MŰEGYETEM 1782

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

Table of Contents

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 that feature into the current portal. All I knew 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 Marton 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. I accepted his idea because this is an appealing engineering task from designing to implementing and in addition the final portal will be used by hundreds of students. *****TODO***: task helyett challenge? vagy assignment???**

1.1 The Old Administration Portal

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

*****TODO***: Sándortól: Nem Bence irta meg, hanem Benceek. Az indittatas az volt, hogy felkinaltuk nehany gondosan kivalasztott "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

Specification

Before the team's first meeting in August, I looked up all the homework portals I've used during my student years to write a small specification [35] about what I expect from a homework portal as a student. Since September we have weekly meetings where we discussed the followings:

- the old portal's features we don't wish to keep
- the old portal's features we want to keep with modification
- the old portal's features we want to keep without modification
- new features we want to add

After a few meetings we had a list of features as the starting point for our specification. Then I used the 5 Whys technique to filter out some features we don't need.

2.1 Test Suite

2.1.1 5 Whys

The 5 Whys is a technique to find the root cause of a problem simply by asking the question "Why?". With this technique we can find out what does the user want to achieve with using a feature and why does he want to achieve it?

Example:

The student wants to list his commits from every branch in the laboratory's repository.

- **Why?** – Because the student wants to tag one commit as final version.
- **Why?** – Because the evaluator will know which commit contains his homework.
- **Why?** – Because the evaluator has to correct his homework.
- **Why?** – Because the evaluator has to give the student a grade.
- **Why?** – Because the student has to pass the laboratory to pass the course.

When I finished with the filtering, I had the final list of features (see section 2.2) and I started to write user stories to have the basics for test automation.

2.1.2 User Story

*****TODO***: miau** cucumber, given-when-then, scenario, background

Example:

```
Background:
        Given a logged in student named "Jakab"
        and the settings page is loaded
Scenario: Setting a new SSH public key
        Given I am logged in as "Jakab"
        And I have entered a new SSH public key
        When I press the save button
        Then I should see "Your settings have been saved."
```

The user stories are in appendix B.

2.1.3 API Blueprint

*****TODO***: miau2** drakov, api blueprint

2.2 The Final Specification

2.2.1 User Roles

In the new portal there will be three different user roles: student, teacher and administrator.

Student Role

The student can:

- see general informations about his classes
  - when will it be
  - where will it be
  - who will be the teacher
  - when is the deadline
  - how much time is left until the deadline
  - what is the Git remote URL
    - The portal doesn't have an option to upload files, because students use Git repositories to upload their homeworks.
    - With every laboratory they get a new Git repository in the Database Laboratory GitLab.
- check his results
  - his entry test grade
  - his laboratory report grade

    - his laboratory report review
    - who was the evaluator
    - his laboratory grade
    - his laboratory review
- list his commits from every branch in the laboratory's repository
- tag a commit as a final version
    - When the deadline is over, the back end will tag every branch's last commit.
    - If the student didn't tag any commit as final version, the evaluator will check the solution only the master branch's commit, what was tagged by the back end.
- see a summarized reviews of his grades

Teacher Role

The teacher can be a demonstrator and/or an evaluator (see appendix A). If a teacher is both a demonstrator and an evaluator, then the teacher can choose which role's pages the teacher wants to use now.

Demonstrator Role

The demonstrator can:

- see his current class with a list of students, who attended that class
    - This list also contains the students' laboratory report grades and reviews.
- give the students entry test grades
- give the students laboratory grades
- give the students laboratory reviews
- choose another class to list the students, who attended that class
    - save the student's results as a draft
    - continue writing a saved review draft
    - publish the results for the student

Evaluator Role

The evaluator can:

- see maximum 4 lists of homeworks
    - If a list doesn't have any element, it won't appear on the page.

  - The first list contains the homeworks, what are waiting for evaluation and booked by the evaluator.
  - The second list contains the homeworks, what are waiting for evaluation and not booked yet.
  - The third list contains homeworks booked by another evaluator.
  - The fourth list contains the evaluated homeworks.

- book homeworks for himself for evaluation
- choose a homework to start the evaluation
- see the student's name and the Git remote URL as a general information for a homework
- give a laboratory report grade for a homework
- write a review for a homework
  - save the review as a draft
  - continue writing a saved review draft
  - publish the grade and the review for the student
  - If the student didn't tag any commit as final version, the evaluator will check the solution only in the last commit in the master branch.

Administrator Role

The administrator role is a teacher role expanded with some extra features.

The administrator can:

- run SQL queries
- search for users
  - with name
  - with username
  - with id
- can change a user's e-mail
- can change a user's password
- can impersonate any user
- modify an evaluator's homework types
- add a new entry test question
- modify an entry test question
- delete an entry test question
- add a new event
- modify an existing event

- delete an existing event
- see the statistics
  - list the unpublished reviews
  - list the homeworks, what are waiting for evaluation

Default Options

Any type of user can:

- upload a new SSH public key
- set a new e-mail address
- set a new password
- change his mailing list subscription
- change his e-mail notifications subscription

2.3 Meh, kene ide egy jo alcim

hibaágak
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.

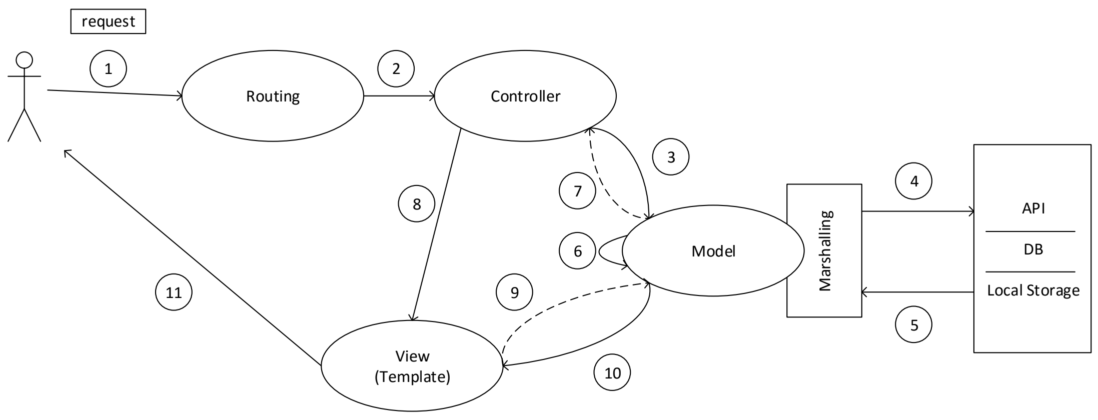


Figure 3.1. Classic MVC Web Application[1]

In web applications 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

[1]Made by Bence Golda.

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

*****TODO***: bevezetőszöveg?**

*****TODO***: ábrán kijavítani, hogy a web server az külön szó**

The main components are the followings:

- **Client:** A web portal, that is the communication bridge between the user and the web server. There will be three different modules: student, teacher and administrator. The different client modules can only communicate with the web server, and they cannot communicate with each other.
- **Database:** A database to store the system's data (see subsection 3.2.2).
- **Git:** A database to store the students' homeworks. Every student will get a different git repository for each laboratory.
- **Load Balancer:** It prevents the client from contacting the web server directly and solves the scalability problem. The client sends its requests to the load balancer, that will forward it to one of the web servers, depending on the client module, request type and the web servers' load.
- **Object-relational mapping:** It converts the data between the representation suitable for the implementation and the database.
- **Message Bus:** A component, what supports messaging between the different components. The web server, the task manager, the message queue and the workers will use this to send tasks to each other.
- **Message Queue:** A component, what will forward the incoming tasks from the task manager to the right worker for processing. *****TODO***: erről beszéljünk, hogy az MB miért nem elég nekünk**
- **Task Manager:** A special worker. It gets tasks from the web server to decide which worker has to process it. After the decision it forwards the task through the message bus.
- **Web Server:** The server that runs the API's implementation. This component processes the incoming requests, creates tasks and forwards them to the task manager. It also provides its clients the data from the databases. The API is written in Ruby on Rails.
- **Worker:** This will process the task, e.g., changes the user's mailing list subscription.

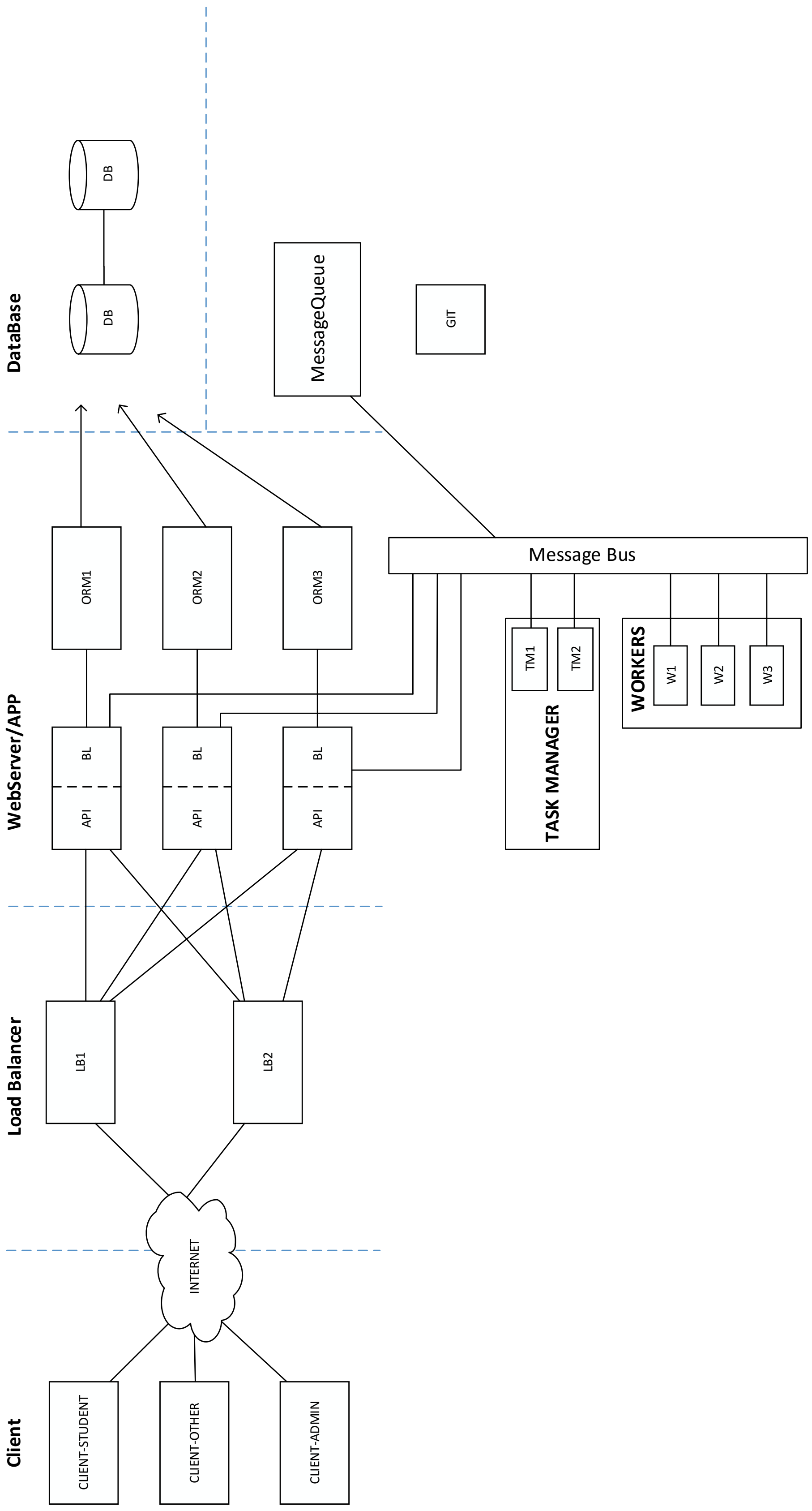


Figure 3.2. Conceptional System Design
Made by the team.

Scalability

Client

To solve the scalability problem, the client's code will run in a web browser for every user. This way, resources for the client's code are provided by the user as he opens the web portal in a web browser.

Web Server

If there were not any load balancer between the client and the web server, then one server would get every request. With thousands of users this could lead to overload and high the response time. With a load balancer, the requests will first arrive at the load balancer, and it will decide which web server will handle that request and forwards it to that web server. The choice depends on the client module, request type and the web servers' load. The load balancer's purpose is to avoid overload and minimize the response time.

3.2.1 Frontend System Design

*****TODO***: frontend ábra visioban**

3.2.2 Entity–relationship model

*****TODO***: magyarázat**

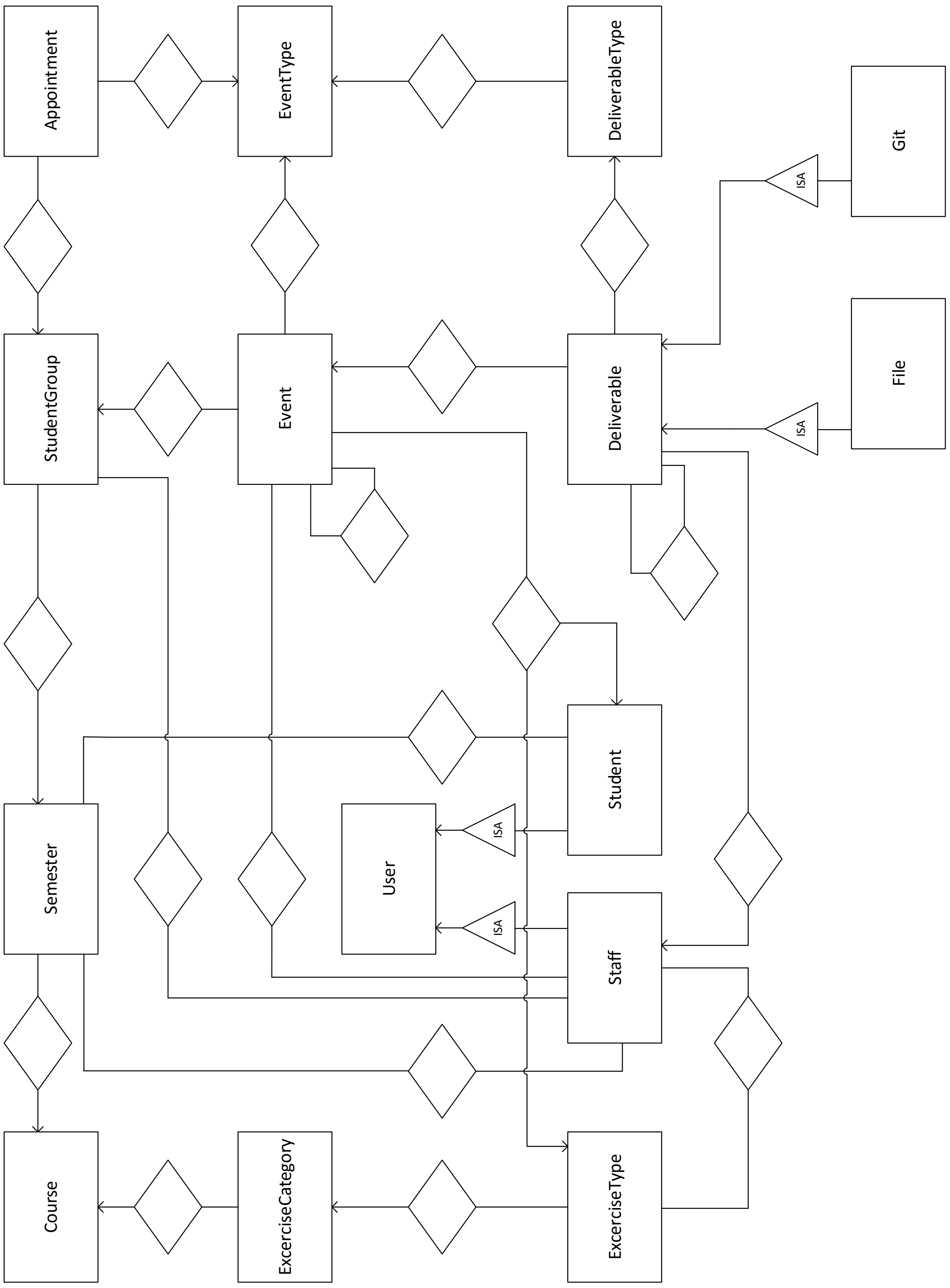


Figure 3.3. Entity–relationship model
Finalized by the team based on my proposal.

Chapter 4

Comparing JavaScript Frameworks

For the project I wanted to choose a JavaScript framework for faster development than using plain JavaScript with jQuery. Technically I don't have to use a framework, because I can write the logic myself too. But writing every component from scratch needs a lot of time. Hundreds of developers contribute in open-source frameworks, and I can use their reviewed and tested work. All I have to do is to learn the framework's usage and I get more tasks done in less time. I have chosen the TodoMVC [1] website to find the currently available frameworks.

To compare these frameworks, I checked the followings:

- **AJAX requests:** the client and the server (see figure 3.2) will communicate via REST using JSON format.
- **data binding:** to connect the data from the model to the view (see figure 3.1).
- **routing:** to build a single-page application.

In a single-page application (SPA) when the user opens the web site in a browser, all the resources will be downloaded with a single page load. From that point when the user interacts with the web site, it will dynamically update previously downloaded single page [25].

In JavaScript with AJAX requests we can send requests to a server asynchronously without reloading a page. In a SPA 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.

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 [11].

Upon URL change an SPA won't download a new page. It will navigate to the right part of the application. Routing takes care of this automatically. The SPA needs a routing table to know which URL belongs to which controller or view.

4.1 React

My first choice was React [23]. It is developed by Facebook and Instagram since 2013.

React creates a virtual DOM instead of always updating the browser's actual document object model (DOM) [37]. The DOM provides a structured representation of HTML and XML documents. The objects are the nodes of the tree, and the tree structure is the DOM tree. In my case the DOM connects the HTML page to the JavaScript code. The virtual DOM is like a blueprint of the real DOM. Instead of containing a div[1] 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 [24].

To create DOM elements, we can choose between JavaScript and JSX [22]. 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-like syntax, and we can insert the desired HTML code as the return statement.

For data binding React has a one-way data flow called Flux [21]. 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 and AJAX requests. If I want to support those too in my application, then I should use it combined with other frameworks or libraries to have a full MVC experience.

4.2 AngularJS

AngularJS [8] 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 [11] 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 [9], 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 [10] [12].

4.3 Mithril

Mithril [13] 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

[1]A div is a generic container in HTML. It helps structuring the HTML document [36].

element from the virtual one [14] [16]. 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 [18]. 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 [20].

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. [19] [15].

For routing Mithril needs a key-value map of possible routes and Mithril modules to connect the routes to the modules. Upon routing the module's controller will be called and passed as a parameter to the view. [17].

4.4 Comparison

During the comparison I made a small program to try AJAX requests, data binding and routing in the chosen frameworks. The source codes are publicly available on Github with a gulp file (see chapter 8) and an API blueprint test file for Drakov (see subsection 2.1.3).

React [34]
The components are separated into different JavaScript files. To represent the views, I used JSX. Before the concatenation I included a JSX transformer in the gulp file, that converts the inline HTML-like JSX to JavaScript code. I used the React components' state as a model. Because React hasn't got an option for AJAX requests, I used jQuery to download some mock data from the mock server. I used the React Router library [31] for routing.

Angular [32]
I separated the controllers into different JavaScript files and the views into different HTML templates. The routing connects the right HTML template to the right controller. In the gulp file I can simply concatenate the JavaScript files, but I had to keep the HTML templates in different HTML files. As a model I used variables inside the controllers. The routing is not included in the core Angular framework, but Google provides a library for it.

Mithril [33]
A Mithril module contains a model, a controller and a view. A controller is a JS constructor and the view is a function, that returns a virtual DOM. I separated them into different JavaScript files and I concatenate them with gulp. I use a MSX transformer to convert the HTML-like MSX to JavaScript code.

4.5 Conclusion

In performance both React and Mithril are faster than Angular, because they use virtual DOM. Angular's public API interface is much bigger than React's or Mithril's. React doesn't have routing and AJAX requests in its core library, so it depends on other libraries. I choose Mithril for this project, because it's self-contained, so it doesn't have to depend on other libraries. It has utilities built-in for routing and AJAX requests. And it has a small API with great documentation.

Chapter 5

Design

5.1 Design Sketches

To be able to draw design sketches, we need to know when will a user login to the Educational Support System and what kind of informations is he looking for. As a student, there are 5 possible scenarios:

1. Before a laboratory
   - To get informations about the laboratory
     - When will it be
     - Where will it be
     - Who will be the teacher
   - To read general informations about the course
2. During a laboratory
   - To upload an SSH public key
   - To get his Git remote URL
3. After a laboratory, before deadline
   - To see the date of the deadline
   - To see how much time is left until the deadline
   - To see the pushed branches, commits and tags
   - To tag a commit as final version
4. After a laboratory, after deadline
   - To check his grades
   - To check his reviews
   - To check the evaluator's name
5. Other scenarios
   - To set a new e-mail address
   - To change his mailing list subscription

- To change his e-mail notification subscription
- To see a summarized table of his grades

With the scenarios and list of actions, we can see how many pages is needed for the student modules and how many states will one page have. I drew sketches (see appendix C) for every state with placeholder data. Because the user is looking for a specific set of information, the page will only contain what he is looking for, and the previous, but still relevant informations will be accessible via submenus. Other pages, e.g., other laboratories, settings and summary, will be accessible via a menu.

5.2 Design template

To show only a specific set of information I have decided to use a minimalist design. A minimalist design is a clear design, focusing on typography, space, color and basic design elements. This way the portal will show as much information as the user needs with as few elements as possible.

To look for templates and ideas I read the Designmodo blog [4] and checked all the popular websites, e.g., Facebook, Github, Twitter and Medium. Designmodo also have purchasable website builders, like Slides [5], but I prefer the simple design of the Bootstrap elements [30].

Bootstrap is a free and open source HTML, CSS and JS framework to create responsive design. It was originally a part of Twitter as Twitter Blueprint, but in 2011 it was released as an open source project. Bootstrap contains elements for responsive web design and mobile design too.

*****TODO***: ez még így nem végleges szerintem**

5.2.1 Colors

After deciding what kind of design framework will I use, I had to chose the colors of the portal. Both the Budapest University of Technology and Economics [26] [27] and the Faculty of Electrical Engineering and Informatics [28] [29] have their own Visual Identity Guidelines.

A visual identity guideline contains the description of which color is the official color of the institution and in what kind of text which fonts and why that font should be used. Because the Software Laboratory 5 course belongs to the Faculty, we will follow that guideline and use blue (#052A4B) as the main color of the portal. *****TODO***: The following fonts will be used: Helvetica /Arial/ as body text and AvantGarde as lead text. - akarjuk mi ezeket a betűtípusokat használni?**

5.3 Elkészítés módszere

html, js, css, bootstrap css

*****TODO***: elkészítés módszere**

5.4 Rendszer kinézete

screenshotok

*****TODO***: hogy néz ki a rendszer**

Chapter 6

Implementation

6.1 verziókezelés

6.2 mire figyeltem a kódolásnál

6.3 tooles, open source projektek

6.4 kód metrika

6.5 teszt lefedettség

6.6 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

9.1 mit értem el

9.2 további tervek

Acknowledgements

leírni, hogy mindenkit nagyon szeretek.

List of Figures

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] Designmodo. Slides Framework: Beautiful Website Builder - Designmodo. http://designmodo.com/. Accessed: 2015-10-30.

[5] Designmodo. Slides Framework: Beautiful Website Builder - Designmodo. http://designmodo.com/slides/?u=2134#mobile. Accessed: 2015-10-30.

[6] Dictionary.com. Dictionary.com | Find the Meanings and Definitions of Words at Dictionary.com. http://dictionary.reference.com/. Accessed: 2015-11-18.

[7] Dictionary.com. Thesaurus.com | Find Synonyms and Antonyms of Words at Thesaurus.com. http://www.thesaurus.com/. Accessed: 2015-11-18.

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

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

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

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

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

[13] Leo Horie. Mithril. http://mithril.js.org/. Accessed: 2015-10-20.

[14] Leo Horie. Mithril, m. http://mithril.js.org/mithril.html. Accessed: 2015-10-20.

[15] Leo Horie. Mithril, m.request. http://mithril.js.org/mithril.request.html. Accessed: 2015-10-20.

[16] Leo Horie. Mithril, Render. https://lhorie.github.io/mithril/mithril.render.html. Accessed: 2015-10-20.

[17] Leo Horie. Mithril, Routing. http://mithril.js.org/routing.html. Accessed: 2015-10-20.

[18] Leo Horie. Mithril, The Auto-Redrawing System. http://mithril.js.org/auto-redrawing.html. Accessed: 2015-10-20.

[19] Leo Horie. Mithril, Web Services. http://mithril.js.org/web-services.html. Accessed: 2015-10-20.

[20] Leo Horie. Mithril, withAttr. http://mithril.js.org/mithril.withAttr.html. Accessed: 2015-10-20.

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

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

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

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

[25] Microsoft. ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. https://msdn.microsoft.com/en-us/magazine/dn463786.aspx. Accessed: 2015-10-19.

[26] Budapest University of Technology and Economics (BME). BME Visual Identity Elements. http://www.bme.hu/mediakit-arculati-elemek?language=hu. Accessed: 2015-10-19, Language: Hungarian.

[27] Budapest University of Technology and Economics (BME). BME Visual Identity Guidebook. http://intranet.bme.hu/arculat/. Accessed: 2015-10-19, Language: Hungarian, Accessible only via BME Intranet.

[28] Budapest University of Technology, Faculty of Electrical Engineering Economics (BME), and Informatics (VIK). BME VIK Visual Identity Elements. https://www.vik.bme.hu/page/523/. Accessed: 2015-10-19, Language: Hungarian.

[29] Budapest University of Technology, Faculty of Electrical Engineering Economics (BME), and Informatics (VIK). BME VIK Visual Identity Guidebook. https://www.vik.bme.hu/files/00006209.pdf. Accessed: 2015-10-19, Language: Hungarian.

[30] Mark Otto and Jacob Thornton. Bootstrap. http://getbootstrap.com/. Accessed: 2015-10-30.

[31] rackt. React/Router. https://github.com/rackt/react-router. Accessed: 2015-11-15.

[32] Nóra Szepes. Angular test program. https://github.com/lordblendi/angular-test. Accessed: 2015-11-12.

[33] Nóra Szepes. Mithril test program. https://github.com/lordblendi/mithril-test. Accessed: 2015-11-12.

[34] Nóra Szepes. React test program. https://github.com/lordblendi/react-test. Accessed: 2015-11-12.

[35] Nóra Szepes. Házibeadó portál, hallgatói elvárások. Technical report, Budapest, 2015. Language: Hungarian. This document is not publicly available.

[36] W3C. The global structure of an HTML document. `http://www.w3.org/TR/html401/struct/global.html#h-7.5.4`. Accessed: 2015-10-20.

[37] W3C. What is the Document Object Model? `http://www.w3.org/TR/DOM-Level-2-Core/introduction.html`. Accessed: 2015-10-20.

Appendix A

Data Dictionary

The data dictionary describes the meaning of the words and terms used in the Educational Support System and the Software Laboratory 5 course. To search synonyms and write definitions I used an online synonym dictionary [7], and an online explanatory dictionary [6].

- **Administrator** A person, who is responsible for running the administration system.
- **Course, subject** A program of instruction in a university.
- **Demonstrator** A person, who teaches a group of students.
- **Entry test, short test, quiz** An evidence that verifies the preparedness of the student.
- **Entry test grade, mark** A number indicating the quality of the student's preparedness.
- **Evaluator** A person, who evaluates the laboratory reports.
- **Event, educational event** An educational event is a class with a date for students to participate.
- **Exercises, tasks** A list of exercises that provides experience to a student with a technology.
- **Laboratory** A type of class held in a computer laboratory by a demonstrator to a group of students.
- **Laboratory grade, mark** A number indicating the quality of the student's laboratory work.
- **Laboratory report, documentation** The documentation about how the student solved the list of exercises.
- **Review, remark** The evaluator's assessment of the quality of the solutions and submitted materials.
- **Semester, term** Half of a school year, lasting about five months.
- **Source code** The program code written by a student to solve a list of exercises.
- **Student, pupil** A person, who is responsible for running the administration system.

Appendix B

User Stories

Feature: Student module

```
As a student
I want to get informations about my laboratories
to know where to upload my homework
and read the remarks of my homeworks
and change my basic settings
```

<u>Before laboratory:</u>

```
Background:
    Given a student named "Jakab"
    and his password and username are entered in the login fields
    and it is one day before the laboratory
    and a finished homework uploaded to the Git repository

Scenario: Getting informations about the next laboratory
    Given I open the Laboradmin page
    When I press the login button
    Then I should see the date of my next laboratory
    And I should see the room number of my next laboratory
    And I should see the name of my teacher
```

<u>During laboratory:</u>

```
Background:
    Given a student named "Jakab"
    and his password and username are entered in the login fields
    and he is sitting at the laboratory

Scenario: Getting a Git remote URL
    Given I open the Laboradmin page
    When I press the login button
    Then I should see a Git remote URL

Scenario: Check how many hours I have left
    Given I open the Laboradmin page
    When I press the login button
    Then I should see a timer with a number between 96 and 92
```

After laboratory, before deadline:

Background:
Given a student named "Jakab"
and his password and username are entered in the login fields
and it’s one day before the deadline
and a finished homework uploaded to the Git repository

Scenario: Getting a list of Git commits
Given I open the Laboradmin page
When I press the login button
Then I should see a list of branches, commits and tags

Scenario: Check how many hours I have left
Given I open the Laboradmin page
When I press the login button
Then I should see a timer with a number between 24 and 0

Scenario: Marking a commit as final
Given I open the main page
And I see a list of branches and commits
When I click on one of the commit in the list
Then I should see "The commit was marked as final."

Scenario: Removing a final mark
Given I open the main page
And I see a list of commits
And one commit is already marked as final
When I click on the master branch in the list
Then I should see "You have succesfully removed your final mark."

After laboratory, after deadline:

Background:
Given a student named "Jakab"
and his password and username are entered in the login fields
and it’s one day after the deadline

Scenario: Getting my grade and review
Given I open the Laboradmin page
When I press the login button
Then I should see my grade and review

Other situations:

Background:
Given a student named "Jakab"
and his password and username are entered in the login fields

Scenario: Getting a summarized list of my grades
Given I am logged in as "Jakab"
When I press the summary button
Then I should see all of my grades

Background:
Given a logged in student named "Jakab"
and the settings page is loaded

```
Scenario: Setting a new SSH public key
    Given I am logged in as "Jakab"
    And I have entered a new SSH public key
    When I press the save button
    Then I should see "Your settings have been saved."

Scenario: Setting a new e-mail address
    Given I am logged in as "Jakab"
    And I have entered a new e-mail address
    When I press the save button
    Then I should see "Your settings have been saved."

Scenario: Changing my subscription for the mailing list
    Given I am logged in as "Jakab"
    And I clicked the checkbox next to "Subscription for mailing list
    When press the save button
    Then I should see "Your settings have been saved."

Scenario: Changing my subscription for notifications
    Given I am logged in as "Jakab"
    And I clicked the checkbox next to "Subscription for notification"
    When press the save button
    Then I should see "Your settings have been saved."
```

Appendix C

Design Sketches

I didn't use a tool, because the design is not standardized, and it's easier and faster drawing by hand then by a tool.

*****TODO***: szebben lerajzolni vonalzóval!!!!! vagy megcsinálni az egészet egy tool-lal, mert ez így csúnya és nem azért, mert nem tudok rajzolni, hanem mert scannelve van**

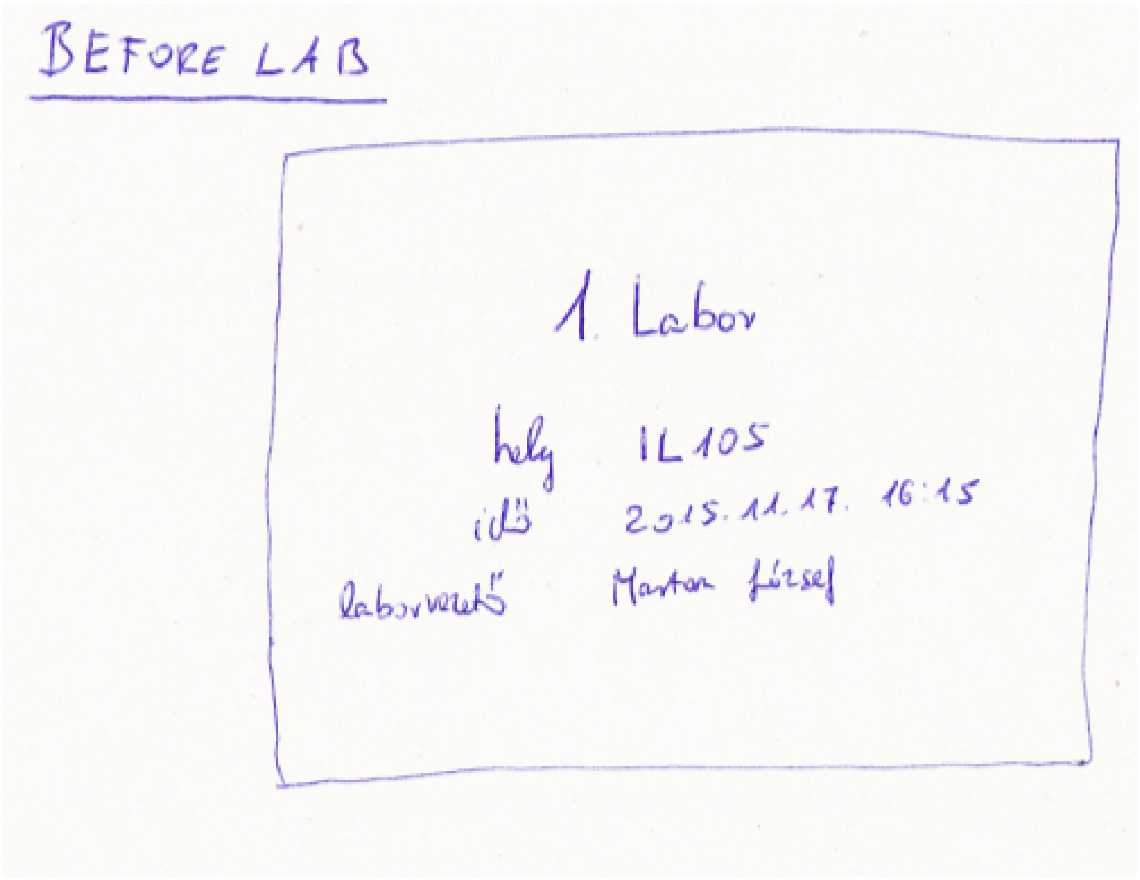


Figure C.0.1. Laboratory page sketch, before lab

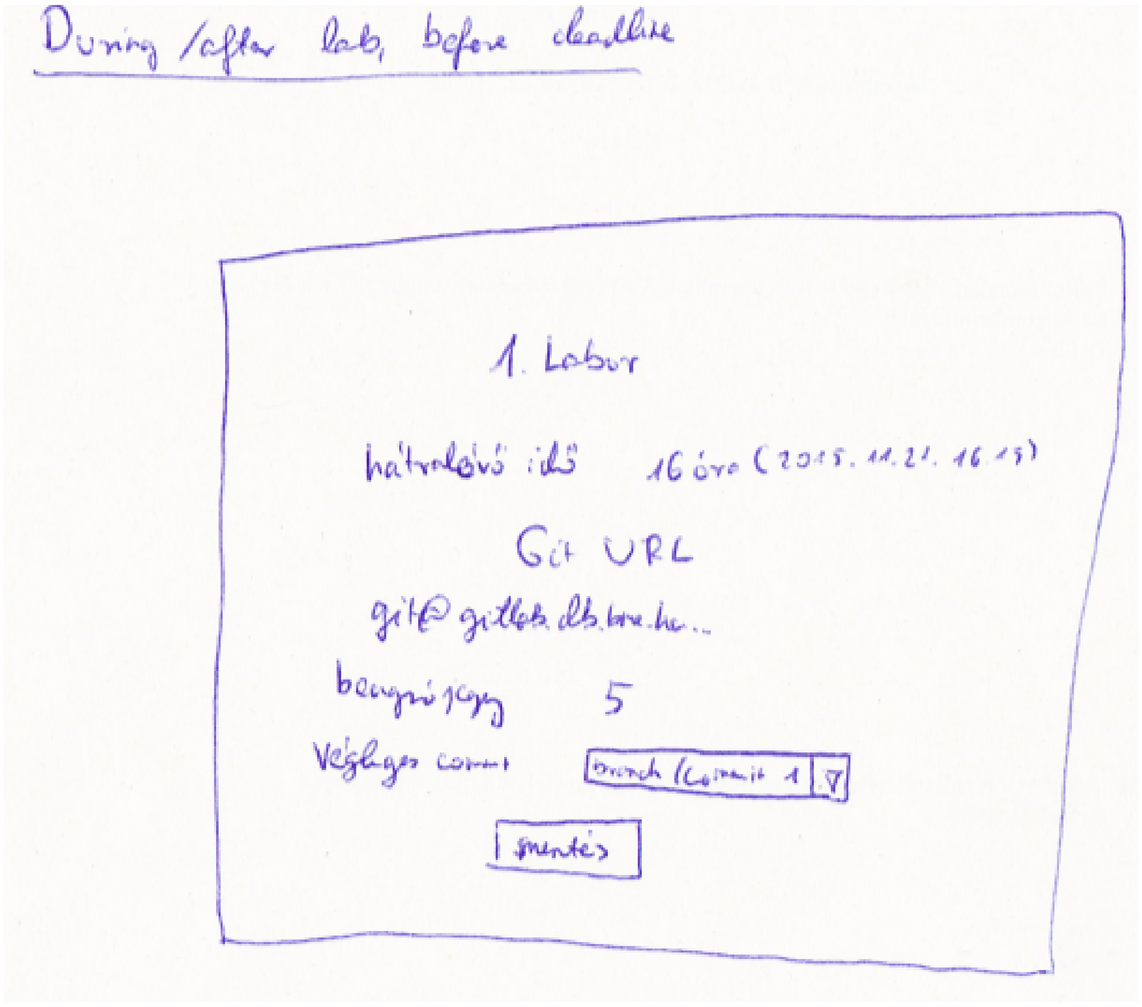
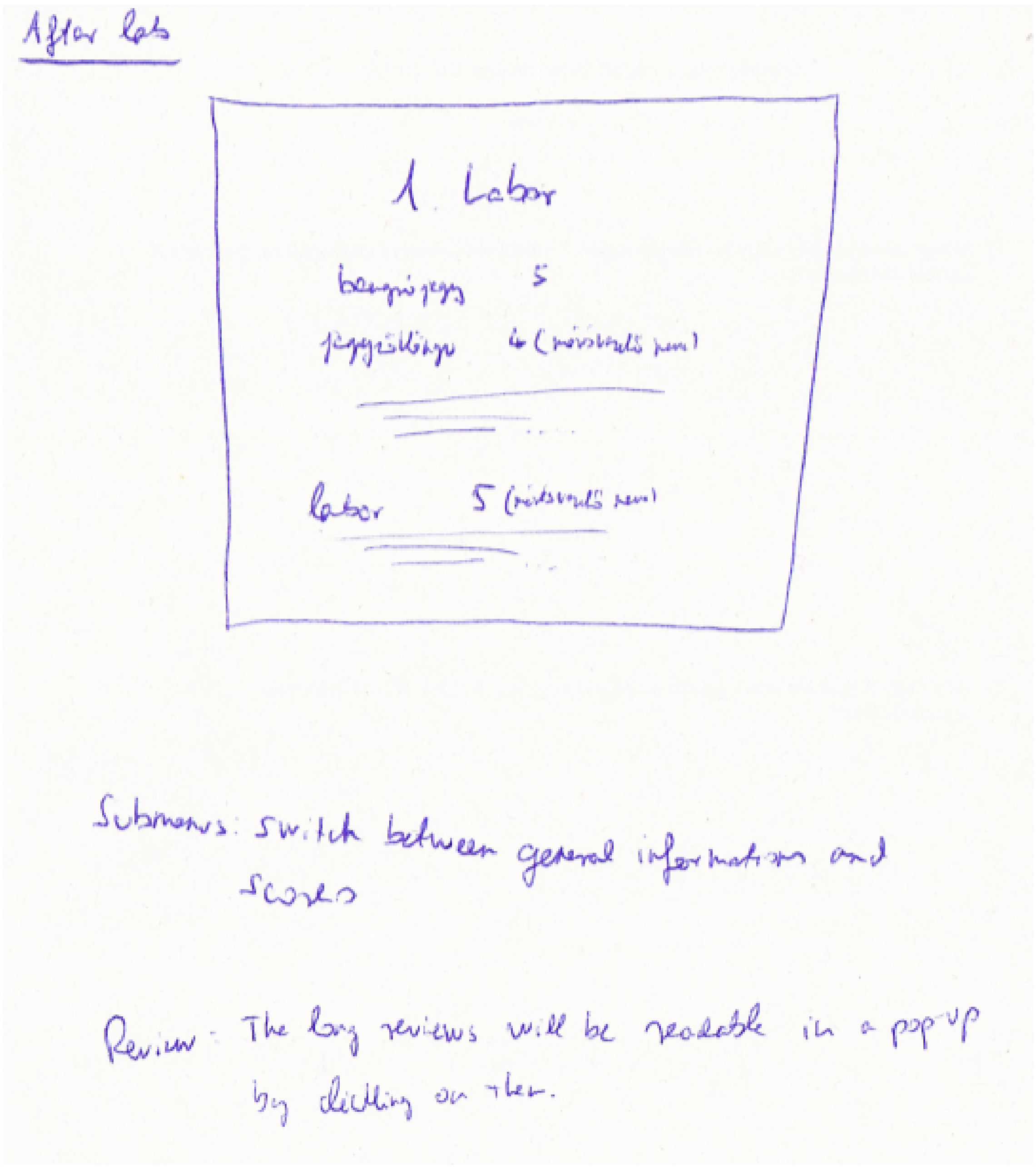


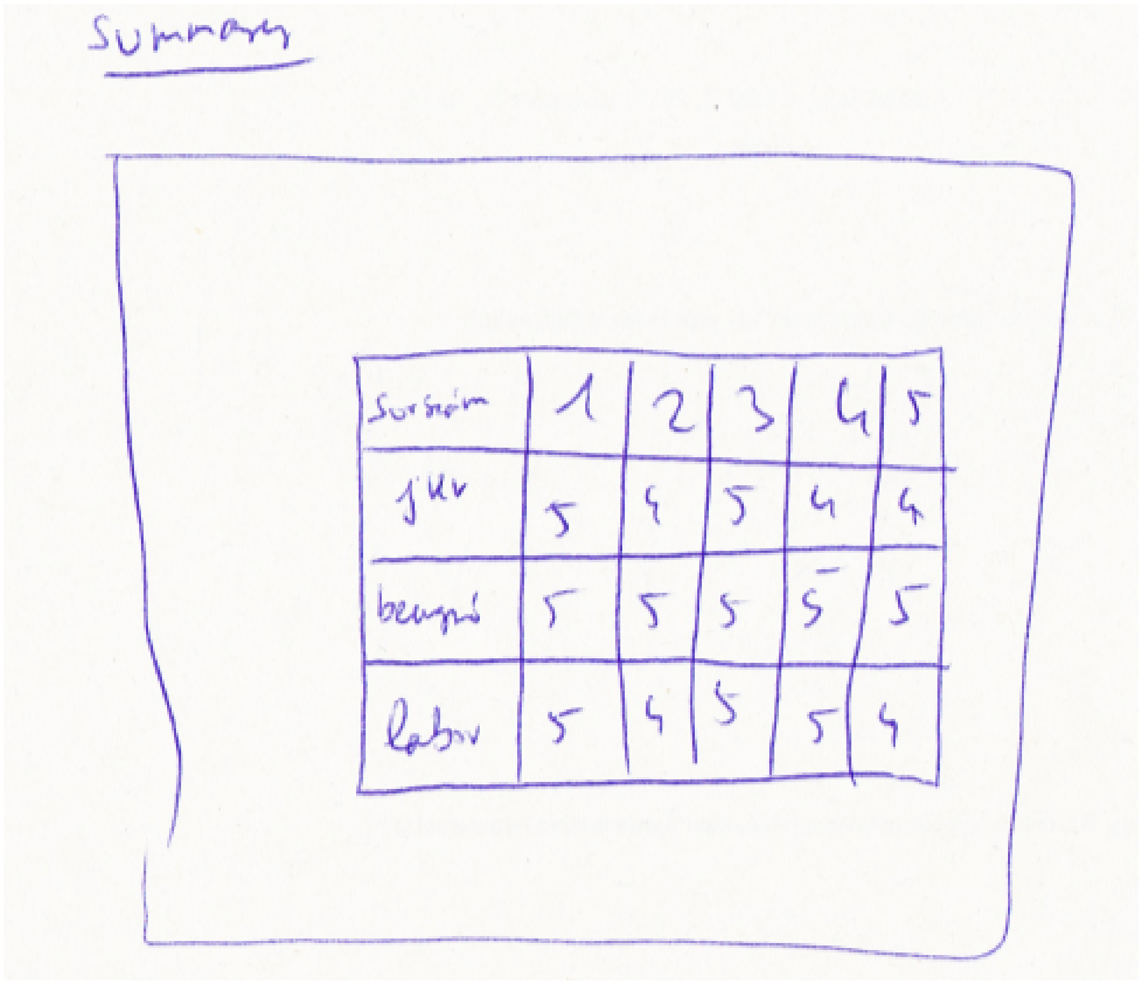
Figure C.0.2. Laboratory page sketch, during/after lab, before deadline



Submenus: switch between general information and scores

Review: The long reviews will be readable in a pop-up by clicking on them.

Figure C.0.3. Laboratory page sketch, after lab



Sorszám	1	2	3	4	5
jkv	5	4	5	4	4
beugró	5	5	5	5	5
labor	5	4	5	5	4

Figure C.0.4. Summary page sketch

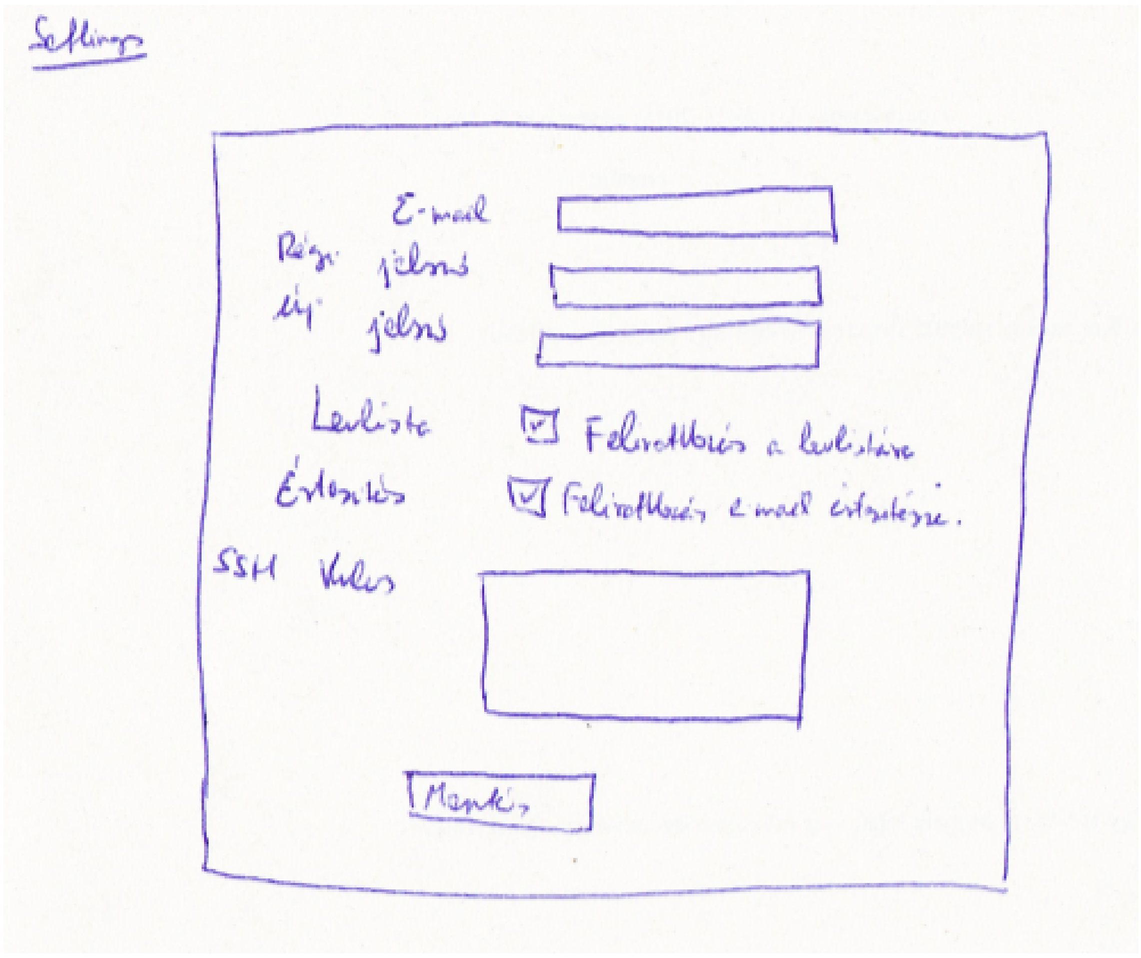


Figure C.0.5. Settings page sketch