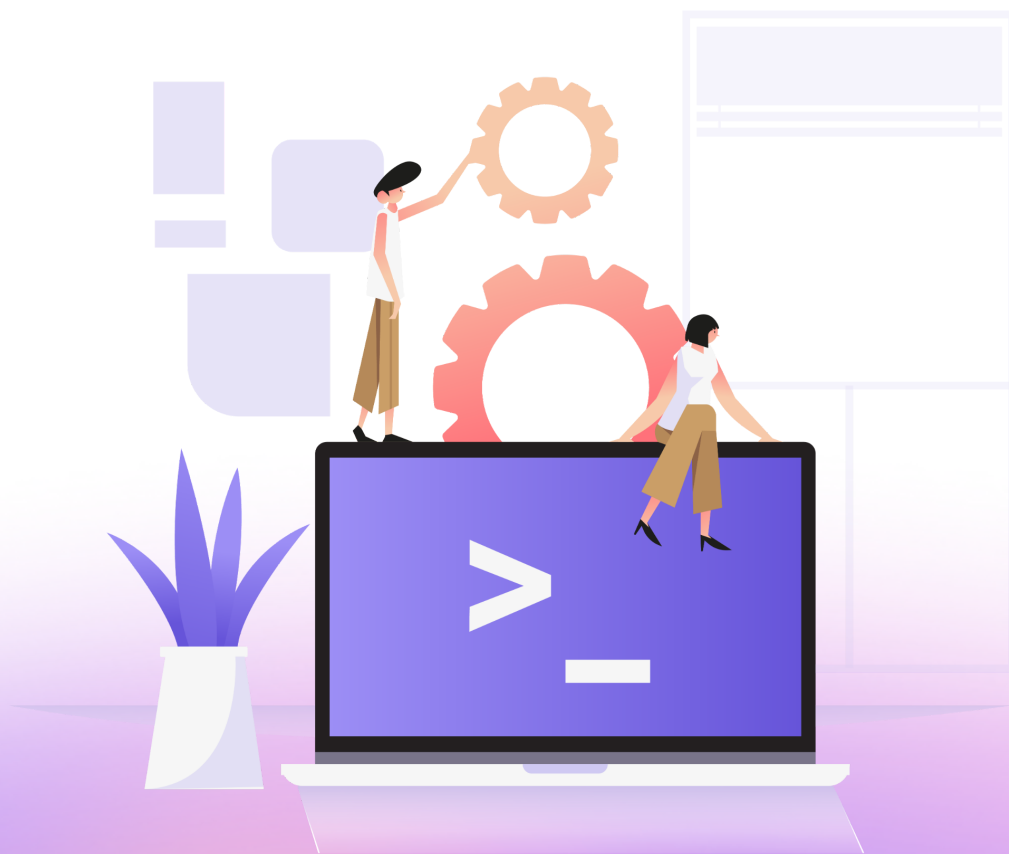


Тестирование веб-приложений

Особенности тестирования веб-приложений



На этом уроке

1. Узнаем о наиболее распространённых ошибках в формах, а также о приёмах тестирования форм.
2. Изучим структуру ссылок, научимся искать битые ссылки.
3. Выясним, что такое кеш, cookie и сессии.

Оглавление

[Формы. Наиболее распространённые ошибки и приёмы тестирования](#)

[Описание форм в HTML](#)

[Тестирование форм](#)

[Регистрация](#)

[Логика](#)

[Валидация полей ввода](#)

[Авторизация](#)

[Поиск](#)

[Фильтры](#)

[Обратная связь](#)

[Обход клиентской валидации](#)

[Ссылки](#)

[Структура URI](#)

[Абсолютные, относительные и другие типы ссылок в HTML](#)

[Инструменты поиска битых ссылок](#)

[Кеш, cookie и сессии](#)

[Кеш браузера](#)

[Cookie](#)

[Типы cookie](#)

[Инструменты для управления cookie](#)

[Подмена cookie](#)

[Веб-хранилища](#)

[Сессии](#)

[Глоссарий](#)

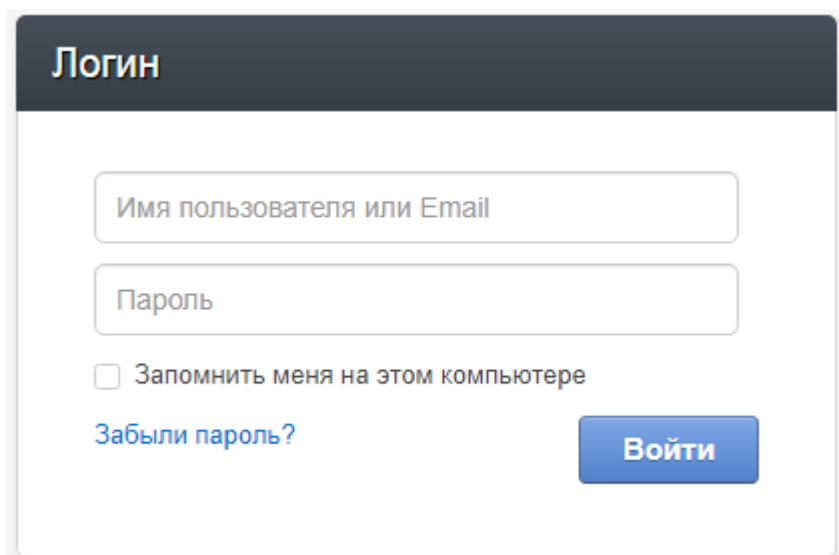
[Практическое задание](#)

Формы. Наиболее распространённые ошибки и приёмы тестирования

Описание форм в HTML

Формы — одна из важнейших частей любого веб-приложения. Регистрация, вход, поиск, отправка обратной связи, создание разных заявок, покупки в интернет-магазинах не обходятся без соответствующих форм.

Пример одной из самых распространённых форм — вход на сайт. Так форма выглядит в браузере:

A screenshot of a login form. At the top, there is a dark grey header with the word "Логин" in white. Below the header, there are two input fields: the first one contains the placeholder text "Имя пользователя или Email", and the second one contains "Пароль". Below the password field, there is a checkbox followed by the text "Запомнить меня на этом компьютере". To the left of the "Войти" button, there is a blue link that says "Забыли пароль?". The "Войти" button is a blue rectangle with white text.

А так — в исходном коде:

```

▼<form id="login-form" action="/user/login-check" method="post" class="form-signin">
  ▼<div class="title-box">
    <h2 class="title">Логин</h2>
  </div>
  ▼<fieldset>
    ▶<script type="text/javascript">...</script>
    ▼<div class="input-prepend">
      <input type="text" id="prependedInput" class="span2" name="_username" value required="required" placeholder="Имя пользователя или Email" autofocus>
    </div>
    ▼<div class="input-prepend">
      <input type="password" id="prependedInput2" class="span2" name="_password" required="required" placeholder="Пароль" autocomplete="off">
    </div>
    ▼<label class="checkbox" oro-remember-me">
      <input type="checkbox" id="remember_me" name="_remember_me" value="on">
      "Запомнить меня на этом компьютере"
    </label>
    ▼<div class="control-group form-row">
      <a href="/user/reset-request">Забыли пароль?</a>
      <button type="submit" class="btn btn-large btn-primary pull-right" id="_submit" name="_submit">Войти</button>
    </div>
  </fieldset>
  <input type="hidden" name="_target_path" value>
  <input type="hidden" name="_csrf_token" value="33w9ksNu5sPS9aqLM-NziBMci0Jawr6UkPo3oY4mDyY">
</form>

```

Обратим внимание на некоторые элементы, которые могут быть полезны в процессе тестирования.

Открывающий тег формы:

```

<form id="login-form" action="/user/login-check" method="post"
class="form-signin">

```

Параметр `action` указывает на скрипт, который будет обрабатывать эту форму, `method` — `post` или `get` — каким HTTP-методом данные станут передаваться на сервер. Подавляющее большинство форм отправляет данные в теле запроса методом `POST`.

```

<input type="text" id="prependedInput" class="span2" name="_username" value=""
required="required" placeholder="Имя пользователя или Email" autofocus="">

```

Это поле ввода имени пользователя или электронной почты. Тип поля — `text`, а это значит, что дополнительной фильтрации со стороны HTML нет. HTML5 имеет встроенную поддержку разных типов полей: `email`, `tel`, `url`, `number`, `time`, `date`, `datetime`, `datetime-local`, `month`, `week`, `range`, `search`, `color`. Есть также атрибут `required="required"`, который означает, что поле обязательно для заполнения. В предыдущих примерах вы видели, что атрибут `required` указывался без его значения. Это не ошибка, а два разных способа указания значения. Сейчас они оба считаются верными, но более рекомендуемый — запись в форме `required`, без значения.

Аналогично с полем пароля:

```
<input type="password" id="prependedInput2" class="span2" name="_password"
required="required" placeholder="Пароль" autocomplete="off">
```

Но здесь тип поля — `password`, а значит, символы пароля должны маскироваться звёздочками. Пароль тоже обязателен, а ещё для него отключено автодополнение, чтобы браузер случайно не подсказал кому-то наш пароль.

Специальная кнопка с типом `submit` служит для отправки формы:

```
<button type="submit" class="btn btn-large btn-primary pull-right" id="_submit"
name="_submit">Войти</button>
```

У неё нет каких-то особых параметров.

А вот ссылка «Забыли пароль?» хоть и расположена внутри формы, но фактически — не её элемент:

```
<a href="/user/reset-request">Забыли пароль?</a>
```

На форме есть элементы с типом `type="hidden"`. Они не видны пользователю, но требуются для передачи различной технической информации.

Тестирование форм

В предыдущем разделе мы рассмотрели формы изнутри: как они устроены и как выглядят в HTML. А сейчас обсудим, как их тестировать с позиции чёрного ящика. Наиболее сложны в этом плане элементы, которые подразумевают ввод пользователем информации в свободном формате,

например, логин, пароль, текст сообщения, заголовок письма. Здесь возможна большая вариативность, а значит, и проблем может быть больше, чем, например, в переключателях или радиокнопках, где надо лишь выбрать один или несколько вариантов из представленных.

Список всех доступных элементов — на страницах [HTML Form Elements](#) и [HTML Input Types](#). Информация на русском языке — [здесь](#).

В версии HTML4 валидация форм осуществлялась через JavaScript на клиентской стороне и посредством различных языков программирования — на серверной. С появлением HTML5 предыдущие методы валидации не исчезли, но в последней версии появились новые атрибуты, и, таким образом, добавилась валидация форм средствами браузера.

Основные атрибуты:

1. `Required`, без которого форма не отправится.
2. Атрибуты `min` и `max`, указывающие, соответственно, минимум и максимум для числового поля ввода.
3. `Minlength`, `Maxlength` — ограничение на длину ввода.
4. `Pattern="\+[0-9]{11}"` — регулярное выражение, которому соответствует вводимый текст.

При тестировании желательно проверить, кто отвечает за валидацию поля на стороне клиента: сам браузер или JavaScript-код, написанный фронтенд-программистом. Для этого надо посмотреть исходный код страницы и определить, есть ли у полей ввода атрибуты из вышеуказанного списка. Если они есть, то валидацию проводит браузер, а значит, не стоит проверять, будет ли поле обязательным, во всех вариациях. Эта проверка стандартна, так что шансы найти там баг примерно нулевые. А вот если проверка поля написана разработчиком на JavaScript, то надо присмотреться к такому полю повнимательнее: собственные методы валидации могут содержать ошибки, и поэтому требуется более тщательное тестирование.

Пройдёмся по типовым формам и посмотрим, на что требуется обращать внимание.

Регистрация

Логика

Пользователь с такими данными не существует в системе

Если форма заполнена верно, пользователь успешно пройдёт регистрацию. По каким данным он должен фильтроваться, зависит от требований, и это надо уточнить заранее:

1. По логину.
2. Телефону.
3. Электронной почте.
4. По комбинации этих полей.

Как правило, телефон и электронная почта уникальны, а если есть логин, то он тоже уникальный.

Пользователь с такими данными существует в системе

Здесь мы должны получить сообщение об ошибке. Но в какой момент это произойдёт? При заполнении? После отправки? И будет ли это сообщение понятно человеку? «Пользователь с таким номером телефона уже существует в системе» — это хорошо. «Error 5. Duplicate user_id» — плохо.

Валидация полей ввода

Ф. И. О.

Сначала надо обратить внимание на обязательность к заполнению. Если эти поля есть, должен ли человек указывать фамилию, имя и отчество? И может ли он их не заполнять? А если указано только имя, то надо ли вводить и фамилию, и отчество? Здесь может быть довольно-таки сложная логика, поэтому потребуется составить список потенциальных проверок. Вот возможный вариант такого списка при условии, что Ф. И. О. помечены как необязательные. Требования: поля необязательные, но если заполнена фамилия, то обязательно указывается и имя.

Фамилия	Имя	Отчество	Результат
НЕТ	НЕТ	НЕТ	ОК
ДА	ДА	ДА	ОК
ДА	ДА	НЕТ	ОК
ДА	НЕТ	НЕТ	?
НЕТ	ДА	НЕТ	?
НЕТ	НЕТ	ДА	?
НЕТ	ДА	ДА	?
ДА	НЕТ	ДА	?

1. Поля «Фамилия», «Имя», «Отчество».

а. Длина.

- Минимум. Какова минимальная длина фамилии? Может ли это быть один символ? Или два? Или три?
- Максимум. Сколько символов будет «необходимо и достаточно»?

б. Диапазон допустимых символов.

- В Ф. И. О. могут быть только буквы. И специальные символы. И иногда цифры.
- А должны ли Ф. И. О. быть только на кириллице? Или могут содержать латиницу? Иероглифы?
- Могут ли Ф. И. О. быть написаны справа налево (иврит, арабская вязь)?

с. Обязательность к заполнению.

- Может ли пользователь не писать фамилию, имя или отчество?

2. Логин.

а. Длина.

- Минимальная длина.
- Максимальная длина.

б. Диапазон допустимых символов.

- с. Необходимость.
 - і. Нужен ли нам логин или можно входить по email, телефону, социальным сетям?
- 3. Email: базовая валидация формата.
- 4. Телефон: базовая валидация формата.
- 5. Дата рождения.
 - а. Проверка формата.
 - б. Проверка валидности возраста.

Про имена и фамилии есть очень хорошая [статья-перевод на «Хабре»](#). Оригинал написан в 2010 году, но с тех пор вряд ли что-то сильно поменялось, и если вы работаете над международной системой, идеи из статьи стоит взять во внимание.

Авторизация

- 1. Пользователь существует в системе с указанными логином и паролем.
- 2. Пользователь с указанным логином не существует в системе.
- 3. Пользователь с указанным логином существует в системе, но пароль неверный.
- 4. Пользователь с указанным логином и паролем существует в системе, но имеет неактивный статус.
 - а. Логин не активирован.
 - б. Логин заблокирован.
- 5. Валидация полей ввода.

Поиск

- 1. По наличию.
 - а. Результаты существуют.
 - і. Найдено всё соответствующее.
 - іі. В поиск не попало ничего не актуального.
 - б. Результаты не существуют.
 - і. В поиск не попало ничего не актуального.
 - іі. Выдано корректное сообщение о пустом результате.
- 2. По типу поиска.
 - а. По точному совпадению.
 - б. Умный поиск, по неточному совпадению.
 - с. С использованием логических операторов, если они поддерживаются.

Фильтры

- 1. По результату.
 - а. Фильтр возвращает непустой результат.
 - і. В фильтр попали все подходящие данные.
 - іі. В фильтр не попали несоответствующие данные.

- b. Фильтр возвращает пустой результат.
 - i. В фильтр не попали несоответствующие данные.
 - ii. Валидное сообщение об ошибке.
- 2. По фильтрам.
 - a. Применён один фильтр.
 - b. Применена группа фильтров.
 - i. Результат должен соответствовать каждому фильтру.

Обратная связь

Эти формы часто содержат лишь несколько текстовых полей «Тема» и «Сообщение», если пользователь зарегистрирован, и «Имя» и «Электронная почта» или «Телефон», когда регистрации на сайте нет. Их тестирование в этом случае мало отличается от тестирования, скажем, полей ввода в формах регистрации.

Обход клиентской валидации

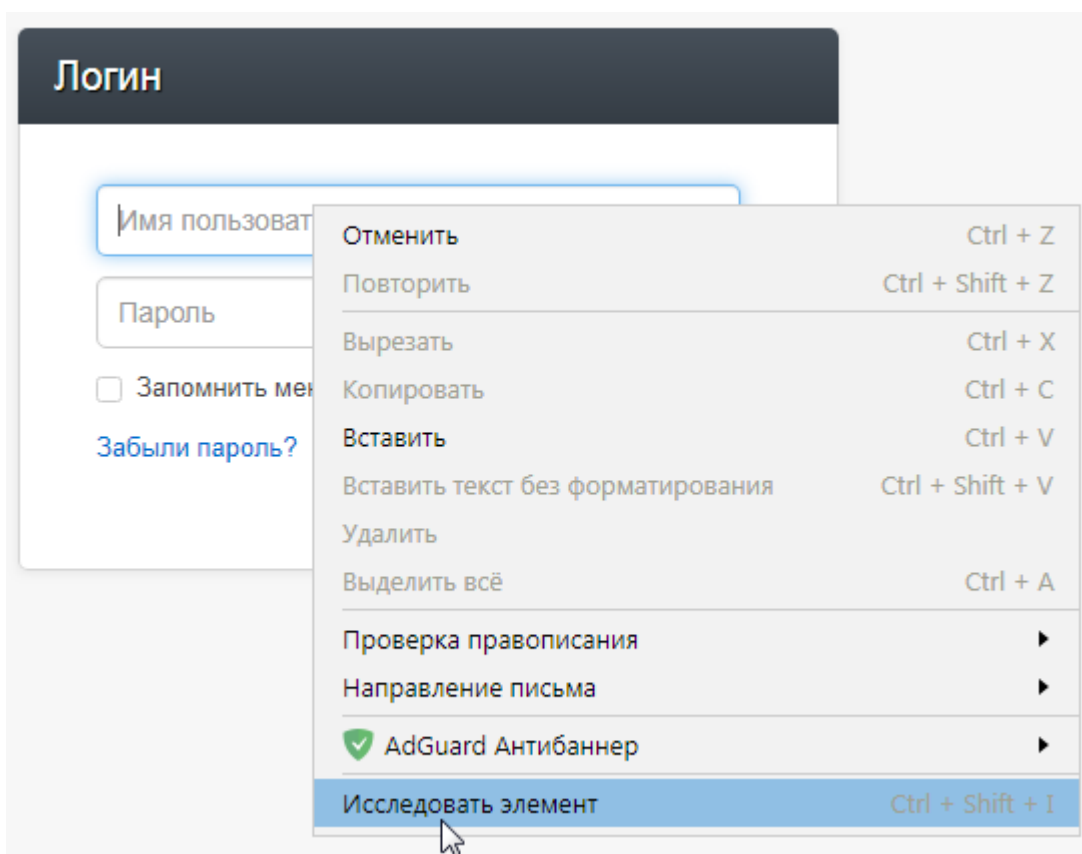
Валидация данных в форме на клиенте — это, безусловно, хорошо, так как пользователь получает мгновенную обратную связь без необходимости отправки данных на сервер. Но этого способа недостаточно, и все полученные данные обязательно надо проверять и на сервере. Если проверки данных, полученных от клиента, нет, это может привести к сбоям в работе приложения или его взлому. Поэтому тестирование форм в обход валидации в браузере — это один из базовых способов тестирования безопасности. Выполняется такая проверка несколькими способами, рассмотрим самый простой и доступный.

Вариант 1. Отключение фильтрации методом редактирования HTML

Современные браузеры позволяют править HTML-код страницы, отображаемой в браузере. Эти изменения «одноразовые», то есть при перезагрузке страницы они потеряются. Но для нашей задачи это неважно. Нам надо изменить код так, чтобы валидации на клиенте не было. Экспериментировать будем на [странице входа](#). Здесь два обязательных к заполнению поля: логин и пароль. Если попытаться отправить пустую форму, то появится сообщение:

Поле обязательно для заполнения. Надо это исправить!

Нажимаем правой кнопкой на поле и выбираем «Исследовать элемент»:



Откроется окно с HTML-кодом. Этот код можно редактировать. Наши изменения отобразятся на открытой странице, но если её перезагрузить или кликнуть на какую-либо ссылку, то изменения потеряются.

```

<form id="login-form" action="/user/login-check" method="post" class="form-signin">
  <div class="title-box">...</div>
  <fieldset>
    <script type="text/javascript">...</script>
    <div class="input-prepend">
      <input type="text" id="prependedInput" class="span2" name="_username" value
        required="required" placeholder="Имя пользователя или Email" autofocus> == $0
    </div>
    <div class="input-prepend">
      <input type="password" id="prependedInput2" class="span2" name="_password"
        required="required" placeholder="Пароль" autocomplete="off">
    </div>
    <label class="checkbox oro-remember-me">...</label>
    <div class="control-group form-row">...</div>
  </fieldset>
  <input type="hidden" name="_target_path" value>
  <input type="hidden" name="_csrf_token" value=
    "0t3Y8wF0lPwmSatYZl0_OiQoMxA__09fb1rdjiDGBsk">
</form>

```

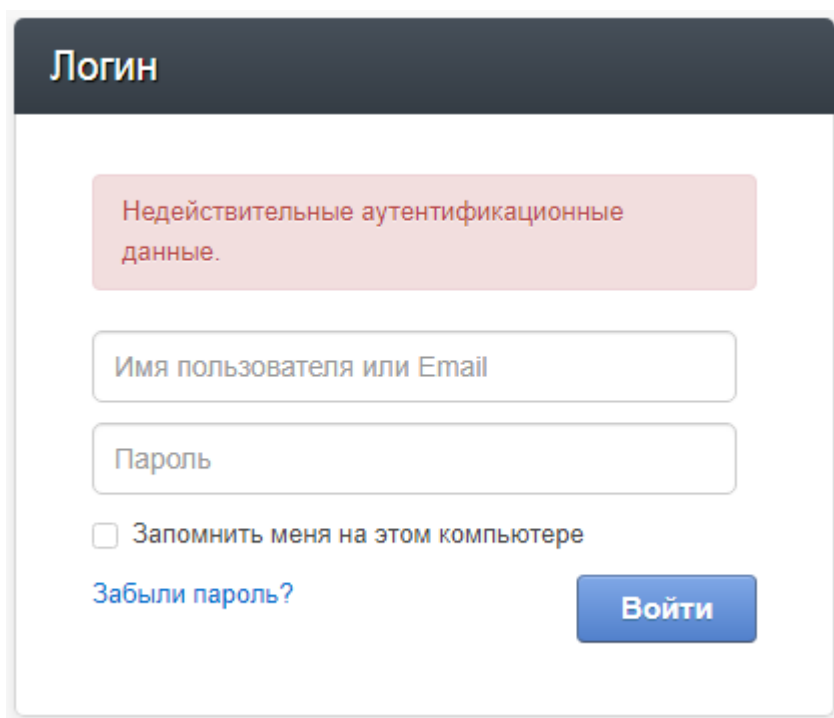
Поля логина и пароля — required. Теперь делаем двойной клик на этом атрибуте, и он переходит в состояние редактирования. Удаляем всё, что там есть, и нажимаем Enter. Получаем такой результат: оба поля стали необязательными.

```

<div class="input-prepend">
  <input type="text" id="prependedInput" class="span2" name="_username" value
    placeholder="Имя пользователя или Email" autofocus>
</div>
<div class="input-prepend">
  <input type="password" id="prependedInput2" class="span2" name="_password"
    placeholder="Пароль" autocomplete="off"> == $0
</div>
<label class="checkbox oro-remember-me">...</label>

```

Возвращаемся в браузер, на вкладку со страницей логина. Нажимаем «Войти» при пустых полях логина и пароля, и форма успешно отправляется! Сервер вернул такое сообщение об ошибке, и это хорошо. Значит, на сервере проводится валидация данных, а мы, по сути, получили то, что ожидали. Тест успешно пройден.



Вариант 2. Отправка GET/POST-запроса через Postman

Этот вариант мы пока оставим на самостоятельное изучение. Подробно рассмотрим его чуть позже, при изучении инструмента Postman и тестирования API.

Небольшая инструкция:

1. Ищем в поисковике «как отправить данные формы через postman» / how to send form data from postman, читаем документацию.
2. Определяем, какие поля надо отправлять.
3. Не забываем про CSRF-токены.

Ссылки

Часто в интернете встречаются два разных термина для описания гиперссылки: URI (Uniform Resource Identifier) — унифицированный идентификатор ресурса, и URL (Uniform Resource Locator) — унифицированный локатор ресурса. По сути, они описывают одно и то же: по какому адресу найти ресурс. Это различие, как некоторое недопонимание, появилось на заре интернета, в первой половине 90-х годов. Сейчас URI и URL по факту считаются синонимами, но стандарты рекомендуют именно термин URI. Подробности — в [документе RFC 3305](#) (на английском языке).

Структура URI

<схема>:[//[<логин>[:<пароль>]@]<хост>[:<порт>]][/<путь>][?<параметры>][#<якорь>]

Схема — схема обращения к ресурсу. В большинстве случаев имеется в виду сетевой протокол, например, http.

Логин — имя пользователя, используемое для доступа к ресурсу.

Пароль — пароль указанного пользователя.

Хост — доменное имя (site.com) или IP-адрес (127.0.0.1).

Порт — порт для подключения.

Путь — информация о месте нахождения ресурса на конкретном хосте.

Параметры — строка запроса с передаваемыми на сервер методом GET параметрами. Начинается с символа «?», в качестве разделителя используется &. Пример:

?параметр1=значение1&параметр2=значение_2.

Якорь — идентификатор якоря внутри документа.

Примеры адресов:

`ftp://user:password@hosting.org:20/transfer/incoming/data`

`https://forum.justforfun.net/forums/view/page.php?skip=20&order=desc`

`http://reader.boooks.ru/read/book/127/chapter3.html#top`

`http://10.20.57.14:8080/resources/`

Эти ссылки — примеры абсолютных путей, то есть они указывают всегда на один и тот же ресурс, где бы мы ни находились. Кроме них есть ещё и относительные: ресурс, на который они указывают, зависит от того, где сейчас находится пользователь.

Абсолютные, относительные и другие типы ссылок в HTML

Абсолютные ссылки указывают полный путь к ресурсу, например, `http://mysite.com/images/photos/photo01.jpg`, и всегда работают одинаково, независимо от того, из какого документа (и даже сайта) по ним переходят.

Относительные ссылки указывают путь к ресурсу относительно текущего URI, то есть местоположения на сайте. Таким образом, одна и та же ссылка, написанная одинаково, с разных страниц ведёт на совершенно разные страницы.

Например, изображение находится на стороннем сервере, поэтому ссылка указана абсолютная:

О Нас



История компании

История компании — 2000 год : Созда
— 2009 год : Закуплено уникальные д
Рязань и пр).
— 2013 год : Запуск цеха тортов на за
моделирование, пищевой принтер и г
— 2014 год : Расширение склада и хс
— 2015 год : Запуск дегустационного
— 2016 год : Открытие 2го цеха

Наша компания производит торты на
и пирожных премиум класса ручной р
традиционные сладости, приготовлен

```

```

Есть четыре типа относительных ссылок:

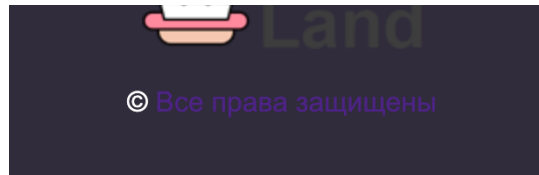
1. **Ссылка сетевого пути.** Например, `//mysite.com/files/`. Эта ссылка перенаправляет на указанный адрес с использованием текущей схемы. То есть, если мы находимся на `https://anysite.com`, откроется адрес `https://mysite.com/files/`, а если на `ftp://anysite.com` — `ftp://mysite.com/files`. Полный адрес ресурса высчитывается относительно URI-схемы.
2. **Ссылка абсолютного пути.** Она имеет вид `/path/to/some/uri` и начинается с символа «/», который указывает, что отсчёт надо начинать с корня домена. В этом случае откроется ресурс с этого же домена, но путь к конечному ресурсу будет одинаков для всех страниц на этом домене.
3. **Ссылка относительного пути.** Такая ссылка высчитывается относительно текущего адреса. Например, ссылка вида `images/photos/photo01.jpg` означает, что для подразделов сайта `mysite.com/events/` и `mysite.com/tours/` она будет указывать на разные файлы. Строка «../» говорит о переходе на один уровень вверх по структуре пути. Если ссылка `../images/photos/photo02.jpg` находится на странице `http://mysite.com/events/about.html`, то полный путь будет `http://mysite.com/images/photos/photo02.jpg`. А если на той же странице есть ссылка `images/photos/photo02.jpg`, то полный путь — `http://mysite.com/events/images/photos/photo02.jpg`.

Например, ссылка на логотип сайта в примере из предыдущего урока — `images/logo.png`:

```
<a href="index.html"> </a>
```

4. **Ссылка внутри документа.** Она задаётся посредством якоря #. Например, можно в начале документа присвоить какому-то элементу `name="top"`, тогда ссылка вида `Наверх` перекинет нас в начало документа.

Переход в самую верхнюю часть сайта при клике на «Все права защищены»:



```
<a href="#top">Все права защищены</a>
```

Разные виды относительных ссылок можно посмотреть [на странице сайта testingcourse.ru](https://testingcourse.ru).

Кроме ссылок типа http(s), есть множество других типов ссылок (схем) в веб-приложениях:

- mailto;
- tel;
- javascript;
- ftp;
- telnet;
- file;
- git;
- xmpp;
- skype;
- steam;
- tg;
- многие другие, зарегистрированные [Uniform Resource Identifier \(URI\) Schemes](https://www.iana.org/assignments/uri-schemes).

А ещё можно создать собственную схему и задать её обработку определённому приложению.

Для работы любого сайта важно, чтобы все ссылки были верными: как минимум не вели в никуда. Если сайт небольшой, это можно проверить вручную, прокликая по всем ссылкам. Но если на сайте сотни и тысячи ссылок, то надо воспользоваться автоматизированными инструментами для проверки сайта.

Инструменты поиска битых ссылок

Их довольно много — как онлайн-сервисов, так и программ, устанавливаемых на компьютер. Сервисы обычно имеют хороший интерфейс, удобны и доступны всегда. Примеры таких сервисов:

- <https://www.deadlinkchecker.com>;
- <https://www.drlinkcheck.com>;
- <https://www.brokenlinkcheck.com>.

И многие другие. Как правило, они имеют некоторый бесплатный план — 1000, 1500, 2000 ссылок. А более крупные проекты пользуются платными тарифами.

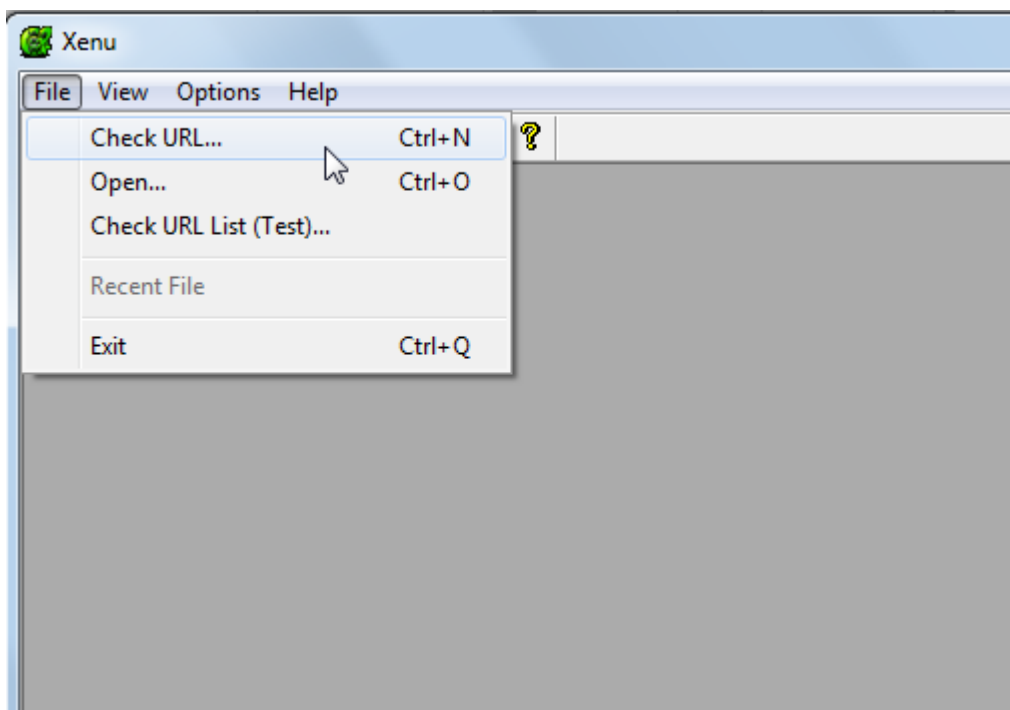
У онлайн-сервисов есть один существенный минус: если мы хотим проверить ссылки не на «живом» сайте, а на разрабатываемой версии, расположенной во внутренней сети компании, то онлайн-сервисы не подойдут, потребуется воспользоваться приложением.

Рассмотрим одно из них — Xenu Link Sleuth. Программа существует давно, и интерфейс выглядит устаревшим, но свою функцию она до сих пор выполняет хорошо.

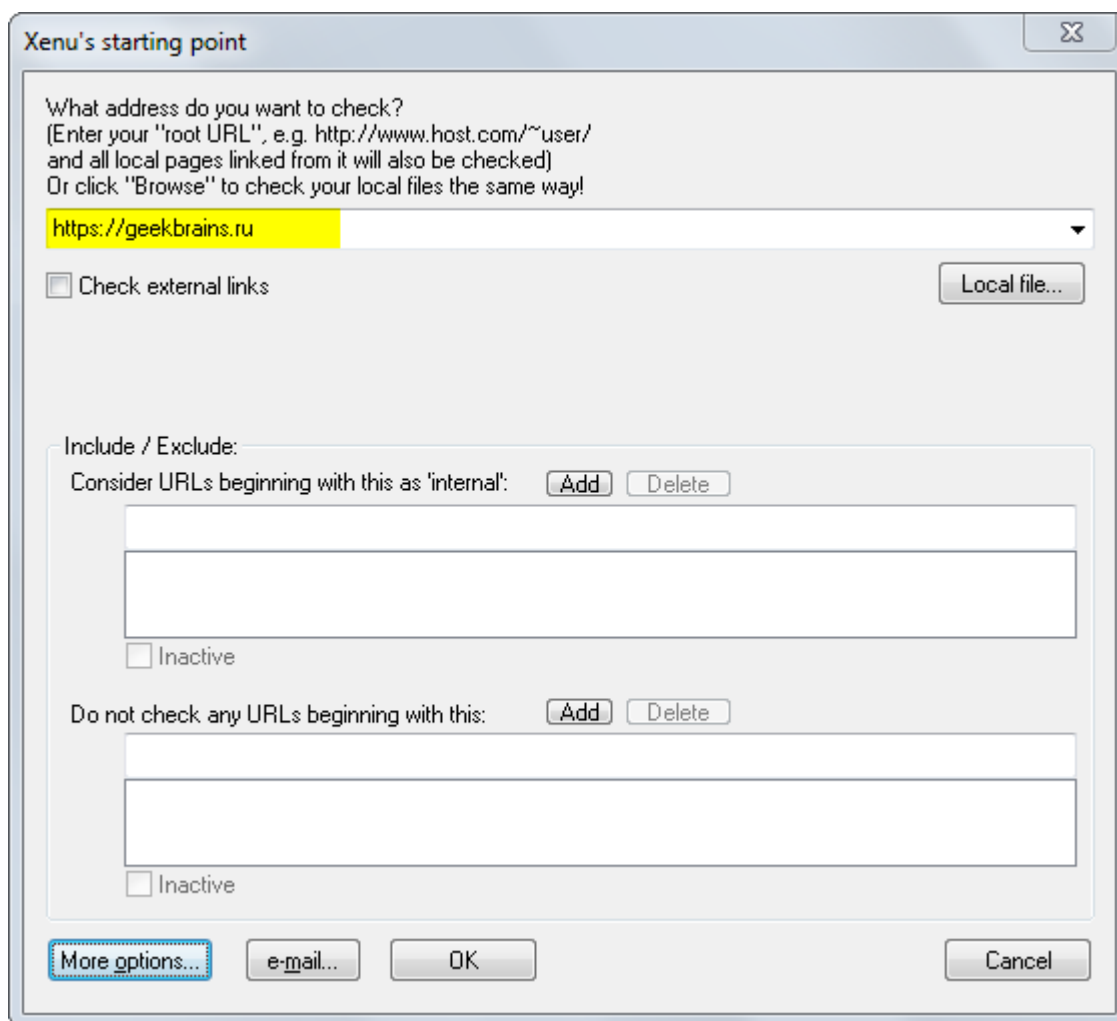
Сайт — [Find broken links on your site with Xenu's Link Sleuth \(TM\)](#).

Скачиваем и устанавливаем программу.

Нажимаем File — Check URL:



В поле адреса указываем URI сайта:



Запускаем проверку кнопкой OK.

Программа выполняет проверку в несколько потоков. Это занимает немного времени, но компьютер нагружается. На момент создания скриншота на сайте появилось 17 521 ссылок, включая ссылки на различные ресурсы: изображения, файлы, CSS, JS. Из них проверено 9 494 ссылки. При дальнейшем продвижении по сайту общее количество также может увеличиваться.

Address	Status	Type	Size	Title
https://geekbrains.ru/dogovor	ok	text/html	385	Пользовательское соглашение
mailto:support@geekbrains.ru	skip external			support@geekbrains.ru
https://geekbrains.ru/courses?tab=professions	ok	text/html	326771	Курсы GeekBrains - образование...
https://geekbrains.ru/courses?tab=free_intensives	ok	text/html	326787	Курсы GeekBrains - образование...
https://geekbrains.ru/courses	ok	text/html	326710	Курсы GeekBrains - образование...
https://vk.com/geekbrainsru	skip external			<svg class="svg-icon icon-vk ico...
https://www.facebook.com/geekbrains.ru	skip external			<svg class="svg-icon icon-faceb...
https://www.instagram.com/geekbrains.ru	skip external			<svg class="svg-icon icon-instag...
https://www.youtube.com/user/progliveru	skip external			<svg class="svg-icon icon-youtu...
https://sk.ru/	skip external			<svg xmlns="http://www.w3.org/...
https://github.com/login/oauth/authorize?client_id=f62ca85f...	skip external			redir
https://oauth.vk.com/authorize?client_id=5447856&display=...	skip external			redir
https://accounts.google.com/o/oauth2/auth?access_type=of...	skip external			redir
https://connect.mail.ru/oauth/authorize?client_id=7657508&r...	skip external			redir
https://d2xzmw6cctk25h.cloudfront.net/staticpage/141/asset...	skip external			
https://www.facebook.com/v2.6/dialog/oauth?client_id=878...	skip external			redir
https://d2xzmw6cctk25h.cloudfront.net/staticpage/142/asset...	skip external			
https://d3ur190xygwd69.cloudfront.net/assets/vendor.0d5f92...	skip external			
https://d3ur190xygwd69.cloudfront.net/assets/app.515e2da7...	skip external			
https://d3ur190xygwd69.cloudfront.net/assets/logo-snow-ny...	skip external			Logo snow ny2020
https://geekbrains.ru/events	ok	text/html	176563	Вебинары GeekBrains - образов...

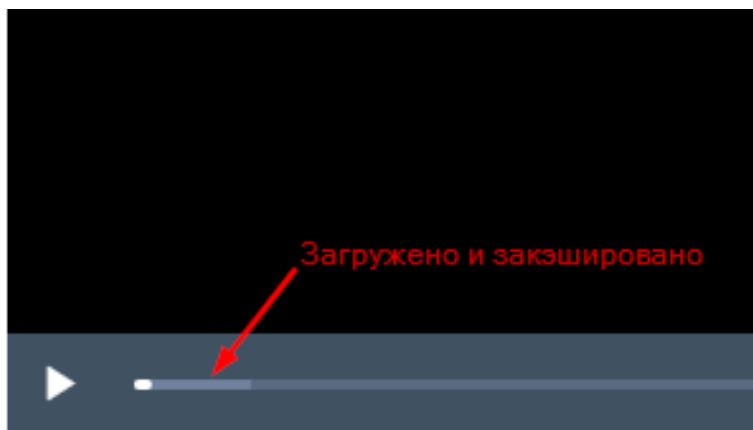
Ready Threads: 21 9494 of 17521 URLs (54 %) done 0:03:36

После проверки мы получим отчёт со списком всех битых ссылок, чтобы передать его разработчикам на исправление или исправить самостоятельно, если знать, что делать.

Кеш, cookie и сессии

Кеш браузера

У каждого браузера есть особая папка на жёстком диске, в которую сохраняются элементы веб-страниц, запрашиваемых в браузере. К этим элементам относятся изображения, видео, стили, скрипты и многое другое — в общем, всё, что нечасто меняется и часто имеет значительный объём. При загрузке страницы браузер сбрасывает этот контент на жёсткий диск. Эта особая папка называется кеш браузера, а сохранённые на диск данные — закешированными данными.



В большинстве случаев кеш используется, чтобы ускорить загрузку страниц, которые мы уже посещали. Если элементы сайта уже сохранились в кеше браузера, то при его повторном открытии часть информации, что занимает большой объём, будет загружаться не из сети, а прямо из кеша браузера (с компьютера). Такая страница откроется очень быстро.

Как правило, папка кеша содержится внутри папки текущего пользователя. Например, для Google Chrome путь к кешу выглядит так:

`C:\Users\<Username>\AppData\Local\Google\Chrome\User Data\Default\Cache.`

А для Firefox — так:

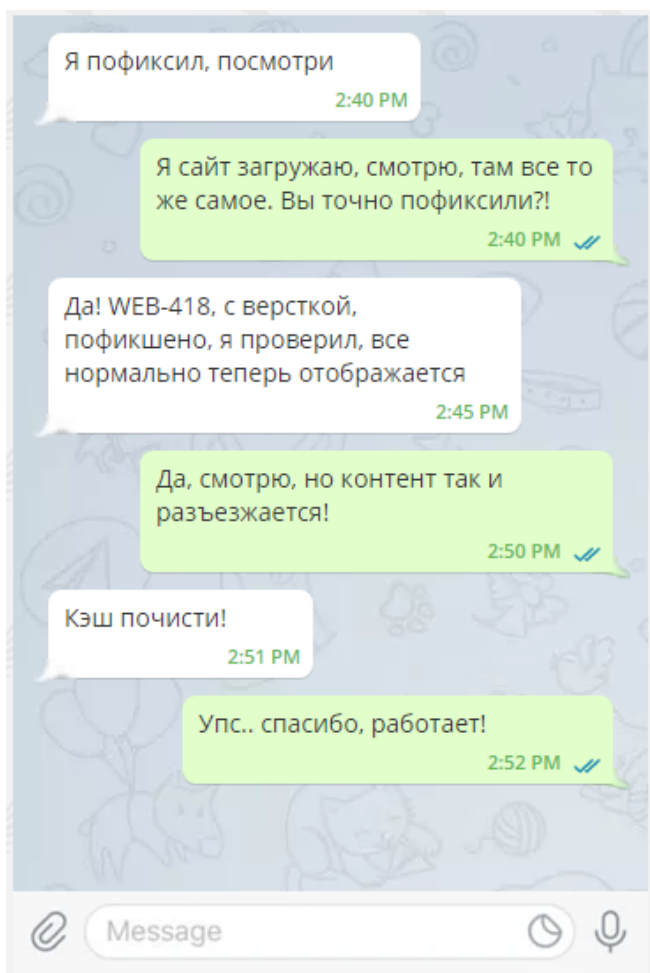
`C:\Users\<Username>\AppData\Local\Mozilla\Firefox\Profiles\<Profile>\cache2.`

Кеш браузера — удобная и полезная вещь, но она имеет свои ограничения по использованию. При активном интернет-сёрфинге кеш быстро увеличивается в размерах. Большой объём сохранённых файлов начинает негативно влиять на производительность браузера. Многим пользователям кажется, что компьютер тормозит и медленно обрабатывает запросы. Чтобы такого не было, кеш требуется время от времени чистить.

У тестировщиков в повседневной работе возникают и другие проблемы, связанные с кешированием. Если стили, изображения или скрипты закешированы, а разработчики внесли незначительные изменения в код и выложили новую версию, есть вероятность, что браузер тестировщика будет брать старую версию стилей, скриптов и изображений из кеша. А это создаст проблемы при тестировании.

Важно! Перед стартом очередной тестовой сессии «почистите» кеш браузера — иногда это позволяет сохранить время и нервы.

Вот пример. Он, конечно, придуманный, но в жизни встречаются подобные ситуации:



Как «почистить» кеш

Для Chrome:

1. Нажать кнопку «Меню» — «Дополнительно» — «Очистить историю» или *Ctrl + Shift + Del*.
2. В списке «Удалить записи» выбрать период, за который надо удалить кеш.
3. Включить опцию «Файлы, сохранённые в кеше».
4. Отключить остальные опции, если надо очистить только кеш браузера.
5. Нажать «Очистить».

Для Firefox:

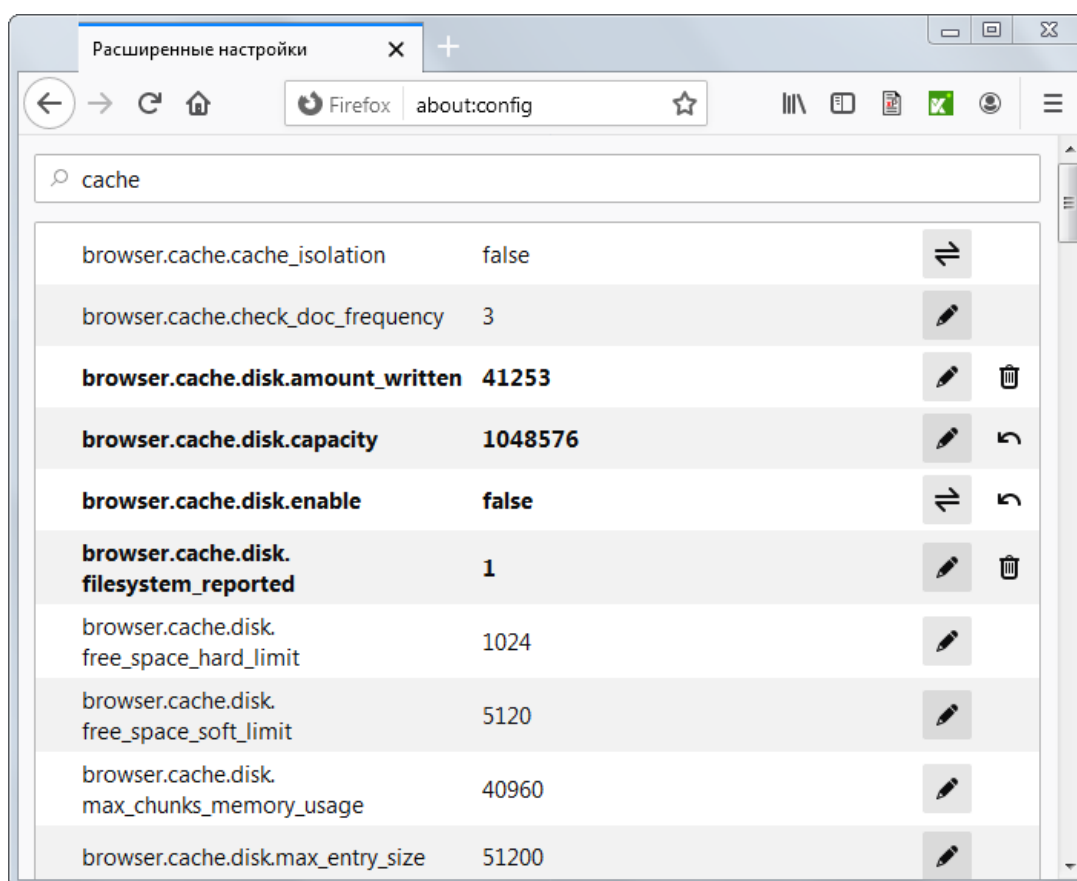
1. Нажать кнопку «Меню» — «Настройки» — «Приватность и защита» — «История» — «Удалить историю» или *Ctrl + Shift + Del*.
2. В списке «Удалить» выбрать период, за который надо удалить кеш.
3. Включить опцию «Кеш».

4. Отключить остальные опции, если надо очистить только кеш браузера.
5. Нажать «Удалить сейчас».

Кроме дискового кеша, есть и более «быстрый» кеш в оперативной памяти, где браузер также временно хранит информацию об открытых страницах и загруженном контенте. Он очищается довольно просто: надо закрыть браузер, дождаться завершения процесса и запустить его снова.

В общем случае отключить кеш совсем нельзя. Все браузеры рекомендуют использовать режим инкогнито, если надо работать в интернете без взаимодействия с кешем. Из браузеров только Firefox позволяет полностью отключить дисковый кеш. Для этого потребуется:

1. Ввести в адресной строке `about:config`.
2. Прочитать предупреждение и согласиться с ним.
3. В настройках найти **`browser.cache.disk.enable`**.
4. Переключить положение на `false`.



Cookie

HTTP-cookie — это небольшой фрагмент данных, отправляемых с веб-сайта и сохраняемых браузером на пользовательском компьютере во время просмотра сайта. Cookie разработаны, чтобы сайты:

- надёжно запоминали информацию о состоянии, например, товары, добавленные в корзину интернет-магазина;
- или сохраняли информацию о действиях пользователя на сайте, включая нажатие на конкретные кнопки, вход в систему или посещённые страницы;
- запоминали информацию, которую пользователь ранее вводил в поля формы — имена, адреса, пароли.

Cookie выполняют важные функции в современном интернете. Возможно, самая важная: cookie — это наиболее распространённый метод, которым веб-серверы определяют, вошёл пользователь в систему или нет, и если да, то с какой учётной записью. Без такого механизма сайт не знал бы, отправлять ли страницы, содержащие конфиденциальную информацию, или требовать от пользователя аутентификацию при входе в систему.

Безопасность cookie-аутентификации обычно зависит от безопасности сайта и браузера пользователя, а также от того, зашифрованы ли данные cookie при передаче. Уязвимости в безопасности приведут к тому, что хакер прочтёт cookie и использует их для получения доступа к пользовательским данным или входа на сайт с учётными данными пользователя. Одна из самых распространённых уязвимостей в веб-приложениях — межсайтовый скриптинг — завязана на получении пользовательских cookie.

Cookie также используются для отслеживания действий пользователей на сайте. Как правило, это требуется для сбора статистики, на основе которой рекламные компании формируют анонимные профили пользователей для более точного таргетирования рекламы.

Типы cookie

Сессионные cookie

Сессионные (временные) cookie есть только в оперативной памяти и только пока пользователь находится на сайте. Браузеры обычно удаляют сессионные cookie после закрытия окна браузера. В отличие от других типов cookie, сессионные cookie не имеют истечения срока действия, и поэтому браузеры понимают их как временные.

Постоянные cookie

Постоянные cookie-файлы удаляются в конкретную дату или через определённый промежуток времени. Это означает, что информация о cookie будет передаваться на сервер каждый раз, когда пользователь посещает веб-сайт, которому эти cookie принадлежат. Такие cookie иногда называются следящими, поскольку рекламодатели обычно записывают с их помощью предпочтения пользователя в течение долгого времени. Однако cookie используются и в «мирных» целях, например, чтобы избежать повторного ввода данных при каждом посещении сайта.

Сторонние cookie

Как правило, атрибут домена cookie совпадает с доменом, отображаемым в адресной строке браузера, однако сторонний cookie принадлежит не тому домену, который указан в адресной строке. Такой тип файлов, как cookie обычно появляется, когда веб-страницы содержат контент с других сайтов, например, рекламные баннеры. Это позволяет отслеживать историю посещений пользователя, что часто используют рекламодатели для предоставления релевантной рекламы.

В качестве примера предположим, что пользователь посещает `www.example.org`. Этот веб-сайт содержит рекламу от `ad.tracking.com`, которая при загрузке устанавливает файл cookie, принадлежащий домену рекламы (`ad.tracking.com`). Затем пользователь посещает другой веб-сайт, `www.foo.com`, который также содержит рекламу от `ad.tracking.com`, и устанавливает файл cookie, принадлежащий этому домену (`ad.tracking.com`). В результате оба этих cookie отправятся рекламодателю при загрузке их рекламы или посещении веб-сайта. Далее рекламодатель использует эти cookie для создания истории просмотров пользователя на всех веб-сайтах, где размещается реклама этого рекламодателя.

Зомби-cookie

Поскольку cookie можно легко удалить, программисты ищут способы идентифицировать пользователей даже после полной очистки истории браузера. Одно из таких решений — зомби-cookie — `evercookie`, или `persistent cookie` — не удаляемые или трудноудаляемые cookie, которые восстанавливаются в браузере через JavaScript.

Это возможно потому, что для хранения cookie сайт одновременно использует все доступные хранилища браузера — HTTP ETag, Session Storage, Local Storage, IndexedDB. В этот список также входят хранилища приложений, таких как:

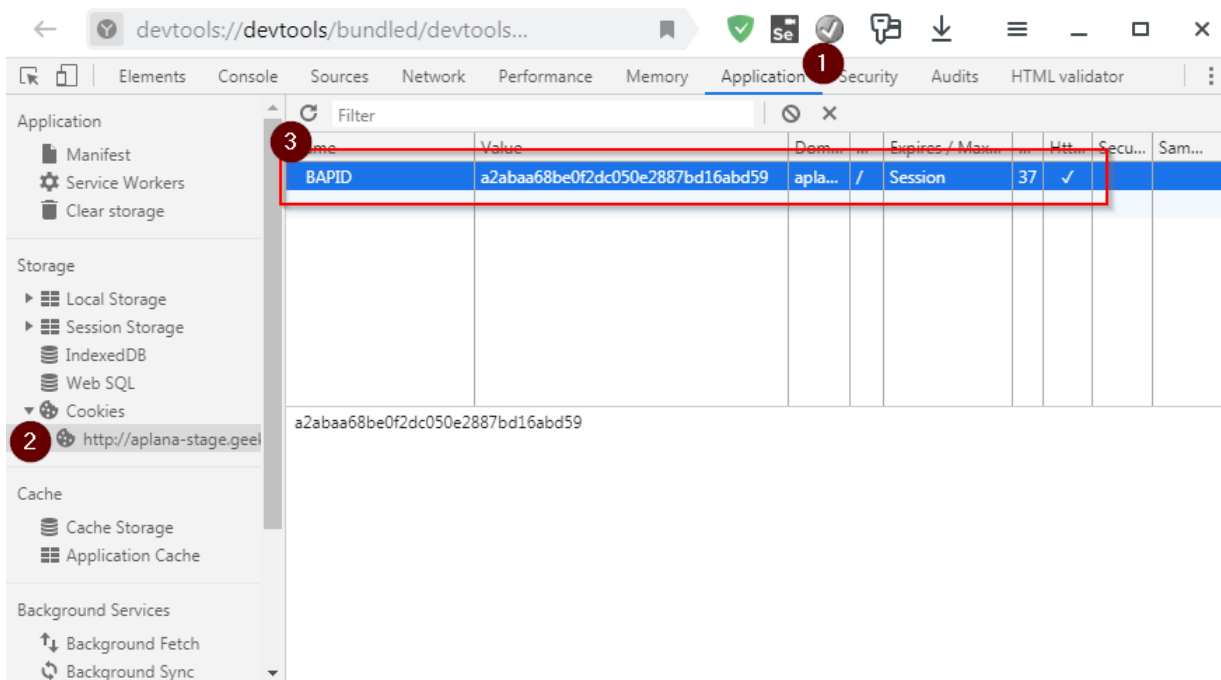
- Flash Player (Local Shared Objects);
- Microsoft Silverlight (Isolated Storage);
- Java (Java persistence API).

Когда программа обнаруживает отсутствие в браузере cookie-файла, информация о котором есть в других хранилищах, она тут же восстанавливает его на место, идентифицируя пользователя для сайта.

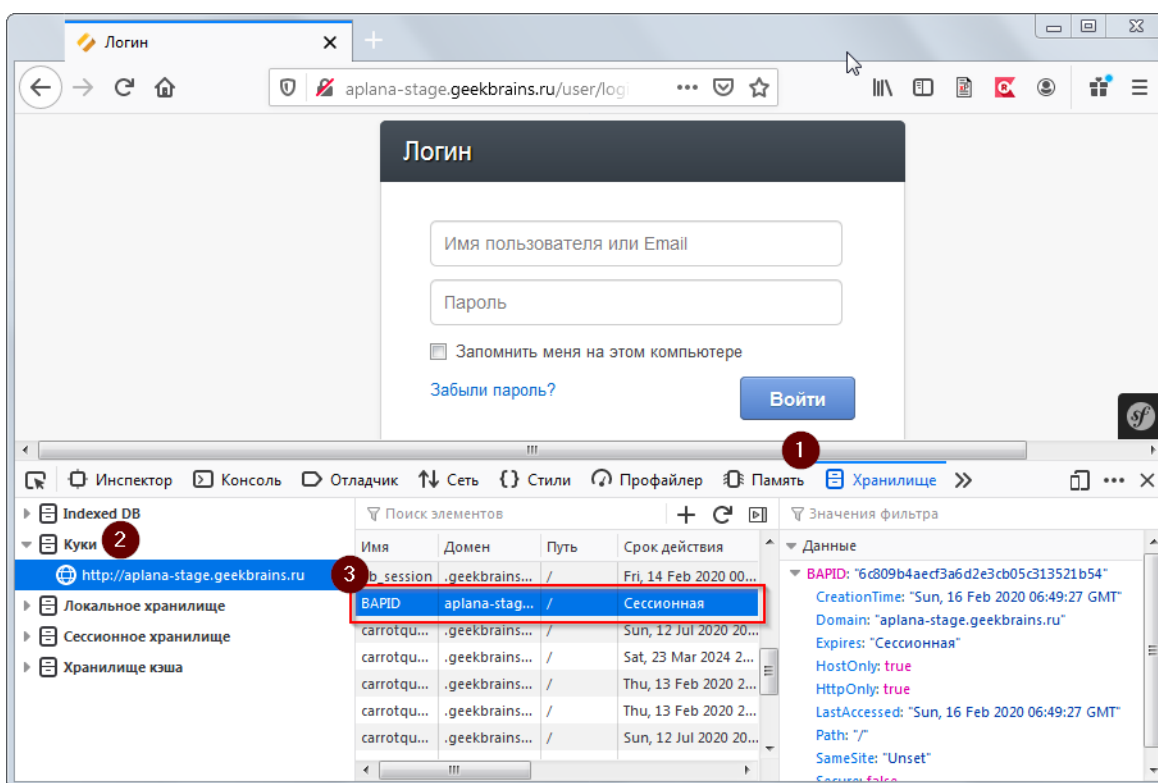
Инструменты для управления cookie

Во всех современных браузерах есть инструменты по управлению cookie. Они позволяют добавлять, изменять и удалять cookie для каждого сайта. Чтобы открыть редактор, надо нажать F12, и появятся инструменты разработчика.

Дальнейшие действия немного различаются в разных браузерах. Для Google Chrome и «Яндекс.Браузера» надо сделать следующее: в верхней части выбираем Application, а в левой находим Cookies и адрес нашего сайта. Теперь в основном блоке отображается список установленных cookie и их параметры.

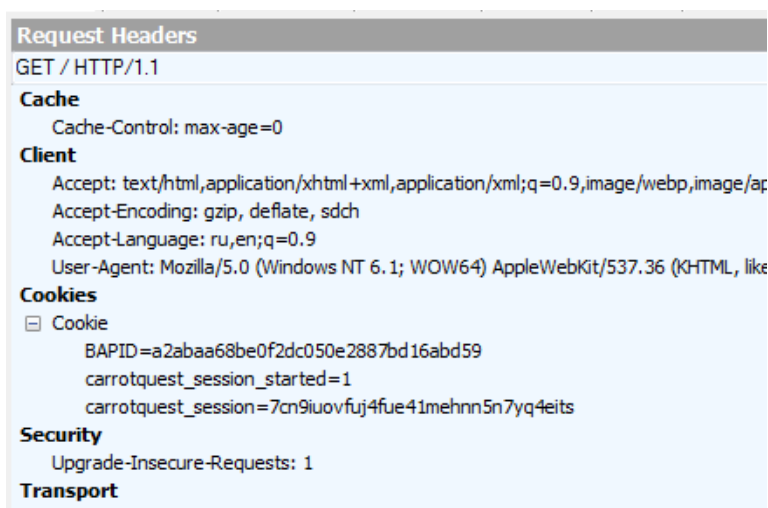


Для Mozilla Firefox: в верхнем списке инструментов выбираем «Хранилище», затем в левом списке — «Куки» и находим подходящий сайт:



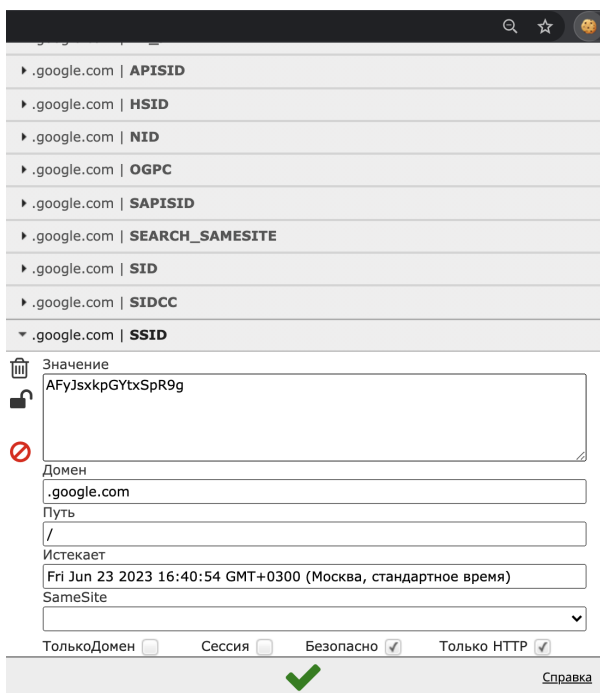
Можно выбрать требуемую cookie и, например, поменять её значение.

Так же просто посмотреть, какие cookie передаются с каждым запросом, применяя инструменты типа Fiddler.



[Edit This Cookie](#) — это расширение для браузеров Chrome, Yandex и Opera, которое позволяет изменять данные в этих файлах.

После установки расширения Edit This Cookie в интерфейсе браузера появится значок в виде «печеньки», нажав на который появится доступ к функциям и настройкам. Чтобы посмотреть куки-файлы, надо зайти на сайт, и при нажатии на значок «изменить этот файл Cookie» появится окно, где отобразятся все активные файлы cookie.



Подмена cookie

Подставим другую cookie и зайдём так на сайт без фактической авторизации. Кстати, подобный метод используют и злоумышленники: они похищают cookie через уязвимости на сайте или незащищённое интернет-соединение.

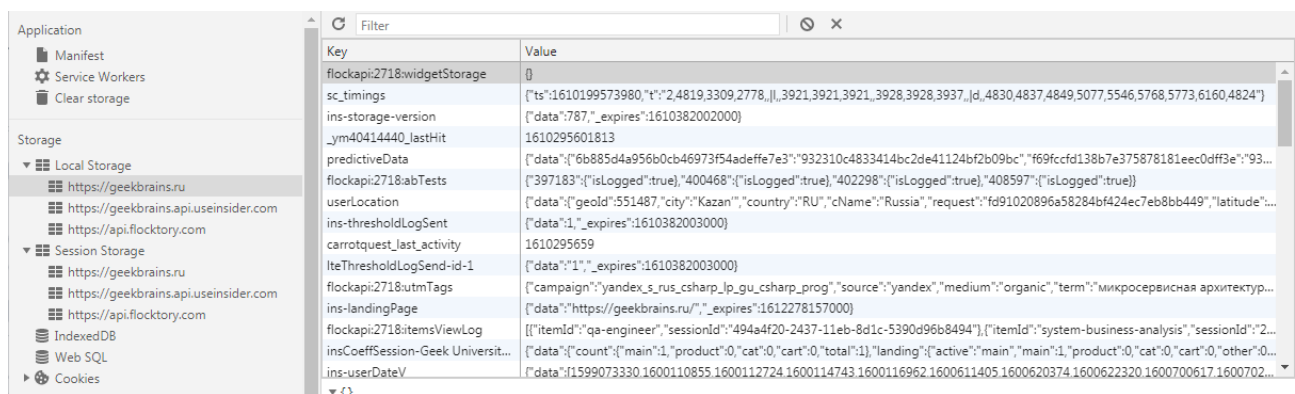
Для чистоты эксперимента нам понадобится два браузера, например, Chrome и Firefox или Опера, «Яндекс.Браузер». Первоначальные условия: пользователь не авторизован на портале ни в одном из браузеров.

1. Открываем начальную страницу [CRM Aplana](#) в любом браузере, например, Chrome.
2. Происходит перенаправление на страницу логина.
3. Вводим логин user, пароль 1234.
4. Открываем инструменты разработчика, в них — вкладку с cookie.
5. Находим cookie с именем **BAPID** и копируем её значение.
6. Открываем второй браузер, например, «Яндекс.Браузер» или Firefox.
7. Во втором браузере также переходим на [CRM Aplana](#).
8. Открываем инструменты разработчика во втором браузере, в них — вкладку с cookie.
9. Находим cookie с именем **BAPID** и вставляем значение, скопированное из первого браузера.
10. Открываем снова начальную страницу [CRM Aplana](#) — именно эту, не страницу логина.
11. Пользователь оказывается авторизован как user, хотя логин и пароль мы не вводили.

Веб-хранилища

Веб-хранилище (DOM-хранилище) предоставляет веб-приложениям методы и протоколы для хранения данных на стороне клиента. Такое хранилище поддерживает постоянное хранение данных, аналогично cookie, но предлагает гораздо больший объём и не передаёт их в заголовках с каждым запросом. Так как они не передаются в запросах, для работы с веб-хранилищем надо использовать функции JavaScript.

Есть два типа хранилищ: локальное и сессионное. Они ведут себя аналогично постоянным и сессионным cookie. Веб-хранилище стандартизовано консорциумом World Wide Web (W3C) и WHATWG — рабочая группа по вебу, гипертексту, приложениям и технологиям.

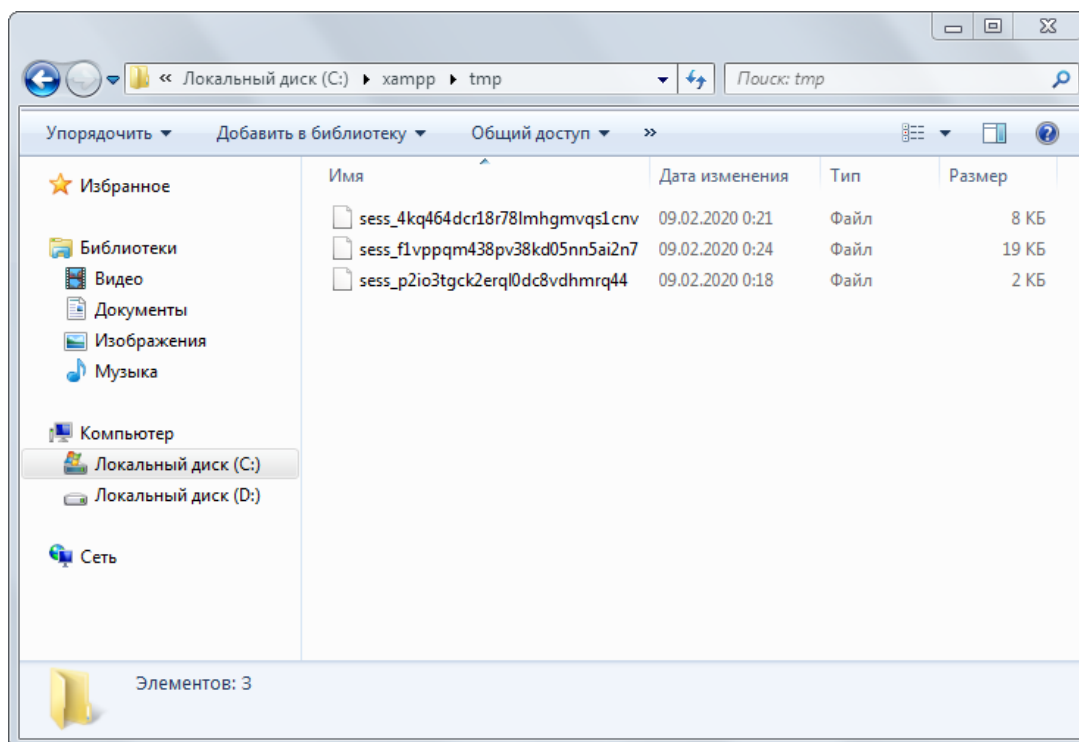


Скриншот из инструментов разработчика Google Chrome, где можно увидеть содержимое веб-хранилища с данными для [geekbrains.ru](#)

Сессии

Как говорилось в первой лекции, HTTP — stateless-протокол, то есть он не хранит информацию о предыдущем состоянии клиента или сервера, об их актуальном статусе. Каждый HTTP-запрос обрабатывается как новый запрос, соответственно, имеющимися средствами HTTP невозможно понять, идут ли запросы от одного пользователя или разных, нельзя сохранить какие-либо данные между запросами. Поэтому для длительного взаимодействия появился механизм сессий. Когда мы открываем сайт и отправляем первый GET HTTP-запрос, сервер его обрабатывает и вместе с ответом передаёт конкретный параметр — идентификатор сессии, который хранится на клиенте и в дальнейшем передаётся с каждым запросом. Так сервер понимает, что запрос относится к той или иной сессии, и не требует повторной авторизации.

А так хранится информация о сессиях на сервере Apache с установленным PHP — в виде текстовых файлов в конкретной папке (по умолчанию — /tmp):



Зачастую сессии используют cookie как хранение информации о пользователе. Поэтому если мы хотим принудительно прервать сессию, то используем редактор cookie и удаляем идентификатор пользователя.

Глоссарий

Кеш (cache) браузера — это папка с копиями некоторых данных со страниц, которые мы посещали.

Сессия (session) — это некоторый отрезок во времени, в пределах которого веб-приложение определяет все запросы от одного клиента.

DOM (Document Object Model) – это объектное представление исходного HTML-документа, попытка преобразовать его структуру и содержимое в объектную модель, с которой смогли бы работать различные программы.

HTTP-cookie — это небольшой фрагмент данных, отправляемых с веб-сайта и сохраняемых браузером на пользовательском компьютере во время просмотра сайта.

Практическое задание

Формат сдачи в следующей форме эксель:
https://docs.google.com/spreadsheets/d/13taaz_r2wfo0hcTbfbk49PeNGu_BL0Yys-HUC16NkNI/edit?usp=sharing

Задание 1. Найти баги в форме.

Найдите баги в форме <http://testingcourse.ru/form/>.

На форме есть как минимум десять багов, которые надо найти и описать. Формат описания багов вам известен с предыдущего семестра.

Возможно, вы найдёте больше десяти багов — это хорошо.

Задание 2. Построить полный путь по относительному.

Этот сайт даётя как пример, его в реальности не существует!

Пользователь находится на странице, которая посвящена одной из пород кошек и имеет такой адрес:

<https://animals.verybigencyclopedia.com/catalog/mammals/felidae/cats/british-shorthair/index.html>

На странице есть относительные ссылки, где в параметрах href указано:

1. `//verybigencyclopedia.com/about/`.
2. `/images/cats/british-cat.jpg`.
3. `/scripts/style-cats.css`.
4. `../ragdoll`.
5. `../images/domestic-cat.jpg`.
6. `../../felinae/`.
7. `images/cats/photo-british-cat-01.jpg`.
8. `verybigencyclopedia.com/catalog/mammals`.
9. `#references`.
10. `../british-shorthair/gallery`.

Для каждой ссылки напишите абсолютный путь.

Задание 3. Проверить отправку формы «Вход» при отключённой валидации на клиенте.

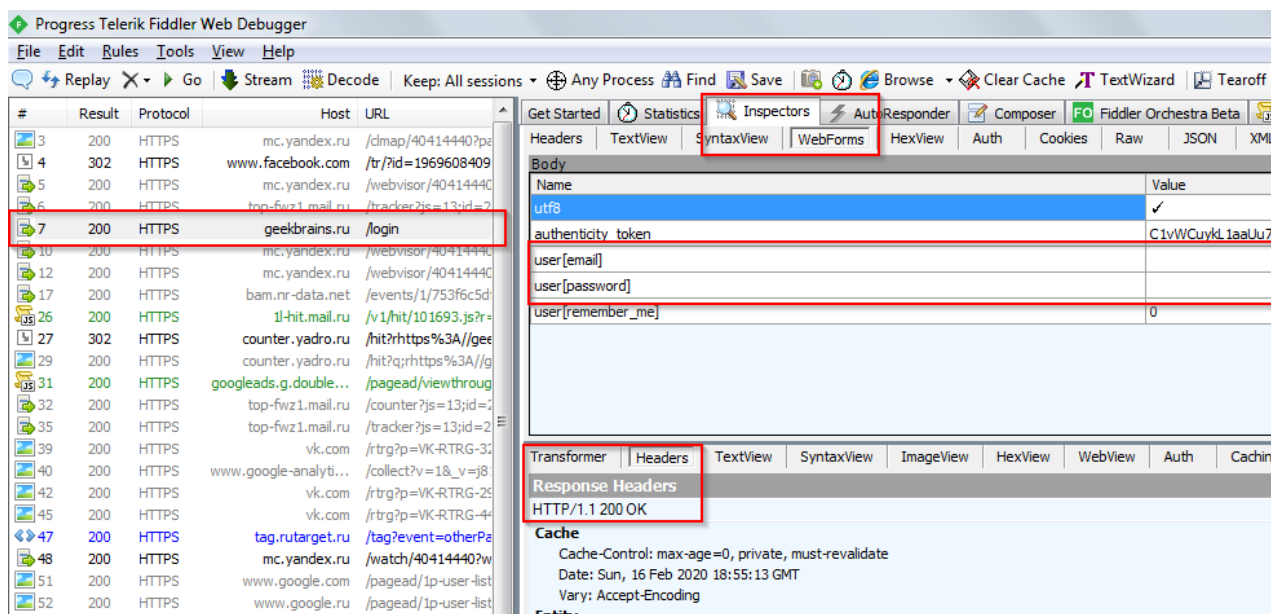
Воспользуйтесь инструкцией из раздела «Обход клиентской валидации» и сделайте то же самое, но для страницы «Вход» сайта GeekBrains (<https://gb.ru/login>).

Необходимо прикрепить:

- скриншоты, на которых демонстрируется изменённый код в DevTools, результат отправки формы с отключённой клиентской валидацией;
- скриншот из Fiddler/Charles, он покажет, что запрос действительно ушёл без этих полей.

Красным выделено то, на что надо обратить внимание:

- выбран верный запрос слева;
- раздел WebForms — во вкладке Inspectors;
- значения полей пустые;
- ответ сервера — 200 OK.



Задание 4. Зайти на сайт через подстановку cookie.

Используйте инструкцию из раздела «Подмена cookie» и зайдите на <http://testingcourse.ru/docs> или на <https://crm.geekbrains.space> в обход страницы логина.

Данные для логина:

user / Password! — testingcourse/docs

user / 1234 — Aplana CRM

Необходимо прикрепить: четыре скриншота — по два из каждого браузера. Один скриншот показывает сайт с залогиненным пользователем, второй — DevTools с cookie. Важно, чтобы значения cookie было видно, в обоих браузерах они должны быть одинаковые.

Задание 5. Тест для самопроверки

<https://coreapp.ai/app/player/lesson/614a22bf8478af554b6f4dbf> (сдавать не нужно)

Задание 6 (необязательное).** Обойти валидацию на клиенте методом отправки POST-запроса к серверу.

Обойдите валидацию для страницы входа <https://qb.ru/login>, но не редактируйте HTML-код, а отправьте POST-запрос к обработчику формы через инструмент Postman/JMeter. В этой методичке информации нет, вы должны найти всё самостоятельно. Ключевые фразы для вбивания в поисковик: «отправка форм через postman», how to send form data from postman и подобные.

Используемые источники

1. [Список зарегистрированных URI-схем.](#)
2. [Использование URI, URL, URN.](#)
3. [Xenu link sleuth.](#)
4. [Онлайн-сервис для проверки ссылок.](#)
5. [Файлы cookie.](#)
6. [Сессии в браузере.](#)
7. [Страница про cookie в «Википедии».](#)
8. [Кеш браузера.](#)
9. [Кеш браузера и другие виды кеширования.](#)