ToiletPaper #159

jambit
WHERE INNOVATION WORKS

Something like Scala's for comprehension in Typescript

Autor: André Petermann / Senior Software Architect / Standort Leipzig

✕ Problem

Monaden wie `Option`, `Result`, `Either` oder `Try` sind gut geeignet, um Programme zu schreiben, die Domänenlogik und Behandlung von Fehlern oder Abwesenheit von Werten klar trennen. Allerdings führt die Umsetzung dieses Ansatzes in der Praxis oft zu großen Closures mit tief verschachtelten Code-Blöcken. Haskells *do notation* oder Scalas *for comprehension* sind Sprachkonstrukte, die dieses Problem lösen. Die Bibliothek *for-comprehension-ts* bietet ein ähnliches Sprachkonstrukt zur *for comprehension* in Typescript. Scalas *for comprehension* sieht zum Beispiel wie folgt aus:

```
val result: Either[String, Int] =
  for {
    dividend <- Right(42)
    divisor <- Right(2)
    divisorVerified <- if (divisor != 0) Right(divisor) else Left("Divisor must not be zero!")
  } yield dividend / divisor

println(result match {
  case Right(value) => value
  case Left(error) => error
})
```

Allerdings gibt es kein Äquivalent in Typescript.

✓ Lösung

Die Bibliothek for-comprehension-ts bietet ein ähnliches Sprachkonstrukt zur for *comprehension* in Typescript:

```
const result: Result<number, string> =
    For._("dividend", success(42))
       ._("divisor", success(2))
       ._("divisorVerified", ({divisor}) => divisor != 0 ? success(divisor) : failure("Divisor must not
          Be zero!"))
       .yield(({dividend, divisorVerified}) => dividend / divisorVerified)
console.log(isSuccess(result) ? result.value : result.error)
```

Die Bibliothek beinhaltet auch eine asynchrone Variante, die eine nahtlose Integration von normalen und asynchronen Funktionen mit denselben Monaden erlaubt. Das funktioniert ganz ohne explizites Handling von Promises.
for-comprehension-ts kann mit allen Implementierungen eines Monad Interfaces verwendet werden, welches die Operationen `map`, `flatMap` und `flatMapAsync` umfasst. Die Bibliothek beinhaltet bereits Implementierungen für die folgenden Monaden:

	`Option<T>`	`Result<T, E>`	`Try<T>`
Abstraktion	An- oder Abwesenheit	Erfolg oder explizit typisierter Fehler	Erfolg oder implizit gefangene Exception
Konstruktoren	`some(value: T)` `none()`	`success(value: T)` `failure(error: E)`	`ok(value: T)` `error(error: any)`

Im Gegensatz zu Pipes erlaubt die Syntax, dass Programme gerichtete azyklische Graphen (DAG) sein können, deren Knoten benannte Werte (z. B. a=3) und deren Kanten Funktionen sind. Der Graph wird lazy ausgewertet, d. h., die tatsächliche Ausführung erfolgt erst, wenn yield aufgerufen wird. Aus diesem Grund können for-comprehension-ts-Programme dupliziert, verzweigt und wiederholt werden. Dabei werden Operationen nur so lange ausgeführt, bis der erste Fehler oder fehlende Wert auftritt. Das genaue Verhalten häng dabei stets von der verwendeten Monade ab.

+ Beitragen

for-comprehension-ts wird von jambit entwickelt und ist bisher noch nicht in produktivem Einsatz. Ihr könnt also gern zum Projekt beitragen oder einfach die Bibliothek benutzen:

- Quellcode: https://github.com/p3et/for-comprehension-ts
- Scalas for comprehension: https://www.baeldung.com/scala/for-comprehension