложения.
2. **Создание проекта Xamarin Forms:**

* Файл > Создать > Проект... > Мобильное приложение (Xamarin).
* Имя проекта > Создать.
* Выберите "Пустой проект Xamarin.Forms" > Создать

1. **Настройка проекта Xamarin Forms:**

* ПКМ по проекту > Свойства > Сборка Android.
* Назначение > Android.
* Версия минимального API > Выберите API > OK.

1. **Создание проекта Xamarin.Android:**

* ПКМ по проекту Xamarin Forms > Добавить > Новый проект... > Android.
* Имя проекта > Создать.
* Выберите "Пустой проект Android" > Создать

1. **Настройка проекта Xamarin.Android:**

* ПКМ по проекту > Свойства > Сборка Android.
* Назначение > Android.
* Версия минимального API > Выберите API > OK.

1. **Запуск приложения:**

* ПКМ по проекту Xamarin Forms > Запустить без отладки/Отладить.
* Приложение запустится на эмуляторе/устройстве

**Билет 8**

1. Контейнеры в Xamarin.Android: ListView, GridView, LinearLayot, RelativeLayot. Описание, назначение, примеры, принцип использования в коде проекта.

**ListView:** ListView используется для отображения списка элементов, в котором пользователь может прокручивать содержимое вертикально. Список может содержать текстовые элементы, изображения или пользовательские представления. Примеры использования ListView включают контактные списки, списки задач или списки новостей. В коде проекта можно создать экземпляр ListView и заполнить его данными из источника данных, например, массива или базы данных.  
**Пример кода:**

ListView listView = FindViewById<ListView>(Resource.Id.myListView);

**string**[] items = **new** **string**[] { "item1", "item2", "item3" };

ArrayAdapter<**string**> adapter = **new** ArrayAdapter<**string**>(**this**, Android.Resource.Layout.SimpleListItem1, items);

listView.Adapter = adapter;

**GridView:** GridView используется для отображения элементов в виде сетки, где каждый элемент представлен в ячейке. GridView обычно используется для отображения коллекции изображений, например, галереи фотографий. В коде проекта можно создать экземпляр GridView, установить адаптер для привязки данных и настроить внешний вид каждого элемента с помощью пользовательских представлений.

**Пример кода:**

<GridView xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/gridView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:numColumns="3" />

**LinearLayout**: LinearLayout используется для расположения элементов пользовательского интерфейса в виде горизонтального или вертикального списка. Линейный макет позволяет управлять порядком и расположением элементов внутри контейнера. В коде проекта можно создать экземпляр LinearLayout и добавить элементы (например, кнопки, текстовые поля или изображения) с помощью методов AddView().

**Пример кода:**

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Элемент 1"/>

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Кнопка"/>

</LinearLayout>

// Добавление элементов в LinearLayout

linearLayout.AddView(button1);

linearLayout.AddView(button2);

**RelativeLayout:** RelativeLayout используется для расположения элементов пользовательского интерфейса с помощью относительных правил и связей между элементами. С помощью RelativeLayout можно создавать сложные макеты, управлять позиционированием и выравниванием элементов. В коде проекта можно создать экземпляр RelativeLayout и задать атрибуты позиционирования для каждого элемента с помощью пользовательских параметров.

**Пример кода:**

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<Button

android:id="@+id/ok"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="OK"/>

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_toLeftOf="@id/ok"

android:layout\_alignTop="@id/ok"

android:text="Cancel"/>

</RelativeLayout>

**Билет 9**

1. Структура типичного мобильного приложения: активности и интенты, архитектура MVC.

**Активности и интенты в Android:**

* Активность (Activity): Активности представляют собой один экран пользовательского интерфейса в приложении Android. Каждая активность обычно представлена в отдельном классе и содержит логику обработки событий, отображение данных и взаимодействие с пользователем. В типичном мобильном приложении может быть несколько активностей, связанных между собой переходами.
* Интент (Intent): Интенты представляют собой объекты, используемые для связи между компонентами приложения, такими как активности, службы, широковещательные приемники и другие. Интенты могут использоваться для запуска активностей, передачи данных между активностями, вызова сторонних приложений и других действий.

**Архитектура MVC (Model-View-Controller):**

* Модель (Model): Модель представляет данные и бизнес-логику в приложении. Она отвечает за получение, обработку и хранение данных, а также взаимодействие с базой данных или удаленными источниками данных.
* Представление (View): Представление отвечает за отображение данных пользователю и управление пользовательским интерфейсом. Оно отображает информацию из модели и обрабатывает действия пользователя, такие как нажатия кнопок, ввод текста и другие события.
* Контроллер (Controller): Контроллер обрабатывает взаимодействие между моделью и представлением. Он реагирует на действия пользователя и обновляет данные в модели, а также обновляет представление на основе изменений в модели.

В типичном мобильном приложении для платформы Android, активности и интенты используются для управления пользовательским взаимодействием и навигацией между экранами, а архитектура MVC помогает организовать код приложения для эффективной работы с данными, отображения информации и управления бизнес-логикой.

**Билет 10**

1. Способы хранения данных: работа с файлами.

**Локальное хранение файлов:**

В Xamarin.Forms можно использовать простой файловый доступ для сохранения и загрузки файлов локально на устройстве. Это может включать в себя сохранение текстовых файлов, изображений, настроек пользователя и других типов данных. Для этого обычно используются стандартные классы C# для работы с файлами, такие как System.IO.File и System.IO.Stream.

**Использование SQLite для хранения данных:**

SQLite является небольшой встраиваемой реляционной базой данных, которую можно использовать в приложениях Xamarin.Forms для хранения структурированных данных. Это позволяет приложению сохранять данные в базе данных SQLite, вместо использования простых файлов для хранения информации.

**Сериализация объектов:**

В Xamarin.Forms можно использовать сериализацию объектов для сохранения и загрузки сложных объектов в файлы. Например, можно использовать JSON или XML для преобразования объектов в строковый формат, который затем можно сохранить в файле на устройстве.

**Использование внешних хранилищ и облаков:**

В некоторых случаях приложения Xamarin.Forms могут использовать внешние хранилища данных, такие как облачные сервисы, для сохранения данных на удаленных серверах. Это позволяет приложениям обмениваться данными между различными устройствами и сохранять информацию в защищенном и надежном хранилище.

При выборе способа хранения данных в Xamarin.Forms важно учитывать требования приложения, типы данных, область применения и требования к безопасности информации.

**Билет 11**

1. Способы хранения данных: работа с базой данных SQLite.

**Установка пакета SQLite:**

Для работы с базой данных SQLite в Xamarin.Forms необходимо установить соответствующий пакет NuGet. В Visual Studio можно открыть менеджер пакетов NuGet и найти и установить пакет SQLite-net-pcl, который является популярной библиотекой для работы с SQLite в Xamarin.Forms.

**Создание модели данных:**

Определите классы данных, которые будут представлять таблицы в базе данных SQLite. Каждый класс должен иметь свойства, которые соответствуют столбцам таблицы.

**Создание БД и таблиц:**

В коде Xamarin.Forms необходимо создать соединение с базой данных SQLite и создать таблицы на основе моделей данных. Для этого можно использовать класс SQLiteConnection и методы CreateTable для каждого класса данных.

**Выполнение операций с данными:**

После создания таблиц можно выполнять различные операции с данными, такие как добавление, обновление, выборка и удаление записей. Для этого можно использовать методы класса SQLiteConnection, такие как Insert, Update, Query и Delete.

**Закрытие соединения:**

После выполнения операций необходимо закрыть соединение с базой данных SQLite. Для этого используется метод Close класса SQLiteConnection.

При работе с базой данных SQLite в Xamarin.Forms важно помнить о практиках безопасности данных и правильном управлении ресурсами.

**Билет 12**

1. Контейнеры компоновки StackLayout, AbsoluteLayout, RelativeLayout, Grid, FlexLayout в Xamarin Forms.

**StackLayout:**

StackLayout располагает элементы вертикально (по умолчанию) или горизонтально и меняет их размеры в соответствии с их содержимым. Элементы могут быть выровнены в центре, по краям или по сторонам контейнера.

**AbsoluteLayout:**

AbsoluteLayout позволяет абсолютно позиционировать элементы внутри контейнера. Каждый элемент можно разместить в заданных координатах (относительно верхнего левого угла контейнера) и задать его размеры.

**RelativeLayout:**

RelativeLayout позволяет размещать элементы относительно друг друга или относительно границ контейнера. Элементы могут быть выровнены вертикально и горизонтально, а также могут иметь привязки к другим элементам, что позволяет строить гибкую разметку.

**Grid:**

Grid позволяет размещать элементы в виде сетки с определенным количеством строк и столбцов. Элементы могут занимать одну или несколько ячеек сетки, а также быть выровнены по горизонтали и вертикали внутри этих ячеек.

**FlexLayout:**

FlexLayout предоставляет гибкий способ компоновки элементов с помощью правил гибкого макетирования. Элементы могут быть управляемыми в зависимости от доступного пространства, размеров контейнера и определенных правил.

**Билет 13**

1. Глобализация и локализация при разработке мобильных приложений.

**Глобализация** - это процесс разработки приложения, который делает его готовым для адаптации к различным языкам, региональным форматам даты и времени, валютам и другим культурным особенностям. Включает в себя проектирование приложения с учетом возможности локализации, таким образом, что весь текст, форматы даты и времени, валюты, и другие локализуемые элементы могут быть легко заменены для соответствия конкретным региональным настройкам.

**Локализация** - это процесс адаптации приложения к конкретному языку, культуре или региону путем замены текстов, изображений, форматов даты и времени, валют и других элементов с учетом конкретных требований конкретной локали.

**При разработке мобильных приложений следует уделять внимание следующим аспектам глобализации и локализации:**

* Язык и текст: использование многоязычных строковых ресурсов для всех текстовых элементов в приложении, чтобы обеспечить их простую замену на другие языки.
* Формат даты и времени: учет форматов даты, времени и календарных систем для разных регионов.
* Формат валют: корректное отображение валюты в зависимости от местной валюты пользователя.
* Изображения и мультимедиа: выбор подходящих изображений и мультимедийных элементов, учитывая культурные особенности и законодательные требования различных стран.

Обеспечение глобализации и локализации в мобильном приложении позволяет увеличить его доступность и привлекательность для широкого международного аудитории, повышая уровень удовлетворенности пользователей и приводя к увеличению числа загрузок и успеху на международных рынках.

**Билет 14**

1. Навигация между страницами в Xamarin Forms: стек навигации, методы, примеры.

**Стек навигации:** представляет собой стек, в котором хранятся страницы приложения. Когда новая страница добавляется в стек, она отображается поверх текущей страницы, а при возврате страницы из стека предыдущая страница становится активной.

**Методы и примеры:**

1. Navigation.PushAsync: Используется для добавления новой страницы в стек навигации.   
   **Пример:** await Navigation.PushAsync(new NewPage());
2. Navigation.PopAsync: Используется для удаления текущей страницы из стека навигации и возврата к предыдущей странице.  
   **Пример:** await Navigation.PopAsync();
3. Navigation.PushModalAsync: Используется для отображения модальной страницы поверх текущей страницы.  
   **Пример:** await Navigation.PushModalAsync(new ModalPage());
4. Navigation.PopModalAsync: Используется для закрытия текущей модальной страницы и возврата к предыдущей странице.  
   **Пример:** await Navigation.PopModalAsync();
5. Navigation.PushAsync с параметрами: Передача параметров между страницами.  
   **Пример:** await Navigation.PushAsync(new DetailPage(parameter));

**Создание главной страницы**

public partial class App : Application {

public App() {

InitializeComponent();

MainPage = new NavigationPage(new MainPage());

}

}

**Билет 15**

1. Триггеры Xamarin Forms: понятие, применение, виды, пример.

**Понятие:** определяет набор условий, если они выполняются, приводят к изменению внешнего вида элемента интерфейса, такого как цвет, размер, видимость и т.д.

**Применение:** Триггеры могут быть использованы для:

1. Изменения стилей элементов в зависимости от определенных условий.
2. Анимации элементов при определенных событиях или изменениях свойств.

**Виды триггеров:**

1. Trigger: Простой триггер, который реагирует на изменение значения свойства элемента.
2. DataTrigger: Триггер данных, который реагирует на изменение значения привязанного свойства данных.
3. EventTrigger: Триггер событий, который реагирует на определенное событие, происходящее в элементе.

**Пример:**

<Label Text="Наведите курсор" BackgroundColor="LightGray">

<Label.Triggers>

<Trigger TargetType="Label" Property="IsFocused" Value="True">

<Setter Property="BackgroundColor" Value="LightBlue" />

</Trigger>

</Label.Triggers>

</Label>

**Билет 16**

1. Привязка данных (data binding) в Xamarin Forms: способы задания, виды, примеры.

Привязка данных в Xamarin Forms — это механизм, который связывает свойства двух объектов, позволяя автоматически обновлять интерфейс при изменении данных. Это упрощает взаимодействие между пользовательским интерфейсом и логикой приложения.

**Способы задания:**

1. Статическое связывание: Привязка данных может быть определена непосредственно в разметке XAML.
2. Динамическое связывание (C#): Привязка также может быть выполнена динамически в коде C#.

**Виды:**

1. OneWay (однонаправленная): Обновление свойства интерфейса происходит при изменении данных, но не наоборот.
2. TwoWay (двунаправленная): Изменения данных приводят к обновлению интерфейса и наоборот.

**Пример привязки:**

public partial class MainPage : ContentPage

{

public MainPage()

{

Label label = new Label

{

FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))

};

Entry entry = new Entry();

// Устанавливаем привязку

// источник привязки - entry, цель привязки - label

label.BindingContext = entry;

// Связываем свойства источника и цели

label.SetBinding(Label.TextProperty, "Text");

StackLayout stackLayout = new StackLayout()

{

Children = { label, entry}

};

Content = stackLayout;

}

}

**Билет 17**

1. Элементы в Xamarin Forms, их свойства и события: кнопки, текстовые поля.

**Кнопка (Button) свойства:**

* Text: Текст, отображаемый на кнопке.
* TextColor: Цвет текста кнопки.
* BackgroundColor: Цвет фона кнопки.
* Command: Команда, которая будет выполнена при нажатии на кнопку.

<Button Text="Нажми меня"

TextColor="White"

BackgroundColor="Blue"

Clicked="OnButtonClicked"/>

**Текстовое поле (Entry) свойства:**

* Text: Текст, отображаемый в текстовом поле.
* Placeholder: Заглушка (подсказка), отображаемая в текстовом поле до ввода текста.
* TextColor: Цвет текста в текстовом поле.
* IsPassword: Булево значение, указывающее, является ли поле ввода полем для ввода пароля.

<Entry Placeholder="Введите ваше имя"

TextChanged="OnTextChanged"/>

**Билет 18**

1. Работа с изображениями: хранение, загрузка, элементы для отображения.

**Хранение изображений:**

* Ресурсы проекта:

Изображения могут быть добавлены в ресурсы проекта Xamarin.Forms. В этом случае они будут включены в сборку приложения.

* Локальное хранение:

Изображения также могут быть сохранены локально в файловой системе устройства.

**Загрузка изображений из ресурсов:**

* Изображения из ресурсов проекта можно загружать в коде: ImageSource imageSource = ImageSource.FromResource("YourNamespace.YourImage.png");
* Изображения могут быть загружены из интернета по URL: ImageSource imageSource = ImageSource.FromUri(new Uri("https://example.com/yourimage.png"));

**Элементы для отображения:**

1. Image:

Source: Свойство, определяющее источник изображения.

* Aspect: Свойство, управляющее режимом масштабирования изображения.

1. CachedImage (от библиотеки FFImageLoading): Позволяет загружать и кэшировать изображения более эффективно.
2. MediaElement (для отображения видео и аудио с возможностью встраивания изображений):
3. SvgImage (от библиотеки SkiaSharp): Позволяет отображать изображения в формате SVG.

**Билет 19**

1. Жизненный цикл мобильного приложения: основные состояния, переход между состояниями.

Жизненный цикл приложения — это процесс, который охватывает все этапы разработки, от первоначальной идеи до завершения использования продукта.

**Основные состояния в жизненном цикле мобильного приложения:**

* Инициализация (Initialization): Запуск и инициализация компонентов приложения.
* Запуск (Launching): Отображение начального интерфейса приложения.
* Работа (Running): Активное взаимодействие пользователя с приложением.
* Приостановка (Pausing): Переход в фоновый режим при сворачивании или переходе на другое приложение.
* Возобновление (Resuming): Возвращение из фонового режима к активной работе.
* Остановка (Stopping): Закрытие приложения или переход в фоновый режим.
* Уничтожение (Destroying): Полное завершение работы приложения.

**Переходы между состояниями:**

* Из Инициализации в Запуск: После успешной инициализации компонентов приложение переходит к отображению начального экрана.
* Из Запуска в Работу: Пользователь начинает взаимодействие с приложением, и оно переходит в активное состояние работы.
* Из Работы в Приостановку: Когда приложение теряет фокус, например, при сворачивании, оно может перейти в состояние приостановки.
* Из Приостановки в Возобновление: Когда пользователь возвращается к приложению, оно может возобновить свою работу с сохраненного состояния.
* Из Работы в Остановку: Приложение может перейти в состояние остановки, когда пользователь закрывает его или переключается на другое приложение.
* Из Остановки в Уничтожение: После остановки приложение может быть уничтожено, освобождая ресурсы и завершая свою работу.

**Билет 20**

1. Работа с ресурсами в Xamrin Forms: виды ресурсов, создание и использование в C# и XAML.

**Виды ресурсов:**

1. Статические ресурсы (Static Resources):

Глобальные ресурсы, определенные в файле App.xaml в разделе <Application.Resources>. Они доступны из любой части приложения и могут содержать цвета, стили и другие элементы.

1. Динамические ресурсы (Dynamic Resources):

Используются для привязки к динамически изменяющимся значениям. Например, цвета или стили элементов управления могут быть установлены как динамические ресурсы.

**Создание и использование в C# и XAML:**

1. В XAML-разметке, статические и динамические ресурсы могут быть использованы для определения стилей, цветов и других параметров элементов интерфейса.
2. В коде C#, ресурсы могут быть получены и установлены динамически. Например, можно получить цвет из статического ресурса и применить его к элементу интерфейса.

**Билет 21**

Supabase — это платформа с открытым исходным кодом, предоставляющая разработчикам набор инструментов для создания приложений. Она включает в себя реляционную базу данных на PostgreSQL, систему аутентификации, API, хранилище данных и возможность работы с данными в реальном времени. Supabase упрощает процесс разработки, позволяя сосредоточиться на клиентской части приложения, в то время как серверная инфраструктура и управление данными берёт на себя сама платформа.

**Процесс подключения к Supabase**

Создание проекта: Перейдите на сайт Supabase и создайте учетную запись. Создайте новый проект, указав имя и регион.

**Настройка базы данных:**

После создания проекта вы получите доступ к интерфейсу управления базой данных. Вы можете создавать таблицы и управлять данными с помощью SQL-редактора или графического интерфейса.

Подключение к проекту: Откройте файл build.gradle и добавьте необходимые зависимости для работы с Supabase:

Вам нужно добавить разрешение на доступ к интернету в файл AndroidManifest.xml.

<uses-permission android:name="android.permission.INTERNET"/>

Шаг 5: Инициализация Supabase

Создайте класс для работы с Supabase, например, SupabaseClient.kt.

import io.github.jan-tennert.supabase.SupabaseClient

object Supabase {

private const val SUPABASE\_URL = "https://your-project-ref.supabase.co"

private const val SUPABASE\_ANON\_KEY = "your-anon-key"

val client = SupabaseClient(SUPABASE\_URL, SUPABASE\_ANON\_KEY)

}

2. Система авторизации через OTP

Шаг 1: Настройка аутентификации в Supabase

В панели управления Supabase перейдите в раздел Authentication.

Включите провайдер Phone для аутентификации через OTP.

Шаг 2: Реализация интерфейса

Создайте основной интерфейс для ввода электронной почты или номера телефона для получения OTP.

activity\_main.xml

<EditText

android:id="@+id/emailEditText"

android:hint="Введите ваш email или номер телефона" />

<Button

android:id="@+id/sendOtpButton"

android:text="Отправить OTP" />

MainActivity.kt

class MainActivity : AppCompatActivity() {

private lateinit var emailEditText: EditText

private lateinit var sendOtpButton: Button

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

emailEditText = findViewById(R.id.emailEditText)

sendOtpButton = findViewById(R.id.sendOtpButton)

sendOtpButton.setOnClickListener {

val input = emailEditText.text.toString()

sendOtp(input)

}

}

private fun sendOtp(input: String) {

Auth.signInWithOtp(input).invokeOnCompletion { result ->

runOnUiThread {

if (result.isSuccess) {

Toast.makeText(this, "OTP отправлен!", Toast.LENGTH\_SHORT).show()

} else {

Toast.makeText(this, "Ошибка при отправке OTP.", Toast.LENGTH\_SHORT).show()

Шаг 3: Проверка OTP

Создайте дополнительный интерфейс для ввода OTP и его проверки.

|  |  |  |  |
| --- | --- | --- | --- |
| **1.Android:**  Создан Android Inc. (2003), куплен Google (2005). Первый телефон – HTC Dream (2008).  Открытый исходный код.  У каждой версии название десерта (например, Android 10, 11, 12).  Основан на ядре Linux.  Плюсы: Более гибкий, много разных устройств, часто дешевле.  **iOS:**  Разработан Apple и представлен вместе с iPhone в 2007.  Закрытый исходный код. Apple полностью контролирует систему.  Версии: Просто iOS 14, iOS 15 и т.д.  Основан на macOS.  Плюсы: Более стабильный, единый стиль, хорошая оптимизация под "железо".  Отличия:  1) Основное отличие между Android и iOS заключается в том, что Android - это открытая операционная система, в то время как iOS - это закрытая операционная система.  2) Интерфейс Android более настраиваемый, пользователь может менять различные настройки, устанавливать сторонние приложения и дополнения. В то время как интерфейс iOS более ограничен и ориентирован на улучшенную работу только с приложениями из App Store.  3) Android поддерживает различные производители устройств, включая Samsung, Huawei, Xiaomi и другие, в то время как iOS работает только на устройствах Apple - iPhone, iPad и iPod Touch.  4) В Android используется язык программирования Java, Kotlin и C++ для разработки приложений, в то время как в iOS используется язык программирования Objective-C или Swift.  Архитектура Android основана на ядре Linux. Она состоит из четырех основных слоев: ядра Linux, аппаратного слоя, фреймворка приложений и пользовательского интерфейса.  Архитектура iOS состоит из ядра операционной системы, фреймворков, аппаратного слоя и пользовательского интерфейса. Она также имеет свою файловую систему и систему безопасности. | **2.Нативные приложения:**  Понятие: Нативные приложения разрабатываются специально для определенной платформы (например, iOS или Android), используя собственные языки программирования и инструменты разработки, предоставленные производителем платформы.  Технология создания: Для разработки нативных приложений на iOS используется язык программирования Swift или Objective-C, а для Android - Java или Kotlin. Примеры приложений: Pinterest и Twitter (Kotlin), WhatsApp и Instagram (Swift)  **Гибридные приложения:**  Понятие: Гибридные приложения создаются с использованием веб-технологий, таких как HTML, CSS и JavaScript, и запускаются внутри специальной обертки, которая позволяет им работать на разных платформах. Примеры: Airbnb, Uber.  Технология создания: Разработка гибридных приложений может осуществляться с использованием различных фреймворков, таких как Apache Cordova (PhoneGap), React Native и Ionic.  **Кроссплатформенные приложения:**  Понятие: Кроссплатформенные приложения разрабатываются с использованием одной базовой кодовой базы и могут работать на разных платформах без изменений в исходном коде. Примеры: Skype, Facebook (React Native), Google Play (Flutter),The World Bank, Alaska Airlines (Xamarin)  Технология создания: Для создания кроссплатформенных приложений широко используются фреймворки, такие как React Native, Flutter и Xamarin. | **3.Веб-приложение** — это программное обеспечение, к которому можно получить доступ через веб-браузер на компьютере или мобильном устройстве. В отличие от нативных приложений, которые устанавливаются на устройство, веб-приложения работают на удаленном сервере и не требуют установки. Пользователи могут взаимодействовать с веб-приложениями, вводя данные и получая результаты.  Технология создания веб-приложений включает использование таких языков программирования, как HTML, CSS и JavaScript. HTML используется для создания структуры веб-страницы, CSS - для оформления и стилизации элементов страницы, а JavaScript - для создания интерактивности и динамического поведения веб-приложения.  **Веб-сервис** — это программная система, предназначенная для поддержки взаимодействия между различными машинами в сети. Веб-сервисы используют стандартизированные протоколы, такие как HTTP, SOAP и REST, для обмена данными между клиентом и сервером.  При создании веб-сервиса используются следующие технологии:   * Языки программирования: Java, Python, PHP и другие. * Форматы данных: XML, JSON. * Протоколы: HTTP, SOAP, REST.   Веб-сервисы могут предоставлять различные функциональные возможности, такие как доступ к базам данных, выполнение бизнес-логики и интеграцию с другими приложениями.  Основным отличием между веб-приложениями и веб-сервисами является то, что веб-приложения выполняются на стороне клиента (мобильного устройства), в то время как веб-сервисы выполняются на стороне сервера и предоставляют функциональность или данные, которые могут быть использованы клиентскими приложениями. | **4.Подготовка:**  Проверить, что приложение готово к публикации и соответствует всем требованиям платформы, для которой оно разработано (например, iOS или Android).  Убедиться, что приложение прошло все необходимые тестирования и исправлены все выявленные ошибки и проблемы.  Подготовить все необходимые ресурсы и файлы, такие как иконки, изображения скриншотов и описания приложения.  **Компиляция:**  Если приложение разрабатывается для iOS, скомпилировать его в Xcode, используя подходящие настройки и профили разработчика. Убедиться, что приложение успешно компилируется без ошибок и предупреждений.  Если приложение разрабатывается для Android, скомпилировать его в Android Studio или другой среде разработки для Android. Убедиться, что компиляция проходит успешно и без ошибок.  **Создание пакета:**  Для публикации приложения в App Store (iOS) необходимо создать подписанный архив (.ipa файл) с использованием соответствующего профиля разработчика. Этот подписанный архив будет загружен в iTunes Connect для дальнейшей публикации в App Store.  Для публикации приложения в Google Play Store (Android) необходимо создать подписанный APK файл. Потребуется ключ для подписи приложения, который можно создать в Google Play Console. После создания подписанного APK файла его можно загрузить в Google Play Console для публикации в Play Store. |
| **9.Активности и интенты в Android:**   * Активность (Activity): Активности представляют собой один экран пользовательского интерфейса в приложении Android. Каждая активность обычно представлена в отдельном классе и содержит логику обработки событий, отображение данных и взаимодействие с пользователем. В типичном мобильном приложении может быть несколько активностей, связанных между собой переходами. * Интент (Intent): Интенты представляют собой объекты, используемые для связи между компонентами приложения, такими как активности, службы, широковещательные приемники и другие. Интенты могут использоваться для запуска активностей, передачи данных между активностями, вызова сторонних приложений и других действий.   **Архитектура MVC (Model-View-Controller):**   * Модель (Model): Модель представляет данные и бизнес-логику в приложении. Она отвечает за получение, обработку и хранение данных, а также взаимодействие с базой данных или удаленными источниками данных. * Представление (View): Представление отвечает за отображение данных пользователю и управление пользовательским интерфейсом. Оно отображает информацию из модели и обрабатывает действия пользователя, такие как нажатия кнопок, ввод текста и другие события. * Контроллер (Controller): Контроллер обрабатывает взаимодействие между моделью и представлением. Он реагирует на действия пользователя и обновляет данные в модели, а также обновляет представление на основе изменений в модели.   В типичном мобильном приложении для платформы Android, активности и интенты используются для управления пользовательским взаимодействием и навигацией между экранами, а архитектура MVC помогает организовать код приложения для эффективной работы с данными, отображения информации и управления бизнес-логикой. | **10.Локальное хранение файлов:**  В Xamarin.Forms можно использовать простой файловый доступ для сохранения и загрузки файлов локально на устройстве. Это может включать в себя сохранение текстовых файлов, изображений, настроек пользователя и других типов данных. Для этого обычно используются стандартные классы C# для работы с файлами, такие как System.IO.File и System.IO.Stream.  **Использование SQLite для хранения данных:**  SQLite является небольшой встраиваемой реляционной базой данных, которую можно использовать в приложениях Xamarin.Forms для хранения структурированных данных. Это позволяет приложению сохранять данные в базе данных SQLite, вместо использования простых файлов для хранения информации.  **Сериализация объектов:**  В Xamarin.Forms можно использовать сериализацию объектов для сохранения и загрузки сложных объектов в файлы. Например, можно использовать JSON или XML для преобразования объектов в строковый формат, который затем можно сохранить в файле на устройстве.  **Использование внешних хранилищ и облаков:**  В некоторых случаях приложения Xamarin.Forms могут использовать внешние хранилища данных, такие как облачные сервисы, для сохранения данных на удаленных серверах. Это позволяет приложениям обмениваться данными между различными устройствами и сохранять информацию в защищенном и надежном хранилище.  При выборе способа хранения данных в Xamarin.Forms важно учитывать требования приложения, типы данных, область применения и требования к безопасности информации. | **11.Установка пакета SQLite:**  Для работы с базой данных SQLite в Xamarin.Forms необходимо установить соответствующий пакет NuGet. В Visual Studio можно открыть менеджер пакетов NuGet и найти и установить пакет SQLite-net-pcl, который является популярной библиотекой для работы с SQLite в Xamarin.Forms.  **Создание модели данных:**  Определите классы данных, которые будут представлять таблицы в базе данных SQLite. Каждый класс должен иметь свойства, которые соответствуют столбцам таблицы.  **Создание БД и таблиц:**  В коде Xamarin.Forms необходимо создать соединение с базой данных SQLite и создать таблицы на основе моделей данных. Для этого можно использовать класс SQLiteConnection и методы CreateTable для каждого класса данных.  **Выполнение операций с данными:**  После создания таблиц можно выполнять различные операции с данными, такие как добавление, обновление, выборка и удаление записей. Для этого можно использовать методы класса SQLiteConnection, такие как Insert, Update, Query и Delete.  **Закрытие соединения:**  После выполнения операций необходимо закрыть соединение с базой данных SQLite. Для этого используется метод Close класса SQLiteConnection.  При работе с базой данных SQLite в Xamarin.Forms важно помнить о практиках безопасности данных и правильном управлении ресурсами. | **12.StackLayout:**  StackLayout располагает элементы вертикально (по умолчанию) или горизонтально и меняет их размеры в соответствии с их содержимым. Элементы могут быть выровнены в центре, по краям или по сторонам контейнера.  **AbsoluteLayout:**  AbsoluteLayout позволяет абсолютно позиционировать элементы внутри контейнера. Каждый элемент можно разместить в заданных координатах (относительно верхнего левого угла контейнера) и задать его размеры.  **RelativeLayout:**  RelativeLayout позволяет размещать элементы относительно друг друга или относительно границ контейнера. Элементы могут быть выровнены вертикально и горизонтально, а также могут иметь привязки к другим элементам, что позволяет строить гибкую разметку.  **Grid:**  Grid позволяет размещать элементы в виде сетки с определенным количеством строк и столбцов. Элементы могут занимать одну или несколько ячеек сетки, а также быть выровнены по горизонтали и вертикали внутри этих ячеек.  **FlexLayout:**  FlexLayout предоставляет гибкий способ компоновки элементов с помощью правил гибкого макетирования. Элементы могут быть управляемыми в зависимости от доступного пространства, размеров контейнера и определенных правил. |
| **13Глобализация** - это процесс разработки приложения, который делает его готовым для адаптации к различным языкам, региональным форматам даты и времени, валютам и другим культурным особенностям. Включает в себя проектирование приложения с учетом возможности локализации, таким образом, что весь текст, форматы даты и времени, валюты, и другие локализуемые элементы могут быть легко заменены для соответствия конкретным региональным настройкам.  **Локализация** - это процесс адаптации приложения к конкретному языку, культуре или региону путем замены текстов, изображений, форматов даты и времени, валют и других элементов с учетом конкретных требований конкретной локали.  **При разработке мобильных приложений следует уделять внимание следующим аспектам глобализации и локализации:**   * Язык и текст: использование многоязычных строковых ресурсов для всех текстовых элементов в приложении, чтобы обеспечить их простую замену на другие языки. * Формат даты и времени: учет форматов даты, времени и календарных систем для разных регионов. * Формат валют: корректное отображение валюты в зависимости от местной валюты пользователя. * Изображения и мультимедиа: выбор подходящих изображений и мультимедийных элементов, учитывая культурные особенности и законодательные требования различных стран.   Обеспечение глобализации и локализации в мобильном приложении позволяет увеличить его доступность и привлекательность для широкого международного аудитории, повышая уровень удовлетворенности пользователей и приводя к увеличению числа загрузок и успеху на международных рынках. | **14Стек навигации:** представляет собой стек, в котором хранятся страницы приложения. Когда новая страница добавляется в стек, она отображается поверх текущей страницы, а при возврате страницы из стека предыдущая страница становится активной.  **Методы и примеры:**   1. Navigation.PushAsync: Используется для добавления новой страницы в стек навигации.  **Пример:** await Navigation.PushAsync(new NewPage()); 2. Navigation.PopAsync: Используется для удаления текущей страницы из стека навигации и возврата к предыдущей странице. **Пример:** await Navigation.PopAsync(); 3. Navigation.PushModalAsync: Используется для отображения модальной страницы поверх текущей страницы. **Пример:** await Navigation.PushModalAsync(new ModalPage()); 4. Navigation.PopModalAsync: Используется для закрытия текущей модальной страницы и возврата к предыдущей странице. **Пример:** await Navigation.PopModalAsync(); 5. Navigation.PushAsync с параметрами: Передача параметров между страницами. **Пример:** await Navigation.PushAsync(new DetailPage(parameter));   Создание главной страницы  public partial class App : Application {  public App() {  InitializeComponent();  MainPage = new NavigationPage(new MainPage());  }  } | **15.Понятие:** определяет набор условий, если они выполняются, приводят к изменению внешнего вида элемента интерфейса, такого как цвет, размер, видимость и т.д.  **Применение:** Триггеры могут быть использованы для:   * Изменения стилей элементов в зависимости от определенных условий. * Анимации элементов при определенных событиях или изменениях свойств.   **Виды триггеров:**   * Trigger: Простой триггер, который реагирует на изменение значения свойства элемента. * DataTrigger: Триггер данных, который реагирует на изменение значения привязанного свойства данных. * EventTrigger: Триггер событий, который реагирует на определенное событие, происходящее в элементе.   **Пример:**  <Label Text="Наведите курсор" BackgroundColor="LightGray">  <Label.Triggers>  <Trigger TargetType="Label" Property="IsFocused" Value="True">  <Setter Property="BackgroundColor" Value="LightBlue" />  </Trigger>  </Label.Triggers>  </Label> | 16Привязка данных в Xamarin Forms — это механизм, который связывает свойства двух объектов, позволяя автоматически обновлять интерфейс при изменении данных. Это упрощает взаимодействие между пользовательским интерфейсом и логикой приложения.  Способы задания:   * Статическое связывание: Привязка данных может быть определена непосредственно в разметке XAML. * Динамическое связывание (C#): Привязка также может быть выполнена динамически в коде C#.   Виды:   * OneWay (однонаправленная): Обновление свойства интерфейса происходит при изменении данных, но не наоборот. * TwoWay (двунаправленная): Изменения данных приводят к обновлению интерфейса и наоборот.   Пример привязки:  public partial class MainPage : ContentPage  {  public MainPage()  {  Label label = new Label  {  FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))  };  Entry entry = new Entry();  // Устанавливаем привязку  // источник привязки - entry, цель привязки - label  label.BindingContext = entry;  // Связываем свойства источника и цели  label.SetBinding(Label.TextProperty, "Text");    StackLayout stackLayout = new StackLayout()  {  Children = { label, entry}  };  Content = stackLayout;  }} |
| **17Кнопка (Button) свойства:**   * Text: Текст, отображаемый на кнопке. * TextColor: Цвет текста кнопки. * BackgroundColor: Цвет фона кнопки. * Command: Команда, которая будет выполнена при нажатии на кнопку.   <Button Text="Нажми меня"  TextColor="White"  BackgroundColor="Blue"  Clicked="OnButtonClicked"/>  **Текстовое поле (Entry) свойства:**   * Text: Текст, отображаемый в текстовом поле. * Placeholder: Заглушка (подсказка), отображаемая в текстовом поле до ввода текста. * TextColor: Цвет текста в текстовом поле. * IsPassword: Булево значение, указывающее, является ли поле ввода полем для ввода пароля.   <Entry Placeholder="Введите ваше имя"  TextChanged="OnTextChanged"/> | **18Хранение изображений:**   * Ресурсы проекта:   Изображения могут быть добавлены в ресурсы проекта Xamarin.Forms. В этом случае они будут включены в сборку приложения.   * Локальное хранение:   Изображения также могут быть сохранены локально в файловой системе устройства.  **Загрузка изображений из ресурсов:**   * Изображения из ресурсов проекта можно загружать в коде: ImageSource imageSource = ImageSource.FromResource("YourNamespace.YourImage.png"); * Изображения могут быть загружены из интернета по URL: ImageSource imageSource = ImageSource.FromUri(new Uri("https://example.com/yourimage.png"));   **Элементы для отображения:**   * Image: Source: Свойство, определяющее источник изображения. * Aspect: Свойство, управляющее режимом масштабирования изображения. * CachedImage (от библиотеки FFImageLoading): Позволяет загружать и кэшировать изображения более эффективно. * MediaElement (для отображения видео и аудио с возможностью встраивания изображений):   SvgImage (от библиотеки SkiaSharp): Позволяет отображать изображения в формате SVG. | 19Жизненный цикл приложения — это процесс, который охватывает все этапы разработки, от первоначальной идеи до завершения использования продукта.  **Основные состояния в жизненном цикле мобильного приложения:**   * Инициализация (Initialization): Запуск и инициализация компонентов приложения. * Запуск (Launching): Отображение начального интерфейса приложения. * Работа (Running): Активное взаимодействие пользователя с приложением. * Приостановка (Pausing): Переход в фоновый режим при сворачивании или переходе на другое приложение. * Возобновление (Resuming): Возвращение из фонового режима к активной работе. * Остановка (Stopping): Закрытие приложения или переход в фоновый режим. * Уничтожение (Destroying): Полное завершение работы приложения.   **Переходы между состояниями:**  Из Инициализации в Запуск: После успешной инициализации компонентов приложение переходит к отображению начального экрана.  Из Запуска в Работу: Пользователь начинает взаимодействие с приложением, и оно переходит в активное состояние работы.  Из Работы в Приостановку: Когда приложение теряет фокус, например, при сворачивании, оно может перейти в состояние приостановки.  Из Приостановки в Возобновление: Когда пользователь возвращается к приложению, оно может возобновить свою работу с сохраненного состояния.  Из Работы в Остановку: Приложение может перейти в состояние остановки, когда пользователь закрывает его или переключается на другое приложение.  Из Остановки в Уничтожение: После остановки приложение может быть уничтожено, освобождая ресурсы и завершая свою работу. | 20Supabase — это платформа с открытым исходным кодом, предоставляющая разработчикам набор инструментов для создания приложений. Она включает в себя реляционную базу данных, систему аутентификации, API, хранилище данных и возможность работы с данными в реальном времени. Supabase упрощает процесс разработки, позволяя сосредоточиться на клиентской части приложения, в то время как серверная инфраструктура и управление данными берёт на себя сама платформа.  **Процесс подключения к Supabase**  Создание проекта: Перейдите на сайт Supabase и создайте учетную запись. Создайте новый проект, указав имя и регион.  **Настройка базы данных:**  После создания проекта вы получите доступ к интерфейсу управления базой данных. Вы можете создавать таблицы и управлять данными с помощью SQL-редактора или графического интерфейса.  Подключение к проекту: Откройте файл build.gradle и добавьте необходимые зависимости для работы с Supabase:  Вам нужно добавить разрешение на доступ к интернету в файл AndroidManifest.xml.  <uses-permission android:name="android.permission.INTERNET"/>  Шаг 5: Инициализация Supabase  Создайте класс для работы с Supabase, например, SupabaseClient.kt.  object Supabase {  private const val SUPABASE\_URL = "https://your-project-ref.supabase.co"  private const val SUPABASE\_ANON\_KEY = "your-anon-key"  val client = SupabaseClient(SUPABASE\_URL, SUPABASE\_ANON\_KEY)  }  2. Система авторизации через OTP  Шаг 1: Настройка аутентификации в Supabase  В панели управления Supabase перейдите в раздел Authentication.  Включите провайдер Phone для аутентификации через OTP.  Шаг 2: Реализация интерфейса  Создайте основной интерфейс для ввода электронной почты или номера телефона для получения OTP.  Шаг 3: Проверка OTP  Создайте дополнительный интерфейс для ввода OTP и его проверки |
| **5.Java:**  Один из самых популярных языков для разработки Android-приложений.  Обладает высокой универсальностью и поддерживает объектно-ориентированное программирование.  Преимущества включают богатую экосистему и большое количество библиотек, что облегчает разработку сложных приложений. Недостатком является более сложный синтаксис по сравнению с современными языками, такими как Kotlin[1](https://fortech.dev/blog/yazyki-programmirovaniya-dlya-mobilnyh-platform/)[3](https://firecode.ru/blog/na-chem-razrabatyvat-mobilnoe-prilozhenie).  **Objective-C:**  Основной язык для разработки приложений под iOS до появления Swift.  Поддерживает объектно-ориентированное программирование и имеет мощные возможности для работы с графикой и пользовательским интерфейсом.  Хотя Objective-C все еще используется, большинство новых приложений разрабатываются на Swift, который предлагает более современный синтаксис и улучшенную безопасность[2](https://sky.pro/wiki/javascript/na-kakom-yazyke-pisat-mobilnye-prilozheniya-sovety-i-primery/)[4](https://creazard.ru/mobilnye-prilozheniya/yazyki-i-tehnologii-razrabotki-prilozhenii/).  **C#:**  Используется в основном для разработки кроссплатформенных приложений с помощью фреймворка Xamarin.  Позволяет создавать приложения как для iOS, так и для Android с использованием одного кода.  Преимущества включают возможность использования существующих библиотек .NET и удобный синтаксис. Однако производительность может быть ниже, чем у нативных приложений[5](https://notissimus.com/16-luchshih-yazykov-programmirovaniya-dlya-razrabotki-mobilnyh-prilozhenij-2023/)[6](https://practicum.yandex.ru/blog/yazyki-programmirovaniya-dlya-mobilnoy-razrabotki/).  **Взаимодействие XAML и C#**  XAML используется для описания пользовательского интерфейса в приложениях, разработанных с использованием C#. Это позволяет разделить логику приложения (написанную на C#) от его визуального представления (определяемого в XAML).  **Преимущества взаимодействия:**  Упрощение разработки: разработчики могут сосредоточиться на логике приложения, в то время как дизайнеры могут работать над интерфейсом.  Поддержка привязки данных: XAML позволяет легко связывать данные с элементами интерфейса, что упрощает обновление UI при изменении данных в C#. | 6.JDK (Java Development Kit) - это набор инструментов, необходимых для разработки Java-приложений, включая мобильные приложения на Android.  Android Studio - это интегрированная среда разработки (IDE), специально созданная для разработки приложений под Android. Она предоставляет удобные инструменты для создания, отладки и тестирования мобильных приложений Android.  **Установка JDK:**  Перейдите на официальный сайт Oracle и загрузить последнюю версию JDK для нужной ОС. Запустить инсталлятор и следуйте инструкциям по установке.  **Установка Android Studio:**  Перейдите на официальный сайт Android Studio и загрузить последнюю версию Android Studio для нужной ОС. Запустить инсталлятор и следуйте инструкциям по установке.  **Компоненты Android Studio:**  Android SDK - коллекция инструментов и библиотек для разработки приложений под Android.  Android Virtual Device - эмулятор Android-устройств для тестирования приложений.  Gradle - система сборки и управления зависимостями проекта.  Android Debugger - инструмент для отладки приложений под Android.  Android Profiler - инструмент для анализа производительности и использования ресурсов приложения.  **Компоненты JDK:**  JDK включает следующие компоненты: среда выполнения Java, компилятор Java и API Java.  **Настройка Android Studio:**  Запустить Android Studio и следовать инструкциям мастера установки.  Выбрать путь для установки Android SDK и установите необходимые платформы Android.  Создать или войти в учетную запись разработчика Google для доступа к дополнительным функциям и сервисам.  **Структура проекта Android Studio:**  app/src/main/java - исходный код приложения.  app/src/main/res - ресурсы приложения (макеты, изображения, строки, стили)  app/src/main/AndroidManifest.xml - манифеста, содержащий информацию о приложении, его компонентах и разрешениях.  app/build.gradle - конфигурация проекта, в котором определяются зависимости, настройки версий и другие параметры.  **Запуск приложения Android studio:**  Подключите устройство или запустите AVD.  Выберите устройство в Android Studio и нажмите "Запустить".  **Запуск приложения JDK:**  Создайте новый проект, выбрав JDK.  Выберите шаблон "Command Line App".  Запустите проект через меню "Run". | **7.Создание проекта Xamarin Forms:**   * Файл > Создать > Проект... > Мобильное приложение (Xamarin). * Имя проекта > Создать. * Выберите "Пустой проект Xamarin.Forms" > Создать   **Настройка проекта Xamarin Forms:**   * ПКМ по проекту > Свойства > Сборка Android. * Назначение > Android. * Версия минимального API > Выберите API > OK.   **Создание проекта Xamarin.Android:**   * ПКМ по проекту Xamarin Forms > Добавить > Новый проект... > Android. * Имя проекта > Создать. * Выберите "Пустой проект Android" > Создать   **Настройка проекта Xamarin.Android:**   * ПКМ по проекту > Свойства > Сборка Android. * Назначение > Android. * Версия минимального API > Выберите API > OK.   **Запуск приложения:**   * ПКМ по проекту Xamarin Forms > Запустить без отладки/Отладить. * Приложение запустится на эмуляторе/устройстве | **8.ListView:** ListView используется для отображения списка элементов, в котором пользователь может прокручивать содержимое вертикально. Список может содержать текстовые элементы, изображения или пользовательские представления. Примеры использования ListView включают контактные списки, списки задач или списки новостей. В коде проекта можно создать экземпляр ListView и заполнить его данными из источника данных, например, массива или базы данных.  ListView listView = FindViewById<ListView>(Resource.Id.myListView);  string[] items = new string[] { "item1", "item2", "item3" };  ArrayAdapter<string> adapter = new ArrayAdapter<string>(this, Android.Resource.Layout.SimpleListItem1, items);  listView.Adapter = adapter;  **GridView:** GridView используется для отображения элементов в виде сетки, где каждый элемент представлен в ячейке. GridView обычно используется для отображения коллекции изображений, например, галереи фотографий. В коде проекта можно создать экземпляр GridView, установить адаптер для привязки данных и настроить внешний вид каждого элемента с помощью пользовательских представлений.  <GridView xmlns:android="http://schemas.android.com/apk/res/android"  android:id="@+id/gridView"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:numColumns="3" />  **LinearLayout**: LinearLayout используется для расположения элементов пользовательского интерфейса в виде горизонтального или вертикального списка. Линейный макет позволяет управлять порядком и расположением элементов внутри контейнера. В коде проекта можно создать экземпляр LinearLayout и добавить элементы (например, кнопки, текстовые поля или изображения) с помощью методов AddView().  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="vertical">  <TextView  android:text="Элемент 1"/>  <Button  android:text="Кнопка"/>  </LinearLayout>  **RelativeLayout:** RelativeLayout используется для расположения элементов пользовательского интерфейса с помощью относительных правил и связей между элементами. С помощью RelativeLayout можно создавать сложные макеты, управлять позиционированием и выравниванием элементов. В коде проекта можно создать экземпляр RelativeLayout и задать атрибуты позиционирования для каждого элемента с помощью пользовательских параметров.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content">  <Button  android:id="@+id/ok"  android:text="OK"/>  <Button  android:layout\_toLeftOf="@id/ok"  android:layout\_alignTop="@id/ok"  android:text="Cancel"/>  </RelativeLayout> |