Введение

Войти в айти

В 1991 году интернет стал общедоступным, и стали появляться первые провайдеры. В период с 1991 по 1996 годы появились стандарты HTML, CSS, JS.

За период 1990-2010 интернет превращался в платформу для наиболее перспективных направлений бизнеса, а технологические компании по своему влиянию постепенно приближались к компаниям нефтяного сектора, автомобильным концернам или финансовым организациям.

До повсеместного распространения интернета по сути не было “айти” в том понимании, которое мы знаем сегодня, а компании делились на множество отдельных индустрий. В наши дни во многих крупных фирмах есть IT-департаменты, а крупнейшие технологические компании по умолчанию называются айтишными вне зависимости от их профиля. Банки хотят стать айти-компаниями, айти-компании хотят стать банками, а сами айтишники, кажется, уже устали от повышенного внимания к себе, особенно со стороны государства.

“Интернетизация” сделала мир немного другим, став катализатором развития различных технологий. Сегодня для многих айти ассоциируется в первую очередь с какой-либо работой в интернете, или зависящей от него. Конечно же, микроэлектроника и компьютерная техника существовали задолго до эпохи интернета, но это тема для совсем другого разговора.

Раньше в айти входили не по ссылкам на рекламу курсов, а по зову сердца. Типичной историей было начать на коленке стряпать странички для друзей, просто вдохновляясь красивыми примерами, увиденными где-то на просторах тогдашнего интернета “по карточкам”, а потом делая это уже за деньги для первых заказчиков и открывая веб-студии. Примерно так начинал известный всем Артемий Лебедев.

В наши дни пресловутое айти и интернет пронизывают практически все сферы деятельности человечества, начиная от высокотехнологичных сфер, и заканчивая сельским хозяйством. Было бы легче сказать, где его нет. Да и само слово “айти” уже набило оскомину, изрядно прожужжав уши любому, у кого они есть. Множество новых индустрий в разное время появились или стали востребованы на волне развития интернета и цифровых технологий: социальные сети, интернет вещей, машинное обучение, бигдата, блокчейн, кибербезопасность, онлайн-ритейл, смартфоны, облачные сервисы, виртуальная реальность и многое другое.

Автомобили набиты электронными системами, банки работают онлайн, медиа-контент распространяется по воздуху и даже софт для космических аппаратов пишут на тех же технологиях, что и в каком-нибудь интернет-магазине. Многие вещи стало на порядок проще реализовать благодаря высокоуровневым языкам, фреймворкам, другим инструментам и массе готовых строительных блоков для решения задач на любой вкус.

Кодеры, программисты, разработчики и другие представители сферы заняли нишу тех самых инженеров прошлого, которые раньше с помощью паяльников “программировали” аналоговые приборы. Раньше в крупных промышленных центрах основой рабочего класса были слесари у станков, а сегодня это айтишники со своими макбуками и линуксами.

Но как всё вышесказанное относится к вебу и фронтенду в частности?

Фронтенд-разработка по сути произошла из так называемого “сайтостроения” в эпоху Web 2.0 примерно в конце нулевых, когда веб-мастера, выросшие во времена становления индустрии, стали разбиваться по более узким специализациям и веб отчетливо разделился на бекенд, фронтенд и дизайн.

Тогда в бекенде наиболее мейнстримовыми были фреймворки и CMS на PHP, обретавшим в те времена всё больше возможностей для разработки в ООП стиле. А веб-дизайн стал отдельным видом искусства и люди выбирали эту стезю, осознанно отказываясь развиваться в техническую сторону, и двигаясь по пути UI/UX, рисуя макеты в Fireworks и Photoshop. Фронтендеры же пилили свои велосипеды на ванильном JavaScript.

В переходный момент между классической веб-разработкой на условном PHP и новым подходом SPA довольно широко была распространена профессия верстальщика, подразумевающая специалиста, владеющего HTML, CSS и начальными знаниями JS. В дальнейшем верстальщики под влиянием трендов или условий рынка часто переходили в фронтенд или дизайн.

С ростом функциональности веб-сервисов один специалист “мастер на все руки” уже с трудом справлялся с широким спектром задач и расширяющимся стеком технологий. Разработка сайтов всё больше смещалась от монолитных CMS в сторону SPA, что способствовало развитию клиентских фреймворков, таких как jQuery, Backbone, Ext JS и др.

Но все эти инструменты на JavaScript выглядели неряшливо на фоне своих старших братьев из Java, .NET и даже PHP. Долгое время JavaScript со всеми своими детскими болячками не воспринимался всерьёз, и на то были основания. Любой, кто пытался “въехать” в фронтенд-разработку тех лет, отмечал ужасный developer-experience от всех принятых тогда стандартных инструментов, а также низкую пригодность самого JavaScript к применению в “кровавом энтерпрайзе“. Так и прошла первая декада 21-го века.

В 2009 вышел NodeJS и определил вектор развития всего JavaScript мира. Наконец фронтендеры смогли на родном языке писать себе инструментарий, а самые продвинутые стали осваивать фулстек разработку. В 2010 году вышел в свет AngularJS, запустивший новый виток гонки фронтенд-фреймворков и библиотек, которые стали расти как грибы после дождя. Вкупе с популяризацией GitHub эти события стали почвой для развития массы хороших и не очень библиотек, некоторые из которых дошли до наших дней, но в большинстве своем быстро ставших никому не нужными.

Стоит упомянуть также выход первого iPhone в 2007 году, ставшего предвестником повальной миграции всего и вся на мобильные платформы на волне быстрого развития сотовых сетей передачи данных. Еще тогда Стив Джобс пророчил новый мир, где веб-технологии станут основой интернета будущего (привет от PWA!), и, кажется, он не сильно ошибся. С развитием смартфонов тогдашние верстальщики, всё еще страдающие поддержкой Internet Explorer версий 5+, уже начинали делать первые шаги в сторону адаптивной вёрстки под весь зоопарк мобильных браузеров и экранов.

Если в начале 10-х годов фронтенд или JavaScript-разработчик, как правило, был молодым человеком не старше 30, и часто воспринимался окружающими не очень серьезно (подумаешь, формочки красит на пару с дизайнером, да JSON-ы гоняет по сети), то сегодня ввиду разнообразной специфики JavaScript-приложений такое восприятие - моветон.

Не хочу равнять всех под одну гребёнку, и, разумеется, есть в профессии люди разного возраста, но, судя по коллективам, в которых приходилось работать, конференциям и другим мероприятиям, мне кажется, что именно миллениалы стали костяком JavaScript-сообщества, по крайней мере в нашей стране.

Эпоха евангелистов 2010-х

Несмотря на все новомодные по тем временам технологии фронтенд всё еще оставался отсталым по меркам “серьезных” платформ. Крупные корпорации и новое поколение евангелистов веб-технологий под их началом изо всех сил продвигали в массы фронтенд-фреймворки и новые стандарты, привносящие в платформу невиданные ранее возможности. К ним относятся, например, HTML5 и CSS3, позволяющие без всяких Flash-плееров воспроизводить на страницах видео и делать продвинутую анимацию.

Медийные персоны от технологических компаний, например Дэн Абрамов, сами того не ведая, создавали монстров, таких как React/Redux, а их конкуренты еще сильнее разгоняли эту безумную карусель “убийц” jQuery и прочих унылых инструментов вчерашнего дня.

Трудно переоценить уровень парадоксальности всего происходящего тогда в авангарде фронтенд технологий. Вроде бы все по-отдельности хотят создать лучшие инструменты для решения всех проблем, но каждый тянет одеяло на себя, и в итоге имеем по несколько реализаций какого-нибудь Flexbox, десять одинаково безнадежных фреймворков для решения простых проблем сложным способом, и так далее.

Не отставали и браузеры: все эти годы шла ожесточенная борьба между основными игроками на рынке браузерных движков. Internet Exporer быстро терял позиции, рынок занимали Firefox, Chrome, Opera и Safari. Интереснейшие метаморфозы происходили с Opera и Chrome, поменявшими движок прямо на ходу, что в конце концов привело к низвержению первого и воцарению второго. На этом фоне рассказывать про Edge даже не хочется, уж больно комичной выглядит его история “успеха”.

Появление таких ключевых технологий, как NPM, TypeScript, Electron, React, Webpack и некоторых других (пусть многие из них и вторичны) привело к немыслимому проникновению веб-технологий во все сферы: десктопные, мобильные и какие угодно приложения стало удобно и выгодно делать на стеке, изначально придуманном для страничек в интернете. Не стоит забывать и WebAssembly, привносивший еще больше возможностей для запуска почти чего угодно в браузере. Хорошо это или плохо - тема для бесконечного холивара, но одно можно сказать точно - фронтендеры изобрели программирование заново и произвели революцию, которую вроде бы никто не просил.

Если вспомнить историю создания самого JavaScript, многих веб-стандартов и фреймворков, то можно уловить определенный “стартаперский вайб”. Это очень по-фронтендерски начинать с ужасного и полного нестыковок API, годами придумывать для него костыли, и в конце концов героически представлять миру что-то “блейзингли фаст”, будто это ты сам только что придумал.

Короче говоря, у фронтенда свой уникальный путь - заново изобретать велосипеды.

**Кругом JavaScript**

К концу 10-х годов веб-технологии перешагнули невидимую черту, за одно десятилетие совершив эволюцию, которую другие платформы проходили в разы дольше, так и не приблизившись к успеху JavaScript и его друзей. Под успехом, разумеется, я имею в виду не совокупную пользу для человечества, а хайп вокруг технологии и степень ее востребованности, пусть и нарочито раздутой.

Взгляните на любой актуальный роадмап развития фронтенд-разработчика: HTML и CSS там занимает процентов 10, а остальное поделили мастодонты React, Vue и Angular, а также всевозможные инструменты в их экосистеме, либо на их основе. А если фронтендеру становится скучно, то он открывает для себя мир бекенд-разработки на Node.js.

Айтишные курсы и инфоцыгане в основном учат, как работать с фреймворками, и начисто отбивают у людей желание разбираться с основами. Отсюда и обоснованные стереотипы, мол фронтендеры поголовно “формошлёпы” и только красят свои кнопки, да распивают смузи. Конечно, даже в реалиях России 2023 года подобное всё еще может иметь место, но на мой взгляд осталось лишь в воспоминаниях старожилов, которые сами эти легенды и распространяют.

Зачем всё делать на веб-технологиях?

Не хочется в очередной раз перечислять пресловутые мессенджеры, социальные сети, мобильные приложения и редакторы кода, созданные на веб-технологиях. Стоит лишь понимать, что на JavaScript в наше время можно написать практически всё. И этот факт нам намекает, что это самое “всё” будет делаться на JavaScript всё с большей скоростью.

Что “нового” придумали фронтендеры? Для этого давайте вспомним былые подвиги апологетов веб-технологий.

**React против Angular**

Помните историю React vs Angular? В незапамятные времена до 20-х годов нам отовсюду рассказывали, что React - это тот самый глоток свежего воздуха и решение всех проблем фронтенда после ужасного универсального комбайна, коим нарекли Angular.

React - это всего лишь библиотека для рендеринга, говорили они. React очень прост в освоении в отличие от Angular, говорили они. Делайте как мы, и всё у вас будет летать, говорили они.

Что имеем сегодня? Писать что-то, отвечающее современным стандартам, на React без TypeScript не представляется возможным. Просто написать клиентские компоненты уже недостаточно, нужны изоморфные. Изучение всевозможных хуков и асинхронных API уже далеко не сопоставимо с низким порогом входа. Написать приложение, не подключив целый зоопарк дополнительных библиотек - нереально.

Что там c Angular? Был похожий на бекендерский фреймворк на все случаи жизни для задач любой сложности. Было трудно разбираться с зонами, инъекцией зависимостей и модулями. Что имеем сегодня? Зоны, модули и прочие сложные концепции признаны неоправданно сложными и готовятся на выход. Вместо них предлагаются более понятные паттерны, подсмотренные в других инструментах.

Какая мораль? Писать типизированный код изначально было хорошей идеей - кто бы мог подумать! Подсаживать мир на иглу “простоты” было плохой идеей. Героически побеждать в борьбе за производительность, давая в руки новичкам неполноценные инструменты без внятного руководства - абсурдно, но достойно восхищения.

*Если вы хотите собрать новое приложение или сайт полностью на React, мы рекомендуем выбрать один из React-powered фреймворков, популярных у сообщества.*(с) Документация [React](https://react.dev/learn/start-a-new-react-project)

<https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/>

**SPA против SSR**

“Как же классно перенести всю работу с сервера на клиент!” Подумали фронтендеры и ударились изобретать для этого клиентские фреймворки. И в итоге за несколько лет пришли к раздутым бандлам, ожиревшему DOM-дереву и лагающему интерфейсу на простейших приложениях. SEO-оптимизация страдала, пользователи жаловались на требовательность веб-приложений к батарее и ужасные тормоза при загрузке и обычном скролле.

Как же стали решать эти проблемы? “А давайте производить рендеринг частично на сервере!” Подумали фронтендеры и стали изобретать фреймворки для фреймворков. Теперь у нас есть JavaScript на сервере, уйма сомнительных кеширующих механизмов и размазанная по всему стеку бизнес-логика. Браво, мы изобрели PHP.

Create-react-app постепенно уходит в прошлое, и теперь знаний обычного React вам хватит, чтобы написать сайт-визитку для кота. А если нужно что-то большее, то извольте брать за основу Next.js, Remix, Expo или какой-то еще комбайн. С одной стороны вам всё так же дают выбор, какие технологии тащить в проект, но с другой стороны теперь придется выбирать не между вспомогательными библиотеками, а между платформами, с которых уже просто так не спрыгнуть. Хотя раньше такая возможность была одним из достоинств React.

Риторический вопрос: помните, чем в своё время вдохновлялись разработчики React?

Долго и упорно фронтендеры шли по граблям, придумывая всё новые инструменты для решения проблем, которые клином сошлись на пресловутой производительности. Корпорация Google день за днём закручивала гайки, повышая требования к оптимизации скорости загрузки и рендеринга сайтов. И все под эту дудку были вынуждены плясать.

**Electron против десктопов**

Первые приложения на этой платформе вызвали массу негодования со стороны программистов старой школы, и небезосновательно. При этом публикой они были приняты вполне позитивно, что и убедило многие компании не только вложиться в разработку новых приложений на Electron, но и переписывать на нем старые.

Этот “убийца” нативных приложений клал на лопатки не только всех своих конкурентов, таких как GTK+, Qt и JavaFX, но и достаточно мощные ПК своих нерадивых пользователей. Прожорливость без лишних премудростей объяснялась наличием полноценного браузера под капотом, что и делало Electron настолько же могущественным, насколько подверженным всем проблемам веб-технологий.

К 2023 году появилось много похожих инструментов для кроссплатформенной разработки на веб-стеке, а крупные компании научились готовить их так, чтобы память больше не текла, а размер дистрибутива исчислялся десятками мегабайт вместо сотен. Сюда же можно отнести и технологию PWA, т.к. по большому счету она тоже призвана заместить мобильные и десктопные приложения.

В итоге, если раньше у десктопного приложения была веб-версия, то теперь наоборот - у веб-приложения есть десктопная версия, а еще под мобилки, smart-TV и прочие радости.

**React Native против мобилок**

Как и в истории с Electron, однажды появился великий и ужасный “убийца” нативных мобильных приложений, хотя и без этого многие нативные приложения давно использовали WebView в качестве дешевого варианта отобразить то, что на мобильных платформах делать сложно и бессмысленно.

Поначалу у React Native было мало возможностей интеграции с ОС, но со временем эти проблемы решались. Множество его аналогов стали также заполонять рынок и делать нативную разработку неоправданно дорогой для владельцев бизнеса. Судя по трендам последних лет интерес к ним не собирается падать. Один только [Project Fugu](https://developer.chrome.com/fugu-showcase/) чего стоит.

Кто только не ругал приложения, написанные для мобилок грязными ручками фронтендеров. Но ничего, втянулись.

**TypeScript против всех**

Пожалуй, главным убийцей всех убийц стал TypeScript.

От стадии “писать типы это сложно и бессмысленно” до стадии “как можно писать не типизированный код” проходит примерно пару дней, после чего дороги назад нет. Несмотря на все сложности внедрения типизации, польза окупается в долгосрочной перспективе, что на своей шкуре ощущают как разработчики приложений, так и их конечные пользователи.

Всё, что раньше компилировалось в JavaScript, довольно быстро потеряло актуальность, уступив место новому стандарту. TypeScript пытается быть той версией JavaScript, которую хотели бы иметь прямо в браузере визионеры веб-технологий и простые разработчики.

От появления в 2012 году до сдвига парадигмы в головах у JavaScript-сообщества прошло совсем немного времени по меркам технологий. Теперь фронтендеры с гордостью заявляют, что пишут компилируемый код, совсем как взрослые.

За период 10-х годов веб-технологии хладнокровно выдавили с рынка многих игроков. В итоге теперь мы имеем массу возможностей для того, чтобы для коммерческой разработки вообще забыть о каких-либо альтернативах и всецело отдаться в объятия безумных JavaScript фреймворков на любой вкус и цвет.

**Причины популярности**

В чем заключается успех всех этих PWA, Electron, React Native, если они так плохи? Для ответа на этот вопрос нам стоит разобраться в причинах успеха веб-технологий в целом.

**Универсальность**

Веб всегда был открытым - стандарты в его основе разрабатываются совместно и по принципам опенсорса. Промолчим про то, какие именно компании в массе своей проталкивают новые стандарты. Так или иначе все заинтересованные лица осознают важность веб-стандартов для интернета и человечества в общем. Доступ в интернет в наше время считается чуть ли не священным правом каждого современного человека, и эта идея помогает распространению как самого интернета, так и средств доступа в него.

Браузер или браузерный движок можно обнаружить практически где угодно, и он будет работать на самом медленном железе при условии наличия выхода в сеть, конечно. Бортовые компьютеры, одноплатники, умные холодильники, телевизоры и кнопочные телефоны - все эти устройства скорее всего имеют под капотом браузер с поддержкой стандартов как минимум 10-летней давности, а значит вы с большой вероятностью сможете худо-бедно посетить нужный вам сайт и что-то там увидеть.

Просто откройте вкладку и вбейте нужный адрес, либо поисковый запрос. Ничего не надо устанавливать и обновлять. Всё необходимое загрузится на лету и будет работать (конечно, в зависимости от прямоты рук веб-разработчиков), подстраиваясь под возможности вашей ОС и браузера. Мы об этом даже не задумываемся. Мобильным разработчикам такое только снится.

**Обратная совместимость**

В отличие от любых других платформ всё, что когда-либо заработало в браузере, будет продолжать работать в нём вечно. Прямо сейчас вы запросто можете открыть какую-нибудь раритетную страничку из 90-х, и она будет работать практически так же, как 30 лет назад. Задумайтесь - JS, HTML и CSS код, пролежавший на сервере и не обновлявшийся столько лет, просто работает.

Нет проблем обратной совместимости, нет проблем горячей замены кода на сервере прямо в момент, когда в нему обращаются посетители. В совокупности со стандартами HTTP получается абсолютно бесплатное и гибкое решение проблем доставки продукта до конечного пользователя. Помешать вашим планам может только РКН.

Дешевизна

Тут всё просто - вы можете быстро найти одного единственного специалиста за 5 копеек (относительно стоимости отдельных специалистов под несколько платформ, конечно), который сварганит вам худо-бедно работающее приложение и завернет его во все необходимые оболочки. Если ограничиться только веб-версией, то даже не надо заставлять пользователей ничего устанавливать. Вы даже сможете выйти с этим творением на рынок, а дальше сами понимаете…

В последнее время можно наблюдать во многих сферах некий регресс, откат на шаг назад, в том числе и технологическом плане. Все вокруг стали больше ценить решения, менее подверженные санкциям и другим угрозам. Забавно читать новости такого плана:



Фрагмент текста с сайта [CNews](https://www.cnews.ru/news/top/2023-08-11_rossii_srochno_trebuyutsya)

https://www.cnews.ru/news/top/2023-08-11\_rossii\_srochno\_trebuyutsya

Получается, бизнес на своей шкуре ощущает нецелесообразность инвестиций в нестабильные платформы и на их место ждёт дешевую замену в виде джунов веб-мастеров, прямо как в благословенные нулевые, когда трава была зеленее!

Но почему это дёшево?

Низкий порог входа

В целом даже сегодня вам не нужно знать ничего, чтобы постепенно “вкатиться в айти” на правах фронтендера. Ну подумаешь, не изучал информатику, алгоритмы и основы сетей. Зато по туториалу с Youtube можешь в блокноте слепить работающий сайтик, который даже не стыдно впарить потенциальному инвестору. Опять же курсы “вайтишников” создают нездоровый интерес вокруг сферы и рисуют в головах обывателей безоблачный мир, где айтишникам платят за стук пальцами по клавиатуре с важным видом.

Программирование в целом становится более казуальным. Можно это явление сравнить с ситуацией в игровой индустрии. Считается, что раньше игры были хардкорнее, и игроки соответственно. А в современные игры стало играть слишком легко, просто жми левую кнопку мыши и побеждай, что очень удобно для превращения игр в донатные помойки. Но хардкорные игры никуда не делись, просто теперь мейнстримом стали однокнопочные дофаминовые генераторы.

Так же и с технологиями - в лидерах сейчас максимально высокоуровневые языки для соответствующих задач. Рынок постоянно подстраивается под запрос общества, языки эволюционируют, программисты осваивают именно то, что сейчас востребовано. А для более глубокого погружения требуются годы, которые из кармана не вынешь.

Среда исполнения

Node.js, браузеры, облака, устройства интернета вещей, блокчейны, смартфоны и еще немало интересных мест, где можно исполнять JavaScript, делают язык одним из наиболее востребованных в мире. Построение приложений на базе Chromium и/или Node.js стало нормальной практикой. Браузерные расширения также являются интересным вариантом замены десктопных приложений или обхода ограничений доступа к заблокированным ресурсам.

Не менее прекрасно то, что в случае с веб-приложениями вам ничегошеньки не надо собирать, компилировать, транспилировать и т.д., если вам лень. Подключаете к вашему index.html готовые библиотеки и рядом пару строк со Stackoverflow, чтобы это всё запустить. Вам даже не нужно, чтобы этот HTML был полностью валидным. Казалось бы, это же халтура, и так не будет делать уважающий себя инженер. Но я вас уверяю, так действительно делают, и это действительно просто работает. Абсолютно любой компьютер из коробки готов для разработки фронтенда.

Приятный UI из коробки

Веб-технологии изначально развивались ради возможности делать удобный и красивый пользовательский интерфейс. Поэтому здесь доступен наиболее широкий спектр инструментов и готовых решений по сравнению с другими платформами. А пользователям, как известно, подавай красивые кнопочки и чтобы МЕГАЛО.

Новые API привносят всё больше возможностей делать веб-приложения ближе к нативным, чтобы пользователь не смог отличить их. Доступ к файловой системе, шаринг контента, уведомления, буфер обмена, перехват и чтение файлов через свой протокол и многие другие привычные фичи могут работать не хуже, чем на iOS или Android, хотя в зависимости от платформы реализация может отличаться.

Мобильные и десктопные ОС всё больше становятся похожи, как по функциональности, так и по внешнему виду. Это очень на руку веб-технологиям, потому что они изначально развиваются в парадигме максимальной независимости от окружения и из коробки готовы для адаптации под любые типы устройств и способов взаимодействия с контентом.

Свобода выбора

Хочешь, делай SVG-анимации. Хочешь, рендерь на WebGL и пиши шейдеры. Хочешь, упарывайся в WebAssembly и портируй что угодно в чертов браузер. Или бери какой-нибудь из тысяч бойлерплейт-проектов, и просто подставляй свои данные.

Количество вариантов решения абстрактной задачи на веб-технологиях обычно позволяет выбирать между вариантами “фигак-фигак и в продакшен”, “установлю #name#.js и оно сделает всё само” или “я у мамы инженер, напишу свой фреймворк”, помимо скучных опций почитать документацию где-нибудь на MDN и включить рациональное мышление.

Есть, конечно, один исполинский малюсенький минус во всех этих свободах: делать нормально могут не только лишь все, мало кто может это делать (с). Если серьезно, то порой и правда хочется иметь тот самый единственный способ, как реализовать какой-нибудь кастомный select или ленивую загрузку данных, но каждый раз приходится идти на компромисс между производительностью, расширяемостью, удобством и прочими нефункциональными свойствами отдельно взятого решения.

**Итого**

Мы редко замечаем, когда приложения становятся быстрее, но при этом четко отмечаем падение производительности, такая у нас природа. Приложения 30 лет назад по субъективным ощущениям могли быть быстрее, но отнюдь не за счет идеальной оптимизации, а скорее за счет меньших объемов данных и других характеристик контента, который мы в этих приложениях обрабатываем.

Да, скорость интернета выросла и мощь процессоров тоже. Но и наши требования тоже постоянно растут, в том числе к разрешению экранов, функциональности различных сервисов, плавности анимаций, безопасности передачи данных и прочим вещам, требующим избыточных ресурсов, чтобы угодить взыскательному вкусу пользователей.

Периодически производительность софта не успевает за новыми требованиями, поэтому мы замечаем просадки частоты кадров или скорости загрузки. Потом незаметно для всех ситуация выравнивается, когда разработчики софта оптимизируют приложения. И снова выходит очередной стандарт какого-нибудь кодека, повышающий требования к ширине канала, объему памяти и другим параметрам, а потребители крайними выставляют разработчиков софта. И так по кругу.

Веб-технологии шагают семимильными шагами вперед, а ваши ожидания от них, как говорится - это ваши проблемы.

Все перечисленные факторы делают веб весьма привлекательным для бизнеса, хоть это и не очень нравится борцам за мифическую компилируемость или строгую статическую типизацию.

Почему админка для управления вашим роутером не сделана приложением под все платформы? Потому что бизнесу кратно дешевле сделать ее на коленке, чтобы вы так же на коленке открыли однажды эту админку в браузере и забыли её как страшный сон.

Зачем делать мессенджер под все платформы, запихивая в него браузер и лишая лампового шарма нативных приложений? Получается, что это просто дешевле. Зарплаты у JavaScript-программистов меньше, чем у тех же джавистов, да и специалистов больше. Красивый UI на каком-нибудь C++ сделать сложно, да и не для этого он. JavaScript-фреймворки душат все остальные платформы для дешевой разработки UI, и нанимать специалистов на других стеках - скорее всего дурная затея.

Ну а пользоваться консолью или текстовым UI могут себе позволить не все, мягко говоря. Поэтому рыночек порешал так, что народу нужен GUI на веб-технологиях.

Но это всё жалкие оправдания. Я, конечно же, **не** буду апеллировать к опыту “бигтехов”, делающих PWA в довесок к нативным приложениям, написанным также на веб-технологиях. И **не** буду вспоминать, какой язык находится в верхушке всех мыслимых рейтингов.

**Мораль**

Перестаньте жаловаться на ненастоящих программистов фронтендеров. Они вам написали половину всего, чем вы пользуетесь в повседневной жизни, пусть вы этого и не просили. Мир безумен, жизнь боль, а вы прекрасны.

Если вы всё еще сомневаетесь в нормальности происходящего, на такой случай давным-давно придумали умный совет на все случаи жизни: не можешь победить - возглавь.