

Этапы подготовки к тестированию мобильных приложений

Процесс тестирования
мобильных приложений



Оглавление

Введение	3
О чём этот курс	3
На этом уроке	4
Термины, используемые в лекции	4
Процесс тестирования мобильных приложений	5
Этапы подготовки	6
Изучение требований	7
Стратегия выбора реальных девайсов для тестирования	8
Составление тестовой документации	14
Функциональные чек-листы	14
Нефункциональные чек-листы	15
Сервисы дистрибуции приложений	17
Firebase App Distribution	17
Visual Studio App Center	18
TestFairy	19
Appaloosa	20
TestFlight	21
Google Play	22
Firebase: пошаговый процесс	23
Итоги предыдущих глав	26
В какой момент стоит начинать автоматизацию	27
Выводы	28
Домашнее задание	29
Что можно почитать	29
Используемая литература	29

Введение

Добро пожаловать на программу **«Тестирование мобильных приложений»**! На ней вам предстоит пройти несколько курсов:

- Процесс тестирования мобильных приложений
- Объектно-ориентированное программирование
- Автоматизация тестирования мобильных приложений

К началу обучения на этой программе у вас уже должны быть вводные знания об основах тестирования мобильных приложений, которые вы получили на курсе Специализации «Основы тестирования мобильных приложений». На нём вы также разобрали тестирование мобильных приложений на Android и iOS, поработали с Android Studio, изучили, как анализировать трафик с помощью Charles Proxy.

На курсе «Процесс тестирования мобильных приложений» вас ждёт более глубокое погружение в тестирование мобильных приложений, новые инструменты, лайфхаки и гайды для будущей работы.

На курсе по Java вы сможете вспомнить основы языка программирования Java и ООП и углубиться в его изучение для последующей автоматизации тестирования.

На курсе «Автоматизация тестирования мобильных приложений» вы сможете приступить к созданию автотестов, используя Appium.

О чём этот курс

Курс «Процесс тестирования мобильных приложений» посвящён процессу тестирования, релизу и пострелизной поддержке приложения. Мы погрузимся в каждый из этапов и пройдем путь тестирования мобильного приложения в теории и на практике.

По результатам прохождения курса вы:

- Сможете выбирать устройства для тестирования
- Познакомитесь с работой мобильных ферм
- Увидите, как происходит работа с мобильными сторами
- Сможете составить тест-кейсы для прохождения функционального и нефункционального тестирования приложения
- Проведёте тестирование мобильного приложения

На схеме вы можете увидеть подробное описание уроков и их расположение на нашей карте:



На этом уроке

На этом уроке мы пошагово разберём процесс подготовки к тестированию мобильных приложений и рассмотрим главные сервисы, которые будут помогать организовать процесс тестирования:

- Подберём обязательные и желательные устройства (на основе статистики, пользователей и т. д.)
- Разберём, какие проверки нужно добавлять в тестовую документацию
- Рассмотрим сервисы дистрибуции мобильных приложений
- Поближе посмотрим на Firebase
- Поймём, когда на проекте пора вводить автоматизацию

Термины, используемые в лекции

Легаси-код (от англ. legacy — наследие) — это код, который одна команда разработчиков получила в «наследство» от предыдущей.

Матрица совместимости устройств — это документ, чётко определяющий объём усилий по разработке и тестированию, связанных с конкретными платформами и телефонами.

Процесс тестирования мобильных приложений

Обычно процесс тестирования мобильного приложения включает в себя следующие шаги:

- **Подготовка к тестированию**

Анализ требований. Первый шаг — это ознакомление с требованиями и понимание целей приложения, его особенностей и ожиданий пользователей.

Планирование тестирования, выбор устройств и разработка тестовой документации. Включает определение объёма тестирования, создание сценариев тестирования и определение подхода к тестированию, включая типы тестирования, такие как функциональное, тестирование производительности, безопасности и тестирование удобства использования.

- **Проведение тестирования**

Создание тест-кейсов. На основе требований и сценариев тестирования создаются тест-кейсы, чтобы обеспечить покрытие всех аспектов приложения в процессе тестирования.

Выполнение тестирования. Тест-кейсы выполняются на разных устройствах, операционных системах и сетях, чтобы выявить дефекты и проверить функциональность приложения.

Заведение и отслеживание дефектов. Все обнаруженные дефекты во время тестирования заводятся в систему баг-трекинга (это система, где обычно хранится информация по дефектам и, как правило, задачам), и команда разработчиков уведомляется. Дефекты отслеживаются до тех пор, пока они не будут исправлены и проверены.

Регрессионное тестирование. После исправления дефектов выполняется регрессионное тестирование, чтобы убедиться, что изменения, внесённые в приложение, не повлияли на его существующую функциональность.

Пользовательское тестирование. После того как приложение становится стабильным, выполняется пользовательское тестирование, чтобы убедиться, что приложение соответствует требованиям и ожиданиям пользователей.

- **Пострелизная поддержка**

Релиз. После успешного завершения всех этапов тестирования приложение выпускается для пользователей.

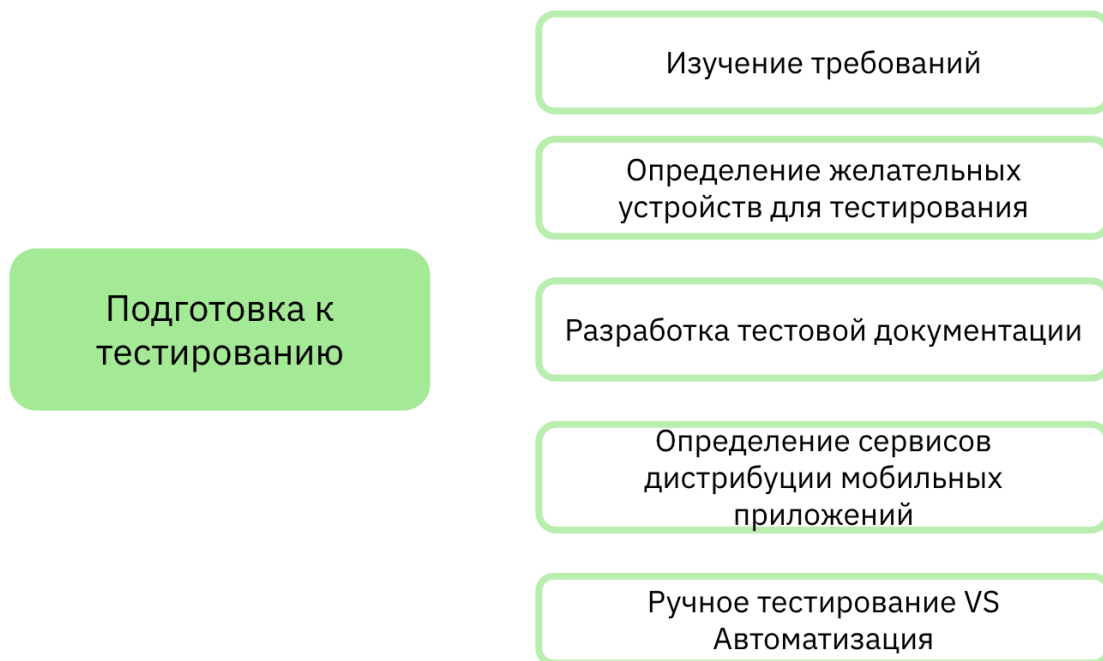
Мониторинг после выпуска. После выпуска приложения выполняется мониторинг, чтобы выявить возможные проблемы или дефекты и убедиться, что приложение работает корректно на всех устройствах и платформах.

В целом, процесс тестирования мобильного приложения включает в себя тщательное планирование, выполнение и мониторинг, чтобы обеспечить высокое качество приложения, соответствующее ожиданиям пользователей и разработчиков.


Наш курс нацелен на то, чтобы мы детально разобрали каждый из этапов и совершили наше первое полноценное тестирование мобильного приложения. На этом уроке мы начнём с подготовительного этапа и всего того, что он в себя включает.

Этапы подготовки

Перед тем как мы приступим к тестированию нашего приложения, мы должны выполнить несколько подготовительных этапов.



Они будут немного отличаться в зависимости от того, пришли ли мы на новый проект, на котором приложение только разрабатывается, или это старый проект с легаси-кодом.

 **Легаси-код** (от англ. legacy — наследие) — это код, который одна команда разработчиков получила в «наследство» от предыдущей.

Но если мы пришли на старый проект, нам в любом случае нужно проанализировать приложение, чтобы понять его узкие места, сложности в реализации функций приложения. Это поможет составить оптимальную стратегию и охватить все вещи, которые, возможно, были пропущены ранее.

Изучение требований

В первую очередь изучаются требования к продукту. Для этого мы изучаем всю доступную документацию по нашему мобильному приложению. Видами документации могут быть:

- Требования и Спецификации
- Технические характеристики приложения
- Руководства пользователя
- Руководства по эксплуатации

Сбор требований является основой любого типа тестирования. Это помогает нам определить методики тестирования и даёт понимание о необходимом тестовом окружении.

В результате изучения документации и всех требований мы можем составить общий план тестирования (понять, какие виды тестирования применимы к нашему проекту, и определиться с подходом к тестированию) и добавить в него те виды тестирования мобильных приложений, которые применимы и необходимы в нашем проекте.

Типичные виды, которые применяются в мобильном тестировании:

- Функциональное тестирование
- Тестирование совместимости
- Тестирование производительности
- Стресс-тесты
- Тестирование безопасности
- Тестирование юзабилити
- Тестирование локализации и глобализации
- Тестирование установки и обновления

Мы проходимся по списку и смотрим, актуально ли нам составлять тест-кейсы по этому виду тестирования?

Отдельно отмечу ограничения по нашему мобильному приложению, то есть какие версии операционной системы приложение не должно поддерживать, какие есть ограничения на оперативную память и прочее. Нужно будет добавить проверки на то, что приложение не поддерживается там, где не должно.

После **изучения документации** важным шагом будет **выбор устройств для реального тестирования**.

Стратегия выбора реальных девайсов для тестирования

Перед тем как обсудить этот этап, надо добавить, что тестирование может проводиться на:

- реальных девайсах
- симуляторах
- эмуляторах
- мобильных фермах

Тестирование на реальных девайсах является критически важным, а остальные инструменты идут, как правило, в помощь. Мы поговорим про них чуть позже на нашем курсе.

Также вспомним, что мобильные приложения бывают трёх видов:

- **Нативные приложения** — это приложения для конкретной платформы. Разрабатываются с помощью SDK, инструментов разработки, датчиков и функций конкретных платформ.

Они загружаются, устанавливаются и обновляются из магазинов приложений (например, App Store или Google Play). Таким приложениям может потребоваться тестирование на всех поддерживаемых устройствах.

- **Веб-приложения** — это приложения, использующие веб-технологии для работы на мобильном устройстве.

Одно приложение (с одним и тем же исходным кодом) может работать на различных устройствах и платформах, что существенно ускоряет процесс разработки. При запуске приложения в браузере оно запускается независимо от модели телефона.

- **Гибридные приложения** — это мобильные приложения, представляющие собой сочетание между нативными и веб-приложениями.

Они используют встроенную оболочку приложения, которая содержит веб-представление для запуска веб-приложения внутри приложения для конкретной платформы. Возможные слабые места таких приложений — это проблемы с производительностью из-за использования оболочки и возможные отклонения от ожидаемого внешнего вида и поведения приложения из-за особенностей платформы. Поэтому необходимо проводить тестирование на разных устройствах.

Почему нам важно выбрать корректные мобильные устройства? Сейчас на рынке представлено огромное количество устройств с разными характеристиками.

Основные из них:

- Разные экраны и конфигурации;
- Разные типы моделей и производителей, например, Apple, Samsung, Google, Xiaomi, OnePlus и т. д.;
- Разные операционные системы, включая Android, iOS и другие;
- Различные версии ОС. Например, iOS 15, Android 13 и так далее. При выборе девайсов с различными версиями ОС необходимо учитывать то, насколько регулярно и часто обновляются девайсы. При каждом обновлении требуется новый цикл тестирования, чтобы гарантировать, что приложение работает корректно.

В отличие от iOS, выбор девайсов на Android куда сложнее. Поскольку Android поддерживается многими производителями мобильных устройств, разнообразие версий больше, чем в iOS. При этом одновременное существование разных версий Android тоже сильно больше. Как это может повлиять на пользователей?

Большая часть клиентов судит о надёжности продукта на основе диапазона поддерживаемых устройств и ОС. То есть когда пользователь видит, что его девайс или версия ОС не подходит — это, конечно, разочарованный клиент и потери для бизнеса. Более половины пользователей удалят приложение или не будут повторно заходить в него из-за проблем на устройстве и скачают приложение конкурента.

Перечислим важные варианты при выборе корректных мобильных устройств и дальнейшей работе с ними.

Сужение списка мобильных устройств

Для сужения области и определения списка тестовых устройств, которые на 100% подходят для теста, необходимо выполнить следующие шаги:

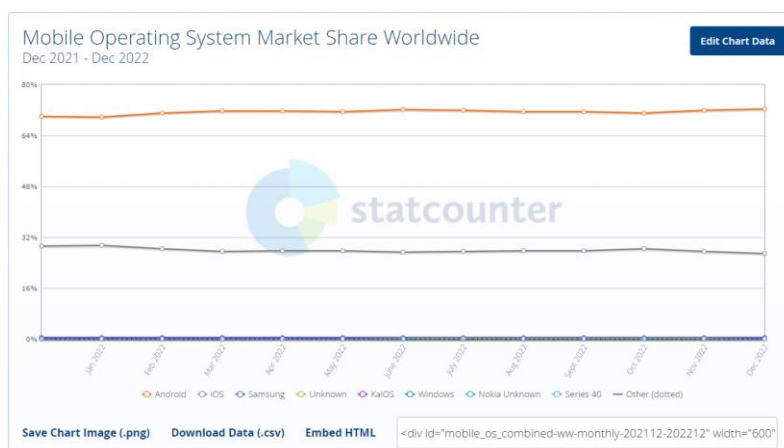
1. **Сбор требований.** Важно собрать все требования к продукту и проверить, есть ли устройство или устройства, которые мы хотим обязательно включить в список тестируемых девайсов. Например, если в приложении есть функционал, который поддерживает 3D touch на iOS (технология Apple, позволяющая определять силу нажатия на дисплей), то нужно включить в список те модели айфона, которые поддерживают эти функции.
2. **Определение доли рынка для разных платформ.** Крайне важно проанализировать рынок и ваших потенциальных клиентов, чтобы легко определить долю рынка для разных платформ. Как правило, данные о потенциальных клиентах мы можем получить из документации, которую предоставляет аналитик. А вот с помощью разных сервисов, которые мы рассмотрим дальше, можно будет подобрать популярные у пользователей устройства.

Несколько сервисов, которые могут помочь нам собрать статистику:

- [Google Analytics](#) собирает статистику по разным производителям (Apple, Samsung и прочие). Также есть статистика, откуда приходят пользователи.
- [Statcounter Global Stats](#) — очень объёмный и популярный сервис для сбора статистики. Можно получить данные как по популярным ОС, так и по разрешениям экранов, наиболее активно используемым в интересных нам странах. Удобные фильтры и наглядные графики делают сервис популярным.

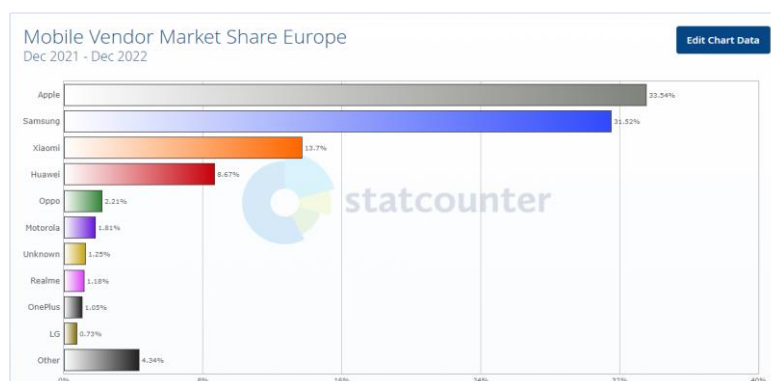
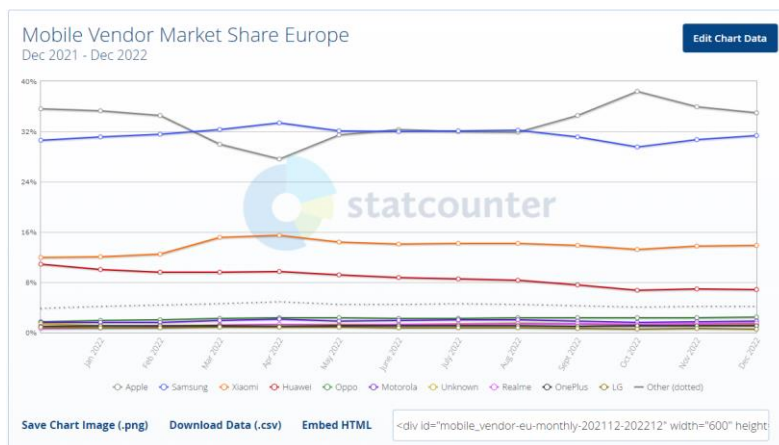
Рассмотрим примеры.

Распределение по мобильным ОС в двух видах графиков:





Распределение по производителям мобильных устройств в Европе (тоже для наглядности в двух видах отчётов):



- [AppBrain.com](https://appbrain.com) — простой в использовании и бесплатный сервис для получения статистики по Android-устройствам.
- Порталы сторов: крупные сторы, такие как Google Play и App Store, собирают свою статистику и предоставляют разработчикам данные.

Анализ целевой аудитории

Целевая аудитория играет существенную роль при выборе устройства для тестирования мобильного приложения. Нам нужно собрать статистику по тому,

какие устройства используют наши потенциальные клиенты. На этом этапе полезно изучать клиентов наших конкурентов и статистику по ним.

Выбирая количество устройств по перечисленным выше пунктам, нужно понимать размер команды (от этого зависит, на каком количестве девайсов мы можем протестировать наше приложение) и установленный лимит на приобретение всех устройств. Если у вас небольшой стартап, то велика вероятность, что все устройства приобрести не получится, и нужно будет выбрать только важные по каждому из пунктов.

Создаём матрицу устройств

Создание матрицы устройств, сравнивающей характеристики каждого устройства, поможет нам ещё больше сузить выбор до точной ОС, разрешения экрана, ЦП, памяти, ОЗУ, камеры и прочих параметров.

Составление матрицы устройств — это более комплексный подход при выборе устройства.

При разработке мобильного приложения разработчики, скорее всего, будут иметь список версий мобильных ОС и спецификаций, с которыми будет совместимо наше приложение. После того как этот список завершён и задокументирован, его часто называют матрицей совместимости устройств.

Матрица совместимости устройств — это документ, чётко определяющий объём усилий по разработке и тестированию, связанных с конкретными платформами и телефонами.

Note: 1. Minimum supported version is 4.2 2. Distribution percentage is derived from: https://developer.android.com/about/dashboards						
Manufacturer	Model	Firmware	Screen Size (Inches)	Resolution (pixels)	Tier	Test Type
Google	Pixel	8.1.0 / 8.0.0	5.0	1080 * 1920	Primary Device	Every version of the application will be tested on this devices
Samsung	S7 Edge	7.0.0	5.5	1440 * 2560	Primary Device	Every version of the application will be tested on this devices
Nexus	6P	7.1.2	5.7	2560x1440	Tier 1	Regression Testing on the closed beta builds will be performed on this devices
Samsung	Galaxy S6	6.0.1	5.1	1440 * 2560	Tier 2	Sanity Test on the release candidate version will be performed on this devices
Motorola	Moto G4 Plus	7.0.0	5.5	1080 x 1920	Tier 1	Regression Testing on the closed beta builds will be performed on this devices
Samsung	Note 3	5.0.0	5.7	1080 x 1920	Tier 2	Sanity Test on the release candidate version will be performed on this devices
HTC	Nexus 9 (Tablet)	7.0.0	8.9	1536 x 2048	Tier 2	Regression Testing on the closed beta builds will be performed on this devices
HTC	Nexus 4	4.4.x	4.7	1280 x 768	Tier 2	Sanity Test on the release candidate version will be performed on this devices
Motorola	G	4.3	4.5	720x1280	Tier 2	Sanity Test on the release candidate version will be performed on this devices
Samsung	Nexus 10	5.1	10.1	2560 * 1600	Tier 1	Regression Testing on the closed beta builds will be performed on this devices
Xiaomi	MI A1	7.1.2	5.5	1080x1920	Tier 1	Regression Testing on the closed beta builds will be performed on this devices

Создаётся он с той же целью: мы не можем протестировать наше приложение на всех девайсах, и нам нужно их как-то группировать.

Ключевые факторы при создании матрицы устройств:

- **Платформы и производители.** Здесь мы учитываем не только нужные ОС, вроде iOS и Android. В случае с Android это могут быть популярные оболочки (вроде тех, что стоят на Самсунгах и Хуавеях поверх чистого андроида).
- **Поддерживаемые экраны и разрешения.**
- **Прочие технические характеристики.** Например, оперативная память, процессор и прочее.

На выходе мы получаем список устройств с прописанными характеристиками. И, совмещая полученные характеристики по разным устройствам, мы можем понять, что лучше брать для тестирования, а что будет дублировать характеристики уже выбранных девайсов.

Матрица совместимости устройств ограничивает охват разработки и тестирования мобильных приложений определённым набором версий операционной системы и наборов спецификаций. То есть это иногда полезный шаг ещё на этапе проектирования продукта, когда мы понимаем, что, возможно, нам вообще не нужно поддерживать некоторые устройства. Когда мы решаем, что не будем поддерживать некоторые устройства, исходя из их специфики, мы сильно экономим время на разработку и тестирование, при этом не делая костыли или заглушки в нашем приложении.

Ограничение количества совместимых версий мобильных ОС помогает командам сосредоточиться на самых популярных версиях, соответствующих их целевой аудитории. Это также помогает уменьшить вероятность наличия кроссплатформенных проблем в нашем приложении. Кроме того, тестирование и оптимизация приложений на последних комбинациях устройств и ОС позволит нам привести приложение в соответствие с последними мобильными версиями на глобальном рынке.

А чётко определённая область помогает обеспечить наилучшее взаимодействие с пользователем для выбранных платформ.

Зависимость от типа тестирования

Стратегия тестирования также влияет на то, какое количество мобильных устройств мы можем взять для тестирования. При ручном тестировании нам нужно стараться ограничиваться минимальным парком девайсов, чтобы проверки не занимали слишком много времени (один из частых вариантов на проектах — новое сильное

устройство и устройство с очень слабыми характеристиками). Когда дело касается автоматизации тестирования, мы, конечно, можем сильно расширить парк устройств, так как в этом случае тесты будут подгоняться автоматически, и всё, что от нас будет требоваться — анализировать результаты тестов.

Регулярный пересмотр тестовых устройств

Рынок развивается достаточно быстро, и каждый год появляются новые модели с новыми спецификациями и функционалом. Важно держать руку на пульсе и периодически пересматривать наш пакет мобильных устройств с целью актуализации. Это касается и поддержки приложением устаревших устройств и ОС в принципе. Когда мы видим, что какие-то старые устройства полностью теряют свою актуальность, — это хороший момент, чтобы пересмотреть список поддерживаемых устройств.

Составление тестовой документации

Очень важный этап подготовки к тестированию — это составление тестовой документации. Напомню, что в тестовую документацию могут входить наборы чек-листов, тест-кейсов, тестовых данных и прочее. Рассмотрим основные виды чек-листов и разберём, какие проверки нам нужно добавить именно для мобильных приложений.

Сейчас мы в общих чертах рассмотрим функциональные и нефункциональные проверки. Подробнее про это и составление тест-кейсов поговорим на следующих уроках.

Функциональные чек-листы

- Корректный вызов клавиатуры
- Корректная работа клавиатуры
- Проверка, чтобы убедиться, что навигация между соответствующими модулями в приложении соответствует требованиям
- Корректная работа функции прокрутки, чтобы проверить плавность страницы во время прокрутки
- Проверка, что пользователь получает соответствующее сообщение об ошибке, например, «Ошибка сети. Пожалуйста, попробуйте через некоторое время», всякий раз, когда возникает какая-либо сетевая ошибка

- Проверка, что приложение выполняет функцию автоматического запуска в соответствии с требованиями

Нефункциональные чек-листы

Здесь мы добавляем проверки по тем видам тестирования, которые мы выбрали, когда изучали документацию и требования по продукту.

Разберём несколько примеров тестов по каждому виду:

1. Проверка производительности:

- Время отклика приложения соответствует требованиям
- Приложение работает в соответствии с требованиями при различных условиях нагрузки
- Проверка производительности приложения при изменении сети на WI-FI с 2G/3G или наоборот
- Проверка потребления батареи, утечки памяти

2. Проверка безопасности:

- Приложение не позволяет третьим лицам получить доступ к конфиденциальному содержимому или функциям без надлежащей аутентификации
- Приложение имеет надёжную систему защиты паролем и не позволяет злоумышленнику получить, изменить или восстановить пароль другого пользователя
- Кэш клавиатуры — для предотвращения небезопасного хранения данных в кэше клавиатуры приложений
- Надёжная система защиты приложения паролем
- Время истечения сеанса приложения соответствует требованиям

3. Проверка совместимости:

- Проверка приложения на последних мобильных устройствах и версиях ОС
- Проверка приложения на старых устройствах
- Если приложению требуется веб-браузер: как оно будет работать в различных браузерах и его более ранних версиях
- Читаемость текста на разных устройствах и версиях ОС

4. Тестирование юзабилити:

- Кнопки и поля имеют правильный размер
- Процесс навигации прост и интуитивно понятен
- Меню расположено корректно
- Кнопки, имеющие одинаковую функцию, имеют одинаковый цвет
- Контекстные меню не перегружены
- Пользователю предоставлено руководство пользователя, которое помогает понять и использовать приложение

5. Стресс-тестирование:

- Работа приложения при высокой загрузке ЦП (процессора телефона)
- Работа приложения при нехватке памяти
- Работа приложения при нехватке места на диске
- Работа приложения при активном использовании батареи
- Работа приложения при плохой пропускной способности
- Большое количество взаимодействий с пользователем (для этого может потребоваться моделирование реальных сетевых условий)

6. Тестирование восстановления:

- Способность приложения восстанавливаться после внезапного сбоя (например, приложение крашится, или пользователи какое-то время не могли зайти в приложение)
- Поведение приложения во время сбоя, который происходит между денежными транзакциями
- Приложение ведёт себя корректно после сбоя питания (например, если садится телефон)
- Приложение способно восстанавливать данные после неудачного сценария подключения

7. Тестирование локализации и глобализации:

- Сообщения об ошибках корректны и выглядят корректно на всех поддерживаемых языках
- Время и дата отображаются корректно для каждого региона
- Все элементы пользовательского интерфейса приложения изолированы
- Статические элементы пользовательского интерфейса приложения синхронизированы с пользовательским интерфейсом приложения

После составления тестовой документации мы переходим непосредственно к тестированию. И первое, что нам нужно, — получить тестовые сборки приложения на наши устройства. Разные версии сборок приложения часто называют билдами.

Сервисы дистрибуции приложений

Для удобства распространения тестовых билдов существует довольно большое количество сервисов дистрибуции. Мы посмотрим самые популярные из них. Но сначала разберёмся в том, как вообще они работают.

Эти сервисы позволяют распространять тестовые версии нашего приложения среди тестировщиков по беспроводной сети (OTA — Over The Air). Ими мы пользуемся, чтобы не собирать каждый раз приложение непосредственно на устройствах или не передавать сборку приложения просто файлом (это может быть неудобно и небезопасно).

Глобально все сервисы работают примерно по одному принципу:

1. **Настройка проекта.** На этом этапе, как правило, создаём проект и заполняем информацию по нашему продукту.
2. **Создание групп рассылки.** На этом этапе мы добавляем в проект тех пользователей, которые должны иметь доступ к приложению, как правило, это тестировщики, руководитель проекта, возможно, заказчики. В зависимости от инструментария мы можем настраивать разные группы и даже отправлять пользователям разные версии нашего приложения.
3. **Загрузки билда в сервис.** В большинстве сервисов, чтобы корректно загрузить билд, нужно интегрировать SDK (Software development kit. Подробнее о том, что это, расскажем на следующем уроке) в приложение.
4. **Выпуск версии для настроенных групп.**

Firebase App Distribution

Firebase App Distribution является платформой Google и частью проекта Firebase. Это довольно простой сервис, к тому же большинство функций в нём бесплатны, поэтому он очень популярен.

Используя все сервисы Firebase, мы можем просто получить доступ к крэшлогам пользователей (это удобно в случае, когда у кого-то из тестеров крэшит приложение. Посмотрев крэшлоги, разработчик может локализовать проблему). Крэшлоги — это

данные о том, что происходило с приложением в момент сбоя, а также данные о разделе приложения, в котором произошёл сбой.

Приглашения для вступления в тестовую группу можно отправлять по электронной почте. При этом можно создавать разные группы и давать доступ или, наоборот, запрещать его нужным группам тестировщиков и пользователей.

Основные фишки:

- Работает для iOS и Android
- Простая и быстрая настройка
- Мгновенная доставка наших сборок пользователям
- Простое управление группами
- Возможность смотреть, кто принимает приглашение и кто его загружает
- Сочетается с crashlytics (сервис для получения статистики) для получения информации о стабильности приложения

Из минусов можно отметить то, что сервис не поддерживает обратную связь в приложении от тестировщиков. То есть тестировщик не может оставить комментарий напрямую через этот сервис (а это может быть удобно, если у нас большое количество пользователей, которые смотрят наши тестовые сборки).

Позже мы пошагово рассмотрим процесс дистрибуции приложения через этот сервис.

Visual Studio App Center

App Center можно интегрировать с нашим приложением для Android, iOS, macOS или Windows.

- Можно настроить группы пользователей или сделать общедоступную ссылку на наше приложение. Помимо распространения, в сервисе удобно собирать отчёты о сбоях, сегментированные уведомления в приложении, отзывы пользователей и аналитику.
- В App Center есть удобные функции управления пользователями, что позволяет вам контролировать, кто имеет доступ к каждой сборке. Тестировщики могут быть организованы в группы, и несколько сборок могут быть отправлены в несколько групп одновременно. Можно предоставить или отозвать доступ к отдельным сборкам для группы и для отдельных тестировщиков.

- У App Center есть возможность настроить интеграцию с сервисами вроде Jira для автоматического отслеживания крашей и сбоев. Разработчики могут быть быстро проинформированы о важных событиях App Center. Простыми словами, вот как это работает: у тестировщика во время работы с приложением произошёл сбой. App Center автоматически собрал необходимую информацию и крэшлоги. После этого в Jira создаётся тикет и падает разработчикам. С уже собранной информацией и логами разработчики могут быстро исправить проблему и добавить фикс в следующую сборку.

Подведем итоги: App Center — отличная платформа для дистрибуции приложений, он покрывает большинство потребностей разработки и тестирования.

TestFairy

TestFairy — это платформа бета-распространения, ориентированная на крупных корпоративных пользователей. Её также удобно использовать при проведении юзабилити-тестирования. Сервис доступен для Android и iOS.

Какие у него плюсы:

- TestFairy позволяет нам приглашать тестировщиков по электронной почте, SMS или через целевую страницу, которую он создаёт для каждого из наших приложений. Эта целевая страница может быть открыта для всех, кто хочет присоединиться. Или закрыта, в таком случае тестировщики могут запросить доступ. Как только тестировщики начинают тестировать наше приложение, TestFairy записывает видеосессии, чтобы мы могли точно увидеть, что произошло в приложении, прежде чем возникла проблема. SDK TestFairy также предоставляет нам встроенный в приложение инструмент для создания отчётов об ошибках и сбоях, облегчающий процесс для тестировщиков и других пользователей.
- В TestFairy есть функции для управления тестировщиками, которые позволяют объединять тестировщиков в группы и управлять их разрешениями. Однако пользователи не могут тестировать две разные сборки одного и того же приложения одновременно.
- TestFairy также поддерживает единый вход на самых популярных платформах. Кроме того, его можно разместить в облаке или локально, если требуется дополнительная безопасность.
- TestFairy интегрируется со многими популярными инструментами для отслеживания ошибок, коммуникации и непрерывной интеграции.

- Что касается безопасности, у сервиса есть интеграция с Google, Okta, Onelogin и другими популярными инструментами.

Подведем итоги: можно сказать, что TestFairy — действительно очень простой и удобный инструмент. У него есть функции, которые не поддерживаются Firebase или App Center (например, запись сессии). Но при этом есть минус: это платный инструмент, и, скорее всего, он не подойдёт для независимых разработчиков и небольших компаний, которым не нужны некоторые из его функций.

Appaloosa

Appaloosa — это корпоративный магазин приложений, который также можно использовать для распространения бета-приложений на Android и iOS. Используя этот сервис, мы можем загрузить свою бета-версию и распространить её среди тестировщиков через собственный магазин приложений.

Тестировщики направляются в наш частный магазин приложений по ссылке, которую мы можем отправить по электронной почте. Appaloosa не предполагает каких-либо инструментов обратной связи в приложении, отчётов об ошибках или сбоях, но тестировщики могут отправлять отзывы на нашу страницу в магазине приложений.

Рассмотрим его плюсы:

- Appaloosa поддерживает SAML, OAuth и SAML SSO для повышения безопасности. Appaloosa также поддерживает push-уведомления и автообновление приложения, которые могут быть полезны, чтобы убедиться, что все наши тестировщики используют последнюю сборку.
- Также поддерживается управление тестировщиками и возможность организовывать их в группы с разными полномочиями и разрешениями, чтобы мы могли контролировать, кто имеет доступ к нашему приложению.
- Appaloosa интегрируется с большинством популярных инструментов для отслеживания ошибок, коммуникации и непрерывной интеграции. Например, можно настроить интеграцию с корпоративным мессенджером и включить для всей команды оповещения о важных вещах.
- У Appaloosa есть бесплатный тарифный план, но с большими ограничениями. Также есть несколько тарифов с расширением доступного функционала.

Подведем итоги: Appaloosa — это ещё один инструмент, который подходит более крупным компаниям. Из плюсов — конечно, магазин приложений и автообновление. А большой минус — это отсутствие отчётов о сбоях и крашах.

TestFlight

Если до этого мы рассматривали кроссплатформенные инструменты, то сейчас посмотрим сервисы, которые поддерживаются непосредственно площадками. Первый — TestFlight, продукт Apple.

Apple приобрела TestFlight в 2014 году, интегрировала его в iTunes Connect и приняла в качестве своего инструмента для распространения бета-приложений. Естественно, TestFlight поддерживает исключительно платформы iOS, watchOS и tvOS.

TestFlight позволяет добавить до 25 внутренних тестировщиков из числа пользователей iTunes Connect, если у них есть роль администратора, менеджера приложений, разработчика, маркетолога или технического специалиста. Что касается внешних тестировщиков, мы можем добавить до 10 000 тестировщиков, используя их электронные адреса. Тестировщики получают электронное письмо со ссылкой, которая предлагает им загрузить приложение TestFlight.

Затем тестировщики могут использовать приложение TestFlight для загрузки нашего бета-приложения и получать уведомления в приложении, когда придёт время для обновления.

Пройдёмся по плюсам:

- TestFlight поддерживает открытое бета-тестирование, создавая общедоступную ссылку для подписки и распространения нашего приложения среди тестировщиков.
- Мы также можем установить максимально допустимое количество тестировщиков для нашего открытого бета-тестирования, чтобы не превысить возможности команды.
- TestFlight поддерживает одновременное тестирование до 100 приложений. Сборки доступны в течение 90 дней после загрузки, а внутренние тестировщики имеют доступ ко всем сборкам наших приложений. Мы можем выбрать, какие приложения и сборки сделать доступными для внешних тестировщиков, и организовать этих тестировщиков в разные группы. Это позволяет одновременно тестировать разные сборки одного и того же приложения с разными группами.
- Основная проблема, с которой сталкивается большинство разработчиков при работе с TestFlight, заключается в том, что, прежде чем сборки можно будет отправить внешним тестировщикам, они должны сначала пройти процесс утверждения Apple. Утверждение первой сборки занимает больше времени,

чем последующие сборки, а время обработки зависит от платформы разработки. Но тем не менее для того, чтобы отправить первую сборку внешним тестировщикам, нам обычно приходится ждать несколько дней.

- Ещё большим плюсом использования TestFlight является то, что мы можем тестировать покупки внутри приложения (IAP — In App Purchase). А также то, что в TestFlight мы загружаем сборки, которые потом могут идти в App Store. То есть мы можем проверить версию, которая будет отправлена непосредственно в стор (App Store) до открытия приложения на всех пользователей.

Подведем итоги: TestFlight — отличный вариант для малых и средних команд, разрабатывающих для среды Apple. Группы тестировщиков и возможность тестировать сборки одновременно делают его хорошим инструментом для не слишком сложных бета-тестов.

На что стоит обратить внимание:

- Если нам требуется более подробная аналитика о наших тестировщиках или производительности нашего приложения, придётся переходить на сторонние инструменты.

Google Play

Google Play, соответственно, является инструментом Google и помогает проводить бета-тесты билдов для магазина приложений Google Play. По сути, это и является магазином. Только используя консоль Google Play, мы можем настраивать распространение билдов для пререлизного тестирования (то есть распространять билды через Google Play до того, как они станут доступны всем пользователям).

Его основные плюсы:

- Консоль разработчика Google Play поддерживает распространение внутренних, закрытых и открытых тестовых версий.
- Мы можем добавлять тестировщиков по электронной почте, если это учётная запись Gmail. Внутренние тесты ограничены 100 тестировщиками на приложение, а в закрытых бета-версиях не ограничено количество тестировщиков, которых мы можем добавить. Получив ссылку, тестировщики смогут загрузить приложение из Google Play Store.
- Google не устанавливает ограничений на количество приложений, но не позволяет одновременно тестировать разные сборки одного и того же приложения в открытой и тестовой версии.

- Что касается управления группами тестировщиков — тут довольно ограниченное количество функций. Мы можем просто задать список пользователей, у которых будет доступ к магазину приложений.
- Сборки, отправленные в Google Play, также должны пройти проверку, но обычно это занимает пару часов.

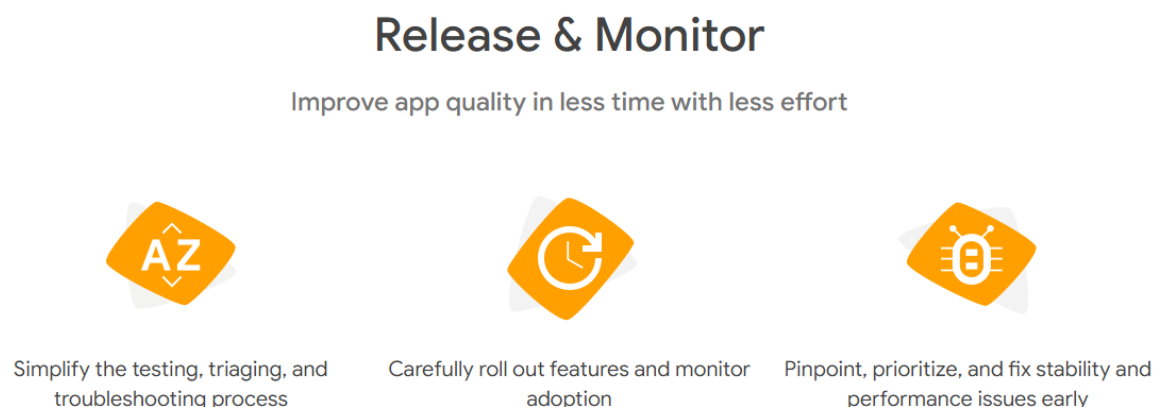
В итоге Google Play является довольно удобным инструментом для дистрибуции приложений. Особенно он удобен тем, что тестировщикам не нужны никакие сторонние приложения — билд можно скачать прямо из официального магазина приложений. При выпуске обновления приложение может обновиться автоматически. Также есть возможность тестировать покупки внутри приложения. Но если нам нужно точнее настроить группы пользователей и разные версии билдов — это не самый удобный инструмент.

Подробно работу со сторями App Store и Google Play мы рассмотрим на следующем уроке.

Firebase: пошаговый процесс

А сейчас пошагово разберём процесс дистрибуции приложений через Firebase. Для этого представим, что у нас есть готовая сборка приложения для Android — APK-файл.

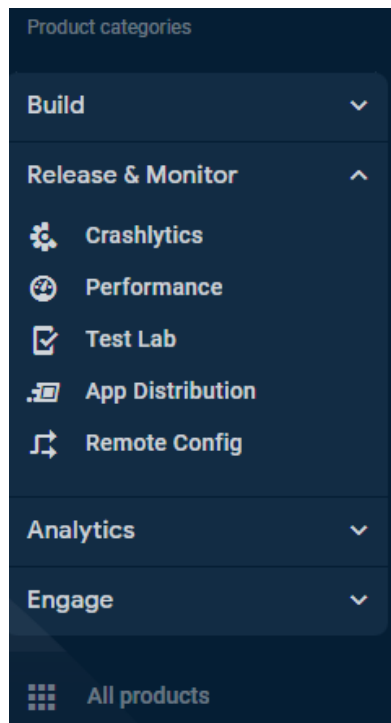
Шаг 1. Заходим в Firebase, используя гугл-аккаунт.



Шаг 2. Далее нам нужно создать проект.

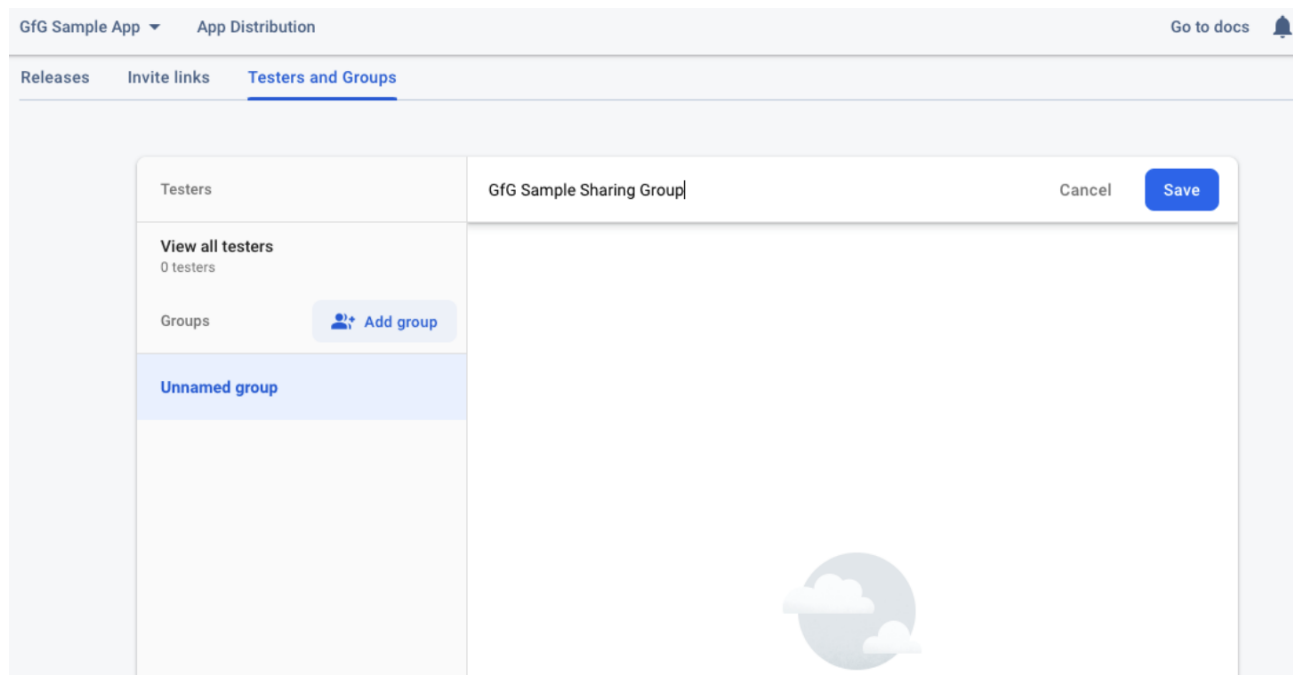
Прежде чем распространять приложение, мы должны создать группы и добавить тестировщиков в каждую из них.

Шаг 3. В веб-консоли Firebase находим **Firebase Distribution** в группе **Release and Monitor**.



Шаг 4. Затем создаём группы во вкладке **Verifiers and groups** и добавляем в каждую группу адреса электронной почты тех, кто хочет протестировать приложение.

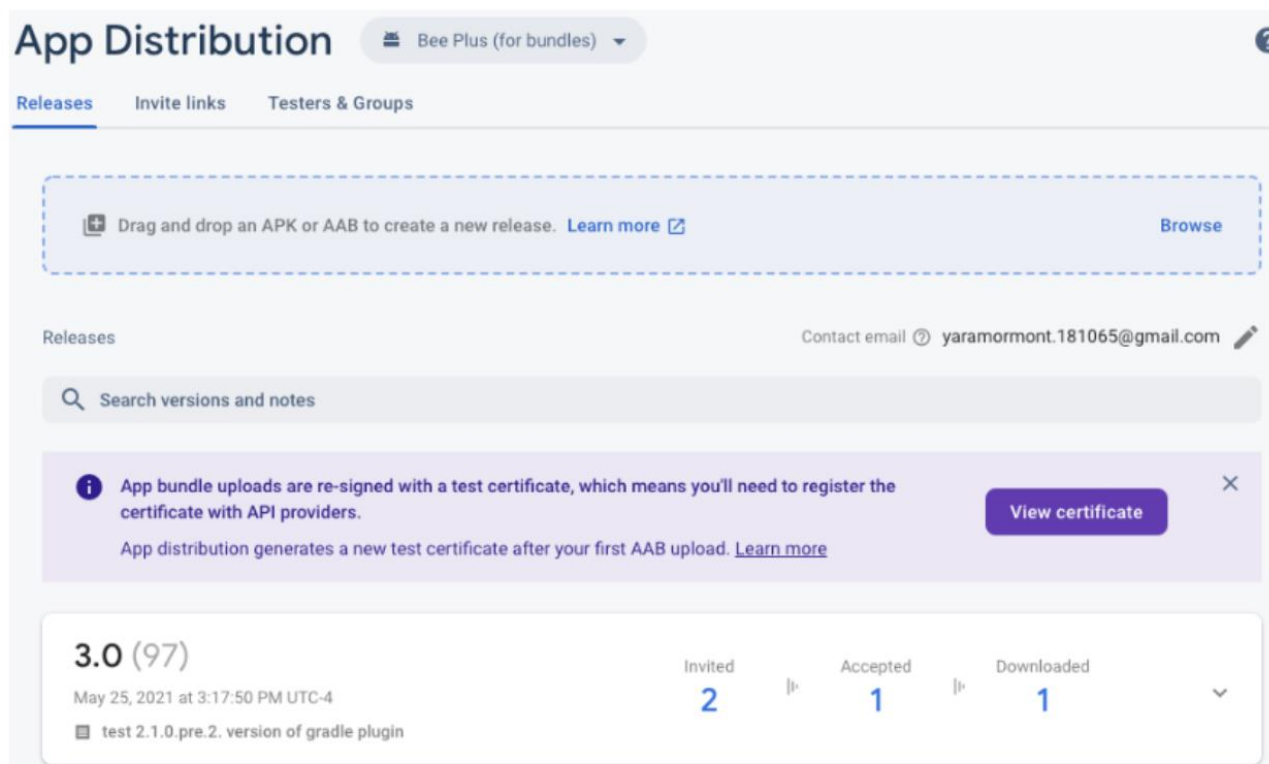
Это удобный способ, однако мы можем не создавать группы и просто добавлять электронные адреса.



Мы также можем загружать электронные адреса через CSV. Это удобно, когда у нас большой список тестировщиков: мы вносим всех в один документ и загружаем его.

Шаг 5. И рассмотрим два варианта того, как мы будем давать доступ тестировщикам.

- Через группу:



- После создания группы просто загружаем подписанный или отладочный APK-файл из консоли Firebase, при этом в примечание можно добавить любые заметки. На этом же этапе можно выбрать ранее созданную группу.

После добавления групп каждый участник получает приглашение по электронной почте.

В этом письме будет ссылка для загрузки приложения из **Firebase Distribution**. В этом приложении мы можем просматривать приложения, к которым у нас есть доступ, и смотреть версии приложений. При этом в консоли Firebase мы можем просматривать, кто принял приглашение и установил тестовую версию. Как мы видим, очень простой и быстрый способ.

Ещё бывают случаи, когда мы не знаем электронные адреса наших тестировщиков (допустим, когда мы запускаем открытый тест).

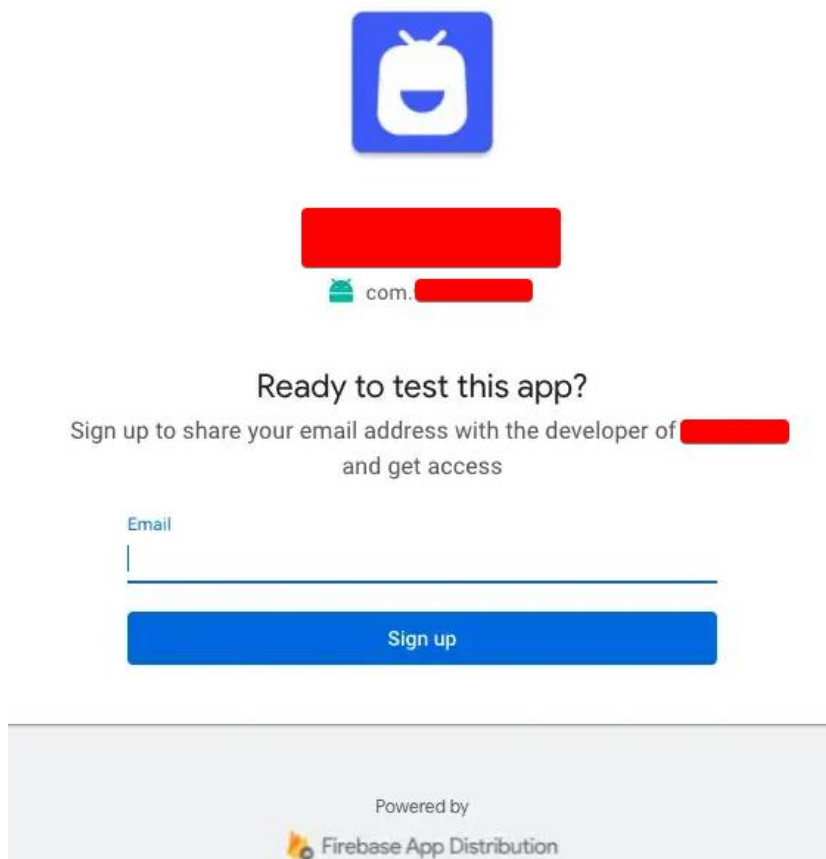
В таком случае мы прибегаем ко второму способу:

- Ссылка для скачивания приложения.

В этом случае мы точно так же загружаем APK-файл, потом нажимаем на кнопку **Создать ссылку-приглашение**.

Далее мы можем просто дать эту ссылку нашим потенциальным тестировщикам, либо размещать её на каком-либо ресурсе.

Когда пользователь перейдёт по этой ссылке, он увидит форму, в которой он может оставить свой адрес электронной почты.



Итоги предыдущих глав

Итак, теперь у нас всё готово для тестирования. Мы с вами:

- изучили требования для продукта
- выбрали тест-кейсы для тестирования
- выбрали необходимые устройства
- настроили дистрибуцию приложений

При выполнении тестов мы можем проводить наши тесты вручную, а можем настроить автоматизацию тестирования. Теперь разберём, в какой момент тестирование мобильных приложений стоит автоматизировать.

В какой момент стоит начинать автоматизацию

Как вы уже знаете, тестирование бывает ручное и автоматизированное. С мобильными приложениями всё то же самое. Существует большое количество фреймворков, которые помогают создавать автотесты для мобильных приложений. Эти фреймворки бывают кроссбраузерные (то есть можно писать тесты и для Android, и для iOS). А также существуют фреймворки, которые работают с одной платформой. Подробнее мы изучим их дальше.

Написание и поддержка автотестов — долгий и часто дорогой процесс. Поэтому нам надо понимать, целесообразно ли начинать заниматься автотестами.

- Если у нас небольшое приложение, которое находится в стадии активной разработки, то чаще нет смысла вводить автоматизацию.
- Если в приложении небольшой функционал, то количество тест-кейсов тоже небольшое, соответственно, ручное прохождение тест-кейсов занимает немного времени.
- А если приложение находится в активной стадии разработки с постоянно меняющимися требованиями, то нам придётся постоянно переписывать код для автотестов. То есть на этом этапе тоже не очень хорошо заниматься автоматизацией.
- Но если у нас приложение побольше, и к нему выходят регулярные обновления, то есть смысл задуматься об автоматизации.

Существует несколько предпосылок к тому, когда нам следует внедрять автоматизацию тестирования.

1. Цикл тестирования приложения становится всё больше с каждым разом

Если время прохождения тестов становится всё больше, и это влияет на время выполнения проекта, это признак того, что мы тратим слишком много времени на что-то повторяющееся, и логичным решением может быть внедрение автоматизации мобильного тестирования.

2. Увеличение внутреннего парка мобильных устройств

В одном из предыдущих пунктов мы разбирали, какие устройства нам нужны для тестирования. При этом с расширением требований к продукту нам может потребоваться всё больше мобильных устройств. То же может произойти, когда у нас растёт аудитория пользователей приложения, и нужно смотреть приложение на всех

популярных устройствах. В итоге это увеличивает затраты на внутреннюю лабораторию устройств. А время тестирования может вырасти в разы, либо нам нужно будет расширять команду тестирования. В этом случае автоматизация тестирования может помочь. Мы можем настроить облачную лабораторию, и благодаря ей можно будет настроить прогон автотестов сразу на всех необходимых устройствах. А некоторые сервисы даже позволяют пользоваться устройствами, которые предоставляет сам сервис.

3. Большое количество регрессионных тестов

В мобильном приложении количество регрессионных тестов может быть очень большим не только за счёт того, что растёт функционал нашего приложения, но и из-за того, что нам нужно прогонять большое количество разнообразных функциональных тестов перед релизом. К тому же их нужно прогонять на различных устройствах. Автоматизация в этом плане сильно сокращает время и затраты усилий команды.

4. Поддержка большого числа версий операционных систем

Возьмём, к примеру, Android. У андроида большое количество версий ОС, которыми сейчас пользуются. Наше приложение может поддерживать версии Android от 6 до 13. И чтобы убедиться, что наше устройство работает нормально на всех версиях (а ошибки, возникающие на разных версиях ОС, — распространённая проблема), нам нужно просмотреть все ключевые пункты приложения на всех версиях. В этот момент разумнее подключить автоматизацию.

Ручное тестирование лучше всего использовать при выполнении исследовательского тестирования, тестирования удобства использования и специального тестирования. Совместимость устройств и взаимодействие с пользовательским интерфейсом нельзя проверить с помощью автоматического тестирования.

Итак, мы прошли по основным моментам, которые являются особенными для процесса тестирования мобильных приложений. На следующем уроке мы подробно рассмотрим предрелизное тестирование, работу с мобильными сторами, пострелизное тестирование и поддержку.

Выводы

Сегодня мы:

- Подробно разобрали стратегию выбора устройств для тестирования

- Узнали, что такое матрица устройств
- Разобрали примеры тестов, которые необходимо добавлять в наши тест-кейсы
- Узнали про самые популярные сервисы дистрибуции и поближе познакомились с Firebase

Домашнее задание

Для закрепления материала мы предлагаем вам выполнить домашнее задание:

- Взять приложение «Почта» и посмотреть, какие виды тестирования можно применить к приложению.
- Используя StatCounter, подумать, на каких устройствах мы могли бы тестировать приложение (представим, что большинство целевой аудитории находится в России).

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

Что можно почитать

- [ISTQB CTFL Mobile Application Tester Syllabus](#) — базовая программа сертификации тестировщиков мобильных приложений, версия 2019

Используемая литература

- [Choose your Test Devices Wisely. Every mobile app is unique. So is the... | by Krishna Chaitanya | Medium](#)
- [Apple Developer](#)
- [Google Play Console](#)