Hochschule Bremen

Exposè

Media-science Bsc.

Roman Quistler

February 18, 2019

Contents

1 Introduction

2 Problem and solution

2.1 Problem

Today's applications seem to be increasing in complexity over the last few years. Especially in the field of frontend development, the range of functions has increased [1]. The web development has transitioned from server rendered, page-reloading websites to modern, so called single page applications.

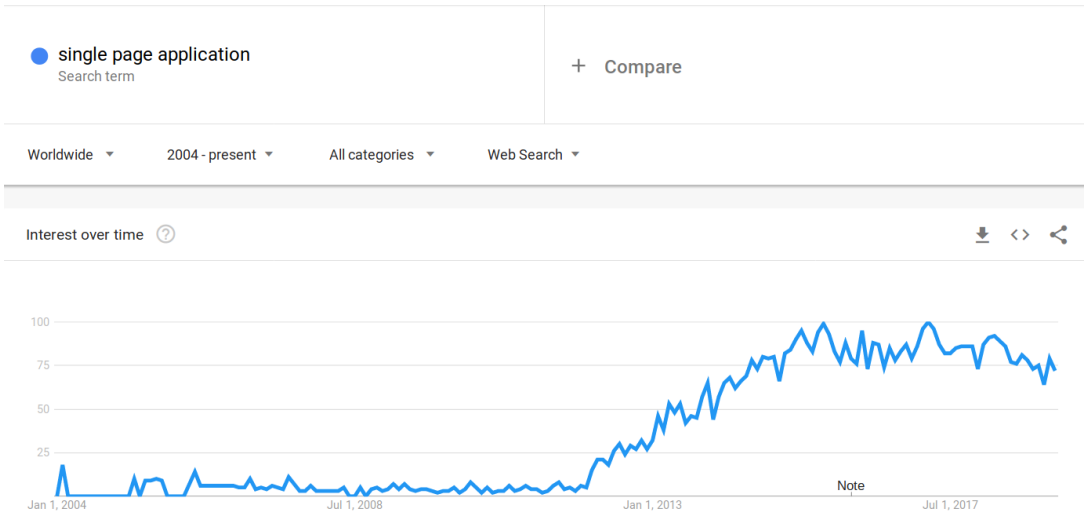

Figure 1: Google trends: single page application

In most cases, an application communicates with other interfaces, such as a back-end or local database. On mobile devices it's also often necessary to know the GPS status, whether mobile data is turned on off, and similar. A click of a user thus can cause many changes inside and outside of the application. A developer must be aware of all the possible outcomes, react to them and save what has happened. The developer must manage the state. The problem is to find a good design pattern, which makes this task feasible.

shared mutable state = bad

2.2 Solution

A solution to the problem can lie in giving the developer a small and lightweight set of tools to help managing the state, which also takes care of side effects caused by user interaction.

Talk about routing/user flow/navigation

in such a way that it respects SOLID/DRY etc

mention reactive programming

1. State
2. Events
3. S.O.L.I.D
4. Functional
5. Reactive
6. Clean code
7. DRY
8. MVI

Since user interactions are inherently asynchronous, in that a user clicks on the screen which trigger some action. That being accessing a remote or local data source, or simply toggle between dark and light themes, one could leverage the concepts of reactive programming. To address the race-condition prone shared mutable state that plagues concurrent/reactive programming we want to use immutable state -https://kotlinlang.org/docs/reference/corou mutable-state-and-concurrency.html To be able to know which events and state the developer has to handle, a finite state machine [2] fits well into this problem. By writing a 'state transition table' the developer immediately knows all possible events and states that needs to be handled. This is useful because it makes it easier to plan the architecture of the application. To take it a step futher, one can implement a deterministic state machine so that we always know what to expect . This will be solvedlby writing a sort of helper library, which rather than advocating strict software design patterns guides the user into a type of workflow that is approporiate for a reactive environment with the concepts of Events, states and transitions between those.

what

List of Figures

References

[1] Kevin Ball. *The increasing nature of frontend complexity*. Jan. 30, 2018. url: https://blog.logrocket.com/the-increasing-nature-of-frontend-complexity-b73c784c09ae (visited on 02/04/2019).

[2] Wikipedia. *Finite-state machine*. Feb. 18, 2019. url: https://en.wikipedia.org/wiki/Finite-state_machine (visited on 02/18/2019).