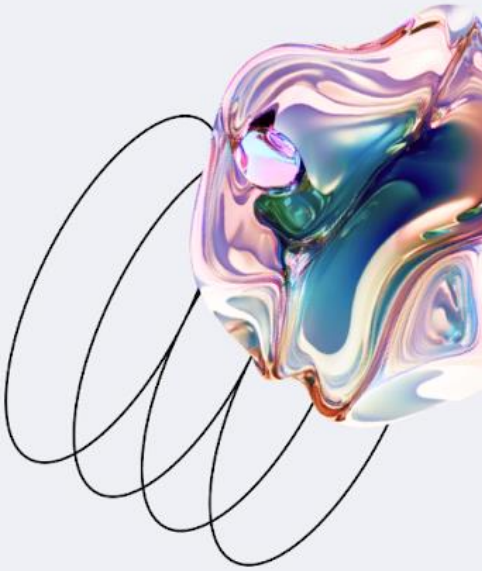
GeekBrains



Проблемы и сложности при тестировании мобильных приложений

Процесс тестирования мобильных приложений

Оглавление

На этом уроке

Итак, мы успели разобрать основные виды тестирования мобильных приложений, рассмотреть инструменты, которые помогут при тестировании, и составить тест-кейсы.

На заключительной лекции мы узнаем, какие типы багов чаще всего встречаются в мобильных приложениях, и разберем сложности при тестировании.

Сегодня на уроке:

- Приложения для iOS и Android: где больше ошибок?
- Распространенные баги при тестировании мобильных приложений.
- Приоритеты багов.
- Сложности при тестировании мобильных приложений.
- Регресс-тесты.
- Пострелизная поддержка приложения.

Термины, используемые в лекции

Неотвечающие уведомления — уведомления, при тапе на которые приложение открывается слишком долго либо вообще не открывается.

Блокирующие баги — баги, которые блокируют или препятствуют выполнению другой работы, пока не будут исправлены.

Критические баги — баги, которые серьезно влияют на приложение и его функциональность. Могут привести к сбою приложения, потере данных или раскрытию конфиденциальной информации.

Баги с высоким приоритетом — баги, которые оказывают значительное влияние на взаимодействие с пользователем, но не так серьезны, как критические ошибки.

Баги со средним приоритетом — баги, которые оказывают умеренное влияние на функциональность приложения и взаимодействие с пользователем.

Баги с низким приоритетом — баги, которые незначительно влияют на функциональность или удобство приложения.

Приложения для iOS и Android: где больше ошибок?

Как вы думаете, в каких приложениях обычно больше багов — на iOS или Android?

Как правило, на Android, хотя бывают и исключения. На это есть несколько причин:

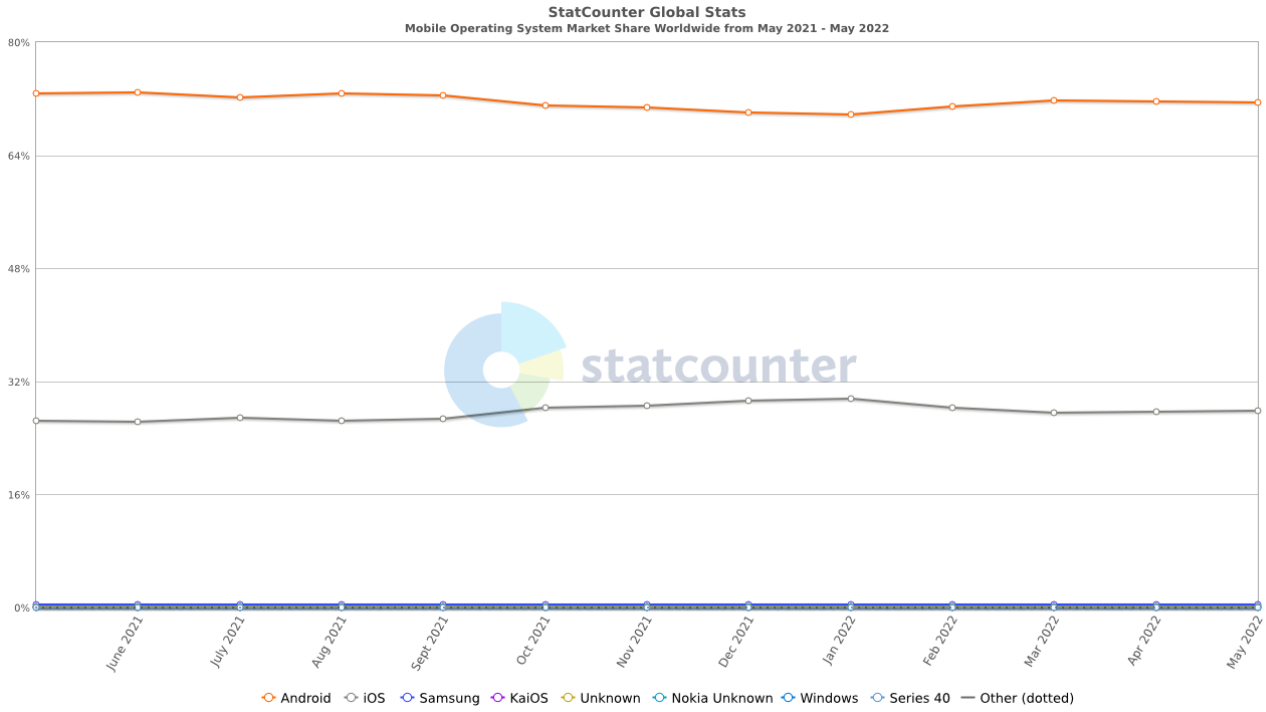
- **Фрагментация**. В Android более фрагментированная экосистема с широким спектром производителей устройств и версий операционной системы. Разработчикам бывает сложно обеспечить безупречную работу приложения на всех устройствах, что может привести к ошибкам.

  Множество ОС Android вызывают множество проблем с совместимостью. Кроме того, пользователи Android редко обновляют свое ПО (отчасти из-за того, что сами устройства Android поддерживаются не так долго). Около 50% пользователей используют версии двухлетней давности — это серьезная проблема безопасности и основная причина низкой производительности приложений для Android.

- **Открытый код.** Android — это платформа с открытым исходным кодом. Это значит, что у разработчиков больше вариантов для изменения и настройки ОС. С одной стороны, это хорошо, с другой — больше вероятностей для появления ошибок.

- **Процесс проверки в магазине приложений.** Проверка в Play Store обычно менее строгая, чем в App Store, поэтому в магазине Google больше некачественных приложений с ошибками.

Как мы видим, самая большая проблема — разнообразие устройств, так что тестирование на Android будет занимать больше времени.



Источник: [Statcounter Global Stats]

Скриншот выше — это глобальная статистика с процентами устройств, которыми пользуются на рынке. Как мы видим, большая часть — Android.

График еще раз доказывает, что тестирование на Android должно проводиться тщательней и на большем количестве устройств.

Распространенные баги при тестировании мобильных приложений

Рассмотрим, где в мобильном приложении могут быть баги.

В логике приложения. Пример — нереализованный функционал, отсутствие необходимых разделов в меню и прочее, что касается функционала приложения.

Зависящие от ОС. Иногда баг воспроизводится только на одной платформе. Например, пуш-уведомления могут корректно работать на одной ОС, но не работать на другой.

Важно учитывать особенности платформ разных ОС. Хороший пример — наличие кнопки «Назад» в меню приложения. В момент реализации приложений дизайнеры или программисты могут не продумать этот момент и пользователи iOS не смогут вернуться назад с любого экрана. Важно учитывать это при тестировании. Знание и понимание таких моментов приходит с опытом работы и с изучением информации об операционных системах.

Зависящие от версии ОС. Например, когда баг воспроизводится только на последней версии iOS. Частый пример — работа с пермишенами приложений. На определенной версии iOS при выдаче разрешения на доступ, например, к камере, приложение может крешить.

Важно отслеживать, что нового в обновлениях ОС, особенно глобальных. Так тестирование на обновленных устройствах будет более тщательным.

Зависящие от устройства. Воспроизводятся на определенном устройстве либо группе устройств.

У устройств есть множество параметров, которые могут повлиять на стабильность работы приложения. Пример бага — неработающая камера при сканировании кода на iPhone 13 Max.

Еще пример — баги, связанные с производителем. Например, в приложении есть доступ к галерее, чтобы вставить картинку в сообщение, но на моделях Samsung он не будет работать.

Итак, мы разобрали виды багов по частоте воспроизведения. Теперь пройдемся по основным багам, которые можем встретить в процессе тестирования мобильных приложений.

Креши и зависания

С одной стороны, найти некоторые креши не так сложно. Например, когда мы нажимаем кнопку, а приложение принудительно закрывается, аварийно завершает работу, неоднократно зависает, перестает отвечать на запросы или работать должным образом.

С другой стороны, приложение может крешить по огромному количеству причин. Причем креши могут быть как связаны с логикой приложения (например, креш при нажатии кнопки «Настройки», который воспроизводится на всех устройствах), так и зависеть от ОС, версии ОС, устройства.

Факторов, влияющих на производительность мобильного приложения, много. Выделим основные из них:

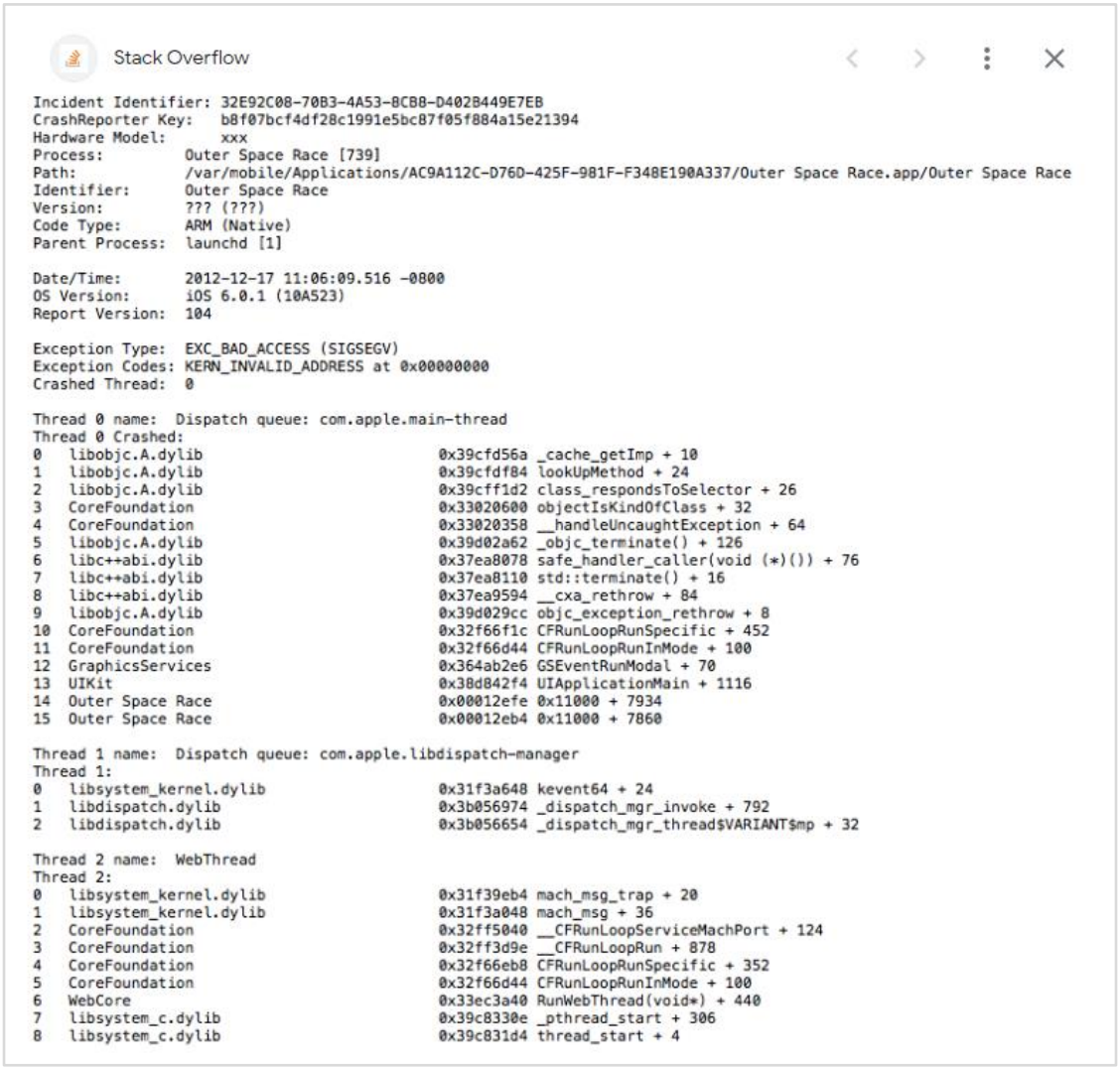
- **Ошибки в логике приложения.** Если в коде приложения есть ошибки, они могут привести к сбою. Например, если приложение пытается получить доступ к памяти, которая уже была освобождена, или пытается выполнить операцию с несуществующим объектом, возможен креш на всех типах устройства.

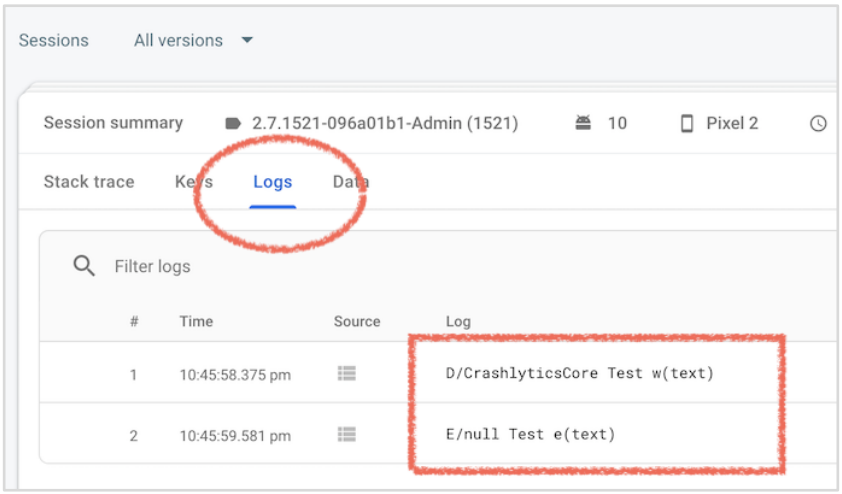
- **Совместимость с устройствами.** Приложения могут работать не на всех устройствах и не на всех версиях операционной системы. Проблемы совместимости могут привести к сбою или неожиданному поведению.
- **Недостаточно памяти.** Если приложению требуется больше памяти, чем доступно на устройстве, может произойти сбой.
- **Проблемы с сетью.** Если приложение зависит от сетевого подключения, а сеть нестабильна или недоступна, это может привести к сбою.
- **Пользовательский ввод.** Если пользователь вводит неверные данные или выполняет действия, которые приложение не поддерживает, это может привести к сбою.
- **Сбои самих устройств.** Иногда аппаратное обеспечение устройства может выйти из строя, что становится причиной сбоя. Например, неисправный модуль памяти может привести к сбою приложения, если оно попытается получить доступ к этой памяти.

Важно понимать, что мы не можем найти 100% крешей в приложении. Всегда есть вероятность, что приложение крешит у пользователей. Можно смотреть статистику крешей от Firebase: программисты проанализируют крешлоги и попросят нас воспроизвести баг.

Обычно это выглядит так: в крешлоге видно, на каком устройстве и в каком модуле приложения происходит сбой. Нам нужно внимательно просмотреть все возможные действия в этом участке приложения на таком же либо очень похожем по характеристикам устройстве.

Рассмотрим, как может выглядеть крешлог. На скриншоте часть кода, которая вызывает креш. Именно ее и анализируют разработчики, чтобы понять проблему:





Чтобы предотвратить сбои приложения, команда разработки может проводить тщательное тестирование и отладку, оптимизировать производительность приложения и использование памяти, а также обеспечивать совместимость с различными устройствами и версиями операционной системы.

UI-баги

UI-баги могут быть общими. Например, экраны приложений не соответствуют макету: неправильное расположение кнопок, неправильные надписи или заголовки, неправильные цвета фонов или кнопок.

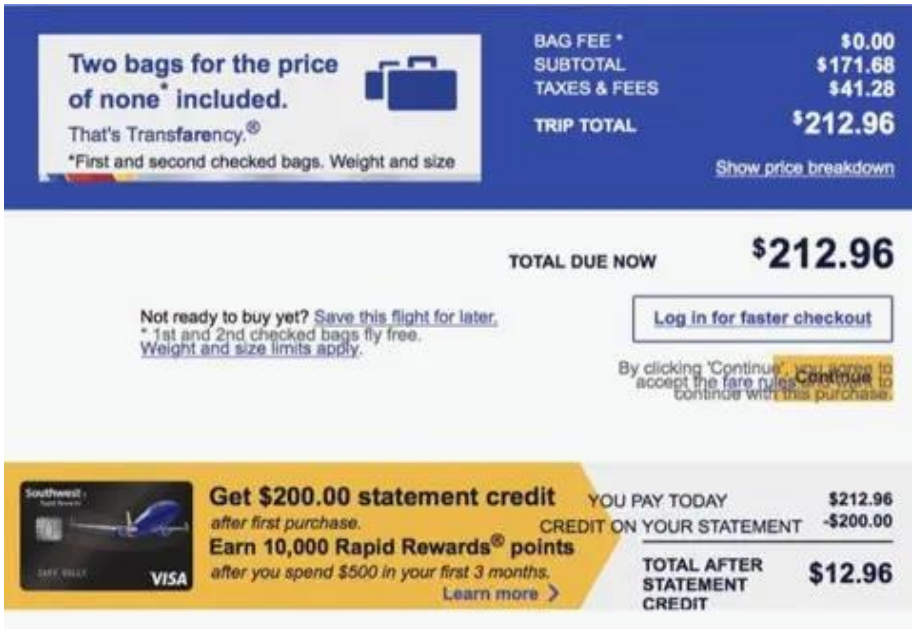
С этим довольно просто. Такие ошибки возникают при неправильной реализации приложения и при проверке наша задача — тщательно сверить все разделы приложения с макетом, если макет есть. Если макета нет, проверяем, что все элементы расположены нормально и не перекрывают друг друга.

Помимо общих проблем с UI, есть проблемы, связанные с тем, как приложение выглядит на разных устройствах.

UI-баги на разных разрешениях/размерах экрана

Самый распространенный сценарий — один элемент UI перекрывает другой: например, иконка, кнопка или цена заказа. Другие распространенные ошибки — обрезанные заголовки и изображения.

На скриншоте пример бага — надпись смещена и закрывает кнопку:



Источник: Lambdatest

Некорректное отображение UI на разных экранах может произойти по нескольким причинам:

- **Нет адаптивного дизайна.** Если приложение сделано без адаптивного дизайна, могут быть проблемы несовместимости при просмотре на экранах с

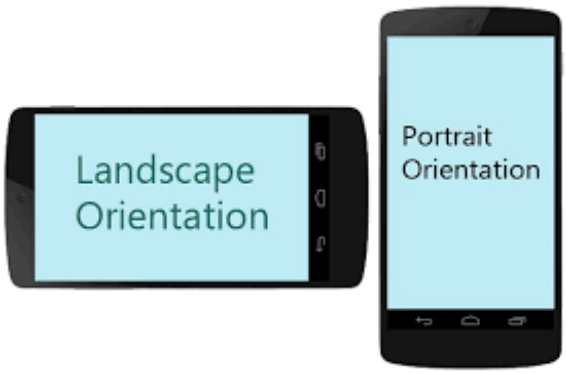
разным разрешением или плотностью. Адаптивный дизайн может настроить макет в зависимости от размера экрана и гарантировать, что он будет хорошо выглядеть на всех устройствах.

- **Разные соотношения сторон экрана:** например, 16:9 и 18:9. Если макет страницы разработан для определенного соотношения, он может плохо выглядеть на экранах с другим (частый случай — один элемент закрывает другой).
- **Различия в плотности пикселей.** У разных устройств разная плотность пикселей, она может влиять на размер и расположение элементов на странице. Например, элемент, который выглядит большим на экране с низкой плотностью, может выглядеть маленьким на экране с высокой плотностью. Из-за этого отображение разных элементов на страницах приложения будет неудобным и некорректным.
- **Неподдерживаемые функции.** Некоторые устройства могут не поддерживать функции, используемые в макете страницы: например, определенные стили CSS или библиотеки JavaScript. Все это может привести к проблемам несовместимости.

Чтобы предотвратить проблемы несовместимости с макетами страниц при разных разрешениях экрана и плотности, важно создавать страницы с адаптивными макетами, которые можно адаптировать к разным размерам экрана, тестировать на различных устройствах и использовать лучшие практики веб-дизайна и разработки.

Проблемы, связанные с портретным и ландшафтным режимом

Часто приложения разрабатываются только под один режим: портретный или ландшафтный (вертикальный и горизонтальный).



Источник: [Detect Screen Orientation in Android using Jetpack Compose]

Наша задача — проверить, что приложение действительно поддерживает только один режим, то есть перевернуть устройство с запущенным приложением.

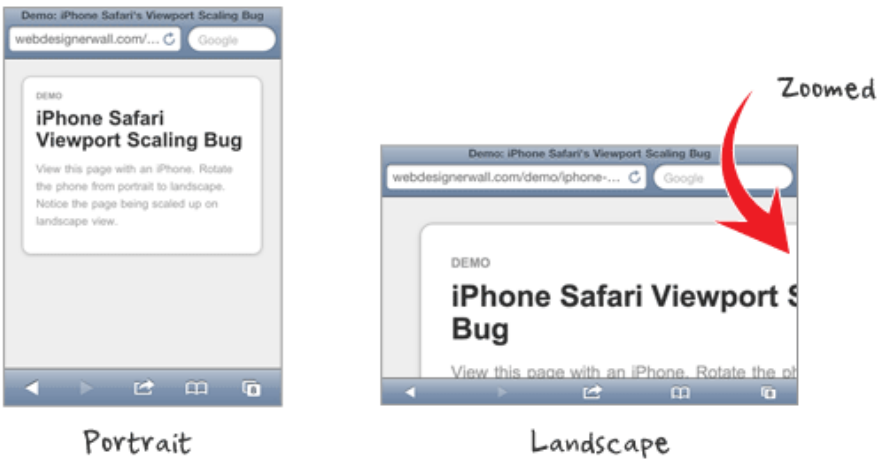
Важно учитывать, что если у приложения есть доступ к другим разделам телефона, оно нормально вернется в изначальное состояние при переключении между разделами.

Например, у приложения, разработанного только для портретного режима, есть доступ к галерее. Есть смысл проверить сценарий: открыть галерею через приложение, перевернуть телефон, выполнить действия в галерее (например, выбрать картинку), вернуться в приложение и посмотреть, что будет в результате.

Если приложение разработано и в портретном, и в ландшафтном режиме, нужно проверить, как оно поведет себя в обоих, потому что переключение между режимами часто приводит к UI-багам: например, к обрезанному интерфейсу или наложению элементов.

Другой пример: в системе Android определенный элемент экрана — Activity — может быть сломан поворотом экрана. В этом случае он потеряет свое текущее состояние. После поворота все выбранные флажки или раскрывающиеся списки могут быть сброшены.

На скриншоте пример бага со сменой ориентации экрана. В портретном режиме все отображается корректно, а при переключении в ландшафтный часть информации обрезается:



Источник: [iPhone Safari Viewport Scaling Bug]

Push-уведомления

Примеры багов в push-уведомлениях мобильных приложений:

- **Дублирующиеся уведомления.** Пользователи могут получать одно и то же уведомление несколько раз.
- **Отложенные уведомления.** Уведомления могут задержаться или вообще не прийти.
- **Неверный контент.** В уведомлении может быть неверный текст или картинки, из-за чего пользователи получат неправильную информацию. Важно проверять уведомления на соответствие макетам.
- **Непредусмотренные получатели.** Уведомления могут быть отправлены не тем пользователям. Это приведет к путанице и потенциальному нарушению конфиденциальности пользователей.
- **Неотвечающие уведомления.** Это уведомления, при тапе на которые приложение открывается слишком долго либо вообще не открывается.

Проблемы с кнопками

Одна из самых распространенных ошибок, связанных с кнопками, — они не реагируют на нажатие. Причины могут быть разными: например, есть проблемы с сетевым подключением или сбой программного обеспечения.

Иногда проблема с неработающими кнопками появляется, если в приложении нет обработки неправильных действий. Например, если в поле ввода пароля мы укажем неправильный пароль, приложение просто не отреагирует при нажатии кнопки «Подтвердить».

Еще одна частая проблема с кнопками — возможность повторного нажатия. Это довольно важно, особенно в приложениях, где совершаются платежи, отправляются сообщения или информация на сервер. Если пользователь по ошибке дважды нажал кнопку «Оплатить», деньги с него не должны списываться дважды.

Есть несколько причин таких проблем:

- **Медленное время отклика.** Если у приложения медленное время отклика, пользователи могут раздраженно нажимать кнопку несколько раз. Это может стать причиной непреднамеренных действий и сбоев.
- **Проблемы с сетевым подключением.** Если приложение использует сетевое подключение, плохое или нестабильное соединение может вызвать задержки во времени отклика, что приведет к повторному нажатию кнопки.

- **Пользовательский интерфейс не отвечает или зависает.** Пользователи могут нажимать кнопку несколько раз, думая, что приложение не регистрирует на их ввод.

Команда разработки должна работать над оптимизацией приложений, чтобы таких проблем было как можно меньше.

Источник: [UI Design: Basic Types of Buttons in User Interfaces]

Индикатор прогресса

Индикаторы должны быть прописаны в требованиях. Но мы, как тестировщики, всегда должны обращать внимание на то, реализованы ли они и, если нет, уточнять требованиях насчет них.

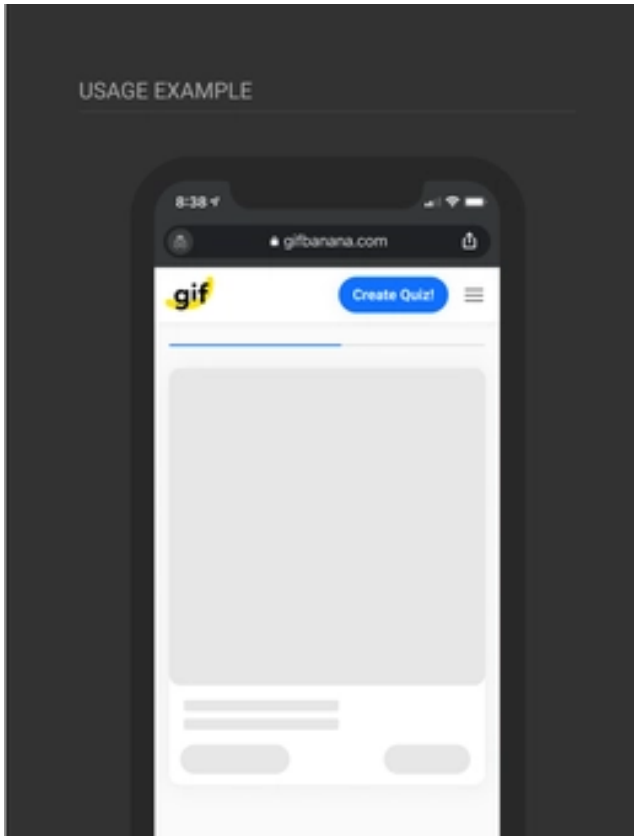
В приложении должен быть индикатор прогресса или загрузки (спиннер), чтобы уведомить пользователя о том, что действие выполняется. Есть множество обстоятельств, когда выполнение команды в приложении задерживается надолго. В таких случаях нужно показать индикатор выполнения, чтобы уведомить пользователя о том, что команда обрабатывается.

Примеры сценариев: загрузка экрана/контента и длительные сетевые действия при нажатии кнопок (например, редактирование профиля, загрузка изображений и так далее). Пользователь хочет обновить изображение своего профиля и нажимает кнопку «Загрузить».

Спиннер:

Источник: Loading Spinner

Прогресс-бар:

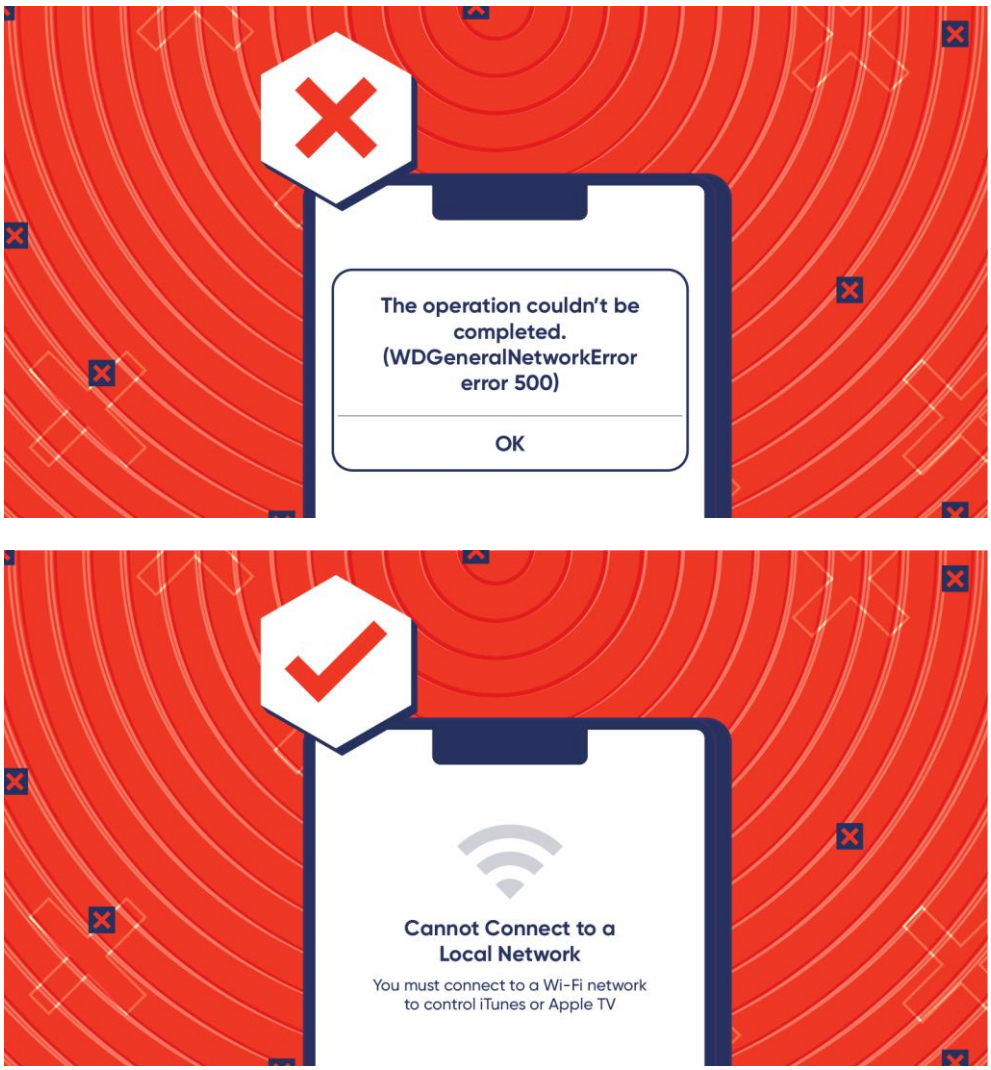


Источник: Progress Bar Loading

Непонятные сообщения об ошибках

Еще один раздражающий пользователь тип багов — непонятные сообщения об ошибках. Иногда приложение должно выдать ошибку, но пользователю непонятно, что это за ошибка. Таких случаев надо избегать.

Пример непонятного и понятного для пользователя сообщения:



Источник: [The Do's and Don'ts of Mobile Error Messages]

> 💡 Список багов в мобильных приложениях на этом не ограничен. Но мы рассмотрели самые распространенные из них.

Приоритеты багов

Баги в мобильных приложениях можно классифицировать по приоритетам:

Приоритеты определяют исходя из важности ошибки или проблемы. Чем сильнее баг влияет на функциональность, взаимодействие с пользователем, безопасность и другие аспекты, тем выше приоритет, тем быстрее его нужно исправить.

Блокирующие баги

Самый критичный вид багов — они блокируют или препятствуют выполнению другой работы, пока не будут исправлены.

Пример — ошибки или проблемы, которые препятствуют тестированию, сборке или развертыванию приложения. Блокирующая ошибка на экране входа в мобильное приложение может помешать нам тестировать другие функции — мы просто не сможем войти в приложение и получить к ним доступ.

Критические баги

Баги, которые серьезно влияют на приложение и его функциональность. Могут привести к сбою приложения, потере данных или раскрытию конфиденциальной информации.

Пример — креш при открытии настроек профиля.

С высоким приоритетом

Баги, значительно влияющие на взаимодействие с пользователем, но не такие серьезные, как критические. Могут привести к зависанию приложения или повлиять на удобство использования некоторых функций.

Пример — в приложении социальной сети баг может помешать пользователям загружать фотографии или видео.

Со средним приоритетом

Баги, которые умеренно влияют на функциональность приложения и взаимодействие с пользователем. Могут влиять на менее важные функции или вызывать незначительные проблемы с удобством использования.

С низким приоритетом

Баги, которые незначительно влияют на функциональность или удобство приложения. Могут влиять на незначительные функции или затрагивать только отображение.

Пример — неправильный цвет фона в меню. Но если цвет фона сливается с надписями, это уже баг со средним приоритетом, так как он влияет на удобство пользования.

В разных командах разный подход к тому, какие баги должны быть исправлены до выпуска релиза или обновления, а какие нет. Как правило, баги с блокирующим, критическим и высоким приоритетом фиксятся до релиза (обновления), а баги со средним и низким — в следующих релизах.

Сложности при тестировании мобильных приложений

В процессе тестирования мобильных приложений есть несколько уникальных проблем, нехарактерных для других типов тестирования ПО.

Фрагментация устройств

Фрагментация устройств — самая большая проблема, с которой сталкивается команда разработчиков приложений.

Мобильные приложения нужно тестировать на разных устройствах с разными операционными системами, размерами экрана и аппаратными конфигурациями. Это может затруднить правильную работу приложения на всех устройствах и привести к проблемам совместимости.

Фрагментацию мы уже неоднократно обсуждали. Здесь важно выбрать подходящие устройства для тестирования.

Быстрые циклы разработки

Обычно мобильные приложения разрабатываются по гибкому или итеративному подходу с частыми обновлениями и выпусками.

Разработка ПО разбивается на мелкие и более управляемые части — спринты или итерации. Каждый спринт длится от одной до четырех недель и фокусируется на наборе функций. В конце спринта команда анализирует свой прогресс и при необходимости вносит коррективы на основе отзывов пользователей или заинтересованных сторон.

Одно из ключевых преимуществ быстрых циклов разработки — они позволяют командам быстро реагировать на изменения в потребностях пользователей или условиях рынка. Если функция не работает как надо, команда может быстро скорректировать свой подход и повторить попытку в следующем спринте. Потеря времени и ресурсов сводится к минимуму, а работа над проектом сохраняется в нужном русле.

Однако быстрые циклы разработки могут создавать проблемы для тестировщиков. Новые функции и обновления выпускаются регулярно, тестировщики должны быть в состоянии идти в ногу с темпами разработки и быстро выявлять возникающие ошибки или проблемы. Это особенно сложно при разработке мобильных приложений, где нужно проводить тестирование на разных устройствах и операционных системах.

Тестировщикам может быть сложно следить за изменениями и обеспечивать стабильность и функциональность приложения на протяжении всего цикла разработки. Поэтому важно правильно планировать процессы тестирования и разработки в принципе, а также составлять хорошие наборы регресс-тестов. Если регресс-тестов становится много, а обновления выпускаются часто, есть смысл автоматизировать часть регресс-тестов или их все.

Варианты сети

Мобильные приложения должны работать в разных сетевых условиях, включая низкую пропускную способность, высокую задержку и прерывистые соединения. Тестирование этих условий может быть затруднено, поскольку они часто непредсказуемы и их трудно смоделировать в рабочих условиях.

Тем не менее мы всегда можем отловить много ошибок, имитируя плохую сеть и связь.

Варианты местоположения

Важный аспект тестирования, поскольку многие приложения полагаются на GPS и геолокацию, чтобы предоставить пользователям соответствующую информацию и функции. Некоторые приложения могут давать разную информацию пользователям в зависимости от местоположения.

Тестировщикам нужно учитывать способ, с помощью которого приложение отслеживает локацию пользователей, и имитировать нужное местоположение.

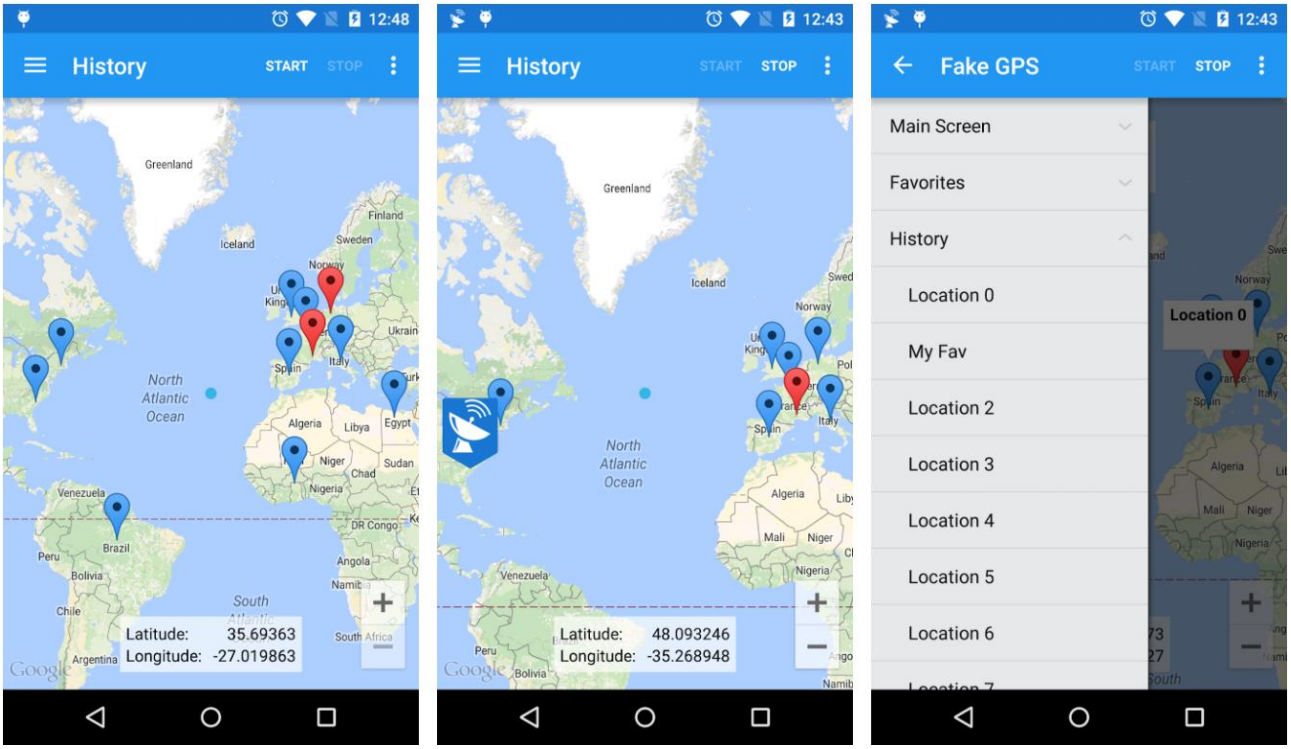
У мобильных приложений есть несколько способов отслеживать регион и местоположение:

- **IP-адрес** (интернет-протокол) — базы данных IP-адресов сопоставляют IP-адреса с регионами, вычисляя приблизительное местоположение.
- **Информация о SIM-карте** — некоторые мобильные приложения могут получать доступ к информации с SIM-карты для определения региона пользователя. Информация может включать код страны и оператора мобильной связи.
- **GPS** — у большинства устройств есть встроенный GPS (система глобального позиционирования), который позволяет определять местоположение устройства на основе спутниковых сигналов. GPS очень точен и может определить местоположение пользователя вплоть до нескольких метров.
- **Разрешения** — некоторые приложения запрашивают доступ к региону пользователя. В этом случае регион определяется по настройкам языка и часового пояса устройства.

Чтобы проверить поведение приложения в другом регионе, мы можем использовать:

- **VPN** (виртуальную частную сеть) — она может направлять интернет-трафик через сервер в другом регионе, что позволяет менять местоположение.
- **Изменение настроек устройства** — мы можем изменить язык, часовой пояс и другие параметры устройства на другой регион. Некоторые приложения это заставит думать, что мы в другом регионе.
- **SIM-карта другого региона** — поможет изменить регион для приложения в случае, если приложение отслеживает локацию по информации о SIM-карте.

- **Приложение-спуфинг** (фейковый GPS) — они могут имитировать другое местоположение на устройстве, манипулируя его GPS-координатами. Пример приложения — Fake GPS:



Источник: [7 Android приложений, которые помогут подделать местоположение]

Поведение пользователей

У пользователей мобильных приложений множество разных предпочтений и вариантов поведения. Приложения нужно разрабатывать с учетом их потребностей и ожиданий. Тестирование поведения пользователей может быть сложным: от тестировщиков требуется имитации реальных сценариев использования и сбор отзывов от реальных пользователей.

Команда разработки должна понимать свою целевую аудиторию. При проработке требований и разработке нужно учитывать ключевые факторы поведения пользователей и их привычки. Для этого команды разработки (чаще аналитики) тщательно анализируют приложения конкурентов и изучают лучшие практики.

Важно собирать отзывы и пожеланий пользователей через магазины приложений и другие каналы связи. Команде разработки важно учитывать все жалобы и пожелания и при необходимости вносить правки в новых версиях приложений.

Регресс-тесты в тестировании мобильных приложений

Место и процесс проведения регрессионных тестов для мобильных приложений глобально не отличается от любых других.

Регрессионное тестирование в мобильных приложениях — это процесс повторного тестирования, цель которого — убедиться, что ранее разработанные и протестированные функции по-прежнему работают правильно после внесения изменений или введения новых функций. Помогает выявить баги или проблемы, которые могли возникнуть из-за изменений в приложении.

Важно изучить и при необходимости актуализировать регресс-тесты, если они уже есть на проекте.

При выборе кейсов для регресс-тестов мы должны отталкиваться от нескольких факторов:

- **Основная функциональность:** тест-кейсы, связанные с основными и широко используемыми функциями приложения. Должны иметь высокий приоритет.
- **Области высокого риска:** нужно определить области, которые склонны к появлению багов, имеют сложную логику или взаимодействуют с внешними системами. Их нужно тщательно тестировать во время регресс-тестов.
- **Недавно измененные области:** важно сосредоточиться на функциях или модулях, которые недавно менялись или дорабатывались.
- **История дефектов:** нужно тщательнее покрывать тестами и тестировать области, где ранее были обнаружены проблемы.
- **Интеграционное тестирование:** важно включать тест-кейсы, охватывающие интеграцию со сторонними сервисами или API. Во время проведения регресс-тестов нужно убедиться, что эти интеграции работают правильно после любых изменений или обновлений.
- **Общие пользовательские сценарии:** те действия, которые пользователи выполняют регулярно. Нужно определить и составить тест-кейсы, охватывающие распространенные пользовательские действия. Сюда входит регистрация, вход в систему, просмотр контента, совершение транзакций и доступ к настройкам учетной записи. Чаще всего они дублируют основную функциональность, так что нужно избегать повторения тест-кейсов.

И, конечно, важно учитывать охват устройств и платформ. Нужно выбирать и составлять тест-кейсы, охватывающие разные устройства, операционные системы и размеры экрана, чтобы обеспечить совместимость с широким спектром мобильных устройств.

Помните: регрессионное тестирование — это повторяющийся процесс, который следует регулярно выполнять, чтобы поддерживать качество и стабильность мобильного приложения, особенно при введении обновлений или изменений.

Пострелизная поддержка приложения

Мы разобрали основные аспекты и процессы тестирования мобильных приложений. Теперь поговорим о том, что происходит, когда мы выпускаем приложение на рынок.

Во-первых, приходится работать над обновлениями. На основе отзывов пользователей, требований бизнеса и тенденций рынка команда разработчиков может определить новые функции, которые стоит добавить в приложение.

Аналитики собирают требования и пишут документацию, разработчики реализуют все, что было запланировано, а тестировщики тестируют новые фичи, проводят регресс-тесты и дополняют тестовую документацию. Этот процесс нам хорошо знаком.

Что еще важно делать команде разработчиков и тестировщиков, когда продукт вышел на рынок?

Мониторинг отзывов пользователей

Команда разработчиков должна активно отслеживать каналы обратной связи. Ими могут быть отзывы в магазине приложений, социальные сети и контактная электронная почта.

Отзывы часто помогают выявить распространенные проблемы пользователей, отслеживать их пожелания и фидбек о том, удобно ли пользоваться приложением. Основываясь на отзывах, команда может расставить приоритеты для новых функций и улучшений.

Как правило, в больших командах есть сотрудники, которые разбирают отзывы, передают фидбек и проблемы ответственным сотрудникам, а также отвечают пользователям.

Исправление ошибок

После выпуска обновлений пользователи часто сталкиваются с новыми ошибками. Они дают обратную связь, и в отдел тестирования могут приходить запросы от поддержки с описанием проблем.

В этом случае тестировщикам нужно:

1. **Воспроизвести баг.** Как правило, у нас есть общая информация о том, что делает пользователь, прежде чем получает ошибку, и данные о его устройстве. На этой основе мы должны попытаться воспроизвести проблему и убедиться, что это действительно баг, а не ошибка пользователя или проблема, связанная с конкретным устройством.

   Например, иногда пользователь выполняет неверные действия, действия, недоступные в его регионе, или хочет получить другой результат. А иногда проблема может быть связана с тем, что на телефоне пользователя сломан какой-то модуль или устройство взломано.

2. **Определить причину.** Когда баг воспроизведен, мы должны проанализировать его причину. Не всегда мы можем сделать это сами и часто прибегаем к помощи программистов. Но можем самостоятельно проверить, уходят ли запросы от приложения к серверу и приходят ли нужные ответы, а также посмотреть логи и проанализировать их. Это поможет программистам быстрее исправить проблему и не затронуть ненужные модули приложения.

3. **Завести баг.** То есть задокументировать его в системе (например, Jira). Баг должен включать в себя подробное описание проблемы, шаги по воспроизведению, операционную систему и устройство, на котором воспроизводится, а также любые необходимые сопутствующие данные (например скриншоты и логи).

4. **Определить приоритет.** В первую очередь следует устранять баги с высокими приоритетами, которые влияют на многих пользователей или препятствуют работе основных функций. Бывают случаи, что для блокирующих или критичных багов нужно выпускать хотфиксы. Чуть позже разберем, что это такое.

5. **Проверить фикс.** После фикса бага мы должны убедиться, что проблема устранена. Может понадобиться провести регрессионное тестирование. С его помощью мы сможем понять, что исправление не привело к возникновению новых проблем.

6. **Обновить тестовую документацию.** После исправления ошибки нужно добавить соответствующие тест-кейсы в регрессионные тесты. Они помогут избежать таких же ошибок у пользователей при дальнейших обновлениях.

Хотфикс

Хотфикс (от англ. hotfix) — это быстрое и временное решение срочной проблемы или бага, обнаруженного в приложении после его выпуска. Обычно представляет собой небольшое исправление, которое включает в себя только решение проблемы и восстановление функциональности приложения. То есть хотфикс — это обновление приложения, но срочно выпущенное и с очень ограниченным количеством изменений.

Хотфиксы часто используются в экстренных ситуациях, когда функциональность приложения серьезно нарушена или пользовательские данные могут быть подвержены риску. В этих ситуациях хотфикс может быть выпущен без полного цикла выпуска, который занимает несколько недель или месяцев.

Конечно, перед выпуском хотфикса обновление проходит внутреннее тестирование. Но из-за критичности исправленных багов и ограниченного времени регресс-тестирование может проводиться не полностью — тестируются только затронутые модули. Соответственно, есть риск, что хотфиксы могут вызвать непредвиденные последствия или проблемы в будущем.

Выводы

На этом уроке мы:

- Узнали, в каких типах приложений больше ошибок.
- Разобрали самые распространенные баги при тестировании мобильных приложений.
- Рассмотрели приоритеты багов.
- Узнали, какие есть сложности при тестировании мобильных приложений.
- Определили принципы составления и выбора тест-кейсов при регрессионном тестировании.
- Разобрали основные моменты пострелизной поддержки приложения.

В курсе мы пошагово изучили процесс тестирования мобильных приложений. Тезисно вспомним основные моменты:

- Мы разобрали этап подготовки к тестированию, который включает анализ требований, планирование тестирования, выбор устройств и разработку тестовой документации.
- Узнали, какие есть средства дистрибуции приложений для удобства распространения тестовых билдов, и поработали с одним из них — Firebase.
- Разобрались, что такое эмуляторы и симуляторы, как они могут помочь в тестировании приложений.
- Познакомились с фермами мобильных устройств, с помощью которых можем подключиться к реальным устройствам для тестирования.
- Посмотрели, как работать с гайдами мобильных сторов.
- Разобрали основные виды тестирования мобильных приложений: функциональные и нефункциональные.
- Изучили основные баги при тестировании приложений.
- Узнали, как строится пострелизная поддержка мобильного приложения и почему так важно проводить регресс-тестирование.

Домашнее задание

- Попробуйте вспомнить какие баги вам чаще всего встречаются в мобильных приложениях.
- Возьмите несколько приложений и пройдите основные тестовые сценарии: позитивные и негативные. Обратите внимание на те аспекты приложения, где чаще всего встречаются баги.

> 💡 Задание является необязательным, но крайне полезным для получения практического опыта и подготовки к семинару. Сдавать задание не нужно.

Используемая литература

1. 7 Android приложений, которые помогут подделать ваше местоположение
2. Daniel Knott Hands On Mobile App Testing -A Guide for Mobile Testers and Anyone Involved in the Mobile App Business Addison Wesley Professional 2015