- **Модуль:** Typescript. Generics and type manipulations
- **Урок:** Работа с запросами на сервер, Promise и JSON

TypeScript Handbook
from
Иван Петриченко
udemy

РАБОТА С JSON И ЗАПРОСАМИ

Чтобы мы не делали, максимально типизировать парсинг json-строки в TS **мы не сможем.** Эта операция происходит в **рантайме**, то есть при запуске кода и в самой строке может быть **что угодно.** Так что такой код особого смысла не имеет:

```ts
const jsonTest = '{ "name": "Test", "data": "dfdg"}';

interface JSONTest {
    name: string;
    data: number;
}

const objFromJson: JSONTest = JSON.parse(jsonTest);
```

Результат работы будет **any** и подсказок никаких TS не даст. Для правильной обработки таких данных **можно использовать стандартные техники:** проверка на наличие нужных свойств, значений, типов и тп.
В TS можно **вместо any вернуть unknown** и обрабатывать его через сужение типов:

```ts
const userData =
    '{"isBirthdayData": true, "ageData": 40, "userNameData": "John"}';

function safeParse(s: string): unknown {
    return JSON.parse(s);
}

const data = safeParse(userData);

function transferData(d: unknown): void {
    if (typeof d === "string") {
        console.log(d.toLowerCase());
    } else if (typeof d === "object" && d) {
        console.log(data);
    } else {
        console.error("Some error");
    }
}
```

При работе с запросами и получении данных с сервера та же схема. **Получили данные и проверяете их:**

```ts
fetch("https://jsonplaceholder.typicode.com/todos/1")
    .then((response) => response.json())
    .then((json) => {
        if ("id" in json) {
            toDoList.push(json);
        }
        console.log(toDoList);
    });
```

Такие условия в запросах можно расширять по вашим нуждам. Проверять **на тип данных и содержимое в них**:

```
fetch("https://jsonplaceholder.typicode.com/todos")
    .then((response) => response.json())
    .then((json) => {
        if ("id" in json && "userId" in json) {
            toDoList.push(json);
        } else if (Array.isArray(json)) {
            toDoList = json;
        } else {
            console.log(`${json} - is a string`);
        }
        console.log(toDoList);
    });
```

Типизация кода внутри .then **никак не убережет нас от ошибок** в рантайме. Но она можем повысить **семантику** вашего кода: сказать всем разработчикам, что именно вы тут ожидаете и получить подсказки внутри функции. Так что тут на ваше усмотрение

```
let toDoList: ToDo[] = [];

interface ToDo {
    userId: number;
    id: number;
    title: string;
    completed: boolean;
}

fetch("https://jsonplaceholder.typicode.com/todos")
    .then((response) => response.json())
    .then((json: ToDo) => {
        if ("id" in json && "userId" in json) {
            toDoList.push(json);
            toDoList[0].  // тут будут подсказки о существующих свойствах
```

ИНТЕРФЕЙС PROMISE

При выполнении запроса с fetch можно обнаружить, что он нам возвращает **Promise<Response>** А значит, что в TS существует интерфейс Promise, **принимающий тот тип**, который он будет возвращать при успешном выполнении

Для примера можно создать свой пример со строкой:

```
const promise = new Promise<string>((resolve, reject) => {
    resolve("Test"); // resolve принимает только string
});

promise.then((value) => {
    console.log(value.toLowerCase()); // value - string
});
```

Во время запроса TS автоматически подставляет туда интерфейс Response, который содержит методы .json() и тп.