# AntiBotBrowser: Теория обхода антибот-детекторов и разбор кода

## Введение

Современные веб-сайты широко применяют системы выявления ботов, чтобы блокировать автоматизированные скрипты доступа. Даже такие мощные инструменты, как Playwright, по умолчанию легко распознаются — например, при безголовом режиме (headless) он устанавливает флаг navigator.webdriver = true, что сразу выдает автоматизацию[[1]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=The%20main%20issue%20with%20the,true). Антибот-механизмы отслеживают множество признаков: специальные свойства браузера, шаблоны HTTP-заголовков, частоту запросов, наличие пользовательских действий на странице и др. Например, отсутствие движений мыши или клавиатуры, неестественно **частые запросы** или **статический IP-адрес** могут вызвать подозрение у системы защиты[[2]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Raw%20behavioral%20signals%20can%20be,can%20indicate%20a%20bot%20activity)[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix). Чтобы обходить такие защиты, класс **AntiBotBrowser** реализует комплекс мер: использование прокси-серверов для смены IP, рандомизация заголовков User-Agent, запуск браузера в стелс-режиме (скрытие факта автоматизации), эмуляция действий пользователя и регулирование скорости запросов. Ниже мы подробно разберем каждую из этих мер, объясним теоретические основы и укажем, на что обратить внимание.

## Цели и возможности AntiBotBrowser

**AntiBotBrowser** создан для безопасного запуска браузера **Playwright** с минимальным риском быть обнаруженным как бот. Класс оборачивает открытие браузера в контекстный менеджер Python, который автоматически настраивает и закрывает браузер. Его основные возможности и стратегии обхода антибот-детекторов:

* **Смена IP через прокси.** При запуске браузера класс может подключаться через последовательность прокси-серверов, меняя IP-адрес до тех пор, пока не будет найден рабочий прокси. Это предотвращает блокировки по IP и распределяет нагрузку между адресами[[4]](https://blog.apify.com/rotating-proxies/#:~:text=If%20you%27re%20serious%20about%20web,requests%20from%20a%20single%20IP).
* **Ротация заголовка User-Agent.** Перед открытием страницы выбирается случайный **User-Agent**, имитирующий разные браузеры и устройства. Это позволяет не светиться одним и тем же агентом, что могло бы вызвать флаги у антибот-систем[[5]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Great%21%20You%27ve%20successfully%20set%20up,a%20custom%20Playwright%20User%20Agent).
* **Стелс-режим (скрытие автоматики).** AntiBotBrowser старается скрыть признаки автоматизации: например, через *stealth*-скрипт, который отключает видимые флаги вроде navigator.webdriver и убирает отметку "HeadlessChrome" из сигнатуры браузера[[6]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=,in%20headless%20mode%20by%20default)[[7]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=is%20the%20,undefined). Таким образом скрипт маскирует безголовый браузер под обычный.
* **Эмуляция действий пользователя.** После загрузки страницы AntiBotBrowser **имитирует движения мыши и прокрутку** с небольшими случайными задержками. Это добавляет человеческих поведенческих сигналов, чтобы бот не выглядел “сверхъестественно” быстрым и статичным[[8]](https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a#:~:text=Websites%20often%20track%20user%20interactions,mouse%20movement%20function%20can).
* **Обработка капчи и лимитов.** Класс отслеживает, не вернул ли сайт ответ *«429 Too Many Requests»* (слишком много запросов). При обнаружении он предпримет повторную попытку загрузки страницы после паузы и не станет бездумно спамить запросами[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix). Кроме того, предусмотрена возможность подмешивать заранее полученные **cookies** (например, Google) для обхода капчи или повторного входа без дополнительной проверки[[9]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=name%3Dvalue%3B%20name2%3Dvalue2%3B%20name3%3Dvalue3).
* **Безопасное завершение и очистка.** При выходе из контекстного менеджера AntiBotBrowser гарантированно закрывает страницу, контекст и сам браузер, освобождая ресурсы. Также он логирует текущее прокси-IP и ошибки, что помогает контролировать процесс.

Далее рассмотрим каждую из этих составляющих, а также теорию, почему они необходимы и как работают.

## Использование прокси для смены IP

Один из главных признаков бота — большое число запросов, исходящих с одного IP-адреса. Многие сайты ограничивают количество запросов с одного клиента (IP) в единицу времени и при превышении выдают ошибку **HTTP 429 (Too Many Requests)**[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix). Чтобы избежать такого блокирования, **AntiBotBrowser** применяет прокси-серверы.

При инициализации класс загружает список прокси (например, из переменных окружения) и пытается последовательно подключиться через каждый. Для каждого прокси выполняется проверка соединения – открывается специальная страница https://api.ipify.org для получения текущего внешнего IP. Если ответ получен, значит прокси работает, и этот IP будет использоваться (класс даже выводит его: *«ℹ️ Используется IP: ...»*). В случае ошибки или недоступности прокси, он логируется (*«⚠️ Прокси не работает...»*) и пробуется следующий в списке. Такой механизм гарантирует, что браузер запустится только через рабочий прокси, и вы не начнете scraping со сбоя или, что хуже, с реального IP.

**Теория и преимущества.** Ротация IP-адресов – **обязательное условие** для крупномасштабного сбора данных. Без смены IP даже легитимный парсинг быстро столкнется с блокировкой[[4]](https://blog.apify.com/rotating-proxies/#:~:text=If%20you%27re%20serious%20about%20web,requests%20from%20a%20single%20IP). Прокси позволяют распределить запросы между разными адресами, скрывая истинный источник. Особенно ценны *резиденциальные прокси* (IP обычных пользователей) или *мобильные*, так как они обладают высокой репутацией и меньше подозрений со стороны антибот-систем[[10]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=This%20fingerprint%20includes%20details%20about,be%20one%20of%20three%20types)[[11]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=track%2C%20cell%20tower%20connections%20are,IPs%20to%20browse%20the%20web). Напротив, **датацентровые прокси** (IP крупных хостингов) часто имеют низкую «оценку доверия», потому что известны как источники бот-трафика[[11]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=track%2C%20cell%20tower%20connections%20are,IPs%20to%20browse%20the%20web). AntiBotBrowser не различает типы прокси, но подразумевается, что вы можете передать ему список качественных адресов.

**Замечания и подводные камни.** Используя прокси, стоит учитывать: скорость ответа (некоторые прокси медленные), географию (сайт может показывать разный контент по регионам), а главное – **лимиты по IP всё равно остаются**, просто вы их распределяете. Если без прокси сайт позволял 100 запросов в минуту, то с 5 прокси можно условно сделать 5× больше, но **каждый отдельный прокси** не должен превышать лимит. Также важно не использовать один и тот же прокси слишком долго – **ротация** нужна не только при старте, но и в ходе работы (наш класс пока меняет IP только при запуске нового браузера). Если планируется длительный парсинг, нужно программно менять прокси или перезапускать AntiBotBrowser через определенное число запросов. Наконец, помните про авторизацию на прокси: класс поддерживает прокси с логином/паролем (формирует соответствующие настройки), но убедитесь, что эти креденшелы верны и не истекли.

## Ротация User-Agent заголовков

HTTP-заголовок **User-Agent** сообщает сайту информацию о браузере и ОС клиента. Многие антибот-системы отслеживают, какой User-Agent делает запросы. Если десятки или сотни запросов приходят с одним и тем же агентом, это подозрительно (реальные пользователи разнообразны)[[5]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Great%21%20You%27ve%20successfully%20set%20up,a%20custom%20Playwright%20User%20Agent). Более того, стандартные User-Agent от автоматизированных браузеров могут содержать явно бот-подобные маркеры. Например, безголовый Chrome добавляет строку "HeadlessChrome" в User-Agent, что служит явным индикатором бота[[6]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=,in%20headless%20mode%20by%20default). Поэтому AntiBotBrowser при каждом запуске контекста задает **случайный User-Agent** из заранее подготовленного списка.

В коде определен массив USER\_AGENTS с десятком строк, представляющих популярные браузеры: современные Chrome и Firefox на Windows и Linux, Safari на iPhone и Mac, мобильный Chrome на Android, даже Яндекс.Браузер. При запуске новый контекст Playwright создается с параметром user\_agent=random.choice(self.user\_agents\_list), т.е. выбирается один случайный вариант. В результате все запросы в этом браузерном контексте будут иметь данный агент.

**Почему это работает.** Рандомизация User-Agent затрудняет выявление бота по последовательности запросов. Вместо того чтобы все запросы шли как, скажем, *Chrome/103 Windows*, сегодня они могут выглядеть как **Chrome/114** на Android, завтра как **Firefox/102** на Ubuntu и т.д. Это имитирует трафик от разных устройств и пользователей[[5]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Great%21%20You%27ve%20successfully%20set%20up,a%20custom%20Playwright%20User%20Agent)[[12]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=Websites%20also%20use%20the%20User,avoiding%20CAPTCHAs%20while%20web%20scraping). Более того, установка кастомного агента позволяет убрать нежелательные пометки вроде "HeadlessChrome" – мы явно подставляем строку, похожую на реальный браузер (наш список UA уже не содержит слова "Headless"). Согласно рекомендациям, **необходимо менять User-Agent достаточно часто**, иначе даже уникальный, но статичный агент со временем могут заподозрить[[13]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=Websites%20also%20use%20the%20User,avoiding%20CAPTCHAs%20while%20web%20scraping).

**На что обратить внимание.** При смене User-Agent важно, чтобы остальные характеристики запроса и среды не противоречили ему. Если вы выдаёте себя за Safari на Mac, но при этом запускаете безголовый браузер на Linux-сервере, некоторые продвинутые системы это вычислят. Они могут сверить, например, шрифты и рендеринг Canvas или WebGL с заявленным агентом. Как отмечают специалисты, **несоответствие между заголовками и реальным окружением приводит к блокировке** – например, если User-Agent указывает Windows, а HTTP-заголовок Sec-CH-UA-Platform или другие признаки говорят о Linux[[14]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Remember%20to%20check%20that%20your,mismatch%20and%20block%20your%20requests). В нашем случае Playwright сам ставит связанные заголовки под выбранный User-Agent, но **стоит убедиться**, что ваш stealth-скрипт также маскирует наиболее явные нестыковки. Обычно безопаснее выбирать User-Agent, близкий к реальной среде запуска: если скрипт работает на сервере с Linux, можно использовать агенты Chrome/Firefox под Windows/Linux, а не мобильный Safari. Также регулярно обновляйте список агентов – софт выявления ботов знает самые популярные и устаревшие строки, лучше имитировать актуальные версии браузеров.

## Стелс-режим: сокрытие факта автоматизации

Даже при использовании прокси и реалистичного User-Agent, **браузер под управлением Playwright всё равно может быть распознан**, если не спрятать ряд специфических “меток” автоматизации. Разработчики антибот-систем накопили большой список трюков для выявления Selenium/WebDriver, Playwright и т.п. Например: свойство navigator.webdriver, отсутствие реальных плагинов, особые devtools-протоколы, наличие специфических JS-объектов, искажения в WebGL/Canvas, и многое другое[[15]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=,server%2C%20to%20appear%20more%20human)[[16]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=This%20involves%20signals%2C%20such%20as%3A). При безголовом запуске Chrome сам выставляет navigator.webdriver=true и может отразить в User-Agent слово "Headless". Если оставить всё как есть, сайт сразу узнает о боте.

**AntiBotBrowser** решает эту проблему через подключение **stealth-скрипта**. В конструкторе, после создания контекста, идет блок:

stealth\_path = "stealth.min.js"  
if os.path.exists(stealth\_path):  
 self.context.add\_init\_script(path=stealth\_path)

То есть, если в папке есть файл stealth.min.js, он будет выполнен на каждой загружаемой странице *до* любого пользовательского скрипта. Такой подход аналогичен подключению **Playwright Stealth Plugin** – набора патчей, скрывающих автоматизацию. В частности, типичный stealth-скрипт делает следующее:

* **Отключает флаг webdriver.** Например, выполняет код Object.defineProperty(navigator, 'webdriver', { get: () => undefined }), чтобы свойство navigator.webdriver возвращало undefined вместо true[[7]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=is%20the%20,undefined). Сайт, проверяющий window.navigator.webdriver, решит, что перед ним не автоматизированный браузер.
* **Удаляет метку Headless из User-Agent.** В случае если браузер запущен с опцией headless без кастомного UA, Playwright/Chromium добавляют "HeadlessChrome" в строку агента. Stealth-скрипты вырезают эту подстроку, делая User-Agent неотличимым от обычного[[6]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=,in%20headless%20mode%20by%20default).
* **Эмулирует наличие плагинов, WebGL, других объектов.** Часто добавляются фиктивные объекты в navigator.plugins и navigator.languages, исправляются свойства типа navigator.vendor (для Chrome это "Google Inc."), подставляются правильные platform и userAgentData. Также могут **подменяться функции**, которые выдавали бы типичные сигнатуры Selenium (например, в HTMLCanvasElement.toDataURL или func.toString() удаляются строки вроде "function () { [native code] }" или ссылки на драйвер)[[17]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Detecting%20forged%20and%20modified%20fingerprints%3A,server%2C%20to%20appear%20more%20human). Цель — устранить несовпадения и убрать признаки, которые **невозможны в обычном браузере** либо типичны только для ботов.
* **Маскировка DevTools-протокола.** Некоторые детекторы, как показано в примерах, могут через хитрые JS-ошибки выявлять использование CDP (Chrome DevTools Protocol)[[18]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Specially%20crafted%20bot%20challenges%3A%20For,like%20Puppeteer%2C%20Playwright%20and%20Selenium)[[19]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=This%20JavaScript%20detection%20technique%20leverages,when%20they%20communicate%20with%20WebSocket). Полный стелс-плагин старается и такие вещи обходить, хотя на практике это сложнее и выходит за рамки простого скрипта.

По сути, **стелс-режим пытается пройти “тест Тюринга” для браузера**. Например, существуют специальные страницы с набором проверок (как sannysoft, источник Puppeteer Stealth): цель в том, чтобы автоматизированный браузер успешно прошёл все тесты, показывая только ожидаемые для человека данные[[20]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=The%20objective%20of%20the%20Stealth,mentioned%20in%20the%20official%20documentation). Согласно BrightData, на начало 2024 года Playwright Stealth уже способен обойти большинство стандартных проверок[[20]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=The%20objective%20of%20the%20Stealth,mentioned%20in%20the%20official%20documentation). Однако гонка вооружений продолжается: **антибот-провайдеры** тоже добавляют новые проверки, поэтому важно держать stealth-скрипт актуальным. Наш AntiBotBrowser загружает файл *если он существует*, то есть вы сами можете обновлять stealth.min.js по мере появления новых версий.

**Примечание:** Playwright сам по себе предлагает некоторые опции для маскировки (например, можно запускать с флагом --disable-blink-features=AutomationControlled, который скрывает webdriver-флаг частично[[21]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Some%20signals%20are%20simpler,features%3DAutomationControlled%27%60%20chrome%20flag)). Тем не менее, встроенных средств мало, и использование стороннего скрипта или плагина значительно повышает шансы не быть заблокированным. В коде также предусмотрена функция \_inject\_google\_cookies — её можно использовать, чтобы добавить в контекст авторизационные куки или куки, полученные из реального браузера, что тоже повышает **легитимность** сессии. Например, если вы заранее прошли reCAPTCHA в обычном браузере и сохранили куки, их загрузка перед парсингом может избавить от необходимости решать капчу снова[[9]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=name%3Dvalue%3B%20name2%3Dvalue2%3B%20name3%3Dvalue3).

## Эмуляция действий пользователя

Одного технического маскирования недостаточно. Много современных антибот-систем анализируют **поведение** клиента на странице: двигает ли он мышкой, скроллит ли страницу, кликает ли элементы, как быстро происходят действия и пр.[[16]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=This%20involves%20signals%2C%20such%20as%3A). Чисто машинное поведение – например, мгновенная загрузка и извлечение данных без единого движения курсора или без пауз – сильно отличается от поведения живого пользователя и может быть зафиксировано скриптами. В ответ страница может потребовать капчу или вовсе заблокировать дальнейшие действия.

AntiBotBrowser старается придать автоматизации более «человеческий» облик. После загрузки каждой страницы (метод goto) выполняется \_emulate\_human(). Этот метод программно дергает мышь и колесо прокрутки:

* **Случайные перемещения курсора.** Код несколько раз выбирает случайные координаты в пределах окна браузера (x и y в диапазоне от 0 до ширины/высоты окна) и вызывает page.mouse.move(x, y). Между перемещениями делается небольшая пауза (от 100 до 500 мс). Такие рывки курсора создают иллюзию, что пользователь водит мышкой по экрану. Алгоритм не сложный – курсор просто прыгает в случайные места, – но это лучше, чем ничего. Известно, что боты часто двигают указатель идеально прямолинейно или телепортируют между элементами, тогда как реальные пользователи водят его более хаотично[[22]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=,have%20a%20less%20perfect%20behavior). Рандомные точки и задержки добавляют непредсказуемости, снижая линейность движения.
* **Прокрутка страницы.** После движений курсора метод выбирает случайную величину (например, 200–1000 пикселей) и выполняет page.mouse.wheel(0, scroll\_down) – прокручивает страницу вниз, затем через паузу прокручивает чуть вверх (scroll\_up на 50–300 пикселей). Это похоже на поведение читающего человека: немного пролистал, потом, возможно, вернулся назад. Резкие дерганья или непрерывная прокрутка сверху донизу могли бы выглядеть ненатурально, поэтому используется два шага с небольшой коррекцией вверх.
* **Случайные паузы.** Между всеми действиями (move и wheel) стоят time.sleep на случайный промежуток (десятые доли секунды). Это имитирует время реакции человека. Без задержек бот бы выполнял всё мгновенно, что тоже аномально. Дополнительные паузы можно добавить, например, имитируя время «размышления» перед кликом на ссылку.

**Эффективность подхода.** Хотя эти мероприятия довольно примитивны, они дают базовый уровень **поведенческой маскировки**. По крайней мере, скрипты, ожидающие увидеть хоть какое-то движение или прокрутку, не найдут страницу полностью статичной. Исследования показывают, что **природные, неидеальные движения** (с небольшими дрожаниями, разной скоростью) значительно усложняют задачу детектирования по поведению[[23]](https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a#:~:text=Bot%20detection%20mechanisms%20have%20become,curved%20mouse%20movements%20can%20help)[[8]](https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a#:~:text=Websites%20often%20track%20user%20interactions,mouse%20movement%20function%20can). Наш код не реализует сложные траектории — есть библиотеки, генерирующие реалистичные кривые Безье для движения мыши с «дрожью» (напр., Ghost Cursor)[[8]](https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a#:~:text=Websites%20often%20track%20user%20interactions,mouse%20movement%20function%20can), — но даже простые случайные сдвиги уже лучше, чем полностью отсутствующие.

Стоит упомянуть, что **поведенческий анализ** может включать и другие факторы. Например, время, проведенное на странице, последовательность действий (не заходит ли бот сразу на страницу покупки, минуя просмотр товара), ритмичность запросов на сервер[[24]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=such%20as%3A)[[25]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=,of%20request%20is%20almost%20zero). Боты часто работают по таймеру или очень равномерно. Наш AntiBotBrowser, помимо движений на странице, косвенно решает эту проблему путем **рандомизации интервалов**: время задержки перед загрузкой страницы (goto) и небольшие паузы после — все это разбавляет четкий ритм. Хорошей практикой будет и **внешнее** управление частотой запусков: не вызывать десятки страниц в секунду, а делать случайные паузы между навигациями. Исследования подтверждают, что отправка запросов с **постоянным интервалом** или без перерывов (24/7) однозначно указывает на бот-скрипт[[25]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=,of%20request%20is%20almost%20zero). Поэтому интервалы должны меняться, а ночью, например, трафик может вовсе останавливаться (имитируя сон пользователя)[[26]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=One%20of%20the%20most%20common,scale%20web%20scraping).

## Ограничение скорости и обработка блокировок (429 / CAPTCHA)

Даже приняв все меры предосторожности, можно столкнуться с ситуацией, когда сайт начал ограничивать доступ. Часто первым признаком служит **HTTP 429 Too Many Requests** – ответ сервера, сигнализирующий о превышении лимита частоты запросов[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix). В AntiBotBrowser реализовано обнаружение этого сигнала. Метод \_is\_rate\_limited() получает HTML содержимое страницы и проверяет наличие подстроки "429" и текста "That’s an error." (фраза, типичная для страниц ошибки Google). Если такая комбинация найдена, считается, что сработало ограничение.

В коде это используется в goto(): после загрузки страницы и эмуляции пользователя, если \_is\_rate\_limited() вернул True, класс выводит предупреждение и пытается **перезагрузить страницу** (page.reload()) один раз. Идея в том, что иногда разовый сбой или требование можно обойти повторным запросом (например, если в фоне сменился прокси или прошло достаточно времени). После перезагрузки AntiBotBrowser снова имитирует мышь и проверяет, исчезла ли ошибка. Если 429 остался, в логе будет сообщение, что страница по-прежнему ограничена. В таком случае дальнейшие попытки не делаются (нужно принять меры снаружи: сменить IP, подождать дольше или отказаться от запроса).

**Как поступать при 429.** Сервер дает понять, что вы сделали слишком много за отрезок времени[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix). Лучшее решение – отдохнуть. Например, некоторые ответы 429 содержат заголовок Retry-After с числом секунд ожидания[[27]](https://oxylabs.io/resources/error-codes/429#:~:text=,After%3A%203600). Можно парсить его и спать указанный промежуток. Если же такого заголовка нет, разумно сделать паузу несколько минут. AntiBotBrowser пока не реализует авто-паузы. **Ни в коем случае нельзя бесконечно перезагружать** страницу, игнорируя 429, – это лишь усугубит блок и может привести к более жестким мерам (например, **капче** или временной полной блокировке IP). Эксперты советуют прекращать или замедлять парсинг, если увидели всплеск ошибок, и адаптироваться перед продолжением[[28]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=One%20of%20the%20most%20suspicious,the%20cause%20and%20start%20investigating).

Говоря о капче: обычно появлению CAPTCHA предшествуют признаки подозрительной активности. Если вы достаточно “замаскировались” (IP, агенты, поведение) и не превышаете лимиты, капча может не появляться вовсе[[29]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=The%20recommended%20approach%20is%20to,services%20like%20web%20scraping%20APIs). Но если она возникла, то без решения задачи (вручную или через капча-сервис) дальше не пройти. AntiBotBrowser не содержит встроенного решения капчи (что ожидаемо, это сложная отдельная тема), однако предоставляет **возможность работать с cookies**, что может помочь. Метод \_inject\_google\_cookies() может принять список словарей с куками и добавить их в контекст браузера. Идея в том, что вы можете заранее получить **cookie**, подтверждающие что вы человек. Например, Google и Cloudflare после прохождения капчи ставят специальные cookies (GFSD, cf\_clearance и т.п.), по которым несколько последующих запросов не требуют капчи. Если такие куки сохранить и подгружать перед началом навигации, скрипт будет восприниматься как уже проверенный пользователь[[9]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=name%3Dvalue%3B%20name2%3Dvalue2%3B%20name3%3Dvalue3). Даже более простое применение — использовать куки авторизованного сеанса (если парсите сайт, где у вас есть логин), чтобы не проходить лишние проверки. AntiBotBrowser не заполняет куки сам (по умолчанию он даже вызывает self.context.add\_cookies([]) для демонстрации пустого списка), но вы можете расширить этот метод.

**Дополнительные меры.** При частых блокировках стоит пересмотреть стратегию: уменьшить скорость запросов, увеличить пул прокси, либо прибегнуть к специализированным сервисам (например, scraping API, которые берут решение капчи на себя). Также контролируйте **лог-файлы**: AntiBotBrowser выводит предупреждения о каждом сбое прокси или ограничении 429 — эти сигналы к тому, чтобы оператор скрипта вовремя заменил прокси или приостановил парсинг. Успешное обходение блокировок — это итеративный процесс: заметили, что текущие настройки больше не работают (возникли капчи, 429) — внесли коррективы, продолжаем.

## Завершение работы и очистка ресурсов

Класс AntiBotBrowser спроектирован как контекстный менеджер (методы \_\_enter\_\_ и \_\_exit\_\_), что облегчает безопасное закрытие браузера. Когда вы выходите из блока with AntiBotBrowser() as browser:, метод \_\_exit\_\_ автоматически выполнится и сделает:

* page.close() – закроет открытое окно страницы;
* context.close() – закроет контекст (профиль браузера);
* browser.close() – полностью завершит процесс браузера (Chromium/Firefox/WebKit, в зависимости от использования).

Также в коде предусмотрено закрытие при исключениях: если при попытке подключения через прокси произошла ошибка, класс ловит её, закрывает любые уже открытые страницы/контексты/браузеры, и переходит к следующему прокси. Это гарантирует, что не останется «висячих» браузерных процессов в фоне и не будет утечек памяти или дескрипторов. После исчерпания списка прокси без успеха, \_\_enter\_\_ бросит исключение, информируя, что ни один прокси не сработал. Такой подход лучше, чем молча использовать локальный IP – скриптер сразу узнает о проблеме и не начнёт невольно парсить с реального адреса.

**Практические советы:** Всегда оборачивайте использование AntiBotBrowser в with блок, как и задумано. Если нужно запустить много потоков или параллельных браузеров, создавайте соответствующее число контекстов (но учтите нагрузку на CPU/RAM). Не забывайте, что **каждый контекст** в Playwright изолирован: у них разные cookies, кэш, и даже User-Agent можно задавать разный. Это удобно для одновременного парсинга под разными личинами. Однако не запускайте сотни браузеров сразу – это само по себе может выдать бота по косвенным признакам на сервере (всплеск одновременных запросов даже с разных IP). Планируйте постепенный старт, следите за расходом ресурсов и соблюдайте лимиты.

## Рекомендации и подводные камни

AntiBotBrowser значительно облегчает обход типичных антибот-механизмов, но для успеха в реальных условиях следует учесть весь комплекс факторов. Ниже перечислим ключевые рекомендации и возможные трудности:

* **Используйте качественные прокси.** Сам скрипт умеет менять IP, но **качество прокси имеет значение**. Если вы взяли случайные бесплатные прокси, велика вероятность, что они уже в черных списках сайта. Предпочтительнее использовать доверенные источники: резидентские IP или надежные датацентровые пулы. Помните, что даже лучшие прокси не спасут, если вся нагрузка пойдет через один IP – ротация и еще раз ротация[[4]](https://blog.apify.com/rotating-proxies/#:~:text=If%20you%27re%20serious%20about%20web,requests%20from%20a%20single%20IP).
* **Не превышайте лимиты сайта.** Узнайте (эмпирически или из правил сайта), сколько запросов в минуту/час допустимо. AntiBotBrowser помогает распределить запросы между IP и замедлить их, но вы сами должны задать **правильный темп** парсинга. Например, если сайт начинает возвращать 429 после 5 запросов в секунду, перейдите на 1 запрос в 2 секунды плюс случайные паузы. Избегайте длительных серий запросов без пауз – даже с разными IP это может проявляться в серверной статистике. **Нерегулярность – ваш друг**: добавляйте sleep разной длительности, делайте технологические перерывы, не стреляйте запросами как метроном[[2]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Raw%20behavioral%20signals%20can%20be,can%20indicate%20a%20bot%20activity)[[26]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=One%20of%20the%20most%20common,scale%20web%20scraping).
* **Следите за согласованностью данных о клиенте.** Как отмечалось, убедитесь, что ваш заявленный User-Agent и окружение не противоречат друг другу[[14]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Remember%20to%20check%20that%20your,mismatch%20and%20block%20your%20requests). Стелс-скрипт берёт часть работы на себя, но проверяйте и вручную: например, если у вас окно безголового браузера всегда 1280×720, а вы притворяетесь iPhone, некоторые сайты могут заподозрить неладное. Настройте размер viewport (в Playwright можно через context = browser.new\_context(viewport={...})) или хотя бы используйте агенты десктопных браузеров для десктоп-режима.
* **Обновляйте User-Agent список.** В мире браузеров всё быстро меняется – выходят новые версии Chrome, появляются новые популярные устройства. Старайтесь раз в несколько месяцев пересматривать перечень строк в USER\_AGENTS. Также **не злоупотребляйте экзотикой**: агент YaBrowser или старого Opera хорош для разнообразия, но если вы скрапите, к примеру, Google, возможно логичнее чаще выдавать себя за Chrome или Safari, т.е. за наиболее типичного посетителя.
* **Держите stealth-скрипт в актуальном состоянии.** Как только замечаете, что вас стали чаще блокировать, или в сообществе парсеров обсуждают новую уловку антиботов – обновите stealth.min.js. Проекты вроде **puppeteer-extra-plugin-stealth** постоянно дополняются, и Playwright-аналоги тоже. Помните, что антиботы могут ловить то, что вы не поправили: например, если завтра Chromium начнет ставить новый флажок для автоматизации, а вы его не отключили, вас начнут видеть. Стелс-режим должен эволюционировать вместе с антибот-средствами.
* **Мониторьте результаты и логируйте всё.** Успешный обход – это не что-то гарантированное; нужно постоянно **следить за признаками блокировки**. Ведите лог запросов: коды ответов, время ответа, содержание страниц (например, не вернулась ли вместо реальных данных страница с текстом "Please enable JavaScript" или "Are you human?"). AntiBotBrowser уже выводит в консоль основное (не получилось подключиться к прокси, пойман 429). На основе этих сигналов автоматизируйте реакцию: смена прокси-пула, более долгий сон, или даже уведомление вас (в телеграм, почту), что, мол, **нужна капча**.
* **Уважайте правила сайтов.** С этической и правовой точки зрения, даже если технически вы обходите блокировки, убедитесь, что не нарушаете закон и правила пользования ресурсом. Некоторые сайты прямо запрещают автоматический сбор данных в своих **Terms of Service**. Веб-скрапинг попадает в серую зону: общедоступные данные собирать можно, но обход технических средств защиты может противоречить законам (например, CFAA в США) или условиям. Всегда взвешивайте риски. По возможности, запрашивайте только открытые данные, не превышайте разумную нагрузку (чтобы не нанести вред сайту), и соблюдайте **robots.txt** если это применимо.
* **Дополнительные уловки антиботов.** Имейте в виду, что помимо перечисленного, сайты могут использовать *honeypot*-ловушки и другие хитрости. Например, скрытые поля формы или скрытые ссылки: обычный пользователь их не видит и не нажмет, а наивный бот может попытаться взаимодействовать – и сразу себя выдать[[30]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=Avoid%20Web%20Scraping%20Honeypots). Наш браузер-бот сам по себе ничего не кликает, кроме того, что явно прописано в вашем коде, но если вы будете автоматизировать ввод данных, обратите внимание на невидимые поля. Также иногда на страницах внедряют **капча-генераторы действий** – скрипты, требующие, чтобы пользователь прокрутил страницу, или подождал N секунд. В таких случаях убедитесь, что ваша эмуляция пользователя (scroll, паузы) достаточно правдоподобна и, возможно, специфична (например, прокрутить до конца, если известно, что страница грузит контент на прокрутке).

## Заключение

AntiBotBrowser объединяет в себе несколько уровней защиты от обнаружения: сетевой (прокси), идентификационный (User-Agent и заголовки), маскировочный (стелс-скрипт), поведенческий (движения и задержки) и управленческий (обработка ошибок). Такой комплексный подход существенно повышает шансы на успешный **web scraping** без блокировок. Как метко заметил один из экспертов, парсинг без прокси и других мер сегодня практически невозможен: *«Web scraping without proxy rotation? Good luck! But even that isn't enough to not get blocked.»*[[31]](https://blog.apify.com/rotating-proxies/#:~:text=Rotating%20proxies%3A%20why%20you%20need,them%20for%20web%20scraping). То есть одного прокси недостаточно – нужны и другие уловки, которые мы разобрали.

Важно понимать, что борьба между ботами и антиботами динамична. Ваша стратегия должна адаптироваться: что работало вчера, завтра может потребовать корректировки. Тем не менее, освоив теорию — как сайты выявляют ботов и как этому противодействовать — вы вооружены для решения практически любой такой проблемы. **AntiBotBrowser**, по сути, является реализацией многих лучших практик в одном месте. Используйте его возможности, дополняйте своими улучшениями и анализируйте поведение скрипта. Тогда вы сможете собирать нужные данные, оставаясь под радаром систем киберзащиты.

**Источники и ссылки:**

* Anti-bot детекция и уязвимости безголовых браузеров[[1]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=The%20main%20issue%20with%20the,true)[[6]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=,in%20headless%20mode%20by%20default)[[7]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=is%20the%20,undefined)
* Ротация прокси и необходимость распределения запросов[[4]](https://blog.apify.com/rotating-proxies/#:~:text=If%20you%27re%20serious%20about%20web,requests%20from%20a%20single%20IP)[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix)
* Значение и смена User-Agent при парсинге[[5]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Great%21%20You%27ve%20successfully%20set%20up,a%20custom%20Playwright%20User%20Agent)[[12]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=Websites%20also%20use%20the%20User,avoiding%20CAPTCHAs%20while%20web%20scraping)
* Скрытие автоматизации (Stealth / Puppeteer-extra-plugin)[[6]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=,in%20headless%20mode%20by%20default)[[32]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Use%20the%20Playwright%20Stealth%20Plugin)
* Анализ поведения пользователей vs. ботов[[16]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=This%20involves%20signals%2C%20such%20as%3A)[[8]](https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a#:~:text=Websites%20often%20track%20user%20interactions,mouse%20movement%20function%20can)
* Ограничение скорости, 429 ошибки и борьба с ними[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix)[[28]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=One%20of%20the%20most%20suspicious,the%20cause%20and%20start%20investigating)
* Использование cookies для обхода капчи[[9]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=name%3Dvalue%3B%20name2%3Dvalue2%3B%20name3%3Dvalue3)
* Практические советы по избеганию блокировок[[2]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Raw%20behavioral%20signals%20can%20be,can%20indicate%20a%20bot%20activity)[[26]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=One%20of%20the%20most%20common,scale%20web%20scraping)

[[1]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth" \l ":~:text=The%20main%20issue%20with%20the,true) [[6]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=,in%20headless%20mode%20by%20default) [[20]](https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth#:~:text=The%20objective%20of%20the%20Stealth,mentioned%20in%20the%20official%20documentation) Avoiding Bot Detection with Playwright Stealth

<https://brightdata.com/blog/how-tos/avoid-bot-detection-with-playwright-stealth>

[[2]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Raw%20behavioral%20signals%20can%20be,can%20indicate%20a%20bot%20activity) [[15]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=,server%2C%20to%20appear%20more%20human) [[16]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=This%20involves%20signals%2C%20such%20as%3A) [[17]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Detecting%20forged%20and%20modified%20fingerprints%3A,server%2C%20to%20appear%20more%20human) [[18]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Specially%20crafted%20bot%20challenges%3A%20For,like%20Puppeteer%2C%20Playwright%20and%20Selenium) [[19]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=This%20JavaScript%20detection%20technique%20leverages,when%20they%20communicate%20with%20WebSocket) [[21]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=Some%20signals%20are%20simpler,features%3DAutomationControlled%27%60%20chrome%20flag) [[22]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=,have%20a%20less%20perfect%20behavior) [[24]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=such%20as%3A) [[25]](https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/#:~:text=,of%20request%20is%20almost%20zero) Bot detection 101: How to detect bots In 2025?

<https://blog.castle.io/bot-detection-101-how-to-detect-bots-in-2025-2/>

[[3]](https://oxylabs.io/resources/error-codes/429#:~:text=HTTP%20429%20Too%20Many%20Requests,Error%3A%20Causes%20%26%20Fix) [[27]](https://oxylabs.io/resources/error-codes/429#:~:text=,After%3A%203600) HTTP 429 Too Many Requests Error - Meaning & Fix

<https://oxylabs.io/resources/error-codes/429>

[[4]](https://blog.apify.com/rotating-proxies/#:~:text=If%20you%27re%20serious%20about%20web,requests%20from%20a%20single%20IP) [[31]](https://blog.apify.com/rotating-proxies/#:~:text=Rotating%20proxies%3A%20why%20you%20need,them%20for%20web%20scraping) ROTATING PROXIES: why you need them for web scraping

<https://blog.apify.com/rotating-proxies/>

[[5]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Great%21%20You%27ve%20successfully%20set%20up,a%20custom%20Playwright%20User%20Agent) [[7]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=is%20the%20,undefined) [[14]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Remember%20to%20check%20that%20your,mismatch%20and%20block%20your%20requests) [[32]](https://scrapingant.com/blog/web-scraping-playwright-python-part-4#:~:text=Use%20the%20Playwright%20Stealth%20Plugin) Web Scraping with Playwright Series Part 4 - Avoid Getting Blocked | ScrapingAnt

<https://scrapingant.com/blog/web-scraping-playwright-python-part-4>

[[8]](https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a#:~:text=Websites%20often%20track%20user%20interactions,mouse%20movement%20function%20can) [[23]](https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a#:~:text=Bot%20detection%20mechanisms%20have%20become,curved%20mouse%20movements%20can%20help) Preventing Playwright Bot Detection with Random Mouse Movements | by Manan Patel | Medium

<https://medium.com/@domadiyamanan/preventing-playwright-bot-detection-with-random-mouse-movements-10ab7c710d2a>

[[9]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=name%3Dvalue%3B%20name2%3Dvalue2%3B%20name3%3Dvalue3) [[10]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=This%20fingerprint%20includes%20details%20about,be%20one%20of%20three%20types) [[11]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=track%2C%20cell%20tower%20connections%20are,IPs%20to%20browse%20the%20web) [[12]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=Websites%20also%20use%20the%20User,avoiding%20CAPTCHAs%20while%20web%20scraping) [[13]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=Websites%20also%20use%20the%20User,avoiding%20CAPTCHAs%20while%20web%20scraping) [[30]](https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024#:~:text=Avoid%20Web%20Scraping%20Honeypots) How to Bypass CAPTCHA While Web Scraping in 2025

<https://scrapfly.io/blog/posts/how-to-bypass-captcha-while-web-scraping-in-2024>

[[26]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=One%20of%20the%20most%20common,scale%20web%20scraping) [[28]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=One%20of%20the%20most%20suspicious,the%20cause%20and%20start%20investigating) [[29]](https://www.zenrows.com/blog/web-scraping-without-getting-blocked#:~:text=The%20recommended%20approach%20is%20to,services%20like%20web%20scraping%20APIs) 14 Ways for Web Scraping Without Getting Blocked - ZenRows

<https://www.zenrows.com/blog/web-scraping-without-getting-blocked>