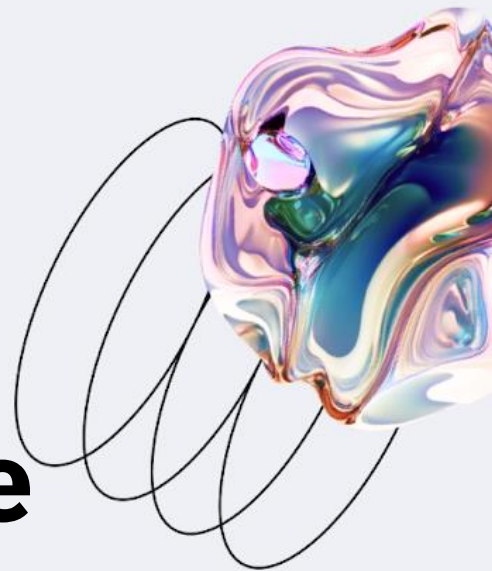


Функциональное тестирование

Процесс тестирования
мобильных приложений



Оглавление

На этом уроке	3
Термины, используемые в лекции	4
Составление тест-кейсов	5
Практика	6
Техники тест-дизайна	7
Классы эквивалентности	8
Граничные значения	9
Тестовая комбинаторика	10
Техника минимальных проверок	13
Перебор значений	13
Попарное тестирование	14
Проверки для мобильных приложений	16
Варианты прерывания приложения	16
Навигация	18
Уведомления	19
Разные условия сети	21
Разрешение	22
Отрабатывание жестов	23
Юзабилити-тестирование	25
Количество устройств для тестов	26
Исследовательское тестирование	26
Хранение тестовой документации и прогон тестов. Системы тест-менеджмента	27
Crashlytics	30
Что можно еще почитать?	33
Используемая литература	33

На этом уроке

На сегодняшнем уроке мы начнем составлять тест-кейсы для тестирования мобильного приложения.

- Рассмотрим примеры тест-кейсов для мобильного приложения.
- Вспомним техники тест-дизайна.
- Разберем разные функциональные тест-кейсы.
- Вспомним, как работать с системами тест-менеджмента.
- Познакомимся с Crashlytics.
- Повторим, что такое исследовательское тестирование.



Как мы знаем, есть два вида тестирования — функциональное и нефункциональное.


Функциональное тестирование проверяет, что каждая функция программы работает в соответствии с требованиями. В основном проходит по методу черного ящика и не касается исходного кода.

Нефункциональное тестирование проверяет нефункциональные аспекты приложения: производительность, надежность и так далее. Помогает убедиться, что приложение готово по нефункциональным параметрам, которые не учитываются при функциональном тестировании.

На этом уроке мы узнаем, как проводить функциональное тестирование. Сразу оговоримся: мы можем проводить его как вручную, так и с помощью автотестов. Но сейчас нам важно разобраться, как проводить его в целом, повторить основные моменты и уточнить детали, специфичные для мобильных приложений.

Не забывайте, что сначала принято проводить ручное тестирование с последующим составлением тест-кейсов, а после переходить к автоматизации. Мы, как и в реальном тестировании, начнем с составления тест-кейсов. Процесс не будет отличаться от аналогичного для любых сервисов и продуктов: мы проанализируем приложение и документацию, а затем составим тест-кейсы.

На помощь придут стандартные техники тест-дизайна: классы эквивалентности, граничные значения, тестовая комбинаторика. А также тест-кейсы, специфичные только для мобильных приложений.

 **Пример специфики:** мобильное приложение обрабатывает входящие вызовы и сообщения. Скорее всего, пользователи захотят, чтобы работа приложения приостановилась или запустилась в фоновом режиме, пока они отвечают на звонок или читают сообщение. Важно создать тест-кейсы для проверки таких ситуаций.

Как и в любом другом тестировании, при тестировании мобильного приложения важно определить критерии приемки и стараться их достичь. Важно составить тест-кейсы, которые помогут убедиться, что все функции работают корректно: например, поля ввода и вывода, сенсорные кнопки, навигация по приложению, датчик телефона.

Функциональное тестирование мобильного приложения выполняется так же, как и тестирование любого другого типа приложений: мы изучаем требования и составляем тест-кейсы. А еще добавляем специфичные для мобильных приложений проверки. О них поговорим на сегодняшнем уроке.

Термины, используемые в лекции

Функциональное тестирование проверяет, что каждая функция программы работает в соответствии требованиями. В основном проходит по методу черного ящика и не касается исходного кода.

Нефункциональное тестирование проверяет нефункциональные аспекты приложения: производительность, надежность и так далее. Помогает убедиться, что

приложение готово по нефункциональным параметрам, которые не учитываются при функциональном тестировании.

Классы эквивалентности — метод тестирования, при котором функционал или данные делятся на наборы, с которыми тестируемое приложение должно работать одинаково.

Граничные значения — значения, в которых один класс эквивалентности переходит в другой.

Комбинаторное тестирование (Combinatorial Testing) — метод выделения тестовых комбинаций. Пригодится, когда в программе много параметров, а внутри этих параметров много значений. Поможет сэкономить время при тестировании множества комбинаций.

Исследовательское тестирование — тип тестирования программного обеспечения, при котором тестировщики не составляют тест-кейсы заранее, а проверяют систему на лету. Идеи о том, что тестировать, могут записывать перед выполнением теста. Основное внимание уделяется тестированию как мыслительной деятельности.

Составление тест-кейсов

Вспомним, из чего состоит тест-кейс.

- **ID** помогает ссылаться на тест-кейс и делать привязки к баг-репортам. В системах тест-менеджмента, как правило, ID создаётся автоматически. Если тест-кейсы хранятся в Excel или текстовых документах, будет не так удобно: придется самостоятельно присваивать ID, что может вызвать путаницу. А если его не указывать, будет сложно ссылаться на конкретный кейс и искать его.
- **Приоритет теста:** низкий, средний или высокий — в зависимости от бизнес-требований. Часто отдел тестирования не может пройти все тест-кейсы перед релизом (например, если на этапе реализации функционала была задержка, а сроки релиза переносить нельзя). В этом случае выполняют только тесты с высоким приоритетом.
- **Название** должно описывать функционал, который проверяет тест. Обычно это цель или краткое описание тест-кейса.
- **Предусловия** — любые требования, которые должны быть выполнены для выполнения тест-кейса.
- **Тестовые данные** — данные, необходимые при выполнении теста.

- **Шаги** с подробным описанием выполнения тест-кейса.
- **Ожидаемый результат**, включая сообщения или ошибки, которые должны появиться на экране.

Не все поля обязательны для каждого проекта или команды и не у каждого тест-кейса могут быть предусловия или тестовые данные. Иногда они не нужны.

Бывает и наоборот: к тест-кейсам могут добавиться другие поля и компоненты. Например, поле зависимости, в котором указываются зависимости от других тест-кейсов.

Практика

Составим простой тест-кейс на часто встречающуюся функцию — **авторизацию**.

Название: успешная авторизация
Приоритет: высокий
Предусловия: приложение установлено на устройство. Пользователь не залогинен в приложении
Тестовые данные: логин — test, пароль — Testtest
Шаги: <ol style="list-style-type: none"> 1. Открыть приложение 2. В поле «Логин» ввести логин 3. В поле «Пароль» ввести пароль 4. Нажать кнопку Sign in
Ожидаемый результат: авторизация прошла успешно, открывается главный экран приложения

Мы также можем составлять тест-кейсы по принципу «результат после каждого шага»:

Название: успешная авторизация
Приоритет: высокий
Предусловия: приложение установлено на устройство. Пользователь не залогинен в приложении

Тестовые данные: логин — test, пароль — Testtest

Шаги:

1. Открыть приложение

Результат: откроется домашний экран с формой логина

2. В поле «Логин» ввести логин

Результат: логин можно ввести, введенный логин выглядит корректно

3. В поле «Пароль» ввести пароль

Результат: пароль можно ввести. Введенный пароль отображается звездочками

4. Нажать кнопку Sign in.

Результат: авторизация прошла успешно, открывается главный экран приложения

Как правило, полезно делать и **негативные тест-кейсы** — проверки потенциально ломающихся мест.

Негативные проверки для авторизации:

- Вход пользователя с неверным именем пользователя и неверным паролем.
- Вход пользователя с неверным именем пользователя и действительным паролем.
- Вход пользователя с действительным именем пользователя и неверным паролем.

Ожидаемые результаты в этих кейсах — пользователь видит корректные сообщения об ошибках.

Техники тест-дизайна

Тест-дизайн — это этап тестирования ПО, на котором проектируются и создаются тестовые случаи (тест-кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

Техники тест-дизайна помогают правильно составить набор тест-кейсов. Мы с ними уже знакомы, но, так как они критически важны и облегчают процесс тестирования, ещё раз вспомним основные из них:

- классы эквивалентности,
- граничные значения,
- тестовая комбинаторика: техника минимальных проверок, попарное тестирование, перебор значений.

Классы эквивалентности

Суть классов эквивалентности — разделение функционала или данных на наборы, с которыми тестируемое приложение должно работать одинаково.

Представим, что в окне регистрации нашего приложения есть поле для ввода возраста. Оно принимает значение от 18 до 100.

Возраст

Выберем значения, которые будем проверять, чтобы убедиться, что поле ввода работает корректно. Выделим несколько групп, в которых поведение приложения будет одинаковым для всех данных, то есть **эквивалентным**.

В нашем случае это группы:

1. до 18
2. 18–100
3. больше 100

При вводе данных из первой группы мы должны получить **ошибку**: данные не соответствуют ограничениям.

При вводе данных из второй группы **ошибки быть не должно**: данные должны корректно сохраниться.

Ввод данных из третьей группы также должен выдать **ошибку**.

Берем по одному значению из каждой группы:

- 2
- 56
- 115

У нас получается три группы. На их основе мы можем составить три тест-кейса.

💡 Правила техники классов эквивалентности:

- Если при тестировании одно значение из класса **выявит ошибку**, остальные тоже это сделают.
- Если при тестировании одно значение из класса **не выявит ошибку**, остальные тоже этого не сделают.

Главное при применении техники классов эквивалентности — выделить группы, данные в которых будут вести себя одинаково.

Это может быть:

- Поле для ввода чисел, где есть ограничения на ввод валидных значений.
- Поле для ввода текста, где есть ограничение на количество символов. Например, в комментариях к заказу.
- Список, где выбор любого значения приведет к одинаковому результату. Например, выбор марки автомобиля при составлении заявки на бронирование.
- Входное условие. Например, номер телефона должен начинаться с символа + или цифры. Мы можем сделать проверки на номер, который начинается с + (один класс эквивалентности), с цифры (второй), с буквы (третий, результатом будет ошибка), с любого другого символа кроме + (четвертый, результатом тоже будет ошибка).

Граничные значения

Граничные значения — это значения, в которых один класс эквивалентности переходит в другой. Анализ граничных значений продолжает технику классов эквивалентности, поэтому они используются вместе.

Рассмотрим на примере. Возьмем уже знакомое поле ввода возраста, которое принимает значение от 18 до 100.

Возраст

Поведение на границе класса эквивалентности с бóльшей вероятностью будет неправильным, чем поведение внутри группы. Границы — это область, где тестирование может выявить дефекты.

Выделим четыре граничных значения для тестирования: 17, 18, 100, 101.

Как мы уже говорили, граничные значения — это продолжение классов эквивалентности. То есть к уже имеющимся трем кейсам, мы прибавляем еще четыре граничных:

- 2
- 17
- 18
- 56
- 100
- 101
- 115

Таким образом, при проверке классов эквивалентности и граничных значений нам нужно 2 тест-кейса на каждую границу + 1 на каждый класс.

Тестовая комбинаторика

Комбинаторное тестирование (Combinatorial Testing) — метод выделения тестовых комбинаций. Пригодится, когда в программе много параметров, а внутри этих параметров много значений. Поможет сэкономить время при тестировании множества комбинаций.

Разберемся на примере. Представим, что нам нужно проверить, как работают фильтры поиска объявлений по мышкам и клавиатурам: все ли корректно при выборе разных значений фильтров, как фильтры работают друг с другом.

У нас есть параметры: фильтры по типу, марке, типу подключения, игровые. У каждого фильтра есть свои значения.

Есть локация (чтоб разобраться было проще — одна). Она подтягивается от местоположения телефона.

Есть поля с вводом минимальных и максимальных значений цены. Ограничений на цену нет. При вводе открывается числовая клавиатура, так что ввести можно только цифры.

Есть четыре переключателя: безопасная сделка, товары со скидкой, товары с доставкой, товары из магазинов. У этих переключателей может быть только два значения — да или нет.

✕ Фильтр	СБРОСИТЬ	← Тип	ВЫБРАТЬ	← Марка	ВЫБРАТЬ
Тип	Не указано >	Клавиатуры	<input type="checkbox"/>	Поиск	
Марка	Не указано >	Мыши	<input type="checkbox"/>	A4Tech	<input type="checkbox"/>
Тип подключения	Не указано >			Apple	<input type="checkbox"/>
Игровые	Не указано >			ASUS	<input type="checkbox"/>
Изменить местоположение				Defender	<input type="checkbox"/>
На расстоянии 50 км				Genius	<input type="checkbox"/>
Цена от Мин.	Цена до Макс.			HP	<input type="checkbox"/>
Безопасная сделка	<input type="checkbox"/>			Logitech	<input type="checkbox"/>
Товары со скидкой	<input type="checkbox"/>			Microsoft	<input type="checkbox"/>
Товары с доставкой	<input type="checkbox"/>			Oklick	<input type="checkbox"/>
Товары из магазинов	<input type="checkbox"/>			Razer	<input type="checkbox"/>
Показать 202 объявления				SteelSeries	<input type="checkbox"/>
По умолчанию		ОЧИСТИТЬ		Sven	<input type="checkbox"/>
				ОЧИСТИТЬ	

← Тип подключения	ВЫБРАТЬ	← Игровые	ВЫБРАТЬ	✕ Фильтр	СБРОСИТЬ
Проводные	<input type="checkbox"/>	Да	<input type="checkbox"/>	Тип подключения	Не указано >
Беспроводные	<input type="checkbox"/>	Нет	<input type="checkbox"/>	Игровые	Не указано >
				Изменить местоположение	
				На расстоянии 1 км	
				Цена от Мин.	Цена до Макс.
				Показать 4 объявления	
				1	2
				4	5
				7	8
				0	Далее
ОЧИСТИТЬ		ОЧИСТИТЬ			

Выделим список параметров, которые будем использовать при составлении тест-кейсов:

Параметр	Значение 1	Значение 2	Значение 3	Значение 4	Значение 5	Значение 6
Тип	Не указано	Клавиатуры	Мыши			
Марка	Не указано	A4Tech	Apple	ASUS	Defender	Genius
Тип подключения	Не указано	Проводные	Беспроводные			
Игровые	Не указано	Да	Нет			
Расстояние	1	16	больше 200			
Цена от	Не указано	0	2000			
Цена до	Не указано	800	9000			
Безопасная сделка	Да	Нет				
Товары со скидкой	Да	Нет				
Товары с доставкой	Да	Нет				
Товары из магазинов	Да	Нет				

Мы занесли значения фильтров в таблицу — так они выглядят удобнее и нагляднее. Для простоты выписали чуть меньше значений, чем есть в фильтре. В реальной работе нужно будет выписать всё.

Как видим, значений много. Давайте посмотрим, как мы можем применить на них разные техники тестовой комбинаторики.

Техника минимальных проверок

Техника минимальных проверок подразумевает, что по одному значению из набора тестовых данных должно быть использовано хотя бы в одном тест-кейсе.

Таким образом, результирующее количество кейсов будет равно количеству значений параметра с наибольшим диапазоном.

В нашем кейсе параметр с самым большим диапазоном — марка. Следовательно, число тест-кейсов будет равно количеству значений этого параметра. Как мы уже говорили, каждое значение каждого параметра мы должны использовать минимум один раз.

В итоге мы получим такие тест-кейсы:

Параметр	Значение 1	Значение 2	Значение 3	Значение 4	Значение 5	Значение 6
Тип	Не указано	Клавиатуры	Мыши	Не указано	Клавиатуры	Мыши
Марка	Не указано	A4Tech	Apple	ASUS	Defender	Genius
Тип подключения	Не указано	Проводные	Беспроводные	Не указано	Проводные	Беспроводные
Игровые	Не указано	Да	Нет	Не указано	Да	Нет
Расстояние	1	16	больше 200	1	16	больше 200
Цена от	Не указано	0	2000	Не указано	0	2000
Цена до	Не указано	800	9000	Не указано	800	9000
Безопасная сделка	Да	Нет	Да	Нет	Да	Нет
Товары со скидкой	Да	Нет	Да	Нет	Да	Нет
Товары с доставкой	Да	Нет	Да	Нет	Да	Нет
Товары из магазинов	Да	Нет	Да	Нет	Да	Нет

Каждая строка — это каждый кейс. Мы просто берем каждое значение, а оставшиеся поля заполняем рандомными данными из пула значений параметров.

Перебор значений

Перебор значений подразумевает тестирование всех комбинаций всех тестовых данных. Чтобы посчитать количество комбинаций, нужно перемножить количество всех значений всех параметров.

Перемножим количество комбинаций в нашем примере и получим:

$$3 * 6 * 3 * 3 * 3 * 3 * 3 * 2 * 2 * 2 * 2 = 69\,984$$

Конечно, мы не можем проверить такое количество кейсов — на это уйдет слишком много времени. Такой тип проверок используется, когда нужно проверить очень критичный функционал и когда число параметров и их значений совсем небольшое.

Попарное тестирование

Попарное тестирование — это метод, в котором тест-кейсы разрабатываются для выполнения всех возможных комбинаций каждой пары входных параметров.

То есть нам нужно перебрать каждое значение и хотя бы один раз протестировать его с каждым другим значением. Мы выбираем уникальные пары всех параметров и на их основе составляем набор тестовых данных.

Этот метод опирается на два принципа:

1. К большинству ошибок ПО приводит взаимодействие всего двух значений между собой (какой-то один компонент не работает с каким-то другим).

Дефекты проявляются скорее как сочетание двух параметров, чем трех и более — отсюда и получается попарное тестирование.

2. Каждый конкретный тестовый пример может проверять более одной уникальной пары. Мы получим очень маленькое и плотное подмножество тестовых случаев.

Вручную составлять тест-кейсы для техники попарного тестирования долго (весь алгоритм вы проходите при изучении техник тест-дизайна). Чтобы ускорить процесс, можно воспользоваться **Pairwise Tool** — инструментом для составления готовых тестовых наборов.

1. Зайдём на сайт [Pairwise Online Tool](#). Откроется страница, на которой нужно ввести данные:

The screenshot shows the Pairwise Tool interface. At the top, there are buttons: "Edit Conditions", "Generate Pairwise", "Generate All Combinations", and "Create Permalink". Below the buttons is a 3x3 grid for inputting data. The grid has a header row and a header column, both labeled "Column 1" and "Row 1" respectively. The cells contain placeholders like "{your value}" and symbols like "x" and "+".

	× Column 1	+
× Row 1	{your value}	+
+	+	+

2. Введём данные, которые уже вносили в таблицу:

The screenshot shows the Pairwise Tool interface with a 12x8 grid filled with test data. The grid has a header row and a header column, both labeled "Column 1" through "Column 7" and "Row 1" through "Row 11" respectively. The cells contain various test data points, including product names, features, and prices. The last row and column are labeled with "+" symbols.

	× Column 1	× Column 2	× Column 3	× Column 4	× Column 5	× Column 6	× Column 7	+
× Row 1	Тип	Не указано	Клавиатуры	Мыши	{your value}	{your value}	{your value}	+
× Row 2	Марка	Не указано	A4Tech	Apple	ASUS	Defender	Genius	+
× Row 3	Тип подключения	Не указано	Проводные	Беспроводные	{your value}	{your value}	{your value}	+
× Row 4	Игровые	Не указано	Да	Нет	{your value}	{your value}	{your value}	+
× Row 5	Расстояние	1	16	больше 200	{your value}	{your value}	{your value}	+
× Row 6	Цена от	Не указано	0	2000	{your value}	{your value}	{your value}	+
× Row 7	Цена до	Не указано	800	9000	{your value}	{your value}	{your value}	+
× Row 8	Безопасная сделка	Да	Нет	{your value}	{your value}	{your value}	{your value}	+
× Row 9	Товары со скидкой	Да	Нет	{your value}	{your value}	{your value}	{your value}	+
× Row 10	Товары с доставкой	Да	Нет	{your value}	{your value}	{your value}	{your value}	+
× Row 11	Товары из магазинов	Да	Нет	{your value}	{your value}	{your value}	{your value}	+
+	+	+	+	+	+	+	+	+

3. После заполнения данных нажимаем **Generate Pairwise**.

4. Скачается таблица с готовыми данными:

Тип	Не указано	Клавиатуры	Мыши	ASUS	Defender	Genius
Тип	Не указано	A4Tech	Apple	ASUS	Defender	Genius
Тип	Не указано	Проводные	Беспроводные	ASUS	Defender	Genius
Тип	Не указано	Да	Нет	ASUS	Defender	Genius
Тип	1	16	больше 200	ASUS	Defender	Genius
Тип	Не указано	0	2000	ASUS	Defender	Genius
Тип	Не указано	800	9000	ASUS	Defender	Genius
Тип	Да	Нет	Нет	ASUS	Defender	Genius
Марка	Не указано	Проводные	Нет	ASUS	Defender	Genius
Марка	Не указано	Да	больше 200	ASUS	Defender	Genius
Марка	Не указано	16	2000	ASUS	Defender	Genius
Марка	1	0	9000	ASUS	Defender	Genius
Марка	Не указано	800	Мыши	ASUS	Defender	Genius
Марка	Да	Нет	Мыши	ASUS	Defender	Genius
Марка	Да	Клавиатуры	Apple	ASUS	Defender	Genius
Марка	Не указано	A4Tech	Беспроводные	ASUS	Defender	Genius
Тип подключения	Не указано	16	9000	ASUS	Defender	Genius
Тип подключения	Не указано	0	Мыши	ASUS	Defender	Genius
Тип подключения	1	800	Apple	ASUS	Defender	Genius
Тип подключения	Да	Нет	Apple	ASUS	Defender	Genius

[Pairwise Pict Online](#) — похожий инструмент. Он немного сложнее, но позволяет добавлять разные условия для тестовых данных. Если вы с ним не работали, советуем познакомиться самостоятельно.

Подведём итоги раздела: важно понимать, когда применять ту или иную технику тест-дизайна: где-то достаточно техники минимальных проверок (для проверки некритичного функционала), где-то нужно проверять все данные, в том числе с перебором значений. Попарное тестирование тоже часто используется в работе.

Проверки для мобильных приложений

Мы вспомнили, как составлять тест-кейсы с помощью техник тест-дизайна. Теперь пройдемся по проверкам, специфичным для мобильных приложений:

1. [Варианты прерывания приложения](#)
2. [Навигация](#)
3. [Уведомления](#)
4. [Разные условия сети](#)
5. [Разрешения](#)
6. [Отрабатывание жестов](#)

Варианты прерывания приложения

Пользователи часто используют мобильные устройства в режиме многозадачности. Поэтому при тестировании важно проверить, как приложение справляется с прерываниями работы, переключением на другие приложения, блокировкой, выключением устройства и так далее.

Добавив эти кейсы в тестовые наборы, мы сможем избежать плохих отзывов и возможного удаления приложения пользователями.

Что нужно сделать при тестировании:

1. Переключиться на другое приложение и вернуться к нашему, чтобы посмотреть, как оно реагирует.
2. Ответить на входящий вызов или отклонить его во время использования приложения. Убедиться, что приложение не выйдет из строя или не перезапустится.
3. Нажать на push-уведомление другого приложения и переключиться на него, чтобы проверить, вернется ли приложение в то же состояние и на тот же экран.
4. Если приложение требует подключения к Bluetooth-устройству или другому оборудованию, отключить устройство и посмотреть, как приложение справляется с прерыванием.
5. Закрыть приложение во время совершения покупки и снова открыть его. Приложение должно показать, какой была транзакция: успешной или неудачной.
6. Установить будильник на время использования приложения. Проверить, что приложение не зависает.
7. Оставить устройство с запущенным приложением, пока оно не заблокируется или не перейдет в спящий режим. После разблокировки устройства приложение должно быть в том же состоянии, в котором его оставили.
8. Использовать приложение, когда батарея приближается к порогу низкого заряда. Убедиться, что приложение не зависнет, когда мыотреагируем на нотиф (уведомление, от англ. notification) о низком заряде: перейдем в режим энергосбережения или просто закроем уведомление.

Составим тест-кейс на переключение приложения.

Название: корректное возобновление работы после переключения на другое приложение

Приоритет: высокий

Предусловия: приложение установлено на устройство.

Шаги:

1. Открыть приложение
2. Перейти в раздел «Мои заказы»
3. Свернуть приложение
4. Зайти в телефонную книгу
5. Вернуться в приложение

Ожидаемый результат: приложение корректно возвращается после сворачивания. Приложение остается в том же разделе, в котором было до сворачивания

Навигация

Важно убедиться, что пользователь может без проблем переходить между экранами приложения.

Что нужно сделать при тестировании:

1. Проверить, что во всех нужных модулях приложения есть прокрутка (например, прокрутка ленты сообщений или длинного списка меню). Убедиться, что она работает корректно.
2. Проверить, что навигация между модулями в приложении соответствует требованиям.
3. Проверить, что на всех нужных экранах есть кнопка «Назад» и что пользователь может вернуться к предыдущему модулю.
4. Проверить, что на всех нужных экранах есть кнопка «Меню» и пользователь может перейти в любой из доступных в меню разделов.

Подсказка по названию разных вариантов иконок меню:

Menu Icons



Hamburger



Oreos



Kebab



Candy Box



Chocolate



Cheese Burger



Hot Dog



Veggie Burger



Strawberry



Fries



Stairs



Cake

Источник: [weareconflux](https://weareconflux.com/)

Подробнее про типы меню — в статье [The Ultimate Guide to the Hamburger Menu and Its Alternatives | by Apptimize | UX Planet](#).



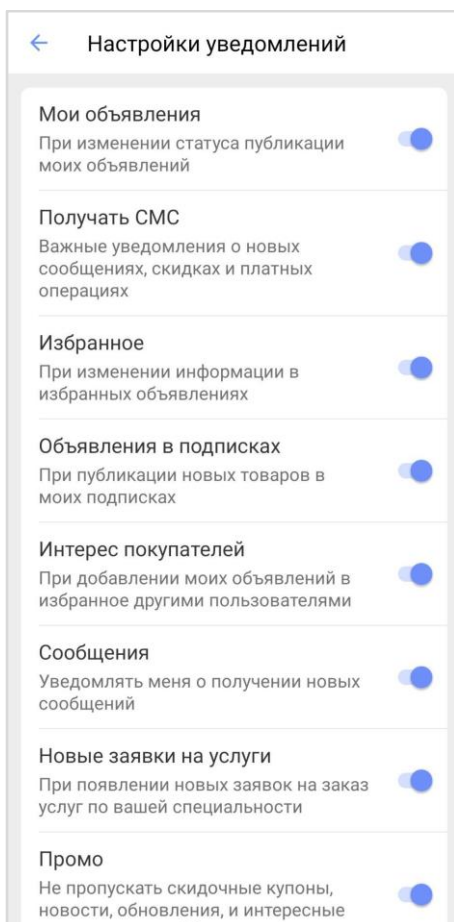
На английском языке есть много хороших источников информации для IT-специалистов. Даже если вы поверхностно знаете язык или не знаете его совсем, это не должно вас останавливать. Встроенный в браузер переводчик или сервисы от [Яндекса](#) или [Google](#) смогут помочь.

Уведомления

Сложно представить мобильное приложение без уведомлений. Магазины присылают уведомления об акциях и статусах заказа, мессенджеры — о новых сообщениях, игры — о новых уровнях и так далее.

Уведомления бывают нескольких типов. Одни приходят с сервера, другие — по таймеру. Во втором случае неважно, есть ли на устройстве интернет.

Кроме того, уведомления могут относиться к разным категориям. Например, на скриншоте мы видим разные категории уведомлений в приложении объявлений:



Хорошая статья о том, какими бывают уведомления и из чего они состоят: [Подробный гайд по мобильным пуш-уведомлениям | Блог MyTracker.](#)

Что нужно сделать при тестировании уведомлений в зависимости от реализации:

1. Проверить, что уведомление приходит, если приложение не запущено.
2. Проверить, что сообщение и заголовок, которые отображаются в уведомлении, соответствуют требованиям.

3. Проверить, что уведомление приходит и корректно отображается, когда приложение запущено (если это требуется).
4. Проверить уведомление, когда пользователь не вошел в приложение. Для этого нужно выйти из аккаунта и попробовать отправить нотиф на устройство, где пользователь не залогинен.
5. Проверить работу уведомления при отсутствии интернета или после его появления.
6. Если есть требование на звук и вибрацию в уведомлении, убедиться, что они воспроизводятся.
7. Проверить, что уведомление корректно отображается с большим количеством текста или изображениями.
8. Проверить, что уведомление корректно отображается, когда телефон заблокирован.
9. Проверить значок уведомлений на иконке приложения.
10. Проверить, что при получении нескольких уведомлений счетчик уведомлений корректно работает (если это предусмотрено в требованиях).
11. Проверить, правильно ли работают настройки уведомлений, могут ли пользователи настраивать их в соответствии со своими предпочтениями.
12. Проверить, могут ли пользователи взаимодействовать с интерактивными уведомлениями и открывать соответствующий раздел приложения по клику.
13. Проверить, сохраняет ли приложение историю всех полученных уведомлений, может ли пользователь получить к ней доступ.
14. Проверить, запрашивает ли приложение права для отправки и получения уведомлений, работают ли они правильно.

Составим тест-кейс на получение уведомления:

Название: получение уведомления при получении нового сообщения

Приоритет: высокий

Предусловия: приложение установлено на устройство. Приложение свернуто. Подготовлен другой аккаунт, с которого можно отправить сообщение

Тестовые данные: логин — test, пароль — Testtest

Шаги:

1. Используя второй аккаунт, отправить сообщение на тестируемое устройство
2. Проверить уведомление о входящем сообщении

Ожидаемый результат: уведомление выглядит корректно, соответствует макету. При получении уведомления проигрывается звук и вибрация

Разные условия сети

Часто приложение завершает работу и крешит из-за проблем с сетью. При этом отсутствие подключения или переключение между вайфаем и мобильным интернетом — частые пользовательские ситуации.

Чтобы убедиться, что наше приложение корректно отработает, нужно проверить:

1. Переключение в режим полета во время работы приложения. Еще одна хорошая проверка — включение режима полета во время скачивания файлов или загрузки приложения.
2. Изменение сети с 4G на 3G (5G) и наоборот.
3. Переключение с вайфая на мобильный интернет и наоборот. Полезно проделать несколько проверок в разных состояниях приложения.
4. Проверить, что приложение корректно восстанавливает работу после появления соединения.
5. Проверить, что при хорошем соединении приложение корректно обрабатывает большое количество данных.

Составим тест-кейс на отключение интернет-соединения.

Название: запуск приложения при отсутствии интернет-соединения

Приоритет: высокий

Предусловия: приложение установлено на устройство. Приложение не запущено

Шаги:

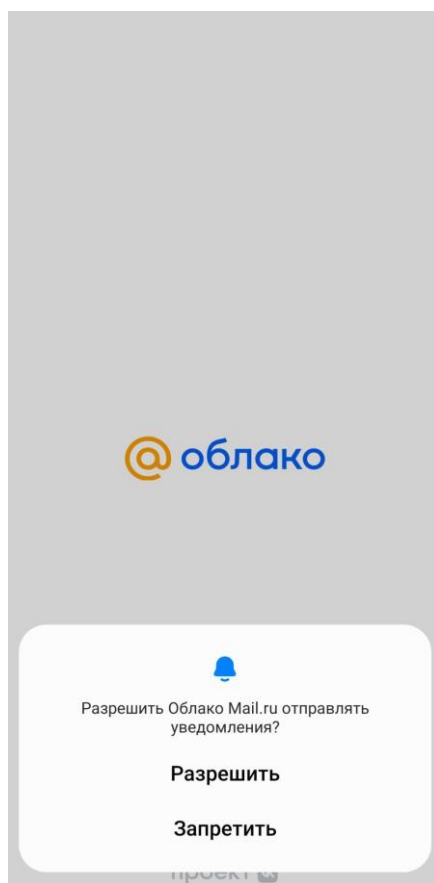
1. Включить режим полета на устройстве
2. Запустить приложение

Ожидаемый результат: появляется сообщение «Проверьте интернет-соединение» и кнопка «Обновить»

Важно! Все примеры тест-кейсов — только примеры. В зависимости от требований и документации будут разные ожидаемые результаты.

Разрешение

Современные приложения могут запрашивать много разрешений (пермишенов, от англ. permission): например, на доступ к местоположению, хранилищу устройства, камере, звонкам и многому другому.



Важно проверить, что приложение не запрашивает лишних пермишенов. Если запросов много и непонятно, для чего они нужны, пользователь может отклонить разрешение на использование тех или иных функций.

Другая важная проверка — как поведет себя приложение, если пользователь не дает разрешение на использование функций телефона. В этом случае пользователь должен получить сообщение, что определенные функции приложения ему будут недоступны.

Что нужно сделать при тестировании:

1. Проверить, что приложение запрашивает только необходимые пермишены.
2. Проверить, что приложение объясняет, зачем нужно то или иное разрешение (в соответствии с рекомендациями Android или iOS).
3. Проверить, что приложение корректно обрабатывает ситуации, когда пользователь не дает доступ к функции. Корректные сценарии обработки должны быть предусмотрены требованиями.

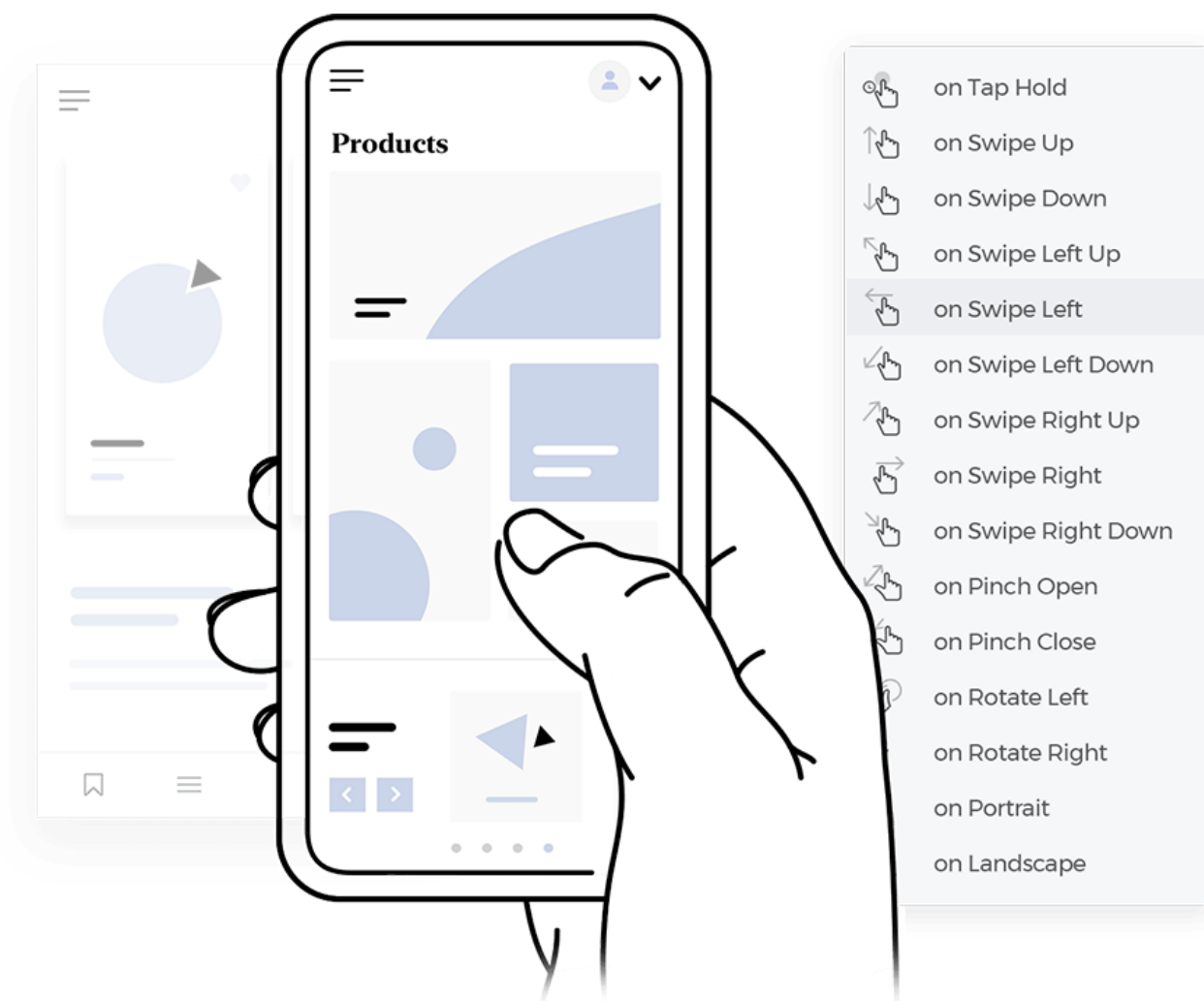
Пример — запретить приложению использовать камеру и вызвать функцию приложения, которая использует камеру. Из пермисшена должно быть понятно, зачем мы его запрашиваем. Обычно пояснение реализуют окном внутри приложения с комментарием. Дизайны и текст могут быть разными — зависит от документации к продукту.

4. Проверить, что если пользователь сначала отказался давать пермишен, а потом разрешил, приложение может корректно использовать функции.

Обработка жестов

Что нужно сделать при тестировании:

1. Проверить, что скролл работает в нужных разделах.
2. Проверить, что зум работает в нужных разделах: например, товар в магазине увеличивается и уменьшается.
3. Проверить, что если приложение поддерживает мультитач, пользователь может корректно им пользоваться.
4. Проверить, что приложение корректно обрабатывает долгие и короткие нажатия.



Источник: [justinmind](https://justinmind.io/)

Статья о типах жестов с описаниями и примерами хорошей реализации: [Mobile navigation: patterns and examples - Justinmind](https://justinmind.io/en/mobile-navigation-patterns-and-examples/)

Составим тест-кейс на скролл ленты новостей.

Название: скролл ленты новостей

Приоритет: высокий

Предусловия: приложение установлено на устройство. Пользователь подписан на несколько новостных каналов

Шаги:

1. Открыть приложение
2. Перейти в раздел «Новости»
3. Проскроллить новости

Ожидаемый результат: скролл работает корректно, при пролистывании подтягиваются новые новости

Юзабилити-тестирование

Юзабилити-тестирование — это тестирование удобства интерфейса для пользователя.

При тестировании обращаем внимание на несколько аспектов:

- **Навигация.** Проверяем, интуитивно ли она понятна для пользователя, последовательна ли на всех экранах. Для проверки опираемся на требования, а если есть места, которые нас смущают, — общаемся с ответственными за дизайн командами.

Навигация не должна сбивать с толку, в приложении должна быть возможность вернуться или отменить действие, если пользователь нажал не ту кнопку.

- **Интерфейс пользователя.** Проверяем, выглядит ли интерфейс визуально привлекательным, легкочитаемым и согласованным с рекомендациями по дизайну.

Кнопки должны располагаться в одной области экрана, у них должен быть нормальный размер и, если они выполняют одну функцию, — одинаковый цвет. Текст не должен быть слишком мелким, чтобы пользователь мог без проблем его прочитать.

- **Обработка ошибок.** Проверяем, показывает ли приложение четкие сообщения об ошибках, помогает ли пользователям быстро их устранять.
- **Обратная связь с пользователем.** Проверяем, дает ли приложение пользователям четкую обратную связь об их действиях: например, визуальные подсказки или сообщения о подтверждении.
- **Доступность.** Проверяем, доступно ли приложение всем пользователям, включая людей с ограниченными возможностями, в том числе с нарушением зрения.
- **Пользовательский опыт.** Проверяем, дает ли приложение положительный и приятный пользовательский опыт, соответствует ли он потребностям целевой аудитории. То есть может ли пользователь легко и без лишних проблем

воспользоваться функционалом приложения и выполнить все нужные действия.

- **Руководство по удобству использования.** Проверяем, что приложение соответствует руководствам по удобству использования мобильных приложений (например, руководствам [Apple](#) и [Google](#)).

💡 Помимо перечисленных тестов важно убедиться, что приложение корректно выполняет все свои функции, что все реализовано по требованиям.

Мы изучаем требования и составляем тест-кейсы, которые покрывают каждое требование из задач/документации.

Количество устройств для тестов

Как правило, функционал в командах полностью смотрят на одном-двух устройствах (для каждой ОС). При этом должны быть чек-листы, которые тестировщики проходят на большем количестве устройств.

Во втором случае мы смотрим, что приложение в целом работает корректно, что девайсозависимые функции (например, уведомления) работают на каждой интересующей нас версии ОС, проверяем визуал приложения на всех имеющихся разрешениях экранов.

Исследовательское тестирование

При составлении всех типов тест-кейсов полезно применять креативное мышление. Конечно, это не системный подход к тестированию, но и баги в приложениях часто возникают не систематически.

Исследовательское тестирование — это тип тестирования программного обеспечения, при котором тестировщики не составляют тест-кейсы заранее, а проверяют систему на лету. Они могут записывать идеи о том, что тестировать, перед выполнением теста. Основное внимание уделяется тестированию как мыслительной деятельности.

В исследовательском тестировании многое зависит от навыков и опыта тестировщика. Но даже недолгое проведение исследовательского тестирования помимо системного полезно.

Подход к исследовательскому тестированию довольно простой — нужно отойти от уже составленных тестовых сценариев и попытаться сломать приложение. Попробуйте придумать пограничные случаи, которые, скорее всего, и произойдут, когда множество людей будут использовать приложение иначе, чем ожидает команда.

Возьмем для примера приложение интернет-магазина. Представим, что нам нужно протестировать кнопку добавления в корзину, оформление заказа и варианты связи со службой поддержки клиентов.

1. Исследовательский тест может начаться так же, как сценарий обычного пользователя — с просмотра товара.
2. Мы можем выбрать товары и добавить их в корзину.
3. Переключаемся к тестированию всех функций корзины, включая добавление новых товаров, редактирование количества, удаление из корзины и переход к оформлению заказа.
4. Проходим по сценариям вроде постоянных отмен и возвращений к предыдущим экранам на каждом этапе заказа.
5. Проверяем отключение интернета в момент оплаты и сворачивания приложения.

Хранение тестовой документации и прогон тестов. Системы тест-менеджмента

В рамках курса вы уже работали с системами тест-менеджмента. Давайте освежим знания и оформим созданные тест-кейсы в системе тест-менеджмента qase.io.

Глобально все TMS (test management system) работают по одному принципу.

1. Создаем новый проект:

Create new project

Project name *

Mobile Testing

Project code * ?

MT

Description

Write a few sentences about your project

Project access type

☒ Private

☐ Public

Member access

☒ Add all members to this project

☐ Add members from specific group

☐ Don't add members

Cancel

Create project

2. Заполняем репозиторий тест-кейсами:

MP repository

1 suite | 1 test

+ Suite

+ Case

Search for cases

+ Add filter

Suites

Авторизация 1 ...

Авторизация + ✎ 📄 🗑

↑ 🖐 MP-1 Успешная авторизация

+ Create quick test

Тест-кейсы (case) можно объединять в тест-сьюты (suite) — тестовые наборы. Как правило, в один тестовый набор входят все тест-кейсы, которые относятся к одному модулю (блоку, разделу) приложения.

Например, можно создать suite «Авторизация» и внести в него все релевантные кейсы: регистрацию нового пользователя, вход с корректными данными, попытки неуспешного входа с некорректными данными. При составлении тест-кейсов мы заполняем все поля, нужные на проекте.

Успешная авторизация

Edit

Clone

Delete

AI Test Case

General

Properties

Runs

History

Defects

Приложение установлено на устройство.

Пользователь не залогинен в приложении

Post-conditions

Not set

Parameters

логин

○ test

пароль

○ Testtest

Steps

Shown as table ☒

	Action	Data	Expected result	Att
○ 1	Открыть приложение		Откроется домашний экран с формой логина	
○ 2	В поле Логин ввести логин		Логин можно ввести, введенный логин выглядит корректно	
○ 3	В поле пароль ввести пароль		Пароль можно ввести. Введенный пароль отображается звездочками	
○ 4	Нажать кнопку "Sign in".		Авторизация прошла успешно, открывается главный экран	

3. После заполнения тест-кейсов, когда нужно перейти к этапу прохождения кейсов, создаем Test Run (тест-ран, тестовый прогон):

Start a new test run

Manual

Test run 2023/04/10

Description

For example: we can authorize on page https://example.com

Environment

Not set

Milestone

Not set

Default assignee

Select value

Tags

Select value

Tests

0 test cases will be added to the run

Choose tests from test plan

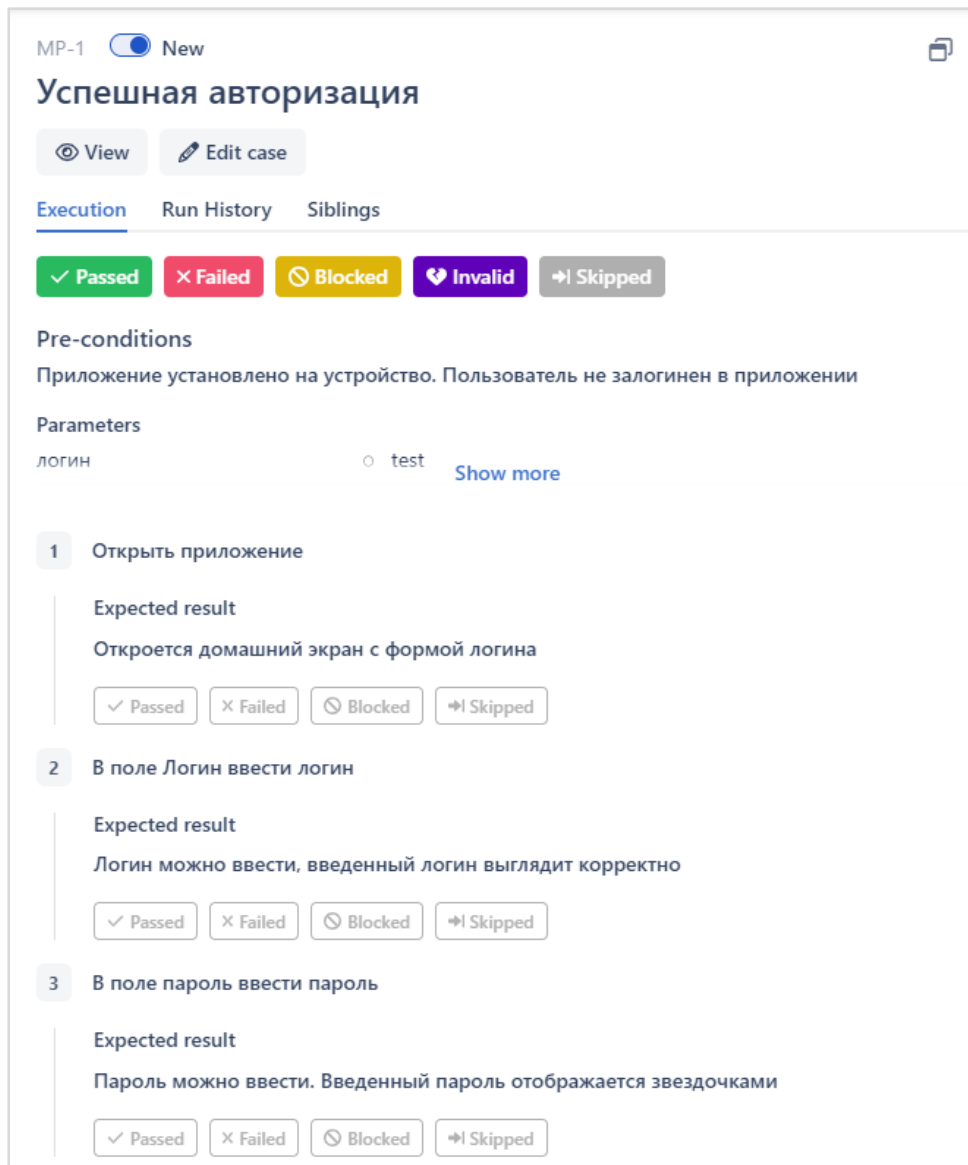
+ Add/modify tests from repository

Cancel

Start a run

При создании тестового прогона нужно выбрать, какие тест-кейсы будут в него включены.

4. После создания тест-рана мы можем приступить к прохождению тест-кейсов.



Мы можем ставить статус как для каждого шага, так и для всего тест-кейса:

- **passed** — успешно пройдено,
- **failed** — фактический результат не совпал с ожидаемым, есть баг,
- **blocked** — прохождение тест-кейса/шага заблокировано,
- **skipped** — прохождение тест-кейса/шага пропущено.

Crashlytics

Часто во время тестирования наше тестовое приложение может крешить. И не всегда мы с легкостью можем воспроизвести этот момент, хотя важно заметить паттерн поведения приложения и подробно описать его.

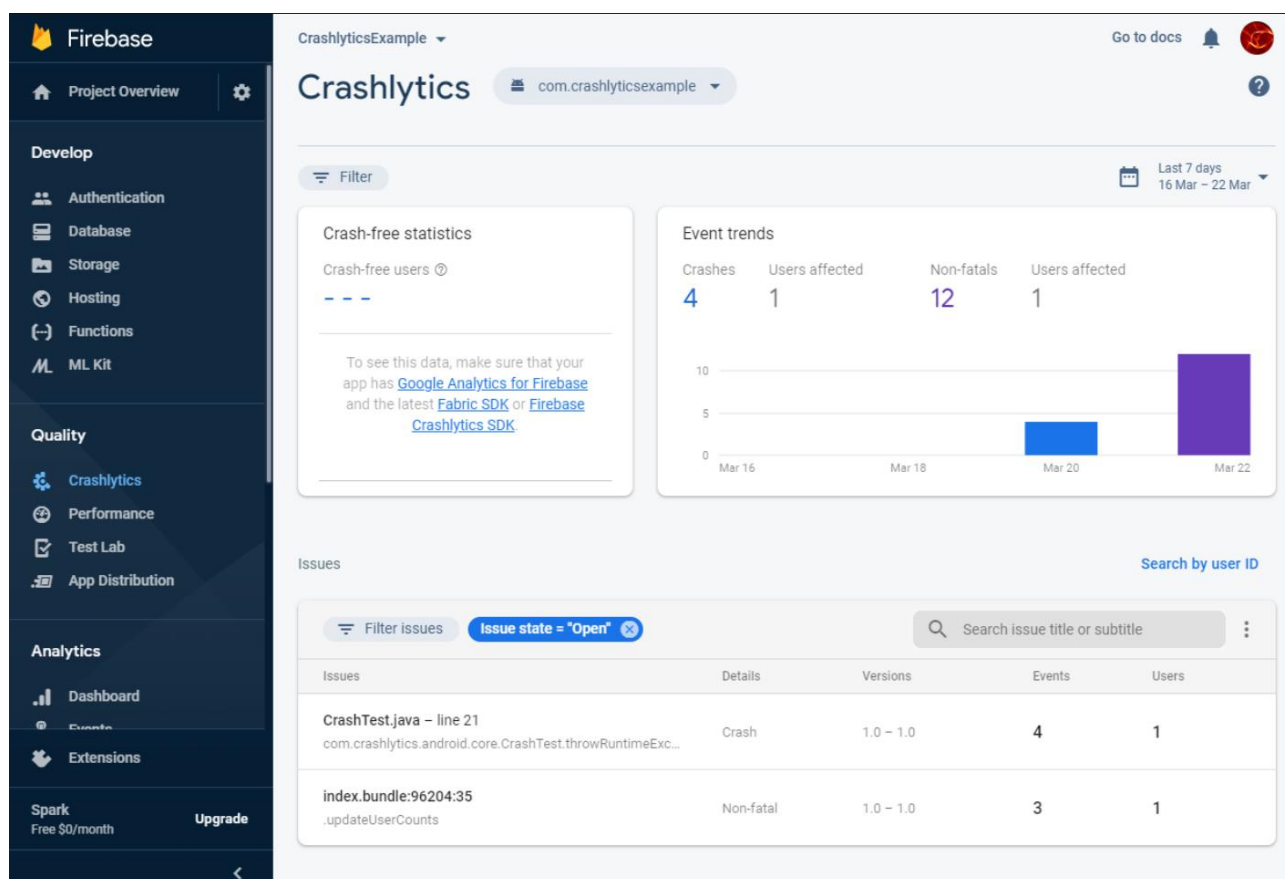
Обычно для анализа проблем и более быстрой правки разработчикам помогают логи, которые хранят информацию о том, какой раздел приложения вызвал сбой.

Мы уже умеем снимать логи, когда приложение подключено к компьютеру (для Android мы используем ABD или Android Studio, для iOS — Xcode). Но во время тестирования приложения оно есть не всегда. В этом случае могут помочь инструменты, которые отслеживают крэши и собирают всю необходимую информацию.

Частый сценарий использования таких инструментов: если приложение уже доступно пользователям, можно смотреть статистику сбоев и информацию по крэшам пользователей. Один из самых популярных инструментов для этого — Crashlytics.

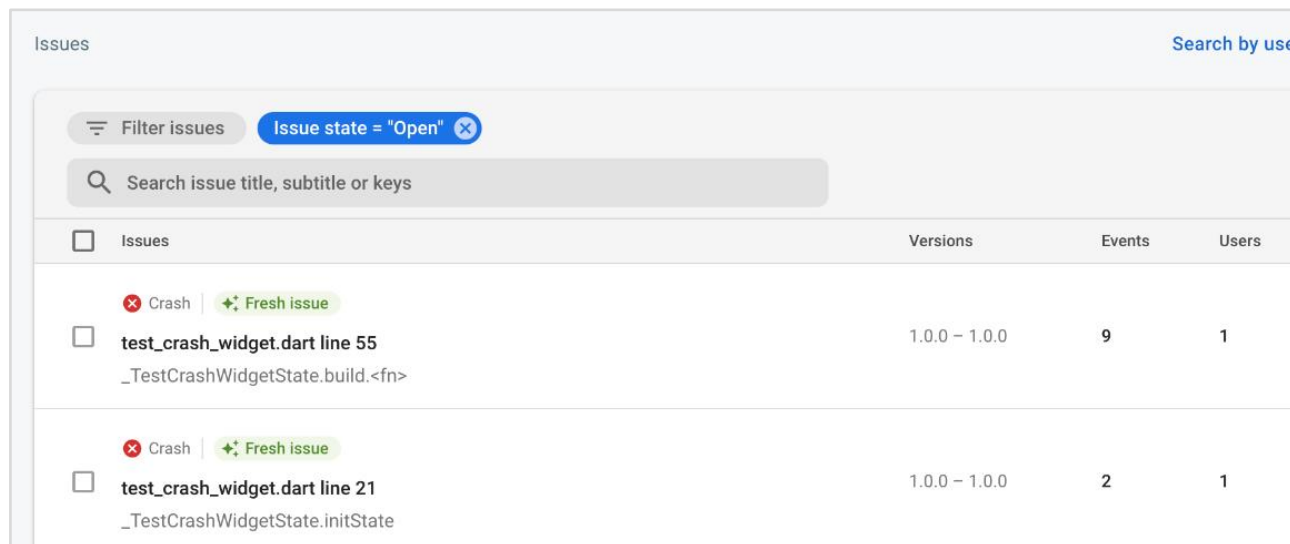
Crashlytics — это бесплатная облачная служба отчетов о сбоях для приложений iOS и Android. Она автоматически создает простые, но информативные отчеты, которые содержат строку кода, вызывающую сбой, а также подробные данные о тестовой среде: устройство, режим, в котором произошел сбой, и так далее. Так что не придется тратить много времени на поиск корня проблемы.

Crashlytics — часть уже знакомого нам Firebase. В консоли он находится в разделе «Качество».



Crashlytics показывает все зарегистрированные ошибки/сбои во время тестирования и у пользователей. Для каждого сбоя есть детали. Можно посмотреть строки кода, на которых приложение дало сбой. Все сбои и крэши можно фильтровать по версии приложения, устройствам, ID пользователя и другим критериям.

Если наше приложение крэшит, мы можем приложить ссылку на крэшлог к дефекту. Крэшлог — это основная информация о том, что происходило в приложении до того, как произошел сбой и в момент сбоя. Крэшлоги помогают разработчикам локализовать проблему и быстрее ее исправить.



Issues Search by use				
<div>Filter issues Issue state = "Open" X</div> <div>Search issue title, subtitle or keys</div>				
<input type="checkbox"/> Issues		Versions	Events	Users
<input type="checkbox"/> Crash Fresh issue test_crash_widget.dart line 55 _TestCrashWidgetState.build.<fn>		1.0.0 – 1.0.0	9	1
<input type="checkbox"/> Crash Fresh issue test_crash_widget.dart line 21 _TestCrashWidgetState.initState		1.0.0 – 1.0.0	2	1

💡 Когда приложение крэшит, мы берем крэшлог и прикладываем его к баг-репорту.

Выводы

Сегодня мы:

- Повторили виды тестирования и обратили особое внимание на функциональные, так как начинаем тестирование именно с них.
- Вспомнили техники тест-дизайна.
- Повторили процесс составления тест кейсов, а также подробно рассмотрели виды проверок, специфичных для мобильных устройств:
 - [Варианты прерывания приложения](#)
 - [Навигация](#)
 - [Уведомления](#)

- [Разные условия сети](#)
- [Разрешения](#)
- [Отрабатывание жестов](#)
- [Юзабилити](#)
- Повторили флоу работы с TMS.
- Вспомнили о важности исследовательского тестирования.
- Узнали, как снимать крешлоги с помощью Crashlytics.

На следующем уроке мы рассмотрим нефункциональное тестирование и познакомимся с инструментами, которые помогут проводить нефункциональные тесты.

Домашнее задание

Возьмите приложение социальной сети (VK, LinkedIn и др.) и попробуйте обратить внимание навигацию, работу уведомлений, разрешений, жестов и в целом удобство использования. Варианты проверок можно найти в конспекте к лекции.

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

Что можно еще почитать?

- «Практическое руководство по тест-дизайну», Ли Коуплэнд — книга поможет лучше запомнить техники тест-дизайна.

Используемая литература

- [Firebase Crashlytics | A powerful Android and iOS crash reporting solution](#)
- [ISTQB](#)