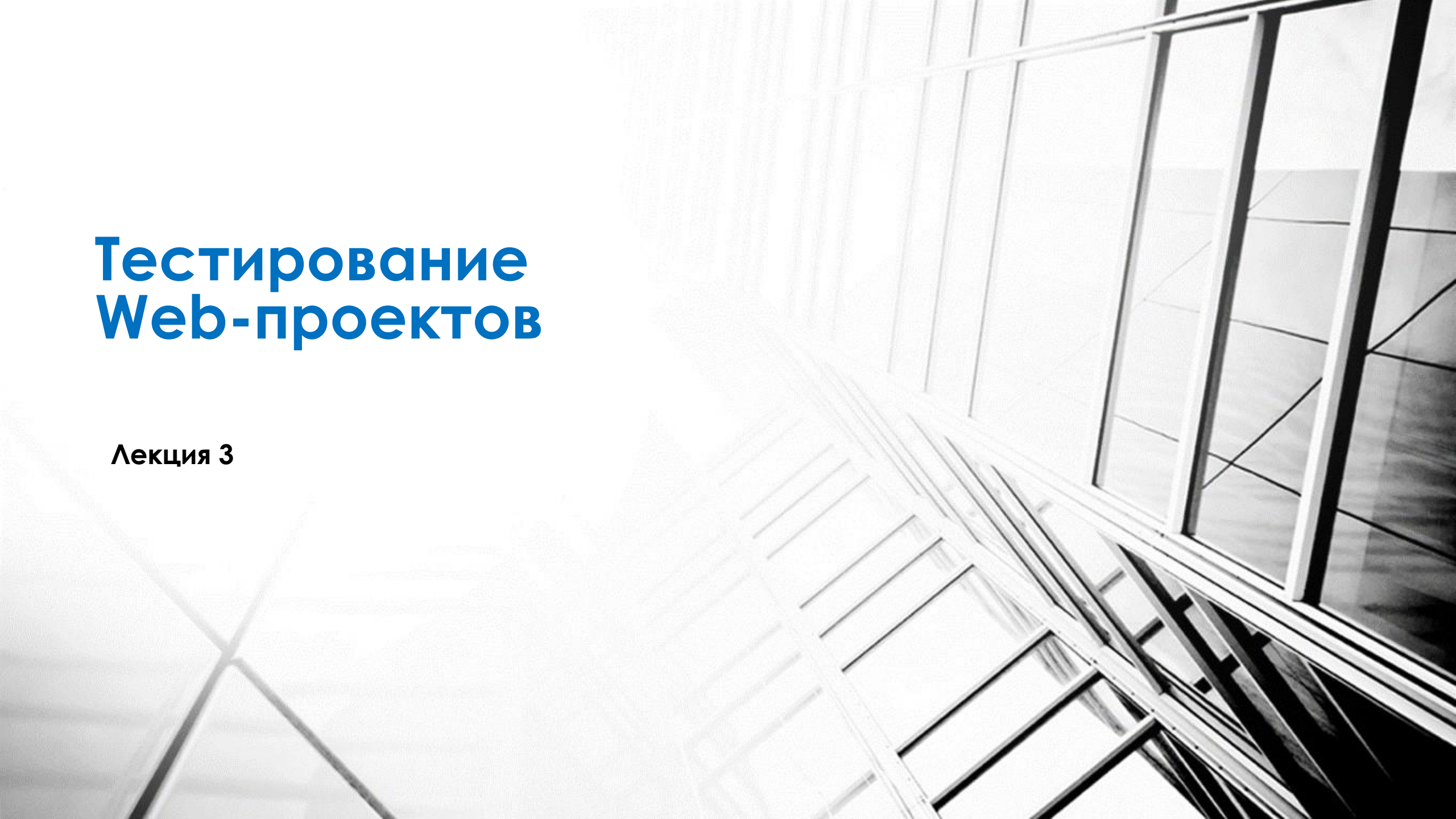


Тестирование Web-проектов

Лекция 3



- Дефекты;
- Тестирование web – приложений;
- Тестирование API;
- Тестирование совместимости;
- Тестирование юзабилити;
- Тестирование форм.

Дефекты



Определение дефекта

Дефект – это несоответствие фактического и ожидаемого результата.

Дефекты могут встречаться в любой документации, в архитектуре и дизайне, в коде программы и т.д. Иногда дефекты на самом деле являются не ошибкой в программе, а результатом неверного конфигурирования программы и/или окружения.

Поиск и документирование дефектов

Тестирование – это не поиск ошибок!

Главное правило тестирования – не надо стараться найти как можно больше ошибок, надо стараться пропустить как можно меньше!

Кто может задокументировать дефект?

Задокументировать дефект может кто угодно, обнаруживший некорректное поведение программы:

- Тестировщики и специалисты по обеспечению качества
- Разработчики
- Представители службы технической поддержки
- Продавцы и специалисты по маркетингу
- Представители заказчика
- Конечные пользователи

Отчеты об ошибках

- **Баг/Дефект Репорт (Bug Report)** - это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.
- **Отчёт об ошибке («баг-репорт», «bug report»)** – один из основных результатов работы тестировщиков. И, к слову, именно этот результат работы видят коллеги (другие тестировщики и люди, не входящие в команду тестировщиков).

Цель написания отчета об ошибке

Отчеты об ошибках пишутся с целью:

- предоставить информацию о проблеме, ее свойствах и последствиях;
- приоритизировать проблему по важности и скорости устранения;
- помочь программистам обнаружить и устранить источник проблемы.

Пример отчета об ошибке

Field Name	Content
Title:	«Корзина»: система не реагирует на нажатие кнопки «Оформить заказ»
Bug ID:	WS_VELC_20080625_001
Build Number:	Version Number 5.0.1
Environment:	Windows 7, Chrome 31.0.1650.63 m
Severity:	Low
Priority:	Blocker
Assigned to:	Developer-X
Reported By:	Your Name
Reported On:	Date
Type:	Bug
Status:	New/Open/Active (Depends on the Tool you are using)

Пример отчета об ошибке

Field Name	Content
Steps To Reproduce:	1) Выбрать любой товар 2) Нажать кнопку «Добавить в корзину» 3) Перейти в корзину 4) Нажать кнопку «Оформить заказ»
Expected result:	Пользователь перенаправлен на страницу оформления заказа
Actual Result:	Пользователь остался на странице «Корзина»
Attachment:	Attached video and screenshot

Атрибуты отчета об ошибке

Основные (обязательные) атрибуты:

- Идентификатор (id)
- Заголовок (title)
- Шаги воспроизведения (steps to reproduce, STR)
- Фактический результат (actual result)
- Ожидаемый результат (expected result)
- Приоритет (priority)
- Важность (severity)

Идентификатор (id)

- У каждого отчета об ошибке должен быть уникальный идентификатор.

Например:

- Аббревиатура проекта + дата + порядковый номер
WSVELC20080625001 Или
WS_VELC_20080625_001

Заголовок (title)

Хороший заголовок должен давать ответы на три вопроса:

- Что?
- Где?
- При каких условиях? Или когда?

Например:

- Где? – На странице «Корзина»
- Что? – Система не реагирует на нажатие кнопки «Оформить заказ»
- При каких условиях? Или когда? – Когда в корзине пользователя есть товар

Шаги воспроизведения

Несколько рекомендаций:

- Начинайте каждый шаг с глагола в инфинитиве.
- Используйте пункты с нумерацией.
- Используйте простые предложения.
- Избегайте лишних шагов.
- Перечитайте, убедитесь, что все корректно и достаточно для понимания.

Пример:

1. Выбрать любой товар
2. Нажать кнопку «Корзина»
3. Перейти в корзину
4. Нажать кнопку «Оформить заказ»

Приоритет

Обычно, выделяют такие значения срочности:

- **Наивысшая** (*ASAP, as soon as possible* иногда пишут *blocker*). Присваивается ошибкам, наличие которых делает невозможным дальнейшую работу над проектом или передачу заказчику текущей версии проекта.
Высокая (*high*). Присваивается ошибкам, которые нужно исправить в самое ближайшее время.
Обычная (*normal*). Присваивается ошибкам, которые следует исправлять в порядке общей очереди.
Низкая (*low*). Присваивается ошибкам, которыми отделу разработки следует заниматься в последнюю очередь (когда и если на них останется время).

Важность

Обычно, выделяют следующие уровни важности:

- **Блокирующая (blocker).** Это самые страшные ошибки, выражающиеся в крахе приложения или операционной системы, серьёзных повреждениях базы данных, падению веб-сервера или сервера приложений.
- **Критическая (critical).** Серьёзные ошибки, такие как: потеря данных пользователя, падение значительной части функциональности приложения, падение браузера или иного клиента и т.п.
- **Значительная (major).** Ошибки, затрагивающие небольшой набор функций приложения. Как правило, такие ошибки можно «обойти», т.е. выполнить требуемое действие иным способом, не приводящим к возникновению ошибки.
- **Низкая (minor).** Ошибки, не мешающие непосредственно работе с приложением. Как правило, сюда относятся всевозможные косметические дефекты, опечатки и т.п.

Атрибуты отчета об ошибке

Дополнительные (необязательные) атрибуты:

- Тестовое окружение (environment)
- Предварительные условия (precondition)
- Воспроизводимость (reproducible)
- Приложения (attachments)

Тестовое окружение

Это поле заполняется тогда, когда дефект воспроизводится при конкретных условиях.

- Операционная система
- Браузер
- Ссылка на окружение

Предварительные условия

Условия окружения и состояния, которые должны быть выполнены перед началом выполнения определенного теста или процедуры тестирования.

Воспроизводимость

Это поле показывает,
воспроизводится ли баг всегда
(«always») или лишь иногда
(«sometimes»).

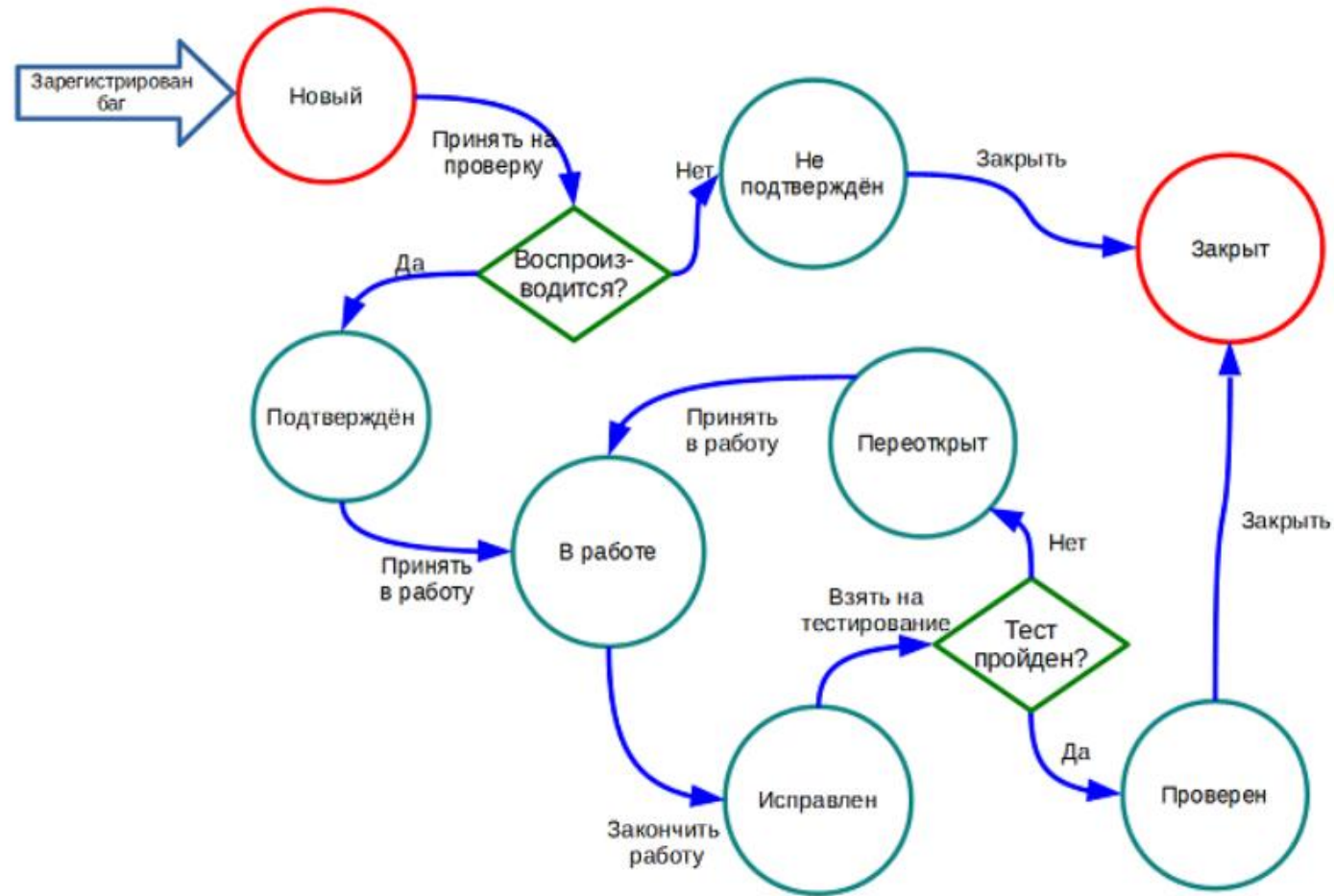
Приложения

Лучший способ указать на баг – приложить к баг-репорту некую наглядную информацию: скриншоты, видеоролики, логи (журналы событий), изображения и файлы.

Жизненный цикл дефекта

- Обнаружен (submitted).
- Назначен (assigned).
- Исправлен (fixed).
- Проверен (verified).
- Закрыт (closed)
- Открыт заново (reopened).
- Рекомендован к отклонению (to be declined).
- Отклонён (reject).
- Отложен (deferred).

Жизненный цикл дефекта



Какой отчёт об ошибке является плохим?

- Отчёт, который не даёт достаточной информации «Программа не работает», «Приложение виснет».
- Отчёт об ошибках в той части функциональности, которая не заявлена как реализованная в данном билде.
- Отчёт, дающий некорректную информацию (в любом из полей).
- Отчёт, написанный с грамматическими ошибками и/или с использованием жаргонной, непонятной большинству лексики.
- Отчёт, критикующий работу программиста.

Преимущества хорошего отчёта об ошибках

Хороший отчёт об ошибках помогает:

- Сократить количество ошибок, «возвращаемых» разработчиками (отклонённых или открытых заново).
- Ускорить устранение ошибки.
- Сократить стоимость исправления ошибки.
- Повысить репутацию тестировщика.
- Улучшить взаимоотношения между командами тестирования и разработки.

Баг-трекинг-овые системы

- JIRA
- Redmine
- Bugzilla
- QC

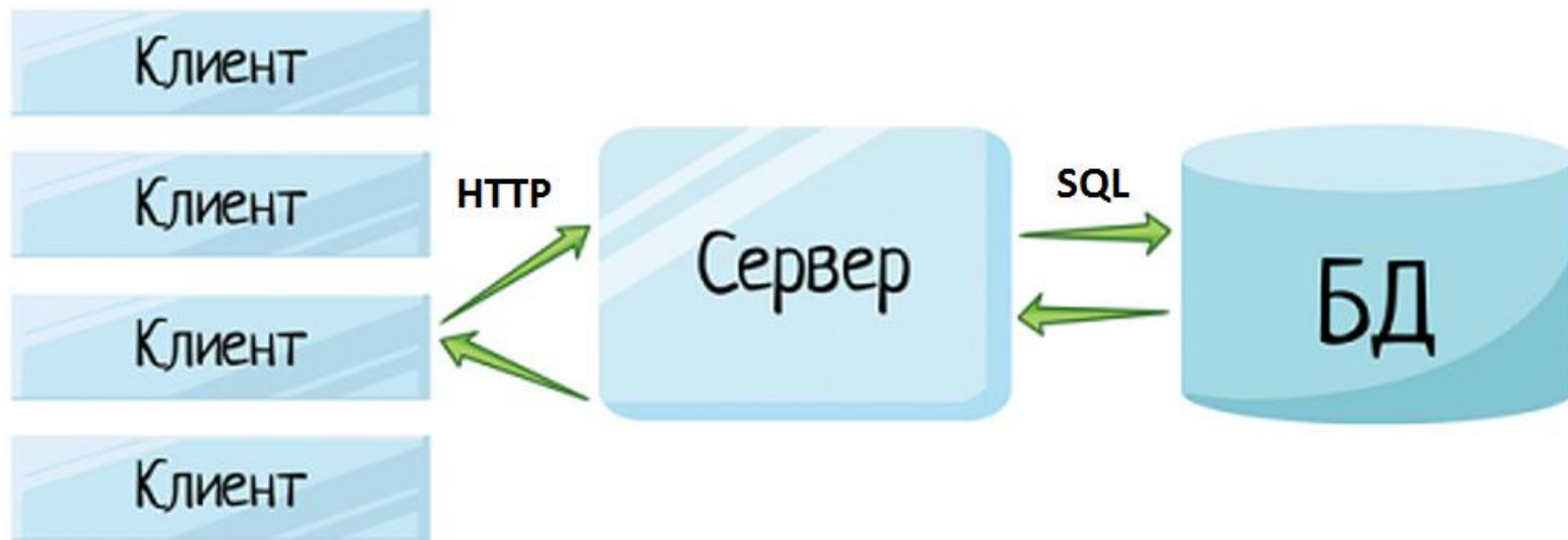


Тестирование web - приложений

Что такое web – приложение?

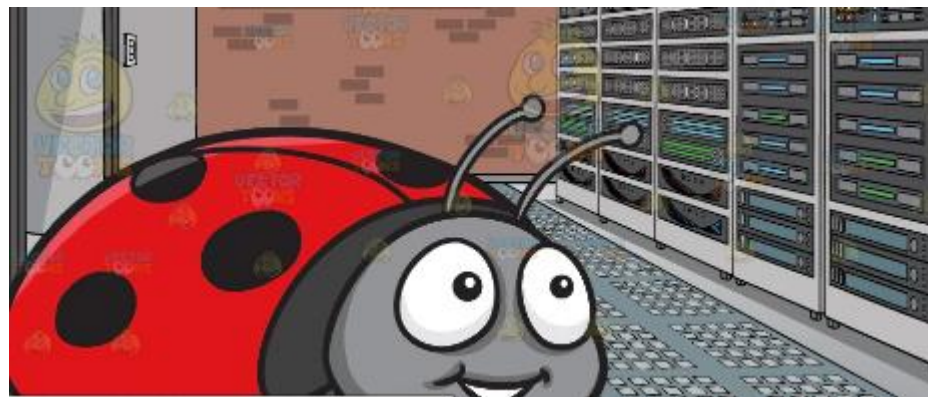
Клиент-серверное приложение, основная часть которой содержится на удаленном сервере, а пользовательский интерфейс (UI) отображается в браузере в виде web-страниц.

Трехуровневая архитектура



Что тестируем

Баг может находиться в каждом из частей приложения и во взаимодействии между ними.



Что тестируем



FRONTEND

- Вёрстка на HTML, CSS, программирование на JavaScript
- Адаптивность
- Кроссплатформенность
- Поисковая оптимизация
- Редактирование изображений
- Кроссбраузерные проблемы
- Асинхронные запросы и AJAX

BACKEND

- Программирование на языках PHP, Python, Ruby...
- Администрирование БД
- Масштабирование
- Безопасность
- Серверная архитектура
- Резервное копирование
- Обработка данных



Тестирование UI и верстки

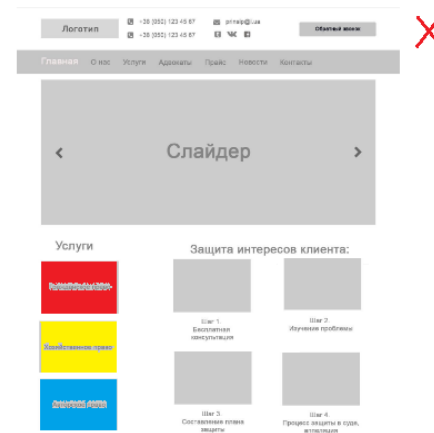
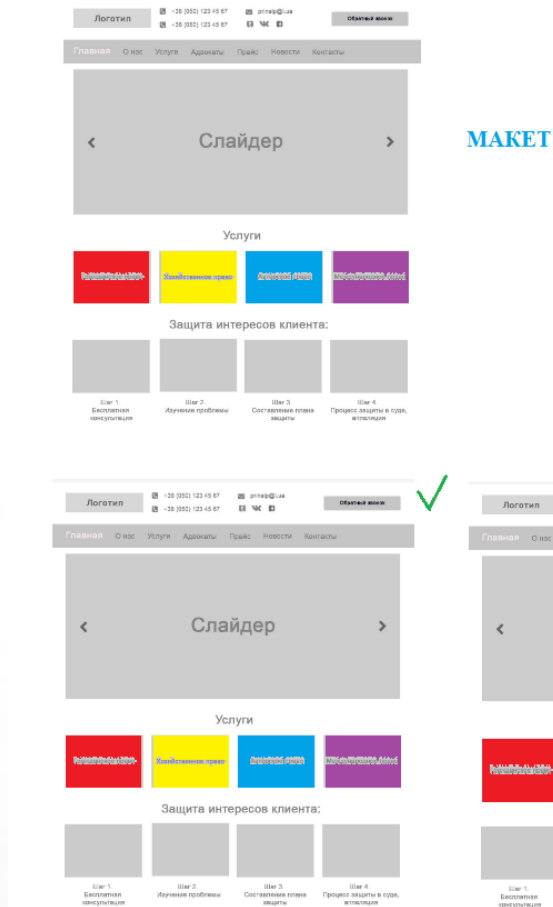
UI (user interface — пользовательский интерфейс) - является точкой взаимодействия человека и продукта. Дизайн кнопок, полей ввода и т.д. - это место, где пользователь взаимодействует с системой.

Тестирование интерфейса пользователя осуществляется вместе со следующими видами тестирования (UI):

- Тестирование на соответствие стандартам графических интерфейсов.
- Тестирование с различными разрешениями экрана.
- Тестирование кроссбраузерности или совместимости с разными интернет браузерами и их версиями.
- Тестирование локализованных версий: точность перевода (мультиязычность, мультивалютность), проверка длины названий элементов интерфейса и т.д..
- Тестирование графического интерфейса пользователя на целевых устройствах (смартфоны, планшеты и т.д.).

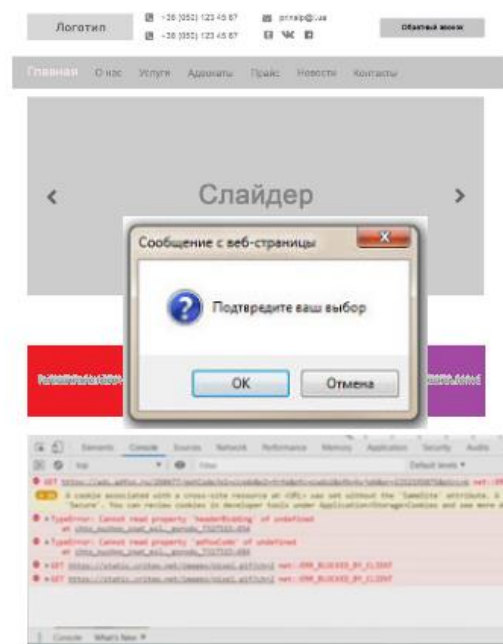
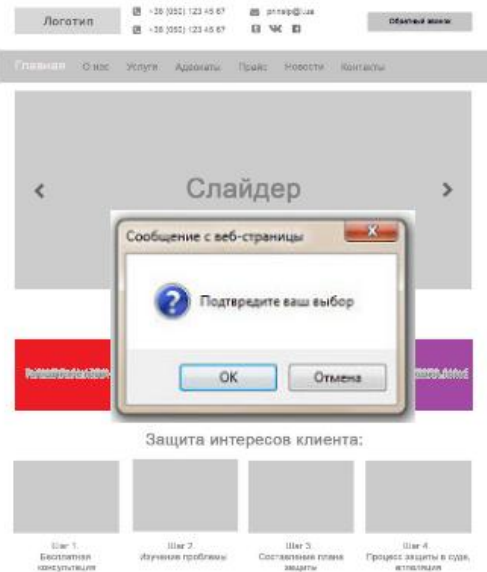
Что тестируем

Элементы web-страницы могут выглядеть и располагаться неправильно.



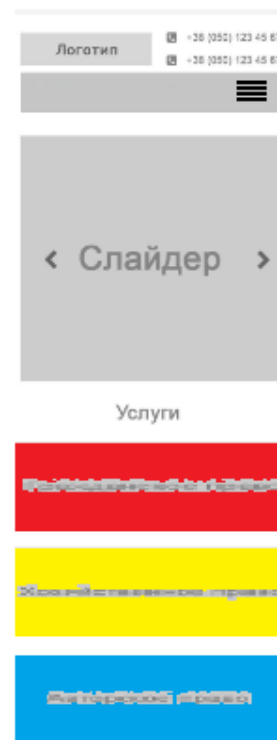
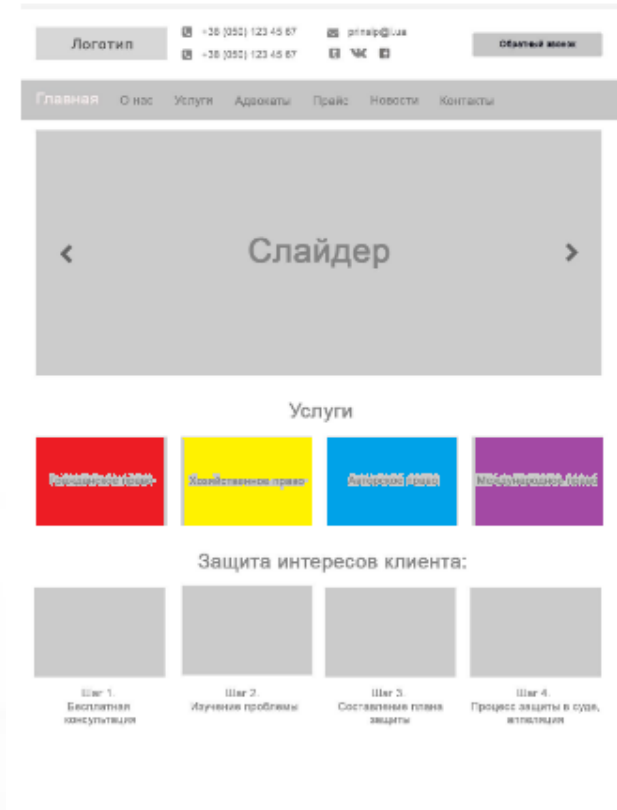
Что тестируем

Интерактив web-приложения может работать некорректно или не работать вообще.



Что тестируем

Взаимное расположение элементов может зависеть от размера экрана браузера



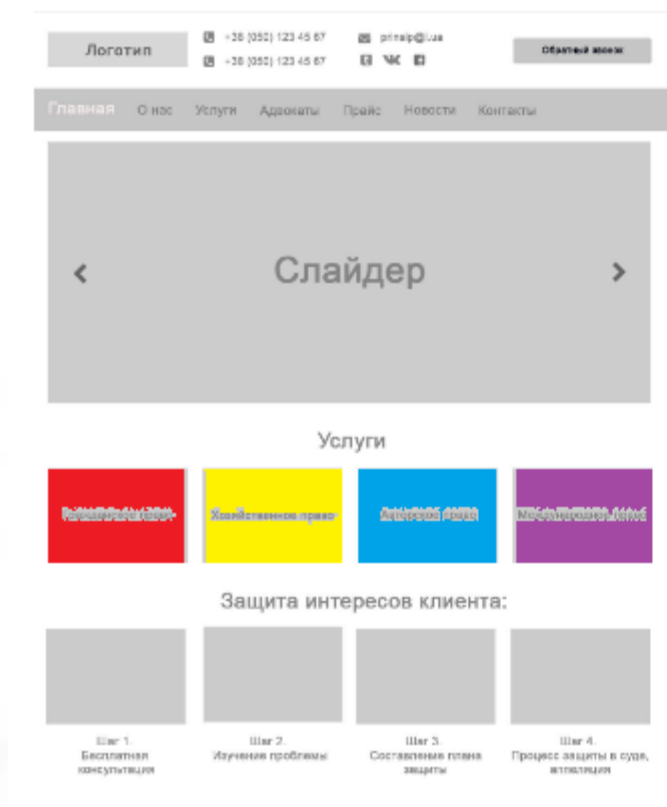
Что тестируем

**Web-страницы могут по-разному отображаться
в разных браузерах**



Что тестируем

Кастемизация (Customization). Данные пользователя могут задавать поведение приложения.



Что тестируем

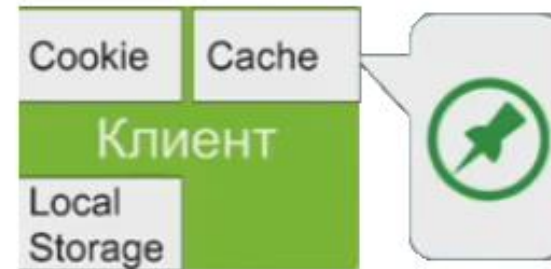
1



2

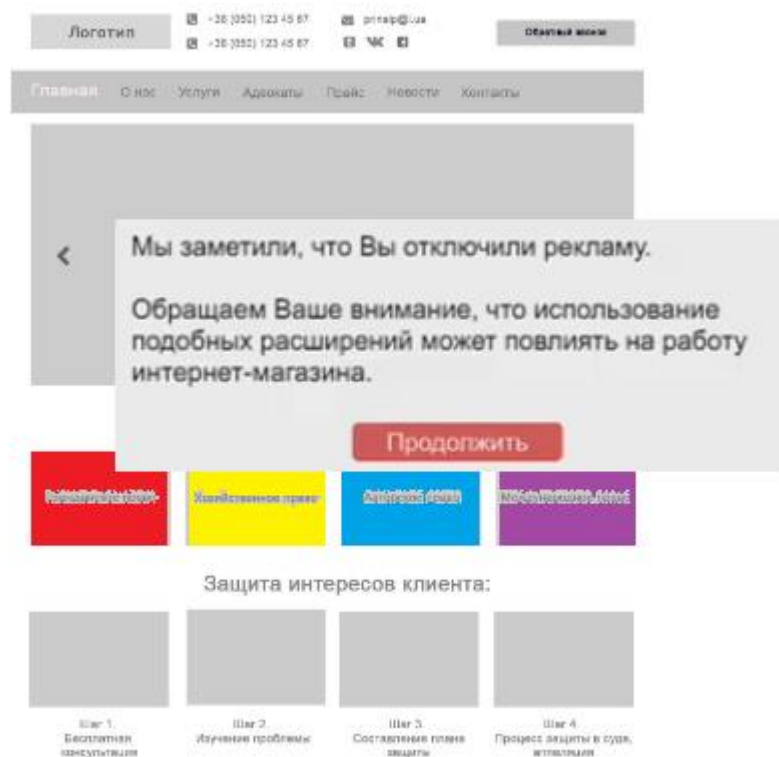


3



Что тестируем

Плагины могут влиять на работу приложения



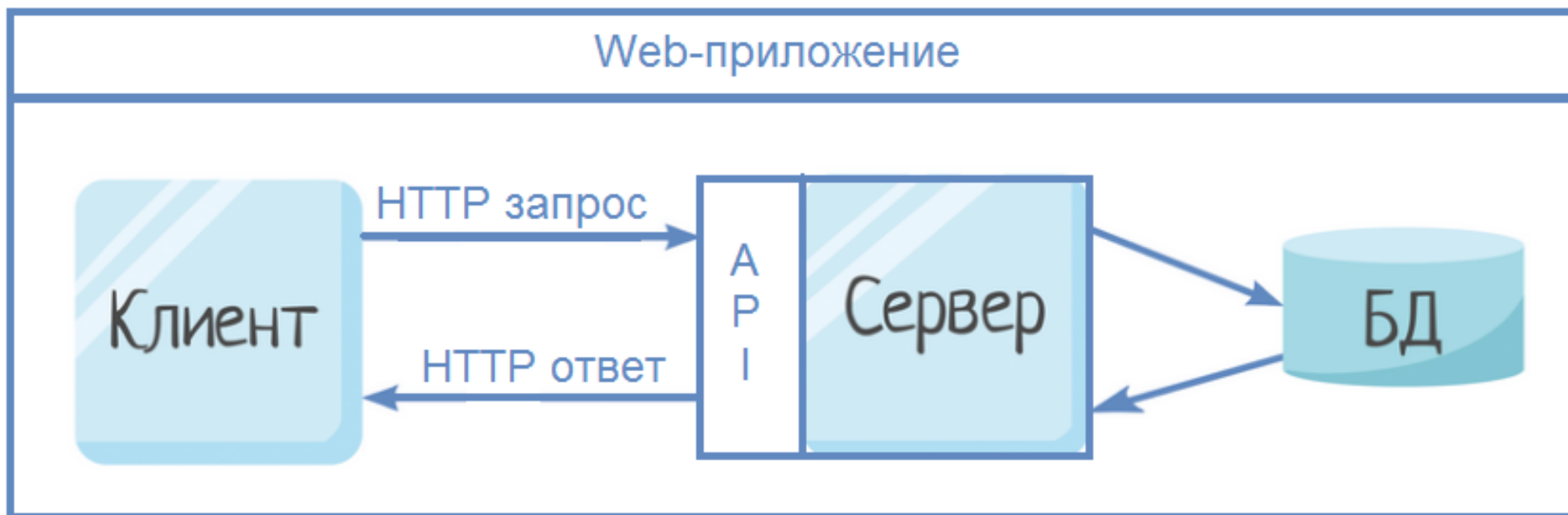
Тестирование API

Что такое API

Application programming interface (API) –
программный интерфейс приложения.

- Правила по которым общаются две системы;
- Он определяет что можно сделать;
- Как именно должно выглядеть сообщение.

Что такое API



Как устроен HTTP

Hyper Text Transfer Protocol (HTTP) –
протокол передачи гипертекста.

- Работает по схеме запрос-ответ;
- Определенная структура запроса;
- Определенная структура ответа.



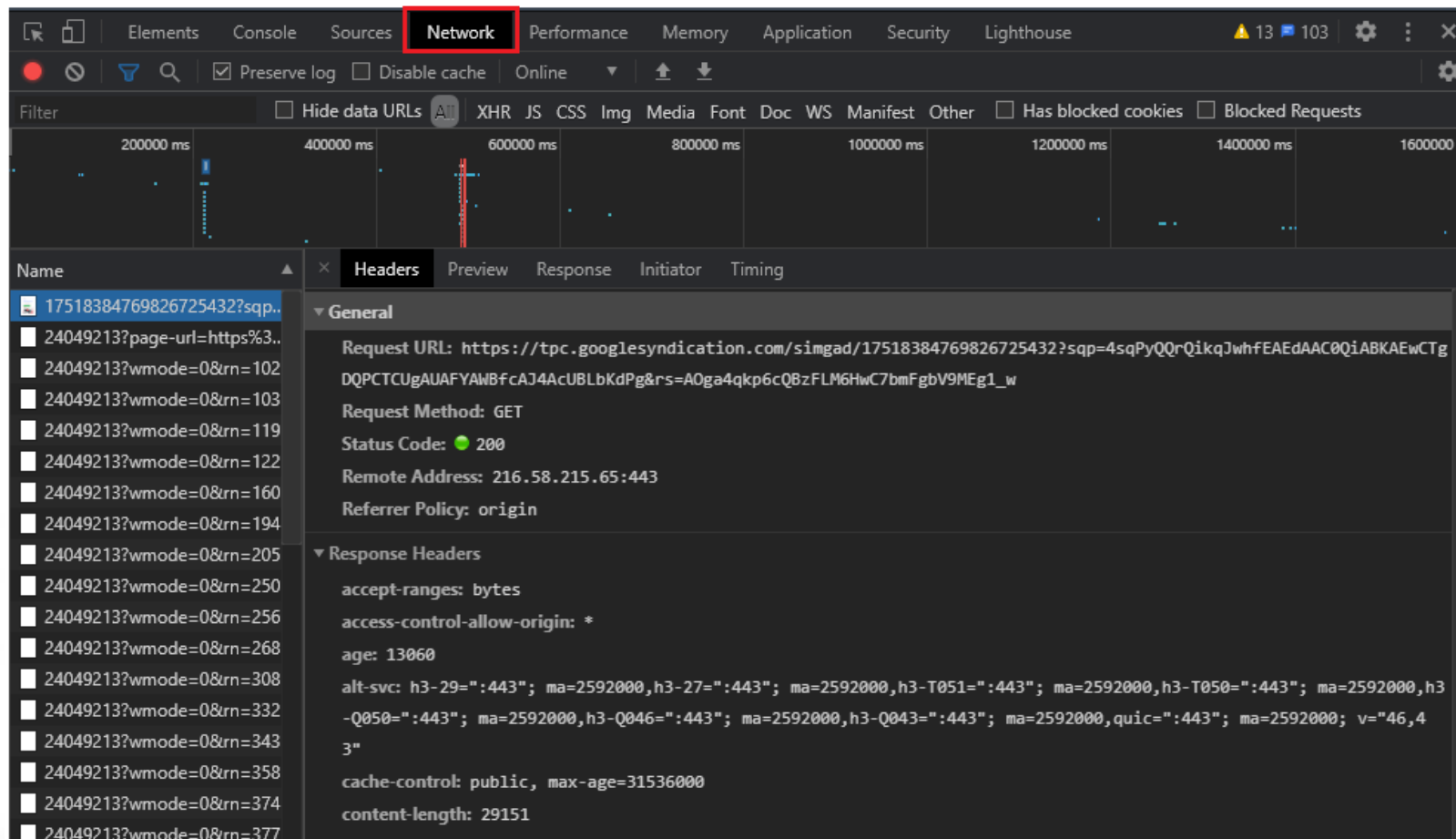
Как устроены запрос и ответ

HTTP	
Запрос (Request)	Ответ (Response)
Request URL <code>https://lms2.bsuir.by/my/</code>	
Request Method: GET	Status cod: 200 OK
Request Headers: Content-Type: text/html; charset=UTF-8	Response Headers: Content-Type: text/html; charset=UTF-8
Request Body: (empty)	Response Body: <!doctype html>

Как устроены запрос и ответ

HTTP Request/Response	
Request URL	С чем взаимодействовать
Request Method	Какое действие
Response Status Cod	Результат обработки
Body	Необходимая информация
Headers	Служебная информация

Инструменты разработчика

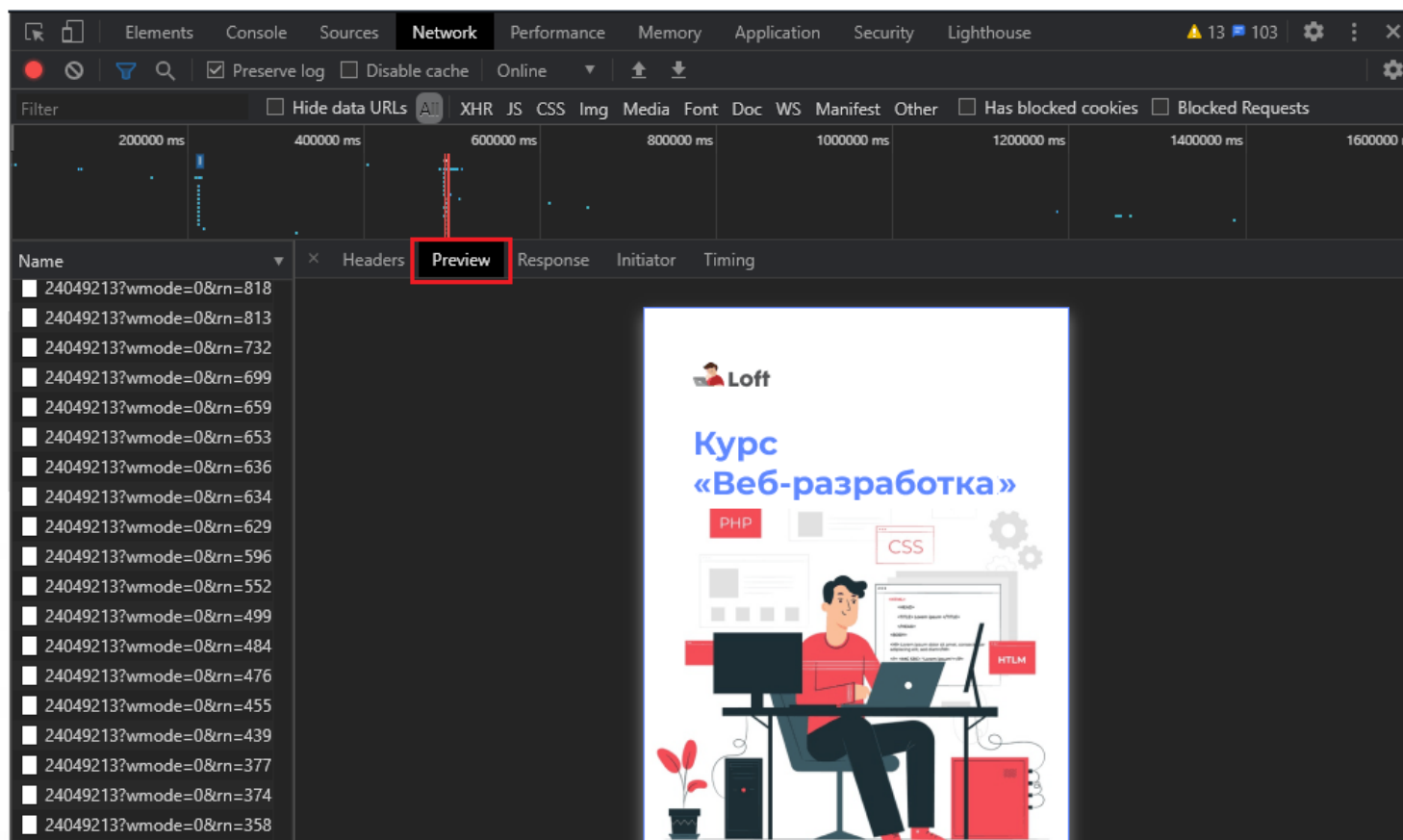


Инструменты разработчика

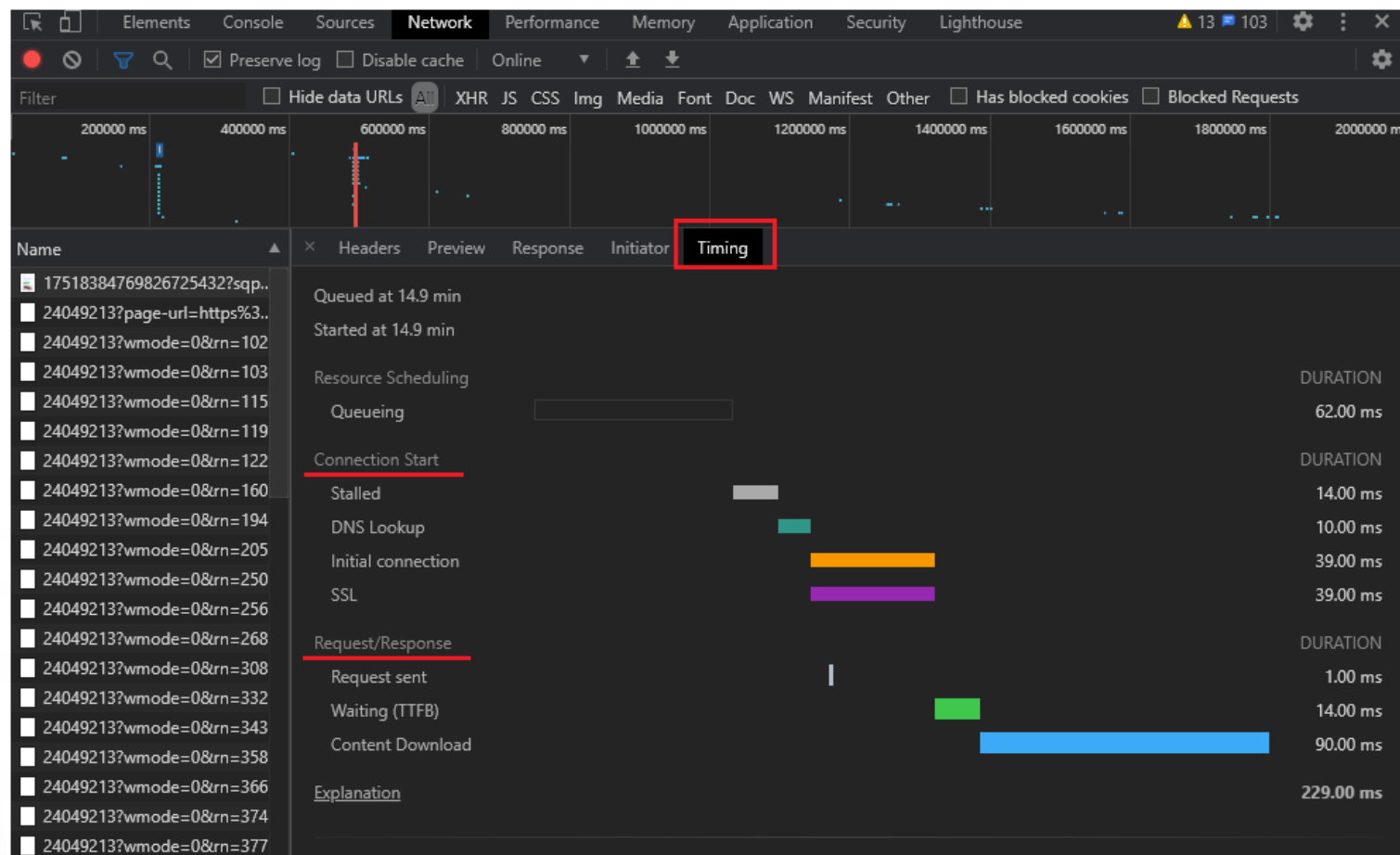
The screenshot shows the Chrome DevTools Network tab. The 'Headers' sub-tab is selected and highlighted with a red box. The left sidebar lists various resources, with '17518384769826725432?sqp..' selected. The main panel displays the following information:

- General**
 - Request URL: https://tpc.googlesyndication.com/simgad/17518384769826725432?sqp=4sqPyQQrQikqJwhfEAEdAAC0QiABKAewCTgDQPCTCUGAUAFYAWBfcAJ4AcUblbKdPg&rs=A0ga4qkp6cQBzFLM6HwC7bmFgbV9MEg1_w
 - Request Method: GET
 - Status Code: 200
 - Remote Address: 216.58.215.65:443
 - Referrer Policy: origin
- Response Headers**
 - accept-ranges: bytes
 - access-control-allow-origin: *
 - age: 13060
 - alt-svc: h3-29=":443"; ma=2592000, h3-27=":443"; ma=2592000, h3-T051=":443"; ma=2592000, h3-T050=":443"; ma=2592000, h3-Q050=":443"; ma=2592000, h3-Q046=":443"; ma=2592000, h3-Q043=":443"; ma=2592000, quic=":443"; ma=2592000; v="46,4"
- Request Headers**
 - :authority: tpc.googlesyndication.com
 - :method: GET
 - :path: /simgad/17518384769826725432?sqp=4sqPyQQrQikqJwhfEAEdAAC0QiABKAewCTgDQPCTCUGAUAFYAWBfcAJ4AcUblbKdPg&rs=A0ga4qkp6cQBzFLM6HwC7bmFgbV9MEg1_w

Инструменты разработчика



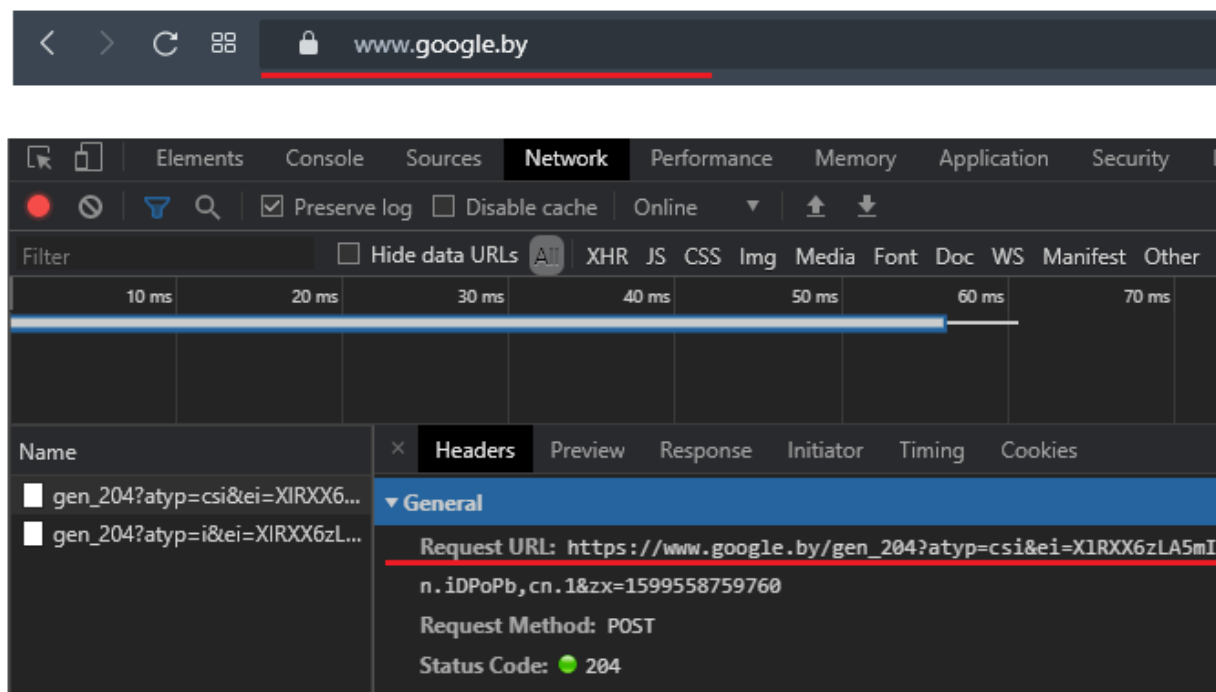
Инструменты разработчика



Структура запроса и ответа

Request URL - адрес запроса

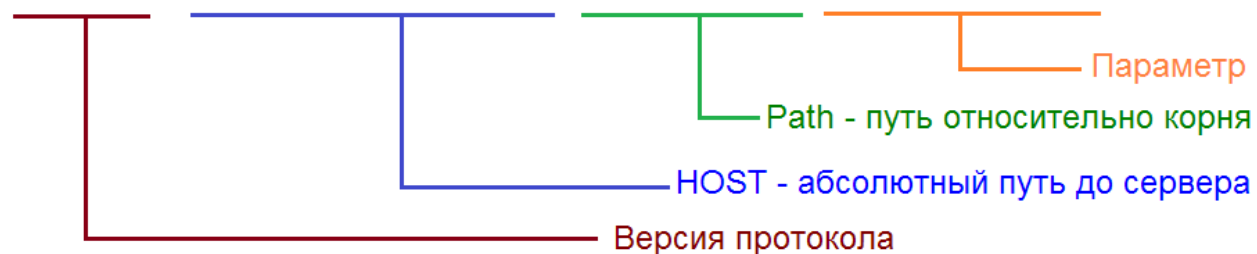
https://www.google.by/gen_204



Структура запроса и ответа

Request URL

https://www.google.by/gen_204?imgeld=44



Параметры после знака ?:

- Формат **КЛЮЧ = значение**;
- Ключ – уникальное название параметра;
- Параметров может быть несколько, между ними ставят **&**.

<https://lms2.bsuir.by/my?imgeld=9¶m17=6>

Структура запроса и ответа

Hyper Text Transfer Protocol Secure (HTTPS) –
безопасный протокол передачи гипертекста.

HTTP	
HTTP	HTTPS
Request Body:	Request Body:
{ “email”:“Petrov@mail.ru”. “password”: 1234567 }	D45lemm7823g8cw6gt98bcxtkbme[bm8498xltyby9874

Структура запроса и ответа

Status Code – результат обработки

Класс	Пример
1XX - Information	100 Continue
2XX - Success	200 OK 201 Created
3XX - Redirection	301 Moved
4XX – Client Error	400 Bad Request 401 Unauthorized 404 Not Found 418 I'm a teapot
5XX - Server Error	502 Bad Gateway

<https://developer.mozilla.org/ru/docs/Web/HTTP/Status>

Структура запроса и ответа

Позитивные и негативные тесты

Request	Response
GET/images?imageld=1 GET/images?imageld=10 GET/images?imageld=15	Status: 200 OK { "image": "id.jpeg" }
GET/images?imageld=1 GET/images?imageld=12 GET/images?imageld=18	Status: 404 Not Found { "message": "Image doest not exist" }

Структура запроса и ответа

Body – тело запроса

- HTML страница
- Файл
- Структурированная информация
-

Структура запроса и ответа

Структурированное тело сообщения.

На практике это формат JSON. Текстовый формат для обмена данными имеющий свою строго определенную структуру.

```
{
  "Persons": [
    {
      "name": "Petya Petrov",
      "gender": "male"
    },
    {
      "name": "Inna Ivanova",
      "gender": "female"
    }
  ]
}
```

Структура запроса и ответа

Важно!

Структура

- ключ:значение
- через запятую
- вложенность

Название параметров


Типы данных


- [] для списков
- "" для строк
- формат дат

Обязательные и не обязательные параметры

Структура запроса и ответа

 JSON Schema Validator

 Save

 newtonsoft.com

An online, interactive JSON Schema validator. Supports JSON Schema Draft 3, Draft 4, Draft 6, Draft 7 and Draft 2019-09.

Select schema: Custom

```
1 {
2   "title": "Person",
3   "type": "object",
4   "properties": {
5     "firstName": {
6       "type": "string",
7       "description": "The person's first name."
8     },
9     "lastName": {
10      "type": "string",
11      "description": "The person's last name."
12    },
13    "age": {
14      "description": "Age in years which must be
15      equal to or greater than zero.",
16      "type": "integer",
17      "minimum": 0
18    }
19  }
```

Input JSON:

```
1 {
2   "firstName": "John",
3   "lastName": "Doe",
4   "age": 0
5 }
```

✓ No errors found. JSON validates against the schema

<https://www.jsonschemavalidator.net>

Структура запроса и ответа

Headers – служебная информация

Ключ значения

Request Headers:

Host: Bsuir.by

Connections: keep-alive

User Agent: Mozilla/5.0(X11; Linux x86_64)

Response Headers:

Cache-Control: no store, no cache, must revalidate

Connections: keep-alive

Content Type: text/html; charset=UTF-8

Структура запроса и ответа

Request Method – методы запросов

GET	Получить
POST	Передать
PUT	Создать
PATCH	Изменить
DELETE	Удалить

Структура запроса и ответа

GET – **получить** ресурс, находящийся по указанному Request URL адресу

- Параметры передаются в URL

POST – **передать** по указанному Request URL адресу данные пользователя из тела запроса

- Параметры передаются в URL
- Данные содержатся в теле запроса
- Ответственное сообщение, как правило, имеет статус код 200 и содержит в теле итог выполнения запроса

Структура запроса и ответа

PUT – создать ресурс согласно параметрам из Request URL, используя данные из тела запроса

- Параметры передаются в URL
- Данные содержатся в теле запроса
- Ответственное сообщение, как правило, имеет статус код 201 Created и содержит в теле созданный ресурс

PATCH – изменить ресурс, находящийся по указанному в Request URL адресу, используя данные из тела запроса

- Параметры передаются в URL
- Данные содержатся в теле запроса
- Ответственное сообщение, как правило, имеет статус код 200 и содержит в теле ресурс после изменений

Структура запроса и ответа

Delete – **удалить** ресурс, находящийся по указанному в Request URL адресу

- Параметры передаются в URL
- Ответственное сообщение, как правило, имеет статус **код 204** и сопроводительное сообщение

Особенности тестирования API

Список Request URLs (end points) и их методов.

Для каждого запроса:

- что должен сделать сервер;
- какие есть параметры;
- какие параметры обязательные;
- что должны получить в ответ.

Для каждого параметра:

- какой формат данных;
- какие значения может принимать.

Ожидаемое сообщение об ошибках.

Особенности тестирования API

UI должен быть рабочим.
Локализация дефектов.
Создание тестовых данных.
Авторизация.


Инструменты

Swagger

- Посмотреть какие запросы есть.
- Посмотреть параметры запросов и типы данных.
- Посмотреть примеры запросов и ответов.
- Выполнить тестовый запрос.

Инструменты

https://editor.swagger.io

 **Swagger Editor**
Sponsored by SMARTBEAR

File

Edit

Generate Server

Generate Client

```
1 swagger: "2.0"
2 info:
3   description: "This is a sample server Petstore server.
   You can find out more about Swagger at [http
   ://swagger.io](http://swagger.io) or on [irc.freenode
   .net, #swagger](http://swagger.io/irc/). For this
   sample, you can use the api key `special-key` to test
   the authorization filters."
4   version: "1.0.0"
5   title: "Swagger Petstore"
6   termsOfService: "http://swagger.io/terms/"
7   contact:
8     email: "apiteam@swagger.io"
9   license:
10     name: "Apache 2.0"
11     url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12 host: "petstore.swagger.io"
13 basePath: "/v2"
14 tags:
15 - name: "pet"
16   description: "Everything about your Pets"
17   externalDocs:
18     description: "Find out more"
19     url: "http://swagger.io"
20 - name: "store"
21   description: "Access to Petstore orders"
22 - name: "user"
23   description: "Operations about user"
24   externalDocs:
25     description: "Find out more about our store"
26     url: "http://swagger.io"
27 schemes:
28 - "https"
29 - "http"
30 paths:
31 /pet:
32   post:
33     tags:
34     - "pet"
35     summary: "Add a new pet to the store"
36     description: ""
37     operationId: "addPet"
38     consumes:
```

Swagger Petstore 1.0.0

[Base URL: petstore.swagger.io/v2]

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [#swagger](http://irc.freenode.net). For this sample, you can use the api key `special-key` to test the authorization filters.

[Terms of service](#)

[Contact the developer](#)

[Apache 2.0](#)

[Find out more about Swagger](#)

Schemes

HTTPS

Authorize

pet Everything about your Pets Find out more

POST /pet Add a new pet to the store

PUT /pet Update an existing pet

GET /pet /findByStatus Finds Pets by status

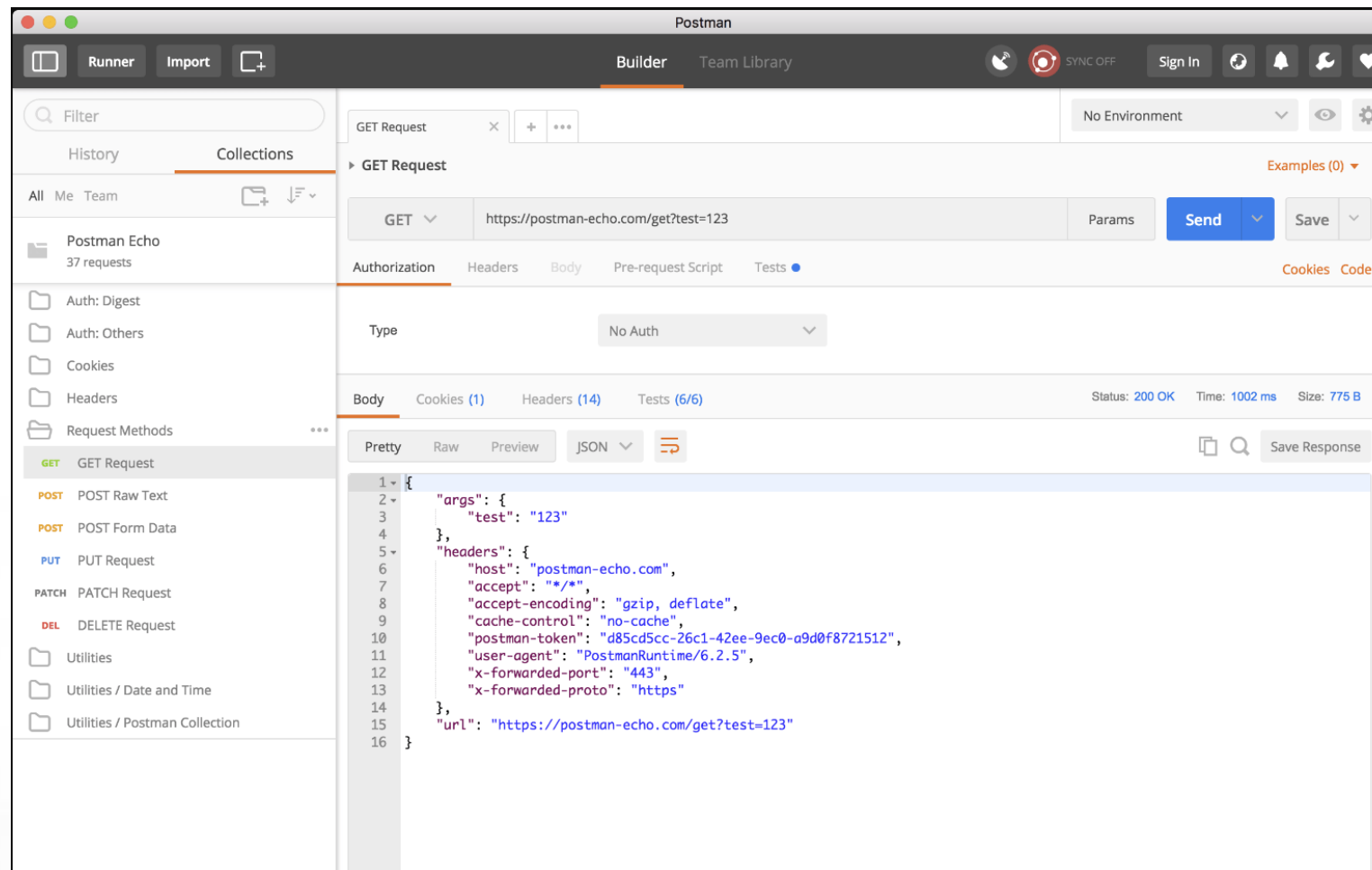
Инструменты

Postman

- Посмотреть какие запросы есть.
- Посмотреть параметры запросов и типы данных.
- Посмотреть примеры запросов и ответов.
- Выполнить тестовый запрос.

Инструменты

<https://www.postman.com>

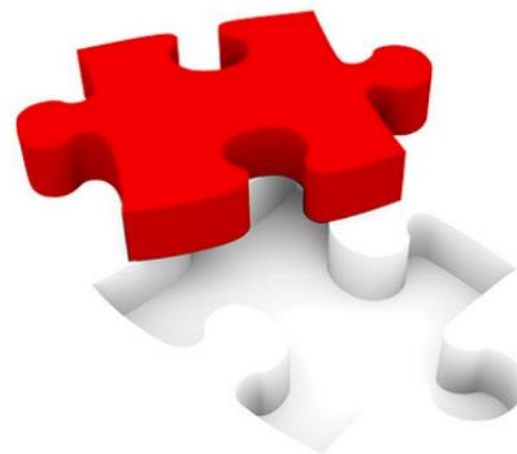


Тестирование совместимости

Тестирование совместимости

Тестирование совместимости (compatibility testing) - проверка того, как приложение взаимодействует с другими приложениями и операционной системой.

В случае веб-ориентированных приложений особое внимание уделяется совместимости с различными браузерами.



Кроссбраузерное тестирование

Кроссбраузерное тестирование - проверка способности веб-ориентированного приложения идентично работать и отображаться во всех заявленных поддерживаемых браузерах (обычно учитываются лишь наиболее распространённые).

Следует обратить внимание на тот факт, что кроссбраузерная совместимость не обязательно означает «попиксельное соответствие» внешнего вида приложения в разных браузерах.

Как обеспечить кроссбраузерность

Наиболее распространённый (но не самый лучший) способ - написание **специального CSS/JS-кода**, который будет выполняться только в определённых версиях определённых браузеров.

Иногда генерацию соответствующего кода перекладывают на серверную часть приложения.

Как обеспечить кроссбраузерность

Поскольку браузеров **ОЧЕНЬ** много, писать свою логику поведения и отображения приложения для каждого из них - нерационально.

Поэтому **рекомендуется использовать** такие элементы вёрстки, которые одинаково **хорошо совместимы** с большинством версий наиболее распространённых браузеров.



Как проверить кроссбраузерность

Научный подход: поставить мощный сервер, на котором в виртуальных машинах под разными операционными системами выполняются разные браузеры. Подключаться удалённо к этим виртуальным машинам и открывать сайт.

Упрощённый подход: просто подготовить много образов виртуальных машин, запускать их на своём компьютере по мере надобности.

Самый простой подход: использовать готовые сервисы.

Что проверяем

При тестировании кроссбраузерной совместимости нужно проверить:

- как приложение отображается и работает в **различных браузерах** (семейство Mozilla, Internet Explorer, Opera, Safari, Chrome, мобильные браузеры);
- как приложение отображается и работает **при различных разрешениях экрана** (обычно 1024x768, 1280x800 и т.д.)
- как приложение отображается и работает в **различных операционных системах** (Mac OS, Linux, Win).

Он-лайн сервисы

В тестировании кроссбраузерности нам помогут он-лайн сервисы и оффлайновые утилиты.

IE NetRenderer - показывает (генерирует изображения), как сайт будет выглядеть под различными версиями MSIE:

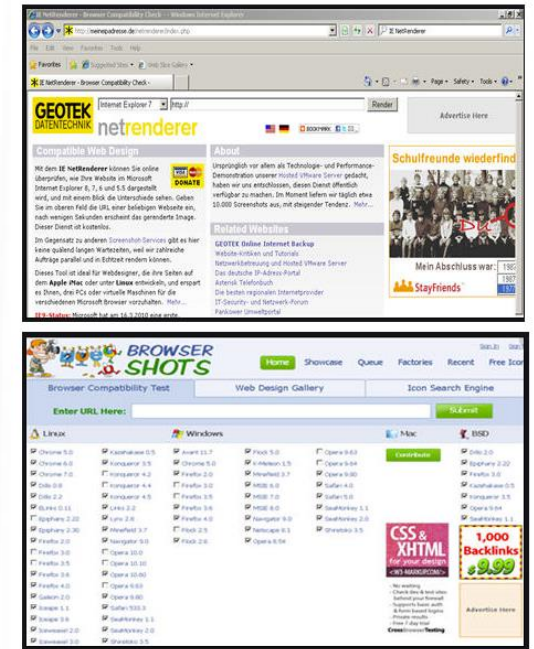
<http://ipinfo.info/netrenderer/>

Browsershots.org - пожалуй, самый известный сервис кроссбраузерного тестирования.

Поддерживает десятки версий самых разнообразных браузеров под самыми разнообразными операционными системами.

В очень удобном виде генерирует изображения:

<http://browsershots.org>



Он-лайн сервисы

В дополнение - список подобных сервисов:

Browser Sandbox (<http://spoon.net/Browsers/>) - позволяет открывать станицы в разных браузерах, таких как IE, Firefox, Safari, Chrome и Opera непосредственно из своего браузера.

Browsrcamp (<http://www.browsrcamp.com/>) - позволяет проверить совместимость приложения с браузерами в Mac OS X.

Adobe BrowserLab (<https://browserlab.adobe.com/en-us/index.html>) - позволяет проверять отображение сайтов в разных браузерах и на разных ОС.

Browsera (<http://www.browsera.com/>) - платный онлайн сервис предназначенный для тестирования сайта в целом, а не только для проверки кроссбраузерности сайта.

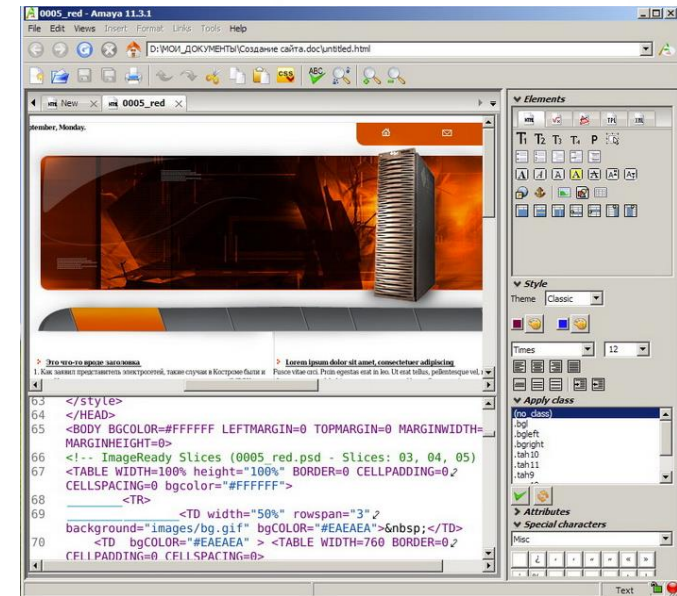
Litmusapp (<http://litmusapp.com/>) - платный сервис с большим количеством разнообразных функций, как мелких и довольно специфичных, так и крупных и очень полезных (в т.ч. возможность тестирования сайта на мобильных устройствах с Windows Mobile, Symbian или iPhone).

Он-лайн сервисы

Amaya - браузер от W3C. Позволяет хорошо изучить клиентскую часть приложения на предмет того, что в ней есть, и как это работает.

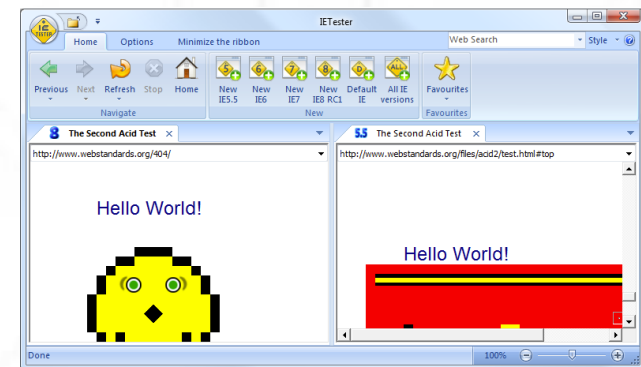
Бесплатно загрузить последнюю версию можно здесь:

<http://www.w3.org/Amaya/>



IETester - бесплатный веб-браузер, который содержит в себе несколько версий Internet Explorer: IE9, IE8, IE7, IE 6 и IE5.5 для Windows 7, Vista и XP.

<http://www.my-debugbar.com/wiki/IETester/HomePage>



О чем стоит помнить

Различные браузеры и операционные системы используют различные способы вывода шрифтов.

Размеры шрифтов тоже не одинаковы в различных системах, а некоторые шрифты могут просто отсутствовать на компьютере пользователя.

В различных браузерах могут быть свои особенности использования JavaScript.

О чем стоит помнить

Помимо вёрстки стоит помнить о:

- **Cookies** - приложение должно быть полностью работоспособным, если они у пользователя выключены.
- **JavaScript** - если он выключен, приложение должно продолжать работать.
- Без крайней необходимости приложение не должно обращаться куда бы то ни было по порту, отличному от 80-го.
- Приложение должно иметь возможность работать БЕЗ установки в браузер дополнительных плагинов.

ПОДВОДИМ ИТОГ

Итак, что может ухудшить совместимость веб-ориентированного приложения с клиентским ПО:

- использование устаревших технологий;
- использование самых новых технологий, ещё не нашедших повсеместной поддержки;
- нарушение стандартов;
- отсутствие учёта особенностей конкретных видов и версий клиентского ПО;
- применение технологий, требующих установки дополнительных компонентов в клиентское ПО;
- использование нетипичного для веб-ориентированных приложений сетевого взаимодействия;
- отсутствие учёта предпочтений пользователей по настройкам того или иного клиентского ПО;
- создание плохо поддерживаемого кода HTML/CSS/JS и серверной части, что приводит к сложнопредсказуемым последствиям при внесении изменений;
- недостаточное внимание вопросам тестирования совместимости.

«Тестирование удобства использования (юзабилити)»

Принципы юзабилити веб-ориентированных приложений

Научно юзабилити (*) трактуется как:

- мера пользовательской реакции и мера пользовательского опыта.

Говоря простым языком, это:

- насколько пользователю нравится приложение и
- насколько быстро пользователь обучается работе с приложением.

* Usability - «возможность быть использованным», «удобство использования».

Юзабилити в веб

Применительно к веб-сайтам юзабилити - это **удобство пользования сайтом**, достигаемое применением концепции построения веб-интерфейсов сайта, направленной на **достижение основной цели (или целей)**, поставленной перед сайтом.

Примеры целей:

- заказ;
- покупка товара;
- сообщение о сайте знакомым.

Принципы юзабилити

Правило одной секунды

Ещё 2-3 года назад это правило называлось «правилом секунд», но сейчас время изменилось.

Если пользователь ждёт реакции на свои действия более секунды - он это замечает и начинает раздражаться.

Если пользователь ждёт более двух секунд - он злится.

Ожидание более пяти секунд большинством пользователей воспринимается как «подвисание» приложения.



Принципы юзабилити

Правило трёх кликов

Пользователь хочет, чтобы навигация была **простой, как кирпич**.

Число «три» в названии правила выбрано как «наиболее показательное», и не является догмой, **однако если для выполнения простой задачи пользователю приходится делать 5-7 и более действий - это плохо. Что-то нужно менять.**

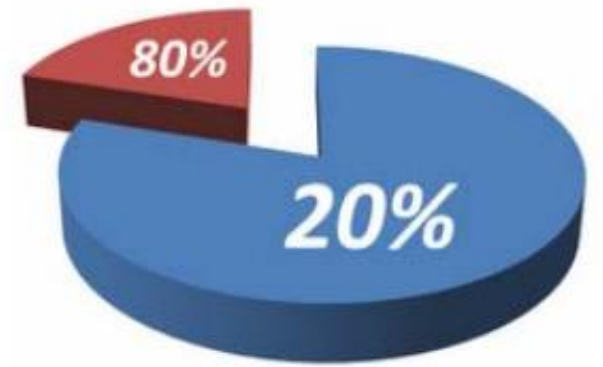


Принципы юзабилити

Правило 80/20

Это правило также известно как «**принцип Парето**» {«20% действий приносят 80% результата; остальные 80% действий дают всего лишь 20% результата»).

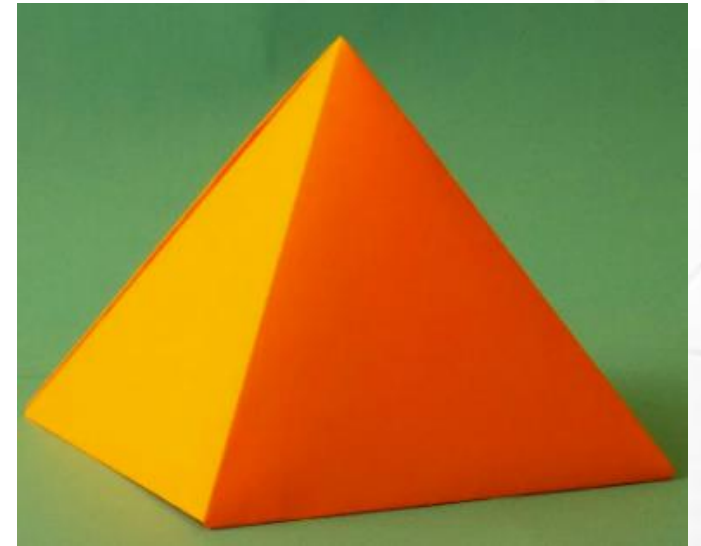
В случае юзабилити веб-ориентированных приложений это правило предписывает.
определить те «20%» аудитории сайта, которые дадут «80%» результата (достижения цели сайта), и оптимизировать юзабилити для нужд этих «20%»



Принципы юзабилити

Правило пирамиды

Суть этого правила состоит в том, чтобы помещать в самом начале информационного блока итоговый вывод, а затем расширять тему.



Принципы юзабилити

Принцип удовлетворённости

Пользователю далеко не всегда нужно самое лучшее, их больше интересует «то, что их устроит».



Принципы юзабилити

Синдром утёнка (*)

Огромное значение для пользователя имеет его первый опыт работы с некоторым классом приложений.

Впоследствии всё новое пользователь сравнивает с тем первым. Сравнение, как правило, оказывается в пользу первого опыта.



Принципы юзабилити

Баннерная слепота

Опытные пользователи («интернетчики со стажем») инстинктивно игнорируют всё, что так или иначе напоминает банеры или иную рекламу.

Поэтому если вы решили разместить что-то полезное, но оформить это «в рекламном стиле», оно многими останется незамеченным.



Принципы юзабилити

Принципы восприятия форм

Закон близости утверждает, что когда мы видим набор объектов, объекты, расположенные ближе друг к другу, мы распознаем как группу.

Закон сходства утверждает, что сходные объекты человек подсознательно группирует.

Закон содержательности утверждает, что один и тот же объект может играть важную роль в одном визуальном поле и быть частью фона в другом.



Mac

Принципы юзабилити

Правила построения интерфейса

1. Интерфейс должен быть **логичным**, не вводящим в заблуждение.
2. Для опытных пользователей должен существовать **быстрый способ выполнения операций** (горячие клавиши, макросы).
3. Обработка ошибок должна быть **простой и информативной**.
4. **ВСЕГДА** должен существовать **ПРОСТОЙ способ отмены действия**.
5. Пользователь должен чувствовать, что **всё под его контролем**.
6. Как можно **меньше загружайте кратковременную память** пользователя.



Полезные советы

Надписи полей форм лучше размещать над ними, а не слева от них.

Первый вариант - ужасно.

Второй вариант - дискомфортно, хоть и достаточно примелькалось.

Третий вариант - хорошо.



Register form

Username:

E-mail:

Password:

Sign Up

This form illustrates a poor design where labels are placed to the left of the input fields. The labels 'Username:', 'E-mail:', and 'Password:' are aligned with the start of their respective text boxes, which is visually awkward and less intuitive for users.



Register form

Username:

E-mail:

Password:

Sign Up

This form is identical to the first one, showing a common but poor practice of left-aligned labels. It serves as a baseline for comparison with the better design shown in the third example.



Register form

Username:

E-mail:

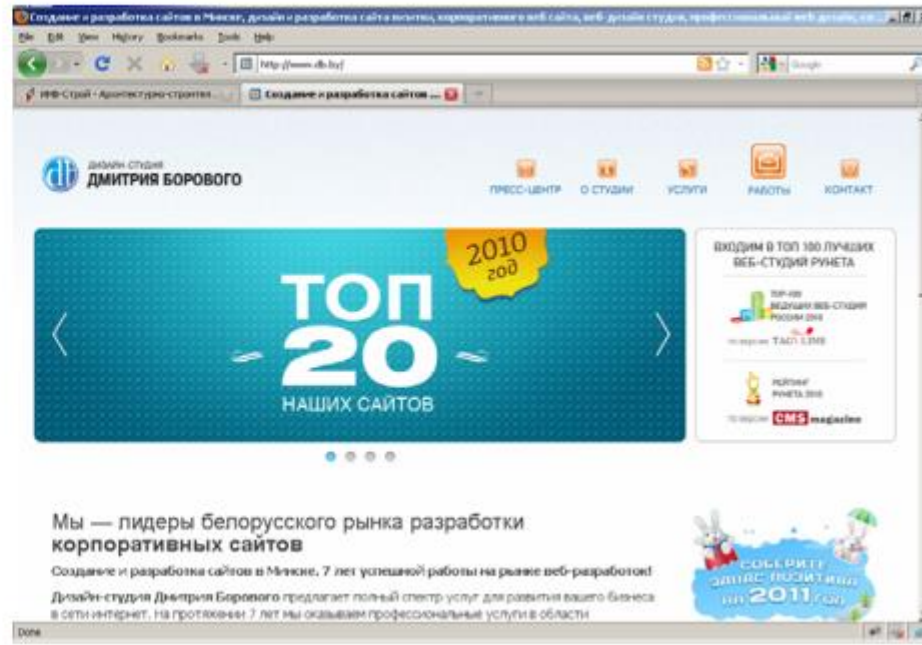
Password:

Sign Up

This form demonstrates a better design where labels are placed directly above each input field. This layout is cleaner, more professional, and easier for users to understand, as the labels are clearly associated with the fields they describe.

Полезные советы

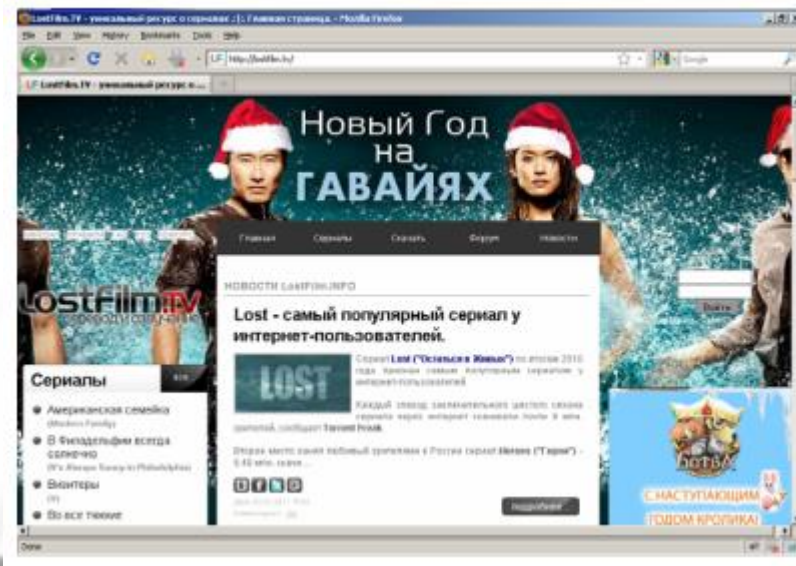
Поскольку многие пользователи сразу открывают несколько сайтов в результатах поиска и бегло просматривают страницы, лучше размещать полезную с точки зрения сайта информацию так, чтобы пользователю не пришлось прокручивать страницу ВНИЗ.



Полезные советы

Пользователи концентрируют своё внимание на лицах людей. Потому дизайн, в котором присутствуют фотографии, выигрывает у дизайна, в котором фотографий нет.

При этом, если «взгляд» людей на фотографии «направлен на некоторую часть страницы», пользователи сами склонны посмотреть туда. Так можно привлечь внимание к важным элементам страницы.



Способы тестирования юзабилити

Карточная сортировка

Это **классификационный метод**, при котором пользователи сортируют различные элементы по нескольким категориям.

Для проведения карточной сортировки создаётся список параметров, каждый из которых выписывается на отдельной карточке.

Карточки показывают пользователям и просят сгруппировать наиболее логичным образом.

Полученную в результате информацию используют для организации пользовательского интерфейса.

Так, например, можно группировать пункты подменю по пунктам меню, информационные блоки по блокам страницы, страницы по разделам и т.п.

Оценочные листы

Оценочные листы помогают удостовериться в том, что продукт выполнен с учётом принципов юзабилити.

При создании оценочного листа следует помнить, что это - **НЕ** анкета. Т.е. вопросы в оценочном листе должны быть предельно короткими, ясными, сгруппированными по категориям и допускать только однозначные ответы вида «ДА» или «НЕТ» (*).

* Иногда в оценочных листах допускаются «ответы с ранжированием», но куда чаще это делается в классических анкетах.

Анкетирование

В то время как оценочные листы применяются на поздних стадиях для контроля качества, анкетирование даёт наибольшую отдачу на стадии прототипирования или сбора требований.

Анкетирование позволяет собрать много сведений о предпочтениях пользователей.

Главное требование к анкетам - их верное составление: вопросы должны быть понятны пользователям, а варианты ответов должны позволять автоматизированную обработку данных.

Прототипирование

Прототипирование в контексте юзабилити в основном сводится к построению макетов (прототипов) дизайна интерфейса.

Прототип демонстрируется заказчику и/или целевой аудитории, после чего на него собираются отзывы. Наибольший эффект даёт сбор отзывов в свободном виде, дополненный оценочным листом.



Фокусные группы

Данный метод заключается в подробном опросе специально отобранной группы пользователей, представляющих собой фрагмент целевой аудитории приложения.

Это довольно затратный способ, но он позволяет не только отследить реакцию отдельных пользователей, но и изменение реакции в процессе обмена мнениями.

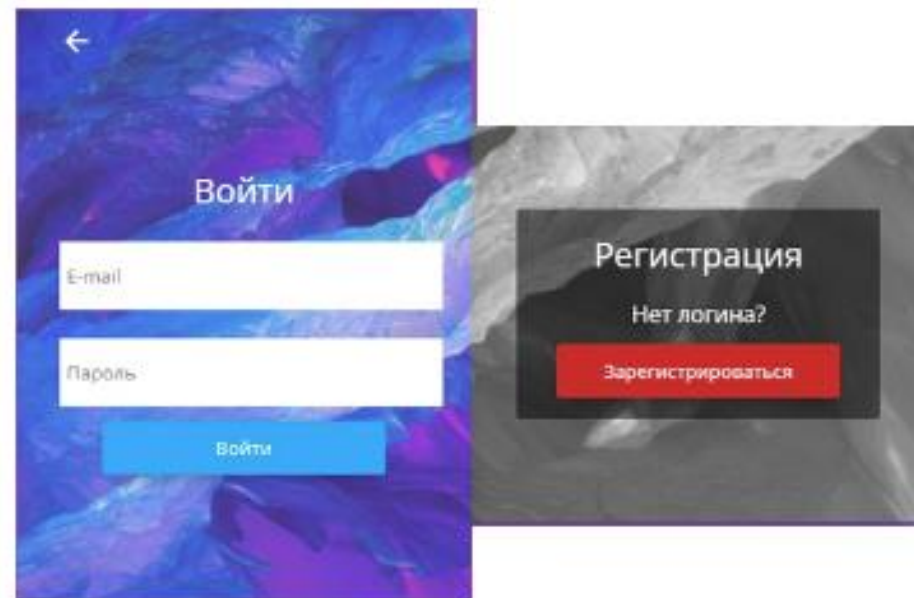
Не следует делать фокусные группы чрезмерно большими. Достаточно 6-9 человек.

Тестирование форм

Формы

Формы – неотъемлемый атрибут web-ориентированного приложения, подразумевающего взаимодействие с человеком.

От качества реализации механизма работы с формами зависит множество аспектов качества web-приложения, от юзабилити до производительности и безопасности.



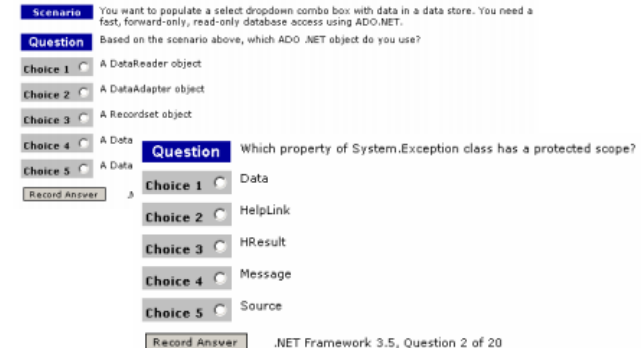
Виды форм

Формы в общем случае делятся на два основных вида:

- **однооконные;**
- **пошаговые формы.**



A screenshot of the WordPress login form. It features the 'WordPress' logo at the top. Below it are two input fields: 'Login:' and 'Password:'. To the right of the password field is a 'Login »' button. At the bottom, there are two links: « Back to blog and Lost your password?.



A screenshot of a scenario-based question form. It starts with a 'Scenario' section describing a task to populate a dropdown menu. This is followed by a 'Question' section asking for the correct ADO.NET object. Below the question are five radio button choices: 'Choice 1' (A DataReader object), 'Choice 2' (A DataAdapter object), 'Choice 3' (A Recordset object), 'Choice 4' (A Data), and 'Choice 5' (A Data). A 'Record Answer' button is at the bottom left. A second 'Question' section asks for the protected scope property of System.Exception, with five radio button choices: 'Choice 1' (Data), 'Choice 2' (HelpLink), 'Choice 3' (HResult), 'Choice 4' (Message), and 'Choice 5' (Source). A 'Record Answer' button is at the bottom left. The footer indicates '.NET Framework 3.5, Question 2 of 20'.

Что проверяем

- 1. Путь (URL).**
- 2. Месторасположение.**
- 3. Ошибочные ситуации.**
- 4. Поля, их значения.**
- 5. JavaScript в формах.**

Пошаговые формы

Основная беда пошаговых форм - неочевидность их функционирования при возврате на предыдущие шаги или восстановлении работы после обрыва связи.

Пункты 1-5 в полной мере относятся к каждому отдельному шагу пошаговой формы.



Переходим к тестированию (Чек-лист)

Перед вами - краткий, «максимально универсализированный» чек-лист:

1. **Расположение формы** на экране перед заполнением, после неверного заполнения, после верного заполнения.
2. **Сохранение/изменение URL** при отправке данных из формы.
3. Отправка формы с **незаполненными обязательными полями**.
4. Отправка формы с **неверно заполненными полями** (числа вне диапазонов, строки превышают допустимую длину и т.п.)
5. Отправка формы с «**нелогичными данными**» (например, дата рождения - в будущем).

Переходим к тестированию (Чек-лист)

6. Отправка формы с полями, содержащими **СПЕЦСИМВОЛЫ**.
7. **Восстановление значений полей** после отправки формы с неверно заполненными полями.
8. Информативность **сообщений об ошибках**.
9. **Функциональная сгруппированность полей** формы.
Информативность надписей, подсказок.
10. **Работоспособность и юзабилити** формы с отключённым JavaScript.
11. **Использовать чек-листы по каждому** стандартному полю (текстовое, числовое, дата и т.п.)

Поля форм, стандартные тестовые случаи

Передача данных

Сейчас мы рассмотрим стандартные тестовые случаи для типичных полей форм.

Прежде всего - проверяем, **верным ли методом** отправляются **данные** из формы (атрибут «**method**» тега «**form**»).

Белый ящик:

`<form... method="post" ...>`

или

`<form... method="get" ...>`

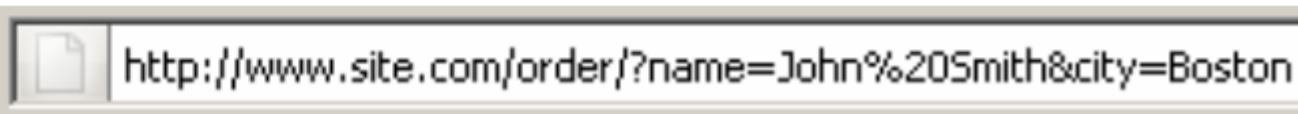
Чёрный ящик:

отправьте данные и
посмотрите на адресную
строку браузера.

Post – никаких «неожиданных
данных»:



Get – вы видите имена и
значения полей:



Текстовые поля

Чаще всего используются три вида текстовых полей:

1) Однострочное поле:
`<input type="text" ...>`

A single-line text input field containing the text "Text Text Text".

2) Пароль:
`<input type="password" ...>`

A password input field represented by a series of black dots.

3) Многострочное поле:
`<textarea ...></textarea>`

A multi-line text area containing three lines of text: "Text1", "Text2", and "Text3". It includes vertical scrollbars on the right side.

Текстовые поля

Для всех текстовых полей стоит проверить:

1. Значение **по умолчанию**.
2. Заполнение поля **корректными данными**.
3. Оставить поле **незаполненным (пустым)**.
4. Заполнить поле **максимально разрешённым количеством символов**.
5. Заполнить поле **максимально разрешённым количеством символов + 1**.
6. Заполнить поле такими **спецсимволами** как “1 < > \; @ и т.п.
7. Заполнить поле **только пробелами**.
8. Вставить **пробелы перед / в середине / в конце текста**.
9. Ввести **цифры**.
10. Ввести **данные на разных языках (в т.ч. одновременно)**.
11. Использовать **разные кодировки (cp1251, utf8 и т.п.)**.
12. Проверить, корректно ли **восстановлено значение поля** после повторной загрузки страницы (кроме полей с паролями и CAPTCHA).

Текстовые поля

Для полей с паролями стоит проверить:

1. Очищается ли значение поля после перезагрузки страницы.
2. Есть ли предупреждение о том, что использованный пароль слишком прост.
3. Есть ли предупреждение о том, что использованный пароль совпадает с логином или некоторой персональной информацией (телефон и т.п.).

Текстовые поля

Для многострочных полей стоит проверить:

1. Сколько данных мы можем вставить в поле из буфера. Если поле «хитрое» и умеет принимать не только текст - проверим, данные какого типа оно принимает.
2. Применяется ли конвертация концов строк (т.е. $\backslash n \rightarrow
$).
Корректно ли она работает. Нужна ли она здесь.

Текстовые поля

Для чек-боксов следует проверить:

1. Состояние по умолчанию.
2. Остаётся ли чек-бокс (не)выбранным после повторной загрузки формы.
3. Изменяется ли состояние чек-бокса при клике по ассоциированной с ним надписи.
4. «Выбрать все», «Снять отметку со всех», «Инвертировать выбор» (если присутствует).
5. Работоспособность взаимосвязи между чек-боксами (при её наличии).

Текстовые поля

Для радио-кнопок следует выполнить следующие проверки:

1. Состояние по умолчанию.
2. Сохранение состояния после повторной загрузки формы.
3. Изменяется ли состояние группы радио-кнопок при клике по ассоциированной с некоторой опцией надписи.
4. Если на странице присутствует несколько групп радио-кнопок - действительно ли все они независимы.
5. Можно ли отправить форму, не выбирая ни одну из опций, представленных радио-кнопками?

Red: ☐
Green: ☐
Blue: ☐

Текстовые поля

Для любых кнопок на форме следует проверить:

1. Понятно ли, что это за кнопка.
2. Состояние по умолчанию (доступна / недоступна).
3. Корректность текста (картинки) на кнопке.
4. Можно ли кликнуть по кнопке дважды, и какова реакция приложения на такое действие.



Текстовые поля

Для поля отправки файлов следует проверить:

1. Есть ли указание на то, **какие файлы (формат, размер) можно отправлять**.
2. Состояние **по умолчанию** (поле доступно/недоступно).
3. Наличие **фильтра по расширению**.
4. Какова **реакция приложения на отставку**: несуществующего файла, пустого файла, файла неверного формата, файла с неверным расширением, повреждённого файла, слишком большого файла.
5. Если при отставке файла появляется **полоса прогресса**, корректно ли она работает.



The image shows a user interface element for file selection. It consists of a rectangular text input field on the left and a button labeled "Browse..." on the right. The button has a light gray background and a thin border. The entire element is set against a background of faint, overlapping architectural lines.

Текстовые поля

Для скрытых полей следует проверить:

1. Значение по умолчанию.
2. Как значение поля меняется в ответ на действия пользователя с формой.
3. Не содержит ли поле информации, просмотр которой пользователем нежелателен.
4. Как можно навредить приложению, используя специально созданное вредоносное значение в этом поле.

Текстовые поля

Существует много разновидностей списков: выпадающие, поле ввода + выпадающий список, список значений с полосой прокрутки, список с возможностью выбрать несколько значений и т.п. Поэтому без хороших требований тестировать списки очень сложно.

Для списков следует проверить:

1. Значение по умолчанию.
2. Сохраняет ли список своё состояние после повторной загрузки формы.
3. Как работает возможность выбрать несколько вариантов.
4. Как форма реагирует на изменение значения списка (если такая реакция предусмотрена).
5. Корректность «заголовков группировки полей» (если они должны быть).

Благодарю за внимание