

Ufaj, ale sprawdzaj, czyli co nieco o dekodkach

Mikołaj Klaman, Virtus Lab

Tworząc aplikacje w TSie chcemy:

- **Type safety at compile-time** - wiemy co jest czym
- **Type safety at run-time** - typy przekładają się na rzeczywistość

Tworząc aplikacje w TSie chcemy:

- **Type safety at compile-time** - wiemy co jest czym
- **Type safety at run-time** - typy przekładają się na rzeczywistość

↑
— [typescript non-goal](#)

Tworząc aplikacje w TSie chcemy:

- **Type safety at compile-time** - wiemy co jest czym
- **Type safety at run-time** - typy przekładają się na rzeczywistość

↑
_____ [typescript non-goal](#)

```
let foo: any // opt-out of type checking 🤔
```

Tworząc aplikacje w TSie chcemy:

- **Type safety at compile-time** - wiemy co jest czym
- **Type safety at run-time** - typy przekładają się na rzeczywistość

↑
_____ [typescript non-goal](#)

```
const data = fetch("/api/data").then(res => res.json())  
  
type Data = typeof data // Promise<any> 🤖
```

I/O to wrota dla **any** do wnętrza naszego systemu

REST APIs, localStorage, third party code, ...

Jak sobie z tym poradzić?

I/O to wrota dla **any** do wnętrza naszego systemu

REST APIs, localStorage, third party code, ...



Jak sobie z tym poradzić?

Strategia #1: YOLO

- Ufamy że API dotrzymuje swojego kontraktu
- Ufamy że kontrakt API jest poprawnie interpretowany po naszej stronie

=> Nie obsługujemy przypadków w których tak nie jest

```
const data = fetch("/api/data").then(res => res.json())  
  
return data as MyData // yey! 🤪
```


Strategia #2: CODEGEN

- Generujemy typy i klienta na podstawie specyfikacji API
- Jako dodatkowe benefity mamy mniej kodu do napisania i jesteśmy na bieżąco ze zmianami w API 😎
- Ufamy że API dotrzymuje swojego kontraktu

Co może pójść nie tak?

- Brak lub niedokładna specyfikacja API
- API Breaking changes 😱
- Nadmiarowe / rozwlekłe wygenerowane typy

Strategia #2: CODEGEN

DepartmentsWhatIfsWhatIfIdProductivityModelRoutinesRoutineIdElementsElementIdTotalSubdepartmentsPutRequest

Strategia #3: Ufaj, ale sprawdzaj

- **Ufamy** na tyle żeby oprzeć pomyślny flow na naszych założeniach co do API
- **Weryfikujemy** założenia w runtime aby obsłużyć ich złamanie w przewidywalny sposób
- **Fail fast** - nie ma co ratować, jak kontrakt API się nie zgadza, to traktujemy to tak jakbyśmy dostali HTTP 4xx

#DefensiveProgramming

#TypeSafety

#HealthyParanoia

#TrustBoundries

Strategia #3.1: Type Guards

Czyli ręcznie sprawdzamy czy typy się zgadzają...

```
function isPost(json: unknown): json is Post {  
    if (typeof json === "object") {  
        if (typeof (json as Post).title === "string") {  
            ...  
        }  
    }  
}
```



Strategia #3.2: Dekodery!

Komponowalne utile do walidacji w runtime:

```
const postDecoder = Decoder.object({  
  title: Decoder.string(),  
  likes: Decoder.number(),  
  comments: Decoder.array(commentDecoder),  
});
```

Strategia #3.2: Jak podejść do dekodowania?

- A) **Type-First** – Piszemy dekodery pod nasze typy
- B) **Decoder-First** – Inferujemy typy z dekodерów
- C) **Just-Type** – Generujemy dekodery do typów

Strategia #3.2: Jak podejść do dekodowania?

- A) **Type-First** – Piszemy dekodery pod nasze typy
- B) **Decoder-First** – Inferujemy typy z dekodерów
- C) **Just-Type** – Generujemy dekodery do typów

DEMO

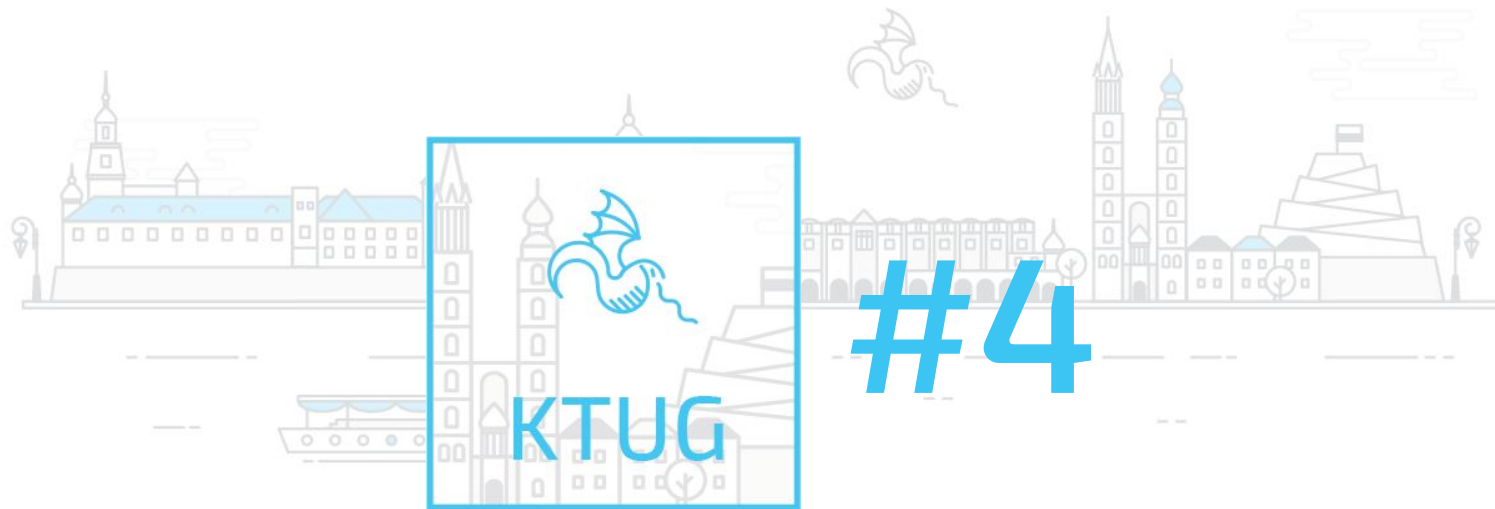
Podsumowanie

Czy warto dekodować?

- Czy kod wewnątrz systemu jest bezpieczny?
- Czy to co pochodzi z zewnątrz jest godne zaufania?
- Na ile mamy paranoję?

Polecam spróbować :)

- [json-type-validation](#) ★85 🏅
- [typescript-is](#) ★321
- [runtypes](#) ★774
- [io-ts](#) ★2.3k

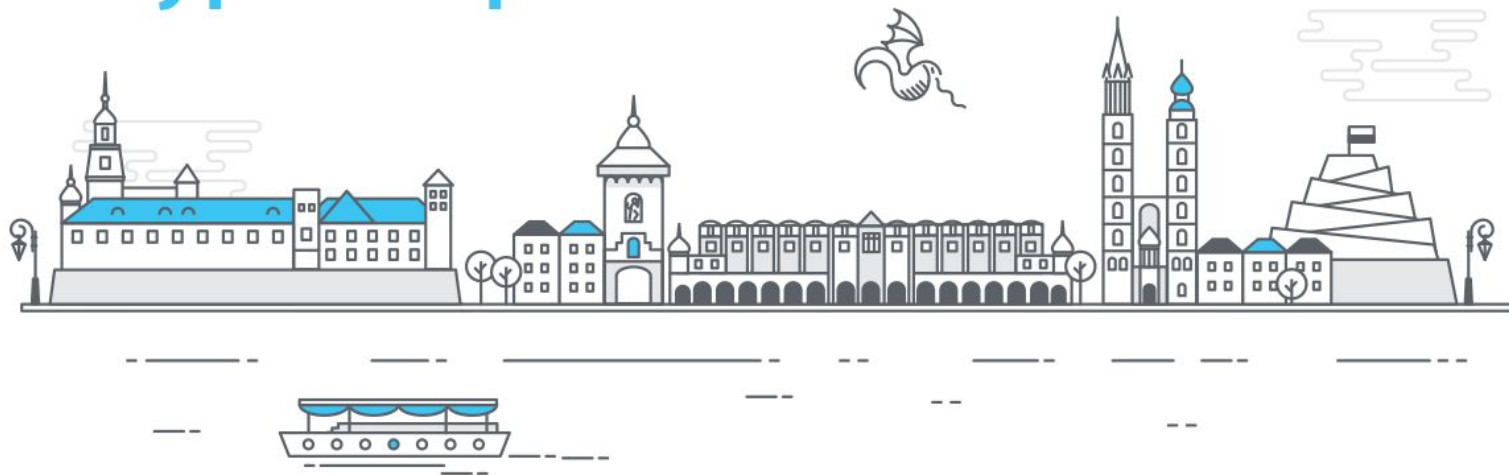


Dziękuję za uwagę!

Pytania?

Kraków TypeScript

A Type-Safe Kingdom



Sponsored by

