ГЛАВА 1

Основы API

Термин *API* в английском языке расшифровывается как *application programming interface* и означает программный интерфейс приложения. В этой книге речь пойдет о сетевых API. В главе 1 мы представим историю API и дадим краткий обзор всех видов API, которые рассмотрим в книге. Далее сравним их по критериям, определяющим стиль API.

Кроме того, мы изучим концепции, общие для всех сетевых API, и цели создания API, в том числе их использование в качестве продуктов и монетизацию. Мы разберем все этапы жизненного цикла разработки API, а также принципы управления и администрирования. Понимание этих этапов поможет вам выбрать API, соответствующий вашим потребностям.

Прежде чем перейти к конкретным видам API, обсудим ключевые понятия и историю создания таких интерфейсов.

Что такое API

В термине *application programming interface* слово *interface* означает точку взаимодействия, где сущности обмениваются данными. Сущностью может быть пользователь, система или организация.

На рис. 1.1 показан пример взаимодействия между сущностями. На электронной микрофотографии игла видно, как взаимодействует с поверхностью виниловой пластинки. Дорожки — это физическое воплощение звуковой волны, выгравированной на виниле. Сила воздействует на иглу и удерживает ее в дорожке, а при вращении пластинки мы слышим результат этого взаимодействия.

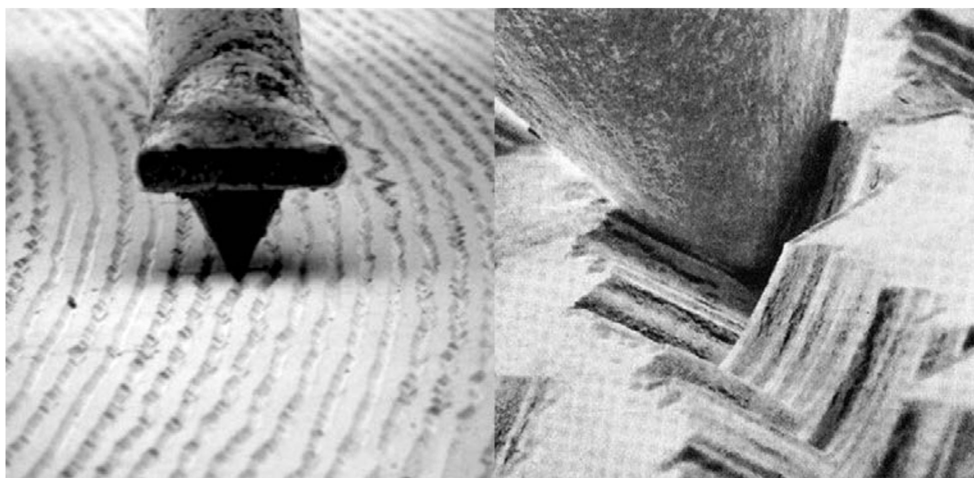


Рис. 1.1. Электронная микрофотография иглы и виниловой пластинки[1]

Вот другие примеры интерфейсов, с которыми мы взаимодействуем ежедневно.

Устройство для взаимодействия человека с компьютерной системой (human interface device, HID)

Устройство, позволяющее человеку взаимодействовать с другим устройством и управлять им, например пульт от телевизора или компьютерная мышь.

Интерфейс командной строки (command-line interface, CLI)

Интерфейс, получающий команды от пользователя в виде текстовых строк и передающий их в компьютерную систему.

Графический пользовательский интерфейс (graphical user interface, GUI)

Интерфейс, позволяющий пользователям взаимодействовать с компьютерными системами через графические элементы.

Чат-интерфейс

Интерфейс приложения для общения через чат.

Интерфейс «мозг — компьютер» (brain-computer interface, BCI)

Интерфейс, обеспечивающий прямое взаимодействие между мозгом и внешним устройством. Известен также как интерфейс «мозг — машина».

---

[1] Изображение взято из поста Electron Microscope Slow-Motion Video of Vinyl LP («Виниловая пластинка: замедленное видео с электронного микроскопа») в блоге Бена Краснова (Ben Krasnow, https://oreil.ly/uxWIh).

API встречаются в разных компьютерных системах.

Библиотеки или фреймворки

Java Platform, Standard Edition и Java Development Kit (JDK, https://oreil.ly/aMYLg) — пример библиотеки, которая предоставляет API для взаимодействия с операционной системой, сетью и другими компонентами; Flask API (https://oreil.ly/OiLFi) — пример веб-фреймворка на Python.

Операционные системы

Portable Operating System Interface (POSIX) — набор стандартов, разработанных IEEE Computer Society (IEEE CS). POSIX унифицирует API для обеспечения совместимости между операционными системами (Unix-подобными и др.).

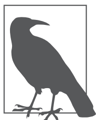
Сетевые приложения

API этой категории предназначены для приложений, взаимодействующих по компьютерной сети. Типичные интерфейсы здесь — *веб-API* (web API), предоставляемые веб-сервером и используемые браузером. Для взаимодействия с сервером браузер чаще всего задействует веб-API JavaScript[1]. Однако есть приложения с сетевыми API, которые работают и без браузера, например инструменты командной строки, мобильные и настольные приложения.

Таким образом, API используются в разных типах компьютерных систем. Они предоставляют конкретные функции, чтобы пользователям не приходилось реализовывать их с нуля, и скрывают сложность системы за интерфейсом. Для пользователя API — что-то вроде «черного ящика». Следуя документации, мы подаем данные на вход и получаем результат на выходе (иногда вместе с «побочными продуктами» взаимодействия).

Сетевые API

Сетевые API, которым посвящена книга, — это точки взаимодействия, позволяющие программам обмениваться данными по сети (обычно через интернет).



Большинство API, описанных в книге, относятся к категории веб-API. API основаны на веб-технологиях — браузерах и протоколах наподобие HTTP (подробнее об HTTP см. в главе 4). Однако термин «веб-API» обозначает лишь те API, которые построены на веб-архитектуре и веб-технологиях, поэтому мы будем пользоваться более широким термином — «сетевые API». Каждый раз, когда в книге встречается термин API, речь идет именно о сетевых API, если не указано иное.

[1] Спецификация веб-API JavaScript доступна на https://oreil.ly/LVCGP.

Теперь разберем аббревиатуру API в контексте сетевых приложений. На рис. 1.2 показаны компоненты стандартного веб-приложения с архитектурой «клиент — сервер»[1].

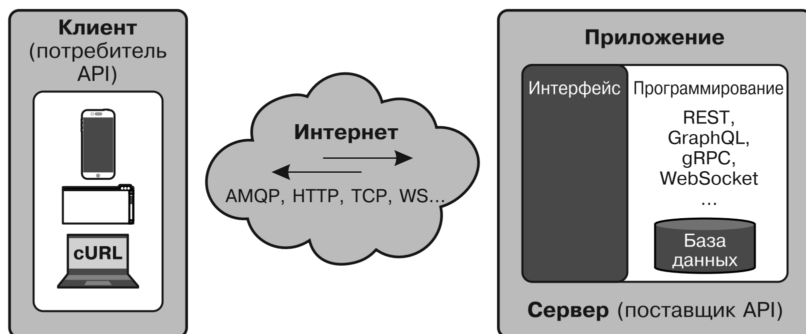


Рис. 1.2. Сетевые API

Проанализируем слова, из которых складывается само название — API.

Приложение (application)

Хотя слово «приложение» на рис. 1.2 появляется только на стороне сервера, оно есть и на стороне клиента. Серверное приложение — это «черный ящик», скрывающий детали реализации. Мы знаем, что входные данные попадают в него, подвергаются преобразованиям (скрытым от внешнего наблюдателя), а на выходе мы получаем желаемый результат. В контексте веб-API серверное приложение называют *бэкендом* (backend), а клиентское — *фронтендом* (frontend). Бэкенд предоставляет данные, а фронтенд их потребляет.

Программирование (programming)

Программирование — это передача компьютеру инструкций для выполнения задач. Как правило, функциональность серверного приложения программируется клиентским приложением. Поэтому программирование происходит и на стороне клиента, и на стороне сервера, хотя на рис. 1.2 термин появляется только на стороне сервера. Программирование (реализацию) функциональности на фронтенде и бэкенде выполняет разработчик, используя соответствующие инструменты и библиотеки.

Интерфейс (interface)

Интерфейс позволяет системам взаимодействовать и обмениваться данными. Это точка взаимодействия, где клиентское приложение начинает использовать функциональность, предоставленную серверным приложением.

[1] Определения терминов «клиент» и «сервер» даны в разделе «Socket API» главы 3.

Наглядно представить сетевой API поможет такая аналогия: многополосный платный мост, соединяющий два города, по которому едут грузовики (рис. 1.3).

Рис. 1.3. Визуальная аналогия сетевых API[1]

Города по обе стороны моста представляют собой клиентское и серверное приложения. Мост, соединяющий города, — это сеть, по которой передаются данные.

Пункт оплаты, через который грузовик въезжает на мост, — стиль API. Грузовик с грузом (сообщением), въезжающий на мост или покидающий его, — это запрос или ответ. Адрес на грузе соответствует эндпоинту API. Грузовик и его груз проходят проверку в пункте оплаты: это гарантирует, что запрос и ответ корректно отформатированы и авторизованы. После проверки грузовик въезжает на мост и двигается в соответствии с правилами дорожного движения (они определяются сетевыми протоколами, о которых речь пойдет в главе 3).

Эта аналогия подойдет также для описания синхронного и асинхронного типов связи (см. подраздел «Синхронная и асинхронная коммуникация» далее в этой главе). В качестве примера рассмотрим процесс доставки груза. При синхронной доставке сообщений грузовик доставляет груз и ожидает квитанцию

---

[1] Изображение сгенерировано моделью DALL·E от OpenAI.

о получении (подтверждение того, что груз был получен). При асинхронной доставке грузовик привозит груз и уезжает. Квитанция о получении отправляется позже на другом грузовике.

Взаимодействие через API: ключевые понятия

Прежде чем переходить к собственно сетевым API, рассмотрим понятия, связанные с обменом данными по сети: сообщение, режимы передачи, а также синхронный и асинхронный типы связи.

Сообщение

Сообщение — это дискретная единица данных (то есть самостоятельная запись), которую отправитель передает одному или нескольким получателям. Оно обычно содержит полезную нагрузку (payload) и метаданные.

В модели взаимодействия открытых систем (Open Systems Interconnection, OSI) термин «сообщение» связывают с протоколом прикладного уровня (см. раздел «TCP/IP и модель OSI» главы 3). Рассмотрим несколько примеров сообщений прикладного уровня.

HTTP

В примере 1.1 показано стандартное сообщение HTTP/1.1 `POST` (подробнее об HTTP см. в главе 4). Оно содержит строку запроса `POST`, за которой следуют три заголовка и тело сообщения (`comment=Hello`).

Пример 1.1. HTTP-сообщение, передаваемое запросом POST

```
POST /submit HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

comment=Hello
```

XMPP

В примере 1.2 приводится сообщение в формате XML для протокола XMPP (Extensible Messaging and Presence Protocol) (https://oreil.ly/dCqrv), которое входит в XMPP-поток. Обратите внимание, как XML-атрибуты используются для описания метаданных сообщения.

Пример 1.2. Сообщение в XMPP-потоке

```
<?xml version='1.0'?>
<stream:stream ...>
  <message from='sender@example.com' to='receiver@example.net' xml:lang='en'>
```

```
    <body>Hello</body>
  </message>
</stream:stream>
```

MQTT

В примере 1.3 показано двоичное сообщение, отправленное по протоколу MQTT (Message Queuing Telemetry Transport) (https://oreil.ly/5K2et). В MQTT сообщение называется пакетом и состоит из фиксированного заголовка, необязательного переменного заголовка и полезной нагрузки. Сообщение в примере — это управляющий пакет (команда `PUBLISH`). Переменный заголовок содержит имя топика `sensor/temperature`, а полезная нагрузка — данные `25.5 C`.

Пример 1.3. Структура пакета PUBLISH в MQTT

```
Fixed Header:     PUBLISH ...
Variable Header:  sensor/temperature
Payload:          25.5 C
```

Режимы передачи

Термин «режим передачи» (transmission mode или data transmission mode) пришел из области телекоммуникаций и описывает, как взаимодействующие стороны обмениваются данными. На рис. 1.4 показаны три режима, которые используются для передачи данных между взаимодействующими сервисами или устройствами. Каждая сторона может находиться в одном из трех состояний: *передача*, *прием* или *передача* и *прием* одновременно.

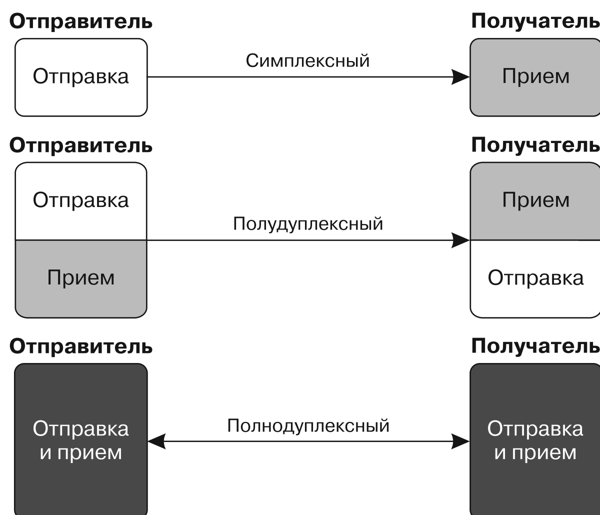
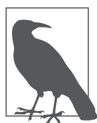


Рис. 1.4. Режимы передачи



Режимы передачи есть не только в API. В любой системе, обменивающейся информацией, — будь то аппаратное устройство или программный компонент — реализуется подходящий режим передачи данных.

Рассмотрим различные режимы передачи.

Симплексный (simplex)

Односторонний режим передачи. Данные передаются только в одном направлении — от отправителя к получателю. Например, радио и телевизор работают в симплексном режиме: данные передаются от устройства к зрителю или слушателю.

Полудуплексный (half-duplex/semi-duplex)

Двусторонний режим передачи, в котором данные передаются в обоих направлениях попеременно: от отправителя к получателю. Пример устройства с полудуплексным режимом — рация: при общении обе стороны соблюдают протокол связи (набор команд наподобие «прием» и «конец связи»).

Полнодуплексный (full-duplex/duplex)

Двусторонний режим передачи, при котором данные передаются одновременно в обоих направлениях[1]. Пример устройства, работающего в таком режиме, — телефон: при разговоре мы можем одновременно говорить и слышать собеседника.

Синхронная и асинхронная коммуникация

Термин «синхронный» (synchronous) происходит от греческих слов *syn* — «вместе» и *chronos* — «время». Поэтому *синхронная коммуникация*, в отличие от *асинхронной*, предполагает синхронизацию по времени. Например, в телекоммуникациях при синхронной передаче данных используется общий тактовый сигнал. Он гарантирует, что получатель может принимать и правильно интерпретировать сигнал с частотой, заданной отправителем. Из этого следует, что понятия синхронной и асинхронной коммуникации применимы не только к модели «запрос — ответ».

К синхронной коммуникации относятся телефонные и видеозвонки, где участники взаимодействуют одновременно. Электронная почта и сообщения на форумах — это примеры асинхронной коммуникации, при которой пользователи отправляют и получают сообщения в разное время. При асинхронной коммуникации участники после отправки или получения сообщения могут

[1] Одновременная двунаправленная передача данных также может осуществляться в двойственно-симплексном режиме. В этом режиме используется два симплексных канала связи, в отличие от полнодуплексного режима, в котором используется только один канал.

выполнять другие задачи. В технических терминах это означает, что они могут выполнять *неблокирующую обработку* (nonblocking processing). На рис. 1.5 показано различие между синхронной и асинхронной коммуникацией на примере модели взаимодействия «запрос — ответ».

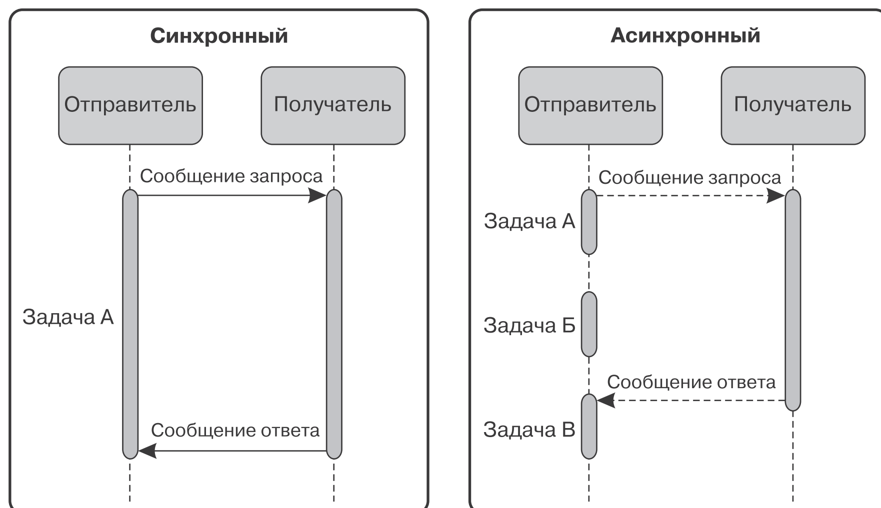


Рис. 1.5. Сравнение синхронного и асинхронного типов коммуникации

При синхронной коммуникации отправитель передает сообщение получателю и остается в заблокированном состоянии до получения ответа. Без ответа отправитель не может продолжить работу или начать выполнять новую задачу. Успешная синхронная коммуникация требует, чтобы обе стороны были доступны и активны во время обмена сообщениями.

При асинхронной коммуникации отправитель передает сообщение получателю и, ожидая ответа, может выполнять другие задачи. В этом случае отправителю и получателю не нужно быть доступными одновременно при обмене сообщениями.

Сетевой протокол может определять, как будет действовать отправитель — синхронно или асинхронно. Если же протокол не содержит таких требований, то отправитель сам решает, какой стиль коммуникации использовать. Например, в HTTP применяется модель «запрос — ответ», где каждому запросу соответствует определенный ответ, поэтому можно предположить, что по умолчанию клиенты, использующие HTTP, общаются синхронно. Однако клиент JavaScript, отправляющий HTTP-запрос с помощью метода `fetch()`, взаимодействует асинхронно, а клиент `curl`, отправляющий такой же HTTP-запрос, — синхронно.

История API

Концепция API появилась задолго до соответствующего термина. Сам термин был введен в 1940-х годах, однако широкое распространение получил лишь в 1960–1970-х[1].

В 1940-х годах британские специалисты в области компьютерных наук Морис Уилкс (Maurice Wilkes), Дэвид Уилер (David Wheeler) и Стэнли Гилл (Stanley Gill) работали над модульной программной библиотекой для одного из первых компьютеров — EDSAC (Electronic Delay Storage Automatic Calculator). Тогда же был создан документ — первая книга по программированию[2], — в котором прозвучала идея использовать подпрограммы в качестве стандартизированных интерфейсов к вычислительным машинам. Понятие «подпрограмма» соотносимо с современным понятием API.

> Чтобы пользоваться подпрограммой, не нужно точно знать, как она устроена и на каких вычислительных процессах основана. Главное условие ее использования — наличие краткой спецификации ее функциональности и способа применения... Чтобы стать частью библиотеки, подпрограмма должна быть разработана в достаточно общей форме, что позволит применять ее в разных контекстах без необходимости внесения внутренних изменений.
>
> *The Preparation of Programs for an Electronic Digital Computer*
> («Подготовка программ для электронного цифрового компьютера», 1957)

На рис. 1.6, *вверху*, — компьютер EDSAC, состоящий из стоек для хранения данных и блоков ввода-вывода. В прямоугольнике справа — перфолента с программой, вставленная в считыватель. Считыватель подключен к телетайпу для вывода результатов. На нижнем изображении оператор (программист) с помощью специальной клавиатуры вбивает команду в перфоленту. Для копирования стандартных подпрограмм из библиотеки используется считыватель (в центре изображения), который управляет клавиатурой и позволяет воспроизводить команды на новой ленте. В прямоугольнике слева — стальной шкаф с библиотекой лент, на которых были вбиты подпрограммы.

Термин *application program interface* без суффикса *-ing* в слове *program* впервые использовали Айра У. Коттон (Ira W. Cotton) и Фрэнк С. Грейторекс (Frank S. Greatorex) в работе *Data Structures and Techniques for Remote Computer*

---

[1] Истории API посвящен доклад Джошуа Блоха (Joshua Bloch) A Brief, Opinionated History of the API («Краткая история API: субъективный взгляд») (https://oreil.ly/hLiqt), представленный на конференции QCon в Нью-Йорке в 2018 году.

[2] *Wilkes M. V., Wheeler D. J., Gill S.* The Preparation of Programs for an Electronic Digital Computer. — Addison-Wesley Press, 1951/1957. https://oreil.ly/49YYe.

Graphics[1] (1968). Термин обозначал взаимодействие между приложением и компьютерной системой. При этом авторы уточняют, что система является аппаратно независимой и может адаптироваться к разным типам устройств при сохранении неизменного программного интерфейса приложения.

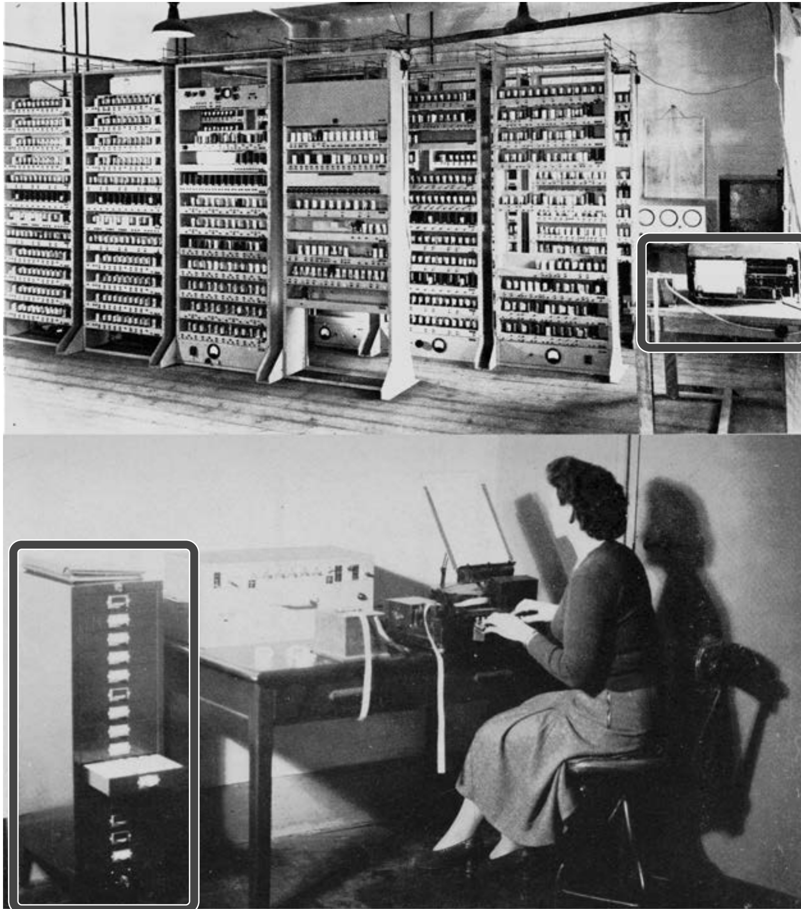


Рис. 1.6. Компьютер EDSAC *(вверху)* и оператор, использующий клавиатуру для перфорации программы на ленту *(внизу)* (1950-е годы)[2]

---

[1] *Cotton I. W., Greatorex F. S.* Data Structures and Techniques for Remote Computer Graphics («Техники и структуры данных в удаленной графической обработке») (https://oreil.ly/k4fek): материалы конференции AFIPS '68: Fall Joint Computer Conference, 9–11 декабря 1968 года.

[2] Описано в книге The Preparation of Programs for an Electronic Digital Computer (https://oreil.ly/Iw6Do).

По мере распространения компьютерных сетей в 1970-х и 1980-х годах программисты начали обращаться к библиотекам не только на локальных компьютерах, но и на удаленных. Понятие API расширилось с появлением *удаленного вызова процедур* (Remote Procedure Call, RPC) (подробнее см. в разделе «Удаленный вызов процедур» главы 8).

В 1990-х годах компьютеры продолжали работать в разных средах, операционных системах и на разных языках программирования. Тогда же консорциум Object Management Group (OMG) (https://www.omg.org) определил стандарт Common Object Request Broker Architecture (CORBA) (https://www.corba.org). CORBA не зависел от языка программирования и позволял удаленно вызывать объекты и выполнять над ними операции, несмотря на их распределенность по разным системам. Связи CORBA обеспечивали межъязыковое взаимодействие объектов, а сами объекты были описаны на языке определения интерфейсов (Interface Definition Language, IDL) (https://oreil.ly/Akrew).

В 1993 году Microsoft создала *модель компонентных объектов* (Component Object Model, COM) (https://oreil.ly/PizNK) для решения аналогичных задач, но в двоичных программах с использованием *двоичного интерфейса приложений* (Application Binary Interface, ABI) (https://oreil.ly/qm08x). В отличие от API, определяющих доступ к программам на уровне исходного (высокоуровневого) кода, ABI зависел от оборудования. Он определял, как обращаться к подпрограммам (функциям) и структурам данных на уровне машинного (низкоуровневого) кода.

В 1998 году был создан протокол SOAP (Simple Object Access Protocol) (https://oreil.ly/h5EGt) для обмена XML-сообщениями между веб-сервисами. В нем использовались протоколы прикладного уровня: в основном HTTP (Hypertext Transfer Protocol) и реже SMTP (Simple Mail Transfer Protocol)[1]. Протокол критиковали за «многословность» (слишком большие сообщения), медленный парсинг (использовались не все возможности HTTP), отсутствие стандартной модели взаимодействия между клиентом и сервером.

В 2000 году Рой Томас Филдинг (Roy Thomas Fielding) представил докторскую диссертацию *Architectural Styles and the Design of Network-Based Software Architectures* («Архитектурные стили и проектирование сетевых программных архитектур») (https://oreil.ly/O1rcQ). Он описал архитектурный стиль REST (Representational State Transfer) и сформулировал концепцию сетевого API.

---

[1] Согласно отчету Postman о рынке API за 2023 год, SOAP занимает четвертое место в рейтинге самых используемых API (https://oreil.ly/knMeq).

API-МАНДАТ

Архитектурный стиль REST, представленный Филдингом в 2000 году, способствовал внедрению сетевых API в публичном интернете. Вскоре после этого был выпущен документ, известный как API-мандат, который показал, что сетевые API распространились и в корпоративной среде.

В 2002 году генеральный директор Amazon Джефф Безос (Jeff Bezos) выпустил меморандум для сотрудников (API-мандат) (https://oreil.ly/YlZ7C). Мандат относился в первую очередь к сервисно ориентированной архитектуре (SOA). На современном языке его основные принципы звучат так.

1. Данные и функциональность сервисов предоставляются через API.
2. Сервисы взаимодействуют только через API.
3. Запрещены любые другие способы связи, кроме сетевых API.
4. API реализуются с помощью технологий, подходящих для конкретной задачи.
5. API проектируются так, чтобы их можно было открывать внешним клиентам.

В 2010-х годах наблюдался рост использования API-технологий — этот период часто называют эпохой API. Вот хронология некоторых API-технологий, появившихся в это время: REST (2000), вебхуки (2007), WebSocket (2011), GraphQL (2015) и gRPC (2015).

Зачем нужны API

Рассмотрим ценность API для специалистов, разработчиков и организаций.

Часто говорят, что нужно развивать так называемые *мягкие навыки* (soft skills) — универсальные профессиональные качества, такие как тайм-менеджмент, коммуникабельность или работа в команде. Благодаря мягким навыкам соискателя могут пригласить на собеседование, но в конечном счете компании ищут кандидатов, которые помогут им достичь их целей. Организациям нужны специалисты с конкретными навыками — это могут быть как *мягкие* навыки (межличностное общение), так и *жесткие*. Если кандидат обладает навыками, которые нужны компании, то получает преимущество. Поэтому, независимо от того, являетесь ли вы разработчиком ПО или архитектором сетевых приложений, необходимо развивать навыки работы с API, поскольку они востребованы на разных позициях в разных отраслях.