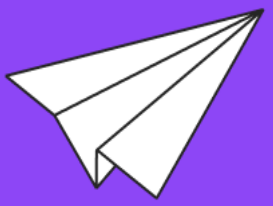
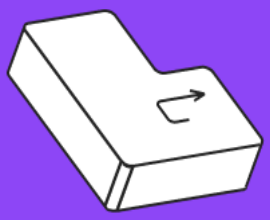
GeekBrains



Тестирование форм

Тестирование веб-приложений



Оглавление

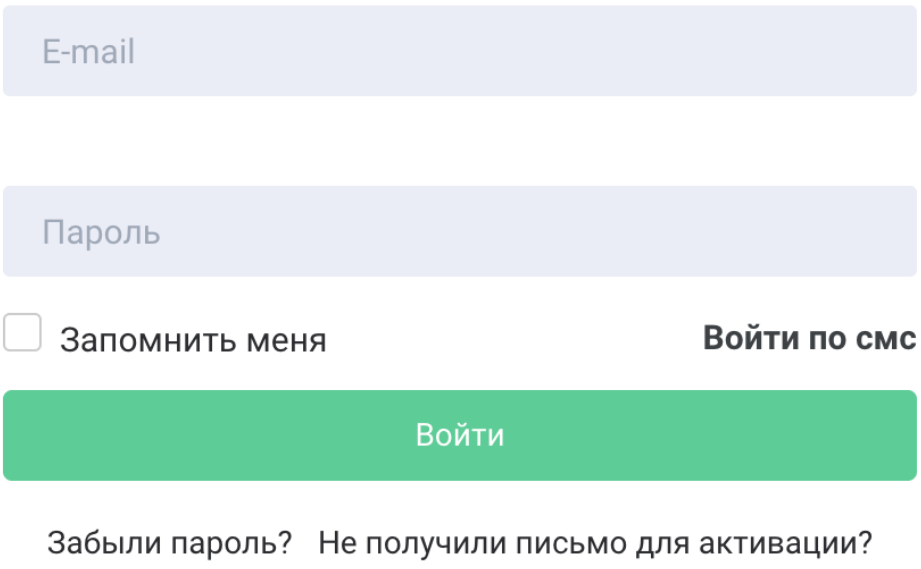
На этом уроке

1. Тестирование форм
2. Тестирование фильтров
3. Тестирование полей поиска

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:



А так — в исходном коде:

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

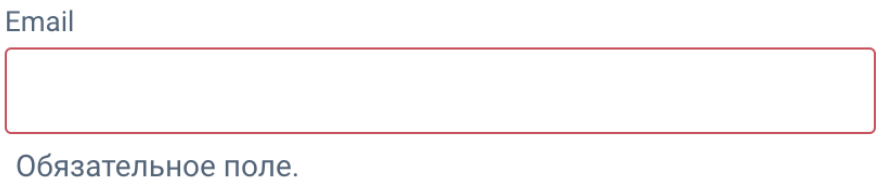
Открывающий тег формы:

```
<form class="new_user" id="new_user" data-parsley-validate="true" ng-non-bindable="true"
action="/login" accept-charset="UTF-8" method="post" novalidate=""
data-gtm-vis-recent-on-screen-1391498_637="592"
data-gtm-vis-first-on-screen-1391498_637="592"
data-gtm-vis-total-visible-time-1391498_637="100"
data-gtm-vis-has-fired-1391498_637="1">
```

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 autofocus="autofocus" class="form-control input-lg" placeholder="E-mail"
required="required" data-testid="login-email" type="email" value="" name="user[email]"
id="user_email">
```

Это поле электронной почты. Тип поля — email, 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. Можно указать атрибут `required` без значения: это не ошибка, а рекомендуемый способ использования данного атрибута.



Аналогично с полем пароля:

```
<input autocomplete="off" class="form-control input-lg parsley-error" placeholder="Пароль"
```

```
required="required" data-testid="login-password" type="password" name="user[password]" id="user_password" data-parsley-id="7">
```

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:

```
<input class="btn btn-block btn-success" type="submit" value="Войти" data-testid="login-submit">
```

У неё нет каких-то особых параметров.

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

```
<a class="h5" href="password/new">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 уникален.

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».
    a. Длина.
        i. Минимум. Какова минимальная длина фамилии? Может ли это быть один символ? Или два? Или три?
        ii. Максимум. Сколько символов будет «необходимо и достаточно»?
    b. Диапазон допустимых символов.
        i. В Ф. И. О. могут быть только буквы. И специальные символы. И иногда цифры.

        ii. А должны ли Ф. И. О. быть только на кириллице? Или могут содержать латиницу? Иероглифы?
        iii. Могут ли Ф. И. О. быть написаны справа налево (иврит, арабская вязь)?
    c. Обязательность к заполнению.
        i. Может ли пользователь не писать фамилию, имя или отчество?
2. Логин.
    a. Длина.
        i. Минимальная длина.
        ii. Максимальная длина.
    b. Диапазон допустимых символов.
    c. Необходимость.
        i. 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.
    a. Проверка формата.
    b. 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, то идеи из статьи стоит принять во внимание.

Авторизация

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

Поиск

1. По наличию искомого.
    a. Результаты существуют.

        i. Найдено всё соответствующее.
        ii. Поисковому запросу не соответствует ничего не актуального.
    b. Результаты не существуют.
        i. В поиск не попало ничего не актуального.
        ii. Выдано корректное сообщение о пустом результате.
2. По типу поиска.
    a. По точному совпадению.
    b. Умный поиск, по неточному совпадению.
    c. С использованием логических операторов, если они поддерживаются.

Язык запросов

Поговорим про язык запросов. Иногда надо найти точную фразу, в этом нам поможет оператор **двойные кавычки**. Если вы заключите слова запроса в кавычки, то мы будем искать документы, которые содержат именно такой порядок слов без каких-либо изменений. Например, ["о чём ещё говорят мужчины"] найдёт документы, которые содержат информацию именно о второй части фильма «О чём говорят мужчины», пропуская страницы с первым фильмом.

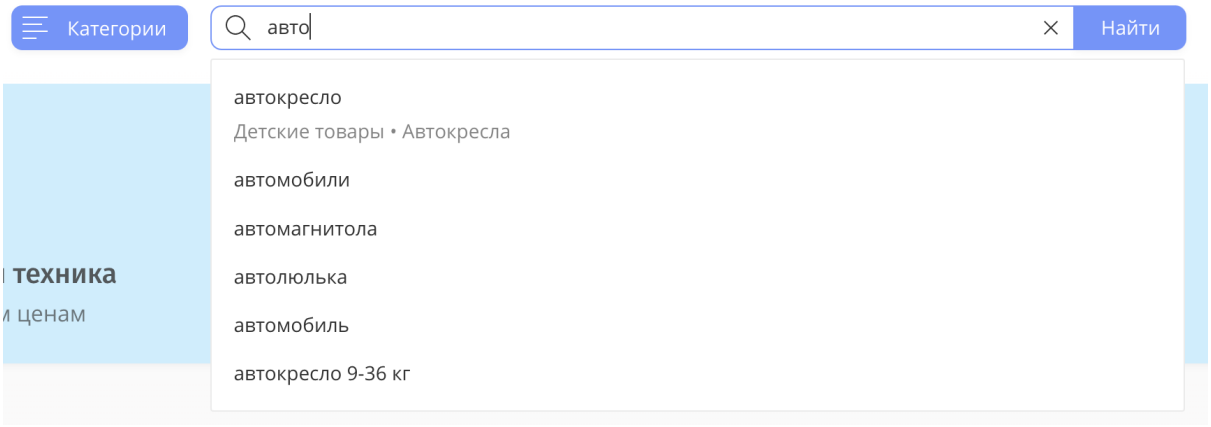
Оператор **плюс** предлагает найти все тексты, содержащие искомое слово в любой форме. Например, запрос [сиамские кошки + подоконник] вернёт все документы с «подоконником» в любой словоформе.

Если поисковая система не находит ничего, удовлетворяющего вашему запросу, она начинает игнорировать действие операторов. Например, ["роза пахнет розой хоть фиалкой назови её хоть нет"]. В отсутствие точно совпадающей цитаты поиск предлагает все документы, где встречаются указанные слова.

Если вы уверены, что интересующая вас информация находится на определённом сайте, используйте оператор «**site:**». Например, на запрос [site:health.mail.ru витамины мифы] вы получите все документы с сайта «Здоровье Mail.ru», в которых есть слова «витамины» и «мифы».

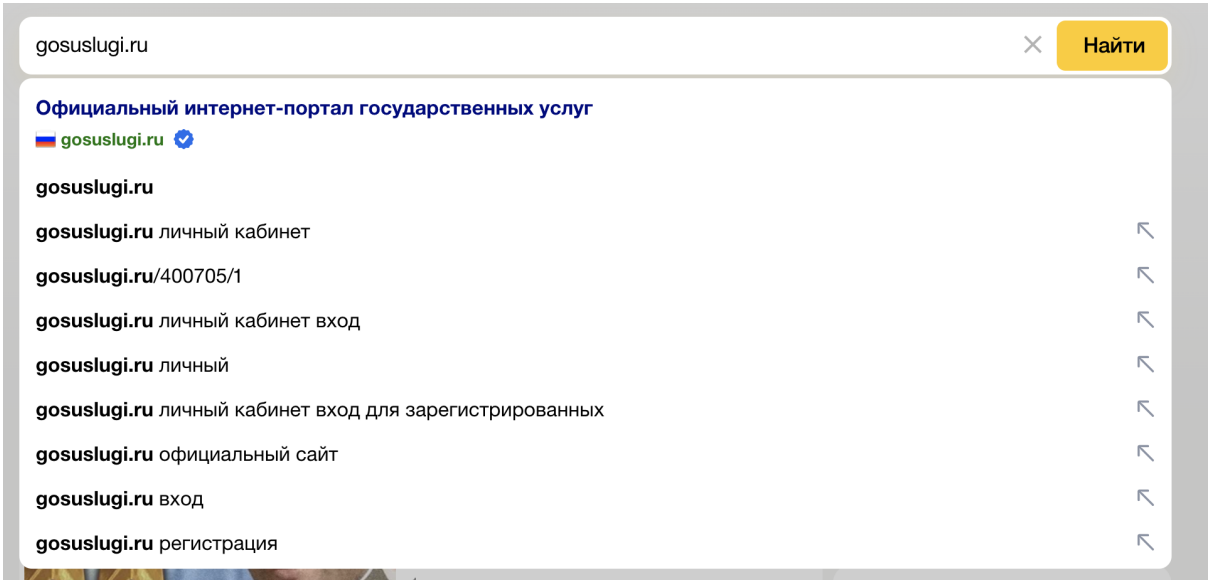
Suggests (подсказки)

Поисковые подсказки («саджесты») — это продолжения запроса, который вы начинаете вводить. Они помогают уточнить и быстро выбрать подходящий вариант запроса, не набирая его целиком. Например, введём слово «автокресло». Начинаем вводить «авто» и видим подсказки:



Саджест обновляется по мере ввода букв или слов. Подходящий запрос можно выбрать как мышкой, кликнув на него, так и передвигаясь при помощи клавиш «вверх/вниз».

Чтобы пользователь быстрее получал искомое, в некоторых случаях подсказки позволяют непосредственно перейти на нужный сайт, не обращаясь к результатам поиска.



Фильтры

1. По результату.
    a. Фильтр возвращает непустой результат.
        i. В фильтр попали все подходящие данные.
        ii. 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

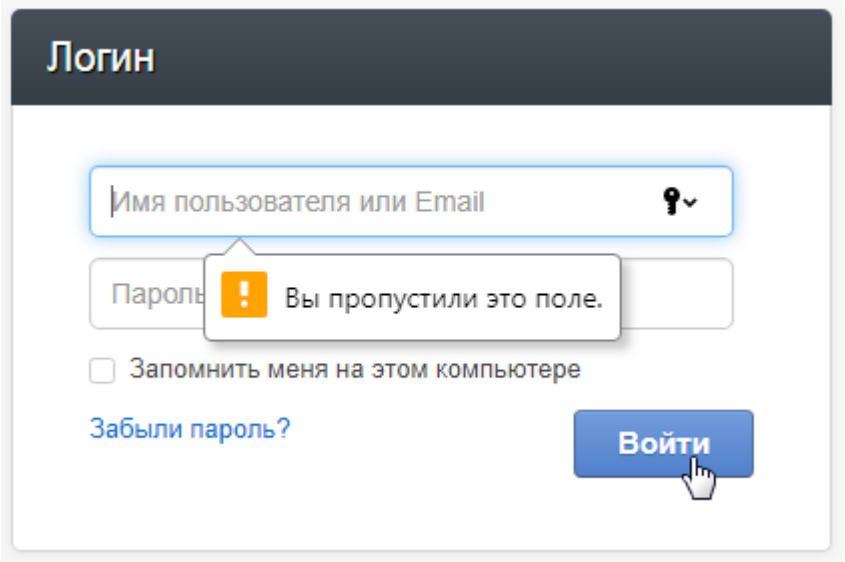
Эти формы часто содержат лишь несколько текстовых полей «Тема» и «Сообщение», если пользователь зарегистрирован, и «Имя» и «Электронная почта» или «Телефон», когда регистрации на сайте нет. Их тестирование в этом случае мало отличается от тестирования, скажем, полей ввода в формах регистрации.

Обход клиентской валидации

Валидация данных в форме на клиенте — это, безусловно, хорошо: пользователь получает мгновенную обратную связь без необходимости отправлять данные на сервер. Но этого способа недостаточно, и все полученные данные обязательно надо проверять на сервере. Без проверки полученных от клиента данных приложение уязвимо для взлома и сбоев в работе. 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.

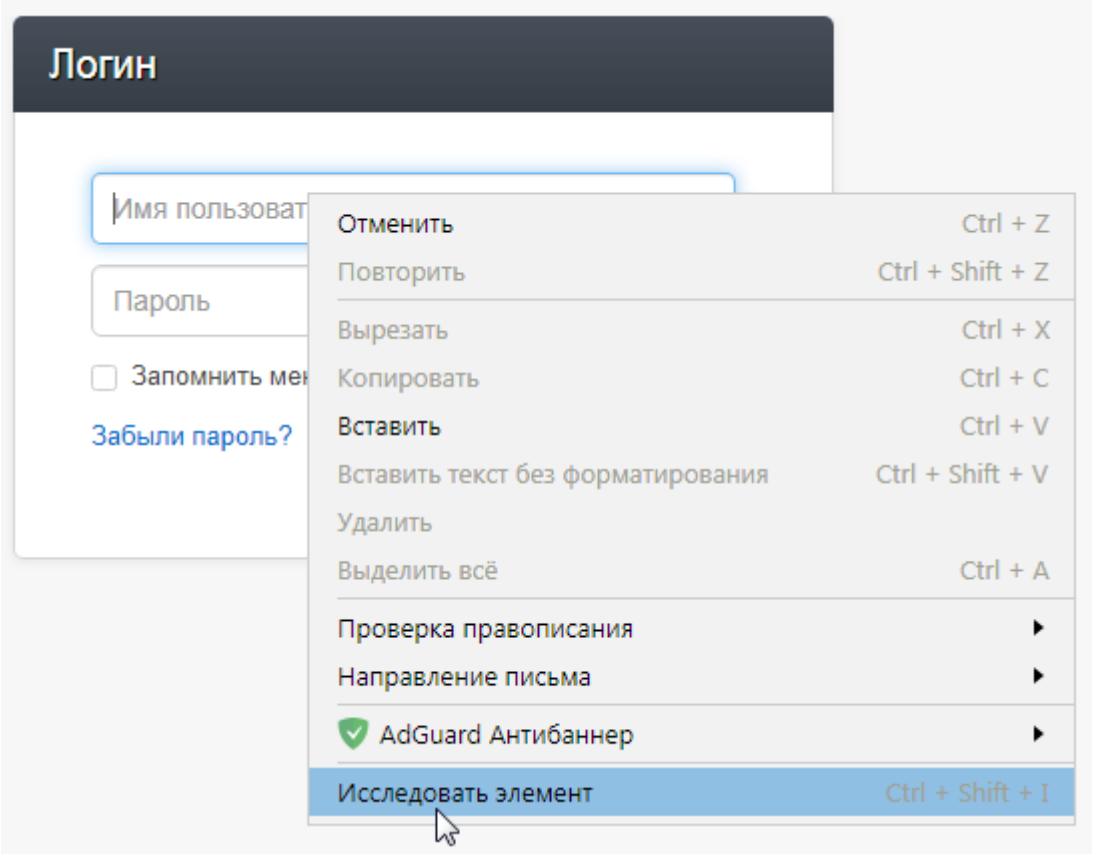
Вариант 1. Отключение фильтрации методом редактирования HTML

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. Например, имеется 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»:



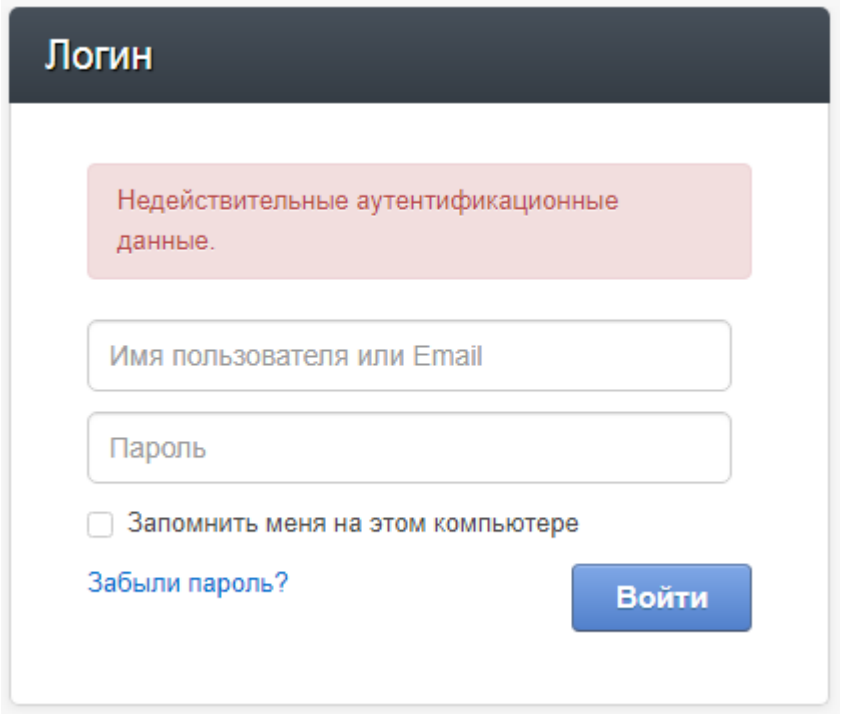
Откроется окно с HTML-кодом. Этот код можно редактировать. Наши изменения отобразятся на открытой странице, но если её перезагрузить или кликнуть на какую-либо ссылку, то изменения потеряются.

```
<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=
  "Ot3Y8wF0lPwmSatYZlO_OiQoMxA__O9fb1rdjiDGBsk">
</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 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 для самостоятельного изучения. Подробно рассмотрим его чуть позже, когда будем знакомиться с инструментом Postman и тестированием 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.

Контрольные вопросы

1) Какие существующие виды валидации вы можете выделить?
2) Для чего необходим язык запросов?
3) Какие способы обхода клиентской валидации имеются?

Дополнительные материалы

1) Тестирование форм оплаты: https://telegra.ph/Payment-gateway-ili-testirovanie-platezhnogo-shlyuza-10-01

2) https://alexeybulat.blogspot.com/2009/05/search.html