Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)**

УТВЕРЖДАЮ
Заведующий кафедрой ИСПИ
_____ И.Е. Жигалов
«20» марта 2025 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ МЕЖДИСЦИПЛИНАРНОГО КУРСА

«ТЕСТИРОВАНИЕ ИНФОРМАЦИОННЫХ РЕСУРСОВ»

В РАМКАХ ПРОФЕССИОНАЛЬНОГО МОДУЛЯ

«ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНФОРМАЦИОННЫХ РЕСУРСОВ»

09.02.09 Веб-разработка
Разработчик веб приложений

Владимир, 2025

Методические указания к лабораторным работам междисциплинарного курса «Тестирование информационных ресурсов» разработал ассистент кафедры ИСПИ Хлызова В.Г.

Методические указания к лабораторным работам рассмотрены и одобрены на заседании УМК специальности 09.02.09 Веб-разработка
протокол № 1 от «10» марта 2025 г.

Председатель УМК специальности _____ И.Е. Жигалов

Методические указания к лабораторным работам рассмотрены и одобрены на заседании кафедры ИСПИ
протокол № 7а от «12» марта 2025 г.

Рецензент от работодателя:
руководитель группы обеспечения
качества программного обеспечения
ООО «БСЦ МСК» _____ С.С. Смирнова

СОДЕРЖАНИЕ

Лабораторная работа №1. Изучение этапов тестирования ПО. Тестирование калькулятора

ЦЕЛЬ ВЫПОЛНЕНИЯ РАБОТЫ

Изучить этапы тестирования ПО, виды тестирования. Провести тестирование предложенного приложения.

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

Цели тестирования:

- Обнаружение дефектов;
- Повышение уверенности в уровне качества;
- Предоставление информации для принятия решений;
- Предотвращение дефектов;
- Оценка тестируемого приложения.

Этапы тестирования:

1. Этап «Планирование».

На этом этапе определяются задачи, сроки тестирования, составляют календарный план работ, пишется тест-план. В тест-план заносится следующая информация:

- Цель тестирования (то, для чего проводится тестирование).
- Процедуры и методики тестирования (используемые техники и технологии, см. таблицу 1).
- Описание тестируемой функциональности (то, что будет протестировано).
- Критерии завершения тестирования (выполнение всех тестовых сценариев будет одним из критериев завершения тестирования).

2. Этап «Анализ и проектирование тестов».

Тестовый сценарий (тест-кейс) – это высокоуровневое описание действий тестировщика для проверки функций системы с целью обнаружения дефектов.

Каждый тестовый сценарий содержит:

- название;
- начальные условия;
- последовательность действий;
- ожидаемый результат.

Методики проектирования тестов.

1. Эквивалентное разбиение – методика тестирования, в котором текстовые сценарии разбиваются в порядке эквивалентности.

Цель: разбиение входных и выходных данных на классы эквивалентности для уменьшения числа тестов.

Предположение: программа реагирует на все элементы одного класса эквивалентности одинаково.

Проектирование тестовых сценариев:

1. комбинирование классов эквивалентности
2. для каждого класса эквивалентности создаётся тестовый сценарий
3. для каждого тестового сценария определяются предусловия и ожидаемые результаты

Пример. Допустим, тестируем Интернет-магазин, продающий карандаши. В заказе необходимо указать количество карандашей (максимум для заказа – 1000 штук). В зависимости от заказанного количества карандашей различается стоимость:

1 – 500 – 10 руб. за карандаш;

501 – 1000 – 8 руб. за карандаш;

Очевидно, что входные данные мы можем разделить на следующие классы эквивалентности:

- Невалидное значение: >1000 штук;
- Невалидное значение: <=0;
- Валидное значение: от 1 до 500;
- Валидное значение: от 501 до 1000.

2. Анализ граничных значений

По статистике много дефектов находится вокруг граничных значений, разделяющих классы эквивалентности. Следовательно, границы необходимо тестировать более тщательно для выявления дефектов, связанных с операторами отношения (<, <=, <>, =, >=, >). Создан для дополнения эквивалентного разбиения.

Пример. Используя пример, приведенный в эквивалентном разбиении, можно определить граничные значения для каждого класса эквивалентности:

- 1000, 1001;
- 0, 1;
- 500, 501.

3. На этапе «Реализация и выполнение тестов» происходит выполнение тестов, запись результатов тестирования, оформление дефектов.

Дефект: 1. частично неверная программа, неверная команда или определение данных, являющееся причиной отказа; 2. состояние программного продукта или одного из его компонентов, которое в определённых условиях (например, при большой нагрузке) может нарушить требуемую функцию продукта или привести к отказу.

Дефект содержит в себе следующие данные:

- Название/краткое описание дефекта
- Описание дефекта. Содержит в себе последовательность действий для обнаружения дефекта, ожидаемый результат, фактический результат.
- Вложения. Например скриншоты.

4. Этап «Оценка и отчетность»

- Оценка результатов тестирования на соответствие критериям выхода;
- Описание протестированной функциональности;
- Количество и критичность найденных дефектов;
- Заключение об использовании приложения.

Таблица 1 Виды тестирования ПО

Вид тестирования	Описание
По объекту тестирования:	
Функциональное тестирование	Тестирование, основанное на анализе функциональной спецификации компонента или системы
Нагрузочное тестирование	Тип тестирования, проводимый с целью оценки поведения компонента или системы при возрастающей нагрузке, например количестве параллельных пользователей и/или интенсивности операций, а также определения какие ресурсы системы и в каком объеме будут при этом использованы.
Тестирование производительности	Процесс тестирования с целью определить производительность программного продукта. Производительность - степень, с которой система или компонент выполняет заложенные в нее функции в установленных рамках на время обработки и пропускную способность.
Тестирование стабильности	Тестирование, целью которого является оценка работы системы при длительной средней нагрузке
Тестирование удобства использования	Это метод тестирования направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий
Тестирование интерфейса пользователя	Предполагает проверку соответствия приложения требованиям к графическому интерфейсу, профессионально ли оно выглядит, выполнено ли оно в едином стиле

Тестирование безопасности	Стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным
Тестирование совместимости	Подход, основной целью которого является обеспечение качественной работы конечного продукта с другими программами, операционными системами, аппаратным обеспечением и т.д.
По знанию системы:	
Тестирование чёрного ящика	Тестирование, функциональное или нефункциональное, без знания внутренней структуры компонента или системы
Тестирование белого ящика	Тестирование, основанное на анализе внутренней структуры компонента или системы
Статическое тестирование	Проводится без запуска программы
Динамическое тестирование	Проводится с запуском программы
По степени автоматизированности:	
Ручное тестирование	Функциональное тестирование, при котором все шаги тестов выполняются вручную
Автоматизированное тестирование	Процесс верификации программного обеспечения, при котором инициализация и выполнение тестов выполняется с помощью инструментов для автоматизации тестирования
По степени изолированности компонентов:	
Компонентное (модульное) тестирование	Метод тестирования, при котором проверяется минимально возможный для тестирования компонент, например, отдельный класс или функция. Чаще всего модульное тестирование осуществляется разработчиками
Интеграционное тестирование	Метод тестирования, при котором проверяются интерфейсы между компонентами, подсистемами.

Системное тестирование	Метод тестирования, при котором проверяется интегрированная система на ее соответствие требованиям.
Приемочное тестирование	Формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью: определения удовлетворяет ли система приемочным критериям, вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет
По уровню независимости тестирования:	
Альфа тестирование	Имитация реальной работы с системой штатными разработчиками, либо реальная работа с системой потенциальными пользователями/заказчиком. Чаще всего альфа-тестирование проводится на ранней стадии разработки продукта, но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приемочного тестирования.
Бета тестирование	Выполняется будущими пользователями для того, что бы получить обратную связь о продукте. Иногда распространяются версии с ограничениями по времени или по функциональности для некоторой группы лиц, с тем, что бы убедиться, что продукт содержит достаточно мало ошибок.
По времени проведения тестирования	
Регрессионное тестирование	Тестирование уже протестированной программы, которое проводится после модификации для уверенности в том, что процесс модификации не внес или не активизировал ошибки в областях, не подвергавшихся изменениям. Проводится после изменений в коде программного продукта или его окружения
Повторное тестирование	Тестирование, во время которого исполняются тестовые сценарии, выявившие ошибки во время последнего запуска, для подтверждения успешности

	исправления этих ошибок
По признаку позитивности сценариев:	
Позитивное тестирование	Это тестирование на данных или сценариях, которые соответствуют нормальному (штатному, ожидаемому) поведению системы. Основной целью является проверка того, что при помощи системы можно сделать то, для чего она создавалась
Негативное тестирование	Это тестирование на данных или сценариях, которые соответствуют нештатному поведению тестируемой системы - различные сообщения об ошибках, исключительные ситуации. Основной целью является проверка устойчивости системы к воздействиям различного рода, валидация неверного набора данных, проверка обработки исключительных ситуаций (как в реализации самих программных алгоритмов, так и в логике бизнес-правил)
По степени подготовленности к тестированию:	
Тестирование по документации	Подход к тестированию, при котором тестовые сценарии разрабатываются на основе целей и условий тестирования, вытекающих из требований, то есть тесты, проверяющие определенные функции или оценивающие нефункциональные атрибуты системы, такие как надежность или практичность.
Тестирование по тест-кейсам	Тестирование производится по заранее написанным тестовым сценариям
Исследовательское тестирование	Тестирование без применения документаций и тестовых сценариев. Данный вид тестирования заранее не определяется в плане тестирования, и такие тесты создаются, выполняются и модифицируются динамически, по мере необходимости.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Составить тест-план для тестирования приложения «Калькулятор».
2. Спроектировать тестовую модель (тест-кейсы) для любых трех операций (например, сложение, умножение, деление).
3. Провести тестирование по тестовой модели и тест-плану.
4. Описать найденные дефекты.
5. Составить отчет по завершению тестирования.

СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы
2. Тест-план
3. Тестовая модель
4. Дефекты
5. Отчет по завершению тестирования
6. Выводы по лабораторной работе

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое тестирование ПО и каковы его основные цели?
2. Какие этапы включает процесс тестирования ПО?
3. Что такое тестовый сценарий (тест-кейс) и какие элементы он включает?
4. Что такое эквивалентное разбиение и как оно используется в проектировании тестов?
5. Какие виды тестирования по объекту тестирования вы знаете и чем они отличаются друг от друга?
6. Что такое граничные значения и почему их важно тестировать?
7. Какие этапы включает процесс реализации и выполнения тестов?
8. Что включает этап оценки и отчетности в тестировании?
9. Какие виды тестирования по степени автоматизированности вы знаете и в чем их отличия?
10. Каковы особенности альфа-тестирования и бета-тестирования?
11. Что такое регрессионное тестирование и когда оно проводится?
12. Какие виды тестирования существуют по признаку позитивности сценариев?
13. Что такое исследовательское тестирование и как оно отличается от тестирования по документации и тест-кейсам?

Лабораторная работа №2. Написание тест-кейсов. Тестирование программы, которая определяет тип треугольника по трем его сторонам.

ЦЕЛЬ РАБОТЫ

Изучить написание тест-кейсов. Протестировать приложение по написанным сценариям.

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

Зачем нужны тест-кейсы? Тест-кейсы должен помочь нам провести проверку продукта без ознакомления с всей документацией. Написанный один раз, удобный в поддержке тест-кейс сэкономит много времени и сил тестировщикам.

Тест-кейсы играют ключевую роль в процессе тестирования программного обеспечения. Вот несколько основных причин, зачем они нужны:

1. Структурированность Тестирования
   - Организация: Тест-кейсы помогают структурировать процесс тестирования, обеспечивая четкий план действий. Это упрощает понимание того, что именно должно быть протестировано.
   - Последовательность: Они обеспечивают последовательность выполнения шагов, что минимизирует вероятность пропуска важных этапов или функций.
2. Повторяемость
   - Реабилитация: Хорошо написанные тест-кейсы можно повторять для проверки изменений в коде или после исправления дефектов, что помогает убедиться, что исправления не вызвали новых проблем.
   - Стабильность: При автоматизации тестирования тест-кейсы служат основой для автоматизированных тестов, которые можно повторять регулярно.
3. Документация и Отчетность
   - Документирование: Тест-кейсы создают документацию о том, что было протестировано, как и какие результаты были получены. Это полезно для анализа и отчетности.
   - История: В случае обнаружения дефектов тест-кейсы служат справочным материалом для анализа проблемы и выявления причин.
4. Тестирование по Требованиям

- Соответствие: Тест-кейсы помогают убедиться, что программное обеспечение соответствует требованиям и спецификациям. Каждый тест-кейс связан с определенной функциональностью или требованием.

5. Улучшение Качества ПО
   - Выявление Дефектов: Они способствуют более тщательному выявлению дефектов и проблем в программном обеспечении, что в итоге повышает его качество.
   - Функциональное Покрытие: Тест-кейсы помогают охватить все функциональные и нефункциональные аспекты системы, уменьшая вероятность упущения важных сценариев тестирования.
6. Командная Работа
   - Согласованность: Тест-кейсы обеспечивают единообразие и согласованность в подходе к тестированию среди разных членов команды. Они помогают новым членам команды быстрее вникнуть в процесс тестирования.
   - Обратная Связь: Командные члены могут обсуждать и улучшать тест-кейсы, что способствует улучшению процесса тестирования в целом.
7. Управление Рисками
   - Идентификация Рисков: Тест-кейсы помогают выявить потенциальные риски и проблемы на ранних стадиях разработки, что позволяет предотвратить более серьезные проблемы в будущем.
8. Клиентская Оценка
   - Удовлетворение Требований: Взаимодействие с клиентами может требовать представления тест-кейсов для демонстрации, что продукт соответствует их ожиданиям и требованиям.

Пример Реальной Применимости

Ситуация: Ваша команда работает над новой функцией в приложении, и вы пишете тест-кейсы для проверки этой функции.

- Без Тест-Кейсов: Вы выполняете тесты наугад, и, возможно, упускаете некоторые критические сценарии. В результате вы можете не заметить ошибки, которые проявятся в производственной среде.
- С Тест-Кейсами: У вас есть четкий план тестирования. Вы протестировали все важные сценарии, зафиксировали результаты и проверили, что все требования выполнены. Это обеспечивает более высокий уровень уверенности в том, что функция работает корректно.

Любой тест-кейс обязательно включает в себя:

- Название — основная тема, или идея тест-кейса. Кратное 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.

Важно: оформляйте тест-кейсы в максимально читаемой форме. Чаще всего тест кейсы оформляют в виде таблицы.

Выполненный тест кейс выдает один из трех результатов:

- Положительный результат, если фактический результат равен ожидаемому результату,
- Отрицательный результат, если фактический результат не равен ожидаемому результату. В этом случае, найдена ошибка.
- Выполнение теста блокировано, если после одного из шагов продолжение теста невозможно. В этом случае так же, найдена ошибка.

Советы для написания тест-кейсов

- Четкость и Конкретность: Каждый шаг и результат должны быть однозначными.
- Повторяемость: Шаги теста должны быть легко повторяемыми.
- Логичность: Тест-кейсы должны быть логически структурированы и понятны для других тестировщиков.
- Полнота: Все аспекты тестируемой функции должны быть охвачены, включая граничные случаи и возможные ошибки.

Пример тест-кейса из Лабораторной работы №1:

Название тест кейса:	Проверка функции сложения.
Предусловия тест кейса:	Приложение запущено, поле ввода пустое.
Шаги тест кейса:	1) Нажать на кнопку со значением «1»; 2) Нажать на кнопку с символом «+»; 3) Нажать на кнопку со значением «2»; 4) Нажать на кнопку с символом «=».

Ожидаемый результат тест кейса:	Значение в поле вывода должно быть равно 3.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Написать минимум 12 кейсов для тестирования программы, которая определяет тип треугольника по трем его сторонам;
2. Получить от преподавателя ссылку на тестируемое приложение;
3. Протестировать приложение по написанным тест кейсам;
4. Описать найденные дефекты;
5. Составить отчет по завершению тестирования;

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Цель работы;
2. Тест кейсы;
3. Скриншот выполнения работы, сделанный из предложенного преподавателем ресурса;
4. Отчет по завершению тестирования;
5. Выводы по лабораторной работе.

ВОПРОСЫ

1. Что такое тест-кейс и какова его основная цель?
2. Почему тест-кейсы важны для процесса тестирования программного обеспечения?
3. Какие основные компоненты должен включать тест-кейс?
4. Какое значение имеют предусловия в тест-кейсе? Приведите пример предусловия для теста, связанного с авторизацией пользователя.
5. Что должно быть указано в разделе "Шаги" тест-кейса?
6. Каков основной принцип определения ожидаемого результата в тест-кейсе?
7. Каковы возможные результаты выполнения тест-кейса? Опишите их.
8. Что делать, если выполнение теста блокировано на каком-либо шаге?
9. Как тест-кейсы помогают в поддержке и повторении тестирования?
10. Почему тест-кейсы важны для документации и отчетности?
11. Какие рекомендации по написанию тест-кейсов помогут сделать их более эффективными?
12. Как тест-кейсы способствуют улучшению качества ПО и командной работе?
13. Приведите пример тест-кейса для проверки функции регистрации нового пользователя. Включите все необходимые компоненты.

14. Как тест-кейсы могут быть использованы для демонстрации клиенту, что продукт соответствует его требованиям?

Лабораторная работа №3. Тест-кейсы и чек-листы на основе бизнес-требований.

ЦЕЛЬ

Закрепить основы тестирования. Потренироваться в написании тест-кейсов и чек-листов по установленным бизнес-требованиям.

Тест-кейсы и чек-листы — это два различных инструмента, используемых в процессе тестирования программного обеспечения. Оба инструмента помогают структурировать тестирование, но они имеют разные цели и способы использования.

Что такое Тест-Кейсы?

Определение: Тест-кейс — это детализированный документ, который описывает конкретные шаги, которые нужно выполнить, чтобы проверить определенную функциональность программного обеспечения, а также ожидаемый результат этих шагов.

Структура тест-кейса:

1. Название: Краткое и ясное описание тестируемого функционала.
2. Предусловия: Условия, которые должны быть выполнены до начала выполнения теста.
3. Шаги: Последовательность действий, которые нужно выполнить.
4. Ожидаемый результат: Результат, который должен быть получен после выполнения шагов.
5. Фактический результат: Результат, который был получен в ходе тестирования (указывается после выполнения теста).

Пример тест-кейса:

- Название: Проверка функции регистрации нового пользователя.
- Предусловия: Пользователь находится на странице регистрации.
- Шаги:
    1. Ввести в поле "Имя" текст "Иван".
    2. Ввести в поле "Email" текст "ivan@example.com".
    3. Нажать кнопку "Зарегистрироваться".
- Ожидаемый результат: Пользователь успешно зарегистрирован и перенаправлен на страницу подтверждения.

Преимущества тест-кейсов:

- Детализация: Предоставляют полную информацию о том, как провести тест.

- Повторяемость: Легко повторить для проверки исправлений или изменений.
- Документация: Полезны для создания документации и отчетов о тестировании.

Определение: Чек-лист — это более простой и общий инструмент, который представляет собой список задач или элементов, которые необходимо проверить. Чек-листы часто используются для быстрой проверки наличия определенных функций или аспектов, без детального описания шагов тестирования.

Структура чек-листа:

1. Название: Краткое описание проверки.
2. Пункты для проверки: Список элементов или функций, которые нужно проверить.

Пример чек-листа:

- Название: Проверка основных функций регистрации.
- Пункты для проверки:
    1. Проверить, что форма регистрации загружается.
    2. Проверить, что все поля формы доступны для ввода.
    3. Проверить, что кнопка "Зарегистрироваться" активна.
    4. Проверить, что отображается сообщение об успешной регистрации.

Преимущества чек-листов:

- Простота: Легко составить и использовать для быстрого тестирования.
- Гибкость: Подходят для проверки общих аспектов или функций.
- Эффективность: Быстрый способ убедиться, что основные функции работают.

Разница между Тест-Кейсами и Чек-Листами

1. Детализация:
    - Тест-кейсы: Подробно описывают шаги и ожидаемые результаты. Подходят для глубинного тестирования и повторного использования.
    - Чек-листы: Более общие и простые, не содержат детализированных шагов. Подходят для поверхностного тестирования.
2. Цель:
    - Тест-кейсы: Используются для проверки конкретных функциональностей и сценариев, где важна точность и последовательность.
    - Чек-листы: Используются для быстрой проверки основных функций и наличия необходимых элементов.
3. Документация:
    - Тест-кейсы: Создают подробную документацию и отчетность.
    - Чек-листы: Обычно не требуют такой детализированной документации.
4. Применение:

- Тест-кейсы: Подходят для сложных тестов и сценариев, где необходима точность и повторяемость.
- Чек-листы: Подходят для проверки общих аспектов и функций, а также для ежедневного тестирования и быстрого обзора.

Примеры ситуаций:

- Тест-кейсы: Подходят для проверки бизнес-логики, функциональных требований и сложных сценариев взаимодействия с пользователем.
- Чек-листы: Подходят для проверки интерфейса пользователя, доступности базовых функций и общей стабильности системы.

Оба инструмента имеют свои области применения и могут дополнять друг друга в процессе тестирования, в зависимости от целей и требований проекта.

ХОД ВЫПОЛНЕНИЯ РАБОТЫ:

Система — http://users.bugred.ru

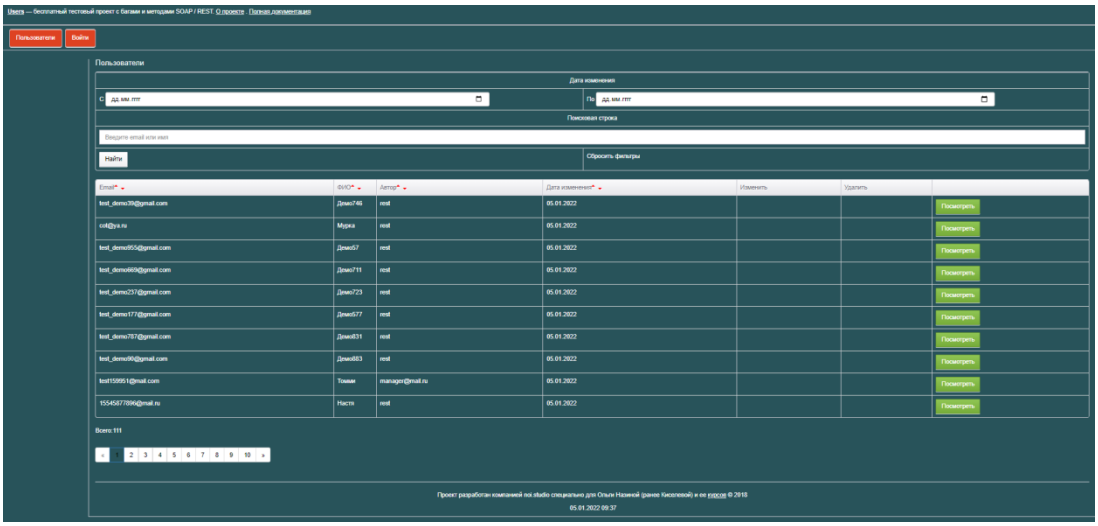


Рисунок 1 - Стартовая страница.

Изучите техническое задание к системе и на его основе выполните задания к данной лабораторной работе.

Техническое задание

1. Регистрация / авторизация

В системе есть возможность регистрации и авторизации, а также изменение своих учетных данных.

2. Список сотрудников

Главная страница содержит список сотрудников, есть возможность создать нового. Вы можешь посмотреть информацию о любом сотруднике, но изменить или удалить только своего.

Доступен поиск по ФИО пользователя, его email или автору. Также есть фильтрация по дате изменения. Все фильтры можно сбросить одной кнопкой.

Колонки в списке сотрудников можно сортировать. Сортировка идет по всей базе, а не только по той странице, на который вы находитесь.

Если сотрудник только что создан или его данные были изменены, он попадает наверх списка.

3. Карточка сотрудника

Поля карточки, в основном просто текстовые поля:

- Email
- ФИО
- Пол
- Дата рождения
- Начал работать в компании
- Хобби
- Телефон
- Адрес

Создавать дубли по нику или ФИО нельзя, считается, что все ФИО в системе уникальны.

4. Менеджер

В системе есть отдельная роль – менеджер. Данные для входа: manager@mail.ru / 1.

Менеджер может наблюдать за сотрудниками:

- Он видит все оповещения + может их сортировать и фильтровать;
- Он видит все текущие задачи, все отложенные и все выполненные;
- Он может отредактировать любую задачу;

5. Список задач

В разделе задач можно посмотреть созданные задачи, автора, кому они назначены и т.д. При создании задачи следует заполнить следующие поля:

— Название string (50)

— Описание string (500)

— Ответственный (поле выбора)

Созданная задача появляется в списке задач, где можно ее изменить, удалить или добавить в расписание. Простой пользователь видит только свои задачи, менеджер - вообще все. И все может менять.

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Изучите представленную преподавателем систему, напишите чек-лист и 5 тест-кейсов по описанному в ТЗ функционалу;
2. Проверьте систему по написанному чек-листу (обязательно отметьте пройденные пункты соответствующими статусами) и тест-кейсам;
3. Оформите найденные дефекты.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Цель работы;
2. Чек-лист;
3. Тест-кейсы;
4. Оформленные дефекты;
5. Вывод.

ВОПРОСЫ

Что такое чек-лист и для чего он используется в тестировании?
Какова структура чек-листа и какие элементы он содержит?
Какие преимущества предоставляют тест-кейсы по сравнению с чек-листами?
В каких ситуациях чек-листы могут быть более эффективны, чем тест-кейсы?
Как тест-кейсы способствуют улучшению документации и отчетности в процессе тестирования?
Почему важно, чтобы шаги тест-кейса были детализированы и конкретны?
Как тест-кейсы помогают в повторении тестирования после исправления дефектов?
Как чек-листы могут быть полезны для быстрой оценки базовых функций системы?
Как тест-кейсы и чек-листы могут быть использованы совместно для достижения лучшего результата в тестировании?
Какие недостатки могут быть у тест-кейсов и чек-листов, и как их можно минимизировать?

Лабораторная работа №4. SoapUI как инструмент эмуляции сервисов.

ЦЕЛЬ РАБОТЫ

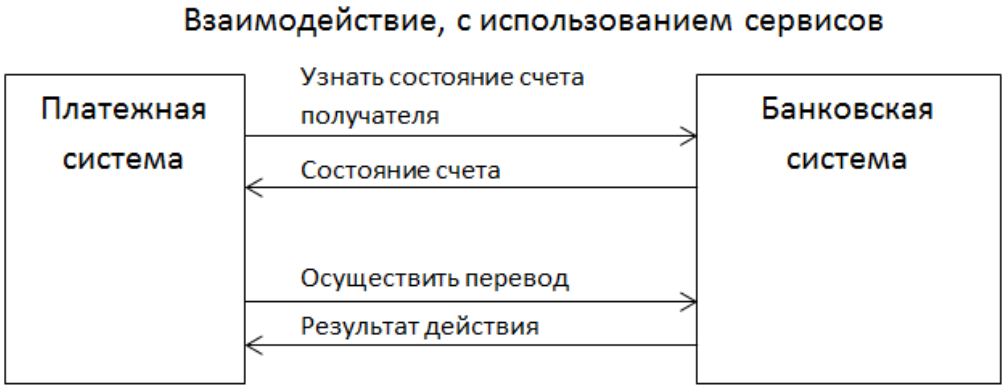
Изучить принципы работы со средством эмуляции работы веб-сервисов SoapUI.

Веб-сервис - идентифицируемая веб-адресом программная система со стандартизированными интерфейсами.

Как правило, системы не существуют сами по себе, они взаимодействуют с другими системами, приложениями, базами данных для того, чтобы получить или передать информацию.

Например, существует платежная система. Она предоставляет возможность осуществления платежей, переводов и другие услуги. И, например, для перевода денег на какой-нибудь счет, необходимо получить информацию о состоянии счета и возможности

его пополнения, и выполнить сам перевод. Все эти задачи (получения информации) легко могут выполнить сервисы.



У каждого веб-сервиса существует адрес, он может принимать сообщения в формате xml, и отправлять ответы так же в формате xml. Взаимодействие осуществляется по протоколам, например, http.

В тестировании ПС часто возникают такие ситуации, что необходимо протестировать только одну систему (например, только платежную или только банковскую систему) и доступа к другой системе нет. Для этого используются эмуляторы (или их еще называют заглушками), которые выполняют те же действия, что выполняла бы реальная система. Т.е. эмуляторы заменяют реальные системы во время тестирования.

Кроме заглушек существуют еще и драйверы. Основное отличие драйвера от заглушки во взаимодействии: драйвер вызывает тестируемую систему, тестируемая система вызывает заглушку.

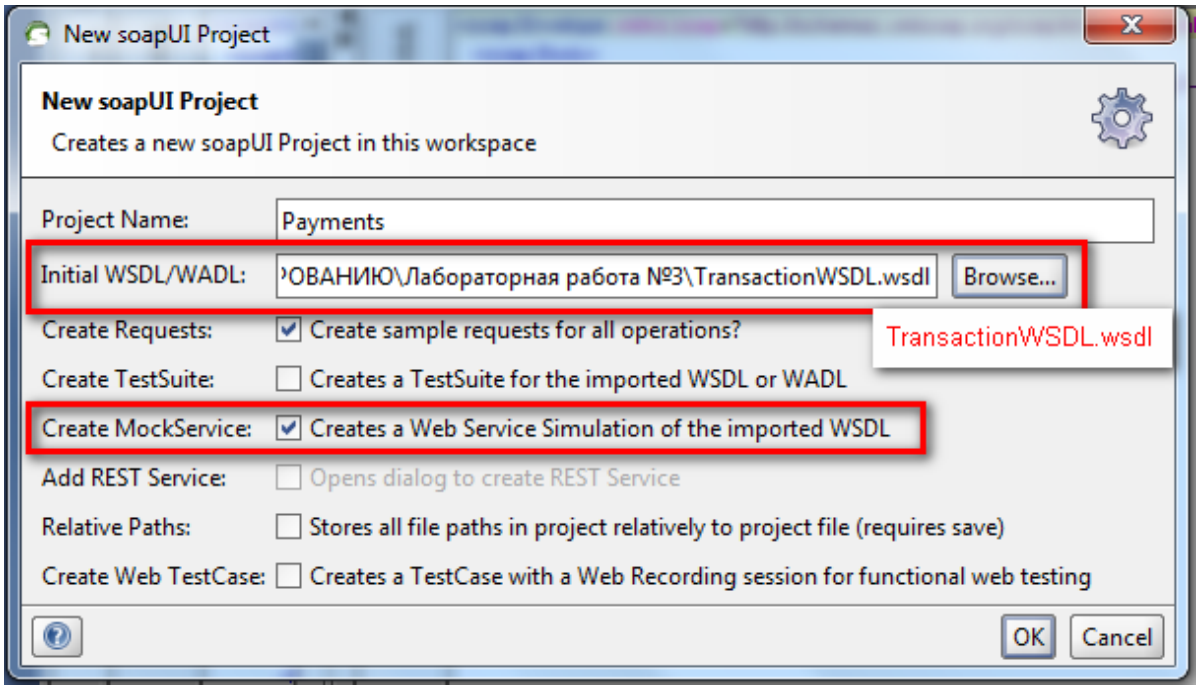
Драйвер – компонент программного обеспечения или средство тестирования, которое заменяет компонент, обеспечивающий управление и/или вызов компонента или системы.

Заглушка – минимальная или специализированная реализация программного компонента. Использующаяся для подмены компонента, от которого зависит разработка или тестирование другого компонента системы.

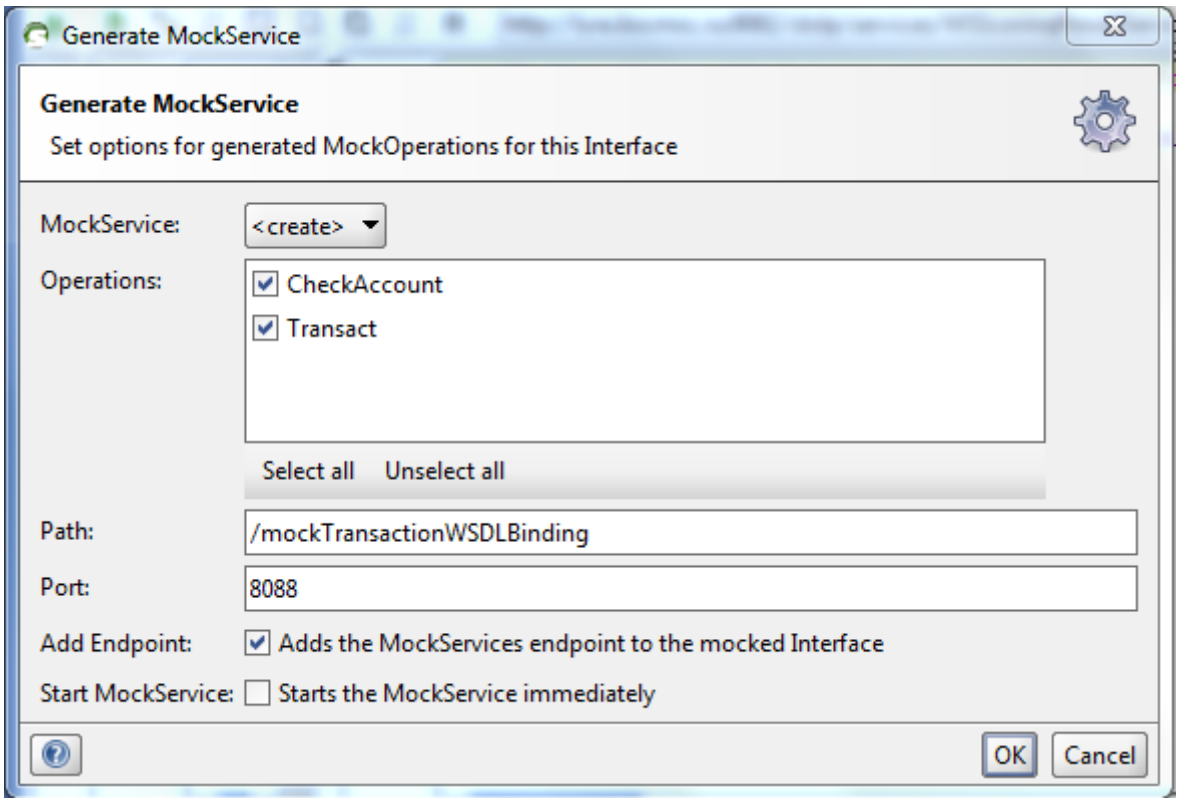
Программа SoapUI – одно из средств эмуляции ответов от веб-сервисов.

Ход работы:

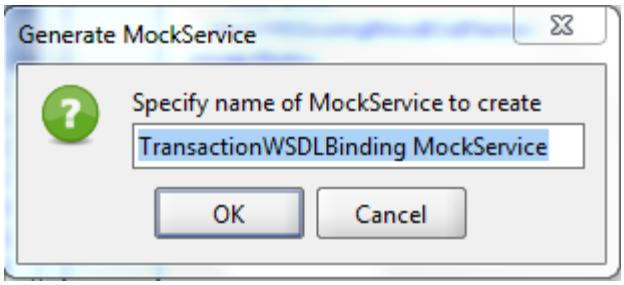
1. Установить программу SoapUI по ссылке https://clck.ru/KW6o5.
2. В открывшемся окне программы сделать: File – New SoapUI Project
3. Добавить wsdl сервиса, указать называние и проставить чек-бокс, что будет Mock-сервис. Нажать «ОК»:



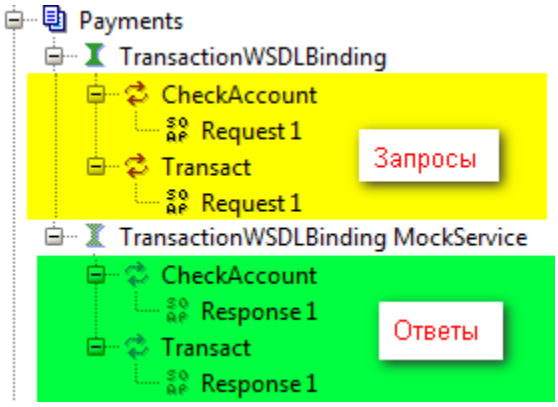
4. В следующем окне будут параметры сервиса. Путь к сервису и порт, на котором будет запущен. Нажать «ОК»:



5. Далее указывается название Mock-сервиса. Нажать «ОК».



6. Появились примеры запросов и Mock-сервис:



В сервисе две операции: проверить счет (CheckAccount) и провести перевод (Transact).

Операция CheckAccount, входные параметры:

- AccountNumber – номер счета;
- Name – имя клиента;
- Surname – фамилия клиента;

- Passport – номер паспорта.

Параметры ответа:

- AccountNumber – номер счета;
- Status – статус (может принимать значения: Close, Open, Blocked).

Операция Transact, входные параметры:

- AccountNumber
- Sum
- Target

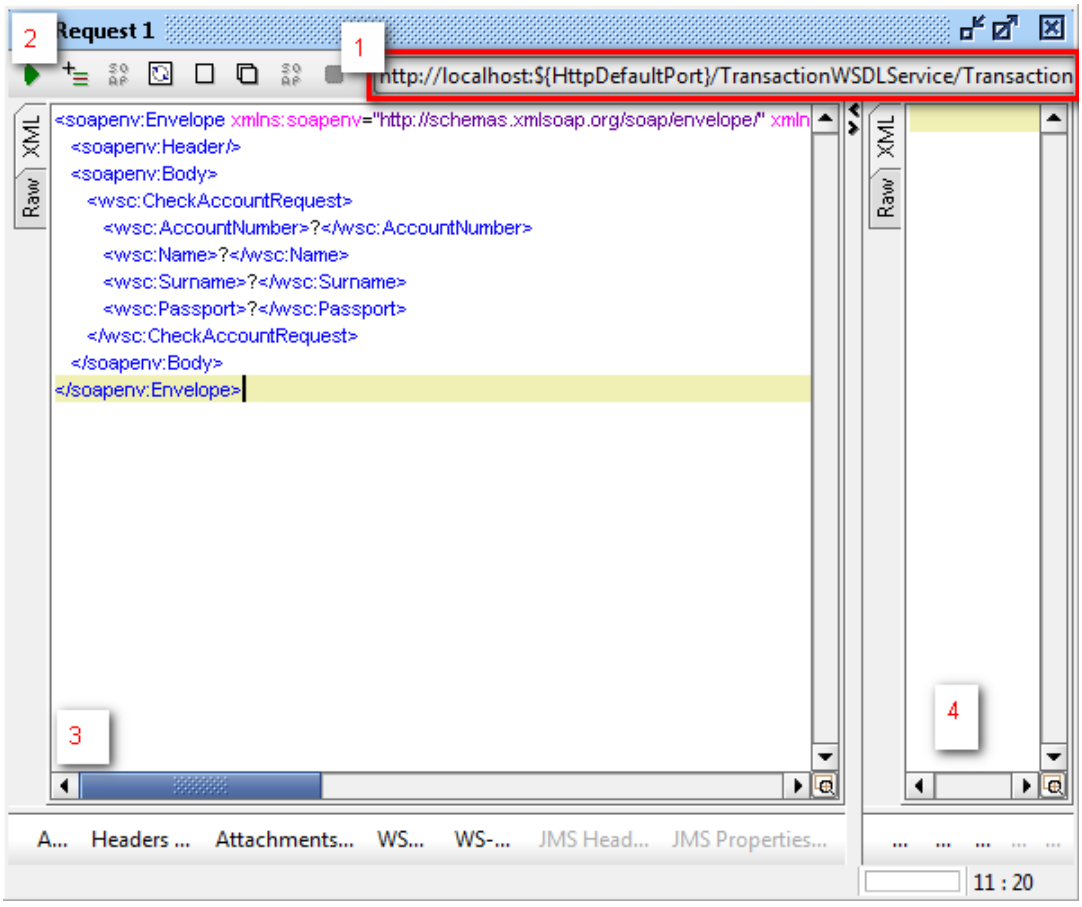
Параметры ответа:

- TransactionId – идентификационный номер транзакции;
- TransactionStatus – статус (Success).

Каждый сервис может вернуть ошибку состоящую из:

- ErrorCode – код ошибки;
- ErrorString – описание ошибки.

7. Если открыть Request1, то увидим окно:



1 – строка-ссылка на сервис (необходимо указать свой, т.е. он будет вторым из списка и содержать: http://<название комп.>:<порт>/<ссылка>, которые указывали в п. 4 при создании сервиса);

2 – кнопка для отправления запроса;

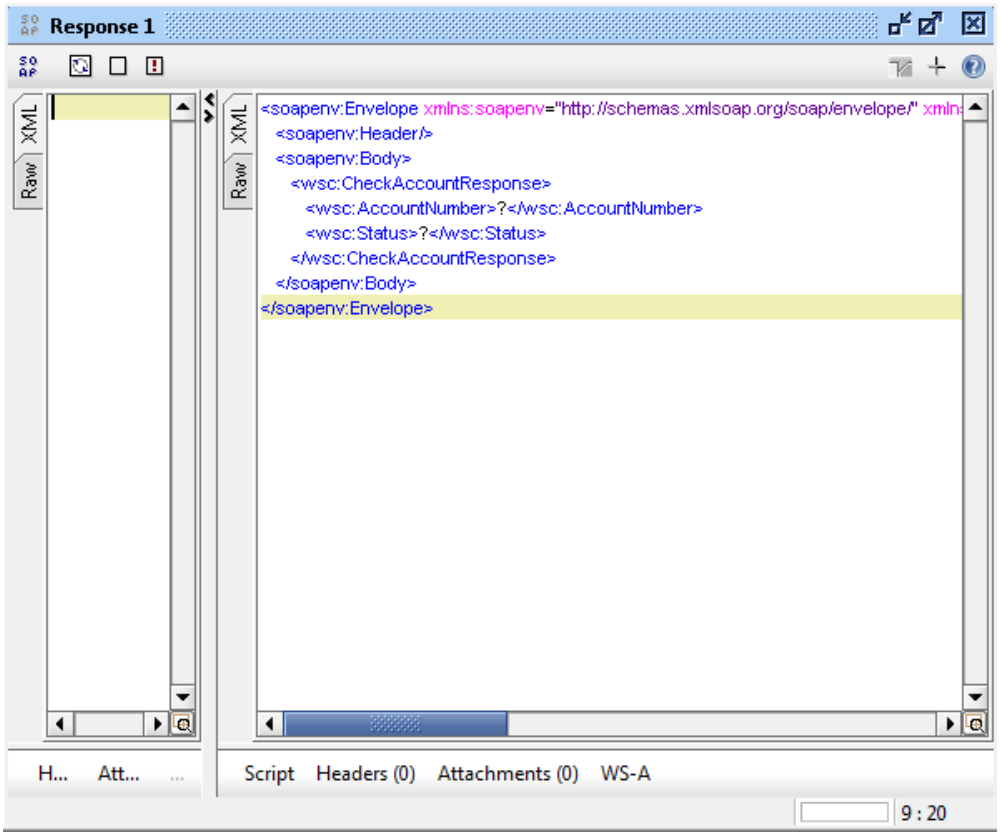
3 – окно запроса;

4 – окно ответа.

Можно изменить параметры запроса, например, сделать его таким:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsc="http://bank/schema/WSCommonTypes">
   <soapenv:Header/>
   <soapenv:Body>
      <wsc:CheckAccountRequest>
         <wsc:AccountNumber>100500</wsc:AccountNumber>
         <wsc:Name>Иванов</wsc:Name>
         <wsc:Surname>Иван</wsc:Surname>
         <wsc:Passport>17 10 020202</wsc:Passport>
      </wsc:CheckAccountRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

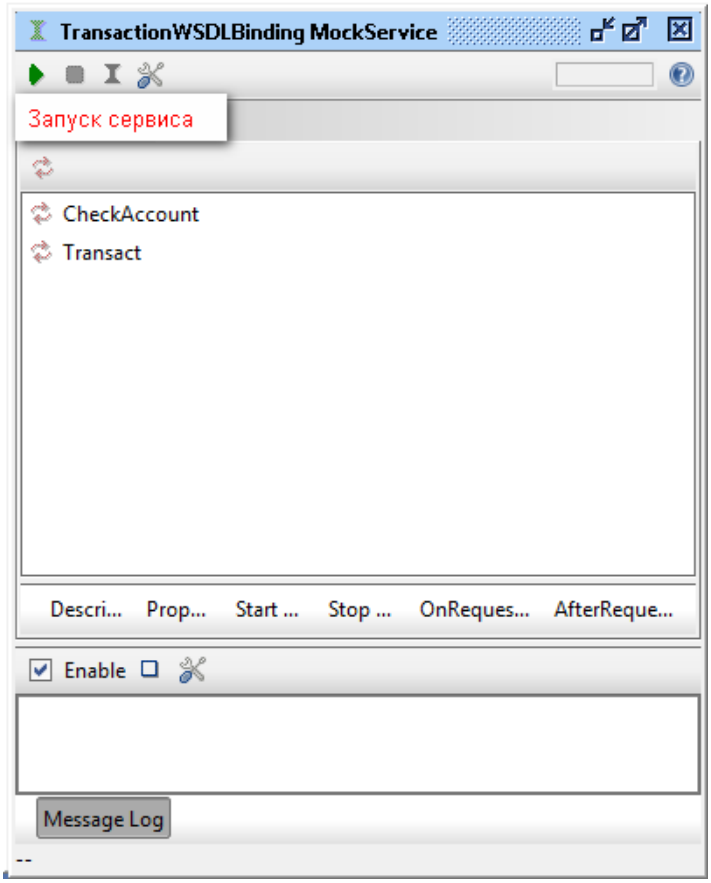
8. Если открыть Response1, то увидим похожее окно, там указывается ответ сервиса:



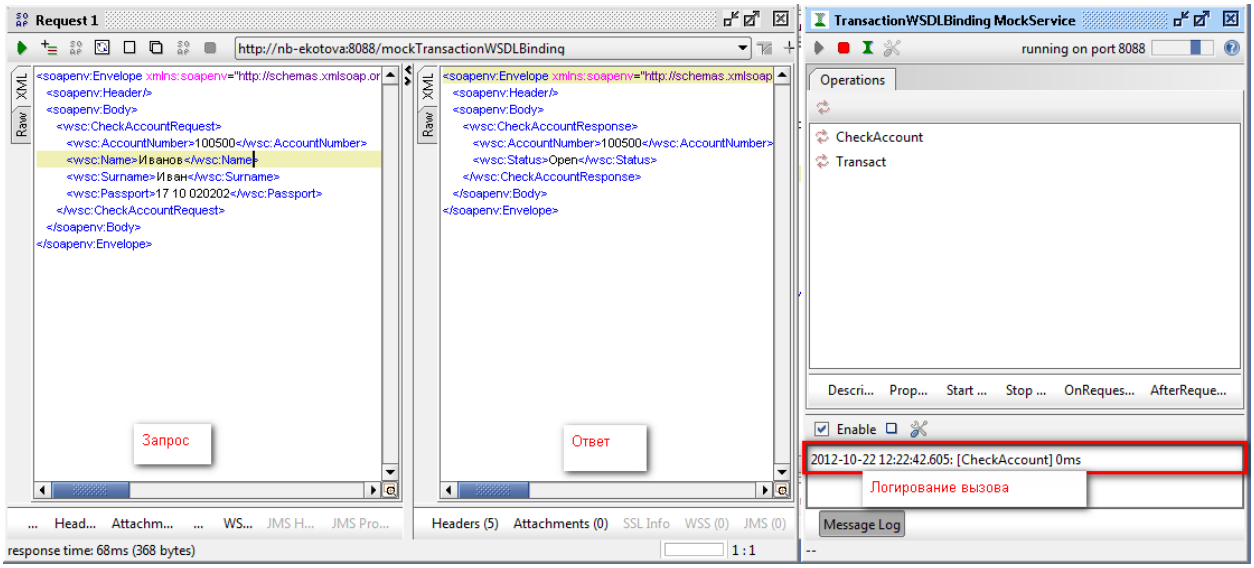
Можно изменить ответ на следующий:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsc="http://bank/schema/WSCommonTypes">
   <soapenv:Header/>
   <soapenv:Body>
      <wsc:CheckAccountResponse>
         <wsc:AccountNumber>100500</wsc:AccountNumber>
         <wsc:Status>Open</wsc:Status>
      </wsc:CheckAccountResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

9. Откроем сам Mock-сервис и запустим его:



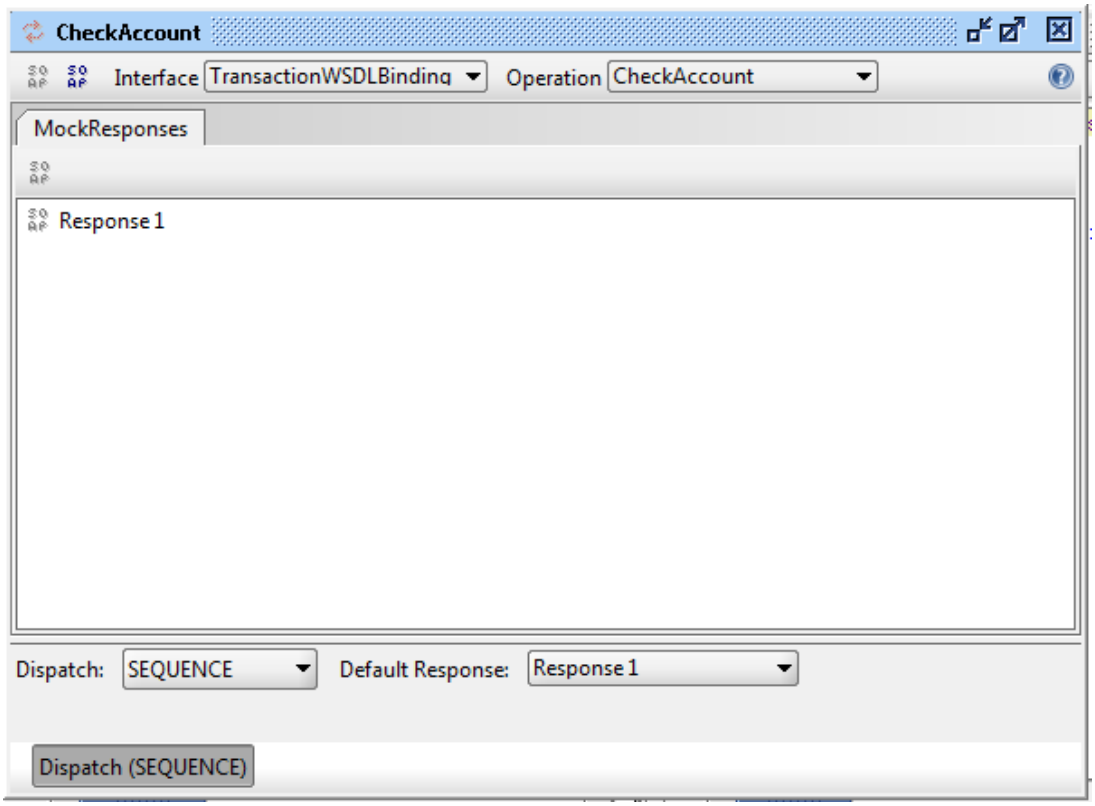
10. Если открыть запрос, который мы написали на п. 7, и попробовать отправить сообщение то получим следующее:



Таким образом, мы осуществили вызов эмулятора, и получили ответ.

Так же в тестировании есть необходимость, что бы на определенные запросы приходили определенные ответы. В SoapUI есть возможность с использованием языка Groovy обрабатывать те или иные запросы, и возвращать те или иные ответы.

Для этого необходимо открыть операцию веб-сервиса на редактирование:



Dispatch – указывает способ получения ответа. Можно выбрать Sequence, тогда все ответы будут выдаваться последовательно (если указано несколько ответов). Так же можно выбрать Script, тогда запрос будет обрабатываться скриптом.

Пример скрипта для операции Transact:

```
//возвращает рандомное значение (для использования в ответе:
<wsc:TransactionId>${TransactionId}</wsc:TransactionId>):

context.TransactionId = Math.random()
context.TransactionId =
context.TransactionId.toString().substring(5,15)

//Определяем name space’s:
def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context);
def holder =
groovyUtils.getXmlHolder(mockRequest.getRequestContent());
holder.declareNamespace("wsc", "http://bank/schema/WSCommonTypes");

//Определяем переменную AccountNumber из входного сообщения:
def accountNumber =
holder.getNodeValue("//wsc:TransactRequest/wsc:AccountNumber");

//Определяем ответ, в зависимости от AccountNumber (Error и Success –
названия ответов)
if (accountNumber == "100500"){
       return "Success";
}
if (accountNumber == "555555"){
  return "Error";
 }
```

Примеры ответов:

```
Success:
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsc="http://bank/schema/WSCommonTypes">
   <soapenv:Header/>
   <soapenv:Body>
      <wsc:TransactResponse>
```

```
         <wsc:TransactionId>${TransactionId}</wsc:TransactionId>
         <wsc:TransactionStatus>Success</wsc:TransactionStatus>
      </wsc:TransactResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

```
Error:
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsc="http://bank/schema/WSCommonTypes">
   <soapenv:Header/>
   <soapenv:Body>
      <wsc:TransactResponse>
         <wsc:ErrorCode>911</wsc:ErrorCode>
         <wsc:ErrorString>Ошибка</wsc:ErrorString>
      </wsc:TransactResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

1. Для операции ChechAccount реализовать:
   - В случае, если AccountNumber = 999999999, возвращать Close;
   - В случае, если AccountNumber = 777777777, возвращать Blocked;
   - В случае, если AccountNumber = 666666666, возвращать ошибочный ответ.
   - Для других значений AccountNumber возвращать Open;
   - В AccountNumber всегда возвращать входное значение.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

7. Цель работы;
8. Описание выбранной операции;
9. Код программы, скриншоты (если требуются);
10. Выводы по лабораторной работе.

ВОПРОСЫ

1. Что такое веб-сервис и какова его основная функция?

2. Какой формат сообщений чаще всего используется веб-сервисами для обмена данными?
3. Какие протоколы могут использоваться для взаимодействия веб-сервисов?
4. Приведите пример использования веб-сервиса в платёжной системе. Как веб-сервис может быть использован для выполнения задачи перевода денег?
5. Что такое эмулятор (или заглушка) в контексте тестирования веб-сервисов и почему он используется?
6. В чем основное отличие между заглушкой и драйвером в контексте тестирования?
7. Как заглушка взаимодействует с тестируемой системой?
8. Как драйвер используется для вызова тестируемой системы?
9. Как эмуляторы и заглушки помогают в тестировании, когда доступ к реальной системе ограничен или невозможен?
11. Как использование эмуляторов может повлиять на результаты тестирования веб-сервиса?
12. Какие потенциальные проблемы могут возникнуть при использовании заглушек и драйверов, и как их можно минимизировать?
14. Каковы преимущества и недостатки использования заглушек и драйверов в процессе тестирования?

Лабораторная работа №5. Регулярные выражения.

ЦЕЛЬ

Изучить механизм для поиска и замены текста. Применить полученные знания на практике.

Регулярные выражения (англ. regular expressions) — используемый в компьютерных программах, работающих с текстом, формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters).

Для поиска используется строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Их используют разработчики в коде приложения, тестировщики в автотестах или при работе в командной строке!

Чем это лучше простого поиска? Тем, что позволяет задать шаблон.

Регулярное выражение позволяет задать шаблон «найди мне цифры в таком-то формате».

Регулярные выражения являются мощным инструментом для работы с текстом и широко используются в тестировании программного обеспечения. Они представляют собой специальный синтаксис для поиска и манипуляции строками, что делает их очень

полезными для различных задач в тестировании. Вот основные причины, почему регулярные выражения важны в тестировании:

Основные Применения Регулярных Выражений в Тестировании

1. Проверка Форматов Данных:
    - Валидация Форматов: Регулярные выражения используются для проверки, соответствует ли данные определенному формату. Например, можно проверить, что электронный адрес имеет правильный формат или что номер телефона соответствует требуемой маске.
    - Пример: Проверка, что введенный текст соответствует формату номера кредитной карты (например, \d{4}-\d{4}-\d{4}-\d{4}).
2. Поиск и Замена:
    - Поиск Ошибок: Регулярные выражения позволяют искать определенные паттерны в логах или текстовых файлах, что помогает быстро находить ошибки или проблемы.
    - Замена Значений: Они также могут использоваться для замены текста или данных в строках. Например, можно заменить все вхождения одной строки на другую в исходном коде или документации.
3. Анализ Результатов Тестирования:
    - Извлечение Данных: Регулярные выражения помогают извлекать нужные данные из результатов тестов. Например, можно использовать регулярные выражения для выделения определенных значений из вывода тестов или логов.
    - Пример: Извлечение числовых значений из текстовых отчетов с помощью паттернов, таких как \d+.
4. Подтверждение Текстовых Результатов:
    - Сравнение Результатов: Регулярные выражения используются для проверки, что текстовые результаты соответствуют ожидаемым паттернам. Это может включать проверку сообщений об ошибках, предупреждений и других текстов, которые возвращаются системой.
    - Пример: Проверка, что сообщение об ошибке содержит определенную текстовую строку или код ошибки.
5. Автоматизация Тестов:
    - Сценарии Тестирования: Регулярные выражения могут быть встроены в тестовые сценарии для динамической проверки входных данных и результатов. Это позволяет создавать более гибкие и мощные тесты, которые могут адаптироваться к различным форматам данных.
    - Пример: Использование регулярных выражений для проверки формата даты, который может меняться в зависимости от локализации.

Одна из любопытных особенностей регулярных выражений в их универсальности, стоит вам выучить синтаксис, и вы сможете применять их в любом языке программирования. Небольшие отличия касаются только наиболее продвинутых функций и версий синтаксиса, поддерживаемых движком.

Для тренировки мы будем использовать следующий ресурс: https://regex101.com/

Якоря — ^ и $

^Привет	Соответствует строке, начинающейся с Привет
пока$	Соответствует строке, заканчивающейся на пока
^Привет пока$	Точное совпадение (начинается и заканчивается как Привет пока)
воробушки	Соответствует любой строке, в которой есть текст воробушки

Квантификаторы — * + ? и {}

abc*	соответствует строке, в которой после ab следует 0 или более символов c
abc+	соответствует строке, в которой после ab следует один или более символов c
abc?	соответствует строке, в которой после ab следует 0 или один символ c
abc{2}	соответствует строке, в которой после ab следует 2 символа c
abc{2,}	соответствует строке, в которой после ab следует 2 или более символов c
abc{2,5}	соответствует строке, в которой после ab следует от 2 до 5 символов c
a(bc)*	соответствует строке, в которой после ab следует 0 или более последовательностей символов bc
a(bc){2,5}	соответствует строке, в которой после ab следует от 2 до 5 последовательностей символов bc

Оператор ИЛИ — | или []

a(b|c)	соответствует строке, в которой после a следует b или c
a[bc]	как и в предыдущем примере

Символьные классы — \d \w \s и .

\d	соответствует одному символу, который является цифрой
\w	соответствует слову (может состоять из букв, цифр и подчёркивания)
\s	соответствует символу пробела (включая табуляцию и прерывание строки)
.	соответствует любому символу

У операторов \d, \w и \s также есть отрицания — \D, \W и \S соответственно (\D соответствует одному символу, который не является цифрой).

Флаги

Регулярное выражение, как правило, записывается в такой форме /abc/, где шаблон для сопоставления выделен двумя слешами /. В конце выражения, мы определяем значение флага (эти значения можно комбинировать):

- g (global) — не возвращает результат после первого совпадения, а продолжает поиск с конца предыдущего совпадения.
- m (multi line) — с таким флагом, операторы ^ и $ вызовут совпадение в начале и конце строки ввода (line), вместо строки целиком (string).
- I (insensitive) — делает выражение регистронезависимым (например, **/aBc/i** соответствует **AbC**).

Скобочные группы — ()

a(bc)	создаём группу со значением bc
a(?:bc)*	оперетор ?: отключает группу

Преимущества Использования Регулярных Выражений в Тестировании

Гибкость: Регулярные выражения позволяют создавать универсальные шаблоны для поиска и проверки данных, что делает их полезными в различных тестовых сценариях.

Экономия Времени: Автоматизация задач поиска и замены с помощью регулярных выражений ускоряет тестирование и уменьшает необходимость в ручной проверке.

Сложные Проверки: Регулярные выражения позволяют выполнять сложные проверки форматов данных, которые трудно реализовать простыми строковыми операциями.

Регулярные выражения играют важную роль в тестировании программного обеспечения благодаря своей способности эффективно работать с текстовыми данными. Они позволяют автоматизировать и улучшить процесс тестирования, особенно при проверке форматов данных, анализе результатов и поиске ошибок.

Выполнение лабораторной работы

1. Перейдите на предоставленный преподавателем ресурс;
2. Скопируйте следующий текст в поле TEST STRING:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi eros justo, molestie vitae dui sed, auctor eleifend 5 nunc. Suspendisse potenti. Etiam tristique vel mauris consectetur tincidunt. Mauris a porta nibh. Duis 88 malesuada aliquam risus, eu venenatis tortor pulvinar id. Mauris ullamcorper arcu vel porttitor consectetur. Mauris at risus vel risus auctor 25 luctus. Sed quam diam, iaculis vel nunc sed, vehicula eleifend massa. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Class aptent taciti sociosqu ad 0 litora torquent per conubia nostra, per inceptos himenaeos. Maecenas ornare, nisi et dapibus aliquam, risus nibh tempus nisl, condimentum sodales dui felis 908 non metus. Vestibulum eu sollicitudin neque, ut condimentum nisi. Nunc ut imperdiet mauris, in ullamcorper ipsum.

3. Выполните следующие примеры:
    a. Найти все цифры: **\d+**
    b. Найти все слова начинающиеся с заглавной буквы: **[A-Z][a-z]***
    c. Найти все слова, которые начинаются на букву «L»: **(^|\s)([L,l][a-z\-0-9]*)**
    d. Найти все слова, заканчивающиеся на букву «r» или «m»: **\w+(m|r)($|\s)**
    e. Найти все слова, в которых встречаются две буквы «tt» подрят: **\w*tt\w***

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

4. Изучить представленный преподавателем материал и выполнить примеры.
5. Написать регулярные выражения для следующих заданий:
    a. Найти все числа в которых есть цифра вашего варианта;
    b. Найти все слова стоящие в конце предложения;
    c. Найти все слова, состоящие только из 3 букв;
    d. Написать регулярное выражение для даты в формате yyyy-MM-dd HH:mm;
    e. Написать регулярное выражение для e-mail.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

6. Цель работы;
7. Выполненные примеры;
8. Выполненные задания к работе;
9. Вывод.

ВОПРОСЫ

1. Что такое регулярное выражение?
2. Какие основные метасимволы используются в регулярных выражениях, и какое у них назначение?
3. Какое назначение у символа ^ в регулярном выражении?
4. Что обозначает символ $ в регулярных выражениях?
5. Какое регулярное выражение можно использовать для поиска всех вхождений слова "тест" в строке, игнорируя регистр?
6. Как с помощью регулярного выражения найти все числа в строке, которые состоят из 3 или более цифр?
7. Каким образом можно использовать регулярные выражения для замены всех пробелов в строке на подчеркивания?
8. Как работает квантификатор * в регулярных выражениях и в чем его отличие от квантификатора +?
9. Что такое "группировка" в регулярных выражениях и как она может быть использована?

10. Как можно использовать регулярные выражения для нахождения повторяющихся последовательностей символов в строке?
11. Как регулярные выражения могут помочь в тестировании веб-сервисов или API?
12. Каковы преимущества и ограничения использования регулярных выражений в тестировании программного обеспечения?

Лабораторная работа №6. Изучение Swagger

ЦЕЛЬ

Познакомиться с инструментом документирования API – Swagger. Написать тест-кейсы для тестирования API запросов, используя Swagger.

При проектировании современных программных систем часто встает задача согласования и разработки интерфейсов для взаимодействия их компонентов друг с другом.

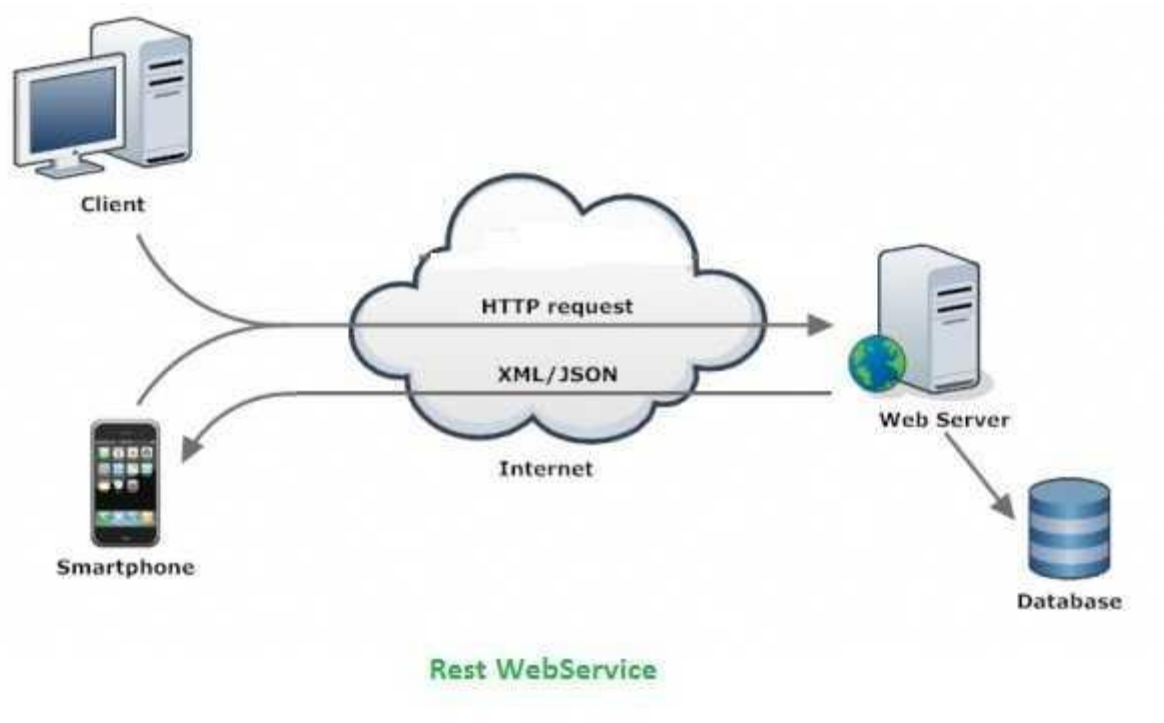
В последнее десятилетие огромную популярность и развитие получили приложения, взаимодействующие с сервером через API интерфейсы.

Один из самых популярных протоколов для обмена сообщениями — это REST. Основная идея данного подхода состоит в том, что клиентская часть приложения отправляет на url-адрес ресурса запросы определенного вида (GET, POST, PUT или DELETE) в формате JSON, на каждый запрос приходит ответ, содержащий запрашиваемые данные.

Виды запросов в архитектуре REST:

- GET – возвращает данные;
- POST – создает новые данные;
- PUT – редактирует данные;
- DELETE – удаляет данные.

Данный подход позволяет четко разграничить логику обработки клиентской части, от логики серверной части, что позволяет переносить разработанное приложение на другие платформы. Более того клиент-серверная архитектура минимизирует возможность полного отказа информационной системы и предоставляет возможность отладки или изменения одного звена без внесения изменений в другое звено.



Для описания REST API обычно используют Swagger. Данный инструмент является языконезависимым, что позволяет создавать документацию, которая воспринимается как человеком, так и машиной.

Вот несколько причин, почему тестировщики могут использовать Swagger:

1. Документирование API

- **Автоматическая генерация документации**: Swagger позволяет автоматически генерировать документацию для API на основе спецификаций OpenAPI. Это помогает тестировщикам и другим членам команды быстро понять, как работает API и какие endpoint доступны.

2. Тестирование API

- **Интерактивные тесты**: С помощью Swagger UI тестировщики могут интерактивно отправлять запросы к API и получать ответы прямо в веб-интерфейсе. Это удобно для проверки того, как API реагирует на различные запросы и входные данные.
- **Проверка параметров и ответов**: Тестировщики могут использовать Swagger для проверки корректности параметров запроса и ответов от API, включая статус-коды, заголовки и тело ответа.

3. Упрощение взаимодействия с API

- **Простота в использовании**: Swagger UI предоставляет удобный интерфейс для тестирования и взаимодействия с API, что упрощает процесс тестирования и позволяет легко обнаружить ошибки или проблемы с API.

- **Быстрая проверка изменений**: Когда API обновляется или меняется, Swagger позволяет быстро проверить изменения и убедиться, что новые функции работают как ожидается.

4. Создание и поддержка тестов

- **Проверка соответствия спецификациям**: Swagger позволяет тестировщикам убедиться, что API соответствует спецификациям, предоставленным в документации. Это особенно важно для обеспечения того, чтобы изменения в API не нарушили существующую функциональность.
- **Автоматизация тестирования**: Инструменты на базе Swagger могут использоваться для автоматизации тестов API, что позволяет регулярно проверять, что API работает корректно после изменений или обновлений.

5. Снижение рисков

- **Поиск ошибок**: Тестировщики могут использовать Swagger для поиска ошибок в API, таких как некорректные параметры или неверные ответы, что помогает снизить риски перед запуском API в производственную среду.
- **Верификация безопасности**: Swagger помогает проверить, что API правильно обрабатывает входные данные и не имеет уязвимостей, которые могут быть использованы для атак.

6. Совместная работа

- **Обратная связь**: Swagger позволяет тестировщикам и разработчикам обмениваться обратной связью о том, как API работает и как его можно улучшить. Документация, созданная с помощью Swagger, может служить общим источником информации для всей команды.
- **Упрощение интеграции**: Тестировщики могут использовать Swagger для проверки интеграции между различными системами, взаимодействующими через API.

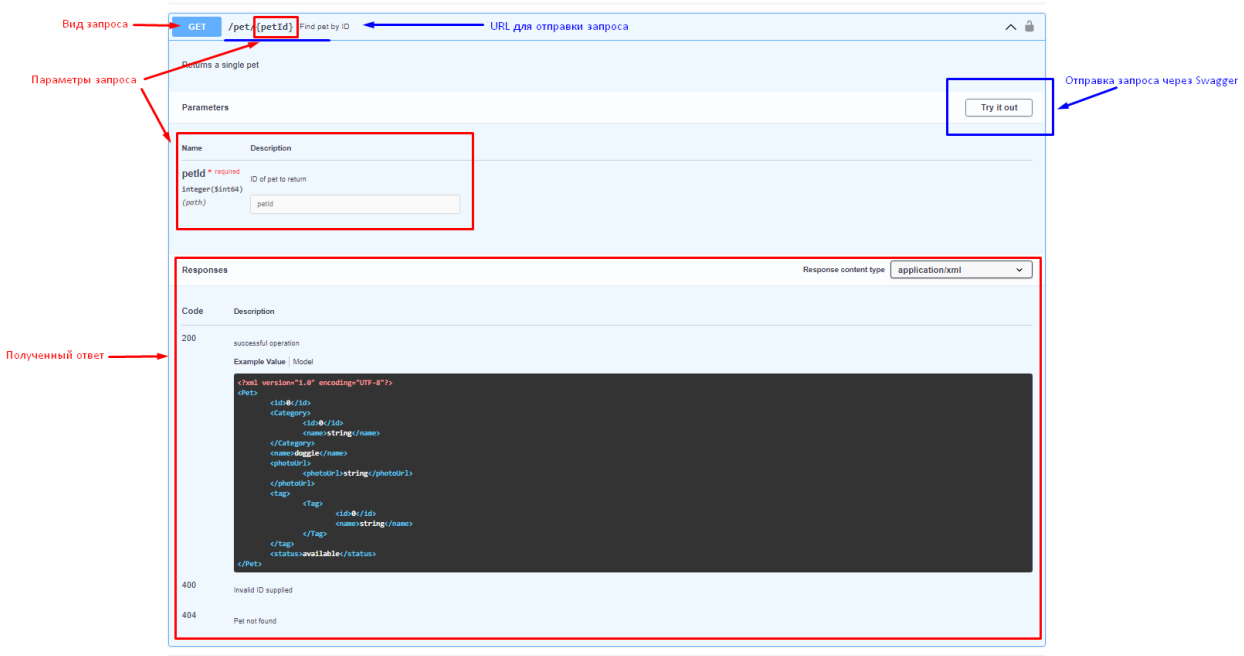
7. Образование и обучение

- **Учебные материалы**: Swagger может быть полезен для обучения новым членам команды или для создания учебных материалов о том, как работать с API и как их тестировать.

8. Снижение затрат на поддержку

- **Упрощение отладки**: С помощью Swagger тестировщики могут быстрее выявлять и исправлять ошибки, что снижает затраты на поддержку и улучшает качество API.

Использование Swagger позволяет тестировщикам более эффективно работать с API, снижать количество ошибок и упрощать процесс тестирования и документирования.



ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить предложенный преподавателем материал и стандартную документацию - Swagger Petstore (https://editor.swagger.io/#);
2. Написать минимум 5 тест-кейсов для тестирования всех видов запросов;
3. Выполнить запросы в Swagger согласно вашему варианту;
4. Составить отчет.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Цель работы;
2. Тест кейсы;
3. Скриншоты выполнения запросов в Swagger;
4. Выводы по лабораторной работе.

ВАРИАНТЫ

1	PUT /pet POST /pet GET /pet/findByStatus
2	GET /pet/findByTags

	GET /pet/{petId} POST /pet/{petId}/uploadImage
3	PUT /store/inventory POST /store/order GET /store/order/{orderId}
4	DELETE /store/order/{orderId} POST /user GET /user/login
5	GET /user/login GET /user/logout PUT /user/{username}
6	DELETE /user/{username} PUT /user/{username} GET /user/login
7	PUT /pet POST /user GET /pet/findByTags
8	PUT /pet POST /pet GET /pet/{petId}
9	PUT /store/inventory POST /store/order PUT /user/{username}
10	PUT /store/inventory PUT /user/{username} GET /store/order/{orderId}
11	PUT /user/{username} POST /pet GET /pet/findByStatus
12	PUT /pet POST /user GET /pet/{petId}
13	DELETE /user/{username} POST /user

	GET /user/login
14	PUT /pet POST /pet GET /pet/findByStatus
15	GET /pet/findByTags GET /pet/{petId} POST /pet/{petId}/uploadImage
16	PUT /store/inventory POST /store/order GET /store/order/{orderId}
17	DELETE /store/order/{orderId} POST /user GET /user/login
18	GET /user/login GET /user/logout PUT /user/{username}
19	DELETE /user/{username} PUT /user/{username} GET /user/login
20	PUT /pet POST /user GET /pet/findByTags
21	PUT /pet POST /pet GET /pet/{petId}
22	PUT /store/inventory POST /store/order PUT /user/{username}
23	PUT /store/inventory PUT /user/{username} GET /store/order/{orderId}
24	PUT /user/{username} POST /pet GET /pet/findByStatus

25	PUT /pet POST /user GET /pet/{petId}
26	DELETE /user/{username} POST /user GET /user/login
27	PUT /pet POST /pet GET /pet/findByStatus
28	GET /pet/findByTags GET /pet/{petId} POST /pet/{petId}/uploadImage
29	PUT /store/inventory POST /store/order GET /store/order/{orderId}
30	DELETE /store/order/{orderId} POST /user GET /user/login

ВОПРОСЫ

1. Что такое Swagger (OpenAPI)?
2. Какую роль играет Swagger UI в тестировании API?
3. Какой формат используется для описания API в Swagger?
4. Как в Swagger описываются эндпоинты API?
5. Как описать ответы API в Swagger?
6. Как можно использовать Swagger для тестирования API?
7. Как Swagger помогает в обеспечении качества API и в управлении изменениями?
8. Как можно использовать Swagger для создания документации API?

Лабораторная работа №7. Изучение инструмента тестирования REST API – Postman

ЦЕЛЬ

Познакомиться с инструментом тестирования REST API - Postman.

Инструмент Postman предназначен для проверки запросов с клиента на сервер и получения ответа от бэкенда. Можно описать общение сервера с Postman в виде диалога:

> ***Postman**: "Дай мне информацию по балансу именно этого пользователя".*
>
> ***Сервер**: "Да, конечно, запрос правильный, получи информацию по балансу этого пользователя".*

Такой позитивный диалог происходит в том случае, если ошибок на сервере нет, и разработчик сделал всё согласно документации. Но не всегда это происходит в таком успешном ключе. Очень часто встречаются следующие диалоги:

> ***Postman**: "Дай мне информацию по балансу именно этого пользователя"*
>
> ***Backend**: "Кто я вообще?"*

или

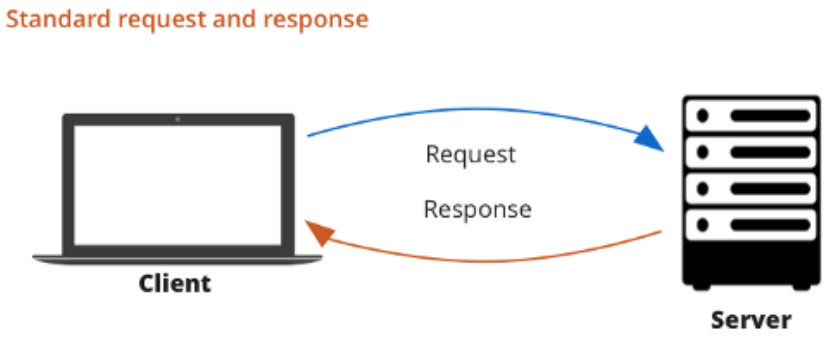
> ***Postman**: "Дай мне информацию по балансу именно этого пользователя"*
>
> ***Backend**: "Пользователь не найден."*

Описанные выше ответы от сервера имеют свой код ошибки, которые приходят в ответе.

В первом случае — это ошибка с кодом **500** *(Internal Server Error)* внутренняя ошибка сервера, которая говорит о том, что сервер столкнулся с неожиданным условием, которое помешало ему выполнить запрос.

Во втором — **404** ошибка *(Not Found)* код ответа HTTP о том, что сервер не может найти данные по запросу, полученному от клиента.

Именно для этого и предназначен Postman — для проверки запросов клиент → сервер по документации, чтобы убедиться, что всё корректно работает на стороне сервера.



JSON (JavaScript Object Notation) — представляет собой текстовый формат обмена данными, основанный на JavaScript. Формат считается независимым от языка и может использоваться практически с любым языком программирования. Важно отметить, что

JSON это лишь один из возможных форматов, в качестве примера можно привести такой формат как SOAP. Пример JSON-сообщения:

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": "101101"
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
}
```

Обычные GET запросы можно посылать при помощи браузера. Но существует множество специальных инструментов, которые предназначены для разработки и тестирования REST API. Они предоставляют возможность не только отправлять различные типы запросов, но и сохранять запросы, показывать результаты в различных форматах, выступать в роли прокси-сервера. Среди таких инструментов:

- **Postman.** Расширение для Google Chrome, которое в бесплатной версии позволяет посылать запросы, записывать их, показывать историю. Удобно и понятно.
- **jMeter.** Инструмент, получивший известность прежде всего благодаря нагрузочному тестированию, которое можно проводить с его помощью.
- **Fiddler.** Позволяет просматривать посылаемые HTTP запросы. И много чего еще.
- **SoapUI.** Мощный продукт для разработки и тестирования веб приложений. Сам я его не использовал, но слышал много отзывов - и хороших, и плохих.

Для того чтобы освоить тестирование REST API достаточно выполнить несколько шагов.

Работа со средствами разработчика в браузере. После совершения обычных действий на сайте будет отображаться множество запросов, которые будут отвечать за каждое из совершенных действий. Далее необходимо исключить запросы, не относящиеся к логике. Для этого можно использовать различные текстовые фильтры. После детального анализа можно сделать заключения о том, что подразумевают в себе эти запросы. Можно попробовать самостоятельно воспроизвести такие запросы и проверить поведение.

Ниже приведен примерный перечень тех тестов, которые необходимо выполнить при тестировании API:

- Обзорное тестирование – набор вызовов для проверки общей работоспособности системы;
- Проверка граничных значений
- Проверка типов полей запроса/ответа
- Минимальный запрос
- Максимальный запрос
- Минимальный ответ
- Максимальный ответ
- Корректный запрос
- Корректный ответ
- Некорректный запрос
- Некорректный ответ
- Успешный код ошибки
    - Неуспешный код ошибки (чем регламентируется?)
    - Не возвращается обязательное поле
    - Возвращается лишнее поле

Postman — это мощный инструмент, который широко используется для тестирования API. Вот несколько основных причин, почему Postman так полезен для тестирования:

1. Интерактивное Тестирование API

- Отправка Запросов: Postman позволяет отправлять запросы к API, используя различные методы (GET, POST, PUT, DELETE и другие). Это позволяет тестировщикам легко проверять ответы API.
- Просмотр Ответов: Вы можете просматривать ответы от сервера в различных форматах, включая JSON, XML, HTML и текст, что помогает анализировать, как API обрабатывает запросы.

2. Упрощение Работы с Запросами

- Конфигурация Запросов: Postman предоставляет удобный интерфейс для настройки заголовков, параметров, тела запроса и аутентификации. Это упрощает процесс создания и отправки запросов.
- Сохранение Запросов: Запросы можно сохранять в коллекции, что позволяет легко повторно использовать их или делиться с другими членами команды.

3. Автоматизация Тестирования

- Тестовые Скрипты: Postman позволяет писать тестовые скрипты на JavaScript для проверки ответа API. Это позволяет автоматизировать процесс тестирования и проверять корректность API на основе заданных критериев.
- Сборка и Запуск Коллекций: Используя Collection Runner, можно запускать тесты для всей коллекции запросов и получать отчет о результатах выполнения.

4. Управление Средами
    - Среды и Переменные: Postman поддерживает использование переменных и сред (environments), что позволяет тестировать API в различных конфигурациях и с разными наборами данных, без необходимости вручную изменять параметры запросов.
5. Документация и Обратная Связь
    - Генерация Документации: Postman позволяет создавать и публиковать документацию для API, что упрощает общение с другими разработчиками и потребителями API.
    - Обратная Связь: Тестировщики могут оставлять комментарии и заметки прямо в запросах и коллекциях, что упрощает совместную работу и обмен обратной связью.
6. Интеграция с CI/CD
    - Интеграция с Платформами CI/CD: Postman может быть интегрирован с системами непрерывной интеграции и доставки (CI/CD), такими как Jenkins и GitHub Actions, что позволяет автоматизировать тестирование API в рамках сборки и развертывания.
7. Проверка Безопасности и Производительности
    - Проверка Аутентификации и Авторизации: Postman позволяет тестировать различные схемы аутентификации и авторизации, такие как OAuth, Basic Auth, и другие.
    - Тестирование Параметров Производительности: Вы можете использовать Postman для проверки различных параметров и нагрузки на API, что помогает в выявлении проблем с производительностью.
8. Упрощение Тестирования Разработки и Интеграции
    - Тестирование Во Время Разработки: Разработчики могут использовать Postman для тестирования API в процессе разработки, что позволяет быстрее находить и исправлять ошибки.

- Интеграция с другими Инструментами: Postman интегрируется с инструментами управления проектами и отчетности, такими как JIRA, что упрощает отслеживание дефектов и задач.

9. Проверка Соответствия Спецификациям

- Импорт и Экспорт Спецификаций: Postman поддерживает импорт и экспорт спецификаций API, таких как OpenAPI и RAML, что помогает в тестировании API, соответствующих заданным спецификациям.

Примеры Конкретных Использований

- Тестирование RESTful API: Создание и выполнение запросов для проверки функциональности различных эндпоинтов.
- Проверка Параметров Запроса: Верификация корректности обработки различных параметров и данных.
- Анализ Ответов: Проверка структуры и содержания ответов от API, включая статус-коды, заголовки и тело ответа.
- Автоматизация Регрессионного Тестирования: Запуск наборов тестов для проверки изменений в API и обеспечения его корректной работы после обновлений.

Postman является мощным и универсальным инструментом для тестирования API, который помогает тестировщикам и разработчикам быстро и эффективно проверять функциональность, производительность и безопасность API, обеспечивая высокий уровень уверенности в их качестве и надежности.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

При помощи postman необходимо протестировать REST API мобильного клиента портала befutsal.ru. Для этого необходимо настроить доступ и выполнить несколько запросов на основе соответствующей спецификации.

1. Зайдите на http://editor.swagger.io/# и скопируйте в левую часть веб-сайта предложенные преподавателем код swagger.

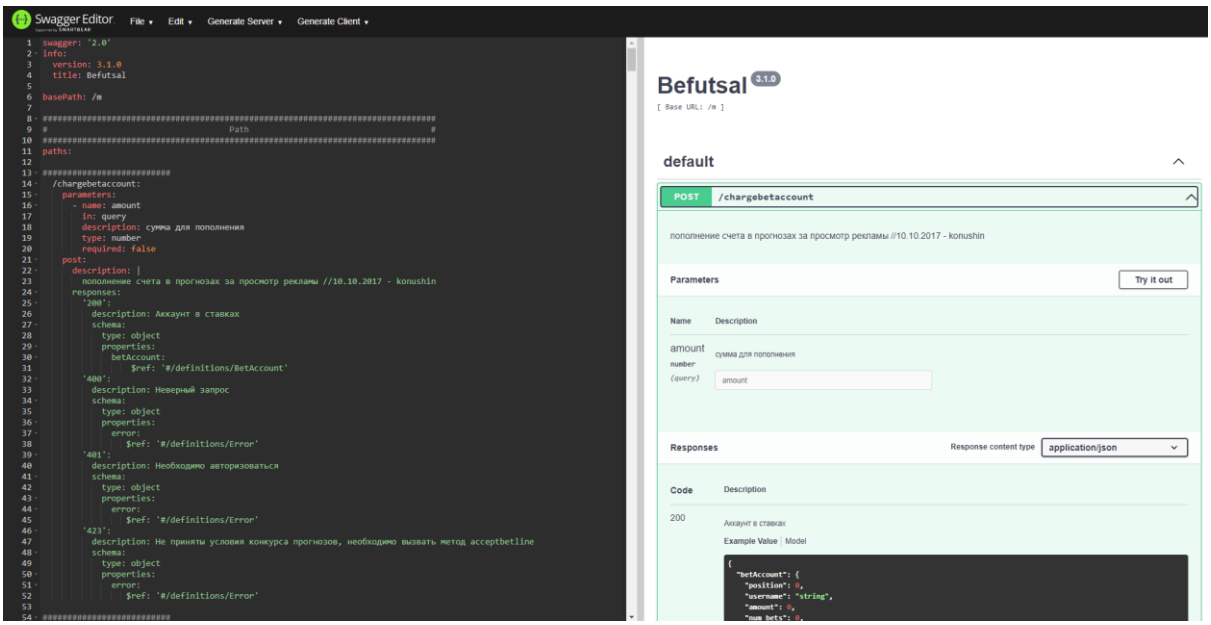


Рисунок 2 - Настройка swagger.

2. Открыть postman, создать проект и новое рабочее пространство. Затем в рабочем пространстве создайте новый запрос.

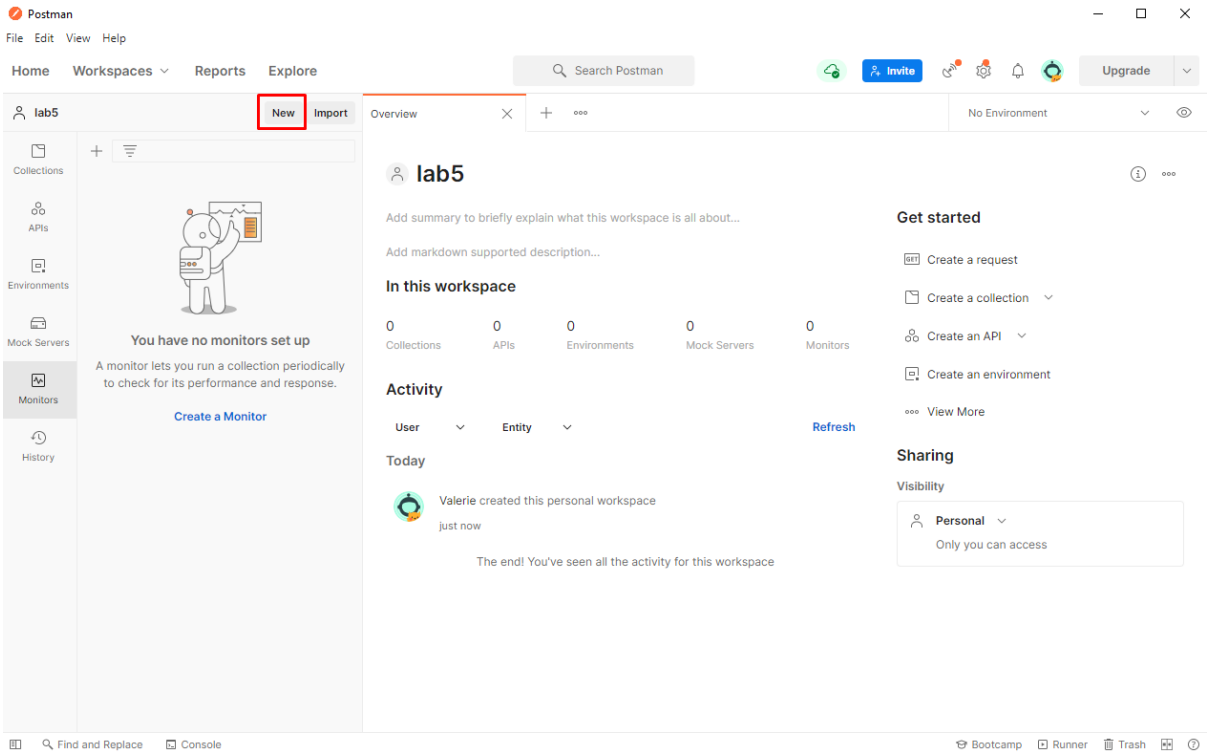


Рисунок 3 - Создание нового запроса.

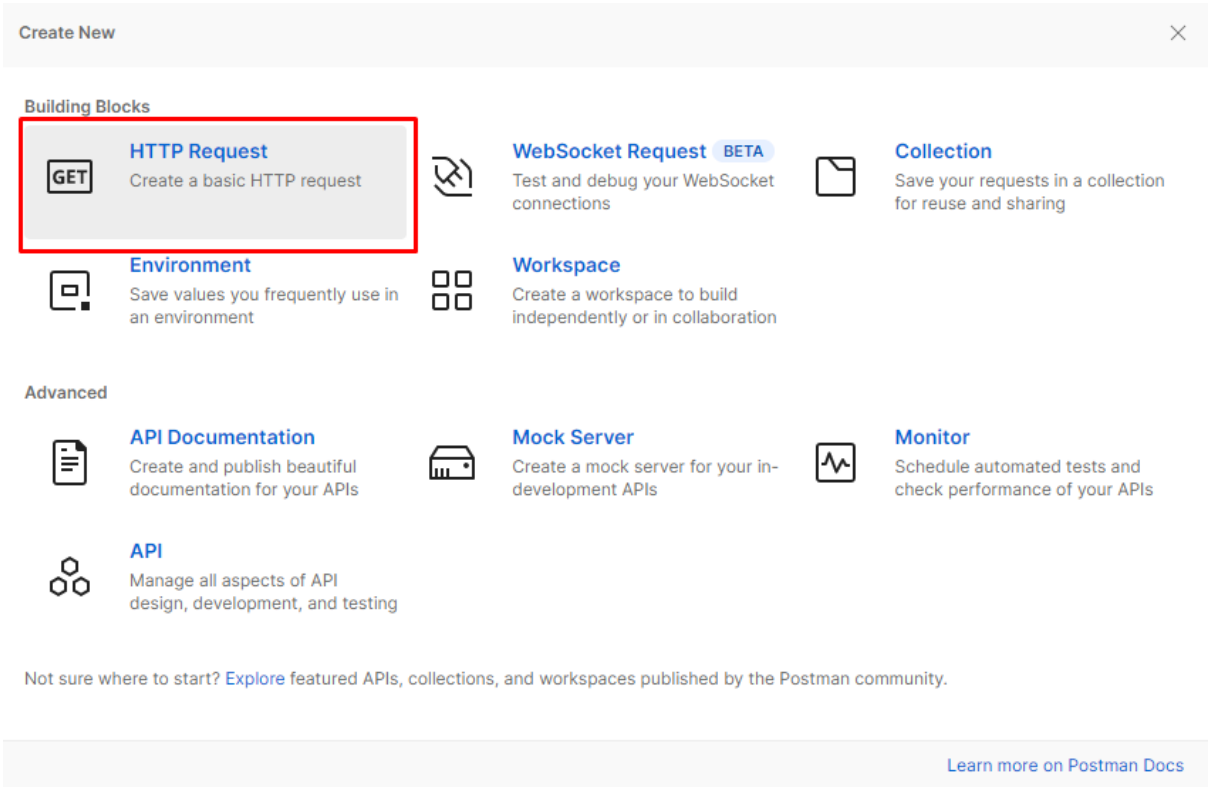


Рисунок 4 - Создание нового запроса.

3. Используя swagger, выбираем любой запрос и выполняем его, используя инструмент Postman. Первым запросом разумно выбрать запрос на регистрацию. После выполнении запроса сохраните его. Так вы сможете сохранить тестовые данные, и переиспользовать их в будущем. Для регистрации можно использовать сервис временной почты, например: https://temp-mail.org/ru/

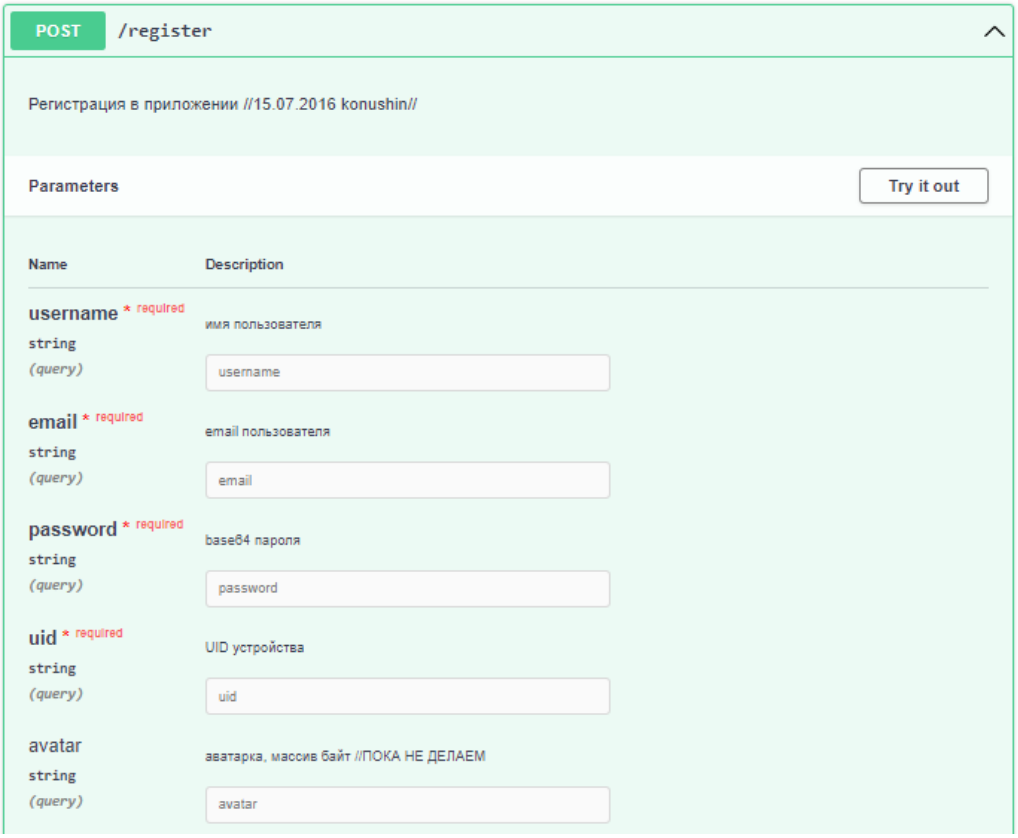


Рисунок 5 - Описание запроса в Swagger.

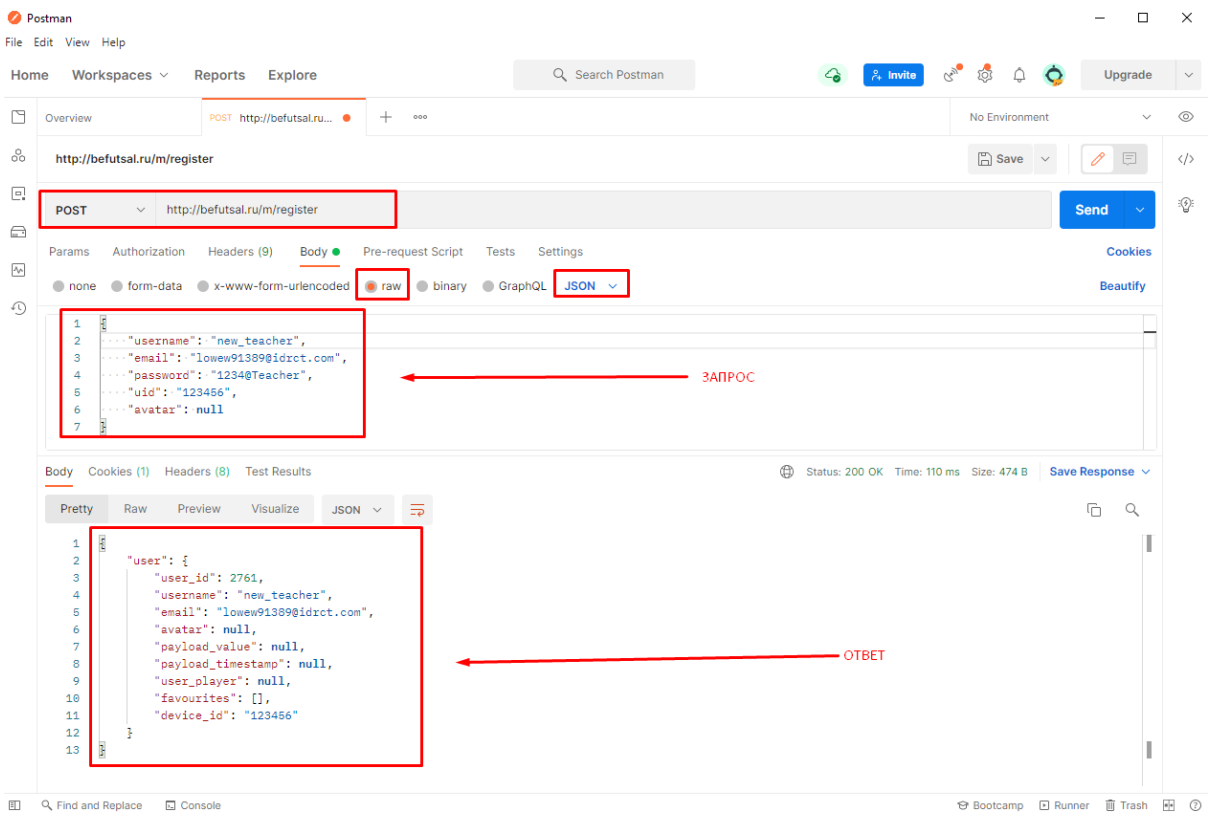


Рисунок 6 - Выполнение запроса.

4. После регистрации или авторизации, можно выполнять запросы к системе.

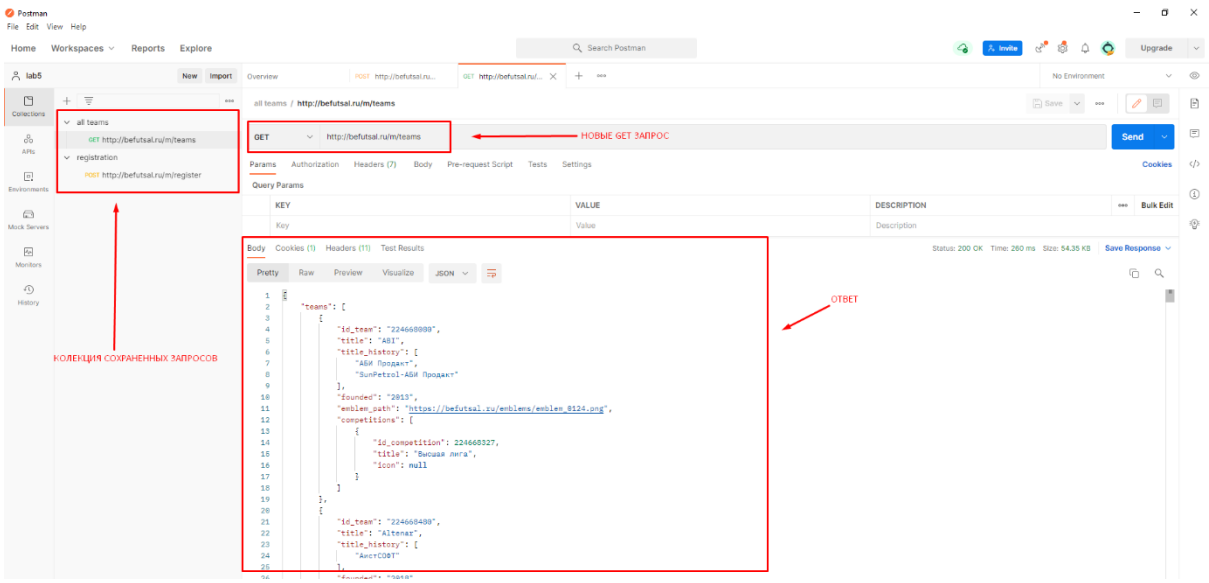


Рисунок 7 - Выполнение GET запроса.

ЗАДАНИЕ

1 Изучить спецификацию API портала;

2 Составить тест кейсы и чек лист по одному сценарию в соответствии с вашим вариантом;

3 Подготовить тестовые данные (по необходимости);

4 Провести тестирование;

5 Проанализировать результат.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Титульный лист;
2. Цель;
3. Выбранный сценарий;
4. Ход работы;
    a. Чек лист;
    b. Тест кейсы;
    c. Примеры запроса по заданию;
    d. Примеры ответа по заданию;
    e. Результаты тестирования;
5. Вывод.

ВАРИАНТЫ

№	Задание

1	Зарегистрироваться, вспомнить пароль, войти с новым паролем
2	Зарегистрироваться, получить список всех команд, добавить в избранное 2 команды;
3	Войти, получить список избранных команд, удалить из избранного 1 команду;
4	Войти, получить информацию о контактах федераций;
5	Войти, получить список всех команд, получить результаты игр любой команды;
6	Войти, получить список всех команд, получить турнирную таблицу любой команды;
7	Войти, получить список всех команд, получить состав любой команды;
8	Войти, получить список всех команд, получить список предстоящих игр для любой команды;
9	Зарегистрироваться, вспомнить пароль, войти с новым паролем
10	Зарегистрироваться, получить список всех команд, добавить в избранное 2 команды;
11	Войти, получить список избранных команд, удалить из избранного 1 команду;
12	Войти, получить информацию о контактах федераций;
13	Войти, получить список всех команд, получить результаты игр любой команды;
14	Войти, получить список всех команд, получить турнирную таблицу любой команды;
15	Войти, получить список всех команд, получить состав любой команды;
16	Войти, получить список всех команд, получить список предстоящих игр для любой команды;
17	Зарегистрироваться, вспомнить пароль, войти с новым паролем
18	Зарегистрироваться, получить список всех команд, добавить в избранное 2 команды;
19	Войти, получить список избранных команд, удалить из избранного 1 команду;
20	Войти, получить информацию о контактах федераций;
21	Войти, получить список всех команд, получить результаты игр любой команды;
22	Войти, получить список всех команд, получить турнирную таблицу любой команды;
23	Войти, получить список всех команд, получить состав любой команды;
24	Войти, получить список всех команд, получить список предстоящих игр для любой команды;
25	Зарегистрироваться, вспомнить пароль, войти с новым паролем

26	Зарегистрироваться, получить список всех команд, добавить в избранное 2 команды;
27	Войти, получить список избранных команд, удалить из избранного 1 команду;
28	Войти, получить информацию о контактах федераций;
29	Войти, получить список всех команд, получить результаты игр любой команды;
30	Войти, получить список всех команд, получить турнирную таблицу любой команды;
31	Войти, получить список всех команд, получить состав любой команды;
32	Войти, получить список всех команд, получить список предстоящих игр для любой команды;

ВОПРОСЫ

1. Что такое Postman и для чего он используется?
2. Как создать новый запрос в Postman?
3. Какие методы HTTP поддерживает Postman?
4. Как сохранить запрос в коллекцию Postman?
6. Что такое Collection Runner в Postman и как его использовать?
7. Как использовать переменные в Postman для параметризации запросов?
8. Как экспортировать и импортировать коллекции и окружения в Postman?
9. Как создать документацию для API в Postman?
10. Опишите, как бы вы использовали Postman для тестирования RESTful API в условиях разработки.
11. Как вы бы протестировали API, которое требует взаимодействия с несколькими эндпоинтами?

Лабораторная работа №8. Тестирование базы данных

ЦЕЛЬ

Изучить язык запросов к базе данных - SQL. Протестировать запросы к базе используя полученные знания.

База данных — специальная структура для хранения информации, организации и обмена большого объема данных. Они используются в разных сферах деятельности ИТ-сообществ: начиная от простых веб-магазинов и заканчивая центрами управления космическими полетами.

Для создания, применения и управления БД был создан специальный комплекс приложений и системных средств, который именуется системой управления базами данных (СУБД).

СУБД — особая совокупность программ, которые применяются для разработки, отображения, заполнения и модифицирования информации.

Далее детально рассмотрим некоторые характерные особенности, на которые необходимо обращать внимание при тестировании баз данных.

1. **Отображение информации:** тестировщик должен проверять сопоставление полей в форме пользовательского интерфейса с аналогичными полями внутри БД.
2. **Обновление записи в таблице:** каждый раз, когда определенное действие будет выполняться во внешнем интерфейсе ПО, соответствующее действие должно вызываться во внутреннем интерфейсе. Для теста обновления записи в таблице, проверяющему необходимо руководствоваться специальной мнемоникой CRUD C (Create) — когда пользователь сохраняет любую операцию, применяется операция INSERT, R (Read) — когда человек может просматривать или выполнять поиск сохраненной операции, выполняется операция SELECT, U (Update) — когда пользователь редактирует существующую запись, в базе данных автоматически выполняется операция Update, D (Delete) — когда человек удаляет запись из БД, выполняется операция Delete;
3. **Сохранность и целостность данных:** вся информация, а также ее модификация, должны отображаться одинаково и в понятной для пользователя форме. Необходимо проверить, чтобы система демонстрировала крайние изменения в таблицах идентично во всех местах, где они отображаются.

Проверку работоспособности БД можно поделить на 3 вида, в зависимости от параметров и структуры БД:

- **Структурная проверка БД** основывается на тестировании таблиц, столбцов, схем БД, а также на тестировании хранимых процедур и (или) триггеров;
- **Функциональные проверки БД** — это проверка работоспособности БД с точки зрения рядового пользовтеля. Можно выделить сразу 2 вида функциональных проверок — тестирование белого и черного ящиков;
- **Нефункциональные проверки БД** — это проверка производительности БД, от нагрузочных тестов, тестирования рисков до стресс-тестирования, а также аналитики минимальных требований системы.

Таблица 2 - Основные SQL команды.

Команда	**Обозначение**
SHOW DATABASES	SQL-команда, которая отвечает за просмотр доступных баз данных.
CREATE DATABASE	Команда для создания новой базы данных.
USE	С помощью этой SQL-команды USE <database_name> выбирается база данных, необходимая для дальнейшей работы с ней.
DROP DATABASE	Стандартная SQL-команда для удаления целой базы данных.
SHOW TABLES	С помощью этой несложной команды можно увидеть все таблицы, которые доступны в базе данных.
CREATE TABLE	SQL-команда для создания новой таблицы: CREATE TABLE <table_name1> (<col_name1><col_type1>, <col_name2><col_type2>, <col_name3><col_type3> PRIMARY KEY(<col_name1>), FOREIGN KEY(<col_name2>) REFERENCES <table_name2>(<col_name2>));
INSERT	Команда INSERT INTO <table_name> в SQL отвечает за добавление данных в таблицу: INSERT INTO <table_name> (<col_name1>, <col_name2>, <col_name3>, …) VALUES (<value1>, <value2>, <value3>, …); При добавлении данных в каждый столбец таблицы не требуется указывать названия столбцов. INSERT INTO <table_name> VALUES (<value1>, <value2>, <value3>, …);
UPDATE	SQL-команда для обновления данных таблицы: UPDATE <table_name> SET <col_name1> = <value1>, <col_name2> = <value2>, ... WHERE <condition>;
DELETE	SQL-команда DELETE FROM <table_name> используется для удаления данных из таблицы.
DROP TABLE	А так можно удалить всю таблицу целиком.
SELECT	Получения данных из выбранной таблицы: SELECT <col_name1>, <col_name2>, … FROM <table_name>; Следующей командой можно вывести все данные из таблицы: SELECT * FROM <table_name>;
WHERE	Можно использовать ключевое слово WHERE в SELECT для указания условий в запросе: SELECT <col_name1>, <col_name2>, … FROM <table_name> WHERE <condition>; В запросе можно задавать следующие условия: сравнение текста; сравнение численных значений; логические операции AND (и), OR (или) и NOT (отрицание).
GROUP BY	Оператор GROUP BY часто используется с агрегатными

	функциями, такими как COUNT, MAX, MIN, SUM и AVG, для группировки выходных значений. SELECT <col_name1>, <col_name2>, … FROM <table_name> GROUP BY <col_namex>;
ORDER BY	ORDER BY используется для сортировки результатов запроса по убыванию или возрастанию. ORDER BY отсортирует по возрастанию, если не будет указан способ сортировки ASC или DESC. SELECT <col_name1>, <col_name2>, … FROM <table_name> ORDER BY <col_name1>, <col_name2>, … ASC|DESC;

Ход выполнения работы

Для выполнения данной лабораторной работы мы будем использовать онлайн-редактора SQL:

https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all

При выполнении работы нам доступны следующие таблицы:

- Customers
- Categories
- Employees
- OrderDetails
- Orders
- Products
- Shippers
- Suppliers

Самый простой запрос, чтобы выбрать всю информацию о всех заказчиков:

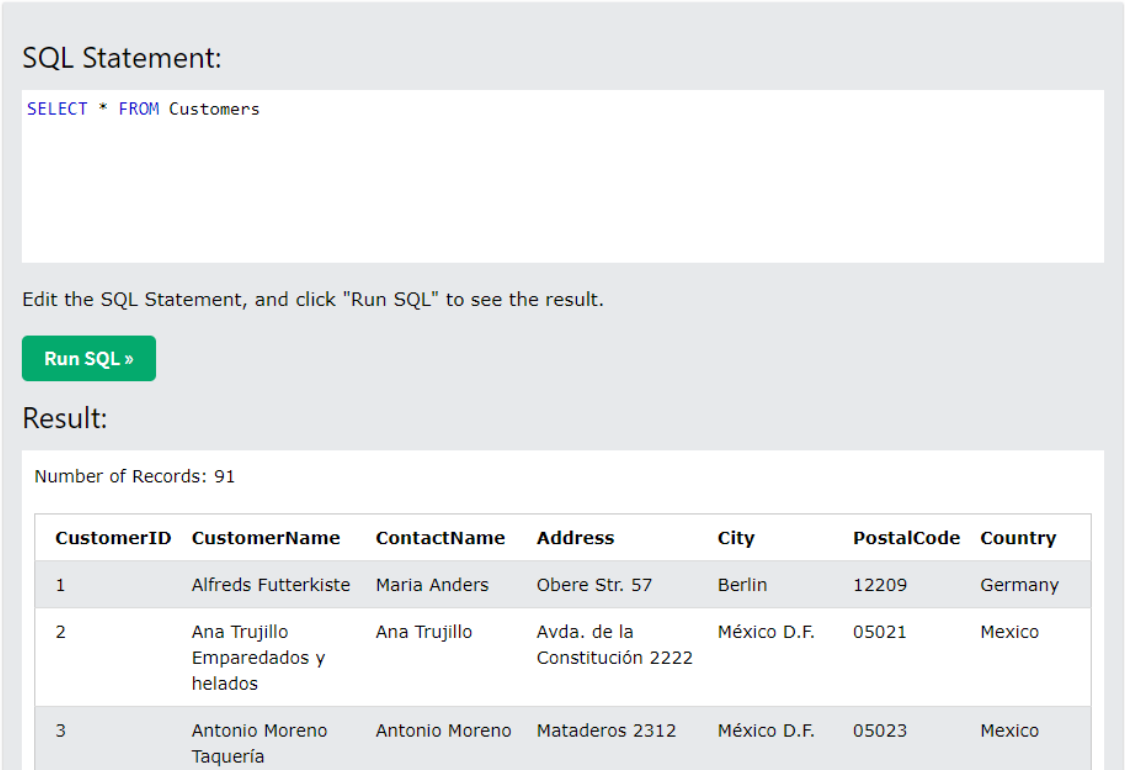


Рисунок 8 - Выбор всех данных из таблицы.

Чтобы выбрать заказчика, у которого PostalCode равен определенному значению, используем команду WHERE.

Рисунок 9 - Выбор данных по условию.

Попробуйте выполнить следующие команды на добавление и обновление данных в таблице.

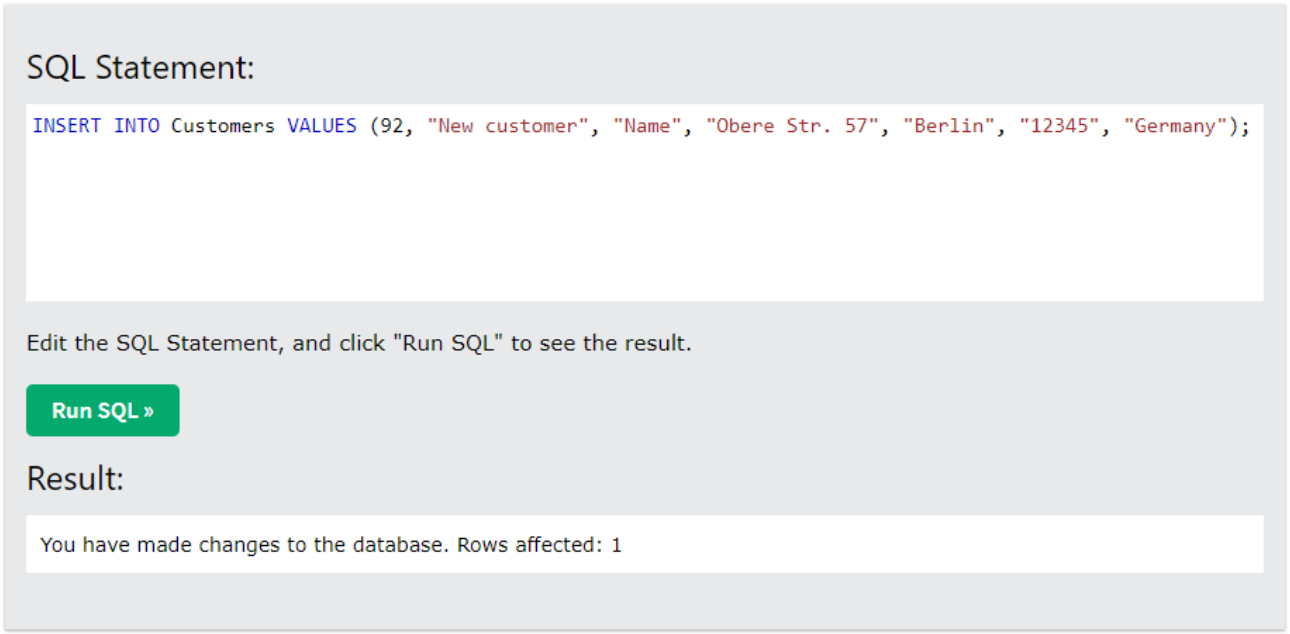


Рисунок 10 - Добавление данных.

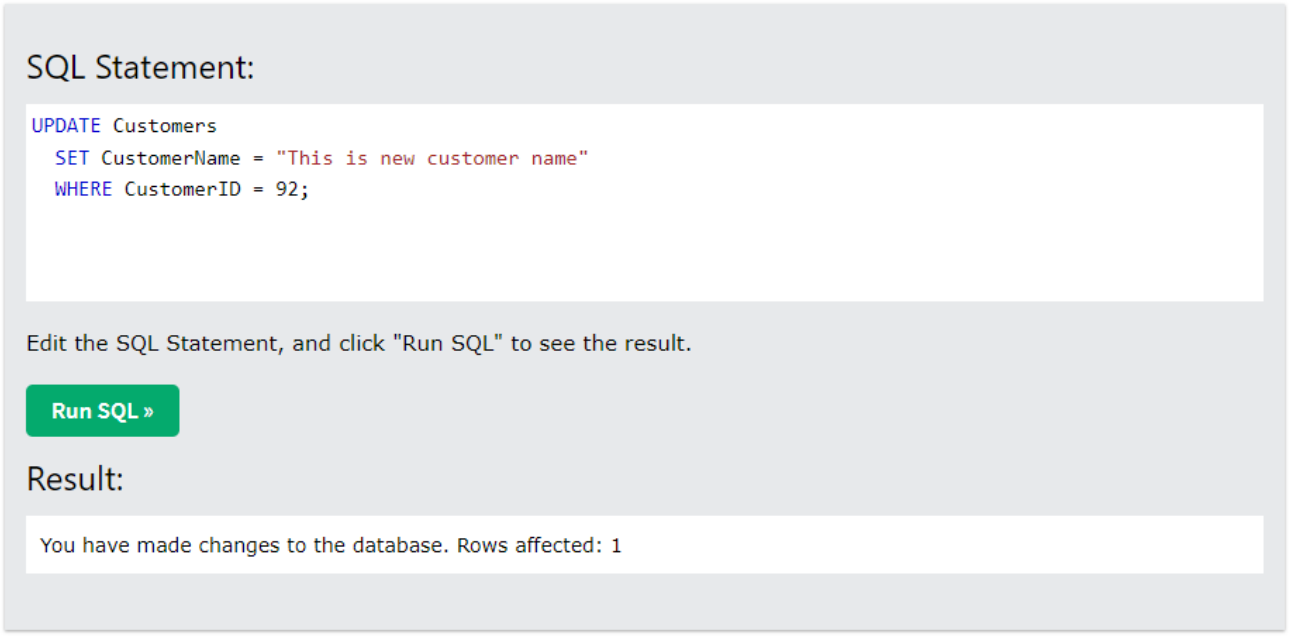


Рисунок 11 - Обновление данных.

Чаще всего используются сложные операции с объединением нескольких таблиц. При объединении таблиц используют команду JOIN. Рассмотрим небольшой пример: нам нужно узнать какие заказы выполнил заказчик Hanari Carnes и в какие даты.

Для этого нам нужно объединить таблицы Orders и Customers, и выбрать только заказы конкретного заказчика (WHERE CustomerName = "Hanari Carnes").

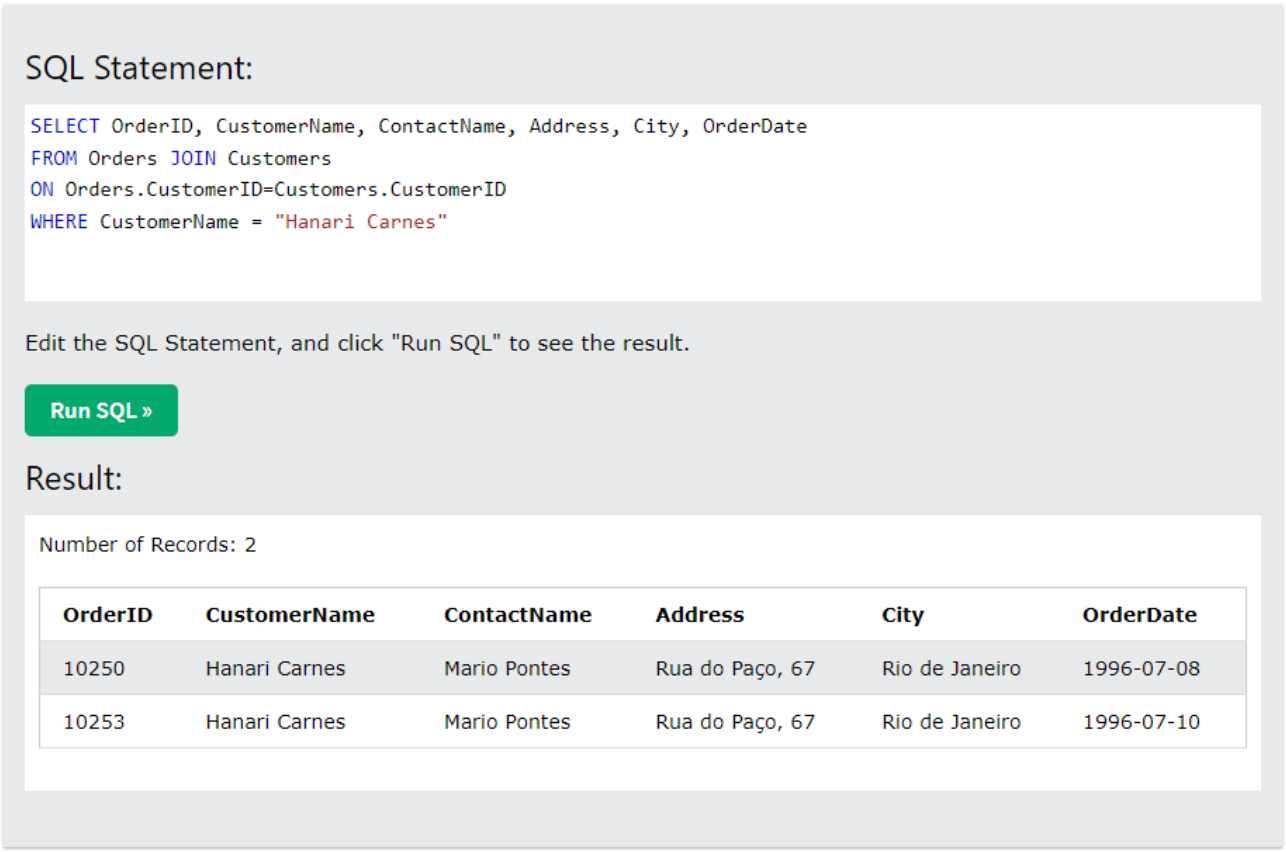


Рисунок 12 - Пример запроса с объединением нескольких таблиц.

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Изучите представленный преподавателем материал;
2. Напишите запросы для всех CRUD операций, в качестве источника данных используйте таблицы в соответствии с номером вашего варианта;
3. Напишите минимум два запроса, которые бы включали объединение двух таблиц (хотя бы одна из таблиц при объединении должна быть из вашего варианта), сортировку или условие выбора;
4. Выполните запросы в онлайн-редакторе.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Цель работы;
2. Вариант задания (соответствует вашему номеру в списке группы);
3. Выполненные запросы, скриншоты;
4. Вывод по работе.

ВОПРОСЫ

1. Что такое реляционная база данных и как она отличается от нереляционной базы данных?
2. Что такое SQL и какие основные команды SQL вы знаете?
3. Какие виды тестирования базы данных вы знаете?
4. Как вы проверяете целостность данных в базе данных?
5. Что такое тестирование производительности базы данных и как его проводить?
6. Как вы можете объединить данные из двух таблиц, используя SQL?
7. Как вы проверяете, что базы данных защищены от SQL-инъекций?

Лабораторная работа №9. Selenium IDE как инструмент автоматизации тестирования

ЦЕЛЬ РАБОТЫ

Изучить принципы работы со средством автоматизированного тестирования Selenium IDE.

Selenium IDE — это инструмент для автоматизации действий веб-браузера. В большинстве случаев используется для тестирования Web-приложений, но этим не ограничивается. В частности, он может быть использован для решения рутинных задач администрирования сайта.

Он записывает определенный сценарий поведения пользователя на сайте, а потом воспроизводит записанные действия в автоматическом режиме.

Selenium IDE также обладает более 500-ми встроенными командами, которые позволяют зафиксировать практически любые ваши действия на сайте и воспроизвести их вновь или осуществить проверку какого-то элемента.

Например, как узнать, что в вашем интернет-магазине исправно работает оформления заказа? Нужно зайти в карточку какого-либо товара, положить его в корзину, перейти на страницу оформления заказа, заполнить данные заказа (фио, телефон, адрес, e-mail и тд), нажать кнопку "Оформить заказ", убедиться, что заказ оформлен.

В следующий раз, чтоб проверить тот же сценарий, вам придется проделать эти действия снова. Но на самом деле, вам можно сделать это всего один раз, а Selenium IDE даст возможность многократно повторить эти действия в автоматическом режиме.

Тестировщики, которые не хотят программировать, используют Selenium IDE как самостоятельный продукт, без преобразования записанных сценариев в программный код. Это, конечно, не позволяет разрабатывать достаточно сложные тестовые наборы, но часто

хватает и простых линейных сценариев. Вот несколько ключевых аспектов, зачем он нужен в тестировании:

1. **Простота Использования**:
    - **Графический Интерфейс**: Selenium IDE предоставляет удобный графический интерфейс, который позволяет создавать тесты без необходимости писать код. Это делает его доступным для тестировщиков, которые могут не иметь глубоких знаний программирования.
2. **Быстрое Создание Тестов**:
    - **Запись и Воспроизведение**: Вы можете записывать свои действия в браузере и воспроизводить их для создания тестов. Это ускоряет процесс создания тестов для простых сценариев.
3. **Отладка и Анализ**:
    - **Инструменты для Отладки**: Selenium IDE предоставляет функции для отладки тестов, такие как возможность остановки теста на определённом шаге, проверки состояния элементов и просмотра результатов тестов.
4. **Легкость в Обучении**:
    - **Дружественный Интерфейс**: Благодаря простоте интерфейса и возможности записи действий, новичкам легче освоиться и начать автоматизировать тестирование.
5. **Поддержка Различных Команд**:
    - **Разнообразие Команд**: Selenium IDE поддерживает широкий набор команд и методов для взаимодействия с элементами веб-страниц, таких как заполнение форм, клики, навигация и проверки значений.
6. **Поддержка Скриптов**:
    - **JavaScript и JUnit**: Для более сложных тестов можно использовать JavaScript и JUnit, что расширяет возможности тестирования.
7. **Портативность и Совместимость**:
    - **Плагины для Браузеров**: Selenium IDE доступен как расширение для браузеров Firefox и Chrome, что упрощает интеграцию в существующую среду тестирования.

Типичные Сценарии Использования

1. **Тестирование Пользовательских Интерфейсов**:
    - Selenium IDE идеально подходит для автоматизации тестов пользовательского интерфейса, таких как проверки корректности отображения элементов и их функциональности.

2. **Регрессионное Тестирование**:
    - Использование Selenium IDE для автоматического выполнения регрессионных тестов помогает убедиться, что изменения в коде не нарушили существующую функциональность.
3. **Быстрое Прототипирование Тестов**:
    - Тестировщики могут быстро создавать и запускать тесты для проверки новой функциональности или выявления проблем в процессе разработки.

Ограничения Selenium IDE

1. **Ограниченная Функциональность**:
    - Selenium IDE не предоставляет таких мощных возможностей для тестирования, как Selenium WebDriver. Он лучше подходит для простых тестов и сценариев.
2. **Невозможность Параллельного Выполнения**:
    - В отличие от Selenium Grid, Selenium IDE не поддерживает параллельное выполнение тестов на разных устройствах и браузерах.
3. **Отсутствие Поддержки Несколько Пользовательских Сессий**:
    - Selenium IDE не поддерживает тестирование с несколькими пользовательскими сессиями или сложными сценариями взаимодействия с веб-приложением.

Selenium IDE является отличным инструментом для начинающих тестировщиков и для быстрого прототипирования тестов, но для более сложных тестов и интеграции в большие тестовые комплексы чаще используются более мощные инструменты, такие как Selenium WebDriver.

Ход выполнения работы:

1. Установите расширение Selenium IDE для своего браузера. Например для Google Chrome можно перейти по следующей ссылке https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd

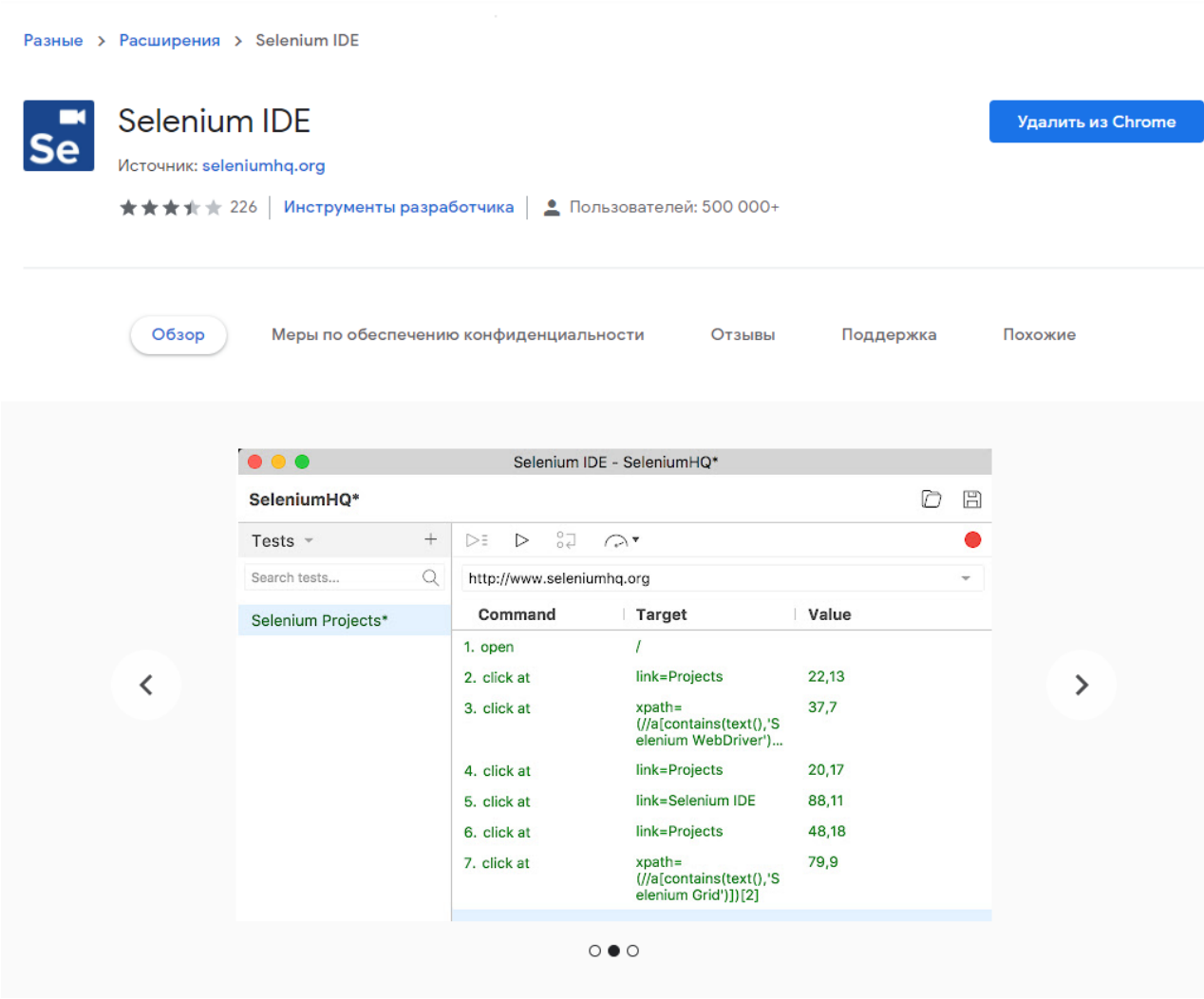


Рисунок 13 - Установка расширения.

2. Чтобы использовать установленное расширение нажмите на значок пазла (Рисунок 2).

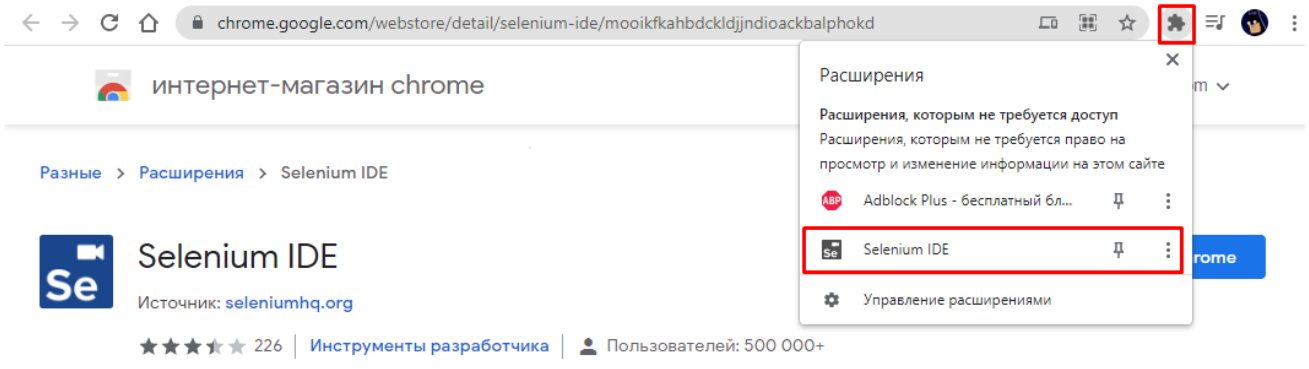


Рисунок 14 – Запустить Selenium IDE.

3. После открытия выберите «Создать новый проект». Дайте проекту любое название.
4. Для примера рассмотрим поисковую строку Google. Вставим в поле «Playback base URL» url-адрес поисковой строки: https://www.google.ru/ и нажмём на кнопку REC.

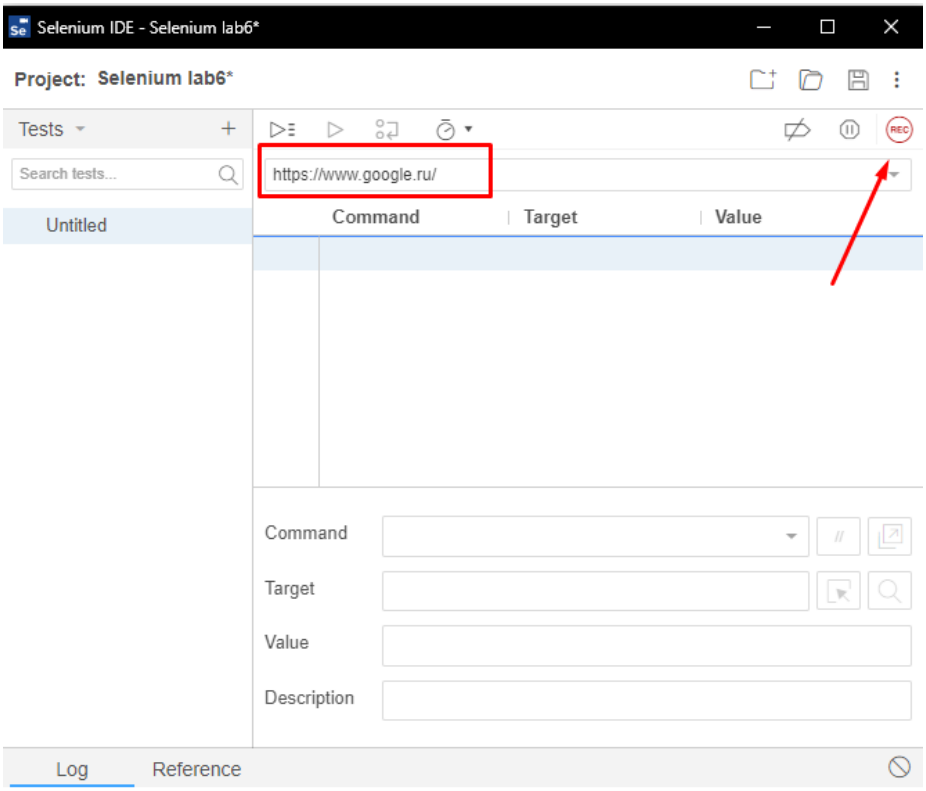


Рисунок 15 - Подготовка к выполнению теста.

5. После запуска записи откроется дополнительное окно, где можно будет записать сценарий (например, введите в поисковую строку «Учебный сайт кафедры ИСПИ», выполните поиск и кликните на предложенную поисковиком ссылку). Чтобы закончить запись нажмите кнопку Стоп (Рисунок 4). Сохраните выполненный тест.

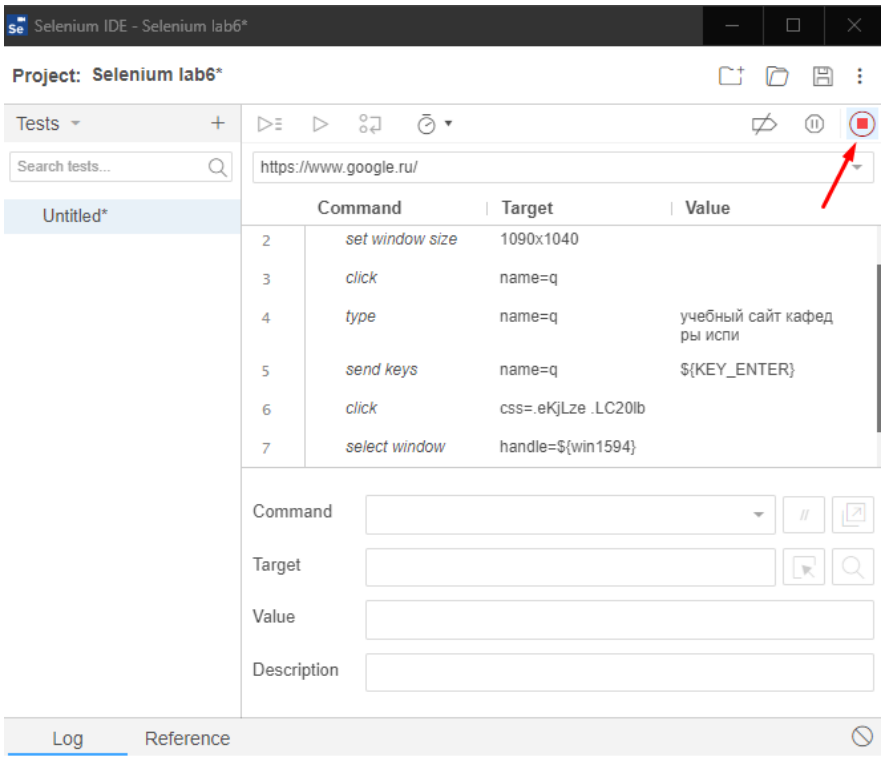


Рисунок 16 - Остановка записи.

6. После записи все проделанные вами действия были записаны в основное окно. Воспроизведите записанный ранее тест (Рисунок 5).
7. Сделайте дубликат данного теста и в новом тесте измените поисковый запрос (Рисунок 6).
8. Запустите новый сценарий. Вы можете заметить, что при выполнение нового сценария поисковый запрос изменился.

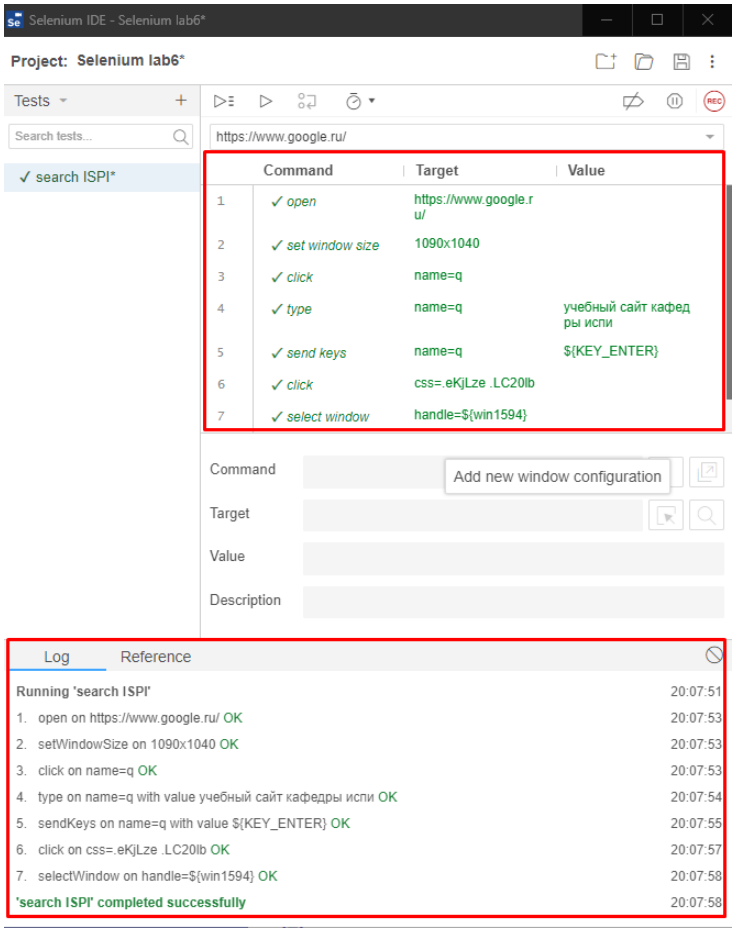


Рисунок 17 - Окно плагина после выполнения теста.

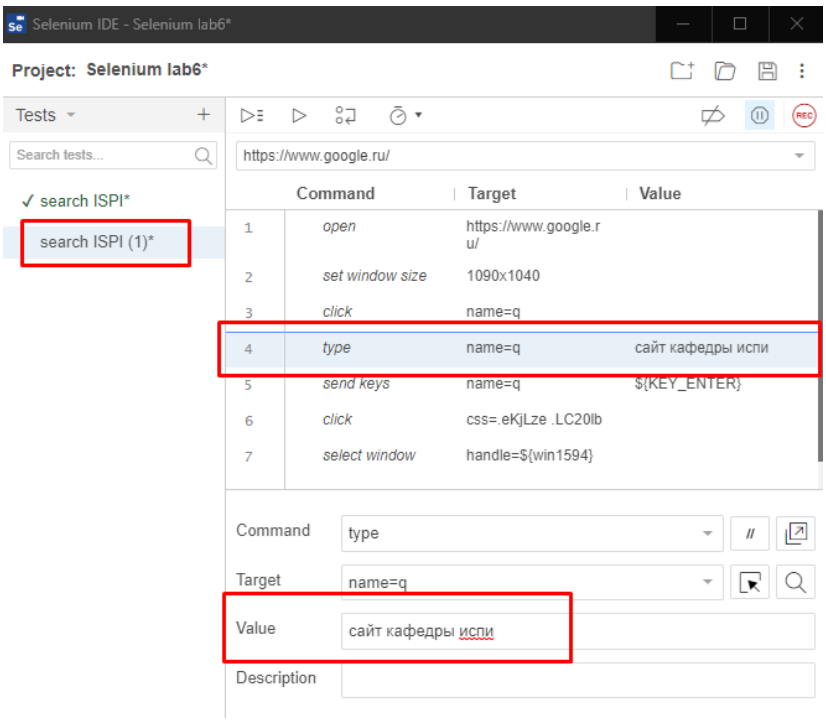


Рисунок 18 - Изменение запроса.

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Выберите любой сайт и придумайте сценарий для пользователя (опишите все действия пользователя последовательно);
2. Запишите сценарий используя Selenium IDE;
3. Сделайте несколько дубликатов сценария, измените данные;
4. Попробуйте провести небольшое нагрузочное тестирование, запустив сразу несколько сценариев;
5. Проанализируйте полученные результаты.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Цель работы;
2. Выбранный сайт;
3. Сценарий пользователя (в сценарии должно быть минимум 5 действий);
4. Скриншоты выполнения работы и пояснения;
5. Вывод.

ВОПРОСЫ

1. Что такое Selenium IDE и каковы его основные функции?
2. Как вы создаете тестовый сценарий в Selenium IDE?
3. Как вы можете отладить тесты в Selenium IDE?
4. Какие ограничения у Selenium IDE по сравнению с Selenium WebDriver?
5. Как вы можете сохранить и экспортировать тесты из Selenium IDE?

Лабораторная работа №10. Автоматизированное тестирование. Selenium WebDriver.

ЦЕЛЬ РАБОТЫ

Познакомиться с инструментов автоматизированного тестирования Selenium WebDriver.

Selenium WebDriver – это программная библиотека для управления браузерами. WebDriver представляет собой драйверы для различных браузеров и клиентские библиотеки на разных языках программирования, предназначенные для управления этими драйверами.

Библиотеки WebDriver доступны на языках Java, .Net (C#), Python, Ruby, JavaScript, драйверы реализованы для браузеров Firefox, InternetExplorer, Safari, Andriod, iOS (а также Chrome и Opera).

Чаще всего Selenium WebDriver используется для тестирования функционала вебсайтов/веб-ориентированных приложений.

Автоматизированное тестирование удобно, потому что позволяет многократно запускать повторяющиеся тесты. Регрессионное тестирование, то есть, проверка, что старый код не перестал работать правильно после внесения новых изменений, является типичным примером, когда необходима автоматизация.

WebDriver предоставляет все необходимые методы, обеспечивает высокую скорость теста и гарантирует корректность проверки (поскольку человеческий фактор исключен). В официальной документации Selenium приводятся следующие плюсы автоматизированного тестирования веб-приложений:

- возможность проводить чаще регрессионное тестирование;
- быстрое предоставление разработчикам отчета о состоянии продукта;
- получение потенциально бесконечного числа прогонов тестов;
- обеспечение поддержки Agile и экстремальным методам разработки;
- сохранение строгой документации тестов;
- обнаружение ошибок, которые были пропущены на стадии ручного тестирования.

Как работать с Selenium WebDriver: https://selenium-python.readthedocs.io/getting-started.html

Напишем небольшой автотест для сайта кафедры ИСПИ.

Во время изучения будем использовать следующие технологии:

- Python 3.10.4

- Установить можно с официального сайта: https://www.python.org/downloads/

- pip модуль selenium
  - команда установки: pip install selenium
- Google Chrome
- драйвер для браузера Chrome
  - https://chromedriver.chromium.org/downloads

Перед началом выполнения работы установите все описанные выше технологии и создайте отдельную папку на вашем компьютере.

Selenium требует наличия драйвера для взаимодействия с выбранным браузером. Скачайте драйвер для Google Chrome и поместите его в созданную для лабораторной работы папку.

Для других поддерживаемых браузеров будут доступны собственные драйверы. В таблице приведены ссылки на некоторые из наиболее популярных драйверов для браузеров.

Chrome	https://chromedriver.chromium.org/downloads
Edge	https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Firefox	https://github.com/mozilla/geckodriver/releases
Safari	https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Создайте в папке новый фал для скрипта и дайте ему название test.py. Откройте созданный файл и вставьте следующие строки:

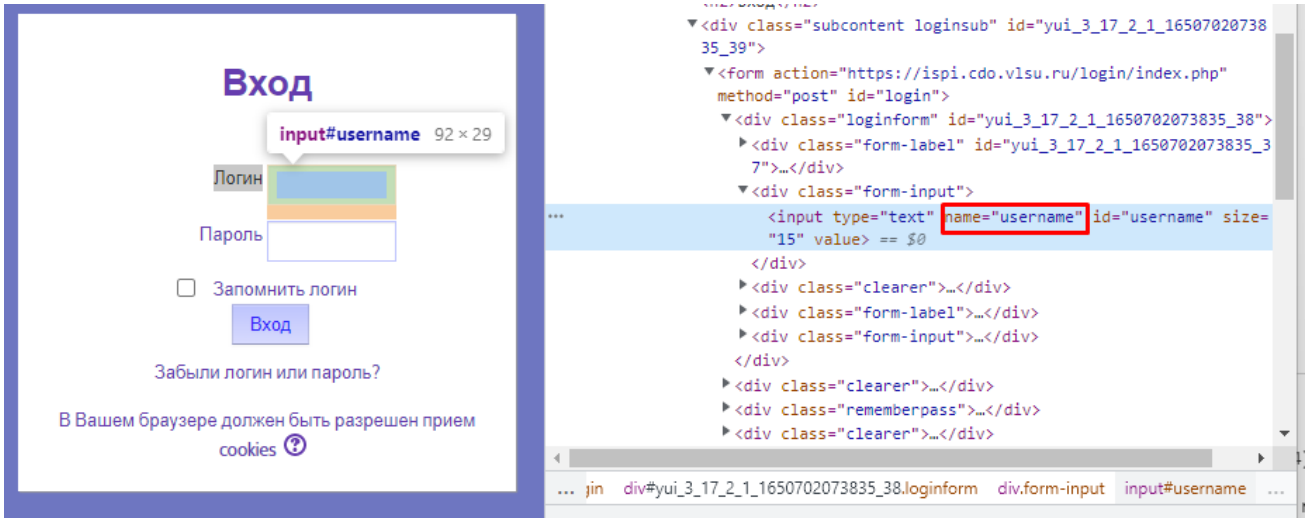
```
from selenium import webdriver

# Получаем в переменную browser указатель на браузер
# В новых версиях можно использовать webdriver.Chrome()
browser=webdriver.Chrome('./chromedriver')

# Переходим на страницу, на которой находится форма для авторизации
browser.get('https://ispi.cdo.vlsu.ru/login/index.php')

# Закрываем браузер
```

```
browser.close()
```

В конце скрипта можно закрыть браузер командой browser.close(). Запустите созданный скрипт через консоль используя команду: python test.py

Результатом выполнения будет являться открытие нового окна браузера и переход на учебный сайт кафедры ИСПИ.

Чтобы привязаться к определенному элементу на странице, например к полю ввода логина, мы будем использовать инструменты разработчика. Откройте вкладку Elements и найдите input поле Логин.



Для привязки нам нужно найти определенные маркеры этого элемента: id, name или css_selector. Чтобы заполнить поле определенным значением мы будем использовать метод send_keys().

Для действий с кнопками на странице используется стандартный метод click().

Добавляем в наш скрипт новые строки и запускаем его.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

 # Получаем в переменную browser указатель на браузер
browser=webdriver.Chrome()

# Переходим на страницу, на которой находится форма для авторизации
browser.get('https://ispi.cdo.vlsu.ru/login/index.php')
```

```
# заполняем поле логин, привязываемся к элементу через его имя
username=browser.find_element(by=By.NAME, value='username')
username.send_keys('k1179e6jt')

# заполняем поле пароля, привязываемся к элементу через его id
password=browser.find_element(by=By.ID, value='password')
password.send_keys('NA498bli')

#Получаем указатель на кнопку "Вход", привязываемся к элементу через его css_selector
button=browser.find_element(by=By.CSS_SELECTOR, value='#loginbtn')
#Нажимаем на кнопку входа
button.click()

# Закрываем браузер
browser.close()
```

Результатом выполнения данного скрипта может быть появление на странице надписи «Неверный логин или пароль, попробуйте заново.» или успешная авторизация – это два возможных сценария авторизации. Напишем два тест-кейса и для каждого тест-кейса расширим написанный ранее скрипт, добавив дополнительные условия.

Номер	1
Название	Авторизация с некорректными учетными данными.
Предусловие	Пользователь не авторизирован в системе.
Шаги	1. Переходим на сайт кафедры ИСПИ на страницу авторизации; 2. В поле «Логин» вводим 'k1179e6jt'; 3. В поле «Пароль» вводим 'NA498bli'; 4. Нажимаем на кнопку «Вход».
Ожидаемый результат	Пользователь находится на странице авторизации. На странице отображается сообщение «Неверный логин или пароль,

	попробуйте заново.».

Чтобы проверить что пользователь находится на странице авторизации после всех выполненных действий можно проверить заголовок страницы, он должен быть равен следующей надписи: 'Учебный сайт кафедры ИСПИ: Вход на сайт'. Добавим в написанный ранее скрипт проверку ожидаемого результата перед закрытием браузера:

```
# Проверка результата
try:
    # Проверка что пользователь находится на странице авторизации
    assert 'Учебный сайт кафедры ИСПИ: Вход на сайт' in browser.title
    errormessage=browser.find_element(by=By.ID, value='loginerrormessage')
    # Проверка сообщения об ошибке на странице
    assert 'Неверный логин или пароль, попробуйте заново.' in errormessage.accessible_name
    print('The test was completed successfully')
except Exception as err:
    print('The test was failled')
```

Номер	2
Название	Авторизация с корректными учетными данными.
Предусловие	Пользователь не авторизирован в системе.
Шаги	1. Переходим на сайт кафедры ИСПИ на страницу авторизации; 2. В поле «Логин» вводим <ваш логин>; 3. В поле «Пароль» вводим <ваш пароль>; 4. Нажимаем на кнопку «Вход».
Ожидаемый результат	Пользователь успешно авторизовался в системе. На странице отображается имя текущего пользователя.

Добавим в скрипт новую проверку ожидаемого результата перед закрытием браузера:

```
try:
    # Проверка что пользователь находится на главной странице сайта
    assert 'Учебный сайт кафедры ИСПИ' in browser.title
    # Проверка что на странице присутствует полное имя пользователя
    assert "<Фамилия Имя Отчество Пользователя>" in browser.page_source
    print('The test was completed successfully')
except Exception as err:
    print('The test was failled')
```

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

6. Изучить представленный преподавателем материал и выполнить пример;
7. Выбрать любой веб-сайт и описать основные его функции. Для учебного сайта кафедры ИСПИ (для роли «студент») это:
    a. Авторизация;
    b. Просмотр курсов;
    c. Скачивание учебного материала;
    d. Прикрепление заданий;
    e. Прохождение тестирования;
    f. Отправка сообщений.
8. Опираясь на основные функции системы написать три тест-кейса для автоматизированного тестирования.
9. Согласно написанным тест-кейсам реализовать автоматизированные тесты используя Selenium WebDriver;
10. Создать репозиторий на GitHub и выложить реализованные скрипты;
    a. Добавить README файл с описанием тест-кейсов;
11. Оформить лабораторную работу.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

10. Цель работы;
11. Выполненный пример;
12. Описание функций выбранного веб-сайта;
13. Тест-кейсы;

14. Код;
15. Скриншоты выполнения;
16. Ссылка на GitHub;
17. Вывод.

ВОПРОСЫ

1. Что такое Selenium WebDriver и как он работает?
2. Как установить Selenium WebDriver для Python?
3. Как начать работу с Selenium WebDriver в Python?
4. Как открыть веб-страницу с помощью Selenium WebDriver?
5. Как взаимодействовать с элементами веб-страницы с помощью Selenium WebDriver?
6. Как справиться с ожиданиями в Selenium WebDriver?
7. Как можно обрабатывать алерты и модальные окна с помощью Selenium WebDriver?
8. Как можно управлять несколькими окнами и вкладками в Selenium WebDriver?
9. Как выполнять тесты на различных браузерах с помощью Selenium WebDriver?
10. Как можно использовать XPath и CSS-селекторы для поиска элементов в Selenium WebDriver?

Лабораторная работа №11. Инструменты разработчика в браузере

ЦЕЛЬ РАБОТЫ

Изучить инструменты разработчика в браузере.

В каждом браузере есть свой инструмент разработчика – это то, что многие называют просто «консолью», рассмотрим Консоль в браузере Chrome.

Консоль в браузере Chrome — это инструмент, с помощью которого можно не только посмотреть наполнение страницы, выводимой браузером, а также существующие ошибки (что чаще всего и делают тестировщики), но и исправить эти ошибки намного проще и быстрее (что чаще всего делают разработчики), замерять различные показатели и манипулировать страницей.

Инструменты разработчика в браузере (или DevTools) — это мощный набор инструментов, встроенных в современные веб-браузеры, такие как Google Chrome, Firefox, Microsoft Edge и другие. Они предоставляют тестировщикам и разработчикам возможность анализировать, отлаживать и оптимизировать веб-приложения. Вот основные функции и их значение для тестировщика, а также вопросы для контроля.

Зачем тестировщику нужны инструменты разработчика в браузере?

1. **Отладка и анализ ошибок**
    - Позволяют тестировщику отлаживать и анализировать ошибки в веб-приложении, включая JavaScript-ошибки, ошибки загрузки ресурсов и проблемы с сетевыми запросами.
2. **Анализ сетевых запросов**
    - Тестировщик может просматривать и анализировать сетевые запросы и ответы, что помогает в проверке правильности передачи данных между клиентом и сервером, а также в отладке проблем с API.
3. **Изменение и тестирование CSS**
    - Позволяют тестировщику изменять стили на лету, проверять различные состояния и адаптивность страницы, что помогает в выявлении и исправлении проблем с дизайном и версткой.
4. **Работа с DOM**
    - Предоставляют возможность просматривать и изменять структуру документа, что полезно для проверки правильности отображения элементов и отладки проблем с отображением.
5. **Профилирование и оптимизация производительности**
    - Позволяют тестировщику измерять производительность веб-приложения, анализировать загрузку страницы и выявлять узкие места, влияющие на скорость работы.
6. **Эмуляция различных устройств и сетевых условий**
    - Помогают тестировщику проверить, как веб-приложение работает на разных устройствах и в условиях ограниченной скорости сети, что важно для обеспечения кросс-браузерной и кросс-платформенной совместимости.
7. **Тестирование безопасности**
    - Позволяют тестировать аспекты безопасности, такие как защищенность данных и возможность выполнения небезопасных операций на стороне клиента.

Как открыть консоль в браузере Chrome:

- клавиша F12;
- нажав одновременно клавиши Ctrl + Shift + I;
- ПКМ по элементу страницы –> Просмотреть код;
- меню браузера –> Дополнительные Инструменты –> Инструменты Разработчика.

Располагаться консоль может внизу страницы или сбоку, можно открепить в отдельное окно. Итак, рассмотрим по отдельности самые важные вкладки.

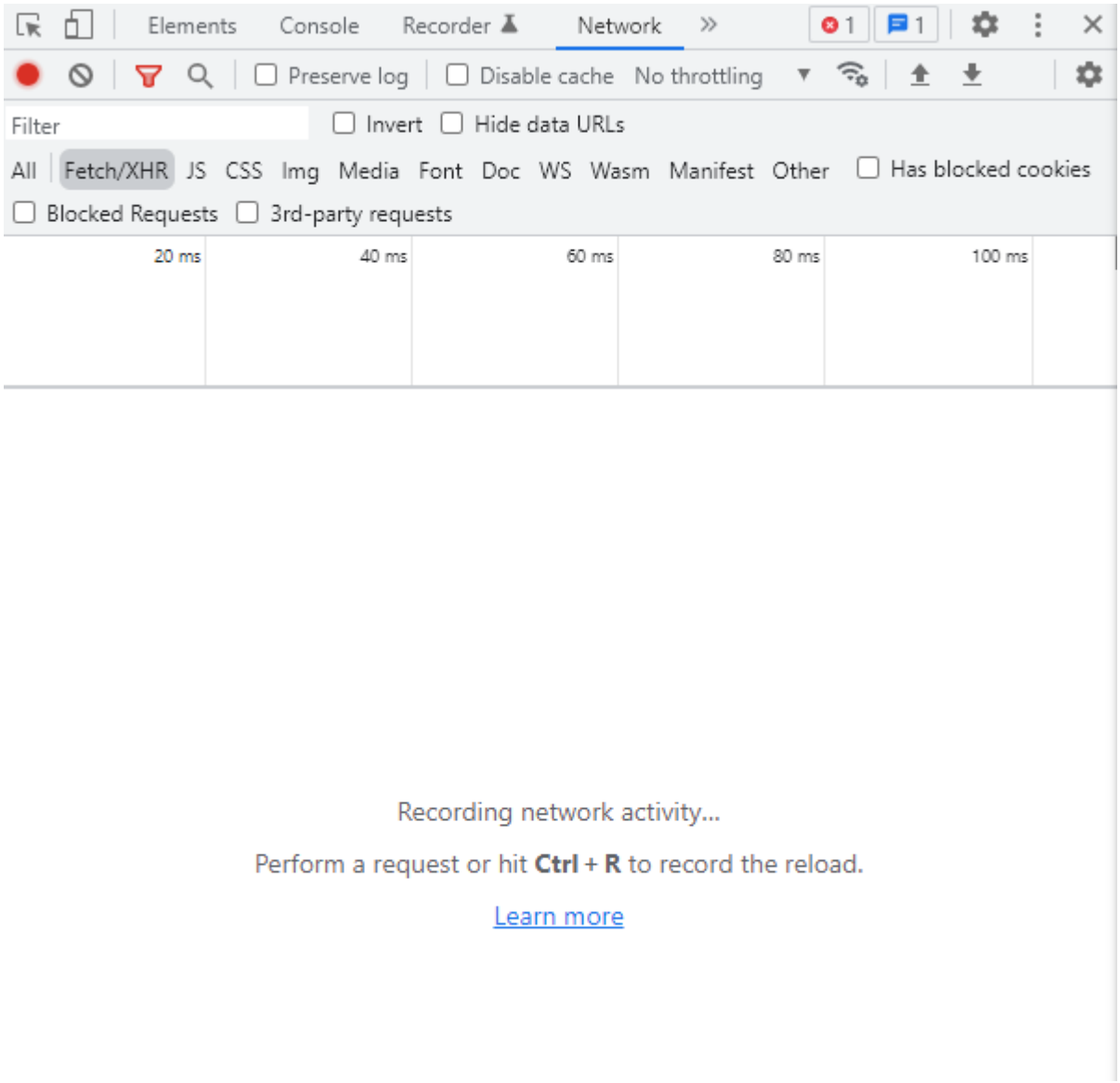


Рисунок 19 - Инструменты разработчика.

Панель Elements показывает разметку страницы точно так же, как она отображается в браузере. Можно визуально менять наполнение сайта посредством изменения html/css кода в панели элементов. Проводя нехитрые манипуляции с данными можно изменить наполнение и дизайн открытой страницы. Например, можно поменять текст в окне, если редактировать его в теле html, а также изменить шрифт страницы поменяв его значение в поле css. Но это не сохранит введенных данных, а поможет просто визуально оценить примененные изменения.

Рассмотрим пример изменения текста на странице.

1. Зайдите на любой сайт (мы будем рассматривать сайт кафедры ИСПИ, курс «Тестирование ИС») и откройте консоль разработчика, вкладку Elements;

2. Выберете возможность просмотра конкретного кода элемента, используя значок со стрелочкой в левой верхней части вкладки Elements;

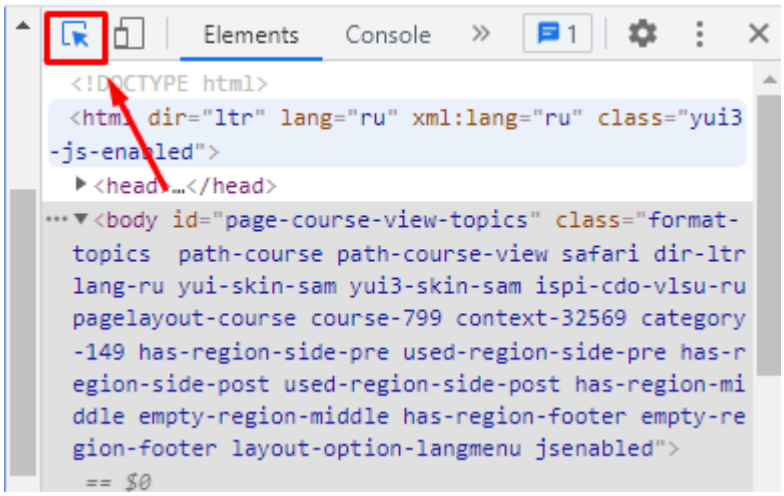


Рисунок 20 - Выбор конкретного элемента на странице.

3. Кликните на любой текст на странице. Во вкладке Elements код нужного фрагмента подсветиться;

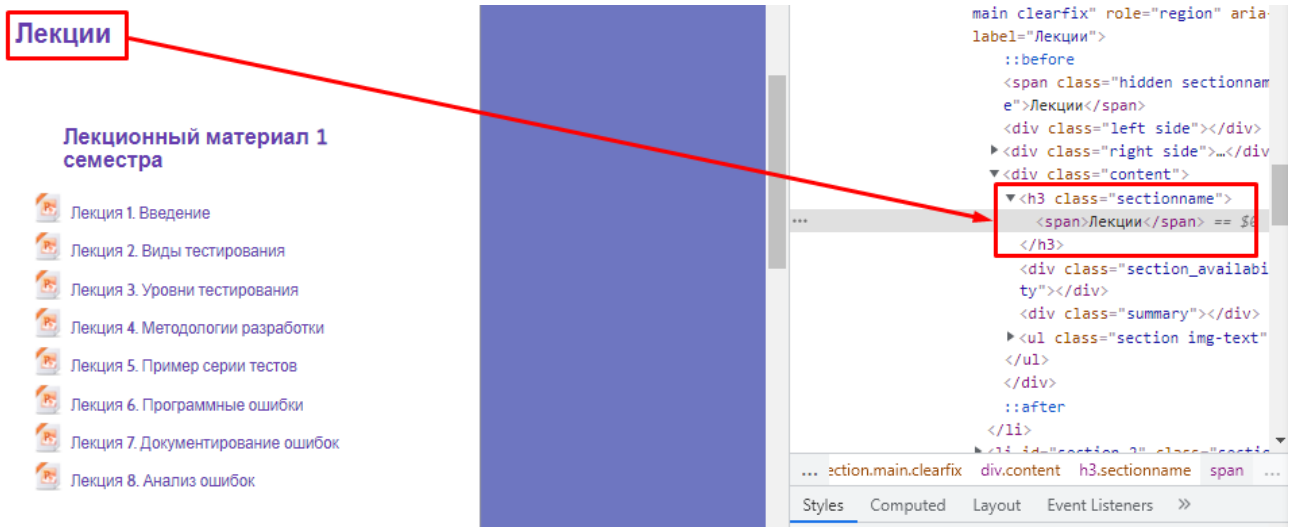


Рисунок 21 - Код текста на странице.

4. Кликните по тексту в коде дважды и измените его;

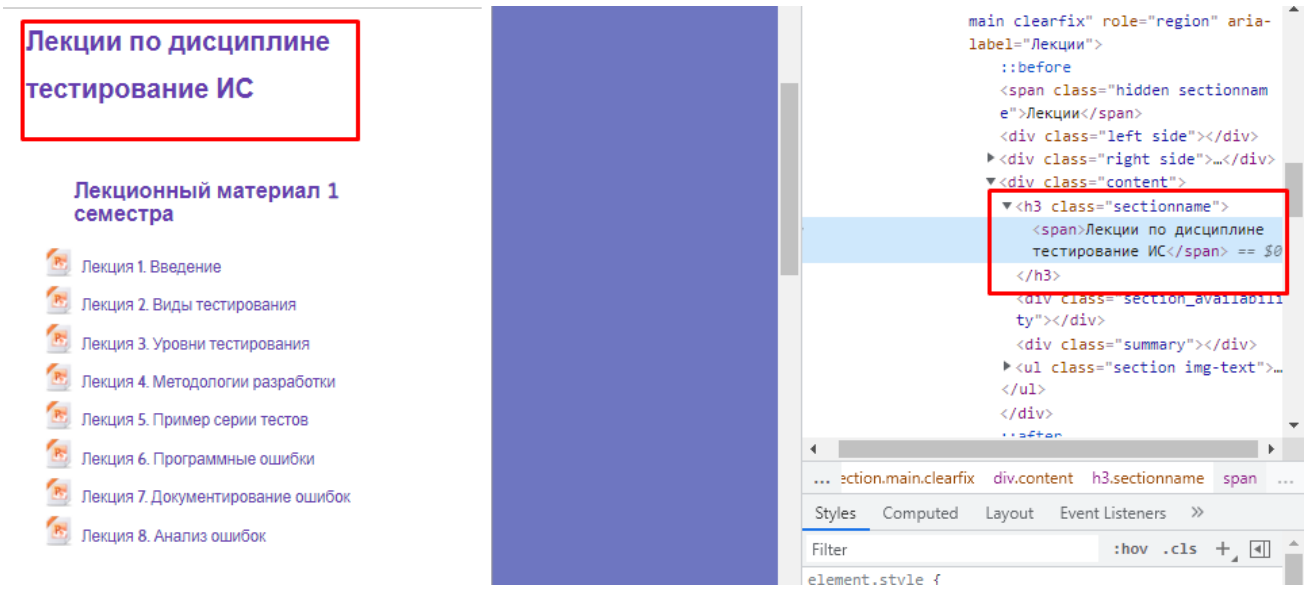


Рисунок 22 - Измененный на странице текст.

5. Изменение стиля текста происходит на вкладке Styles;

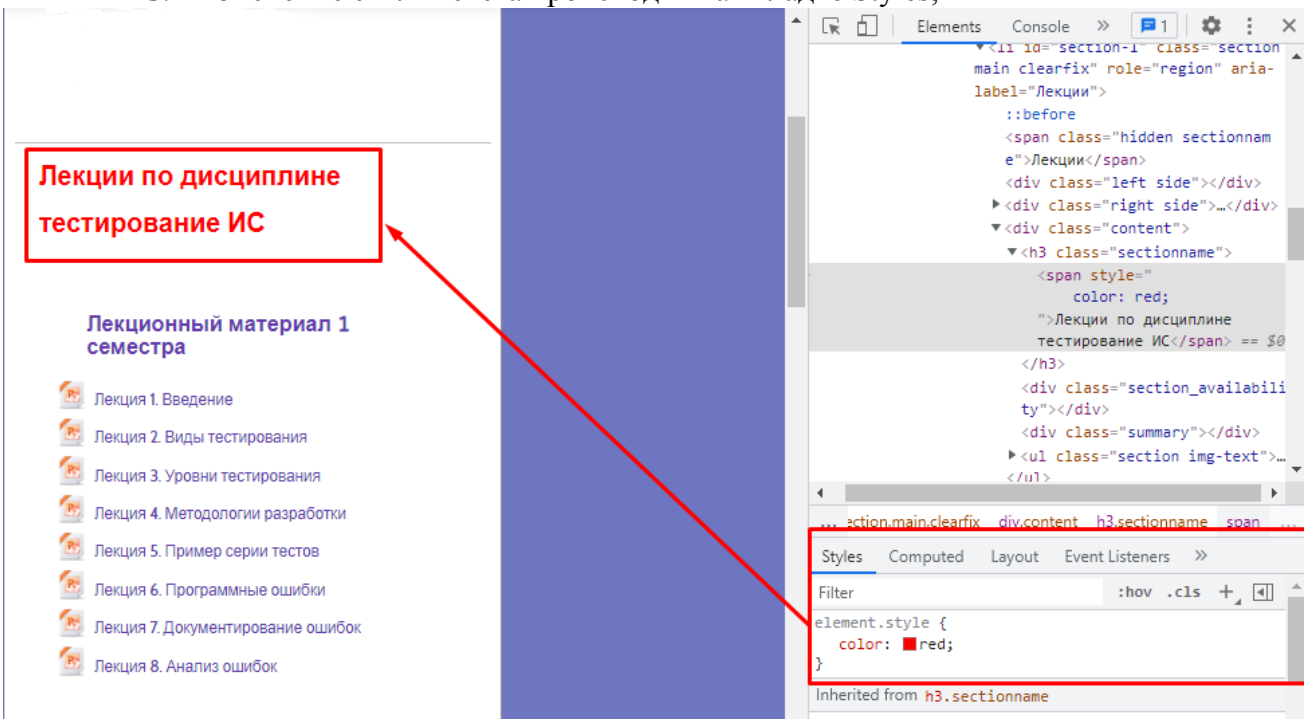
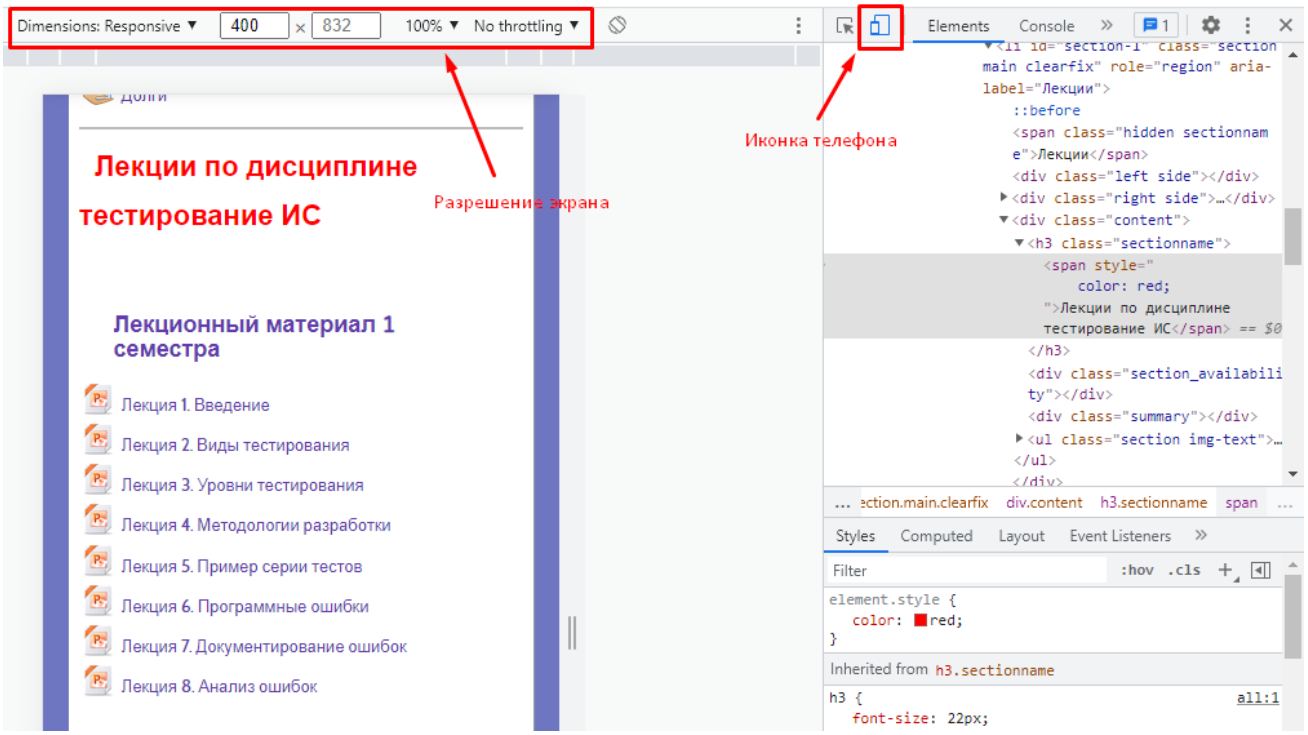


Рисунок 23 - Изменение стиля.

В панели Elements есть одна очень прекрасная функция. Можно посмотреть, как бы выглядела открытая страница на каком-нибудь девайсе с другим разрешением экрана. Кликом по иконке телефона слева от вкладки Elements вызывается окно, в котором можно менять размер предполагаемого экрана, таким образом эмулируя тот или иной девайс и контролировать отображение страницы на нем.



Панель Console - самая популярная вкладка тестировщиков, поскольку именно здесь мы можем видеть найденные при выполнении скрипта ошибки в коде. Также данная панель отображает различного рода предупреждения и рекомендации. Все выводимые во вкладке сообщения можно фильтровать. Фильтровать сообщения в консоли можно по типу — ошибки, предупреждения, инфо, стандартный вывод, сообщения отладчика, исправленные — выбрав одну из доступных опций консоли.

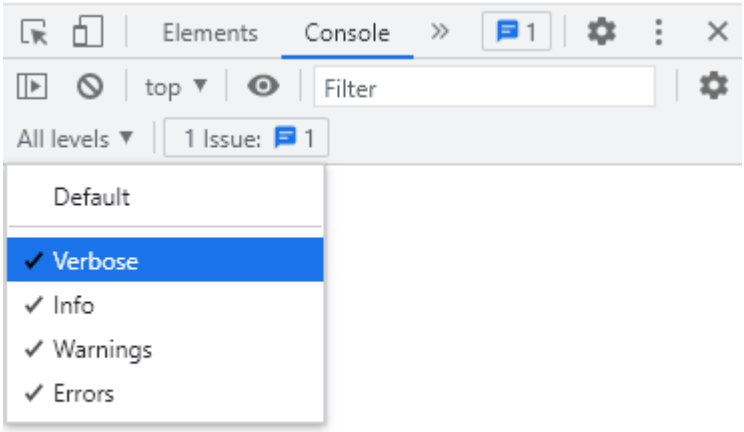


Рисунок 24 - Фильтрация сообщений об ошибках.

Очистить поле вкладки Console (в случае, если вам требуется удалить сообщения о предыдущих ошибках) можно кликнув иконку перечеркнутого круга  .

Так же в консоли можно выполнять простейший код на js.

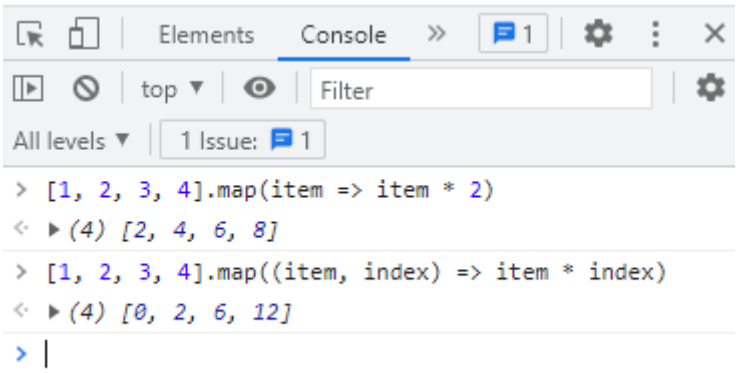


Рисунок 25 – Выполнение кода в консоли.

Панель Network – запись сетевого журнала. Она дает представление о запрашиваемых и загружаемых ресурсах в режиме реального времени. Можно выявить, загрузка и обработка каких именно ресурсов занимает большее количество времени, чтобы впоследствии знать где и в чем именно можно оптимизировать страницу.

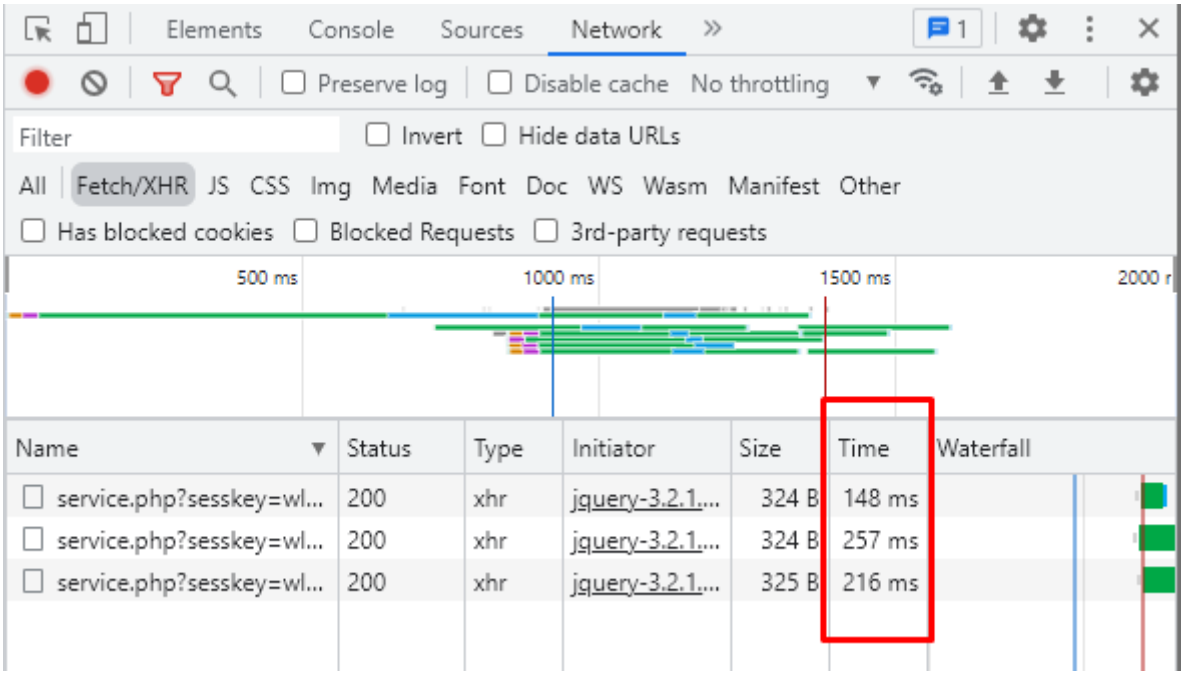


Рисунок 26 - Время выполнения запросов.

Также стоит отметить, что именно в этой вкладке можно просмотреть запросы, которые отправляются на сервер, а также ответы, которые приходят с него, их содержание и характеристики.

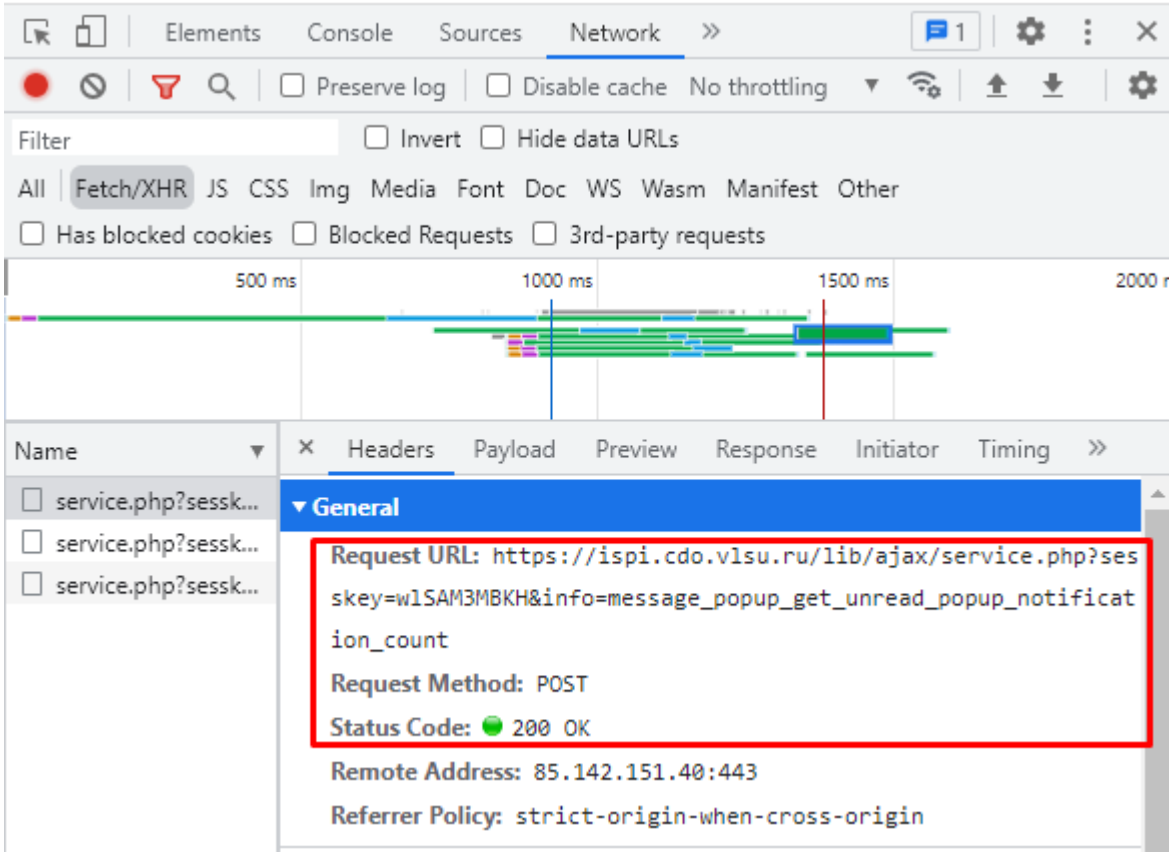


Рисунок 27 - Информация о запросе.

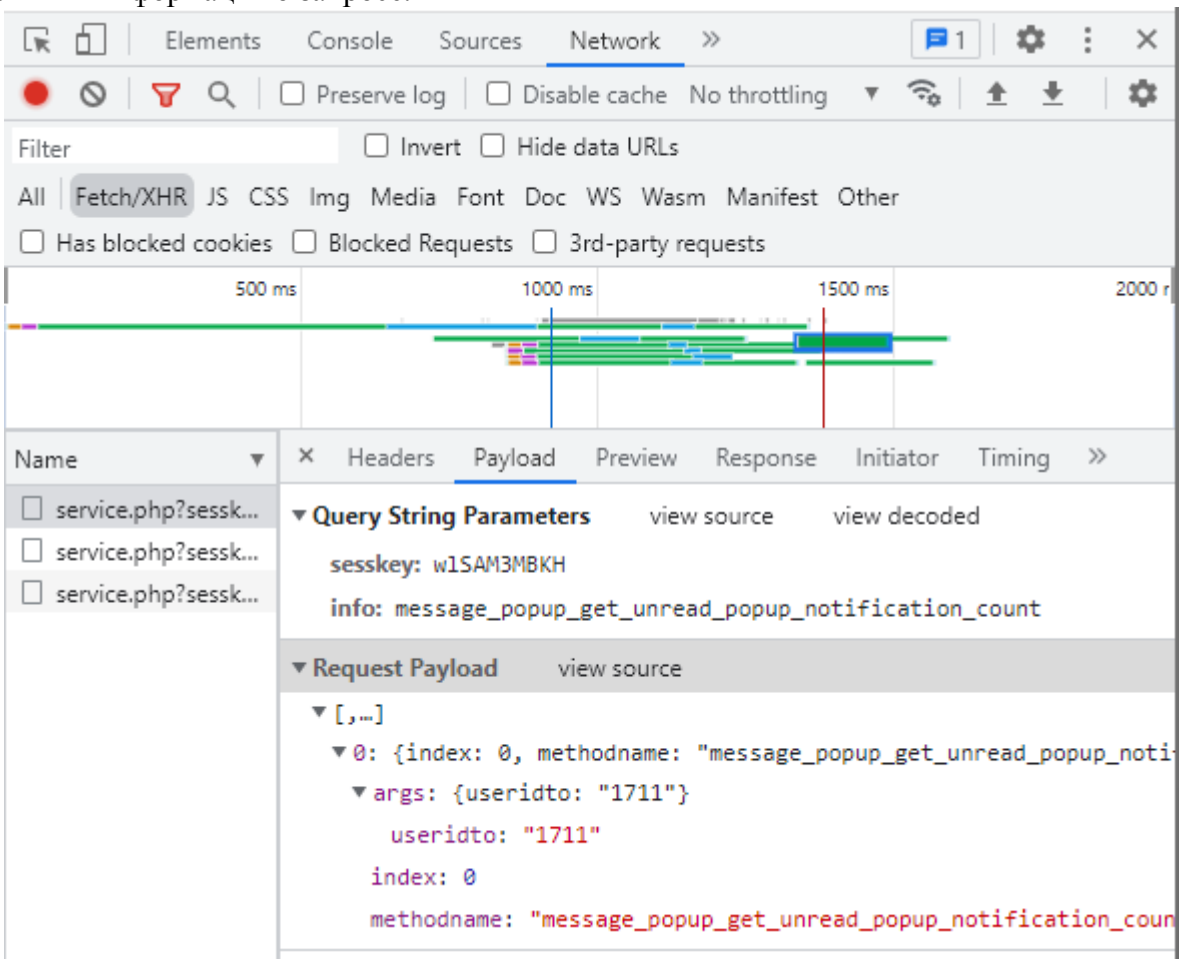


Рисунок 28 - Информация о запросе.

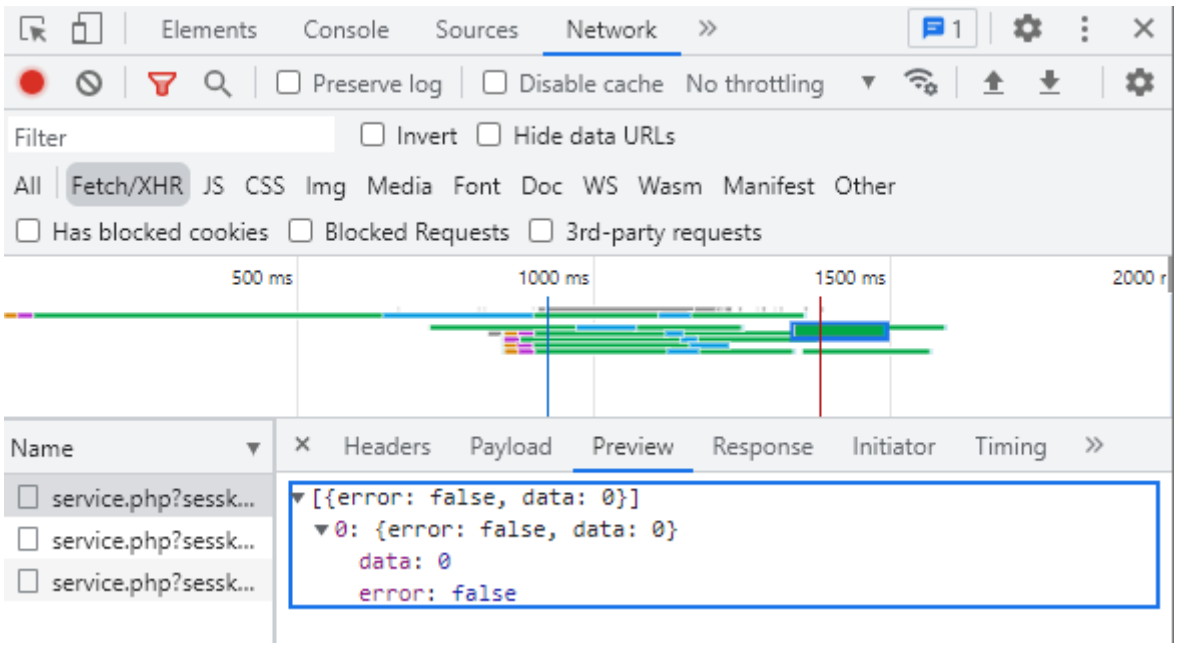


Рисунок 29 - Ответ.

Работать в Network удобно, если хорошо знать ее возможности. Вы можете по желанию изучить статью [Как эффективно использовать Network в Chrome DevTools]

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Изучите представленный преподавателем материал, выберите любой веб-сайт для выполнения заданий к лабораторной работе. Опишите выбранный веб-сайт и прикрепите на него ссылку;
2. Используя вкладку Elements, консоли разработчика, измените любой текст на странице на ваше имя, фамилию и номер группы, и стиль текста в соответствии с номером вашего варианта;
3. Изучите вкладку Console, отфильтруйте все ошибки и изучите минимум две из них;
4. Изучите вкладку Network: выявите, загрузка и обработка каких именно ресурсов занимает большее количество времени, опишите минимум два разных запроса на сервер;
5. Самостоятельно изучите ещё одну любую вкладку инструментов разработчика, опишите её предназначение.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Цель работы;
2. Вариант задания (соответствует вашему номеру в списке группы);
3. Описание выбранного веб-сайта, ссылка на веб-сайт;
4. Выполнение задания во вкладках Elements, Console, Network. Скриншоты;
5. Информация о самостоятельно изученной вкладке инструментов разработчика (выделите только главное);
6. Вывод по работе.

ВАРИАНТЫ

Номер варианта	Стиль текста (цвет текста, стиль текста, размер текста соответствует номеру варианта в пикселях)
Вариант 1	Зеленый, подчеркнутый
Вариант 2	Красный, подчеркнутый
Вариант 3	Синий, подчеркнутый
Вариант 4	Черный, подчеркнутый
Вариант 5	Розовый, подчеркнутый
Вариант 6	Желтый, подчеркнутый
Вариант 7	Оранжевый, подчеркнутый
Вариант 8	Фиолетовый, подчеркнутый
Вариант 9	Голубой, подчеркнутый
Вариант 10	Зеленый, перечеркнутый
Вариант 11	Красный, перечеркнутый
Вариант 12	Синий, перечеркнутый
Вариант 13	Черный, перечеркнутый
Вариант 14	Розовый, перечеркнутый
Вариант 15	Желтый, перечеркнутый
Вариант 16	Оранжевый, перечеркнутый
Вариант 17	Фиолетовый, перечеркнутый
Вариант 18	Голубой, перечеркнутый
Вариант 19	Зеленый, нет стиля
Вариант 20	Красный, нет стиля
Вариант 21	Синий, нет стиля
Вариант 22	Зеленый, линия проходит над текстом
Вариант 23	Красный, линия проходит над текстом
Вариант 24	Синий, линия проходит над текстом
Вариант 25	Черный, линия проходит над текстом
Вариант 26	Розовый, линия проходит над текстом
Вариант 27	Желтый, линия проходит над текстом
Вариант 28	Оранжевый, линия проходит над текстом
Вариант 29	Фиолетовый, линия проходит над текстом
Вариант 30	Голубой, линия проходит над текстом

ВОПРОСЫ

1. Что такое инструменты разработчика в браузере и в каких браузерах они доступны?
2. Как открыть инструменты разработчика в Google Chrome и других популярных браузерах?
3. Какие вкладки в инструментах разработчика используются для анализа сетевых запросов и ответов?
4. Как можно отследить и проанализировать JavaScript-ошибки с помощью инструментов разработчика?
5. Как изменять и тестировать CSS стили на лету с помощью инструментов разработчика?

6. Как использовать инструменты разработчика для анализа производительности веб-приложения?
7. Как можно просмотреть и изменить структуру DOM с помощью инструментов разработчика?
8. Как можно эмулировать различные устройства и размеры экрана в инструментах разработчика?
9. Как можно просматривать и изменять куки и локальное хранилище с помощью инструментов разработчика?
10. Как можно использовать инструменты разработчика для анализа и тестирования безопасности веб-приложения?
11. Как можно настроить и использовать фильтры для сетевых запросов в инструментах разработчика?

Лабораторная работа №12. Тестирование мобильных приложений

ЦЕЛЬ РАБОТЫ

Изучить основы тестирования мобильных приложений и применить полученных знаний на практике.

Тестирование мобильных приложений – это процесс, с помощью которого прикладное ПО, разработанное для портативных мобильных устройств, проверяется на его функциональность, удобство использования и совместимость. Тестирование может быть мануальным или автоматизированным.

Функциональное тестирование является самым базовым тестом для любого приложения, для проверки соответствия требованиям. Подобно другим приложениям, основанным на пользовательском интерфейсе, мобильные приложения требуют ряда взаимодействий человека в пользовательских сценариях.

Тестирование совместимости имеет самую высокую важность, когда дело доходит до тестирования мобильных приложений. Цель теста на совместимость мобильного приложения, как правило, состоит в том, чтобы ключевые функции приложения работали должным образом на конкретном устройстве. Сама совместимость должна занимать всего несколько минут и может быть спланирована заранее.

Localization Testing. В настоящее время большинство приложений предназначены для глобального использования, и очень важно заботиться о региональных особенностях,

таких как языки, часовые пояса и т.д. Важно проверить функциональность приложения, когда кто-то меняет часовой пояс.

Laboratory testing, обычно проводимые сетевыми операторами, выполняются путем моделирования всей беспроводной сети. Этот тест выполняется для обнаружения каких-либо сбоев, когда мобильное приложение использует передачу голоса и / или данных для выполнения некоторых функций.

Stress Testing является обязательным тестированием на пути обнаружения исключений, зависаний и взаимоблокировок, что может остаться незамеченными во время тестирования функциональности и пользовательского интерфейса.

Вот список некоторых критериев:

- Загрузите в свое приложение как можно больше данных, чтобы попытаться достичь его предела.
- Выполняйте одни и те же операции снова и снова.
- Выполняйте повторные операции на разных скоростях, очень быстро или очень медленно.
- Оставьте ваше приложение работающим в течение длительного периода времени, одновременно взаимодействуя с устройством и просто оставляя его бездействующим, или выполняя некоторую автоматическую задачу, которая занимает много времени, например, слайд-шоу.
- На вашем устройстве должно быть запущено несколько приложений, чтобы вы могли часто переключаться между приложением и другими приложениями на устройстве.

Security Testing помогает выявить все возможные уязвимости в отношении политик взлома, аутентификации и авторизации, безопасности данных, управления сеансами и других стандартов безопасности. Приложения должны шифровать имя пользователя и пароли при аутентификации пользователя по сети.

Usability Testing оценивает приложение на основе следующих трех критериев для целевой аудитории: эффективность; точность и полнота; удовлетворенность. Очень важно провести юзабилити-тестирование с самого раннего этапа разработки приложения. Этот вид тестирования требует активного участия пользователей, и результаты могут повлиять на дизайн приложения, что очень трудно изменить на более поздних этапах проекта.

Тестирование мобильных приложений представляет собой сложную задачу по нескольким причинам. Вот основные сложности, с которыми сталкиваются тестировщики:

1. Разнообразие устройств и платформ

Многообразие устройств: Мобильные приложения должны работать на множестве различных устройств с разными размерами экранов, разрешениями и аппаратными характеристиками. Это усложняет тестирование, поскольку необходимо проверять приложение на множестве конфигураций.

Операционные системы: iOS и Android имеют разные API, особенности и требования, что требует отдельного тестирования для каждой платформы. Разные версии операционных систем также могут иметь свои особенности и проблемы.

2. Различные разрешения экранов и размеры устройств

Адаптивность интерфейса: Приложение должно корректно отображаться на экранах разных размеров и разрешений. Это требует тщательной проверки адаптивности интерфейса и корректности отображения всех элементов.

3. Различия в аппаратных характеристиках

Производительность и ресурсы: Устройства имеют различную производительность, объем оперативной памяти и мощности процессора. Приложение должно быть протестировано на устройствах с разными характеристиками, чтобы убедиться, что оно работает стабильно и эффективно.

4. Различия в версиях операционных систем

Обновления и совместимость: Новые версии операционных систем могут вносить изменения, которые могут повлиять на работу приложения. Необходимо тестировать приложение на разных версиях ОС, чтобы убедиться в его совместимости.

5. Мобильные сети и их нестабильность

Разные типы соединений: Мобильные приложения могут использовать различные типы сетевых соединений (Wi-Fi, 4G, 5G и т.д.), и производительность и поведение приложения могут значительно отличаться в зависимости от качества соединения.

Проблемы с подключением: Сетевые сбои и нестабильное соединение могут повлиять на работу приложения, особенно если оно зависит от постоянного интернет-соединения.

6. Различные модели поведения пользователя

Мобильный опыт: Пользователи могут использовать мобильное приложение в различных условиях (на улице, в транспорте и т.д.), что может влиять на поведение приложения и его стабильность.

7. Особенности мобильных интерфейсов

Жесты и сенсорные экраны: Мобильные приложения часто используют жесты (тапы, свайпы и т.д.), которые могут быть сложны для тестирования и воспроизведения. Тестировщикам нужно убедиться, что все жесты работают корректно на всех устройствах.

8. Безопасность и конфиденциальность

Права доступа и конфиденциальность: Мобильные приложения часто требуют доступ к различным системным функциям и данным (камера, геолокация, контакты и т.д.). Тестирование должно учитывать правильность работы с этими функциями и защиту пользовательских данных.

9. Процесс публикации и развертывания

App Store и Google Play: Процесс публикации приложений в магазинах приложений включает в себя проверку и одобрение, что может добавить сложности и задержки в процессе тестирования.

Ход выполнения работы

1. Установите приложение Expo(https://expo.dev/client) на ваше мобильное устройство;

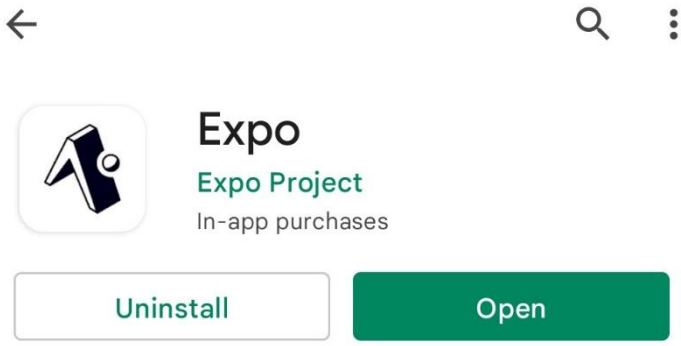


Рисунок 30 - Вид приложения в Google Play

2. Перейдите по ссылке https://snack.expo.dev/@arranay/calculator и выберите опцию My Device в правой части экрана;

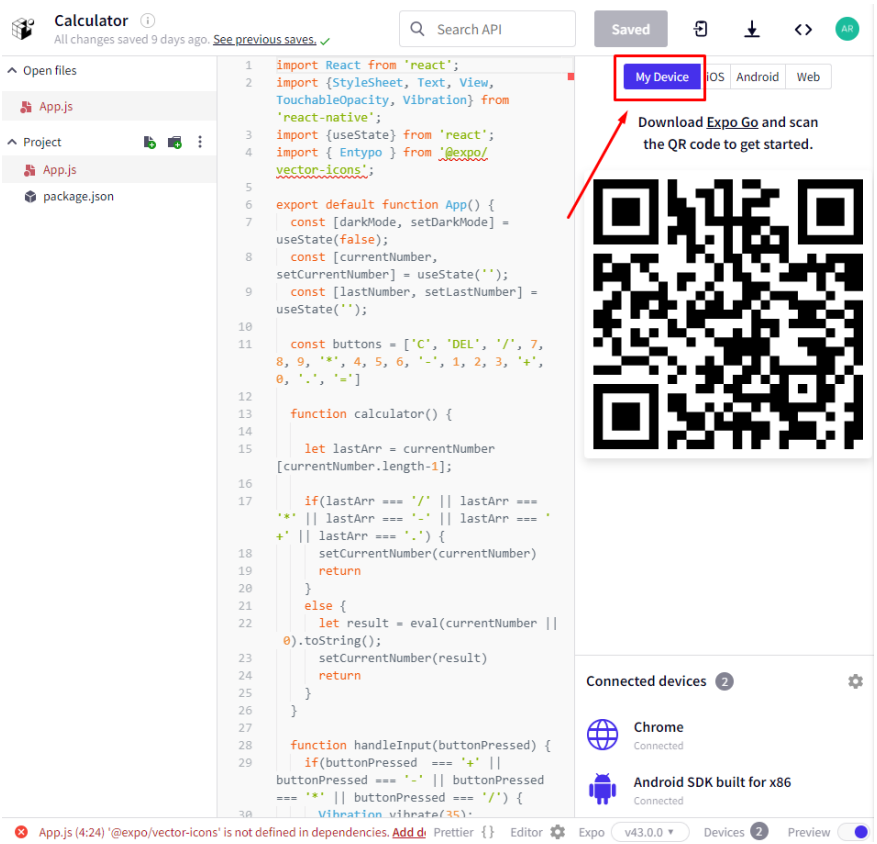


Рисунок 31 - Сформированный QR код для приложения.

3. Откройте установленное приложение и отсканируйте QR коде;

4. Для проверки работоспособности введите простейшую операцию на калькуляторе.

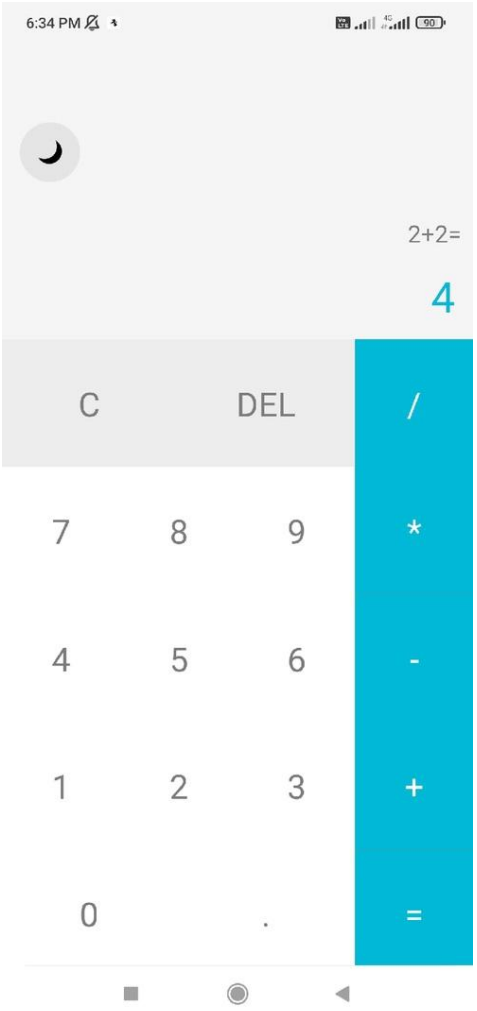


Рисунок 32 - Проверка работы калькулятора.

Задание к лабораторной работе

1. Изучите представленный преподавателем материал;
2. Используя тест-кейсы из первой лабораторной работы протестируйте мобильное приложение;
3. Опишите найденные дефекты.

СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

1. Цель работы;
2. Тест кейсы;
3. Скриншоты выполнения операций;
4. Дефекты;
5. Вывод по работе.

ВОПРОСЫ

1. Почему тестирование мобильных приложений требует проверки на различных устройствах и платформах?
2. Какие трудности связаны с тестированием приложения на разных версиях операционных систем?
3. Как нестабильность мобильных сетей может повлиять на тестирование?
4. Какие особенности мобильного интерфейса могут усложнить тестирование?
5. Почему важно тестировать мобильное приложение на устройствах с различными аппаратными характеристиками?
6. Как особенности поведения пользователей могут повлиять на тестирование мобильного приложения?
7. Что такое права доступа в контексте мобильного тестирования и почему их важно проверять?
8. Какие шаги нужно предпринять для тестирования адаптивности интерфейса мобильного приложения?
9. Как процесс публикации в App Store и Google Play может влиять на тестирование?

СПИСОК ЛИТЕРАТУРЫ

1. Старолетов, С. М. Основы тестирования и верификации программного обеспечения / С. М. Старолетов. — 3-е изд., стер. — СПб : Лань. URL: https://e.lanbook.com/book/319445 (дата обращения: 10.03.2025).
2. Морозова, Ю. В. Тестирование программного обеспечения : учебное пособие / Ю. В. Морозова. - Томск : Эль-Контент. URL: https://znanium.com/catalog/product/1845910 (дата обращения: 10.03.2025).
3. Игнатьев, А. В. Тестирование программного обеспечения / А. В. Игнатьев. — 3-е изд., стер. — СПб : Лань. URL: https://e.lanbook.com/book/269873 (дата обращения: 10.03.2025).