Типы данных в API

Справочник тестировщика: типы JSON и OpenAPI, их форматы и что с ними проверять

База: типы данных JSON

Любой ответ API — это JSON, а в JSON всего шесть типов:

Тип	Пример
`string`	`"John"`
`number`	`25.5`
`object`	`{ "id": 1 }`
`array`	`[1, 2, 3]`
`boolean`	`true`
`null`	`null`

OpenAPI уточняет их через ключевые слова: `integer` — целое, `format` — подтип (дата, uuid, email…), а также ограничения: `minimum`, `maxLength`, `pattern`, `enum` и другие. Типы полей описаны в документации — это часть контракта API, и каждое отклонение от них — баг.

String — строка

```
"name": "John Doe"
```

Форматы (format)

- `date` — дата: `"2026-08-23"`
- `date-time` — дата и время: `"2026-08-23T12:00:00Z"`
- `email`, `uuid`, `uri`, `ipv4`, `hostname` — по соответствующим стандартам

Примеры

```
"John Doe"            // валидно
""                      // пустая строка — допустима ли по доке?
"John123"             // цифры, если pattern только буквы
"  John  "            // пробелы по краям — триммит ли сервер?
"John" O'Neil"       // кавычки внутри — экранирование
"aaaaaaaa…(300 симв.)" // превышение maxLength
"Иван Петров"          // другая раскладка/юникод
```

Что проверять

- Длина: `minLength` / `maxLength` — пустая строка, 1 символ, граница, превышение
- `pattern` — регулярка допустимых символов; цифры, спецсимволы, другая раскладка
- Пробелы по краям (тримминг), регистр, кодировки и юникод, экранирование кавычек и переводов строк

Integer — целое число

```
"age": 25
```

Форматы

- `int32` — от −2 147 483 648 до 2 147 483 647
- `int64` — от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807

Примеры

```
25                      // валидно
0                       // ноль — допустим ли по доке?
-1                      // отрицательное при minimum: 0
25.5                    // дробное — не целое
"25"                    // число строкой — классика
2147483648             // переполнение int32
```

Что проверять

- Границы: `minimum` / `maximum` и значения сразу за ними
- Ноль, отрицательные, очень большие числа (переполнение int32)
- Дробное значение 25.5 — не целое; число строкой `"25"` — классический баг типа в ответе

Number — дробное число

```
"price": 19.99
```

Форматы

- `float` — одинарная точность (~7 значащих цифр)
- `double` — двойная точность (~15 значащих цифр)

Примеры

```
19.99                    // валидно
19.999                   // лишний знак при шаге 0.01
19,99                    // запятая вместо точки
-42.7                    // отрицательная цена?
0.1 + 0.2 = 0.30000000000000004 // бинарная арифметика
1234567.891             // точность float: хвост потеряется
1e10                     // экспоненциальная запись
```

Что проверять

- Точка как разделитель, количество знаков после запятой
- Отрицательные, ноль, экспоненциальная запись, потеря точности float (7 значащих цифр)
- `multipleOf` — кратность (например, шаг 0.01 для цен)
- **Округление при трансформации:** сервер может гонять число через float→double или decimal в БД — 19.999 может прийти как 20.00 или 19.99. Сверь, сколько знаков реально приходит, и какое округление используется (вверх, вниз, банковское)

Boolean — логическое

```
"is_active": true
```

Примеры

```
true                    // валидно
false                   // валидно
"true"                  // строка вместо булева
1                       // число вместо булева
"yes"                   // сервер кастует? контракт должен решать
null                    // допустим ли null для обязательного флага
```

Что проверять

- Только `true` или `false`, без кавычек: `"true"` строкой — баг
- Не путать с числами: 0/1 вместо false/true — частое расхождение
- Неявные касты: принимает ли сервер "yes"/"no"/1/0 и что отвечает в ответе

Array — массив

```
"items": [ {"id": 1}, {"id": 2} ]
```

Примеры

```
[]                      // пустой — валидный ответ, не ошибка
[1, 2, 3]               // элементы одного типа
[1, "two", 3]           // смешанные типы
[1, 1, 1]               // дубликаты — допустимы ли?
[[1], [2]]              // вложенные массивы по схеме?
null                    // null вместо массива
```

Что проверять

- Элементы одного типа по схеме (`items`)
- Пустой массив `[]` — валидный ответ, когда данных нет; не ошибка
- Ограничения `minItems` / `maxItems`, порядок, дубликаты
- Пагинация и сортировка для списков — отдельная зона проверок

Object — объект

```
"user": { "id": 1, "name": "John", "email": null }
```

Примеры

```
{ "id": 1, "name": "John" }        // валидно
{}                                   // пустой объект без required-полей
{ "id": 1 }                          // нет обязательного name
{ "id": 1, "name": "John", "hack": 1 } // лишнее поле
{ "Name": "John", "id": 1 }          // регистр ключа
"user": null                         // null вместо объекта
```

Что проверять

- Обязательные ключи (`required`) — отсутствие обязательного поля это баг
- Лишние поля (`additionalProperties`) — пропускает сервер или режет?
- Вложенные объекты — типы проверяются на каждом уровне
- Ключи регистрозависимы: `"Name"` и `"name"` — разные поля

Null — отсутствие значения

```
"middle_name": null
```

Примеры: это РАЗНЫЕ вещи

```
null        // явный null
(нет поля)  // поле вообще отсутствует в ответе
""          // пустая строка
0           // ноль
[]          // пустой массив
```

Что проверять

- null ≠ 0, null ≠ `""`, null ≠ `[]` — не подменяй при проверке
- Где null допустим — решает документация: `nullable: true` или `type: [string, null]`
- Отличай «поле отсутствует» от «поле = null» — сервисы часто трактуют по-разному

Enum — перечисление

```
"status": "active"   // допустимы только: active, blocked, deleted
```

Примеры

```
"active"     // валидно
"ACTIVE"     // регистр
"unknown"    // значение вне списка
""           // пустая строка
null         // null вместо enum
```

Что проверять

- Только значения из списка; всё остальное — ошибка 400
- Регистр: `"ACTIVE"` не должен проходить, если задокументирован нижний
- Проверь каждое значение списка — все должны обрабатываться

Дата и время

```
"created_at": "2026-08-23T12:00:00Z"
```

Примеры

```
"2026-08-23"                // дата
"2026-08-23T12:00:00Z"      // UTC (суффикс Z)
"2026-08-23T15:00:00+03:00" // со смещением таймзоны
"2026-08-23T12:00:00.123Z"  // миллисекунды
"23.08.2026"                // локальный формат вместо ISO
"2026-13-45"                // невалидные месяц и день
"2025-02-29"                // 29 февраля в невисокосный год
```

Что проверять

- Стандарт ISO 8601 / RFC 3339: `YYYY-MM-DDTHH:MM:SSZ`
- Таймзоны: суффикс `Z` (UTC) или смещение `+03:00`
- Границы: даты в прошлом и будущем, високосные годы, точность (с секундами, миллисекундами или без)
- Сроки действия — любимое место багов: «истёкшее» значение всё ещё активно

UUID и другие идентификаторы

```
"id": "550e8400-e29b-41d4-a716-446655440000"
```

Примеры

```
"550e8400-e29b-41d4-a716-446655440000"  // валидный UUID v4
"550e8400"                               // слишком короткий
"550e8400-e29b-41d4-a716-44665544000G"  // буква G — не hex
"550e8400e29b41d4a716446655440000"       // без дефисов
null                                     // null вместо id
```

Что проверять

- UUID (RFC 4122): 36 символов, 8-4-4-4-12, шестнадцатеричные
- Формат кодов часто задан жёстко (префикс, регистр, длина) — сверяй с документацией
- Целочисленные ID: отрицательные, нулевые, очень большие, несуществующие

Комбинации схем

- `oneOf` — ровно одна из схем
- `anyOf` — хотя бы одна из схем
- `allOf` — все схемы одновременно

Встречаются в сложных ответах: поле может быть строкой ИЛИ объектом. Проверяй все ветви каждой комбинации.

Типы данных — часть контракта API

Документация описывает не только методы, но и типы каждого поля. Любое отклонение — баг.

Больше полезного про тестирование — в телеграм-канале @eddytester.