Шпаргалка: введение в API

Что такое архитектура

Архитектура — это способ организовать компоненты приложения.

В клиент-серверной архитектуре выделяют три компонента:

- Клиент — приложение, с которым работает пользователь.
- Сервер — система, к которой обращаются клиенты, чтобы получить данные.
- Сеть — система из нескольких устройств, которая помогает клиенту и серверу обмениваться данными.

В веб-приложениях клиент — это фронтенд, видимая часть приложения. Бэкенд — это серверная часть: отвечает за логику и данные.

Клиент и сервер обмениваются сообщениями по протоколам HTTP и HTTPS. Клиент отправляет запрос (request); сервер возвращает ответ (response).

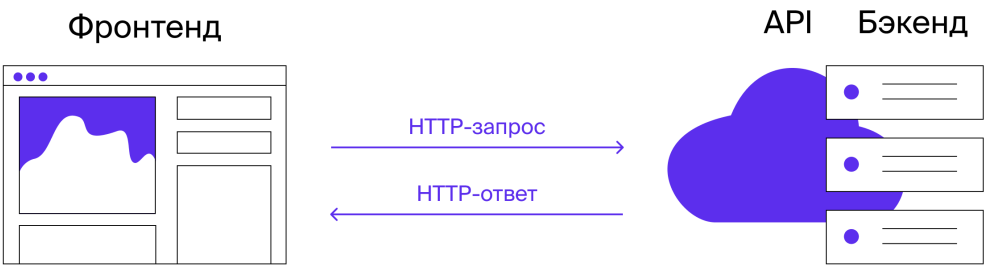
Что такое API и база данных

API — это интерфейс, который помогает приложениям взаимодействовать. Через него они обмениваются данными.

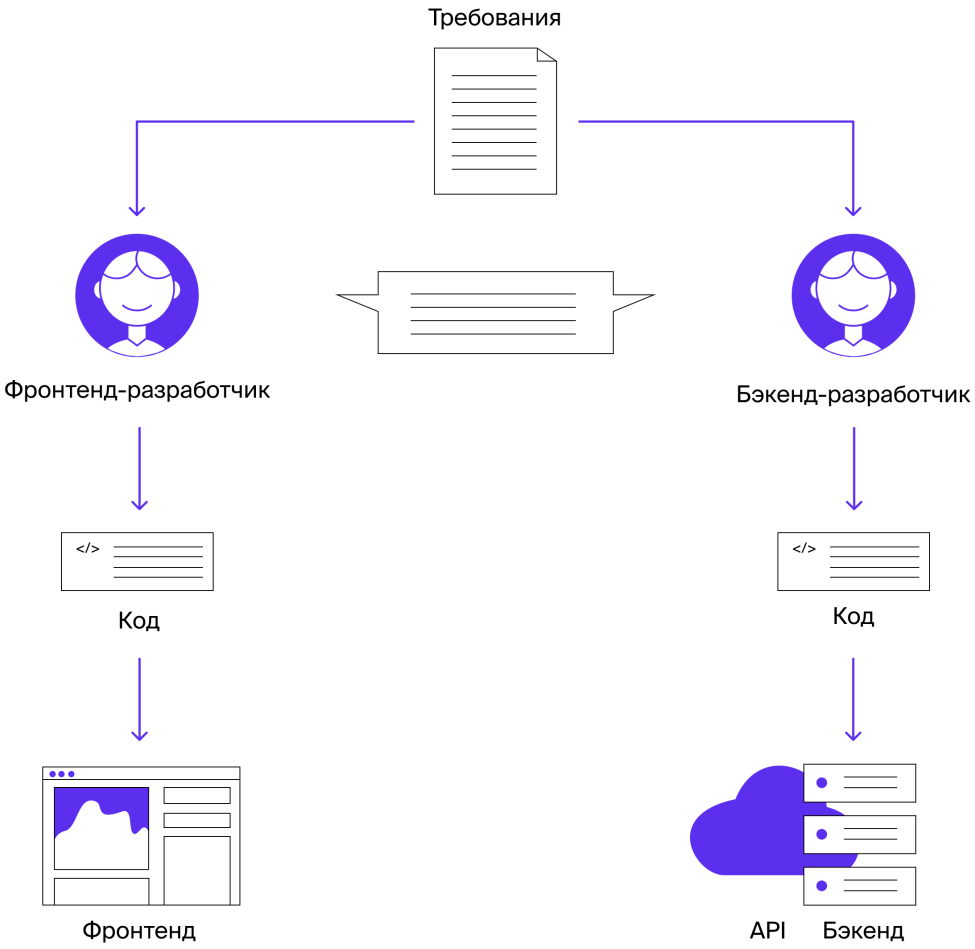
Например, если фронтенду приложения Яндекс.Маршруты нужно получить координаты маршрута, он обращается к API сервиса Яндекс.Карты.

Фронтенд общается с бэкендом, отправляя запросы и получая ответы. Чтобы сделать запрос к бэкенду, фронтенд обращается к API бэкенда.

Здесь API бэкенда — это фиксированный набор URL-адресов. По ним можно передавать данные в определённом формате.

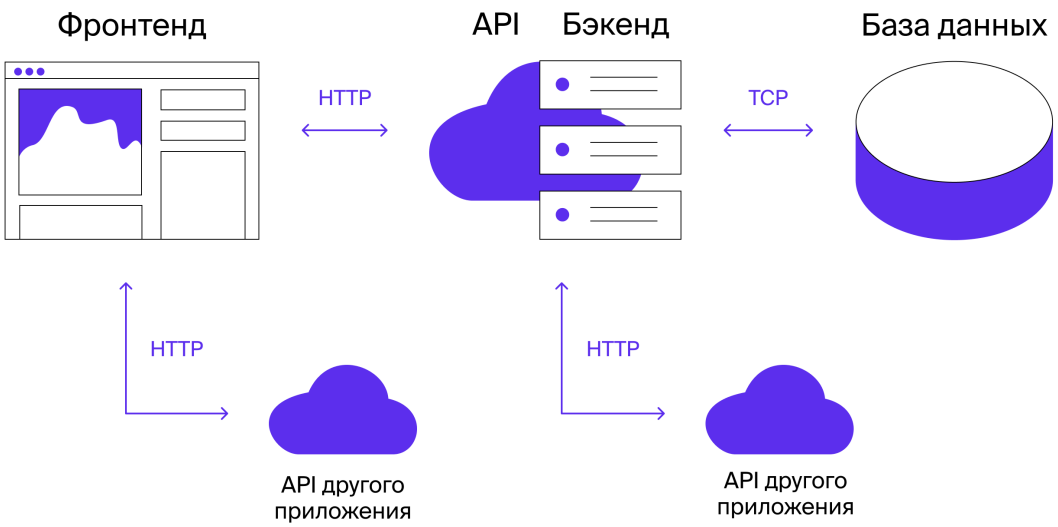


API как часть бэкенда «пишет» разработчик. Обычно он делает это по спецификации или техническому заданию приложения.

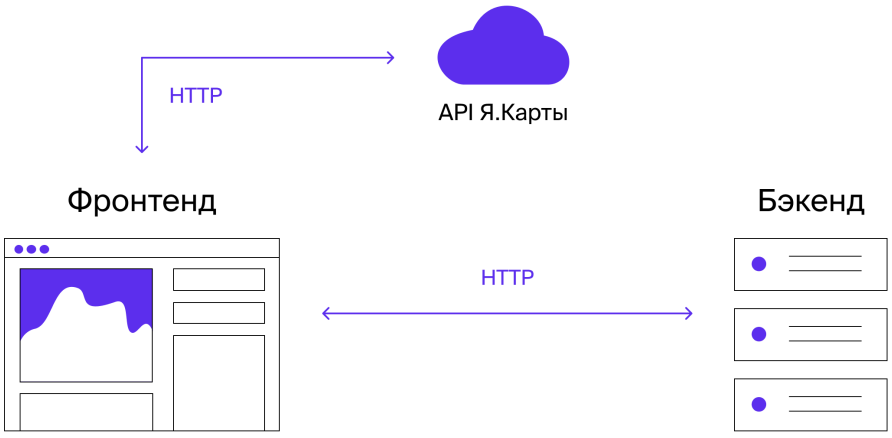


База данных (БД) — это набор данных, которые чаще всего хранятся в электронном виде, доступ к которым пользователь получает с помощью компьютера. Например, в базе данных интернет-магазина может храниться информация о товарах и зарегистрированных покупателях.

Упрощённая схема архитектуры с API и БД может выглядеть так:



Как взаимодействуют Яндекс.Маршруты с API Яндекс.Карт: карту и координаты маршрута возвращает API Яндекс.Карт; расчёт цены и длительности поездки отдаёт бэкенд Яндекс.Маршрутов.



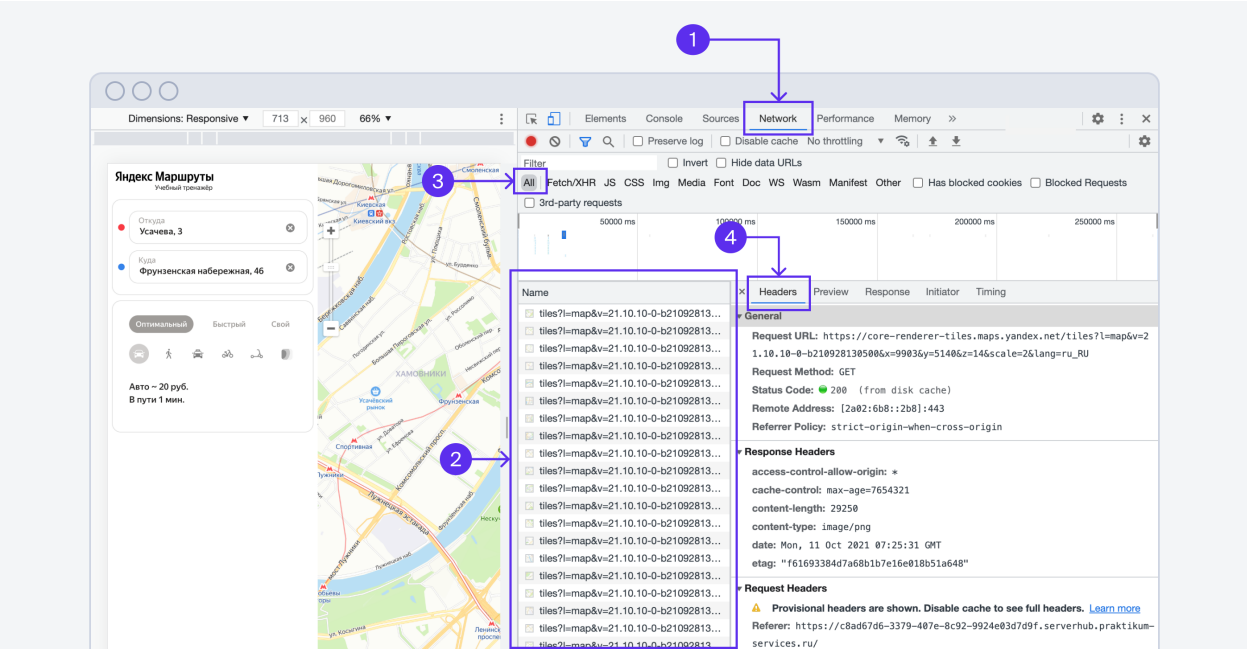
Как работать с такой архитектурой

Вот как распределяются запросы от фронтенда к API бэкенда и API Яндекс.Карт — на примере приложения Яндекс.Маршруты.

Посмотри на запросы на вкладке Network, чтобы увидеть, как компоненты архитектуры приложения Яндекс.Маршруты взаимодействуют между собой.

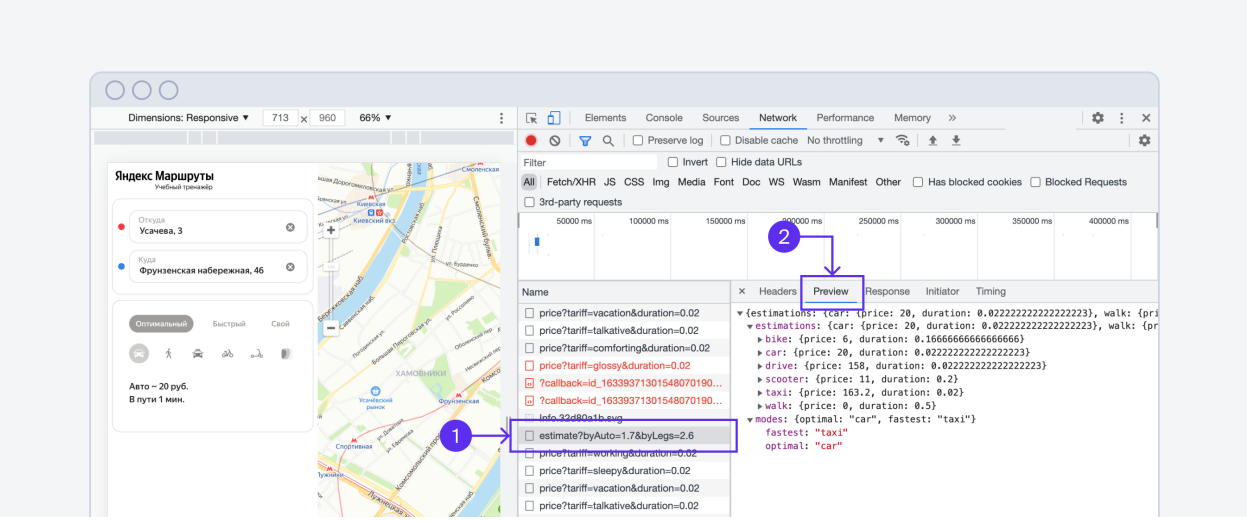
Открой Яндекс.Маршруты и вкладку Network в Devtools.

В Devtools видны отправленные запросы — слева. Если кликнуть на любой запрос, появится вкладка с ответом на него — он находится справа на вкладке Preview.

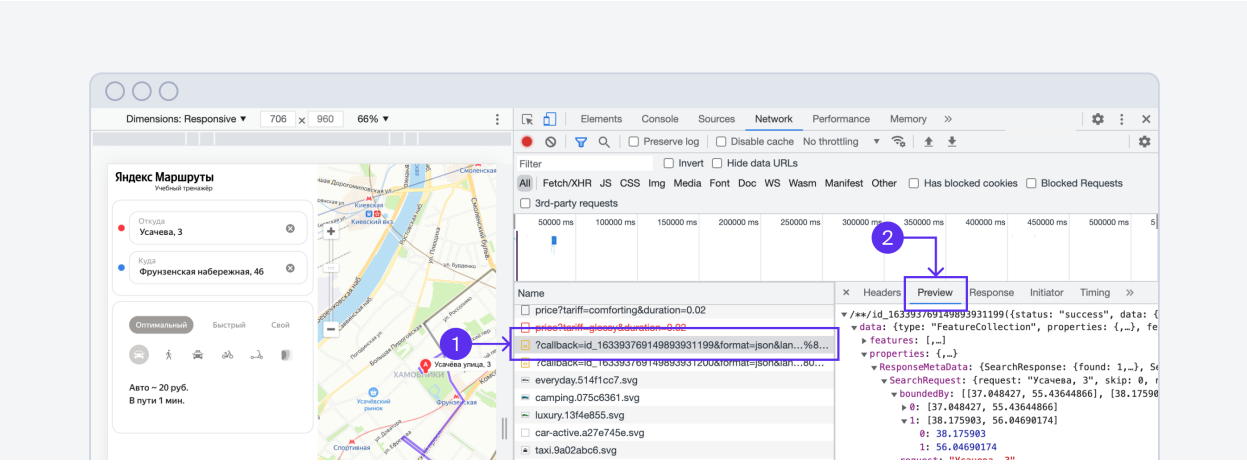


В списке запросов ты увидишь строку, в которой есть URL https://qa-routes.praktikum-services.ru/api/estimate?distanceWalk=2.6&distanceAuto=1.7&hour=10&minute=00.

Это запрос, который отправил клиент, чтобы получить информацию о цене и продолжительности поездки. Справа в Preview — ответ бэкенда с этой информацией.



Это запросы от клиента к API Я.Карт — справа видны ответы с информацией.



На вкладке Headers можно увидеть код ответа сервера на запросы. В обоих случаях Status Code: 200 OK. Это значит, сервер успешно обработал запрос.

Архитектурные решения — REST

Архитектуру приложения можно выстраивать по-разному. Самые распространённые способы — по SOAP и REST.

REST — один из способов спроектировать приложение. Его главная особенность — простота отображения данных. В других архитектурных стилях данные «обёрнуты» в дополнительный слой разметки. В REST дополнительного слоя нет: по запросам и ответам сразу понятно, что происходит с данными.

Например, метод POST создаёт данные, DELETE — удаляет.

Особенности REST

Сервер не хранит информацию о клиенте. Запрос клиента несёт ту информацию, которая нужна, чтобы сервер отправил ответ.

Многослойная структура. В архитектуре есть компоненты, которые обмениваются данными, — клиент и сервер; кроме них есть ещё и компоненты, которые распределяют потоки данных.

Единообразные компоненты. Каждый компонент архитектуры в REST построен по чётким правилам. К любому компоненту можно обратиться с запросом по определённому URL. В запросе будет вся информация, чтобы компонент смог обработать его корректно.

Кэширование. Когда сервер отправляет ответ, у него остаётся информация, какие данные нужно кэшировать, а какие нет.

Зачем знать о REST

От архитектурного стиля API зависит, как компоненты архитектуры обмениваются сообщениями: по какому протоколу и в каком формате. Это определяет то, как именно и какими инструментами ты будешь тестировать приложение.

Например, тебе нужно протестировать API. Ты знаешь, что оно построено по REST. Значит, оно принимает запросы по HTTP в форматах JSON или XML. Эта информация помогает определиться с инструментами: скорее всего, здесь лучше тестировать через Postman.

HTTP: структура запросов и ответов

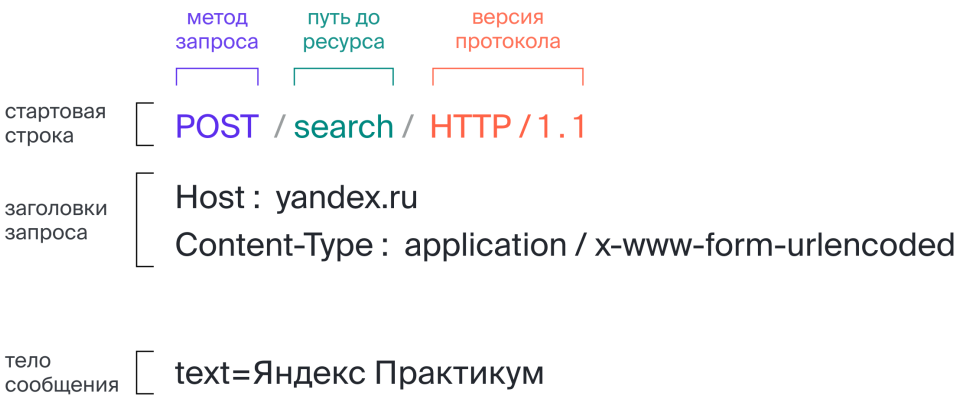
Клиенту и серверу нужно обмениваться сообщениями. Чтобы это происходило, клиент отправляет серверу **HTTP-запрос,** а сервер возвращает **HTTP-ответ**.

HTTP-запрос

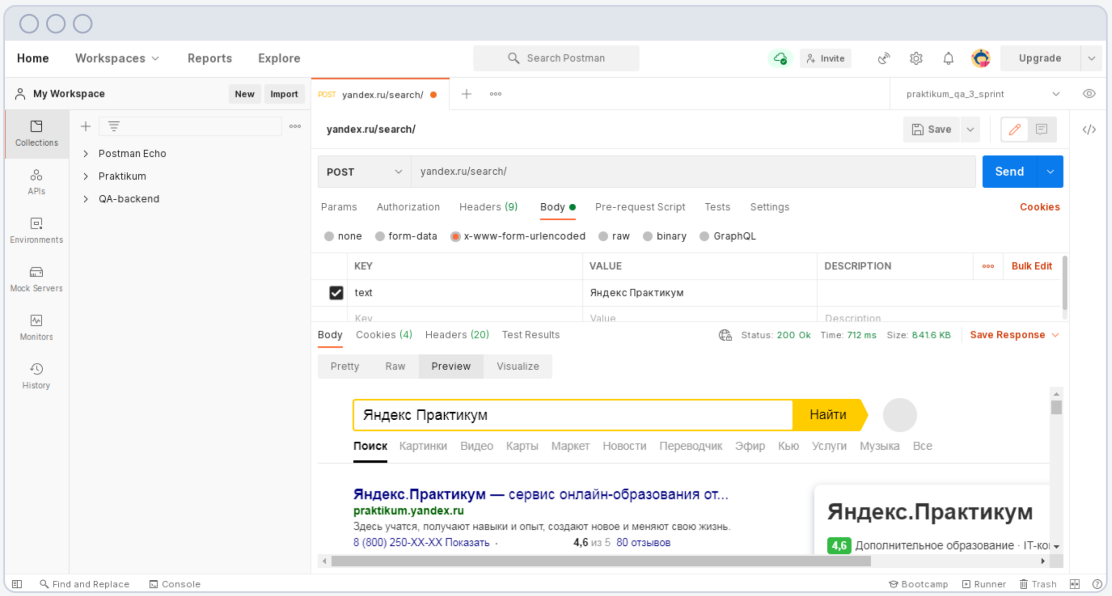
HTTP-запрос (Request) состоит из трёх блоков:

- **стартовая строка** содержит метод запроса, URL и версию протокола;

- **заголовки** передают дополнительную информацию. Например, доменное имя ресурса или длину контента в байтах;
- **тело сообщения** содержит данные, которые передаёт клиент. Например — текст, который пользователь ввёл в форме регистрации.



Если отправить такой запрос, откроется поиск по тексту «Яндекс Практикум».



Метод сообщает серверу, какое действие нужно выполнить. Самые распространённые методы:

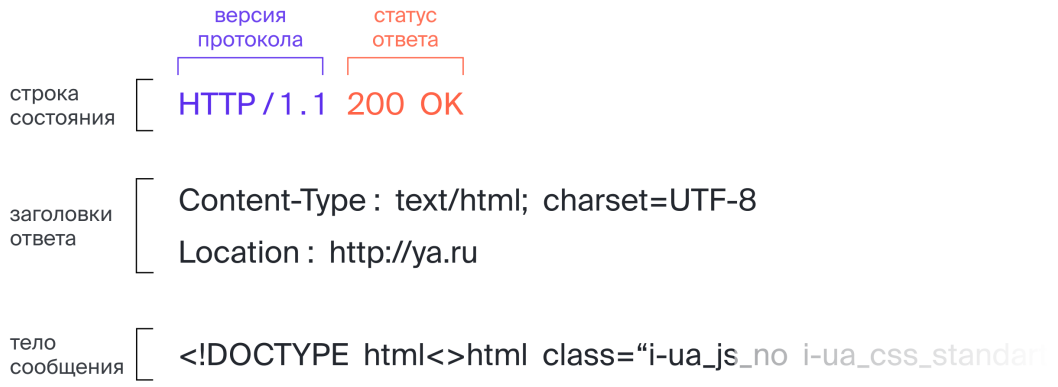
- GET — запрашивает данные;
- POST — создаёт или получает данные;
- PATCH — обновляет данные;
- DELETE — удаляет данные;
- PUT — обновляет данные.

URL — адрес, по которому клиент запрашивает данные, а сервер передаёт.

HTTP-ответ

Из чего состоит HTTP-ответ (Response):

- **строка состояния** содержит версию протокола, код состояния и текст состояния;
- **заголовки** передают дополнительную информацию: например, дату и время создания ответа;
- **тело** содержит набор данных, которые клиент получает в ответ на свой запрос. Например, информацию о всех зарегистрированных в приложении пользователях.



Код состояния (Status Code) сообщает, успешно ли сервер обработал запрос. Например, код 200 означает, что всё прошло успешно. А код 500 указывает на внутреннюю ошибку сервера.

Текст состояния — текст сообщения кода состояния от сервера. Например, ошибка 500 — «внутренняя ошибка сервера».

Какие есть коды состояния:

Коды	Описание	Пример
1xx	Информационные сообщения	102 — запрос принят, но обработка ещё не завершена
2xx	Сообщения об успехе	200 — OK, запрос обработан успешно
3xx	Перенаправление	302 — запрошенный ресурс временно доступен по другому адресу
4xx	Клиентские ошибки	404 — страница, которую запросил клиент, не найдена на сервере
5xx	Ошибки сервера	500 — внутренняя ошибка сервера

JSON

JSON (JavaScript Object Notation) — это «лёгкий» формат данных, который часто используется при обмене данными между приложениями. Основное преимущество — JSON легко писать и читать.

С помощью JSON можно структурно описать практически любой объект, чтобы использовать в системе.

Например, ты хочешь описать информацию о человеке — его ФИО. В формате JSON она может выглядеть так:

```
{
  "first_name": "Иван",
  "last_name": "Иванов",
  "middle_name": "Иванович"
}
```

Перед тобой пары вида ключ:значение. «Ключ» — это имя параметра объекта. Например, фамилия, отчество или имя. «Значение» указывается после двоеточия:

- first_name — «Иван»;
- last_name — «Иванов»;
- middle_name — «Иванович».

Параметры перечисляют через запятую и описывают в определённом формате. Например, ключ всегда берут в двойные кавычки; значение — в зависимости от типа данных. Текстовые данные заключают в кавычки.

Обрати внимание: набор пар заключён в фигурные скобки — так ты можешь указать, что они относятся к одному и тому же объекту.

Нужно добавить в структуру возраст человека:

```
{
  "first_name": "Иван",
  "last_name": "Иванов",
  "middle_name": "Иванович",
  "age": 35
}
```

Появился ключ `age`, но его значение — это число. Числа обычно не включают в кавычки.

Но иногда бывает нужно передать «число» (по смыслу) как «строку» (по записи в JSON) — например, `"age": "100"`.

Тут `100` — это строка в контексте формата JSON (и будет обрабатываться приложением как строка), но число по смыслу (пользователь знает, что 100 — это число).

Следующий шаг — добавить информацию о семейном положении и наличии детей:

```
{
  "first_name": "Иван",
  "last_name": "Иванов",
  "middle_name": "Иванович",
  "age": 35,
  "is_married": true,
  "has_kids": false
}
```

В «Семейном положении» действует условная бинарная логика:

- `true` — правда;
- `false` — ложь.

Эти значения называют «булевыми» (от слова boolean). Их тоже не заключают в кавычки. Ключ описан в виде условия: «Женат ли» человек, «Есть ли дети»; значение отвечает на эти вопросы — да или нет.

Что делать, если один параметр может содержать несколько значений? Например, доступные email-адреса указывают так:

```
{
  "first_name": "Ivan",
  "last_name": "Ivanov",
  "middle_name": "Ivanovich",
  "age": 35,
  "is_married": true,
  "has_kids": false,
  "emails": ["ivanov.i.i.@yandex-work.ru", "ivan@yandex-home.ru"]
}
```

Запись значения в квадратных скобках указывает на список, или, по-другому, массив. Список может содержать одно значение — `["type":"dog"]`, несколько — `["type":"dog", "type":"cat"]` или быть пустым. Например, если у человека нет почтовых адресов, но данные об этом всё равно передаются. Это будет выглядеть так: `"emails": []`.

Значения внутри массива могут быть представлены не только текстом, но и числами, а ещё другими объектами. Например, появились данные о наличии

домашних животных:

```
{
  "first_name": "Ivan",
  "last_name": "Ivanov",
  "middle_name": "Ivanovich",
  "age": 35,
  "is_married": true,
  "has_kids": false,
  "emails": ["ivanov.i.i.@yandex-work.ru", "ivan@yandex-home.ru"],
  "favorite_numbers": [0, 7, 42],
  "pets": [{"type": "dog", "name": "Шарик"}, {"type": "cat", "name": "Мурлыка"}]
}
```

Внутри массива данные просто перечисляют через запятую, и порядок имеет значение. Для сравнения: внутри объектов, где параметры определяются парами `ключ:значение`, порядок может быть любой.

Объект может быть пустым. Так же, как и массив. В этом случае значение будет записано просто фигурными скобками без содержимого между ними: `{}`.

Если нужно явно указать, что параметру не присвоено никакое значение, применяют запись вида `"pets": null` — слово `null` говорит, что значение этого параметра не определено.

Структура данных в формате JSON может быть очень разнообразной и зависит от информации, которую нужно описать. От очень простого списка — например, артикулов товаров, которые купил пользователь `[10429, 23294, 12357]` — до длинного набора параметров, где каждый — это отдельный объект со своей структурой. Например:

```
{
  "user": {
    "name": "Ivan",
    "lastVisited": "2020-02-11",
    "previous_order_ids": [10032, 12158, 77923]
  },
  "last_order": {
    "article": 76240,
    "prepaid": true,
    "price": 999.99,
    "need_delivery": false,
    "comments": null
  },
```

```
  "additional_info": {}
}
```