TypeScript is great.

TypeScript is great.

Introduces types to JavaScript

Increases type safety and reduces bugs

As long as your code compiles, there will be no type errors during runtime

TypeScript is great.

Introduces types to JavaScript

Increases type safety and reduces bugs

~~As long as your code compiles, there will be no type errors during runtime~~

```javascript
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)







  } catch (error) {
    // handle error...
  }
}
```

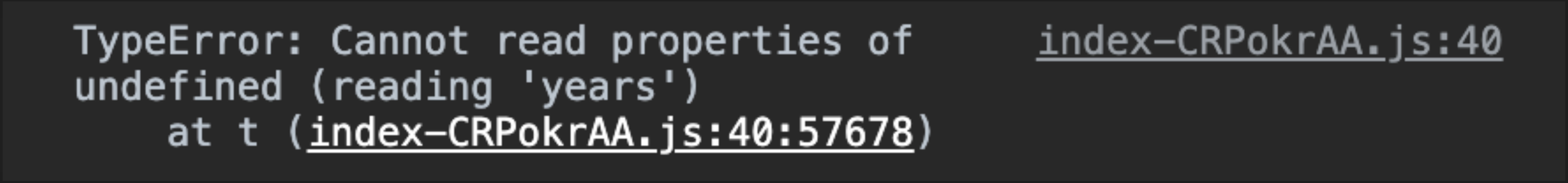
```javascript
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)
    if (!response.ok) {
      throw new Error(`${response.status} - ${response.statusText}`)
    }




  } catch (error) {
    // handle error...
  }
}
```

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)
    if (!response.ok) {
      throw new Error(`${response.status} - ${response.statusText}`)
    }

    const data: UserApiResponse = await response.json()
    return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`

  } catch (error) {
    // handle error...
  }
}
```

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)
    if (!response.ok) {
      throw new Error(`${response.status} - ${response.statusText}`)
    }

    const data: UserApiResponse = await response.json()
    return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`
```



```
TypeError: Cannot read properties of        index-CRPokrAA.js:40
undefined (reading 'years')
    at t (index-CRPokrAA.js:40:57678)
```

```
}
```

1. The problem

TypeScript's limitations during runtime: External Systems

1. The problem

TypeScript’s limitations during runtime: External Systems

2. The solution

Runtime validation (libraries)

1. The problem

TypeScript’s limitations during runtime: External Systems

2. The solution

Runtime validation (libraries)

3. Do I actually need it?

Tradeoffs & Evaluation

```
const object = {
  character: {
    about: {
      name: 'abc',
      age: {
        years: 20,
        months: 2,
      },
    },
  },
}
```

```
function double(x: number) {
  return (x * 2).toString()
}

function splitBySpace(str: string) {
  return str.split(" ")
}
```

```
const object = {
  character: {
    about: {
      name: 'abc',
      age: {
        years: 20,
        months: 2,
      },
    },
  },
}
```

```
function double(x: number) {
  return (x * 2).toString()
}

function splitBySpace(str: string) {
  return str.split(" ")
}
```

```
const a = double(object.character.about.age.years)
const object = {
  character: {
    about: {
      name: 'abc',
      age: {
        years: 20,
        months: 2,
      },
    },
  },
}
```

```
function double(x: number) {
  return (x * 2).toString()
}

function splitBySpace(str: string) {
  return str.split(" ")
}
```

```
const a = double(object.character.about.age.years)

const b = splitBySpace(a)
const object = {
  character: {
    about: {
      name: 'abc',
      age: {
        years: 20,
        months: 2,
      },
    },
  },
}
```

```
function double(x: number) {
  return (x * 2).toString()
}

function splitBySpace(str: string) {
  return str.split(" ")
}
```

```
const a = double(object.character.about.age.years)

const b = splitBySpace(a)

const c = double(b[0])
```

```
Argument of type 'string' is not assignable
to parameter of type 'number'. ts(2345)
const b: string[]
```

TypeScript is really good at catching type errors.

```ts
const object = {
  character: {
    about: {
      name: 'abc',
      age: {
        years: 20,
        months: 2,
      },
    },
  },
}

function double(x: number) {
  return (x * 2).toString()
}

function splitBySpace(str: string) {
  return str.split(" ")
}

const a = double(object.character.about.age.years)

const b = splitBySpace(a)

const c = double(b[0])
```

```
Argument of type 'string' is not assignable
to parameter of type 'number'. ts(2345)
const b: string[]
const object = {
  character: {
    about: {
      name: 'abc',
      age: {
        years: 20,
        months: 2,
      },
    },
  },
}
```

```
function double(x: number) {
  return (x * 2).toString()
}

function splitBySpace(str: string) {
  return str.split(" ")
}

const a = double(object.character.about.age.years)

const b = splitBySpace(a)

const c = double(b[0])
const object = {
  character: {
    about: {
      name: 'abc',
      age: {
        years: 20,
        months: 2,
      },
    },
  },
}
```

```
function double(x: number) {
  return (x * 2).toString()
}

function splitBySpace(str: string) {
  return str.split(" ")
}

const a = double(object.character.about.age.years)

const b = splitBySpace(a)

const c = double(b[0])
```

The entire data flow is **internal** (contained within code)

```typescript
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```typescript
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)
    if (!response.ok) {
      throw new Error(`${response.status} - ${response.statusText}`)
    }

    const data: UserApiResponse = await response.json()
    return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`

  } catch (error) {
    // handle error...
  }
}
```

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

External System

```
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)
    if (!response.ok) {
      throw new Error(`${response.status} - ${response.statusText}`)
    }

    const data: UserApiResponse = await response.json()
    return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`

  } catch (error) {
    // handle error...
  }
}
```

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

Only known at *runtime*

External System

```
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)
    if (!response.ok) {
      throw new Error(`${response.status} - ${response.statusText}`)
    }

    const data: UserApiResponse = await response.json()
    return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`

  } catch (error) {
    // handle error...
  }
}
```

Only known at *runtime*

External System

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

TypeScript is really good at catching type errors, *except data whose type is only available at runtime.*

```
async function getUserInfo() {
  try {
    const response = await fetch(API_URL)

    if (!response.ok) {
      throw new Error(response.statusText)
    }

    const data: UserApiResponse = await response.json()
    return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`

  } catch (error) {
    // handle error...
  }
}
```

```typescript
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```typescript
const data: UserApiResponse = await response.json()
return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`
```

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```
const data: UserApiResponse = await response.json()
return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`
```

Dev: data ***is*** of type UserApiResponse, trust me

TypeScript: Okay

Solution: Type data correctly

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```
const data: unknown = await response.json()
return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`
```

```
'data' is of type 'unknown'. ts(18046)
const data: unknown
```

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

Solution: Check manually

```
const data: unknown = await response.json()

if (typeof data === "object" && data !== null) {
  if ("name" in data && typeof data["name"] === "string") {
    const typed = data.name
  }
}
```

```typescript
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

Solution: Check manually

```typescript
const data: unknown = await response.json()

if (typeof data === "object" && data !== null) {
  if ("name" in data && typeof data["name"] === "string") {
    const typed = data.name
  }      const typed: string
}
```

Solution: Check manually

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```
const data: unknown = await response.json()

if (typeof data === "object" && data !== null) {
  if ("name" in data && typeof data["name"] === "string") {
    const typed = data.name
  }       const typed: string
}
const untyped = data.name
                  'data' is of type 'unknown'. ts(18046)
                  const data: unknown
```

Solution: Check manually

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```
const data: unknown = await response.json()

if (typeof data === "object" && data !== null) {
 if (
     'name' in data &&
     typeof data['name'] === 'string' &&
     'age' in data &&
     typeof data['age'] === 'object' &&
     data['age'] !== null &&
     'years' in data['age'] &&
     typeof data['age']['years'] === 'number' &&
     'months' in data['age'] &&
     typeof data['age']['months'] === 'number'
   ) {
     return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`
   }
}
```

Solution: Check manually

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

```
const data: unknown = await response.json()

if (typeof data === "object" && data !== null) {
 if (
    'name' in data &&
    typeof data['name'] === 'string' &&
    'age' in data &&
    typeof data['age'] === 'object' &&
    data['age'] !== null &&
    'years' in data['age'] &&
    typeof data['age']['years'] === 'number' &&
    'months' in data['age'] &&
    typeof data['age']['months'] === 'number'
  ) {
    return `${data.name} is ${data.age.years} years and ${data.age.months} months old today.`
  }
}
```

✅ Type safe

Runtime Validation Libraries

		No logo :(	
Zod	Joi	Yup	AJV

Zod Demo

```
interface UserApiResponse {
  name: string
  age: {
    years: number
    months: number
  }
}
```

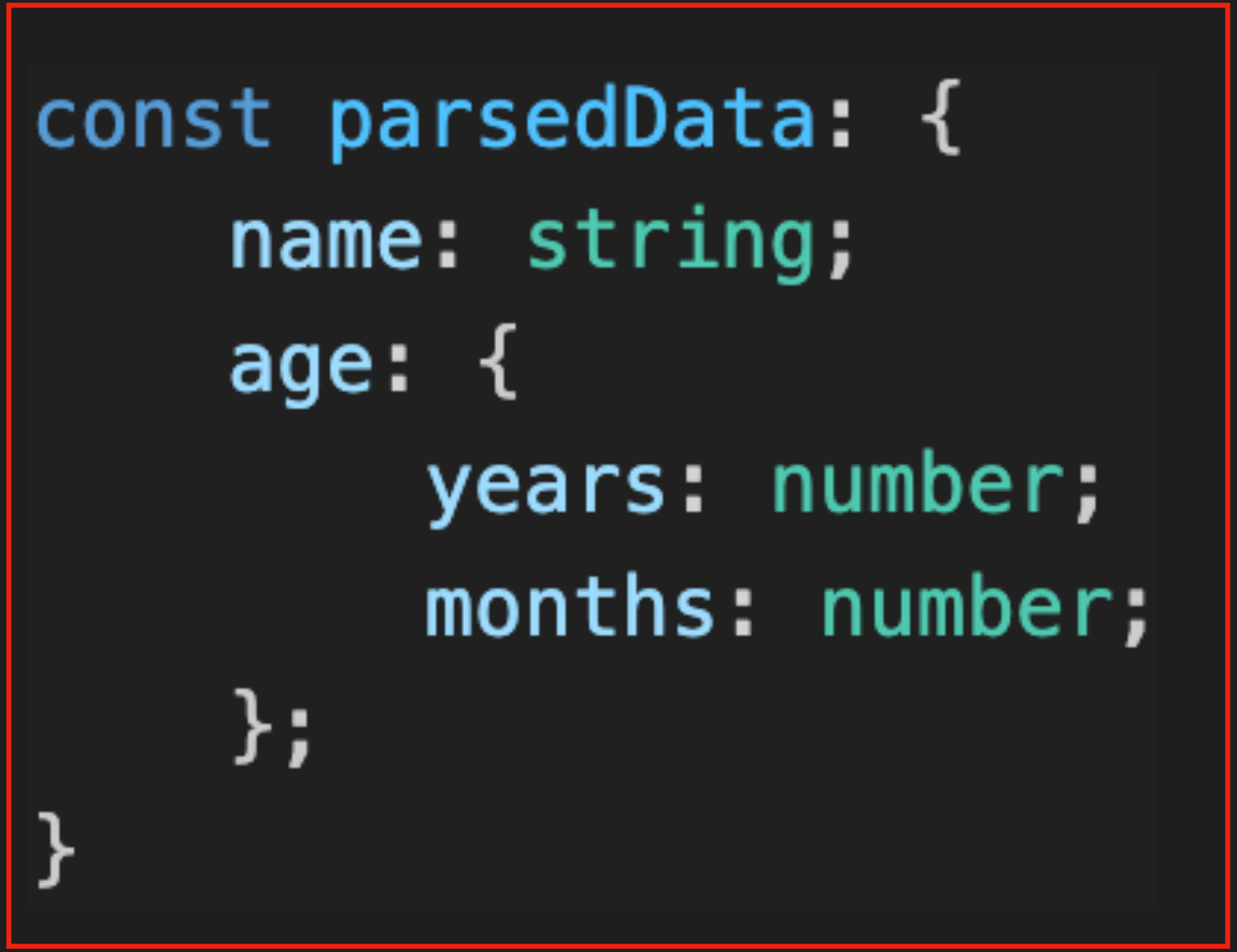
```
import { z } from 'zod'

const UserApiResponse = z.object({
  name: z.string(),
  age: z.object({
    years: z.number(),
    months: z.number()
  })
})

type UserApiResponse = z.infer<typeof
 UserApiResponse>
```

Zod Demo

```
const data: unknown = await response.json()

const parsedData = UserApiResponse.parse(data)

return `${parsedData.name} is ${parsedData.age.years} years and $
{parsedData.age.months} months old today.`
```

Zod Demo

```
const data: unknown = await response.json()

const parsedData = UserApiResponse.parse(data)

return `${parsedData.name} is ${parsedData.age.years} years and $
{parsedData.age.months} months old today.`
```



```
const parsedData: {
    name: string;
    age: {
        years: number;
        months: number;
    };
}
```

Zod Demo

```
const parsedData: {
    name: string;
    age: {
        years: number;
        months: number;
    };
}
```

```
try {

        const data: unknown = await response.json()

        const parsedData = UserApiResponse.parse(data)

        return `${parsedData.name} is ${parsedData.age.years} years and $
{parsedData.age.months} months old today.`

}
 catch {
// handle error...

}
```

Do I actually need it?

Do I actually need it?

Time

Bundle Size

(runtime)

Type Safety

Benchmark (simple object)

Library	**Operations per second**	**Average**
validator.js	1 023 397	977ns
validate.js	529 948	1µs
validatorjs	370 442	2µs
joi	323 164	3µs
ajv	13 859 044	72ns
mschema	1 303 805	766ns
parambulator	50 519	19µs
fastest-validator	14 880 485	67ns
yup	83 749	11µs
nope	3 480 196	287ns
jsvalidator	1 772 962	564ns
Valibot	4 153 651	240ns
Zod	2 053 469	486ns
Typia	9 518 755	105ns

source: https://github.com/icebob/validator-benchmark

Bundle Size

(Minified)

zod: 57kB

axios: 31kB

lodash: 70kB

@mui/material: 507kB

source: bundlephobia

Value of Type Safety

Does your application get data from external systems?

How much do you trust and/or control this external data?

How critical is it that your data conforms to its expected schema?

Recap

1. You can’t rely on TypeScript for type safety when dealing with data that only exists at runtime

2. Runtime validation can be used to re-achieve type safety, and you can use libraries to do the validation

3. You’re trading time and bundle size for better type safety - decide if it makes sense for your specific project

Consider using runtime validation in your projects!