Effect.ts

Przemysław Jan Beigert

- **Github** - https://github.io/przemyslawjanpietrzak
- **Dev.to** - https://dev.to/przemyslawjanpietrzak
- **Stackoverflow** - https://stackoverflow.com/users/5914352/przemyslaw-jan-beigert
- **Linkedin** - https://www.linkedin.com/in/przemysław-beigert-b0b149b4/

Intro

- node http service
- authenticate
- send some request to external services
- filtering & normalization
- safe in DB

And…

- 500
- debugging…
- auth token is missing in the vault
- let’s handle that

Handler

```
class AuthTokenIsMissingError extends Error {}

const loadAuthToken = async () => {
  const response = await fetch();
  if (!respose.token) {
    throw AuthTokenIsMissingError();
  }

  return response.token;
}
```

Handler

```
const main = async () => {
  try {
    await loadAuthToken()
  } catch (e) {
    if (e instanceof AuthTokenIsMissingError) {
      // handler
    }
    throw e;
  }
}
```

A FEW MOMENTS LATER

Later

```
class AuthTokenIsMissingError extends Error {}
class AuthTokenWrongFormatError extends Error {}
class AuthTokenExpiredError extends Error {}
```

Catch

```
} catch (e) {
  if (e instanceof AuthTokenIsMissingError) {
    // handler
  }
  if (e instanceof AuthTokenWrongFormatError) {
    // handler
  }
  if (e instanceof InvalidAuthError) {
    // handler
  }
  throw e;
}
```

Try

```
let value1;
try {
  value1 = fn1();
} catch (e) {

}

const value2 = fn2(value1)

try {
  const value3 = fn3(value3);
} catch (e) {

}
```

Let’s fix that

Effect.ts

- The best way to `` `handle errors` `` in TypeScript
- The best way to `` `manage complexity` `` in TypeScript
- The best way to `` `ship faster` `` in TypeScript

Effect

```typescript
Effect<Success, Error, Requirements>;

const value = Effect.succeed(42);
type Value = Effect<number, never, never>;

const fail = Effect.fail(new Error(''));
type Fail = Effect<never, Error, never>;
```

Program

```
import { Effect, pipe } from 'effect';

const program = pipe(
  Effect.succeed(42),
  Effect.map(item => item + 1),
)

const result = Effect.runSync(program); // 43
```

FlatMap

```typescript
import { Effect, pipe } from 'effect';

const program = pipe(
  Effect.succeed(42),
  Effect.flatMap(item => Effect.succeed(item + 1)),
) satisfies Effect<number, never, never>;

const result = Effect.runSync(program);
```

Fail

```typescript
import { Effect, pipe } from 'effect';

const program = pipe(
  Effect.succeed(42),
  Effect.flatMap(item => item % 2 == 0 ? Effect.succeed(item + 1) : Effect.fail(new ValueError()))),
) satisfies Effect<number, ValueError, never>;

const result = Effect.runSync(program);
```

Many errors

```
import { Effect, pipe } from "effect"

const program = pipe(
  Effect.succeed(42),
  Effect.flatMap(item => item % 2 == 0 ? Effect.succeed(item + 1) : Effect.fail(new FirstError())),
  Effect.flatMap(item => item % 3 == 0 ? Effect.succeed(item + 1) : Effect.fail(new SecondError())),
) satisfies Effect<number, FirstError | SecondError, never>;
```

Map

```
import { Effect, pipe } from 'effect';

const program = pipe(
  Effect.succeed(42),
  Effect.flatMap(item => item % 2 == 0 ? Effect.succeed(item + 1) : Effect.fail(new ValueError()))),
  Effect.map(item => item - 1),
) satisfies Effect<number, ValueError, never>;

const result = Effect.runSync(program);
```

Error handler

```
import { Effect, pipe } from "effect"

class AuthTokenIsMissingError {
  readonly _tag = "AuthTokenIsMissingError"
}

const program = pipe(
  Effect.succeed(42),
  Effect.flatMap(item => item % 2 == 0 ? Effect.succeed(item + 1) : Effect.fail(new AuthTokenIsMissingError())),
  Effect.map(item => item - 1),
  Effect.catchTag("AuthTokenIsMissingError", () => Effect.succeed(0))
) satisfies Effect<number, never, never>;
```

Typed errors

```
import { Effect, pipe } from "effect"

class AuthTokenIsMissingError  {
  readonly _tag = "AuthTokenIsMissingError"
}

const program = pipe(
  Effect.succeed(42),
  Effect.flatMap(item => item % 2 == 0 ? Effect.succeed(item + 1) : Effect.fail(new AuthTokenIsMissingError())),
  Effect.catchTag("InvalidAuthError", () => Effect.succeed(0))
  // Argument of type '"InvalidAuthError"' is not assignable to parameter of type '"AuthTokenIsMissingError"'
) satisfies Effect<any, never, any>;
```

Die!

```typescript
import { Effect, pipe } from "effect"

const program = pipe(
  Effect.succeed(42),
  Effect.flatMap(item => item % 2 == 0 ? Effect.succeed(item + 1) : Effect.die("Angry message")),
) satisfies Effect<number, never, number>;
```

Cons

Scopes

```
const program = pipe(
  Effect.succeed(42),
  Effect.map(value => value + 1),
  Effect.map(newValue => newValue + value),
  // Cannot find name 'value'
);
```

Generator

```
import { Effect, pipe } from "effect"

const program = Effect.gen(function*() {
  const value = 42;
  const newValue = value + 1;
  return newValue + value;
});
```

Generator & effects

```typescript
import { Effect, pipe } from "effect"

const asyncValue = Effect.promise(() => Promise.resolve(42))

const asyncIncrease = (value: number) => Effect.promise(() => Promise.resolve(value + 1))

const program = Effect.gen(function*() {
  const value = yield* asyncValue;
  const newValue = yield* asyncIncrease(value);
  return newValue + value;
});
```

Contexts

```typescript
import { Context, Ref } from "effect";

export class AuthContext extends Context.Tag("Auth")<AuthContext, Ref<string>>() {}
```

Usage

```
const program = pipe(
  Effect.promise(() => fetch('')),
  Effect.flatMap((token) =>
    Effect.gen(function* () {
      const authContext = yield* AuthContext;
      yield* Ref.update(authContext, () => token);

      return 0;
    }),
  ),
);
```

Error

```
const result = Effect.runPromise(program);
// Argument of type 'Effect<number, never, AuthContext>'
// is not assignable to parameter of type 'Effect<number, never, never>'
// Type 'AuthContext' is not assignable to type 'never'.
```

Runnable

```
const runnable = program.pipe(
  Effect.provideService(AuthContext, {
    next: Effect.sync(() => "Default")
  }),
);

const result = Effect.runPromise(runnable);
```

#How?

```
const program = pipe(
  Effect.promise(() => fetch('')),
  Effect.flatMap((token) =>
    Effect.gen(function* () {
      const authContext = yield* AuthContext;
      yield* Ref.update(authContext, () => token);

      return 0;
    }),
  ),
);
```

How???

TS understands

- `return`

```
let fn = () => { return 42; };
```

- `throw`

```
let fn = () => throw new Error();
```

- `yield*`

```
const authContext = yield* AuthContext;
```

Benefits

- Contexts
- Strongly typed
- Dependency injection
- Wrapping libs into services
- Unit tests++

Retry

```
import { Effect, Data } from 'effect';

class RedeployError extends Data.Error {}

const task = Effect.gen(function* () {
  if (42 % 2 === 0) {
    return 43;
  } else {
    yield* new RedeployError();
  }
});

const program = Effect.retry(task, {
  times: 7,
  schedule: Schedule.fixed('1 seconds')
  until: (err) => !(err instanceof RedeployError),
});
```

Timeout

```
import { Effect, Data } from 'effect';

const task = Effect.gen(function* () {
  if (42 % 2 === 0) {
    yield* Effect.sleep("2 seconds");

    return true;
  } else {
    yield* Effect.sleep("5 seconds");

    return false;
  }
});

const program = task.pipe(Effect.timeout("4 seconds"))
```

Final effect

```
const program = pipe(
  Effect.all(
    [fetchExistingWebhookEventsEffect, fetchWebhookEventsEffect, fetchAuthEffect(tokenCredentialId)],
    { concurrency: 3 },
  ),
  Effect.map(filterToNotExistingWebHookEvents),
  Effect.flatMap((webhookEvents) =>
    Effect.all(webhookEvents.map((webhookEvent) =>
      Effect.gen(function* () {
        const [pipelineTimelines, pipelineBridges] = yield* Effect.all([
          fetchPipelineTimelineEffect(webhookEvent),
          fetchPipelineBridgesEffect(webhookEvent),
        ], {concurrency: 2});
        const reRun = calculatePipelineRerun(pipelineTimelines, pipelineBridges, webhookEvent);
        const jobs = filterDuplicatedJobs(webhookEvent, [...pipelineTimelines, ...pipelineBridges]);
        const normalizeAndIngestOriginPipelineEffect = normalizeAndIngestOriginPipelineEffectFactory({
          jobs,
          webhookEvent,
          reRun,
        });
        const loadAndIngestStepsEffect = loadAndIngestStepsEffectFactory(webhookEvent, jobs, reRun);
        return Effect.all([normalizeAndIngestOriginPipelineEffect, loadAndIngestStepsEffect], { concurrency: 2 });
      }),
    ), { concurrency: 8 }),
  ),
);
```

Benefits

- Typed errors
- Elastic error handling
- Declarative
- Modular
- DI without classes
- Time utils

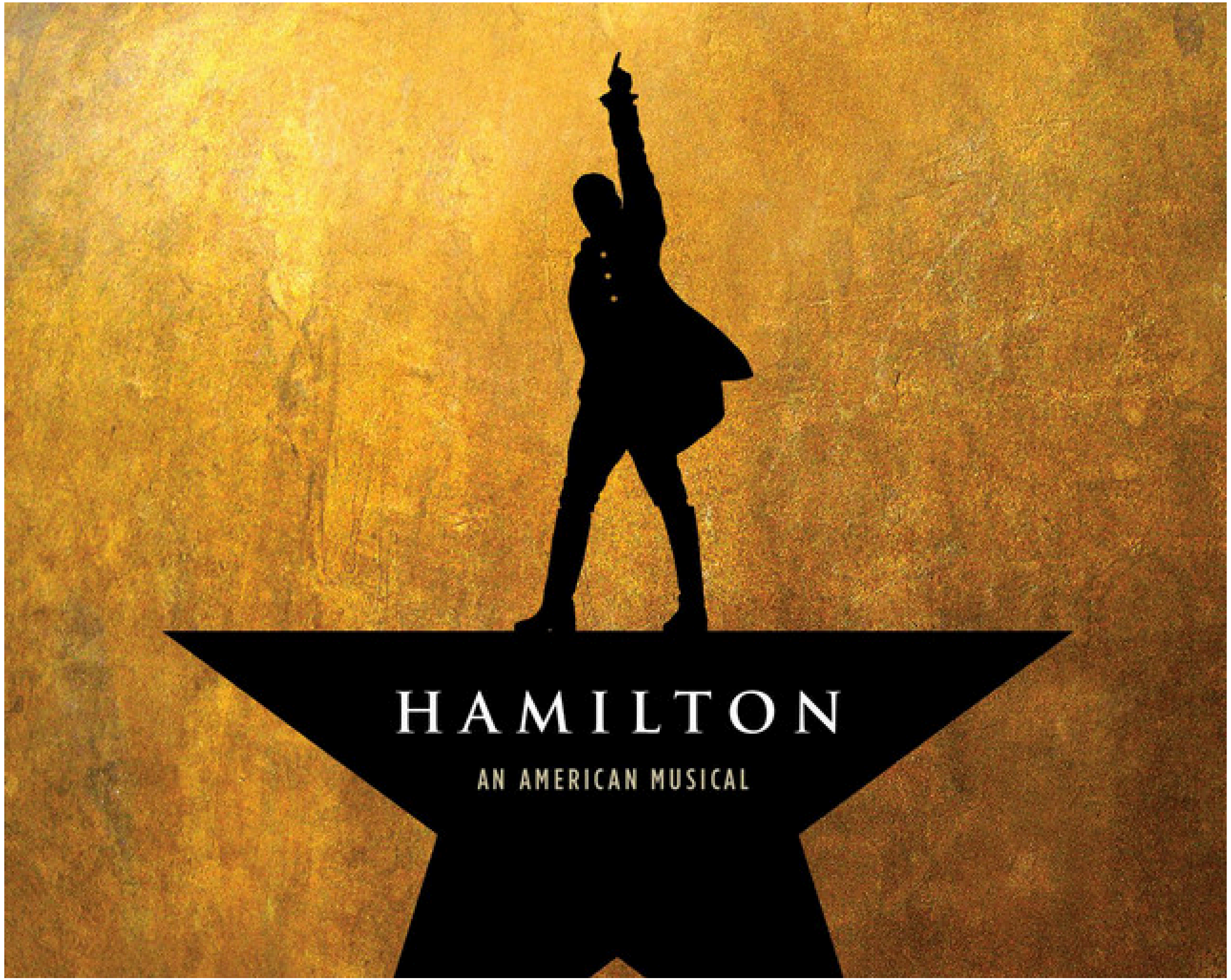
Cons

- NOT easy to migrate *

- Opinionated

```
export declare function pipe<A>(a: A): A;
export declare function pipe<A, B = never>(a: A, ab: (a: A) => B): B;
export declare function pipe<A, B = never, C = never>(a: A, ab: (a: A) => B, bc: (b: B) => C): C;
export declare function pipe<A, B = never, C = never, D = never>(a: A, ab: (a: A) => B, bc: (b: B) => C, cd: (c:
export declare function pipe<A, B = never, C = never, D = never, E = never>(a: A, ab: (a: A) => B, bc: (b: B) =>
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never>(a: A, ab: (a: A) => B, bc:
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never>(a: A, ab: (a: A
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never>(a: A
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
export declare function pipe<A, B = never, C = never, D = never, E = never, F = never, G = never, H = never, I =
```

Summary


HAMILTON
AN AMERICAN MUSICAL

Cost

- Generators instead of methods
- No more classes
- Stateless
- Wrap everything

Benefits

- Typed errors
- Optimistic approach
- Typed DI
- Stateless
- Time utils
- Schema validator
- Unit test support
- Debug support

Ewolucja czy ślepy zaułek?

Recommended

- wanna be senior dev
- intro to functional languages
- long pipeline with branches

Questions?

Thank you :*