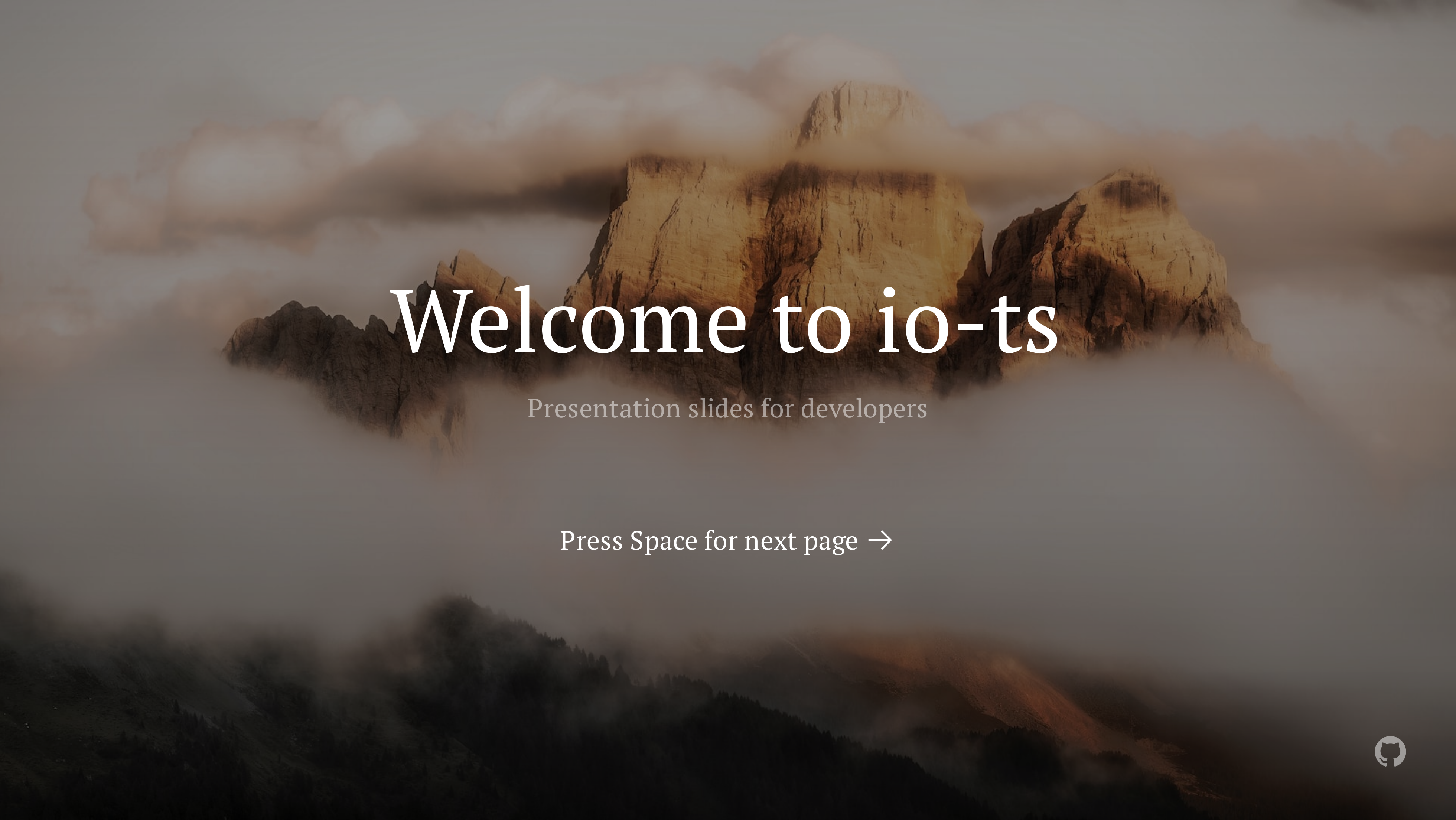

Welcome to io-ts

Presentation slides for developers

Press Space for next page →

What problem do we want to solve?

We want the types to always be consistent with the value

We have

```
type User = {
  name: string;
  age: number;
}
```

We can get

```
const userFromNetwork: unknown = { name: 'foo', age: 33 };
```

We want to achieve

```
const user: E.Either<Error, User> = validate(userFromNetwork);
```

or

```
const user: User = validate(userFromNetwork);
```

How can io-ts help us?

What can we do?

We can build a codec

```
type User = t.TypeOf<typeof User>

const User = t.type({
  name: t.string,
  age: t.number,
});
```

We can validate (decode) the value

```
const user: t.Validation<User> = User.decode(userFromNetwork);
```

or

```
const user: User = pipe(userFromNetwork, User.decode, E.fold((e) => { throw e; }, identity));
```

This is it.

Can we encode & decode an instance of the Date class?

Yes

We have

```
type User = t.TypeOf<typeof User>

const User = t.type({
  name: t.string,
  birthday: DateFromISOString, // npm: io-ts-types
});

const userFromNetwork: unknown = User.encode({ name: 'foo', birthday: new Date() });
```

We can validate (decode) the value

```
const user: t.Validation<User> = User.decode(userFromNetwork);
```

This is it.

Can we encode & decode an object as a string?

Yes

We have

```
type User = t.TypeOf<typeof User>

const User = t.string.pipe(JsonFromString).pipe(t.type({
  name: t.string,
  birthday: DateFromISOString, // npm: io-ts-types
}));
```

and

```
const userFromNetwork: unknown = User.encode({ name: 'foo', birthday: new Date() });
```

We can validate (decode) the value

```
const user: t.Validation<User> = User.decode(userFromNetwork);
```

This is it.

What about an end-to-end experience?

It depends. Most of the time it is easy.

We can do

```
const Request = t.type({ foo: t.literal("foo") });

const Response = t.type({ bar: t.literal("bar") });

export const doSomethingOnClient = (
  request: t.TypeOf<typeof Request>
): E.Either<Error, t.TypeOf<typeof Response>> =>
  pipe(request, Request.encode, fetcher.post, Response.decode, E.mapLeft(E.toError));

export const doSomethingOnServer = (
  request: t.OutputOf<typeof Request>,
  response: { write: (response: unknown) => void }
): void =>
  pipe(
    request, Request.decode, E.fold((e) => { throw E.toError(e); }, identity),
    () => ({ bar: "bar" as const }),
    Response.encode, response.write);
```

This is it.

Table of contents

```
<Toc minDepth="1" maxDepth="1"></Toc>
```

Learn More

Documentation · GitHub

The End

Telegram Channel (RU) · GitHub · LinkedIn