Во всех лабораторных кроме 5 и 9 приложения для тестирования должны написать сами студенты!!!

В отчеты необходимо включать как код разработанного ПО, так и все что сделано в лабораторной работе в плане тестирования ПО.

Лабораторная работа 1: Основы тест-дизайна и создание тестовой документации

Цель: Отработать навыки написания тестовой документации (тест-план, чек-лист, тест-кейс) на основе требований к простому функционалу.

Объект тестирования: Калькулятор (веб- или десктопное приложение) или форма регистрации на сайте.

Задание (Ручное тестирование):

1. На основе предоставленных требований составьте тест-план (цель, объем, стратегия, риски).

2. Разработайте чек-лист для проверки основных функций (арифметические операции, очистка поля).

3. Напишите 5-10 детальных тест-кейсов (с шагами, ожидаемым результатом) для позитивных и негативных сценариев (например, деление на ноль).

Лабораторная работа 2: Функциональное ручное тестирование UI и составление баг-репортов

Цель: Закрепить навыки исследования приложения, выявления дефектов и их оформления в баг-трекинговой системе.

Объект тестирования: Простое веб-приложение (например, Todo List, трелло-доска) с преднамеренно допущенными багами.

Задание (Ручное тестирование):

1. Проведите исследовательское (ad-hoc) тестирование приложения.
2. Используя чек-лист из ЛР №1 (адаптированный под новое приложение), проведите тестирование по сценариям.
3. Обнаруженные дефекты оформите в виде баг-репортов (Jira, Youtrack или просто шаблон в Excel). В отчете должны быть: заголовок, шаги воспроизведения, фактический/ожидаемый результат, severity/priority, окружение.

Лабораторная работа 3: Написание первых автотестов на Python (или Java) + Selenium WebDriver

Цель: Познакомиться с базовым синтаксисом языка, инструментами и написать первый UI-автотест.

Объект тестирования: Страница логина любого публичного веб-сайта или специально подготовленный стенд.

Задание (Автоматическое тестирование):

1. Установите и настройте окружение: Python, pip, библиотеки `selenium`, `webdriver-manager`, `pytest`.
2. Напишите скрипт, который инициализирует браузер (Chrome/Firefox) с помощью WebDriver, открывает страницу и закрывает браузер.
3. Напишите автотест, который вводит валидные credentials на странице логина и проверяет успешный вход (например, появлению элемента "Logout").

Задание (Ручное тестирование): Проанализируйте, какие элементы на странице имеют стабильные селекторы (id, name) для использования в автотестах.

Лабораторная работа 4: Автоматизация позитивных и негативных UI-сценариев

Цель: Научиться структурировать код автотестов (Page Object Pattern) и handle-ить различные UI-сценарии.

Объект тестирования: Веб-форма (например, форма оформления заказа или контактная форма).

Задание (Автоматическое тестирование):

1. Реализуйте базовый класс `BasePage` и класс `ContactPage` по паттерну Page Object.
2. Напишите позитивный автотест: заполнение всех полей формы валидными данными, отправка и проверка успешного сообщения.
3. Напишите негативный автотест: попытка отправить форму с пустым обязательным полем и проверка текста ошибки.

Задание (Ручное тестирование): Составьте матрицу принятия решений для полей формы, чтобы определить все возможные валидные/невалидные комбинации.

Лабораторная работа 5: Тестирование REST API с помощью Postman и Python

Цель: Освоить основы тестирования API: отправка запросов, валидация ответов, написание скриптов для автоматических проверок.

Объект тестирования: Публичное REST API (например, https://reqres.in/ или https://jsonplaceholder.typicode.com/).

Задание (Ручное тестирование):

1. В Postman создайте коллекцию запросов: GET (получить пользователя), POST (создать пользователя), PUT (обновить).
2. Для каждого запроса напишите тесты на JavaScript (проверка status code, структуры JSON, значений полей).
3. Запустите коллекцию как collection run и проанализируйте результаты.

Задание (Автоматическое тестирование):

1. Напишите скрипт на Python с использованием библиотеки `requests` и `pytest`, который повторяет логику тестов из Postman.

Лабораторная работа 6: Написание модульных тестов (Unit Tests)

Цель: Понять принципы unit-тестирования и научиться тестировать изолированные функции и методы.

Объект тестирования: Класс или набор функций с бизнес-логикой (например, класс `Calculator` с методами `add`, `divide`, `is_prime_number`).

Задание (Автоматическое тестирование):

1. Используя фреймворк `unittest` или `pytest`, напишите модульные тесты для всех методов.
2. Используйте технику параметризации для тестирования с разными наборами данных.
3. Для метода `divide` убедитесь, что тест проверяет возникновение исключения (exception) при делении на ноль.

Лабораторная работа 7: Тестирование мобильного приложения

Цель: Получить базовый опыт в тестировании мобильных приложений на эмуляторе/реальном устройстве.

Объект тестирования: Простое мобильное приложение (например, стандартные "Часы" или "Заметки" на iOS/Android).

Задание (Ручное тестирование):

1. Составьте чек-лист для тестирования мобильного приложения с учетом специфики: жесты, ориентация экрана, прерывания (звонки, SMS), работа с памятью.
2. Проведите тестирование по чек-листу.

Задание (Автоматическое тестирование):

1. Установите Appium Server и настроить окружение.
2. Напишите простой скрипт (на основе ранее изученного Selenium WebDriver), который запускает приложение на эмуляторе и выполняет одно простое действие (тап по кнопке).

Лабораторная работа 8: Основы нагрузочного тестирования с k6

Цель: Познакомиться с концепцией нагрузочного тестирования и написать простой скрипт.

Объект тестирования: Простое API или статичная веб-страница.

Задание (Автоматическое тестирование):

1. Установите и настройте k6.
2. Напишите скрипт, который создает виртуальных пользователей (VUs), отправляющих GET-запрос на выбранный endpoint.
3. Запустите тест с разными параметрами нагрузки (например, 10 VUs на 30s и 50 VUs на 10s).
4. Проанализируйте выходные данные: количество запросов, время ответа, процент ошибок.

Задание (Ручное тестирование): Сформулируйте цели теста (что мы хотим проверить? пропускную способность? устойчивость?).

Лабораторная работа 9: Интеграция автоматизированных тестов в CI/CD (GitHub Actions)

Цель: Научиться запускать автотесты автоматически при коммите кода в репозиторий.

Объект тестирования: Код автотестов из предыдущих лабораторных работ (например, UI-тесты или API-тесты).

Задание (Автоматическое тестирование):

1. Создайте публичный репозиторий на GitHub и загрузите код ваших автотестов.
2. Напишите YAML-конфигурационный файл для GitHub Actions (.github/workflows/run-tests.yml).
3. Настройте workflow для: установки ОС, установки Python, зависимостей, запуска ваших тестов с помощью `pytest`.
4. Сделайте push в репозиторий и убедитесь, что workflow запустился и тесты прошли успешно (зеленая галочка).