Все приложения делятся на три типа.

Мобильные веб-приложения Наиболее распространённый тип, который с лёгкостью позволяет отображать сайты на различных устройствах. Это не приложения в чистом виде, а адаптированные под мобильные устройства интерфейсы сайтов с активной функциональностью. Веб-приложения отличаются кросс-платформенностью, их не нужно устанавливать, и они достаточно просты в использовании. К другим преимуществам этого типа ПО относится достаточно простая и оперативная разработка. Но есть у веб-приложений и недостатки: необходимость подключаться к интернету для работы, ограниченная безопасность, невысокая производительность.

Нативные приложения Такие приложения разработаны только под определённую платформу и по максимуму используют возможность той или иной операционной системы. Нативные приложения могут задействовать и иные ресурсы гаджета: камера, навигатор, список контактов и прочее. Они характеризуются широкой функциональностью и высокой скоростью работы. Но есть у подобных приложений и недостатки: низкий по сравнению с мобильными веб-приложениями охват платформ, высокая стоимость разработки и необходимость регулярно выпускать обновления.

Гибридные приложения Комбинация двух вышеупомянутых типов. В отличие от нативных, гибридные приложения разрабатываются для нескольких платформ одновременно и пишутся на универсальном языке. Такой продукт можно размещать в сторах, а для его обновления не нужно каждый раз выпускать новую версию. Достаточно лишь добавить все изменения на сервер. Слабое место подобных приложений — визуальный стиль. Ведь разработка приложения для конкретной платформы ведётся на основе единого гайдлайна. А интерфейс гибридного приложения теряет узнаваемые черты операционной системы. Конечно, разработка таких приложений обходится дешевле, но и потенциально уязвимых мест больше.

Особенности тестирования web-приложений:

Мобильным веб-приложением является веб-сайт, который открывается в гаджете (смартфоне или планшете) с помощью мобильного браузера.

Достоинства мобильных веб-приложений:

- Простая разработка.
- Легкий доступ.
- Простое обновление.
- Мобильные веб-приложения не требует установки.

Недостатки мобильных веб-приложений:

- Нет поддержки автономных функций.
- Ограниченная функциональность в сравнении с гибридными и нативными приложениями (нет доступа к файловой системе и локальным ресурсам)
- Проблемы с перераспределением: Google Play и App Store не поддерживают перераспределение мобильных веб-приложений.

Нативное приложение — это приложение, разработанное специально для одной платформы (Android, iOS, BlackBerry).

Достоинства нативных приложений:

- Нативное приложение работает в автономном режиме.
- Оно может использовать все функции своего устройства.

- Продвинутый пользовательский интерфейс.
- Push-уведомления для удобства пользователей.

Недостатки нативных приложений:

- Разработка нативных приложений обходится дороже в сравнении с мобильными веб-приложениями.
- Требуется больших затрат на техническое обслуживание.

Гибридное приложение — это сочетание нативного и мобильного веб-приложений. Его можно определить как отображение содержимого мобильного сайта в формате приложения.

Достоинства гибридных приложений:

- Более рентабельно в сравнении с нативным приложением.
- Простое распространение.
- Встроенный браузер.
- Особенности устройства.

Недостатки гибридных приложений:

- Работает не так быстро, как нативное приложение.
- Графика менее адаптирована к ОС в сравнении с нативным приложением.

Требования к разработке приложений

Android

Приложения для Android можно писать на языках Kotlin, Java и C++. Инструменты Android SDK компилируют ваш код вместе с любыми файлами данных и ресурсов в APK или Android App Bundle.

Пакет *Android* , который представляет собой архивный файл с .apkсуффиксом, содержит содержимое приложения Android, необходимое во время выполнения, и это файл, который устройства на базе Android используют для установки приложения.

Пакет Android App Bundle, представляющий собой архивный файл с .aabсуффиксом, содержит содержимое проекта приложения Android, включая некоторые дополнительные метаданные, которые не требуются во время выполнения. AAB — это формат публикации, который нельзя установить на устройствах Android. Он откладывает создание и подписание APK на более поздний этап. Например, при распространении вашего приложения через Google Play серверы Google Play генерируют оптимизированные APK-файлы, содержащие только те ресурсы и код, которые требуются конкретному устройству, запрашивающему установку приложения.

Каждое приложение для Android находится в собственной изолированной программной среде, защищенной следующими функциями безопасности Android:

- Операционная система Android — это многопользовательская система Linux, в которой каждое приложение является отдельным пользователем.
- По умолчанию система назначает каждому приложению уникальный идентификатор пользователя Linux (идентификатор используется только системой и неизвестен приложению). Система устанавливает разрешения для всех файлов

в приложении, чтобы только идентификатор пользователя, назначенный этому приложению, мог получить к ним доступ.
- У каждого процесса есть собственная виртуальная машина (ВМ), поэтому код приложения выполняется изолированно от других приложений.
- По умолчанию каждое приложение работает в своем собственном процессе Linux. Система Android запускает процесс, когда необходимо выполнить какой-либо из компонентов приложения, а затем останавливает процесс, когда он больше не нужен или когда системе необходимо восстановить память для других приложений.

В системе Android реализован *принцип наименьших привилегий* . То есть каждое приложение по умолчанию имеет доступ только к тем компонентам, которые ему необходимы для работы, и не более того. Это создает очень безопасную среду, в которой приложение не может получить доступ к частям системы, для которых у него нет разрешения. Однако есть способы, которыми приложение может обмениваться данными с другими приложениями и получать доступ к системным службам:

- Два приложения могут использовать один и тот же идентификатор пользователя Linux, и в этом случае они смогут получать доступ к файлам друг друга. Для экономии системных ресурсов приложения с одним и тем же идентификатором пользователя также могут запускаться в одном процессе Linux и совместно использовать одну и ту же виртуальную машину. Приложения также должны быть подписаны одним и тем же сертификатом.
- Приложение может запросить разрешение на доступ к данным устройства, таким как местоположение устройства, камера и соединение Bluetooth. Пользователь должен явно предоставить эти разрешения. Дополнительные сведения см. в разделе Работа с системными разрешениями .

В остальной части этого документа вводятся следующие понятия:

- Основные компоненты фреймворка, определяющие ваше приложение.
- Файл манифеста, в котором вы объявляете компоненты и необходимые функции устройства для вашего приложения.
- Ресурсы, которые отделены от кода приложения и позволяют вашему приложению изящно оптимизировать свое поведение для различных конфигураций устройств.

Компоненты приложения

Компоненты приложения — это основные строительные блоки приложения для Android. Каждый компонент — это точка входа, через которую система или пользователь могут войти в ваше приложение. Одни компоненты зависят от других.

Существует четыре различных типа компонентов приложения:

- мероприятия
- Услуги
- Вещательные приемники
- Контент-провайдеры

Каждый тип служит определенной цели и имеет свой жизненный цикл, который определяет, как компонент создается и уничтожается. В следующих разделах описываются четыре типа компонентов приложения.

мероприятия

Активность — это точка входа для взаимодействия с пользователем. Он представляет собой один экран с пользовательским интерфейсом. Например, приложение электронной почты может иметь одно действие, отображающее список новых сообщений электронной почты, другое действие для составления электронного письма и еще одно действие для чтения электронных писем. Несмотря на то, что действия работают вместе, чтобы сформировать целостный пользовательский интерфейс в почтовом приложении, каждое из них не зависит от других. Таким образом, любое из этих действий может быть запущено другим приложением, если это разрешено приложением электронной почты. Например, приложение камеры может запустить действие в приложении электронной почты, которое составляет новую почту, чтобы пользователь мог поделиться изображением. Действие облегчает следующие ключевые взаимодействия между системой и приложением:

- Отслеживание того, что в данный момент волнует пользователя (что находится на экране), чтобы убедиться, что система продолжает выполнять процесс, в котором выполняется действие.
- Знание того, что ранее использованные процессы содержат вещи, к которым пользователь может вернуться (остановленные действия), и, следовательно, более высокий приоритет сохранения этих процессов.
- Помочь приложению справиться с уничтожением его процесса, чтобы пользователь мог вернуться к действиям с восстановленным предыдущим состоянием.
- Предоставление приложениям способа реализовать потоки пользователей между собой, а системе — координировать эти потоки. (Самый классический пример здесь — поделиться.)

Вы реализуете действие как подкласс Activityкласса. Дополнительные сведения о Activityклассе см. в руководстве разработчика Activity .

Услуги

Служба — это универсальная точка входа для поддержания работы приложения в фоновом режиме по разным причинам . Это компонент, который работает в фоновом режиме для выполнения длительных операций или выполнения работы для удаленных процессов. Служба не предоставляет пользовательский интерфейс. Например, служба может воспроизводить музыку в фоновом режиме, пока пользователь находится в другом приложении, или может получать данные по сети, не блокируя взаимодействие пользователя с действием. Другой компонент, например действие, может запустить службу и позволить ей работать или привязываться к ней для взаимодействия с ней.

Существует два типа служб, которые сообщают системе, как управлять приложением: запущенные службы и связанные службы.

Запущенные службы сообщают системе о необходимости продолжать их работу до тех пор, пока их работа не будет завершена. Это может быть синхронизация некоторых данных в фоновом режиме или воспроизведение музыки даже после того, как пользователь покинет приложение. Синхронизация данных в фоновом режиме или воспроизведение музыки также представляют два разных типа запущенных служб, которые изменяют то, как система их обрабатывает:

- Воспроизведение музыки — это то, о чем пользователь непосредственно знает, поэтому приложение сообщает об этом системе, говоря, что оно хочет быть на

переднем плане с уведомлением, чтобы сообщить об этом пользователю; в этом случае система знает, что она должна изо всех сил стараться поддерживать процесс этой службы в рабочем состоянии, потому что пользователь будет недоволен, если он исчезнет.

- Обычная фоновая служба не является чем-то, о чем пользователь непосредственно знает, как работает, поэтому система имеет больше свободы в управлении своим процессом. Это может позволить его убить (а затем перезапустить службу через некоторое время), если ему требуется ОЗУ для вещей, которые имеют более непосредственное значение для пользователя.

Связанные службы запускаются, потому что какое-то другое приложение (или система) заявило, что хочет использовать службу. По сути, это служба, предоставляющая API другому процессу. Таким образом, система знает, что между этими процессами существует зависимость, поэтому, если процесс A связан со службой в процессе B, она знает, что ей необходимо поддерживать работу процесса B (и его службы) для A. Кроме того, если процесс A является чем-то о чем заботится пользователь, то он также знает, что процесс B следует рассматривать как нечто, о чем также заботится пользователь.

Благодаря своей гибкости (к лучшему или к худшему) сервисы оказались действительно полезным строительным блоком для всех видов системных концепций более высокого уровня. Живые обои, прослушиватели уведомлений, экранные заставки, методы ввода, службы специальных возможностей и многие другие основные системные функции построены как службы, которые реализуются приложениями, и система привязывается к ним, когда они должны работать.

Служба реализована как подкласс Service. Дополнительные сведения о Servicеклассе см. в руководстве разработчика служб .

Примечание. Если ваше приложение предназначено для Android 5.0 (уровень API 21) или более поздней версии, используйте **`JobScheduler`**класс для планирования действий. Преимущество JobScheduler заключается в экономии заряда батареи за счет оптимального планирования заданий для снижения энергопотребления и работы с Doze API. Дополнительные сведения об использовании этого класса см. в **`JobScheduler`** справочной документации.

Вещательные приемники

A *приемник*— это компонент, который позволяет системе доставлять события в приложение вне обычного пользовательского потока, позволяя приложению реагировать на общесистемные широковещательные объявления. Поскольку широковещательные приемники — это еще одна четко определенная запись в приложении, система может доставлять широковещательные сообщения даже приложениям, которые в данный момент не запущены. Так, например, приложение может запланировать будильник, чтобы опубликовать уведомление, чтобы сообщить пользователю о предстоящем событии... и, доставив этот будильник в BroadcastReceiver приложения, нет необходимости, чтобы приложение продолжало работать до тех пор, пока срабатывает сигнализация. Многие широковещательные сообщения исходят из системы — например, широковещательное сообщение о выключении экрана, низком уровне заряда батареи или о том, что была сделана фотография. Приложения также могут инициировать широковещательные рассылки, например, чтобы сообщить другим приложениям, что некоторые данные были загружены на устройство и доступны для использования. Хотя широковещательные приемники не отображают пользовательский интерфейс, они могутсоздать уведомление в строке состояния, чтобы предупредить пользователя о возникновении широковещательного события. Однако чаще широковещательный приемник является просто *шлюзом* для других компонентов и предназначен

для выполнения очень минимального объема работы. Например, он может запланировать JobServiceвыполнение некоторой работы на основе события с помощьюJobScheduler

Широковещательный приемник реализован как подкласс, BroadcastReceiver и каждая широковещательная рассылка доставляется как Intentобъект. Для получения дополнительной информации см BroadcastReceiver. класс.

Контент-провайдеры

Поставщик *контента* управляет общим набором данных приложения, которые вы можете хранить в файловой системе, в базе данных SQLite, в Интернете или в любом другом постоянном хранилище, к которому может получить доступ ваше приложение. Через поставщика содержимого другие приложения могут запрашивать или изменять данные, если поставщик содержимого разрешает это. Например, система Android предоставляет поставщика контента, который управляет контактной информацией пользователя. Таким образом, любое приложение с соответствующими разрешениями может запрашивать поставщика контента, например ContactsContract.Data, читать и записывать информацию о конкретном человеке. Заманчиво думать о поставщике контента как об абстракции базы данных, потому что для этого распространенного случая в них встроено множество API и поддержки. Однако они имеют другую основную цель с точки зрения проектирования системы. Для системы поставщик контента — это точка входа в приложение для публикации именованных элементов данных, идентифицируемых схемой URI. Таким образом, приложение может решить, как оно хочет сопоставить содержащиеся в нем данные с пространством имен URI, передавая эти URI другим объектам, которые, в свою очередь, могут использовать их для доступа к данным. Есть несколько конкретных вещей, которые это позволяет системе делать при управлении приложением:

- Назначение URI не требует, чтобы приложение продолжало работать, поэтому URI могут сохраняться после выхода из приложений, которым они принадлежат. Системе нужно только убедиться, что приложение-владелец все еще работает, когда ей нужно получить данные приложения из соответствующего URI.
- Эти URI также обеспечивают важную детализированную модель безопасности. Например, приложение может поместить URI своего изображения в буфер обмена, но оставить поставщика содержимого заблокированным, чтобы другие приложения не могли получить к нему свободный доступ. Когда второе приложение пытается получить доступ к этому URI в буфере обмена, система может разрешить этому приложению доступ к данным через предоставление временного *разрешения URI* , чтобы ему был разрешен доступ к данным только за этим URI, но не во втором приложении. .

Поставщики контента также полезны для чтения и записи данных, которые являются частными для вашего приложения и недоступны для общего доступа.

Поставщик содержимого реализуется как подкласс ContentProvider и должен реализовывать стандартный набор API, которые позволяют другим приложениям выполнять транзакции. Дополнительные сведения см. в руководстве разработчика поставщиков контента .

Уникальный аспект дизайна системы Android заключается в том, что любое приложение может запускать компонент другого приложения. Например, если вы хотите, чтобы пользователь сделал снимок с помощью камеры устройства, вероятно, есть другое приложение, которое делает это, и ваше приложение может использовать его вместо того, чтобы разрабатывать действие для самостоятельного захвата фотографии. Вам не нужно включать или даже ссылаться на код из

приложения камеры. Вместо этого вы можете просто начать действие в приложении камеры, которое делает снимок. По завершении фотография даже возвращается в ваше приложение, чтобы вы могли ее использовать. Пользователю кажется, что камера на самом деле является частью вашего приложения.

Когда система запускает компонент, она запускает процесс для этого приложения, если оно еще не запущено, и создает экземпляры классов, необходимых для компонента. Например, если ваше приложение запускает действие в приложении камеры, которое делает снимок, это действие выполняется в процессе, принадлежащем приложению камеры, а не в процессе вашего приложения. Поэтому, в отличие от приложений в большинстве других систем, приложения для Android не имеют единой точки входа (нет main()функции).

Поскольку система запускает каждое приложение в отдельном процессе с правами доступа к файлам, которые ограничивают доступ к другим приложениям, ваше приложение не может напрямую активировать компонент из другого приложения. Однако система Android может. Чтобы активировать компонент в другом приложении, отправьте в систему сообщение, в котором указывается ваше *намерение* запустить определенный компонент. Затем система активирует компонент для вас.

Активация компонентов

Три из четырех типов компонентов — действия, службы и широковещательные приемники — активируются асинхронным сообщением, называемым *намерением* . Намерения связывают отдельные компоненты друг с другом во время выполнения. Вы можете думать о них как о мессенджерах, которые запрашивают действие у других компонентов, независимо от того, принадлежит ли компонент вашему приложению или другому.

Намерение создается с помощью Intentобъекта, который определяет сообщение для активации либо определенного компонента (явное намерение), либо компонента определенного *типа* (неявное намерение).

Для действий и служб намерение определяет действие, которое нужно выполнить (например, *просмотреть* или *отправить* что-либо), и может указать URI данных, с которыми нужно действовать, среди прочего, что может потребоваться знать запускаемому компоненту. Например, намерение может передавать запрос активности на отображение изображения или открытие веб-страницы. В некоторых случаях вы можете запустить действие для получения результата, и в этом случае действие также возвращает результат в формате Intent. Например, вы можете указать намерение позволить пользователю выбрать личный контакт и вернуть его вам. Намерение возврата включает URI, указывающий на выбранный контакт.

Для широковещательных приемников намерение просто определяет транслируемое объявление. Например, широковещательная рассылка, указывающая, что батарея устройства разряжена, включает только известную строку действия, указывающую, что *батарея разряжена* .

В отличие от действий, служб и приемников вещания, поставщики контента не активируются намерениями. Скорее, они активируются по запросу от ContentResolver. Преобразователь содержимого обрабатывает все прямые транзакции с поставщиком содержимого, поэтому компоненту, выполняющему транзакции с поставщиком, это не нужно, и вместо этого он вызывает методы ContentResolverобъекта. Это оставляет уровень абстракции между поставщиком контента и компонентом, запрашивающим информацию (для безопасности).

Существуют отдельные способы активации каждого типа компонента:

- Вы можете запустить действие или дать ему что-то новое, передав Intentto startActivity()или startActivityForResult() (если вы хотите, чтобы действие возвращало результат).
- В Android 5.0 (уровень API 21) и более поздних версиях вы можете использовать JobSchedulerкласс для планирования действий. В более ранних версиях Android вы можете запустить службу (или дать новые инструкции текущей службе), передав Intentв startService(). Вы можете привязаться к службе, передав Intentto bindService().
- Вы можете инициировать широковещательную рассылку, передав Intentметоды to, такие как sendBroadcast(), sendOrderedBroadcast()или sendStickyBroadcast().
- Вы можете выполнить запрос к поставщику содержимого, вызвав query()файл ContentResolver.

Дополнительные сведения об использовании намерений см. в документе « Намерения и фильтры намерений » . В следующих документах содержится дополнительная информация об активации конкретных компонентов: Действия , Службы , BroadcastReceiveri Поставщики содержимого .

Файл манифеста

Прежде чем система Android сможет запустить компонент приложения, система должна узнать, что компонент существует, прочитав *файл манифеста* приложения , AndroidManifest.xml. Ваше приложение должно объявить все свои компоненты в этом файле, который должен находиться в корне каталога проекта приложения.

В дополнение к объявлению компонентов приложения манифест выполняет ряд действий, например следующее:

- Определяет любые пользовательские разрешения, необходимые приложению, например доступ в Интернет или доступ для чтения к контактам пользователя.
- Объявляет минимальный уровень API , требуемый приложением, в зависимости от того, какие API использует приложение.
- Объявляет аппаратные и программные функции, используемые или требуемые приложением, такие как камера, службы Bluetooth или мультисенсорный экран.
- Объявляет библиотеки API, с которыми должно быть связано приложение (кроме API платформы Android), например библиотеку Google Maps .

Объявление компонентов

Основная задача манифеста — информировать систему о компонентах приложения. Например, файл манифеста может объявить действие следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
              android:label="@string/example_label" ... >
    </activity>
    ...
  </application>
</manifest>
```

В <application> элементе android:iconатрибут указывает на ресурсы для значка, идентифицирующего приложение.

В <activity>элементе android:nameатрибут указывает полное имя класса Activityподкласса, а android:labelатрибут указывает строку, используемую в качестве видимой для пользователя метки для действия.

Вы должны объявить все компоненты приложения, используя следующие элементы:

- <activity>элементы для занятий.
- <service>элементы для обслуживания.
- <receiver>элементы для широковещательных приемников.
- <provider>элементы для контент-провайдеров.

Действия, службы и поставщики контента, которые вы включаете в исходный код, но не объявляете в манифесте, не видны системе и, следовательно, никогда не могут быть запущены. Однако широковещательные приемники могут быть либо объявлены в манифесте, либо созданы динамически в коде как BroadcastReceiverобъекты и зарегистрированы в системе с помощью вызова registerReceiver().

Дополнительные сведения о том, как структурировать файл манифеста для вашего приложения, см . в документации по файлу AndroidManifest.xml .

Объявление возможностей компонента

Как обсуждалось выше, в разделе « Активация компонентов » вы можете использовать Intentдля запуска действий, служб и широковещательных приемников. Вы можете использовать Intent , явно назвав целевой компонент (используя имя класса компонента) в намерении. Вы также можете использовать неявное намерение, которое описывает тип выполняемого действия и, при необходимости, данные, с которыми вы хотите выполнить действие. Неявное намерение позволяет системе найти на устройстве компонент, который может выполнить действие, и запустить его. Если есть несколько компонентов, которые могут выполнять действие, описанное намерением, пользователь выбирает, какой из них использовать.

Предупреждение. Если вы используете намерение для запуска **`Service`**, убедитесь, что ваше приложение защищено с помощью явного намерения. Использование неявного намерения для запуска службы представляет собой угрозу безопасности, поскольку вы не можете быть уверены, какая служба ответит на намерение, а пользователь не может видеть, какая служба запускается. Начиная с Android 5.0 (уровень API 21), система выдает исключение, если вы вызываете **`bindService()`** с неявным намерением. Не объявляйте фильтры намерений для своих служб.

Система определяет компоненты, которые могут реагировать на намерение, сравнивая полученное *намерение с фильтрами намерений* , предоставленными в файле манифеста других приложений на устройстве.

Когда вы объявляете действие в манифесте своего приложения, вы можете дополнительно включить фильтры намерений, которые объявляют возможности действия, чтобы оно могло реагировать на намерения других приложений. Вы можете объявить фильтр намерений для своего компонента, добавив <intent-filter>элемент в качестве дочернего элемента элемента объявления компонента.

Например, если вы создаете приложение электронной почты с действием для составления нового электронного письма, вы можете объявить фильтр намерений для ответа на намерения «отправить» (для отправки нового электронного письма), как показано в следующем примере:

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Если другое приложение создает намерение с ACTION_SENDдействием и передает его startActivity(), система может запустить вашу активность, чтобы пользователь мог составить и отправить электронное письмо.

Дополнительные сведения о создании фильтров намерений см. в документе « Намерения и фильтры намерений » .

Объявление требований к приложению

Существует множество устройств на базе Android, и не все из них обладают одинаковыми функциями и возможностями. Чтобы ваше приложение не было установлено на устройствах, на которых отсутствуют функции, необходимые вашему приложению, важно четко определить профиль для типов устройств, которые поддерживает ваше приложение, объявив требования к устройствам и программному обеспечению в файле манифеста. Большинство этих объявлений носят информационный характер, и система их не читает, но внешние сервисы, такие как Google Play, считывают их, чтобы обеспечить фильтрацию для пользователей, когда они ищут приложения со своего устройства.

Например, если вашему приложению требуется камера и используются API, представленные в Android 8.0 (уровень API 26), вы должны объявить эти требования.

Значения для minSdkVersionи targetSdkVersionустанавливаются в build.gradleфайле вашего модуля приложения:

```
android {
  ...
  defaultConfig {
    ...
    minSdkVersion 26
    targetSdkVersion 29
  }
}
```

Примечание. Не устанавливайте **`minSdkVersion`**и **`targetSdkVersion`**непосредственно в файле манифеста, так как они будут перезаписаны Gradle в процессе сборки. Дополнительные сведения см. в разделе Указание требований к уровню API .

Объявите функцию камеры непосредственно в файле манифеста вашего приложения:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    ...
</manifest>
```

С объявлениями, показанными в этих примерах, устройства *без* камеры или с версией Android *ниже* 8.0 не могут установить ваше приложение из Google Play. Однако вы можете объявить, что ваше приложение использует камеру, но не *требует* ее. В этом случае ваше приложение должно установить для required атрибута значение falsei во время выполнения проверить, есть ли на устройстве камера, и при необходимости отключить все функции камеры.

Дополнительные сведения о том, как вы можете управлять совместимостью вашего приложения с различными устройствами, приведены в документе « Совместимость устройств» .

Ресурсы приложения

Приложение для Android состоит не только из кода — ему требуются ресурсы, которые отделены от исходного кода, например изображения, аудиофайлы и все, что связано с визуальным представлением приложения. Например, вы можете определить анимацию, меню, стили, цвета и компоновку пользовательских интерфейсов активности с помощью XML-файлов. Использование ресурсов приложения упрощает обновление различных характеристик вашего приложения без изменения кода. Предоставление наборов альтернативных ресурсов позволяет оптимизировать приложение для различных конфигураций устройств, таких как разные языки и размеры экрана.

Для каждого ресурса, который вы включаете в свой проект Android, инструменты сборки SDK определяют уникальный целочисленный идентификатор, который вы можете использовать для ссылки на ресурс из кода вашего приложения или из других ресурсов, определенных в XML. Например, если ваше приложение содержит файл изображения с именем logo.png(сохраненный в res/drawable/каталоге), инструменты SDK создают идентификатор ресурса с именем R.drawable.logo. Этот идентификатор сопоставляется с конкретным целым числом приложения, которое вы можете использовать для ссылки на изображение и вставки его в свой пользовательский интерфейс.

Одним из наиболее важных аспектов предоставления ресурсов отдельно от исходного кода является возможность предоставления альтернативных ресурсов для различных конфигураций устройств. Например, определяя строки пользовательского интерфейса в XML, вы можете переводить строки на другие языки и сохранять эти строки в отдельных файлах. Затем Android применяет соответствующие языковые строки к вашему пользовательскому интерфейсу на основе *квалификатора* языка , который вы добавляете к имени каталога ресурсов (например, res/values-fr/для французских строковых значений) и параметрах языка пользователя.

Android поддерживает множество различных *квалификаторовд*ля ваших альтернативных ресурсов. Квалификатор — это короткая строка, которую вы включаете в имя ваших каталогов

ресурсов, чтобы определить конфигурацию устройства, для которой должны использоваться эти ресурсы. Например, вы должны создавать разные макеты для своих занятий в зависимости от ориентации и размера экрана устройства. Когда экран устройства имеет портретную ориентацию (высокий), вам может понадобиться, чтобы макет с кнопками был вертикальным, но когда экран имеет альбомную ориентацию (широкий), кнопки могут быть выровнены по горизонтали. Чтобы изменить макет в зависимости от ориентации, вы можете определить два разных макета и применить соответствующий квалификатор к имени каталога каждого макета. Затем система автоматически применяет соответствующий макет в зависимости от текущей ориентации устройства.

Дополнительные сведения о различных типах ресурсов, которые вы можете включить в свое приложение, и о том, как создавать альтернативные ресурсы для различных конфигураций устройств, см. в разделе Предоставление ресурсов . Чтобы узнать больше о передовых методах и разработке надежных приложений производственного качества, см. Руководство по архитектуре приложений .

Windows Phone

Если вы хотите создать новое приложение для Windows 11 или Windows 10, сначала нужно выбрать его тип. С помощью средств разработки Windows и .NET в Visual Studio можно создавать приложения нескольких разных типов, каждому из которых соответствуют собственные типы проектов Visual Studio и у каждого из которых есть свои преимущества. Каждый тип приложения включает модель приложения, определяющую жизненный цикл приложения, платформу пользовательского интерфейса по умолчанию и доступ к комплексным API для использования компонентов Windows.

Создание приложения WinUI 3

Библиотека пользовательского интерфейса Windows (WinUI) 3 — это новейшая и рекомендуемая платформа пользовательского интерфейса для классических приложений Windows, включая управляемые приложения, которые используют C# и .NET, и собственные приложения, которые используют C++ с API Win32. Включая систему Fluent Design во все интерфейсы, элементы управления и стили, WinUI предоставляет единообразные, интуитивно понятные и доступные возможности на основе новейших шаблонов пользовательского интерфейса.

WinUI 3 предоставляется в **пакете SDK для приложений Windows** . Пакет SDK для приложений для Windows включает единый набор API и инструментов, которые могут согласованно использоваться любым приложением Win32 на C++ или .NET на C# в различных целевых версиях ОС Windows.

Варианты кросс-платформенной поддержки

WinUI также служит основой для кросс-платформенных технологий, обеспечивающих великолепные возможности Windows с использованием разных языков программирования. Эти платформы используют возможности WinUI в Windows, а также разрешают выполнение в других операционных системах.

.NET MAUI для Windows станет отличным выбором, если:

- Вы хотите сохранить как можно больший объем кода .NET в мобильных и классических приложениях.

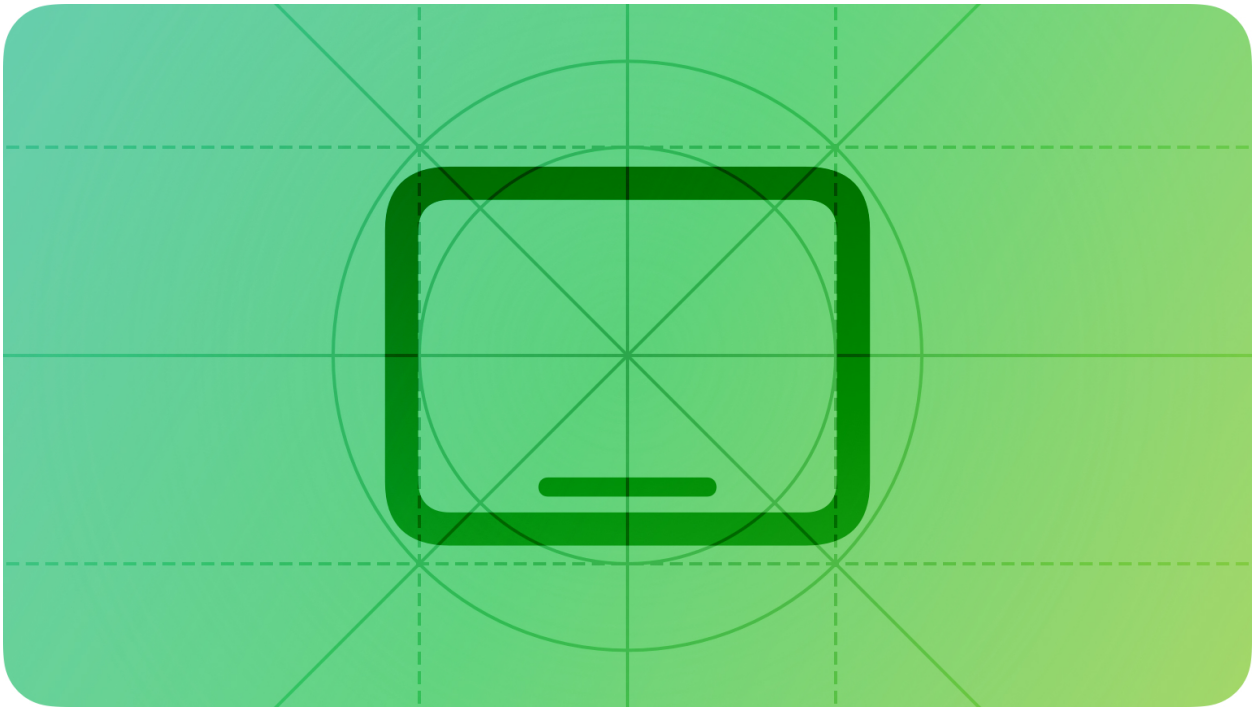
- Вы хотите поставлять приложения на другие компьютерные и мобильные платформы, помимо Windows, с поддержкой собственных возможностей платформы.
- Вы хотите использовать C# и (или) XAML для создания кросс-платформенных приложений.
- Вы используете Blazor для веб-разработки и хотите включить все или одну часть этого решения в мобильное или классическое приложение.

Вот несколько основных особенностей **интерфейса API Win32 и C++** предложений, позволяющих создавать высокопроизводительные приложения.

- Оптимизация на уровне оборудования, включающая тщательный контроль за выделением ресурсов, время существования объектов, макет данных, выравнивание, байтовую упаковку и многое другое.
- Доступ к наборам инструкций, ориентированных на производительность, например SSE и AVX, с помощью встроенных функций.
- Эффективное и строго типизированное универсальное программирование с помощью шаблонов.
- Эффективные и надежные контейнеры и алгоритмы.
- DirectX, в частности Direct3D и DirectCompute (обратите внимание, что UWP также предлагает межпрограммное взаимодействие с DirectX).
- Используйте C++/WinRT для создания современных классических приложений Win32 с эффективным доступом к интерфейсам API среды выполнения Windows (WinRT).

IOS

Люди ценят мощность, мобильность и гибкость iPad, поскольку они наслаждаются мультимедиа, играют в игры, выполняют сложные задачи по повышению производительности и воплощают свои творения в жизнь.



Приступая к разработке приложения или игры для iPad, начните с понимания следующих основных характеристик устройства и шаблонов, отличающих iPadOS. Использование этих характеристик и шаблонов для информирования ваших дизайнерских решений может помочь вам создать приложение или игру, которые оценят пользователи iPad.

Отображать. iPad имеет большой дисплей с высоким разрешением.

Эргономика. Люди часто держат свой iPad во время его использования, но они также могут положить его на поверхность или поставить на подставку. Размещение устройства по-разному может изменить расстояние просмотра, хотя люди обычно находятся в пределах 3 футов от устройства, когда они взаимодействуют с ним.

Входы. Люди могут взаимодействовать с iPad, используя жесты Multi-Touch и экранную клавиатуру , подключенную клавиатуру или указывающее устройство , Apple Pencil или голос , и они часто сочетают несколько режимов ввода.

Взаимодействие приложений. Иногда люди выполняют несколько быстрых действий на своем iPad. В другое время они проводят часы, погруженные в игры, мультимедиа, создание контента или задачи по повышению производительности. Люди часто имеют несколько приложений, открытых одновременно, и им нравится просматривать более одного приложения на экране одновременно и использовать возможности между приложениями, такие как перетаскивание.

Особенности системы. iPadOS предоставляет несколько функций, которые помогают людям взаимодействовать с системой и их приложениями привычными и последовательными способами.

Лучшие практики

Отличные возможности iPad объединяют возможности платформы и устройства, которые люди ценят больше всего. Чтобы вы чувствовали себя в iPadOS как дома, расставьте приоритеты для следующих способов включения этих функций и возможностей.

- Воспользуйтесь преимуществами большого дисплея, чтобы поднять интересующий людей контент на новый уровень, минимизировав модальные интерфейсы и полноэкранные переходы, а также расположив элементы управления на экране там, где они легко доступны, но не мешают.
- Используйте расстояние просмотра и режим ввода, чтобы определить размер и плотность отображаемого на экране содержимого.
- Позвольте людям использовать жесты Multi-Touch, физическую клавиатуру или трекпад или Apple Pencil, а также рассмотрите возможность включения уникальных взаимодействий, сочетающих несколько режимов ввода.
- Плавно адаптируйтесь к изменениям внешнего вида, таким как ориентация устройства, режимы многозадачности, темный режим и динамический тип, и без особых усилий переходите к работе в macOS, позволяя людям выбирать наиболее подходящие для них конфигурации.

Касаемо тестирования приложения(нефункциональное)

Тестирование установки (Installation testing)

Тестирование удобства пользования (Usability Testing)

Тестирование на отказ и восстановление (Failover and Recovery Testing)

Конфигурационное тестирование (Configuration Testing)

Нагрузочное тестирование (Performance and Load Testing)— подвид тестирования производительности, сбор показателей и определение производительности и времени отклика

программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе (устройству).

Тестируемое приложение Gloopt – сервис по созданию канала с возможностью делится своими видеороликами длительностью до 1 минуты. Трафик мы записали через Fiddler, так как jMeter не позволил нам воспроизвести видеоролик и загрузить клип на сервер. Для конвертирования запросов в скрипт jMeter'а мы использовали плагин авторства нашего коллеги, Дениса Скорнякова (https://git.performance-lab.ru/d.skornyakov/FiddlerJmxCreator) – он сохраняет сессию в .jmx файл. Предварительно прямо из Fiddler можно коррелировать значения.

Если создавать тест-план с помощью плагина, а не вручную, сессию Fiddler лучше сохранить, потому что не все заголовки могут быть перенесены корректно.

Обмен сообщениями между клиентским приложением и сервером выглядит примерно так:

1. Сначала запрашивается информация о файле, из респонза мы узнаем его размер в байтах.

2. Далее приложение посылает запрос на некий отрезок файла, например, от нуля.

Выглядит как "Content-range: bytes был = 0-", сервер возвращает уже конкретный отрезок, " Content-range: bytes был = 0-37895".

Следующий запрос клиента запрашивает часть от 37896 и так далее, пока не дойдем до конца.

В Fiddler, кстати, реквесты со стороны клиента уходили с определенным диапазоном. Поначалу было непонятно, каким образом приложение формирует запрос. Зато в деббагере Chrome можно было увидеть, что запрашиваемый диапазон зависит от респонза сервера.

Соответственно, получая длину контента, делим его на n частей и через While симулируем просмотр видео.

Загрузка видео в нашем приложении может быть в двух вариантах – записать ролик сразу с камеры либо загрузить уже готовый видеоролик. В первом случае он все равно сначала сохраняется в файловую систему, поэтому для нас оба процесса с точки зрения нагрузки идентичные.

Эту транзакцию не удалось выполнить из jMeter после конвертации – то ли Fiddler специфично воспринимает подобную операцию, то ли в jMeter приходится выкручиваться по-особому. В итоге пришлось именно эту часть записать помощью jMeter, чтобы были правильно проставлены параметры сэмплера.

В частности, для загрузки видео надо было удалить Content-Type из header'a, прописать MIME Type у отправляемого файла, поставить галочки у "Use multipart/form data for POST" и "Browser-compatible headers", что не было сделано при конвертации запроса из Fiddler (там сам запрос построен немного по-другому).

Стресс-тести́рование (англ. Stress Testing) — один из видов тестирования программного обеспечения, которое оценивает надёжность и устойчивость системы в условиях превышения пределов нормального функционирования. Стресс-тестирование особенно необходимо для «критически важного» ПО, однако также используется и для остального ПО. Обычно стресс-

тестирование лучше обнаруживает устойчивость, доступность и обработку исключений системой под большой нагрузкой, чем то, что считается корректным поведением в нормальных условиях.

Тестирование внешним скриптом

Суть в следующем:

- Все решения и генераторы помещаются в отдельные файлы — которые теперь не обязательно исполняются в одной среде.
- Передача тестов происходит через перенаправление потоков ввода-вывода. Ввод в программах считывается так, как бы он считывался естественно в тестирующей системе.
- Запускается внешний скрипт, который n*n* раз запускает генератор, записывает его вывод в файл, а затем кормит этот файл двум решениям и построчно сравнивает выводы.

Файлы `stupid.cpp`, `smart.cpp` и `gen.py` содержат уже понятный нам код. Вот примерный код скрипта `checker.py`:

```
import os, sys

_, f1, f2, gen, iters = sys.argv
# первый аргумент это название программы, "checker.py",
# поэтому "забудем" его с помощью "_"

for i in range(int(iters)):
    print('Test', i + 1)
    os.system(f'python3 {gen} > test.txt')
    v1 = os.popen(f'./{f1} < test.txt').read()
    v2 = os.popen(f'./{f2} < test.txt').read()
    if v1 != v2:
        print("Failed test:")
        print(open("test.txt").read())
        print(f'Output of {f1}:')
        print(v1)
        print(f'Output of {f2}:')
        print(v2)
        break
```

Автор обычно запускает его командой `python3 checker.py stupid smart gen.py 100`, предварительно скомпилировав `stupid` и `smart` в ту же директорию, что и сам `checker.py`. При желании можно также прописать компиляцию прямо внутри скрипта.

Скрипт написан под Linux / Mac. Для Windows нужно убрать `./` во внешних командах и вместо `python3` писать `python`.

Не забывайте, что если хотя бы одна из программ не выводит перевод строки в конце файла, то чекер посчитает, что вывод разный.

Тестирование стабильности или надежности (Stability / Reliability Testing) — один из видов нефункционального тестирования ПО, целью которого является проверка работоспособности приложения при длительном тестировании с ожидаемым уровнем нагрузки.

Тестирование прерываний — это раздел Тестирование мобильных приложений, в котором рассматривается, как приложение реагирует на прерывание и возвращается к своему предыдущему состоянию.

Ожидаемое поведение в случае этих прерываний является одним из следующих:

1. **Запуск в фоновом режиме:** прерывание вступает во владение, пока приложение отходит на задний план. Он получает контроль после окончания прерывания. Например, телефонный звонок / Facetime, который вы посещаете, когда читаете цифровую книгу в iBooks (или аналогичном приложении). Когда пользователь отвечает на телефон, iBooks ждет, пока это не будет сделано, и затем возобновляет работу, когда звонок заканчивается.
2. **Показать оповещение. Оповещение исчезает, и вы работаете как обычно. «** SMS получены» — сообщения появляются в заголовке. Пользователь не беспокоится об этом и продолжает работать с приложением в обычном режиме. Другие оповещения мобильного приложения, такие как запрос нового друга в Facebook или сообщение WhatsApp, также попадают в эту категорию. Но если пользователь решает прочитать сообщение, то следует поведение, описанное в пункте 1. Если игнорируется, состояние приложения не изменяется.
3. **Призыв к действию** : Сигналы тревоги должны быть выключены или отодвинуты перед продолжением работы То же самое с сообщениями об обновлении приложения. Вы должны отменить или принять изменения, прежде чем продолжить. Другим примером является предупреждение о низком заряде батареи. Вы можете продолжить обычную работу или перейти в режим низкого энергопотребления (если устройство это позволяет).
4. **Не влияет:** Пример: если сетевое соединение становится доступным и ваше устройство подключается к нему. Кроме того, когда вы подключаете свое устройство для зарядки, нет необходимости в предупреждении или призыве к действию. Вероятно, он выполнит свою работу, пока вы продолжаете использовать свое приложение.

Тестирование безопасности – это метод тестирования, позволяющий определить, защищает ли информационная система данные и поддерживает ли функциональность, как предполагалось. Тестирование безопасности не гарантирует полную безопасность системы, но важно включить тестирование безопасности как часть процесса тестирования.

пример

Обнаружение недостатка безопасности в веб-приложении требует сложных шагов и творческого мышления. Иногда простой тест может подвергнуть самую серьезную угрозу безопасности. Вы можете попробовать этот самый базовый тест в любом веб-приложении –

- Войдите в веб-приложение, используя действительные учетные данные.
- Выйдите из веб-приложения.
- Нажмите кнопку НАЗАД браузера.
- Убедитесь, что вас попросили снова войти в систему или вы можете вернуться на страницу входа снова.

Войдите в веб-приложение, используя действительные учетные данные.

Выйдите из веб-приложения.

Нажмите кнопку НАЗАД браузера.

Убедитесь, что вас попросили снова войти в систему или вы можете вернуться на страницу входа снова.

Тестирование установки

Тестирование установки подтверждает, что процесс установки проходит без каких-либо проблем.

Пример тестовых сценариев –

- Убедитесь, что процесс установки проходит гладко и не займет много времени.
- Убедитесь, что установка прошла успешно через корпоративный магазин приложений.

Тестирование локализации

В настоящее время большинство приложений предназначено для глобального использования, и очень важно заботиться о региональных следах, таких как языки, часовые пояса и т. Д. Важно проверить функциональность приложения, когда кто-то меняет часовой пояс. Необходимо учитывать, что иногда западные дизайны могут не работать с аудиторией из восточных стран или наоборот.

Пример тестовых сценариев –

- Убедитесь в отсутствии проблем с пользовательским интерфейсом или усечением данных, когда мы используем мобильное приложение на разных языках (или, скажем, не на английском языке).
- Убедитесь, что изменения часового пояса корректно обрабатываются для вашего мобильного приложения.

Убедитесь в отсутствии проблем с пользовательским интерфейсом или усечением данных, когда мы используем мобильное приложение на разных языках (или, скажем, не на английском языке).

Убедитесь, что изменения часового пояса корректно обрабатываются для вашего мобильного приложения.

Тестирование совместимости

Тестирование совместимости имеет самый высокий стек, когда дело доходит до тестирования мобильных приложений. Цель теста на совместимость мобильного приложения, как правило, состоит в том, чтобы ключевые функции приложения работали должным образом на конкретном устройстве. Сама совместимость должна занимать всего несколько минут и может быть спланирована заранее.

Это не будет легкой задачей, решить, какие тесты на совместимость мобильных устройств следует выполнить (поскольку тестирование со всеми доступными устройствами просто невозможно). Поэтому подготовьте тестовую матрицу с каждой возможной комбинацией и расставьте приоритеты для клиента.

Пример тестовых сценариев –

- Убедитесь, что поиск авиабилетов успешно выполняется на устройстве Android.
- Убедитесь, что поиск авиабилетов успешно выполнен для Apple iPad.

Тестирование производительности

Мобильный тест производительности охватывает производительность клиентских приложений, производительность сервера и производительность сети. Важно убедиться, что сценарии тестирования производительности охватывают все эти области. С помощью инструментов тестирования производительности нетрудно идентифицировать существующие сети, серверы и узкие места серверных приложений, учитывая предопределенную нагрузку и сочетание транзакций.

Пример тестовых сценариев –

- Убедитесь, что проверка доступного рейса занимает только разумное количество времени.
- Убедитесь, что во время проверки доступности рейса мобильный телефон работает нормально и не зависает.

Убедитесь, что проверка доступного рейса занимает только разумное количество времени.

Убедитесь, что во время проверки доступности рейса мобильный телефон работает нормально и не зависает.

Тестирование конфигурации

Тестирование конфигурации — это подход к тестированию программного обеспечения, при котором программное приложение тестируется с различными комбинациями программного и аппаратного обеспечения для анализа функциональных требований и определения наилучших конфигураций, в которых программное приложение работает без ошибок или недостатков.

Тестирование конфигурации, как указывалось ранее, представляет собой тип тестирования программного обеспечения, при котором тестируемое приложение должно оцениваться с использованием различных комбинаций программного и аппаратного обеспечения.

Пример тестирования конфигурации

Давайте посмотрим на пример настольного приложения, чтобы увидеть, как это работает —

Как правило, настольные приложения бывают двухуровневыми или трехуровневыми. Здесь мы рассмотрим трехуровневое настольное приложение, созданное с использованием Asp.Net и состоящее из клиента, сервера бизнес-логики и сервера базы данных, каждый из которых поддерживает перечисленные ниже платформы.

Windows XP, Windows 7, Windows 8 и другие клиентские платформы

Windows Server 2008 R2, Windows Server 2008 R2 и Windows Server 2012R2 являются серверными платформами.

SQL Server 2008, SQL Server 2008R2, SQL Server 2012 и другие базы данных

Чтобы убедиться, что приложение работает правильно и не дает сбоев, тестировщик должен протестировать комбинацию клиента, сервера и базы данных, используя комбинации вышеупомянутых платформ и версий базы данных.

Тестирование конфигурации не ограничивается программным обеспечением, но также применяется к оборудованию, поэтому оно также известно как тестирование конфигурации оборудования, при котором мы тестируем различные аппаратные устройства, такие как принтеры, сканеры, веб-камеры и другие устройства, которые позволяют тестируемому приложению.

Показательный пример

Рассмотрим случай, когда ваша компания создала настольное приложение на C#, основанное на платформе .NET.

И это приложение построено на трехуровневой архитектуре с внешним интерфейсом (клиент), уровнем приложения (сервер) и уровнем базы данных. Соответственно, каждый слой будет поддерживать определенные платформы.

Предположим, что каждый уровень поддерживает платформы, перечисленные ниже —

Клиент — Microsoft Windows 10, Windows 7, Windows XP и Linux.

Ubuntu Server, Windows Server 2016 и Novell Open Enterprise Server являются примерами серверов.

Microsoft SQL Server, IBM DB2 и MySQL являются примерами баз данных.

Теперь, как тестировщик программного обеспечения, вы должны протестировать программу на каждой из многочисленных комбинаций вышеупомянутых клиентских, серверных и баз данных платформ, чтобы гарантировать, что она работает должным образом на всех из них.

Например, вы проверите, как приложение работает с ОС Windows 10, базой данных Windows Server 2016 и базой данных MySQL, а затем вы проверите, как программа работает с ОС Windows 10, базой данных Windows Server 2016 и базой данных IBM DB2.

И так далее, пока не перепробуете все мыслимые комбинации. Наше тестирование не будет ограничиваться программным обеспечением; он также будет включать аппаратное обеспечение, в котором нам придется исследовать каждую из многочисленных комбинаций аппаратных устройств. В результате это тестирование также называется тестированием конфигурации оборудования.

Чтобы провести **комплексное тестирование интернет-соединения**, следует проверить:

1. текущее соединение с интернетом, на котором мы будем работать;
2. конфигурации соединения с интернетом тестируемого приложения;
3. устойчивость приложения к неполадкам в сети;
4. устойчивость приложения к перегрузкам (нагрузочное тестирование).

Проверка соединения с интернетом

Перед началом работы стоит протестировать текущее соединение, чтобы определить, будут ли тесты в данной среде валидными. Тестирование интернет-соединения среды включает в себя **проверку скорости** отправки и получения пакетов, а также **качество подключения** с помощью вычисления колебания в серии последовательных измерений пинга при посылке запросов на один и тот же адрес.

Последнее является одним из базовых умений пользователя консоли. Команда, позволяющая это делать, стоит в начале списка команд, которые узнает новичок. Речь идет, конечно, о команде ping. Она идентична для Windows и Linux.

Команду следует вписать в консоли, а после нее IP-адрес или http-адрес страницы, которую мы хотим «пинговать». Команда посылает несколько тестовых пакетов по протоколу TCP/IP на указанный сервер и измеряет время ответа (если таковой последует). Таким образом, мы проверяем работоспособность сети, а также качество соединения с конкретным IP-адресом, указанным после команды, ведь может так случиться, что соединение с интернетом у нас рабочее, а на вызываемом сервере как раз проблемы, из-за чего машины не могут соединиться друг с другом. Перед началом тестирования полезно «пропинговать» сервер нашего приложения, чтобы убедиться, что соединение надежное и среда настроена правильно.

На скриншоте видно, как мы даем в консоли команду пропинговать страницу google.com. Программа сообщает нам, что совершила попытку отправить по указанному адресу пакет с 32 байтами данных (небольшой тестовый пакет, просто чтобы получить ответ). Утилита делает это 4 раза и высвечивает результаты попыток на экране. Далее приводится отчет об операциях со средними показателями. Чтобы получить ответ от сайта программе понадобилось в среднем около 13 миллисекунд. Неплохое время! Наш интернет работает быстро и стабильно (ни одного пакета не потеряно).

Проверка пропускной способности канала связи (насколько быстро посылаются/получаются пакеты) осуществляется с помощью специальных программ и утилит, которые легко можно найти в интернете. Например, Test TCP (TTCP) от Cisco. Также можно проверить все три аспекта с помощью онлайн-утилит (например, https://www.nperf.com/ru/ или https://www.speedtest.pl/ru/).

Тестирование интернет-конфигураций приложения

Когда мы убедимся, что среда работает стабильно и правильно, наступает очередь тестирования конкретного приложения. Начать стоит с конфигураций интернет-соединения в данном приложении, если таковые имеются. Чаще всего мы можем столкнуться с такой необходимость в случае десктопных приложений.

В подавляющем большинстве случаев параметры соединения с сетью находятся в специальном конфигурационном текстовом файле, к которому нет доступа “простым” пользователям. Какие параметры необходимо сообщить приложению и в каком формате они должны сообщаться, подскажут разработчики. Задача тестировщика попробовать разные комбинации таких параметров, а также неправильные параметры. Последнее нужно, чтобы проверить, как система будет реагировать на неправильные параметры и какую информацию об ошибке предоставит пользователю.

Например, рассмотрим конфигурационный файл приложения ERP. Приложение должно знать, к какой базе данных подключаться, какой сервер использовать, с каким профилем работать и какие плагины использовать.

Всю эту информацию мы предоставляем ей в конфигурационном файле. Чтобы удобнее было оперировать данными, можно разделить их на секции (названия в квадратных скобках. Вариантов тестирования конфигураций множество: базы на локальном/удаленном сервере, несуществующие пути к БД и серверам, правильные/неправильные логины и пароли, разные комбинации плагинов и так далее.

Проверка устойчивости приложения к перепадам в сети

После теста конфигураций стоит проверить “устойчивость” приложения к неполадкам в сети и перебоям связи. Например, хорошим тестом будет прерывание интернет-соединения во время выполнения каких-либо задач: загрузки данных, обновлений, отправления данных на сервер.

Перебои в связи не должны серьезно нарушить работу приложения и точно не должны привести к потере каких-либо данных. Кроме того, пользователь должен быть проинформирован, что произошло, какая задача не была выполнена и почему, а также что может сделать пользователь, чтобы исправить ситуацию.

Стоит ли упоминать, что информация должна быть правдива (точно диагностировать проблему) и содержать действенные инструкции насчет того, как разрешить сложившуюся ситуацию.