МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(ФГБОУ ВПО МПУ)



Пояснительная записка по дисциплине

«Проектная деятельность»

По теме: «Лаборатория тестирования. Тестирование приложения “Юла”. Модуль: Истории»

Куратор проекта: \_\_\_\_\_\_\_\_\_\_\_\_ / Яковлев Станислав Игоревич, Преподаватель /

подпись ФИО, уч. звание и степень

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Волейко А. В. , №171-371 /

подпись ФИО, группа

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Ивлев Д. С. , №171-371 /

подпись ФИО, группа

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Сидякин В. Д. , №181-352 /

подпись ФИО, группа

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Смирнова Е. Н. , №181-352 /

подпись ФИО, группа

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Халькин Д. А. , №171-371 /

подпись ФИО, группа

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Шмелёв А. Н. , №191-362 /

подпись ФИО, группа

Москва, 2021

# СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ** 3](#_Toc61743304)

[**ПЛАН РАБОТ** 6](#_Toc61743305)

[**1 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** 8](#_Toc61743306)

[1.1 Тестовая документация 8](#_Toc61743307)

[1.2 Описание дефектов 10](#_Toc61743308)

[**2 ТЕСТИРОВАНИЕ ВЕБ-СЕРВИСОВ** 13](#_Toc61743309)

[2.1 Тестирование API 13](#_Toc61743310)

[2.2 Тестирование WEB 15](#_Toc61743311)

[2.3 Нагрузочное тестирование 17](#_Toc61743312)

[2.4 Автоматизированное тестирование 19](#_Toc61743313)

[**3 ТЕСТИРОВАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ** 21](#_Toc61743314)

[3.1 Ручное тестирование 22](#_Toc61743315)

[3.2 Автоматизированное тестирование 24](#_Toc61743316)

[**4 UX ТЕСТИРОВАНИЕ** 26](#_Toc61743317)

[**5 МЕНЕДЖМЕНТ ТЕСТИРОВАНИЯ** 31](#_Toc61743318)

[5.1 Управление тестированием 31](#_Toc61743319)

[5.2 Бизнес-ценность тестирования 31](#_Toc61743320)

[5.3 Управление дефектами 32](#_Toc61743321)

[**6 ОТЧЕТ ПО ТЕСТИРОВАНИЮ ПРОЕКТА** 35](#_Toc61743322)

[6.1 Описание проекта 35](#_Toc61743323)

[6.2 Чек-лист 35](#_Toc61743324)

[6.3 Тест-кейсы 35](#_Toc61743325)

[6.4 Найденные дефекты 38](#_Toc61743326)

[**ЗАКЛЮЧЕНИЕ** 39](#_Toc61743327)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 40](#_Toc61743328)

# ВВЕДЕНИЕ

Проект по тестированию мобильного приложения «Юла» от разработчиков «Mail.ru» представляет собой поиск ошибок в работе данного приложения.

Нашей команде достался модуль историй.

**Цель** проекта: ознакомиться с базовыми понятиями и методиками, применяемыми в тестировании, изучить тестируемое приложение, написать чек-листы, тест-кейсы и рассмотреть выявленные ошибки.

Для успешного выполнения проекта, необходимо выполнить ряд следующих **задач**:

1. пройти введение в тестирование;
2. изучить клиент-серверную архитектуру и научиться тестированию API;
3. научиться тестированию WEB-приложений;
4. протестировать мобильное приложение «Юла» в модуле историй.

## Аннотация

В данной работе представлены методологии и методы исследования предметной области тестирования мобильных приложений. Также представлены тест-кейсы и чек-листы протестированного мобильного приложения раздела чаты и сообщения, с представлением найденных дефектов.

## Участники и их роли

Волейко А. В. 171-371 – тимлид, написание отчёта, тестировщик

Ивлев Д. С. 171-371 – разработка сайта, подготовка git, тестировщик

Сидякин В. Д. 181-352 – подготовка презентации, тестировщик

Смирнова Е. Н. 181-352 – подготовка постера, тестировщик

Халькин Д. А. 171-371 – дизайн сайта, тестировщик

Шмелёв А. Н. 191-362 – подготовка презентации, тестировщик

Общий план работ представлен на таблице 1 «Общий план работ».

**Индивидуальные планы участников:**

Волейко А.В.

* Изучение теоретического материала в объеме 56-часов
* Тестирование подписки из истории в объеме 9-часов
* Тестирование порядка историй в объеме 7-часов

Сидякин В.Д.

* Изучение теоретического материала в объеме 56-часов
* Тестирование удаления истории в объеме 7-часов
* Тестирование жалобы в объеме 10-часов

Халькин Д.А.

* Изучение теоретического материала в объеме 56-часов
* Тестирование перехода в профиль из истории в объеме 8-часов
* Тестирование перехода на объявление из истории в объеме 8-часов
* Создание макета сайта в объеме 5-часов

Смирнова Е.Н.

* Изучение теоретического материала в объеме 56-часов
* Тестирование функции добавления видео в историю в объеме 10-часов
* Тестирование функции перехода к следующей истории в объеме 6-часов
* Тестирование функции перехода к предыдущей истории в объеме 6-часов

Ивлев Д.С.

* Изучение теоретического материала в объеме 56-часов
* Тестирование функции добавления ссылки на профиль в истории в объеме 8-часов
* Тестирование функции добавления ссылки на объявление в истории в объеме 8-часов
* Верстка сайта в объеме 7-часов

Шмелев А.Н.

* Изучение теоретического материала в объеме 56-часов
* Тестирование модерации в объеме 7-часов
* Тестирование лимита историй в объеме 9-часов

# ПЛАН РАБОТ

Таблица 1 – Общий план работ

|  |  |  |  |
| --- | --- | --- | --- |
| **Тема занятия** | **Кол-во часов** | **Из них практических** | **Кол-во часов самостоятельной работы** |
| Введение в тестирование |  |  |  |
| Основные понятия | 18 | - | - |
| Знакомство с Jira, Redmine, Trello. | 12 | 12 | - |
| Тестирование в жизненном цикле разработки ПО. | 12 | 12 | - |
| Тестовая документация. Test Rail. | 12 | 12 | - |
| Составление чек-листов и тест-кейсов. | 12 | 12 | - |
| Правила составления отчетов об ошибках. | 12 | 12 | - |
| Техники тест-дизайна. | 12 | 12 | - |
| Клиент-серверная архитектура. Тестирование API |  |  |  |
| Краткий обзор сети | 12 | - | - |
| Клиент-серверная архитектура  Протокол HTTP | 12 | 12 | - |
| Микросервисная архитектура | 12 | 12 | - |
| Тестирование API с помощью Postman | 12 | 12 | - |
| Тестирование WEB-приложений |  |  |  |
| Основы HTML, CSS, JS для тестировщиков | 12 | 12 |  |
| Особенности тестирования WEB-приложений | - | 12 | 12 |
| DEV-Tools Google Chrome | 6 | 12 | 6 |
| Тестирование мобильных приложений |  |  |  |
| Особенности тестирования мобильных приложений. | 12 | 12 | - |
| Тестирование Android приложений | 6 | 12 | 18 |
| Среда разработки Android Studio и ADB | 6 | 12 | 6 |
| Тестирование мобильных приложений iOS | 6 | 6 | 18 |
| Сбор и анализ трафика (Charles Proxy, Fiddler) | 6 | 18 | - |
|  |  |  |  |
| Подготовка отчета |  |  |  |
| Создание отчета |  |  | 36 |
| Создание сайта | - | - | 38 |
| Подготовка постера | - | - | 10 |
| Подготовка презентации | - | - | 4 |
| Запись видеопрезентации |  | - | 1 |
| Подготовка Git | - | - | 1 |

# 1 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Тестирование – процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ, с целью определить, что они соответствуют описанным требованиям, показать, что они подходят для заявленных целей и определения дефектов.

Тестирование необходимо, чтобы понять, насколько продукт соответствует предъявляемым к нему требованиям, вписывается ли в общую концепцию видения. Тестирование рассматривается как один из аспектов обеспечения качества.

Тестировщик – специалист, который занимается тестированием. В его обязанности входит поиск возможных ошибок и сбоев в функционале тестирования объекта, например приложения. Тестировщик моделирует ситуации, вероятные при использовании тестируемого объекта, чтобы потом разработчики могли устранить обнаруженные неполадки.

Баг (Дефект) – недостаток в компоненте или системе, способный привести к ситуации сбоя или отказа.

Необходимо понимать разницу между дефектом и ошибкой. Ошибка возникает вследствие действий пользователя, которые не предусмотрены в логике работы системы. Дефект же – это недостаток в самой системе, когда при корректных действиях пользователя возникает некорректный результат.

## 1.1 Тестовая документация

План тестирования (Test Plan) — это документ или совокупность документов, расписывающих все тестовую активность пределах одного проекта, все работы проводимые командой тестирования или одним тестировщиком. Поверхностно можно выделить моменты описываемые в тест-плане: объект тестирования, расписания работ, критерии начала и окончания тестирования, стратегию, риски и список проводимых работ. [2]

Хороший тест план должен описывать следующее:

1. Что надо тестировать? Описание объекта тестирования: системы, приложения, оборудования.

2. Что будете тестировать? Список функций и описание тестируемой системы и её компонент в отдельности.

3. Как будете тестировать? Стратегия тестирования, а именно: виды тестирования и их применение по отношению к объекту тестирования.

4. Когда будете тестировать? Последовательность проведения работ: подготовка (Test Preparation), тестирование (Testing), анализ результатов (Test Result Analisys) в разрезе запланированных фаз разработки

Критерии начала тестирования:

- готовность тестовой платформы (тестового стенда);

- законченность разработки требуемого функционала;

- наличие всей необходимой документации.

Критерии окончания тестирования - результаты тестирования удовлетворяют критериям качества продукта:

- требования к количеству открытых багов выполнены;

- выдержка определенного периода без изменения исходного кода приложения Code Freeze (CF);

- выдержка определенного периода без открытия новых багов Zero Bug Bounce (ZBB).

Тест дизайн (Test Design) — процесс проектирования и создания тестовых случаев, для проведения в дальнейшем проверки ПО с учетом спецификации проекта и требований.

Тестовый случай (Test-case) — строго сформированный документ (на каждой организации могут быть свои правила, в той или иной мере похожие) с подробным описанием шагов и действий, для проверки одной части функционала, со строго заданными критериями прохождения.

Юз-кейс (Use-case) — документ для тестирования продукта или ПО менее официален и используется для предугадывания и построения на основе предполагаемых сценариев использования путей тестирования приложения. Часто разрабатывается на основе бизнес-задач и проверки критического пути тестирования.

Баг-репорт (Bug report) — документ, случай — в котором описывается ситуация, последовательность шагов, приведшая к возникновению ошибки на тестируемом ПО. Чаще всего требуется описание ожидаемого результата для такой конкретной последовательности шагов который будет являться правильным.

Чек-лист (Check-list) — документ который содержит в себе краткое описание функциональности всего приложения, для проверки его (ПО). Создание и написание чек-листа обычно заменяет написание тестовых случаев, в большинстве случаев ускоряет подготовку к проведению тестирования.

Спецификация требований (Software Requirements Specification) — это полное, четкое описание разрабатываемой программы. Ключевыми особенностями является полное описание всей функциональности программы, четкость требований, отсутствие неточностей в их описании, простота понятий, и достаточно четкая детализации. (Тестирование требований).

## 1.2 Описание дефектов

**Что такое ошибка и что такое дефект?**

Баг/ошибка является следствием ошибки кодирования, в то время как дефект — это отклонение от первоначальных бизнес-требований.

Эти два термина имеют очень тонкую черту различия. Оба эти сущности являются недостатками, которые должны быть исправлены и, поэтому, иногда, эти термины взаимозаменяемо используются в разных компаниях и командах.

Когда тестер выполняет тестовые случаи, он может столкнуться с результатом теста, который противоречит ожидаемому результату. Это изменение в результатах теста называется дефектом программного обеспечения. Эти дефекты или вариации обозначаются разными именами в разных организациях, таких как проблемы, проблемы, ошибки или инциденты.

**Отчет об ошибке**

Сообщая об ошибке разработчику, ваш отчет об ошибке должен содержать следующую информацию

• Defect\_ID - уникальный идентификационный номер для дефекта.

• Описание дефекта - подробное описание дефекта, включая информацию о модуле, в котором обнаружен дефект.

• Версия - версия приложения, в котором был обнаружен дефект.

• Шаги - подробные шаги вместе со скриншотами, с помощью которых разработчик может воспроизвести дефекты.

• Дата повышения - дата возникновения дефекта

• Ссылка - где в вас предоставить ссылку на документы, как. требования, дизайн, архитектура или даже скриншоты ошибки, чтобы помочь понять дефект

• Обнаружено - имя / идентификатор тестера, который выявил дефект

• Состояние - состояние дефекта, подробнее об этом позже

• Исправлено - Имя / ID разработчика, который это исправил

• Дата закрытия - дата закрытия дефекта

• Серьезность, которая описывает влияние дефекта на приложение

• Приоритет, связанный с срочностью устранения дефектов. Приоритет серьезности может быть Высокий / Средний / Низкий в зависимости от срочности воздействия, при которой дефект должен быть исправлен соответственно.

Управление дефектами — это систематический процесс выявления и устранения ошибок. Цикл управления дефектами содержит следующие этапы:

1. Обнаружение дефекта

2. Категоризация дефекта

3. Устранение дефекта разработчиками

4. Проверка тестерами

5. Закрытие дефекта

6. Отчеты о дефектах в конце проекта

**Категории**

Классификация дефектов помогает разработчикам программного обеспечения определять приоритеты своих задач и в первую очередь устранить те дефекты, которые больше прочих угрожают работоспособности продукта.

Критический - дефект должен быть устранены немедленно, иначе это может привести к большим потерям для продукта

Например: функция входа на сайт не работает должным образом.

Вход в систему является одной из основных функций банковского сайта, если эта функция не работает, это серьезные ошибки.

Высокий - дефект негативно влияет на основные функции продукта

Например: производительность сайта слишком низкая.

Средний - дефект вносит минимальные отклонения от требований к к продукту

Например: не корректно отображается интерфейс на мобильных устройствах.

Низкий - минимальное не функциональное влияние на продукт

Например: некоторые ссылки не работают.

**Решение**

После того, как дефекты приняты и классифицированы, вы можете выполнить следующие шаги, чтобы исправить их.

• Назначение: проблемы отправлены разработчику или другому техническому специалисту для исправления и изменило статус на отвечающий.

• График устранения: сторона разработчика берет на себя ответственность на этом этапе, они создадут график для устранения этих дефектов в зависимости от их приоритета.

• Исправление: пока группа разработчиков устраняет дефекты, диспетчер тестов отслеживает процесс устранения проблем, исходя из графика.

• Сообщить о решении: получите отчет об устранении бага от разработчиков, когда дефекты устранены.

# 2 ТЕСТИРОВАНИЕ ВЕБ-СЕРВИСОВ

## 2.1 Тестирование API

Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с API удалённого сервера. API — это составляющая часть сервера, которая получает запросы и отправляет ответы.

HTTP – протокол для обмена текстовыми сообщениями, описывает взаимодействие между двумя компьютерами (клиентом и сервером), построенное на базе сообщений, называемых запрос (Request) и ответ (Response). Каждое сообщение состоит из трех частей: стартовая строка, заголовки и тело. При этом обязательной является только стартовая строка.

HTTPS – версия HTTP, использующая шифрование данных.

Толстый клиент для выполнения операций использует вычислительные мощности машины, на которой он установлен. Пример: почтовые программы, например Microsoft Outlook, Mozilla Thunderbird.

Тонкий клиент использует мощности удаленного компьютера, предоставляя пользователю только интерфейс для взаимодействия с ним. Пример: электронная почта в браузере.

Так как RESTful-сервис основан на HTTP, он использует те же стандартные методы протокола:

* GET. Используется для получения содержимого указанного ресурса. В тело ответа может содержаться дополнительная информация;
* HEAD. Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело;
* POST. Применяется для передачи пользовательских данных заданному ресурсу;
* PUT. Обновляет содержимое указанного ресурса;
* PATCH. Аналогично PUT, но обновляет данные выборочно, а не целиком;
* DELETE. Удаление содержимого указанного ресурса;
* TRACE. Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные серверы добавляют или изменяют в запросе.;
* OPTIONS. Преобразует соединение запроса в прозрачный TCP/IP-туннель, обычно чтобы содействовать установлению защищенного SSL-соединения через нешифрованный прокси;
* CONNECT. Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса.

Когда сервер возвращает HTTP ответ, в нем обязательно содержится код состояний. Примеры некоторых из них.

* 200 ОК – ответ успешно получен;
* 201 Created – элемент создан;
* 204 No Content – нет содержимого. Может быть прислан сервером, если в результате POST или DELETE-запроса по той или иной причине не был обработан ресурс, но URI корректен, запрос корректен;
* 400 Bad Request – задан неверный запрос;
* 401 Unauthorized – пользователь не авторизован;
* 403 Forbidden – доступ запрещен Если данному пользователю запрещен доступ к запрашиваемому ресурсу, то вернётся код 403. Он означает, что аутентификация прошла успешно, но у пользователя нет соответствующих прав;
* 404 Not Found – не найдено, ресурс по данному URI отсутствует;
* 404. Во-первых, этот код возвращается, когда задан действительно неверный адрес и по нему в принципе нет (и не должно быть) ресурса;
* 405 Method Not Allowed – метод не разрешен Данный код ответа может прийти в том случае, если на URI ресурса отправляется запрос с методом, который там применяться не может;
* 409 Conflict – конфликт. Этот код ответа может отправляться в ответ на POST/PUT/PATCH-запросы, если невозможно создать или обновить указанную запись. Например, если какое-то поле должно быть уникальным.

Simple Object Access Protocol – основанный на XML протокол, предназначенный для обмена информацией в распределенных системах. SOAP устанавливает стандарт взаимодействия клиент-сервер и регламентирует,как должен осуществляться вызов, как должны передаваться параметры и возвращаемые значения.

REST (сокр. от англ. Representational State Transfer – «передача состояния представления») – архитектурный стиль взаимодействия компонентов распределенного приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределенной системы. REST не является стандартом сам по себе, это архитектурный стиль. [1]

RESTful-сервисы работают с объектами на сервере, которые называются «ресурсы». Эти сервисы позволяют внешним системам получать доступ к текстовым представлениям ресурсов и управлять ими с помощью определенного набора операций. Зачастую это стандартные действия: «создать», «прочитать», «обновить», «удалить».

Для тестирования API необходимы инструменты, где можно отправить входные данные в запросе и проверить точность выходных данных. Одним из таких инструментов является Postman. Его возможности:

* Написание и отправка запросов;
* Сохранение запросов;
* Добавление контрольных точек к вызову API;
* Создание разных окружений для одних и тех же запросов;
* Запуск коллекций и использование их как автоматизированных тестов.

## 2.2 Тестирование WEB

Веб-приложение – это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер (в широком смысле). Основная часть приложения, как правило, находится на стороне веб-сервера, который обрабатывает полученные запросы в соответствии с бизнес-логикой продукта и формирует ответ, отправляемый пользователю. На этом этапе в работу включается браузер, именно он преобразовывает полученный ответ от сервера в графический интерфейс, понятный рядовому пользователю.

Основные виды тестирования сайта (веб-приложения)

1. Тестирование функциональности;

2. Тестирование удобства использования;

3. Тестирование интерфейса;

4. Тестирование совместимости;

5. Тестирование производительности и скорости загрузки сайта;

6. Тестирование безопасности.

Остановимся поподробнее на первом пункте

«Клиент-сервер» —это вид архитектуры распределенного приложения. В этой архитектуре есть два типа элементов: клиенты и сервер. Они взаимодействуют определенным способом, который называется протоколом. К одному серверу могут подключаться и одновременно запрашивать нужную информацию миллионы клиентов.

Обычно клиенты и сервер расположены на разных вычислительных машинах и взаимодействуют через сеть посредством сетевых протоколов, но они могут быть и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или сервисов, а клиенты инициализируют сеансы с серверами.

Для взаимодействия с веб приложением необходим проколов http.

Протокол HTTP (сокр. от англ. HyperText Transfer Protocol) —протокол прикладного уровня, используемый в веб-приложениях для передачи данных между клиентом и сервером. По умолчанию используется порт 80. HTTP чаще всего использует возможности протокола TCP для пересылки своих сообщений.

HTTPS (сокр. от англ. HyperText Transfer Protocol Secure) —расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов SSL или TLS. HTTPS по умолчанию использует порт 443.HTTP —это так называемый stateless-протокол. Он не имеет состояния, то есть в этом протоколе не существует связи между двумя запросами, которые последовательно выполняются по одному соединению.

Для обхода этого ограничения могут применяться Cookie, которые позволяют использовать сессии с сохранением состояния. Cookie добавляются к рабочему потоку благодаря расширяемости заголовков и позволяют сессии на каждом HTTP-запросе делиться состоянием.

За рендеринг HTML-документов и работу с JavaScript в браузере отвечают специальные компоненты, так называемые движки браузера (browser engine).

На сегодняшний день существует несколько актуальных движков:

* Microsoft Trident – используется в Internet Explorer 11 и 12. Планировалось, что в дальнейшем будет заменен на Blink;
* EdgeHTML – являлся основой для Microsoft Edge, браузера в Windows 10. Сейчас заменён на Blink;
* Gecko – на данный момент используется в Firefox и других продуктах, входящих в Mozilla, например, в почтовом клиенте Thunderbird;
* Blink – наиболее популярный движок, основе браузера с открытым кодом Chromium, на базе которого был разработан браузер Chrome и многочисленные сторонние браузеры: Opera, Яндекс.Браузер, Atom и другие;
* WebKit – движок для браузера Safari, это стандартный браузер в macOS.

## 2.3 Нагрузочное тестирование

Нагрузочное тестирование - важный элемент комплексного тестирования производительности, цель которого - проанализировать скорость отклика системы на внешний запрос. Этот тест позволяет определить, соответствует ли тестируемое приложение или устройство заявленным требованиям. Кроме того, можно понять, как программа отреагирует, если ее используют несколько пользователей одновременно. Основная идея нагрузочного тестирования - создать определенную нагрузку с помощью программного и аппаратного обеспечения, а затем отслеживать индекс производительности продукта.

Основные этапы нагрузочного тестирования:

1. Подготовка — Проводится анализ целей и статистики эксплуатации системы. Определяются бизнес-операции, имеющие значение с точки зрения нагрузки на систему. Создается и согласуется документ «Методика нагрузочного тестирования», который включает: стратегию тестирования, список и описание тестов, критерии успешного завершения, описание средств мониторинга и инструментов нагрузочного тестирования. Осуществляется подготовка тестовых данных, настраивается мониторинг, наполняется база данных.

2. Проведение — Выполняется запись и отладка скриптов нагрузочного тестирования, и реализация сценариев нагрузочного тестирования. С помощью разработанной системы нагрузочного тестирования выполняется тестирование приложений. В рамках нагрузочных испытаний собирается различная статистическая информация, с помощью которой выполняется анализ производительности.

3. Отчет — После проведения нагрузочного тестирования компании клиенту предоставляется отчет, который описывает результаты тестирования, отступления от методики (если имеются), список ошибок, предложения по оптимизации работы системы, общие замечания.

Основными целями нагрузочного тестирования являются:

1. Оценка производительности и работоспособности приложения на этапе разработки и передачи в эксплуатацию

2. Оценка производительности и работоспособности приложения на этапе выпуска новых релизов, патч-сетов

3. Оптимизация производительности приложения, включая настройки серверов и оптимизацию кода

4. Подбор соответствующей для данного приложения аппаратной (программной платформы) и конфигурации сервера

## 2.4 Автоматизированное тестирование

Автоматизированное тестирование ПО — процесс тестирования программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, производятся автоматически с помощью инструментов для автоматизированного тестирования.

Основной целью автоматизации является оптимизация временных и человеческих ресурсов, затрачиваемых на проведение тестирования. Исключен «человеческий фактор». Сильное достоинство. Все мы люди и никто из нас не застрахован от ошибок. Выполняемый же тест-скрипт не пропустит тест по неосторожности и ничего не напутает в результатах.

* Быстрое выполнение – автоматизированному скрипту не нужно сверяться с инструкциями и документациями.
* Меньшие затраты на поддержку – когда скрипты уже написаны, на их поддержку и анализ результатов требуется, как правило, меньшее время чем на проведение того же объема тестирования вручную.
* Отчеты – автоматически рассылаемые и сохраняемые отчеты о результатах тестирования.
* Выполнение без вмешательства – во время выполнения тестов инженер-тестировщик может заниматься другими полезными делами, или тесты могут выполняться в нерабочее время.

В процессе автоматизации можно выделить три основных этапа:

* начальный этап - этап подготовки и планирования. На этом этапе принимается решение о необходимости автоматизированного тестирования, оцениваются потенциальные возможности и экономический эффект, устанавливаются цели и стратегии автоматизации, а также определяются типы тестов, подходящие для автоматизации. На этом же этапе после тщательного изучения свойств, оценки степени соответствия конкретной задаче и оценки ресурсов, необходимых для поддержания нормальной работы, выбирается инструмент автоматизации. В результате первоначальной разработки устанавливаются требования к описанию тестов, проверяется совместимость средств автоматизации и тестируемого программного обеспечения и тестовой среды, а также разрабатываются точные методы оценки затрат на внедрение. этап активной разработки;
* этап поддержки автоматических тестов. Большая часть времени на этом этапе посвящена описанию, разработке, тестированию и выполнению автоматизированных тестов. Уменьшает количество ресурсов, необходимых для разработки общих функций. В случае крупных проектов команда разработчиков тестов может быть значительно увеличена, а успешно завершенная фаза подготовки и планирования гарантирует минимальные риски. При разработке тестов важно обеспечить возможность автоматического документирования обнаруженных ошибок и составления общего отчета по результатам автоматических тестов. После успешного выполнения автоматических тестов результаты анализируются и при необходимости корректируются. Активная фаза разработки может быть довольно продолжительной, в зависимости от размера проекта.
* этап поддержки автоматических тестов. Для автоматизации тестов выбираются только те тесты, которые тестируют фиксированную часть программы. Однако изменение требований к входу, обновление настроек или структуры среды тестирования может привести к тому, что автоматические тесты будут давать ошибочные результаты. Следовательно, изменения в системе всегда следует отслеживать, и при необходимости автоматизированные тесты следует адаптировать или модифицировать, чтобы поддерживать их в актуальном состоянии.

# 3 ТЕСТИРОВАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Очевидно, что мобильное приложение сильно отличается от веб-версии. Поэтому необходимо учитывать это при планировании процесса тестирования.

Рассмотрим основные различия между мобильными и веб-версии:

* Мобильное устройство – это система, которая не обладает мощными характеристиками. Таким образом, он не может работать как персональный компьютер.
* Тестирование мобильных приложений проводится на мобильных телефонах (Apple, Samsung, Nokia), в то время как веб-версия тестируется на центральном процессоре.
* У мобильных устройств бывают разные разрешения. Размер экрана мобильного телефона меньше, чем у веб-версий.
* Выполнение и прием вызовов является основной задачей телефона, поэтому приложение не должно вмешиваться в эту важную функцию.
* Широкий спектр конкретных операционных систем и компонентных конфигураций: Android, iOS, BlackBerry.
* Операционная система мобильного телефона быстро устаревает.
* Мобильные устройства используют сетевые подключения (3G, 4G, Wi-Fi), широкополосное подключение к настольному ПК или Wi-Fi.
* Мобильные устройства постоянно осуществляют поиск сети. Вот почему необходимо протестировать приложение с разной скоростью передачи данных.
* Инструменты, которые хорошо подходят для тестирования веб приложений, не полностью подходят для тестирования мобильных приложений.
* Мобильные приложения должны поддерживать несколько входных каналов (клавиатура, голос, жесты и т. д.), мультимедийные технологии и другие функции, повышающие их удобство использования.

Тестирование мобильных приложений можно разделить на ручное и автоматизированное. Необходимо понимать, что в краткосрочной перспективе выгоднее использовать ручное, но в долгосрочной - автоматизированное.

Конечно, нет однозначных ответов на то, какую стратегию лучше всего выбрать. Однако сочетание различных вариантов наиболее оптимально. Например, вы можете использовать симуляторы на самых ранних этапах вашего тестирования. Но лучше использовать реальные устройства (физические или облачные) на заключительных этапах. Автоматизированное тестирование предпочтительнее для нагрузочного и регрессионного тестирований.

## 3.1 Ручное тестирование

Самый большой из недостатков ручного тестирования — человеческий фактор. Он, конечно, влияет на всё, происходящее в команде — и на работу участников, и на события, происходящие на стороне клиента. В случае QA причиной того, что тестировщик будет слабо погружен в процесс и пропустит потенциальную ошибку может стать всё что угодно — недостаток опыта, семейные проблемы или головная боль.

Один и тот же сценарий два тестировщика могут проверить разными способами. Что с этим делать? Важно, чтобы каждый непредусмотренный или отличающийся от ожидаемого результат был зафиксирован в виде кейса. Чтобы любой тестировщик мог проверить его, совершив тот же набор действий. Но и этого может быть мало — если кейс окажется недостаточно подробным, а тестировщик — не разберётся в описании. Гарантировать стопроцентное исключение человеческого фактора, конечно, невозможно — но можно постараться максимально снизить вероятность проблем, которые он вызывает.

Это тоже негативно сказывается на сроках поставки фичи в продакшн и трудозатратах: ведь теперь к периодически проводимым базовым кейсам и регрессии добавляются и “хитрые” кейсы, придуманные тестировщиками в процессе.

Усугубляет ситуацию вероятность того, что часть встреченных багов ещё не будет иметь строгого описания, потому что причины их возникновения пока не понятны. Как бороться с такими повторными тестированиями? Либо найти уже источник ошибки и устранить его, либо — всё-таки автоматизировать прохождение проблемных кейсов — в этом случае переход к программным тестам будет вполне оправдан как с точки зрения времени, так и финансово. (Нет, это не противоречит вышесказанному — потому что такие ситуации возникают уже в ходе активной разработки и к этому времени вы уже в любом случае развернёте процессы автотестирования).

В любом случае — появление первых кейсов, нуждающихся в регрессивных тестах или релиз второй версии приложения и соответствующее этим событиям масштабирование команды — тот момент, когда автоматизация станет актуальна (но не исключит ручное тестирование новых возможностей). На этом этапе автоматизация уже начнёт экономить время: разработчик сможет сам, ещё до передачи QA-отделу запустить регресс-тесты новой фичи, чтобы убедиться, что она не ломает существующий функционал, а тестировщику не придётся лишний раз проходить по набившим оскомину базовым кейсам. Ещё одно преимущество внедрения автотестов на этом этапе — их можно запускать по определённому расписанию — каждую ночь, в дни окончания спринтов или при публикации новых сборок приложения.

При этом нельзя забывать несколько вещей:

* создание кейсов и написание автотестов будет требовать времени — закладывайте его в спринты;
* убедитесь, что кейс автотеста прописан хорошо и подробно и описывает возможную проблему или существующий сценарий во всей полноте;
* проверьте, правильно ли работает автотест и действительно ли он проверяет то, что нужно и делает это качественно.

Достоинства ручного тестирования мобильных приложений:

1. Это более экономически выгодно в краткосрочной перспективе.
2. Ручное тестирование более гибкое.
3. Лучшее моделирование действий пользователя.
4. Недостатки ручного тестирования мобильных приложений:
5. Ручные тестовые примеры трудно использовать повторно.
6. Менее эффективно выполнение определенной постоянной задачи.
7. Процесс тестирования медленный.
8. Некоторые типы тестовых случаев не могут быть выполнены вручную (нагрузочное тестирование).

## 3.2 Автоматизированное тестирование

Автоматизация тестирования помогает решить сразу несколько проблем — в том числе если речь идёт о мобильных приложениях. Вместо того чтобы вручную проводить рутинные трудоемкие процедуры, специалисты могут делегировать значительную их часть фреймворкам. Автоматизация упрощает проверку и помогает ускорить регрессионное тестирование, а также даёт возможность использовать ранее недоступные типы тестирования.

В идеале вы должны настроить как можно больше кейсов, что позволит вам автоматизировать около 80% ваших процессов тестирования. Существуют конкретные тестовые случаи, которые должны быть автоматизированы, список перед вами:

* автоматизируйте самые частотные тест-кейсы;
* автоматизируйте тестовые случаи, которые легко автоматизировать;
* автоматизируйте тест-кейсы, которые имеют предсказуемые результаты;
* автоматизируйте самые утомительные ручные тесты;
* автоматизируйте тест-кейсы, которые невозможно выполнить вручную;
* автоматизируйте тест-кейсы, которые выполняются на нескольких различных аппаратных или программных платформах и конфигурации;
* автоматизируйте часто используемые функции.

Преимущества автоматизированного тестирования приложений:

* Процесс тестирования занимает мало времени.
* Экономичность в долгосрочной перспективе использования.
* Автоматизированные тестовые случаи легко использовать повторно.
* Единственное решение для некоторых видов тестирования (тестирование производительности).
* Результаты испытаний легко доступны.
* Недостатки автоматизированного тестирования приложений:
* У некоторых мобильных средств тестирования есть ограничения.
* Процесс тестирования занимает много времени.

Автоматизированное тестирование наименее эффективно в определении удобства пользования, что подводит нас к следующем разделу.

# 4 UX ТЕСТИРОВАНИЕ

Проверка эргономичности (юзабилити-тестирование, англ. *Usability testing*) — исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (такой как веб-станица, пользовательский интерфейс или устройство) для его предполагаемого применения. Таким образом, проверка эргономичности измеряет эргономичность объекта или системы. Проверка эргономичности сосредоточена на определённом объекте или небольшом наборе объектов, в то время как исследования взаимодействия человек-компьютер в целом — формулируют универсальные принципы. [4]

Цели проведения тестирования у каждого разные: кто-то тестирует прототип или концепцию, кто-то проверяет гипотезы, кто-то исследует пользовательские сценарии, поэтому методы и метрики различаются. Но правила проведения, этапы подготовки и комплект сопроводительной документации схожие:

1. Цели и задачи. Постановка главной цели тестирования, которая определит дальнейшее направление: задачи, задания, методы и выбор респондентов. Исходя из цели формулировка проблемы или задачи. Это может быть проверка разработанного продукта или поиск недочетов после редизайна. Например, компания поменяла дизайн формы заказа, после этого снизилась конверсия. С помощью тестирования исследователи будут разбираться, почему так произошло и что делать.

2. Гипотезы. Составление гипотезы, которую исследование подтвердит или опровергнет. Допустим, пользователи при бронировании отеля заказывают трансфер из аэропорта отдельным сообщением, не используя специальную форму заказа. В таком случае вариант гипотезы может быть такой: «пользователи не понимают, что это форма для заказа трансфера, или считают ее громоздкой для заполнения».

3. Сценарии. Отдельное тестирование сценария поведения пользователей — то, как люди взаимодействуют с сайтом. Для каждой страницы работает собственный сценарий. Для его составления ответьте на четыре вопроса:  
- Откуда пришел пользователь?  
- Что он должен увидеть на этой странице?  
- С какой целью он пришел на страницу?  
- Чем должно закончиться посещение?  
Сценарий пользователя не обязательно должен быть длинным и сложным. Иногда чем короче взаимодействие, тем лучше конверсия. Например, для бизнеса по доставке картриджей, зарядок, линз важна скорость, поэтому желательно, чтобы пользователь сразу понимал, что служба доставка ему подходит.

* Тест юзабилити поможет узнать причины, по которым:  
  у интернет-магазина низкая конверсия;
* пользователи звонят в колл-центр с вопросами, которые уже есть на сайте;
* мобильное приложение получает негативные отзывы в App Store и Google Play;
* сотрудники, работающие с СЭД, ненавидят ее и жалуются, что работать с ней слишком сложно;
* и т.п.

Также тестирование понадобится, если нужно:  
сравнить два интерфейса, например, старый и новый, ваш и конкурента, чтобы выяснить, какой из них лучше;

* сравнить удобство интерфейса для двух групп пользователей, например, для новичков и опытных;
* выявить потенциальные юзабилити-проблемы нового продукта еще до того, как он будет выпущен на рынок;
* оценить соответствие продукта определенным KPI (например, «процесс оформления заказа занимает не более 5 минут»).

**Что можно измерить?**

**Количественные исследования** всегда конкретны и сфокусированы, нацелены на получение числовых показателей. Это может быть время выполнения действий на сайте или доля респондентов, выполнивших задание. Результаты «да/нет» тоже можно представить в виде цифр. Например, привести их в бинарную систему: да — 1 балл, нет — 0 баллов.

* Часто в тестировании применяют метод Якоба Нильсена, который переводит результаты в процентное соотношение и считает процент успешности. Рекомендуем упростить шкалу оценок и оперировать тремя вариантами:  
  выполнил самостоятельно — 100%;
* выполним с помощью модератора — 50%;
* не выполнил — 0%.

Чтобы определить, насколько часто пользователи сталкиваются с проблемами, вычисляйте их частотность. Для этого посчитайте количество респондентов, которые не смогли выполнить задание из-за одной и той же проблемы. Давайте участникам тестирования одинаковые задания, тогда показатель частотности будет достоверным.

**Качественные исследования** выбирают, чтобы получить много разных замечаний, понять мышление пользователей, найти скрытые проблемы. Тестирование основано на открытых и гибких вопросах. Для этого проводится интервью, которое показывает степень удовлетворенности респондентов. Существует множество методик и опросников для проведения качественных исследований.

Например, модель Кано, которая была разработана японским ученым. С ее помощью узнавайте не только удовлетворенность текущей версией сайта, но и ожидания пользователей. Все ответы респондентов переводятся в баллы и ранжируются по шкале ожиданий от «мне нравится» и «я ожидаю это» до «мне не нравится, и я не могу это принять». В результате исследователи строят график, который показывает, что именно аудитория считает:

* само собой разумеющимся;
* конкурентным преимуществом сайта;
* функциями, которые вызывают у них восторг;
* неважным.

По итогу проведения качественных исследований необходимо правильно интерпретировать полученные результаты. Возможно, респонденты дадут много интересных предложений, но оцените их с точки зрения технической реализации и затрат на их разработку. В любом случае постарайтесь понять, какую именно потребность закрывает их предложение. Это нужно, чтобы найти способ улучшить юзабилити сайта, подходящий для вашей компании.

**Методы тестирования**

**Наблюдение** — самый простой метод: респондент работает в обычном режиме, модератор смотрит и анализирует его действия. В завершении респондент заполняет опросник и делится впечатлениями о сайте. Метод хорош тем, что пользователь взаимодействует с сайтом естественно, на него не давят окружающие обстоятельства. [5]

Но есть и недостаток: респондент заполняет опросник после завершения тестирования, поэтому может не вспомнить точно, почему он поступил так или иначе. Тогда это приведет к неверному толкованию действий респондента.

**Мысли вслух.**Этот популярный метод был предложен Якобом Нильсеном. Суть его заключается в том, что пользователь все свои действия проговаривает вслух. Однако при подобном поведении респонденты начинают более вдумчиво подходить к выполнению заданий и часть естественности теряется.

**Диалог с модератором.**Метод больше всего подходит для проведения качественных исследований прототипов и концепций. Во время тестирования респонденты активно общаются с модератором, задают ему вопросы и сразу дают фидбек.

**Теневой метод.** Одновременно работают три участника: респондент, модератор и эксперт. Респондент свободно и независимо выполняет задания, модератор фиксирует, эксперт комментируют действия респондента.

**Ретроспектива.**Это метод, который сочетает в себе наблюдение и «мысли вслух». Сначала респондент выполняет задания, затем — смотрит видеозапись своих действия и комментирует их. Основной недостаток — значительное увеличение времени тестирования.

# 5 МЕНЕДЖМЕНТ ТЕСТИРОВАНИЯ

## 5.1 Управление тестированием

Системы управления тестированием используются для хранения информации о том, как должным образом проводить тестирование, осуществление очередности проведения тестирования в соответствии с его планом, а также для получения информации в виде отчетов о стадии тестирования и качестве тестируемого продукта.

Инструменты имеют различные подходы к тестированию и, таким образом, включают в себя различные наборы функций. Обычно они используются для планирования ручного тестирования, сбора данных о результатах прохождения чек-листов и тест-кейсов, а также для получения оперативной информации в виде отчетов. Системы управления тестированием помогают оптимизировать процесс тестирования и обеспечивают быстрый доступ к анализу данных, средствам совместной работы и более качественному взаимодействию между несколькими проектными группами. Многие системы управления тестированием включают в себя возможность работы с требованиями. [6]

Например, TestRail. Это программное обеспечение удобно как для команд QA, так и для разработки. План тестирования можно выстроить как по сценарию гибкой методологии, так и для более традиционного подхода. Инструмент позволяет получить представление о ходе тестирования в реальном времени. [8] Возможности:

* отслеживание состояния и результатов отдельного теста,
* сравнение результатов нескольких тестов, конфигураций и этапов,
* отслеживание рабочей нагрузки команды для корректировки задач и ресурсов,
* развёрнутые отчёты и метрики,
* широкие возможности настройки, облачные или локальные варианты установки,
* интеграция с JIRA, Redmine, YouTrack, GitHub, Jenkins, Selenium и Visual Studio,
* удобный REST API.

## 5.2 Бизнес-ценность тестирования

[Тестирование программного обеспечения](https://www.a1qa.ru/" \t "_self" \o "Тестирование программного обеспечения) – это один из самых важных этапов в процессе создания продукта. Не уделив ему должного внимания, нельзя добиться высокого качества на выходе. Все крупные компании, занимающиеся разработкой программ и приложений для различных бизнес-целей, имеют в штате группу высококвалифицированных тестировщиков, которые привлекаются к участию в проекте на разных этапах. Каждый из них может специализироваться на каком-то конкретном тесте или группе тестов, что позволяет добиться более качественного результата.

Стоит отметить, что в настоящее время нельзя выделить какой-то конкретный вид тестирования, применение которого гарантирует высокое качество программного продукта. Чтобы добиться хорошего результата, необходимо использовать различные типы тестов, включая [юзабилити тестирование](https://www.a1qa.ru/usability_testing/). При этом применять стоит как ручные, так и автоматические тесты, комбинирование которых позволит выявить как можно больше ошибок и недочетов. Однако даже использование различных методов тестирования еще не гарантирует того, что в продукте абсолютно не будет ошибок. [7]

Независимо от того, какие методы используются для проверки качества программы и ее соответствия первоначально установленным требованиям, тестировщик обязательно должен изучить и проанализировать следующие факторы: функциональность, эффективность, надежность, мобильность, практичность и сопровождаемость разрабатываемого программного обеспечения или приложения. При этом в ходе проведения тестов нужно сформировать комплексный отчет, который будет подробно отражать информацию обо всех найденных ошибках и дефектах программного продукта. Этот отчет необходимо предоставить для анализа программистам, чтобы они могли устранить все дефекты, обнаруженные тестировщиками. Кроме того, такой отчет необходимо будет предоставить заказчику.

Чем позже тестировщики приступят к работам по проекту, тем менее качественным будет продукт. Соответственно, привлекать их к работе стоит уже на стадии обсуждения требований заказчика. Это поможет предотвратить некорректный выбор стратегий разработки продукта и значительно снизить количество возможных ошибок. Если не устранять ошибки на ранних этапах, а делать это после проведения основных работ по проекту, то стоимость всей разработки может возрасти в несколько раз, что абсолютно недопустимо.

## 5.3 Управление дефектами

После завершения процесса тестирования необходимо составить отчет, в котором описывают результаты тестирования, найденные дефекты.

В Процессе тестирования можно выделить 3 основные стратегии тестирования: тестирование черного ящика, серого ящика и белого ящика.

Под тестированием методом белого ящика традиционно понимают использование исходного, когда приложения в качестве основы для его отладки. Обычно такой вид тестирования выполняется непосредственно программистом, так как он подразумевает хорошее знание исходного кода. Такой вид тестирования может включать в себя отслеживание путей выполнения программы путем заранее известных входных значений на этих путях. [3]

Тестирование методом белого ящика не поможет определить несоответствие работы программы с заявленными требованиями, но оно может помочь выявить некоторые ошибки проектирования в исходном коде, такие как проблема управления потоком данных (бесконечные циклы) или в самих данных (использование неопределенной переменой). Статический анализ кода (с помощью инструментариев) так же поможет найти проблемы в программе, но не сможет помочь разработчику/тестеру код в той степени, в которой это делает тестирование белого ящика.

Как правило, тестирование белого ящика связано с тестированием компонентов системы на предмет полного покрытия всех требований кодом и что эти компоненты находятся в исправном состоянии. Но такие тесты не могу быть полными, так как тестовые примеры в таком виде тестирования являются искусственными и не могут выявить всех ошибок поведения программы в реальной среде.

Тестирование методом черного ящика применяется, когда тестер не имеет представления о внутреннем составе программы. Он может подавать какие-либо данные в качестве входных условий в тестируемое программное обеспечение и по полученным значениям определять, соответствует ли это обеспечение заявленным требованиям. [9]

Основными стратегиями тестирования черного ящика являются:

* разбиение на классы эквивалентности (используются, когда необходимо проверить множество входных условий, которые можно разбить на некоторые группы по формальным признакам и затем протестировать эти группы, взяв из каждой из них по 1-2 значения),
* анализ граничных значений (проверка работоспособности программы при работе с данными, выходящими за пределы допустимых для ввода значений или находящихся близко к этому рубежу),
* проверка причинно-следственных связей (проверка корректной работы пользовательских сценариев),
* свободный поиск ошибок (прохождение нетипичных сценариев, которые могут выявить ошибки в функционале работы программы, как правило, не очень высокого приоритета).

Тестирование методом серого ящика объединяет в себе методы тестирования методом «черного» и «белого» ящика. По методу черного ящика обычно осуществляется ввод данных, например - запуска программы внутри отладчика. [10]

# 6 ОТЧЕТ ПО ТЕСТИРОВАНИЮ ПРОЕКТА

## 6.1 Описание проекта

Целью данного проекта является тестирование историй в мобильном приложении «Юла». Необходимо протестировать их, составить чек лист проверок и тест-кейсы.

## 6.2 Чек-лист

Ниже приведены выполненные чек-листы:

* Добавление медиафайлов в историю
* Переход к соседним историям
* Подписка из истории
* Добавление ссылок в историях
* Проверка лимита историй
* Отправка жалоб на истории
* Удаление истории
* Переходы из истории
* Тестирование удаления историй
* Тестирование жалобы

## 6.3 Тест-кейсы

В таблице 2 содержатся тест-кейсы для тестирования историй.

Таблица 2 – Тест-кейсы для тестирования историй

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ теста** | **Описание** | **Начальные условия** | **Шаги** | **Ожидаемый результат** |
| 1 | Тестирование перехода на объявление из истории | Открыт главный экран приложения | 1. Тап на историю  2. Свайпнуть историю вверх  3. Увидеть объявление | Пользователь открыл историю  История корректно отобразилась  Открылся экран объявления |
| 2 | Тестирование перехода на свое объявление из истории | Открыт главный экран приложения | 1. Тап на свою историю 2. Свайпнуть историю вверх 3. Увидеть свое объявление | Пользователь открыл свою историю  История корректно отобразилась  Открылся экран своего объявления  Высвечивается уведомление о возможности продать быстрее за бонусы |
| 3 | Тестирование перехода в профиль из истории | Открыт главный экран приложения | 1. Тап на историю 2. Нажать на иконку или имя пользователя 3. Увидеть профиль и объявления пользователя | Пользователь открыл историю  История корректно отобразилась  Открылся экран профиля пользователя  Отобразились все объявления пользователя |
| 4 | Тестирование перехода в свой профиль из истории | Открыт главный экран приложения | 1. Тап на историю 2. Нажать на иконку или имя пользователя 3. Увидеть профиль и объявления пользователя | Пользователь открыл свою историю  История корректно отобразилась  Открылся экран собственного профиля пользователя  Отобразились все объявления пользователя |
| 5 | Тестирование функции добавления видео в историю | Открыт профиль в приложении | 1. Нажать иконку “Добавить историю” 2. Нажать кнопку “Добавить медиафайл” 3. Выбрать любое видео 4. Нажать кнопку “Создать историю” | Создаётся история с выбранным видео |
| 6 | Тестирование функции перехода к следующей истории | Открыта любая история | 1. Свайп справа налево | Показана следующая история |
| 7 | Тестирование функции перехода к предыдущей истории | Открыта любая история | 1. Свайп слева направо | Показана предыдущая история |
| 8 | Тестирование функции добавления ссылки на профиль в истории | Открыто окно добавления истории | 1. Фото или видео добавлено для загрузки  2. Тапнуть на пункт Ссылка в истории  3. Выбрать пункт «мой профиль» | Ссылка на профиль прикреплена |
| 9 | Тестирование функции добавления ссылки на объявление в истории | Открыто окно добавления истории | 1. Фото или видео добавлено для загрузки  2. Тапнуть на пункт Ссылка в истории  3. Выбрать пункт «объявление» | Ссылка на объявление прикреплена |
| 10 | Тестирование функции перехода по ссылке на профиль из истории | Открыта любая история с прикрепленной ссылкой на профиль | 1. Тапнуть по кнопке «Показать профиль» | Профиль пользователя открылся |
| 11 | Тестирование функции перехода по ссылке на объявление из истории | Открыта любая история с прикрепленной ссылкой на объявление | 1. Тапнуть по кнопке «Показать объявление» | Объявление открылось |
| 12 | Тестирование удаления истории | Открыто окно с загруженной историей | 1.Тапнуть по кнопке дополнительное меню (три точки)  2.Нажать удалить историю | История удалена |
| 13 | Тестирование жалобы | Открыто окно с любой историей | 1.Тапнуть по кнопке дополнительное меню (три точки)  2.Нажать на кнопку пожаловаться | Сообщение, жалоба успешно отправлена. |

## 6.4 Найденные дефекты

Фактический результат соответствует ожидаемому во всех случаях.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работ были пройдено введение в тестирование, изучена клиент-серверная архитектура изучено тестирование WEB-приложений, а также протестировано мобильное приложение «Юла» в модуле историй. Были составлены чек-лист и тест-кейсы.

**Ссылки003A**

Testrail:

Address: https://historyyula.testrail.io/

Username: gdr92660@cuoly.com

Password: BTekaG6L20qFE0htW2Z1

GitHub: https://github.com/ivlevme/Testing-stories

Пост в социальной сети: https://vk.com/club196909331?w=wall-196909331\_3

Trello: https://trello.com/b/x6f6Dipe/юла-команда-по-тестиров..

Google-диск на видео процессов тестирования:

https://drive.google.com/drive/folders/15udw4saNOfDrU..

Сайт (лендинг): http://pd-2020-2.std-1313.ist.mospolytech.ru/

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Список обучающих лекций по работе над тестирование под редакцией «Lime Lab»
2. Тестовая документация [Электронный ресурс]. – Режим доступа: <https://zhenyabrik.gitbooks.io/matrixcopy/content/lektsiya-4-ch1-testovaya-dokumentatsiya.html>
3. Управление дефектами в тестировании программного обеспечения [Электронный ресурс]. – Режим доступа: https://logrocon.ru/news/defects
4. Юзабилити-тестирование [Электронный ресурс]. – Режим доступа: https://clck.ru/EwrES
5. Юзабилити-тестирование [Электронный ресурс]. – Режим доступа: https://usabilitylab.ru/usability/yuzabiliti-testirovanie/
6. Важность тестирования в разработке ПО [Электронный ресурс]. – Режим доступа: https://mycompplus.ru/news/53-obzor-softa/2420-vazhnost-testirovaniya-vrazrabotke-po.html
7. Управление дефектами в тестировании программного обеспечения [Электронный ресурс]. – Режим доступа: https://studwood.ru/1653964/informatika/upravlenie\_defektami
8. Системы управления тестированием [Электронный ресурс]. – Режим доступа: https://clck.ru/Sx2Ac
9. Лучшие системы управления тестированием [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/461205/
10. Управление дефектами [Электронный ресурс]. – Режим доступа: https://studwood.ru/1653964/informatika/upravlenie\_defektami