Store-Technologien unter Angular

Eine Übersicht über Best-Practises und Auswahl der besten Möglichkeiten

Im folgenden Text schauen wir, wie eine Angular-App am besten um einen Store erweitert werden kann, im dem wir uns verfügbare Module anschauen und deren Einsatz etwas unter die Lupe nehmen.

Es kommen drei Store-Module in Frage, wobei es sicherlich noch viele weitere gute Implementierungen gibt, aber unter dem Gesichtspunkt der Dokumentation und der gesicherten Weiterentwicklung idealerweise über Jahre reduziert sich die Auswahl auf:

NgRx (https://ngrx.io/)

NGXS (https://www.ngxs.io/)

Elf (https://ngneat.github.io/elf/)

Zu NgRx lässt sich sagen, dass ein Einsatz mit Angular am sinnvollsten ist, wenn die gesamte Anwendung vollständig an den Store angebunden ist, im Enddefekt spielt das System seine Stärken aus, wenn auch die Backend-Requests in den Store eingebunden sind und alle Actions mit Enum-Werten definiert sind. Eine bestehende App lässt sich nur schwerlich um NgRx erweitern und sorgt für eine gewisse Anzahl an Boilerplate-Code.

NGXS nimmt sich zwar NgRx als Vorbild, unterscheidet sich neben seiner Einfachheit aber grundlegend. Reducer können direkt benutzt werden, um zum Beispiel auch direkt auf Requests zu reagieren und Daten zu aktualisieren. Zudem gibt es diverse Erweiterungen zu Themen wie Storage und Routing:

```
@Action(GetTodos)
getTodos(ctx: StateContext<Todo[]>) {
  return this.todoService.getTodos().pipe(
   tap(todo => {
   ctx.setState(todo);
  }),
catchError(() => {
  return ctx.dispatch(new HandleApiError(new GetTodos()));
  })
);
}
```

Das System umgeht die Definition von Actions und in der Folge auch von Reducern und greift direkt über seine Annotationen auf die Daten und Definitionen zu.

Die durchgängige Verwendung von Annotation wie für Actions ist Geschmackssache, ist aber positiv zu sehen, da es die Implementierung durchaus verbessert und einen wartbaren und guten Code erzeugt. State-Klassen werden mit Hilfe von statischen Methoden benutzt.

Elf ist eigebettet in die Store-Welt von Netanel Basal und der Community. Es unterschiedet sich von den klassischen Umgebungen darin, dass es eine saubere Repository-Pattern-Implementierung ermöglicht und die jeweiligen Store-Umgebungen und Methoden wie eine Datenbank-Tabelle betrachtet. Dabei folgt der Nutzer der SQL-Logik im Sinne von Ursprungs- und erweiterten Tabellen über die jeweiligen Fremdschlüssel. So erwartet das Interface, welches die Store-Daten definiert, einen eindeutigen Schlüssel, im Standard wird dieser mit ‚id' bezeichnet, es kann aber auch eine andere Bezeichnung konfiguriert werden. Wenn eine entsprechende Business-Logik implementiert wird, die neue Datensätze erzeugt und ablegt, werden diese Werte komfortabel über den Store verwaltet, da dafür starke Store-Methoden zur Verfügung stehen. So steht zum Beispiel die Methode upsertEntities (https://ngneat.github.io/elf/docs/features/entities-management/entities#upsertentities) zur Verfügung, die einen vorhandenen „Datensatz" aktualisiert oder bei einer Nicht-Existenz neu anlegt:

```
import { upsertEntitiesBy } from '@ngneat/elf-entities';
todosStore.update(upsertEntities({ id: '1', happy: true }));

todosStore.update(
  upsertEntities([
    { id: '1', happy: true },
    { id: '2', name: 'elf 2', happy: false },
  ])
);
```

Es existieren diverse Zusatzmodule, die einen möglichen vorhandenen Basis-Store um Möglichkeiten wie persistente Daten oder UI-spezifische Entitäten erweitern.

Die Produktion des Store-Codes folgt einem intuitiven und sauberen Weg. Zunächst sollten konzeptionelle Strukturen definiert werden. Dadurch entstehende Sub-Domains im Sinne der Domain-driven Design-Modellierung führen zu entsprechenden Interfaces. Je Store wird anschließend ein Repository geschrieben, dass einen Store erzeugt und die Store-Methoden involviert. Unter Elf lassen sich etwas bessere Tests schreiben, weil durch die direktere Verwendung des Stores ohne Action-Klassen weniger Mockdaten geschrieben werden müssen.

Für alle hier aufgeführten und weitere Store-Umgebungen finden sich unter https://github.com/ihrwebentwickler entsprechende Beispiel-Implementierungen einer Shop-Basket-Umgebung.

Fazit:

Elf hat sich in Projekten bewährt und unterstützt die Herstellung zeitnaher und leistungsfähiger Store-Umgebungen. Der Einsatz von Elf macht bereits Sinn in kleineren Anwendungen, wenn ein gewisser Anteil an GUI-Formularen vorliegt. NGXS ist etwas weiterverbreitet.