Билет 1

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

История развития:

**Android:**

Android является операционной системой, разработанной компанией Google и выпускается с 2008 года. Первой версией Android была Android 1.0, также известная как "Android Beta". Затем следовали Android 1.1, Android 1.5 Cupcake, Android 1.6 Donut, Android 2.0/2.1 Eclair, Android 2.2 Froyo и так далее.

В 2011 году была представлена версия Android 4.0 Ice Cream Sandwich, объединившая функциональность смартфонов и планшетов. За ней последовали Android 4.1/4.2/4.3 Jelly Bean, Android 4.4 KitKat, Android 5.0/5.1 Lollipop, Android 6.0 Marshmallow, Android 7.0/7.1 Nougat, Android 8.0/8.1 Oreo, Android 9.0 Pie и Android 10.0 Q.

Каждая новая версия Android вносит улучшения в интерфейс, функциональность, производительность и безопасность системы.

**iOS:**

iOS - это мобильная операционная система, разработанная компанией Apple. Она была представлена в 2007 году вместе с первым iPhone. Первой версией iOS была iPhone OS 1.0, затем она была переименована в iOS.

Затем последовали версии iOS 2.0, iOS 3.0, iOS 4.0, iOS 5.0, iOS 6.0, iOS 7.0, iOS 8.0, iOS 9.0, iOS 10.0, iOS 11.0, iOS 12.0, iOS 13.0 и настоящая версия - iOS 14.0.

Каждая новая версия iOS также вносит улучшения в интерфейс, функциональность, производительность и безопасность системы.

**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, гибридные и кроссплатформенные приложения мобильных устройств являются технологиями разработки программного обеспечения для мобильных устройств. Они отличаются друг от друга в терминах используемых языков программирования, доступности для разных платформ и производительности.

**Нативные приложения:**

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

Примеры: Instagram для iOS, Facebook для Android.

Технология создания: Для разработки нативных приложений на iOS используется язык программирования Swift или Objective-C, а для Android - Java или Kotlin.

**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: WhatsApp, Slack.

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-приложения мобильных устройств, также известные как мобильные веб-приложения, представляют собой программные приложения, которые доступны через интернет и выполняются на мобильных устройствах, таких как смартфоны или планшеты.

**Технология создания веб-приложений** включает использование таких языков программирования, как HTML (язык разметки гипertext markup language), CSS (каскадные таблицы стилей) 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.

**Технология создания веб-сервиса** отличается от создания веб-приложения. Веб-сервис предоставляет функциональность или данные через интернет и может быть использован в различных приложениях и сервисах. Технология создания веб-сервиса обычно включает использование таких языков программирования, как XML (расширяемый язык разметки) и SOAP (простой обьектный доступ в протоколах).

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:**

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

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

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

**Компиляция:**

Если вы разрабатываете приложение для iOS, скомпилируйте его в Xcode, используя подходящие настройки и профили разработчика. Убедитесь, что приложение успешно компилируется без ошибок и предупреждений.

Если вы разрабатываете приложение для Android, скомпилируйте его в Android Studio или другой среде разработки для Android. Опять же, убедитесь, 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. Вам потребуется ключ для подписи приложения, который можно создать в Google Play Console. После создания, подписанного APK файла, вы сможете загрузить его 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#.

**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 (https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) и загрузите последнюю версию JDK для вашей операционной системы.

Запустите загруженный инсталлятор и следуйте инструкциям по установке.

**Установка Android Studio:**

Перейдите на официальный сайт Android Studio (https://developer.android.com/studio) и загрузите последнюю версию Android Studio для вашей операционной системы.

Запустите загруженный инсталлятор и следуйте инструкциям по установке.

**Компоненты Android Studio:**

Android SDK (Software Development Kit) - коллекция инструментов и библиотек для разработки приложений под Android.

Android Virtual Device (AVD) - 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:**

Запустите Android Studio и следуйте инструкциям мастера установки.

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

Создайте или войдите в учетную запись разработчика Google для доступа к дополнительным функциям и сервисам.

**Настройка JDK:**

1. Откройте IntelliJ IDEA.
2. Перейдите в раздел «Файл» > «Структура проекта» > «SDKs».
3. Нажмите на знак «+» для добавления нового JDK.
4. Перейдите в каталог установки JDK, который вы установили ранее, и выберите его. IntelliJ загрузит JDK.
5. Убедитесь, что JDK выбран для вашего проекта: перейдите в раздел «Файл» > «Структура проекта» > «Проект».
6. В разделе «Project SDK» выберите JDK, который вы только что добавили.
7. Примените изменения и закройте окно.

**Структура проекта Android Studio:**

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

app/src/main/res - содержит ресурсы приложения, такие как макеты, изображения, строки, стили и т. д.

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

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

**Структура проекта JDK**

1. Разделение исходного кода и тестового кода. Это позволяет лучше управлять зависимостями и обеспечивает чёткое различие между логикой приложения и кодом, используемым для проверки его корректности.
2. Следование стандартам именования пакетов. Это улучшает читаемость кода и облегчает навигацию в базе кода.
3. Использование Maven или Gradle для управления зависимостями. Это позволяет легко добавлять или обновлять зависимости, отслеживать транзитивные зависимости и управлять версиями конфликтов.
4. Применение принципов модульного проектирования. Это способствует улучшению ремонтопригодности и повышению повторного использования кода.
5. Документация кода. Это помогает понять проект.

**Запуск приложения Android studio:**

Подключите Android-устройство или запустите эмулятор Android Virtual Device (AVD).

В Android Studio откройте проект и выберите нужное устройство из списка доступных устройств.

Нажмите на кнопку "Запустить" (зеленый треугольник) или используйте комбинацию клавиш Shift + F10 для запуска приложения.

**Запуск приложения JDK:**

1. В стартовом окне выберите пункт «Create New Project».
2. В следующем окне выберите JDK, который будет использован в проекте. Нажмите кнопку «New...» и укажите путь к установленному ранее JDK.
3. В следующем окне выберите пункт «Create project from template» и выберите «Command Line App».
4. Укажите название проекта и Base Package проекта.
5. После завершения мастера нового проекта IDEA сгенерирует новый пустой проект.
6. Запустите проект. Для этого можно выбрать пункт «Run» — «Run „Main“», нажать комбинацию клавиш Shift+F10 или на зелёный треугольник справа в верхней части редактора.

**Билет 7**

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

• Откройте Visual Studio.

• Выберите «Файл» > «Создать» > «Проект...».

• В диалоговом окне «Новый проект» выберите «Мобильное приложение (Xamarin)» в разделе «Шаблоны».

• Введите имя проекта и нажмите «Создать».

• Выберите шаблон «Пустой проект Xamarin.Forms».

• Нажмите «Создать».

1. **Настройка проекта Xamarin Forms:**

• В обозревателе решений щелкните правой кнопкой мыши проект Xamarin Forms и выберите «Свойства».

• В диалоговом окне «Свойства» перейдите на вкладку «Сборка Android».

• В разделе «Назначение» выберите «Android».

• В разделе «Версия минимального API» выберите минимальную версию API Android, которую вы хотите поддерживать.

• Нажмите «ОК».

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

• В обозревателе решений щелкните правой кнопкой мыши проект Xamarin Forms и выберите «Добавить» > «Новый проект...».

• В диалоговом окне «Новый проект» выберите «Android» в разделе «Шаблоны».

• Введите имя проекта и нажмите «Создать».

• Выберите шаблон «Пустой проект Android».

• Нажмите «Создать».

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

• В обозревателе решений щелкните правой кнопкой мыши проект Xamarin.Android и выберите «Свойства».

• В диалоговом окне «Свойства» перейдите на вкладку «Сборка Android».

• В разделе «Назначение» выберите «Android».

• В разделе «Версия минимального API» выберите минимальную версию API Android, которую вы хотите поддерживать.

• Нажмите «ОК».

1. **Запуск приложения:**

• В обозревателе решений щелкните правой кнопкой мыши проект Xamarin Forms и выберите «Запустить без отладки» или «Отладить».

• Приложение запустится на эмуляторе или устройстве Android.

**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:**

// Создание экземпляра ListView

ListView listView = new ListView(this);

// Источник данных - массив строк

string[] data = new string[] { "Элемент 1", "Элемент 2", "Элемент 3" };

// Создание адаптера для привязки данных к ListView

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

// Установка адаптера для ListView

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:**

// Создание экземпляра GridView

GridView gridView = new GridView(this);

// Источник данных - массив изображений

int[] images = new int[] { Resource.Drawable.image1, Resource.Drawable.image2, Resource.Drawable.image3 };

// Создание адаптера для привязки данных к GridView

ImageAdapter adapter = new ImageAdapter(this, images);

// Установка адаптера для GridView

gridView.Adapter = adapter;

**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:**

// Создание экземпляра LinearLayout

LinearLayout linearLayout = new LinearLayout(this);

// Установка ориентации линейного макета на вертикальную

linearLayout.Orientation = Orientation.Vertical;

// Создание элементов пользовательского интерфейса

Button button1 = new Button(this);

button1.Text = "Кнопка 1";

Button button2 = new Button(this);

button2.Text = "Кнопка 2";

// 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:**

// Создание экземпляра RelativeLayout

RelativeLayout relativeLayout = new RelativeLayout(this);

// Создание элемента пользовательского интерфейса

Button button = new Button(this);

button.Text = "Кнопка";

// Задание параметров позиционирования кнопки

RelativeLayout.LayoutParams layoutParams = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WrapContent, RelativeLayout.LayoutParams.WrapContent);

layoutParams.AddRule(LayoutRules.CenterInParent);

// Установка параметров для кнопки

button.LayoutParameters = layoutParams;

// Добавление кнопки в RelativeLayout

relativeLayout.AddView(button);

**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 (internationalization)** - 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 (localization)** - 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));

**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.

**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.

**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.

**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.

**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**

1. Элементы управления в Xamarin.Android: виды, описание, назначение, примеры, принцип использования в коде проекта.

**Виды элементов управления в Xamarin.Android:**

* TextView:

Описание: Элемент для отображения текста.

Назначение: Используется для вывода статического текста или динамического текста из кода.

* EditText:

Описание: Поле для ввода текста.

Назначение: Позволяет пользователю вводить текст с клавиатуры.

* Button:

Описание: Кнопка, на которую пользователь может нажать.

Назначение: Используется для обработки событий нажатия, например, для запуска действий при клике.

* ImageView:  
  Описание: Виджет для отображения изображений.

Назначение: Предназначен для показа графических изображений.

* CheckBox:

Описание: Флажок для отметки/снятия отметки.

Назначение: Используется, когда нужно предоставить пользователю возможность выбора из нескольких вариантов.

* RadioButton:  
  Описание: Кнопка-переключатель для выбора одного из нескольких вариантов.  
  Назначение: Позволяет пользователю выбрать один из нескольких взаимоисключающих вариантов.

**Принцип использования в коде проекта:**

// Пример работы с TextView

TextView textView = FindViewById<TextView>(Resource.Id.textView);

string textValue = textView.Text;

// Пример обработки нажатия кнопки

Button button = FindViewById<Button>(Resource.Id.button);

button.Click += (sender, e) =>

{

// Обработка нажатия кнопки

};

// Пример получения текста из EditText

EditText editText = FindViewById<EditText>(Resource.Id.editText);

string enteredText = editText.Text;

**Билет 22**

1. Тестирование мобильных приложений: юзабилити тестирование, ручное и автоматическое тестирование.

**Юзабилити тестирование:**

* Определение удобства использования: Это тестирование оценивает, насколько приложение удовлетворяет потребности пользователей, насколько легко они могут взаимодействовать с приложением и выполнять необходимые задачи.
* Тестирование пользовательского интерфейса (UI) и пользовательского опыта (UX): Оценка элементов дизайна, навигации, цветовой схемы, шрифтов и других аспектов, влияющих на общее восприятие приложения.

**Ручное тестирование:**

* Функциональное тестирование: Проверка работы отдельных функций приложения, убеждаясь, что они выполняются правильно.
* Тестирование совместимости: Проверка работы приложения на различных устройствах, разрешениях экрана и версиях операционных систем.
* Тестирование производительности: Оценка отклика приложения при различных условиях использования (например, при низкой батарее, слабом интернет-соединении).

**Автоматическое тестирование:**

* Тестирование интерфейса (UI тестирование): Автоматизированная проверка элементов пользовательского интерфейса для выявления ошибок в их работе.
* Тестирование API: Проверка взаимодействия приложения с внешними сервисами через API.
* Тестирование производительности: Использование автоматизированных средств для оценки производительности приложения при различных условиях.

**Билет 23.**

1. Тестирование мобильных приложений: функциональное тестирование, тестирование производительности.

**Функциональное тестирование мобильных приложений:**

Цели функционального тестирования:

1. Проверка работы основных функций:

Проверка, что все основные функции приложения работают корректно. Например, если это приложение для социальных сетей, функции отправки сообщений, просмотра новостей, добавления друзей и т.д.

1. Тестирование пользовательского ввода:

Проверка корректности обработки ввода пользователя через различные элементы интерфейса: кнопки, текстовые поля, жесты, и так далее.

1. Взаимодействие с внешними системами:

Проверка корректной интеграции с внешними сервисами, если они используются (например, платежные системы, социальные сети, геолокация).

1. Обработка ошибок:

Проверка того, как приложение обрабатывает ошибки и нестандартные ситуации, такие как отсутствие интернет-соединения, неверные данные и т.д.

1. Тестирование совместимости:

Проверка работы приложения на различных устройствах (различные модели телефонов и планшетов), разрешениях экранов и версиях операционных систем.

**Тестирование производительности мобильных приложений:**

Цели тестирования производительности:

1. Отклик приложения:

Оценка времени ответа приложения на пользовательский ввод. Это включает в себя время загрузки страниц, открытия окон и выполнения основных функций.

1. Потребление ресурсов:

Измерение потребления ресурсов устройства, таких как CPU, память и батарея, в различных сценариях использования приложения.

1. Стабильность приложения:

Проверка стабильности приложения в течение длительного времени использования. Тестирование на утечки памяти, перегрев устройства и другие проблемы, которые могут привести к сбоям.

1. Тестирование при нагрузке:

Проверка поведения приложения при максимальной нагрузке, например, при одновременном использовании множеством пользователей.

1. Тестирование сети:

Оценка производительности приложения при различных условиях сети, таких как медленное 3G-соединение или отсутствие сети.