Тестирование ПО – проверка соответствия ожидаемого и реального поведения программы (ни в коем случае не поиск багов!!!!).

# Модели разработки

* Водопадная модель (ступенчатая; изначально дорогое удовольствие, однако используется);
* V-образная модель (чем дальше этап, тем дороже продукт, его разработка);
* Инкрементальная модель (редко используется);
* Спиральная модель (тоже редко);
* Гибкая модель (SCRUM, много где работают по ней или же стараются со спринтом);
* Модель хаоса (самая важная задача в первую очередь, часто юзается).

Этапы SCRUM’а:

1. Идея (особо не принимаем участие)
2. Разработка некой документации, проработка/анализ требований (тестирование документации, требований);
3. Передача требований на разработку (тестеры пишут тест-кейсы, чек-листы, по чему в будущем будет тестироваться продукт);
4. Тестирование
5. Отчет (резолюция – готов/не готов)

Заведение дефектов (баги) – на каждом этапе может произойти.

# Объекты тестирования

1. Код (Code) (модульное тестирование);
2. Программное обеспечение (Software) – сайты, чат-боты, игры, мобильные приложения, интернет вещей (технологий Умный дом, часы), десктоп (приложения на компьютер) и другое (CRM???);
3. Прототипы проекта (Prototype) (перед релизом, тест продукта, которые еще полностью не готов);
4. Документация (Documentation) (проверка требований, проверка описание продукта) – ручное тестирование;
5. Железо (Hardware)

# Тестирование требований

Требования:

1. Прямые
2. Косвенные

Типы требований

* Функциональные, уровни:
  + Бизнес-требования;
  + Требования пользователей;
  + Функциональные требования.
* Нефункциональные (примеры: производительность, интерфейсы работы)

# Источники требований:

1. Заказчик;
2. Мозговой штурм;
3. Документы;
4. Фокус группа;
5. Наблюдение;
6. Моделирование;
7. Анкетирование;
8. Прототип;
9. Описание;
10. Нормы;
11. Лучшие практики;
12. Конкуренты.

# Характеристики требований:

* Завершенность;
* Непротиворечивость;
* Корректность;
* Недвусмысленность;
* Проверяемость;
* Атомарность (одно требование – для одного действия);
* Выполнимость;
* Обязательность.

# Тест-план

Тест-планы бывают:

1. Мастер план (высокоуровневая вещь, то есть без подробностей, на проекте есть только мастер-план) – требования к заводимым дефектам, условия принятия сборки, критерии принятия продукта к релизу, инструменты;
2. План (что надо протестировать, что БУДЕМ тестировать, как будем это делать, когда это будем делать, критерии сдачи, критерии приемки);
3. План приемо-сдаточных испытаний;

Что находится в тест-плане:

* Перечень работ;
* Критерии качества;
* Оценка рисков;
* Документация;
* Стратегия;
* Ресурсы;
* Расписание;

# Виды тестирования

Тестирование:

* По степени автоматизации:
  + Ручное;
  + Автоматизированное;
  + Полу автоматизированное.
* Виды тестирования по объектам:
  + Функциональное тестирование. Этапы:
    - Функциональное тестирование – поведение вашего продукта, что умеет делать;
    - GUI (графический интерфейс, UI – пользовательский интерфейс) тестирование элемента интерфейса;
    - Безопасность (продукт отвечает за безопасность – это его функционал);
  + Нефункциональное
    - Безопасность (защищенность продукта);
    - Тестирование интерфейса (UI);
    - Usability тестирование (как быстро пользователь достигнет цели, размер кнопок);
    - Локализации (по каждую страну – текст (смысловая нагрузка, синтаксис, грамматика, пунктуация, наличие текста), функционал (системы мер, валюта, формат времени или даты, региональные настройки, часовой пояс, календарь (буддийский, японский, григорианский), начало недели, формат бумаги, особенности имен, сортировка), региональные особенности (для этого лучше настоять на помощи со стороны (юриста); правовые, контроль символики и цветов, перевод на поддерживаемые языки, толкование текста, символов, знаков), контент (аудио, видео, изображения, текст, документация));
    - Интернационализация (локализации с переработками, например окна под текст, так как он может быть слева-направо, сверху-вниз – его нужно поменять, когда есть требование изначально разработать продукт с возможностью переделать на другую страну);
    - Конфигурационное (разная конфигурация под каждый браузер);
    - Совместимости (кроссбраузерное тестирование);
    - Инсталляционное (установка, обновление, удаление (но не всегда, например сайты));
    - Производительности (нагрузочное, стрессовое, стабильности (надежности, то есть может выдержать нагрузку в течение большего времени ~24 часа), масштабируемости ());
    - Тестирование на отказ и восстановление (помехоустойчивость);
    - Тестирование документации.

# Функциональное тестирование

СМЫСЛ: нужно проверить все функции программы и сравнить реальность с ожидаемыми результатами.



# Вид тестирований, связанных с изменениями

Виды:

* Дымовое тестирование (смок тест) – это вообще работает, надо проверить критически важный функционал и сделать это насквозь, то есть от начала и до конца, если не прошел этот тест билд, то он отправляется обратно на доработку, не на дальнейший тест;
* Регрессионное тестирование – проверка старого функционала с новыми доработками, может придется тестировать и новый функционал, если не уверены;
* Тестирование сборки – тестирование конкретной сборки, определенной версии до конца, скорее всего подразумевается уже готовый продукт, реже какую-то часть под сборку;
* Санитарное тестирование – тестирование конкретного небольшого участка, обычно это проверка той части, которая чаще всего выходит из строя;
* Также есть ретесты – когда нашли дефект, передали его разработчику, он исправил и отдал обратно, надо снова проверить, убедиться, что изменения внесены и дальше отдать на развертывание или, если есть ошибка, то обратно разрабу.

# Уровни тестирования, системное, интеграционное, модульное

* Модульное тестирование (юнитесты) – проверка маленького кусочка программы, маленькую логику;
* Интеграционное тестирование;
* Системное тестирование;
* Приемочное тестирование.

# Команды и их обозначения по порядку

1. const puppeteer = require('puppeteer');

Вот это строчка обозначает подключение модуля

Т.е в данном случае мы подключили к библиотеки Puppeteer

1. function name (variable) { ... } - это объявление глобальной функции js.

У нас в коде это функция имеет отношение к планированию - setTimeout. То есть мы, например, хотим поставить паузу в какой-то момент выполнения кода.

1. async -это объявления блока асинхронным. К этому мы еще вернемся что он вообще делает. Это особенность для puppeteer
2. const browser = await puppeteer.launch({headless: false})

здесь же мы объявляем переменную(const)

browser

1. await puppeteer.launch({headless: false})

Здесь задаем команду запуск браузера. У нас прописано {headless: false} - это значит что безголовой(headless) режим отключен. Безголовой режим - это запуск браузера без оболочки

1. const page = await browser.newPage()

Здесь объявили переменную page.

это переменная обращается к ранее созданной переменной browser.

и browser прописана функция newPage(). Нужен для взаимодействия с браузером.

1. await page.goto(link) – Задаем команду перейти по ссылке
2. await page.keyboard.press(' имя\название кнопки клавиатуры '); - Чтобы использовать эмуляцию кнопки, надо ввести в команду
3. console.log(“”Текст”) – вывод на консоль текста
4. let namePages = [''] - Туда мы вносим переменные в массив. Массивы это место, где хранятся несколько данных со строковыми ключами
5. for (...){

...

}

Это объявления цикла. Внутри в скобках прописываем условия цикла.

У нас данном случае условие такое - for (const namePage of namePages) {}

Мы из массива namePages берем по одному значению и присваиваем его как namePage.

И цикл будет будет работать до тех пор, пока не закончатся списки из массива

1. const { scrollPageToBottom } = require('puppeteer-autoscroll-down'); - подключение библиотеки пролистывания странички
2. await scrollPageToBottom(page, {

size: 500,

delay: 250,

stepsLimit: 3

}) - scrollPageToBottom внутри прописаны параметры

size: 500 - это размер пикселей, которые буду проматывать вниз

то есть пропуск 500 пикселей

delay - пауза пропуска

stepsLimit - это сколько раз надо пропускать

то есть, пропускаем 500 раз по три раза. в итоге 1500

# Узнать координаты нажатия

1. Выбрать вкладку "Console"

2. Вставить код

document.onclick = function(e)

{

var x = e.pageX;

var y = e.pageY;

alert("User clicked at position (" + x + "," + y + ")")

};

3. Нажать на кнопку "Ентер"

4. Нажать левой кнопкой мышки на любой элемент сайта

# Обновить страницу

await page.reload();

# Условия

try{

await (await page.waitForXPath ('//input [@id = "searchge554656tyugygjh"]')).type("QA")

console.log("Не работает")

} catch(err){

await (await page.waitForXPath ('//input [@id = "search"]')).type("QA")

console.log("Работает")

}

# Команда для создания скриншота

await page.screenshot({path: 'screen.png'});

# Функция выделения текста в консоли цветом

async function redLog(str){

console.log("\x1b[41m%s\x1b[0m", str)

}

# Отправлять код в правильной табуляции

Когда присылаешь код - выделяй все(ctrl+a) и нажми ctrl+shift+m.

# Количество строк с XPathом

const countText = await page.$x("//div[@class='inventory\_item\_name']")

// const read = (await page.$x("(//div[@class='inventory\_item\_name'])")).length

const read = countText.length

console.log(read)

# Drag and Drop

сначала надо использовать move(xy) на кубик А, чтоб навестить мышкой

потом down() - это удержать кнопкой мыши

снова move(xy), куда мы хотим перенести. В данном случае перекидываем на куб Б

потом up(), чтобы отпустить мышку

# Код от 27.06.2022 с циклом

const puppeteer = require('puppeteer');

function delay(time) {

  return new Promise(function(resolve) {

      setTimeout(resolve, time)

  });

}

(async () => {

  const browser = await puppeteer.launch({headless: false})

  const page = await browser.newPage()

  await page.goto('https://shazoo.ru/')

  await page.setViewport({width: 1000, height: 600})

  await delay(5000)

  let namePages = ['Игры', 'Гайды', 'VR', 'Кино', 'Сериалы', 'Аниме', 'Дизайн и Арт', 'Гаджеты', 'Железо', 'Наука', 'IT и Технологии'];

  for (const namePage of namePages) {

      try {

        await delay (3000)

        await page.mouse.move (176,30)

        await (await page.waitForXPath("//nav[@class='flex h-full gap-3 px-4']/div/ul/li/a[text()='"+namePage+"']")).click()

          console.log("\x1b[32m%s\x1b[0m","Раздел "+namePage+" найден");

      } catch (err){

          console.log("\x1b[41m%s\x1b[0m","Раздел "+namePage+" не найден");

      }

  }

})()

# Код от 27.06.2022 с использованием двумерного массива

const puppeteer = require('puppeteer');

const { scrollPageToBottom } = require('puppeteer-autoscroll-down');

function delay(time) {

  return new Promise(function(resolve) {

      setTimeout(resolve, time)

  });

}

async function redLog(str){

  console.log("\x1b[41m%s\x1b[0m", str)

}

(async () => {

  const browser = await puppeteer.launch({headless: false})

  const page = await browser.newPage()

  await page.goto('https://garrypotter.net/')

  await delay(5000)

  let nameParts = [{name: 'Книги о Гарри Поттере', good: '1'}, {name: 'Гарри Поттер на английском языке', good: '2'},

  {name: 'Настольные игры Гарри Поттер', good: '3'}, {name:'Новогодние игрушки', good:'4'}, {name:'Ручки Гарри Поттер', good:'5'}]

  let qty = ['1', '2', '3', '4', '5']

  //Добавляем все товары в корзину

for (const namePart of nameParts){

  await delay(1000)

  await(await page.waitForXPath("//button//span[text()='Каталог']")).click()

  await delay(1000)

  await(await page.waitForXPath("//div[@class='catalog-btn\_\_menu-item']/a[text()='"+namePart.name+"']", {timeout:70000})).click()

  //Проверка на правильность открывшегося раздела

  try{

    console.log("Открыт верный раздел")

    await page.waitForXPath("//h1[text()='"+namePart.name+"']")

} catch(err){

    redLog("Открыт неверный раздел")

}

//Добавление в корзину товара

await(await page.waitForXPath("(//button[@title='Купить'])["+namePart.good+"]")).click()

await(await page.waitForXPath("//button[@class='modal\_\_close']")).click()

//Переход к главной странице

await(await page.waitForXPath("//a[@title='Каталог товаров']")).click()

}

//Переход к корзине

await(await page.waitForXPath("//button[@title='Корзина']")).click()

try{

    console.log("Корзина открыта")

    await page.waitForXPath("//span[text()='Корзина заказов']")

} catch(err){

    redLog("Корзина не открыта")

}

//Проверка на наличие товаров в корзине

for (num of qty){

    await delay(1000)

    try{

        console.log("Товар в корзине")

        await page.waitForXPath("(//div[@class='cart\_\_product cart\_\_row'])["+num+"]")

    } catch(err){

        redLog("Товар не найден в корзине")

    }

}

//Удаляем первый товар

console.log ("Удаляем товар")

await(await page.waitForXPath("(//div[@class='cart\_\_product-table']//span[@title='Удалить товар'])[1]")).click()

for (num of qty){

    await delay(1000)

    try{

        await page.waitForXPath ("(//div[@class='cart\_\_product-table']//span[@title='Удалить товар'])["+num+"]")

    } catch(err){

        console.log("Товар удален")

    }

}

await delay(1000)

await browser.close()

})()