Puppeteer

Часть 1

Василий Петров
Разработчик Python, JavaScript

Василий Петров

О спикере:

- Стаж работы в IT более 25 лет
- Разрабатывал корпоративные приложения
- Руководил проектами и IT-подразделениями
- Руководил собственным бизнесом
- Участвует в различных проектах с применением Python и JavaScript

Цели занятия

1. Узнаем, что такое Puppeteer, для чего он нужен
2. Разберёмся, как его устанавливать и настраивать
3. Сравним синхронную и асинхронную работу приложений
4. Напишем первые тесты с Puppeteer, используя дебаггинг, хуки и различные конфигурации

План занятия

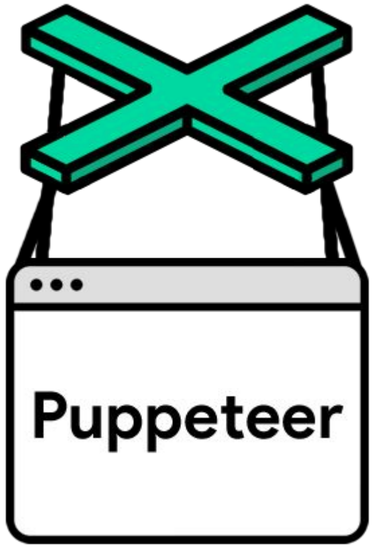
1. Что такое Puppeteer
2. Установка и настройка
3. Структура тестов
4. async/await
5. Первый тест
6. Итоги
7. Домашнее задание

Что такое Puppeteer

1

Что такое Puppeteer

- Фреймворк для end-to-end тестирования
- Разработан командой Google Chrome Team
- Основывается на JavaScript
- Настраивается за 5 минут
- Быстрый и стабильный



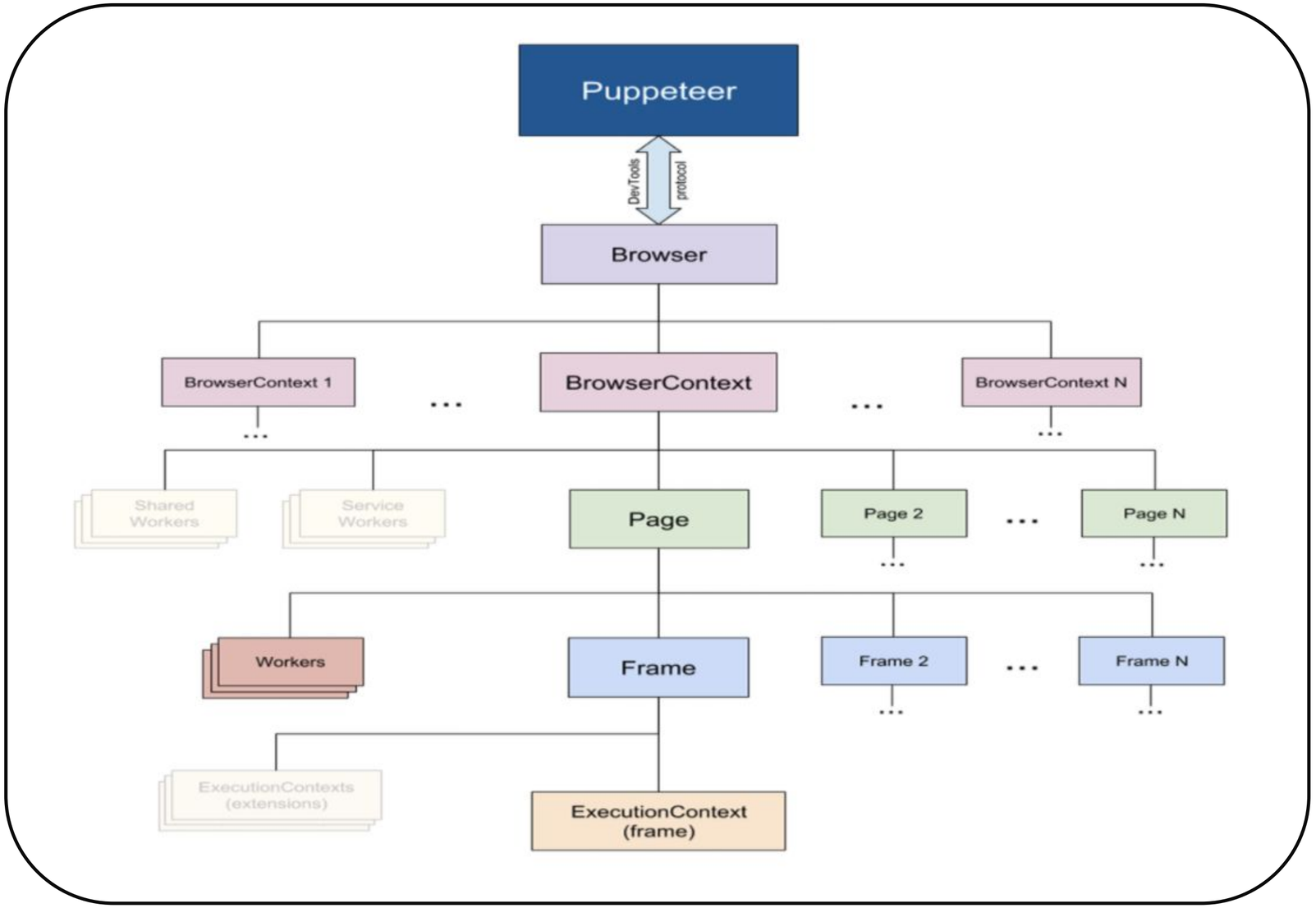
Что может Puppeteer

- Работает с Chromium и Chrome. С Firefox — экспериментально
- Работа с веб-элементами
- Поддержка разных браузеров и устройств
- Скриншоты
- Управление окружениями
- Тестирование расширений
- И многое другое

Как работает Puppeteer

- Puppeteer взаимодействует с браузером через DevTools Protocol
- Browser позволяет использовать несколько контекстов для браузера
- BrowserContext проясняет контекст браузера, позволяет использовать несколько окон браузера
- Page — сущность, которая имеет как минимум один frame. Может быть несколько pages
- Frame имеет как минимум один контекст — там, где выполняется JavaScript. Frame может иметь дополнительный контекст, когда работает с расширениями Chrome
- Worker имеет только один контекст, с которым работает, а также управляет взаимодействием с WebWorkers

Как работает Puppeteer



Установка и настройка

2

Устанавливаем puppeteer

В командной строке выполняем:

```
npm init
...
npm install puppeteer
```

Получаем package.json

```
{
  "name": "puppeteer-test",
  "version": "1.0.0",
  "description": "First puppeteer tests",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "puppeteer","netology","test"
  ],
  "author": "Oksana Melnikova",
  "license": "ISC",
  "dependencies": {
    "puppeteer": "^10.1.0"
  }
}
```

Структура тестов

3

Структура

Puppeteer работает с асинхронным кодом, поэтому мы используем анонимную асинхронную функцию async/await:

```
(async () => {
  //тесты с await...
})();
```

async/await

async/await

async/await были добавлены в ES8 как альтернатива promises.
Использование async-функций позволяет избежать цепочек promises и большой вложенности кода.
С помощью async-функции мы можем работать с асинхронным кодом синхронно.

Синтаксис — два ключевых слова async и await:

- async — делает функцию асинхронной
- await — остановит выполнение асинхронной функции до тех пор,

  пока не будет выполнен код, возле которого используется await

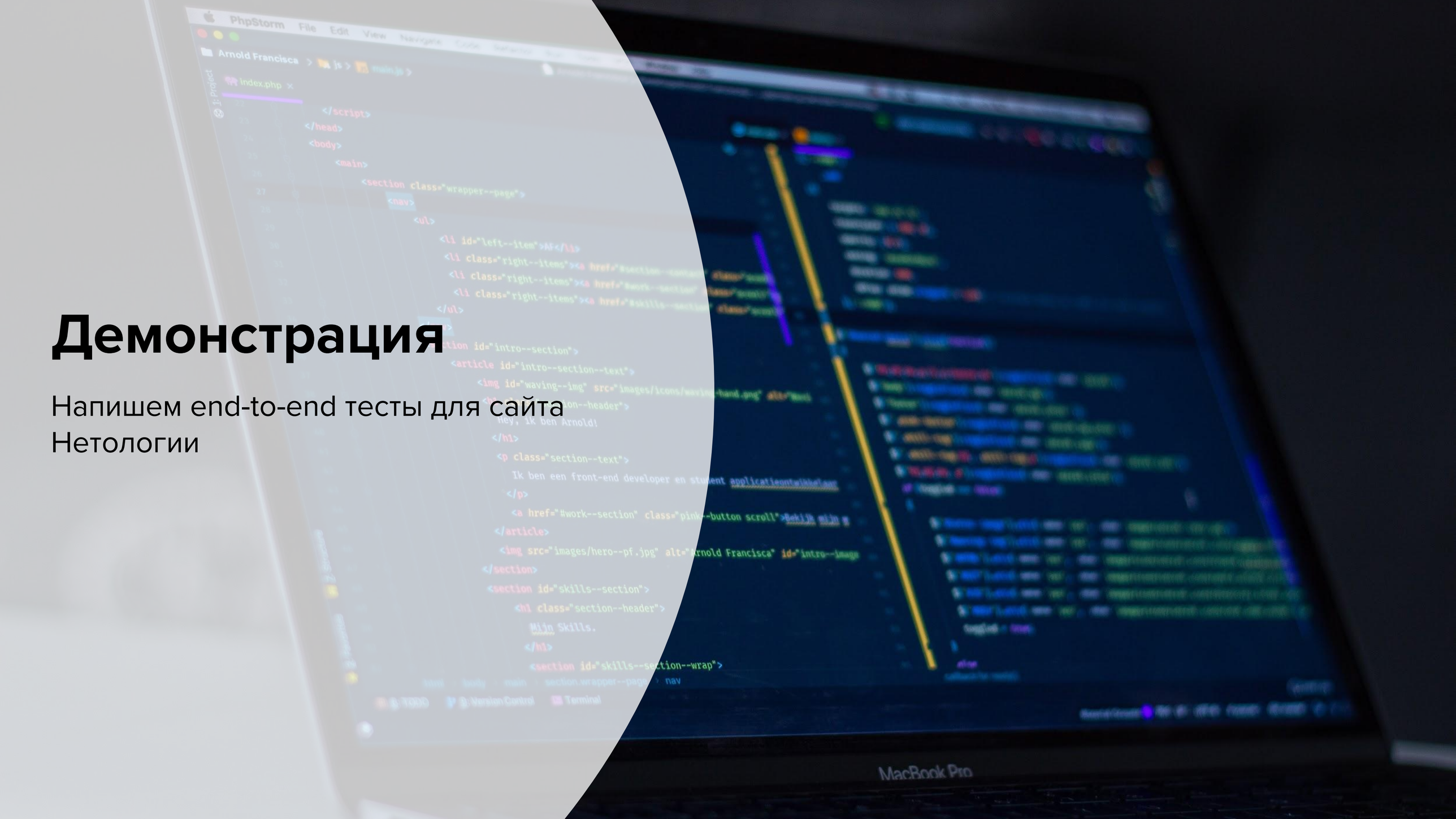
Пример использования async/await

```
//объявляем асинхронную функцию sum
async function sum() {
        //останавливаем выполнение sum до тех пор, пока a не будет присвоено значение 5
    const a = await 5;
        //останавливаем выполнение sum на 1 сек
        //и до тех пор, пока b не будет присвоено рандомное значение
    const b = await delayAndGetRandom(1000);
        //останавливаем выполнение sum на 1 сек
    await delayBeforeContinue(1000);
    return a + b;
  }
  // Вызов fn
  sum().then(console.log);


// delayAndGetRandom() and delayBeforeContinue() реализованы отдельно
```

Первый тест

5



Демонстрация

Напишем end-to-end тесты для сайта Нетологии

Первый тест

Создаём файл с нашим первым тестом с расширением *.test.js:

```
//импортируем библиотеку Puppeteer
const puppeteer = require("puppeteer");
// получаем нужный url как аргумент из командной строки
const [url] = process.argv.slice(2);
(async () => {
  // запускаем браузер
  const browser = await puppeteer.launch();
  // открываем новую страницу
  const page = await browser.newPage();
  // переходим на заданный URL
  await page.goto(url);
  // закрываем страницу и браузер
  await page.close();
  await browser.close();
})();
```

Запускаем первый тест

```
node first-test.test.js https://netology.ru
```

После запуска мы не видим результата тестов

Добавляем режим headed

По умолчанию тесты запускаются в режиме headless.

Для отладки, чтобы видеть работу браузера, необходимо добавить параметр headless: false в команду запуска браузера

(документация по launch()):

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

После запуска мы не видим результата тестов

Расширяем функционал теста

Page предоставляет множество методов для тестов.

Например, можно получить title страницы:

```
const title = await page.title();
console.log("Page title: " + title);
```

Перерыв

Разрешение браузера

Для тестирования мобильной версии браузера мы можем указать необходимый девайс:

```
await page.emulate(puppeteer.devices['iPhone 6']);
```

Список доступных девайсов.

Либо можно указать необходимое нам разрешение экрана

Работа с селекторами

Puppeteer работает с селекторами по аналогии с document.querySelector и document.querySelectorAll, используя $ и $$ соответственно:

```
const firstLink = await page.$("header a + a");
```

Чтобы получить текст элемента, воспользуемся методом $eval:

```
const firstLinkText = await page.$eval("header a + a", link => link.textContent);
```

Для взаимодействия с элементами мы должны использовать соответствующие методы:

```
await firstLink.click();
```

или:

```
await page.click("header a + a");
```

Работа с селекторами

Puppeteer работает с селекторами по аналогии с document.querySelector и document.querySelectorAll, используя $ и $$ соответственно:

```
const firstLink = await page.$("header a + a");
```

Чтобы получить текст элемента, воспользуемся методом $eval:

```
const firstLinkText = await page.$eval("header a + a", link => link.textContent);
```

Для взаимодействия с элементами мы должны использовать соответствующие методы:

```
await firstLink.click();
```

или:

```
await page.click("header a + a");
```

Работа с селекторами

Puppeteer работает с селекторами по аналогии с document.querySelector и document.querySelectorAll, используя $ и $$ соответственно:

```
const firstLink = await page.$("header a + a");
```

Чтобы получить текст элемента, воспользуемся методом $eval:

```
const firstLinkText = await page.$eval("header a + a", link => link.textContent);
```

Для взаимодействия с элементами мы должны использовать соответствующие методы:

```
await firstLink.click();
```

или:

```
await page.click("header a + a");
```

Работа с другой страницей

Puppeteer может работать с контекстом нескольких страниц. Откроем вторую страницу в другой вкладке:

```
const pageList = await browser.newPage();
await pageList.goto("https://netology.ru/navigation");
```

Добавим проверку, что страница загрузилась:

```
await pageList.waitForSelector('h1');
```

Также добавим команду закрытия новой вкладки:

```
await pageList.close();
```

Убедимся, что тест не будет проходить, если селектор неверный:

```
await pageList.waitForSelector('h123');
```

Работа с другой страницей

Puppeteer может работать с контекстом нескольких страниц. Откроем вторую страницу в другой вкладке:

```
const pageList = await browser.newPage();
await pageList.goto("https://netology.ru/navigation");
```

Добавим проверку, что страница загрузилась:

```
await pageList.waitForSelector('h1');
```

Также добавим команду закрытия новой вкладки:

```
await pageList.close();
```

Убедимся, что тест не будет проходить, если селектор неверный:

```
await pageList.waitForSelector('h123');
```

Работа с другой страницей

Puppeteer может работать с контекстом нескольких страниц. Откроем вторую страницу в другой вкладке:

```
const pageList = await browser.newPage();
await pageList.goto("https://netology.ru/navigation");
```

Добавим проверку, что страница загрузилась:

```
await pageList.waitForSelector('h1');
```

Также добавим команду закрытия новой вкладки:

```
await pageList.close();
```

Убедимся, что тест не будет проходить, если селектор неверный:

```
await pageList.waitForSelector('h123');
```

Работа с другой страницей

Puppeteer может работать с контекстом нескольких страниц. Откроем вторую страницу в другой вкладке:

```
const pageList = await browser.newPage();
await pageList.goto("https://netology.ru/navigation");
```

Добавим проверку, что страница загрузилась:

```
await pageList.waitForSelector('h1');
```

Также добавим команду закрытия новой вкладки:

```
await pageList.close();
```

Убедимся, что тест не будет проходить, если селектор неверный:

```
await pageList.waitForSelector('h123');
```

devtools

Puppeteer может открывать консоль разработчика во время исполнения теста.

Для этого необходимо добавить в команду запуска браузера:

```
devtools: true;
```

Улучшаем запуск тестов

Чтобы организовать более сложную структуру тестов и их запуск, воспользуемся уже знакомой нам библиотекой Jest.

Подключим её, а также библиотеку для работы Jest с puppeteer к нашему проекту:

```
npm install jest-puppeteer jest
```

В наш проект автоматически добавились зависимости:

```
"jest": "^27.0.6",
"jest-puppeteer": "^5.0.4",
```

Добавляем в package.json команду для запуска тестов, используя Jest

```
"scripts": {
    "test": "jest"
 }
```

Настраиваем конфигурации запуска новых библиотек

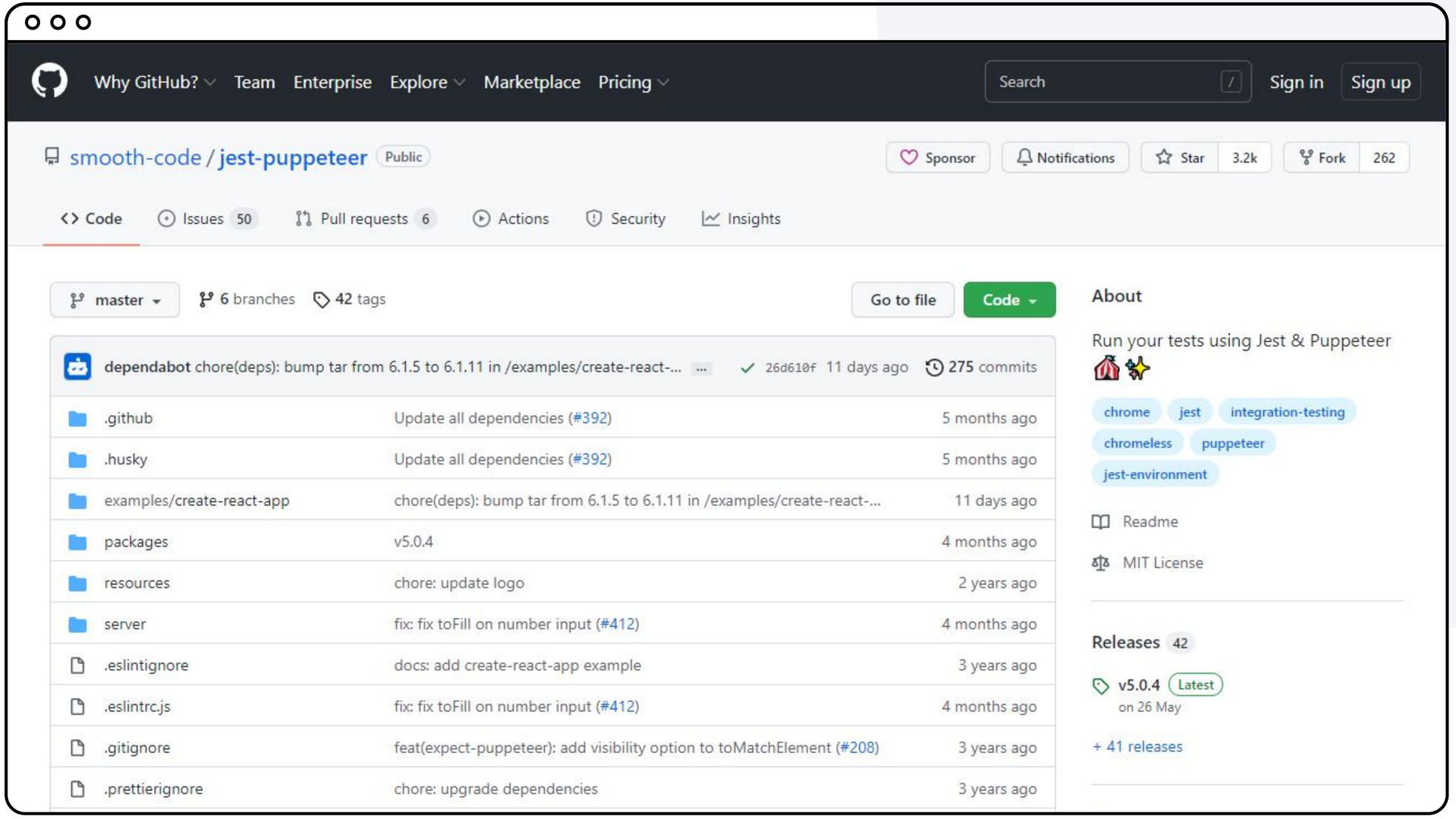
Создаём файл jest.config.js с конфигурацией Jest:

```
module.exports = {
    verbose: true, //указывает на то, что каждый тест будет показан в отчете о процессе запуска
     preset: "jest-puppeteer" //указываем, что будем использовать эту библиотеку
 };
```

И для второй библиотеки jest-puppeteer.config.js:

```
module.exports = {
  launch: {
    // здесь можем указывать все глобальные параметры запуска браузера для функции launch()
    slowMo: 1000
  }
};
```

Документация jest-puppeteer



https://github.com/smooth-code/jest-puppeteer

Перенесём настройки запуска браузера в config

jest-puppeteer.config.js:

```
module.exports = {
  launch: {
    slowMo: 1000,
    headless: false,
    defaultViewport: null,
    args: ['--start-maximized'] — используем максимальный размер окна браузера
  },
};
```

Теперь мы можем удалить эту команду запуска браузера из самих тестов:

```
const browser = await puppeteer.launch();
```

Запустим наши тесты

```
npm test
```

Теперь визуализация результатов гораздо лучше, но сами тесты не запускаются:

```
...
 FAIL  ./first-test.test.js
  ● Test suite failed to run
    Your test suite must contain at least one test.
...
Test Suites: 1 failed, 1 total
Tests:       0 total
Snapshots:   0 total
Time:        11.039 s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
```

Добавление структуры тестов Jest: 1 из 2

Теперь нам необходимо организовывать наши тесты, используя стандартную структуру Jest.

Организуем наши тесты в несколько тест-кейсов:

```
test("The title is 'Нетология...'", async () => {
  page = await browser.newPage();
  await page.goto("https://netology.ru");
  const actual = await page.title();
  expect(actual).toEqual("Нетология – курсы и обучение интернет-профессиям онлайн");
});
```

Добавление структуры тестов Jest: 2 из 2

```
test("The first link text 'Медиа'", async () => {
    page = await browser.newPage();
    await page.goto("https://netology.ru");
    const actual= await page.$eval("header a + a", link => link.textContent);
    expect(actual).toContain("Медиа Нетологии");
  });

  test("The first link leads on 'Медиа' page", async () => {
    page = await browser.newPage();
    await page.goto("https://netology.ru");
    await page.click("header a + a");
    const actual = await page.title();
    expect(actual).toEqual("Медиа Нетологии: об образовании в диджитале");
  });
```

Организуем набор тестов

Определим набор тестов при помощи describe():

```
describe("Netology.ru tests", () => {
    ///тест-кейсы
});
```

Применим подход DRY

DRY — Don’t Repeat Yourself.

Вынесем повторяющийся код в хуки beforeEach(), afterEach() перед всеми тест-кейсами внутри describe():

```
let page;
describe("Netology.ru tests", () => {
      beforeEach(async () => {
    page = await browser.newPage();
    await page.goto("https://netology.ru");
  });
      afterEach(() => {
        page.close();
      });
      ///тест-кейсы
});
```

Добавим ещё тесты

Добавим тесты для другой страницы вне блока describe():

```
test("The h1 should contain 'Работа'", async () => {
  const expected = "Работа";
  await page.goto("https://netology.ru/job");
  const actual = await page.$eval("h1", (link) => link.textContent);
  expect(actual).toContain(expected);
});
```

Добавим новые хуки

Наши хуки beforeEach() и afterEach() не будут действовать вне describe(), поэтому мы перенесём часть общих команд для запуска вкладки и закрытия браузера в хуки beforeEach() и afterEach() вне describe(). Тогда они будут срабатывать для всех тестов в этом файле:

```
let page;
beforeEach(async () => {
  page = await browser.newPage();
});
afterEach(() => {
  page.close();
});
describe("Netology.ru tests", () => {
      beforeEach(async () => {
    await page.goto("https://netology.ru");
///тест-кейсы
  });
});
///тест-кейсы
```

Итоги

1. Познакомились с библиотекой Puppeteer
2. Узнали основную структуру тестов
3. Ближе освоили async/await
4. Разобрались с before/after хуками
5. Подключили Jest для управления тестами и ассершенами

Домашнее задание

Давайте посмотрим ваше домашнее задание:

- Вопросы по домашней работе задавайте в чате группы
- Задачи можно сдавать **по частям**
- Зачёт по домашней работе проставляется после того, как приняты **все задачи**

Задавайте вопросы и пишите отзыв о лекции