ToiletPaper #150

Try but don't catch: Elegant error handling with TypeScript

Author: Robert Gruner / Software Engineer / Office Leipzig

✗ Problem

Tired of *throw*? No longer checking each return value via *If* to see whether the value is really present? Tired of boilerplate code with *try/catch*? You are looking for an implementation of the *Result/Optional/Maybe* monad? Or have you heard about *railway oriented programming* and want to try it out?

✓ Solution

If you are using TypeScript, there is a nice tool for you – **neverthrow**! As the name suggests, it lets you write code that no longer throws or catches errors. Instead, there is a *Result* type that can take the states *Ok* or *Err*. The best thing about it, in the sense of *railway oriented programming*, is that you can "chain" different *Results* together and switch between the states, i.e. switch to the *Err* state or even "recover" from the *Err* state back to the *Ok* state. As if that weren't enough, the library also provides interoperability with *Promise* through the *ResultAsync* type.

➔ Example

```
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');
}
```

```
// Using the synchronous Result API
async function storeCoffee(candidate?: Coffee): Promise<void> {
  const validationResult: Result<Coffee, string> = exists(candidate)
    // Only if Ok is returned, the code is continued in "andThen".
    .andThen((existingCandidate) => hasKeys(existingCandidate, ['name', 'origin']));
  if (validationResult.isOk()) {
    await saveCoffeeInDb(validationResult.value);
  }
}
```

```
// Converting a Promise to ResultAsync
function get<T>(url: string): ResultAsync<T, ApiError> {
  return fromPromise(
    httpClient.get<T>(url, { responseType: 'json' }).then((result) => result.body),
    (e) => mapToApiError(e)); // Replacement for the "catch" method of 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 with fallback value
    }).match((coffeeList: Coffee[]) => {
        logger.info(`There are ${coffeeList.length} types of coffee available.`);
        return coffeeList;
      }, (fallbackList) => fallbackList));
}
```

✚ Further Aspects

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