Шпаргалка 2 спринта

Клиент-серверная архитектура

Веб-приложение устроено по определённым принципам. Они определяют, из каких элементов состоит приложение и как они связаны.

Такие принципы называют **архитектурой**. Это способ организации работы приложения.

Самая распространённая архитектура веб-приложений — **клиент-серверная**.

Клиент, сервер и интернет — элементы клиент-серверной архитектуры. Клиент и сервер работают по отдельности: клиент отвечает за взаимодействие с пользователем, сервер — за логические операции, вычисления и хранение данных, а интернет, или сеть — за связь клиента и сервера.

Клиент — это система, которая связывается с сервером и запрашивает нужную пользователю информацию.

Сервер — система, которая обрабатывает запросы клиента и формирует ответ. Например, сохраняет заказ или передаёт информацию о цене.

Клиент и сервер «общаются» по сети.

Интернет, или **Сеть** — система связанных между собой устройств, которая помогает клиенту и серверу обмениваться данными.

Веб-приложение

Браузер — это ПО, которое отправляет запросы на сервер, получает ответы, обрабатывает информацию и отображает результат на странице веб-приложения.

Именно через браузер пользователь выполняет действия в веб-приложении. Браузер отправляет запросы на сервер, получает ответ, обрабатывает информацию и отображает результат на странице веб-приложения.

Само веб-приложение — это:

- Набор веб-страниц, которые пользователь видит в браузере. Например: главная страница Яндекс.Практикума, страница профессии «Инженер по тестированию».

- Набор логических операций и вычислений, которые обрабатывают запросы пользователя. Например, когда ты нажимаешь на кнопку «Выбрать курс», браузер отправляет твой запрос на сервер, получает от него ответ и отображает на странице список всех курсов Практикума.

Код веб-страниц пишет фронтенд-разработчик. Код операций и вычислений, которые обрабатывают запросы и формируют ответы, пишет бэкенд-разработчик.

Фронтенд и бэкенд

Фронтенд (frontend) — видимая часть приложения. С ней пользователь взаимодействует: например, нажимает на кнопки или вводит текст. В модели «клиент-сервер» фронтенд — это код, который обрабатывается на стороне клиента.

Бэкенд (backend) — скрытая часть приложения. Она отвечает за вычисления, логику, хранение данных. В модели «клиент-сервер» бэкенд — это код, который работает на удалённом сервере.

URL

URL (Uniform Resource Locator — унифицированный указатель ресурса) — это адрес веб-ресурса. URL показывает, где находится веб-приложение, веб-страница или фрагмент веб-страницы и как к ним обратиться.

Доменное имя — это адрес веб-приложения, под которым оно зарегистрировано в интернете. Доменное имя состоит из символов, которые присвоили приложению в системе доменных имён (DNS)

Когда ты вводишь URL в адресной строке, браузер инициирует запрос к **DNS**-серверу (Domain Name System — система доменных имён).

IP-адрес — это уникальный идентификатор сервера, на котором находится нужная информация.

Протоколы HTTP и HTTPS

Протокол передачи данных — набор правил, по которым устройства обмениваются информацией.

Один из таких протоколов — **HTTP** (HyperText Transfer Protocol — протокол передачи гипертекста). Сейчас по HTTP передаётся не только, но и другие данные: картинки, аудио, видео.

Чтобы обеспечить безопасность, применяют протокол HTTP с расширением защиты — **HTTPS** (HyperText Transfer Protocol Secure — протокол защищённой передачи гипертекста). Он шифрует соединение по криптографическим протоколам: так клиент и сервер смогут безопасно передавать сообщения друг другу.

Структура HTTP-запроса

По протоколу HTTP фронтенд передаёт запрос (request), а бэкенд — ответ (response).

Запрос — это специальное сообщение. В нём фронтенд «просит» у бэкенда совершить определённые действия с данными: например, отправить, сохранить или изменить их.

Запрос структурирован по правилам передачи данных HTTP. Он состоит из трёх блоков:

- стартовая строка,
- заголовки,
- тело сообщения.

Стартовая строка

Стартовая строка (start line) состоит из трёх элементов: метода, пути до ресурса и версии протокола.

- **Метод** (method) указывает действие: бэкенд принимает его в обработку. Самые распространённые — GET, POST, PUT, DELETE.
- **Путь до ресурса** — адрес, по которому фронтенд отправляет запрос на бэкенд.
- **Версия протокола** — номер версии HTTP. Сейчас применяют версию HTTP/1.1.

Заголовки запроса

Заголовки запроса (Request Headers) — дополнительная информация от фронтенда бэкенду.

Различают:

- общие заголовки (general headers);

- заголовки запроса (request headers);
- заголовки сущности (entity headers).

Заголовки запроса — дополнительная информация о клиенте.

Общие заголовки содержат параметры, которые передаются как в запросах, так и в ответах.

Заголовки сущности задают характеристики контента в теле.

Тело сообщения

Тело сообщения — это данные, которые передаёт фронтенд.

Структура HTTP-ответа

HTTP-ответ включает:

- строку состояния,
- заголовки,
- тело сообщения.

Строка состояния

Строка состояния включает версию протокола, код состояния, текст состояния.

Например: `HTTP/1.1 200 OK`.

- **Версия протокола** — номер версии HTTP, для которой сформирован ответ. Например: HTTP/1.1.
- **Код состояния** (Status Code) — трёхзначное число, которое указывает код результата: успешно ли сервер обработал запрос. Например: 200.
- **Текст состояния** (Reason Phrase) — текст, который сопровождает код состояния. Например, «OK».

Разработчики используют стандартизированный список кодов и текстов состояний: Hypertext Transfer Protocol — HTTP/1.1.

Заголовки ответа

Заголовки ответа (Response Headers) — дополнительная информация о бэкенде и типе данных для фронтенда.

Структура заголовков похожа на ту, что у клиента. Различают:

Общие заголовки содержат параметры, передаваемые как в запросе, так и ответе. Например: Connection — состояние соединения между клиентом и сервером, Date — дата создания ответа от сервера.

Заголовки ответа позволяют серверу передавать об ответе дополнительную информацию, которую нельзя поместить в строку состояния. Например: Server — какое ПО использует сервер, чтобы обработать запрос и сформировать ответ, Location — точное указание местоположения ресурса.

Заголовки сущности задают характеристики контента в теле. Например, Content-Type — тип данных в ответе.

Тело сообщения

В теле сообщения бэкенд отдаёт результат выполнения запроса.

Просмотр запросов фронтенда и ответов бэкенда

Чтобы просматривать запросы и ответы, обычно пользуются DevTools (Tools for Web Developers — инструменты разработчика). **Инструменты разработчика** доступны в современных браузерах.

- Чтобы открыть инструменты разработчика: в Яндекс.Браузере нужно кликнуть правой кнопкой мыши в любой области страницы и выбрать пункт «Исследовать элемент». В Chrome — «Просмотреть код». Или используй сочетание клавиш: для Windows — `Ctrl+Shift+i`, для macOS — `Cmd+Opt+J`.
- В верхней части страницы ты увидишь вкладки: Elements, Console, Sources, Network, Performance, Memory, Application, Lighthouse. Это и есть инструменты разработчика. У каждой вкладки свои возможности.

На вкладке Network доступна следующая информация:

- названия ресурсов, которые фронтенд и бэкенд передают по сети (колонка Name);
- длительность загрузки (колонка Time);
- размер файлов (колонка Size);
- сводная информация о запросах к серверу: например, объём трафика.

Типовые задачи тестировщика веб-приложений

Задачи тестировщика можно разделить на три типа:

- тестирование новых фич;
- проверка исправленных багов;
- прочие задачи.

Тестирование новой функциональности

Фича (feature — особенность) — изменение или дополнение к функциональности приложения.

Например, проджект-менеджер решает добавить в Яндекс.Маршруты новый вид транспорта. Прежде чем эта задача попадёт к тестировщику, она пройдёт несколько этапов разработки — **жизненный путь**.

Проверка исправленных багов

Разработчики исправляют баги, а тестировщик проверяет приложение ещё раз.

Баги можно найти на двух этапах:

- До релиза фичи — в ходе тестирования. Их видит только команда.
- После релиза фичи. Их видят пользователи. Например, пользователь заметил баг и написал в техподдержку, а тестировщик составил баг-репорт и сообщил разработчикам.

Обычно баг-репорты составляют в трекере задач.

Задач на изменение функциональности и исправление багов может быть много. Для их отслеживания команда использует специальные системы управления задачами — **трекеры**. Например: Яндекс.Трекер, Trello, Jira.

Прочие задачи

Иногда руководитель ставит тестировщику задачу напрямую: например, провести определённый вид тестирования или выполнить локальную задачу.

С видами тестирования тебе удалось ознакомиться в бесплатной вводной части. Задачи по разным видам тестирования могут могут выглядеть так:

- нагрузочное тестирование — проверить, что приложение выдерживает сценарий оплаты заказа при 100 000 пользователей;
- интеграционное тестирование — проверить, как работает интеграция между Яндекс.Маршрутами и банком, когда пользователь оплачивает заказ;

- регрессионное тестирование — проверить, что в новой версии Яндекс.Маршрутов все элементы сервиса работают так же, как и до обновления;
- смоук-тестирование — проверить, не сломались ли основные элементы приложения.

Локальные задачи могут выглядеть так:

- разработать тестовые сценарии под функциональность оплаты заказа;
- проанализировать критические баги с продакшена и оформить чек-лист;
- подготовить отчёт по тестированию новой версии приложения.

Типовые изменения в веб-приложениях

В веб-приложении могут изменить интерфейс, логику фронтенда и логику бэкенда.

Изменение интерфейса

Есть два вида интерфейсов:

- графический пользовательский интерфейс — GUI,
- интерфейс командной строки — CLI.

GUI (Graphical User Interface — «графический пользовательский интерфейс») — это привычный для пользователей интерфейс. Информация выводится на компьютере в виде графических элементов: меню, кнопок, форм ввода данных, иконок. Через элементы на экране пользователь отправляет команды компьютеру.

CLI (Command Line Interface — «интерфейс командной строки») — пользователь взаимодействует с компьютером, отправляя ему текстовые команды. В ответ компьютер выводит результаты — тоже текстом. Среда, в которой пользователь общается с компьютером, — это консоль. Её можно использовать для работы с приложениями, у которых нет графического интерфейса.

Изменение логики фронтенда

Логика фронтенда — это программный код, который написал фронтенд-разработчик по требованиям. Требования определяют, как элементы интерфейса реагируют на действия пользователя.

К логике фронтенда относят:

- Возможность элементов интерфейса изменять состояние. Например, кнопка может менять форму при нажатии.
- Возможность элементов интерфейса проверять, что пользователь ввёл данные корректно. Такую проверку называют **валидацией**. Например, в Яндекс.Почте не получится зарегистрироваться, если пароль короче шести символов.
- Выполнение операций и вычислений на стороне самого фронтенда. Эти операции и вычисления зафиксированы в требованиях к приложению. Вычисление суммы может производиться на стороне фронтенда или бэкенда — это решает команда, которая разрабатывает приложение. Например, пользователь добавляет товары в корзину и видит сумму к оплате.

Изменение логики бэкенда

Логика бэкенда — это программный код, который написал бэкенд-разработчик. К логике бэкенда относят:

- Возможность бэкенда проверить, что данные из запроса фронтенда соответствуют требованиям. Такую проверку называют валидацией API. Например, если пользователь ввёл недопустимые по требованиям символы в поле логина, в ответе бэкенда будет сообщение об ошибке. Подробнее о валидации в API ты узнаешь в третьем спринте.
- Выполнение операций и вычислений на стороне бэкенда. Вспомни пример с расчётом стоимости заказа в интернет-магазине: вычисления могут производиться и на стороне бэкенда.

Инструмент для работы с макетами

Макет — визуальное представление того, как должен выглядеть интерфейс веб-страницы.

На макет ориентируются и разработчики, и тестировщики.

- Фронтенд-разработчик использует макеты, чтобы написать код интерфейса. Дизайнер должен заранее спроектировать удобный и простой интерфейс. Такое свойство называют **юзабилити** (usability — удобство использования).

- Тестировщик сопоставляет готовый интерфейс и макеты: например, ищет несоответствия в расположении элементов или в цветовой гамме.

Figma

Один из самых распространённых инструментов работы с макетами — **Figma**. Это полноценный графический редактор, который доступен в браузере. В Figma можно совместно работать над проектом онлайн — как в Google Docs или Яндекс.Документах.

Элементы интерфейса

Каждый элемент интерфейса проверяют по двум срезам: визуальному и функциональному.

Визуальная проверка

Визуальную проверку ещё называют тестированием вёрстки.

Вёрстка — это соединение и расположение элементов интерфейса, которое при этом соответствует макету.

Тестировщик проверяет, что веб-страница в браузере выглядит как задумано. Если замечаешь расхождения — например, цвет или размер кнопки не совпадает с макетом — нужно завести баг-репорт.

Функциональная проверка

Тестировщик должен убедиться, что элементы интерфейса работают корректно: кнопки нажимаются, ссылки ведут на нужные веб-страницы.

Если несколько элементов формирую единую функциональность, их объединяют в группы, или **формы**.

Посмотри, какие элементы интерфейса выделяют:

Общие элементы	Элементы выбора и ввода данных	Элементы действия	Вспомогательные
Главное меню	Радиокнопка	Иконка	Надпись
Контекстное меню	Чекбокс	Кнопка	Всплывающая подсказка
Полоса прокрутки	Поля ввода	Ссылка	Плейсхолдер
	Область ввода текста		
	Выпадающий список		
	Раскрывающийся список		

Общие элементы

Главное меню (main menu) — основной элемент пользовательского интерфейса в приложении, который делает возможной навигацию по разделам.

Контекстное меню (popup menu) — меню действий, которое можно вызвать у выбранного элемента. Например, если ты выделяешь текст и нажимаешь правую кнопку мыши, всплывает контекстное меню.

Полоса прокрутки (scrollbar) — элемент, который передвигает визуальную область окна браузера вверх-вниз, вправо-влево.

Элементы выбора данных

Радиокнопка (radiobutton) — элемент круглой формы. Выделяется точкой внутри. Рядом с радиокнопкой всегда добавляют обозначение либо описание. Иногда их размечают группами по несколько штук. Радиокнопка предполагает, что можно выбрать только один вариант. Её название и произошло от функциональности старых радиоприёмников, на которых нажатие одной кнопки приводило к тому, что все остальные «отжимались».

Чекбокс, флажок, галочка (checkbox) — ☑ включено и ☐ отключено. Почти всегда это элемент квадратной формы, реже — круглой. Внутри можно установить флажок или галочку. Обычно рядом добавляют описание.

Раскрывающийся список, комбинированный список (combobox) — подвид элемента «Список». Состоит из поля ввода текста и выпадающего списка. Список раскрывается, если нажать на кнопку раскрытия или начать вводить текст. Позволяет ввести текст значения вручную или раскрыть список и выбрать значение.

Выпадающий список (select menu, drop-down list) — подвид элемента «Список». Раскрывается, если нажать в любую его область. Вводить данные не позволяет — только выбирать. Бывает только с единственным выбором значения из списка, а бывает с множественным.

Элементы ввода данных

Поле ввода — область, в которой пользователь вводит данные. В текстовые поля можно вводить любые значения. При этом разработчик в коде фронтенда

может ограничивать ввод значений согласно требованиям. Например, задать формат ввода автомобильного номера: только русские буквы и цифры.

Область ввода текста, многострочный текст (textarea) — поле ввода из нескольких строк. Применяют, чтобы вводить и редактировать длинный текст.

Элемент действий

Иконка, картинка (icon) — небольшое изображение, которое иллюстрирует действие. При нажатии это действие и вызывается.

Кнопка (button) — элемент с текстом внутри, вызывает определённое действие при нажатии.

Ссылка — связующий элемент веб-приложения. Может отсылать к элементу: например, к картинке или блоку текста внутри веб-страницы. Или к объекту: файлу, папке, другой веб-странице.

Вспомогательные элементы

Надпись, метка (label) поясняет значения элементов: например, чек-боксов.

Всплывающая подсказка (hint) — текст, который появляется при наведении на элемент и подсказывает его значение.

Плейсхолдер (placeholder) — указатель, где нужно заполнять форму; или подсказка внутри формы. Плейсхолдер должен исчезать, если кликнуть по полю или начать вводить данные.

Формы

Форма — это компонент страницы для получения информации от пользователя. Форма регистрации, область заказа в интернет-магазине, окно для написания поста в Facebook — всё это формы.

У формы есть три особенности.

- Чтобы пользователь мог вводить и отправлять данные, в форме размещают элементы: кнопки, радиобаттоны, чекбоксы, поля с выпадающими списками и поля ввода текста.
- У формы есть цель — пользователь должен совершить в веб-приложении определённое действие. Например, зарегистрироваться.
- Пользователь должен понимать, когда цель достигнута. Например, после регистрации в Яндекс.Паспорте на экране появляется сообщение: «Ваш

аккаунт готов!»

Тестирование форм

Формы проверяют целиком. Например, в форме регистрации нужно ввести имя, фамилию, логин, пароль и номер телефона.

Данные выполняют одну задачу — фронтенд не отправит запрос, пока ты не заполнишь все поля. Поэтому тестировать элементы нужно вместе.

Сначала проводят позитивные проверки, потом негативные — об этом ты знаешь из первого спринта.

Валидация

Валидация — механизм, который проверяет, что пользователь ввёл данные согласно требованиям.

Если валидация прошла успешно, приложение будет работать корректно. Если нет — пользователь увидит сообщение, что заполнил поле неверно.

Правила валидации разработчики прописывают в коде. Это делают как на стороне фронтенда, так и на стороне бэкенда.

- В первом случае фронтенд проверяет данные сразу. Например, Яндекс.Паспорт выдаёт сообщение «Пароль содержит запрещённые символы», как только пользователь заполнил поле «Придумайте пароль».
- Во втором случае бэкенд проверяет все данные только после того, как получил запрос. Например, в Яндекс.Формах валидация полей происходит, когда пользователь заполнил форму целиком и нажал кнопку «Сохранить».

Тестирование текста на ошибки

Тексты часто готовят по такому сценарию:

- Менеджер ставит задачу редактору. Например: «Текст для кнопки заказа такси должен содержать заголовок и дополнительную информацию о расстоянии и времени в пути».
- Редактор готовит текст. Например: «**Заказать** Маршрут составит [3] км и займет [3] мин».
- Когда тексты готовы, дизайнер встраивает их в макеты.

- Разработчик на этапе написания кода добавляет этот текст в нужные элементы интерфейса: в примере выше — на кнопку заказа такси.

На каждом этапе с текстом может что-то пойти не так:

- Текст непонятен пользователю.

Например, для кнопки заказа такси редактор написал: «Готово». Непонятно, что будет после нажатия этой кнопки — произойдёт ли заказ такси или откроется ещё одна форма ввода.

- Дизайнер перепутал тексты на макете, а разработчик перенёс их в код.

Например, текст на кнопке заказа такси стал таким: «**Заказать** Маршрут составит [3] мин и займёт [4] км». Правильный вариант — «**Заказать** Маршрут составит [4] км и займёт [3] мин».

- Разработчик допустил опечатку и перенёс в код текст с ошибкой.

Например, поменял местами две буквы в слове «Маршрут» — получилось «Маршурт».

- Разработчик не перенёс в код финальный вариант текста.

Например, редактор ещё не подготовил текст, и разработчик перенёс в код временный вариант, но забыл его поменять. В результате в интерфейсе осталась подсказка: «Тут будет текст кнопки».

Именно поэтому тестировщику нужно обращать внимание не только на элементы вёрстки, но и на тексты в интерфейсе. Они влияют на юзабилити приложения и репутацию компании.

Кроссплатформенное тестирование

Платформы

Пользователь может открыть приложение на ноутбуке, смартфоне или планшете. Это и есть **платформы** — типы устройств, с которых пользователь работает с приложением.

Платформы бывают двух типов:

- **Десктопная** — компьютеры и ноутбуки.
- **Мобильная** — смартфоны и планшеты.

Одно и то же приложение адаптируют под разные платформы — разрабатывают десктопную и мобильную версию. Это делают, чтобы было удобно использовать сервис и на маленьких, и на больших экранах.

Мобильную и десктопную версии нужно тестировать отдельно. Бывает так, что в десктопной версии всё соответсвует макетам и требованиям, а в мобильной «поехала» вёрстка: кнопки, подсказки и другие элементы оказались не там, где нужно.

Операционные системы

Чтобы устройство «общалось» с пользователем, нужен целый набор программ. Например, одни обрабатывают сигналы с клавиатуры и мыши, другие позволяют создавать и удалять файлы.

Набор программ для «общения» устройства и пользователя называется **операционной системой** — сокращённо **ОС**. Примеры ОС — Windows, Android, macOS.

Приложение на разных операционных системах нужно тестировать отдельно. Если приложение соответствует макетам и требованиям на iOS, на Android что-то может пойти не так: может отличаться вёрстка, кнопки, расположение элементов.

Комбинации платформ и операционных систем

Некоторые операционные системы подходят только для десктопных платформ, а другие — только для мобильных.

Нужно проверять разные комбинации платформ и операционных систем. Если в одном случае приложение работает без багов, в другом могут появиться ошибки.

Чтобы убедиться, что приложение соответствует требованиям и макетам на любых платформах и ОС, проводят **кроссплатформенное тестирование**. Тестировщик проверяет, что:

- пользовательский интерфейс соответствует макету;
- функциональность работает так, как описано в требованиях.

Как понять, на каких ОС проверять приложение

Посмотреть в требованиях

Часто в требованиях к приложению уже прописан набор операционных систем для десктопных и мобильных устройств. Тогда нужно протестировать только их.

Спросить у менеджера

Иногда бывает, что ОС не указаны в требованиях. Тогда можно спросить у менеджера проекта, какие ОС поддерживает приложение: их и надо тестировать.

Посмотреть в локальной статистике

Если поддерживаемые ОС не указаны в требованиях и менеджер тоже про них не знает, обратись к **локальной статистике по приложению**.

Большинство компаний собирают данные о том, как пользователи используют приложение. Например, сколько людей его открывают, на какие кнопки нажимают чаще всего, какой функциональностью пользуются.

Разработчик добавляет в приложение код, который отправляет данные в специальные инструменты — например, Яндекс.Метрику или Гугл.аналитику. Они собирают аналитику и переводят ее в наглядный вид.

На основе этих данных аналитики и менеджеры понимают, как развивать приложение. А тестировщик может определить, какие платформы и операционные системы стоит проверить.

Посмотреть в общей статистике по ОС

Иногда локальной статистики по приложению нет — например, если сервис только запускается. Тогда можно посмотреть общую статистку по операционным системам в стране, для которой разрабатывается приложение.

Кроссбраузерное тестирование

Разработчики не всегда могут предугадать, как элементы интерфейса будут выглядеть в разных браузерах. Например, в Яндекс.Браузере вёрстка может соответствовать макетам, а в Safari — нет. Это связано с особенностями разных браузеров, о них ты узнаешь в следующем уроке.

Нужно проверить **кроссбраузерность** — способность приложения работать одинаково в разных браузерах. Провести **кроссбраузерное тестирование** — значит убедиться, что в разных браузерах расположение, цвет, форма элементов и текст в интерфейсе такие же, как на макете.

Когда проводят кроссбраузерное тестирование

Ты знаешь, что в приложении могут меняться:

- логика работы,
- вёрстка.

Кроссбраузерное тестирование нужно проводить, если были доработки или изменения **в вёрстке**.

Если добавилась или изменилась только логика работы приложения, кроссбраузерность проверять не нужно — достаточно протестировать изменённую или новую функциональность в одном браузере.

Если меняется и логика работы, и вёрстка, тестировщик проводит все проверки в одном из поддерживаемых браузеров. В остальных браузерах нужно протестировать только вёрстку.

Как выбрать браузеры для тестирования

Выбрать браузеры для тестирования можно так же, как операционные системы:

- посмотреть в требованиях к приложению;
- спросить у менеджера проекта;
- посмотреть в локальной статистике, через какие браузеры пользователи заходят в приложение;
- посмотреть в общей статистике, какие браузеры популярны в соответствующем регионе.

Выбор браузеров по движку

Популярных браузеров может быть довольно много. Чтобы сэкономить время на кроссбраузерное тестирование, можно выбирать браузеры по популярности **движка**.

Браузерный движок — это программа внутри браузера, которая помогает отображать веб-страницы и её элементы.

Некоторые браузеры работают на одном движке: это значит, что логика отображения у них одинаковая. Например, Google Chrome, Яндекс.Браузер, Opera и Microsoft Edge. Для каждого движка можно выбрать самый популярный браузер и проверять только в нём.

Некоторые баги возникают только в одном браузере и не воспроизводятся в других, даже на том же движке. Такие баги называют **специфичными**.

Если всегда проверять только в самом популярном браузере на одном движке, можно не заметить некоторые ошибки: например, баг может воспроизводиться в Яндекс.Браузере, а в Google Chrome — нет.

Версии браузеров

Как и у операционных систем, у браузеров есть версии. В разных версиях одного браузера элементы интерфейса могут по-разному работать или отображаться.

Проверить все версии всех популярных браузеров сложно — их слишком много.

Если есть статистика по приложению, можно посмотреть, какие версии браузеров — самые популярные, и проверить приложение в них.

Если такой статистики нет, можно тестировать в последней версии. Обычно она самая популярная: у большинства пользователей включено автоматическое обновление. Последнюю версию браузера можно скачать на официальном сайте.

Подходы к вёрстке веб-приложения

Версий веб-приложения — всего две, а видов устройств много: компьютеры, ноутбуки, смартфоны, планшеты. Чтобы пользователю было комфортно взаимодействовать с приложением везде, его делают адаптивным.

Адаптивность — это способность приложения подстраиваться под экраны с разными разрешениями.

Разрешение экрана — это размер изображения на экране в пикселях.

Веб-приложение подстраивается под разрешение экрана так, чтобы на экране помещались все элементы и сохранялась доступная функциональность.

Чтобы «подогнать» вёрстку под разные разрешения, применяют один из подходов: **адаптивный** и **респонсивный** дизайн**,** или **адаптив** и **респонсив**. И то, и другое часто просто называют «адаптивным» дизайном.

Адаптивный дизайн

Чтобы адаптироваться под разрешение экрана, приложение использует специально заданные точки — **брейк-поинты** (от англ. break-point — «точка

разрушения», «переломная точка»). Эти точки дизайнер задаёт на макете, а разработчик расставляет в коде. Когда ширина экрана приближается к каждой из них, то дизайн как бы «ломается» и перестраивается под новую ширину.

Респонсивный дизайн

Если в адаптиве есть брейк-поинты, то респонсивный дизайн адаптируется к ширине экрана в любой его точке. Такую вёрстку ещё называют «отзывчивой» или «резиновой» — макет тянется во все стороны без резких скачков в изображении.

Пропорции и размеры элементов в отзывчивом дизайне задают в процентах, а не в пикселях. В итоге все элементы сайта «растягиваются» вслед за изменением экрана — плавно. Поэтому такой метод адаптации и называют резиновым.

Тестирование вёрстки на адаптивность

Тестировщику всегда нужно проверять как вёрстка подстраивается под устройства с разными разрешениями. Это называется **тестирование адаптивности.**

Toggle Device Toolbar

Чтобы проверить вёрстку на разных устройствах, у тестировщики есть тестовые девайсы — смартфоны, планшеты, ПК.

Но моделей и разрешений много, всё физически трудно иметь под рукой. Тогда пригодится инструмент Toggle Device Toolbar. Он встроен в DevTools — имитирует разрешения и ориентацию экрана мобильных устройств.

Ориентация экрана — это положение экрана мобильного устройства относительно пользователя. Ориентация может быть вертикальной и горизонтальной.

На чём тестировать мобильную версию веб-приложения

Тестирование на мобильном устройстве

На реальных девайсах проверяют:

- Кроссплатформенность и кроссбраузерность: веб-приложение должно корректно работать и отображаться во всех поддерживаемых

операционных системах и мобильных браузерах.

- Как на вёрстку влияет встроенная клавиатура: нужно убедиться, что вёрстка не разъезжается, когда появляется клавиатура.
- Некоторые жесты, которые невозможно воспроизвести на компьютере. Например, пинч — сжимающее действие двумя пальцами, оно уменьшает изображение на экране.

Тестирование в DevTools

В DevTools можно проверить только часть особенностей мобильной версии приложения:

- Как вёрстка веб-приложения адаптируется под разные разрешения экрана.
- Как вёрстка ведёт себя, когда меняется ориентация экрана.
- Некоторые жесты. Например, свайп — скольжение пальцем в сторону.

Часто тестировщики комбинируют оба варианта. Сначала проводят проверки на реальном устройстве. В DevTools тестируют вёрстку в тех разрешениях, которые не удалось проверить на девайсе.

Как выбрать разрешения экрана для тестирования адаптивности вёрстки

А в начале пути пригодится специальный алгоритм. Он уже встречался тебе в уроках по кроссплатформенному и кроссбраузерному тестированию.

Изучи, как подобрать разрешение экрана:

1. Посмотри требования и макеты: если там указано, какие разрешения поддерживает приложение, тестируй только в них.
2. Если разрешения не указаны, уточни у менеджера проекта. Часто в команде заранее договариваются, в каких разрешениях тестировать — их выбирают таким образом, чтобы покрывать и большие, и маленькие экраны.
3. Если договорённости нет, попробуй посмотреть локальную статистику — она подскажет, какие разрешения экрана встречаются у пользователей приложения.
4. Если статистики по приложению нет, поищи общую статистику по разрешениям. Так ты поймёшь, какие разрешения самые популярные.

Посмотри на пример такой статистики по ссылке: https://www.hotlog.ru/global/screen.

Как подобрать конфигурацию окружения для тестирования

Конфигурация окружения — это набор параметров. Они характеризуют среду, в которой тестировщик проверяет приложение. Такие параметры — это платформа, ОС и версия ОС, браузер и версия браузера, разрешение.

Матрица поддерживаемых окружений

Чтобы покрыть проверками все параметры окружений, тестировать 48 комбинаций необязательно. Тебе пригодится матрица поддерживаемых окружений.

Матрица поддерживаемых окружений — это таблица, строками которой являются операционные системы, а столбцами — браузеры.

Алгоритм построения матрицы

Тебе пригодится специальный алгоритм:

- Последовательно сопоставь каждый браузер с одной десктопной и одной мобильной ОС. Например, для браузера Chrome можно выделить ОС Windows 10 и Android 9, для Яндекс.Браузера — Windows 7 и Android 10, и так далее.
- Не забудь, что некоторые браузеры работают не со всеми ОС.
- У тебя получится таблица — опирайся на неё, чтобы хотя бы раз проверить каждую ОС и каждый браузер.
- Выдели зелёным цветом ячейки, в которых пересекаются ОС и браузер, — это поддерживаемые окружения.
- Впиши поддерживаемые разрешения в те зелёные ячейки, в которых можно их проверить. Разрешения можно тестировать на реальных устройствах или в DevTools.

Cookie

Cookie (куки, печенье) — данные, которые хранятся на устройстве пользователя. Эти данные приходят с сервера и сохраняются на компьютере

или смартфоне, а когда используешь приложение повторно — снова передаются на сервер.

Представь: ты едешь в отпуск и выбираешь гостиницу в веб-приложении — агрегаторе отелей. Когда открываешь главную страницу агрегатора, браузер не только отправляет запрос на сервер, но и ищет на твоём устройстве куки приложения. Если ты не в первый раз используешь агрегатор отелей и куки для него сохранились, браузер отправит их на сервер. Например, в куках может быть информация, что пользователь авторизован.

Сервер обработает запрос и сформирует ответ с учётом этих данных: например, добавит в ответ список твоих бронирований. Без кук сервер их не пришлёт, потому что идентифицирует тебя как нового посетителя.

Что хранят в куках

В куках можно хранить разные данные:

1. Данные для аутентификации. Например, ты авторизуешься в веб-приложении, затем закрываешь вкладку, а через какое-то время открываешь приложение снова. Второй раз вводить логин и пароль не придётся, потому что данные об аккаунте сохранились в куках.
2. Данные о пользователе. Например, информацию о просмотренных отелях, геолокацию, язык, валюту.
3. Данные для сбора статистики. Например, об устройстве пользователя или просмотренных страницах. На основе статистики команда приложения делает выводы и принимает решения о работе функциональности.

Как просматривать куки

Куки — это небольшие текстовые файлы. Их можно просматривать прямо в браузере.

Local Storage

Local Storage (локалсторадж, локальное хранилище) — хранилище данных, которое встроено в браузер. Оно отличается от кук: данные из Local Storage не передаются на сервер, их использует только клиентская часть приложения.

Например, ты заполняешь форму бронирования отеля: вводишь ФИО, почту, телефон. Если фронтенд приложения сохраняет данные в Local Storage, они не пропадут даже после обновления страницы. Если решишь забронировать

другой отель и снова перейдёшь на форму бронирования — твои ФИО, почта и телефон «подтянутся» из Local Storage. Тогда вводить их заново не придётся.

Как просматривать данные из Local Storage

Данные из Local Storage можно просмотреть в DevTools.

Выбери вкладку Application (Приложение) → найди раздел «Storage» (Хранилище) → раскрой выпадающий список «Local Storage» (Локальное хранилище) → кликни по URL приложения.

Как очистить данные из Local Storage

А теперь представь: тебе нужно проверить, как приложение будет работать для нового пользователя. Он ещё ни разу не выбирал тариф и требования, поэтому данные из Local Storage (Локальное хранилище) предстоит очистить:

1. Перейди в DevTools → найди выпадающий список Local Storage → нажми правой кнопкой мыши по URL приложения.
2. Нажми Clear (Сбросить).

Кэш в тестировании

Кэш (Cache Storage) — это данные веб-страниц. Они сохраняются на компьютере, когда ты открываешь изображения, аудио-, видео-, CSS-, HTML-, JS-файлы.

Cookie и Local Storage упрощают работу с приложением. Пользователю не нужно каждый раз вводить одни и те же данные, например: логин, пароль, номер телефона.

Браузерный кэш нужен для другого. Когда ты откроешь веб-страницу повторно, она загрузится быстрее: браузер возьмёт файлы из кэш-хранилища, а не с сервера.

Иногда причина бага кроется именно в кэше. Представь: ты находишь баг и создаёшь карточку в трекере. Разработчик исправляет ошибку и возвращает задачу в тестирование. Теперь нужно убедиться, что всё в порядке. Ты проверяешь приложение ещё раз, но изменений нет — баг по-прежнему воспроизводится. А разработчик утверждает, что всё починил.

Скорее всего, данные просто закэшировались у тебя в браузере. Поэтому ты видишь старую версию страницы со своего компьютера, а не новую версию с

сервера. Чтобы исправить проблему, нужно почистить кэш. Выполни эти шаги, чтобы потренироваться:

1. Открой браузер Google Chrome или Яндекс.Браузер.
2. В правом верхнем углу разверни меню браузера и перейди в раздел «История».
3. Кликни на кнопку «Очистить историю».
4. Отметь галочкой только один пункт: в Chrome — «Изображения и другие файлы, сохраненные в кеше», в Яндекс.Браузере — «Файлы, сохраненные в кеше».
5. Установи временной диапазон — «За всё время».
6. Нажми «Удалить данные» в Chrome или «Очистить данные» в Яндекс.Браузере.

Если у тебя другой браузер — можешь найти инструкции в интернете.

Иногда после обновления приложения часть файлов загружается с сервера, а часть — из кэш-хранилища. Из-за этого могут появиться проблемы — например, с вёрсткой.

Если заметишь такой баг в продакшене, не торопись чистить кэш — возможно, пользователи столкнутся с такой же проблемой. Сообщи разработчикам, они помогут разобраться.

Charles

Сниффер (от англ. sniff — нюхать), или анализатор трафика — приложение, которое перехватывает данные (трафик), передающиеся по сети.

Анализаторы трафика помогают:

- Разобраться, как работает приложение: ты поймёшь взаимосвязь между действиями пользователя и отправкой запроса.
- Быстрее тестировать: со сниффером тебе не нужно перенастраивать бэкенд, чтобы получить конкретный ответ от него. Можно просто подменить запрос в инструменте и получить нужный на клиенте результат.
- Локализовать ошибку. Сниффер позволяет просматривать запросы и ответы, которые не видны пользователю при обычной работе.

- Тестировать клиент и сервер раздельно. Например, когда клиент уже написан, а бэкенд — ещё нет. Тогда нужно настроить инструмент, чтобы он имитировал ответы сервера.

Распространённый сниффер — Charles.

В нём можно просматривать запросы в режимах Sequence и Structure.

Режим Sequence

В режиме Sequence ресурсы отображаются в хронологическом порядке: от недавнего — к старым. Интерфейс в режиме Sequence делится на два блока. В верхнем можно посмотреть все запросы, а в нижнем — изучить запрос или ответ детально.

Режим Structure

В режиме Structure ресурсы группируются по хостам (host) и путям (path). Это помогает быстрее ориентироваться в данных. Интерфейс Structure отличается от Sequence. Здесь окно делится вертикально: слева — запросы, рассортированные по хостам, справа — детали конкретного ресурса.

Какой режим выбрать

Sequence удобно использовать, если ресурс известен целиком. Его легко найти: достаточно ввести его в поле Filter.

Structure больше подходит для исследования: когда знаешь, с какого хоста должен прийти запрос, но не знаешь, какой конкретно. Тогда можно отслеживать запросы с определённого хоста.

Charles: ручное изменение ответа от бэкенда

В Charles можно перехватывать и подменять запросы или ответы. Это полезно, когда нужно быстро протестировать отображение данных на фронтенде

Как подменить ответ бэкенда и подставить нужный параметр

Чтобы загрузить данные для отображения страницы, браузер делает запросы на сервер. В ответе приходят картинки, тексты, результаты расчётов и многое другое.

Открой DevTools: в ресурсах можно посмотреть, какой запрос отправил фронтенд и какой ответ дал бэкенд. Твоя задача — найти в Charles ресурс,

перехватить ответ и подставить нужное значение. Тогда фронтенд получит уже исправленную версию ответа.

Изменить ответ можно разными способами:

- вручную;
- автоматически.

Breakpoints

Изменить один из параметров ответа ручным способом можно через Breakpoints. Breakpoints — это остановка запроса. Когда его останавливают, можно редактировать содержимое. Чтобы завершить запрос, пользователь должен дать команду продолжить запрос.

Rewrite: автоматическая подмена

Этот инструмент помогает настроить правила, по которым содержимое определённых запросов и ответов будет меняться автоматически.

Укажи точное значение, которое предстоит менять → настрой инструмент Rewrite → настрой Location → задай правила автоматической подмены.

Map Local: автоматическая подмена из файла

Чтобы автоматически подменить ответ на запрос, можно также заменить содержимое прямо из готового файла. С этим поможет функция Map Local. Данные в формате JSON заранее готовят в текстовом файле — ты можешь создать библиотеку файлов для подмены и пользоваться ими. Map Local применяют, если нужно заменить не несколько параметров, как в Rewrite, а весь запрос или ответ.

Таблица принятия решений

Таблица принятия решений (таблица решений, decision table) — техника проектирования тест-кейсов. Комбинации условий из требований можно оформить в виде таблицы — так ты покроешь проверками все сценарии и ничего не упустишь.

Чтобы посчитать количество комбинаций, умножь количество значений всех параметров друг на друга.

Такая таблица нагляднее, чем текстовое описание. Ты сразу поймёшь, что нужно проверить и какого результата ожидать.

Чтобы спроектировать тест-кейсы, следуй простому алгоритму:

- значения параметров из таблицы подставь в шаги тест-кейса,
- ОР из таблицы подставь в ОР кейса.

Попарное тестирование

Попарное тестирование (pairwise testing) — техника тест-дизайна, в которой проверяют все комбинации пар параметров.

Если пара параметров вызвала ошибку в комбинации с другими параметрами, с большой вероятностью та же пара вызовет ошибку и в другой комбинации. Поэтому достаточно протестировать только уникальные пары вариантов.

Из урока про таблицу принятия решений ты знаешь: чтобы посчитать количество всех возможных комбинаций, нужно перемножить значения параметров между собой.

Технику попарного тестирования не используют, когда параметров и их значений мало: нужно построить таблицу, правильно её заполнить и только потом приступать к тестированию. Быстрее сразу проверить все комбинации.

Но когда параметров и их значений много, попарное тестирование сокращает количество проверок в несколько раз.

Pairwise Tools

Чтобы построить таблицу быстрее, используй специальные инструменты. Например, Pairwise Tools. Тебе нужно ввести параметры и их значения, а система просчитает все комбинации.