sugarpirate_
poteto

Learning to Love Type Systems

Swipe Left, Uncaught TypeError 💔

PRESENTED BY
Lauren Tan (she/her)

Cinemagraph by /u/orbojunglist

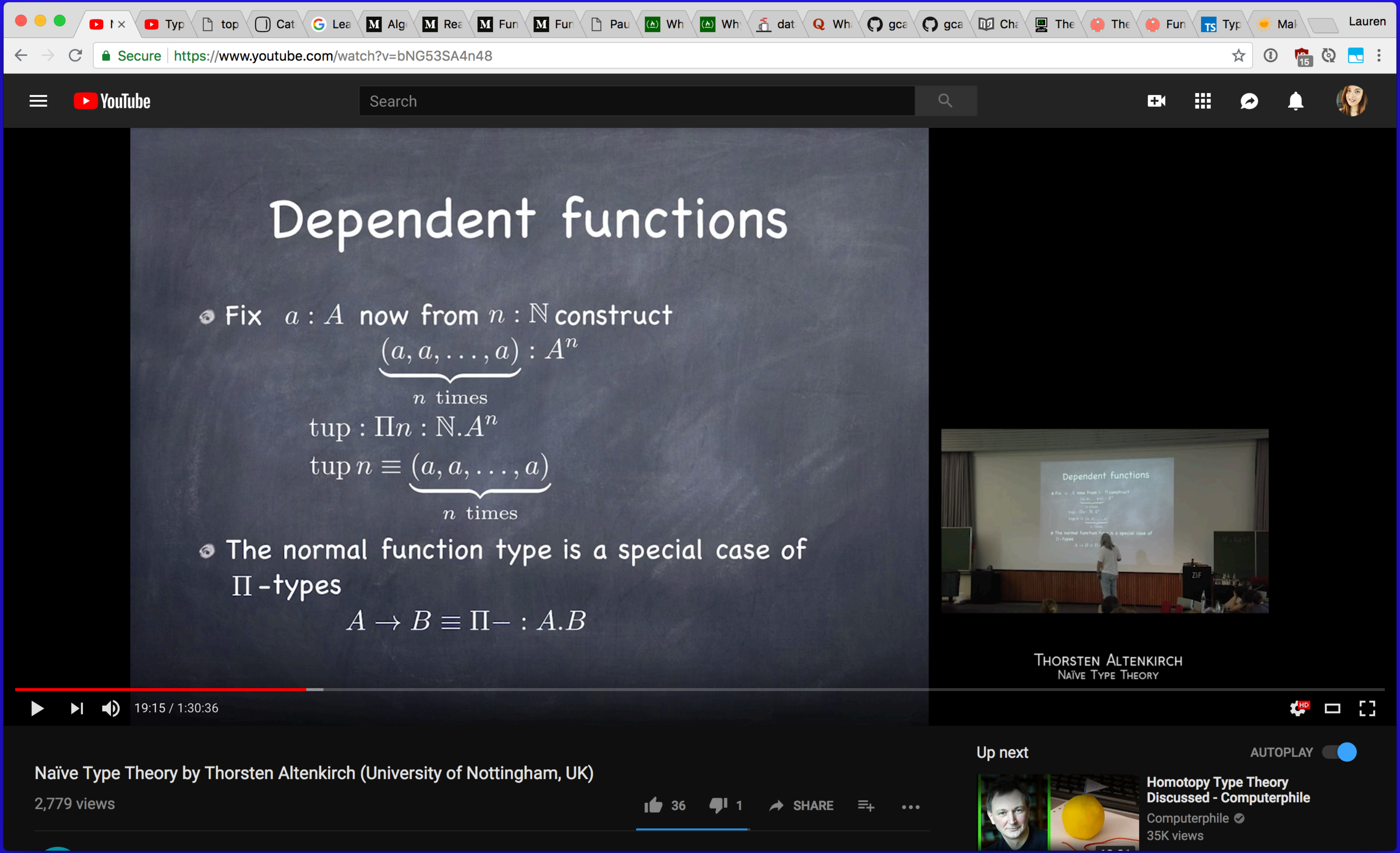
sugarpirate_
poteto

Learning to Love Type Systems

Swipe Left, Uncaught TypeError 💔

PRESENTED BY
Lauren Tan (she/her)

Cinemagraph by /u/orbojunglist



https://www.youtube.com/watch?v=bNG53SA4n48

Dependent functions

- Fix $a : A$ now from $n : \mathbb{N}$ construct
$$\underbrace{(a, a, \ldots, a)}_{n \text{ times}} : A^n$$
$$\text{tup} : \Pi n : \mathbb{N}.A^n$$
$$\text{tup}\, n \equiv \underbrace{(a, a, \ldots, a)}_{n \text{ times}}$$
- The normal function type is a special case of Π-types
$$A \to B \equiv \Pi - : A.B$$

Thorsten Altenkirch
Naïve Type Theory

19:15 / 1:30:36

Naïve Type Theory by Thorsten Altenkirch (University of Nottingham, UK)

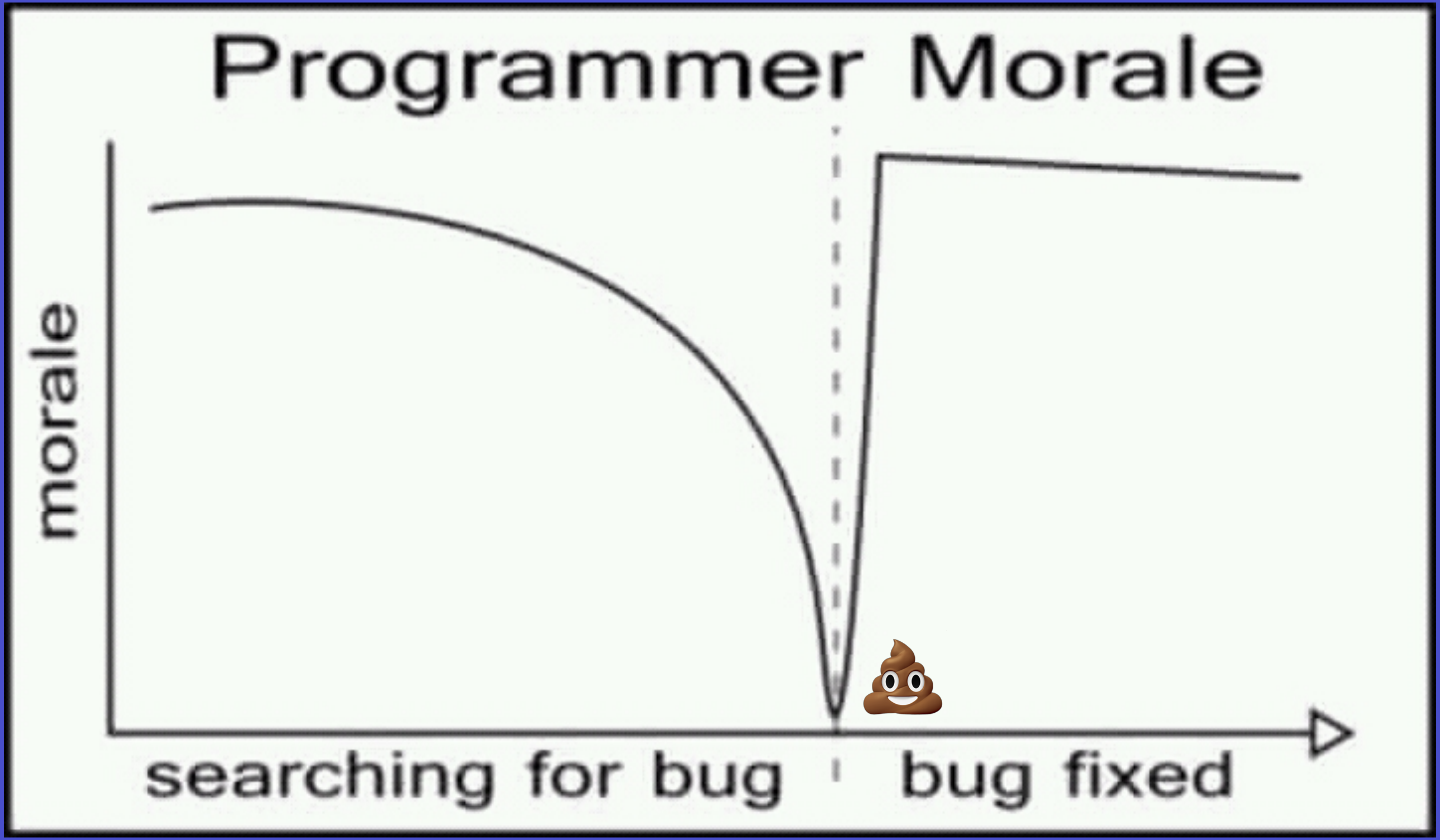
2,779 views

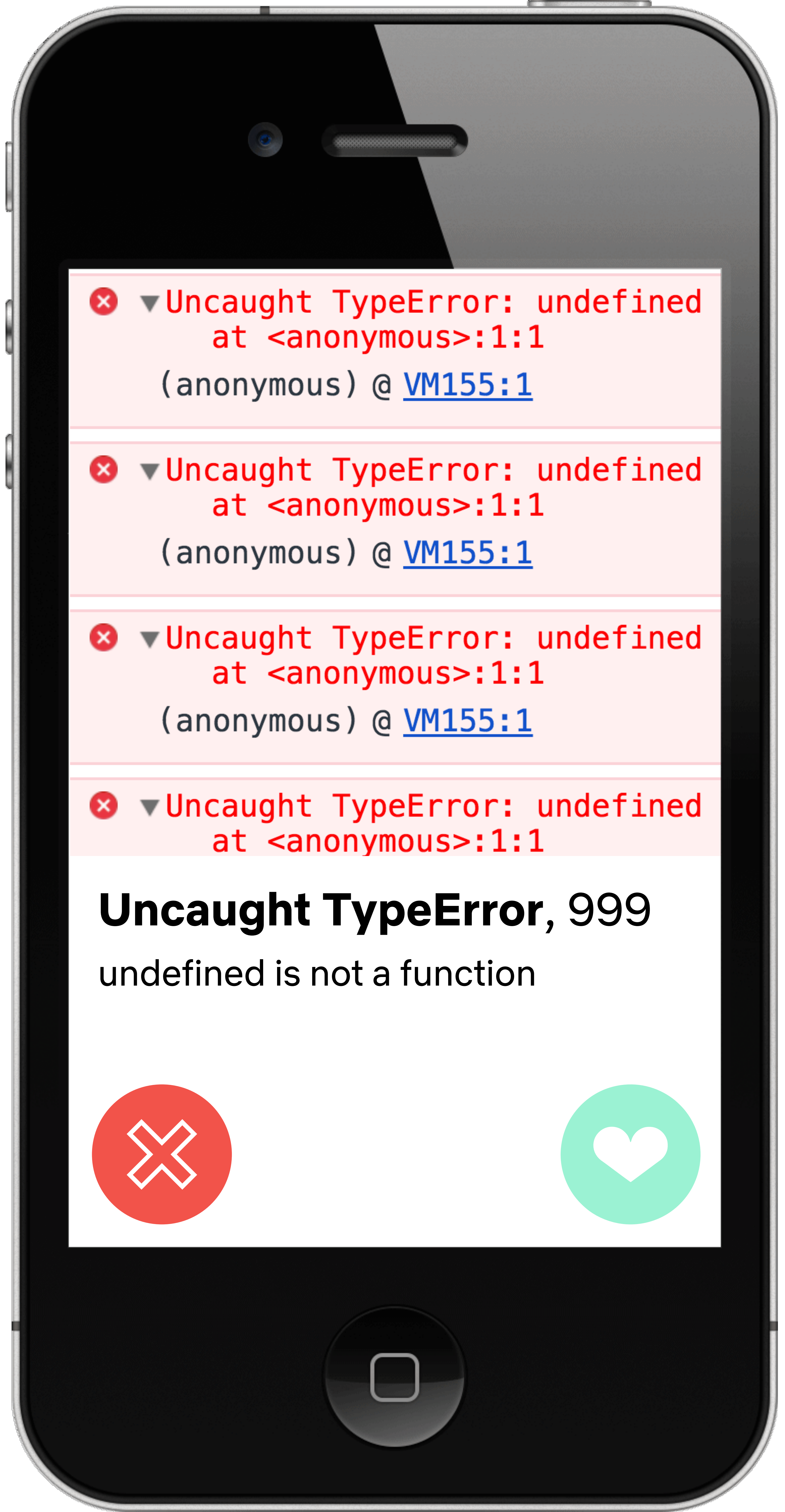
cos
(t − t₀)
4K(1 + K)
p_K
p_SF
21 = (A + η)² + κ² and

$\sum$
$\sqrt{\frac{4K(1+K)}{p_{SF}}p_K}$
$21 = (A + \eta)^2 + \kappa^2$ and

"If debugging is the process of removing software bugs, then programming must be the process of putting them in."

Djikstra, supposedly 🤷🏻‍♀️

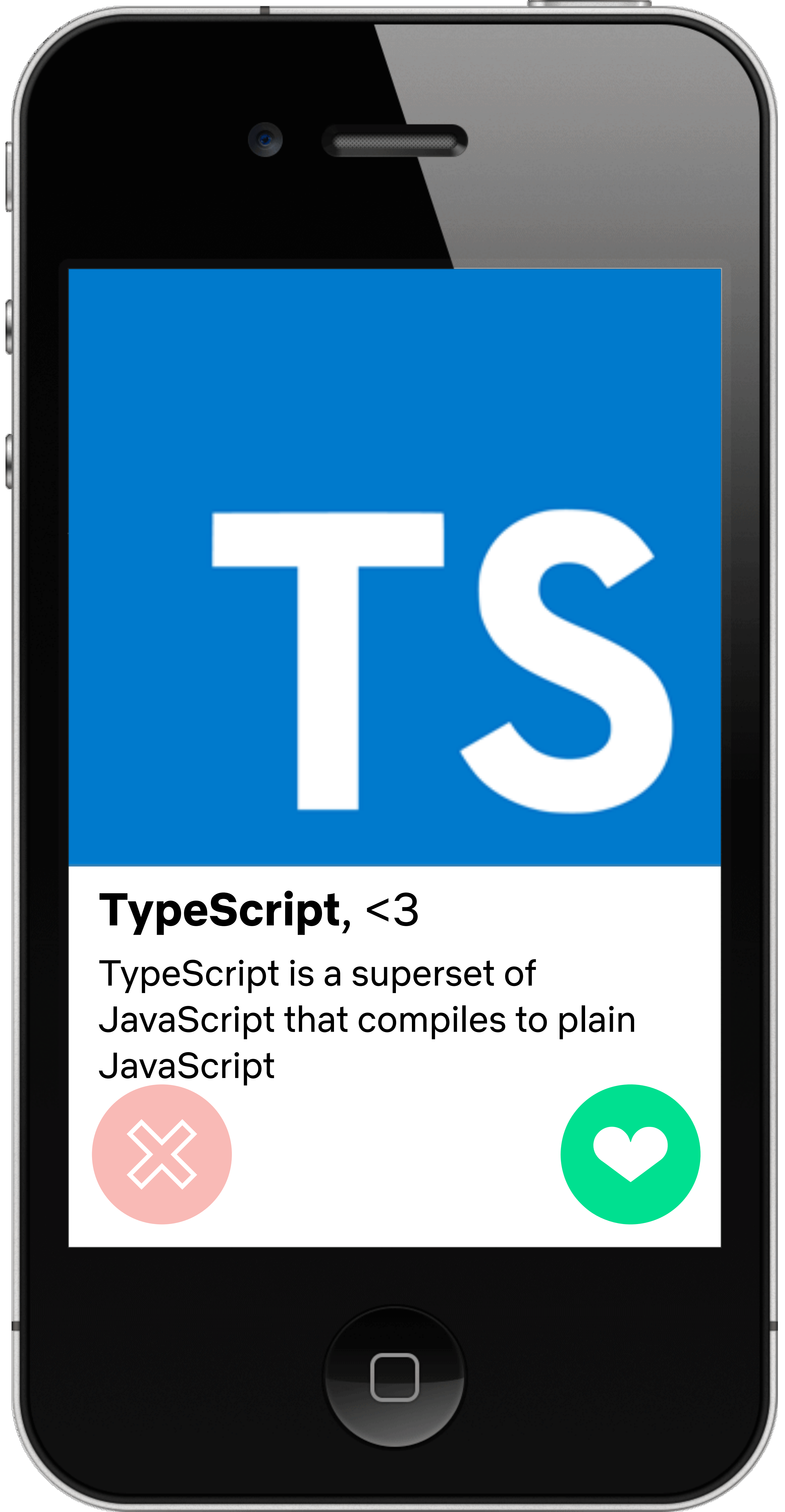

Programmer Morale
morale
searching for bug
bug fixed



```
▼Uncaught TypeError: undefined
    at <anonymous>:1:1
  (anonymous) @ VM155:1
▼Uncaught TypeError: undefined
    at <anonymous>:1:1
  (anonymous) @ VM155:1
▼Uncaught TypeError: undefined
    at <anonymous>:1:1
  (anonymous) @ VM155:1
▼Uncaught TypeError: undefined
    at <anonymous>:1:1
```

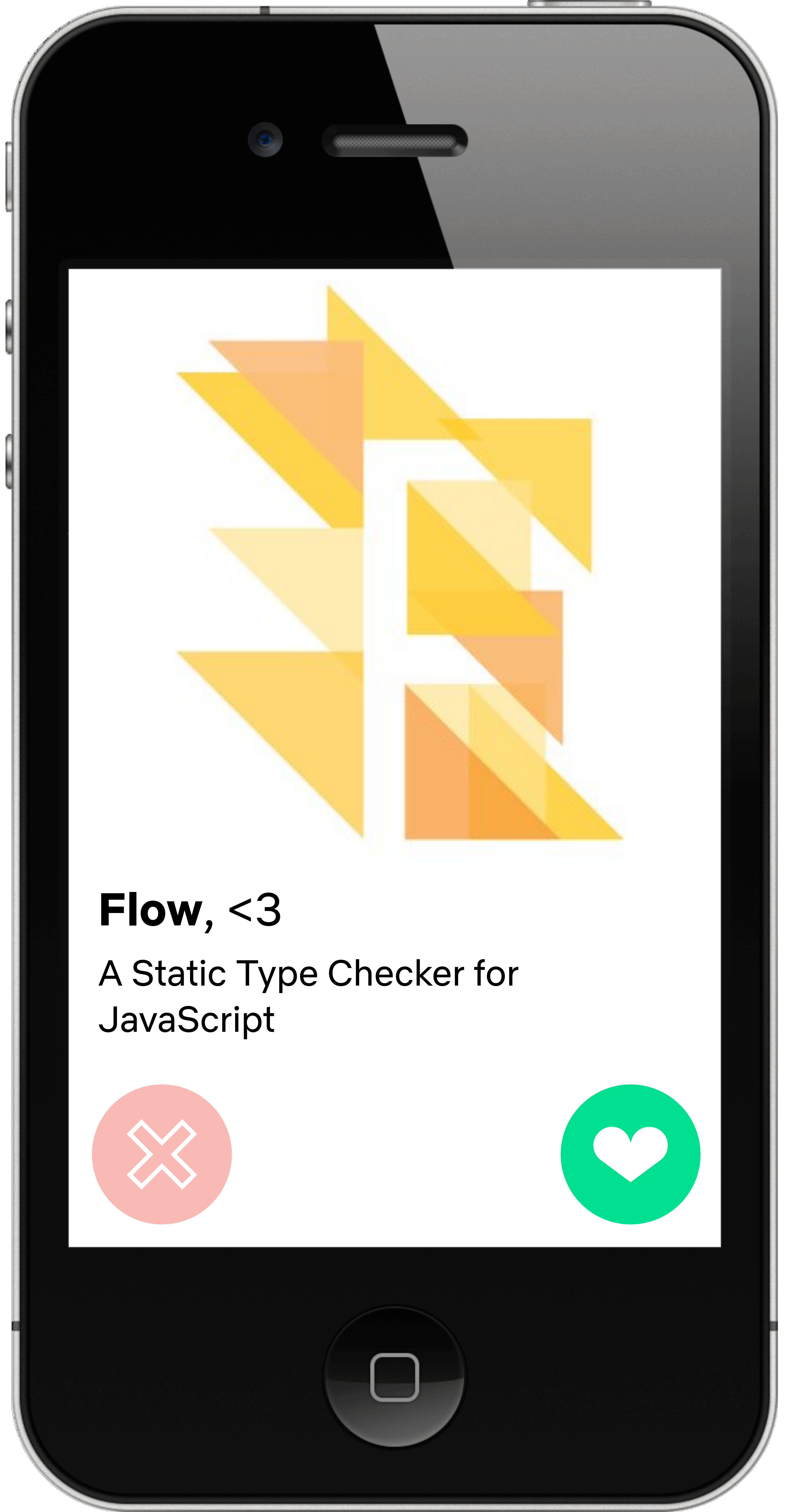
Uncaught TypeError, 999

undefined is not a function



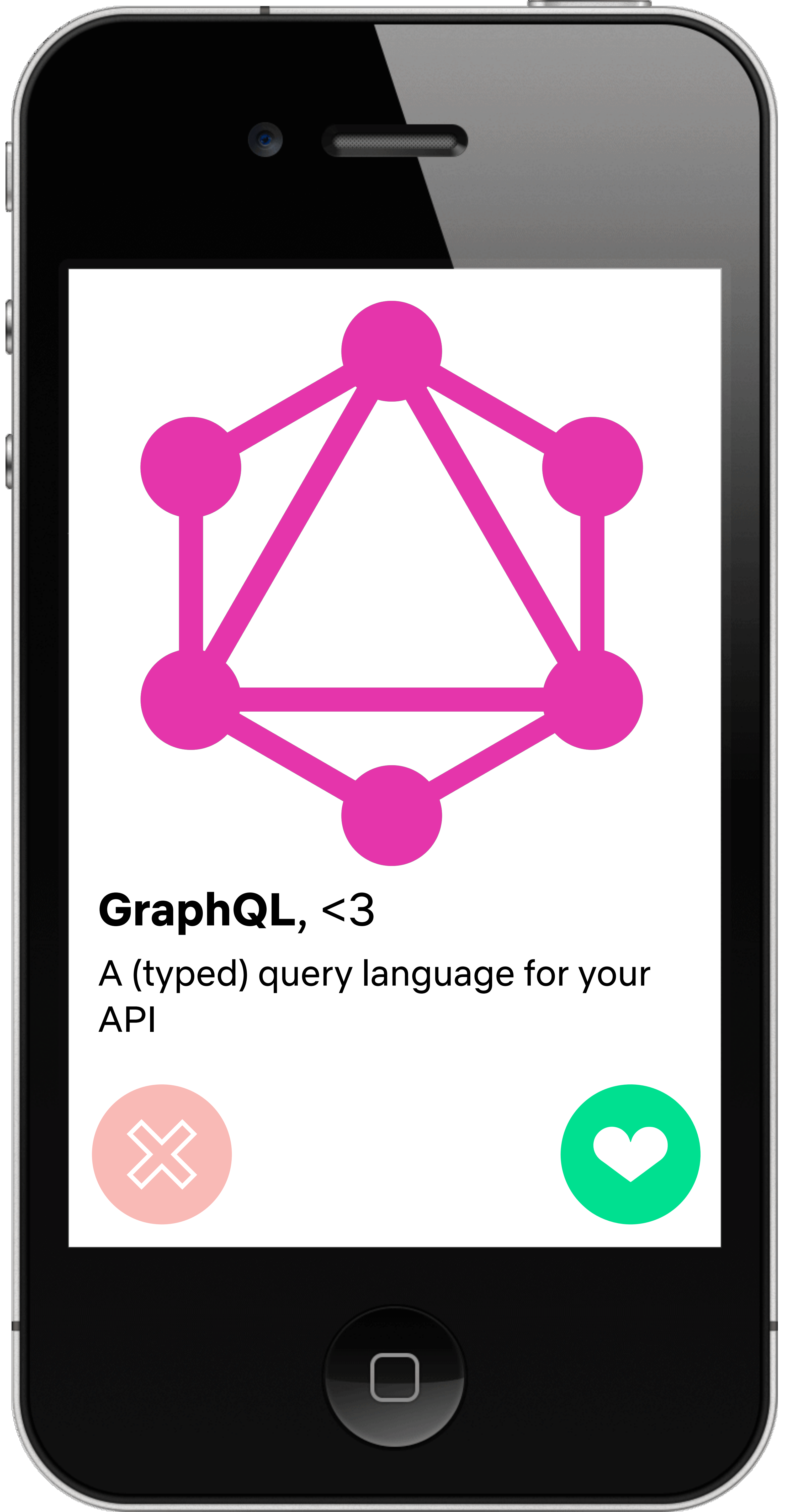
TypeScript, <3

TypeScript is a superset of JavaScript that compiles to plain JavaScript



Flow, <3

A Static Type Checker for JavaScript



GraphQL, <3

A (typed) query language for your API

Lauren Tan

sugarpirate_

poteto

"A type system is a tractable syntactic method for **proving the absence of certain program behaviors** *by classifying phrases according to the kinds of values they compute."*

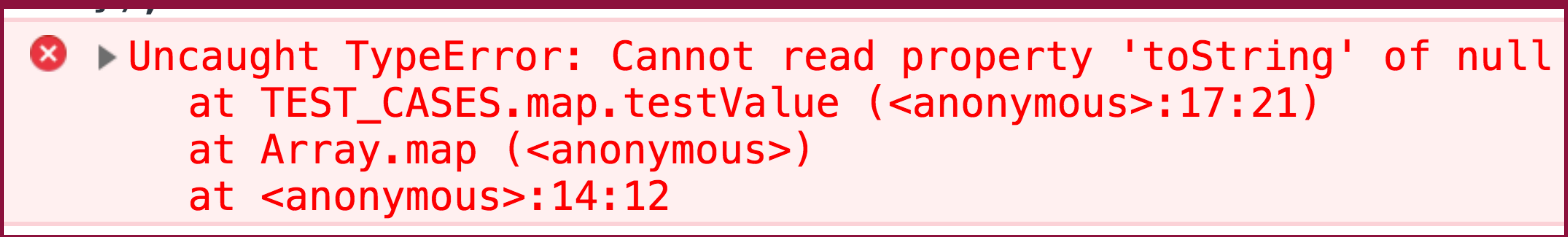
Types and Programming Languages, Benjamin C. Pierce

How many ways can this program fail?

```
const half = x => x / 2;
const TEST_CASES = [
  null,
  undefined,
  Symbol(1),
  10,
  '10',
  'hello world',
  { name: 'Lauren' },
  [1, 2, 3],
  x => x * x
];
```

```
TEST_CASES.map(testValue ⇒ {
  return {
    result: half(testValue),
    test: testValue.toString()
  }
});
```

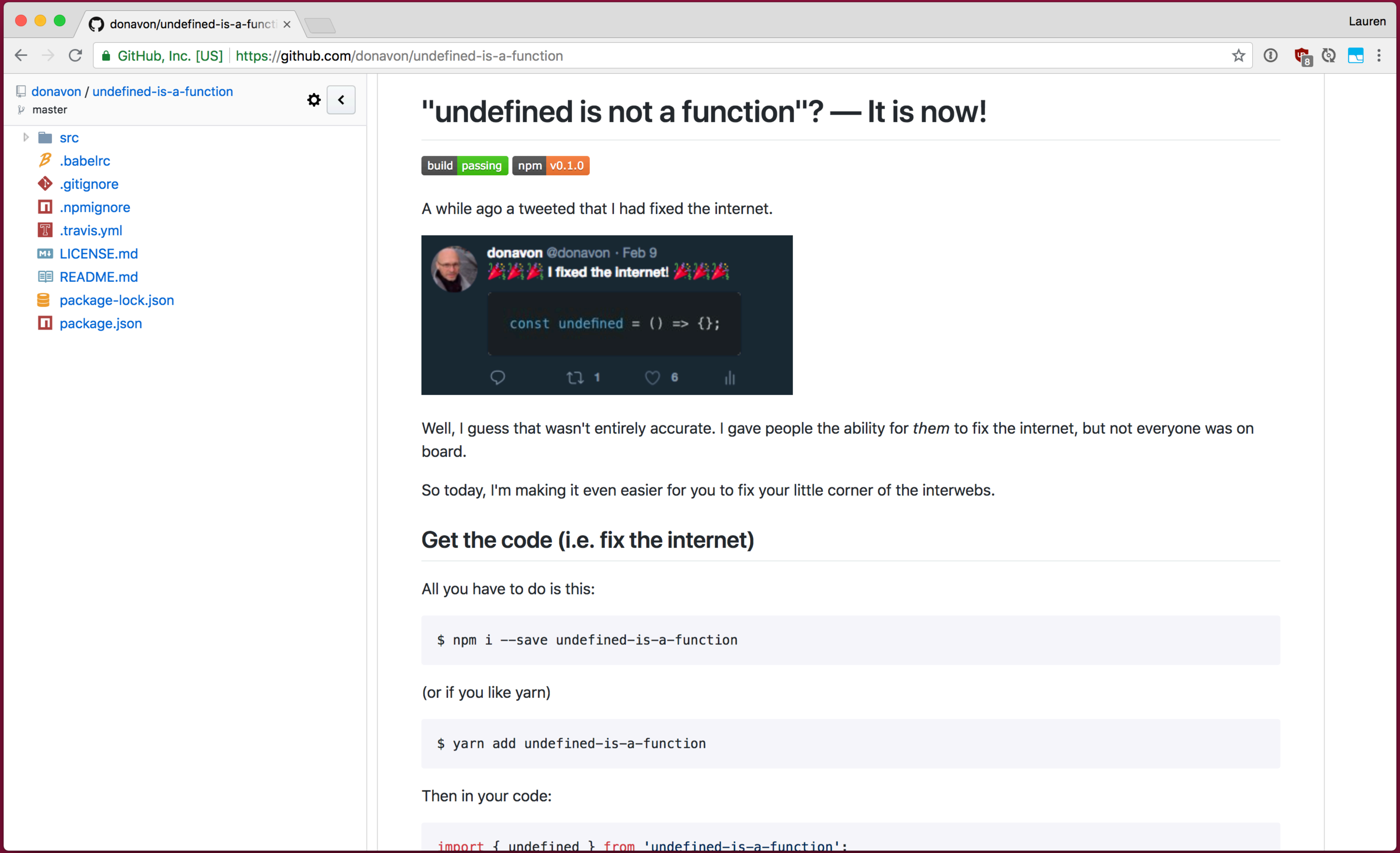
not my type!



```
Uncaught TypeError: Cannot read property 'toString' of null
    at TEST_CASES.map.testValue (<anonymous>:17:21)
    at Array.map (<anonymous>)
    at <anonymous>:14:12
```

OK

OK



donavon/undefined-is-a-funct ×

Lauren

GitHub, Inc. [US] https://github.com/donavon/undefined-is-a-function

donavon / undefined-is-a-function
master

- src
- .babelrc
- .gitignore
- .npmignore
- .travis.yml
- LICENSE.md
- README.md
- package-lock.json
- package.json

"undefined is not a function"? — It is now!

build passing npm v0.1.0

A while ago a tweeted that I had fixed the internet.

donavon @donavon · Feb 9
🎉🎉🎉 I fixed the internet! 🎉🎉🎉

```
const undefined = () => {};
```

1 6

Well, I guess that wasn't entirely accurate. I gave people the ability for *them* to fix the internet, but not everyone was on board.

So today, I'm making it even easier for you to fix your little corner of the interwebs.

Get the code (i.e. fix the internet)

All you have to do is this:

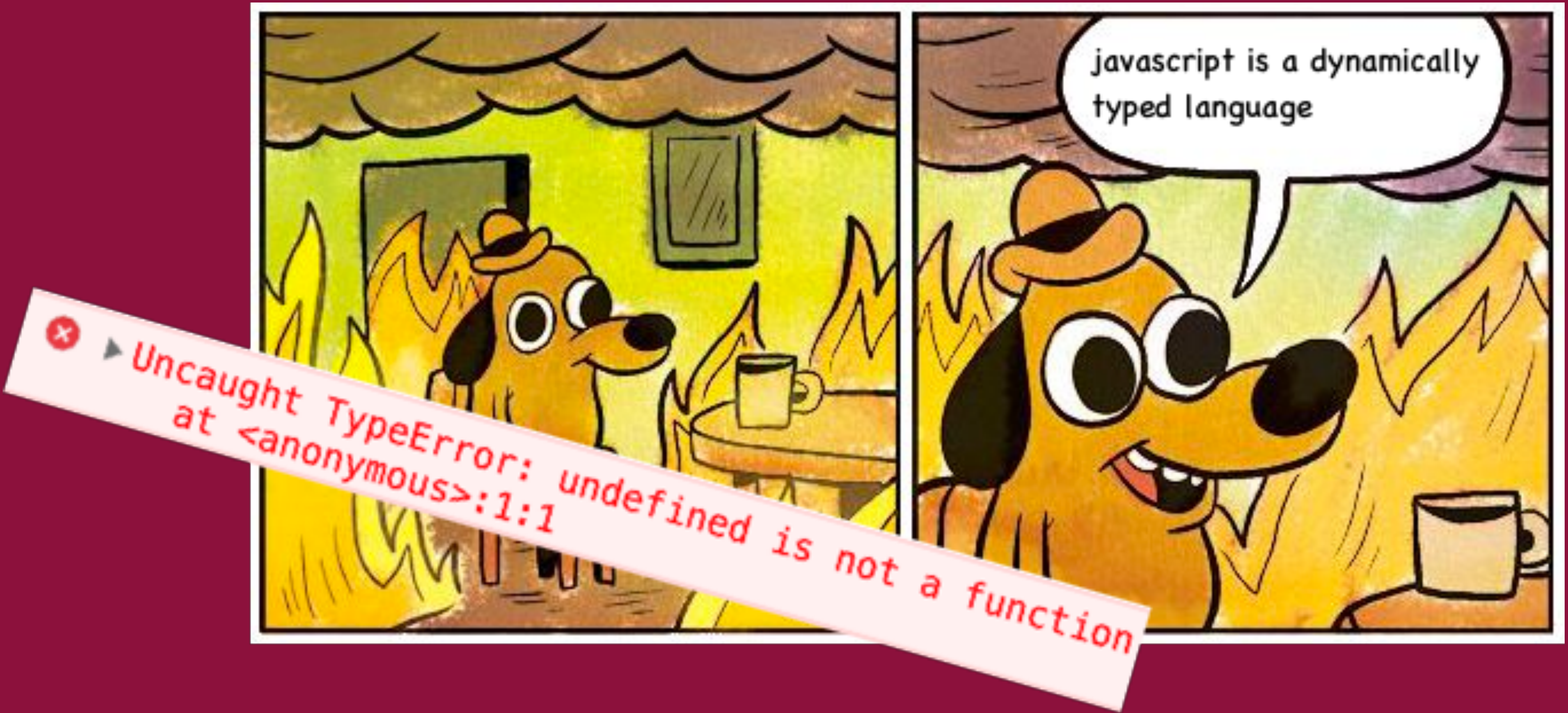
```
$ npm i --save undefined-is-a-function
```

(or if you like yarn)

```
$ yarn add undefined-is-a-function
```

Then in your code:

```
import { undefined } from 'undefined-is-a-function';
```



```
▶ Uncaught TypeError: undefined is not a function
    at <anonymous>:1:1
```

```javascript
const TEST_CASES = [
  null, // Uncaught TypeError
  undefined, // Uncaught TypeError
  Symbol(1), // Uncaught TypeError
  10, // 5
  '10', // 5 🤷🏻‍♀️
  'hello world', // NaN
  { name: 'Lauren' }, // NaN
  [1, 2, 3], // NaN
  x => x * x // NaN
];
```

How many ways can this program fail?

(Infinity)

How fail?

```
const half = (x: number) ⇒ x / 2;
```

How many ways can this program fail (at compile time)?

0*

Agenda

A Gentle Introduction to Types

Why Less is Better

Types Over the Network

A Gentle Introduction to Types

Why You Should Care About Types

A Gentle Introduction to Types

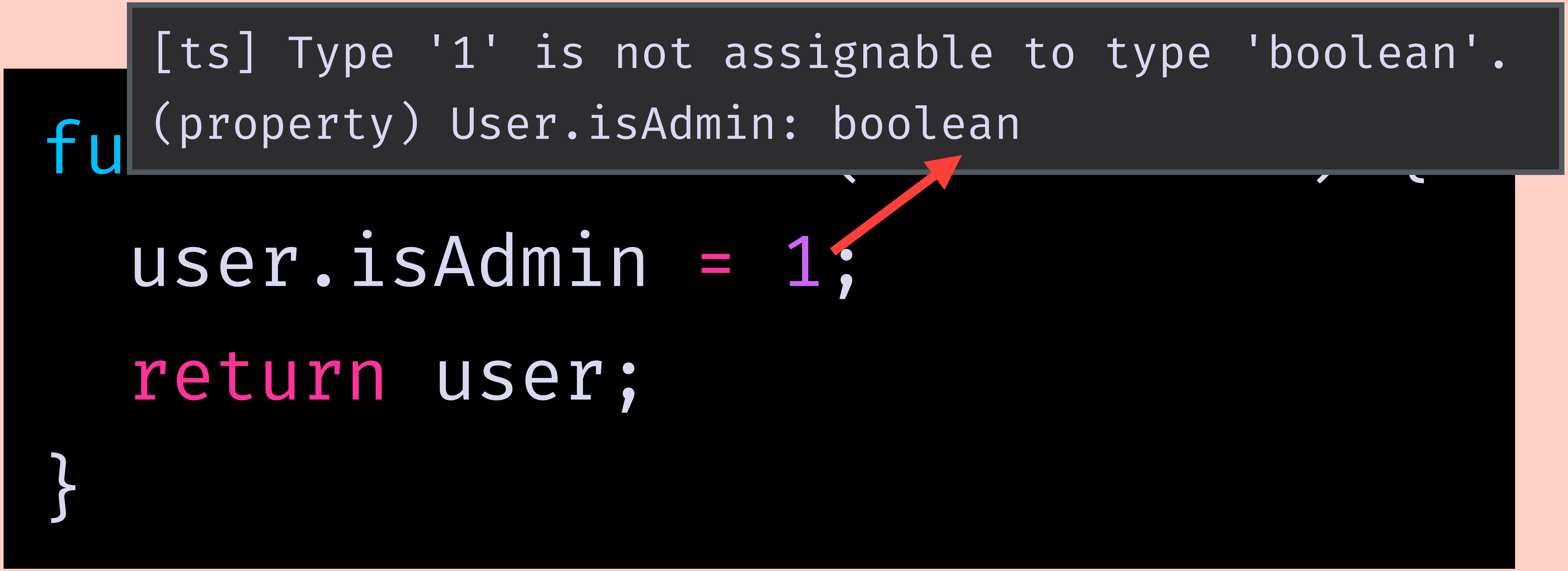
Why You Should Care About Types

The Basics

```
const text1 = 'hello react rally';
const text2: string = 'hello react rally';
const list1 = [1, 2, 3];
const list2: number[] = [4, 5, 6];
```

```
type ServerStacks = 'canary' | 'beta' | 'production'
interface User {
  id: number;
  name: string;
  isAdmin: boolean;
}
```

```
function makeAdmin(user: User) {
  user.isAdmin = true;
  return user;
}
```



```
[ts] Type '1' is not assignable to type 'boolean'.
(property) User.isAdmin: boolean
```

```
fu
  user.isAdmin = 1;
  return user;
}
```

Generics

(Parametric Polymorphism)

```
function makeArray(x: number): number[] { return [x]; }
function makeArray(x: string): string[] { return [x]; }
function makeArray(x: boolean): boolean[] { return [x]; }
function makeArray<T>(x: T): T[] { return [x]; }
function makeArray<T>(x: T): T[] { return [x]; }
function makeArray<T>(x: T): T[] { return [x]; }
function makeArray<T>(x: T): T[] { return [x]; }
function makeArray<number>(x: number): number[]
```

```
makeArray(1);
```

```
function makeArray<string>(x: string): string[]
```

```
makeArray('hello');
```

```
function map<A, B>(fn: (item: A) => B, items: A[]): B[] {
  // ...
}
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  // ...
}
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  // ...
}
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  // ...
}
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  // ...
}
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  // ...
}
```

```
map(x => x * x, [1, 2, 3]); // number[]
map(x => x.toUpperCase(), ['hello', 'react', 'rally']); // string[]
```

Why Less Is Better

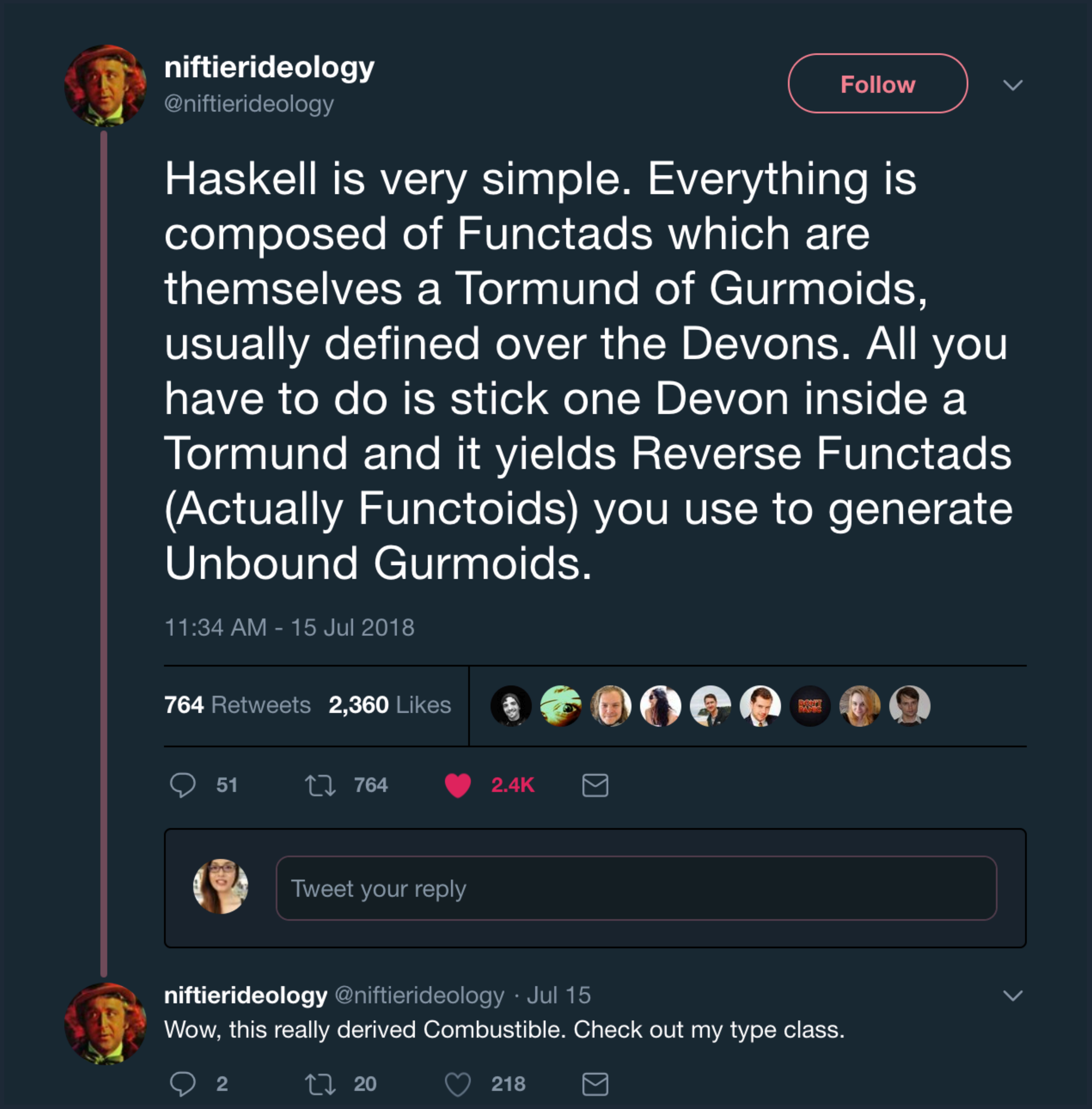
Precise Types Means Less Bugs

Why Less Is Better

Precise Types Means Less Bugs

Learning from Functional Programming

https://www.youtube.com/watch?v=ev7AYsLljxk&index=5&list=PL8Ky8lYL8-Oh7awp0sqa82o7Ggt4AGhyf



niftierideology
@niftierideology

Follow

Haskell is very simple. Everything is composed of Functads which are themselves a Tormund of Gurmoids, usually defined over the Devons. All you have to do is stick one Devon inside a Tormund and it yields Reverse Functads (Actually Functoids) you use to generate Unbound Gurmoids.

11:34 AM - 15 Jul 2018

764 Retweets 2,360 Likes

51 764 2.4K

Tweet your reply

niftierideology @niftierideology · Jul 15
Wow, this really derived Combustible. Check out my type class.

2 20 218

$$\begin{tikzcd}
& \begin{array}{c}\textbf{Proof theory}\\ \textit{Logic}\end{array} \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
\begin{array}{c}\textbf{Category theory}\\ \textit{Algebra}\end{array} \arrow[rr, leftrightarrow] & & \begin{array}{c}\textbf{Type theory}\\ \textit{Programs}\end{array}
\end{tikzcd}$$

Curry–Howard–Lambek correspondence

Stop with the jargon, Lauren

Stop with the jargon, Lauren

```
declare function Addition(x: number, y: number): number;      // proposition
function add(x: number, y: number): number { return x + y; }  // proof
```

```
Proposition: If x and y are numbers, a number exists
```

```
declare function Addition(x: number, y: number): number;      // proposition
function add(x: number, y: number): number { return x + y; }  // proof
```

Proposition: If x and y are numbers, a number exists

```
declare function Addition(x: number, y: number): number;      // proposition
function add(x: number, y: number): number { return x + y; }  // proof
```

Proof: x + y proves that a number exists

What is a function?

$$f : A \to B$$

$$\begin{tikzcd} a : A \arrow[r] & \boxed{f} \arrow[r] & b : B \end{tikzcd}$$

```
f :: function from type A to type B
```

$$\begin{tikzcd}
\text{an object* of type A} \arrow[r] & \boxed{f} \arrow[r] & \text{an object* of type B}
\end{tikzcd}$$

** not a JS object*

Types are propositions
Programs are proofs

Curry-Howard Correspondence

Let the type system suggest the implementation

```
function head<T>(list: T[]): T {
  // ...
}
```

```
[ts] Type 'T[]' is not assignable to type 'T'.
(parameter) list: T[]
```

```
function head<T>(list: T[]): T {
  return list;
}
```

```
function head<T>(list: T[]): T {
  return list[0];
}
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  // ...
}
```

```
[ts]
Type 'A[]' is not assignable to type 'B[]'.
  Type 'A' is not assignable to type 'B'.
(parameter) items: A[]
```

```
function map<A, B>(fn: (item: A) => B, items: A[]): B[] {
  return items;
}
```

```
[ts] Type 'B' is not assignable to type 'B[]'.
(parameter) fn: (item: A) ⇒ B
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  return fn(items[0]);
}
```

```
function map<A, B>(fn: (item: A) ⇒ B, items: A[]): B[] {
  return items.reduce((acc, curr) ⇒ {
    acc.push(fn(curr));
    return acc;
  }, [] as B[]);
}
```

```
myStatelessComponent :: Props → React.ReactNode
```

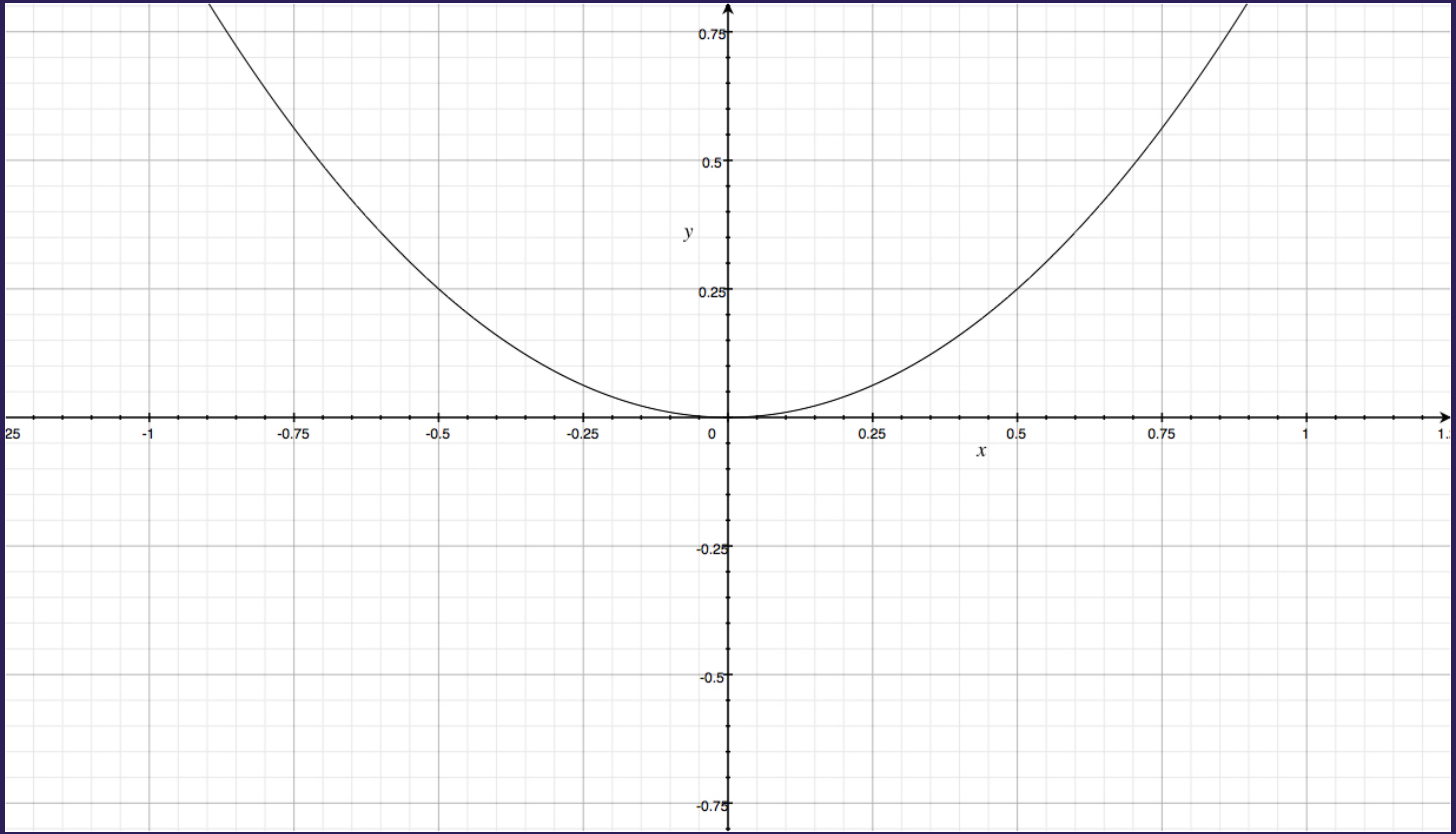
```
[ts] Type '1' is not assignable to type
'StatelessComponent<MyProps>'.
const MyStatelessComponent: React.StatelessComponent<MyProps>
```

```
const MyStatelessComponent: React.SFC<MyProps> = 1;
```

```
[ts]
Type '() ⇒ number' is not assignable to type
'StatelessComponent<MyProps>'.
  Type 'number' is not assignable to type 'ReactElement<any>'.
const MyStatelessComponent: React.StatelessComponent<MyProps>
```

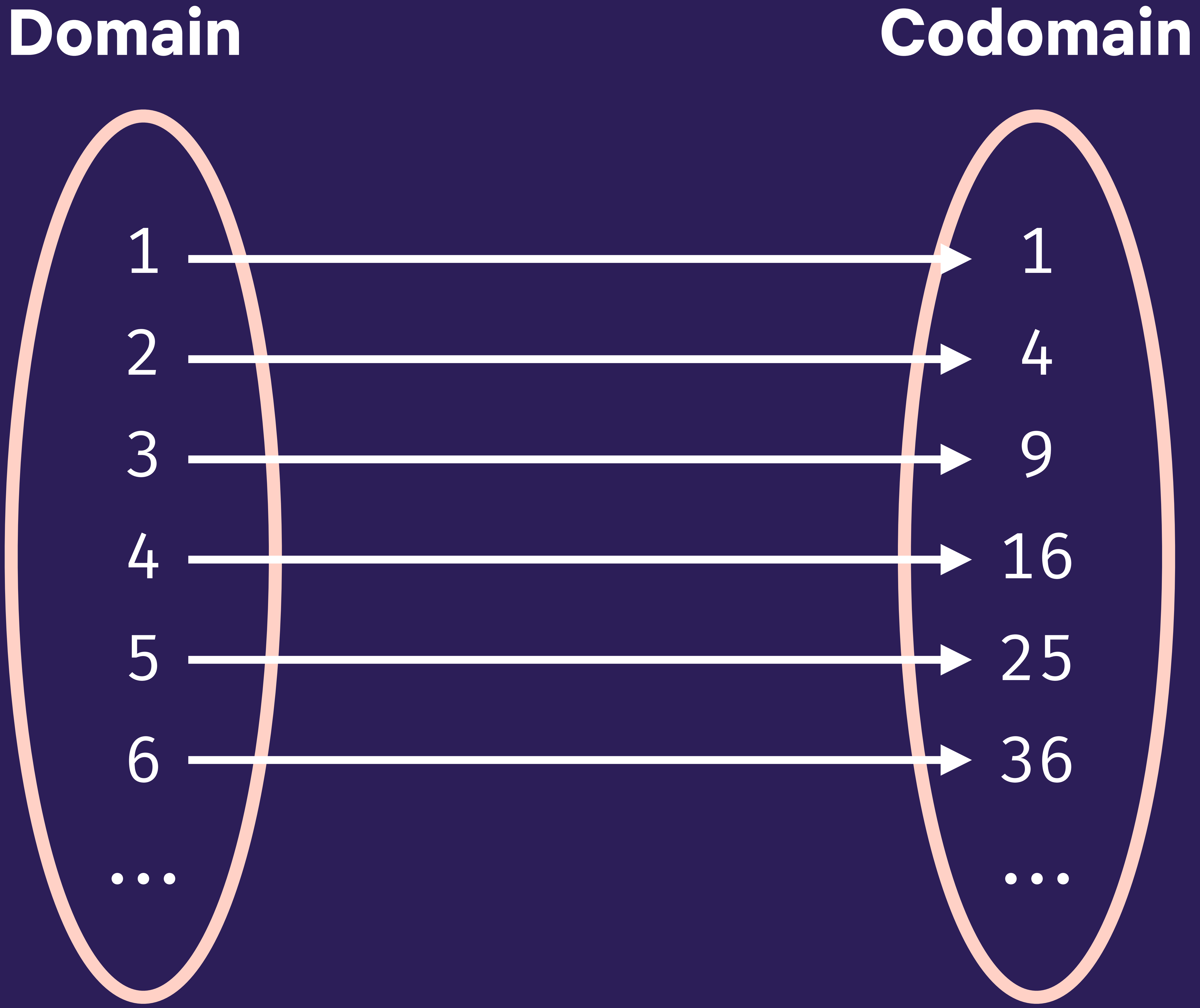
```
const MyStatelessComponent: React.SFC<MyProps> = () ⇒ 1;
const MyStatelessComponent: React.SFC<MyProps> = props => <div>...</div>;
```

Writing better functions



$$f(x) = x^2$$

$$f(x) = x^2$$



Total vs Partial functions

	Total	Partial
Impure	Impure & Total	Impure & Partial
Pure	Pure & Total	Pure & Partial

Partial

A partial function is a function that is not defined for all possible input values.

```
const half = x => x / 2;
const half = x => x / 2;
```

Possible Domains **Possible Codomains**

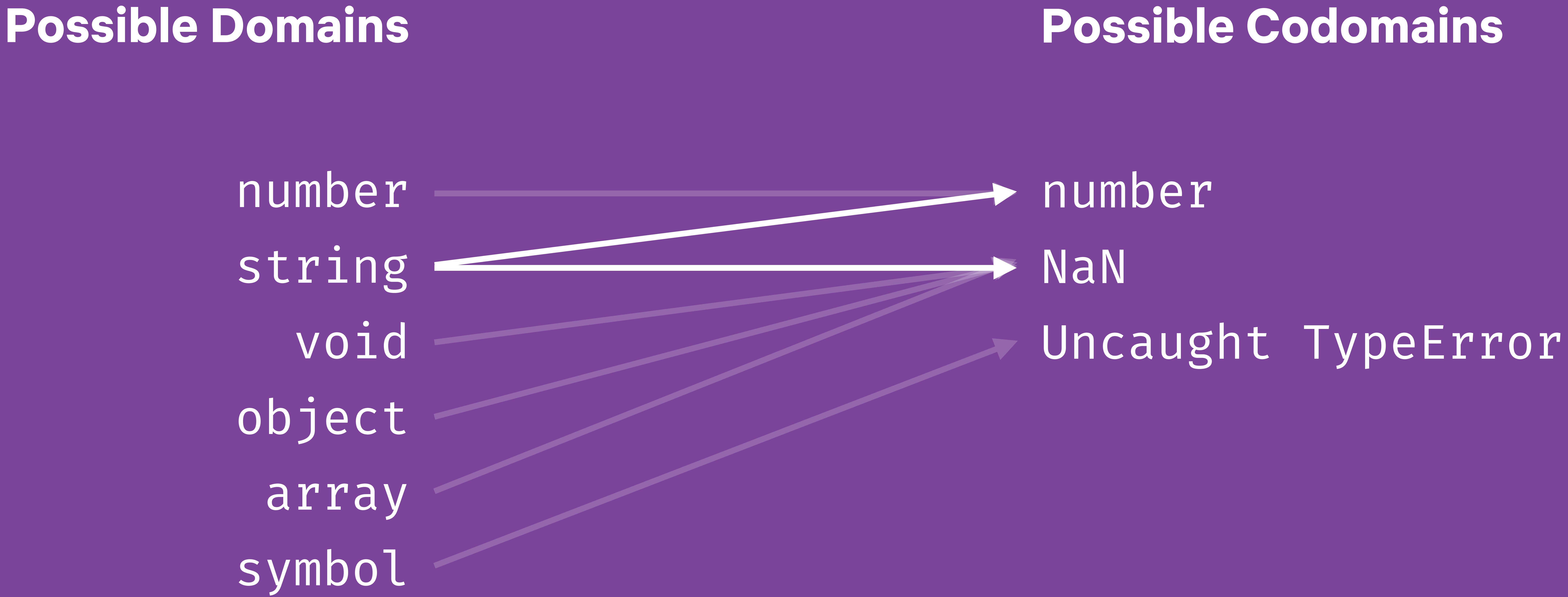
$$\begin{tikzcd}
\texttt{number} \arrow[r] & \texttt{number} \\
\texttt{string} \arrow[ur] \arrow[r] & \texttt{NaN} \\
\texttt{void} \arrow[ur] & \texttt{Uncaught TypeError} \\
\texttt{object} \arrow[uur] & \\
\texttt{array} \arrow[uuur] & \\
\texttt{symbol} \arrow[uuur] &
\end{tikzcd}$$

```js
const half = x => x / 2;
```

Possible Domains **Possible Codomains**

$$\begin{tikzcd}
\texttt{number} \arrow[r] & \texttt{number} \\
\texttt{string} \arrow[ur] \arrow[r] & \texttt{NaN} \\
\texttt{void} \arrow[ur] & \texttt{Uncaught TypeError} \\
\texttt{object} \arrow[uur] & \\
\texttt{array} \arrow[uuur] & \\
\texttt{symbol} \arrow[uuur] &
\end{tikzcd}$$

```
const half = x ⇒ x / 2;
```



```
const half = x ⇒ x / 2;
```

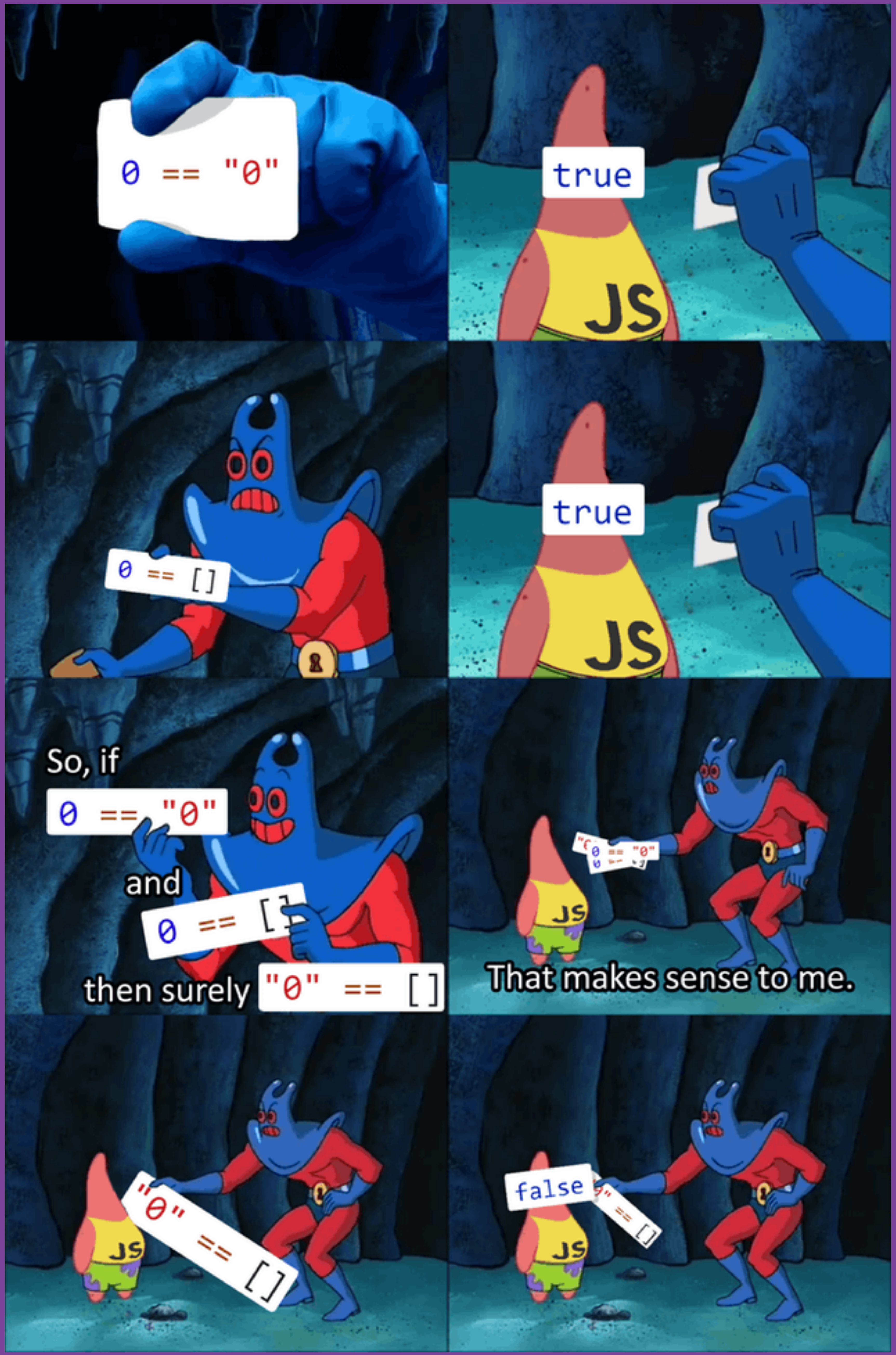
Possible Domains

Possible Codomains

```
half('10') // 5
half('hello world') // NaN
```

```
void
object
array
symbol
```

```
Uncaught TypeError
```


0 == "0"
true
JS
0 == []
true
JS
So, if
0 == "0"
and
0 == []
then surely "0" == []
That makes sense to me.
"0" == []
false
JS

```
const half = x => x / 2;
```

Possible Domains **Possible Codomains**

$$\begin{tikzcd}
\text{number} \arrow[r] & \text{number} \\
\text{string} \arrow[ur] \arrow[r] & \text{NaN} \\
\text{void} \arrow[ur] & \text{Uncaught TypeError} \\
\text{object} \arrow[uur] & \\
\text{array} \arrow[uuur] & \\
\text{symbol} \arrow[uuur] &
\end{tikzcd}$$

```
const half = x => x / 2;
```

Possible Domains **Possible Codomains**

$$\begin{tikzcd}
\texttt{number} \arrow[r] & \texttt{number} \\
\texttt{string} \arrow[ur] \arrow[r] & \texttt{NaN} \\
\texttt{void} \arrow[ur] & \texttt{Uncaught TypeError} \\
\texttt{object} \arrow[uur] & \\
\texttt{array} \arrow[uuur] & \\
\texttt{symbol} \arrow[uuur] &
\end{tikzcd}$$

```
const half = (x: number) ⇒ x / 2;
```

Possible Domains		Possible Codomains
number	$\longrightarrow$	number
string		NaN
void		Uncaught TypeError
object		
array		
symbol		

Total

A total function is a function that is defined for all possible values of its input. That is, it terminates and returns a value.

```
function fetchUser(username: string): Promise<User>
```

Possible Domains **Possible Codomains**

$$\begin{tikzcd}
\texttt{string} \arrow[r] \arrow[dr] & \texttt{Promise<User>} \\
\texttt{number} & \texttt{Uncaught Error} \\
\texttt{void} & \\
\texttt{object} & \\
\texttt{array} & \\
\texttt{symbol} &
\end{tikzcd}$$

```
function fetchUser(username: string): Promise<Either<FetchError, User>>
```

Possible Domains **Possible Codomains**

$$\begin{tikzcd}
\texttt{string} \arrow[r] & \texttt{Promise<Either<FetchError, User>>}
\end{tikzcd}$$

```
  string     Promise<Either<FetchError, User>>
  number     Uncaught Error
    void
  object
   array
  symbol
```

```
type Either<L, A> = Left<L, A> | Right<L, A>
```

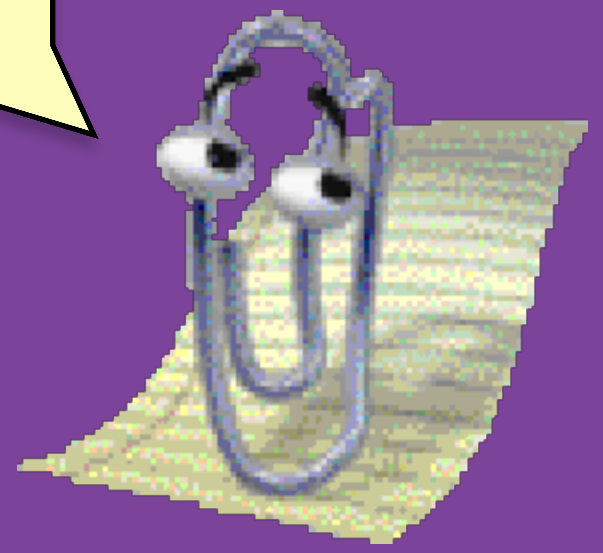
It looks like you're
trying to use a monad.

Would you like help?

```
type Either<L, A> = Left<L, A> | Right<L, A>
```

It looks like you're trying to use a monad.

Would you like help?



```
import { Either, left, right } from 'fp-ts/lib/Either';
import fetch from 'node-fetch';

async function fetchUser(username: string): Promise<Either<FetchError, User>> {
  const res = await fetch(`https://api.sugarpirate.com/users/${username}`);
  if (!res.ok) { return left(new FetchError(`[${res.status}] ${res.statusText}`)) }
  return right(await res.json());
}
```

https://github.com/gcanti/fp-ts

```typescript
import { Either, left, right } from 'fp-ts/lib/Either';
import fetch from 'node-fetch';

async function fetchUser(username: string): Promise<Either<FetchError, User>> {
  const res = await fetch(`https://api.sugarpirate.com/users/${username}`);
  if (!res.ok) { return left(new FetchError(`[${res.status}] ${res.statusText}`)) }
  return right(await res.json());
}
```

https://github.com/gcanti/fp-ts

```
import { Either, left, right } from 'fp-ts/lib/Either';
import fetch from 'node-fetch';

async function fetchUser(username: string): Promise<Either<FetchError, User>> {
  const res = await fetch(`https://api.sugarpirate.com/users/${username}`);
  if (!res.ok) { return left(new FetchError(`[${res.status}] ${res.statusText}`)) }
  return right(await res.json());
}
```

https://github.com/gcanti/fp-ts

```
import { Either, left, right } from 'fp-ts/lib/Either';
import fetch from 'node-fetch';

async function fetchUser(username: string): Promise<Either<FetchError, User>> {
  const res = await fetch(`https://api.sugarpirate.com/users/${username}`);
  if (!res.ok) { return left(new FetchError(`[${res.status}] ${res.statusText}`)) }
  return right(await res.json());
}
```

https://github.com/gcanti/fp-ts

```
async function doIt() {
  const maybeLauren = await fetchUser('lauren');
  const maybeNoOne = await fetchUser('asdjasjdashjdkahjksd');
  maybeLauren
    .map(lauren ⇒ lauren.projects)
    .map(projects ⇒ console.log(projects.map(p ⇒ p.name)));
  maybeNoOne
    .map(noOne ⇒ noOne.projects)
    .map(projects ⇒ console.log(projects.map(p ⇒ p.name)));
}
```

```
async function doIt() {
  const maybeNoOne = await fetchUser('asdjasjdashjdkahjksd');
  maybeNoOne
    .mapLeft(e ⇒ console.log(e.message)); // e: FetchError
}
```

```
export function firstVisibleElement(
  selector: string,
  scrollableAreaSelector: string
): Element | undefined
```

Possible Domains

Possible Codomains

$$\begin{tikzcd} \texttt{[string, string]} \arrow[r] \arrow[dr] & \texttt{Element} \\ \texttt{number} & \texttt{undefined} \\ \texttt{void} & \\ \texttt{object} & \\ \texttt{array} & \\ \texttt{symbol} & \end{tikzcd}$$

🙅‍♀️

```
export function firstVisibleElement(
  selector: string,
  scrollableAreaSelector: string
): Option<Element>
```

Possible Domains		Possible Codomains
[string, string]	→	Option<Element>
number		undefined
void		
object		
array		
symbol		

```
type Option<A> = None<A> | Some<A>
```

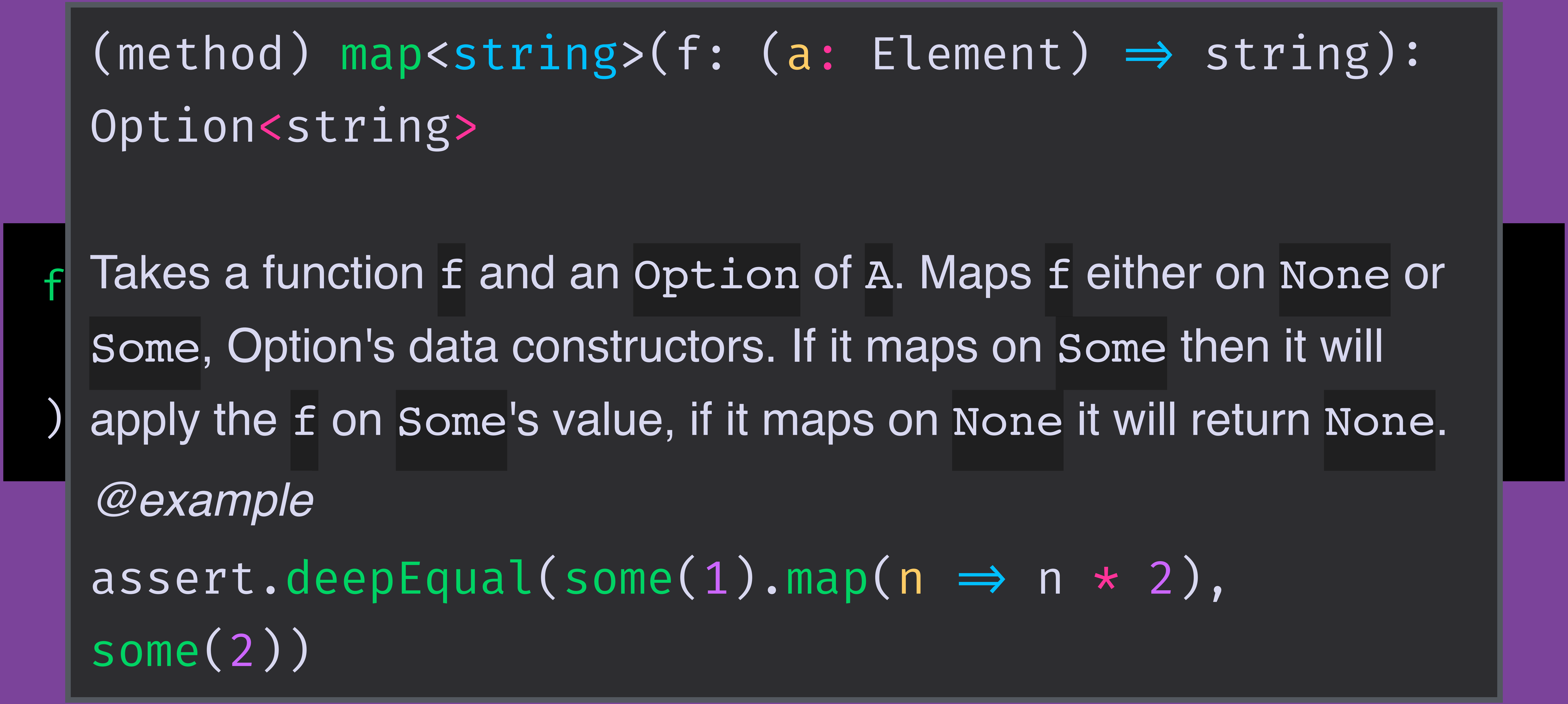
https://github.com/gcanti/fp-ts

```typescript
export function firstVisibleElement(
  selector: string,
  scrollableAreaSelector: string
): Option<Element> {
  const scrollableElement = document.querySelector(scrollableAreaSelector);
  if (!scrollableElement) return none;
  const scrollableBounds = scrollableElement.getBoundingClientRect();
  const firstVisibleItem = Array.from(document.querySelectorAll(selector)).find(
    el => {
      const elBounds = el.getBoundingClientRect();
      const isInViewport = detectInViewport(elBounds, scrollableBounds);
      return isInViewport && elBounds.top - scrollableBounds.top <= 0;
    }
  );
  return firstVisibleItem ? some<Element>(firstVisibleItem) : none;
}
```

```
export function firstVisibleElement(
  selector: string,
  scrollableAreaSelector: string
): Option<Element> {
  const scrollableElement = document.querySelector(scrollableAreaSelector);
  if (!scrollableElement) return none;
  const scrollableBounds = scrollableElement.getBoundingClientRect();
  const firstVisibleItem = Array.from(document.querySelectorAll(selector)).find(
    el => {
      const elBounds = el.getBoundingClientRect();
      const isInViewport = detectInViewport(elBounds, scrollableBounds);
      return isInViewport && elBounds.top - scrollableBounds.top <= 0;
    }
  );
  return firstVisibleItem ? some<Element>(firstVisibleItem) : none;
}
```

```
export function firstVisibleElement(
  selector: string,
  scrollableAreaSelector: string
): Option<Element> {
  const scrollableElement = document.querySelector(scrollableAreaSelector);
  if (!scrollableElement) return none;
  const scrollableBounds = scrollableElement.getBoundingClientRect();
  const firstVisibleItem = Array.from(document.querySelectorAll(selector)).find(
    el => {
      const elBounds = el.getBoundingClientRect();
      const isInViewport = detectInViewport(elBounds, scrollableBounds);
      return isInViewport && elBounds.top - scrollableBounds.top <= 0;
    }
  );
  return firstVisibleItem ? some<Element>(firstVisibleItem) : none;
}
```

```
firstVisibleElement('.item', '.item-container').map(el ⇒
  el.getAttribute('data-whatever') // string
);
```



```
(method) map<string>(f: (a: Element) ⇒ string):
Option<string>
```

Takes a function `f` and an `Option` of `A`. Maps `f` either on `None` or `Some`, Option's data constructors. If it maps on `Some` then it will apply the `f` on `Some`'s value, if it maps on `None` it will return `None`.

@example

```
assert.deepEqual(some(1).map(n ⇒ n * 2),
some(2))
```

```tsx
import { NoData, Pending, Failure } from './MyPlaceholders';
import { TCustomer } from './MyModel';

type TCustomersList = { entities: RemoteData<TCustomer[]>; };

const CustomersList: React.SFC<TCustomersList> = ({ entities }) => entities.foldL(
  () => <NoData />,
  () => <Pending />,
  err => <Failure error={err} />,
  data => <ul>{data.map(item => <li>{item.name}</li>)}</ul>
);
```

https://github.com/devex-web-frontend/remote-data-ts

Cardinality

cardinality · number of elements of the set

Lower cardinality = Less bugs*

Pragmatic Set Theory

set · collection of objects

```
type Conferences = 'QConSF' | 'dotJS' | 'React Rally';
```

(not real syntax)

```
|Conferences| = 3
```

```
type Conferences = string;
```

(not real syntax)

```
|Conferences| = Infinity
```

Primitive types are not precise

(not real syntax)

```
|string|    = Infinity
|number|    = Infinity
|symbol|    = Infinity
|boolean|   = 2
|null|      = 1
|undefined| = 1
```

(not real syntax)

```
|object| = Infinity
```

Be precise

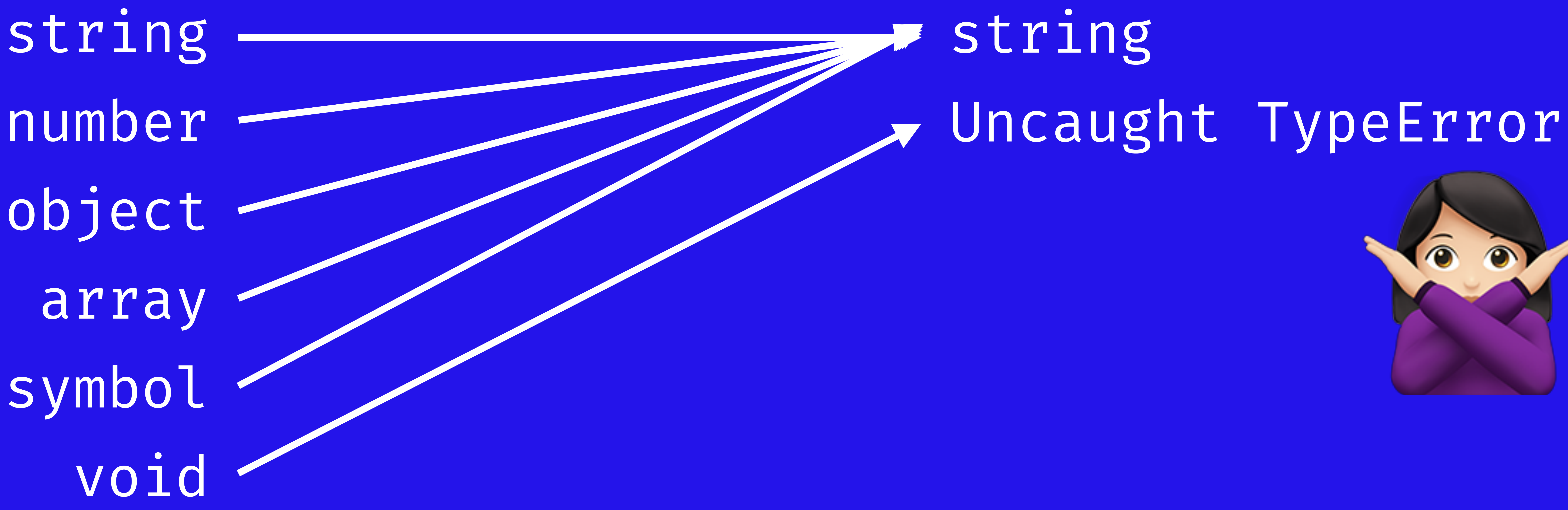
```
function toString<T>(x: T): string { return x.toString(); }
toString(undefined);
toString(null);
```

```
function toString<undefined>(x: undefined): string
```

```
function toString<T>(x: T): string { return x.toString(); }
toString(undefined);
toString(null);
```

```
function toString<null>(x: null): string
function toString<T>(x: T): string
```

$$\begin{tikzcd}[row sep=small]
\text{string} \arrow[r] & \text{string} \\
\text{number} \arrow[ru] & \text{Uncaught TypeError} \\
\text{object} \arrow[ruu] & \\
\text{array} \arrow[ruuu] & \\
\text{symbol} \arrow[ruuuu] & \\
\text{void} \arrow[ruuuu] &
\end{tikzcd}$$



```
function toString<T>(x: NonNullable<T>): string { return x.toString(); }
toString(undefined);
toString(null);
```

```
function toString<T>(x: NonNullable<T>): string { return x.toString(); }
toString(undefined);
toString(null);
```

```
[ts] Argument of type 'undefined' is not
assignable to parameter of type '{}'.
```

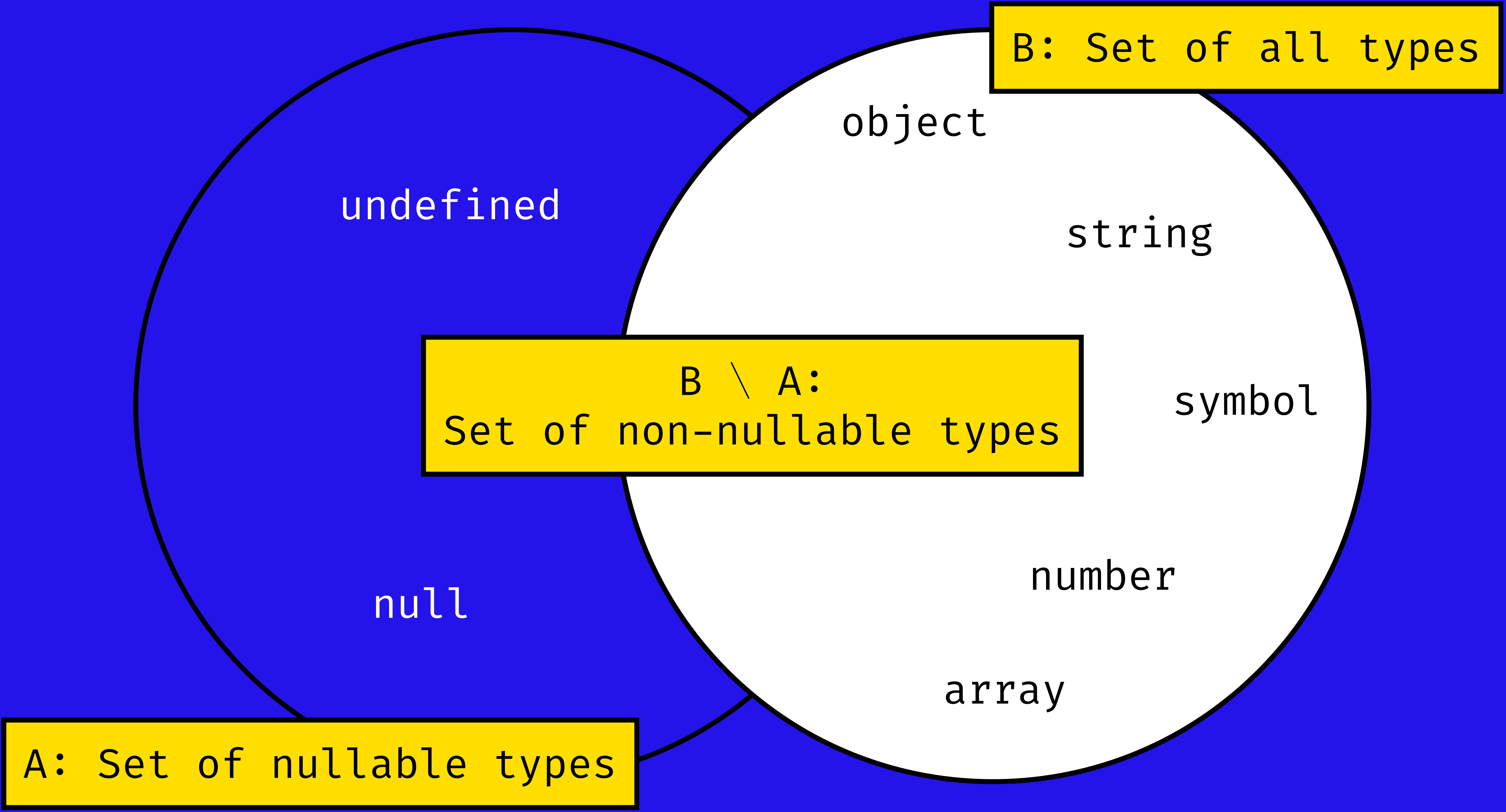
```
function toString<T>(x: NonNullable<T>): string { return x.toString(); }
toString(undefined);
toString(null);
```

```
[ts] Argument of type 'null' is not
assignable to parameter of type '{}'.
```

```
function toString<T>(x: NonNullable<T>): string
```

Possible Domains **Possible Codomains**

$$\begin{tikzcd}
\texttt{string} \arrow[r] & \texttt{string} \\
\texttt{number} \arrow[ru] & \texttt{Uncaught TypeError} \\
\texttt{object} \arrow[ruu] & \\
\texttt{array} \arrow[ruuu] & \\
\texttt{symbol} \arrow[ruuuu] & \\
\texttt{void} &
\end{tikzcd}$$


B: Set of all types
object
undefined
string
B \ A:
Set of non-nullable types
symbol
number
null
array
A: Set of nullable types

```
type NonNullable<T> = T extends null | undefined ? never : T
type T34 = NonNullable<string | number | undefined>;  // string | number
type T35 = NonNullable<string | string[] | null | undefined>;  // string | string[]
```

```typescript
type Partial<T> = { [P in keyof T]?: T[P]; };
type Required<T> = { [P in keyof T]-?: T[P]; };
type Readonly<T> = { readonly [P in keyof T]: T[P]; };
type Pick<T, K extends keyof T> = { [P in K]: T[P]; };
type Record<K extends keyof any, T> = { [P in K]: T; };
type Exclude<T, U> = T extends U ? never : T;
type Extract<T, U> = T extends U ? T : never;
type NonNullable<T> = T extends null | undefined ? never : T;
type ReturnType<T extends (...args: any[]) => any> = T extends (...args: any[]) => infer R ? R : any;
```

https://github.com/Microsoft/TypeScript/blob/v3.0.1/src/lib/es5.d.ts

Be pragmatic

"No matter what language you work in, programming in a functional style provides benefits. You should do it whenever it is convenient, and you should think hard about the decision when it isn't convenient."

John Carmack

Types Over the Network

GraphQL, Your New BFF

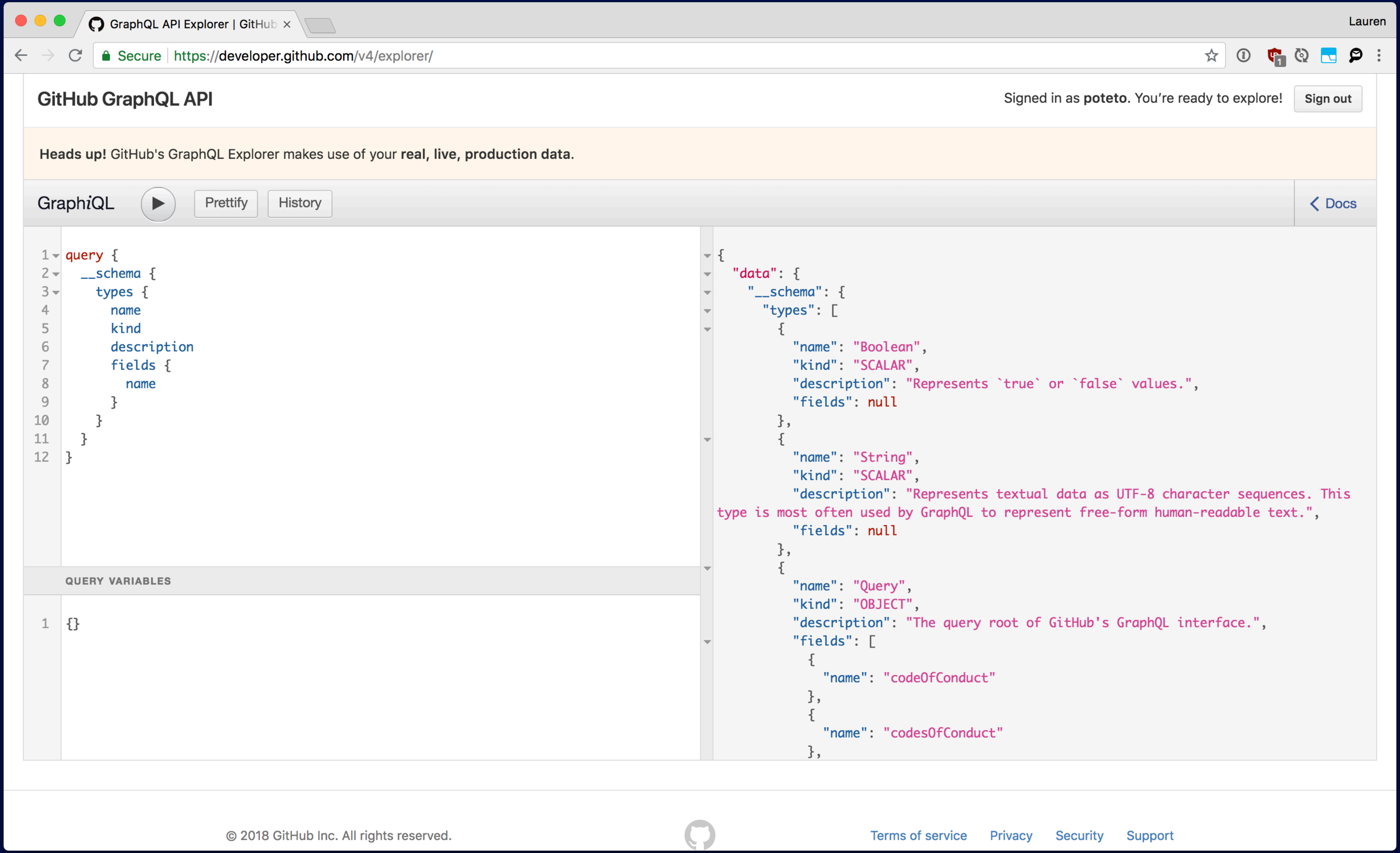
Types Over the Network

GraphQL, Your New BFF

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```



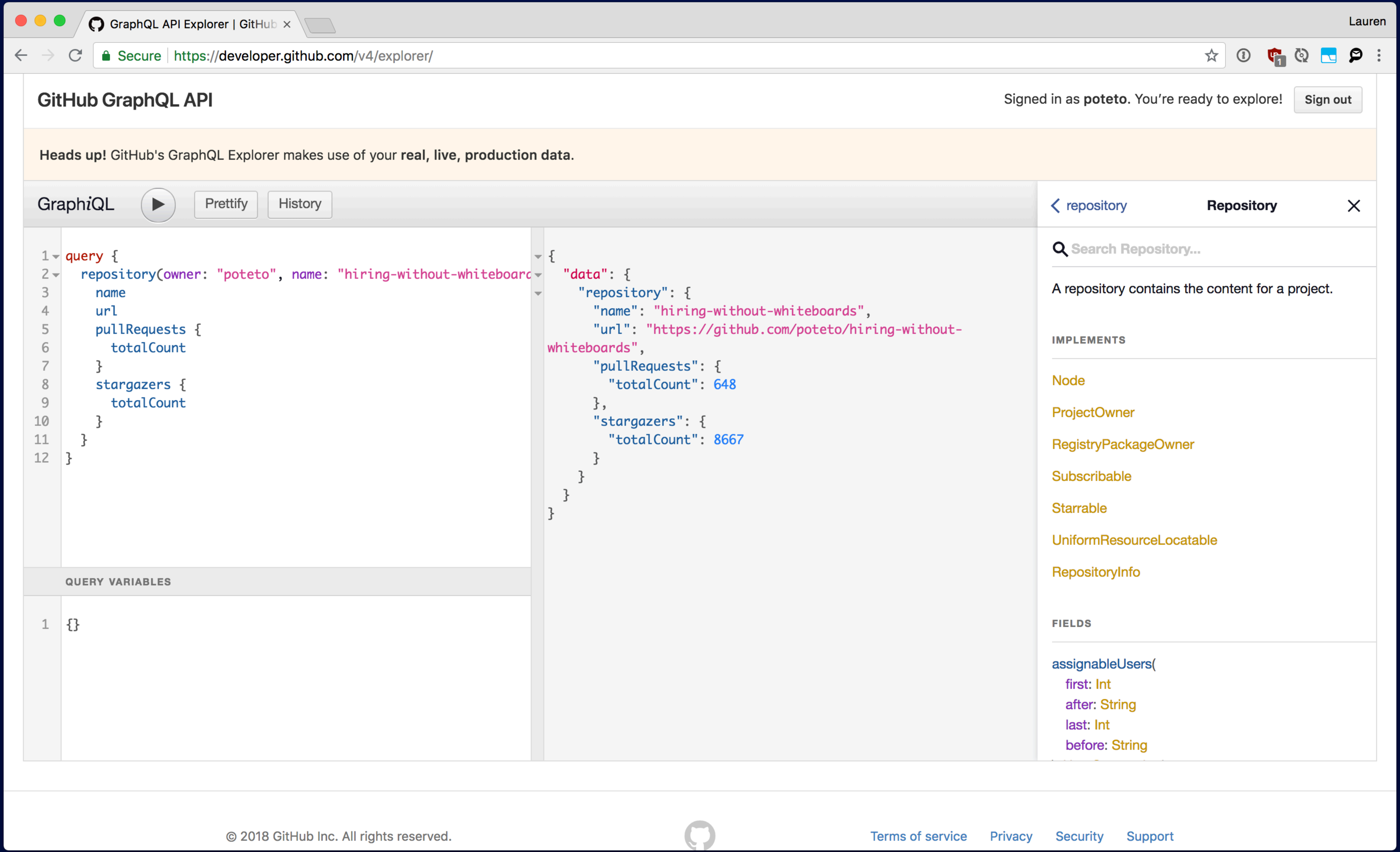
Heads up! GitHub's GraphQL Explorer makes use of your **real, live, production data**.

```
query {
  __schema {
    types {
      name
      kind
      description
      fields {
        name
      }
    }
  }
}
```

```
{}
```

```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "Boolean",
          "kind": "SCALAR",
          "description": "Represents `true` or `false` values.",
          "fields": null
        },
        {
          "name": "String",
          "kind": "SCALAR",
          "description": "Represents textual data as UTF-8 character sequences. This type is most often used by GraphQL to represent free-form human-readable text.",
          "fields": null
        },
        {
          "name": "Query",
          "kind": "OBJECT",
          "description": "The query root of GitHub's GraphQL interface.",
          "fields": [
            {
              "name": "codeOfConduct"
            },
            {
              "name": "codesOfConduct"
            },
```

© 2018 GitHub Inc. All rights reserved.



GitHub GraphQL API

Signed in as **poteto**. You're ready to explore!

Heads up! GitHub's GraphQL Explorer makes use of your **real, live, production data**.

```
query {
  repository(owner: "poteto", name: "hiring-without-whiteboarc
    name
    url
    pullRequests {
      totalCount
    }
    stargazers {
      totalCount
    }
  }
}
```

QUERY VARIABLES

```
{}
```

```
{
  "data": {
    "repository": {
      "name": "hiring-without-whiteboards",
      "url": "https://github.com/poteto/hiring-without-
whiteboards",
      "pullRequests": {
        "totalCount": 648
      },
      "stargazers": {
        "totalCount": 8667
      }
    }
  }
}
```

Repository

A repository contains the content for a project.

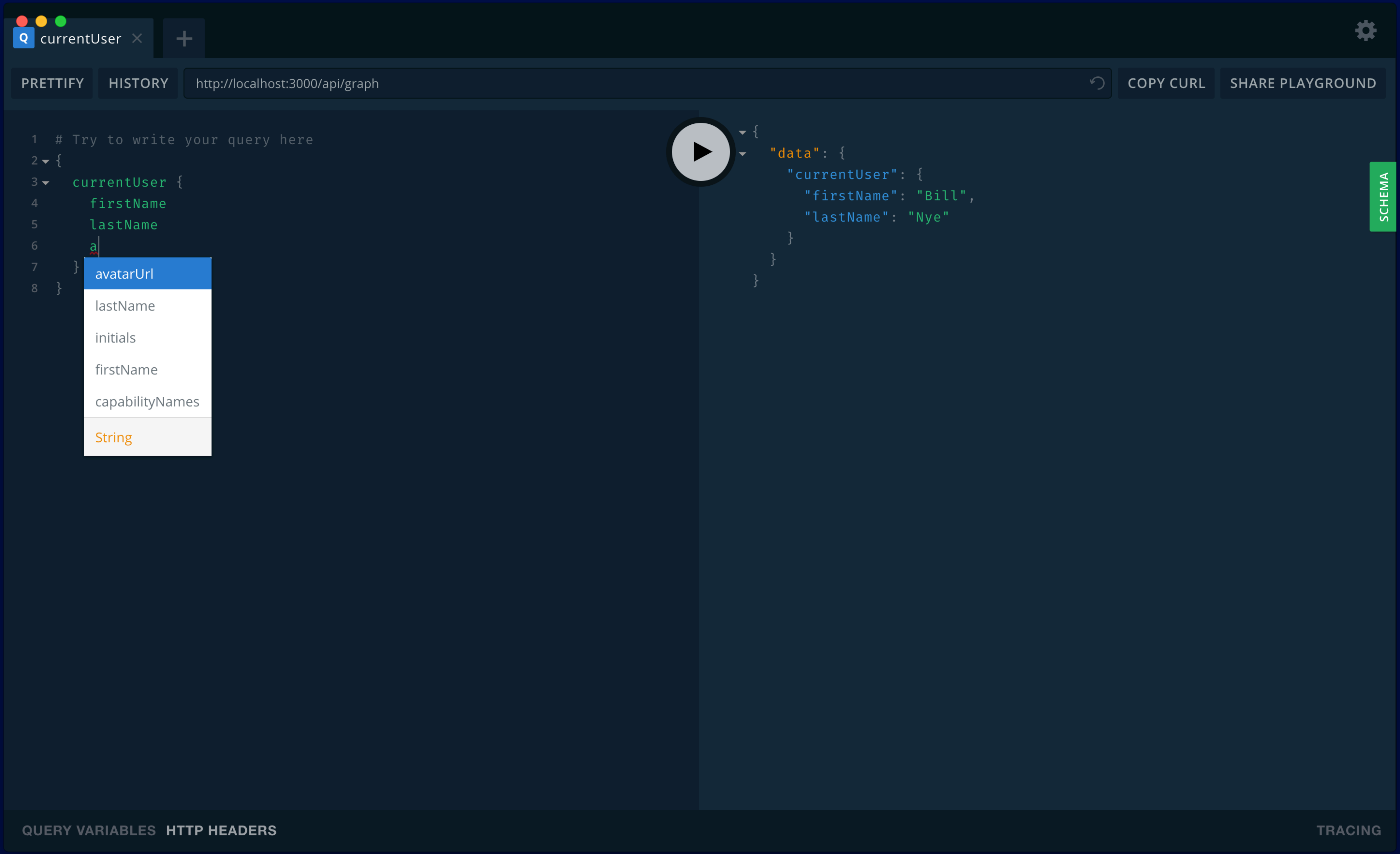
IMPLEMENTS

Node
ProjectOwner
RegistryPackageOwner
Subscribable
Starrable
UniformResourceLocatable
RepositoryInfo

FIELDS

```
assignableUsers(
  first: Int
  after: String
  last: Int
  before: String
```

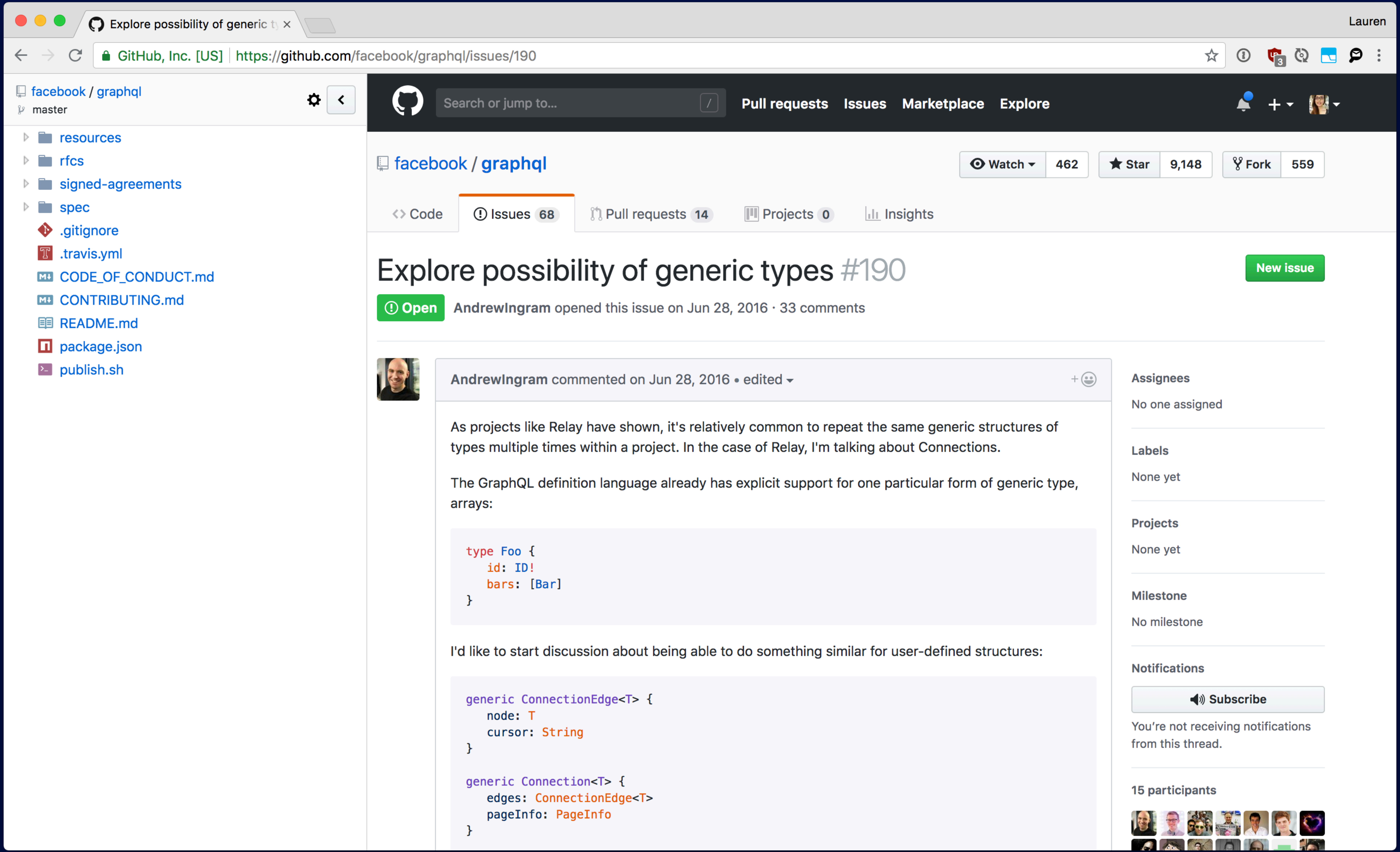
© 2018 GitHub Inc. All rights reserved.



```
# Try to write your query here
{
  currentUser {
    firstName
    lastName
    a
  }
}
```

avatarUrl
lastName
initials
firstName
capabilityNames
String

```
{
  "data": {
    "currentUser": {
      "firstName": "Bill",
      "lastName": "Nye"
    }
  }
}
```



Explore possibility of generic t ×

Lauren

GitHub, Inc. [US] https://github.com/facebook/graphql/issues/190

facebook / graphql
master

- resources
- rfcs
- signed-agreements
- spec
- .gitignore
- .travis.yml
- CODE_OF_CONDUCT.md
- CONTRIBUTING.md
- README.md
- package.json
- publish.sh

Search or jump to... Pull requests Issues Marketplace Explore

facebook / graphql

Watch 462 Star 9,148 Fork 559

Code Issues 68 Pull requests 14 Projects 0 Insights

Explore possibility of generic types #190

New issue

Open AndrewIngram opened this issue on Jun 28, 2016 · 33 comments

AndrewIngram commented on Jun 28, 2016 • edited

As projects like Relay have shown, it's relatively common to repeat the same generic structures of types multiple times within a project. In the case of Relay, I'm talking about Connections.

The GraphQL definition language already has explicit support for one particular form of generic type, arrays:

```
type Foo {
   id: ID!
   bars: [Bar]
}
```

I'd like to start discussion about being able to do something similar for user-defined structures:

```
generic ConnectionEdge<T> {
   node: T
   cursor: String
}

generic Connection<T> {
   edges: ConnectionEdge<T>
   pageInfo: PageInfo
}
```

Assignees
No one assigned

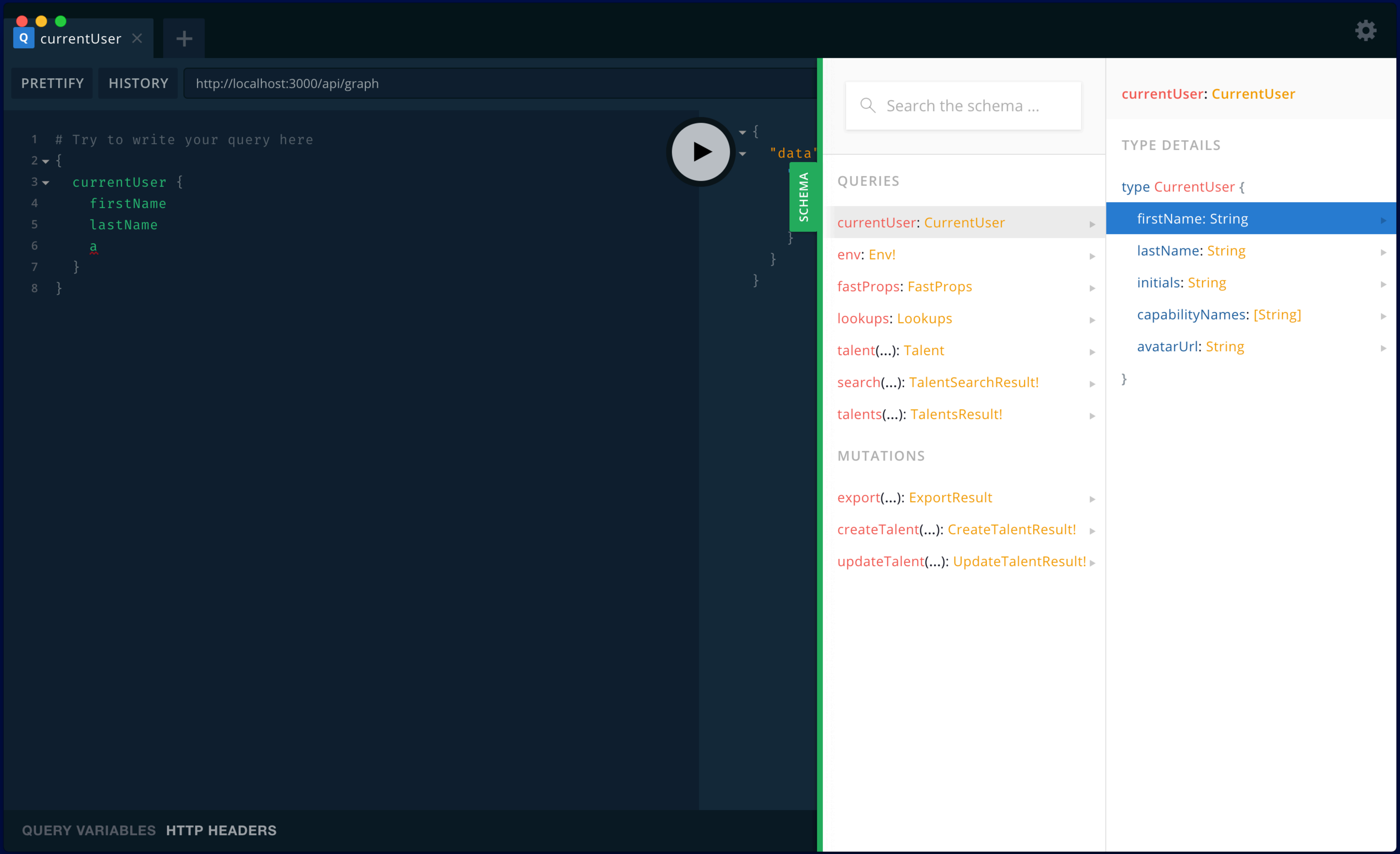
Labels
None yet

Projects
None yet

Milestone
No milestone

Notifications
Subscribe
You're not receiving notifications from this thread.

15 participants



http://localhost:3000/api/graph

```
# Try to write your query here
{
  currentUser {
    firstName
    lastName
    a
  }
}
```

```
{
  "data"
```

QUERIES

currentUser: CurrentUser
env: Env!
fastProps: FastProps
lookups: Lookups
talent(...): Talent
search(...): TalentSearchResult!
talents(...): TalentsResult!

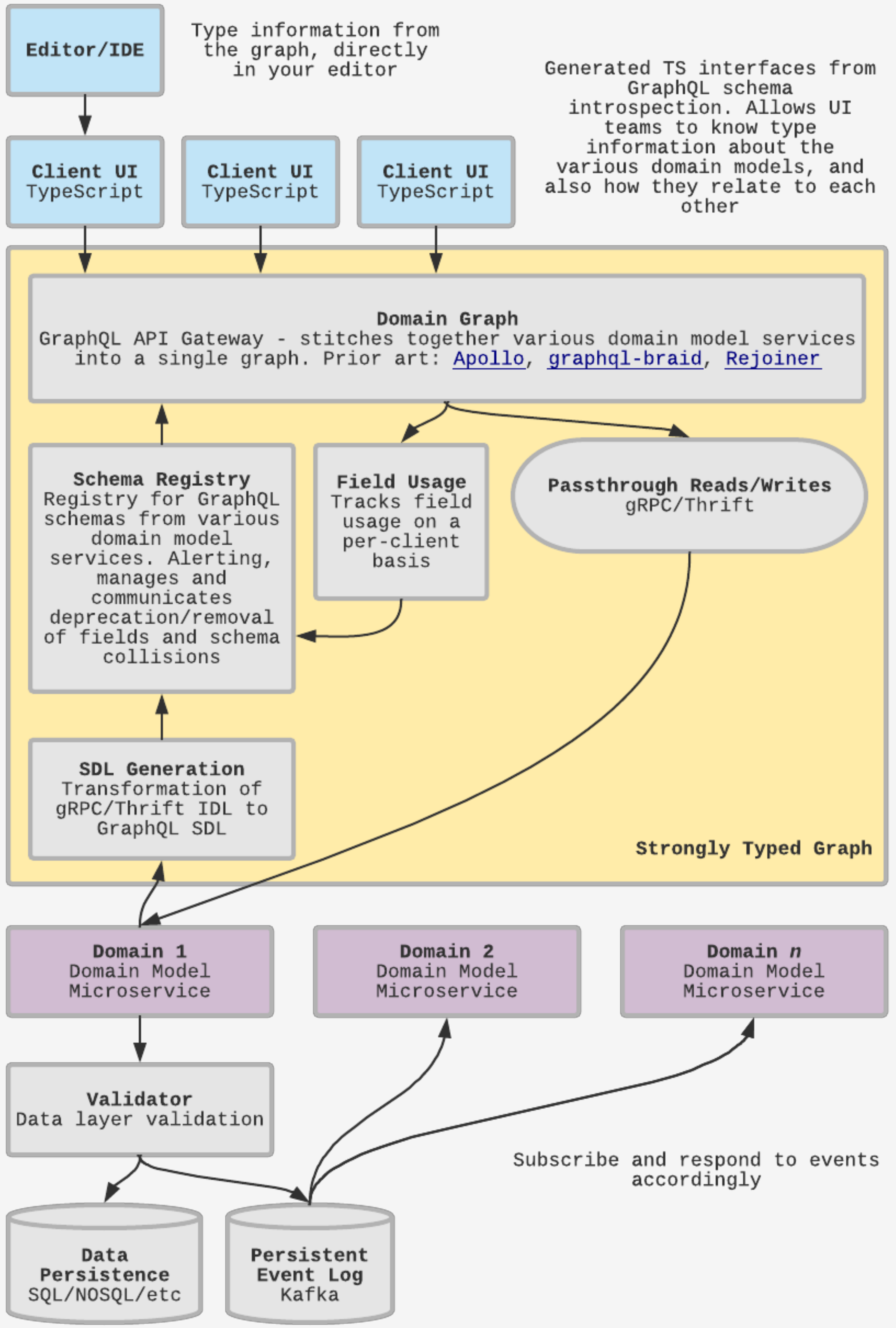
MUTATIONS

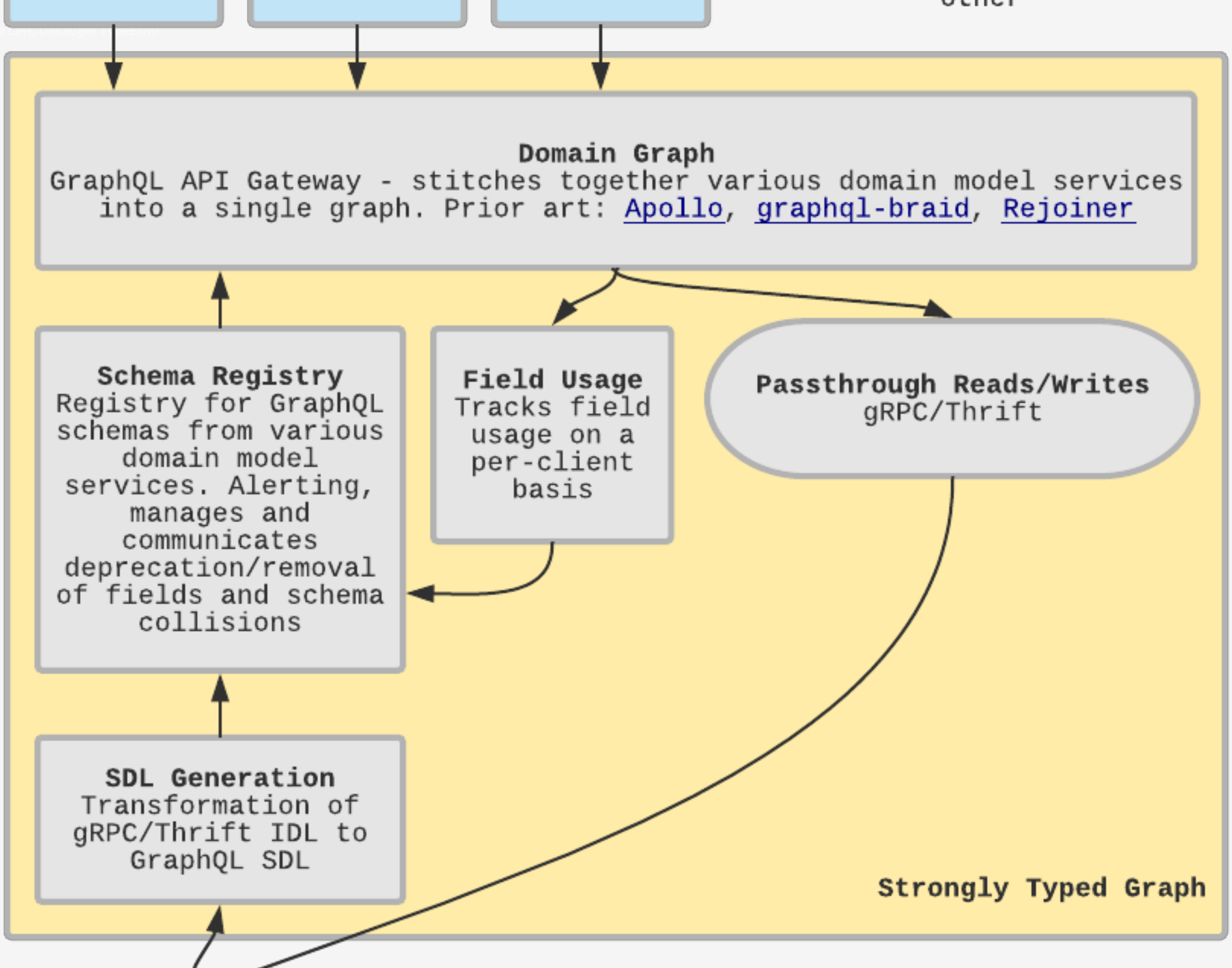
export(...): ExportResult
createTalent(...): CreateTalentResult!
updateTalent(...): UpdateTalentResult!

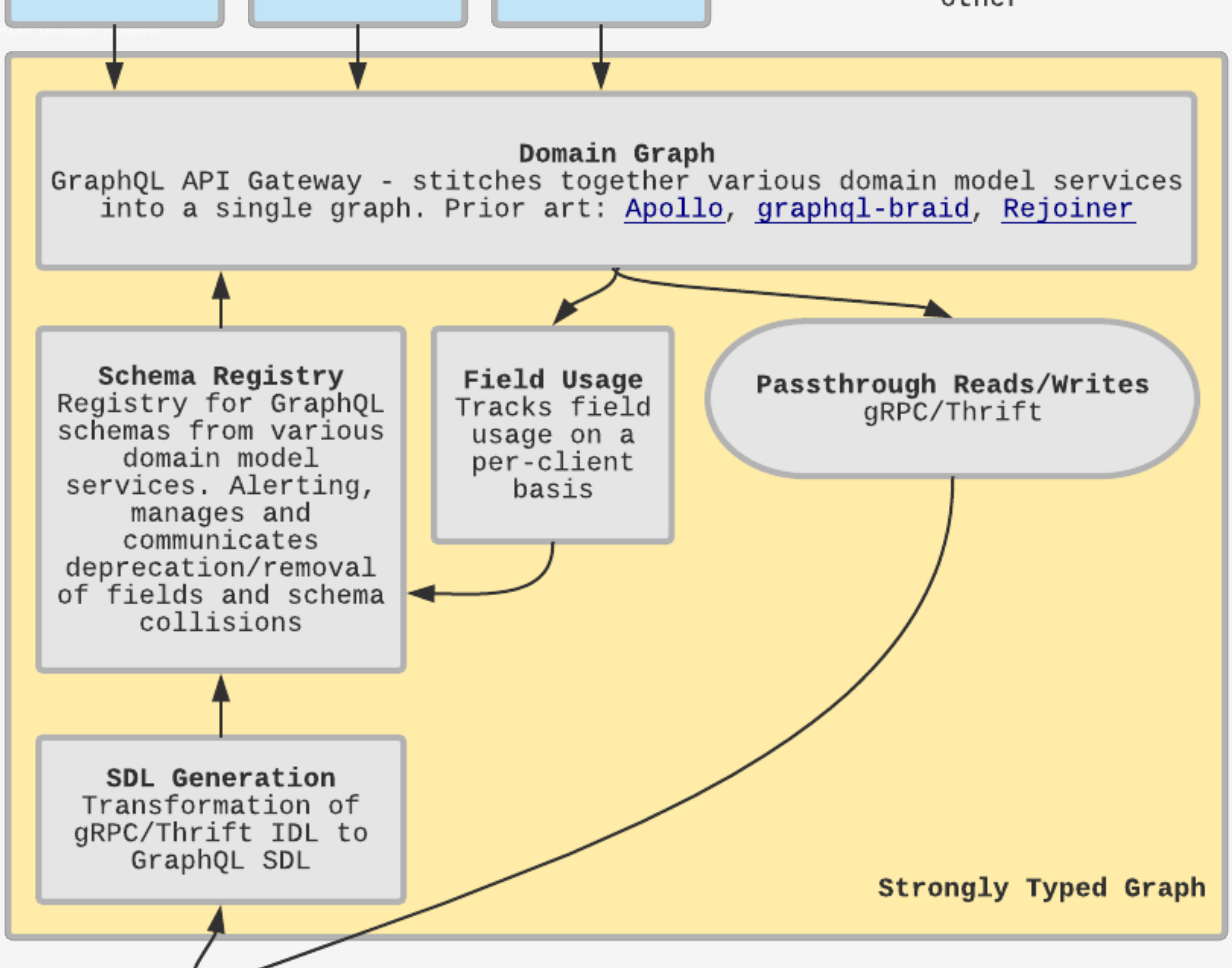
currentUser: CurrentUser

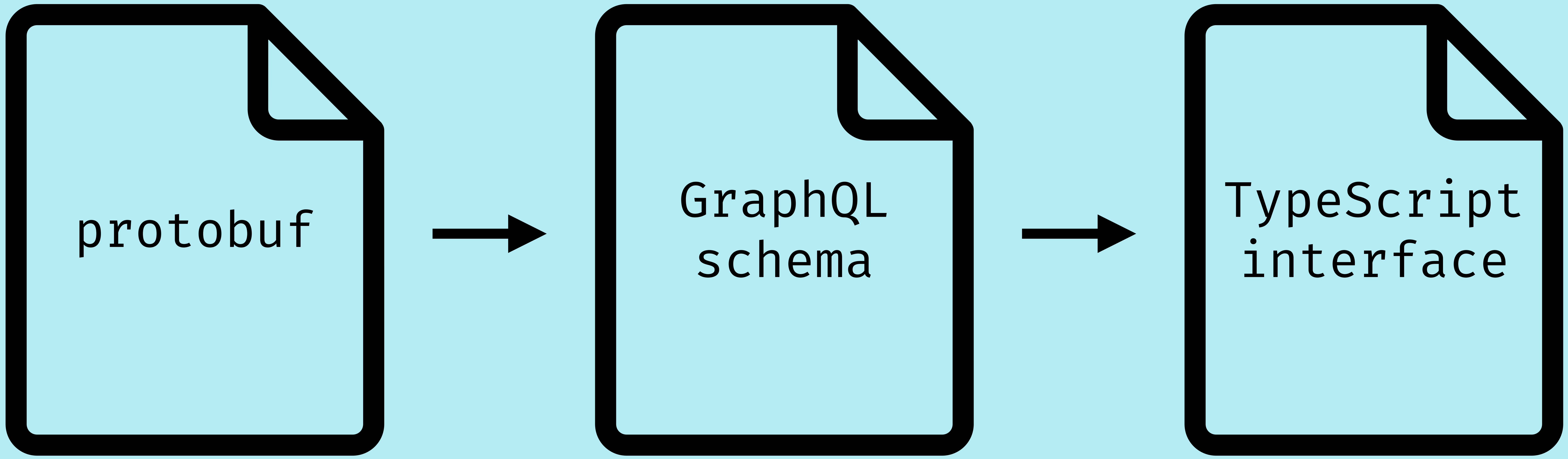
TYPE DETAILS

```
type CurrentUser {
  firstName: String
  lastName: String
  initials: String
  capabilityNames: [String]
  avatarUrl: String
}
```

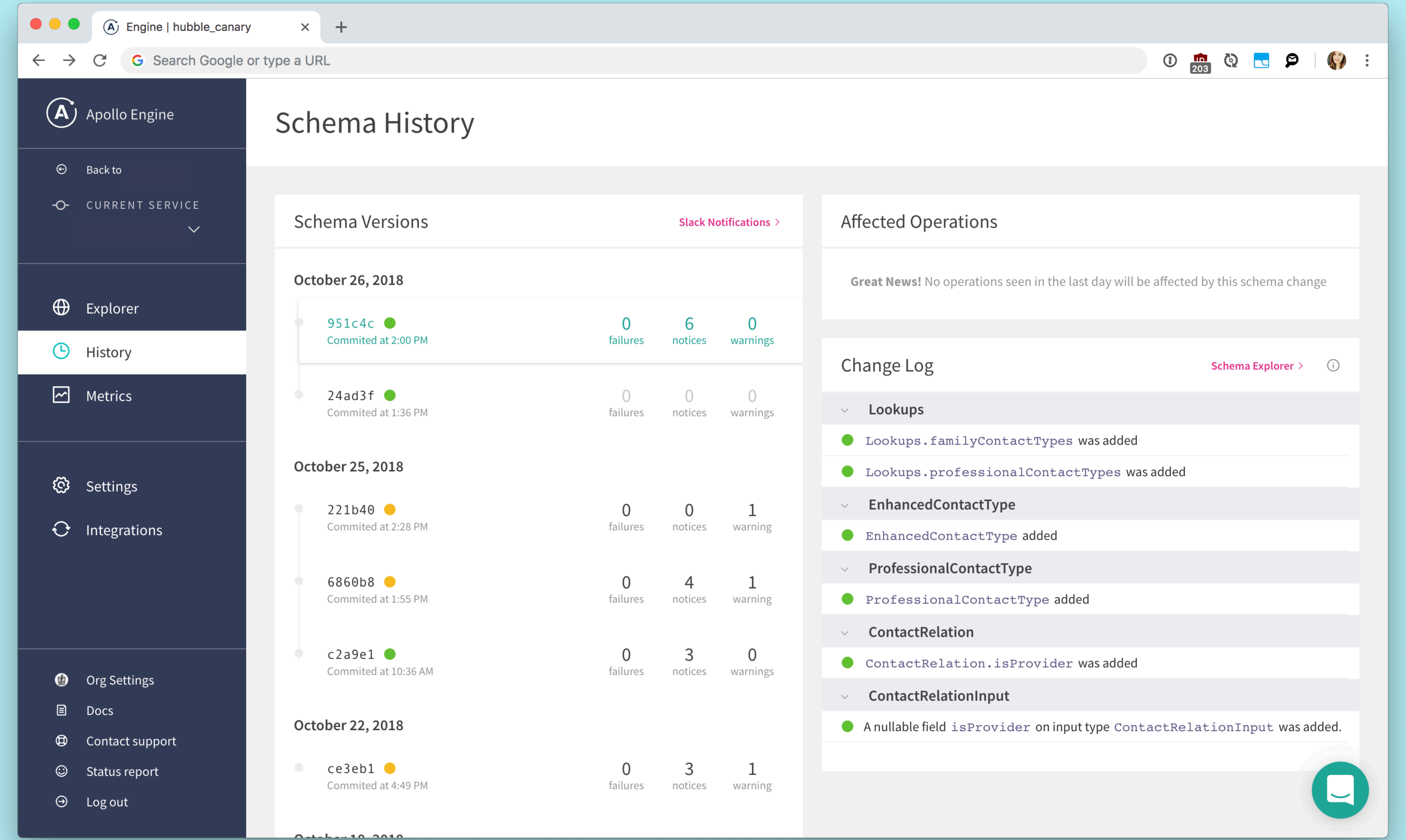

Editor/IDE
Type information from the graph, directly in your editor
Client UI TypeScript
Client UI TypeScript
Client UI TypeScript
Generated TS interfaces from GraphQL schema introspection. Allows UI teams to know type information about the various domain models, and also how they relate to each other
Domain Graph
GraphQL API Gateway - stitches together various domain model services into a single graph. Prior art: Apollo, graphql-braid, Rejoiner
Schema Registry
Registry for GraphQL schemas from various domain model services. Alerting, manages and communicates deprecation/removal of fields and schema collisions
Field Usage
Tracks field usage on a per-client basis
Passthrough Reads/Writes
gRPC/Thrift
SDL Generation
Transformation of gRPC/Thrift IDL to GraphQL SDL
Strongly Typed Graph
Domain 1 Domain Model Microservice
Domain 2 Domain Model Microservice
Domain n Domain Model Microservice
Validator
Data layer validation
Subscribe and respond to events accordingly
Data Persistence SQL/NOSQL/etc
Persistent Event Log Kafka


Other
Domain Graph
GraphQL API Gateway - stitches together various domain model services into a single graph. Prior art: Apollo, graphql-braid, Rejoiner
Schema Registry
Registry for GraphQL schemas from various domain model services. Alerting, manages and communicates deprecation/removal of fields and schema collisions
Field Usage
Tracks field usage on a per-client basis
Passthrough Reads/Writes
gRPC/Thrift
SDL Generation
Transformation of gRPC/Thrift IDL to GraphQL SDL
Strongly Typed Graph


Other
Domain Graph
GraphQL API Gateway - stitches together various domain model services into a single graph. Prior art: Apollo, graphql-braid, Rejoiner
Schema Registry
Registry for GraphQL schemas from various domain model services. Alerting, manages and communicates deprecation/removal of fields and schema collisions
Field Usage
Tracks field usage on a per-client basis
Passthrough Reads/Writes
gRPC/Thrift
SDL Generation
Transformation of gRPC/Thrift IDL to GraphQL SDL
Strongly Typed Graph


protobuf
GraphQL schema
TypeScript interface

$$\begin{tikzcd}
\text{gRPC Microservice} \arrow[r] & \text{GraphQL Gateway} \arrow[r] & \text{Strongly Typed UI}
\end{tikzcd}$$



Schema History

Schema Versions

Slack Notifications >

October 26, 2018

951c4c ● Commited at 2:00 PM	0 failures	6 notices	0 warnings
24ad3f ● Commited at 1:36 PM	0 failures	0 notices	0 warnings

October 25, 2018

221b40 ● Commited at 2:28 PM	0 failures	0 notices	1 warning
6860b8 ● Commited at 1:55 PM	0 failures	4 notices	1 warning
c2a9e1 ● Commited at 10:36 AM	0 failures	3 notices	0 warnings

October 22, 2018

ce3eb1 ● Commited at 4:49 PM	0 failures	3 notices	1 warning

Affected Operations

Great News! No operations seen in the last day will be affected by this schema change

Change Log

Schema Explorer >

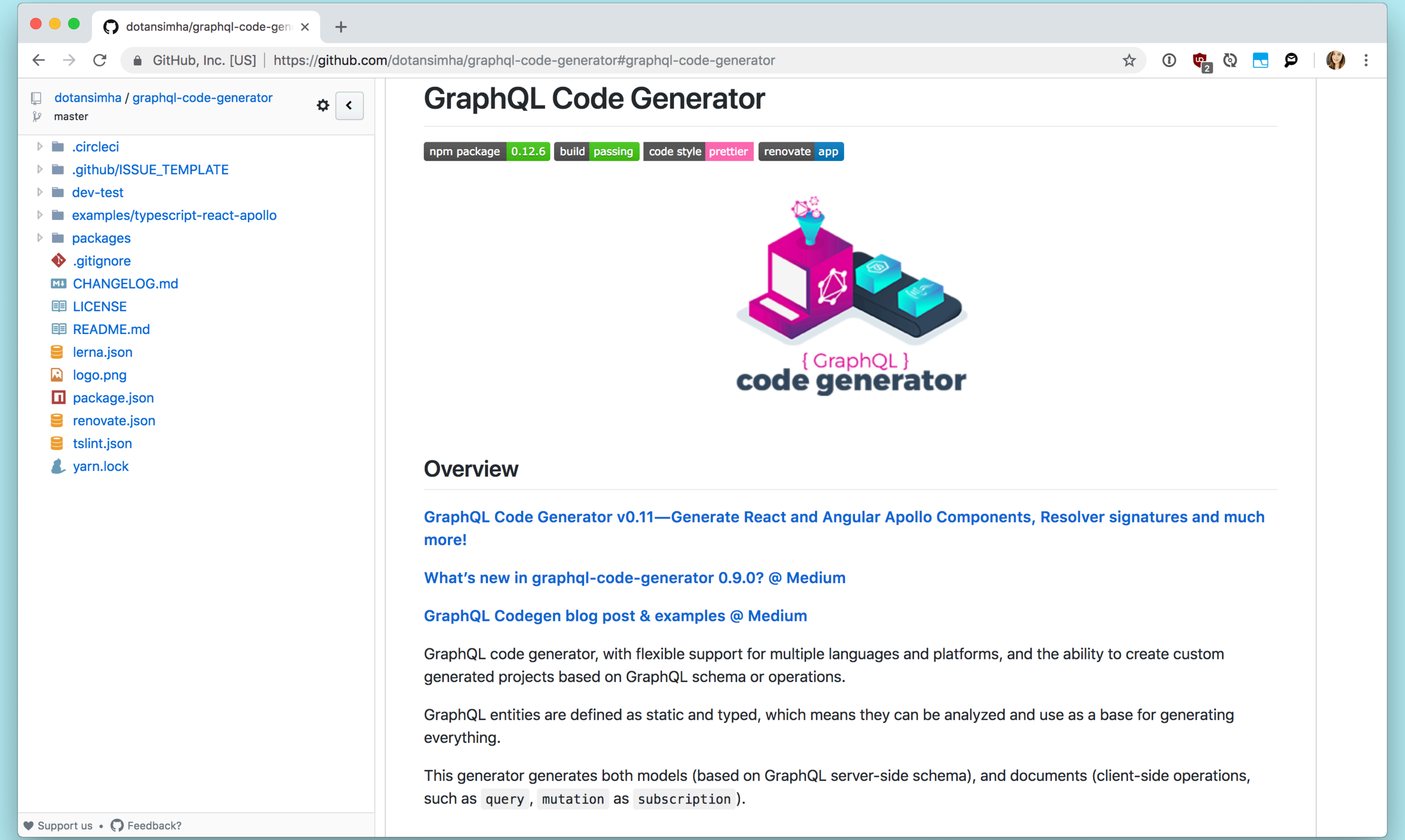
Lookups
- ● `Lookups.familyContactTypes` was added
- ● `Lookups.professionalContactTypes` was added

EnhancedContactType
- ● `EnhancedContactType` added

ProfessionalContactType
- ● `ProfessionalContactType` added

ContactRelation
- ● `ContactRelation.isProvider` was added

ContactRelationInput
- ● A nullable field `isProvider` on input type `ContactRelationInput` was added.



GraphQL Code Generator

Overview

GraphQL Code Generator v0.11—Generate React and Angular Apollo Components, Resolver signatures and much more!

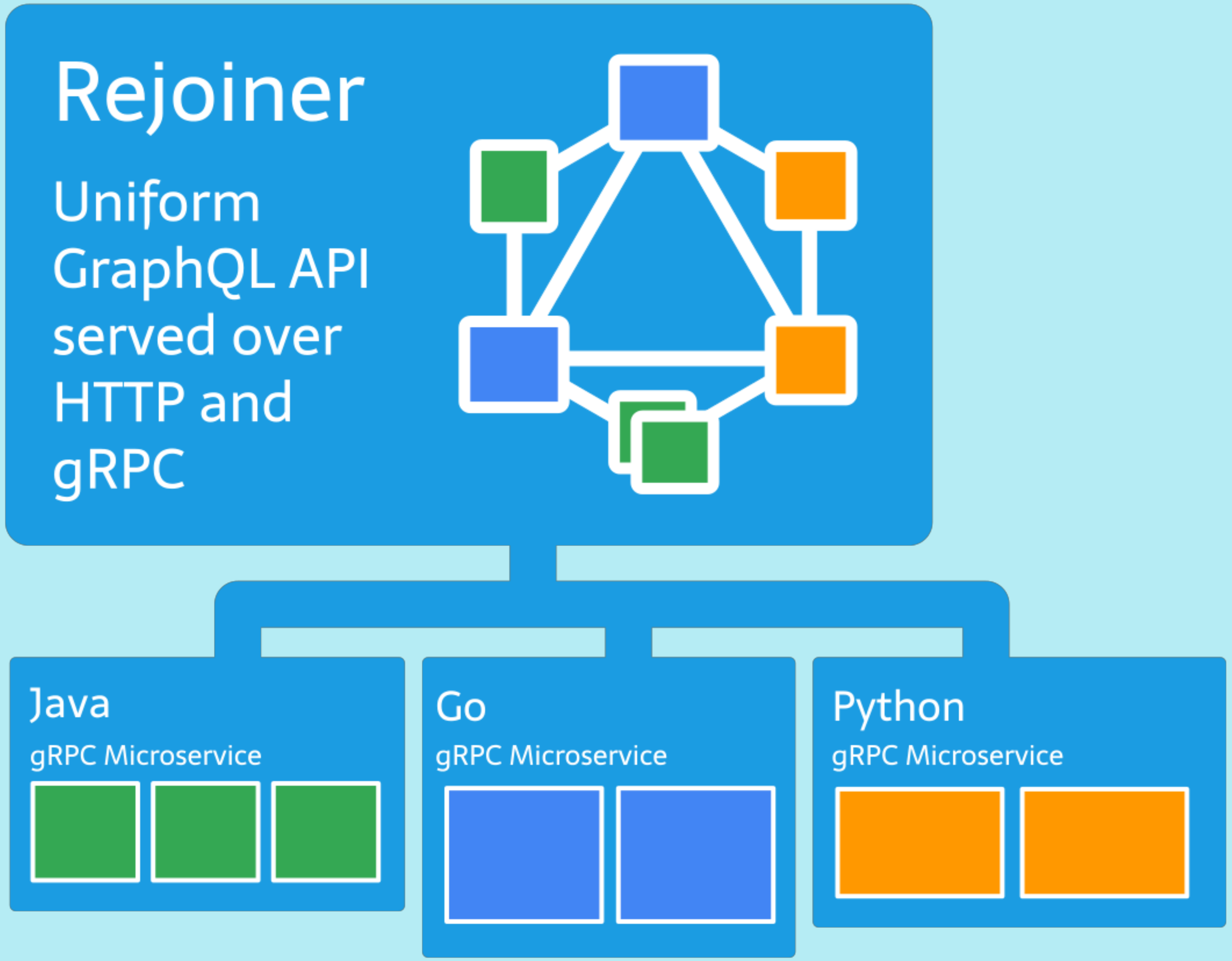
What's new in graphql-code-generator 0.9.0? @ Medium

GraphQL Codegen blog post & examples @ Medium

GraphQL code generator, with flexible support for multiple languages and platforms, and the ability to create custom generated projects based on GraphQL schema or operations.

GraphQL entities are defined as static and typed, which means they can be analyzed and use as a base for generating everything.

This generator generates both models (based on GraphQL server-side schema), and documents (client-side operations, such as `query`, `mutation` as `subscription`).


Rejoiner
Uniform GraphQL API served over HTTP and gRPC
Java
gRPC Microservice
Go
gRPC Microservice
Python
gRPC Microservice

Agenda

A Gentle Introduction to Types

Why Less is Better

Types Over the Network

Agenda

A Gentle Introduction to Types

Why Less is Better

Types Over the Network

Agenda

A Gentle Introduction to Types

Why Less is Better

Types Over the Network

sugarpirate_

poteto

Thank you

Cinemagraph by /u/fezzo

Thank you

Cinemagraph by /u/fezzo