ToiletPaper #150

Try but don't catch: Elegantes Error Handling mit TypeScript

Autor: Robert Gruner / Software Engineer / Standort Leipzig

✗ Problem

Keine Lust mehr auf *throw*? Keine Lust mehr darauf, für jeden Rückgabewert mittels *If* zu checken, ob der Wert auch wirklich angekommen ist? Keine Lust mehr auf Boilerplate Code mit *try/catch*? Ihr sucht eine Implementierung der *Result-/Optional-/Maybe*-Monade? Oder habt mal etwas von *Railway Oriented Programming* gehört und wollt das mal ausprobieren?

✓ Lösung

Wenn ihr im TypeScript unterwegs seid, gibt es da ein schönes Tool für euch – **neverthrow**! Wie der Name schon sagt, schreibt ihr damit Code, der keine Fehler mehr wirft oder fängt. Stattdessen gibt es einen *Result*-Typen, welcher die Zustände *Ok* oder *Err* annehmen kann. Das Beste daran ist, dass ihr ganz im Sinne des *Railway Oriented Programming* verschiedene *Results* aneinander "ketten" und zwischen den Zuständen wechseln könnt, d.h. in den *Err*-Zustand übergehen oder sogar aus dem *Err*-Zustand wieder in den *Ok*-Zustand "recovern". Als wäre das noch nicht genug, bietet die Lib auch Interoperabilität mit *Promise* durch den *ResultAsync*-Typen.

➔ Beispiel

```
  function hasKeys<T extends object>(item: T, keys: Array<keyof T>): Result<T, string> {
    const requiredKeysArePresent = keys.every((key) => key in item);
    return requiredKeysArePresent ? ok(item) : err('Object to validate has missing keys');
  }
```

```
// Nutzung der synchronen Result API
async function storeCoffee(candidate?: Coffee): Promise<void> {
  const validationResult: Result<Coffee, string> = exists(candidate)
    // Nur wenn Ok zurückkommt, wird mit dem Code im "andThen" fortgefahren
    .andThen((existingCandidate) => hasKeys(existingCandidate, ['name', 'origin']));
  if (validationResult.isOk()) {
    await saveCoffeeInDb(validationResult.value);
  }
}
```

```
// Umwandlung einer Promise zu ResultAsync
function get<T>(url: string): ResultAsync<T, ApiError> {
  return fromPromise(
    httpClient.get<T>(url, { responseType: 'json' }).then((result) => result.body),
    (e) => mapToApiError(e)); // Ersatz für die "catch" Methode der Promise
}
```

```
function getCoffeeList(): Promise<Coffee[]> {
  return get<Coffee[]>('api.jambit.com/coffee')
    .mapErr((e: ApiError) => {
      logger.error(`Could not get coffee list because of: ${e}`);
      return [coffeeA, coffeeB]; // Recover mit dem fallback value
    }).match((coffeeList: Coffee[]) => {
        logger.info(`There are ${coffeeList.length} types of coffee available.`);
        return coffeeList;
      }, (fallbackList) => fallbackList));
}
```

+ Weiterführende Aspekte

- https://github.com/supermacro/neverthrow