Railway Oriented Programming

어떻게 에러를 핸들링 할 것인가?

2023.09.24 민경민

아래 코드의 문제점이 뭘까요?

```ts
interface User {
  id: number;
  name: string;
}

const userSchema = {
  parse: function (json: any): User {
    const { id, name } = json;

    if (typeof id !== "number") throw new Error("id must be a number");
    if (typeof name !== "string") throw new Error("name must be a string");

    return Object.assign({}, { id, name });
  },
};

async function getUser(): Promise<User> {
  const data = await fetch("https://example.com/api/user");
  const json = await data.json();
  return userSchema.parse(json);
}
```

3줄의 코드에

잠재적 버그가 3개

```
17  async function getUser(): Promise<User> {
18    const data = await fetch("https://example.com/api/user");
19    const json = await data.json();
20    return userSchema.parse(json);
21  }
```

JS의 고질적인 문제점

- fetch( )와 json( ) 는 promise를 리턴함
- promise가 reject 되면 async function 내에서 error 발생
- parse( )는 error throw 할 수 있음

- **함수의 시그니처를 살펴봐도 이놈들이 에러를 야기할 수 있는 것을 알 수 없음!**

일차원적인 해결

try/catch

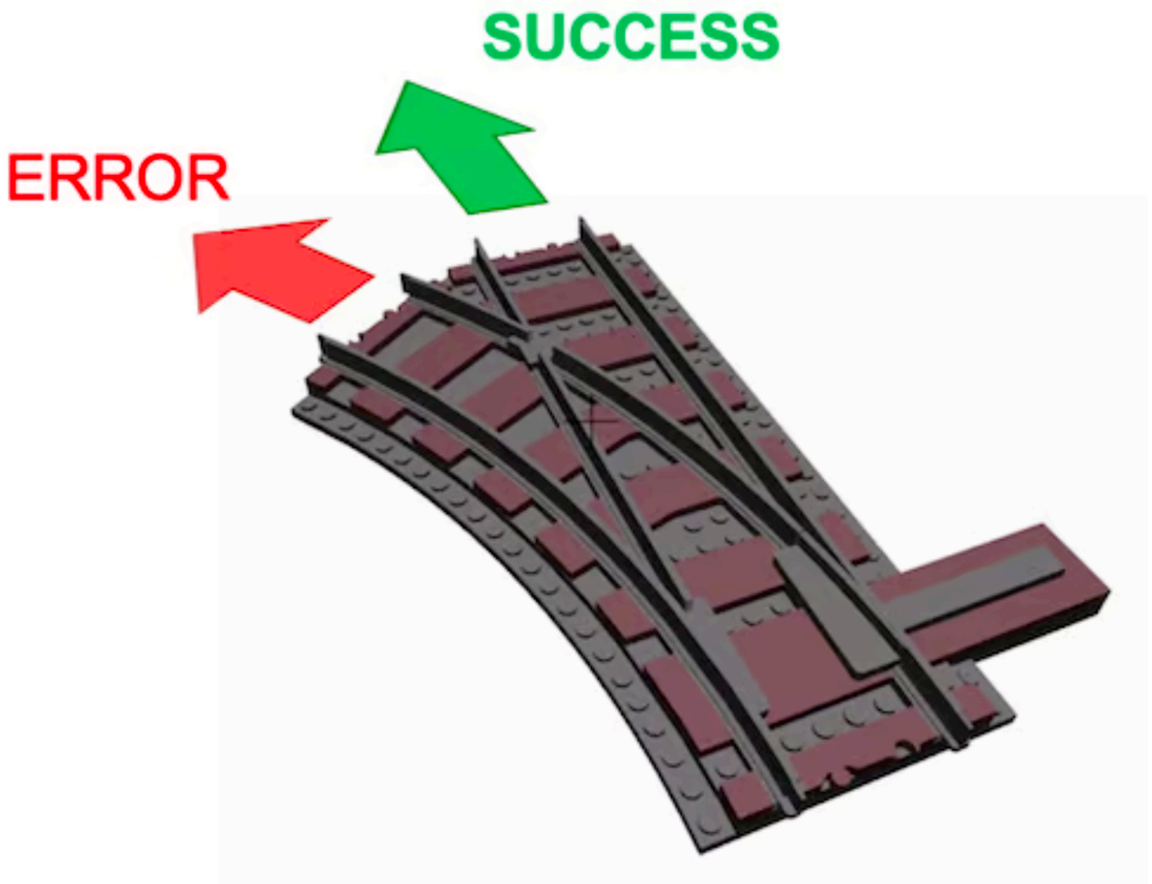
```
17  async function getUser(): Promise<User | null> {
18    try {
19      const data = await fetch("https://example.com/api/user");
20      const json = await data.json();
21
22      return userSchema.parse(json);
23    } catch (error) {
24      console.error(error);
25
26      return null;
27    }
28  }
29
```

암묵지 의존하는 방식

어떤 에러가 언제 발생하는지 알아야함

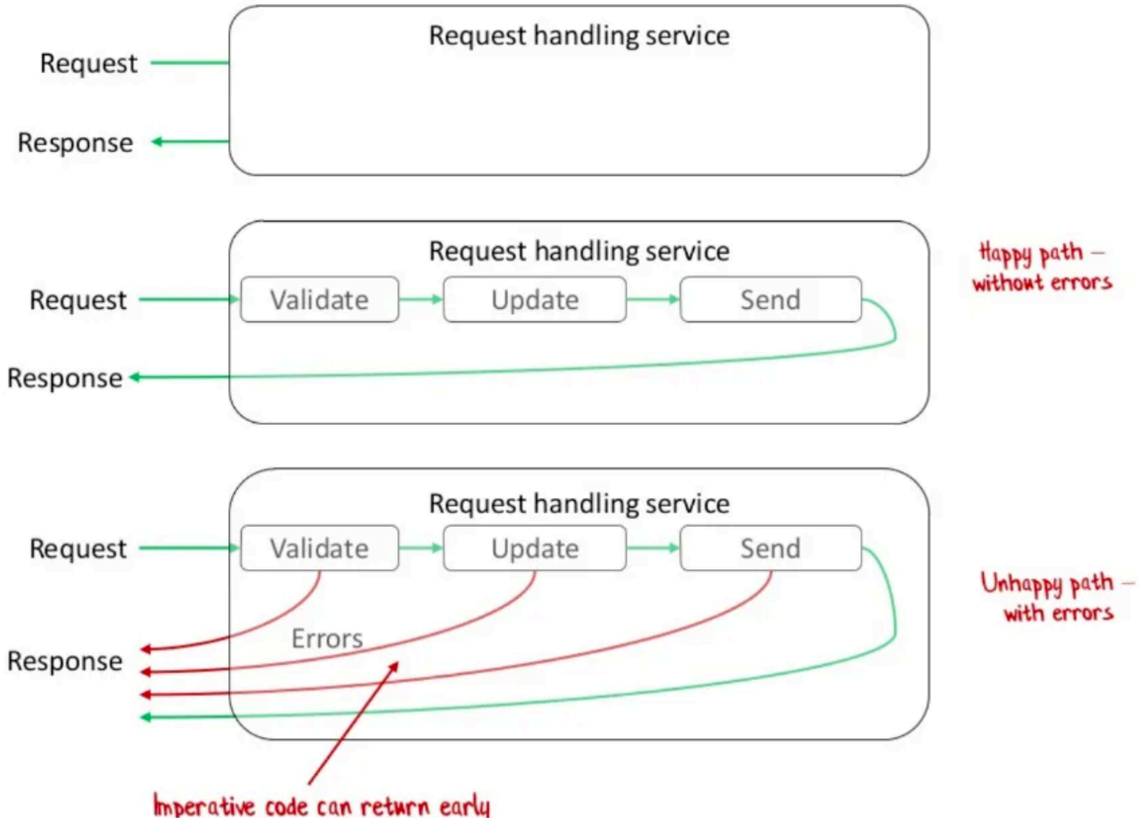
Railway Oriented Programming

함수형 파라다임에서 에러를 핸들링



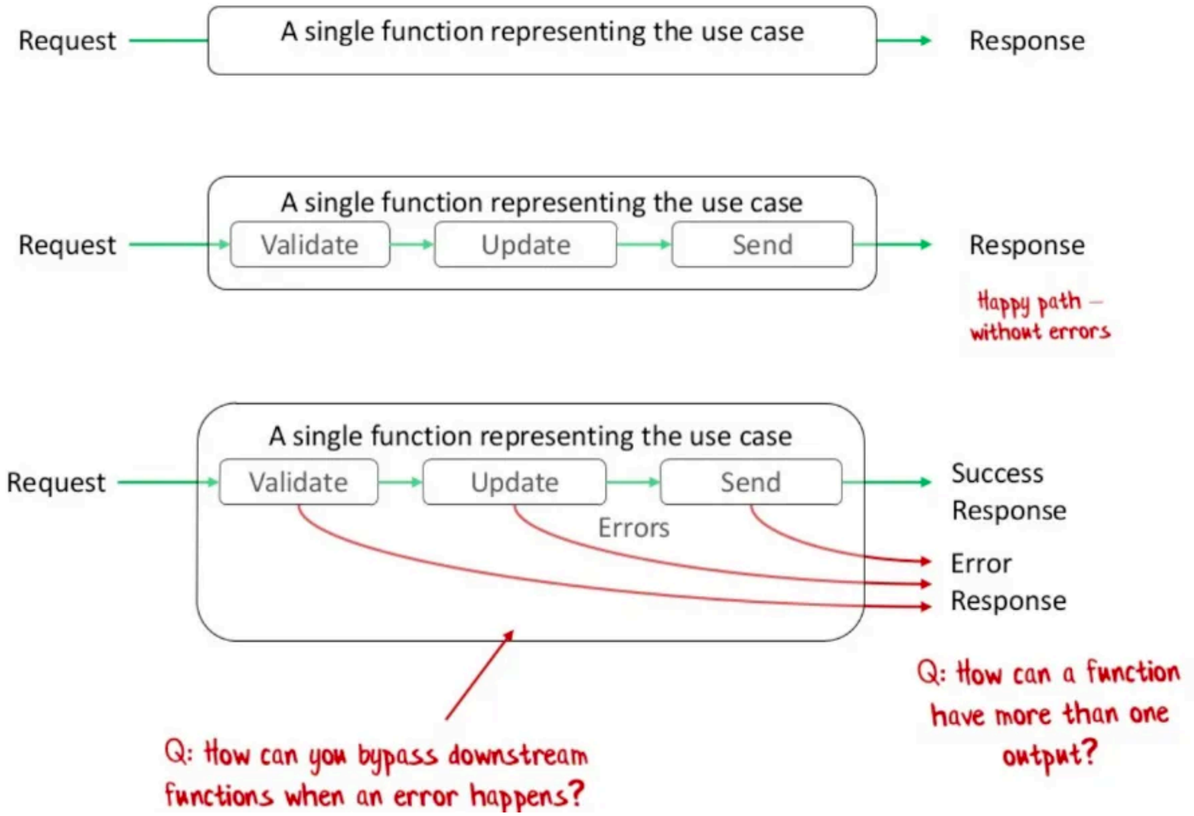
일반적 디자인

error를 throw

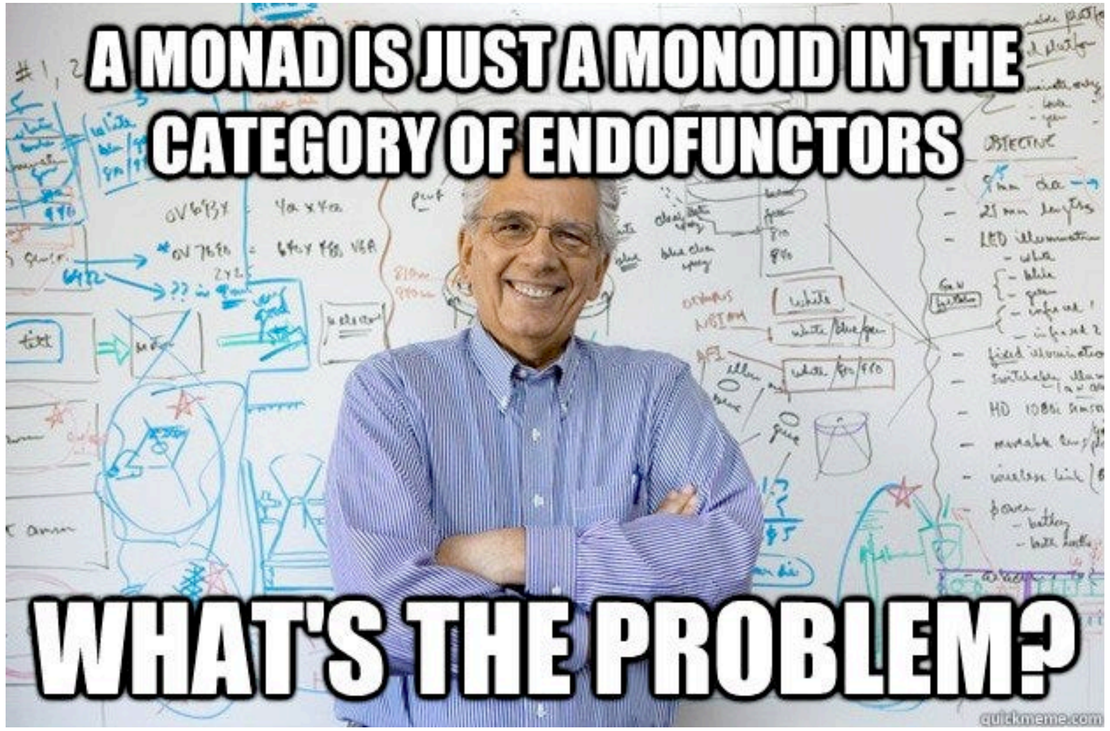


함수형 디자인

error를 타입으로 취급



모나드를 사용하자


A MONAD IS JUST A MONOID IN THE CATEGORY OF ENDOFUNCTORS
WHAT'S THE PROBLEM?
quickmeme.com

모나드의 인터페이스

```
// Monad Interface
interface Monad<T> {
  unit(val: T): Monad<T>;
  bind<R>(f: (val: T) => Monad<R>): Monad<R>;
}
```

- Unit Function : 특정한 값을 받고 해당 값을 문맥(모나드 구조)로 감싸는 함수
- Bind Function : 모나드를 리턴하는 연산을 인자로 받고, 체이닝을 가능하게 하는 함수

Result 모나드

Success와 Failure의 Union(합집합)

```typescript
// monadic interface
interface Success<T> {
  readonly success: true;
  readonly value: T;
  readonly bind: <U>(func: (value: T) => Result<U>) => Result<U>;
}

interface Failure {
  readonly success: false;
  readonly error: string;
  readonly bind: <U>(func: (value: any) => Result<U>) => Result<U>;
}

type Result<T> = Success<T> | Failure;

```

인터페이스 구현

unit과 bind

```
16  // helper function to create a success or failure
17  function success<T>(value: T): Result<T> {
18    return { success: true, value, bind };
19  }
20
21  function failure<T>(message: string): Result<T> {
22    return { success: false, error: message, bind };
23  }
24
25  // monad functions
26  function unit<T>(value: T): Result<T> {
27    return success(value);
28  }
29
30  function bind<T, U>(this: Result<T>, func: (value: T) => Result<U>): Result<U> {
31    if (this.success) {
32      return func(this.value);
33    } else {
34      return failure<U>(this.error);
35    }
36  }
```

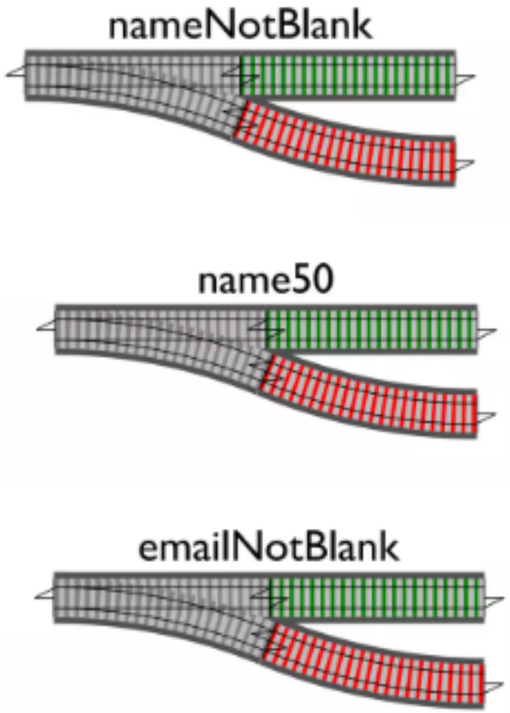
실제로 사용해보자

Request validation

- 입력값의 이름은 빈값이면 안된다
- 입력값의 이름은 50자가 넘으면 안된다
- 입력값의 이메일은 빈값이면 안된다

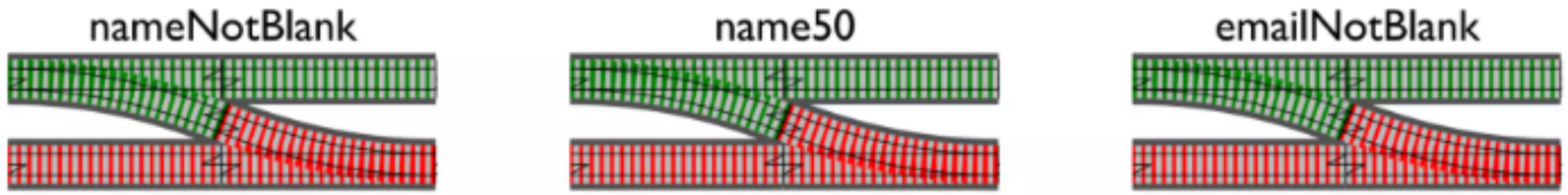
Validations

```ts
// validations
interface Input {
  readonly name: string;
  readonly email: string;
}

function nameNotBlank(input: Input): Result<Input> {
  return input.name.trim() === ""
    ? failure("Name cannot be blank")
    : success(input);
}

function name50(input: Input): Result<Input> {
  return input.name.length > 50
    ? failure("Name cannot be longer than 50 characters")
    : success(input);
}

function emailNotBlank(input: Input): Result<Input> {
  return input.email.trim() === ""
    ? failure("Email cannot be blank")
    : success(input);
}
```



모나드를 사용한 체이닝

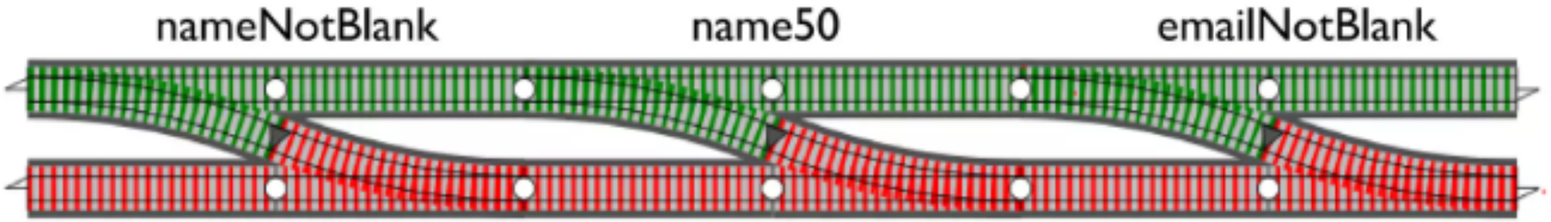
bind( ) function




```
30  function bind<T, U>(this: Result<T>, func: (value: T) => Result<U>): Result<U> {
31    if (this.success) {
32      return func(this.value);
33    } else {
34      return failure<U>(this.error);
35    }
36  }
37
```

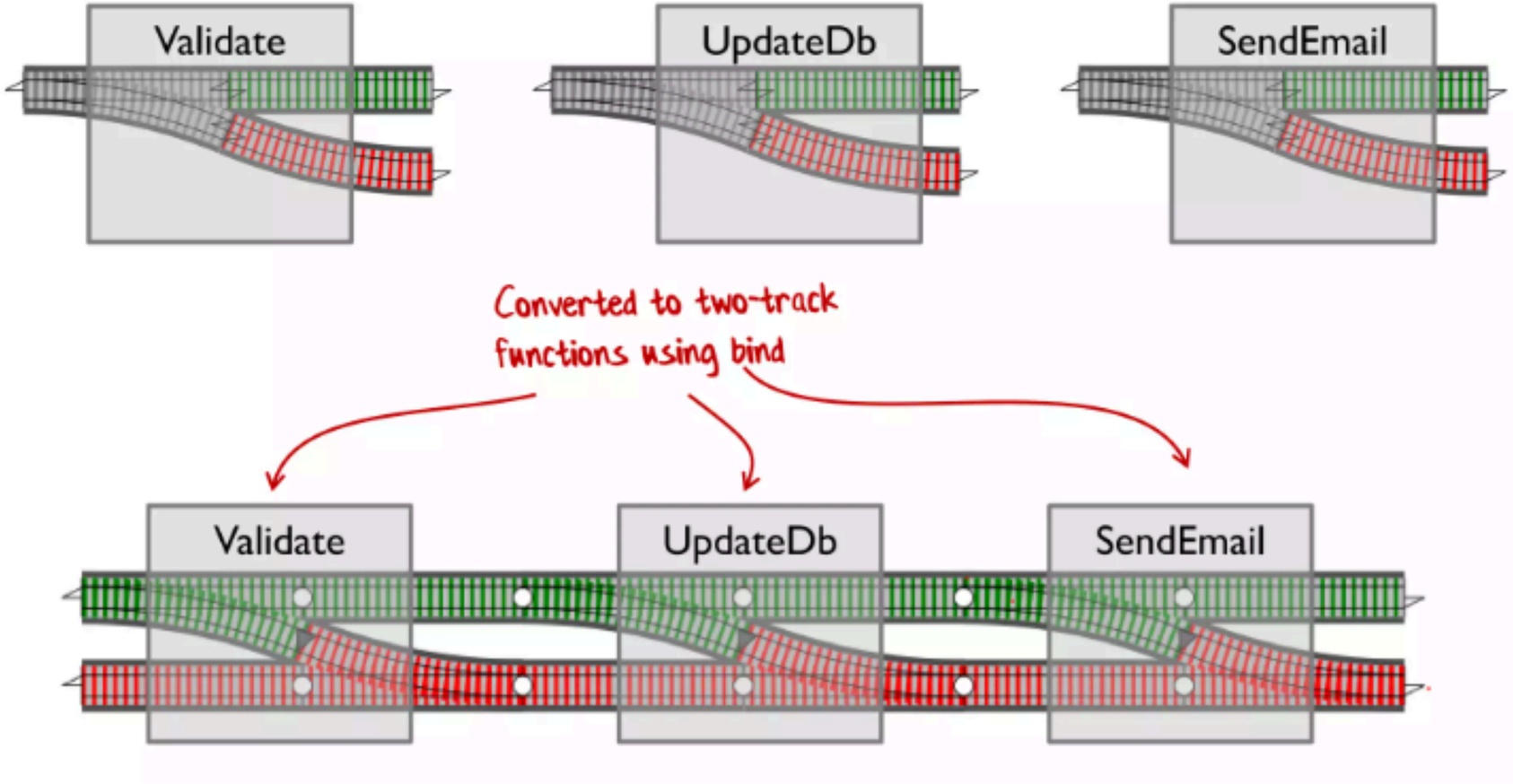
- Bind Function : 모나드를 리턴하는 연산을 인자로 받고, 체이닝을 가능하게 하는 함수

칙칙폭폭




```
65  function validateRequest(input: Input): Result<Input> {
66    return unit(input).bind(nameNotBlank).bind(name50).bind(emailNotBlank);
67  }
```

ROP 확장


Validate
UpdateDb
SendEmail
Converted to two-track
functions using bind
Validate
UpdateDb
SendEmail

ROP 장단점

모든 것은 트레이드 오프

- Clear Error Handling
- Composability
- Funcional Programming

- Complexity
- Overhead
- Debugging Challenges

참고자료

- https://www.slideshare.net/ScottWlaschin/railway-oriented-programming