Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe L-31)

Il flusso dei dati nel front-end di una applicazione web complessa

Laureando
Diego Pasquali
Matricola 093341

Relatore
Prof. Luca Tesei

A.A. 2016/2017

Indice

1. Introduzione

Il *Flusso dei dati* nel front-end di una applicazione web rappresenta tutti gli input e gli eventi che si muovono attraverso i suoi vari livelli logici. Più questo flusso è intenso e disorganizzato più diventa complicato gestire lo stato dell'applicazione.

Un'interfaccia utente mette a disposizione dell'utilizzatore una grande quantità di interazioni, sia volontarie che involontarie, che cambiano in continuazione lo stato dell'applicazione e che devono essere opportunamente gestite e sincronizzate. La struttura del codice diventa quindi prioritaria al fine di ottenere un prodotto che sia soddisfacente a livello di prestazioni e che riesca a mantenere un adeguato livello di scalabilità.

1.1 Background

La gestione del flusso dei dati all'interno di una applicazione web è un argomento molto discusso dopo l'avvento di tecnologie front-end sempre più complesse e potenti come *React* o *Angular*. La causa di ciò è la necessità di avere un codice che sia il più possibile scalabile ed il più facilmente testabile a prescindere dal numero di features che verranno successivamente aggiunte. Codebase vaste come potrebbero essere quelle di Facebook, Twitter o YouTube necessitano di una architettura di fondo che sia altamente chiara e comprensibile per evitare confusione tra i vari servizi. Come vedremo successivamente, architetture datate come l'MVC pur essendo molto efficienti lato back-end non rendono allo stesso modo lato front-end, dove c'è una quantità maggiore di azioni che l'utente può intraprendere e che possono avere ripercussioni differenti su più componenti diversi all'interno di una view. In questo documento verrà discussa l'alternativa attualmente più gettonata che è quella dell'architettura a flusso unidirezionale, implementata in prima battuta da Flux e successivamente ottimizzata da Redux.

1.2 Lo stato dell'arte

Possiamo paragonare la creazione della prima applicazione web con la messa online del primo sito da parte di Tim Berners-Lee nel 1991 dal Cern di Ginevra [Gra12]. Stiamo tuttavia parlando di una applicazione statica costruita solamente in HTML. La svolta avvenne il 5 maggio del 1995 con l'avvento di Javascript [W3C12], il linguaggio che anche adesso è alla base di tutte le tecnologie web più nuove e potenti. Da qui in poi l'evoluzione è andata avanti in maniera esponenziale partendo da un utilizzo banale del linguaggio fino a giungere alla situazione attuale con framework ed architetture complesse.

1.2.1 Single-page web application

La problematica della gestione del flusso dei dati è nata con la comparsa delle così dette *Single-page web application* (SPA), ossia applicazioni organizzate all'interno di un'unica pagina web e gestite interamente tramite Javascript. Nel loro stadio iniziale queste non erano costruite sopra una struttura ben definita e la gestione del flusso dei dati avveniva in maniera disordinata gestendo ogni azione dell'utente in maniera diretta.

Andando avanti con il tempo e con l'aumentare della complessità di queste applicazioni, si è sentito il bisogno di creare un'architettura ben definita che aiutasse a gestire in maniera più consistente lo stato del servizio e tutti i suoi eventuali cambiamenti.

1.2.2 I primi framework

La necessità di avere struttura più solida lato front-end, specialmente per applicazioni più esigenti, ha portato alla nascita dei primi framework basati sull'architettura MVC (Model - View - Controller).

> Single page apps are distinguished by their ability to redraw any part of the UI without requiring a server roundtrip to retrieve HTML. This is achieved by separating the data from the presentation of data by having a model layer that handles data and a view layer that reads from the models. [Tak13]

Nel modello MVC per front-end (e come anche in quello back-end), che riprenderemo in dettaglio più avanti nel documento, abbiamo una netta distinzione tra i dati e la presentazione di quest'ultimi (ossia tra Model e View). Questo ci permette di avere un controllo maggiore sullo stato globale e di ogni singolo componente dell'interfaccia oltre ad un codice più robusto e facile da modificare nel tempo [Ven08].

Javascript ha a disposizione un numero considerevolmente alto di framework MVC. Uno dei più famosi è sicuramente *AngularJS* (parliamo della versione 1) mantenuto da Google e dalla vasta community formatasi intorno. L'architettura MVC lato front-end non è considerata però ottimale. Col passare del tempo si sono venute a creare delle strutture derivate da questa che tendono a sviluppare la gestione dell'interfaccia utente in maniera differente, con i propri vantaggi e svantaggi. Una di queste è MVP (Model - View - Presenter) utilizzata da *Backbone.js*. In questa il Presenter, che sostituisce il Controller, ha una responsabilità minore di quest'ultimo e si occupa solamente di passare dati alla View la quale deciderà cosa e come mostrare. Un'altra architettura derivata da MVC è MVVM (Model - View - ViewModel), utilizzata da framework come *Ember.js* e *Knockout*, in cui il ViewModel si occupa di mantenere i dati del Model (che sono grezzi) nella forma richiesta dalla View, ed espone a quest'ultima metodi e funzioni per la gestione dello stato dell'applicazione. [Cha16].

Ancora una volta tuttavia ci troviamo di fronte ad un muro, dove le architetture venutesi a creare sono tante ma tutte derivate da MVC traendone i difetti. Il problema più grande di questo paradigma, specialmente nei suoi derivati MVVM e MVP, è la comunicazione bidirezionale: la View ha la possibilità di modificare direttamente o per mezzo di un livello intermedio il Model. Tenendo presente che stiamo parlando di applicazioni complesse e quindi con un grande numero di features, questo genera un effetto cascata tra i vari componenti dell'interfaccia che sono direttamente o indirettamente collegati a quel Model causando una codebase molto difficile da gestire ed analizzare. Per la prima volta a questo punto si inizia a discutere di flusso di dati in maniera attiva

e diretta riconoscendolo come difetto principale del modello MVC e trovando in React la libreria perfetta per risolverlo [Sal15].

1.2.3 React e Flux

...

Bibliografia

[Cha16] Shailendra Chauhan. Understanding mvc, mvp and mvvm design patterns, 2016.

[Gra12] Dino Grandoni. World's first website, created by tim berners-lee in 1991, is still up and running on 21st birthday, 2012.

[Sal15] Amir Salihefendic. Flux vs. mvc (design patterns), 2015.

[Tak13] Mikito Takada. Single page apps in depth, 2013.

[Ven08] Bill Venners. The importance of model-view separation, a conversation with terence parr, 2008.

[W3C12] W3C. A short history of javascript, 2012.