МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические рекомендации к лабораторным работам для студентов специальности 6-05-0612-03 «Системы управления информацией» очной и заочной форм обучения

Могилев 2026

УДК 004.415.2
ББК 32.973.202-018.2
Т36

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «16» декабря 2025 г., протокол № 5

Составители: д-р техн. наук, доц. А. И. Якимов;
нач. отдела ЗАО «Итранзишэн» И. А. Ковальчук

Рецензент канд. техн. наук, доц. С. К. Крутолевич

В методических рекомендациях к лабораторным работам по дисциплине «Тестирование программного обеспечения» приведены задания, контрольные вопросы и список литературы для самостоятельной подготовки.

Учебное издание

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Ответственный за выпуск	А. И. Якимов
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Цель преподавания дисциплины «Тестирование программного обеспечения» – познакомить с концепциями и принципами тестирования программного обеспечения; изучить различные методы и подходы к тестированию, такие как функциональное, регрессионное и нагрузочное тестирование; овладеть навыками написания тест-кейсов, тест-планов и других необходимых документов; освоить на практике современные инструменты автоматизации тестирования и управления тестированием; научить проводить анализ дефектов и оценивать качество программного продукта; сформировать способность к критическому мышлению и аналитическому подходу в процессе тестирования.

Цель методических рекомендаций – помочь студентам в самостоятельной подготовке и выполнении заданий к лабораторным занятиям по дисциплине.

Порядок выполнения работы.

1 Изучить теоретические сведения.

2 Получить задание у преподавателя, выполнить типовые задания.

3 Сделать выводы по результатам исследований.

4 Оформить отчет.

Содержание отчета.

1 Цель работы.

2 Постановка задачи.

3 Результаты исследования.

4 Выводы.

1 Лабораторная работа № 1. Основы тестирования и классификация ошибок. Составление спецификации требований

Цель работы: ознакомить студентов с основами тестирования программы; изучить классификацию ошибок и составление отчётов по ошибкам.

1.1 Основные теоретические положения

Процесс тестирования программного обеспечения можно отчасти назвать интуитивным, но то же время в основе его лежит вполне систематизированный подход. Хорошо протестировать программу означает нечто гораздо более серьезное, чем просто «погонять» ее несколько минут, чтобы убедиться, что она работает. Эффективное тестирование требует тщательного анализа и строгого системного подхода.

Вместе с программой, которую следует протестировать, необходимо иметь на неё спецификацию (описание).

Пример спецификации программы

На вход программа принимает два параметра: *x* – число, *n* – степень. Результат вычисления выводится на консоль.

Значения числа и степени должны быть целыми.

Значения числа, возводимого в степень, должны лежать в диапазоне [0, …, 999].

Значения степени должны лежать в диапазоне [1, ..., 100].

Если числа, подаваемые на вход, лежат за пределами указанных диапазонов, то должно выдаваться сообщение об ошибке.

Рассмотрим для демонстрации процесса тестирования программу, спецификация которой приведена ниже:

«*Назначение программы*: *сложить два введённых вами числа. В каждом из чисел должна быть одна или две цифры. Программа выполняет эхо-отображение вводимых чисел, а затем выводит их сумму. Ввод каждого числа завершается нажатием клавиши <Enter>. Запускается программа с помощью команды ADDER*».

Первый цикл тестирования.

Шаг 1. Начало тестирования с простого и наиболее очевидного теста.

Для начала с программой нужно познакомиться и посмотреть, достаточно ли она стабильна, чтобы её можно было тестировать. В программах, предоставленных для первого формального тестирования, часто сразу же происходит сбой. В первом тесте складываются числа 2 и 3. Последовательность действий и результаты приведены в таблице 1.1.

На рисунке 1.1 видно, как выглядит экран после окончания теста. Курсор указывает, где будет отображаться следующее вводимое число.

Отчёт о проблемах, выявленных первым тестом:

«*Программа работает – она приняла числа 2 и 3 и вернула 5*». Но проблемы всё же есть. Для их описания составляется отчёт, форма которого представлена

в приложении А.

Таблица 1.1 – Порядок действия и результаты сложения чисел 2 и 3

Ввод	Вывод
Вводите ADDER и нажимаете клавишу **<*Enter*>**	Экран мигает. Вверху экрана появляется знак вопроса
Нажимаете 2	За знаком вопроса появляется цифра 2
Нажимаете <*Enter*>	В следующей строке появляется знак вопроса
Нажимаете 3	За вторым знаком вопроса появляется цифра 3
Нажимаете <*Enter*>	В третьей строке появляется цифра 5. На несколько строк ниже появляется еще один знак вопроса

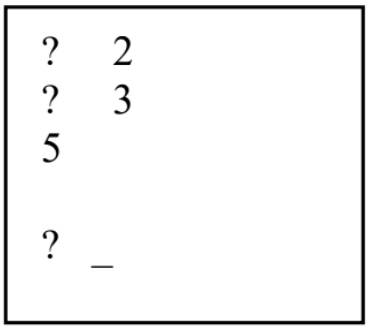

Рисунок 1.1 – Вид экрана после окончания теста

Типы ошибок.

1 *Ошибка проектирования.* Нет никаких указаний на то, с какой программой работаете.

2 *Ошибка проектирования.* На экране нет никаких инструкций. Откуда знать, что нужно делать? Что, если вводите недопустимые числа? Отобразить инструкцию на экране не трудно, и она всегда будет перед глазами, в то время как печатная документация может потеряться.

3 *Ошибка проектирования.* Как остановить программу? Эта инструкция тоже должна быть на экране.

4 *Ошибка кодирования.* Сумма (число 5) выведена в стороне от слагаемых.

Примечание – Обязательно представляется отдельный «*Отчёт о проблеме*» по каждой ошибке. Описание всех четырех ошибок можно было бы поместить в один отчёт, но лучше этого не делать. Ошибки могут исправляться в разное время и сведения о тех из них, которые остались неисправленными, могут просто потеряться. Если программист захочет их сгруппировать, он сам рассортирует отчеты. Чтобы привлечь внимание к взаимосвязанным проблемам, просто поместите в отчеты соответствующие ссылки.

Шаг 2. Составление заметок о том, что ещё должно быть протестировано.

Выполнив первые, и самые очевидные, тесты, следует подумать о том, что ещё следует протестировать. Свои соображения нужно записать: одни из записей примут форму заметок, другие же могут представлять собой достаточно строго

формализованные описания серий тестов. Такие документированные группы тестов в дальнейшем могут послужить для проверки следующих версий программы. Примером может быть серия тестов, представленная в таблице 1.2, в которой для тестирования программы предлагается девять примеров. Тесты подобраны так, чтобы каждая цифра встречалась в них хотя бы один раз и было по одной комбинации чисел на каждую из вероятных проблем. А чтобы определить, на каких значениях вероятнее возникнут проблемы, эффективнее всего проверить граничные условия.

Таблица 1.2 – Серия тестов для программы сложения двух чисел

Входные данные	Ожидаемый результат	Замечание
99 + 99	198	Пара наибольших чисел, которые может складывать программа
-99 + -99	-198	В документации не сказано, что нельзя складывать отрицательные числа
99 + -14	85	Большое первое число может повлиять на интерпретацию программой второго числа
-38 + 99	61	Сложение отрицательного числа с положительным числом
5 + 99	104	Проверка на то, влияет ли слишком большое второе число на интерпретацию первого
9 + 9	18	Наибольшим числом из одной цифры является 9
0 + 0	0	Программы часто дают сбои при вводе нулей
0 + 23	23	Программа может особым образом обрабатывать 0, поэтому его нужно проверить в виде первого слагаемого
-78 + 0	78	Программа может особым образом обрабатывать 0, поэтому его нужно проверить в виде второго слагаемого

Классом можно назвать группу значений, которые программа обрабатывает одним и тем же способом. Ограниченными значениями класса являются те входные данные, на которых программа меняет своё поведение. *Работа программиста* – проверять те практические точки, которые можно определить по листингу. *Задача тестировщика* – проанализировать программу с другой точки зрения, чтобы выявить те критические точки, которые программист пропустил. Поэтому классы возможных тестов следует выделять, исходя, прежде всего, из внешнего поведения программы. В результате набор тестов будет отличаться от того, который можно составить по листингу программы (именно в этом суть задачи тестировщика).

Важным моментом здесь является то, что границу значений обязательно нужно протестировать с двух сторон.

Шаг 3. Проверка допустимых значений и наблюдение за реакцией программы.

В таблице 1.2 приведены только допустимые значения входных данных программы. На следующем этапе тестирования можно создать такую же серию тестов для недопустимых значений. Ещё одна серия тестов может быть предназначена для проверки редактирования чисел: вводите значение, затем изменяете его

и только после этого нажимаете <*Enter*>.

Шаг 4. Немного тестирования в режиме «свободного полёта».

Всегда записывайте, что делаете и что происходит во время исследовательских тестов.

От формальных тестов следует перейти к неформальным тестам. В таблице 1.3 видно, что при малейшей провокации программа работает неверно (она «зависает»). Это приводит к затрате большего времени на перезапуск компьютера, нежели чем на тестирование.

Таблица 1.3 – Неформальные тесты для программы сложения двух чисел

Тест	Особенность теста	Замечание
100 + 100	Граничное условие числа больше максимального допустимого значения (99)	Программа приняла 10. Когда ввели второй 0, чтобы получилось 100, программа повела себя так, как будто вы нажали <Enter >. Также было и со вторым числом 100. В результате по окончании теста на экране было следующее: ? 10 ? 10 20
<*Enter*> + <*Enter*>	Проверка реакции программы на отсутствие числовых данных	Когда нажали <*Enter*>, программа напечатала 10 – последнее введенное вами число. То же было и после второго нажатия <*Enter*>, и в качестве суммы программа напечатала 20
123456 + 0	Ввод более длинных чисел	Программа приняла первые две цифры и проигнорировала остальные так же, как было с числом 100. В будущей версии программа будет принимать большие числа.
1,2 + 5	Числа с десятичной частью	Реакция на десятичный разделитель такая же, как и на <*Enter*>
A + *b*	Недопустимые символы	Когда нажали <*Enter*>, после <*A*> программа «зависла». Для продолжения тестирования приходится перезапускать компьютер
<*Ctrl* + *A*>, <*Ctrl* + *B*>, <*Ctrl* + *C*>, <*Ctrl* + *D*>	Управляющие символы и функциональные клавиши часто являются источниками проблем	Для всех комбинаций клавиш, кроме <*Ctrl* + *C*>, программа отобразила графические символы. Затем, после нажатия <Enter>, «зависла». Нажатие <*Ctrl* + *C*> привело к завершению программы и выходу в операционную систему

Столкнувшись с очередной проблемой, составляете о ней отчет. О сданных отчётах лучше написать для себя итоговые заметки. На этом тестирование первой версии программы можно считать завершенным.

Шаг 5. Подведение итогов о программе и выясненных недостатках.

Эта работа не всегда необходима, но часто оказывается очень полезной. До этого времени все было сконцентрировано на конкретных деталях – анализиро-

вали допустимые данные, продумывали граничные условия и составляли тестовые примеры. В будущем больше времени будет тратиться на выполнение уже подготовленных тестов, чем на придумывание новых.

Итоги первого цикла тестирования.

Начали с простейшего из возможных тестов. Поскольку программа его прошла, разработали серию формальных тестов, чтобы проверить, как она работает с допустимыми данными. Эти тесты будут использованы и дальше. Поскольку часть проверок программа не прошла, то на планирование дальнейших серий тестов времени можно не тратить. Вместо этого проведём несколько неформальных экспериментов и выясним, что программа вообще очень нестабильна. Записали несколько замечаний, к которым обратимся при тестировании следующей версии программы.

Если бы программа успешно прошла первую серию тестов, то разработали бы вторую, более обстоятельную. Если бы программа снова показала себя надежной, продолжали бы ее тестирование, пока не исчерпались бы идеи или отведенное время. Напоследок провели бы несколько беглых тестов, не входивших в ранее разработанные серии, и записали замечания на будущее. Завершив тестирование и всю бумажную работу, стоит потратить еще немного времени, чтобы обдумать результаты.

Практические задания

Задание 1

Дано практическое задание EP & BVA Practice Assignment в свободном доступе: https://qa-ep-bva-practice-assignment.vercel.app

EP & BVA Practice Assignment – тренажер для тестировщиков. Страница содержит простую спецификацию для методов разработки тестов разделения эквивалентности (Equivalence Separation (EP)) и анализа граничных значений (Boundary Value Analysis (BVA)), а также несколько различных реализаций этой спецификации. Реализация 1 правильная, остальные – не очень.

Спецификация:

– программа принимает любое целое число, большее или равное -10 000 и меньше или равное 10 000;

– все отрицательные целые числа должны храниться в базе данных вместе с другими отрицательными целыми числами;

– все положительные целые числа должны храниться в базе данных вместе с другими положительными целыми числами;

– ноль не должен храниться.

Ваша цель – найти значения ON, OFF, OUT, IN, которые позволят найти ошибки в каждой из них. Составить отчет по каждой реализации (рисунок 1.2).

	imp10	
test	respons	result
-10001	REJECTED: An integer must be >= -10000 and <= 10000, it is not stored.	true
-10000	ACCEPTED: Stored as a negative integer.	true
-9999	ACCEPTED: Stored as a negative integer.	true
-1	ACCEPTED: Not stored (0).	false
0	ACCEPTED: Not stored (0).	true
1	ACCEPTED: Not stored (0).	false
9999	ACCEPTED: Stored as a positive integer.	true
10000	ACCEPTED: Stored as a positive integer.	true
10001	REJECTED: An integer must be >= -10000 and <= 10000, it is not stored.	true
aaaaa	REJECTED: The value must be an integer.	true
AAAAA	REJECTED: The value must be an integer.	true
FFFFFF	REJECTED: The value must be an integer.	true
ffffff	REJECTED: The value must be an integer.	true
!@#$%	REJECTED: The value must be an integer.	true
null	REJECTED: The value must be an integer.	true

Рисунок 1.2 – Фрагмент оформления отчета

Задание 2

1 Написание программы и спецификации требований к ней в соответствии с вариантом задания к лабораторной работе. Каждое функциональное требование должно быть описано.

2 Написание тестовых сценариев по спецификации требований; тестирование функциональных требований разработанной программы: провести тестирование программы в соответствии с шагами тестирования, представленными в теоретической части и описание найденных дефектов/недочётов/ошибок.

Проект может быть написан на любом языке программирования, должен являться прикладной программой (реализованной с помощью интерфейса, input/output файлов, или связанной с базой данных), также это может быть сайтом и т. д.

Тестирование включает:

– написание тестовых сценариев для проверки функциональности;

– описание найденных ошибок в отчетах;

– исправление ошибок в программе и тестирование её вновь.

Варианты задания 2

1 Нахождение наибольшего общего делителя при условии, что начальные значения $x1$ и $x2$ положительны.

2 Вычисление факториала неотрицательного целого числа.

3 Вычисление чисел Фибоначчи.

4 Выполнение конкатенации (сцепления) строк.

5 Определить частное q и остаток r от деления x на y.

6 Линейный поиск в упорядоченном по возрастанию элементов массиве $A[1..n]$. Определить порядковый номер m некоторого значения x.

7 Поиск значения *x* в двумерном массиве.

8 Написать программу, которая по данному фиксированному массиву $B[0...n-1]$, где $n > 0$, записывает в *d* число нечётных значений в $B[0...n-1]$.

9 Суммирование элементов массива $B[0...n-1]$, где $n > 0$.

10 Удаление лишних пробелов в тексте.

11 Суммирование комплексных чисел.

12 Вводится строка слов. Вывести слова в обратном порядке.

13 В одномерном массиве найти максимальный из отрицательных элементов и поменять его местами с последним элементом массива.

14 Составить программу печати прямоугольного треугольника из звездочек

```
*
**
***
****
*****
```

используя цикл for. Введите переменную, значением которой является размер катета треугольника.

15 Написать программу, подсчитывающую число символов, поступающих со стандартного ввода.

16 Составить программу перекодировки вводимых символов со стандартного ввода по следующему правилу:

a -> *b*
b -> *c*
c -> *d*
...
z -> *a*
другой символ -> *

Коды строчных латинских букв расположены подряд по возрастанию.

17 Написать программу, печатающую квадраты и кубы целых чисел.

18 Написать программу, печатающую сумму квадратов первых *n* целых чисел.

19 Написать программу, которая переводит секунды в дни, часы, минуты и секунды.

20 Написать программу, переводящую скорость из километров в час в метры в секундах.

Содержание отчета.

1 Постановка задачи.

2 Составление спецификации требований.

3 Алгоритм программы.

4 Шаги тестирования.

5 Результаты работы программы.

6 Приведение классификации найденных ошибок.

7 Приложение. Листинг программы.

Контрольные вопросы

1 Что вы понимаете под тестированием и верификацией программных продуктов?

2 Какие типы ошибок существуют?

3 Что такое классы тестов и для чего они предназначены?

4 На какие шаги можно разбить первый цикл тестирования любой программы?

5 Для чего нужен второй цикл тестирования?

6 Что такое тестирование и какие цели оно преследует?

7 Какие основные виды ошибок могут возникать в процессе разработки программного обеспечения?

8 Чем отличается функциональное тестирование от нефункционального?

9 Каким образом составляется спецификация требований к программному продукту?

10 Какие этапы включает процесс интуитивного тестирования?

11 Какие методы интуитивного тестирования существуют?

12 Как классифицируются ошибки по их происхождению?

13 Какие критерии используются для оценки качества программного продукта?

14 Как оценивается эффективность тестирования?

15 Какие инструменты могут использоваться для поддержки процесса тестирования?

2 Лабораторная работа № 2. Проектирование тест-кейсов

Цель работы: приобрести практические навыки создания тестов и тест-кейсов; научиться создавать тест-кейсы для приложений.

2.1 Основные теоретические положения

1 **Smoke testing** (встречаются названия intake test, build verification test) – тестирование, проводимое на начальном этапе (например, после нового билда) и в первую очередь направленное на проверку готовности разработанного продукта к проведению более расширенного тестирования, определения общего состояния качества продукта.

Это короткий цикл тестов, подтверждающий (отрицающий) факт того, что приложение стартует и выполняет свои основные функции. Данный тип тестирования позволяет на начальном этапе выявить основные быстро находимые критические дефекты. Исходя из того, что данные проверки практически всегда одинаковы и редко претерпевают изменениям, целесообразно будет их автоматизировать.

Стоит понимать, что данный тип тестирования является видом тестирования продукта по глубине, а не просто видом тестовых испытаний. Как говорилось

выше, данный тип тестирования определяет, пригоден ли продукт для дальнейшего, более полного тестирования.

В случае, если он не проходит smoke testing – продукт необходимо отправить на доработку. Обязательно необходимо записывать результаты прохождения теста. Это необходимо для того, чтобы сохранить записи того, что работает, а что нет. Можно разделить результаты на пройденные и проваленные (достаточно таблицы Excel).

2 **Чек-лист для критического пути (**critical path test**)** – основной тип тестовых испытаний, во время которого значимые элементы и функции приложения проверяются на предмет правильности работы при стандартном их использовании.

Чаще всего на практике на данном уровне тестирования проверяется основная масса требований к продукту. Пример: возможность набора текста, вставки картинок, возможность войти на сайт, создать запись и т. д.

Для данного вида тестирования пишутся наиболее подробные и глубокие тест-кейсы, чтобы покрыть всю возможную функциональность приложения. Тест критического пути может быть как позитивным, так и негативным.

Позитивный тест критического пути – это проверка работоспособности функций программного продукта, с которыми пользователь сталкивается ежедневно.

Негативный тест критического пути – это проверка всевозможных вариантов нестандартного использования функциональности, используемой пользователем каждый день.

Тест критического пути является одним из самых распространенных видов функционального тестирования, позволяет выявить самые быстро находимые дефекты и исправить приложение в более сжатые сроки.

Возможный пример оформления приведен на рисунке 2.1.

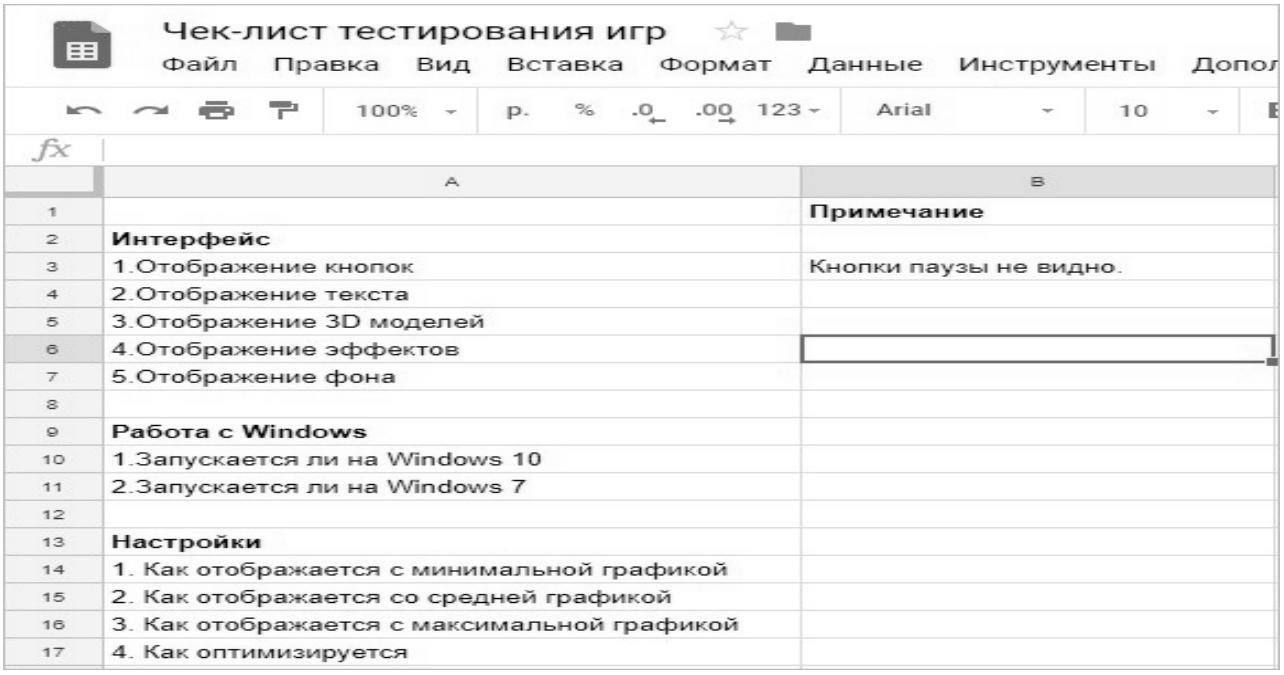

Рисунок 2.1 – Оформление чек-листа

3 **Тест-кейс** – это профессиональная документация тестировщика, последовательность действий, направленная на проверку какого-либо функционала, описывающая как прийти к фактическому результату.

Набор тест-кейсов называют тест-комплектом. Иногда тест-набор путают с тест-планом. Тест-план описывает какие работы, как и когда должны быть проведены в рамках тестирования продукта, а также, что необходимо для их выполнения.

Любой тест-кейс обязательно включает в себя:

– уникальный идентификатор тест-кейса – необходим для удобной организации хранения и навигации по нашим тест-наборам;

– название – основная тема, или идея тест-кейса. Краткое описание его сути;

– предусловия – описание условий, которые не имеют прямого отношения к проверяемому функционалу, но должны быть выполнены. Например, оставить комментарий на вашем портале может только зарегистрированный пользователь. Значит для тест-кейса «Создание комментария» будет необходимо выполнение предусловия «пользователь зарегистрирован» и «пользователь авторизован»;

– шаги – описание последовательности действий, которая должна привести нас к ожидаемому результату;

– ожидаемый результат – результат, что мы ожидаем увидеть после выполнения шагов.

Чего не должно быть в тест-кейсе:

1) зависимостей от других тест-кейсов;

2) нечеткой формулировки шагов или ожидаемого результата;

3) отсутствия необходимой для прохождения тест-кейса информации;

4) излишней детализации.

Возможный план по разработке тест-кейсов приведен на рисунке 2.2.

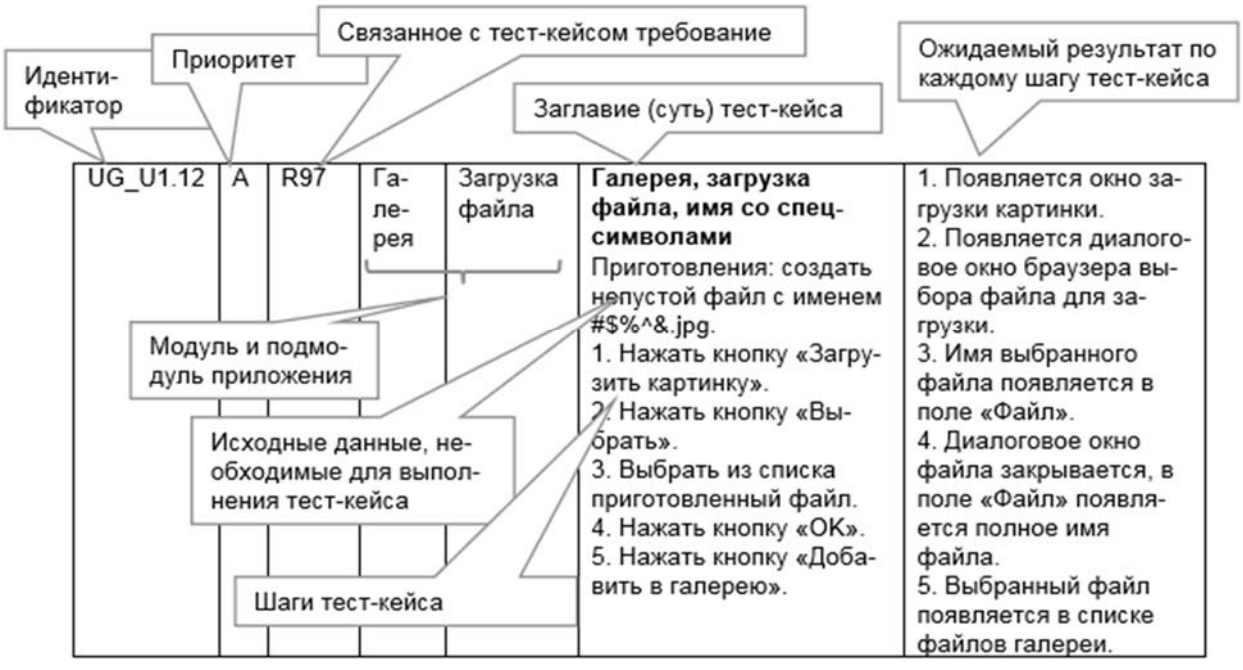

Рисунок 2.2 – Элементы тест-кейсов

Практические задания

Задание 1

На основе представленного ниже набора требований сформируйте для разрабатываемых приложений (на любом языке программирования):

– смоук-тест (оформить в виде таблицы Excel);

– чек-лист для теста критического пути (оформить в виде таблицы Excel);

– оформить тест-кейс.

Вариант 1. Ваш проект «Реализация онлайн почтового клиента». Нужно описать тест-кейсы на функционал: Отправка сообщения. Путь к форме отправки сообщения: нажатие кнопки «Новое сообщение» на панели инструментов. Дизайн с комментариями прилагается (рисунок 2.3).

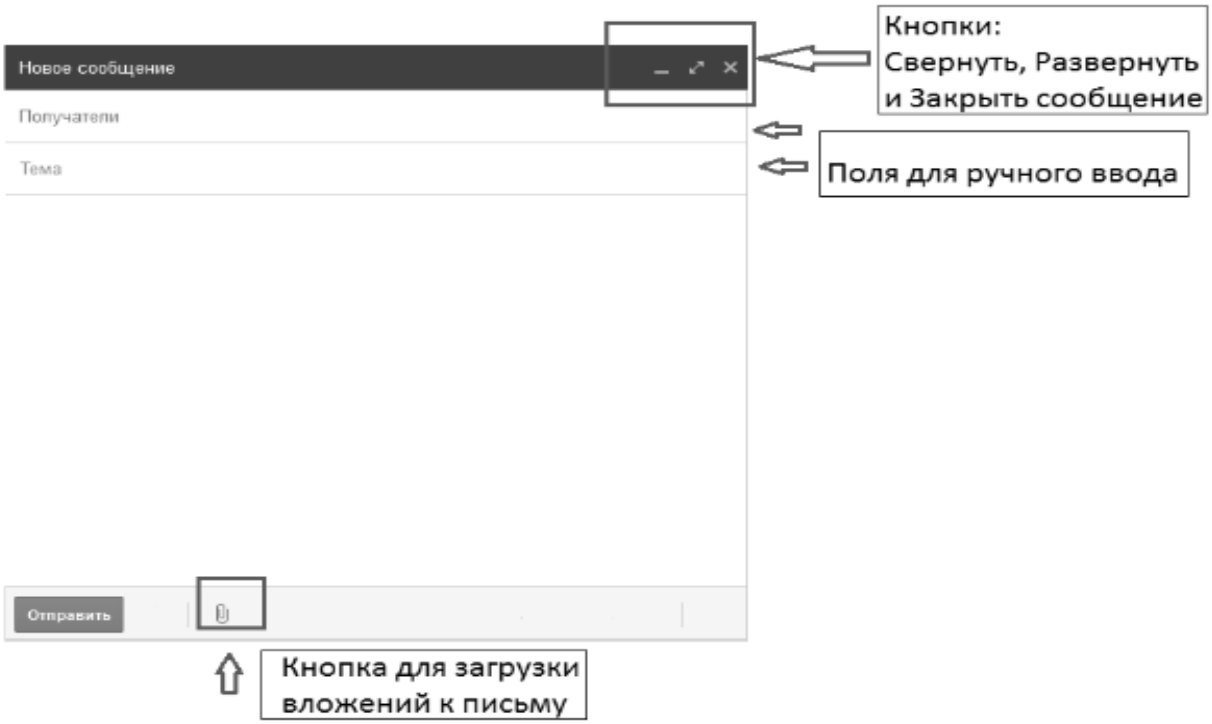

Рисунок 2.3 – Реализация онлайн почтового клиента

Вариант 2. Ваш проект «Реализация системы библиотека». Нужно описать тест-кейсы на функционал «Регистрация пользователя». Путь к форме регистрации: главная страничка сайта, кнопка «Регистрация».

Вариант 3. Ваш проект «Реализация системы поиска работы и сотрудников». Необходимо написать тест-кейсы на функционал «Поиск вакансии». Путь к форме поиска: главная страничка – кнопка «Поиск» на панели инструментов. В каждом выпадающем списке соответственно список городов Беларуси, разделы сайта и зарплаты от 0 до 1 000, от 1 000 до 2 000, 2 000 и выше.

Вариант 4. Требования к разрабатываемому приложению.

Приложение должно выполнять математические вычисления.

1 Приложение должно работать под всеми версиями ОС Windows.

2 Приложение должно быть максимально похоже на стандартный калькулятор Windows (рисунок 2.4) за исключением некоторых особенностей.

3 Несколько приложений должны иметь возможность работать одновременно.

4 При запуске приложения должно отображаться окно со стандартными для калькулятора кнопками и полем ввода и отображения данных.

Рисунок 2.4 – Стандартный калькулятор Windows

5 Для начала вычислений пользователь должен нажать кнопку «Начать».

6 Приложение должно позволять легко сохранять вычисления в выбранном пользователем формате.

7 Опционально предусматривается поддержка нескольких языков.

8 Приложение должно позволять выполнять вычисления сразу же после запуска.

9 Скорость вычислений должна быть максимально высокой.

10 Приложение должно позволять выполнять следующие операции: сложение, умножение, вычитание и деление чисел.

11 Приложение должно позволять строить графики простых функций.

12 Приложение должно запрашивать подтверждение («Результат не сохранён. Выйти?») в случае, если пользователь не сохранил результаты работы.

Вариант 5. Требования к разрабатываемому приложению.

Приложение должно выполнять математические вычисления.

1 Приложение должно работать под версиями ОС Windows10 и Windows11.

2 Несколько приложений должны иметь возможность работать одновременно, т. е. можно открыть несколько калькуляторов и вести в них невзаимосвязанные вычисления.

3 При запуске приложения должно отображаться окно с кнопками калькулятора (см. рисунок 2.4) и полем отображения данных.

4 Данные в приложение могут вводиться как с помощью кнопок приложения, так и с помощью клавиатуры.

5 Приложение должно позволять сохранять вычисления во внешний файл с расширением, задаваемым пользователем.

6 Должна быть предусмотрена поддержка английского и русского языков. Отображается тот язык, который выбран в ОС по умолчанию.

7 Вычисления должны производиться со скоростью не более 1 с.

8 Приложение должно позволять выполнять следующие операции: сложение, умножение, вычитание и деление чисел, взятие квадратного корня, возведение в степень, вычисление процентов, ввод отрицательного числа.

Контрольные вопросы

1 В какой последовательности рекомендуется разрабатывать тесты?

2 Что такое смоук-тест? Приведите пример.

3 Что такое чек-лист?

4 Перечислите элементы тест-кейса.

5 Оба тест-кейса делают одну и ту же проверку. Какой из них лучше (рисунок 2.5)?

6 Обязательно ли описывать ожидаемый результат в тест-кейсе и в чек-листе?

1.В поле A ввести 10 2.В поле B ввести 15 3.Нажать кнопку «Сложить» 4.Проверить значение в поле C	4. Значение в поле C равно 25

1.Проверить, что программа суммирует два числа корректно	4. Суммирует корректно

Рисунок 2.5 – Варианты тест-кейса

3 Лабораторная работа № 3. Тестирование веб-приложений

Цель работы: тестирование веб-приложения, составление чек-листа для тестирования веб-приложения.

3.1 Основные теоретические положения

Функциональное тестирование.

В данном пункте важно убедиться, что наш продукт соответствует нужной функциональной спецификации, упомянутой в документации по разработке. Проверке подлежат следующие функции.

1 Тестирование форм.

1.1 Регистрация (пользователь с данными существует в системе; пользователь с данными не существует в системе; пользователь, заблокированный в системе, не может пройти повторную регистрацию).

1.2 Авторизация (пользователь существует в системе с введенным логином и паролем; пользователь с введенным логином не существует в системе; пользователь с введенным логином существует в системе, но пароль неверный; пользователь с введенным логином и паролем существует в системе, но заблокирован модерацией (страница заморожена); валидация полей ввода.

1.3 Тестирование валидации всех обязательных полей (максимальная и минимальная длина; диапазон допустимых символов, спецсимволы; обязательность к заполнению; убедитесь, что астериск (знак звездочки) отображается у всех обязательных полей; убедитесь, что система не отображает окно ошибки при незаполненных необязательных полях.

1.4 Формы обратной связи.

1.5 Ссылки на пользовательские соглашения.

2 Поиск.

2.1 Результаты существуют/не существуют.

2.2 Корректное сообщение о пустом результате.

2.3 Пустой поисковой запрос.

2.4 Поиск по эмодзи.

3 Поля.

3.1 Числовые поля: они не должны принимать буквы, в этом случае должно отображаться соответствующее сообщение об ошибке.

3.2 Дробные значения, например, как система валидирует 1.1 и 1,1.

3.3 Отрицательные значения в числовых полях, если они разрешены.

3.4 Деление на ноль корректно обрабатывается.

3.5 Протестируйте максимальную длину каждого поля, чтобы убедиться, что данные не обрезаются или скрываются под многоточие.

3.6 Протестируйте все поля ввода на спецсимволы.

3.7 Проверить что текст не выезжает за границы поля.

4 Всплывающие сообщения.

4.1 Протестируйте всплывающие сообщения («Это поле ограничено N знаками»).

4.2 Подтверждающие сообщения отображаются для операций обновления и удаления.

4.3 Сообщения об ошибках ввода.

5 Фильтры.

5.1 Протестируйте функциональность сортировки (по возрастанию, по убыванию, по новизне).

5.2 Задать фильтры с выдачей.

5.3 Задать фильтры, по которым нет выдачи.

5.4 Фильтры по категориям/подкатегориям.

5.5 Фильтры с радиусом поиска.

5.6 Данные в выпадающих списках.

6 Протестируйте функциональность доступных кнопок.

7 Наличие favicon.

8 Проверка обработки различных ошибок (страница не найдена, тайм-аут, ошибка сервера и т. д.).

9 Протестируйте, что все загруженные документы правильно открываются.

10 Пользователь может скачать/прикрепить/загрузить файлы/медиа (картинки, видео и т. д.). А также удалить эти файлы из вложений. Убедитесь, что файлы уходят на сервер только после нажатия соответствующей кнопки.

11 Протестируйте почтовую функциональность системы.

12 Кэш, cookie и сессии (пользователь очистил кэш браузера; посмотрите, что будет, если пользователь удалит куки, находясь на сайте; посмотрите, что будет, если пользователь удалит куки после посещения сайта).

13 DevTools (ошибки в Console; все стили загружаются; картинки загружаются).

Инструменты разработчика (от англ. development tools или сокращённо **DevTools**) – это программы, позволяющие создавать, тестировать и отлаживать (debug) программное обеспечение. Открыть их можно следующими способами:

– клавишей *F*12 (на ноутбуках *fn* + *f*12);

– *ctrl+shift+j* (в ОС *Windows*);

– кликнуть на странице правой кнопкой мыши и выбрать пункт *Inspect* (если язык браузера английский) или «Посмотреть код» (если язык браузера русский).

Современные браузеры (например, Safari, Firefox, Microsoft Edge, Chrome, Яндекс и др.) имеют встроенные инструменты разработчика, позволяющие просмотреть исходный код сайта. Отдельно устанавливать их не требуется. С их помощью можно просматривать и отлаживать HTML сайта, его CSS и Javascript. Также можно проверить сетевой трафик, потребляемый сайтом, его быстродействие и много других параметров.

Вкладка Elements.

Вкладка Elements содержит две кнопки.

1 Выбор элемента с помощью курсора.

2 Переключение в режим выбора устройств.

Вкладка пригодится для тестирования верстки. Также тут можно посмотреть размеры того или иного объекта на экране, его цвет и шрифт текста.

При выборе любого DOM элемента на вкладке Styles будут отображаться все CSS-правила, применяемые к нему, в том числе и неактивные. Все правила разбиты на блоки и упорядочены по убыванию специфичности селектора. Можно налету менять значения, деактивировать и дописывать новые правила и смотреть как это влияет на отображение элемента.

Как посмотреть шрифт, цвет или размер любого элемента на странице?

1 Открыть вкладку Elements в DevTools.

2 Нажать на курсор выбора элемента в левом верхнем углу.

3 Навести на любой элемент и откроется окно с информацией о нем.

Вкладка Console.

Вкладка Console необходима для логирования диагностической информации в процессе разработки или взаимодействия с JavaScript на странице. Все

ошибки в JavaScript коде будут выводиться здесь с указанием файла и конкретного места в нем, где произошла ошибка (речь идет об ошибках на фронте).

Боковая панель сообщений.

1 **Errors** (ошибки) – этот уровень используется для записи ошибок и проблем, которые могут привести к некорректной работе приложения. Логи с уровнем ERROR указывают на проблемы, которые требуют вмешательства и исправления.

2 **Warnings** (предупреждения) – этот уровень указывает на предупреждения и потенциальные проблемы, которые не являются критическими ошибками. Логи с уровнем WARN могут включать сообщения о неправильном использовании приложения, некорректных данных или других ситуациях, требующих внимания.

3 **Info** – этот уровень предоставляет информацию о ходе работы приложения и важных событиях. Логи с уровнем INFO содержат сообщения, которые помогают отслеживать основные операции и состояние приложения. Например, они могут сообщать о начале и окончании определенных операций, загрузке ресурсов, отправке и получении запросов, изменении состояния приложения и других событиях, которые могут быть полезны для отслеживания хода выполнения программ.

4 **Filter** – для фильтрации поиска.

Вкладка Network.

Вкладка Network позволяет мониторить процесс загрузки страницы и всех файлов, которые подгружаются при этом. Ее удобно использовать для оптимизации загрузки страниц и мониторинга запросов.

1 Stop Recording Network – останавливает запросы и ответы.

2 Clear Network – нажав на нее, можно подчистить историю запросов.

3 Filter – активирует панель фильтров.

4 Preserve log – не очищает данные при перезагрузке страницы.

5 Disable cache – отключает кэш браузера (работает, пока открыт DevTools).

6 Throttling – можно регулировать скорость интернета.

7 Fetch/XHR – покажет только запросы к бэкенду.

8 Настройки вкладки Network.

Вкладка Performance.

Вкладка Performance отображает тайм-лайн использования сети, выполнения JavaScript-кода и загрузки памяти. Она применяется для улучшения производительности работы страницы в целом.

На вкладке Performance после первоначального построения графиков будут доступны данные о выполнении кода и жизненном цикле страницы. Пользователь может выбрать отдельный промежуток на временной шкале и увидеть, какие процессы происходили в этот момент.

Вкладка Application.

Вкладка для инспектирования и очистки всех загруженных ресурсов, включая IndexedDB или Web SQL базы данных, local и session storage, куков, кэша приложения, изображений, шрифтов и таблиц стилей.

1 Storage – хранилище данных (в данном случае кэша).

2 Local Storage – хранилище для данных на постоянной основе. Данные из Local Storage автоматически не удаляются и не имеют срока действия. Эти данные не передаются на сервер в запросе HTTP. Кроме того, объем Local Storage для домена составляет в Chrome и Firefox 5 Мб.

3 Session Storage – временное хранилище информации, которая удаляется после закрытия браузера.

4 Cookies – тут хранятся все куки, можно их изменить или удалить.

5 Cache storage – тут можно увидеть весь кэш, который хранится в браузере.

Вкладка Lighthouse.

Lighthouse – это инструмент от компании Google, с помощью которого можно проверить производительность сайта. Он не только тестирует сайт и показывает оценку производительности, но и даёт конкретные рекомендации: что можно улучшить, чтобы сделать сайт быстрее.

1 Radiobatton Device – можно выбрать, какую версию страницы будем проверять, мобильную или десктоп.

2 Check box Categories – можно выбрать категории для анализа. Например, производительность, доступность и т. д.

3 Analyse page load – начать анализ страницы (занимает 10…20 с).

Тестирование безопасности.

Данная проверка нацелена на поиск недостатков и пробелов с точки зрения безопасности нашего приложения.

Что проверяем?

1 Пользователь не может авторизоваться под старым паролем, заблочен в сервисе, достиг лимита авторизаций, ввел чужой код верификации.

2 Страницы, содержащие важные данные (пароль, номер кредитной карты и CVC, ответы на секретные вопросы и т. п.), открываются через HTTPS.

3 Пароль скрыт астерисками на страницах регистрация, «забыли пароль», «смена пароля».

4 Корректное отображение сообщений об ошибках.

5 Завершение сессии после разлогина.

6 Доступ к закрытым разделам сайта.

7 SQL-инъекции.

8 Cross-Site Scripting уязвимости.

9 HTML-инъекции.

10 Cookie должны храниться в зашифрованном виде.

11 Роли пользователей и доступ к контенту.

Тестирование локализации и интернационализации.

Тестирование интернационализации веб-приложения включает тестирование приложения для различных местоположений, форматов дат, чисел и валют. Тестирование локализации включает тестирование веб-приложения с локализованными строками, изображениями и рабочими процессами для определенного региона.

Что проверяем?

1 Дата и время. Например, отображение времени, даты в соответствии с часовым поясом пользователя.

2 Смена языка и проверка перевода всех элементов веб-приложения исходя из выбранного языка.

3 Выбор номера телефона с разными кодами стран.

4 Определение местоположения пользователя и отображение соответствующего пермишена.

5 Отображение соответствующих символов валюты.

Тестирование удобства использования.

Тестирование удобства использования подразумевает проверку навигации, контента и другой информации для пользователя.

Что проверяем?

1 Отсутствие орфографических и грамматических ошибок, все страницы имеют корректные заголовки.

2 Выравнивание картинок, шрифтов, текстов.

3 Информативные ошибки, подсказки.

4 Подсказки существуют для всех полей.

5 Отступы между полями, колонками, рядами и сообщениями об ошибках.

6 Кнопки имеют стандартный размер, цвет.

7 На сайте нет битых ссылок и изображений.

8 Неактивные поля отображаются серым цветом.

9 Проверьте сайт при разных разрешениях экрана.

10 Скролл должен появляться только тогда, когда он требуется.

11 Отображение чек-боксов и радиокнопок. Кнопки должны быть доступны с клавиатуры и пользователь должен быть в состоянии пользоваться сайтом, используя только клавиатуру.

12 Отображение выпадающих списков.

13 Длинный текст скрывается под многоточием.

14 Корректный выбор даты.

15 Наличие плейсхолдеров в полях.

16 Логотип ведет на главную страницу сайта.

17 Переходы и навигация между страницами и разделами меню.

Кросс-платформенное тестирование.

Кросс-платформенное тестирование проводится, чтобы убедиться, что ваше приложение совместимо с другими браузерами, различными оболочками, аппаратным обеспечением устройства.

Что проверяем?

1 Тестирование в различных браузерах (Firefox, Chrome, Safari – это минимальный набор): анимация, верстка, шрифты, уведомления и т. д.

2 Тестирование в различных версиях ОС: Windows, Mac, Linux.

3 JavaScript-код работает в разных браузерах.

4 Просмотр на мобильных устройствах.

Примеры для тестирования различных веб-форм и элементов.

Всегда помните, что сначала нужно проводить позитивное тестирование с валидными данными, а уже после использовать негативные кейсы и вызывать ошибки. Не забывайте проверять валидации как на клиенте, так и на сервере.

1 Тестирование текстовых полей.

Примеры – Тестирование ввода специальных символов, валидация пустых полей.

2 Тестирование тегов в полях ввода.

Примеры – Ввод HTML-тегов, например <script>, для проверки уязвимостей XSS, ввод CSS для проверки его обработки.

3 Тестирование полей для URL.

Примеры – Тестирование ввода невалидных URL, например с неправильными протоколами или символами.

4 Тестирование загрузки файлов.

Примеры – Загрузка файлов с невалидными расширениями, тестирование ограничений на размер файла.

5 Тестирование полей ввода номеров телефонов.

Примеры – Ввод номеров с недопустимыми символами, тестирование международных форматов номеров.

6 Тестирование полей для email.

Примеры – Тестирование ввода корректного email, email без символа «@», ввод email с некорректным доменом.

7 Тестирование полей для ввода паролей.

Примеры – Тестирование паролей с недостаточной сложностью, тестирование функции восстановления пароля.

8 Тестирование капчи.

Примеры – Проверка работы капчи при неверном вводе, тестирование обновления изображения капчи.

9 Тестирование динамических элементов.

Примеры – Проверка обновления содержимого страницы при изменении данных в форме без перезагрузки, тестирование слайдеров на чувствительность и точность.

10 Многостраничные формы.

Примеры – Проверка сохранения введенных данных при переходе между страницами формы, тестирование возвращения к предыдущим шагам без потери данных.

11 Тестирование чек-боксов и радиокнопок.

Примеры – Проверка возможности выбора и снятия выбора, тестирование взаимоисключающего выбора в группе радиокнопок.

12 Тестирование календаря.

Примеры – Тестирование ограничений на выбор дат, проверка корректности ввода дат вручную и через виджет календаря.

13 Тестирование выпадающих списков (Dropdowns).

Примеры – Проверка всех доступных опций в выпадающем списке, тестирование поведения при выборе различных опций.

14 Адаптивность.

Примеры – Тестирование отображения форм на мобильных устройствах, проверка работы форм в разных ориентациях экрана (портретной и ландшафтной).

15 Кросс-браузерное тестирование.

Примеры – Проверка совместимости форм в различных версиях и типах браузеров, включая старые версии.

16 Доступность и Usability.

Примеры – Тестирование использования формы с помощью скринридеров, оценка интуитивности интерфейса и легкости взаимодействия с элементами.

Практические задания

Задание

Тренажер для тестировщиков в открытом доступе http://testingchallenges.thetestingmap.org/index.php.

Спецификация: пользователь должен заполнить необходимые данные, чтобы получить доступ к форуму в качестве обычного пользователя.

Сейчас можно протестировать только поле «Имя». Поле имеет максимальную длину 30.

Что не стоит тестировать: разные браузеры, чрезвычайно большие запросы, увеличение и уменьшение масштаба браузера. Не используйте инструменты автоматизации. Сервер отключит доступ при скорости более 30 запросов в секунду на каждый IP-адрес.

Ваша цель – составить чек-лист для тестирования поля First Name. Применить данные проверки на форме ввода данных. Результаты валидных проверок будут отображены ниже формы ввода – Checks found: N out of 18. Постарайтесь по максимуму покрыть проверками поле.

Контрольные вопросы

1 Для чего составляется чек-лист?

2 Предложите пять проверок для тестирования поля «Дата»?

3 Что такое DevTools? Для чего нужно знать это тестировщику?

4 Чем полезна вкладка Console?

5 Как изменить скорость загрузки страницы?

6 Где посмотреть скорость загрузки страницы?

7 В чем разница в тестировании локализации и интернационализации? Приведите примеры дефектов.

4 Лабораторная работа № 4. Тестирование API

Цель работы: понять, что такое API, как они функционируют и какие виды API существуют (REST, SOAP и т. д.); научиться использовать инструменты для тестирования API, такие как Postman или SoapUI; сформулировать тест-кейсы для проверки функциональности, производительности и безопасности API; изучить, как правильно интерпретировать ответы API, включая статус-коды и тела ответов.

4.1 Основные теоретические положения

Веб-сервисы представляют собой программные системы, которые предназначены для обработки и обмена данными по сети с использованием стандартных протоколов. Они позволяют различным приложениям взаимодействовать друг с другом, независимо от платформы и языка программирования, на которых они реализованы.

Пример веб-сервиса: Взаимодействие между авиакомпаниями и бюро путешествий. Первые предоставляют через веб-службы полезную информацию, которую вторые используют при поиске оптимальных предложений своим клиентам.

Веб-сервисами обычно называют приложения, которые предлагают функциональность через **API** (Application Programming Interface – интерфейс программирования приложений) и используют такие стандартные форматы, как XML или JSON для передачи данных.

Работу API можно описать следующим образом (рисунок 4.1): интерфейс представляет собой промежуточный слой между двумя приложениями. Он позволяет двум программам обмениваться информацией и выполнять функции, не раскрывая своего внутреннего API. Скрытие части функций называется инкапсуляцией.

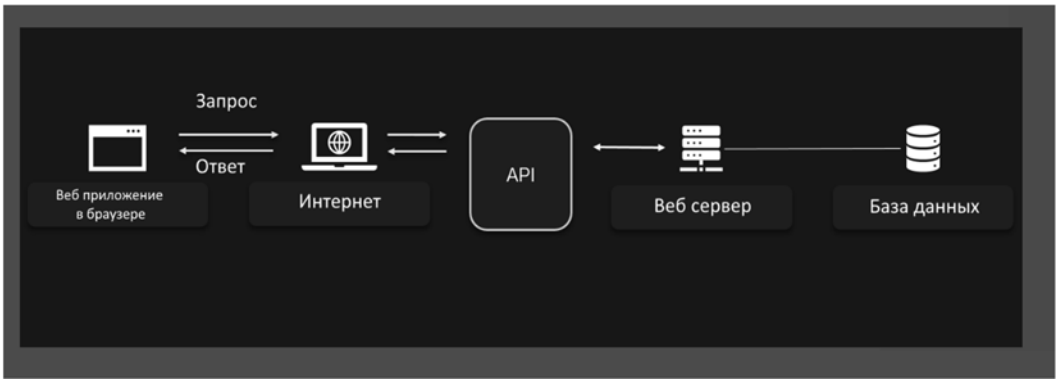

Рисунок 4.1 – Работа API

Есть три метода взаимодействия с API.

1 Процесс, который может выполнять программа при помощи этого интерфейса.

2 Данные, которые нужно передать интерфейсу для выполнения им функции.

3 Данные, которые программа получит на выходе после работы с API.

Разработчик имеет полную свободу в выстраивании функций API. Например, отдельный набор функций может определять возможность регистрироваться и авторизоваться в программе.

Программный интерфейс может реализовываться сервисом ОС, интернет-протоколом, программной библиотекой, фреймворком, множеством иных способов.

Типы программных архитектур.

Архитектура веб-приложений – это план одновременных взаимодействий между компонентами, базами данных, промежуточными системами, пользовательскими интерфейсами и серверами в приложении. Его также можно описать как макет, который логически определяет соединение между сервером и клиентской частью для лучшего взаимодействия с интернетом.

Поэтому, прежде чем приступить к проекту разработки веб-приложений, важно выбрать тип архитектуры веб-приложения, а также модель компонентов. Ведь хорошо продуманная архитектура веб-приложения может справляться с различными нагрузками и умело адаптироваться к изменяющимся бизнес-требованиям, обеспечивая быстрое взаимодействие с пользователем, что еще больше повышает производительность приложения. Также разработчик может взять на себя несколько задач одновременно, разделив структуру на несколько небольших модулей, что в конечном итоге также сократит время разработки. Кроме того, становится проще интегрировать новые функции, не влияя на общую структуру.

***Основные виды архитектур*.**

1 Монолитная архитектура. Монолитная архитектура представляет собой единую программу, где все компоненты плотно связаны. Пример: классическое веб-приложение с одной кодовой базой, где фронтенд и бэкенд работают в одной системе. Изменения в одной части приложения могут повлиять на другие части, и обновление требует перезапуска всего приложения.

2 Сервисно-ориентированная архитектура (SOA). В SOA приложения состоят из множества взаимодействующих сервисов, которые могут работать независимо друг от друга. Пример: в интернет-магазине один сервис отвечает за управление продуктами, другой – за обработку платежей, третий – за отправку уведомлений. Эти сервисы могут быть написаны на разных языках программирования и разворачиваться независимо.

3 Микросервисная архитектура. Микросервисы представляют собой небольшие, независимые и самодостаточные сервисы, которые выполняют свою конкретную задачу. Пример: в онлайн-кинотеатре отдельные микросервисы могут отвечать за аутентификацию пользователей, управление контентом и обработку рекомендаций. Микросервисы могут масштабироваться и обновляться независимо друг от друга.

4 Бессерверная архитектура (Serverless). В бессерверной архитектуре разработчики пишут код, который выполняется в ответ на события, и не беспокоятся о серверах. Пример: использование AWS Lambda для обработки изображений. Вместо того чтобы управлять сервером для обработки загруженных изображений, код автоматически выполняется при загрузке, и за это отвечает облачный провайдер.

Практические задания

Задание 1

Запрос 1: Создать питомца.

Условие: Питомец должен быть успешно создан с уникальным ID и корректным случайным данным для имени категории. ID должен генерироваться автоматически при каждой отправке запроса.

Ожидаемый результат: Ответ должен содержать статус 200 (OK) и данные о созданном питомце.

Запрос 2: Получить информацию о питомце по ID.

Условие: Запрос должен вернуть информацию о питомце, используя ID, полученный из первого запроса.

Ожидаемый результат: Ответ должен содержать статус 200 (OK) и тест, который проверяет, что имя категории из первого запроса совпадает с именем категории из ответа.

Задание 2

Запрос 1: Создать пользователя.

Условие: Пользователь должен быть успешно создан с уникальным именем и корректными данными (имя, email, пароль и т. п.). Все данные должны генерироваться автоматически при каждой отправке запроса.

Ожидаемый результат: Ответ должен содержать статус 200 (OK) и данные ответа.

Запрос 2: Получить информацию о пользователе по имени.

Условие: Запрос должен вернуть информацию о пользователе, используя имя, полученное из первого запроса.

Ожидаемый результат: Ответ должен содержать статус 200 (OK) и тест, который проверяет, что email из первого запроса совпадает с email из текущего ответа.

Задание 3

Запрос 1: Создать заказ для питомца.

Условие: Заказ должен быть успешно создан с уникальным ID и корректными данными (ID питомца, количество). ID и quantity должны генерироваться автоматически при каждой отправке запроса (ID должно быть значение от 1 до 10).

Ожидаемый результат: Ответ должен содержать статус 200 (OK) и данные о созданном заказе.

Запрос 2: Получить информацию о заказе по ID.

Условие: Запрос должен вернуть информацию о заказе, используя ID, полученный из первого запроса.

Ожидаемый результат: Ответ должен содержать статус 200 (OK) и тест, который проверяет, что quantity из первого запроса совпадает с quantity из текущего ответа.

Контрольные вопросы

1 Что такое API и для чего оно используется в программировании? Объясните основные принципы работы API.

2 Какие виды тестирования API вы знаете? Укажите, в чем состоит различие между функциональным, нагрузочным и безопасном тестировании.

3 Что такое метод HTTP и какие основные методы существуют? Перечислите и опишите методы GET, POST, PUT, DELETE.

4 Какова структура URL, которую вы будете тестировать? Укажите компоненты URL, такие как протокол, хост, порт и путь.

5 Что такое HTTP-заголовки и какую роль они играют в запросах API? Приведите примеры существенных заголовков.

6 Как можно обработать ошибки при тестировании API? Рассмотрите карты типов ошибок и методы их диагностики.

7 Какие инструменты вы используете для тестирования API и почему? Приведите примеры: Postman, Curl, JMeter и их особенности.

8 Что такое документация API и почему она важна? Объясните, как документация помогает в тестировании и разработке.

9 Каковы основные критерии успешного тестирования API? Обсудите, как вы определяете, что API работает правильно.

10 Как вы можете проверить производительность API? Опишите методики, которые вы можете использовать для оценки быстродействия.

Список литературы

1 **Пероцкая, В. Н.** Основы тестирования программного обеспечения : учеб. пособие / В. Н. Пероцкая, Д. А. Градусов. – Владимир : ВлГУ, 2017. – 100 с.

2 **Морозова, Ю. В.** Тестирование программного обеспечения : учеб. пособие / Ю. В. Морозова. – Томск : Эль Контент, 2019. – 120 с.

3 Тестирование, оценка программного обеспечения : учеб.-метод. пособие / М. М. Меженная, Т. В. Гордейчук, М. М. Борисик [и др.]. – Мн. : БГУИР, 2016. – 64 с. : ил.

4 **Куликов, С. С.** Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Мн. : Четыре четверти, 2020. – 312 с.