Тестирование ПО. Уровень 1.

Тестирование пользовательского интерфейса

Принципы тестирования графического интерфейса пользователя. Этапы тестирования. Виды интерфейса.

Оглавление

[GUI и GUI testing](#_88glw5ola6id)

[Frontend](#_2et92p0)

[Браузеры](#_s4wfyq9nph9f)

[Отображение HTML-страницы, или её рендеринг](#_2zqb1c4454pc)

[Этапы тестирования веб-приложений](#_9rwdz9ih6yge)

[Пример разбора тестирования одной из форм графического интерфейса пользователя](#_4x93v7ekwh68)

[Практическое задание](#_clx8y3gby0rl)

[Требования к выполненной работе](#_inqahd9gtupg)

[Глоссарий](#_17dp8vu)

[Дополнительные материалы](#_3rdcrjn)

[Используемые источники](#_lnxbz9)

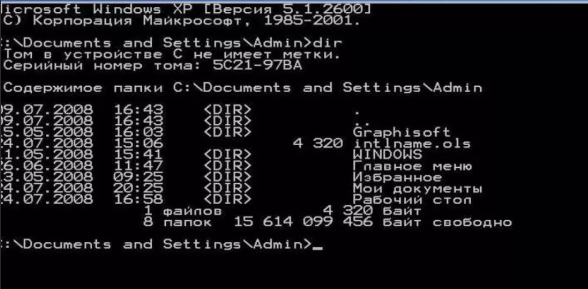
# GUI и GUI testing

Есть два вида тестирования пользовательского интерфейса, с которым может взаимодействовать пользователь:

* Интерфейс командной строки (command line interface).
* Графический интерфейс пользователя (graphical user interface).

Примерами командной строки могут быть:

* консоль командной строки ОС, например Windows;
* консоль командной строки сервера базы данных, например MySQL.



Примерами графического интерфейса могут служить все типовые экранные формы, посредством которых пользователь взаимодействует с программными приложениями. Наглядный пример — окна операционной системы Windows.



В лекции мы будем говорить про графический интерфейс пользователя и его тестирование.

Графический интерфейс пользователя (graphical user interface) — тип интерфейса, который позволяет пользователям взаимодействовать с компонентом или системой посредством графических иконок и других графических элементов интерфейса.

Тестирование графического интерфейса пользователя (GUI testing) — тестирование, выполняемое путём взаимодействия с тестируемым приложением посредством графического интерфейса пользователя.

Поскольку существует ещё программный интерфейс приложения (application programming interface или API), в ISTQB выделяют такой вид тестирования, как тестирование интерфейсов, который включает и GUI testing и API testing.

Тестирование интерфейса (interface testing) — вид интеграционного тестирования, выполняемый для проверки интеграции между компонентами или системами.

# Frontend

Выделяют два основных направления разработки веб-приложений.

Это разработка frontend-части (уровень представления, или presentation layer) и разработка backend-части (или уровень доступа к данным, по-другому, data access layer). В данном случае frontend подразумевает разработку той части приложения, с которой будет непосредственно взаимодействовать пользователь. Разработка backend-части подразумевает разработку серверной части, которая запускается на выделенном сервере, то есть создание бизнес-логики приложения.

Как правило, когда говорят про фронтенд, выделяют три основных компонента:

* **HTML (HyperText Markup Language)** — язык разметки документов, при помощи которого формируется структура страницы: заголовки, абзацы, списки и так далее.
* **CSS (Cascading Style Sheets)** — язык для описания и стилизации внешнего вида документа. Благодаря CSS-коду браузер понимает, как именно отображать элементы. CSS задаёт цвета и параметры шрифтов, определяет, как будут располагаться разные блоки сайта, и так далее. Еще он позволяет выводить один и тот же документ в разных стилях, например, для печати (обычной или шрифтом Брайля), вывода передачи на экран или чтения голосом.
* **JavaScript** — язык, который создавался, чтобы оживить веб-страницы. Его задача — реагировать на действия пользователя, обрабатывать клики мышкой, перемещения курсора, нажатия клавиш. Ещё он посылает запросы на сервер и загружает данные без перезагрузки страницы, позволяет вводить сообщения и многое другое.

Инженеру по тестированию необходимо знать хотя бы на базовом уровне вышеуказанные технологии.

Если говорить более обширно, то этими тремя технологиями фронтенд-разработка не ограничивается. Она может включать гораздо больше целей или направлений работы (задач или технологий).

В целом, список может быть таким:

1. Вспомогательные библиотеки, обычно используемые в языках программирования (JavaScript), например, Sass или JQuery.
2. Асинхронная обработка запросов и AJAX.
3. Одностраничные приложения (с такими фреймворками, как React, AngularJS или Vue.js).
4. Производительность frontend (первая значимая прорисовка, время до интерактива, анимации и взаимодействия 60 кадров в секунду, использование памяти и т. д.).
5. Адаптивный веб-дизайн.
6. Вопросы совместимости браузеров (кроссбраузерность).
7. Пользовательский интерфейс и т. д.

Со многими вещами вы будете сталкиваться на проектах каждый день.

# Браузеры

Бра́узер (web browser), или веб-обозреватель — прикладное программное обеспечение для просмотра веб-страниц, содержания веб-документов, компьютерных файлов и их каталогов, управления веб-приложениями, а также для решения других задач.

Браузер используется как клиент для отправки запросов на получение определённых данных. К примеру, доступ к какому-либо ресурсу, размещенному в интернете. Это может быть доступ к веб-магазину, информационному сайту, сайту государственных или образовательных учреждений и т. д.

Среди наиболее популярных браузеров (как мобильных, так и для ПК) выделяют следующие:

1. Chrome.
2. Safari.
3. Firefox.
4. Opera.
5. MS Edge.
6. Internet Explorer.

Ниже — статистика использования наиболее популярных браузеров:

|  |  |  |  |
| --- | --- | --- | --- |
| Браузер | Источник данных | | |
| StatCounter  Октябрь 2019 | NetMarketShare  Июнь 2019 | Wikimedia  Июнь 2019 |
| [Chrome](https://en.wikipedia.org/wiki/Google_Chrome) | 64.92% | 64.62% | 45.4% |
| [Safari](https://en.wikipedia.org/wiki/Safari_(web_browser)) | 15.97% | 17.62% | 24.9% |
| [Firefox](https://en.wikipedia.org/wiki/Firefox) | 4.33% | 3.77% | 4.9% |
| [Samsung Internet](https://en.wikipedia.org/wiki/Samsung_Internet_for_Android) | 3.29% | N/A | 3.1% |
| [UC](https://en.wikipedia.org/wiki/UC_Browser) | 2.94% | 1.04% | 0.3% |
| [Opera](https://en.wikipedia.org/wiki/Opera_(web_browser)) | 2.34% | 1.39% | 1.3% |
| [Edge](https://en.wikipedia.org/wiki/Microsoft_Edge) | 2.05% | 2.41% | 1.7% |
| [IE](https://en.wikipedia.org/wiki/Internet_Explorer) | 1.98% | 2.87% | 5.9% |
| [AOSP](https://en.wikipedia.org/wiki/Android_browser) | 0.59% | 1.32% | 0.4% |
| [Others](https://en.wikipedia.org/wiki/List_of_web_browsers) | 1.59% | 4.96 % | 12.1% |

Знание статистики использования браузеров помогает сконцентрировать работу, в частности, тестирование, на наиболее популярных браузерах при кроссбраузерном тестировании. Хотя, как правило, веб-ресурсы собирают статистику, какие клиенты (здесь — веб-браузеры) чаще всего используются конечными пользователи для доступа к ресурсам в сети интернет.

Как уже говорилось ранее, клиент (веб-браузер) отправляет запрос на сервер для получения определенных данных. В соответствии со стандартами RFC, выделяют базовые типы передаваемых данных:

* application;
* audio;
* example;
* image;
* message;
* model;
* multipart;
* text;
* video.

Во время тестирования вам, скорее всего, будут встречаться следующие:

* text/html;
* text/css;
* text/javascript;
* text/plain;
* text/xml;
* image/png или image/jpg;
* video/mp4;
* application/json.

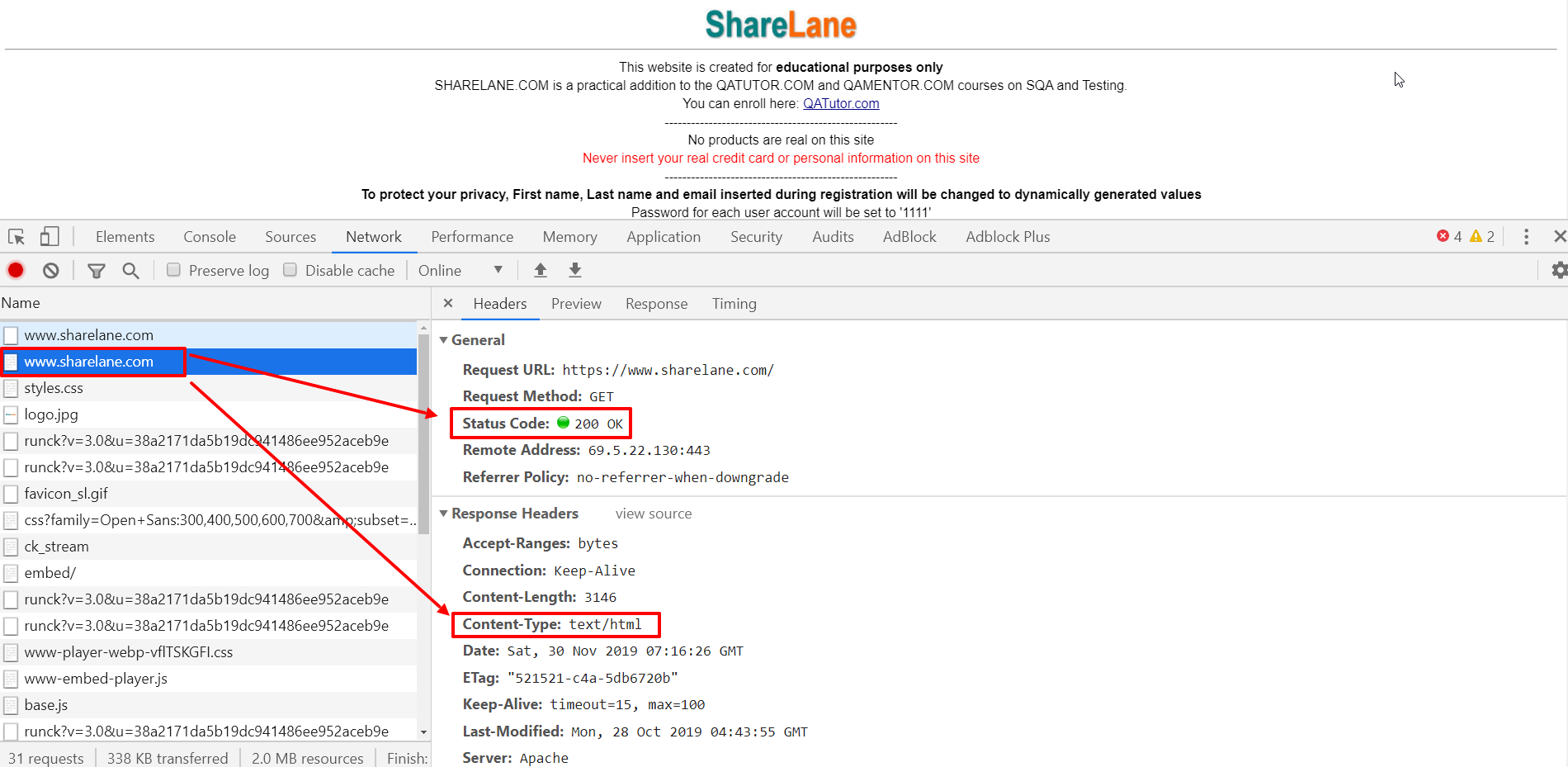
Как это выглядит на практике? Для этого необходимо открыть панель разработчика. Чтобы открыть панель разработчика, мы нажимаем Ctrl+Shift + I или кликаем правой кнопкой мыши и выбираем опцию «Посмотреть код». Далее необходимо перейти во вкладку Network.

К примеру, давайте перейдем на сайт <http://www.sharelane.com/>.

Как видим, при отправке запроса на открытие страницы <http://www.sharelane.com/> мы получили ответ 200, это статус кода ответа на запрос.

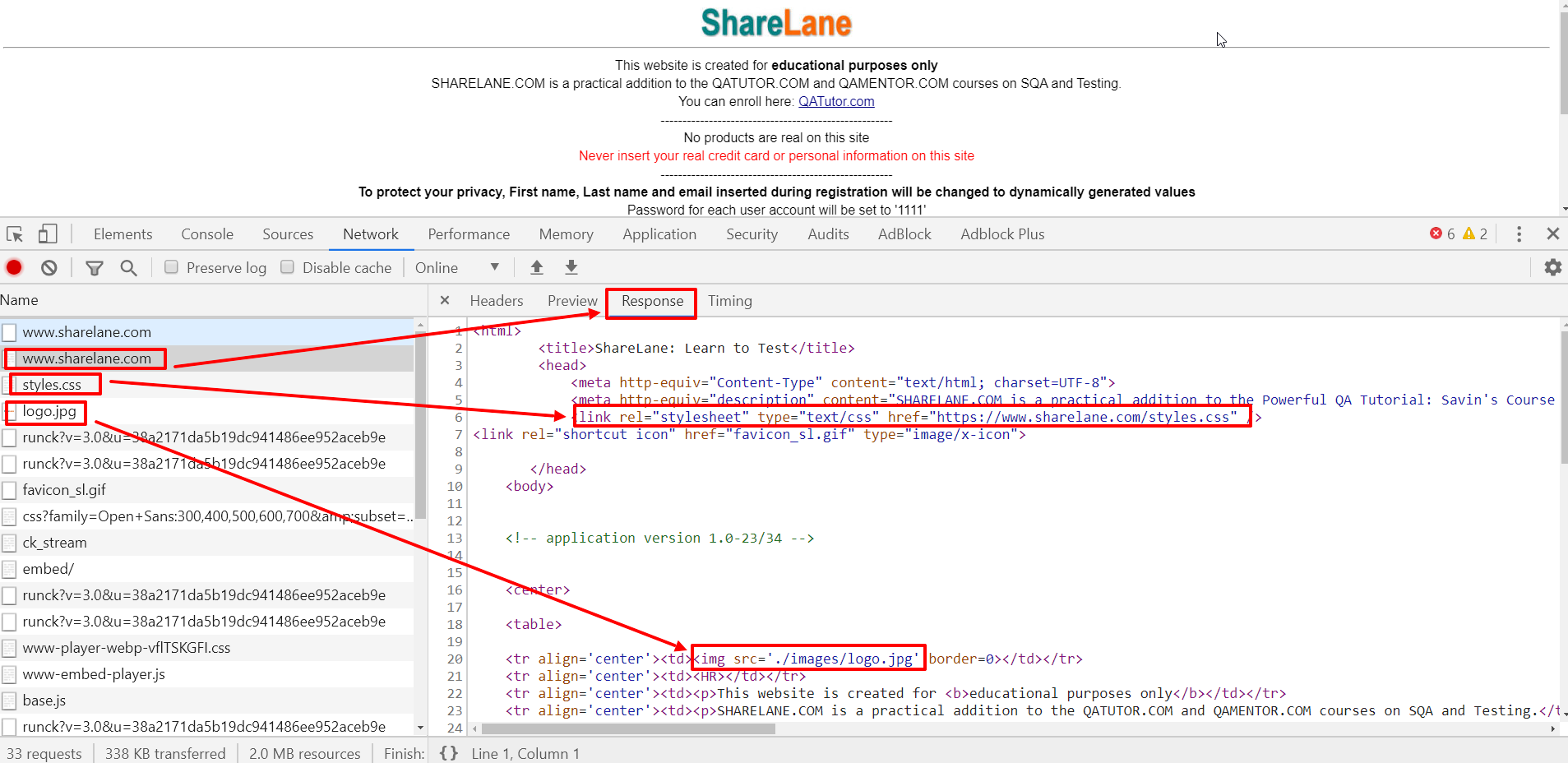
Чтобы детально посмотреть, какой это тип документа, надо открыть сам запрос.

Здесь мы видим Content-Type: text/html документа, полученного в ответ на запрос. То есть браузер получил HTML-страницу:



Если мы откроем вкладку RESPONSE, то увидим, что для полного и правильного отображения страницы, необходимо запросить или загрузить другие дополнительные файлы.

К примеру, это файл со стилями и картинка с изображением логотипа сайта или страницы.



Как результат, мы видим, что клиент, в данном случае наш браузер, отправляет запросы к серверу на получение этих файлов. Итак, следующий запрос — получение файла со стилями <http://www.sharelane.com/styles.css>, здесь у нас Content-Type: text/css.

И следующий запрос — на получение картинки с логотипом <http://www.sharelane.com/images/logo.jpg>. Здесь у нас Content-Type: image/jpeg.

Таким образом, мы можем видеть также и другие запросы, и ответы на них.

Очень важно понимать, как работает браузер, или какие шаги происходят для отображения веб-страницы на экране пользователя (на клиенте).

Основные этапы отображения контента:

1. Запрос ресурса пользователем, то есть ввод URL;
2. Получение веб-страницы или документа в формате HTML от удалённого сервера;
3. Парсинг HTML документа и загрузка дополнительных необходимых файлов с сервера на клиент;
4. Построение DOM-страницы;

DOM (Document Object Model), или объектная модель документа — не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять их содержимое, структуру и оформление. Или, по-другому, это структура данных, используемая браузером для представления документа и отражения его формы.

### Отображение HTML-страницы, или её рендеринг

После получения HTML-страницы браузер производит парсинг документа, грубо говоря, его анализ. Далее строит конструкцию для отображения страницы и её элементов на UI. Дальше, используя файлы со стилями CSS, производит конечную отрисовку страницы.

# Этапы тестирования веб-приложений

Тестирование веб-приложений необходимо проводить в соответствии с определенной логикой и последовательностью шагов. Ниже — один из примеров последовательности тестирования веб-приложения при помощи пользовательского интерфейса.

1. Изучение документации
2. Тестирование вёрстки
3. Тестирование функциональных элементов вёрстки.
4. Тестирование удобства использования.
5. Тестирование безопасности.
6. Тестирование производительности.

На этапе изучения документации тестировщик по максимуму собирает доступную информацию о требованиях. Это могут быть:

1. Описание задачи из Task Management System (Jira).
2. Технические требования из спецификации или RQS (requirement for quality system или требования).
3. Это могут быть макеты или примеры внешнего дизайна пользовательского интерфейса.

На этапе тестирования вёрстки проводят кроссбраузерное тестирование и тестирование вёрстки на при разных масштабах изображения и разном увеличении изображения.

Вёрстка (Layout) — этап дизайна страницы сайта. Представляет собой пространственное размещение на ней текстовых элементов и графических изображений в соответствии с концепцией оформления ресурса.

Вы можете встретиться с такими типами вёрстки, как адаптивная и отзывчивая.

Адаптивная вёрстка (Adaptive layout) — вёрстка, при которой шаблоны и функционал для различных устройств с разной шириной экрана создаются и определяются заранее. Сервер будет определять устройство пользователя и выводить нужный шаблон и контент.

Отзывчивая вёрстка (Responsive layout) — вёрстка, при которой размеры элементов и их пропорции задаются в процентах. Такая структура страницы называется гибкой или резиновой. При уменьшении ширины страницы с такой структурой, её элементы будут плавно сжиматься и уменьшаться относительно друг друга.

Ниже приведен список проверок, которые необходимо проводить при тестировании вёрстки, как краткий чек-лист:

1. Нет ли заметных глазу несоответствий: сломанные блоки, несостыковки цвета, некорректное отображение текста вокруг изображений.
2. Точность соответствия макета (накладка слоёв в Photoshop).
3. Проверка сетки (вертикальные/горизонтальные выравнивания).
4. При уменьшении размера окна меньше минимального по ТЗ не должно ничего ломаться.
5. Фоны не должны смещаться.
6. Проверка масштабирование страницы. В разумных масштабах (диапазон 75-150%) страница должна выглядеть без визуальных недочетов.
7. Изменение размера текстового поля не должно ломать вёрстку.
8. Выделение полей в фокусе, выделение полей с ошибками.
9. Наличие favicon.ico.
10. Проверка в разных разрешениях: (1024x600, 1024x768, 1152x864, 1280x800, 1280x1024, 1440x900, 1680x1050, 1920x1080):
    * не должно ничего ломаться,
    * не должна появляться горизонтальная прокрутка для оговоренных в ТЗ разрешениях,
    * не должны резко обрываться фоны при больших разрешениях.

Этап тестирования функциональных элементов вёрстки может включать следующие проверки:

1. Выделяется ли текст в текстовых блоках.
2. Нажимаются ли кликабельные элементы (ссылки/кнопки).
3. Нажимается ли иконка логотипа.
4. При наведении на кликабельные элементы курсор мыши должен приобретать форму pointer или resize, на недоступные — курсор default.
5. В идеале все активные элементы должны реагировать на наведение.
6. Недоступные/неактивные — не должны.
7. Кликабельные элементы, назначение которых не очевидно, должны быть снабжены подсказками (tooltip) (также относится и к тестированию удобства использования).

Какие элементы пользовательского интерфейса можно (нужно) тестировать:

1. Размер/длина.
2. Текстовое поле ввода.
3. Числовые поля.
4. Радиокнопки.
5. Чекбоксы.
6. Файлы.
7. Дата и время.

Когда мы тестируем размер или длину допустимого вводимого значения в поле, мы проверяем следующие варианты:

1. Минимум.
2. Среднее значение.
3. Максимум.
4. Больше максимального.
5. Меньше минимального.
6. Ноль.

Когда мы тестируем текстовое поле на различные допустимые символы, то мы можем использовать следующие варианты:

1. Буквы, цифры, специальные символы.
2. Тесты для длины строки.
3. Текст с пробелом.
   1. В начале строки.
   2. В середине.
   3. В конце.
4. Только пробел.
5. Не ASCII – ♣☺♂.
6. Редактирование, копирование.
7. Символы конца строки (\n, \r).
8. Перенос строки.

Когда мы тестируем числовые поля, можем использовать следующие значения:

1. То же, что и для текстовых полей.
2. Положительные, отрицательные, ноль.
3. Целые, дробные.
4. Степени двойки.
5. Научная запись чисел (1Е-10).
6. Вычисляемые выражения (2+2).

Когда мы тестируем radio-buttons, проводим следующие проверки:

1. Любое значение можно выбрать.
2. Можно выбрать только одно значение.
3. Можно ли не выбрать ни одного?

Когда мы тестируем check-boxes, проводим следующие проверки:

1. Любое значение можно выбрать;
2. Можно выбрать любое количество значений;
3. Можно ли не выбрать ни одного?

Когда мы тестируем файлы (загрузку файлов), мы проводим следующие проверки:

1. Размер.
2. Тип.
3. Допустимые и недопустимые типы.
4. Требования к содержанию файла.
5. Имя.
6. Длина.
7. Язык.
8. Специальные символы.
9. Существование файла.
10. Права на редактирование/удаление.

Когда мы тестируем дату и время, проводим следующие проверки:

1. Границы минуты, часа, дня, недели, месяца, года, столетия.
2. Переход на летнее/зимнее время.
3. Несуществующие значения.
4. 29 февраля.
5. Часовые пояса.
6. Соотношение с «сейчас».
7. Несколько связанных дат.

На этапе тестирования удобства использования проверяем, насколько конечному пользователю будет понятна работа с приложением и насколько удобно ему будет им пользоваться. Здесь выделяют тестирование удобства использования или UX-тестирование.

Тестирование удобства использования (UX тестирование) направлено на исследование того, насколько конечному пользователю понятно, как работать с продуктом а также на то, насколько ему нравится использовать продукт.

Основные принципы, связанные с удобством использования приложения:

1. Простота изучения: насколько быстро пользователь, который никогда не видел приложения, может его изучить, чтобы решать базовые задачи.
2. Эффективность использования: насколько быстро опытный пользователь, однажды изучивший систему, может решать базовые задачи.
3. Запоминаемость: если пользователь работал с системой раньше, будет ли он помнить достаточно, чтобы эффективно с ней работать в следующий раз.
4. Частота и серьезность ошибок: насколько часто пользователи ошибаются, работая с системой, насколько серьёзны последствия ошибок.
5. Личное удовлетворение: насколько пользователю нравится работать с системой.

10 правил пользовательского интерфейса:

1. Информативность системы — пользователь всегда должен знать текущий статус приложения.
2. Приближенность приложения к реальному миру — диалог с пользователем должен вестись на понятном ему языке, без специфической терминологии.
3. Система должна иметь выходы — приложение всегда должно иметь «запасные выходы» из любой функциональности, которые пользователь по ошибке запустил.
4. Однозначность. Все термины, функции и понятия должны описываться в едином толковании — у пользователя не должно возникнуть путаницы.
5. Предусмотрительность. Система должна всячески «оберегать» пользователя от возможных ошибок.
6. Наглядность. Пользователь не должен ломать голову в попытках понять, что ему нужно делать, или пытаясь вспомнить, как он достиг того или иного состояния системы. Возможные манипуляции с программой должны быть постоянно наглядными.
7. Гибкость и эффективность. Предоставляйте опытным пользователям возможность избегать рутинных действий, и, в то же самое время, скрывайте расширение функционала от неопытных.
8. Лаконичность и точность. Диалоги должны содержать только ту информацию, которую необходимо донести до пользователя, ничего лишнего.
9. Лояльность к ошибкам. Информация об ошибках должна быть понятной и содержать подсказки к дальнейшим действиям.
10. Постоянная справка. Как бы информативно ни была спроектирована система, она всегда должна содержать раздел справки и документации.

На этапе тестирования безопасности проводится проверка, насколько приложение устойчиво к злонамеренным попыткам получения неправомерного доступа к данным или функциям.

Тестирование безопасности (security testing) — тестирование, направленное на проверку способности приложения противостоять злонамеренным попыткам получения доступа к данным или функциям, права на доступ к которым у злоумышленника нет.

Несколько примеров тестирования безопасности, используем текстовые поля:

1. Попытка ввода SQL-инъекций (SELECT \* FROM blog WHERE code LIKE ‘a%’;) — структура запроса зависит от БД.
2. Попытка выполнения XSS-атак — <script>alert(«Alert text»)</script>.
3. Попытка ввода спецсимволов — ”‘~!@#$%^&\*()?>,./\<][ /\*<!–”.
4. Попытка ввода html-тегов — (<form action=”http://live.hh.ru”><input type=”submit”></form>).

Внедрение SQL-кода (англ. SQL injection) — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

На этапе тестирования производительности определяют эффективность производительности системы в целом или отдельных компонентов.

Тестирование производительности (performance testing) — исследование показателей скорости реакции приложения на внешние воздействия при различной по характеру и интенсивности нагрузке.

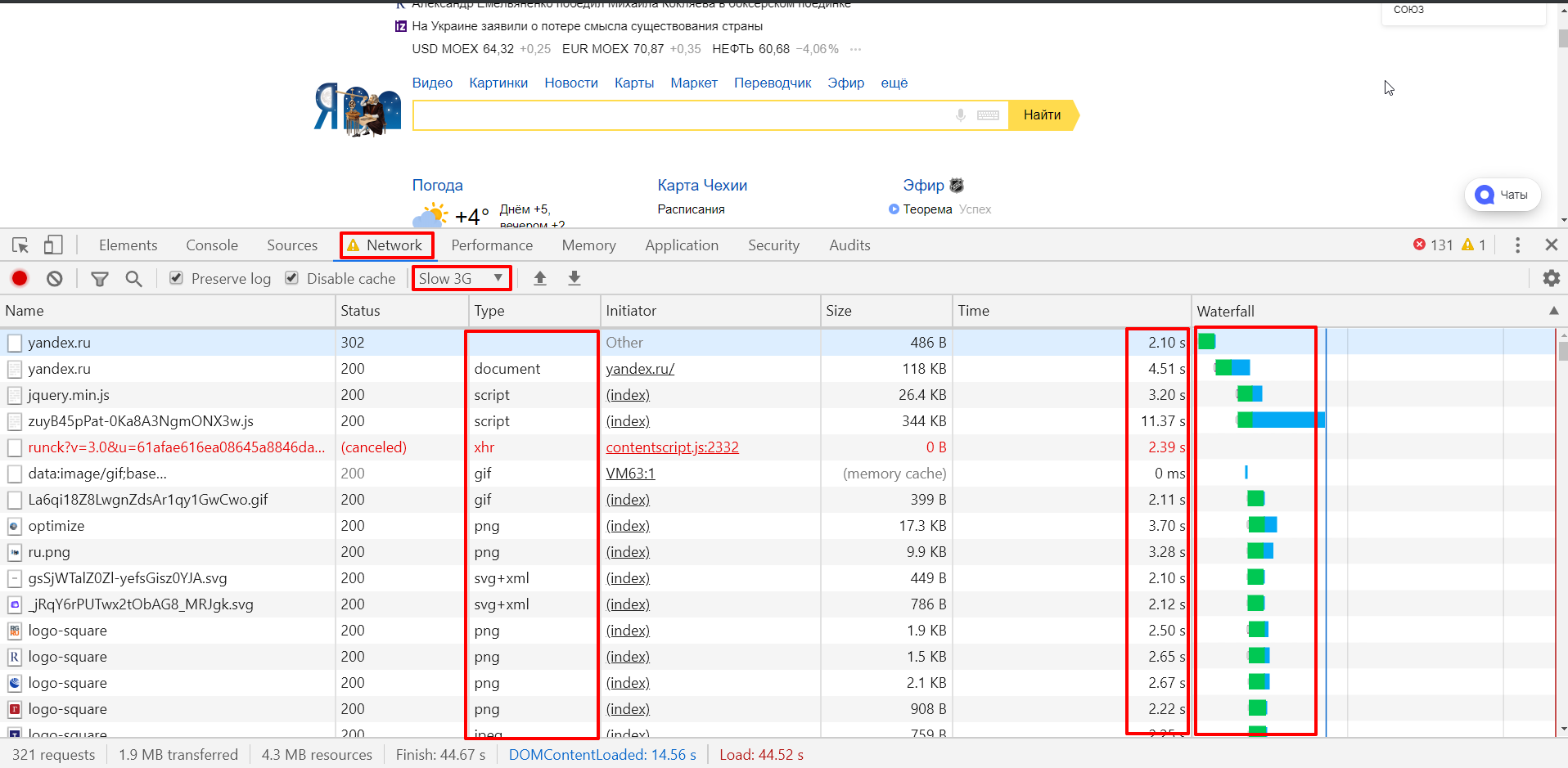
В рамках тестирования производительности выделяют следующие подвиды:

1. Нагрузочное тестирование (load testing, capacity testing) — исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов (определение «запаса прочности»)
2. Тестирование масштабируемости (scalability testing) — исследование способности приложения увеличивать показатели производительности в соответствии с увеличением количества доступных приложению ресурсов.
3. Объемное тестирование (volume testing) — исследование производительности приложения при обработке различных (как правило, больших) объёмов данных.
4. Стрессовое тестирование (stress testing) — исследование поведения приложения при нештатных изменениях нагрузки, значительно превышающих расчетный уровень, или в ситуациях недоступности значительной части необходимых приложению ресурсов.
5. Конкурентное тестирование (concurrency testing) — исследование поведения приложения в ситуации, когда ему приходится обрабатывать большое количество одновременно поступающих запросов, что вызывает конкуренцию между запросами за ресурсы (базу данных, память, канал передачи данных, дисковую подсистему и т. д.).

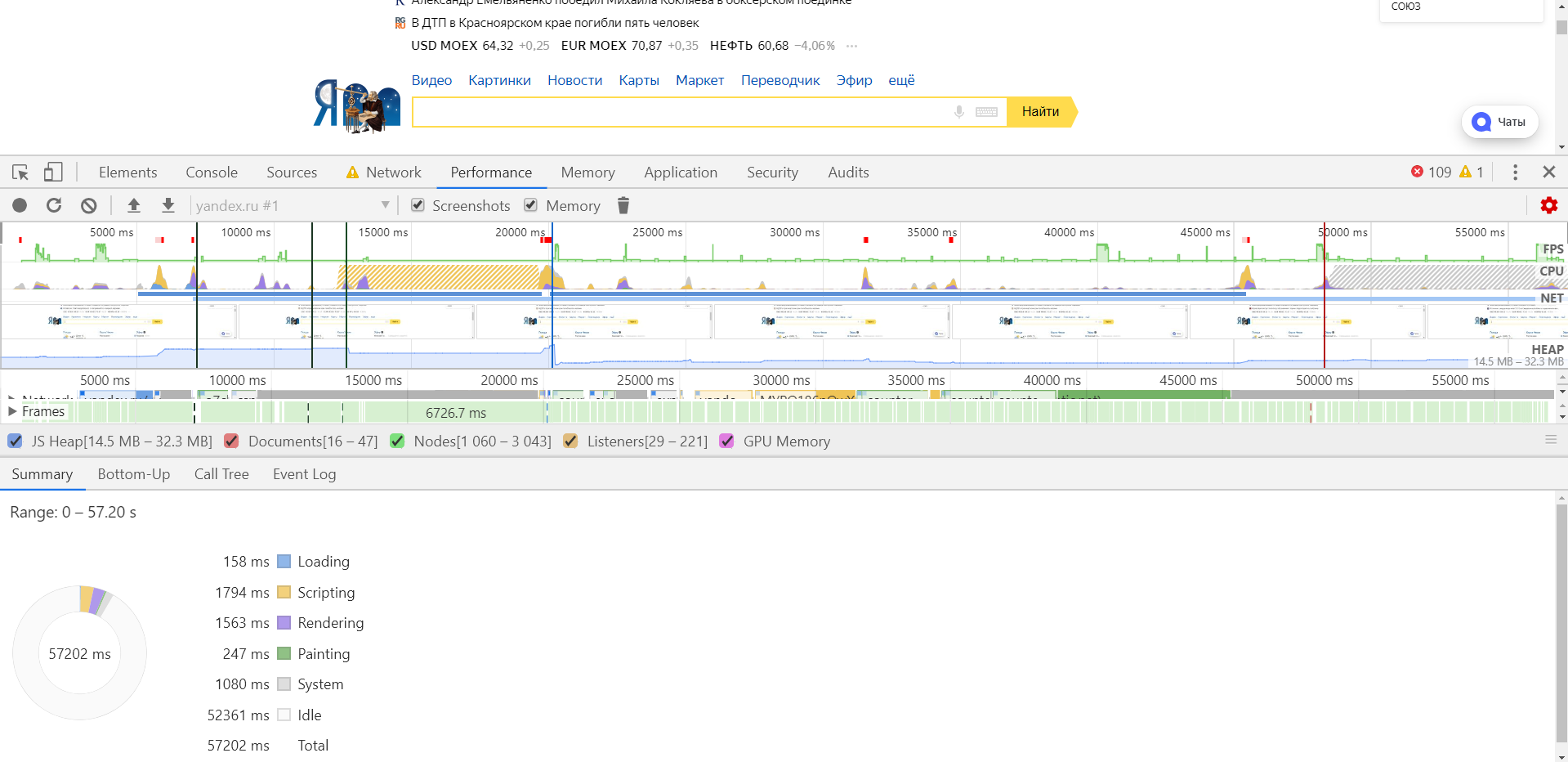
Список инструментов используемых для нагрузочного тестирования:

1. Apache JMeter.
2. LoadRunner.
3. LoadComplete.
4. Rational Performance Tester.
5. WebLOAD.
6. LoadNinja.
7. SmartMeter.io.
8. LoadView.
9. LoadUI NG Pro.
10. Appvance.
11. NeoLoad.
12. Loadster.

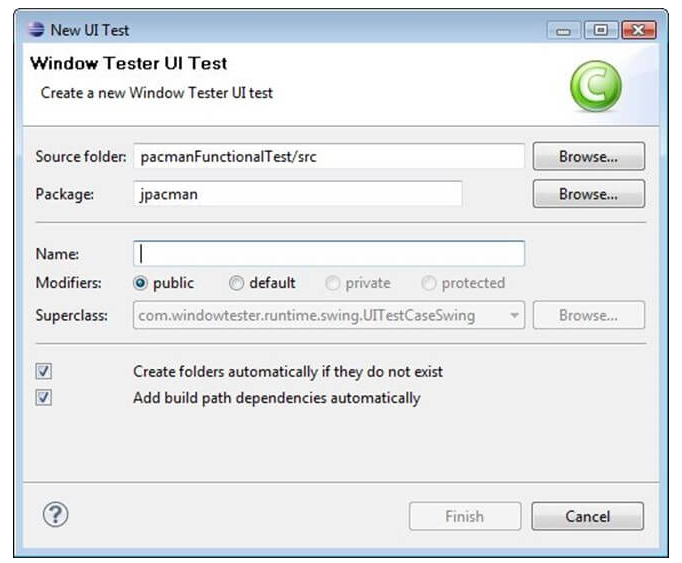
Используя панель разработчика (Dev Tools), можно при помощи встроенных инструментов определить, каким файлам нужно больше времени для загрузки.



Или в течение какого времени браузер отобразит загруженную страницу.



## Пример разбора тестирования одной из форм графического интерфейса пользователя



Примерный список проверок для приведенной формы:

TC 01 — убедитесь, что текстовое поле с надписью **Source Folder** выровнено правильно.

TC 02 — убедитесь, что текстовое поле с надписью **Package** выровнено правильно.

TC 03 — убедитесь, что метка с именем **Browse** — кнопка, которая расположена в конце TextBox с именем **Source Folder**.

TC 04 — убедитесь, что метка с именем **Browse** — кнопка, которая расположена в конце TextBox с именем **Package**.

TC 05 — убедитесь, что текстовое поле с надписью **Name** выровнено правильно.

TC 06 — убедитесь, что метка **Modifiers** состоит из 4 переключателей с названиями public, default, private или protected.

TC 07 — убедитесь, что метка **Modifiers** состоит из 4 переключателей, которые правильно выровнены в ряд.

TC 08 — убедитесь, что метка **Superclass** под меткой **Modifiers** состоит из раскрывающегося списка, который правильно выровнен.

TC 09 — убедитесь, что метка **Superclass** состоит из кнопки с меткой **Browse**, которая правильно выровнена.

TC 10 — убедитесь, что при нажатии любой radiobutton mouse pointer по умолчанию меняется на hand mouse pointer.

TC 11 — убедитесь, что пользователь не может вводить в раскрывающемся списке **Superclass**.

TC 12 — убедитесь, что отображается правильная ошибка, если что-то было неверно выбрано.

TC 13 — убедитесь, что ошибка сгенерирована красным цветом там, где это необходимо.

TC 14 — убедитесь, что в сообщениях об ошибках используются правильные метки.

TC 15 — убедитесь, что отдельные radiobutton выбираются по умолчанию каждый раз.

TC 16 — убедитесь, что кнопка TAB работает должным образом при переходе на другое поле рядом с предыдущим. По правилу — сверху вниз и справа налево.

TC 17 — убедитесь, что все страницы содержат правильный заголовок.

TC 18 — убедитесь, что текст страницы правильно выровнен.

TC 19 — убедитесь, что после обновления любого поля отображается соответствующее подтверждающее сообщение.

TC 20 — убедитесь, что выбрана только одна radiobutton и может быть установлено более одного checkbox.

# Практическое задание

**Используя приложенную таблицу, выполните следующие задания:**

1. Ответить на вопросы в приложенной форме. При проведении какого вида тестирования вы бы нашли тот или иной дефект?
2. Приложите скриншоты из панели разработчика (DevTools) для перечисленных в файле проверок. Сделайте скриншот и выделите необходимую часть требуемой информации (на примере любого из сайтов). Например, статус-код для того или иного ответа.

## Требования к выполненной работе

1. Практическое задание должно быть выполнено в приложенной форме.
2. Расширение файла должно быть формата Excel (XLS, XLSX).
3. Название файла не менять, и вместо ФИО укажите свои фамилию и имя.

# Глоссарий

**Графический интерфейс пользователя** (graphical user interface) — тип интерфейса, который позволяет пользователям взаимодействовать с компонентом или системой посредством иконок и других графических элементов.

**Тестирование графического интерфейса пользователя** (GUI testing) — тестирование, выполняемое путём взаимодействия с приложением через графический интерфейс пользователя.

**Тестирование интерфейса** (interface testing) — вид интеграционного тестирования для проверки интеграции межд компонентами или системами.

**HTML (HyperText Markup Language)** — язык разметки документов, при помощи которого формируется структура страницы: заголовки, абзацы, списки и так далее.

**CSS (Cascading Style Sheets)** — язык для описания и стилизации внешнего вида документа. Благодаря CSS-коду браузер понимает, как именно отображать элементы. CSS задаёт цвета и параметры шрифтов, определяет, как будут располагаться разные блоки сайта, и так далее. Ещё он позволяет выводить один и тот же документ в разных стилях, например, для печати (обычной или шрифтом Брайля), вывода передачи на экран или чтения голосом.

**JavaScript** — язык, который создавался, чтобы оживить веб-страницы. Его задача — реагировать на действия пользователя, обрабатывать клики мышкой, перемещения курсора, нажатия клавиш. Ещё он посылает запросы на сервер и загружает данные без перезагрузки страницы, позволяет вводить сообщения и многое другое.

**Бра́узер, или веб-обозреватель** — прикладное программное обеспечение для просмотра веб-страниц, содержания веб-документов, компьютерных файлов и их каталогов, управления веб-приложениями, а также для решения других задач.

**Вёрстка (Layout)** — этап дизайна страницы сайта. Представляет собой пространственное размещение текстовых элементов и графических изображений в соответствии с концепцией оформления ресурса.

**Адаптивная вёрстка (Adaptive layout)** — вёрстка, при которой шаблоны верстки и функционал для различных устройств с разной шириной экрана создаются и определяются заранее. Сервер будет определять устройство пользователя и выводить нужный шаблон и контент.

**Отзывчивая вёрстка (Responsive layout)** — вёрстка, при которой размеры элементов и их пропорции задаются в процентах. Такая структура страницы называется гибкой или «резиновой». При уменьшении ширины страницы с такой структурой её элементы будут плавно сжиматься и уменьшаться относительно друг друга.

**Тестирование безопасности (security testing)** — тестирование, направленное на проверку способности приложения противостоять злонамеренным попыткам получения доступа к данным или функциям, права на доступ к которым у злоумышленника нет.

**Внедрение SQL-кода (SQL injection)** — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

**Тестирование производительности (performance testing)** — исследование показателей скорости реакции приложения на внешние воздействия при различной по характеру и интенсивности нагрузке.

**Нагрузочное тестирование (load testing, capacity testing)** — исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов (определение «запаса прочности»).

**Тестирование масштабируемости (scalability testing)** — исследование способности приложения увеличивать показатели производительности в соответствии с увеличением количества доступных приложению ресурсов.

**Объёмное тестирование (volume testing)** — исследование производительности приложения при обработке различных (как правило, больших) объёмов данных.

**Стрессовое тестирование (stress testing)** — исследование поведения приложения при нештатных изменениях нагрузки, значительно превышающих расчётный уровень, или в ситуациях недоступности значительной части необходимых приложению ресурсов;

**Конкурентное тестирование (concurrency testing)** — исследование поведения приложения в ситуации, когда ему приходится обрабатывать большое количество одновременно поступающих запросов, что вызывает конкуренцию между запросами за ресурсы (базу данных, память, канал передачи данных, дисковую подсистему и т. д.).

# Дополнительные материалы

1. [Эволюция интернета](http://www.evolutionoftheweb.com/?hl=ru).
2. [Основные функции браузера](https://www.html5rocks.com/ru/tutorials/internals/howbrowserswork/#The_browser_main_functionality).
3. [GUI Testing Tutorial: User Interface (UI) TestCases with Examples](https://www.guru99.com/gui-testing.html).
4. [GUI and Usability Test Scenarios](http://swaretesting.blogspot.com/2013/01/gui-and-usability-test-scenarios.html).
5. [Особенности тестирования веб-приложений](https://quality-lab.ru/blog/key-principles-of-web-testing/).
6. [Is Your Website Fully Usable? Go Through This Checklist To Find Out [57 Things To Do]](https://html.com/blog/website-testing-usability-checklist/).
7. [15 BEST Performance Testing Tools (Load Testing Tools) In 2020](https://www.softwaretestinghelp.com/performance-testing-tools-load-testing-tools/).
8. [13 BEST Performance(Load) Testing Tools in 2020](https://www.guru99.com/performance-testing-tools.html).
9. [Apache JMeter™ Training Academy](https://www.blazemeter.com/jmeter-tutorial/).
10. [Applications Testing](https://support.smartbear.com/testcomplete/docs/app-testing/index.html).

# Используемые источники

1. [Software Engineering Body of Knowledge (SWEBOK)](https://www.computer.org/education/bodies-of-knowledge/software-engineering).
2. [UI Testing Tutorial: User Interface (UI) TestCases with Examples](https://www.guru99.com/gui-testing.html).
3. [ISTQB Glossary.](https://glossary.istqb.org/)
4. [Front and back ends](https://en.wikipedia.org/wiki/Front_and_back_ends).
5. [Usage share of web browsers](https://en.wikipedia.org/wiki/Usage_share_of_web_browsers).
6. [Список MIME-типов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_MIME-%D1%82%D0%B8%D0%BF%D0%BE%D0%B2).
7. [A Guide on Layout Types in Web Design](https://www.webfx.com/blog/web-design/a-guide-on-layout-types-in-web-design/).
8. [Document Object Model](https://ru.wikipedia.org/wiki/Document_Object_Model).