Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Designing and Implementing an Educational Support System

Bachelor's Thesis

Author
Nóra Szepes

Advisors
Dr. Sándor Gajdos
Bence Golda

2015

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Szepes Nóra*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2015. október 19.

Szepes Nóra
hallgató

Kivonat

Abstract

Chapter 1

Introduction

During the summer of 2015 my teacher, Sándor Gajdos contacted me to give him a review about his subject, Software Laboratory 5. I told him what I thought was good and bad in the subject, not only about the tasks, but also about the administration portal. It really bothered me that the portal didn't have e-mail notification, so I told Sándor, that I'd like to develop it into the current portal. All I knew was that it was written in php. I told him about my ideas and he contacted the creator of the old portal, Bence Golda, to ask for some information about the old portal's code and József Márton to create a "noreply" e-mail address for the notification module. József gave us an idea for creating a new portal and other team members, Bence, Gábor Szárnyas and I agreed with the idea.

In the beginning of August we had our first meeting. Before that I decided to look up all the different homework portals I've ever used during my student years. I asked for an account to Zoltán Czirkos's InfoC [6], because that website started after I've finished the subject Basics of Programming 1. After some research I made a small specification for an ideal homework portal and some ideas of how we could use one portal for more than one subject's administration.

During the meeting we talked about this, and what others expect from a new portal. It started as a department project but József asked for some ideas about what students want from a portal. I wanted to participate but I said that besides my thesis I won't have that much time to work on the portal. At the end Sándor offered me that this could be my thesis topic and he would be my advisor. Bence liked this idea and since I didn't have any thesis topic, I accepted the idea.

1.1 The Old Administration Portal

*****TODO***: ask Bence about details** Sándortól: Nem Bence irta meg, hanem Benceek. Az indittatas az volt, hogy felkinaltuk nehany gondosan kivalasztot "taltos" hallgatonak a targy teljesitesenek ezt a formajat, nekik pedig tetszett a lehetoseg/kihivas.

1.2 Purpose of the Thesis

mit tudok, mit fogok végrehajtani, mi az elérendő cél

1.3 ?

hogy álltam neki a fejlesztésnek - melyik fejezetben mit fogok bemutatni

Chapter 2

Comparing JavaScript Frameworks

For the project I wanted to choose a JavaScript framework for faster developement than using plain JavaScript with jQuery. I chose the TodoMVC [5] website to find the most popular frameworks.

I tried these frameworks to see how fast and easily can I build a basic website, how can I access a server with AJAX requests and how routing and data binding works.

In JavaScript with AJAX requests we can send requests to a server asynchronously without reloading a page. In a single-page application we want to make the browser think it is always on the same page. When the user clicks on a new link, the browser won't reload the whole page, it will just simply load the new view into the old frame. Everything happens in the background so the application won't force the user to wait while it sends data to a server. If the application is retrieving data, then when it arrives, the application can process it and show the result to the user.

There are two types of routings. Routing can be either a way to manipulate the browser's URL and the part of a web application what decides which controller will handle the requests. I was looking for a solution for URL manipulation.

The classic data binding model is when the view template and the data from the model are merged together to create the to be displayed view. Any data changes in the view won't automatically sync into the model. The developer has to write the controller what syncs the changes between the model and the view [10].

2.1 React

My first choice was React [14]. It is developed by Facebook and Instagram since 2013.

React's performance is really good because instead of always updating the browser's actual DOM it creates a virtual DOM. The virtual DOM is like a blueprint of the real DOM. Instead of containing a DIV element, the virtual DOM contains a React.div element what is just data and not a rendered content. React is able to find out what are the changes on the real DOM. It makes changes to the virtual DOM, because that is faster and then re-render the real DOM [15].

To create DOM elements you can choose between JavaScript and JSX [13]. If you use JavaScript, then the code will render the HTML code for you. If you choose JSX, then you can mix JavaScript and HTML syntax, and you can insert the desired HTML code as the return statement.

React has a one-way data flow called Flux [12]. Flux supports data flow only in a single direction, downstream. This means if something is changed in the component tree, then it will cause the element to re-render itself and all of its descendants.

React focuses only on building views. The core React version doesn't have an option for routing or AJAX requests. If I want to support those too in my application, then I should use it combined with other frameworks to have a full MVC experience.

2.2 AngularJS

AngularJS [7] is one of the most famous JavaScript frameworks nowadays. It is maintained by Google for 6 years now. It focuses mostly on dynamic views in web-applications.

Creating a website is done with an extended HTML vocabulary, like Android Layouts where you declare everything in XML. Building a website wasn't that hard but the data binding is different. It uses a two-way data binding template [10] which means whenever either the View or the Model is changed, it will update the other one.

Angular AJAX requests are similar to the AJAX methods in jQuery, but Angular takes care of setting headers and converting the data to JSON string. It can also be used in unit tests with ngMock [8], because it can create a mock server.

For routing Angular uses a special listener. It binds these listener to links. If the user clicks on a link, Angular will simply push the page to the browser's history and replace the view with the new page. This will even allow the back button to operate. This method works only if the website is loading from a server, because it allows Angular to load into the memory otherwise the listeners can't navigate through pages [9] [11].

2.3 Mithril

My last choice was Mithril [1]. It is really similar to React. With the help of MSX [4], you can use the same HTML-like syntax to create websites. MSX uses JSX, but transforms the output to be compatible with Mithril.

It provides a simple way to work with AJAX requests [2]. A basic request returns a special *m.prop* getter-setter, what is a utility class.

Mithril provides utilites to handle routing [3]. It's organized around URL routes. For a URL we can create a new module with a controller and a view. When the user clicks on the link a new instance will be made and passed to the view.

2.4 Chosen Tools

ide vagy máshova, de valahova leírni milyen toolokat fogok használni

Chapter 3

specification

Chapter 4

conceptual

Chapter 5

design

Chapter 6

implementation

Chapter 7

test

Chapter 8

deployment

Chapter 9

conclusion

Acknowledgements

List of Figures

List of Tables

Bibliography

[1] Mithril. http://mithril.js.org/.

[2] Mithril, m.request. http://mithril.js.org/mithril.request.html.

[3] Mithril, routing. http://mithril.js.org/routing.html.

[4] MSX. https://github.com/insin/msx.

[5] TodoMVC. http://todomvc.com/. Accessed: 2015-10-18.

[6] Dr. Zoltán Czirkos. Infoc. https://infoc.eet.bme.hu. Accessed: 2015-10-18.

[7] Google. AngularJS. https://www.angularjs.org/. Accessed: 2015-10-19.

[8] Google. AngularJS, API Reference, http. https://docs.angularjs.org/api/ng/service/$http. Accessed: 2015-10-19.

[9] Google. AngularJS, API Reference, location. https://docs.angularjs.org/api/ng/service/$location. Accessed: 2015-10-19.

[10] Google. AngularJS, Developer Guide, Data Binding. https://docs.angularjs.org/guide/databinding. Accessed: 2015-10-19.

[11] Google. AngularJS, Tutorial 7, Routing and Multiple Views. https://docs.angularjs.org/tutorial/step_07. Accessed: 2015-10-19.

[12] Facebook Inc. Flux website. https://facebook.github.io/flux/docs/overview.html. Accessed: 2015-10-19.

[13] Facebook Inc. JSX Specification. https://facebook.github.io/jsx/. Accessed: 2015-10-19.

[14] Facebook Inc. React. https://facebook.github.io/react/. Accessed: 2015-10-19.

[15] Facebook Inc. React, working with the browser. http://facebook.github.io/react/docs/working-with-the-browser.html. Accessed: 2015-10-19.

Appendix