МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА ПО ДИСЦИПЛИНЕ

«ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ»

Лаборатория Тестирования

Куратор проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Яковлев Станислав Игоревич, Преподаватель/ подпись ФИО, уч. звание и степень

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Летов Григорий, 201-351

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Николаев Владислав,

201-351

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Ласкин Владислав,

191-331

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Рогожин Егор,

201-331

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Коротаев Кирилл, 201-331

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Штырба Владислав, 201-725

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Колбая Роберт, 191-725

Москва, 2022

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc107341887)

[**Введение** 4](#_Toc107341888)

[План работ 8](#_Toc107341889)

[1 Тестирование программного обеспечения 10](#_Toc107341890)

[1.1 Тестовая документация 10](#_Toc107341891)

[1.2 Отчет о дефекте 11](#_Toc107341892)

[2 Тестирование веб-сервисов 15](#_Toc107341893)

[2.1 Тестирование API 15](#_Toc107341894)

[2.2 Тестирование WEB 17](#_Toc107341895)

[2.3 Нагрузочное тестирование 18](#_Toc107341896)

[2.4 Автоматизированное тестирование 19](#_Toc107341897)

[3 ТЕСТИРОВАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ 22](#_Toc107341898)

[3.1 Ручное тестирование 23](#_Toc107341899)

[3.2 Автоматизированное тестирование 25](#_Toc107341900)

[4 UX ТЕСТИРОВАНИЕ 27](#_Toc107341901)

[5 МЕНЕДЖМЕНТ ТЕСТИРОВАНИЯ 30](#_Toc107341902)

[5.1 Управление тестированием 30](#_Toc107341903)

[5.2 Бизнес-ценность тестирования 30](#_Toc107341904)

[5.3 Управление дефектами 31](#_Toc107341905)

[6 Отчет по тестированию проекта 33](#_Toc107341906)

[6.1 Описание проекта 33](#_Toc107341907)

[6.2 Баг-репорты 33](#_Toc107341908)

[6.3 Device farm. 35](#_Toc107341909)

[6.4 Практика с Postman и Google DevTools 38](#_Toc107341910)

[6.5 Тестирование YaSpeller при помощи SoapUI 42](#_Toc107341911)

[Заключение 46](#_Toc107341912)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47](#_Toc107341913)

**Введение**

Проект по тестированию мобильного приложения «Юла» представляет собой поиск ошибок в работе приложения, которые могут помешать пользователю комфортно использовать данный программный продукт.

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

Для учета обнаруженных ошибок будут использоваться такие программные средства как:

TestRail – инструмент для управления основными процессами тестирования, начиная от регистрации и заканчивая сроками решения и выполнения работы.

Google Таблицы – для ведения учёта основных аспектов работы.

Trello – программа для менеджмента и учёта сроков выполнения основных работ.

Postman – программное средство для тестирование API.

Qase.io – средство управления тест-кейсами.

SoapUI – приложение для тестирования веб-сервисов сервис-ориентированных архитектур и передачи состояний представлений.

Charles Proxy – кроссплатформенное приложение прокси-сервера отладки http.

Данная работа очень актуальна, так как тестирование приложения напрямую влияет на выявление ошибок и критических сбоев в работе приложения, а также помогает обезопасить приложение, сохранить деньги компании и представить пользователю хороший качественный продукт.

**Аннотация**

В данной работе представлены методологии тестирования мобильных приложений и API. Также представлены результаты выполнения заданий.

**Участники и их роли**

Николаев В. 201-351 – стажер-тестировщик

Летов Г. 201-351 – стажер-тестировщик

Ласкин В. 191-331 – стажер-тестировщик

Рогожин Е. 201-331 – стажер-тестировщик

Коротаев К. 201-331 – стажер-тестировщик

Штырба В. 201-725 – стажер-тестировщик

Колбая Р. 191-725 – стажер-тестировщик

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

**Индивидуальные планы участников:**

Николаев Владислав:

* Изучение теоретического материала в объеме 46 часов
* Выполнение практических заданий 10 часов
* Тестирование мобильной версии WEB-приложения «Юла» 19 часов

Летов Григорий:

* Изучение теоретического материала в объеме 46 часов
* Выполнение практических заданий 10 часов
* Тестирование мобильной версии WEB-приложения «Юла» 19 часов

Ласкин Владислав.

* Изучение теоретического материала в объеме 46 часов
* Выполнение практических заданий 10 часов
* Тестирование мобильной версии WEB-приложения «Юла» 19 часов

Рогожин Егор.

* Изучение теоретического материала в объеме 46 часов
* Выполнение практических заданий 10 часов
* Тестирование мобильной версии WEB-приложения «Юла» 19 часов

Коротаев Кирилл:

* Изучение теоретического материала в объеме 46 часов
* Выполнение практических заданий 10 часов
* Тестирование мобильной версии WEB-приложения «Юла» 19 часов

Штырба Владислав:

* Изучение теоретического материала в объеме 46 часов
* Выполнение практических заданий 10 часов
* Тестирование мобильной версии WEB-приложения «Юла» 19 часов

Колбая Роберт:

* Изучение теоретического материала в объеме 46 часов
* Выполнение практических заданий 10 часов
* Тестирование мобильной версии WEB-приложения «Юла» 19 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, Trello. | 14 | 14 | - |
| Тестирование в жизненном цикле разработки ПО. | 12 | 12 | - |
| Тестовая документация. Qase.io, TestRail. | 13 | 13 | - |
| 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. | 14 | 14 | - |
| 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 | 18 | 12 | 6 |
| Тестирование мобильных приложений |  |  |  |
| Особенности тестирования мобильных приложений. | 16 | 16 | - |
| Тестирование Android приложений | 20 | 12 | 8 |
| Тестирование мобильных приложений iOS | 20 | 12 | 8 |
| Сбор и анализ трафика (Charles Proxy) | 18 | 14 | 4 |
|  |  |  |  |
| Подготовка отчета |  |  |  |
| Создание отчета | 36 | - | - |
| Подготовка постера | 10 | - | - |
| Подготовка презентации | 6 | - | - |
| Запись видеопрезентации | 4 | - | - |
| Подготовка Git | 3 | - | - |

# 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.

Цель тестирования – проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.

Дефект – отклонение фактического результата от ожиданий наблюдателя, сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.

Ошибка – действие человека, приводящее к некорректным результатам.

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

Сбой – самоустраняющийся или однократный отказ, устраняемый незначительным вмешательством.

Отказ – событие, нарушающее работоспособность приложения

## Тестовая документация

Среди видов тестовой документации выделяют:

* Требования. Регламент того, какие функции должно выполнять приложение в процессе решения полезной для пользователя задачи. Требования делятся на бизнес-требования, пользовательские и системные. Бизнес-требования описывают основные цели разработки, а также бизнес-задачи, которые должна решать система. Пользовательские требования – функции, которые должна иметь система с точки зрения пользователя. Системные требования – описание основных условий, необходимых для запуска и работы ПО. Например, аппаратное обеспечение;
* Тест-план. Документ, в котором описывается объем работ, используемые подходы к тестированию, ресурсы, расписание, а также определяются функции и модули для тестирования, тестовые задачи, ответственные лица и необходимые метрики. Необходим для эффективного распределения работ, понимания итогового объема работы и временных рамок;
* Чек-листы и тест-кейсы. Чек-лист – это набор (список) проверок. Создается и используется, как напоминание о важных тестах. Тест-кейс – полноценный тестовый сценарий для определенной функции, содержащий набор предварительных условий, входных данных, действий (где применимо), ожидаемых результатов и постусловий, разработанных на тестовых условиях.

## Отчет о дефекте

Любой дефект, обнаруженный в приложении или системе, должен быть задокументирован и передан разработчику либо другому ответственному специалисту для ознакомления и исправления. Документ, с помощью которого о дефекте становится известно миру или отдельному специалисту, называется отчётом о дефекте. Есть несколько определений этого понятия.

Отчёт о дефекте (bug report) – это документ, описывающий ситуацию, которая привела к обнаружению дефекта, с указанием причин и ожидаемого результата.

Отчёт о дефекте – документ, описывающий и приоритизирующий обнаруженный дефект, а также содействующий его устранению.

Корректно составленный отчёт о дефекте очень важен в разработке приложения. Он позволяет не только зафиксировать обнаруженную неисправность в программе и информировать о ней, но и помогает её исправить. Чем грамотнее и точнее описан дефект, а также возможные причины его возникновения, тем проще и быстрее он будет исправлен.

Кроме того, отчёты о дефектах – инструменты для сбора статистики на проекте. Они помогают определить, в каких областях приложения и при каких условиях концентрируются дефекты. Ещё одна важная функция отчётов о дефектах – приоритизация проблем, обнаруженных в приложении. При большом количестве найденных дефектов и ограниченных временных резервах разработчику важно понимать, какие дефекты хуже всего влияют на работу приложения и должны быть исправлены в первую очередь.

Качественный отчёт о дефекте не только предоставляет все необходимые подробности для понимания сути случившегося, но также может содержать анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

Атрибуты отчёта о дефекте

1. Уникальный идентификатор (ID) –присваивается автоматически, может содержать в себе данные о требовании, на которое ссылается дефект.
2. Тема (краткое описание, Summary) – кратко сформулированная суть дефекта по правилу «Что? Где? Когда?»
3. Подробное описание (Description) –более широкое описание сути дефекта (при необходимости).
4. Шаги для воспроизведения (Steps To Reproduce) –последовательное описание действий, которые привели к выявлению дефекта (которые нужно выполнить для воспроизведения дефекта). Описываются максимально подробно, с указанием конкретных вводимых значений.
5. Фактический результат (Actual result) – указывается, что не так работает, в каком месте продукта и при каких условиях. Описывая фактический результат, необходимо ответить на три вопроса: что? где? когда?
6. Ожидаемый результат (Expected result) – указывается, как именно должна работать система по мнению тестировщика, основанному на требованиях и прочей проектной документации.
7. Вложения (Attachments) – скриншоты, видео или лог-файлы.
8. Серьёзность дефекта (важность, Severity) – характеризует влияние дефекта на работоспособность приложения.
9. Приоритет дефекта (срочность, Priority) – указывает на очерёдность выполнения задачи или устранения дефекта. Чем выше приоритет, тем быстрее нужно исправить дефект.
10. Статус (Status) – определяет текущее состояние дефекта. Отражает жизненный цикл дефекта от начального состояния до завершения. Названия статусов дефектов могут быть разными в разных баг-трекинговых системах.

Дополнительно могут встречаться такие атрибуты, как:

* компонент или среда – указывает, на какой платформе этот дефект воспроизводится (iOS, Android, Windows, Mac);
* версия – указывает, на каком этапе разработки программного продукта был обнаружен дефект;
* назначение – указывается личность, на которую данный баг-репорт назначается для дальнейшей проверки или исправления;
* номер сборки – указывается номер сборки, в котором был обнаружен дефект.

# Тестирование веб-сервисов

## Тестирование API

Web API – составляющая сервера, которая содержит, как правило, определённый набор HTTP-запросов, а также определение структуры HTTP-ответов, для выражения которых используют XML− или JSON−формат.

HTTP – 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. Существует также версия данного протокола, которая использует шифрование данных – HTTPS.

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 – архитектурный стиль взаимодействия компонентов распределённого приложения в сети, иными словами, правила написания API.

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».

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.

В HTTP-ответе сервера содержится код состояния. Ниже приведены 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 запрос корректен;
* 400 Bad Request – задан неверный запрос;
* 401 Unauthorized – пользователь не авторизован;
* 403 Forbidden – доступ запрещен. Он означает, что аутентификация прошла успешно, но у пользователя нет соответствующих прав;
* 404 Not Found – не найдено, ресурс по данному URI отсутствует;
* 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.

Для тестирования API необходимы инструменты, где можно отправить входные данные в запросе и проверить точность выходных данных. Одним из таких инструментов является Postman. Данная программа позволяет писать и отправлять запросы, сохранять их, создавать различные окружения для одинаковых запросов, запускать коллекции запросов для автоматизированных тестов.

## Тестирование WEB

Веб-приложение – клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети.

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.

Клиент-сервер —это вид архитектуры распределенного приложения. В этой архитектуре есть два типа элементов: клиенты и сервер. Они взаимодействуют определенным способом, который называется протоколом.

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 протокол HTTP или HTTPS.

HTTP 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.

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:

* 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, 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.

## 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.

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.

## 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.

Достоинства автоматического тестирования:

* 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.

# ТЕСТИРОВАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

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

Рассмотрим основные различия между мобильным приложением и его веб-версией:

* 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, Xiaomi), 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 оболочек операционных систем и компонентных конфигураций.
* 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.

## 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.

Необходимо, 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. Это тоже негативно сказывается на сроках поставки функционала в конечный продукт и трудозатратах: ведь теперь к периодически проводимым базовым кейсам и регрессии добавляются и 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 новых функций, чтобы убедиться, что они не ломают существующий функционал, а тестировщику не придётся лишний раз проверять базовые кейсы. 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;
* кейс автотеста должен быть прописан подробно и описывать возможную проблему или существующий сценарий во всей полноте;
* необходимо проверить, 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.

Недостатки ручного тестирования мобильных приложений:

1. Ручные тестовые примеры трудно использовать повторно.
2. Менее эффективно выполнение определенной постоянной задачи.
3. Процесс тестирования медленный.
4. Некоторые типы тестовых случаев не могут быть выполнены вручную (нагрузочное тестирование).

## Автоматизированное тестирование

Автоматизация тестирования помогает решить сразу несколько проблем. 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:

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

Преимущества автоматизированного тестирования приложений:

1. Процесс тестирования занимает мало времени.
2. Экономичность в долгосрочной перспективе использования.
3. Автоматизированные тестовые случаи легко использовать повторно.
4. Единственное решение для некоторых видов тестирования (тестирование производительности).
5. Результаты испытаний легко доступны.

Недостатки автоматизированного тестирования приложений:

1. У некоторых мобильных средств тестирования есть ограничения.
2. Процесс тестирования занимает много времени.

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

# UX ТЕСТИРОВАНИЕ

Юзабилити-тестирование – это сценарный план, предложенный специалистом на основе анализа вашего ресурса или виртуального бюро услуг. Цель юзабилити-анализа – экспериментальным путем установить, удобен ли ресурс для клиентов, а если нет – вывести парадигму требуемых улучшений.

Только на основе юзабилити-тестирование вы сможете понять – какие исходные данные нужно использовать для A/B-тестирования (сравнительного анализа существующей и экспериментальной страницы, на который изменен один или ряд фрагментов).

Определив группу задач, которые будут решаться в ходе юзабилити-тестирования, вы даете возможность потенциальным клиентам озвучить свои недовольства и пожелания.

Базовый набор инструментов юзабилити-тестирования включает в себя:

* аналитические системы;
* web-визор;
* эвристический анализ.

К примеру, в процессе юзабилити-тестирования мобильных приложений (такая потребность чаще встает перед маленькими предприятиями), вам желательно заручиться достаточным количеством пользовательских отзывов. Это очень важно, ведь вам нужен ответ на вопрос: почему часть клиентов ушла к другим продавцам, не совершив какие-либо действия именно на вашем ресурсе.

В зависимости от поставленных вами задач по улучшению конверсии сайта и особенностей вашей желаемой целевой аудитории, можно провести:

* классическое юзабилити-тестирование;
* онлайн-юзабилити-тестирование.

Кроме того, вам не составит труда понять, – как проводить юзабилити-тестирование. Последовательность ваших действий будет примерно такой:

* выбор сервиса автоматизации;
* выбор фокус-группы;
* проведение юзабилити-тестирования в течение нескольких дней посредством установки у пользователей софта для захвата и записи экрана.

Хорошо, если разработчики вашего ресурса убедили вас провести юзабилити-тестирование еще до того, как вы окунулись в мир реальных продаж. В этом случае изначально можно быть уверенным в том, что посадочная страница либо страницы каталога, формы выбора товара и оплаты соответствуют устоявшимся представлениям клиентов об удобстве пользования.

Следовательно, обратившись к специалистам, вы сможете быть уверенными в том, что для вашего ресурса подберут тематических клиентов либо профессиональных пользователей.

Еще один плюс проведения юзабилити-тестирования узкопрофильной web-компанией: по факту завершения анализа вам предоставят не просто отчет о выявленных ошибках, а пакет рекомендаций по улучшению юзабилити ресурса.

Нельзя ограничивать фокус-группу небольшим количеством пользователей: безусловно, грубые ошибки помогут установить и первые же пять человек, однако глубинные недочеты могут укрыться от их внимания.

Еще одно преимущество курирования процесса профессионалом – специалист точно знает, как проводить юзабилити-тестирование, иначе говоря, он верно поставит задачи и распределит их последовательность.

Прежде чем юзабилити-тестирование будет запущено, необходимо провести подготовительную работу:

* вычленить KPI (ключевые показатели эффективности) вашего ресурса: как правило, это оформление заказа, запрос на обратный звонок, подписка на рассылку;
* вычленение слабых звеньев (например, страниц, с которых пользователь, открыв, сразу уходит) и «холодных этапов», когда, например, пользователь уходит с сайта, дойдя до фильтра товаров;
* написание сценарного плана тестирования (к примеру, составление модулей предполагаемых действий пользователя: «посмотреть каталог – заказать звонок», «посмотреть каталог – отложить в корзину», «посмотреть каталог – оформить покупку»);
* формулирование ТЗ для пользователя фокус-группы, к примеру, «Вы – девушка 20 лет, ищите оригинальный подарок для шефа своей компании стоимостью от столько-то до столько-то».

Как показывает практика, по итогам юзабилити-тестирования могут вскрыться совершенно неожиданные для владельца ресурса причины непопулярности сайта или интернет-магазина у целевой группы. К примеру, некоторое количество пользователей обескураживает и отталкивает необходимостью пройти многоступенчатую регистрацию, прежде чем оформить покупку. Иногда пользователь прерывает процесс выбора товара, столкнувшись с обилием полей, которые необходимо заполнить для перехода в нужный сегмент каталога.

И самое важное: организация юзабилити-тестирования должна охватить и выявить как можно больше проблемных зон ресурса. Лишь в этом случае у специалиста будет достаточное количество исходных данных для составления для вас указаний по улучшению юзабилити.

# МЕНЕДЖМЕНТ ТЕСТИРОВАНИЯ

## Управление тестированием

Для корректного взаимодействия между членами команды тестировщиков и организации грамотного тестирования продукта, используется единая “СУТ” (Система управления тестированием). Системы управления тестированием обычно 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.

После старта тестирования проекта члены команды могут взаимодействовать через одну из систем управления тестирования путём создания тест-кейсов, чек-листов, назначая ответственных за прохождение их лиц, что упрощает и улучшает качество взаимодействия лиц, проводящих тестирование в рамках конкретного проекта. При создании или прохождении тестов и чек-листов пользователи могут получить доступ к различным функциям систем управления тестированием, которые автоматизируют данную деятельность и благотворно влияют на скорость и качество её выполнения.

## Бизнес-ценность тестирования

На сегодняшний момент сложно отрицать ценность специалистов по тестированию. Основная задача данного специалиста - предотвратить дефекты и, следовательно, обеспечить высокое качество процесса разработки и его результатов. Для более детального рассмотрения функций составим список задач специалиста по тестированию продукта:

* Выявление слабых мест и несоответствия в продукте на всех этапах разработки.
* Помощь в определении требований к проекту.
* Предоставление исчерпывающей информации о качестве продукта.
* Тестирование продукта на протяжении всех фаз жизненного цикла разработки системы (software development lifecycle, SDLC).

Важно отметить, что специалисты по тестированию заинтересованы в том, чтобы сделать любой продукт удобным для пользователя как в плане функциональности, так и в плане дизайна. Для этого они тесно взаимодействуют со всеми членами команды и постоянно обращаются к заданным требованиям.

Влияние тестирования на бизнес:

* Безопасный бизнес.
* Экономия денег.
* Защита репутации.
* Контроль процесса.

## Управление дефектами

Для управления дефектами применяют баг-трекинговые системы, которые упрощают запись и ведение информации о дефектах.

При обнаружении дефект проходит следующий жизненный цикл:

* Дефект открыт, когда его обнаружили и создали отчёт о нем.
* Дефект в работе, когда он назначен на разработчика и разработчик начал работу над ним.
* Дефект исправлен, когда разработчик внес изменения (исправление или fix) в программный код и сохранил эти изменения в общем репозитории.
* Дефект закрыт, когда было проверено, что он действительно исправлен в новой версии продукта.
* Дефект открыт заново в случае, если ранее исправленный дефект снова был обнаружен в системе.

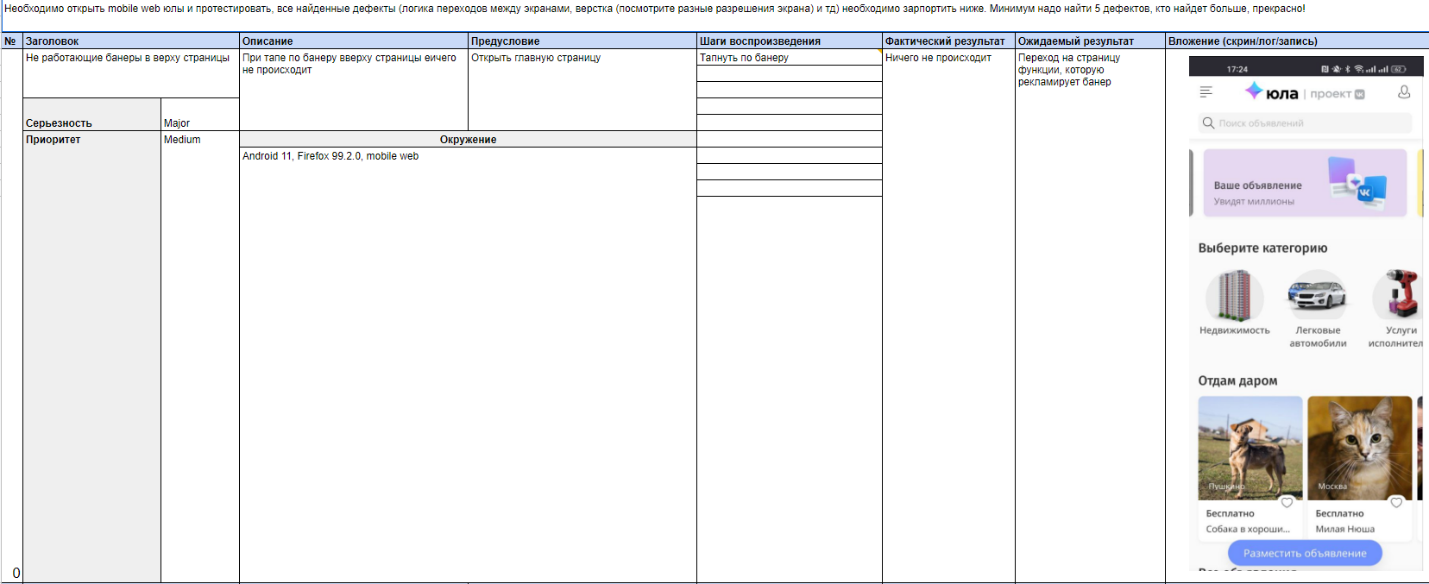
# Отчет по тестированию проекта

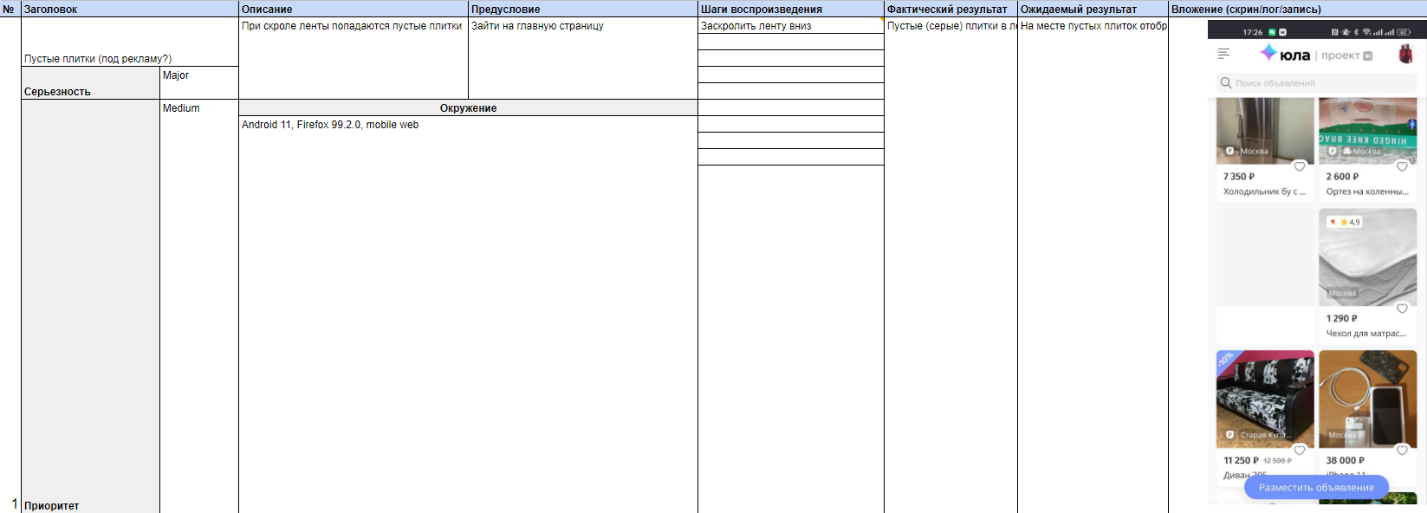
## Описание проекта

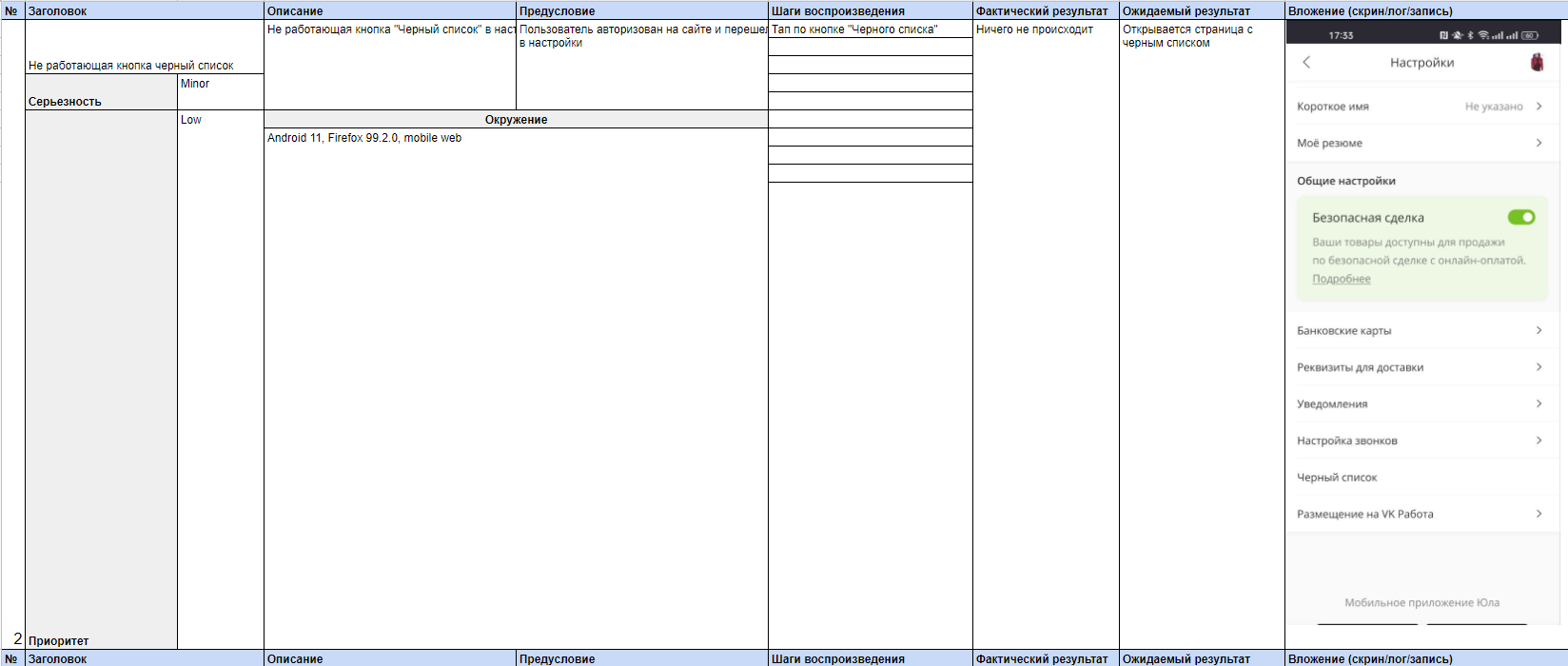
Целью данного проекта является получение практических навыков ручного тестирования ПО. В данном семестре необходимо было выполнить несколько заданий, связанных с применением специализированного ПО для тестирования приложений, а также составить баг-репорты после тестирования мобильной версии web-приложения «Юла».

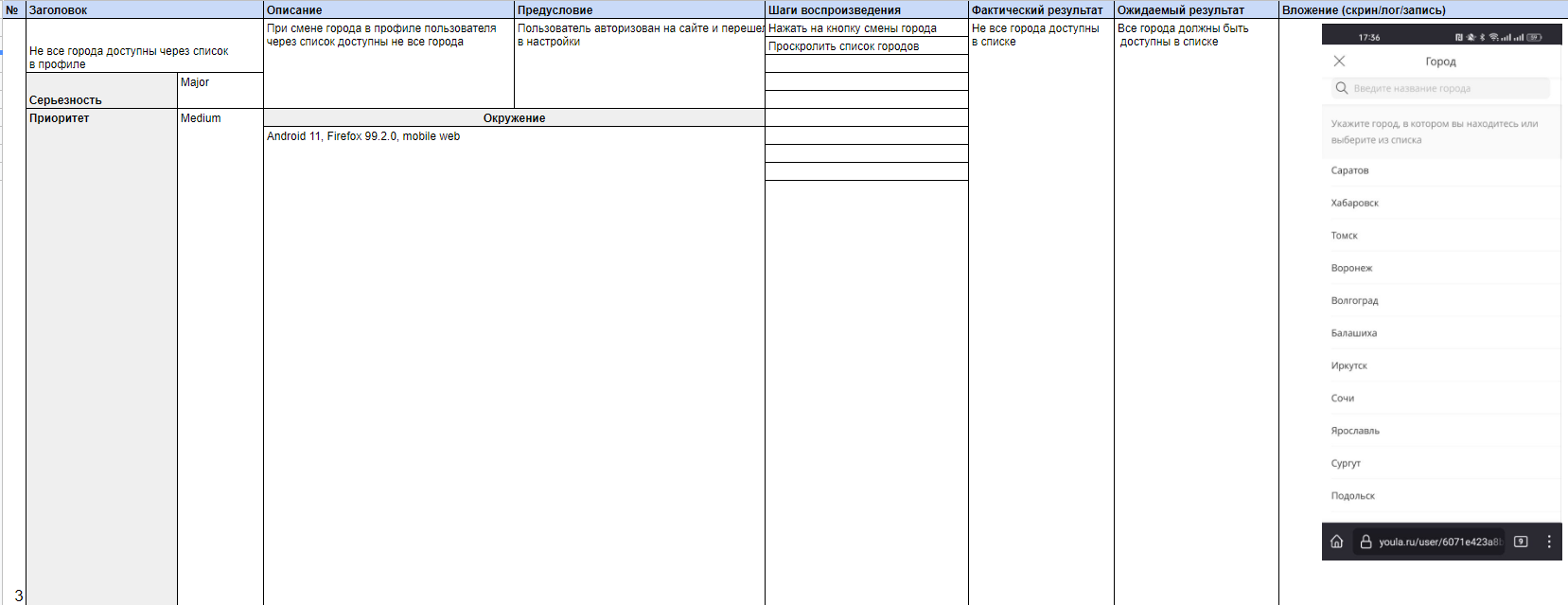
## Баг-репорты

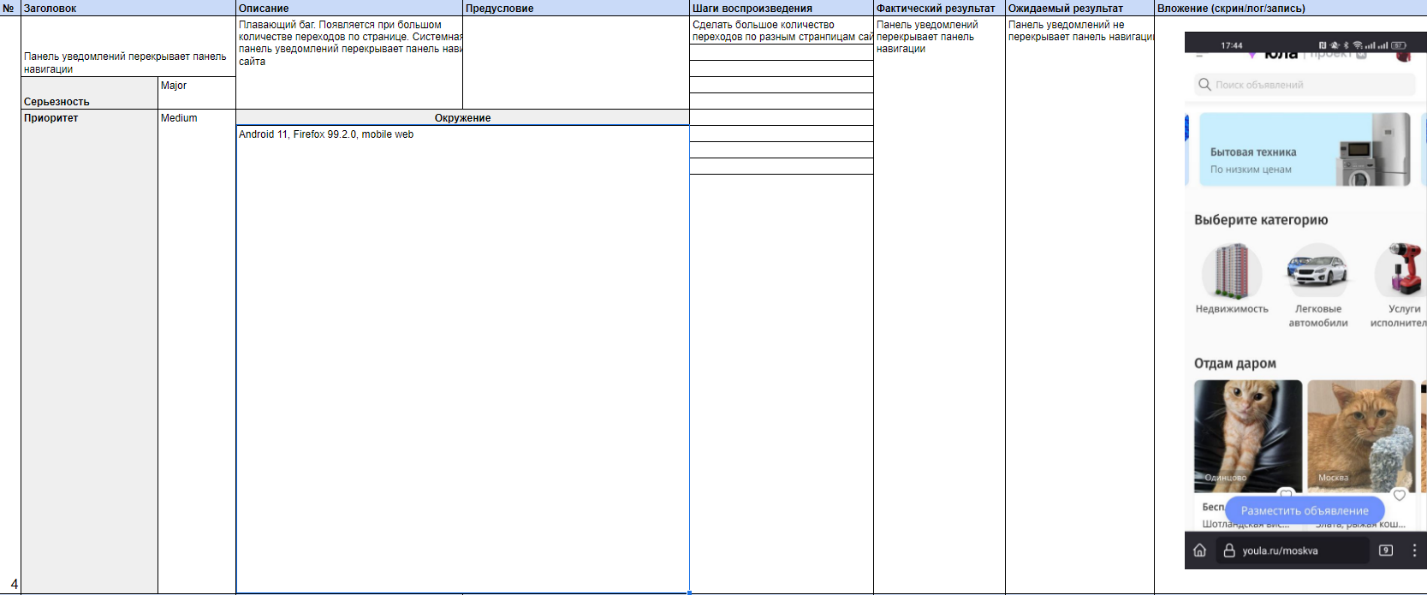
Далее представлены примеры баг-репортов, составленных одним из участников проекта:







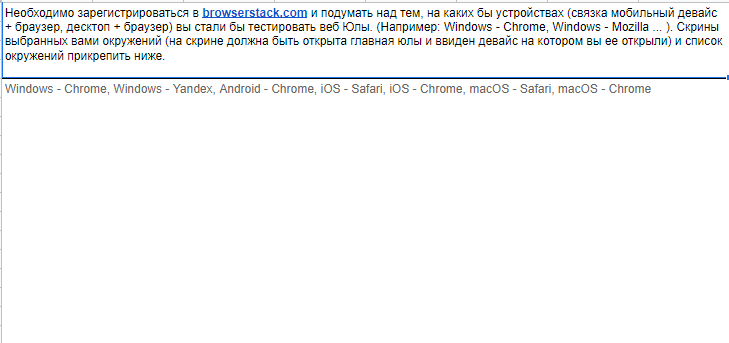


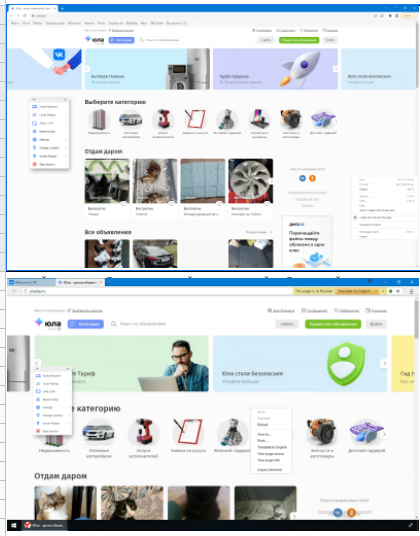


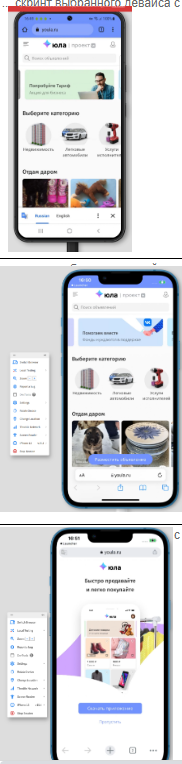
С другими отчетами можно познакомиться по ссылке: <https://docs.google.com/spreadsheets/d/1FqM-aTHtOAaxi5U3kZg-FfqKzMhOz8PKQhb4FO14r1w/edit#gid=882474577>

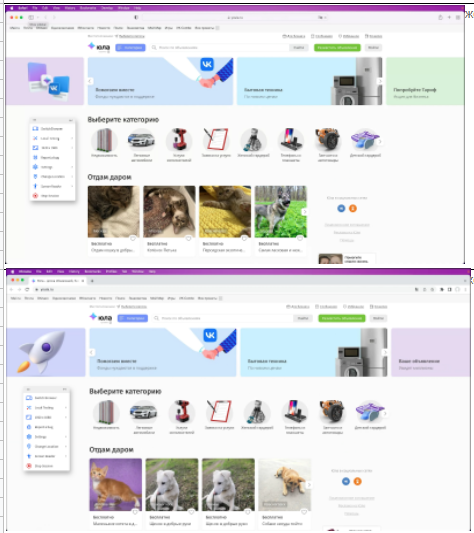
## Device farm.

Данное задание подразумевало знакомство с инструментом для тестирования приложения в разных средах.



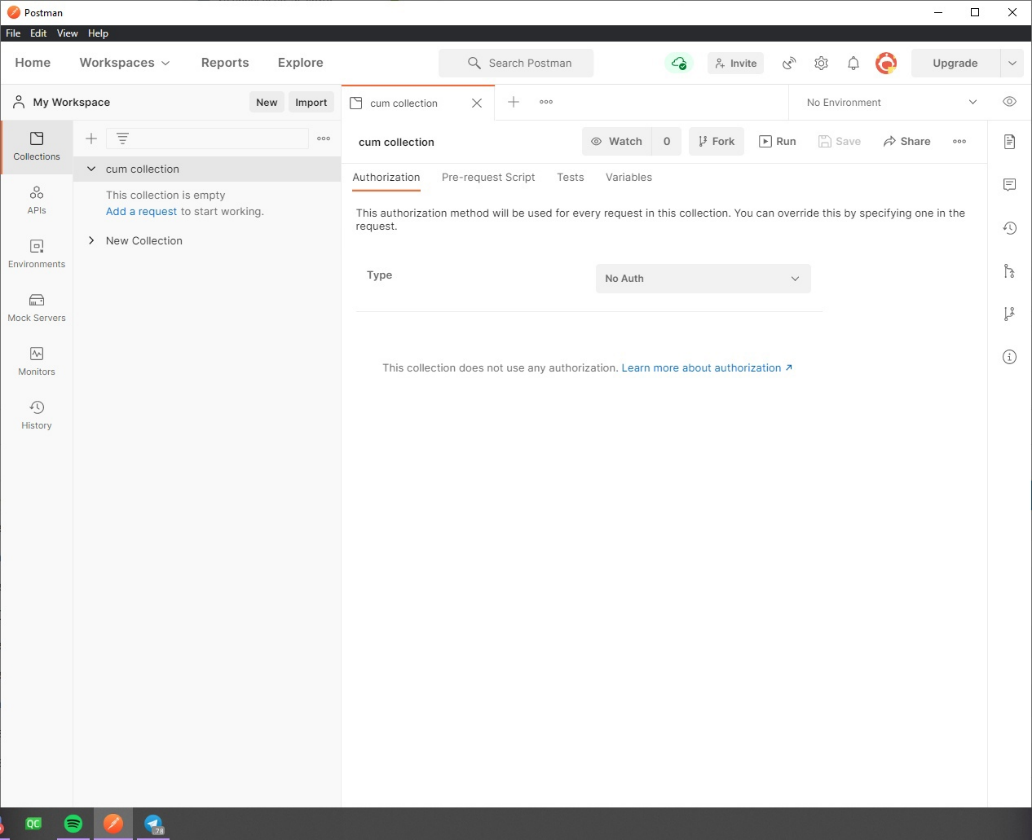


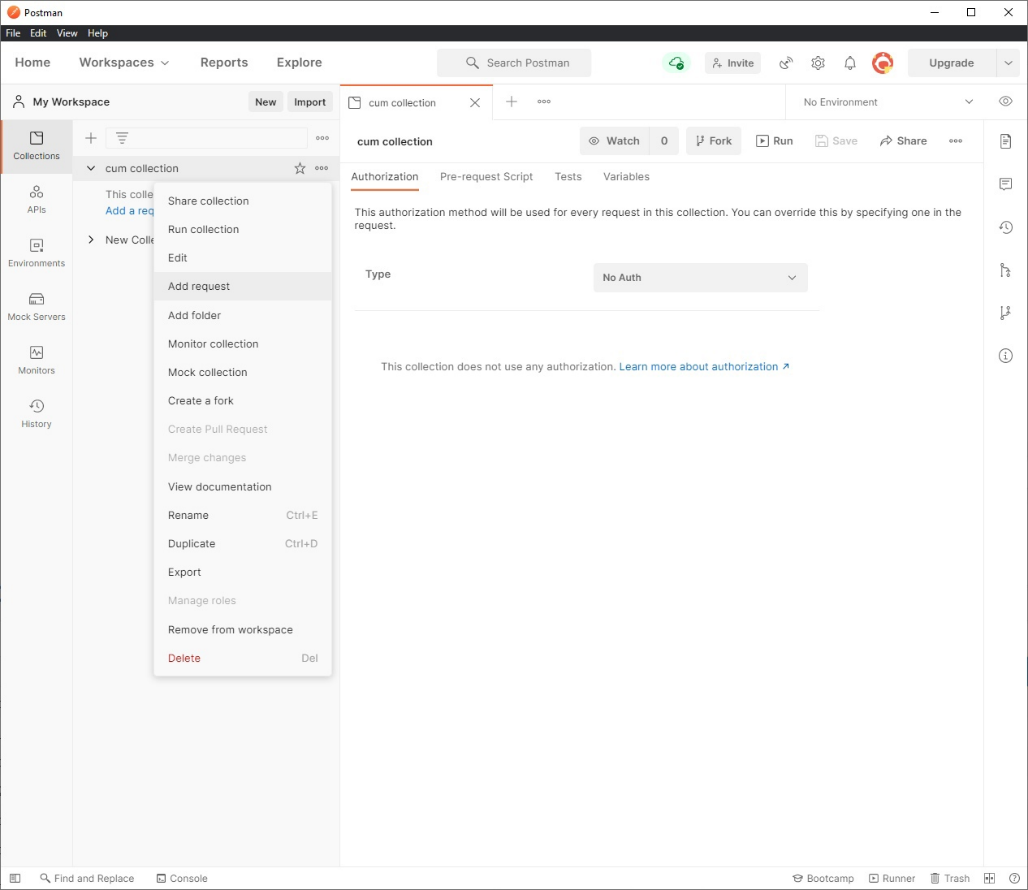


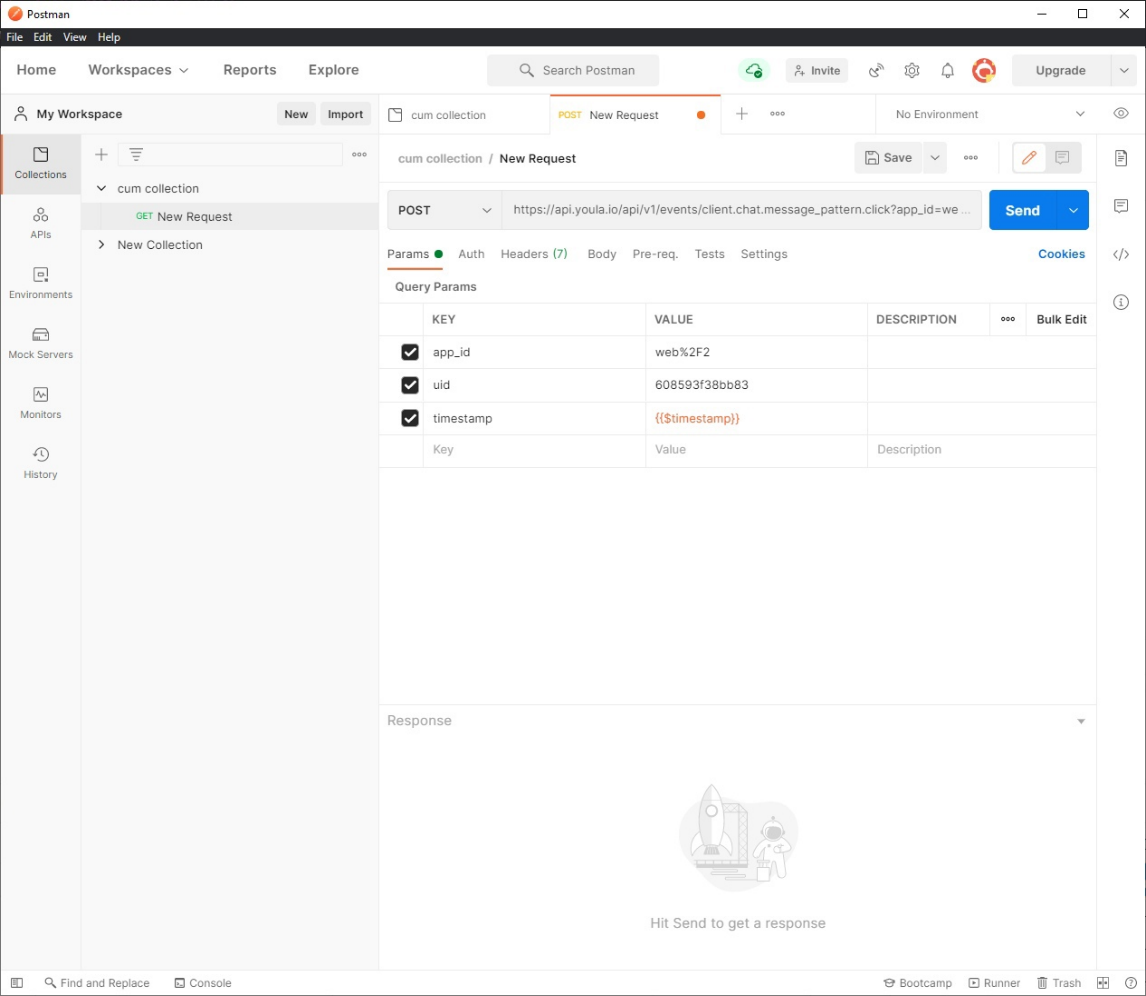


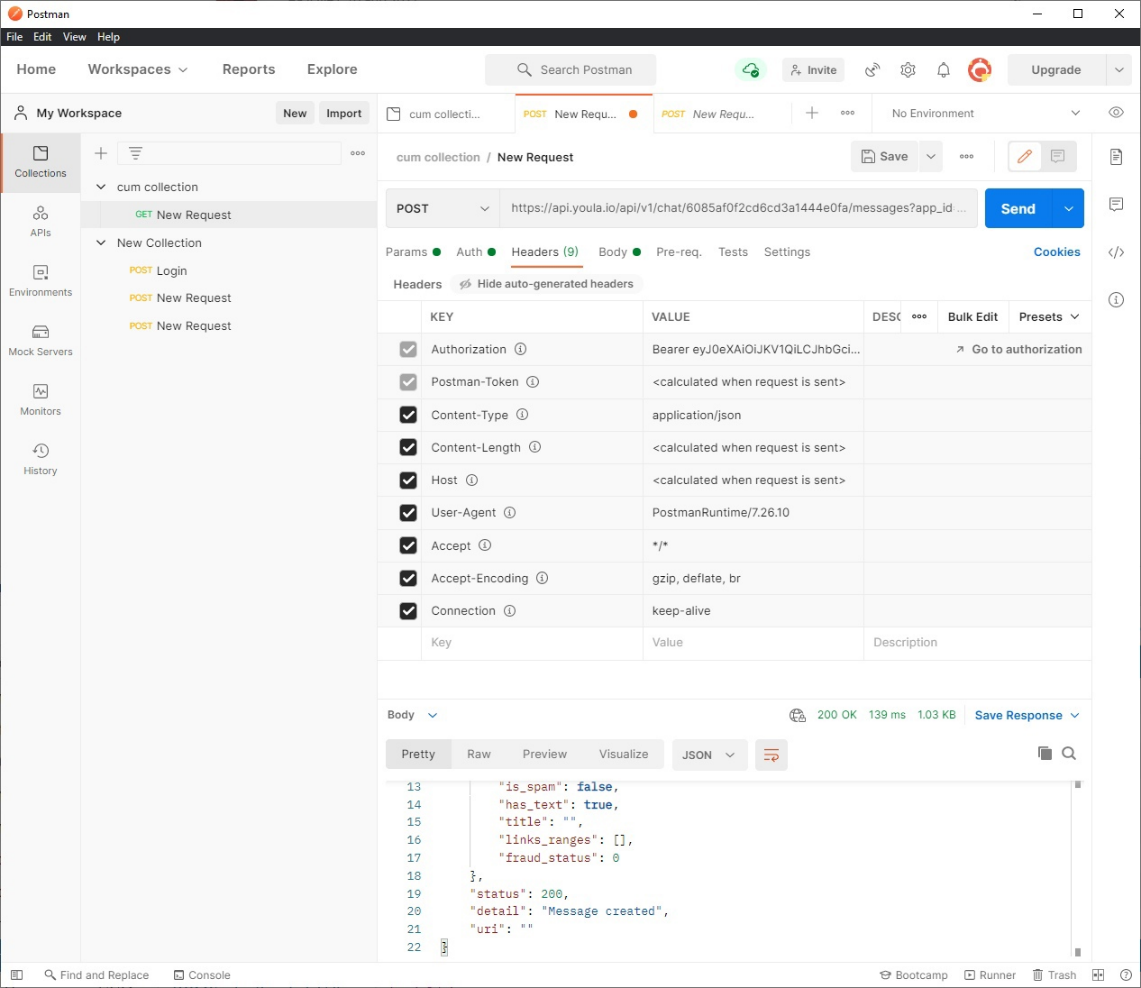
## Практика с Postman и Google DevTools

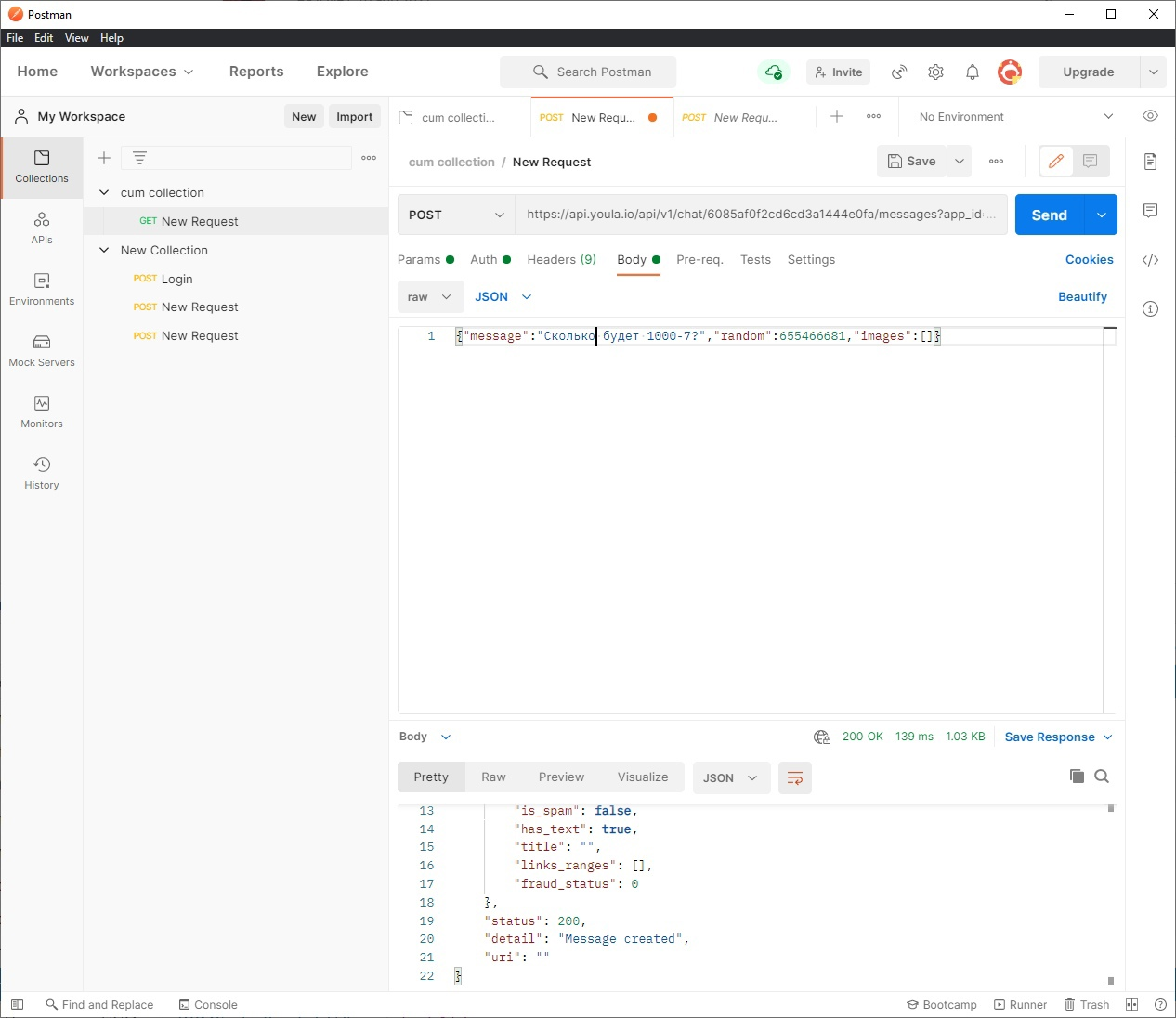
Далее на рисунках представлен результат практической деятельности по данной теме.

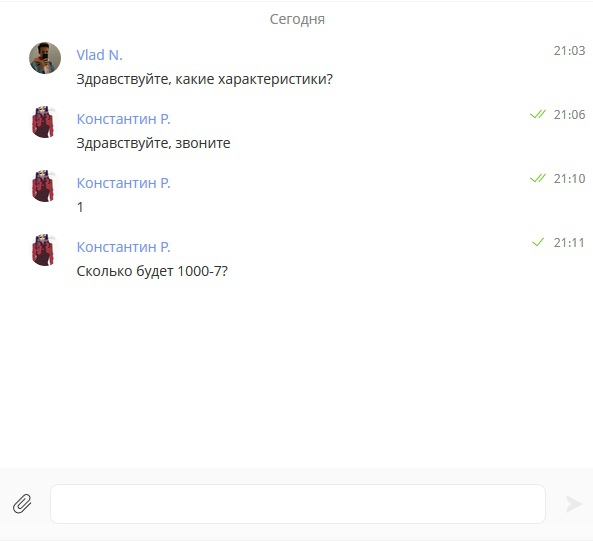
****

****

****

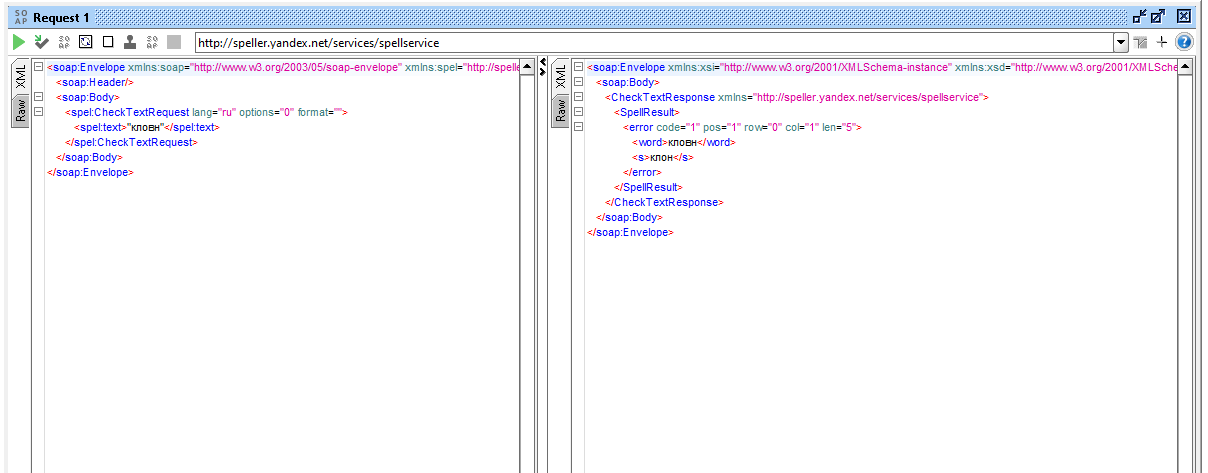
****

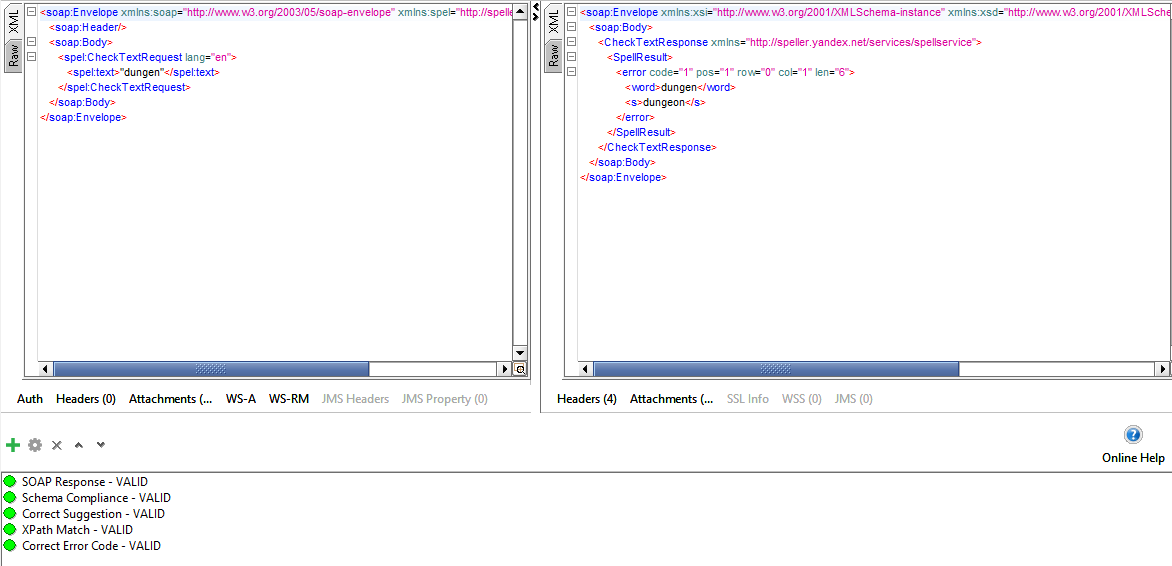
****

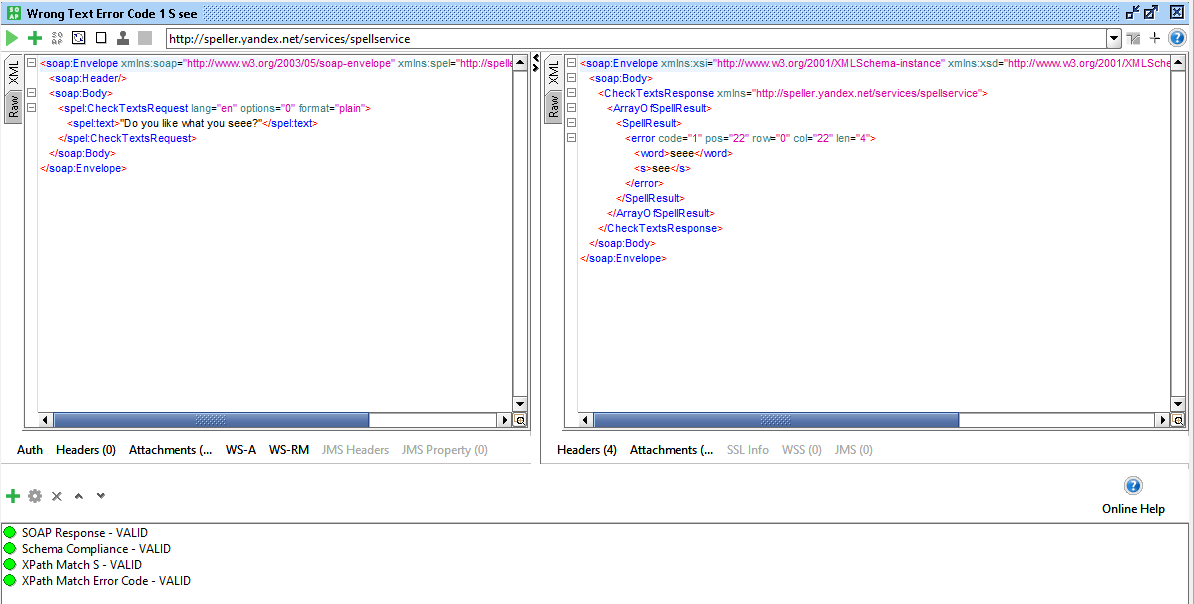
****

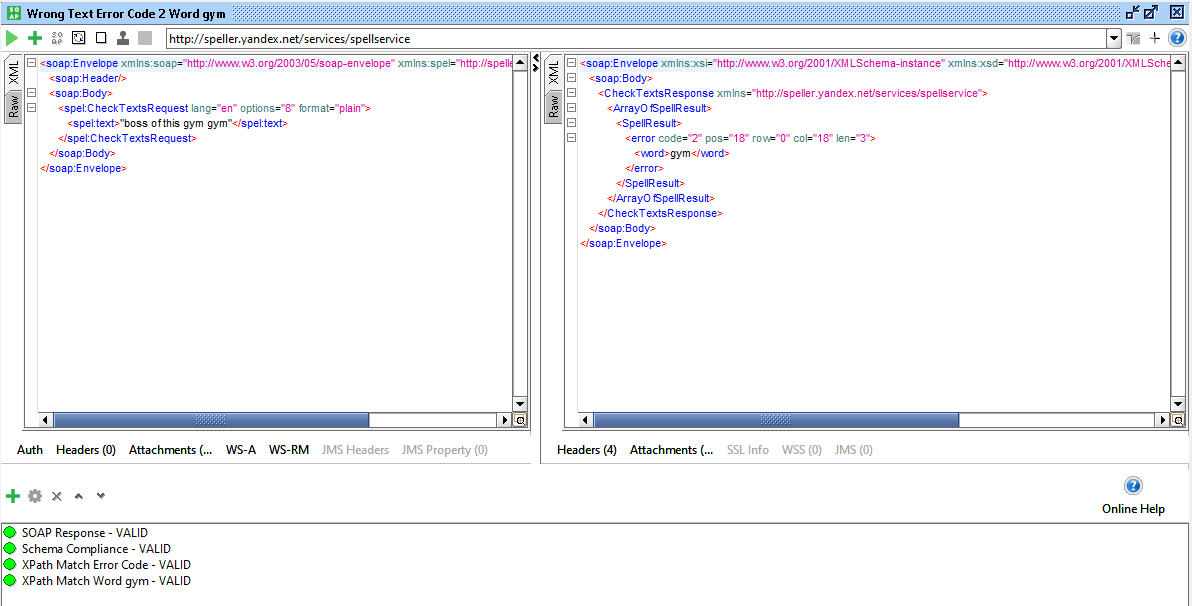
## Тестирование YaSpeller при помощи SoapUI

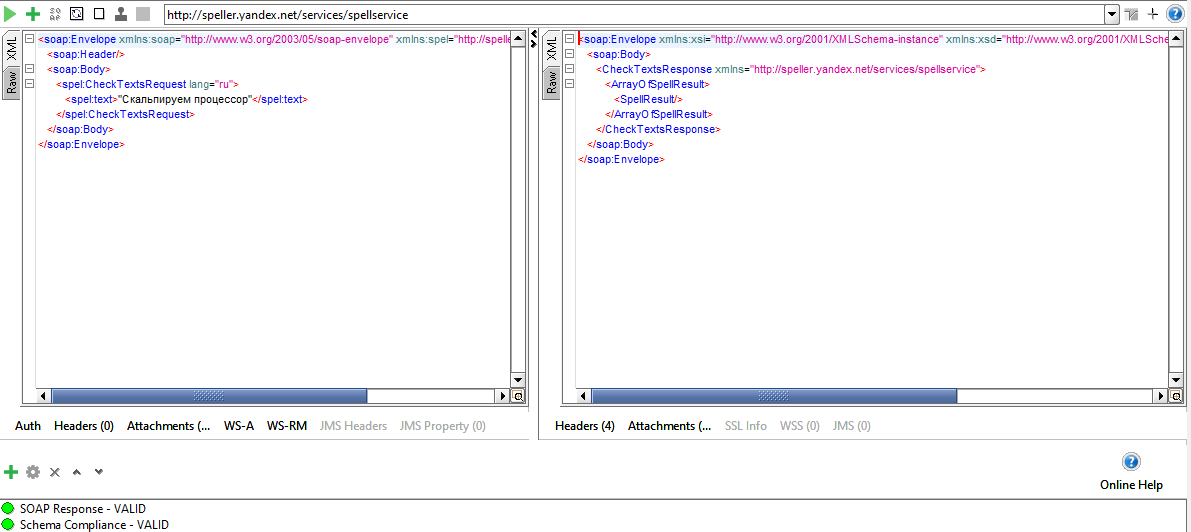
Далее на рисунках будет представлен результат практического изучения инструмента тестирования API – SoapUI.

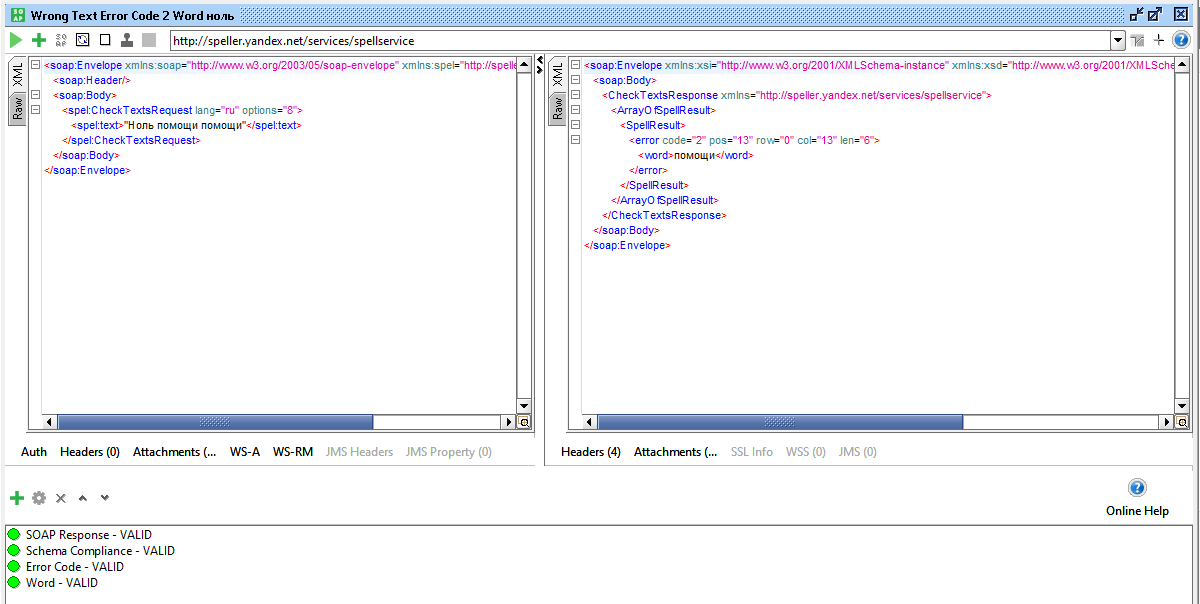


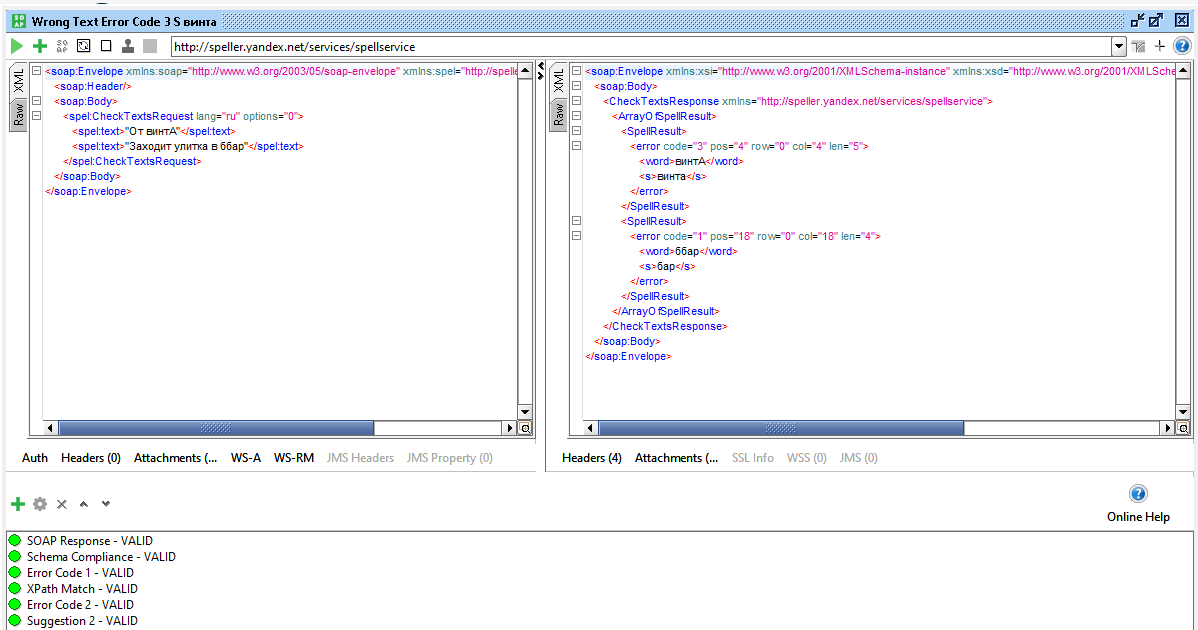












# Заключение

В ходе работы над проектом командой были получены базовые знания о тестировании мобильных и веб-приложений, а также более подробно рассмотрены стандарты написания API для веб-приложений, кроме того был оценен вклад тестирования в бизнес-процессы и изучен менеджмент тестирования. Командой был получен теоретический материал о ручном и автоматизированном тестировании. В ходе выполнения задач проекта команда обрела навыки работы с инструментами для тестирования и системами ведения учета дефектов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Святослав Куликов «Тестирование программного обеспечения. Базовый курс» от 29.09.2015 под редакцией «Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International» https://fktpm.ru/file/113-svyatoslav-kulikov-testirovanie-po-bazovyi-kurs.pdf
2. Список обучающих лекций по работе над тестирование под редакцией «Lime Lab»
3. Статья о SOAP:

https://ru.stackoverflow.com/questions/257184/%D0%A7%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-soap

1. Статья о SOAP:

<https://www.w3schools.com/xml/xpath_syntax.asp>

1. Статья о Chrome DevTools:

<https://htmlacademy.ru/blog/boost/tools/chrome-devtools-1>

1. Cтатья о Chrome DevTools:

<https://habr.com/ru/company/simbirsoft/blog/337116/>

1. Статья о HTTP/HTTPS:

https://javarush.ru/groups/posts/2521-chastjh-3-protokolih-httphttps

1. Статья о работе с Postman:

https://habr.com/ru/company/kolesa/blog/351250/y/kolesa/blog/351250/

1. Статья о связи REST с HTTP:

https://habr.com/ru/post/50147/

1. Статья о работе с Charles Proxy:

https://habr.com/ru/company/youla/blog/527648/