



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

Kivonat	i
Abstract	ii
1 Introduction	1
1.1 The Old Administration Portal	1
1.2 Purpose of the Thesis	2
1.3 ?	2
2 Specification	3
3 Conceptional System Design	4
3.1 MVC Pattern	4
3.2 System Design	5
3.2.1 Frontend System Design	5
3.2.2 Database - ER	5
3.2.3 Server	5
3.2.4 client-server communication	5
3.2.5 client-user communication	5
4 Comparing JavaScript Frameworks	6
4.1 React	6
4.2 AngularJS	7
4.3 Mithril	7
4.4 Conclusion	8
4.5 Chosen Tools	8
5 Design	9
6 Implementation	10
7 Test	11
8 Deployment	12
9 Conclusion	13
Acknowledgements	iii
List of Figures	iv
List of Tables	v

Bibliography

vii

Appendix

viii

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 29.

Szepes Nóra
hallgató

Kivonat

honnan->hova
látszon az eredmény

Abstract

Chapter 1

Introduction

During the summer of 2015 my teacher, Sándor Gajdos contacted me to give him a feedback 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 him, 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 Golda, 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 [3], 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 the same portal for more than one subject.

During the meeting, we talked about this, and what others expect from a new portal. It started as a department project but József Marton 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 Gajdos offered me that this could be my thesis topic and he would be my advisor. Bence Golda liked this idea and *****TODO***: since I didn't have any thesis topic, I accepted the idea. Ird azt, hogy egy vonzo mernoki feladat korvonalazodott, kezdve a tervezestol a megvalositasig, es az eredmenyt raadasul meg tobb szazan elesben is fogjak hasznalni.**

1.1 The Old Administration Portal

*****TODO***: ask Bence about details**

Sándortól: Nem Bence irta meg, hanem Benceek. Az indittatás az volt, hogy felkínáltuk néhány gondosan kiválasztott "taltos" hallgatónak a tárgy teljesítésének ezt a formáját, nekik pedig tetszett a lehetőség/kihívás.

1.2 Purpose of the Thesis

mit tudok, mit fogok végrehajtani, mi az elérendő cél

1.3 ?

hogyan álltam neki a fejlesztésnek - melyik fejezetben mit fogok bemutatni

Chapter 2

Specification

hogy történik a specifikálás folyamata

mire van szükségünk

hibaágak

test suite

mi az én feladatom?

Chapter 3

Conceptional System Design

3.1 MVC Pattern

The Model-View-Controller (MVC) is a software architecture pattern for user interface implementation, where the application logic is separated from the user interface.

In object-oriented programming the Model is the objects where the data from the database is stored. The View is the presentation layer, what the user sees and interacts with. The Controller will process and respond to the user requests and invoke the changes in the Model.

The MVC pattern is memory efficient, because multiple views can share the same underlying data model. Controllers can be separated by events. This let's the developer to create a controller hierarchy, because a controller for a keyboard event is different from a controller for a mouse event. Views implement an instance of a controller, that can be changed at run-time, because we can be disabled and enabled.

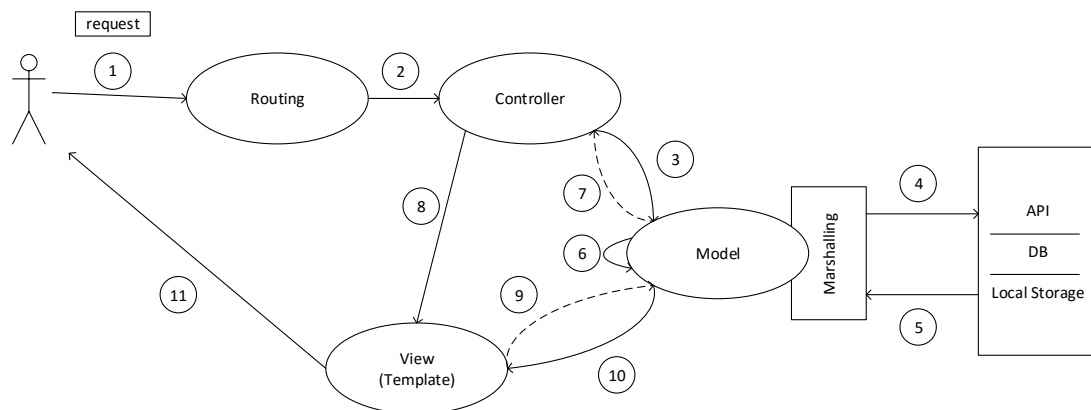


Figure 3.1. Classic MVC Webapplication
Made by Bence Golda

In webapplications the browser communicates with a controller. When the user sends a request, routing will decide which controller will handle the request. The chosen controller talks to the model to get the relevant data. If it's necessary, the model will send data to or ask for data from the database, the API or the local storage. During this process, the data has to be transformed via marshalling. Marshalling is the process, that transforms the data between storable and sendable dataformats. When the model returns the desired

data to the controller, it will forward the data to the view. The presentation layer will decide which page has to be returned to the browser, binds the data to the view template and returns it.

3.2 System Design

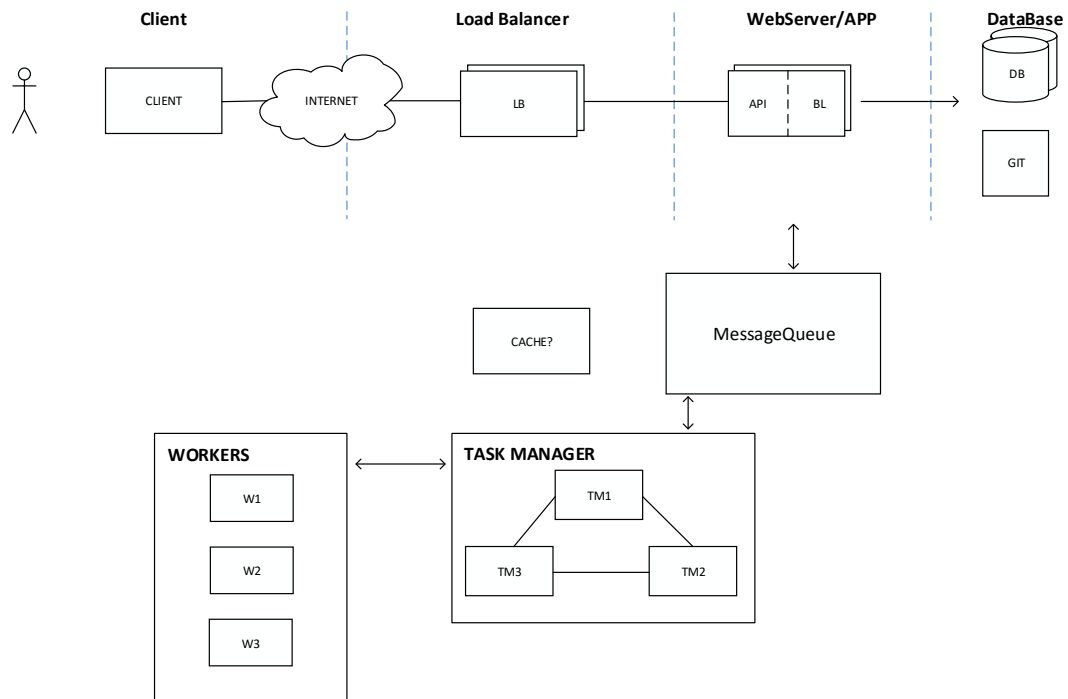


Figure 3.2. Conceptual System Design

TODO: ábramagyarázat

3.2.1 Frontend System Design

TODO: frontend ábra

3.2.2 Database - ER

3.2.3 Server

3.2.4 client-server communication

3.2.5 client-user communication

Chapter 4

Comparing JavaScript Frameworks

For the project I wanted to choose a JavaScript framework for faster development than using plain JavaScript with jQuery. I have chosen the TodoMVC [1] website to find the currently available frameworks. *****TODO***: mire jó egy framework**

I tried these frameworks to see how fast and easily can I build a basic website, how can I access the server with AJAX requests and how routing and data binding works (see figure 3.1). *****TODO***: leírni, miért ezek az összehasonlítási szempontok kellenek, amik -> must have kritériumok, rendszertervre hivatkozás**

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 or the part of a web application what decides which controller will handle the requests. I was looking for a solution for *****TODO***: URL manipulation - single page application!!! leírás.**

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 [7].

4.1 React

My first choice was React [19]. It is developed by Facebook and Instagram since 2013.

React creates a virtual DOM instead of always updating the browser's actual *****TODO***: DOM**. The virtual DOM is like a blueprint of the real DOM. Instead of containing a *****TODO***: 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 [20].

To create DOM elements, we can choose between JavaScript and JSX [18]. If we use JavaScript, then the code will render the HTML code for us. If we choose JSX, then we can mix JavaScript and HTML syntax, and we can insert the desired HTML code as the return statement.

React has a one-way data flow called Flux [17]. 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. *****TODO***: binding kifejtése**

React focuses only on building views. The core React version doesn't have an option for routing and 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.

4.2 AngularJS

AngularJS [4] is one of the most famous JavaScript frameworks nowadays. It has been maintained by Google for 6 years. It focuses mostly on dynamic views in web-applications.

Creating a website is done with an extended HTML vocabulary, like Android Layouts where we declare everything in XML. It uses a two-way data binding template [7] 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 [5], because it can create a mock server.

For routing Angular uses a special listener. It binds these listeners 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 [6] [8].

4.3 Mithril

Mithril [9] is a small MVC framework created by Leo Horie. It uses a similar virtual DOM like React, but also implements controller features like routing.

When we are creating a website, Mithril first creates virtual DOM elements, what is a JavaScript object that represents a DOM element. Rendering will create a real DOM element from the virtual one [10] [12]. If we prefer using HTML syntax, we can use MSX [2]. It uses JSX, but transforms the output to be compatible with Mithril.

Mithril has one-way data flow, from the model to the view. It has an auto-redrawing system to ensure that every part of the UI is up-to-date with the data. It uses a diff algorithm to decide which parts of the DOM needs to be updated and nothing else will be changed. Mithril automatically redraws after all controllers are initialized and will diff after an event handler is triggered. It also supports non-Mithril events to trigger auto-redrawing [14]. If we need view-to-model direction, Mithril provides us an event handler factory. This returns a method that can be bound to an event listener [16].

Mithril provides a utility for AJAX requests. We can set an early reference for the asynchronous response and queue up operations to be performed after the request completes. [15] [11].

For routing Mithril needs a key-value map of possible routes and Mithril modules. A Mithril module contains a controller and a view. A controller is a JS constructor and the view is a function what returns a virtual DOM. When the user clicks on a link, the module's controller will be called and passed as a parameter to the view. [13].*****TODO***: nem komparálható másik két rdsz-el ?**

4.4 Conclusion

Angular uses a HTML syntax what it provides worse debugging support than JavaScript syntax. Mithril's auto-redraw system with the diff algorithm and virtual DOM is faster and the redrawing starts when all controllers are done. Because of this, the user won't see incorrect state during an AJAX response. Although both Angular and Mithril provide solutions for AJAX requests and routing, I'll choose Mithril for this project. *****TODO***: legyen támadhatatlan mérnöki döntés**

4.5 Chosen Tools

ide vagy máshova, de valahova leírni milyen toolokat fogok használni

Chapter 5

Design

hogy néz ki a rendszer

arculati kézikönyv

színek kiválasztásának menete

milyen honlapokat néztem - bootstrap

elkészítés módszere

skiccek függelékbe

Chapter 6

Implementation

verziókezelés

mikre figyeltem a kódolásnál

open source projektek használata

kód metrika

teszt lefedettség

eredmény - kód hol érhető el, melyik commit verziót mutatom be, gulp

Chapter 7

Test

szerver oldali api felületek definiálása automata teszthez

automata rendszert keresni - drakov, api blueprint

teszt környezet leírása és használata

Chapter 8

Deployment

hogy érhető el az az állapot, hogy a fájlok összeállnak -> gulp

Chapter 9

Conclusion

mit értem el, további tervek

Acknowledgements

leírni, hogy mindenkit nagyon szeretek.

List of Figures

3.1	Classic MVC Webapplication Made by Bence Golda	4
3.2	Conceptional System Design	5

List of Tables

Bibliography

- [1] TodoMVC. <http://todomvc.com/>. Accessed: 2015-10-18.
- [2] Jonny Buchanan. MSX. <https://github.com/insin/msx>. Accessed: 2015-10-20.
- [3] Dr. Zoltán Czirkos. InfoC. <https://infoc.eet.bme.hu>. Accessed: 2015-10-18.
- [4] Google. AngularJS. <https://www.angularjs.org/>. Accessed: 2015-10-19.
- [5] Google. AngularJS, API Reference, http. [https://docs.angularjs.org/api/ng/service/\\$http](https://docs.angularjs.org/api/ng/service/$http). Accessed: 2015-10-19.
- [6] Google. AngularJS, API Reference, location. [https://docs.angularjs.org/api/ng/service/\\$location](https://docs.angularjs.org/api/ng/service/$location). Accessed: 2015-10-19.
- [7] Google. AngularJS, Developer Guide, Data Binding. <https://docs.angularjs.org/guide/databinding>. Accessed: 2015-10-19.
- [8] Google. AngularJS, Tutorial 7, Routing and Multiple Views. https://docs.angularjs.org/tutorial/step_07. Accessed: 2015-10-19.
- [9] Leo Horie. Mithril. <http://mithril.js.org/>. Accessed: 2015-10-20.
- [10] Leo Horie. Mithril, m. <http://mithril.js.org/mithril.html>. Accessed: 2015-10-20.
- [11] Leo Horie. Mithril, m.request. <http://mithril.js.org/mithril.request.html>. Accessed: 2015-10-20.
- [12] Leo Horie. Mithril, Render. <https://lhorie.github.io/mithril/mithril.render.html>. Accessed: 2015-10-20.
- [13] Leo Horie. Mithril, Routing. <http://mithril.js.org/routing.html>. Accessed: 2015-10-20.
- [14] Leo Horie. Mithril, The Auto-Redrawing System. <http://mithril.js.org/auto-redrawing.html>. Accessed: 2015-10-20.
- [15] Leo Horie. Mithril, Web Services. <http://mithril.js.org/web-services.html>. Accessed: 2015-10-20.
- [16] Leo Horie. Mithril, withAttr. <http://mithril.js.org/mithril.withAttr.html>. Accessed: 2015-10-20.
- [17] Facebook Inc. Flux website. <https://facebook.github.io/flux/docs/overview.html>. Accessed: 2015-10-19.

- [18] Facebook Inc. JSX Specification. <https://facebook.github.io/jsx/>. Accessed: 2015-10-19.
- [19] Facebook Inc. React. <https://facebook.github.io/react/>. Accessed: 2015-10-19.
- [20] Facebook Inc. React, Working With the Browser. <http://facebook.github.io/react/docs/working-with-the-browser.html>. Accessed: 2015-10-19.

Appendix

ide jönnek majd:

- design tervek
- user storyk és scenariok