**Универсальная схема для тестирования веб-приложений**

Эту схему можно применить к любому приложению, но предлагаю сузить круг до тестирования веб-приложений.

*GUI — >Functional — >Usability — >Security — >Performance (Load/Stress/Recovery) — >Configuration*

**GUI** — у любого тестируемого предмета и веб-приложения есть внешний вид, поэтому тестирование графического интерфейса или попросту, внешнего вида — это самое первое, что мы можем сделать. Сравнить его с требованиями и/или с макетом и все. Или не все? А как насчет верстки?

**Верстка** — размещение элементов веб-приложения (изображения, текст, кнопки, видео...) в соответствии с макетом или требованиями.

Проверяем:

* наличие всех элементов;
* их размер и цвет;
* расположение относительно друг-друга.

У верстки есть еще множество параметров и элементов, которые мы очень часто забываем проверить.

* *Сравнение с макетом* — метод наложения готового эталонного макета (обычно psd-файл) на приложение в экране браузера, все несовпадения можно рассматривать как ошибки (для этого есть хороший инструмент [Pixel Perfect](http://www.welldonecode.com/perfectpixel/)).
* *Измерение размеров элемента* — если это имеет значение, то померять размеры элемента и сравнить их со спецификацией можно с помощью, например [Page Ruler](https://blarg.co.uk/tools/page-ruler).
* *Правильность шрифтов* (название, размер, цвет) — [WhatFont](https://chrome.google.com/webstore/detail/whatfont/jabopobgcpjmedljpbcaablpmlmfcogm).
* *Цвета интерфейса* — [ColorZilla](http://www.colorzilla.com/chrome/screenshots.html).
* *Контент* — проверить на наличие орфографических и грамматических ошибок ([SpellChecker](https://chrome.google.com/webstore/detail/spell-checker-for-chrome/jfpdnkkdgghlpdgldicfgnnnkhdfhocg" \t "_blank)).
* *Появление курсора* — довольно часто мы забываем проверить, появляется ли вообще и как выглядит курсор в полях ввода, на кликабельных элементах.
* *Фавикон* — такая маленькая незначительная вещица, но может изрядно подпортить впечатление пользователя (в моей практике были случаи, когда разработчики или дизайнеры шаблона оставляли фавикон с логотипом своей компании на сайте у заказчика).
* *Обозначение возможности переноса элементов*.
* *Кодировка* (UTF8...).
* *Стандарты HTML/CSS* — достаточно неплохие решения для быстрой проверки предлагает [W3C](https://validator.w3.org/).
* *Заголовки* по всему приложению должны быть приведены к одному стандарту ([пример](https://neilpatel.com/wp-content/uploads/2017/03/image32-2.png)).
* *Title* страницы — о нем мы тоже часто забываем, также как и разработчики :)
* *Back button* — достаточно часто встречается ошибка при переходе на какую-то страницу и нажатии на браузерную кнопку Back, предыдущая страница крашится или возврат на нее вовсе не осуществляется.
* *Масштабируемость* — особенно это важно при тестировании на смартфонах и планшетах. Где пользователь часто меняет масштаб экрана ([Window Resizer](https://chrome.google.com/webstore/detail/window-resizer/kkelicaakdanhinjdeammmilcgefonfh" \t "_blank)), а также режим адаптивного дизайна (например в FireFox Developer Edition).
* *Кроссбраузерность* — одна и та же страница может выглядеть по-разному в разных браузерах ([пример](http://software-testing.ru/images/stories/library/layout-testing/layout-testing66.png)).
* Проверяем *Scroll*.
* *Браузерные расширения*, которые могут влиять на внешний вид приложения (например, AdBlock) — пробуем включить и отключить.
* Проверить контент при отключенных (режим [WebDeveloper](https://chrome.google.com/webstore/detail/web-developer/bfbameneiokkgbdmiekhjnmfkcnldhhm)) изображениях, flash, JavaScript.

**Локализация** — что мы знаем об этом? Обычно наши знания сводятся к невнятным «ну, это язык», «кодировка», «раскладка», еще реже «геолокация». Что еще мы так часто забываем проверять в рамках тестирования локализации?

* Проверяем тестовый образец на правильность перевода — тут, конечно, хорошо бы подключить переводчика или носителя языка, но за неимением таких, берем тестовый образец и переводим через любой онлайн-переводчик (ну и все мы помним, как прекрасно и весело читать описание товаров на русском языке на AliExpress).
* Длина переведенных слов — количество символов в переведенном слове может быть гораздо больше ([пример](https://hsto.org/storage3/d1b/f24/090/d1bf24090668afe4cbf3024e9953419f.png)), что может привести к «расползанию» интерфейса при переводе.
* Сокращения/аббревиатуры — существуют правила, по которым их либо переводят, либо транслитерируют, либо оставляют как есть.
* Валюта.
* Параметры шрифта могут также значительно отличаться в зависимости от языка ввода.
* Проверить работу поиска во всех локализациях — к примеру, на AliExpress результаты поиска одного и того же слова «смартфон» дают разный результат по количеству найденных товаров, причем разница исчисляется десятками тысяч.
* Мета-информация (keywords/title/description) — столь незначительное для пользователя, невидимое, но такое важное для поисковых машин и продвижения сайта в гугле и других поисковиках.
* RTL (right to left languages) — языки c обратным написанием (арабский, иврит) имеют свои особенности: числа пишутся слева направо, значки и иконки отзеркаливаются, названия программ не переводятся, нет переносов, кнопки редактирования Backspace и Delete работают наоборот.

**Functional**

От внешнего переходим к внутреннему — функциональному тестированию. Если в тестировании GUI мы проверяли наличие и внешний вид элементов, то в функциональном тестировании мы проверяем их **работоспособность и взаимодействие**.

Определить основные функции предмета или приложения достаточно просто — нужно понимать его назначение. Задайте себе вопрос — а для чего нужен карандаш? Занавеска? Интернет-магазин? Для чего нам на сайте нужна форма логина? Для чего нам кнопка «Купить»? И тогда все функции приложения открываются как на ладони.

Самый простой способ подготовиться к функциональному тестированию — это выписать список элементов вашего приложения и написать их целевое назначение («зачем?»).

Например:

|  |  |  |
| --- | --- | --- |
| Кнопка | Зачем? | После нажатия происходит какое-то действие. |
| Поле ввода | Для передачи какой-то информации и взаимодействия с приложением. |
| Поиск | Для того, чтобы пользователь мог быстро найти релевантную информацию. |
| Логин-форма | Чтобы пользователи могли иметь доступ к определенным функциям приложения (или наоборот, ограничить их доступ). |
| Календарь | Например, для выбора дат (билеты, бронирование и т. п.). |
| Дата и время | Например, расписание прибытия транспорта. |
| Сообщения об ошибках | Чтобы сообщить пользователю о том, что приложение работает некорректно, либо он делает некорректные действия. |
| Всплывающие окна и подсказки | Направить пользователя по нужному сценарию. |

У вас уже почти готов список тестовых сценариев. Зная целевое назначение любого элемента, мы можем легко описать все позитивные и негативные сценарии, необходимые для тестирования этого элемента.

Но и тут мы можем кое-что забыть. Часто забываемые проверки функциональных элементов приложения:

Кнопки:

* Enter должна срабатывать как submit;
* Tab должен переводить курсор на следующий элемент.

Поля ввода:

* trimming («убирание») пробелов в полях ввода;
* пустота/пробелы в поле ввода;
* все способы редактирования (Insert, Delete, Backspace, Ctrl+C/V/X/Z и т. д.);
* дроби ( 1.5 | 1,5 | ⅕).

Поиск:

* wildcard symbols (\* | ?);
* написание поискового запроса слитно | раздельно | через дефис должно вести к одному результату;
* ввод текста в другой раскладке.

Сообщения об ошибках:

* пробуем отключить в настройках браузера.

Календарь:

* 31 июня;
* 29 февраля + не высокосный год;
* прошлое/будущее (например, купить билет на уже прошедшее число).

Время:

* синхронизация с сервером (на сервере приложения может быть выставлено другое время, отличающееся от таймзоны пользователя);
* временные зоны.

E-mail:

* логин (63 символа) @ домен (253 символа (может быть ip)).

Всплывающие окна / подсказки:

* пробуем закрыть разными способами (нажатие на кнопку (если есть), на «крестик», клавишей ESC, просто нажатием в другую область экрана);
* рефреш страницы особенно в момент запроса на сервер (например, совершение транзакции по покупке) иногда может приводить к появлению ошибок.

**Usability**

За внешним видом и функциональностью следует удобство (Usability). Не менее важная часть, так как от нее зависит, будет ли востребован ваш продукт вообще. О каких моментах нужно помнить при тестировании usability веб-приложения?

* Соответствует ли приложение ожиданиям конечного пользователя;
* Логичность интерфейса;
* Самое нужное «сверху»;
* Продуманная навигация;
* Локализация (да, да, она относится и сюда тоже);
* Совместимость с другим софтом (соцсети) и железом;
* Скорость работы приложения;
* Информативность (сообщения / обязательные поля);
* Возможность отмены действий пользователя;
* Help — должна быть инструкция, как работать с приложением;
* Возможность печати (если нужно).

**Security**

Тестирование безопасности:

* Начинаем всегда с составления [матрицы уровней доступа;](http://quality-lab.ru/wp-content/uploads/2017/04/matrica.png)
* Конфиденциальность — никто не может получить доступ к данным несанкционированно;
* Целостность данных:   
  а) возможность восстановить данные в полном объеме при их повреждении;  
  б) доступ на изменение информации только определенной категории пользователей.

**Performance**

Производительность:

* Имитируем нагрузку пользователями ([JMeter](http://jmeter.apache.org/" \t "_blank));
* Пробуем загрузить большие объемы данных, файлы, медиа;
* Нагружаем БД;
* Понижаем скорость инета ([NetLimiter](https://www.netlimiter.com/" \t "_blank));
* Понижаем скорость передачи данных (Throttling);
* Тестируем восстановление системы после падений.

**Configuration**

Конфигурационное тестирование. Тут все тоже просто:

* Берем у разработчиков/заказчика список софта и железа, на котором и с которым должно работать наше приложение.
* Думаем над тем, с чем еще взаимодействует приложение (например соцсети, почта, возможно, камера на телефоне и т. п.).
* Выписываем это все в список (ОС, браузеры, их версии для ПК, мобильных телефонов, планшетов, также (если это важно) выписываем на каком разрешении или с какими настройками (например, для камеры съемка в режиме HD) нужно проводить тестирование).
* Далее можем использовать метод классов эквивалентности, pairwise или просто руководствуемся тем, что есть в наличии, и настраиваем тестовое окружение с нужными конфигурациями.

*Внешнее — > Внутренее — >Стойкость — >Взаимодействие*

**Внешнее** — проверка внешнего вида и функций, которые доступны только обычному пользователю (GUI, Usability).

**Внутреннее** — все функции приложения (Functional).

**Стойкость** — сюда мы отнесем устойчивость приложения к нагрузкам и к попыткам нарушить его безопасность (Security, Performance (load/stress/recovery)).

**Взаимодействие** — работа приложения с другим софтом и железом (Configuration).

**Памятка**

|  |  |
| --- | --- |
| GUI | * макет * контент * кодировка * элементы (цвет, размер, расположение) * локализация * стандарты HTML/CSS * масштабируемость * курсор * заголовки * шрифты * фавикон * scroll * кроссбраузерность |
| Functional | * работа кнопок * имейл * регистрация/авторизация * поля ввода * время и дата * сообщения об ошибках * поиск * всплывающие окна/подсказки * формы заполнения * календари * взаимодействие всех модулей системы |
| Usability | * навигация * соответствие целям приложения * печать * логичность * локализация * help * информативность * совместимость с другими приложениями * ожидания конечного пользователя * скорость работы |
| Security | * матрица уровней доступа * протоколы передачи данных * конфиденциальность информации * протоколы криптования * доступность информации * авторизация |
| Performance | * нагрузка * имитация количества пользователей * БД нагрузка * стабильность * «тяжелый» медиа-контент * скорость выполнения запросов к БД * стресс * скорость интернета * корректные сообщения об ошибках * восстановление * объем загружаемых файлов * восстановление данных / системы |
| Configuration | * сторонний софт * «железо» * совместимость с другими браузерами * OS |