Geekbrains

**Проведение функционального и нефункционального ручного тестирования мобильного приложения ресторана “Rostic’s” в условиях отсутствия технического задания.**

Разработчик

Тестирование мобильных приложений

Тимошенко Максим Владимирович

Ростов-на-Дону

2024

**Содержание**

Введение……………………………………………………………………………...3

Глава 1. Основы ручного тестирования…………………………………………… 6

1.1 Что такое мобильное приложение, его особенности…………………………..6

1.2 Что такое тестирование, кто такой тестировщик. Зачем нужно тестирование и когда оно начинается.……………………………………………………………………………9

1.3 Этапы ручного тестирования мобильных приложений………………………16

1.4 Выбор устройства для тестирования мобильных приложений………………18

1.5 Источники требований: что такое техническое задание и как проводить ручное тестирование, если техническое задание отсутствует…………………………….19

Глава 2. Подготовка к ручному тестированию мобильного приложения…………………………………………………………………………..24

2.1 Составление чек-листов…………………………………………………………24

2.2 Составление тест-кейсов………………………………………………………..30

2.3 Разработка тест-плана…………………………………………………………...34

2.4 Тест-дизайн и тест-аналитика…………………………………………………..40

2.5 Баг-репорты и багтрекинговая система ………………………………………..50

Глава 3. Проведение тестирования и разработка предложений по улучшению тестирования ресторана Rostic’s……………………………………………………58

3.1 Функциональное тестирование…………………………………………………58

3.1.1 Варианты прерывания приложения…………………………………………..58

3.1.2 Тестирование интерфейса……………………………………………………..59

3.1.3 Навигация………………………………………………………………………67

3.1.4 Уведомления……………………………………………………………………68

3.1.5 Разные условия сети…………………………………………………………...69

3.1.6 Разрешения……………………………………………………………………..70

3.1.7 Отрабатывание жестов………………………………………………………...72

3.2 Нефункциональное тестирование……………………………………………....73

3.2.1 Тестирование установки……………………………………………………....73

3.2.2 Тестирование безопасности……………………………………………..........75

3.2.3 Тестирование локализации и глобализации……………………………........76

3.2.4 Тестирование совместимости…………………………………………….......77

3.3 Подведение итогов. Разработка предложений по улучшению ручного тестирования мобильного приложения Rostic’s……………………………………………...........78

# Введение

Мобильные устройства играют немаловажную роль в современном мире. Сейчас, практически у каждого есть сотовый телефон. С его помощью можно звонить, слушать музыку, смотреть видео, играть, читать и даже заключать сделки в крупных масштабах.

Люди большое количество времени проводят не с семьёй, не с друзьями, а за телефоном. Но в наше время, телефон действительно является неотъемлемой частью нашей жизни.

Самыми распространенными платформами являются: Android; IOS и Windows Mobile.

Мобильный интернет. Востребованность мобильного интернета растет стремительными темпами. Если в 2015 году процент всех пользователей мобильных телефонов,которые получают доступ к интернету со своего карманного устройства составил 35.1%,то уже в 2019 году процент пользователей составил 63.5% [по данным проекта Statista](https://dzen.ru/away?to=https%3A%2F%2Fsdvv.ru%2Farticles%2Felektronnaya-kommertsiya%2Fstatistika-interneta-2019-trafik-sayty-i-blogi-domeny-sotsialnye-media-onlayn-reklama-i-elektronnaya%2F)

Мобильные приложения. Раньше основной платформой были сайты на ПК, а мобильный трафик считался просто одним из дополнительных каналов получения прибыли. Сейчас 50 % продаж генерируют приложения. Для большого количества клиентов различных сервисов, разработка приложений для iPhone, iPad и Android играет большое значение. Именно с их помощью появляется возможность пользоваться теми или иными услугами, совершать покупки без особых усилий и траты времени. Например, каким бы популярным и солидным не был банк, обслуживающий платежные карты, в наше время он будет значительно проигрывать тому, кто взял на вооружение мобильное приложение. Поэтому постепенно растет число компаний, которые понимают ценность и значимость приложений для смартфонов. Если игнорировать этот вопрос, вероятно, очень скоро компания потеряет огромное количество потенциальных клиентов.

Большинство обладателей современных мобильных устройств также используют приложения как элемент досуга в повседневной жизни. Они пользуются приложениями для общения, игр, получения развлекательной информации и т.д. [В App Annie](https://dzen.ru/away?to=https%3A%2F%2Fwww.web-canape.ru%2Fbusiness%2Finternet-2020-globalnaya-statistika-i-trendy%2F) заявляют, что на мобильные приложения теперь приходится 10 из каждых 11 минут пользования мобильным устройством, а на просмотр веб-страниц уходит только 9% нашего «мобильного времени. За прошлые 12 месяцев пользователи смартфонов загрузили более 200 миллиардов мобильных приложений, потратив совокупно 120 миллиардов долларов на приложения и покупки в них [сообщает App Annie.](https://dzen.ru/away?to=https%3A%2F%2Fwww.web-canape.ru%2Fbusiness%2Finternet-2020-globalnaya-statistika-i-trendy%2F)

Продукты питания и товары, а также готовая еда – самые популярные у потребителей категории товаров в доставке. 54% опрошенных потребителей заказывают доставку продуктов питания и товаров FMCG, 42% – доставку готовой еды из кафе и ресторанов.

Цели:

В результате выполнения дипломной работы первоочередными целями стоят закрепить знания и навыки по тестирования мобильных приложений.

Закрепить практические навыки:

1.Провести ручное тестирование мобильного приложения «Rostic’s».

2.Составить чек-лист для проверки приложения.

3.Провести функциональное и нефункциональное тестирование приложения.

4.Составить отчёты о тестировании.

Описание приложения:

Сервис по заказу и доставки еды

Функциональные возможности приложения Rostic’s:

1. Функция регистрации и возможность изменения пользователями своих профилей. Пользователи должны иметь возможность зарегистрироваться и создать свой профиль в приложении, чтобы оформлять заказы и отслеживать их статус.
2. Функция выбора блюд. Пользователи должны иметь возможность выбирать блюда из меню, просматривать информацию о каждом блюде.
3. Функция оформления заказа. Пользователи должны иметь возможность оформить заказ, выбрав блюда из меню, указав адрес доставки и способ оплаты;
4. Возможность отслеживания статуса заказа: пользователи должны иметь возможность отслеживать статус своего заказа, начиная от подтверждения и оплаты до доставки;
5. Онлайн-оплата. Приложение должно поддерживать онлайн-оплату заказов, чтобы пользователи могли оплачивать свои заказы безопасно и удобно.
6. Уведомления. Приложение должно отправлять уведомления пользователям о подтверждении заказа, его готовности и доставке;
7. Служба поддержки. Приложение должно иметь службу технической поддержки, которая готова помочь пользователям в случае возникновения проблем с оформлением заказа или оплатой;
8. Система скидок и бонусов. Приложение может предлагать клиентам скидки и бонусы, чтобы поощрять пользователей делать заказы и привлекать новых пользователей.

Глава 1. Основы ручного тестирования

1.1 Что такое мобильное приложение, его особенности

Моби́льное приложе́ние или приложение для мобильных устройств, также может встречаться в обиходе под названиями [сетевое](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) приложение, [онлайн или офлайн](https://ru.wikipedia.org/wiki/%D0%9E%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD_%D0%B8_%D0%BE%D1%84%D0%BB%D0%B0%D0%B9%D0%BD) приложение, приложение для [смартфона](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD)) — программное изделие, разновидность прикладного [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), предназначенная для работы на смартфонах, [планшетах](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD%D1%88%D0%B5%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) и других мобильных (портативных, переносных, карманных) устройствах. Обеспечивает без привязки к [стационарному компьютеру](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D1%82%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), «на ходу» необходимые пользователю взаимодействия со [Всемирной Сетью](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) (либо устанавливается на устройстве благодаря такому взаимодействию — после скачивания на носимое пользователем устройство). Первоначально мобильные приложения использовались для быстрой проверки электронной почты, но высокий спрос на услуги [мобильного интернета](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) привёл к расширению их применения и в других областях, таких как [игры для мобильных телефонов](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B8%D0%B3%D1%80%D1%8B), [GPS](https://ru.wikipedia.org/wiki/GPS) либо [ГЛОНАСС](https://ru.wikipedia.org/wiki/%D0%93%D0%9B%D0%9E%D0%9D%D0%90%D0%A1%D0%A1)-геолокация, общение, просмотр видео и пользование [Интернетом](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82).

Программные изделия могут быть предустановлены на самом мобильном устройстве или загружаться на него из центров [цифрового распространения](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B4%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%86%D0%B8%D1%8F) — онлайновых [магазинов приложений](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B3%D0%B0%D0%B7%D0%B8%D0%BD_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9). Последние могут представлять собой также точку доступа в среду разработки приложений (в основе которой конкретная [мобильная операционная система](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)) либо стороннюю торговую площадку («[маркетплейс](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D0%BA%D0%B5%D1%82%D0%BF%D0%BB%D0%B5%D0%B9%D1%81)» [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Marketplace*), которая просто предоставляет третьим лицам свой [репозиторий](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9) для размещения готовых приложений — бесплатно или за плату. При этом программное обеспечение доступа к данным [онлайн-платформам](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BC%D0%B0%D0%B3%D0%B0%D0%B7%D0%B8%D0%BD) само по себе является приложением для мобильного устройства, либо предустановленным, либо скачиваемым. Одно и то же изделие может выпускаться разработчиками в нескольких версиях для наиболее распространённых операционных систем мобильных устройств ([iOS](https://ru.wikipedia.org/wiki/IOS), [Android](https://ru.wikipedia.org/wiki/Android), [Windows Phone](https://ru.wikipedia.org/wiki/Windows_Phone) и т. д.).

Виды приложений:

Мобильные приложения можно разделить на три подвида:

* [Нативные приложения](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F), предустановленные либо скачиваемые, но изначально задуманные под определённую операционную систему или устройство. Приложение, написанное для устройства c [прошивкой](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%88%D0%B8%D0%B2%D0%BA%D0%B0) Apple не пойдёт на устройстве с прошивкой Android. Поэтому большинство разработчиков готовят программные изделия в виде набора под несколько операционных систем.
* [Веб-приложения](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F), являющиеся частными случаями [мобильного браузера](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B1%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) для просмотра специально созданных [мобильных сайтов](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%81%D0%B0%D0%B9%D1%82). Работают на языках программирования Сети: разметочного HTML, формального CSS и встраиваемого JavaScript. Преимущество — приложения независимы от операционной системы устройства, так как информация в основном хранится «в облаке» и обрабатывается за счёт ресурсов Всемирной Сети. Недостаток — работают медленнее соответствующих нативных приложений.
* [Гибридные приложения](https://ru.wikipedia.org/w/index.php?title=%D0%93%D0%B8%D0%B1%D1%80%D0%B8%D0%B4%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F&action=edit&redlink=1), представляющие собой смешение двух вышеописанных подходов. Гибридные приложения создают с помощью компонента [WebView](https://ru.wikipedia.org/w/index.php?title=WebView&action=edit&redlink=1). Webview-приложения представляют собой мобильные версии сайтов, отображаемые в интерфейсе мобильного приложения. Webview приложения доступны на платформах [Android](https://ru.wikipedia.org/wiki/Android) (*Android WebView Media Integrity*) и [iOS](https://ru.wikipedia.org/wiki/IOS). Такое приложение может отображать сайт, созданный по технологии [веб-приложения](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Для пользователя такое приложение будет казаться [нативным](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F) и иметь все необходимые функции. Для разработчика использование данной технологии снижает затраты на написание отдельного кода для мобильного приложения, так как сайт по технологии веб-приложения уже предоставляет все необходимые функции. Также webview-приложения могут использовать нативные функции систем iOS и Android, такие как [push-уведомления](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_push), оплату через [GooglePay](https://ru.wikipedia.org/wiki/Google_Pay) или [ApplePay](https://ru.wikipedia.org/wiki/Apple_Pay), и многие другие. Приложения этого подвида разрабатываются с использованием таких [каркасов](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) как [Cordova](https://ru.wikipedia.org/wiki/Cordova), [Flutter](https://ru.wikipedia.org/wiki/Flutter), [React Native](https://ru.wikipedia.org/wiki/React_Native) и ряда других.

Основные центры цифрового распространения

Для старейших торговых площадок приложений характерна специализация на собственной операционной платформе, хотя существуют и кроссплатформенные магазины[[8]](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5#cite_note-8). На начало XXI века рынок приложений для мобильных устройств характеризуется как [дуополия](https://ru.wikipedia.org/wiki/%D0%94%D1%83%D0%BE%D0%BF%D0%BE%D0%BB%D0%B8%D1%8F) мегакорпораций Google и Apple (с операционными системами Android и iOS, соответственно), привлёкших основную массу пользователей и создавших собственные экосистемы разработчиков приложений в мировом масштабе.

Точками цифрового распространения приложений для мобильных устройств, нативных для основных [мобильных операционных систем](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) являются следующие сетевые торговые площадки:

* [App Store](https://ru.wikipedia.org/wiki/App_Store) (для [iOS](https://ru.wikipedia.org/wiki/IOS)/[iPadOS](https://ru.wikipedia.org/wiki/IPadOS_16))
* [Google Play](https://ru.wikipedia.org/wiki/Google_Play) (для [Android](https://ru.wikipedia.org/wiki/Android))
* [Samsung Galaxy Store](https://ru.wikipedia.org/wiki/Samsung_Galaxy_Store) (для Android, [Tizen](https://ru.wikipedia.org/wiki/Tizen))
* [Windows Store](https://ru.wikipedia.org/wiki/Windows_Store) ([Универсальная платформа Windows](https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0_Windows))
* [Amazon Appstore](https://ru.wikipedia.org/wiki/Amazon_Appstore) (для Android, [Windows 11](https://ru.wikipedia.org/wiki/Windows_11))
* [BlackBerry World](https://ru.wikipedia.org/wiki/BlackBerry_World) (Android)
* [Huawei AppGallery](https://ru.wikipedia.org/wiki/Huawei_AppGallery) (Android)

Существуют также крупные маркетплейсы, официально не завязанные на какую-либо операционную систему (сторонние платформы), учреждённые в отдельных странах:

* [Appland](https://ru.wikipedia.org/w/index.php?title=Appland&action=edit&redlink=1) (Android и iOS), Швеция
* [Aptoide](https://ru.wikipedia.org/w/index.php?title=Aptoide&action=edit&redlink=1) (Android), Португалия
* [Cafe Bazaar](https://ru.wikipedia.org/w/index.php?title=Cafe_Bazaar&action=edit&redlink=1) (Android), Иран
* [Cydia](https://ru.wikipedia.org/wiki/Cydia) (iOS, для [взломанных айфонов](https://ru.wikipedia.org/wiki/%D0%94%D0%B6%D0%B5%D0%B9%D0%BB%D0%B1%D1%80%D0%B5%D0%B9%D0%BA), после [легализации этой операции](https://ru.wikipedia.org/wiki/Digital_Millennium_Copyright_Act) в 2010 году в США)
* [F-Droid](https://ru.wikipedia.org/wiki/F-Droid) (Android, только свободное программное обеспечение), Великобритания
* [GetJar](https://ru.wikipedia.org/w/index.php?title=GetJar&action=edit&redlink=1) (Android), Литва
* [MiKandi](https://ru.wikipedia.org/w/index.php?title=MiKandi&action=edit&redlink=1) (Android, для программных изделий, связанных с распространением [порнографии на мобильных устройствах](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D1%80%D0%BD%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)), США
* [Opera Mobile Store](https://ru.wikipedia.org/wiki/Opera_Mobile_Store) (кроссплатформенная, для Android, [Java](https://ru.wikipedia.org/wiki/Java_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0)), [Symbian](https://ru.wikipedia.org/wiki/Symbian_OS), Windows Mobile, iOS), Норвегия

В связи с ростом мировой нестабильности, введением [санкций против России](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%BD%D0%BA%D1%86%D0%B8%D0%B8_%D0%B2_%D1%81%D0%B2%D1%8F%D0%B7%D0%B8_%D1%81%D0%BE_%D0%B2%D1%82%D0%BE%D1%80%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC_%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B8_%D0%BD%D0%B0_%D0%A3%D0%BA%D1%80%D0%B0%D0%B8%D0%BD%D1%83_(2022)), появились и сугубо российские центры сетевого распространения программных изделий данного рода:

* [NashStore](https://ru.wikipedia.org/wiki/NashStore) ([Android](https://ru.wikipedia.org/wiki/Android))
* RuMarket
* [RuStore](https://ru.wikipedia.org/wiki/RuStore) ([Android](https://ru.wikipedia.org/wiki/Android))

1.2 Что такое тестирование, кто такой тестировщик. Зачем нужно тестирование и когда оно начинается.

Тестировщик – это квалифицированный специалист, занимающийся тестированием мобильных приложений, десктопных программ, веб-приложений и другого софта до его официального выхода. В ходе работы тестировщик проверяет наличие ошибок, находит уязвимости, смотрит, адаптировано ли приложение под разные устройства.

Тестирование ПО – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением.

Этапы тестирования:

### Проработка требований к продукту:

На этом этапе тестировщики внимательно изучают требования продукта — это могут быть документы, спецификации, описание того, как пользователь взаимодействует с продуктом (по-другому это называют пользовательскими сценариями). Четкое понимание требований помогает определить области, которые нужно протестировать.

### Анализ требований:

Анализ требований позволяет выяснить, какие возможные риски или сложности могут возникнуть при тестировании. Также на этом этапе можно выявить возможные несоответствия или недостаточно ясные требования, которые требуют уточнения у разработчиков или заказчика.

### Разработка стратегии и плана тестирования:

* + Выбор методов тестирования — ручное, автоматизированное, тестирование на реальных устройствах и другие.
  + Анализ потенциальных рисков, которые могут повлиять на качество и успешность тестирования, и планирование мер по их минимизации.
  + Планирование ресурсов — кто будет тестировать продукт, каким оборудованием и инструментами можно при этом пользоваться и сколько времени займет тестирование, к какому сроку оно должно быть закончено

### Создание тестовой документации:

На этом этапе на основе требований и анализа тестировщики создают тестовые случаи, тест-планы, отчетность и другую документацию, которая будет использоваться во время тестирования. Тестовая документация определяет, какие тесты будут проведены, как будут собраны результаты и как будет оценено качество ПО.

### Тестирование:

После того как команда утверждает стратегию тестирования и тестовую документацию, проводится тестирование. Тестирование программного обеспечения — это длительный и обширный процесс. По ходу составляются отчеты о выявленных недостатках, проводится набор тестовых сценариев, создается тестовая среда и выполняется тестирование согласно заранее задокументированным видам тестов, описанным в тестовой документации.

Найти все ошибки в продукте невозможно. Главная цель заключается не в создании идеального продукта без ошибок, а в обнаружении максимального числа дефектов, которые могут потенциально повлиять на работу системы.

### Эксплуатация и поддержка:

После того как разработчики устраняют дефекты и выпускают продукт, тестировщик переходит к тестированию продукта в рабочей среде. Важно отметить, что на этом этапе не только происходит релиз продукта, но и начинается пост-релизная поддержка.

Невозможно предусмотреть все особенности использования и окружение, в котором будет работать продукт. Поэтому на данном этапе акцент делается на обратной связи пользователей. Устранение дефектов и поиск ошибок проводится быстро, но тщательно.

Виды тестирования классифицируются по:

### Характеру сценариев

### Сценарий в тестировании — это описание того, как пользователь будет взаимодействовать с готовым продуктом. В эту группу входят два вида тестирования:

1. **Тестирование позитивных сценариев** проверяет, как должна работать программа в нормальных условиях. Например, если это веб-приложение, тестирование позитивных сценариев проверит, что пользователь может успешно зарегистрироваться, войти в систему и без проблем использовать основные функции.
2. **Тестирование негативных сценариев** проверяет, как программа ведет себя в необычных или некорректных ситуациях. Такие сценарии показывают, что программа корректно обрабатывает ошибки и не позволяет пользователю выполнить действия, которые не предполагаются в нормальной работе приложения. Возвращаясь к веб-приложению: тестирование негативных сценариев может включать проверку того, что система правильно обрабатывает неправильный ввод данных или отказывается выполнять определенные действия в некорректных условиях.

### Степени автоматизации тестирования

1. **Ручное тестирование** — это проверка программного обеспечения вручную, без использования автоматизированных инструментов. Тестировщик взаимодействует с программой как обычный пользователь.
2. **Автоматизированное тестирование** — это проверка программного обеспечения с использованием специальных программных инструментов, которые выполняют тесты автоматически, без участия человека. Тестировщик создает скрипты или сценарии тестирования, которые содержат инструкции для выполнения определенных действий и проверки результатов.

### Объектам тестирования

1. **Функциональное тестирование** проверяет соответствие программы или системы заранее определенным функциональным требованиям и ожиданиям. Основная цель функционального тестирования — убедиться, что программа выполняет свои функции и операции согласно спецификациям, а также работает правильно и без сбоев.

Во время функционального тестирования тестируются различные сценарии использования, входные данные и выходные результаты, чтобы удостовериться в правильности работы приложения.

Функциональное тестирование делится на подвиды:

* Unit-тестирование (также модульное тестирование) — проводится во время создания исходного кода. На этом этапе тестируются отдельные компоненты приложения. Тестировщики пишут тесты, чтобы убедиться, что каждый компонент будущей программы работоспособен и дает правильные результаты при различных входных данных.
* Интеграционное тестирование. На следующем этапе тестируется то, как компоненты будущего приложения взаимодействуют между собой.
* Системное тестирование (End-to-end тестирование). На этом этапе специалисты тестируют все компоненты программы как единое приложение. Тестировщики проверяют, что продукт корректно обрабатывает различные сценарии и ситуации.
* Приемочное тестирование. На последнем этапе продукт тестирует уже клиент или заказчик. Они проверяют, соответствует ли проект их ожиданиям и требованиям. А еще убеждаются, что программа дает правильные результаты и работает без ошибок.

1. **Нефункциональное тестирование** проверяет нефункциональные аспекты программы — производительность, безопасность, надежность, масштабируемость и совместимость. Основная цель нефункционального тестирования — убедиться, что программа не только выполняет свои функции, но также соответствует требованиям к качеству, производительности и безопасности.

Нефункциональное тестирование часто охватывает атрибуты программы, которые не всегда видны конечному пользователю, но критически важны для обеспечения стабильной и надежной работы приложения.

Нефункциональное тестирование делится на подвиды:

* Нагрузочное тестирование — для проверки производительности приложения под нагрузкой. Во время теста программа подвергается нагрузке: тестировщик, например, увеличивает число пользователей или операций и проверяет, как она будет работать.
* Тестирование на проникновение — для проверки уровня безопасности. Этот вид тестирования проводится, чтобы узнать, насколько безопасна программа или веб-сайт от потенциальных кибератак и несанкционированного доступа.
* Тестирование совместимости. На этом этапе тестировщики проверяют работу программы на различных платформах, устройствах и браузерах, чтобы убедиться в их совместимости.
* Стресс-тестирование — этот вид тестирования помогает выявить уязвимости и слабые места в системе, которые могут проявиться при больших нагрузках.
* Тестирование на отказоустойчивость — помогает удостовериться, что приложение может успешно справляться с различными неполадками, такими как сбои серверов, потеря связи или другие неблагоприятные события, и продолжать функционировать нормально без значительных нарушений или потери данных.
* Тестирование интерфейса пользователя — подразумевает проверку удобства, доступности и правильности работы пользовательского интерфейса программы.
* Тестирование на восстановление. В ходе этого тестирования создаются различные сценарии отказов: отключение серверов или потеря связи, чтобы убедиться, что приложение может быстро и корректно восстановиться и продолжить работу без значительных проблем.

### Степени знания системы

1. **Тестирование «черного ящика»** — это способ проверки программного обеспечения, когда тестировщик не знает внутренней структуры или деталей работы самой программы. Он смотрит на нее как на «черный ящик», и проверяет, как система взаимодействует с внешним миром и выполняет свои функции.
2. **Тестирование «белого ящика»**, предполагает, что тестировщик имеет доступ к внутренней структуре и коду программы. Он изучает, как работает программа «изнутри», чтобы убедиться, что все компоненты и функции написаны правильно и соответствуют требованиям.
3. **Тестирование «серого ящика»** — это комбинация тестирования «черного ящика» и «белого ящика». Тестировщик знает некоторые детали внутренней структуры программы, но не обладает полной информацией о них. Он проверяет как внешнее поведение программы, так и использует некоторые знания о коде для определения эффективности и корректности работы программы
4. **Тестирование по документации** (или формальное тестирование) — основано на анализе документов, созданных во время разработки — спецификации, требования и дизайн. Тестировщики изучают эти документы, чтобы понять, как должно работать программное обеспечение, и затем создают тестовые сценарии и тестовые случаи для проверки соответствия программы этим требованиям и спецификациям.
5. **Интуитивное тестирование** — выполняется на основе интуиции и опыта тестировщика, а не по строгому плану тестирования.

### Времени проведения тестирования

1. **Альфа-тестирование** — это этап тестирования программного обеспечения, который происходит перед его официальным выпуском и предполагает проверку продукта внутри компании-разработчика или ограниченной группой тестировщиков. Альфа-тестирование помогает выявить возможные проблемы и ошибки перед предоставлением продукта пользователю.
2. **Дымовое тестирование** — это быстрая проверка программного обеспечения, которую выполняют после внесения значительных изменений или обновлений в код. Этот вид тестирования напоминает «пробный пуск» программы, чтобы убедиться, что основные функции работают без критических ошибок.
3. Если после дымового тестирования в продукт добавляют какую-то фичу или просто хотят убедиться, что все предыдущие функции работают правильно, то проводят **регрессионное тестирование**. Тестировщики убеждаются, что новая функция работает правильно и выполняет свои задачи так, как ожидается, а все остальное не вызывает новых ошибок.
4. **Приемочное тестирование** выполняют представители заказчика, чтобы удостовериться, что продукт вышел качественным, и что за него можно заплатить деньги. Чтобы успешно пройти приемочное тестирование, обычно нужно просто выполнить тесты, которые доказывают соответствие программы требованиям.
5. **Бета-тестирование**. Тестировщики предоставляют готовую программу ограниченной группе реальных пользователей, которые могут сами с ней повзаимодействовать. Юзеры выявляют дополнительные проблемы, получить обратную связь от пользователей и улучшить программу перед ее окончательным выпуском для широкой аудитории.

1.3 Этапы ручного тестирования мобильных приложений

Ручное тестирование представляет собой метод проверки программного продукта вручную, где тестировщики взаимодействуют с приложением, имитируя действия пользователя. Это включает выполнение тест-кейсов, проверку функциональности и выявление дефектов.

Ручное тестирование необходимо для тщательной проверки функциональности и взаимодействия приложения с пользователем. Это обеспечивает обнаружение неочевидных проблем, адаптацию к изменениям и обеспечение высокого уровня качества продукта.

Ручное тестирование выполняет критическую роль на всех этапах жизненного цикла разработки ПО. Оно помогает выявить дефекты на ранних стадиях, обеспечивает контроль изменений и гарантирует соответствие продукта ожиданиям пользователя.

К этапам тестирования мобильных приложений относятся:

* + Анализ требований

В первую очередь мы изучим требования к продукту и поймем, какие виды тестирования будем применять. Видами документации может быть:

* + - Требования и Спецификации
    - Технические характеристики приложения
    - Руководства пользователя
    - Руководства по эксплуатации
  + Планирование тестирования
  + Создание тест-кейсов
  + Выполнение тестирования

Виды тестирования, 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
  + Заведение и отслеживание дефектов
  + Регрессионное тестирование
  + Пользовательское тестирование
  + Релиз
  + Мониторинг после выпуска

1.4 Выбор устройства для тестирования мобильных приложений

Тестирование мобильных приложений может проводиться на:

* + реальных девайсах
  + симуляторах
  + эмуляторах
  + мобильных фермах

Тестирование на реальных девайсах является критически важным, а остальные инструменты идут, как правило, в помощь.

Исходя из разнообразия видов мобильных приложений, необходимо проводить тестирование на разных устройствах. 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.

Большая часть клиентов судит о надёжности продукта на основе диапазона поддерживаемых устройств и ОС.

Для 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, можно будет подобрать популярные у пользователей устройства. Сервисы, 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 делаю сервис популярным.

1.5 Источники требований: что такое техническое задание и как проводить ручное тестирование, если техническое задание отсутствует

Техническое задание (ТЗ) на разработку программного обеспечения – это документ, определяющий требования к IT-продукту, включая его назначение, функции, поведение, используемые компоненты, технологии, инструменты разработки, а также порядок выполнения работ. ТЗ служит руководством для бизнес- и технических групп, занимающихся созданием IT-решения.

Стоимость сложного устройства или приложения невозможно оценить навскидку. Необходимо учесть множество моментов – затраты труда специалистов, стоимость компонентов и логистики, работы, связанные с сертификацией и т.д. Грамотно составленный документ позволяет и исполнителю, и заказчику видеть и оценивать как процесс разработки целиком, так и отдельные его ступени. Таким образом, заказчик получит представление о предварительной стоимости каждого этапа работ. Более точные данные будут даны в смете проекта.

В ТЗ очерчиваются примерные сроки исполнения заказа. У клиента и аутсорсинговой компании не будет разногласий по поводу тайминга, если с самого начала в документе обозначены временные отрезки для каждого этапа проекта.

Заказчику будет проще оценить готовое решение – электронное устройство, программное обеспечение или программно-аппаратную систему, – сверив его с описанием в техническом задании. Именно поэтому ТЗ должно быть составлено грамотно и максимально подробно.

Качественно написанное техническое задание на разработку прибора или ПО может свидетельствовать о компетенции и опыте специалистов. Вдумчивый подход разработчиков к подготовке проекта, понятная и исчерпывающая информация в ТЗ говорят об общем уровне сервиса компании.

В ТЗ описывается сам продукт, его назначение и функциональность, а также этапы разработки, основные элементы электроники и инструменты для создания ПО. Таким образом, и заказчик, и разработчик имеют полное представление о проектируемом IT-решении, что служит страховкой от разногласий, недопониманий, внесений незапланированных изменений в концепцию продукта.

Работа над проектом идет быстрее и проще, когда команда разработчиков опирается на ТЗ. Нет необходимости согласовывать каждый шаг, теряя время.

Разработкой технического задания на проектирование устройства или создание программного обеспечения занимаются специалисты, знающие все нюансы разработки и того, как будет выполнен проект – этапы работ, сроки, компоненты и конечный продукт. Это ПМы, разработчики, тестировщики. Каждый из них вносит в ТЗ свою информацию, выстраивая общую картину проекта.

Техническое задание, созданное компанией-разработчиком, будет учитывать не только все пожелания клиента, но и возможности подрядчика (экспертизу в разработке, опыт работы с компонентами, используемые инструменты и языки программирования и т.д.). Все пункты ТЗ будут оговорены сторонами и одобрены заказчиком, чтобы по итогу сотрудничества клиент получил удовлетворяющий всем требованиям продукт.

### Что должно быть в ТЗ:

* Термины, сокращения и определения

Использующиеся в тексте термины приводятся в начале документа. Это могут быть как IT-понятия – названия элементов, сред и языков программирования, технические определения, – так и слова и обозначения из той сферы, для которой предназначается IT-решение. Чем тщательнее будет продуман список профессиональных слов, тем лучше поймут друг друга исполнитель и заказчик

#### Назначение продукта

В этом блоке расписываются назначение IT-решения, цели его создания и целевая аудитория. К основным целям могут относиться увеличение клиентской базы, положительный имидж компании, увеличение производительности труда, уменьшение ручных операций и т.д.

Пункт также содержит информацию о задачах проекта – это может быть разработка с нуля и под ключ, участие в отдельном этапе создания (например, работа над программным обеспечением для готовой аппаратной части) или усовершенствование существующего продукта.

Цели должны быть конкретными и понятными, чтобы конечный продукт максимально соответствовал требованиям и был полезен заказчику.

#### Требования к проекту

Основа спецификации, его самая весомая и развернутая часть. Как правило, блок требований содержит следующие подразделы:

1. **Общие требования** определяют последовательность процесса разработки.
2. **Функциональные требования** касаются функций и поведения IT-решения.
3. **Нефункциональные требования** определяют такие критерии, как производительность, масштабируемость, ремонтопригодность, безопасность продукта и многое другое.
4. **Требования к разработке** могут быть представлены несколькими пунктами, где подробно описываются этапы работ и используемые компоненты и инструменты.
5. **Требования к организации** **и качеству работ** определяют то, как будет организована работа над проектом, коммуникация между заказчиком и исполнителем, а также основные моменты, касающиеся качества системы, устройства или программного продукта – время непрерывной работы, поведение системы в аварийной ситуации и пр.
6. **Требования к безопасности** могут содержать требования о защите кода, разграничении доступа, прав и т.д.

**Исследовательское тестирование** лучше всего подходит в ситуациях, когда документация недостаточная, либо вовсе отсутствует, в условиях очень сжатых сроков и как дополнение к другим, более формальным, методам тестирования.

Исследовательское тестирование – это особый вид тестирования программного обеспечения, который имеет множество преимуществ для приложения, позволяя ему полностью раскрыть свой потенциал.

Исследовательское тестирование объединяет этапы разработки и выполнения теста, обеспечивая полную свободу действий для тестировщика и позволяя ему постоянно оптимизировать свою работу.

По мере проверки программного обеспечения, скорее всего, обнаружат новые компоненты, требующие тщательной проверки, и могут легко придумать новые тесты, которые принесут пользу приложению.

Менее структурированный подход помогает тестировщикам определить, как приложение может отреагировать на реалистичные сценарии и тестовые примеры, и служит важным дополнением к сценарному тестированию.

Качество разведочного тестирования команды часто зависит от мастерства отдельных тестировщиков, поскольку проверки требуют творческого подхода и глубокого понимания программного обеспечения. Это процесс непрерывного открытия – процесс, в котором тестировщики используют дедуктивные рассуждения, чтобы направлять свою общую методику.

Исследовательское тестирование особенно полезно, поскольку оно отражает то, как пользователи могут использовать программное обеспечение. Большинство пользователей находят ошибки и проблемы случайно, поэтому такие незаписанные процессы могут помочь тестировщикам найти проблемы, которые заранее определенные проверки могут не выявить.

Глава 2. Подготовка к ручному тестированию мобильного приложения

2.1 Составление чек-листов

Чек-лист является своеобразным списком, с помощью которого большую задачу можно разделить на отдельные подпункты и контролировать их выполнение. После завершения каждого этапа из такого перечня, он помечается как выполненный. Задача состоит в закрытии всех или максимально возможного количества пунктов. Более того, такой чек лист может применяться многократно для повторяющихся процессов.

Этот инструмент был придуман авиаторами. Когда в 1934 году новая модель «Боинга» в ходе испытаний попала в аварию, специалисты задумались над тем, как решить проблему сложности управления самолетами. Пилоты не в состоянии запоминать и тем более осуществлять контроль всех процессов, которые сопутствуют взлету, посадке и маневрированию воздушных судов. Решением этой проблемы стала пошаговая инструкция, в которой были перечислены все необходимые действия летчика.

Различают два варианта чек-листов: Read-Do (прочти и выполни) и Do-Confirm (выполни и сверь). В первом случае речь идет об инструменте для новичков, которые впервые встречаются с определенной процедурой. После ознакомления с инструкцией они выполняют задачи в указанном порядке и отмечают их. Do-Confirm рассчитан на профессионалов для обеспечения самоконтроля.

Чек-листы составляют на основе технического задания (ТЗ) — одного или нескольких документов, определяющих цель, структуру, свойства и методы проекта.

Последовательность действий при составлении чек-листа:

1. Определить все возможные проверки.

2. Сгруппировать проверки в разделы.

3. Определить тестовые данные, если они необходимы.

4. Подумать, какие колонки будут в чек-листе.

Правила составления Чек-листа:

1. Один пункт — одна операция.

Пункты чек-листа — это однозначные атомарные (выполняющее одно действие) и полные (логически завершённые) операции. Например, авторизация в профиль пользователя и редактирование данных пользователя — две разные задачи. В списке проверок их оформляют отдельными пунктами.

2. Пункты всегда начинаются с существительного или глагола неопределённой формы.

При составлении пунктов важно придерживаться унифицированных форм: начинать их с существительного («Проверка», «Добавление», «Отправка») или неопределённой формы глагола («Проверить», «Добавить», «Отправить»). Так пункты будут понятными и однозначными.

3. Соблюдайте структуру.

Чтобы выполнять чек-лист было удобно, лучше составлять тесты так, чтобы они исходили из логики использования функционала. Например:

● в рамках раздела «Регистрация и Личный профиль» — регистрация на сайте, редактирование профиля;

● в рамках раздела «Форма обратной связи» — валидация полей, отправка письма, доставка письма.

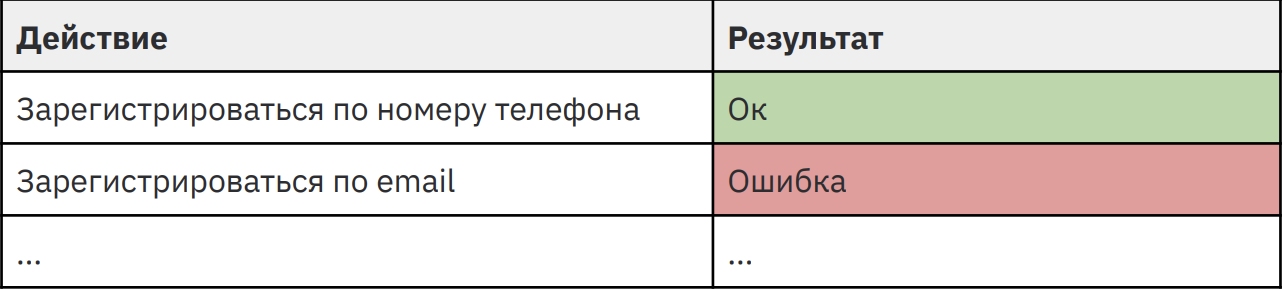
4. Чек-лист важно составлять, опираясь на требования. Не придумывайте то, чего нет в требовании.

5. Общепринятые названия. Хорошо, когда в команде все разговаривают на одном языке, используют общую терминологию. Если один тестировщик в команде называет линию, разделяющую элементы на странице, дивайдером, а остальные — сепаратором, может возникнуть недопонимание.

Варианты колонок в чек-листе:

* Простейший

В чек-листе две колонки: действие и результат — пройдено (passed) или не пройдено (failed). Исполнитель самостоятельно подбирает тестовые данные. Их необязательно фиксировать в чек-листе. Преимущества — быстрое составление.



* Простейший с комментарием

Похож на простейший, но с дополнительной колонкой для комментария. В ней указываются примечания, ссылки на созданные дефекты и прочая полезная информация.



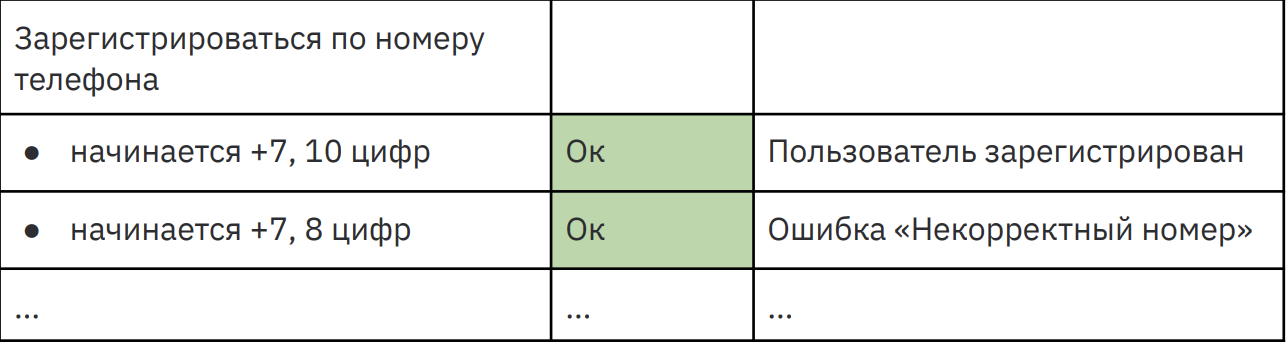
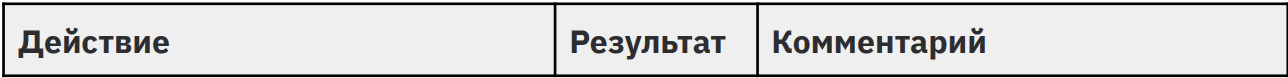
* С тестовыми данными

Кроме действия и результата в чек-листе указано, на каких тестовых данных будет проводиться тестирование.



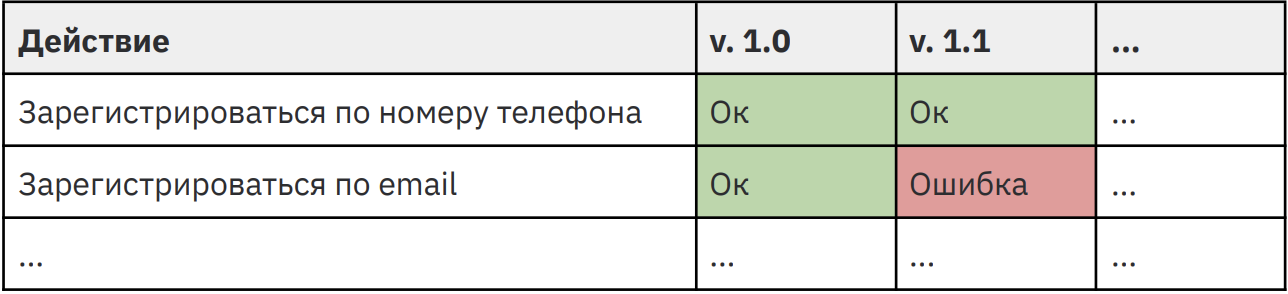
* Проверки с детализацией по тестовым данным

Тестовые данные, на которых тестировщик выполняет проверку, могут указываться в столбце «Действие».



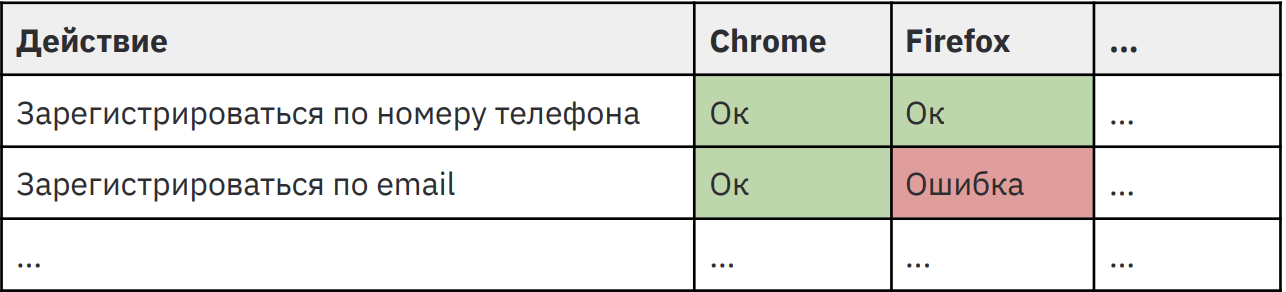
* Проверки с детализацией по версиям

В чек-листе хранится история прохождения для разных версий ПО. Это полезно для отслеживания: в одних местах чаще возникают ошибки, а в других всё относительно стабильно.



* Проверки с детализацией по окружениям

Задача тестирования — проверять работу ПО в разных условиях, то есть с разными окружениями. Веб-приложения должны работать без ошибок во всех браузерах, а мобильные — на всех устройствах и разных операционных системах.



Правила выполнения чек-листа:

1. Не пропускать проверки чек-листа. Если какие-то пункты нельзя проверить из-за багов или недоступности среды, надо отметить их и вернуться к ним позже.

2. Если указываются тестовые данные, нужно использовать именно их.

3. Указывать результат и комментарий.

Резолюция:

Когда исследуемый объект протестирован, нужно выставить кейсу резолюцию, то есть вердикт — прошёл он проверку или нет. Распространённые статусы:

● passed — успешно пройден

● failed — кейс не прошел проверку

● skipped — пропуск проверки

● blocked — проверка заблокирована

● untested — кейс еще не брали в работу

passed ставится, когда исследуемый объект проходит проверку в соответствии с ТЗ. failed – исследуемый объект не соответствует ТЗ. Например, функционал поломан, не работает или его нет (разработчик потерял кнопку). skipped — пропуск проверки кейса. Например, из-за нехватки времени или если в кейсе есть неточности: он устарел, из документации вы понимаете, что его уже изменили в проекте. blocked — проверка заблокирована поломанным функционалом, от которого прямо или косвенно зависит тестируемый. Например, есть функционал «авторизация пользователя» и функционал «редактирование данных пользователя». Разработчик допустил ошибку в коде и функционал авторизации поломан — ему мы выносим резолюцию failed. Проверить редактирование данных пользователя не можем — выставляем статус blocked. В комментарии указываем, почему не можем протестировать этот функционал.

Преимущества и недостатки чек-листов:

Преимущества:

1. Чек-листы затрагивают большее количество кейсов, так как при прохождении их можно по-разному выполнять.

2. Сокращают затраты на содержание и поддержку тестов.

3. Обеспечивают высокую скорость тестирования: не нужно отмечать результат каждого шага тестировщика, достаточно общего результата проверки.

4. Позволяют проходить и комбинировать тесты по-разному, в зависимости от предпочтений сотрудников.

5. Показывают статистику: кто, когда и что проходил — с детализацией по сборке продукта и окружению, на котором проводилось тестирование. 6. Улучшают представление о системе в целом, показывают статус её готовности.

7. Показывают объём проделанной и предстоящей работы по тестированию.

Недостатки:

1. Начинающие тестировщики не всегда эффективно проводят тесты без достаточно подробной документации.

2. Чек-листы невозможно использовать для обучения начинающих сотрудников, так как в них недостаточно подробных сведений.

3. Заказчику или руководству может быть недостаточно того уровня детализации, который предлагают чек-листы.

4. Неопределённость тестового набора: каждый тестировщик выполняет пункт чек-листа по-своему.

2.2 Составление тест-кейсов

Тестовый случай (тест-кейс) — это совокупность шагов, конкретных условий и параметров, нужных для проверки работы тестируемой функции или её части.

В тест-кейсе описывается:

● какие шаги нужно выполнить (например, какие кнопки нажать, чекбоксы активировать, данные внести), чтобы проверить работу участка системы;

● какой результат должен быть у каждого шага. В тест-кейсе есть подробная детализация для каждой из проверок, а в чек-листе — нет.

У тест-кейса есть обязательные атрибуты (необходимые поля). Большая часть заполняется на этапе проектирования тест-кейса, несколько — после прохождения теста.

Атрибуты на этапе проектирования тест-кейсов:

1. ID тест-кейса — уникальный идентификатор, присваивается автоматически. 2. Название — краткое описание сценария, который проверяет тест-кейс. 3. Шаги — последовательность действий, чтобы проверить работу функции или выполнение сценария. 4. Ожидаемый результат — как должна вести себя система после каждого шага тест-кейса, то есть то поведение, которое мы ожидаем получить после выполнения конкретного действия или последовательности действий. 5. Приоритет тест-кейса зависит от приоритета функций и сценариев, которые он проверяет. Определяет очерёдность выполнения теста во время тестирования. Например, при высоком приоритете тест-кейс будет в начале очереди на выполнение, а при низком — в конце. Если во втором случае что-то сломано, для нас это не так критично, как в первом. 6. Тестовые данные — данные, которые используются для проверки. Могут указываться в шагах тест-кейса или в отдельном файле с указанием ссылки на них. Это, например, данные для авторизации пользователя или данные тестовой карты для оплаты заказа. 7. Предусловия — действия, которые нужно выполнить, прежде чем приступать к тест-кейсу, а также настройки приложения и тестовой среды. 8. Постусловия — выполнение тест-кейса, как правило, переводит систему из одного состояния в другое: меняются настройки, производятся расчёты. Состояние, в которое нужно привести систему после прохождения тест-кейса, указывается в постусловии.

Атрибуты на этапе исполнения тест-кейсов:

1. Фактический результат — то, что мы получаем после выполнения всего тест-кейса или его конкретного шага. Если фактический результат совпадает с ожидаемым, в графе с фактическим ставится статус passed — тест-кейс успешно пройден, ошибок не обнаружено. Если результат отличается от ожидаемого — статус failed — проверка провалена, полученный результат не соответствует ожидаемому. К тест-кейсу обязательно добавляется ссылка на дефект, обнаруженный при исполнении. 2. Статус тест-кейса обозначает результат исполнения этого кейса или причину, по которой он не может исполняться. Основные статусы:

● passed — успешно пройден

● failed — кейс не прошел проверку

● skipped — пропуск проверки

● blocked — проверка заблокирована

● untested — кейс еще не брали в работу

passed ставится, когда исследуемый объект проходит проверку в соответствии с ТЗ. failed – исследуемый объект не соответствует ТЗ. Например, функционал поломан, не работает или его нет (разработчик потерял кнопку). skipped — пропуск проверки кейса. Например, из-за нехватки времени или если в кейсе есть неточности: он устарел, из документации вы понимаете, что его уже изменили в проекте. blocked — проверка заблокирована поломанным функционалом, от которого прямо или косвенно зависит тестируемый. Например, есть функционал «авторизация пользователя» и функционал «редактирование данных пользователя». Разработчик допустил ошибку в коде и функционал авторизации поломан — ему выносится резолюцию failed. Проверить редактирование данных пользователя нет возможности — выставляется статус blocked. В комментарии указываем, почему не можем протестировать этот функционал.

Priority в тест-кейсе:

Тестировщик может сортировать тест-кейсы на выполнение по приоритету. Это полезно, если время на тестирование ограничено: можно оставить только тест-кейсы с высоким приоритетом, а остальные не проходить. Приоритет можно выразить в любом удобном виде:

● буквами: A, B, C, D, E;

● цифрами: 1, 2, 3, 4, 5;

● словами: «крайне высокий», «высокий», «средний», «низкий», «крайне низкий» (или английскими аналогами).

Количество градаций не зафиксировано, но часто лежит в диапазоне от 3 до 5. Формат может определяться командой и возможностями системы хранения тест-кейсов. Приоритеты выделяют на основании:

● важности требования, пользовательского сценария или функции, с которыми связан тест-кейс;

● важности потенциального дефекта, который может быть обнаружен в процессе выполнения тест-кейса.

Правила работы с тест-кейсами:

1. Один тест-кейс — одна проверка или один сценарий. 2. Заголовок точно описывает суть тест-кейса. По нему можно определить характер проверок, не открывая шаги теста: то есть что конкретно проверяется в функциональности. Пример: тест-кейс с отправкой сообщения, заголовок будет следующим: «Отправка сообщения другу из раздела “Чаты”» 3. Названия для элементов приложения (кнопок, чекбоксов, элементов меню) точные. Тест-кейсы используют разные люди, всем должно быть понятно их содержание. Не общепринятые названия элементов допускаются только в тех случаях, когда они понятны для всей команды. 4. Простой технический стиль без объяснений базовых понятий работы ПО. Подробности описываются в требованиях или спецификации. Тест-кейс не должен быть сочинением, иначе его будет трудно и долго читать. 5. Нет пропущенных шагов: каждое следующее действие вытекает из предыдущего. Например, в кейсе на проверку подписки на пользователя сразу после шага «Запустить приложение» не должно быть шага «Подписаться на пользователя», ведь нужно ещё перейти на страницу пользователя, на которого будем подписываться. Простой способ проверить, нет ли пропуска — пройти тест-кейс строго по описанным шагам, ничего не додумывая. 6. Нет описания очевидных действий (например, разблокировки девайса). Лучше начинать с ненулевого состояния и определённого экрана. О них сообщить в предусловии. 7. Нет зависимостей от других тест-кейсов. Часто для выполнения тест-кейса нужно попасть в определённое состояние пользователя. В таких случаях высок соблазн в предусловии сослаться на тест-кейс, в котором мы как раз попадаем в нужное состояние. Но это неправильно: спустя время тест-кейс, на который мы ссылаемся, может быть удален или изменён. Поэтому чётко описываем в предусловии, что должно быть сделано до выполнения тест-кейса, в шагах чётко прописываем все действия, даже если они частично дублируют действия из другого тест-кейса 8. Каждый тест-кейс уникален и не дублирует другой. 9. Обнаруженная ошибка становится очевидной. Шаги тест-кейса и ожидаемые результаты описываются таким образом, чтобы отклонение от них явно свидетельствовало о дефекте. 10.Гибкость для модификации. В тест-кейсе легко добавить, изменить или убрать шаг, в том числе в середине теста, а также изменить тестовые данные.

Преимущества и недостатки тест-кейсов:

Преимущества:

● За счёт полной детализации шагов тест-кейсы может проходить новичок. По ним удобно знакомиться с продуктом.

● Достаточно подробное описание бизнес-логики. При неидеальной организации хранения требований уточнение некоторых моментов проще будет найти в тест-кейсах.

Недостатки:

● Нужно больше времени на написание, чем для чек-листов.

● Сложно поддерживать. Если изменится бизнес-логика, будут переименованы разделы или произойдут другие изменения, затрагивающие пользовательский интерфейс и сценарии использования системы, нужно будет актуализировать все тест-кейсы, связанные с измененной частью.

2.3 Разработка тест-плана

Тест план — это артефакт тестирования, описывающий действия, которые будут происходить в процессе тестирования — от разработки стратегии до критериев поиска ошибок. Он также описывает логику завершения задач и оценку рисков со сценариями их разрешения.

Тест план имеет четкую структуру, установленную IEEE 829 — отраслевым стандартом для документации тестирования программ и систем. Это значит, что вы можете подготовить шаблон и использовать его для любого проекта, заполняя конкретными данными.

Создание тест плана в соответствии со стандартом IEEE 829 дает много преимуществ. Прежде всего, когда структура документа всем известна, такой документ и составлять легче, и пользоваться им проще. Стандарт IEEE 829 устраняет любые бесполезные дебаты относительно того, что включать в тест план и в каком порядке. Вместо этого тестировщики могут сосредоточиться на других, более важных вещах.

Структура тест-плана:

1. Идентификатор тест-плана.

В этом разделе мы указываем название и логотип компании, проводящей тестирование, название документа, его версию и год создания. Это титульная страница вашего тест плана.

1. Ссылки.

Дальше идет история документа. Добавьте таблицу изменений со следующими столбцами:

* Дата
* Версия
* Описание
* Автор

С помощью этой таблицы команда сможет эффективно фиксировать и отслеживать изменения в документе и процессе, который он описывает.

1. Введение.

Здесь мы кратко обозначаем, что собираемся делать. Введение — это пояснительная записка для клиента. В паре предложений опишите, какие услуги предоставит команда QA и зачем.

1. Объекты тестирования.

Объекты тестирования — это общие функциональные возможности, которые будут протестированы. Например, установка программы, регистрация в системе, оформление заказа и т. д. По сути, это краткое содержание тест плана. В дальнейшем каждый из объектов будет описан отдельно. Список может быть расширен или сокращен в зависимости от задач или типа тестирования.

1. Проблемы и риски.

В этом разделе мы описываем проблемы, с которыми команда может столкнуться во время тестирования. Например, если дедлайн установлен на летний период, разумно предположить, что люди могут уйти в отпуск, из-за чего возможны задержки. Здесь нужно упомянуть все значимые проблемы и риски: как связанные с кадрами, так и технологические аспекты.

1. Функции, которые нужно протестировать.

Эта часть охватывает то, что, по мнению большинства людей, и должно быть в тест плане: подробный список функций для проверки и время, за которое они должны быть проверены. Например, в «Объектах тестирования» мы упомянули функциональное тестирование. Тогда в функциях, которые необходимо протестировать, мы должны перечислить отдельные компоненты потока: ввод информации о доставке, выбор способа оплаты, подтверждение заказа и т. д. Что касается времени, клиент и компания, проводящая тестирование, обсуждают его до написания документации.

1. Функции, которые не нужно протестировать.

Здесь перечисляются функции, которые тестировщики по каким-то причинам не будут тестировать. Неважно, почему не покрываются тестами эти фичи. Указываются, какие именно функции не охватываются тестированием и остаются в зоне ответственности клиента.

1. Подходы.

Затем описываются методы и виды тестирования, которые будут применять. В этот раздел также включаются тест-кейсы. Благодаря этому клиент может получить полную картину действий по тестированию.

1. Критерии прохождения тестов.

Каждый тест-кейс будет обозначен как «Pass» (пройден) или «Fail» (провален) в зависимости от двух критериев:

* Наличие и серьезность багов
* Уровень успешного выполнения требований

Критерии начала — то, что должно быть и что нужно сделать до начала тестирования. Например, могут понадобиться:

* Законченная справочная документация
* Программный продукт в том виде, в котором его будут получать клиенты
* Специальные программные утилиты, конфигурационные файлы или данные
* требования к продукту и прочая документация

Критерии начала тестирования служат для определения готовности или неготовности к тестированию. Будет полезно составить список того, что будет использоваться в качестве входных данных, и запросить материалы, необходимые для выполнения тестов.

Критерии окончания тестирования — это то, что вы считаете необходимым для завершения процесса тестирования. Тестировщики часто стараются сделать критерии окончания тестирования условием для поставки ПО, но это не реально. Это решение принимает собственник продукта (или другое ответственное лицо).

1. Критерии остановки и требования для возобновления тестирования.

Критерии остановки/возобновления описывают ситуацию, когда тестирование невозможно продолжать из-за найденных багов. Другими словами, если дела идут так плохо, что запланированные тесты нельзя провести, тестирование нужно остановить до устранения блокирующих багов.

1. Результаты тестирования.

Чтобы засвидетельствовать результаты проведенной работы, результаты отправляются клиенту. Обычно результаты тестирования оформляют при помощи различных показателей: количество завершенных тестов, найденные баги и т.д. В некотором смысле эти показатели — индикаторы качества тестирования, хотя и не должны быть единственным критерием для оценки проделанной работы.

1. Оставшиеся задачи тестирования.

Жизненный цикл ПО может быть непредсказуемым. Иногда проверка продукта занимает больше времени, чем первоначально ожидалось. Если времени мало, некоторые части функциональности могут оставаться непроверенными. В таком случае команда включает оставшиеся задачи в тест план. Кроме того, в этом разделе можно описать масштаб необходимой работы на случай, если все задачи будут закрыты до дедлайна.

1. Требования среды.

В целом, в этом разделе описывается, что нужно для тестирования по части аппаратного обеспечения. Здесь перечисляются и инструменты, используемые для тестирования. При необходимости можно описать какое-то особое оборудование и его функционал. Например, если в связи со спецификой проекта вам потребуется использовать комплект VR или какие-то специфические устройства, которые нужно приобрести.

1. Требования по части кадров и их обучения.

Если команда должна протестировать проект из сферы, с которой они не знакомы, имеет смысл провести лекцию или краткий обучающий курс от экспертов. Это поможет тестировщикам понять особенности проекта и сделает их работу более эффективной.

1. Обязанности.

В этом разделе описываются сферы ответственности каждого члена команды QA. Удобно составить таблицу с тремя столбцами — имя, должность и обязанности.

1. Расписание.

Тест план должен также включать дедлайны. Команде нужно как-то оценивать скорость работы. Для этого им нужно знать, сколько времени отводится на тестирование. Если есть несколько этапов тестирования, нужно расписать их порядок и сроки.

1. Планирование рисков и непредвиденных обстоятельств.

Этот раздел перекликается с п. 5 «Проблемы и риски». Здесь, в дополнение к перечню рисков, предоставляются разъяснения о том, как справиться с этими рисками, и что делать в случае форс-мажорных обстоятельств.

1. Утверждение.

Каждый тест план должен содержать информацию о том, кто его составлял (имя, должность), и о том, кто его должен одобрить и дать команде зеленый свет на его использование.

1. Глоссарий.

В этом разделе можно пояснить основные термины, используемые при написании тест плана. Глоссарий помогает предотвратить неправильное толкование используемой терминологии

Виды тест планов:

Несмотря на стандартную структуру, существует несколько типов тестовых планов. Они отличаются особенностями описанных задач и объемом работ. QA-команды, как правило, используют следующую классификацию:

* Тест планы по уровням — планы модульного, интеграционного, системного, приемочного тестирования.
* Тест планы по типам — планы функционального тестирования, тестирования производительности или юзабилити, план автоматизированного тестирования и т.д.
* Мастер тест план — это комплексный план тестирования. Включает высокоуровневую информацию, которая не часто меняется в ходе тестирования и требования к которой не часто пересматриваются.

По сравнению с простым тест планом мастер тест план более статичен. Это ключевое различие. Как правило, команда проекта использует один мастер тест план и несколько более подробных тест планов для разных уровней или типов тестирования, в которых описываются отдельные модули одного приложения.

Написание тест плана требует сильных аналитических навыков, внимания к деталям, а также способности продумывать действия на несколько шагов вперед.

2.4 Тест-дизайн и тест-аналитика

Разработка тест-кейсов — важный этап в жизненном цикле тестирования. От того, насколько правильно написаны тест-кейсы, может зависеть весь процесс тестирования. Создавать эффективные тест-кейсы позволяют техники тест-дизайна. В начале тестирования мы анализируем требования: определяем, насколько они полные, чёткие, тестируемые. Когда требования проверены, проанализированы и приоритизированы, начинается этап написания тест-кейсов — это и есть тест-дизайн.

Тест-дизайн — этап тестирования ПО, на котором проектируются и создаются тестовые случаи (тест-кейсы). Они соответствуют определённым ранее критериям качества и целям тестирования. Важно, что критерии качества и цели тестирования должны быть определены до начала написания тест-кейсов. От этих критериев и целей зависит, какими будут тест-кейсы, для каких модулей они будут описаны в первую очередь, проверка каких функций будет приоритетной. Когда мы пишем тест-кейсы, одна из основных задач — создать оптимальное тестовое покрытие функциональности, то есть не допустить «слепых зон» в системе, которые не покрываются проверками.

Задачи тест-дизайна на проекте:

● максимально покрыть функциональность тестами;

● обнаружить серьёзные баги;

● сократить количество тестов, исключив непродуктивные;

● не пропустить важные тесты.

Существуют следующие техники тест-дизайна:

1. **Класс эквивалентности** — набор данных, которые обрабатываются одинаковым образом и приводят к одному результату.

Например, в требованиях есть условие для посещения онлайн-кинотеатра: «Возраст пользователя — от 16 лет и старше». Результат для пользователей, которые указывают возраст меньше 16 лет (не важно, 5 или 15), всегда должен быть одинаковым — сообщение «Извините, в связи с политикой сайта вы не можете пользоваться сервисом». Так же и со значениями от 16 и выше — не важно, какой возраст укажет пользователь (16, 23, 75, 99 лет), результат будет одинаковым: «Добро пожаловать в наш кинотеатр. Желаем приятного просмотра!» Если известно, что есть группа данных, использование которых приводит систему в одно и то же состояние, нет необходимости проверять каждое значение из этой группы отдельно. Исключения возможны, но мы не можем проверять все данные, так что приходится прибегать к подобным допущениям.

Тестирование на основе классов эквивалентности (equivalence partitioning) — техника тест-дизайна на основе метода чёрного ящика: специалист не знает, как устроена система, и проходит все шаги тестов, используя только те инструменты, которые доступны пользователю. Цель техники — обеспечить максимальную проверку всех требований тестами. Разделяя данные на классы эквивалентности и выбирая лишь несколько значений из каждого, можно существенно повысить эффективность и скорость тестирования, разрабатывать и выполнять меньше тест-кейсов. Есть два признака, что данные в тесте относятся к одному классу эквивалентности:

- Если один тест выявит ошибку, остальные, скорее всего, тоже это сделают. Если в тестах используются значения из одного класса эквивалентности, и один из тестов выявляет ошибку, остальные тесты, построенные на данных из этого класса, тоже должны обнаружить эту ошибку. Например, если онлайн-кинотеатр позволяет пользователю в возрасте 14 лет зарегистрироваться на сайте, то, вероятнее всего, регистрация будет возможна и для пользователей, указавших возраст 5, 10, 12 лет. А по требованиям это ошибка.

- Если один из тестов не выявит ошибку, остальные, скорее всего, тоже этого не сделают. Если пользователю, указавшему возраст 15 лет, было отказано в регистрации на сайте, то нет смысла перебирать все значения от 0 до 15 лет. Вероятнее всего, они тоже обработаются корректно.

Данные, которые можно расположить на числовой прямой — классы эквивалентности этих данных будут **линейными**. Их можно разбить на диапазоны с точными границами начала и конца.

**Нелинейные** классы эквивалентности — это набор неупорядоченных данных. У них нет границ, они являются частью множества данных. Пример — расширения файлов, операционные системы, группы пользователей с различными правами (пользователь, модератор, администратор) и так далее. В этом случае можно выделить только два класса эквивалентности:

● валидный — соответствует требованиям,

● невалидный — не соответствует требованиям или обрабатывается системой отличным от валидного класса образом.

Кроме чисел, на классы эквивалентности можно разбить:

● символы — они могут быть валидными (@ в адресе электронной почты) и невалидными (?, %,\*);

● длину строки — например, валидный класс от 1 до 30 знаков, невалидный — всё остальное (меньше 1 и больше 30);

● объём памяти, который необходим приложению для стабильной работы;

● разрешение экрана — всё, что меньше или больше заявленных требований к разрешению экрана, будет относиться к невалидным классам;

● версии операционных систем, библиотек — также определяются согласно требованиям. Например, приложение должно работать на ОС Windows 7, но поддержка Windows Nt уже не требуется.

● объём передаваемых данных — по требованиям. Например, если мощности сервера не позволяют обработать объём данных больше определённого значения.

Классы эквивалентности — одна из основных техник тест-дизайна. Именно с ней тестировщики и тест-дизайнеры работают чаще всего. Она сокращает число тестов (можно выбрать только несколько значений из класса эквивалентности), но к использованию нужно подходить внимательно: если неверно выделить класс эквивалентности, можно получить некорректные результаты тестирования и пропустить ошибку.

1. **Граничное значение** (border condition, boundary condition) — значение на границе классов эквивалентности.

Когда тестировщик работает с линейными классами эквивалентности (диапазонами значений), может потребоваться определить границы диапазона, чтобы точно отнести значение к конкретному классу эквивалентности. У каждого диапазона будет начальная и конечная граница — это места повышенного риска ошибок, так как разработчик может указать некорректный знак неравенства или задать ошибочную границу диапазона.

Алгоритм определения граничных значений:

- Выделить классы эквивалентности.

- Определить граничные значения этих классов.

- Определить, к какому классу будет относиться каждая граница.

- Для каждой границы провести тесты: проверить значения до границы, на ней и сразу после неё.

1. **Попарное тестирование** (pairwise testing) — техника формирования наборов тестовых данных, при которой каждое тестируемое значение каждого из проверяемых параметров хотя бы раз сочетается с каждым из тестируемых значений всех остальных проверяемых параметров.

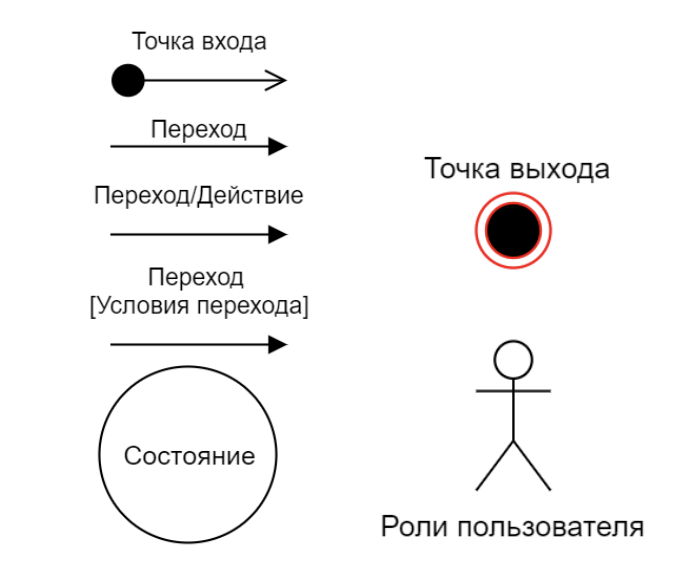
Попарное тестирование — разработка тестов методом чёрного ящика, в которой тестовые сценарии разрабатываются таким образом, чтобы выполнить все возможные отдельные комбинации каждой пары входных параметров. Попарное тестирование — техника тестирования, в которой вместо проверки всех возможных комбинаций значений всех параметров проверяются только комбинации значений каждой пары параметров.

Техника применяется на проектах, где много параметров и их значений.

1. **Тестирование на основе состояний и переходов** (State-Transition Testing) используют для фиксирования требований и описания дизайна приложения.

В проекте может быть большой набор требований с описанием состояния системы и условий, при которых она в них переходит. Без визуального представления этих состояний трудно увидеть всю цепочку событий. А это может привести к дефектам архитектуры и дизайна приложения уже на уровне требований. Техника позволяет составлять тестовые сценарии, основываясь на визуальном представлении состояний и переходов системы.

Основные понятия, которые используются при составлении диаграмм переходов и состояний:



* **Точка входа** — старт работы системы или приложения.
* **Переход** (transition) — переход системы из одного состояния в другое. Происходит в результате действий пользователя или при определённых условиях.
* **Событие** (event) — действие пользователя, которые он выполнил для перевода системы в другое состояние. Или действия самой системы, меняющие её состояние.
* **Действие** (action) — реакция приложения на действия пользователя или самой системы (на событие).
* **Условия перехода** (transition conditions) — условия, которые необходимы для перехода системы в другое состояние. Например, изменение даты для начисления процентов на вклад.
* **Состояние** (state) — состояние системы до или после перехода в результате действий пользователя или при определённых условиях.
* **Точка выхода** — успешное окончание полного цикла работы приложения, то есть выполнение всех переходов и состояний.
* **Роли пользователей** (actors) — пользователи, которые могут по-разному влиять на систему в зависимости от уровня прав доступа (зарегистрированный пользователь, менеджер, администратор).

При построении диаграмм состояний и переходов важно:

● не допускать пересечения линий переходов — это усложняет визуальное восприятие диаграммы и может привести к ошибочному переходу;

● сложные процессы лучше представлять в виде нескольких диаграмм — если охватить всё одной схемой, она будет слишком трудной для понимания;

● главную последовательность состояний следует размещать на одной горизонтальной линии, чтобы прослеживался позитивный сценарий работы системы. Дополнительные состояния можно представить в виде ответвлений и разместить по бокам от основной последовательности.

Плюсы диаграмм состояний:

● позволяют визуализировать состояния продукта;

● демонстрируют варианты переходов, которые можно пропустить;

● помогают отследить дефект, сужая его локацию до конкретного перехода;

● показывают внутреннюю механику продукта.

Минусы:

● можно пропустить неочевидные переходы;

● при слишком сложной структуре продукта диаграммы могут стать громоздкими и запутанными;

● являются только основой к применению других методов;

● бесполезны при плохом знании продукта.

1. **Таблицы принятия решений** (таблицы решений) — способ компактно представить модели со сложной логикой. А ещё это техника тестирования чёрного ящика, которая применяется для систем со сложной логикой.

Таблицы принятия решений используют, чтобы упорядочить и задокументировать сложную логику приложения, а также протестировать все комбинации условий и состояний.

Сущности, из которых состоят таблицы:

* Условия (conditions) — короткое описание входных условий (данных), сформулированное в виде вопроса. Ответ — либо «да/нет», либо ограниченный набор значений. Например: «Пользователь авторизован в системе?», «Вид документа, предоставленный клиентом, — паспорт, водительские права, загранпаспорт?»
* Действия (actions) — чёткое описание ожидаемого результата, действия системы. Формулировка действия — утвердительное предложение. Одно предложение обязательно описывает только одно действие. Например: «Данные заполнены автоматически», «Сообщение об ошибке отображается на экране».
* Значения (values) — значения, допустимые для входных данных, указанных в условии. Например: «да/нет», «паспорт, водительские права, загранпаспорт».
* Правила (rules) — комбинации входных данных, которые отражены в таблице.

1. **Исследовательское тестирование** — это подход, когда тестировщик не использует тест-кейсы, а тестирует приложение по определённому сценарию, который часто составляется прямо во время проверки.

Исследовательское тестирование не всегда относят к техникам тест-дизайна, но по его результатам могут составлять тест-кейсы.

Тестировщик проводит исследовательское тестирование приложения, в результате которого выявляются дефекты. Тот сценарий (тест), который выявил дефект, нужно задокументировать (создать тест-кейс), чтобы в дальнейшем проверять, что дефект исправлен и не появился вновь. Кроме того, стоит создать тест-кейсы (если их нет) и для проверки похожих сценариев, чтобы обнаружить другие подобные дефекты. В некоторых случаях проверки, проведённые при исследовательском тестировании, следует документировать (создавать тест-кейс), даже если они не обнаружили дефект. Это нужно, чтобы повторять проверки в будущем, в том числе при регрессионном тестировании.

Таким образом, исследовательское тестирование как техника тест-дизайна позволяет дополнять наборы тест-кейсов новыми тестами, а также создавать актуальные тест-кейсы, которые выявляют дефекты

Случаи, когда исследовательское тестирование может быть эффективным:

- Нужно быстро понять, насколько качественно выполнена новая функциональность: проверить, что в ней нет критических дефектов.

- Нужно быстро изучить тестируемый продукт (например, новому тестировщику на проекте) и получить общую информацию о его основной функциональности.

- Нужно проконтролировать работу других тестировщиков: проверить без использования тест-кейсов, что приложение работает (с позиции пользователя).

- Недостаточно времени для составления тест-кейсов.

- Отсутствуют требования, на основании которых можно составить тест-кейсы.

- Тестируется небольшой проект, для которого не требуется структурированного подхода к тестированию.

- В проекте произошли внезапные изменения, которые требуют быстрой проверки.

Плюсы исследовательского тестирования:

● не нужно тратить время на предварительное описание всех сценариев; ● не нужна поддержка тестовых сценариев;

● нет привыкания к тестовым сценариям, их прохождение не происходит «не глядя»;

● не теряется цельное видение продукта;

● критические дефекты находятся быстрее;

● повышается скорость тестирования;

● можно сразу начинать тестировать продукт, даже если требований нет вообще;

● исследовательское тестирование интереснее и креативнее (тесты ограничиваются только фантазией и глубиной знаний о продукте).

Минусы:

● сложно планировать время на проведение тестирования без задокументированных заранее сценариев;

● вероятность пропустить ключевые проверки, так как отсутствует ранжирование сценариев по степени важности;

● сложность оценки полноты покрытия требований тестами;

● требуется высокая квалификация тестировщиков и хорошее знание тестируемого приложения;

● сложно использовать для регрессионного тестирования;

● невозможно автоматизировать такое тестирование.

2.5 Баг-репорты и багтрекинговая система

Дефект (баг) — это отклонение фактического результата от ожидаемого.

По легенде, термин «баг» впервые был применён 9 сентября 1947 года, когда учёные Гарвардского университета, тестировавшие вычислительную машину Mark II Aiken Relay Calculator, нашли мотылька, застрявшего между контактами электромеханического реле. Именно тогда Грейс Хоппер произнесла этот термин. Насекомое достали и вклеили в технический дневник, сопроводив надписью: «First actual case of bug being found» (англ. «первый реальный случай, когда жук был найден»).

Как определить, что поведение, которое мы видим, некорректное? Нужно опираться на требования. Но не у всех проектов есть документация: требования, спецификации и технические задания. Поэтому тестировщику приходится опираться:

● на предыдущий опыт тестирования подобных приложений;

● на критерии качества, работающие на рынке;

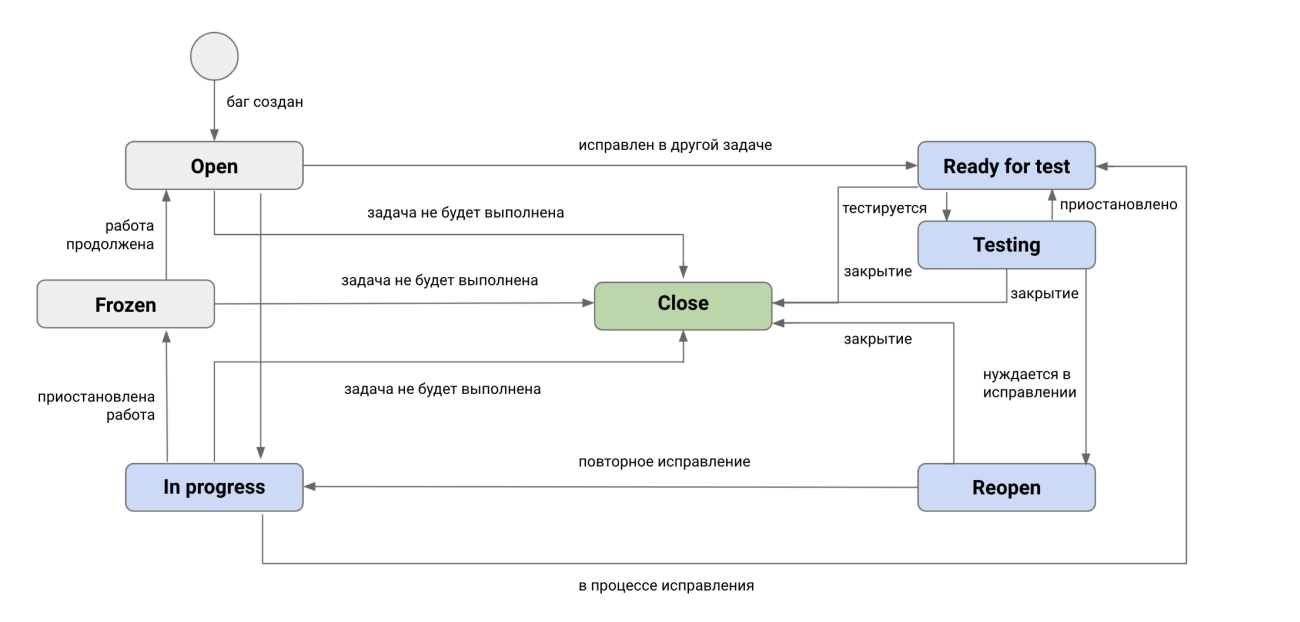
● на логику и здравый смысл.

Причины появления дефектов:

* Проблемы в коммуникациях между членами команды. Если команда не работает слаженно, то разработка тормозится или идёт с ошибками. Разработчик не понял требований, задал вопрос аналитику, но не получил ответ или получил его слишком поздно — вероятно, в системе появились дефекты. Тестировщик обнаружил отклоняющееся поведение, но не нашёл документов с описанием «как надо», не дождался ответа разработчика или заказчика — есть риск получить инцидент.
* Сложность программного обеспечения. Чем сложнее программа, тем больше вероятность появления ошибок. Сложная логика работы, множество взаимодействующих систем, нетривиальные алгоритмы — всё это элементы повышенного риска появления дефектов.
* Изменение требований. ПО подстраивается под меняющийся мир и потребности пользователя. Изменения требований приводят к модификации кода, и чем позже изменились требования, тем больше работ выполняется для трансформации системы. Правки влекут за собой новые дефекты или возврат уже исправленных.
* Ошибки программистов. Разработчики должны писать и документировать код по установленным правилам. Но нехватка времени вынуждает их работать быстро, поэтому срабатывает фактор невнимательности и забывчивости.
* Ошибки тестировщиков. Тестировщики не замечают ошибки из-за недостатка опыта, упускают проблемы из-за сложности программного обеспечения и невозможности исчерпывающего тестирования.

Жизненный цикл дефекта.

Тестировщик создаёт новый дефект в системе, затем его чинит разработчик, и, если работоспособность ПО восстановилась, тестировщик закрывает дефект. Эти этапы составляют жизненный цикл дефекта.



Этапы жизненного цикла дефекта:

● Open — специалист создал баг-репорт.

● In progress — баг назначен на разработчика, разработчик начал его исправлять.

● Ready for test — разработчик всё поправил, задача готова к тестированию.

● Testing — тестировщик взял задачу в тестирование.

● Reopen — тестировщик проверил, задача нуждается в доработках.

● Close — задача работает, как описано в ТЗ.

● Frozen — разработка задачи приостановлена из-за изменений в продукте.

Отчёт о дефекте.

Чтобы сообщить о дефекте, тестировщик описывает его установленным образом. В результате получается отчёт о дефекте, или баг-репорт (bug report). Для краткости его называют «баг».

Отчёт о дефекте — это документ, который описывает шаги воспроизведения дефекта, фактический и ожидаемый результат, серьёзность дефекта и приоритет устранения.

Отчёты о дефектах — инструменты для сбора статистики на проекте. Они помогают определить:

● в каких областях приложения;

● при каких условиях концентрируются дефекты.

Важная функция отчётов о дефектах — приоритизация проблем. Если дефектов много, и времени на все не хватает, разработчику нужно понимать, какие из них хуже всего влияют на работу приложения, и что исправить в первую очередь. Отчёт о дефекте предоставляет важные подробности для понимания сути случившегося, анализирует причины возникновения проблемы и даёт рекомендации для исправления.

Атрибуты отчёта о дефекте:

1. Уникальный идентификатор (ID) — присваивается автоматически.

2. Тема (заголовок) — кратко сформулированная суть дефекта по правилу «Что? Где? Когда?».

Название должно содержать ответы на вопросы:

- Что происходит или не происходит согласно спецификации или представлению тестировщика о нормальной работе продукта?

- В каком месте интерфейса пользователя или архитектуры программного продукта находится проблема?

- В какой момент работы программного продукта, при наступлении какого события или при каких условиях проявляется проблема?

Название содержит предельно краткую, но достаточную для понимания сути проблемы информацию о дефекте. Оптимальная длина — до 10 слов и 80 символов. В названии не употребляются слова «некорректно», «неправильно», «ошибка» и так далее. Тестировщик описывает, что именно некорректно, и в чём именно ошибка.

3. Подробное описание — более широкое описание сути дефекта (может быть, может не быть).

4. Предусловие — описывается подготовка системы для воспроизведения дефекта.

5. Шаги для воспроизведения — последовательное описание действий, которые привели к выявлению дефекта. Описываются максимально подробно с указанием конкретных вводимых значений.

Шаги по воспроизведению — это руководство к действию для тех, кто будет решать проблему. Составляются по правилам:

- Вернуться к началу. Первый шаг — указание, по какой ссылке перейти, какое окно открыть и так далее.

- Шаг отвечает на вопрос «Что сделать?».

○ нажать кнопку «Найти»;

○ ввести валидный email и пароль;

○ заполнить требуемые поля валидными данными.

- Количество шагов — от 2 до 8. Если больше, нужно подумать, какие шаги лишние.

- Последний шаг — указание, на что обратить внимание:

○ осмотреть текст в выпадающем списке (подменю);

○ обратить внимание на Profile-форму.

6. Фактический результат — указывается, что работает не так, в каком месте продукта и при каких условиях.

7. Ожидаемый результат — указывается, как именно должна работать система, по мнению тестировщика, основанному на документации.

8. Вложения — скриншоты, видео или лог-файлы.

9. Серьёзность дефекта — влияние на работоспособность приложения.

Серьёзность показывает степень ущерба, который наносится проекту дефектом.

1. Блокирующий (Blocker) — функция не работает.

2. Критический (Critical) — функция работает, но с ограничениями.

3. Значительный (Major) — функция работает, но неправильно.

4. Незначительный (Minor) — функция работает,но неудобно.

5. Тривиальный (Trivial) — грамматические ошибки в пользовательской документации.

10.Приоритет дефекта — влияние на очерёдность выполнения задачи или устранения дефекта. Чем выше приоритет, тем быстрее нужно исправить дефект.

Приоритет (срочность) показывает, как быстро нужно устранить дефект. Срочность — высокая (High), средняя (Medium), низкая (Low). Порядок исправления ошибок по их приоритетам:

1. High – высокий, дефект нуждается в срочном устранении.

2. Medium – средний, дефект нуждается в устранении, но это не срочно.

3. Low – низкая срочность исправления дефекта.

11.Статус — текущее состояние дефекта: «открыт», «в работе», «исправлен» и так далее.

12.Окружение включает в себя:

- Компонент или среда — на какой платформе или операционной системе дефект воспроизводится: iOS, Android, Windows, Mac.

- Версия — на каком этапе разработки программного продукта обнаружился дефект.

- Назначение — кто ответственен за починку.

- Номер сборки, в которой обнаружился дефект.

Правила хорошего баг-репорта

1. Следовать правилу «Что? Где? Когда?».

2. Одна ошибка — один отчёт о дефекте.

3. Краткость — сестра баг-репорта.

4. Писать техническим языком с применением терминологии, принятой на проекте.

5. Прикреплять дополнительные файлы: логи, скриншоты, видео.

6. Прикреплять ссылки к требованиям, чтобы избежать споров.

7. Избегать дубликатов дефектов — прежде чем сохранить отчёт о дефекте, надо проверить в баг-трекере, есть ли уже такой дефект.

8. Указывать версию ПО и тестовый стенд (окружение), на котором обнаружился дефект.

9. Воспроизводить дефект, следуя собственным шагам.

Багтрекинговая система

Система отслеживания ошибок (от англ. bug tracking system) — прикладная программа, созданная, чтобы разработчикам ПО (программистам, тестировщикам и другим) было проще:

● учитывать и контролировать ошибки и неполадки, найденные в программах, а также пожелания пользователей;

● следить за устранением этих ошибок и выполнением или невыполнением пожеланий.

Багтрекинговые системы используются для создания отчётов о дефектах и для управления ими. Они также применяются при создании задач для команды разработки.

Примеры Багтрекинговых систем:

🐞 Jira

🐞 BugZilla

🐞 Redmine

🐞 YouTrack

🐞 и множество других!

Глава 3. Проведение тестирования и разработка предложений по улучшению тестирования ресторана **Rostic’s**.

3.1 Функциональное тестирование

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

3.1.1 Варианты прерывания приложения

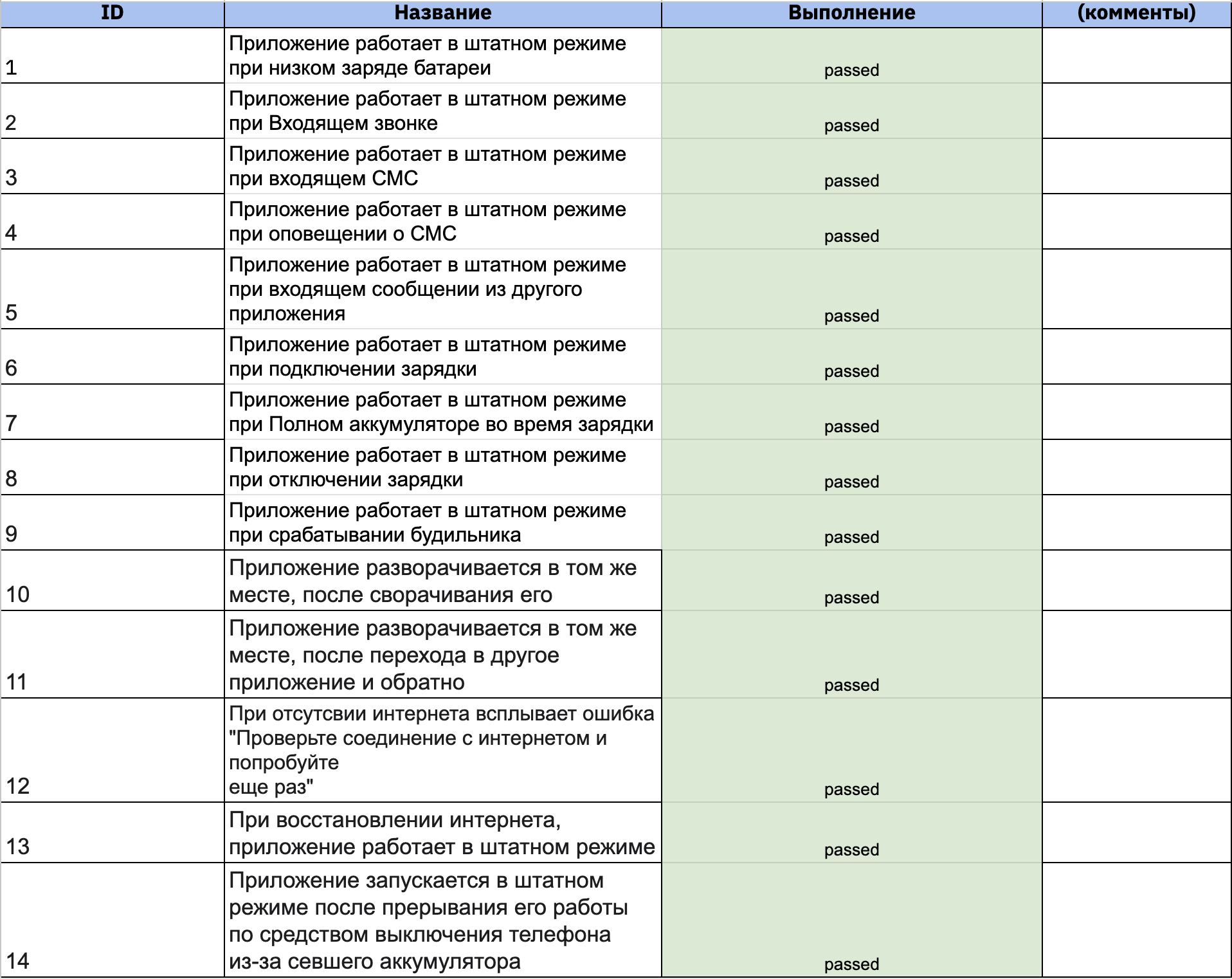
Этот вид мобильного тестирования проверяет, как приложение реагирует на неожиданное прерывание. В зависимости от характера прерывания приложение должно приостановиться, а затем вернуться в исходное состояние или даже отреагировать определенным образом. Виды прерываний зависят от тестируемого приложения, но некоторые общие прерывания, которые следует учитывать при тестировании, следующие:

1. Входящие телефонные звонки во время работы приложения
2. Входящие SMS во время работы приложения
3. Разряженная батарея, когда приложение запущено
4. Устройство подключено к сети или не заряжается во время работы приложения
5. Устройство выключается при запущенном приложении
6. Обновление ОС происходит во время работы приложения
7. Потеря и восстановление сети во время работы приложения.

Тестирование на прерывание гарантирует, что приложение справляется с прерываниями без сбоев и аномалий. При использовании реальными пользователями каждое приложение должно работать слаженно вместе с другими функциями устройства. Все приложения должны быть оптимизированы для работы с этими функциями устройства во время его использования.

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результат проведения Тестирования прерывания приложения Rostic’s:



Ссылка на Чек-лист:

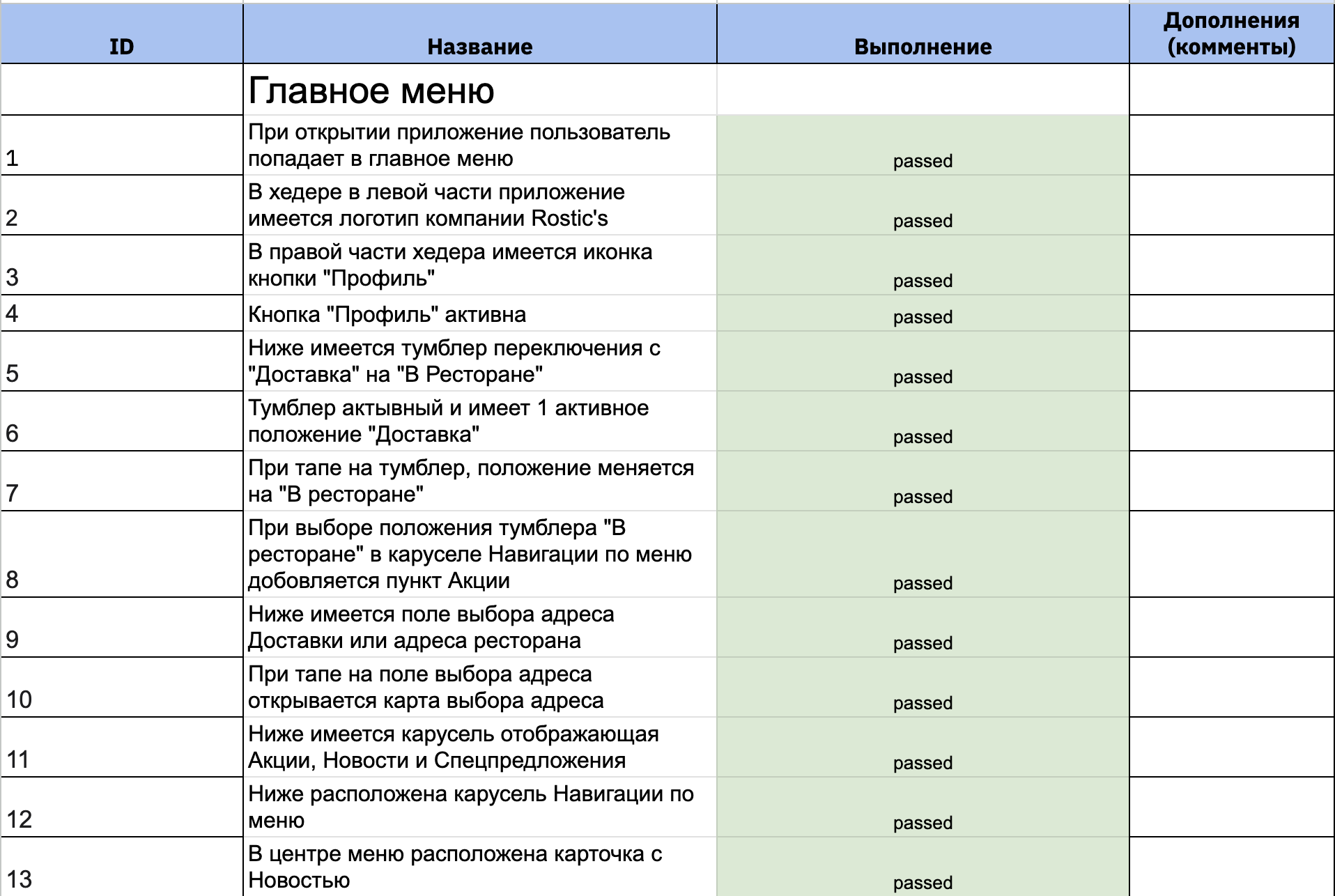
[Тимошенко Тестирования прерывания мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1z06Mr6ZONUPwvGkk501aVRlSnIe_b07NM5AWHdwFXvU/edit?usp=sharing)

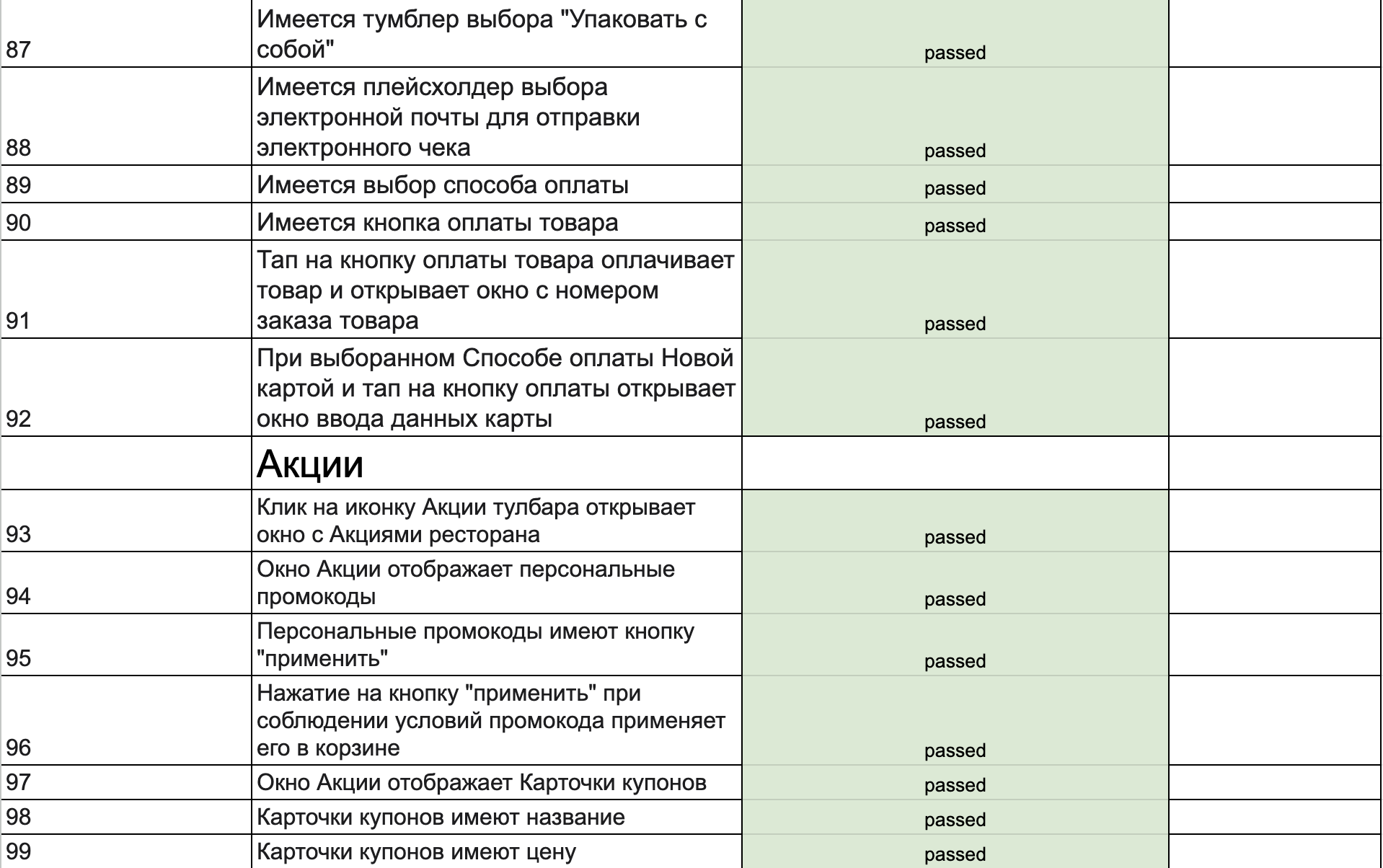
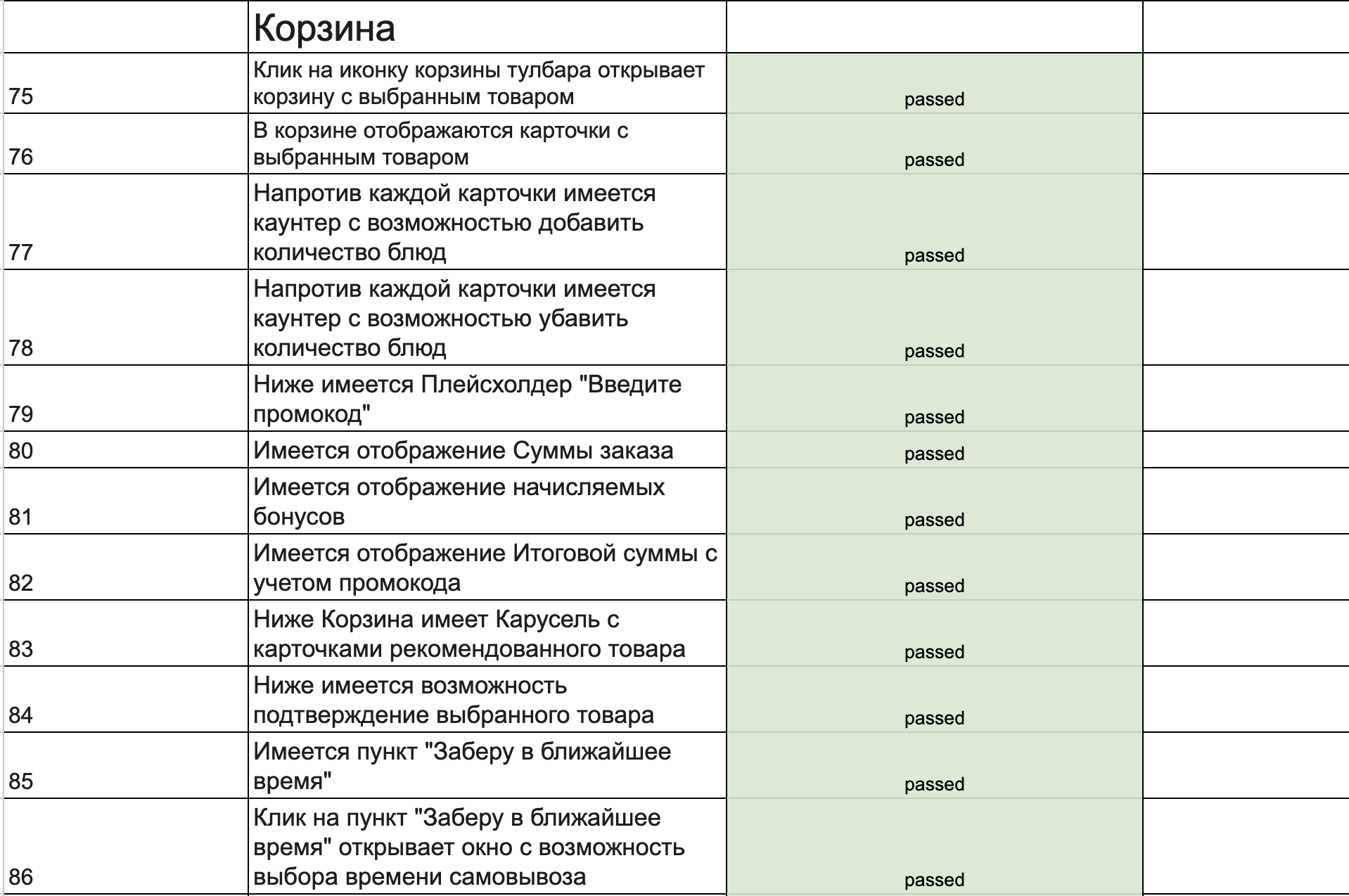
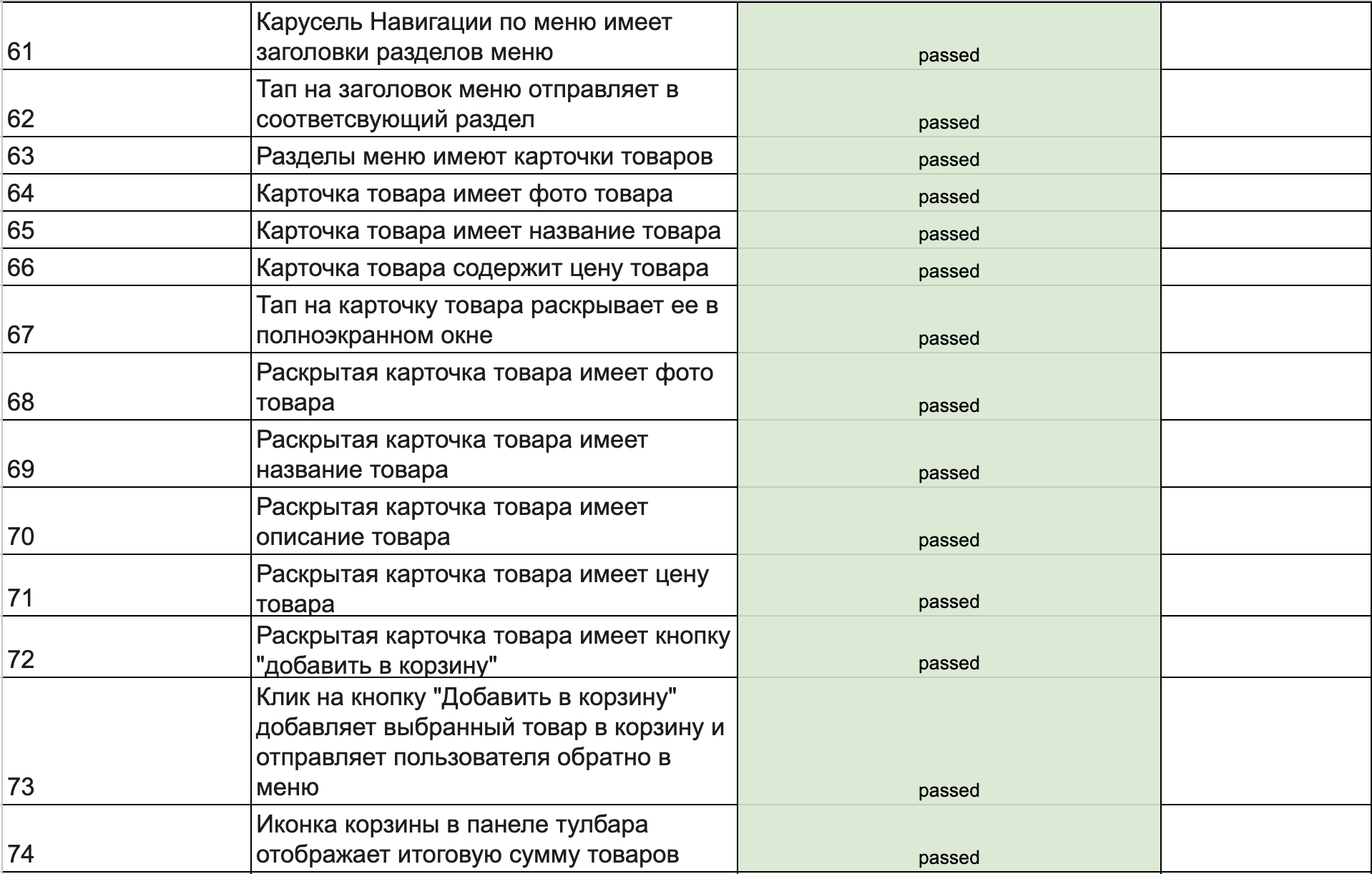
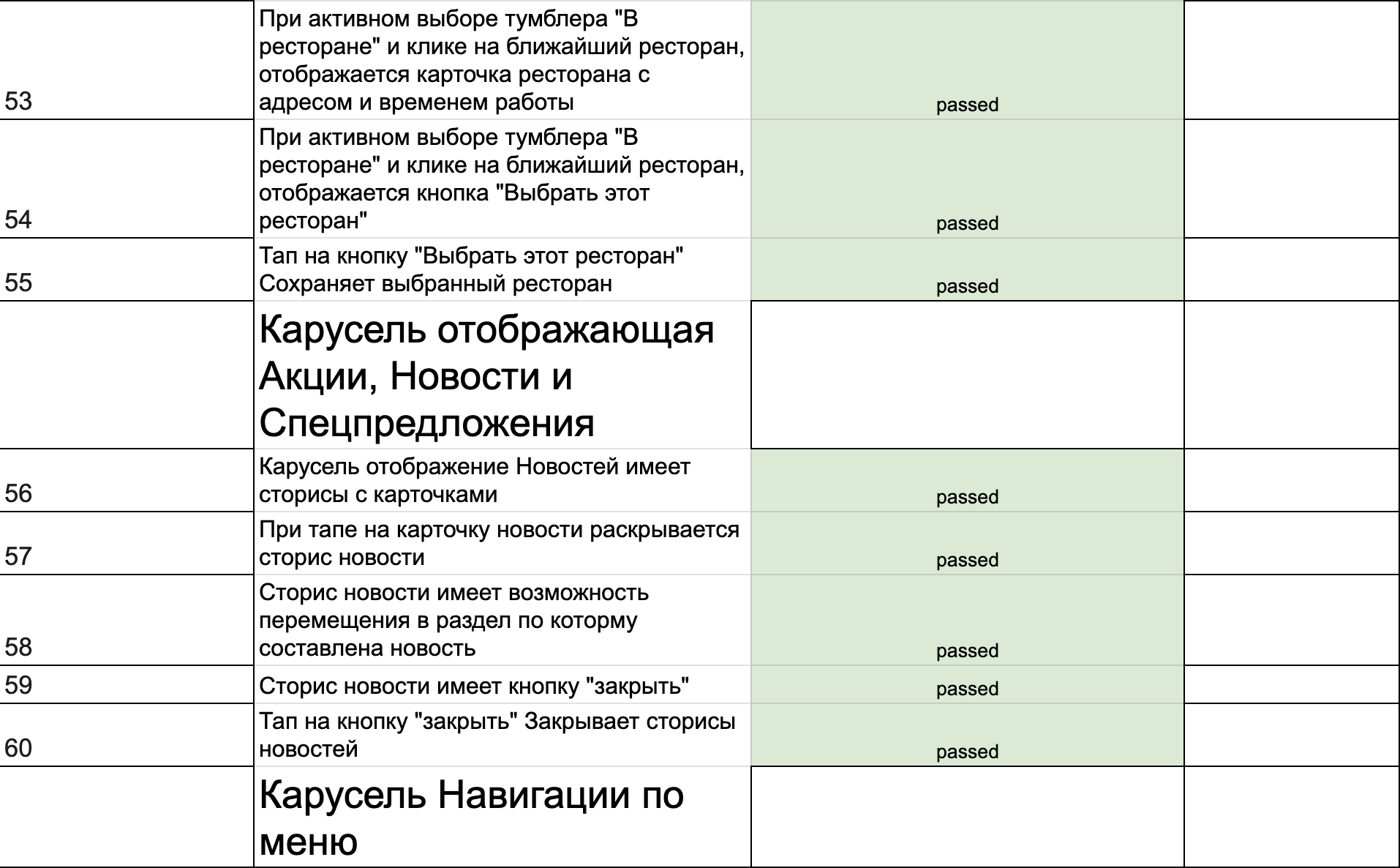
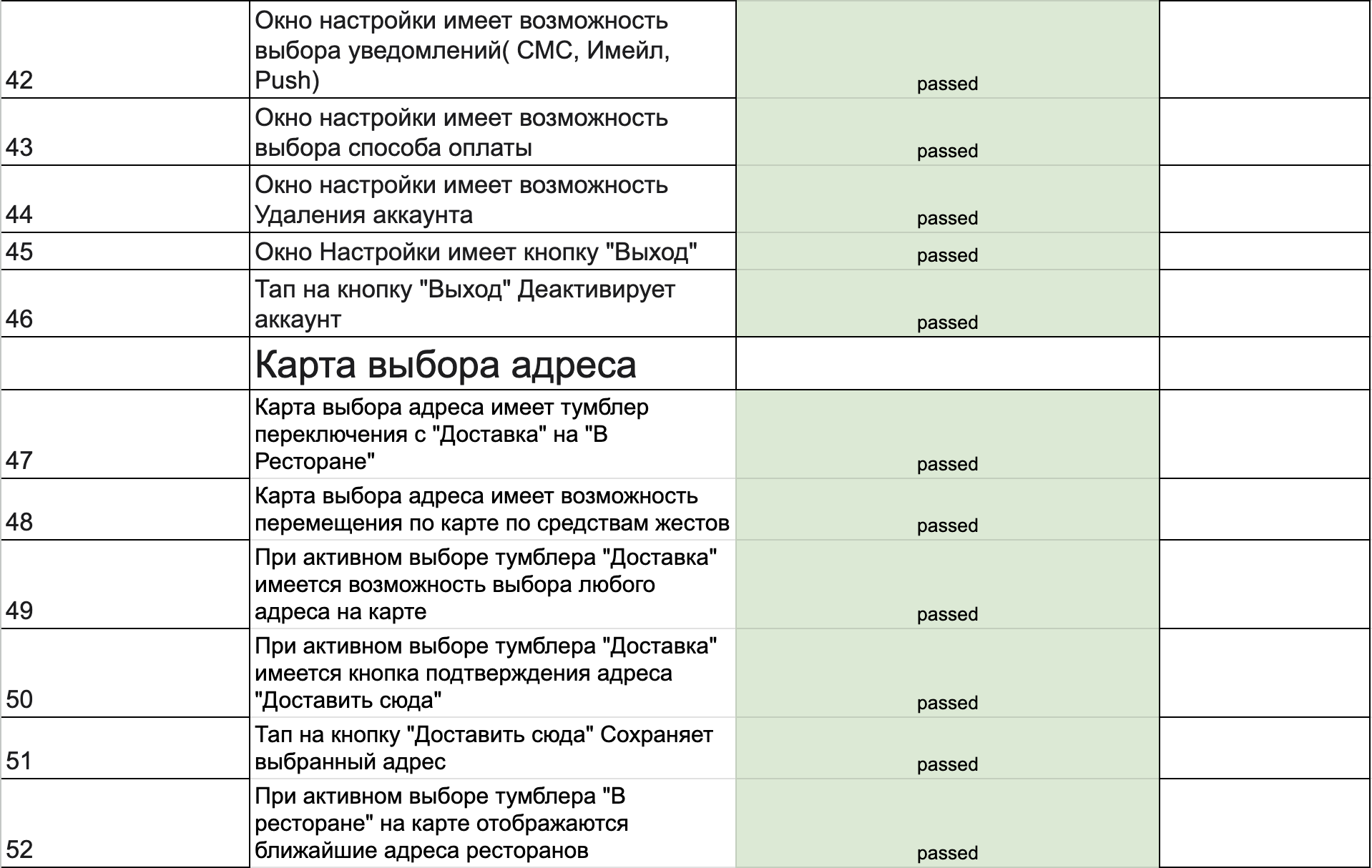
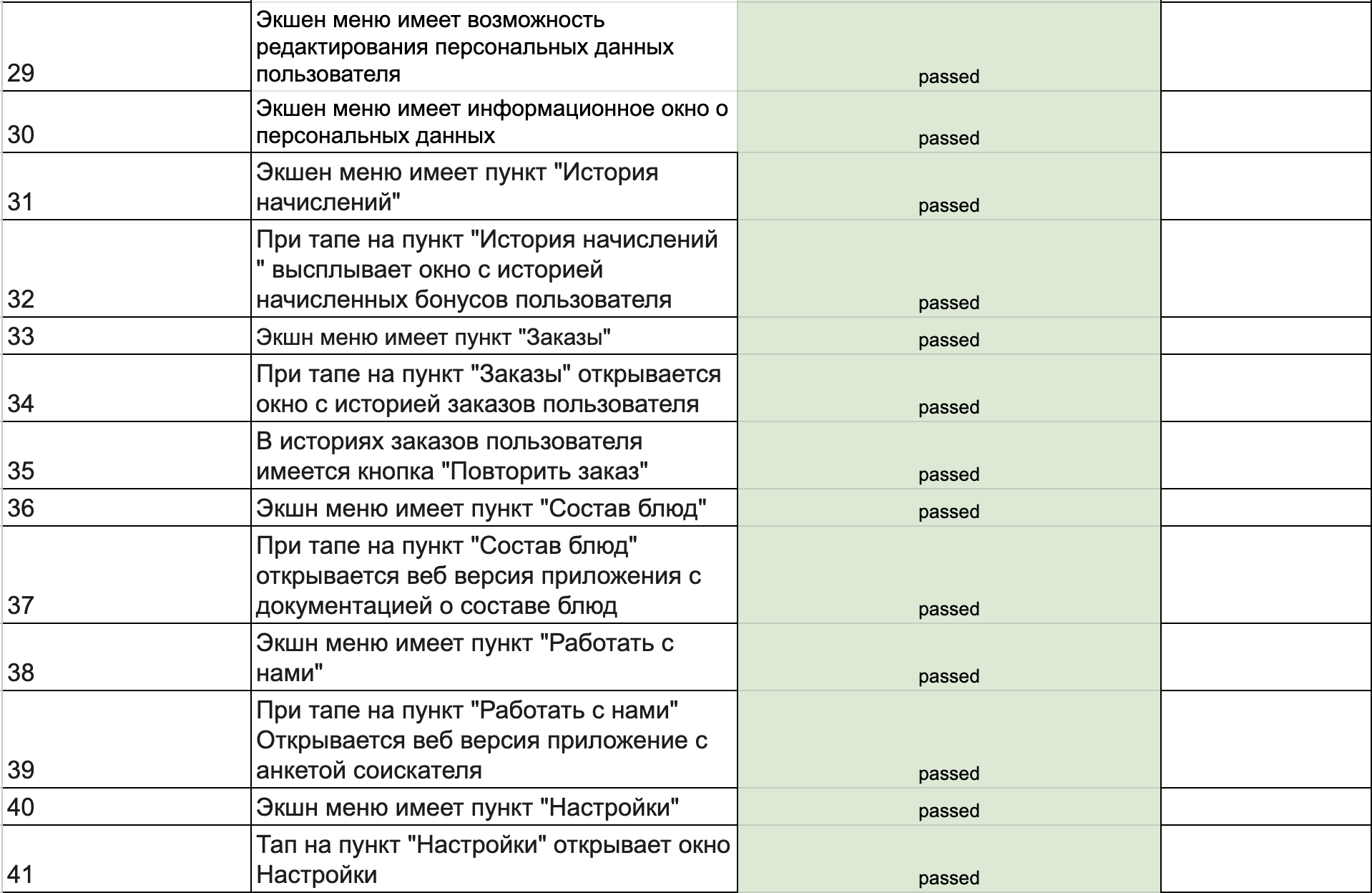
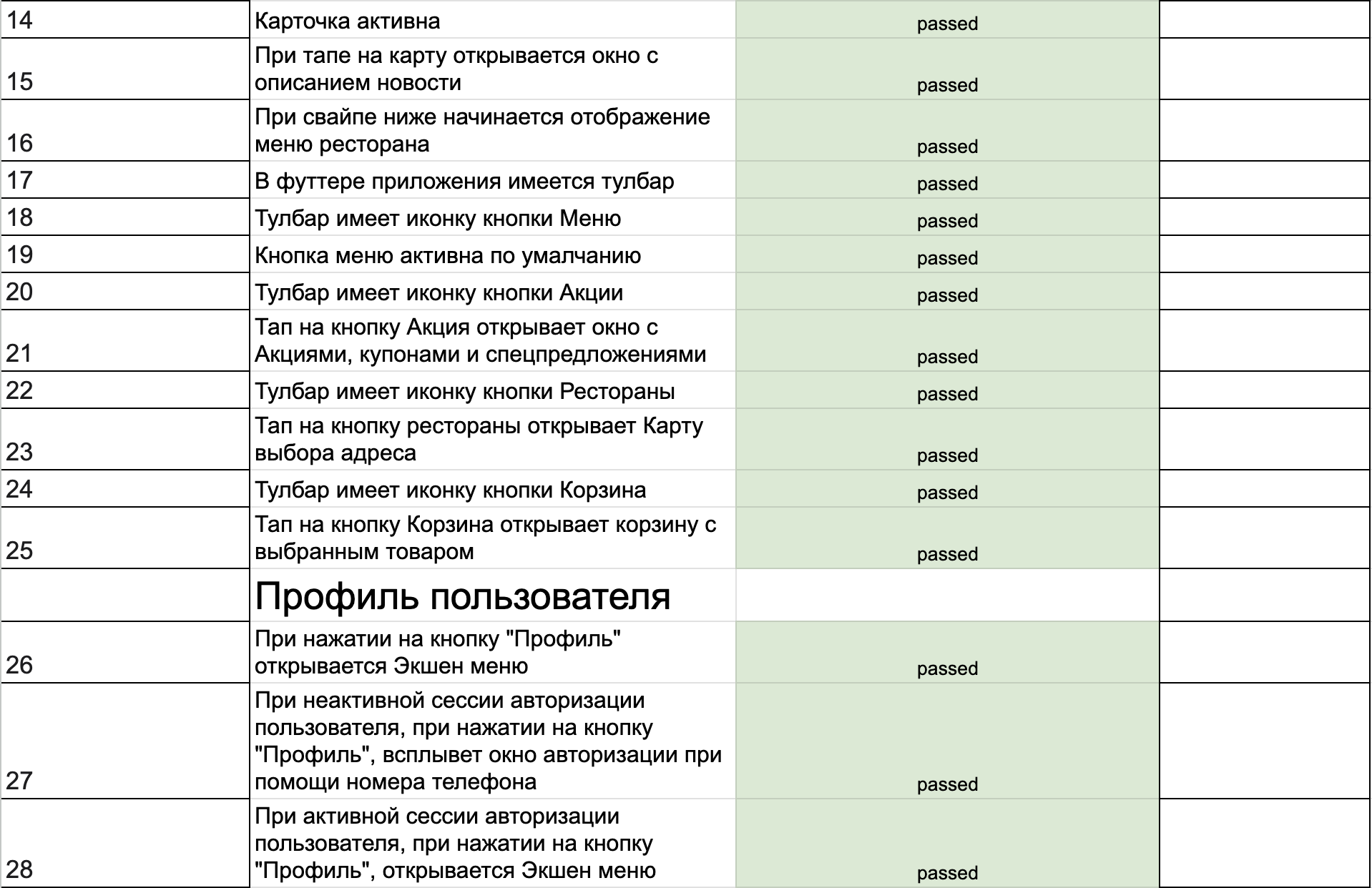
3.1.2 Тестирование интерфейса

Тестирование пользовательского интерфейса (UI testing) является одним из типов тестирования ПО, направленным на проверку корректности работы и удобства использования пользовательского интерфейса приложения. Важность данного вида тестирования обусловлена тем, что качество пользовательского интерфейса напрямую влияет на удовлетворенность пользователей и их готовность продолжать использовать приложение.

Тестирование пользовательского интерфейса включает в себя проверку следующих аспектов:

1. Внешний вид интерфейса: проверка соответствия дизайна макетам, правильность отображения на разных устройствах и разрешениях, соблюдение стилей и цветовой гаммы.
2. Элементы управления: проверка корректной работы кнопок, полей ввода, выпадающих списков и других элементов управления, а также их доступности для пользователя.
3. Навигация: проверка простоты и понятности перемещения между разделами приложения, наличие хлебных крошек, работоспособность ссылок и меню.
4. Отзывчивость: проверка скорости загрузки страниц, отсутствия зависаний и ошибок при взаимодействии с интерфейсом.
5. Сообщения об ошибках и подсказки: проверка наличия и корректности отображения сообщений об ошибках, а также наличия подсказок и инструкций для пользователя.





Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

При проведении тестирования были задействованы следующие техники тест-дизайна: 1. Эквивалентное разбиение, 2. Определение граничных значений, 3. Попарное тестирование. Результат тестирования:

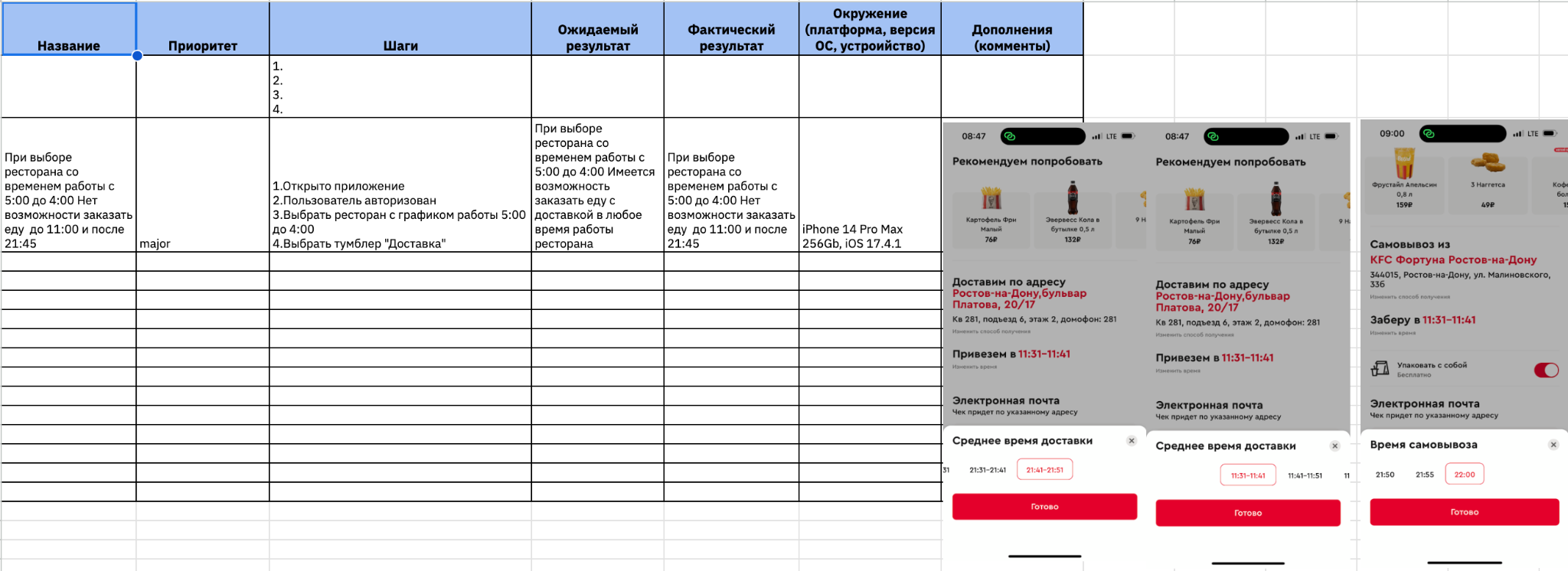


Ссылки на Чек-листы:

[Тимошенко Тестирования Интерфейса мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1yfI01vYDKDedybJiHQlaHcKcjQFf1eVtToyz_Fh8aUw/edit?usp=sharing)

[Тимошенко Тестирование Техник тест-дизайна мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1PKcHrb3hM9H_04Z32Qgjtgy5BuD82_TtdM9_3kSqJjg/edit?usp=sharing)

По результатам тестирования заведен баг репорт:



Ссылка на баг-репорт:

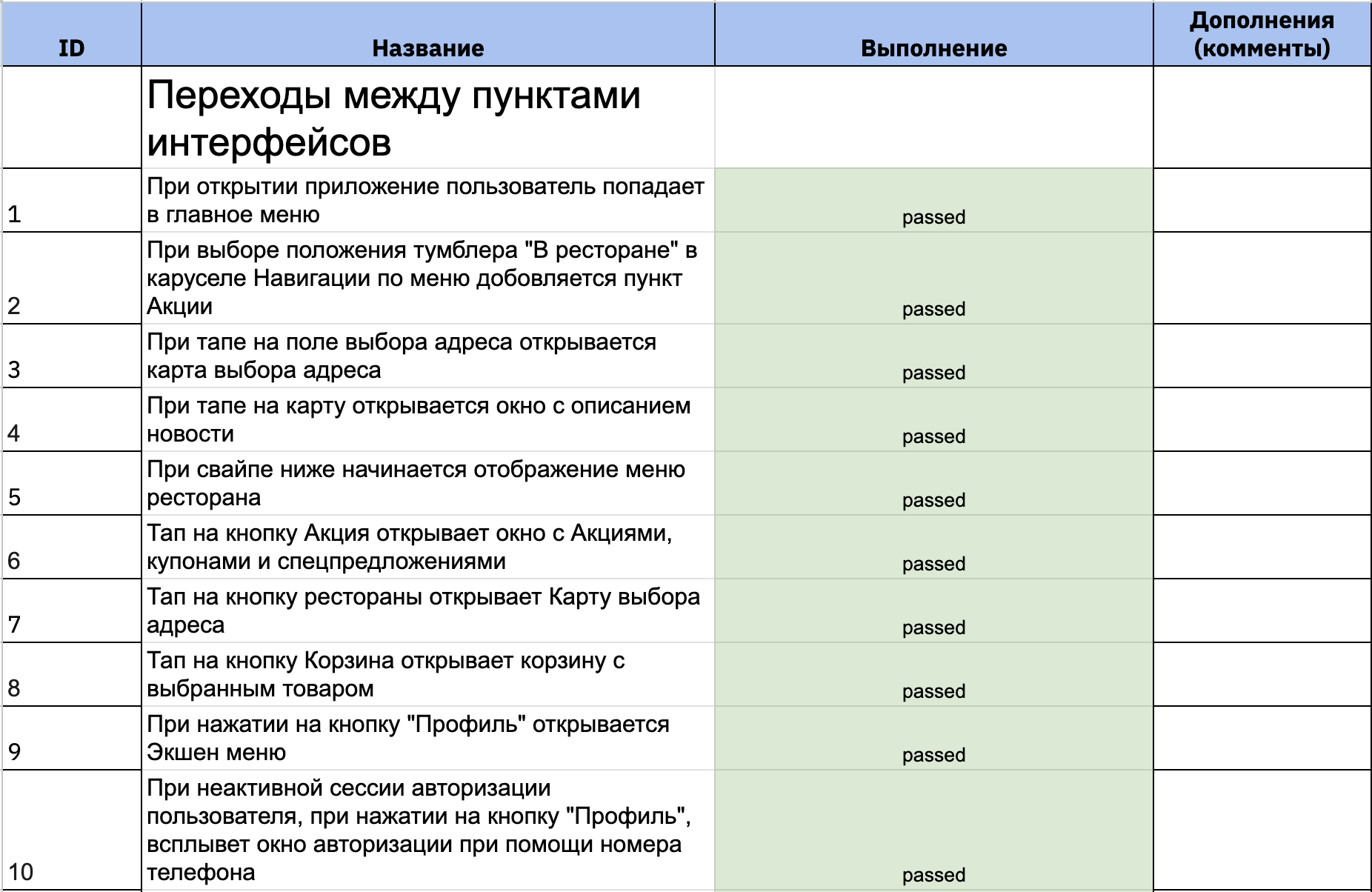
[Тимошенко Тестирование Техник тест-дизайна мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1PKcHrb3hM9H_04Z32Qgjtgy5BuD82_TtdM9_3kSqJjg/edit?usp=sharing)

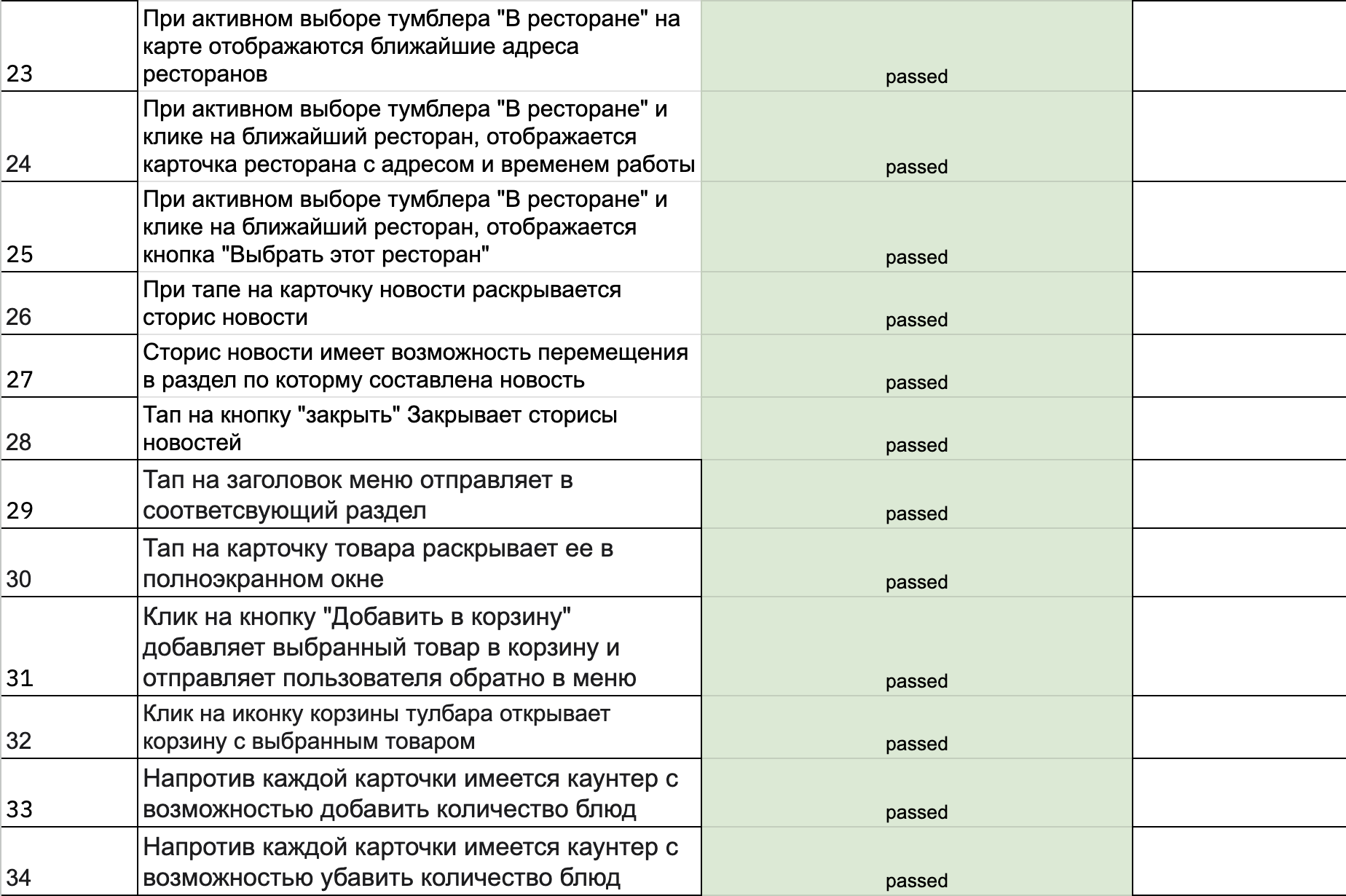
3.1.3 Навигация

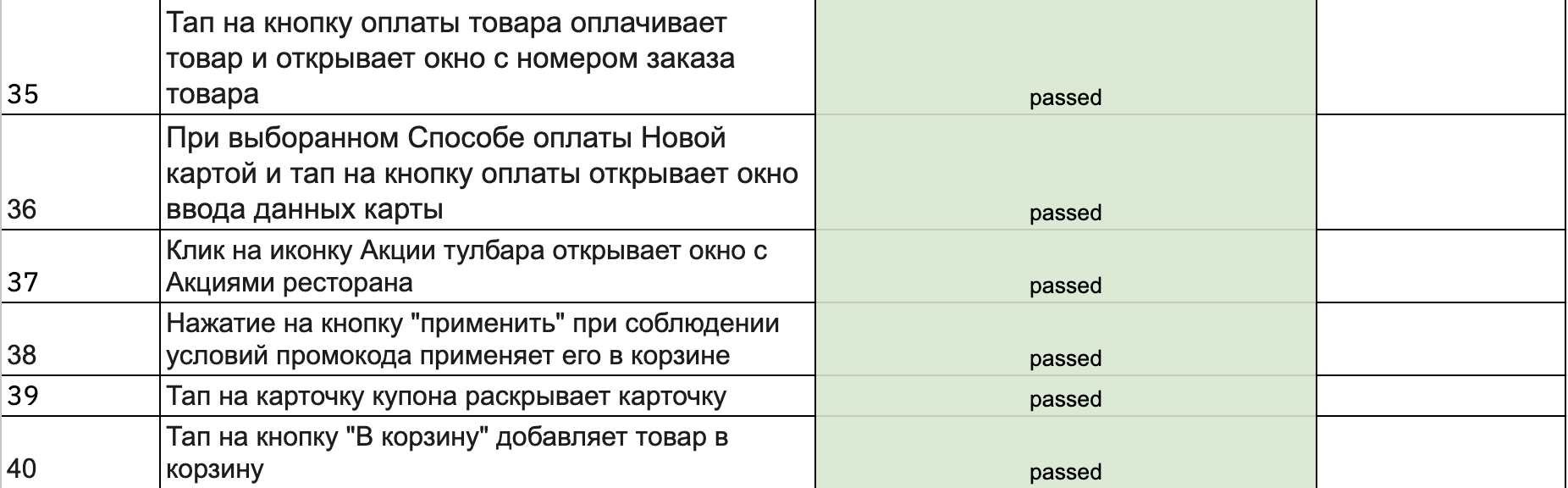
Навигация — это то, как пользователи взаимодействуют с приложением, дизайном и переходят от одного пункта интерфейса к другому. Ее можно сравнить с дорожной системой: в приложении есть десятки магистралей, которые направляют пользователей в нужное им направление.

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результат проведения Тестирования навигации приложения Rostic’s:







Ссылка на Чек-лист:

[Тимошенко Тестирования Навигации мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1P4Sq7kTziLvQpLD2zO8CtsvOudjLkxvGkKimM0otkkI/edit?usp=sharing)

3.1.4 Уведомления

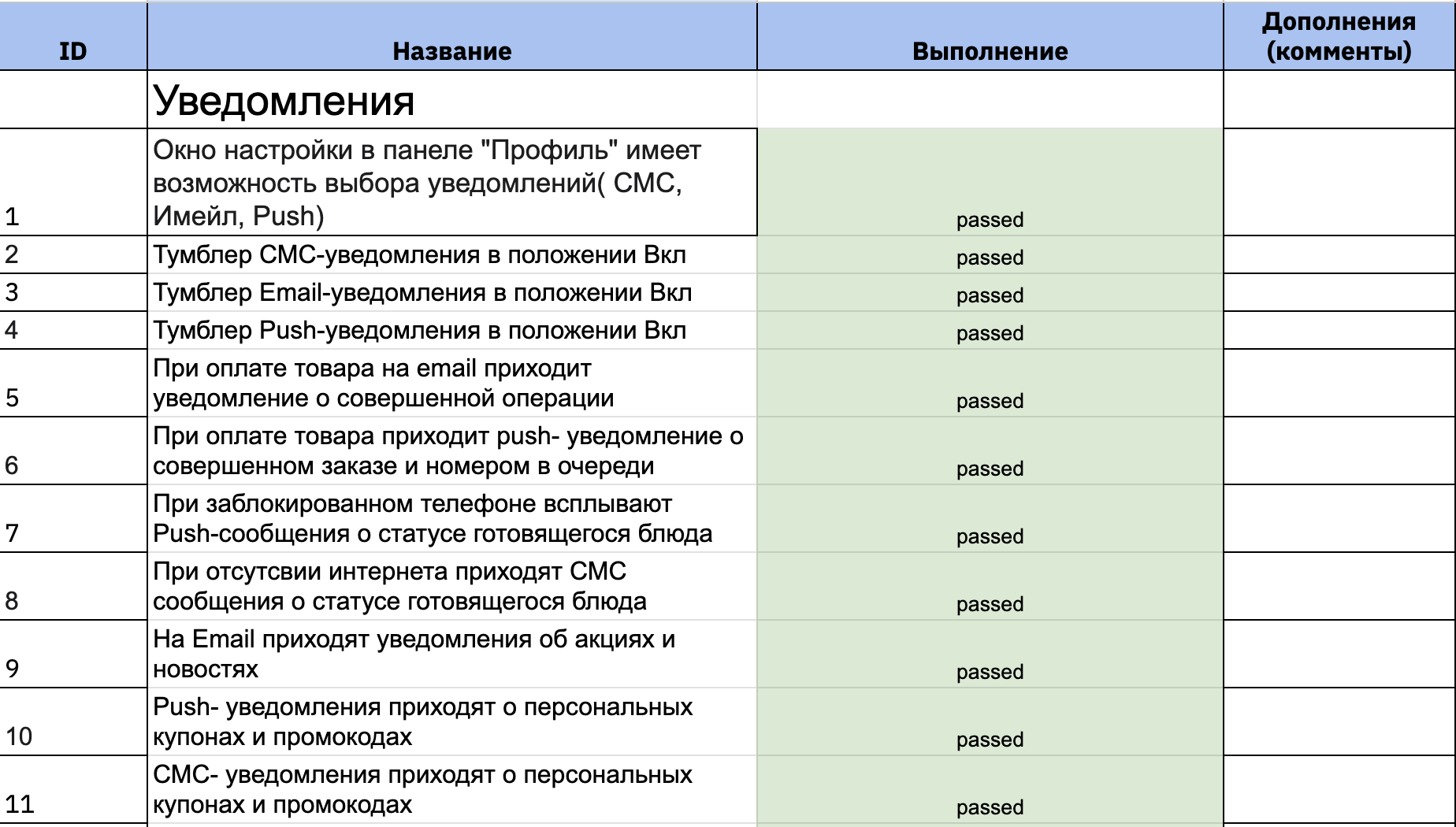
Сложно представить мобильное приложение без уведомлений. Магазины присылают уведомления об акциях и статусах заказа, мессенджеры — о новых сообщениях, игры — о новых уровнях и так далее. Уведомления бывают нескольких типов. Одни приходят с сервера, другие — по таймеру. Во втором случае неважно, есть ли на устройстве интернет.

В случае приложения Rostic’s в настройках есть 3 типа уведомлений:

1. Push-уведомления
2. Email
3. СМС-уведомления

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результат проведения Тестирования уведомлений приложения Rostic’s:



Ссылка на чек-лист:

[Тимошенко Тестирование Уведомлений мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1_B5MzaF2ctsAHqER5w0Ws1bkCctsdYnlt_GjTBfqoUw/edit?usp=sharing)

3.1.5 Разные условия сети

Часто приложение завершает работу и крашит 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 поверить:

1. Переключение в режим полета во время работы приложения. Еще одна хорошая проверка — включение режима полета во время скачивания файлов или загрузки приложения.

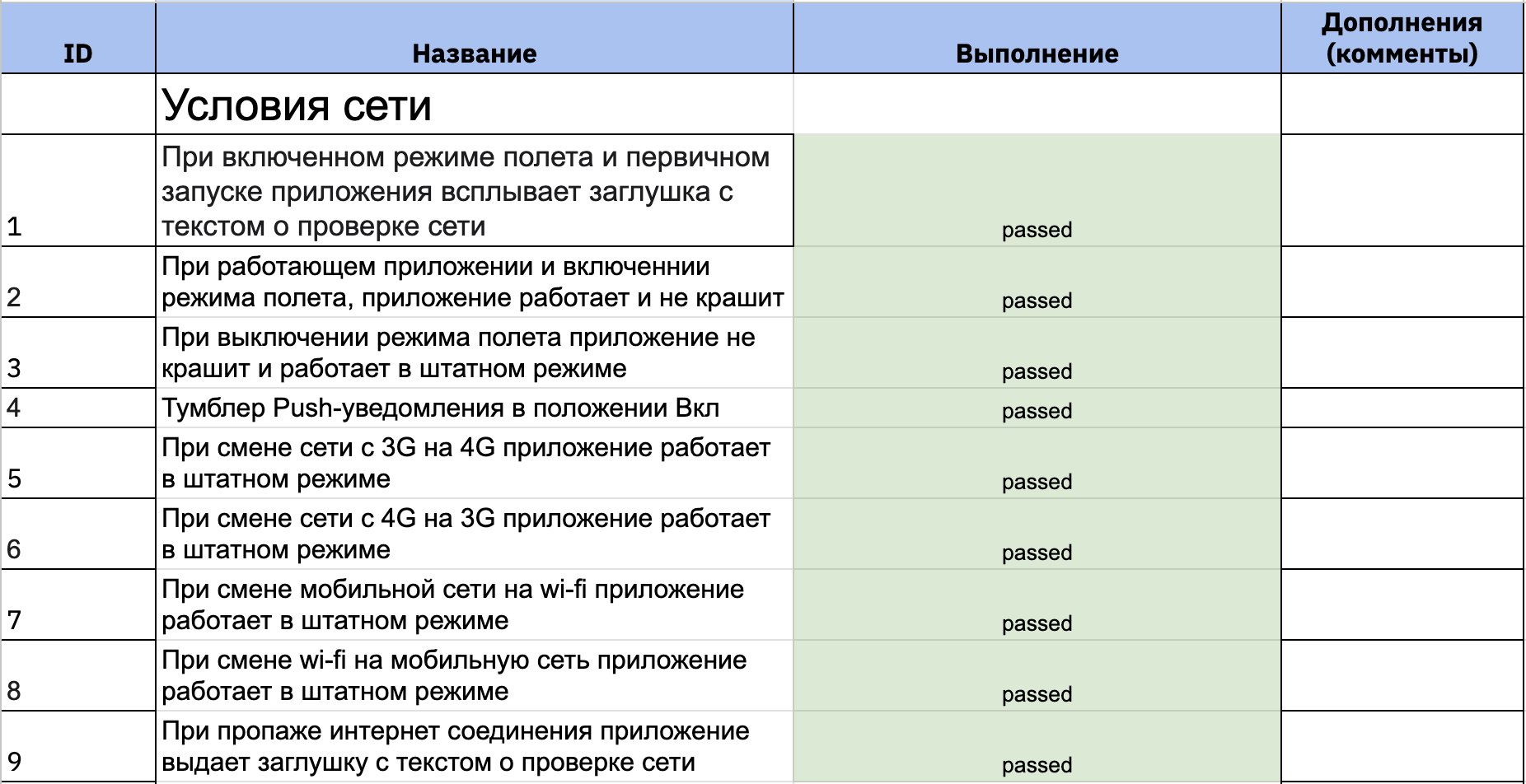
2. Изменение сети с 4G на 3G (5G) и наоборот.

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.

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результат проведения Тестирования условий сети приложения Rostic’s:



Ссылка на чек-лист:

[Тимошенко Тестирование условий сети мобильного приложения KFC](https://docs.google.com/spreadsheets/d/159FygSItna16UKEhzO6fLwoY7H2MUl9OLWAQ5KfGsQ0/edit?usp=sharing)

3.1.6 Разрешения

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 отрабатывает ситуации, когда пользователь не дает доступ к функции. Корректные сценарии отработки должны быть предусмотрены требованиями.

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

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результат проведения Тестирования разрешений приложения Rostic’s:



Ссылка на чек-лист:

[Тимошенко Тестирование условий сети мобильного приложения KFC](https://docs.google.com/spreadsheets/d/159FygSItna16UKEhzO6fLwoY7H2MUl9OLWAQ5KfGsQ0/edit?usp=sharing)

3.1.7 Отрабатывание жестов

Жесты как способ управления электронным устройством появились еще в начале 80-х, а с начала нулевых они распространились повсеместно. Дизайнеры пытаются сделать жесты интуитивно понятными — но этот процесс все еще на стадии бета-тестирования.

Стандартный набор жестов для большинства приложений:

Клик(или тап)

Дабл-тап

Перетаскивание

Щипок

Стягивание

Растягивание

Нажатие

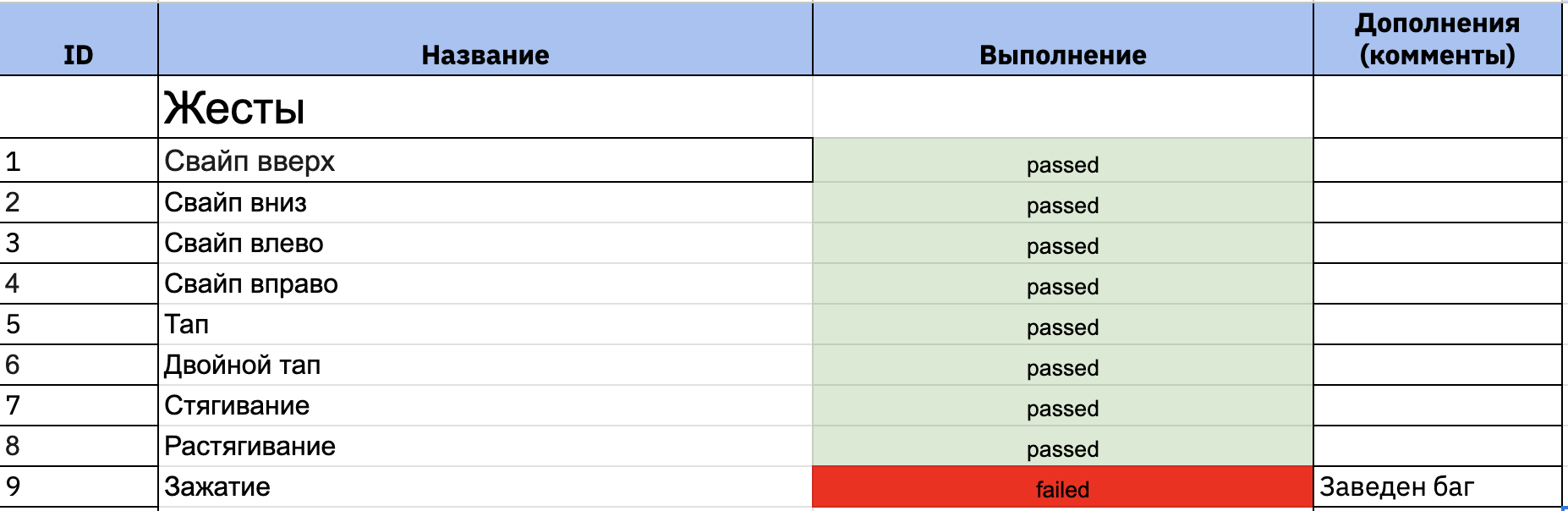
Нажать и зажать

Нажать и перетащить

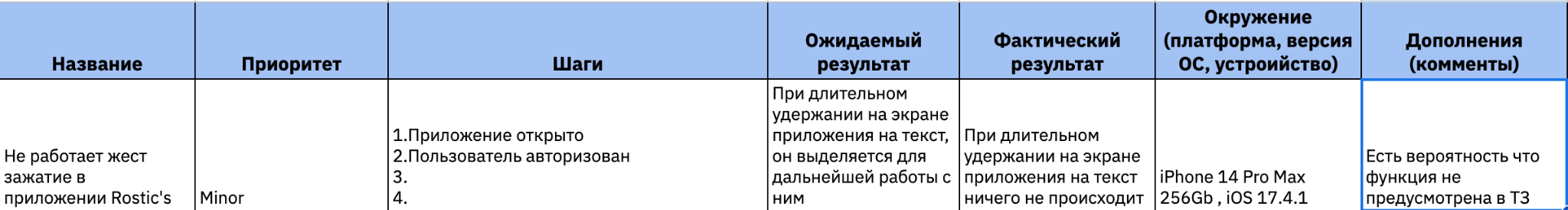
Вращение экрана

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro.

Результат проведения Тестирования жестов приложения Rostic’s:



Заведен баг-репорт:



Ссылка на чек-лист:

[Тимошенко Тестирование жестов мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1Y2-Tdwa1kZ2Mi2nsgvwEc2uSLm2F2n2GlMt3MBsqy00/edit?usp=sharing)

3.2 Нефункциональное тестирование

3.2.1 Тестирование установки

Установочное тестирование

Установочное тестирование проверяет, что процесс установки проходит гладко и пользователь не сталкивается с трудностями. Тестирование установки проверяет, можно ли без ошибок установить и удалить мобильное приложение, корректно ли устанавливаются обновления. Мы можем установить приложение, как опубликованное в магазине (App Store, Google Play), так и не опубликованное. Во втором случае сделать это можно через:

● закрытые тесты в Google Play либо Testflight на iOS;

● сервисы дистрибуции;

● компьютер

Чек-лист проверок:

* Проверка установки.
* Проверка прерывания установки.
* Проверка переустановки приложения — важно учитывать, должно ли приложение сохранять данные в памяти телефона после удаления.
* Проверка удаления.
* Проверка обновления.

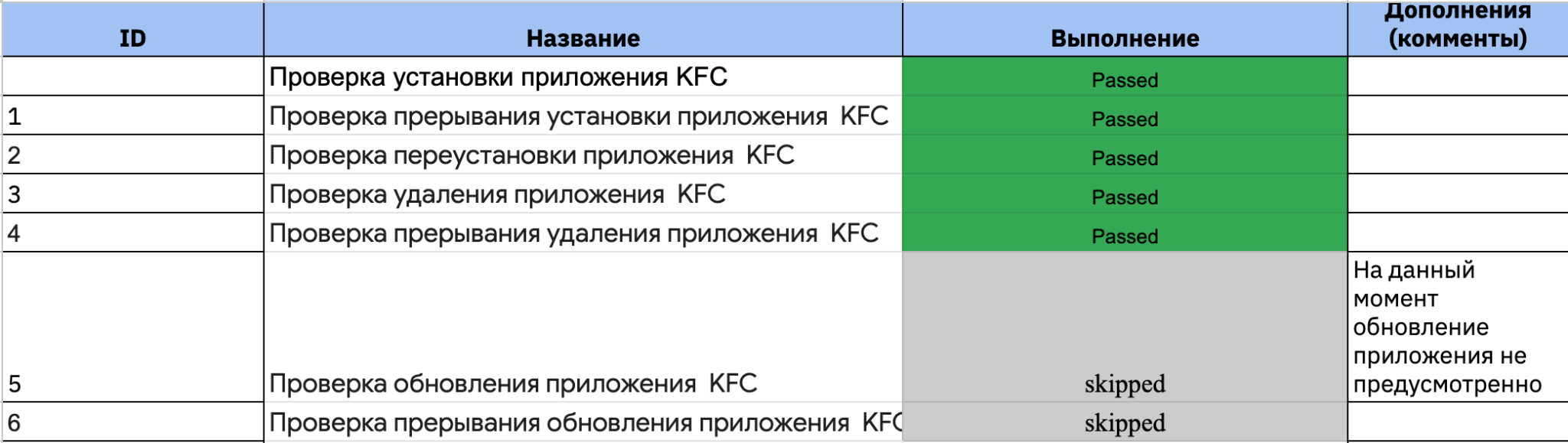
Обновление может проводиться несколькими способами в зависимости от того, как мы тестируем приложение. Например, если тестирование проводится через TestFlight или Google Play, мы просто нажимаем «Установить» в этих приложениях. Если через сервисы мобильной дистрибуции (например, Firebase), скачиваем и устанавливаем приложение «поверх» старой версии, не удаляя ее. Важно учитывать, что после обновления данные пользователя должны сохраняться. Например, после обновления приложения интернет-магазина, пользователь должен остаться залогиненым в нем. После обновления мобильной игры весь прогресс должен сохраниться.

Составим Чек-лист на проверку установки:



Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результаты проверки:



Ссылка на Чек-лист:

[Тимошенко Тестирование установки мобильного приложения КФС](https://docs.google.com/spreadsheets/d/1OgAeexOnIkybsr_F9onv3MbPEOgY0HPcuSWx2b8g_KU/edit?usp=sharing)

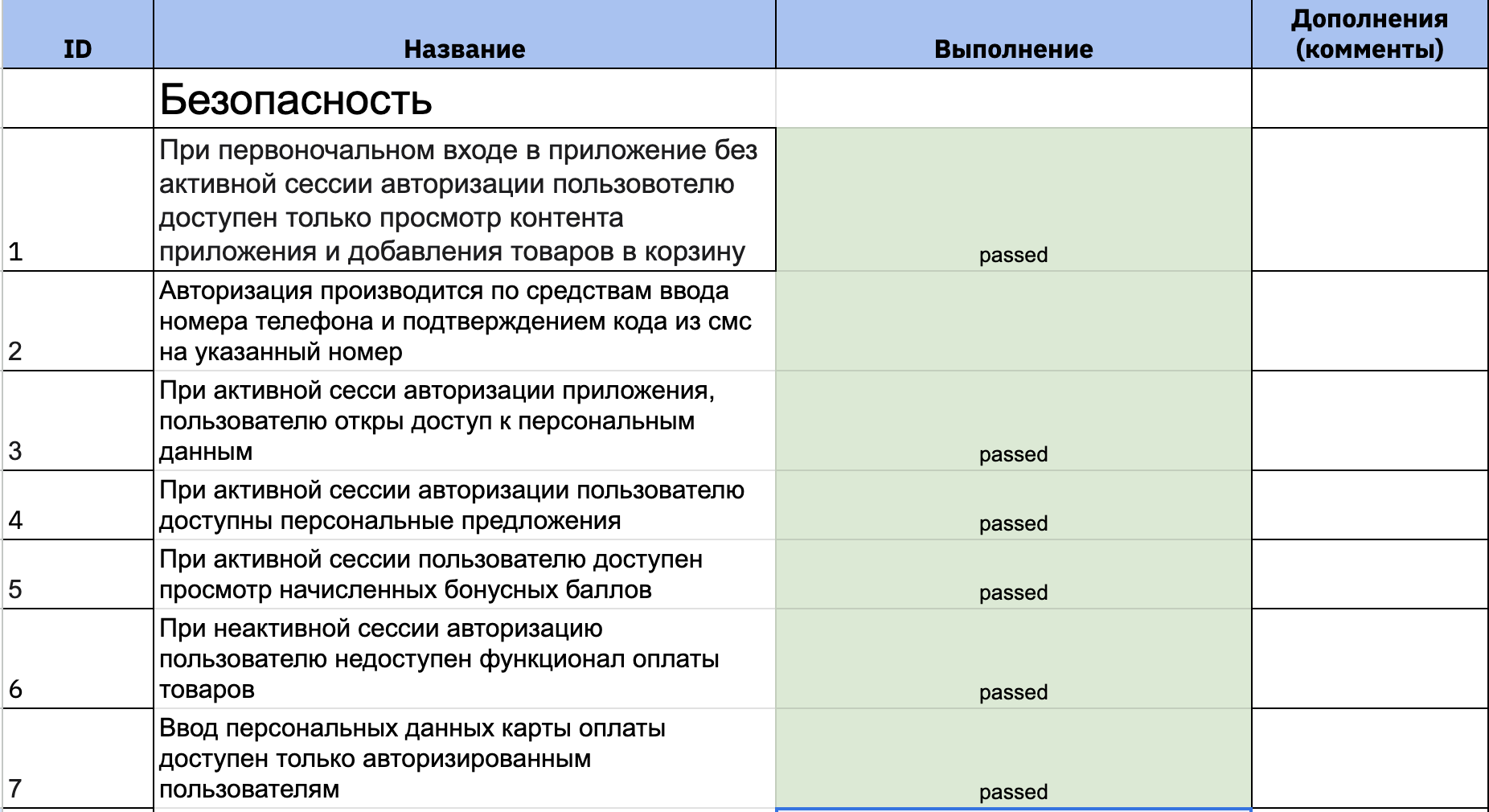
3.2.2 Тестирование безопасности

Безопасность приложений — важный аспект. Пользователи хотят быть уверенными, что их личные данные и платежная информация хранится надежно, а системы защищены от внешних атак. Большая часть пользователей прекратит или не начнет пользоваться приложением, если будут подозрения в плохом обеспечении безопасности.

Основная цель тестирования безопасности — обнаружение и оценка потенциальных уязвимостей в программном обеспечении.

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результаты проверки:



Ссылка на чек-лист:

[Тимошенко Тестирование безопасности мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1kUNj-f9tbgGA3d8hPsMy_LyJCMDettf6UU6ZOkN5-9k/edit?usp=sharing)

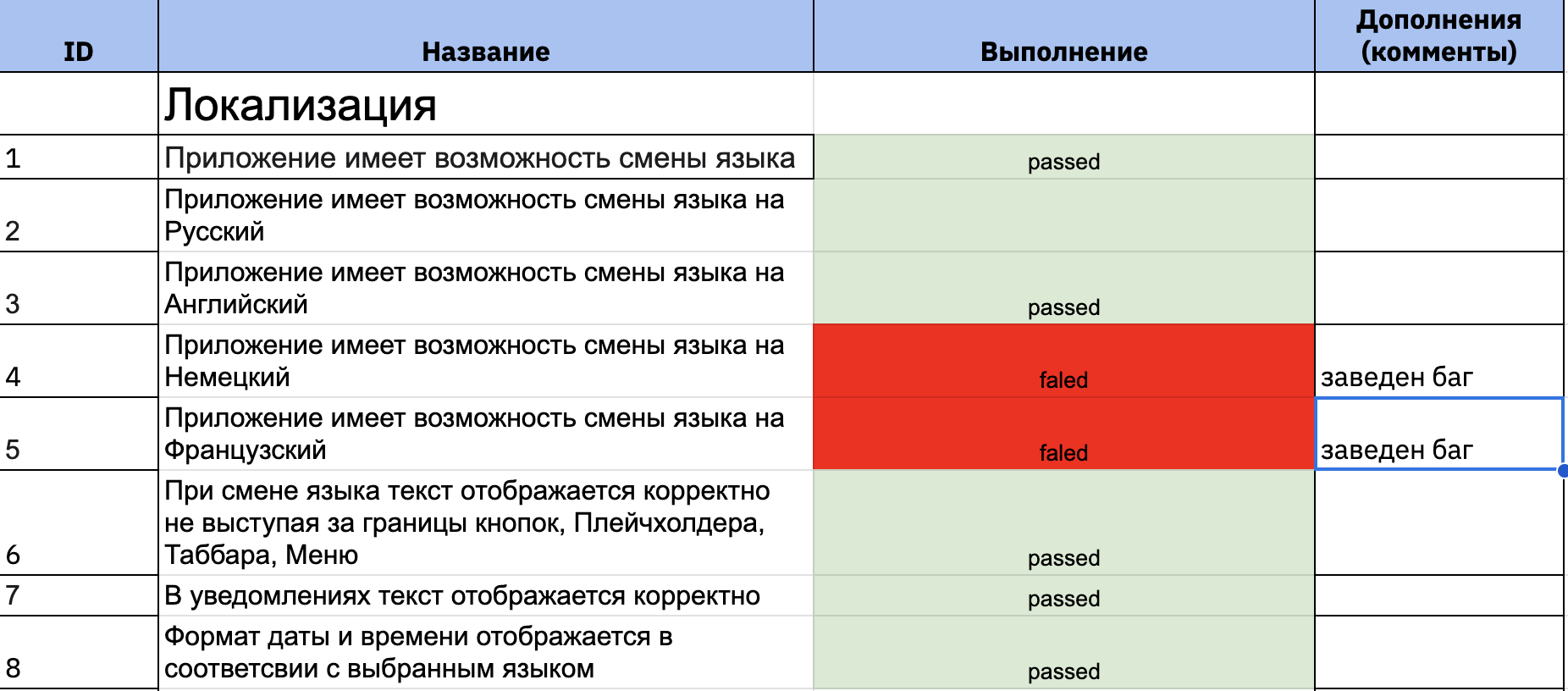
3.2.3 Тестирование локализации и глобализации.

Тестирование глобализации приложения включает в себя тестирование приложения для различных форматов дат, чисел и валюты.

Поскольку у мобильных устройств разные размеры и разрешения экрана, могут случаться проблемы с переведенными строками.

Для проверки было взято устройство: iPhone 14 Pro Max

Результаты проверки:



Заведен баг-репорт:



Ссылка на Чек-лист:

[Тимошенко Тестирование локализации мобильного приложения KFC](https://docs.google.com/spreadsheets/d/1vUQ441SQb5FjrVSgNXobws2KdYoYgEcRwxgadrHmaZg/edit?usp=sharing)

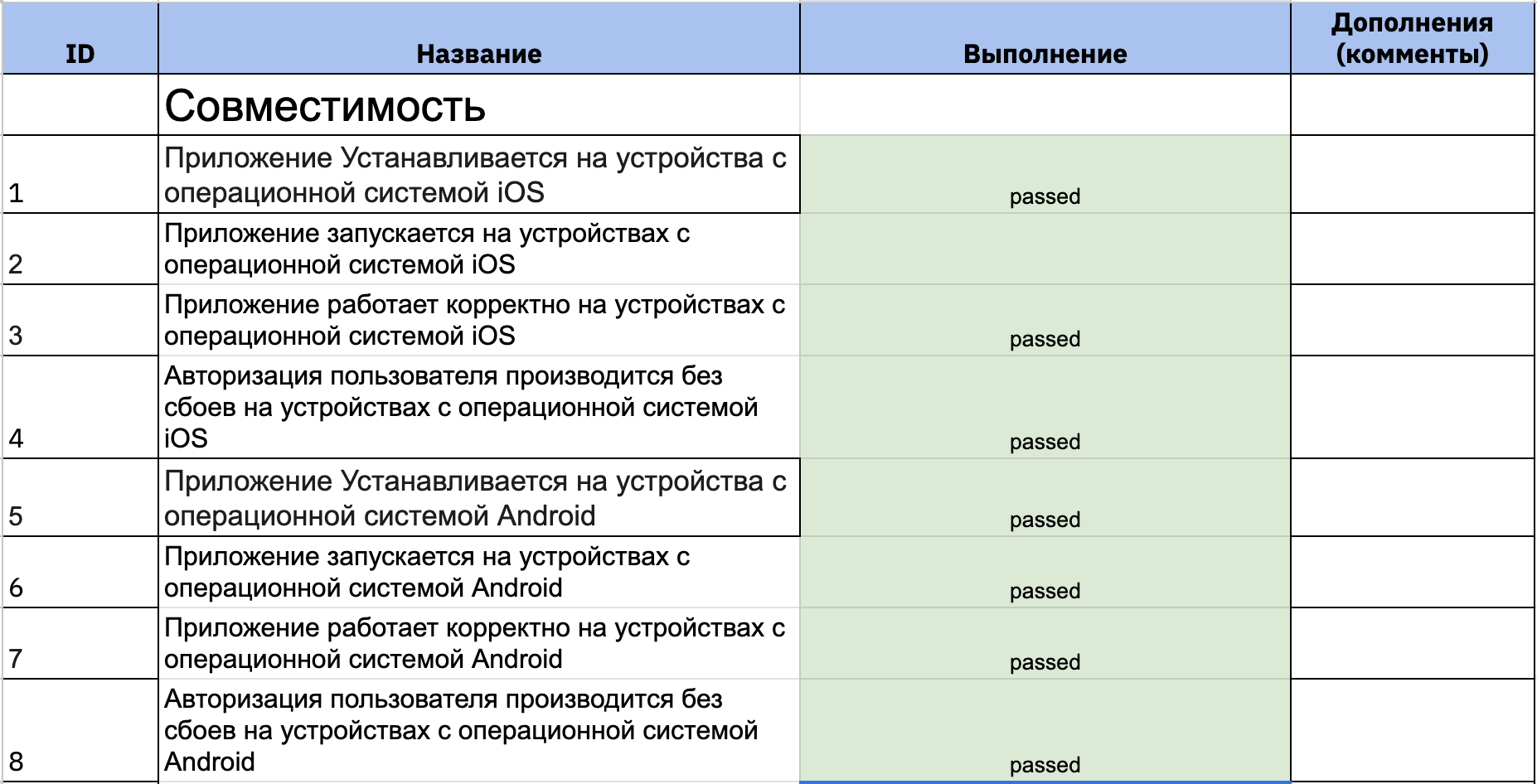
3.2.4 Тестирование совместимости.

[Тестирование на совместимость](https://tquality.ru/compatibility-testing/) (compatibility testing) — это тип тестирования, призванный обеспечить корректную работу и взаимодействие программных приложений с разными типами оборудования, операционными системами, веб и мобильными браузерами, сетями и другими программными элементами. Цель тестирования на совместимость — найти любые дефекты, которые могут образоваться при использовании ПО в различных средах и конфигурациях.

Тест предполагает, что ПО будет совместимо с десктопными и мобильными устройствами при разных настройках. Этот тип тестирования особенно подходит для решений, предназначенных для использования большим количеством пользователей, поскольку тестирование позволяет быть уверенным, что каждый клиент сможет получить доступ к ПО и использовать его без каких-либо трудностей.

Для проверки были взяты устройства: iPhone 14 Pro Max, Xiaomi Redmi Note 8 Pro, Эмулятор Pixel 3a API 34

Результаты проверки:



Ссылка на чек-лист:

[Тимошенко Тестирование совместимости мобильного приложения KFC](https://docs.google.com/spreadsheets/d/17iBAfxktW89rZmrGhfB4qkkPuaG8mB571EusWTidtVk/edit?usp=sharing)

3.3 Подведение итогов. Разработка предложений по улучшению ручного тестирования мобильного приложения **Rostic’s.**

В результате выполнения дипломной работы были закреплены знания и навыки по тестирования мобильных приложений. В ходе работы было выполнено:

Проведено ручное тестирование мобильного приложения «Rostic’s».

Составлены чек-листы для проверки приложения.

В ходе работы был протестирован функционал корректности работы приложения, корректности реализации функциональных требований. Произведена оценка качества приложения с точки зрения производительности, безопасности, надежности и других аспектов, которые не связаны с функциональностью.

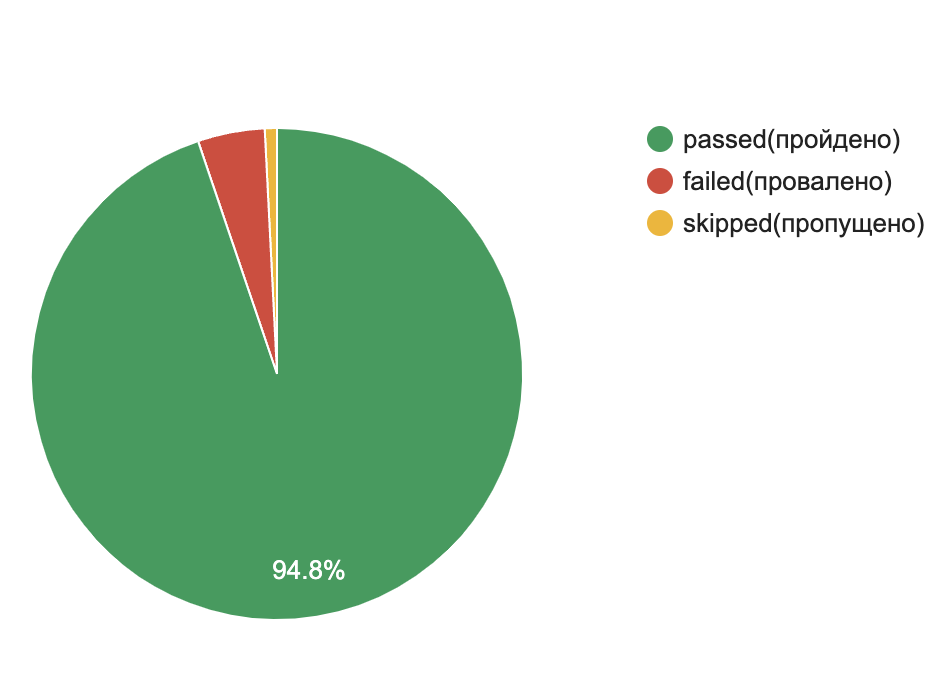
В результат проведения тестирования было выполнено:

Проведено общее количество тестов:250

из них со статусом passed(пройдено): 237

со статусом failed(провалено): 11

со статусом skipped(пропущено): 2



Составлен комплексный отчет: [Тимошенко Комплексный отчет о тестировании](https://docs.google.com/document/d/16mJTJbhADM2PHcPCvJ5FlppIsQhC_LLcaUFGFyCegFc/edit?usp=sharing)

На выявленные баги в ходе тестирования заведены баг-репорты.

# **Рекомендации:**

Критических нарушений не выявлено, что в целом не мешает достижению основных целей при использовании приложения. Замечания, предъявленные к приложению могут быть исправлены в последующих обновлениях. Приложение является достаточным и полнофункциональным. Данное приложение может быть использовано для размещения в сторах в настоящее время.

В ходе Юзабилити-тестирование было вынесены следующие рекомендации:

Доступность: Для людей с ограниченными возможностями было бы неплохо добавить определенные функции, как например для плохо видящих - увеличение шрифта и альбомную ориентацию.

# **Список используемой литературы**

* *Статья: Автор. "Заголовок статьи." Название журнала Том, номер (Год): страницы.*
* *Статья:*[*QAEVOLUTION*](https://qaevolution.ru/author/qaevolutru/)*. ”*[*QA/QC/TESTING*](https://qaevolution.ru/category/qa-qc-testing/)*,* [*ВЕБ ТЕСТИРОВАНИЕ*](https://qaevolution.ru/category/veb-testirovanie/)*,* [*ИНСТРУМЕНТЫ*](https://qaevolution.ru/category/veb-testirovanie/instrumenty/)*,* [*КАЧЕСТВО*](https://qaevolution.ru/category/kachestvo/)*,* [*ОБУЧЕНИЕ ТЕСТИРОВАНИЮ*](https://qaevolution.ru/category/obuchenie-testirovaniyu/) *РУЧНОЕ ТЕСТИРОВАНИЕ”,* [*11 НОЯБРЯ, 2023*](https://qaevolution.ru/ruchnoe-testirovanie/)*,* [*https://qaevolution.ru/ruchnoe-testirovanie/*](https://qaevolution.ru/ruchnoe-testirovanie/)
* *Статья:* [*Илья Смолин*](https://timeweb.com/ru/community/user/39560)*. "Кто такой тестировщик и чем он отличается от QA-инженера.", 03 ноября 2021,* [*https://timeweb.com/ru/community/articles/kto-takoy-testirovshchik-i-chem-on-otlichaetsya-ot-qa*](https://timeweb.com/ru/community/articles/kto-takoy-testirovshchik-i-chem-on-otlichaetsya-ot-qa)
* *Статья:Википедия. ”Мобильные приложения”, 15 апреля 2024,* [*https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5\_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5*](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)
* *Статья:Ирек Насибуллин, Екатерина Самохвалова. ”Какие бывают этапы и виды тестирования: подробный разбор”, 11 августа 2023,*[*https://ru.hexlet.io/blog/posts/vidy-testirovaniya*](https://ru.hexlet.io/blog/posts/vidy-testirovaniya)
* *Статья:Фадеева Александра. ”Процесс тестирования мобильных приложений”, https://gbcdn.mrgcdn.ru/uploads/asset/5688055/attachment/0285b6a40a0dfcea43a78cffeb090bab.pdf*
* *Статья: Андрей Соловьёв. "*[*Производство и разработка электроники*](https://habr.com/ru/hubs/electronics/)”,”[*Подготовка технической документации*](https://habr.com/ru/hubs/technical_writing/)*", 9 ноября 2023, https://habr.com/ru/articles/772732/*
* *Статья: Alex Zap Chernyak. "Исследовательское тестирование – глубокое погружение в типы, процессы, подходы, инструменты, фреймворки и многое другое",* [*https://www.zaptest.com/ru/%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%BE%D0%B5-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B3%D0%BB%D1%83*](https://www.zaptest.com/ru/%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%BE%D0%B5-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B3%D0%BB%D1%83)
* *Статья:* [*Редакция сайта GeekBrains*](https://gb.ru/)*. "Чек-лист: суть, основные форматы, правила составления", 26 мая 2023,* [*https://gb.ru/blog/chek-list/*](https://gb.ru/blog/chek-list/)
* *Лекция: Яковлев Станислав. "Тестирование по чек-листу",* [*https://gbcdn.mrgcdn.ru/uploads/asset/4266580/attachment/68984c348f5732f2a8b50379a09fa54f.pdf*](https://gbcdn.mrgcdn.ru/uploads/asset/4266580/attachment/68984c348f5732f2a8b50379a09fa54f.pdf)
* *Лекция: Яковлев Станислав. "Тест-кейсы",* [*https://gbcdn.mrgcdn.ru/uploads/asset/4266591/attachment/f06c778542a31c7a0e5b8406a9349de8.pdf*](https://gbcdn.mrgcdn.ru/uploads/asset/4266591/attachment/f06c778542a31c7a0e5b8406a9349de8.pdf)
* *Статья:* [*Екатерина Жогаль*](https://testengineer.ru/author/katya-zhogal/)*. "Что такое тест план и как его написать?", 20 января 2020,* [*https://testengineer.ru/chto-takoe-test-plan-i-kak-ego-napisat/*](https://testengineer.ru/chto-takoe-test-plan-i-kak-ego-napisat/)
* *Лекция: Яковлев Станислав. "Техники тест-дизайна", https://gbcdn.mrgcdn.ru/uploads/asset/4266593/attachment/e0ba3daf12e02fd8a826e17c206c7099.pdf*
* *Лекция: Яковлев Станислав. "Создание баг-репортов. Баг-трекинговые системы. Отчеты о тестировании",* [*https://gbcdn.mrgcdn.ru/uploads/asset/4266606/attachment/68e3072b6849798c102e89f1493c8796.pdf*](https://gbcdn.mrgcdn.ru/uploads/asset/4266606/attachment/68e3072b6849798c102e89f1493c8796.pdf)
* *Статья: Екатерина Яковлева. "UI-элементы и жесты в мобильных приложениях", 05 февраля 2021,* [*https://habr.com/ru/companies/youla/articles/540768/*](https://habr.com/ru/companies/youla/articles/540768/)
* *Лекция: Фадеева Александра. "Функциональное тестирование. Процесс тестирования мобильных приложений",* [*https://gbcdn.mrgcdn.ru/uploads/asset/5688050/attachment/0778b53f8b94aa508852d2e45b847d83.pdf*](https://gbcdn.mrgcdn.ru/uploads/asset/5688050/attachment/0778b53f8b94aa508852d2e45b847d83.pdf)
* *Лекция: Фадеева Александра. "Нефункциональное тестирование Процесс тестирования мобильных приложений",* [*https://gbcdn.mrgcdn.ru/uploads/asset/5688049/attachment/845717c7c184d7e518c97802fb605fd9.pdf*](https://gbcdn.mrgcdn.ru/uploads/asset/5688049/attachment/845717c7c184d7e518c97802fb605fd9.pdf)

# 