УДК 004.4'22

В. С. Збитківський[1]
І. В. Богач[1]

ОГЛЯД ФІЛЬТРАЦІЇ КОНТЕНТУ СТРИМІНГОВОГО ВІДЕО СЕРВІСУ

[1] Вінницький національний технічний університет

Анотація
Зроблено огляд способів та інструментів фільтрування для обробки медіа-контенту, такого як відео, субтитри та пов'язані метадані.
Ключові слова: TypeScript, веб-додатки, Angular, NodeJS, NgRx, програмування веб-додатків.

Abstract
An overview of ways and tool to filter and process multimedia content such as video, subtitles, and related metadata.
Keywords: TypeScript, web-applications, Angular, NodeJS, NgRx, web-applications programming.

Вступ

Перед відображенням відео контенту на YouTube для користувача, він має пройти велику кількість фільтрів, що забезпечує найактуальніший та найбезпечніший контент для різних користувачів з різних країн та вікових обмежень.

Зокрема фільтрування відбувається в залежності від вибраної дати народження під час реєстрації користувача сервісу, в залежності від найпопулярніших відео в регіоні користувача та історії переглянутих відео.

Коли справа доходить до фільтрування, NodeJS та NgRx є одними з найбільш широко використовуваних методів для реалізації фільтрації на фреймворку Angular. TypeScript зазвичай використовується як вбудована мова для програмної реалізації. В роботі розглянуто найбільш популярні та практичні методи реалізації фільтрації.

Результати дослідження

NgRx бере ідею Redux та реалізує її з урахуванням специфіки Angular. В цілому Redux не прив'язана до жодного фреймворку, але популярна в основному в react-ком'юніті.

Підхід Redux та NgRx полягає в наступному. Представимо весь стан програми у вигляді одного об'єкта. Цей об'єкт помістимо у сховище – store. Зі сховища ми можемо отримати цей стан, проте безпосередньо змінити його не можемо. Щоб змінити стан, ми повинні відправити в сховище "сигнал" - дію, яка визначається типом (зазвичай це просто рядок) та даними. "Зміни" стану відбуваються шляхом виклику чистих функцій - редьюсерів, які повертають новий стан.

Щоб отримувати дані через API або виконувати якісь побічні дії (наприклад, логування), передбачено механізм Middleware (в термінології NgRx - Effect).

@ngrx/store - це основний модуль: тут містяться засоби управління станом. Центральний об'єкт всього проекту – сервіс Store. Отримання даних та диспетчеризація дій відбувається через нього. Сам сервіс успадковується від Observable, так що працювати зі станом зручно та звично – можна використовувати всі оператори з RxJS та async-пайп. У самому сервісі є весь стан, який ми зареєструємо. Щоб працювати тільки з частиною стану є спеціальний оператор select. Залежно від переданих аргументів це аналог оператора map чи оператора pluck.

Для зміни стану нам спочатку потрібно описати дії та те, як вони змінюють стан. Потім достатньо викликати метод dispatch у сервісі store та передати туди дію.

@ngrx/effects у найпростішому варіанті весь стан може змінюватися тільки синхронно: користувач змінює дані або натискає на щось, і тоді почнеться пов'язана дія. Сховище диспатчеризує події на кшталт, ці дані обробляються в редьюсері, й у результаті повертається новий стан. Воно стає дос-

тупним усім, хто підписаний на відповідний шматок стора в додатку. Основний модуль @ngrx/store дозволяє обробляти лише чистий стан.

Але іноді при виконанні дії потрібно зробити ще щось: запитати дані через API, записати в балку, вивести нотифікацію. У Redux для цього є Middleware, NgRx використовує аналог – ефекти. Кожен ефект – поле у спеціальному сервісі з анотацією @ Effect.

Поля, позначені інструкцією Effect, власне, описують ефект. Ці поля мають бути типу Observable<Action>. Зазвичай вони так виглядають. З сервісу Actions фільтруються необхідні дії, далі запускається якесь завдання: логування, запит до API і т. д. Після завершення завдання потік повертається якесь інше дію.

Селектори - це чисті функції, що використовуються отримання фрагментів стану сховища. Як показано нижче, можна запросити стан навіть без використання селекторів. Але цей підхід знову ж таки має кілька серйозних недоліків.

Ви можете використовувати функції відображення для отримання зрізів стану та виконання будь-яких обчислень, якщо це необхідно.Отримуємо user у дереві об'єктів стану і перетворюємо його на логічне значення, щоб визначити, увійшов користувач у систему чи ні.

Ви можете або вручну підписатися на селектор спостерігається об'єкт, або використовувати його в шаблоні Angular з асинхронним конвеєром для читання значень, що випускаються.

Проте, якщо результат функції зіставлення не змінився з минулого разу, немає необхідності повторно оновлювати інтерфейс користувача. Наприклад, якщо результат map(state => !!state.user)не змінився з моменту останнього виконання, нам не потрібно знову надсилати результат у інтерфейс / передплатника користувача. Для цього NgRx (не RxJS) ввів спеціальний оператор, який називається select. З select оператором наведений вище код зміниться в такий спосіб.

Цей підхід можна покращити. Навіть якщо select оператор не відправляє незмінені значення інтерфейсу користувача / передплатникам, він все одно повинен приймати об'єкт стану і щоразу проводити обчислення для отримання результату.

Як вже пояснювалося вище, об'єкт state генеруватиме об'єкт , коли сховище отримає дію від програми. Дія який завжди оновлює стан. Якщо стан не змінився, результат обчислення функції відображення також не зміниться. Отже, нам не потрібно знову виконувати обчислення, якщо згенерований станоб'єкт не змінився з останнього разу. Тут у гру вступають селектори.

Селектор – це чиста функція, яка зберігає пам'ять про попередні виконання. Поки вхідні дані не змінилися, вихідні дані не будуть перераховані. Натомість висновок буде повернено з пам'яті. Цей процес називається мемоїзацією.

NgRx надає службову функцію, що викликається createSelector для створення селекторів з можливістю запам'ятовування. Нижче наведено приклад створення колекції функції корисності.

Ця createSelector функція приймає функції відображення "один до багатьох", які дають різні зрізи стану, і функцію проектора, яка виконує обчислення. Функція проектора не буде викликана, якщо зрізи стану не змінилися після останнього виконання. Щоб використати створену функцію селектора, ви повинні передати її як аргумент select оператору.

Висновки

Вибір правильного методи реалізації фільтрації контенту багато в чому залежить від вимог, що висуваються до програмного забезпечення. Фреймворк, технології та методи що були розглянуті в даній роботі, є найбільш популярними і призначені для розв’язання різного типу задач, тому їх вибір напряму залежіть від поставленої задачі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Angular, AngularIO [Електронний ресурс]. [Веб-сайт].– 2021. – Режим доступу до ресурсу: https://angular.io/docs.
2. NodeJS, Wikipedia [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://uk.wikipedia.org/wiki/Node.js.
3. NgRx, NgRxIO [Електронний ресурс]. [Веб-сайт]. – 2021. – Режим доступу до ресурсу: https://ngrx.io/docs.
4. Felter data using NgRx [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ngrx.io/guide/store/selectors.

Збитківський Владислав Сергійович *– студент групи 2КН-18Б, кафедра комп'ютерних наук, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м.Вінниця, e-mail: vladz15@ukr.net*

Богач Ілона Віталіївна *– к.т.н., доцент кафедри автоматизації та інтелектуальних інформаційних технологій, факультет комп'ютерних систем і автоматики, Вінницький національний технічний університет, м.Вінниця, e-mail: ilona.bogach@gmail.com*

Zbytkivskyi Vladislav Sergiyvich *– student of 2CS-18b group, Department of Computer Science, Faculty of Information Technology and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: vladz15@ukr.net*

Bogach Ilona Vitaliivna *– Associate Professor of Automation and Intelligent Information Technologies, Faculty of Computer Systems and Automatics Vinnytsia National Technical University, Vinnytsia, e-mail: ilona.bogach@gmail.com*