HAW HAMBURG

BACHELOR THESIS
Soheil Nazari

Robusteres State Management in Frontend Webapplikationen mit DFA Übergängen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Soheil Nazari

Robusteres State Management in Frontend Webapplikationen mit DFA Übergängen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Wirtschaftsinformatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr.-Ing. Lars Hamann

Eingereicht am: 13. Januar 2025

Soheil Nazari

Thema der Arbeit

Robusteres State Management in Frontend Webapplikationen mit DFA Übergängen

Stichworte

State Management, Webapplikationen, Frontend

Kurzzusammenfassung

Arthur Dents Reise in eine neue Zukunft ...

Soheil Nazari

Title of Thesis

Making State Management in Frontend Web Applications Robuster with DFA Transitions

Keywords

State Management, Web Applications, Frontend

Abstract

Arthur Dents travel to a new future ...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Die Rolle des State-Managements in Frontend Webapplikationen

Moderne Webseiten folgen dem Single Page Appliaction (SPA) Ansatz. Dem nach bleibt die gleiche Instanz der Webapplikation solange der Nutzer auf der Webseite ist, bestehen. In der Regel sind mehrere Teile einer Applikation, beispielsweise bei der Komponenten-Architektur, von gleichen Daten abhängig. Außerdem werden die Daten basierend auf Interaktionen des Benutzers modifiziert. Änderungen in den Daten müssen den betroffenen Komponenten mitgeteilt werden. In einigen Fällen ist die Synchronisierung der Daten im Frontend mit den Daten des Servers erforderlich. Um HTTP Aufrufe zu sparen, können verschiedene Mechanismen, wie beispielsweise Caching oder Debouncing verwendet werden. Diese Faktoren erhöhen, die ohnehin schon hohe Komplexität und Fehleranfälligkeit zusätzlich.

Um diese Komplexität effizient zu verwalten, werden State-Management Lösungen wie Redux, NgRx oder Pinia verwendet. Mit Hilfe dieser Open Source JavaScript Bibliotheken, können Daten beim Bedarf von einer API abgerufen, transformiert und gespeichert werden. Die meisten State-Management Bibliotheken sind eng mit einem UI-Framework gekoppelt. Aus diesem Grund sind sie ein fundamentaler Baustein jeder größeren Frontend Webapplikation.

1.2 Ziel der Arbeit

Mit der Komplexität erhöht sich auch die Fehleranfälligkeit. Fehler im Zustand, also Daten der Applikation, haben einen direkten Einfluss auf das Angezeigte. Wenn die Applikation sich in einem „falschen“ Zustand befindet und es keine Laufzeitfehler gab, kön-

nen die Verantwortlichen (in der Regel, die Entwickler) unter Umständen, nicht darüber informiert sein. Dies führt zu langlebigen Bugs.

Ziel dieser Arbeit ist es, einen Ansatz zu erarbeiten, bei dem die Möglichkeit eines Befindens in einem „falschen“ oder „illegalem“ Zustand eliminiert wird. Dazu wird jeder zusammenhänge Teil des Zustands als ein endlicher Automat abgebildet. Dahingehend wird jede Änderung in diesem Zustand wie ein Übergang bei einem endlichen Automaten behandelt. Es wird vorgeschlagen die beliebten State-Management Lösungen um „strikte“ Übergänge, wie bei einem DFA, zu erweitern. Auf diesem Wege wird eine Reduzierung von Bugs in größeren Applikationen bestrebt. Dabei wird insbesondere auf die Lesbarkeit und Wartbarkeit des Quellcodes und die Developer Experience geachtet.

Folgende Forschungsfragen werden behandelt:

1. Können Bugs, die Aufgrund eines falschen Zustandes entstehen, mit Hilfe von „strikten“ Übergängen reduziert werden?
2. Steigt oder sinkt die Developer-Experience?
3. Steigt oder sinkt die Lesbarkeit und Wartbarkeit des Codes?

2 Methodologie

2.1 Code Ausschnitte

TypeScript wird benutzt um, Aufbau von Objekten oder Funktionen zu beschreiben. Längere Strukturen werden mit Hilfe von Code-Bespielen veranschaulicht. Hierfür wird ebenfalls TypeScript verwendet. An viele Stellen wird auf Type-Annotationen verzichtet, damit die Beispiele leicht lesbar bleiben. Es wird auf Semikolon verzichtet, da diese bei JavaScript nicht erforlich sind.

3 State-Management Ansätze

Bei den populären SM Lösungen folgen Redux und NgRx dem Flux-Pattern[4][2], wobei Zustand und Pinia einen anderen, Framework-nahen Ansatz verfolgen.

3.1 Redux

Im Folgenden wird die Funktionsweise und die Eigenschaften von Redux näher beschrieben Die APIs von NgRx unterscheiden sich aufgrund von Bequemlichkeit, allerdings sind die Konzepte immer noch anwendbar.

Redux definiert sich durch folgenden drei Eigenschaften:

1. Unveränderlichkeit (Immutability): Änderung am State sind ausschließlich über die APIs von Redux möglich.
2. Zentralisierung des Zustandes: Der gesamte Applikationszustand lebt in einem zentralen JavaScript Objekt.
3. Nachvollzierbarkeit (Traceability): Während der gesamten Lebensdauer der Applikation sind Änderungen am Zustand auf deren Ursprung verfolgbar.
4. Event basiert: Es wird das Beobachter-Muster (Observer Pattern) verwendet.

Das Verhalten des Stores wird durch *actions* und *reducer* definiert.

3.1.1 Actions

Eine Aktion (Action) beschreibt eine Änderung oder Interaktion in und mit der Applikation. Beispielsweise könnte eine *counter-clicked* Action versendet (dispatch) werden, wenn der Nutzer auf den *Zähler erhöhen* Button drückt. Oder, wenn der Nutzer sich erfolgreich angemeldet hat, kann eine entsprechende Action versendet werden. Intern ist eine Action ein POJO.[5]

Es wird folgende Struktur für Actions empfohlen:

```
type Action<T> = {
  type: string,
  payload: T
}
```

Das Feld *type* beschreibt die Action und das optionale Feld *payload* enthält weiterführende Daten.

3.1.2 Reducer

Ein Reducer übernimmt die Rolle des Beobachters und ist für die Initialisierung und Aktualisierung des Zustandes zuständig. Ein Reducer wird als eine Pure-Funtion mit zwei Parametern definiert. Der erste Parameter ist das Zustandsobjekt und der zweite die Action. Der Rückgabewert der dieser Callback-Funcion ist das neue Zustandsobjekt. Da es sich hier um eine Pure-Funtion handelt, dürfen es hier keine Seiteneffekte stattfinden. Wie anfangserwähnt, ist der Zustand Unveränderlich, daher dürfen hier keine direkten Veränderungen des Zustandes stattfinden. Es wird lediglich ein neues Objekt zurückgegeben. Fall es keine Veränderungen stattfinden sollen, kann das ursprüngliche Objekt aus dem ersten Parameter unverändert zurückgegeben werden.[5]

Es wird folgende Struktur für Reducer empfohlen:

```
type Reducer<S, A> = (state: S, action: A) => S
```

Beispiel reducer:

```
function reducer(state = { user: null }, action) {
  switch (action.type) {
    case 'user-logged-in':
      return {
        ...state,
        user: {
          userId: action.payload.userId
        },
      }
    case 'user-logged-out':
      return {
        ...state,
        user: null,
      }
    default:
      return state
  }
}
```

Es wird die *Spread Syntax: ...* aus ECMAScript 6 genutzt, um as ursprüngliche Zustandsobjekt zu klonen.[3]

3.1.3 Interaktion mit dem Store

Der Store wird mit Hilfe der *createStore* API erstellt. Als Parameter wird die Reducer-Function übergeben. Der Rückgabewert ist das Store-Objekt. Dieses bietet Zugang zu unteranderem *dispatch* und *getState* Methoden. Mit diesen kann jeweils Actions versendet und aus dem Store gelesen werden.

```
import { createStore } from 'redux'

const store = createStore(reducer)
store.dispatch(action)
const user = store.getState().user
```

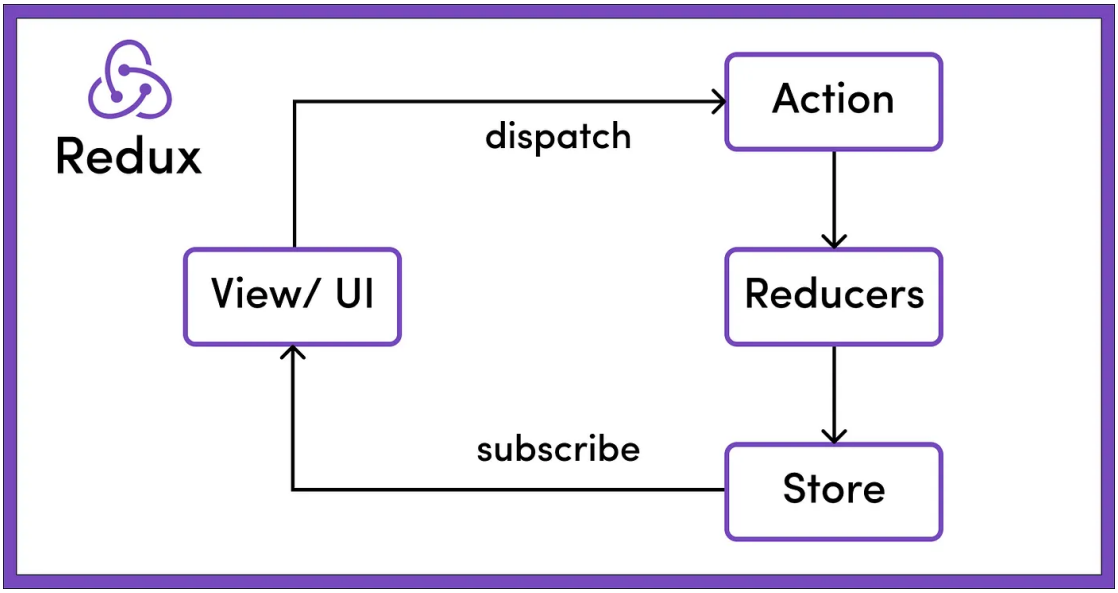

Abbildung 3.1: Redux Datenfluss

3.2 Pinia

Pinia ist sehr eng gekoppelt mit dem Vue Framework und nutzt dessen Mechanismen der Reaktivität zu Datenhaltung. Das führt dazu, dass Pinia selbst minimal bleibt und die Daten ohne weiteres reaktiv sind. Im Gegensatz zu Redux und NgRx setzt diese Store-Lösung nicht das Flux-Pattern um. Dank dieser Praxis, ist weniger Code nötig um einen Store zu definieren. Außerdem folgt Pinia nicht den Single-Store-Ansatz, bei dem alle Daten in einem zentralen Objekt leben. Sondern sind für Teile der Daten eigenständige Store-Instanzen zuständig. Pinia bietet zwei verschiedene APIs zu Definition von Stores an. In dieser Arbeit wird die *Options API* verwendet. Die Konzepte lassen sich auch auf die *Composition API* übertragen.[1] Die zwei essentiellen Konzepte sind *State* und *Action*.

3.2.1 State

State ist eine Funktion, die ein Objekt, das den Zustand beschreibt zurückgibt.

3.2.2 Action

Eine Action ist eine Methode, die den State verändert und in einem *actions* Objekt definiert wird.

3.2.3 Definition eines Stores

Zu Definition eines Stores wird die *defineStore* API genutzt. Als Parameter wird ein eindeutiger Name und eine Beschreibung des Stores in Form eines Objekts übergeben. In dem zweiten Paramter werden die Felder *state* und *actions* definiert.

```
const userStore = defineStore('user-store', {
  state: () => {
    user: null
  },
  actions: {
    updateUser(newUser) {
      this.user = newUser
    }
  }
})
```

Auf die Felder in dem State-Objekt wird in einer Action mit *this* zugegriffen. Das State-Objekt wird seitens Pinia intern jeder Action gebunden.

3.2.4 Interaktion mit dem Store

Literaturverzeichnis

[1] *Defining a Store*. – URL https://pinia.vuejs.org/core-concepts/. – official documentation

[2] (BRANDONROBERTS), Brandon R.: *Getting Started*. 2024. – URL https://ngrx.io/guide/store. – official documentation

[3] (JOSH-CENA), Joshua C.: *Spread syntax (...)*. 2024. – URL https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax. – official documentation

[4] MARK ERIKSON (MARKERIKSON), Eng Zer Jun (.: *A (Brief) History of Redux*. 2023. – URL https://redux.js.org/understanding/history-and-design/history-of-redux. – official documentation

[5] (MARKERIKSON), Mark E.: *Redux Fundamentals, Part 3: State, Actions, and Reducers*. 2024. – URL https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers. – official documentation

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original