Types at the edge of your system

Find and fix your TypeScript blind spots 🔦

Slides

THE VERGE TWITTER FACEBOOK

Mazda head units are getting bricked by a local NPR station in Seattle

This is why we need OTA updates

By Umar Shakir | Feb 9, 2022, 5:37pm EST | 24 comments

Frozen Mazda | Photo by Jakub Porzycki/NurPhoto via Getty Images

Some Mazda drivers in and around Seattle this week discovered they could no longer change the radio station or play anything else in their cars after listening to the local NPR station KUOW

Check the source

Or why it matters

Nicolas Carlo

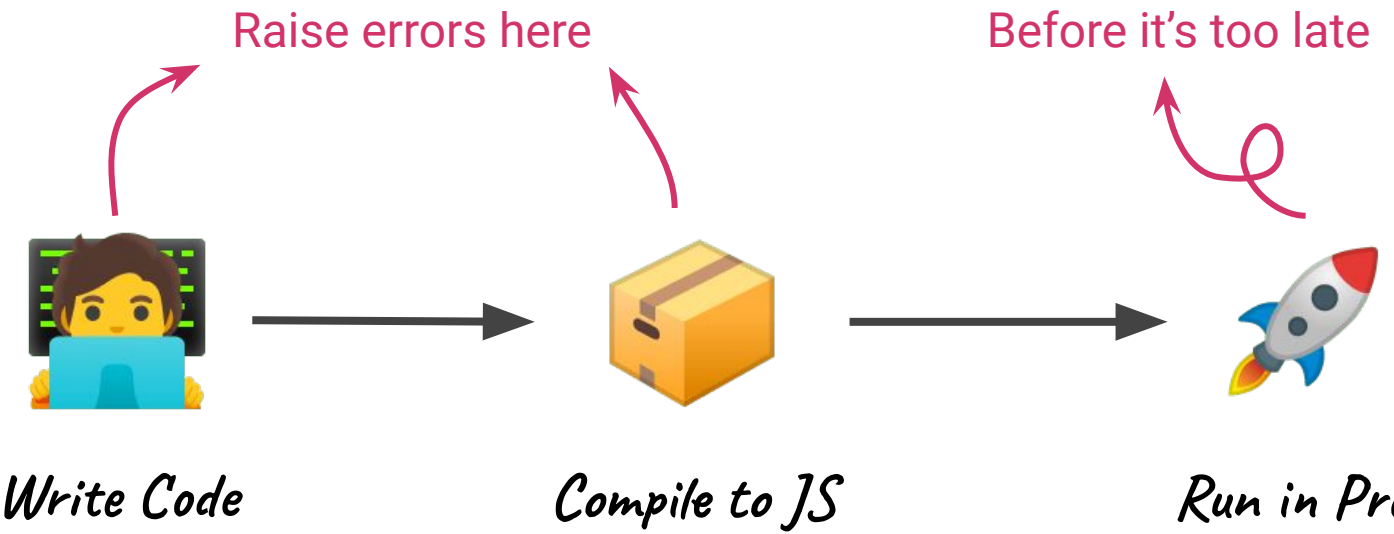
@nicoespeon

understandlegacycode.com

1 month *FREE* `getitdone2022`

TS is great, but…

TS catches type errors at compilation time



Yet, you may still have type errors in Prod with TS

Issues > PROD-APP

TypeError ai([native code])

Cannot read properties of null (reading 'toFixed')

Resolve | Ignore | Mark Reviewed | Open in Discover | Share

“Wait, how is that possible?!!”
— Confused developers

There are blind spots in your TS code 

Examples of blind spots 🙈

`any`

`@ts-ignore`

Wrong types *(eg.* `array[0]`*)*

```
function doSomething(items: Array<Item>) {
  const firstItem = items[0]
    // ^? const firstItem: Item
```

❌

Most of these are solved with TS itself

`any` → unknown

`@ts-ignore` → `@ts-expect-error`

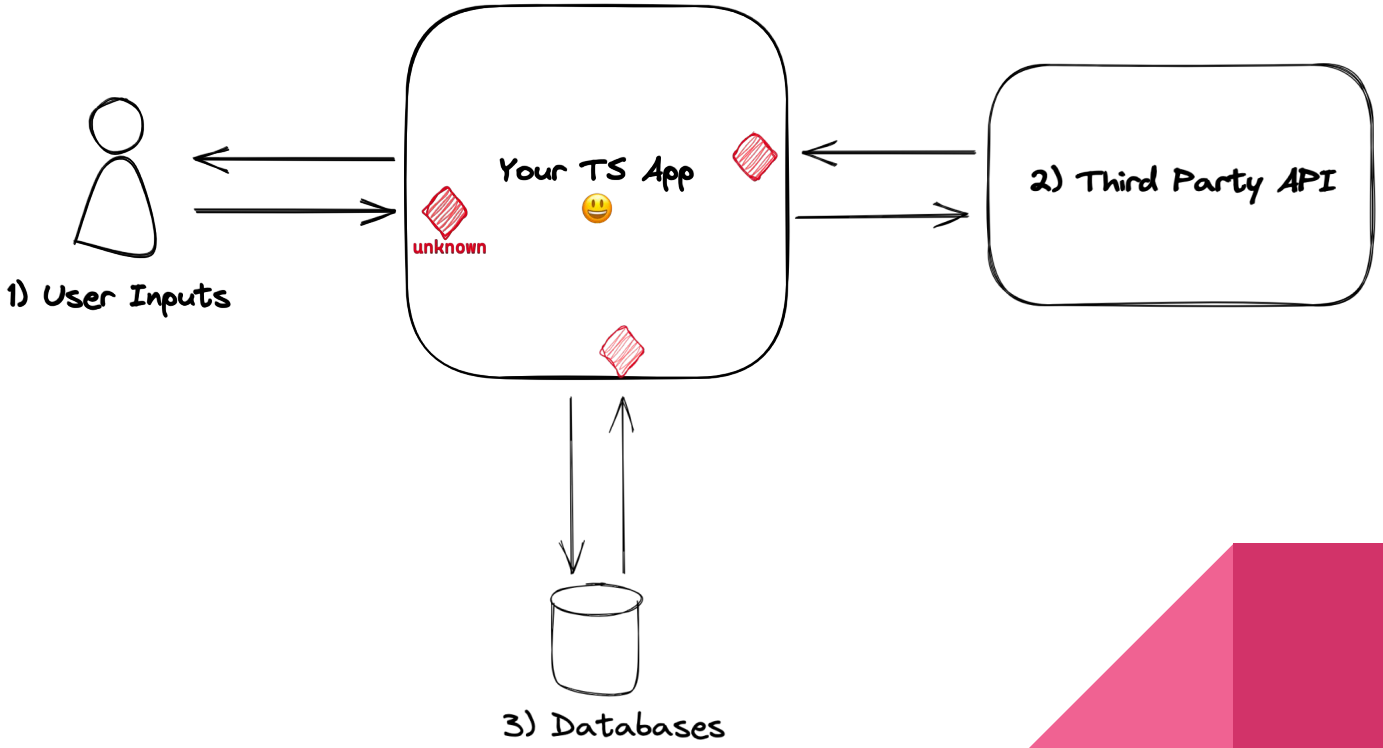
Wrong types (*eg.* `array[0]`)

```
function doSomething(items: Array<Item>) {
  const firstItem = first(items)
    // ^? const firstItem: Item | undefined ✅
```

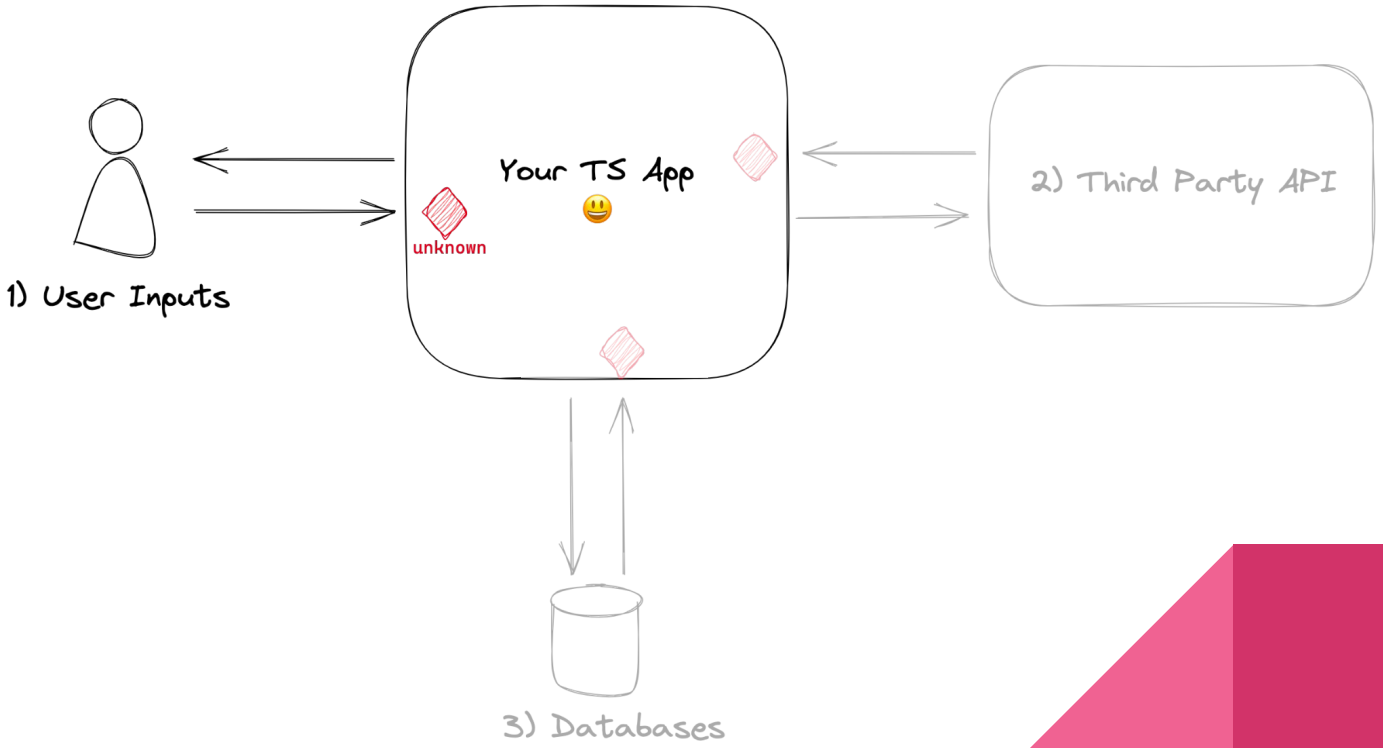
OK...

but is it frequent?

Yes, you have more wrong types than you think…

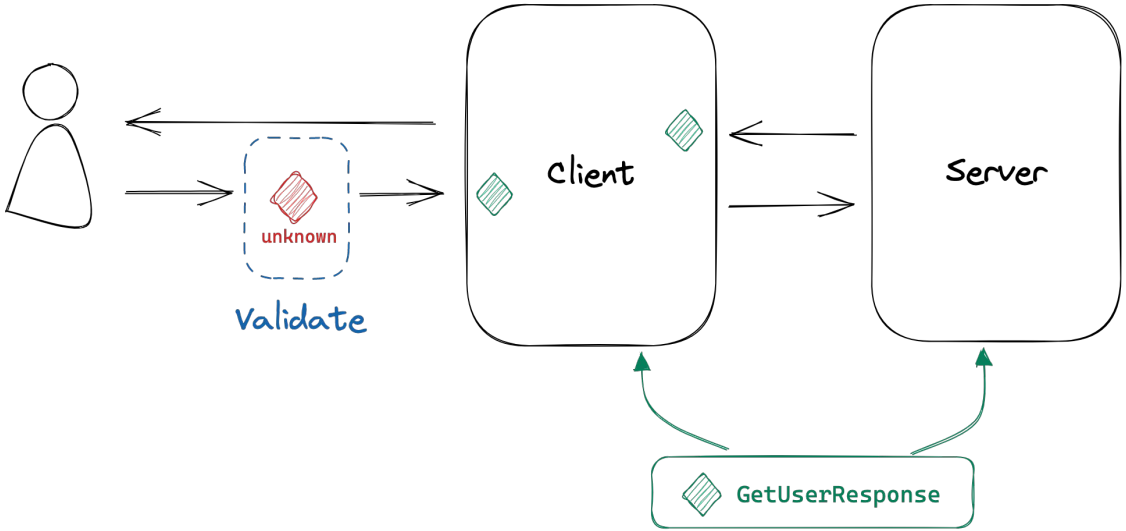


(1) User Inputs



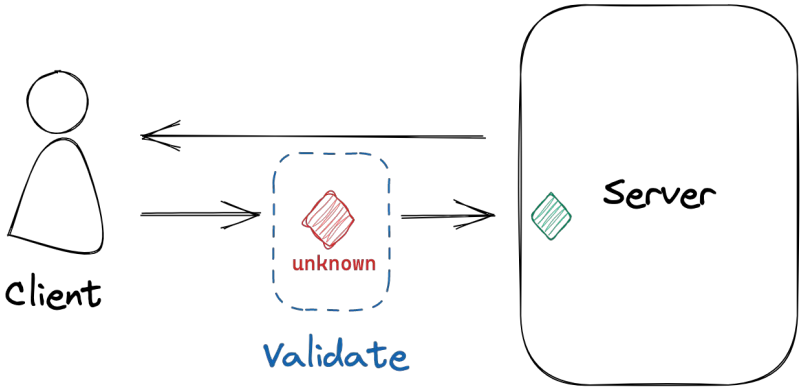
(1A) User inputs, you handle the client & the server

1. Validate user inputs on the client
2. Share API types (eg. Advanced TypeScript Patterns: API Contracts)



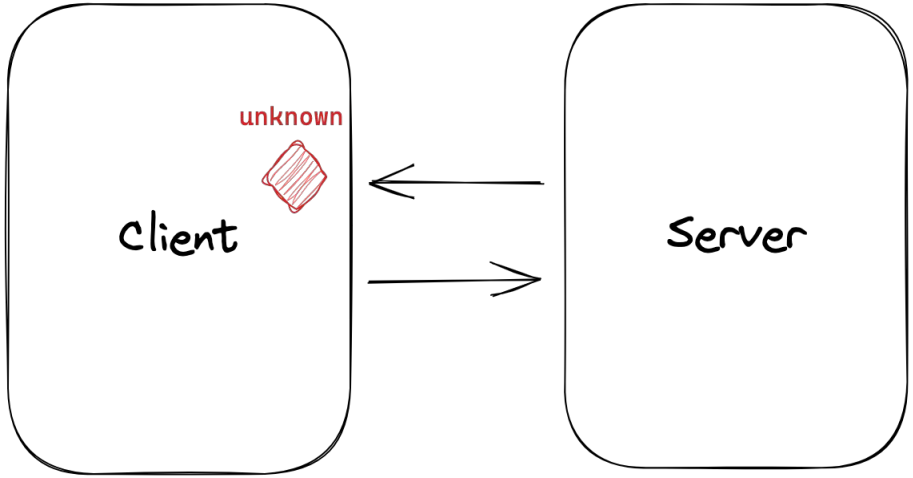
(1B) User inputs, you handle the server

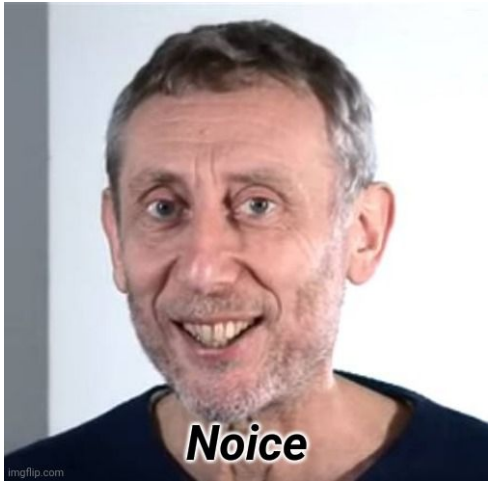
1. Validate user inputs



"How to keep types & validation in sync?"
"And docs?"

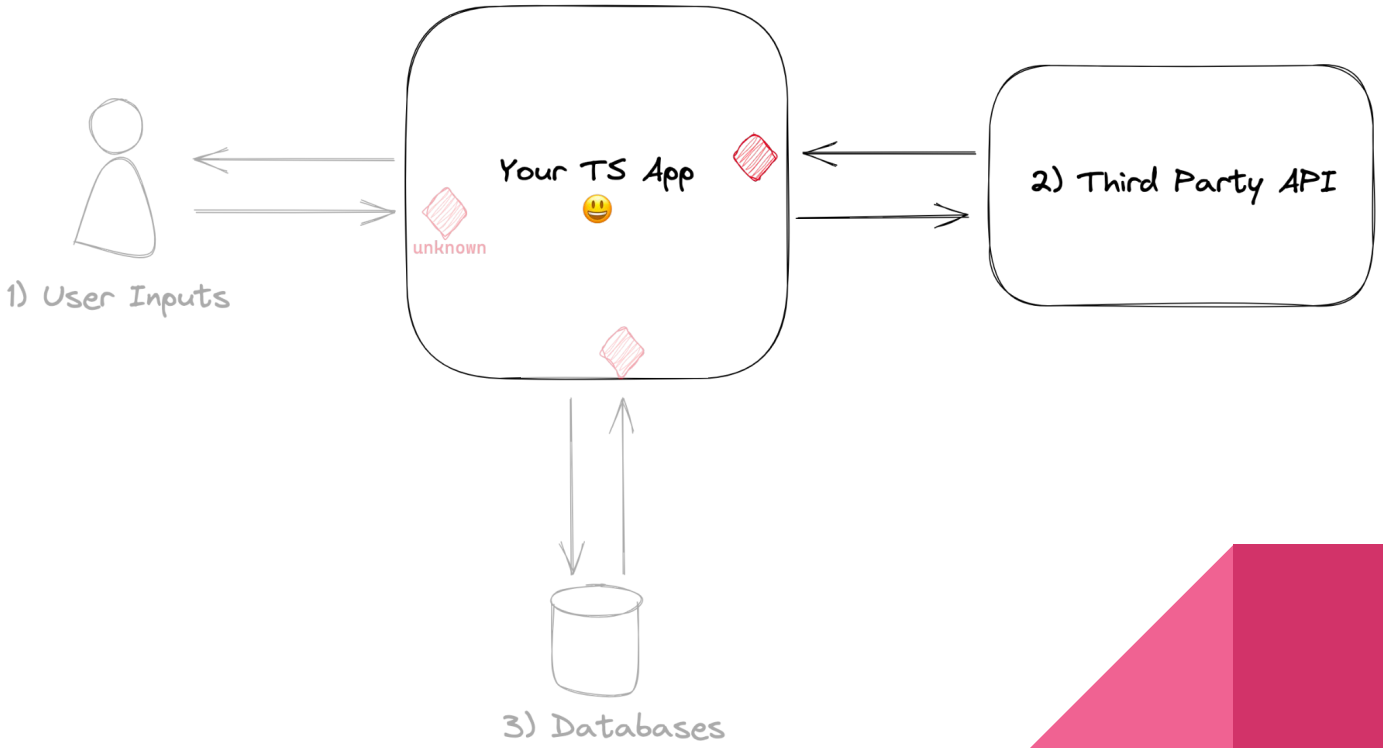
(1C) User inputs, you handle the client





💡 This is the SAME situation as **2) Third Party APIs**

(2) Third Party APIs



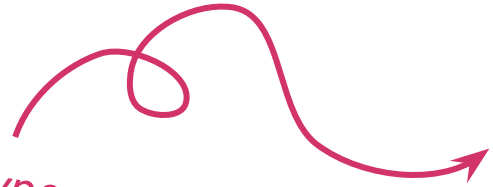
(2) Third Party APIs have different levels of trust


You have a typed SDK


You have online documentation

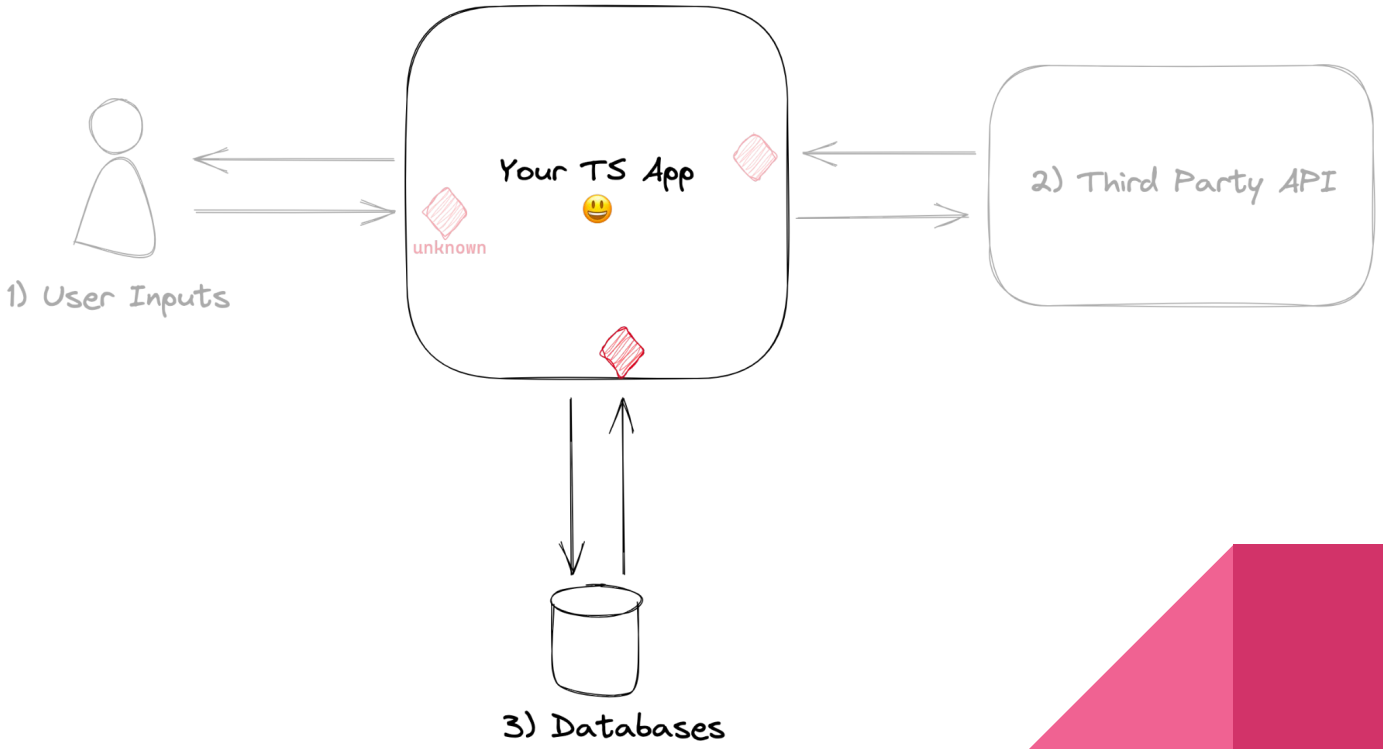

You *some form* of documentation


You don't have up-to-date docs



"How do you ensure types are the ones you expect?"

(2) Databases


Your TS App
😃
unknown
1) User Inputs
2) Third Party API
3) Databases

(3) Databases seem trustworthy. But wait…

- ❏ Are there **enforced** schemas for the stored data?
- ❏ Are the schemas **connected** to your type definitions?
- ❏ Can people change the data **without using your application**?
- ❏ Are there scripts that may update the data **without using your application**?
- ❏ Are there other applications that can update the data?

"How to know if fetched data have the expected format?"

Data entering your system may not respect your type definitions 😬

Tackling TS blind spots

1. Validate entering data
2. Generate types from validation schemas

One tool that can help: zod

github.com/colinhacks/zod

“TypeScript-first schema validation with static type inference”

```
import { z } from "zod";

// 1. Create a schema
const mySchema = z.string();

// 2a. Parse
mySchema.parse("tuna"); // => "tuna"
mySchema.parse(12); // => throws ZodError

// 2b. Safe parsing (doesn't throw error if validation fails)
mySchema.safeParse("tuna"); // => { success: true; data: "tuna" }
mySchema.safeParse(12); // => { success: false; error: ZodError }
```

One tool that can help: zod

github.com/colinhacks/zod

“TypeScript-first schema validation with static type inference”

```
import { z } from "zod";


const userSchema = z.object({
  username: z.string(),
});


// 3. Extract the inferred type
type User = z.infer<typeof userSchema>;
//   ^? { username: string }
```

Answering our questions 

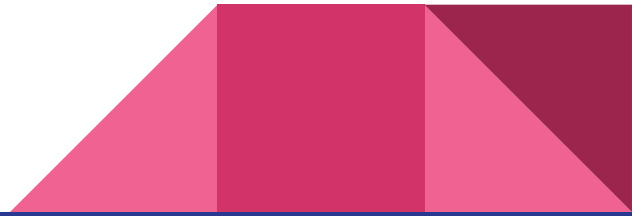
“How to keep types & validation in sync?”

👉 Generate the types from the validation schema.

```
import { z } from "zod";


export const userSchema = z.object({
  username: z.string(),
});


export type User = z.infer<typeof userSchema>;
```



“And docs?”

Generate docs, types and schemas from OpenAPI specs

Tools examples:

- ➔ swagger-api/swagger-ui for interactive docs from OpenAPI specs
- ➔ drwpow/openapi-typescript + fabien0102/ts-to-zod
- ➔ nelsongomes/ts-openapi (using Joi, alternative to Zod)

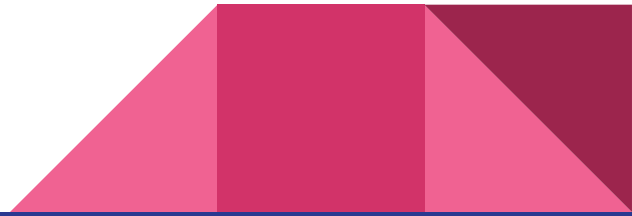
“How do you ensure types are the ones you expect?”

Type `unknown` the responses and validate the data match your expectations

```
const response = await fetch("https://api.github.com/users/octocat");
const data = await response.json() as unknown;

const result = userSchema.safeParse(data);
if (!result.success) {
 throw new ValidationError(result.error);
}

const user = result.data;
// ^? { name: string }
```



“How to know fetched data have the expected format?”

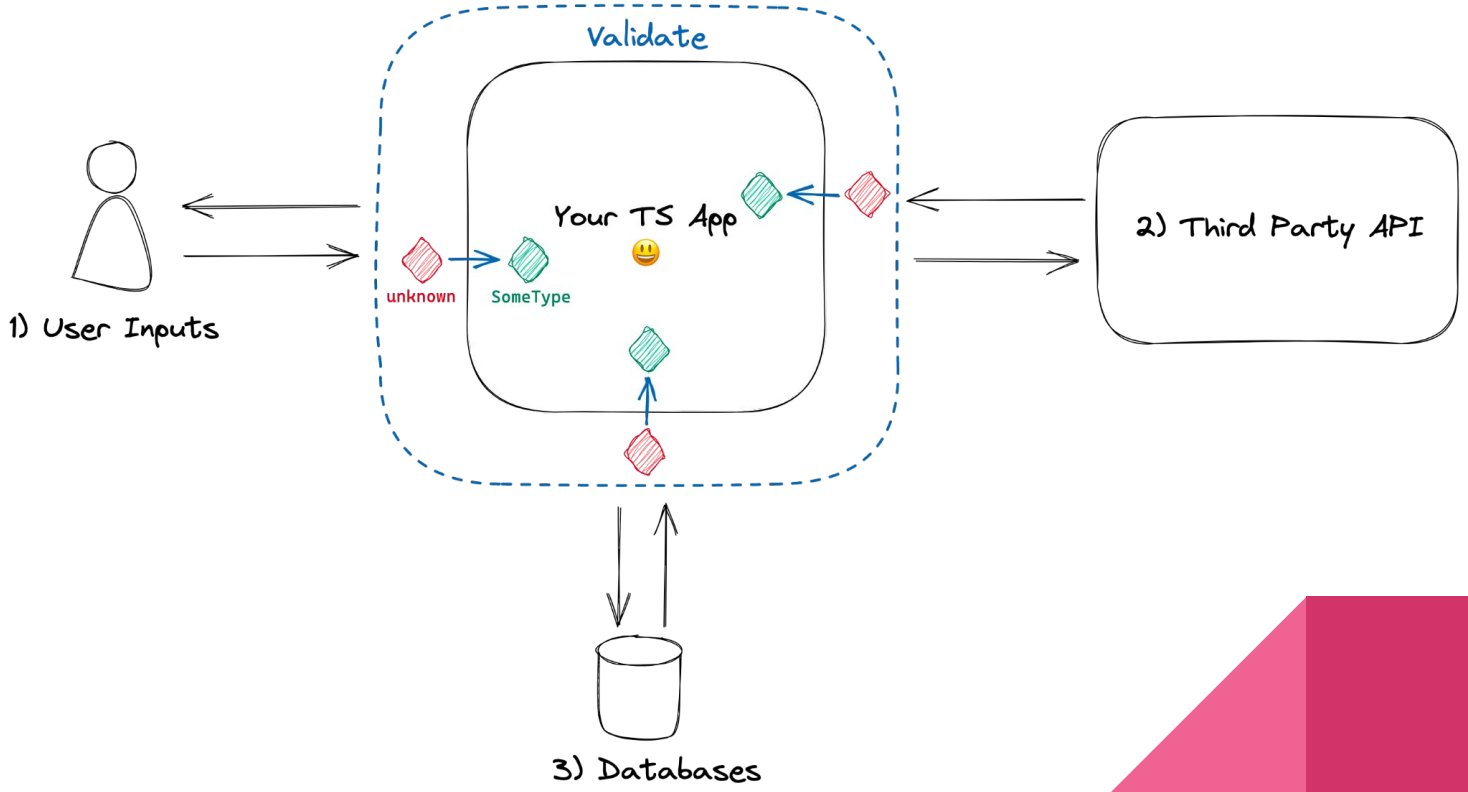
Type `unknown` the fetched data and validate the data match your expectations

```
export async function findUser(id: string): Promise<User> {
 const data = await db.doc(`users/${id}`).get();

 const result = userSchema.safeParse(data);
 if (!result.success) {
   throw new ValidationError(result.error);
 }

 return result.data;
}
```

Your TS app is now safe from type errors!



Don’t type
what you don’t own

1. Validate entering data
2. Generate types from validation schemas

Frequent Questions

Should we do that at all the edges?

It Depends™. Probably not.

How confident you are about these types?

Did you have type errors because you received unexpected data?

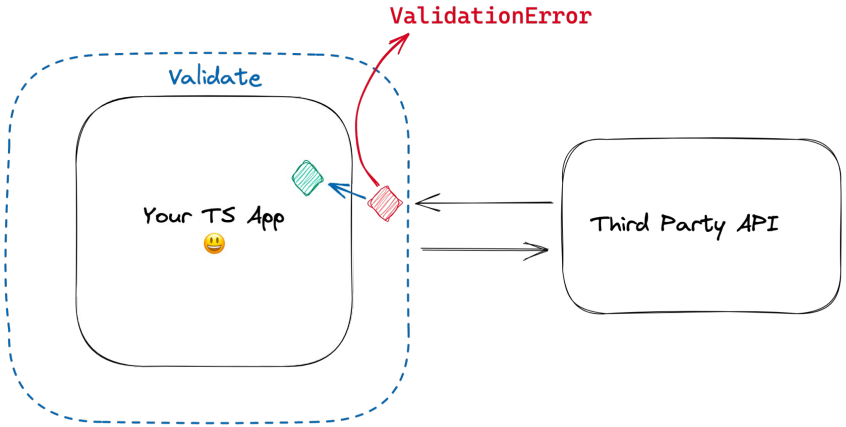
Start with the riskiest part of your app. Iterate.

Will we get fewer runtime errors?

Yes!

You may see more errors at first -> your expectations mismatch reality

Before, these would have gone *ignored* until very late.



To dig further

Related and useful resources

Slides at bit.ly/typing-the-edges

🤔 Why `array[0]` doesn't return undefined by default?

📝 Parse, don't validate

📝 Advanced TS Patterns: API Contracts

🍿 Fixing TypeScript's Blindspot (~15')

colinhacks/zod

TS to Zod (online)

fabien0102/ts-to-zod