Глава 1
ФРОНТЕНД-РАЗРАБОТКА

1.1. С чего начиналась фронтенд-разработка

1.1.1. Осознание способов решения проблем

Сегодня фронтенд-разработчик располагает целым набором инструментов, а также библиотек, которые используются для разработки приложений, выполняемых в браузере. Плюс в самих браузерах появилось довольно много функциональных возможностей. Но так было не всегда, и данная область — разработка фронтенд-приложений — формировалась постепенно.

На заре становления интернета не существовало понятия «фронтенд-приложение». Был термин, вошедший в лексикон, — «клиент-серверное[1] приложение». Чуть позже, когда мы рассмотрим исторические стадии развития функциональности, станет понятно, что такое фронтенд-приложение. Инструменты для его разработки тоже появились со временем.

В эволюции приложений можно выделить три крупных этапа:

- приложения на основе веб-страниц;
- приложения на основе AJAX и библиотек;
- приложения, создаваемые с помощью инструментов и декларативных библиотек.

1.1.2. Приложения на основе страниц

Самые первые приложения — это клиент-серверные приложения, которые назывались сайтами[2]. По сравнению со страницами в браузере они были настолько простыми, что скорее представляли собой электронные книги и каталоги. Да, они были функциональными, но функциональность обеспечивалась не за счет

---

[1] Клиент-серверное приложение — распределенное приложение, основанное на модели вычислений, в которой клиент запрашивает услуги у другой сущности — сервера. В типичном для бизнес-систем клиент-серверном приложении клиент выполняется на персональном компьютере, а расположенный на удаленной и более производительной машине сервер предоставляет ему услуги по доступу к хранящимся на сервере данным. Клиентская часть приложения обычно оптимизируется для взаимодействия с пользователем, в то время как серверная часть предоставляет функциональность, совместно используемую многими пользователями.

[2] Сайт — любой ресурс, который отображается в браузере как документ, а не как код или изображения и появляется в ответ на адрес, набранный в адресной строке браузера. Сайт может быть простым или дополнительно обмениваться данными с сервером.

задействования языка JavaScript и работы с DOM[1], а за счет приложения, которое функционировало на сервере. DOM в том виде, в каком мы знаем ее сегодня, появилась в результате постепенного развития. Движение с одной страницы сайта на другую осуществлялось путем перехода по ссылке. Предыдущая страница выгружалась из браузера, а новая загружалась. То есть происходила полная перезагрузка страниц. Да, использовалась оптимизация запрашиваемых ресурсов — некоторые из них загружались из кэша, но то была оптимизация трафика для сервера. На клиенте все время происходила перезагрузка. В то время клиент-серверное приложение в виде сайта старались разукрасить или сделать как можно более функциональным за счет серверного приложения.

1.1.3. Приложения на основе AJAX и библиотек

AJAX не конкретная технология, а набор технологий, каждая из которых развивается сама по себе.

Чтобы не перечислять все, что предлагалось для использования в AJAX, и не превращать объяснение в доклад о технологиях прошлого, я дам простое определение. Можно сказать, что AJAX подразумевает использование JavaScript в браузере, чтобы обмениваться с сервером данными в формате XML с помощью объекта браузера XMLHttpRequest.

Название AJAX — это первые буквы слов в словосочетании Asynchronous JavaScript And XML, что переводится как «асинхронный JavaScript и XML».

Авторы некоторых книг по JavaScript негодуют из-за присутствия XML и в названии XMLHttpRequest, и в AJAX. Однако появление именно XML вполне оправданно. Какой еще способ представления данных можно было выбрать в начале 2000-х годов? Напомню, что формат JSON Дуглас Крокфорд, можно сказать, «придумал» в 2002 году.

Приложения, которые задействовали подход AJAX, уже не перегружались при каждом взаимодействии пользователя со страницей. Это создавало положительный пользовательский опыт и в целом увеличивало скорость работы клиент-серверного приложения за счет уменьшения трафика[2]. Однако с точки зрения фронтенд-приложения сами приложения, за редким исключением, были основаны на прямой работе с DOM и CSS. DOM браузеров на заре становления интернета не была реализована по единым стандартам, как мы можем видеть это сегодня. И даже сегодня наблюдаются

---

[1] DOM (Document Object Model) — объектная модель документа. Есть целый раздел, посвященный этой модели, где она рассматривается подробно. Сейчас дам общее представление: объектная модель документа — это модель, взаимодействие с которой позволяет менять содержимое документа, отображаемого в браузере. Браузер дает API для изменений, а чтобы разработчики понимали, что менять, документ представляется моделью в виде структуры дерева, чьими элементами являются объекты, созданные на основе тегов из запрошенного с сервера HTML-кода. Таким образом, воздействуя на модель через API, разработчик влияет на то, что отображается в браузере.

[2] Трафик — перемещение, поток данных в передающей среде, например поток сообщений (пакетов передаваемых данных) в Глобальной сети. Также это означает загруженность сети (по аналогии с автотранспортом на дорогах). Трафик состоит из передаваемых данных и служебной информации, необходимой для организации их прохождения.

расхождения со стандартом W3C[1]. А в 2006 году появление библиотеки jQuery осуществило прорыв в скорости создания веб-приложений. Библиотека предлагала множество методов для работы с DOM и AJAX, но главное — разработчикам больше не надо было беспокоиться о разности в реализации DOM в браузерах.

Чуть раньше, в 2005 году, появилась библиотека Prototype.js, которая, пусть и в гораздо меньшей степени, предлагала функции для работы с AJAX и DOM. Начало активному развитию веб-приложений положили 2005–2006 годы, развитие библиотек и инструментов началось чуть позднее. Нас в первую очередь интересуют приложения на стороне фронта. Они в тот момент напрямую манипулировали DOM, CSS и данными. То есть фронтенд-приложение представляло собой реакцию на события, а также прямую манипуляцию DOM и данными. Можно сказать, что после накопления опыта создания таких фронтенд-приложений разработка перешла на новый уровень.

1.1.4. Фронтенд-приложения

Эпоха[2], которую я обозначил как «фронтенд-приложения», началась в 2013 году с появления библиотеки React.js и длится по сей день. Уже есть первые признаки ее завершения. Свое ви́дение данного процесса я изложил в следующем подразделе, а пока вернемся к началу эпохи.

Факторов[3], повлиявших на переход к новой эпохе, четыре. Первый — технологии и подходы, второй — инструменты, третий — библиотеки, четвертый — пользовательский опыт и обратная связь. Каждый из них внес свой вклад в определенный промежуток. В качестве точки отсчета, начала эпохи я выбрал момент появления библиотеки React.js. Взгляните на рис. 1.1.

На инструменты (точнее, на увеличение их количества) сильно повлияло возникновение Node.js. Инструменты позволяют ускорять разработку фронт-приложений плюс дают удобство. Сегодня мы используем их повсеместно. Чуть позже появился репозиторий npm, ускоривший разработку в разы за счет того, что теперь устанавливать зависимости можно одной командой в консоли.

Развитие технологий тоже началось с Node.js. На рис. 1.1 видно, что с 2009 по 2015 год произошел их бурный рост. Эти технологии используются до сих пор и продолжают развиваться. Стоит отметить: несмотря на то что JavaScript уже существовал, в 2015 году стандарт ECMAScript был пересмотрен и произошло качественное изменение языка. В него добавили новые элементы, например, для асинхронного написания программ, а именно — Promise.

---

[1] W3C — World Wide Web Concorcium. Слово «консорциум» означает объединение компаний и частных лиц на основе общего соглашения для осуществления какого-либо мероприятия. Консорциум W3C — это международное сообщество, где члены-организации, постоянные сотрудники и другие люди, участвующие в деятельности, вместе разрабатывают стандарты веба. Директором консорциума является Тим Бернерс Ли.

[2] Эпоха — продолжительный период времени, имеющий какие-либо характерные особенности.

[3] Фактор — существенное обстоятельство, способствующее какому-либо процессу, явлению.

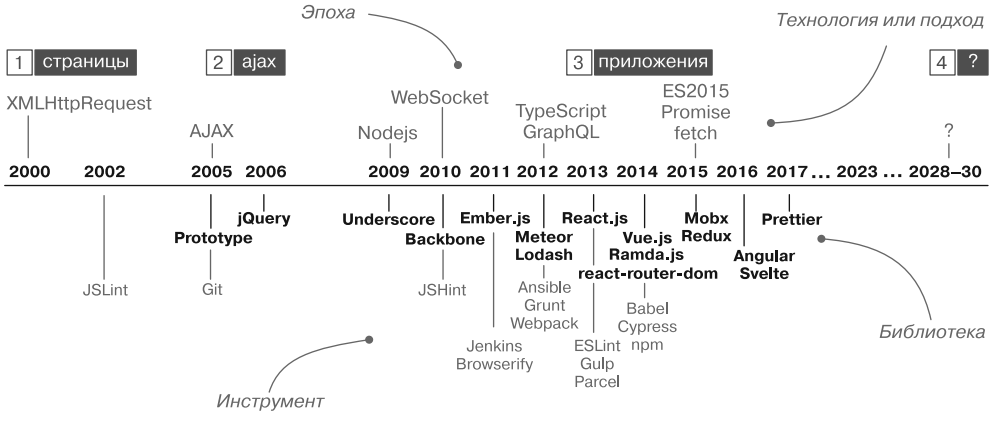

Рис. 1.1. Эпохи разработки, технологий, инструментов и библиотек

Тогда же появились новые библиотеки и стали очевидны недостатки в производительности прямой работы с DOM-моделью. Для лендинга нормально манипулировать DOM и данными напрямую, но не для высоконагруженных сервисов. Компания Facebook[1] к 2013 году разработала библиотеку React.js, которая отличалась от существовавших на тот момент тем, что отделяла создание интерфейса от его воплощения в DOM браузера. Такой подход значительно повысил производительность приложения. Одновременно разработчики выделили среди других компонентов строительный элемент приложения, ускорив саму разработку. Самое интересное, что теперь приложение, работающее в браузере, действительно можно было назвать приложением. Извиняюсь за тавтологию[2], но именно с этой библиотеки, по моему мнению, начался рост веб-приложений как явления. Отныне приложение не манипулировало DOM напрямую. Разработчик описывал, как должен выглядеть интерфейс, с помощью данных и компонентов, а библиотека React в совокупности с библиотекой ReactDOM вычисляла посредством виртуальной DOM[3], что нужно изменить в реальной DOM, чтобы получить предписываемое компонентами. То есть разработчику больше не требовалось манипулировать DOM (императивный подход), достаточно описать состояния и внешний вид (декларативный подход).

Конечно, на рынок стали выходить конкуренты, например Vue.js, Angular. Сообщество, использовавшее React, накопило соответствующий опыт, и стало

---

[1] Компания Meta, которой принадлежит Facebook, признана экстремистской, и ее деятельность запрещена на территории Российской Федерации.

[2] Тавтология — повторение того же самого другими словами без уточнения смысла.

[3] Виртуальная DOM — это структура данных в виде дерева, содержащая точно такие же связи между узлами, как те, что должны быть в реальной DOM в самом браузере. Каждый узел в виртуальной DOM содержит данные, используемые в узлах реальной DOM. Следует отметить, что реальные DOM, выстраиваемые браузером, имеют гораздо больше свойств в каждом узле, чем виртуальные. Главные данные одного узла виртуальной DOM — это связи с другими узлами (отношения «родительский — дочерний»); данные, которые должны быть отображены в виде текста на странице, и данные о стилях.

очевидно, что управлять состоянием всего приложения нужно иным образом, нежели с применением пропсов и стейтов. Так появились библиотеки Redux и Mobx, которые можно использовать совместно с React. Чуть позднее в React (версия 16.3 от 2018 года) появился Context, позволявший не передавать данные вручную в явном виде, через все дерево компонентов от родителя к нужному дочернему компоненту.

Для более предсказуемого управления целыми поддеревьями компонентов была разработана библиотека React-Router-Dom.

Описываемая эпоха продолжает развиваться по сей день. Многие из упомянутых выше библиотек сильно выросли и определили современный стек технологий, которыми пользуются для создания веб-приложений или SPA[1]. В книге описан исключительно современный стек и даже некоторые более передовые сервисы.

Теперь посмотрим, что нас ждет в ближайшем будущем.

1.1.5. Эпоха искусственного интеллекта[2] и виртуальной реальности

На рис. 1.1 я поставил знак вопроса, чтобы вы озадачились тем, что нас действительно ждет. Развитие идет по нескольким направлениям:

- технологии;
- инструменты;
- библиотеки.

Инструменты и библиотеки упрощают использование технологий и вводят дополнительные абстракции для более легкого моделирования работы с данными. В технологиях это также платформы, для которых разрабатываются приложения. На языке информатики это исполнитель: изменится платформа — изменятся и инструмент, и библиотека, и подход к разработке, а также появится другая технология, которую надо будет изучить и использовать. Сейчас активно распространяется браузер. Данная тенденция сохранится на какое-то время, пока не появится другое приложение, вероятно, в совокупности с новым типом девайса[3]. Все тот же, правда уже вчерашний, Facebook, а сегодня — Meta продвигает идею метавселенной, и ведутся диалоги о создании единого стандарта для метавселенных, что правильно. Ведь есть опыт по разработке браузеров, каждый из которых идет своим путем. Единый стандарт позволит сэкономить время и быстрее развиваться.

А как насчет искусственного интеллекта (ИИ)? Что он нам даст? Все просто — ИИ ускорит разработку. Уже сегодня доступны инструменты, позволяющие на

---

[1] SPA (Single Page Application) — приложение, исключающее загрузку страниц при переходе по ссылке. SPA загружается в браузер один раз, а в дальнейшем взаимодействие пользователя обрабатывается так, что хоть и создается видимость смены внешнего вида страниц, но все происходит без загрузки новой HTML-страницы.

[2] Искусственный интеллект — синтетическая система, демонстрирующая «умное» поведение. Определение взято из книги: *Харбанс Р.* Грокаем алгоритмы искусственного интеллекта. — СПб.: Питер, 2023.

[3] Девайс — физическое устройство, позволяющее выполнять определенную функцию. В данном случае будет воспроизводить изображения и, вероятно, даже имитировать тактильные ощущения.

основе промпта[1] получить готовый код. Таковыми являются GigaChat[2] и GigaCode[3] от «Сбера». В ближайшее время можно ожидать появления других инструментов, значительно ускоряющих написание кода и, вероятно, даже создающих полное приложение на основе небольшого количества входных данных. Позволить себе такие крупные разработки может или крупная корпорация, такая как «Сбер», Google, Microsoft, Oracle, SAS и т. п., или группа увлеченных энтузиастов, придумавшая какую-то идею и на основе open source — решений, каковыми являются языковые модели GPT, RuGPT[4], выпустившая продукт, покрывающий потребность в автоматизации, которая ранее была доступна только специалистам. Пока браузеры не будут сдавать свои позиции в силу распространенности, но создание приложений значительно ускорится, а вслед за этим изменится инфраструктура, на которой они работают. Браузеров станет недостаточно. Посудите сами. У вас в руках — инструменты, работающие на основе нейросетей и способные написать типовой код, включая внесение необходимых изменений. В качестве «железа» есть компьютер с огромными ресурсами, а вы по-прежнему пользуетесь браузером, способным разве что воспроизводить картинки и видео. Конечно, нет! Возможно, браузер — как идея, а не как программа — перейдет в девайсы для виртуальной реальности и станет 3D с тактильным интерфейсом.

От чего зависит появление новых технологий? Только от нас с вами — чем больше вы вкладываетесь в разработку новых технологий, тем быстрее они приходят на рынок. Действует прямая закономерность: если вы направляете внимание лишь на изучение библиотек и технологий, то и получаете знания библиотек и технологий, а еще можете создавать продукты на их основе. Если же вы ориентируетесь на разработку и придумывание чего-то нового, что до вас никто не делал, ну разве что высказывал идеи, то и получаете то, что может стать «новым стандартом», как это было с jQuery, React, Redux, React Router, с языком TypeScript. Поэтому надо действовать не только в использовании имеющегося, но и в придумывании нового.

Стоит ли переживать по поводу того, что сегодня из каждого утюга кричат про языковые модели, искусственный интеллект и т. п.? Нет. Приложения не состоят из одного текста и никогда только из него состоять не будут. Они состоят из гипертекста

---

[1] Промпт — запрос к языковой модели, на который предполагается ответ, имеющий смысл для человека.

[2] GigaChat — диалоговая ИИ-модель, которая отвечает на вопросы, сочиняет тексты, пишет код и рисует картинки. Говорит на русском и понимает английский.

[3] GigaCode — ИИ-ассистент, который позволяет генерировать код на разных языках с применением ИИ, выполнять автозаполнение и создавать запросы.

[4] GPT (Generative Pretrained Transformer) — генеративный, предварительно обученный трансформер. Трансформер — это нейронная сеть, обрабатывающая всю последовательность текста сразу, одновременно выявляя взаимосвязи между словами, а не слово за словом. Нейронная сеть представляет собой модель, которая, по мнению исследователей моделей и мира, лежит в основе разума (речь о мыслительной деятельности, которая будто бы происходит в мозгу, состоящем из нейронов) человека и которую можно перенести в сферу обработки данных. Таким образом, создается впечатление, что программа, использующая нейронную сеть, обладает знаниями и способна к разумным ответам. Сеть — это элементы, содержащие связи с другими элементами, по которым могут передаваться данные.

и медиа. Приложения должны иметь структуру, в них воплощаются замыслы людей, а не машин. Без фронтенд-разработчиков это неосуществимо. ИИ ускорит вашу работу. Однако запросов от населения и бизнеса относительно приложений меньше не станет, их количество будет увеличиваться. Растет население, растет осознание и качество жизни, растет потребность. Сами посудите, как бы вы сегодня работали на прошловековых технологиях создания приложений без интернета? Да никак! И работы все прибавляется. Ровно то же самое вас ждет в текущий момент времени. Так что давайте погружаться в область фронтенд-разработки.

1.2. Разработка фронтенд-приложения

1.2.1. Окружение разработки

Знание JavaScript открывает возможность для изучения фронтенд-разработки, где он активно задействуется. Однако написание кода, состоящего из функций и UI-компонентов, не единственное обязательное действие. Разработка приложения включает в себя несколько этапов и шагов в них, состоит из индивидуальной и групповой работы. Конечно, если вы работаете одни, то группового этапа нет. В любом случае вы ведете поиск готовых решений, конструируете компоненты и функции, пишете тесты, проверяете работоспособность и синтаксическую корректность кода. Если приходите в готовый проект, а так происходит чаще всего, вы, кроме того, знакомитесь с кодом. При этом подавляющее большинство проектов осуществляется в группе, поэтому необходимо делать код-ревью изменений, которые вносят в репозиторий другие разработчики. Если вы создаете приложение с нуля, то необходимо его спроектировать и, прежде чем начать писать код, сделать настройки для сборки приложения и проверки корректности синтаксиса.

По мере роста кодовой базы нужно создавать документацию, где описываются соглашения и идеи, закладываемые в основу приложения. И конечно, одно из первых действий в начале любой работы — определить библиотеки и фреймворки, которые будут использоваться, то есть техстек.

Помимо написания или изменения кода, следует научиться пользоваться готовыми инструментами для разработки. Посмотрим, какие области нужно контролировать фронтенд-разработчику и какие существуют инструменты (рис. 1.2).

В прямоугольнике показаны фронтенд-приложение и библиотеки, имеющие к нему прямое отношение. В овалах черным шрифтом прописаны области, с которыми нужно иметь дело при разработке, а бледно-серым — инструменты, часто применяемые в данной области.

Трудно отрицать очевидные вещи, например то, что разработка начинается в какой-то момент и продолжается, продолжается, продолжается, но однажды останавливается. Момент начала важен потому, что перед тем, как начать писать код, нужно сделать настройку будущего окружения. На рис. 1.2 это выглядит именно как ближний круг приложения. Сегодня фронтенд-разработчику надо умело применять и библиотеки, и весь набор инструментов для разработки. Умелое применение начинается со знакомства, понимания и обучения, затем следует успешное применение.

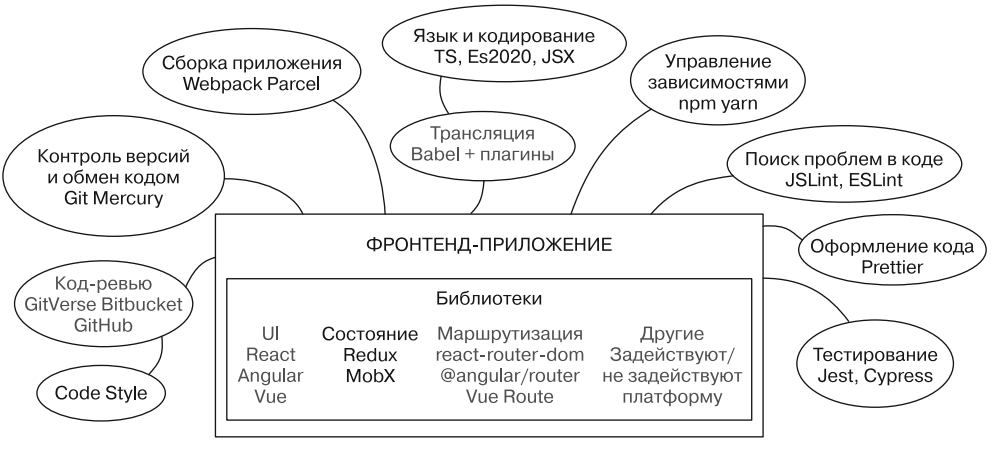

Рис. 1.2. Области, которые нужно контролировать фронтенд-разработчику

Начнем с обзора каждой области. Инструменты, за исключением системы контроля версий и сервисов для код-ревью, работают на node.js. Системы контроля версий работают непосредственно в операционной системе, а сервисы для код-ревью и хранения репозитория[1] приложения доступны в интернете.

1.2.2. Управление зависимостями

Зависимость — это модуль[2]*, обычно называемый пакетом, содержащий выражения на языках JavaScript, TypeScript и импортируемый для использования в программе.* Зависимость может быть любого размера и выполнять одну и более функций или вычислений. Такой модуль, в свою очередь, тоже может иметь зависимости. Пояснения даны на рис. 1.3.

Чтобы не запутаться в словах, надо понимать разницу между реестром и репозиторием. Реестр — просто список чего-либо. Репозиторий — место, где что-либо хранится. Применительно к npm реестр содержит оформленные модули; нам, как пользователям, доступен список всех хранящихся там модулей. Когда пишем код

[1] Репозиторий — место хранения кода в файлах, откуда его можно получить или где можно разместить, а также дополнить или удалить.

[2] Модуль (в программировании) — оформленный специальным образом функционально законченный и самостоятельный (в том числе по отношению к компиляции или загрузке) блок кода, взаимодействие с которым осуществляется через его внешний интерфейс. Раздельная трансляция (separate compilation) модулей важна при создании больших систем. кроме того, каждый отдельный модуль может повторно использоваться в других проектах. Разбиение программы на модули, модульность (modularity), существенно облегчает ее понимание, разработку, документирование, отладку, модификацию и сопровождение. В зависимости от контекста синонимами этого термина являются unit, programm unit и package. Источник: *Пройдаков Э. М., Теплицкий Л. А.* Большой англо-русский толковый словарь по вычислительной технике и информационным технологиям. — М.: РТСофт, 2015.

и с помощью системы контроля версий размещаем его в месте хранения (репозитории), у нас нет специального списка модулей, как в npm, и мы не оформляем каждый JavaScript-модуль как npm-модуль.

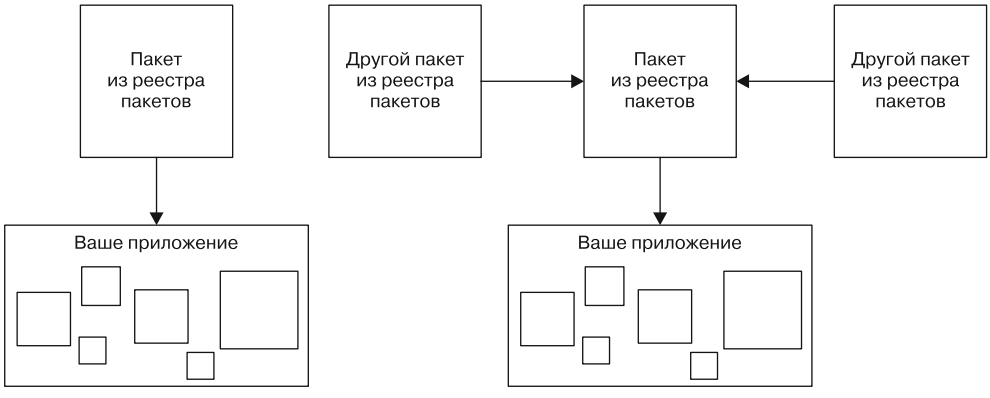

Рис. 1.3. Зависимости приложения

Для управления зависимостями используются программы npm, yarn и др.; это — инструмент. Стоит отметить, что он предназначен для работы с внешними зависимостями — это видно на рис. 1.3.

1.2.3. Язык и кодирование. Трансляция

Под трансляцией понимается два вида процессов. Первый — процесс преобразования исходного кода программы непосредственно в машинный код плюс некоторые другие процессы, что позволяет запускаться программе на компьютере. Например, вы написали программу на алгоритмическом языке C и, используя компилятор gcc, создали исполняемый файл. Второй — процесс перевода программы, написанной на одном языке, в код программы на другом языке. Также ко второму процессу относится перевод программы, написанной на одной версии языка, в код программы, написанный на другой версии того же языка. Самое популярное действие — перевод последней версии языка JavaScript, например версию JavaSript 2024 в версию JavaScript 5.1.

1.2.4. Сборка приложения

Сборка приложения, или упаковывание, — это подготовка кода приложения для его выполнения в браузере и подготовка ресурсов для использования в приложении. Под ресурсами понимаются изображения (PNG, JPEG, GIF и др.), CSS-файлы, аудио и видео (если они располагаются у вас и не управляются особым образом).

Для данной процедуры используется инструмент под названием «сборщик», например webpack.