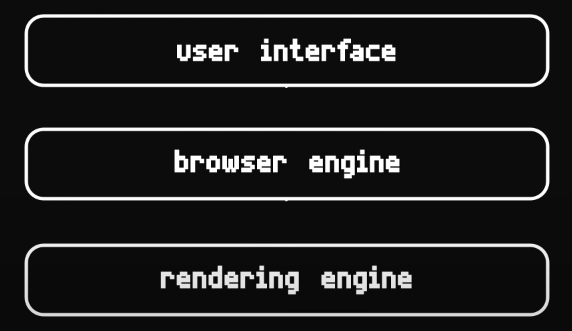
1. **Браузер**…………………………………………………………………………………….
   1. **Архитектура**…………………………………………………………………………..
   2. **Браузерный** **Event** **Loop; основы концепции однопоточного и многопоточного программирования; асинхронная модель**……………………
   3. **Стадии** **рендера. DOM, CSSOM, Render tree, style calculation, layout, paint, composite**………………………………………………………………………………
   4. **Локальное хранение данных**……………………………………………………….
      1. **Local storage**…………………………………………………………………..
      2. **Session storage**…………………………………………………………………
      3. **Browser cache**………………………………………………………………….
      4. **Cookie**………………………………………………………………………….
   5. **Архитектура**

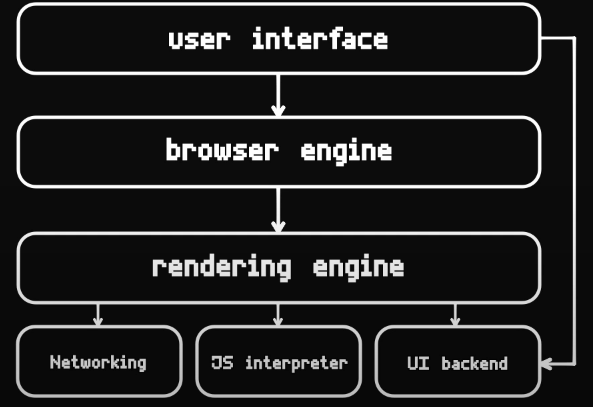
Структурная схема любого браузера



• user interface – вкладки, настройки, навбар, строка URL и т.д.  
• browser engine – соединительная часть между UI и механизмом рендеринга. На основе входных данных UI – browser engine взаимодейсмтвует с механизмом рендеринга и управляет им.

• rendering engine – благодаря ему получаем заветные страницы, приложения с которыми мы можем взаимодействовать. Например WebKit (Chrome) и Gecko (Firefox). Отвечает за обработку HTML, CSS и JS: строится DOM-дерево, строится CSSOM-дерева, определяет расположение элементов, там проходят в общем различные стадии рендера.

В свою очередь rendering engine состоит из компонентов



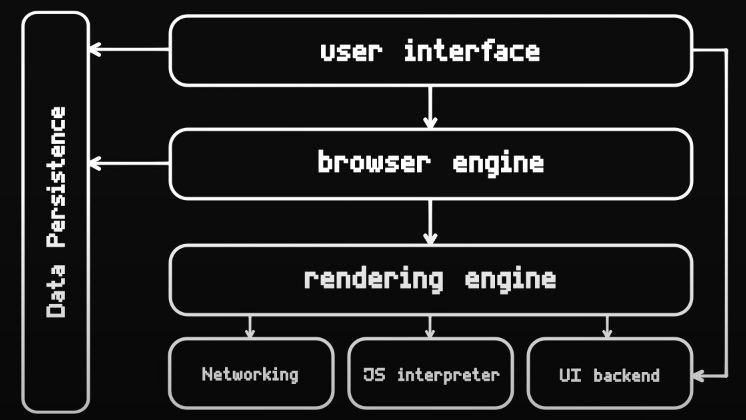
• networking – модуль предназначен для работы с сетью: хранит историю посещаемых сайтов, за доменные имена, за взаимодействие с DNS, работа с HTTP / HTTPS, открытие соединений по протоклам транспортного уровня (UDP / TCP) и т.д.

• js-interpreter – движок обработки JS. Например V8 у Chrome. Поверх этого движка построен Node JS, Google Chrome. Это абстрагированная, не зависимая от платформы технология которая выполняет задачу парсинга JS-кода и его компиляцию в машинный код.

Он также предоставляет heap, call stack, работа с памятью, выделение памяти, сборка мусора.

• ui-backend – логика UI самого браузера.

Есть также хранилище данных. Таким образом выглядит структурная **схема любого браузера**.



Local Storage, session storage, IDB, web SQL, файловая система, например для загрузки картинки в браузер. Также обрабатывает Cookie, хранение вкладок. В общем – локальная база данных.

**Пример**

Имплементация структурной схемы браузеров на примере Google Chrome.



Создать собственный браузер зада вполне реализуемая, поскольку по большей части все браузеры состоят из опенсорсных технологий, важно лишь понимать устройство, структуру браузера и как эти технологии скомпоновать.

**Архитектура движка рендера на примере webkit**

На вход подаётся HTML, CSS. Затем это парсится соответствующими парсерами. На выходе HTML парсера получается DOM дерево, с которым можно взаимождействовать через JS, а на выходе CSS-парсера получается набор стилей. К сгенерированному DOM-дереву применяются получивашееся CSS правила и получается дерево рендера. Это дерево проходит некоторые другие стадии – на выходе получается отрендеренная страница.



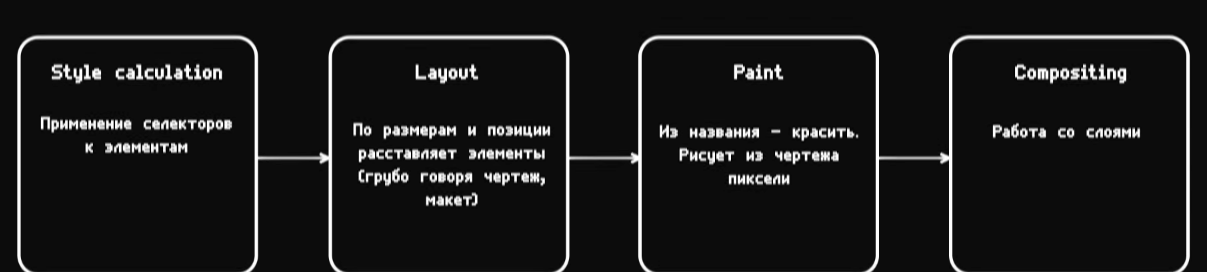
* 1. **Браузерный** **Event** **Loop; основы концепции однопоточного и многопоточного программирования; асинхронная модель**

Event loop – механизм, позволяющий использовать неблокирующую модель ввода/вывода. Решает задачу распараллеливания выполнения кода: например, при отправке запроса на сервер чтобы не повисал весь UI.

Event loop не является частью JS. Пояснение – в хроме используется V8 для обработки JS, в NodeJS также используется V8, но при этом Event Loop в браузере и в Node JS – реализованы по разному, хотя решают одну и ту же задачу.

* 1. **Стадии** **рендера. DOM, CSSOM, Render tree, style calculation, layout, paint, composite**

Как отрисовывается страница, с которой пользователь взаимодействует в браузере. В какой очередности, по каким стадиям этот процесс происходит.



1. Калькуляций стилей. На этом этапе браузер применяет Style Rules к узлам DOM-дерева. Чем сложнее CSS селекторы и чем их больше – тем процесс калькуляции стилей будет дольше и тем больше нагрузка на железо.
2. Layout. На этом этапе браузер знает какие стили к какому элементу должны быть применены, поскольку до этого они были скалькулированы на предыдущем этапе. На этом этапе происходит составление макета, чертежа страницы. На выходе будет Layout-Tree (дерево-макета, дерево-расположения и т.п.). Содержит информацию о том, где элемент должен располагаться, виден он или не виден и т.п.
3. Paint. Применение подсчитанных стилей из первого этапа к макету со второго этапа. Строятся, так называемые, Paint Records.
4. Compositing (композиция). Браузер делит то, что было получено на предыдущем этапе на определенные слои – создает дерево слоев (layer-tree). На этом этапе браузер работает со слоями, задействует для этого видеокарту, например. Располагает элементы в зависимости от z-index и т.д.

DOM-дерево и дерево стилей строится чуток пораньше (как только мы получили HTML и CSS). В этот момент строится DOM, CSSOM и само деревео рендера (Render-tree). Вот таким образом работает всё при первой отрисовке. Вот полная схема:



Но как это работает в случае перерисовок? Например, нажали на кнопку, откроется модальное окно. В таком случае и стили меняются, и отображение элементов на странице меняется и DOM-дерево может меняться из-за модалки.

На самом деле новго ничего не происходит. JS триггерит, например, изменние каких-то стилей, и стади рендера начинаются по новой: перерисоывается **часть** DOM-дерева, обновляется некоторая часть CSSOM, строится новое Render-tree.

**Рендер вызывается:**1) Изменение размера окна

2) изменение шрифта

3) изменение контента

4) Добавление/удаление классов/стилей

5) Манипуляции с DOM

6) Изменение ориентации (альбом/книга)

7) Изменение размеров/позиции

8) Вычисление размеров/позиции

* 1. **Локальное хранение данных**
     1. **Local storage**
     2. **Session storage**
     3. **Browser cache**
     4. **Cookie**

Решают проблему хранения промежуточного состояния между запросами.

**HTTP-куки**

HTTP cookie (web cookie, куки браузера) - это небольшой фрагмент данных, который сервер отправляет браузеру пользователя. Браузер может сохранить этот фрагмент у себя и отправлять на сервер с каждым последующим запросом. Это, в частности, позволяет узнать, с одного ли браузера пришли несколько запросов (например, для аутентификации пользователя). **С помощью кук можно сохранить любую информацию о состоянии**, HTTP-протокол сам по себе этого делать не умеет.

Куки часто используются для:

Управления сеансом (логины, корзины для виртуальных покупок)

Персонализации (пользовательские предпочтения)

Трекинга (отслеживания поведения пользователей)

До недавнего времени куки использовались в качестве хранилища информации на стороне пользователя. Это могло иметь смысл в отсутствии вариантов, но теперь, когда в распоряжении браузеров появились различные API для хранения данных, это уже не так. Из-за того, что куки пересылаются с каждым запросом, они могут ухудшать производительность (особенно при использовании мобильных сетей). В качестве хранилищ данных на стороне пользователя вместо них можно использовать [Web storage API](https://developer.mozilla.org/ru/docs/Web/API/Web_Storage_API) (localStorage и sessionStorage) и [IndexedDB](https://developer.mozilla.org/ru/docs/Web/API/IndexedDB_API).

[**Создание куки**](https://developer.mozilla.org/ru/docs/Web/HTTP/Cookies#%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D1%83%D0%BA%D0%B8)

Получив HTTP-запрос, вместе с ответом сервер может отправить заголовок [Set-Cookie](https://developer.mozilla.org/ru/docs/Web/HTTP/Headers/Set-Cookie). Куки обычно запоминаются браузером и посылаются в HTTP-заголовке [Cookie](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cookie" \o "В настоящее время эта страница доступна только на английском языке) с каждым новым запросом к одному и тому же серверу. Можно задать срок действия кук, а также срок их жизни, после которого куки не будут отправляться. Также можно указать ограничения на путь и домен, то есть указать, в течении какого времени и к какому сайту они будут отсылаться.

**Куки могут быть:**

• сессионные – удаляются при закрытии клиента; существуют на протяжении сеанса (потому что не задаются атрибуты Expires или Max-Age). Иногда в браузере может быть фича автоматического восстановления сеанса – вследствие чего куки могут храниться как если бы браузер и не закрывался;

• постоянные – удаляются уже при наступлении определенной даты (атрибут Expires) или по истечению определенного промежутка времени (атрибут Max-Age).

**HttpOnly куки**

Существуют HttpOnly куки – к ним нельзя получить доступ через document.cookie, что позволяет снизить риск успешной XSS атаки (межсайтовый скриптинг с целью кражи куки).

Область видимости куки задается атрибутами Domain и Path (т. е. определяются те URL адреса, на которые куки могут отсылаться)