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

Design and Implementation of an Educational Support System

Bachelor's Thesis

Candidate
Nóra Szepes

Advisors
Dr. Sándor Gajdos
Bence Golda

2015

Table of 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. december 10.

Szepes Nóra
hallgató

Kivonat

A Szoftver Laboratórium 5 tantárgy lebonyolítása – az órák megtartása mellett – több száz hallgató adminisztratív feladatainak elvégzését igényli. Ezen feladatok elvégzése egy adminisztrációs rendszer segítségével jelentősen megkönnyíthető. A tantárgy eddig is rendelkezett egy portállal, ami azonban az évek során a technológia fejlődésével, illetve a felhasználók újabb elvárásai miatt elavulttá vált.

A szakdolgozatom célja egy olyan új portál és a hozzá tartozó vékony kliens tervezése, amely megfelel a tantárgy követelményeinek, illetve lehetőséget nyújt több tantárgy kezeléséhez is. A végleges rendszer három modult tartalmaz mind a back-end, mind a front-end részen: a hallgatói, az oktatói és az adminisztrációs modult. Szakdolgozatomban a front-end hallgatói moduljának elkészítését mutatom be.

Munkám során egy fejlesztői csapattal kellett együttműködnöm. Az én szerepem a rendszer tervezése, illetve a kliens elkészítése volt a csapattal egyeztetve. A csapat többi tagja a szerver oldali funkciók elkészítéséért felelős.

A vékony kliens tervezése és implementálása előtt megismerkedtem több JavaScript keretrendszerrel. Az összehasonlítás során az útvonalválasztás, az AJAX kérések és az adatkötés lehetőségeit vizsgáltam, hiszen ezek elengedhetetlenek egy egyoldalas webalkalmazás elkészítéséhez. A Mithril keretrendszert választottam, mert gyors, minden szükséges funkciót támogat és nem függ más keretrendszerektől.

A funkcionális specifikálás során először összegyűjtöttem, hogy milyen funkciókat kell mindenképpen nyújtania a rendszernek. Ehhez hozzávettem azokat az új funkciókat, amelyek hiányát az előző portál felhasználói jelezték. Az így kapott funkcionális specifikáció alapján megterveztem a hallgatói kliens architektúráját, funkcionalitását és külalakját.

A rendszertervezés a Design by Contract módszertan elvein alapul. Definiáltam a kliens és a szerver közötti interfészeket, de magukat a szerver oldali komponenseket nem. Mivel a back-end elkészítése nem az én feladatom volt, ezért a fejlesztés és a tesztelés során az elkészített interfészeket egy ún. mock szerverrel működtettem, ami biztosította a szükséges tesztadatokat.

Mithrilben és Bootstrap 3-ban implementáltam a hallgatói klienst. Az MVC minta nézet rétegében Bootstrap 3 szolgáltatja a külalakot. Az elkészült kódot a könnyebb fejlesztés és karbantarthatóság érdekében modulokon belül is tovább bontottam. A futtatható kód előállítását a gulp rendszerrel automatizáltam.

Munkám hangsúlyos részét képezte az elkészült komponensek tesztelése. A specifikáció alapján az implementálandó klienshez elfogadási teszteket és kód lefedettségi tesztet készítettem. A teszteket a Cucumber, Zombie és Istanbul rendszerek segítségével implementáltam és futtattam, amelyek visszaigazolták a tervezés és megvalósítás megfelelő.

Abstract

The Software Laboratory 5 course besides holding classes also requires administration tasks to be carried out for hundreds of students. The completion of these tasks can be disburdened with an administration portal. The course has a portal, but that is out-of-date because of the technological improvements and the users' new requirements.

The objective of my thesis is to design a new portal and a thin client, that meets the course's requirements and supports management of more than one course. The final system will have three modules in both the back end and the front end: student module, teacher module and administration modules. In my thesis I introduce and describe the implementation of the front end student module.

During my work I cooperated with a team of developers. My responsibility was to plan the system and implement the client agreed with the other team members. The rest of the team has to implement the server-side.

Before designing and implementing the thin client I investigated some JavaScript frameworks. During comparison I examined the routing, data-binding and AJAX request features, since these are necessary for a single-page application. I chose the Mithril framework, because it's performance is fast, it is self-contained and provides built-in utilities for the required features.

During the specification phase I collected all the required features, that have to be provided by the portal. Further, I added the required features, that were reported by the old portal's users. Based on this functional specification I designed the student client's architecture, functionality and graphic design.

The conceptual system design is based on the Design by Contract methodology. I defined the communication interfaces between the server and the client, but did not define the server's components. Because the implementation of the back-end is not part of my task, I used a mock server during the implementation and the test phase. This provided the required mock data for the client.

I implemented the student client in Mithril and Bootstrap 3. The graphic style in the MVC pattern's View is served by Bootstrap 3. For improved maintenance and easier implementation I divided the code into smaller parts within the modules, and automatized the build task with gulp.

An important part of my work was to test the implemented components. For this I created acceptance tests and code coverage test based on the specification. The tests were implemented in and run by the Cucumber, the Zombie and the Istanbul frameworks. The results confirmed that the design and the implementation were satisfactory.

Chapter 1

Introduction

During the summer of 2015 my advisor, Sándor Gajdos contacted me to provide him with feedback about his course, Software Laboratory 5. I advised him the strength and weaknesses of the course. I also shed a light on the administration portal which did not have e-mail notification. I offered Sándor Gajdos to develop that feature into the current portal. All I knew that it was written in PHP. After telling him my ideas he contacted Bence Golda, the creator of the old portal, to ask for some information about the portal's code and József Marton to create a "noreply" e-mail address for the notification module. József Marton suggested that we could create a new portal and other members of the team, Bence Golda, Gábor Szárnyas and I agreed with him.

Before our first meeting in early August I decided to look up all the different homework portals I have used during my student years. I asked for an account to Zoltán Czirkos's InfoC [12], because that website was started after I have finished the Basics of Programming 1 course. After the information gathering, I prepared a preliminary specification for an ideal homework portal and some recommendations of how we could use the same portal for more than one course.

During the meeting, we talked about this topic, and also about what others may expect from a new portal. It started as a departmental project but József Marton asked for some ideas about what students would like to gain from a portal. The project sounded exciting and I wanted to participate but since I had to prepare for my thesis I would not have sufficient time to work on the portal. At the point Sándor Gajdos offered me the topic for my thesis and he agreed to serve as my advisor. I accepted his offer as this is an appealing engineering task from design to implementation, and – in addition – the final portal will be used by hundreds of students.

1.1 The Old Administration Portal

The old portal was developed during the spring semester of 2003 by Bence Golda and three other students. In that year the course Software Laboratory 5 had database themed and network themed laboratories. This project was available for some students as advanced database laboratories. The students did not have much software developing experience.

Sándor Gajdos wrote the specification and the students designed and implemented the portal. They used version control, but did not wrote any formal tests. The portal has Oracle SQL basics with a PHP engine and an XSLT templating system.

The first expansion, the repeated laboratory practice handling was added three years later. Around 2008 Bence Golda made Ruby scripts to handle some more related tasks, e.g., initialization.

A group of developers from the university and two intern groups tried to design and implement a new portal before, but they did not succeed.

The old portal is in use since 2003 with minor difficulties, and helps the students to pass their laboratories. Bence Golda spends about 30-40 hours every semester doing operational tasks.

1.2 The Objective of the Thesis

My objective is to design a new administration portal and implement the front end's student module. In chapter 2 I introduce the specification phase. Then I compare frameworks in chapter 3. The conceptual system design, entity-relationship data model, front end system design will be introduced in chapter 4. I introduce the client's graphic design in chapter 5. The deployment will be described in chapter 8. I introduce the method of implementation in chapter 6. The testing phase will be described and the result will be summarized in chapter 7. And at last but not least I summarize my work in chapter 9.

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 up a preliminary specification [84] about what I expect from a homework portal as a student. Since September 2015 we had weekly meetings where we discussed the following related topics:

- the old portal's features we want to keep without modifications
- the old portal's features we want to keep with modifications
- the old portal's features we want to eliminate, and
- new features we want to add

After a couple of meetings we had a list of functional features as the starting point for our specification. A functional feature describes what a user can do while he uses the portal. At this point I wanted to find the common roots of the features. When there are many features with the same common root, then I know I should focus on implementing these features first, because these features are mandatory features, and without them the portal will be useless for the users. I used the 5 Whys technique to find the common roots of the features.

2.1 Specification Phase

In this section I am going to introduce what kind of techniques I used to create the final specification. Some of these techniques serve as the basics of functional testing and acceptance testing. A functional test verifies that the software or features meets the specification. An acceptance test validates that this is the software or feature the customer wanted. I explain the testing in chapter 7.

2.1.1 The 5 Whys

The 5 Whys technique [9] was recommended by Bence Golda.

The 5 Whys is a technique to find the root cause of a problem (in my case a feature) simply by asking the question "Why?" multiple times. With this technique I can find out what does the user would like to achieve by using a feature. Why would the user like to use

this feature? Does the user really want to use this or should the portal do this by default instead of the user? The 5 Whys also helps to decide when I should implement only one general feature instead of many specific features. For example, if the user wants to have 15 specific analytic features, I could consider making a general one instead of specific ones.

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 the student's homework.
- *Why?* – Because the evaluator has to give the student a grade.
- *Why?* – Because the student has to pass the laboratory in order to pass the course.

In consequence of the 5 Whys, I also filtered out some features. For example, in the first version I wanted to have file uploading in the portal, but it has the same root as having a git repository. I have decided to eliminate the file uploading feature, because student can learn the usage of a version control system by submitting their homeworks to a repository. Then I started to write the user stories to establish the basics for test automation.

2.1.2 User Story

A user story is a description of how can the user interact with the system, and what does the user have to do to use a certain feature. User stories are written in everyday language, and I can use them to run automated tests with a software tool.

Given When Then

In test-driven development (TDD) a developer first writes tests and then write then implements. A test is based on the specification, and the purpose of implementation is to pass the tests.

In behavior-driven development (BDD) a developer first writes behaviors and then implements. A behavior describes how the software should behave in certain situations. Tests and behaviors are similar, but behaviors are written in a ubiquitous language, that helps focusing on the behavior of the software and makes the tests fluidly readable [65] [14].

The Given When Then [58] [59] technique is a convention to write user stories in BDD. It was made by Dan North and Chris Matts and contains the following steps [54]:

- **Given:** describes a state of the system.
- **When:** an event, that happens to the system.
- **Then:** the state of the system after the occurred event.
- **And, But:** Multiple steps can
- **Scenario:** is an example of an executable action. It has to contain at least a *When* and a *Then* step.

- **Background:** If you have the same Given statement for many scenarios, you can combine them as a *background* section.

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."

At first I wrote simple user stories without backgrounds and scenarios. I wrote them for all modules, and then I focused only on the student module, because that's the first module I will implement. I finished writing scenarios for only that module (see appendix B).

Cucumber

User stories are good starting points for acceptance tests. For testing I will use Cucumber [55].

Cucumber is a software tool to run automated tests with user stories written in the Given When Then convention. Cucumber uses a parser named Gherkin [56] to parse feature files. A feature file contains user stories written in everyday language in a specific structure. Gherkin can parse user stories written in 65+ languages. In this project I will use English user stories. Although anything can be written in user stories, the more steps is included, the more code have to be implemented. The usage is described in subsection 7.1.1.

Example:

The previous Given When Then example implemented for Cucumber test. The logic for the steps is npt implemented yet

```
var myStepDefinitionsWrapper = function () {
  this.Then(/^I see "([^"]*)" as the page title$/, function (arg1,
  callback) {
    callback.pending();
  });

  this.Given(/^a logged in student named "([^"]*)"$/, function
  (arg1, callback) {
    callback.pending();
  });

  this.Given(/^the settings page is loaded$/, function (callback)
  {
    callback.pending();
```

```
    });

    this.Given(/^I am logged in as "([^"]*)"$/, function (arg1,
    callback) {
      callback.pending();
    });

    this.Given(/^I have entered a new SSH public key$/, function
    (callback) {
      callback.pending();
    });

    this.When(/^I press the save button$/, function (callback) {
      callback.pending();
    });

    this.Then(/^I should see "([^"]*)"$/, function (arg1, callback)
    {
      callback.pending();
    });
};
module.exports = myStepDefinitionsWrapper;
```

2.1.3 Contract

Design by Contract is a software development methodology made by Bertrand Meyer. A contract describes how a software's components a collaborates with each other. A contract has obligations and benefits. An obligation for a component is a benefit for the other component, and vice versa [60] [83] [1].

User stories are lines for a contract, that can be verified with an automatized testing tool, but the components are different: one component is the specification, the other is the client.

I use this methodology to describe the communication between the back end and the front end. I was looking for an API specification tool to design the communication between the back end and the front end and a mock server tool, that implements that API specification.

Mock Server

Because I only develop the front end, I wanted to make sure that I can work on the implementation without waiting for back end modules to be finished. To do that, I have to write an API specification and use a mock server.

A *mock server* simulates the behavior of the back end and offers mock data to work with. With an API specification it will offer the same data as the future back end, and gives the opportunity to test the developed front end modules before the back end is finished.

API Blueprint

I want to use my API specification as a readable documentation for humans and as a mock server specification. *API Blueprint* [5] is an open source project, that gives me the option to write my specification in Markdown. Markdown [11] is a markup language to write plain text format and it can be converted into HTML. Markdown is often used to style messages on forums and readme files.

Example GET:

```
# Message [/student/{userid}/general]
## Get student general informations [GET]
+ Response 200 (application/json)
    {
      "name" : "Teszt Hallgató",
      "neptun" : "Neptun",
      "id" : "1"
    }
```

This example sends the same json response for every HTTP GET request. The *userid* is a wildcard parameter, the server will send the same response for every parameter. I used the parameter *1* for testing.

Example POST:

```
# Message [/student/{userid}/setsettings]
### Post new settings for student [POST]
## Automatic response to OPTIONS requests
+ Request (application/json)
    + Schema
      { "type":"object",
        "required": ["notification", "mailingList"],
        "properties" : {
          "email": {"type":"string"},
          "notification": {"type":"boolean"},
          "mailingList": {"type":"boolean"},
          "sshPublicKey": {"type":"string"},
          "oldpwd": {"type":"string"},
          "newpwd": {"type":"string"}
        }
      }
+ Response 201 (application/json;charset=UTF-8)
    + Body
      {
        status": "ok"
      }
```

This example sends the same json response for every HTTP POST or OPTIONS request, where the body matches the JSON Schema. Only two parameters are required, but the body cannot contain any parameter, that is not defined in the properties section.

The files are available on Github [82].

Drakov

The API Blueprint [5] team offers some tools, that implements API Blueprint specification. *Drakov* [51] is one of these open source mock server tools. Its usage is described in section 8.4.

2.2 The Final Specification

The final specification was made by me and finalized by the team (see subsection 2.2.2).

During the specification phase I eliminated dispensable features. Although we wanted to keep some of the old portal's feature, we wanted to add new ones too. Some of these new features were already supported by the old portal, but in a different way. I have decided to eliminate one of these features to prevent duplication.

For example, the first version of the specification included file uploading for the reports via the portal. This feature is dispensable, because the user can store the reports in the git repository too. I could eliminate the git repository feature and keep the file uploading feature, but the course's goal is to teach the students how to use version control, and file uploading is not a version control system. Eliminating this feature leads to less task for the user, because he does not have use two features to submit his homework.

2.2.1 Authentication

The portal will use the Sibboleth [75] for user authentication and simple login with a username and a password. Sibboleth is a solution to connect the users between applications. The BME offers this for students and university staff members to use only one account for every portal and application. The portal still has to provide another solution for users, that do not have a Sibboleth account, e.g., a user, who is not teaching, working or studying at the university.

2.2.2 User Roles

In the new portal there will be four different user roles: student, demonstrator, evaluator and administrator.

Authorization

A staff member can be a demonstrator and/or evaluator and/or administrator. The portal will divide the different role's features to different pages. This gives an option to choose between the roles, e.g., the menu will contain a button for switching roles. If a user does not have access to a feature, it will not appear in the student.

Student Role

The student can:

- see general information about his classes
  - when will it be
  - where will it be
  - who will be the teacher
  - when is the deadline for submitting the homework
  - 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, that was tagged by the back end.
- see a summarized view of his grades

Teacher

The teacher can be a demonstrator and/or an evaluator (see appendix A).

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 a maximum of 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, that are waiting for evaluation and are booked by the evaluator.
  - The second list contains the homeworks, that are waiting for evaluation and not booked yet.
  - The third list contains homeworks booked by another evaluator, who evaluates the same type of homeworks.
  - 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 additional 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 that 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 Design Sketches

For the graphic design I need to know how many pages are needed for each user roles in each modules. In this thesis I present the implementation of the client's student module. This module is only available for users with student role. In the student module I will implement the following features in priority order:

1. see the general information about the classes
1. see the results
2. see a list of commits and tagging a commit as final version

2. set new password, e-mail and SSH public key
3. summarized view of student's grades

The features with priority level 1 are mandatory, because without these features the portal will be useless for the students.

The features with priority level 2 are important, but the portal is not useless without it. The students can still tag a commit with a Git client, or merge the final commit with the master branch to make it the master branch's last commit.

The features with priority level 3 are not important, but can be useful for students. The portal is usable even without this feature, because the students can check their grades on the educational event's page.

In chapter 7, I will test the features with priority levels 1 and 2.

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

1. Before a laboratory
   - To get information about the laboratory
     - When will it be
     - Where will it be
     - Who will be the demonstrator
   - To read general information about the course
2. During a laboratory
   - To upload an SSH public key
   - To get his Git remote URL
3. After a laboratory, before the deadline for submitting the homework
   - 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 the grades
   - To check the reviews
   - To check the evaluator's name
5. Other scenarios
   - To set a new e-mail address
   - To set a new password
   - To change the mailing list subscription
   - To change the 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 based on the scenarios. This module will have a laboratory page, a summarized view page and a settings page.

Laboratory page, before laboratory: For this information I will use only labels (see figure 5.1).

Laboratory page, during laboratory, after laboratory and before deadline: For the information, like deadline and entry test grade I will use labels. For the list of commits I will use a dropdown menu. When the page is loaded, the last final commit will be the chosen one. If the user did not chose a final commit before, the last commit in the master branch will be the chosen one (see figure 5.2).

Laboratory page, after laboratory and after deadline: For these information I will use labels. The reviews can be long, and I want to keep the design simple. Because of this, there will be a size limit for the label on the view, and the rest of the text will be hidden with an ellipsis. The user can click on the text to show the rest of the review in a pop up window (see figure 5.5).

Settings page, other scenarios: The new e-mail address, password and SSH public keys will have input fields. The subscription will be changeable with check boxes (see figure 5.3).

Summarized view page, other scenarios: The summarized view will be a simple table with the grades in it (see figure 5.4).

Menu: Because the user is looking for a specific set of information, the page will only contain the important information, and the previous, but still relevant information will be accessible with tabs on the laboratory page. To be able to switch between the pages, I will use a navigation bar on the top of the page. The bar will have the logo of the portal, student's name and neptun code and one button for each pages.

I drew sketches (see appendix C) for every scenarios with placeholder data.

Chapter 3

Comparing Frameworks

The Client will be implemented in JavaScript, because all the modern web browsers support JavaScript via an interpreter. JavaScript is a cross-platform scripting language [2]. For the project, I preferred to choose a JavaScript framework for faster development than using plain JavaScript with jQuery. Writing every component from scratch needs a lot of time. Hundreds of developers contribute in open-source frameworks, and developers can utilize their reviewed and tested work. This leads to getting more tasks done in less time. Because this project follows the MVC pattern (see subsection 4.1.1), only JavaScript MVC frameworks will be compared. The TodoMVC [3] website helps developers to select a JavaScript MVC framework for their project. It provides the same example written in JavaScript using different JavaScript MVC frameworks.

During comparison, I checked the followings:

- **AJAX requests:** the client and the server (see figure 4.3) will communicate via REST using JSON format.
- **data binding:** to connect the data from the model to the view (see figure 4.1).
- **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 [61].

HTTP requests are necessary, because the client has to download data from the server and upload data to the server. 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 will not reload the whole page, it will just simply load the new view into the old frame. Everything happens in the background so the application will not 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. This is necessary, because data binding helps separating the model's code from the view's code, and supports code maintainability. Any data changes in the view will not automatically sync into the model. The developer has to write the controller that syncs the changes between the model and the view [27].

Upon URL change an SPA will not 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. This is important, because without routing the application would fetch new pages whenever the URL changes.

During the comparison I made a small test program in every JavaScript framework to try AJAX requests, data binding and routing. The source codes are publicly available on Github with a gulp file (see section 8.1) and an API blueprint test file for Drakov (see subsection 2.1.3).

3.1 React

My first choice was React [45], because it is well-known for it's virtual DOM and Flux. 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) [86]. 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 this 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 that is just data and not a rendered content. React is able to find out what has changed on the real DOM. It makes changes to the virtual DOM, because that is faster and then re-render the real DOM [46].

To create DOM elements, we can choose between JavaScript and JSX [44]. 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 [43]. 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. I did not like that the core React version does not have an option for routing and AJAX requests. If I want to support those in my application, then React should be combined with other frameworks or libraries to have a full MVC experience.

Test Program [80]
The components are separated into different JavaScript files. JSX was used to represent the views. Before the concatenation a JSX transformer converts the inline HTML-like JSX to JavaScript code. The React components' states were used as a model. Because React has not got an option for AJAX requests, jQuery was used to download some mock data from the mock server. React Router was used [73] for routing.

3.2 AngularJS

AngularJS [24] is one of the most advanced JavaScript frameworks nowadays. I chose it because it is famous about the two-way data binding. It has been maintained by Google for 6 years. It focuses mostly on dynamic views in web-applications.

[1] A div is a generic container in HTML. It helps structuring the HTML document [85].

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 [27] which means whenever either the View or the Model is changed, it will update the other one automatically.

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 [25], 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 cannot navigate through the pages [26] [28].

Test Program [77]
The controllers are separated into different JavaScript files and the views into different HTML templates. The routing connects the right HTML template to the right controller. The JavaScript files are concatenated using gulp. The HTML templates are in different HTML files. Variables inside the controllers were used as a basic model. The routing is not included in the core Angular framework, but Google provides a library for it.

3.3 Mithril

Mithril [34] is a small MVC framework created by Leo Horie. I chose it because 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, that is a JavaScript object that represents a DOM element. Rendering will create a real DOM element from the virtual one [35] [37]. If we prefer using HTML syntax, we can use MSX [7]. 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 [39]. 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 [41].

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. [40] [36].

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. [38].

Test Program [79]
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. Modules are separated into different JavaScript files and concatenated with gulp. MSX transformer converts the HTML-like MSX to JavaScript code.

3.4 Comparison Table and Conclusion

	React	**Angular**	**Mithril**
AJAX requests	Not part of the core framework. Requires an external library or framework.	Provides simple AJAX methods. It can be used for unit tests, because it can create a mock server.	Provides a simple utility. An early reference can be set for the response to qeue up operations to be performed after the request completes.
data binding	One-way data flow, downstream. if something is changed in the component tree, it re-renders itself and all of its descendants.	Two-way data binding. Either the model or the View is changed, it will update the other one autimatically.	One-way data flow. Uses a diff algorithm, that decides which part of the DOM needs to be updated. It is triggered by event handlers.
routing	Not part of the core framework. Requires an external library or framework.	A special listener is binded to the links. It pushes the page to the browser's history.	Uses a key-value map as a routing table, that connects the URLs with Mithril module controllers.

Table 3.1. Framework Comparison Table

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 does not have routing and AJAX requests in its core library, so it depends on other libraries. I choose Mithril for this project, because it is self-contained, so it does not 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.

3.5 Supported Web Browsers

At first, I am aiming to develop the portal for desktop browsers. The portal will support every browser, that is supported by Google [31]. All browsers must have JavaScript enabled. The supported browsers are based on the browser usage stats in Hungary [76]:

- Chrome 39.0
- Internet Explorer 10. [30].
- Firefox 33.0
- Safari – previous major releases

3.5.1 Internet Explorer

Because there are some features, that Mithril depends on and are not part of earlier Internet Explorer version, I added a shim JavaScript file to the project [42].

To force Internet Explorer to render in the highest available mode, I added a special meta tag in the html file [63] [10] (see section 6.3).

Chapter 4

Conceptual System Design

The conceptual system design represents a high level structure of the system's architecture. It defines the relations between the components. The components have to be separated from each other, because then every component is easily replaceable without changing any other components. The conceptual system design does not require the components to be implemented with the same technologies. The clients will be implemented in HTML, CSS and JavaScript (see chapter 3) and the back end components will be implemented in Ruby on Rails.

4.1 Frontend System Design

The front end system design depends on the chosen framework's behavior and it also depends on which software architecture pattern is used. This project follows the MVC pattern. In this section I am going to introduce the general MVC pattern and the final front end system design.

4.1.1 MVC Pattern

The Model-View-Controller (MVC) is a software architecture pattern for user interface design and implementation, where the application logic is separated from the user interface [1].

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

MVC Web Applications

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 is 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 serialization. serialization is the process, that transforms the data between dataformats to be sent and stored. When the model returns the desired data to the controller, it will forward the data to the view. The presentation layer will

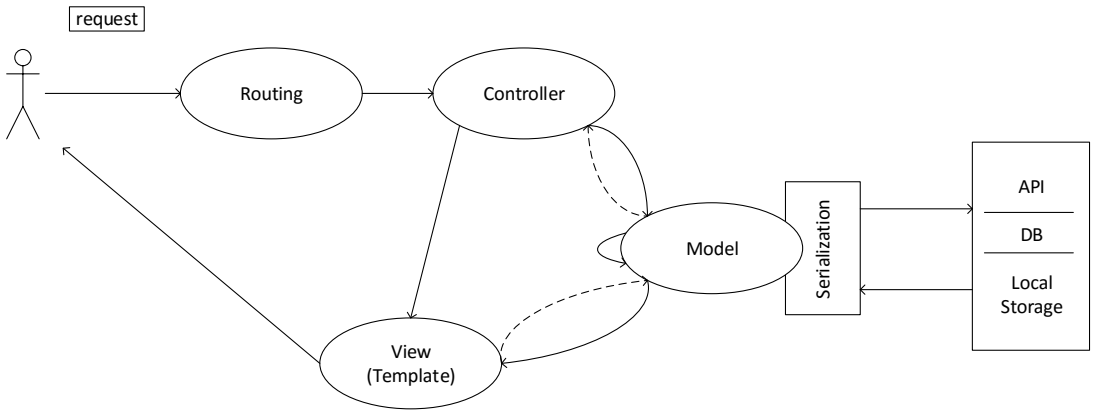

Figure 4.1. Classic MVC Web Application [4]

decide which page has to be returned to the browser, binds the data to the view template and returns it.

4.1.2 Final Front End System Design

The final design is based on the classic MVC Web application (see figure 4.1) and the behavior of Mithril [33]. It describes the design of the client, but does not describe how the clients communicate with each other, because the different clients cannot communicate with each other.

The routing component handles the incoming requests. It uses a routing table to decide which controller has to handle the request.

The controller talks to the model if any data should be changed or needed. The model sends data to the API or asks for data from the API. Every data is converted between the representation suitable for the implementation and the AJAX requests by the serialization component. The communication between the API and the client is handled by the server communication component.

The controller provides helper methods (e.g. getting data from the model) and it is passed as the first parameter to the view [38]. The rendered view is forwarded to the browser that visualized it for the user.

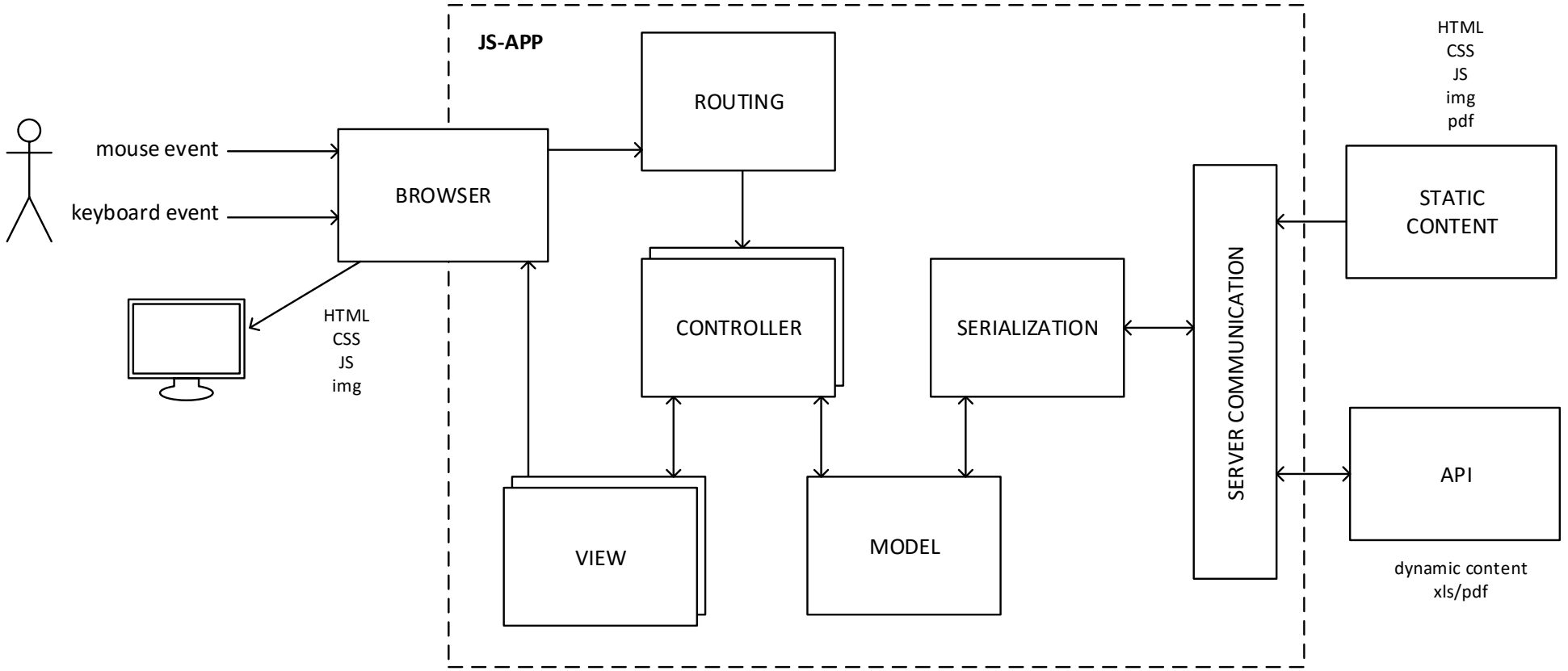

Figure 4.2. Front End System Design
The JS-APP was made by me and the outer layer was made by Bence Golda.

4.2 System Design

In this design the front end is represented as a basic component, because the front end system design was introduced in section 4.1. About 15 percent of the model was created by me, 85 percent of it was created by Bence Golda and it was finalized by the team.

The main components are the followings:

- **Client:** The communication bridge between the user and the web server. There will be three different modules: student, teacher and administrator. The users will be able to reach the client with a web browser. The users can log in with their accounts and features, e.g., getting general information, read reviews, give grades. 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 permanently (see section 4.5).
- **Git:** A database to store the students' homeworks. Every student will get a different git repository for each laboratory.
- **Load Balancer:** A component, that mitigates the chance of a scalability issue (see section 4.3).
- **Object-relational mapping:** It converts the data between the suitable representation for the implementation and the database.
- **Messaging:** A component, that supports sending messages, e.g., tasks between the different components. The web server, the task manager and the workers will use this to send tasks to each other. The Messaging component will have a message queue. In case a worker aborts, the message will be saved in the queue.
- **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 the clients data from the databases. The API is written in Ruby on Rails.
- **Worker:** A worker processes time-consuming tasks that takes longer and does not fit the request-response communication's time limit, e.g., changes the user's mailing list subscription.

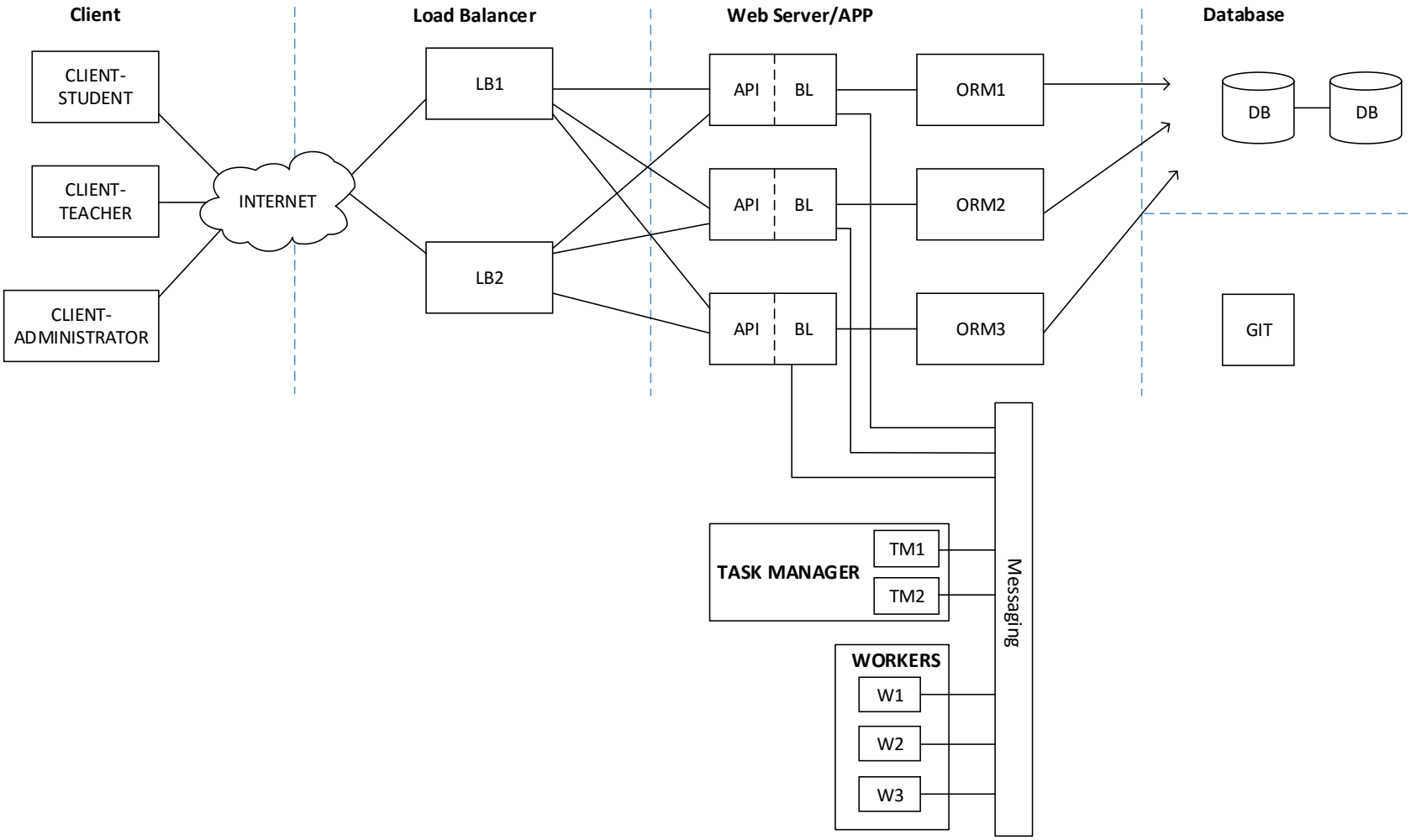

Figure 4.3. Conceptual System Design

4.3 Scalability

Client
To mitigate the scalability issue, 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 overloads and raise the response time. With a load balancer, the requests will first arrive to the load balancer. The load balancer will decide which web server will handle that request and forward 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.

4.4 Data Security

Both the back end and the front end are divided into three big modules: student, teacher and administrator.

Web Server
This gives us the option to run the different modules on different computers or virtual machine instances.

Client
This ensures that one module's source code is not enough to deduce the available data, e.g., the student module will not include some API routing rules, that are included in the teacher module and the administrator module.

4.5 Entity–Relationship Model

A data model describes the structures in which the database stores the data. The *Entity-Relationship model* is a formal notation system, that describes the data structure with entity sets and the relationships sets [19].

This data model is different from the old portal's data model, because it supports the followings:

- more than one course,
- stores data for more than one semester, and
- can manage repeated classes.

The project's model uses Chen's notation. About 65 percent of the model was created by me, 35 percent of it was created by József Marton. This data model is not final yet, because the other modules and the back end are still under development. The attribute list is documented in appendix D.

The main entities are the followings:

- **Appointments:** An appointment connects the student groups with a date and a location.
- **Courses:** The courses, that are managed with the new portal.
- **Deliverables:** The things to be submitted by the students (e.g. homework).
- **Deliverables/Repositories:** A special type of Deliverables that is to be submitted through a repository.
- **Deliverables/Files:** A special type of Deliverables that is to be submitted as a file upload.
- **DeliverableTypes:** Describes the type of the submitted homework, e.g. documentation, source code.
- **Evaluator:** A many-to-many relationship between the ExerciseTypes and the Staffs. It describes which exercise type is being evaluated by a staff member.
- **Events:** An educational event is a class with a date for a particular student to participate.
- **EventTypes:** An event can be any type of class: lecture, laboratory or seminar. The Software Laboratory course has only laboratories, but in case of using it with other courses, other type of classes will be supported too.
- **ExerciseCategories:** Describes the type of the laboratory: DBMS, SQL, JDBC, XML technologies in databases or SOA.
- **ExerciseTypes:** Describe the topic and the language of the exercises.
- **RegisteredStaffs:** A many-to-many relationship between the Semesters and the Staffs. It describes the staff member's role in the semester.
- **RegisteredStudents:** A many-to-many relationship between the Semesters and the Students. It describes which Neptun course the student registered for in the semester.
- **Semesters:** An instance of the course in a particular academic term.
- **StudentGroups:** In Software Laboratory 5 the students are assigned to different groups. A group has one demonstrator and about 20 students.
- **Users:** People, who use the portal during a semester.
- **Users/Staffs:** A type of user, who is not a student. This user can be an administrator and/or a demonstrator and/or an evaluator.
- **Users/Students:** A type of user, who attends the laboratories and solves a list of tasks.

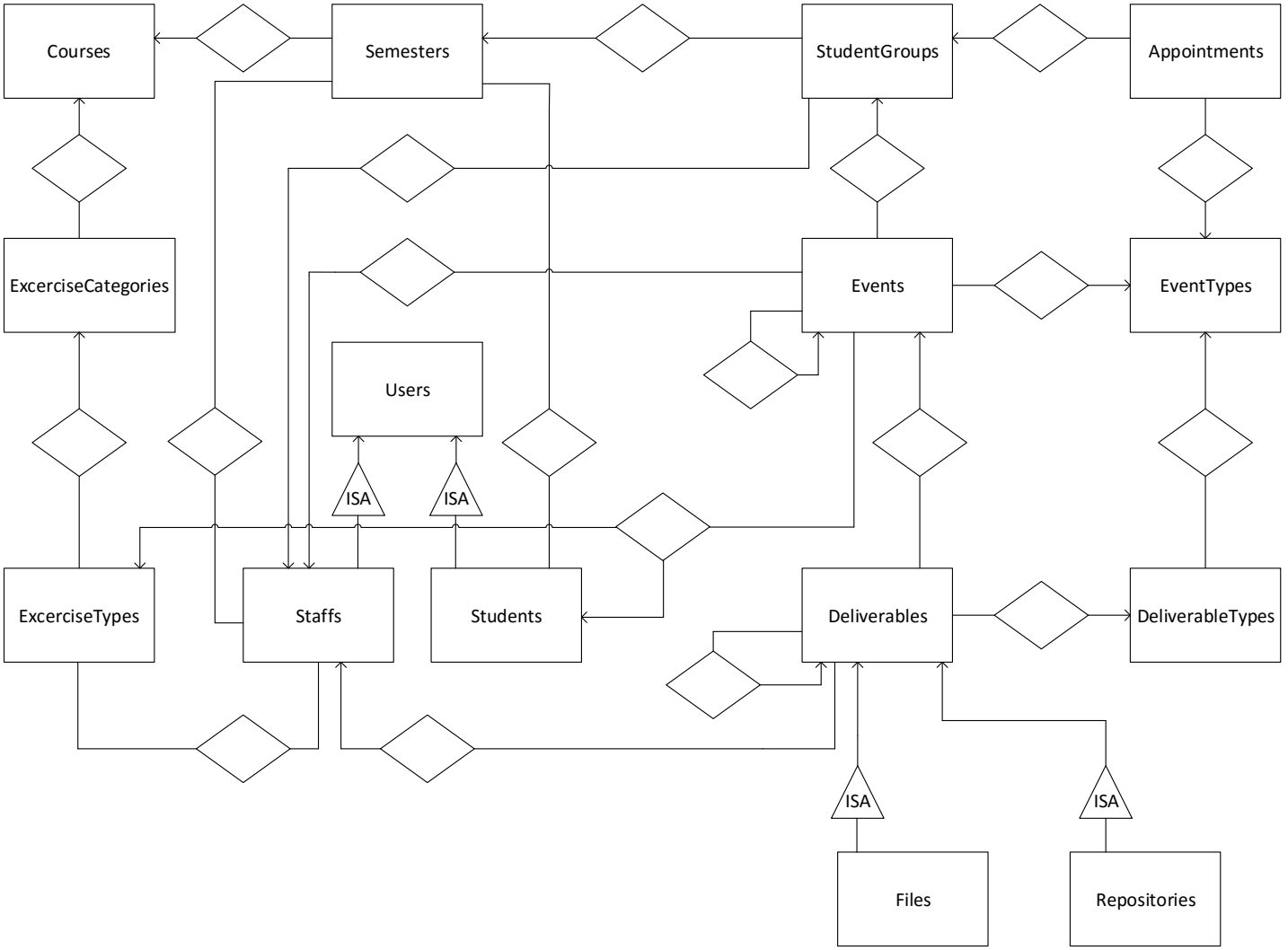

Figure 4.4. Entity–relationship model

Chapter 5

Graphic Design

During the specification phase I prioritized the features and planned how many pages are needed and what kind of components are needed to provide the features for the user (see section 2.3). Then I drew some sketches to see how these components would look like altogether (see appendix C). The next step was to decide what kind of design elements and templates I want to use to create the planned layouts.

5.1 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 [15] and checked all the popular websites, e.g., Facebook, Github, Twitter and Medium. Designmodo also have purchasable website builders, like Slides [16], but I prefer the simple design of the Bootstrap elements [71].

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. Bootstrap 4 alpha was launched in August 2015 alongside with a new side project, Official Bootstrap Themes [72]. Bootstrap Themes are purchasable redesigned collections of Bootstrap components with new components and plug-ins. Although I really like these themes, I will use the free components, because it does not worth buying any theme if I will change some parts of the included components.

5.1.1 Colors

After deciding what kind of design framework will I use, I had to choose the colors of the portal. Both the Budapest University of Technology and Economics [66] [67] and the Faculty of Electrical Engineering and Informatics [69] [70] 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 specific font should be used.

I consulted with my advisor, Sándor Gajdos and he advised me that I should not follow any of these visual identity guidelines. Following any of the strict guidelines can be a problem in the future, if someone would like to use this portal for another course that does not belong to the faculty or the university. Based on my subjective opinion I have decided to use green as the main color of the portal. I used the Google Color palette [29] to choose a nice green, changed it a bit, and I got the final color, #2a623d.

5.2 New Graphic Design

Beside the newer, modern design, I wanted to make sure that the users of the old portal can easily do the same tasks in the new portal too. The pages are separated by features and the different states of the laboratory page will be separated with tabs.

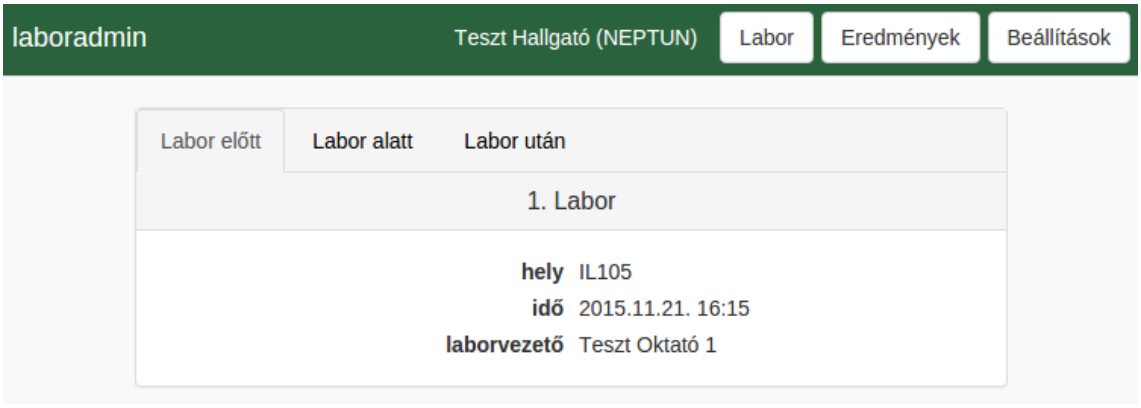

Figure 5.1. Before laboratory

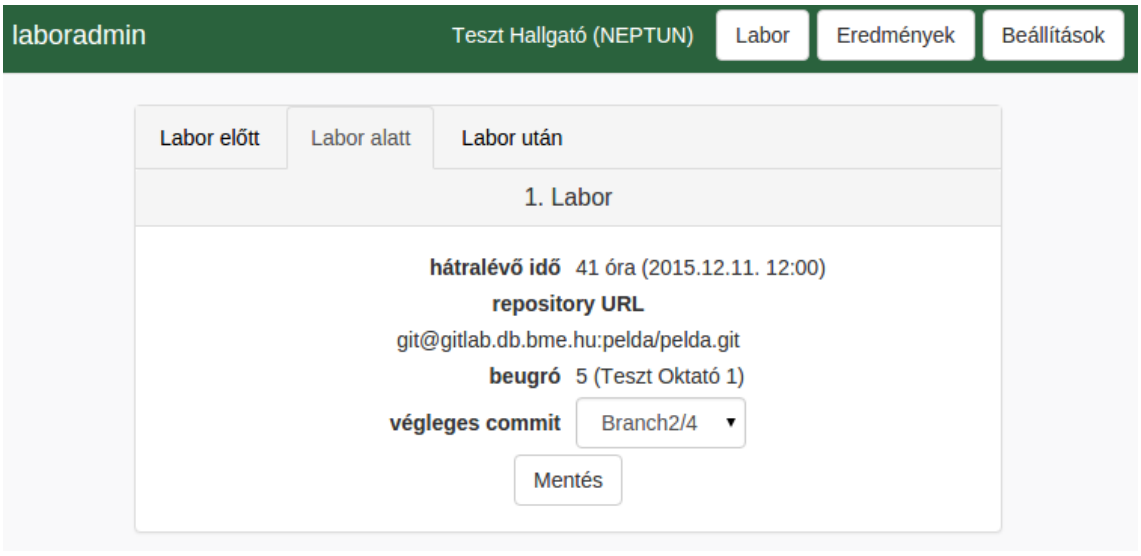

Figure 5.2. During laboratory

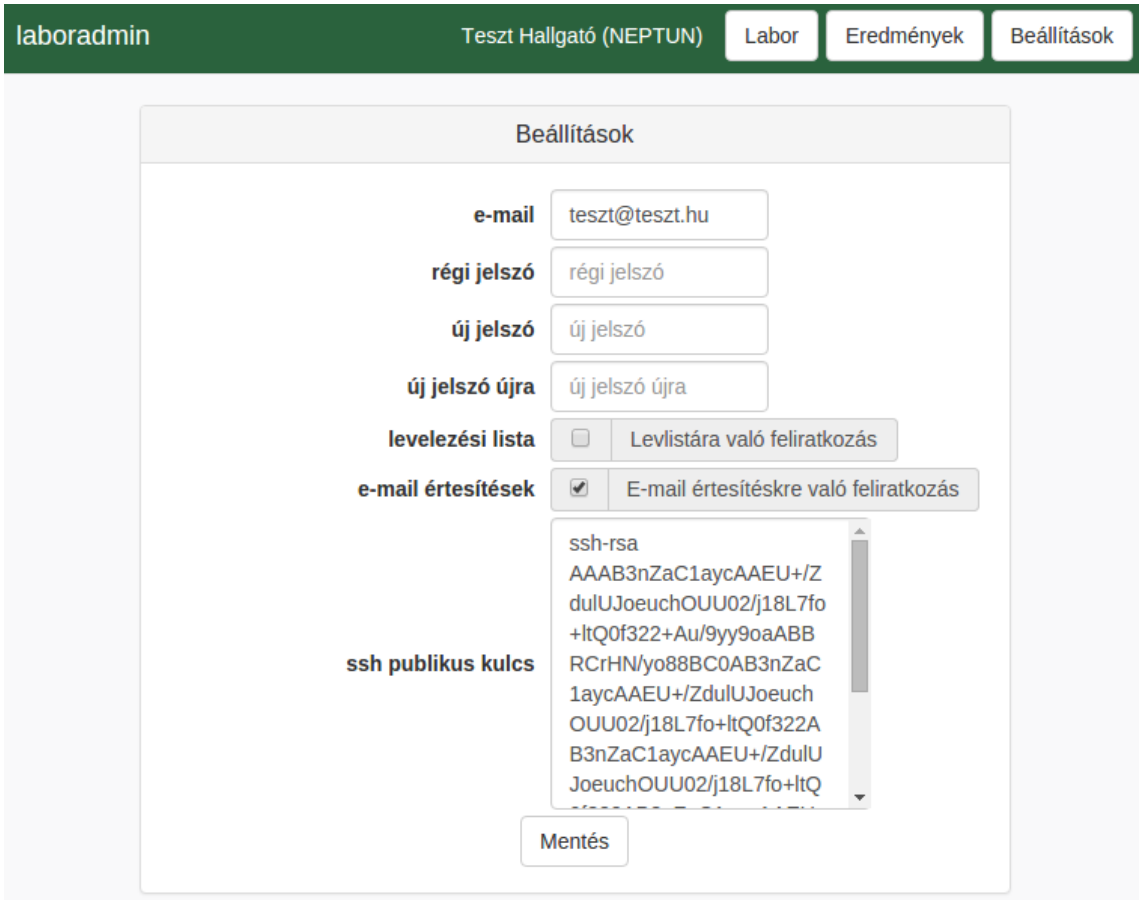

Figure 5.3. Settings

Figure 5.4. Summarized results

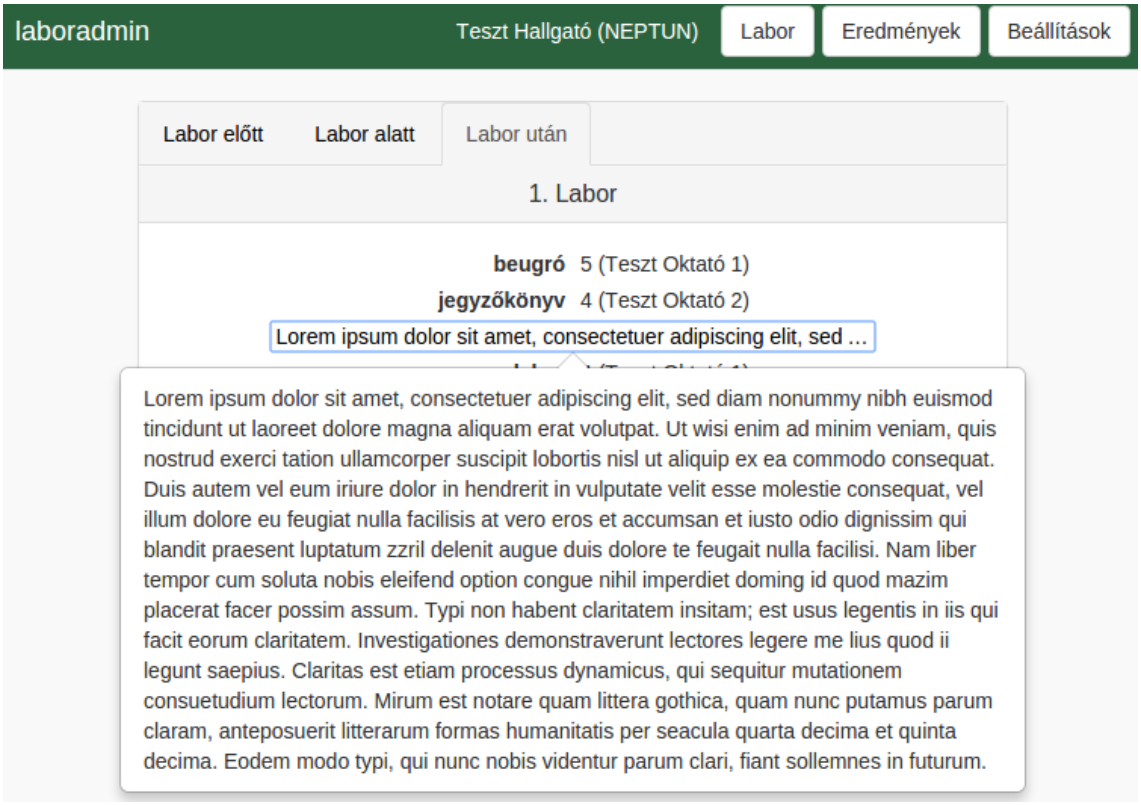

Figure 5.5. Popover for reviews.

Chapter 6

Implementation

In this chapter I introduce the used software tools and open source projects and I describe the method of implementation.

The team has decided, that the portal is an open source project. The source code is available on GitHub [78] [52]. This offers an opportunity for new students and staff to contribute to the project.

6.1 Version Control

GitHub [22] is an online Git repository service. Git is a version control system. In a new commit, it stores snapshots of the file system. If a file did not change, then Git links it to the previous version [20]. GitHub offers private and public repositories too. In this project we use public repositories and the Git Workflow [23].

6.1.1 Story Branch Pattern

When many developers work on the same project, their aim is to keep the master branch clean. A branch is a pointer to one of the commits (snapshots). The default branch is the master branch. To keep the master branch clean, developers can create a new branches. While developing a feature, the developer commits into a second branch. These changes does not affect the master branch. When the feature is finished, then the developer has to merge it into the master branch. If the master branch did not change, it will move the pointer to the new commit. At this point the new branch can be deleted, because the master branch points at the same commit [21].

If the master branch changed before, Git can do a three-way merge between the common ancestor, the last commit in the master branch, and the last commit in the second branch. Git can do this automatically, if a same part of a same file is not changed. In that case the developer can either accept one of the files or merge them by hand [21].

6.2 Used Software Tools and Open Source Projects

The front end uses the following open source projects and software tools:

- **API Blueprint:** API Blueprint [5] is an open source project, that gives me the option to write my specification in Markdown. I use it to describe the contract between the back end and the front end.
- **Bootstrap:** Bootstrap [71] is a free and open source HTML, CSS and JS framework to create responsive design. I use it as a template for the portal's design.
- **Cucumber:** Cucumber [55] is a software tool to run automated tests with user stories written in the Given When Then convention.
- **Drakov:** Drakov [51] is a mock server tool, that implements API Blueprint specifications. I use it as a mock server for the features, that are not provided by the back end yet.
- **GitHub:** GitHub [22] is an online Git repository service. The project's source code is available on GitHub.
- **Gulp:** Gulp [32] is a software tool to automatize tasks. I use it to concatenate and minimize files, transform HTML-like syntax in JavaScript files and move them to another directory and then to start the mock server.
- **Mithril:** Mithril [34] is a small JavaScript MVC framework. The front end is implemented in Mithril.
- **JSLint:** I used the JSLint [50] code verifier plugin to verify that my JavaScript codes match the coding rules [13].

This thesis and all the documents for this project were written in LaTeX [53]. LaTeX is a markup language to create scientific documents. A TeX distribution produces the PDF output.

All the diagrams were made by me in Microsft Visio 2013 [62]. Visio is a part of the Microsoft Office and is an application to create diagrams. Visio is free for students via DreamSpark [68]. For developement, I used IntelliJ IDEA [48], that is free for students.

6.3 Method of Implementation

During implementation I used HTML, CSS and JavaScript to create the web application. For deployment I used a gulp script written by me (see section 8.1).

HTML is a standard language to create websites. CSS describes the style of the HTML elements. The styles are binded to the elements within the class attribute. JavaScript is a script language to make websites interactive. Web applications use JavaScript for AJAX requests and event action handling.

HTML
As the first step I created a basic HTML page. An HTML page has a head and a body section. All the meta data belong to the head section, and anything I want to display on the page goes to the body section.

In the *head section* I included a charset option and set it to utf-8 for Unicode character encoding. I added one more important meta data, the *http-equiv="x-ua-compatible"*. I use this meta tag to force Internet Explorer to render in the highest available mode [63] [10]. This solves the problem, that Internet Explorer wanted to open the website in IE10 Compatibility Mode. I also included a title and linked the main CSS file in the head section.

In the *body section* I created an empty div with an id attribute. The pages are rendered into this div. To make sure that JavaScript can render elements into this div, I included the JavaScript code after the closing tag of this div.

CSS

As the main CSS file I concatenate the Bootstrap CSS file with my own CSS file, called *laboradmin.css*. During concatenation I had to make sure that my CSS code will be after the Bootstrap code. This is important, because in CSS the last style rules overrides the previous ones. The Bootstrap CSS file contains the basic Bootstrap component styles. In the *laboradmin.css* file some parts of the Bootstrap CSS are overwritten, to make the components to have a different appearance than the basic Bootstrap appearance, e.g., font colors, background colors and border visibility.

JavaScript

This module contains the following JavasScript files:

- the shim file, because Mithril relies on some features what are not part of the previous Internet Explorer versions,
- the Mithril framework's minimized JavaScript code,
- the Moment [64] framework's minimized JavaScript code,
  - I use this framework to calculate the time difference between the current time and the deadline.
- the Bootstrap JavaScript file, because some components require it,
- jQuery, because Bootstrap depends on jQuery, and
- my JavaScript files.

During developement for the sake of maintainability, I separated my JavaScript files based on if it is a part of the model's, the controller's or the view's code.

I use MSX in my *view* codes. To make the inline HTML-like syntax more simple, I created different widgets, and use them as HTML tags. The widgets are separated in different directories based on the page that uses the widget. To build a page the followings are needed:

- a page, that contains the menu and the panel,
- a panel, that contains all the widgets, and
- the widgets, that contain the basic HTML elements.

The MSX transformer converts this HTML-like syntax into Mithril JavaScript code during the concatenation. In the HTML-like code, I add classes for every element and ids and names for some element. These class attributes will make the element's style look equal. Every class's style is defined in the CSS files. The names and ids helps finding HTML elements for event handlers and during testing.

The *controllers* are the communication bridges between the model and the views. The controllers have helper methods to get the necessary data from the model, or send data to the model and to change the behavior of the elements, e.g. disable buttons. The view uses these helper methods to bind data or event listeners to the elements.

Because I did not want more dependencies I have decided to implement a simple JavaScript class as the *model*. The model loads the necessary data from the server and stores it. The model can also send data to the server. All data flow between the model and the server go through a marshalling module for conversion.

During deployment I concatenate all the JavaScript files to one file and minimize it. This leads to as few requests to the back end as possible and improves the performance. The website will only download one HTML file, one CSS file and one JavaScript file.

6.4 Final Version

I have uploaded the code to GitHub. The commit, that is presented in this thesis is tagged with *thesis-final2* [81].

Chapter 7

Testing

To test the application I used Cucumber with user stories for acceptance tests with Zombie as a headless browser and Istanbul for code coverage testing. I also used a mock server to get mock data during testing, because the back end modules are not finished yet.

Due the simpleness of the implemented methods, e.g., getters to provide data from the model through the controller to the view, or setters to store the downloaded data in the model, I did not create unit tests.

7.1 Acceptance Tests

An acceptance test validates that this is the software or feature the customer wanted. I wrote user stories (see subsection 2.1.2) for the features I wanted to test based on the user stories I wrote during the specification phase [1].

7.1.1 Cucumber

I used Cucumber (see subsection 2.1.2) to implement acceptance tests. I used the instructions on the Cucumber website to start the implementation [57]. A Cucumber test requires the following files:

- **Features Files:** The user stories are stored in feature files.
- **Support Files:** These will be run before each scenario to create the test environment. In my case it creates a headless browser. One example is included on the Cucumber website, but that does not work. I implemented my own support file.
- **Step Definitions:** These are the implemented feature steps. I generated the functions with a JetBrains plugin [49] and implemented the logic for every step.

7.1.2 Zombie

I wanted to use a browser to test the client's JavaScript code. My first choice was Selenium [74] with a Google Chrome browser, but the tests were slow. To solve this issue I have decided to use a browser without a user interface. I chose Zombie, because it was recommended on the Cucumber website.

Zombie [6] is lightweight framework, that implements a browser without a user interface. It also supports assertions, that makes it possible to text field comparison and HTML element existence testing easier, and pipelines to log the outgoing requests from the Zombie browser.

7.2 Code Coverage Tests

A coverage test measures which functions, statements, lines were executed [1]. I tested which user story step definitions were executed, and how many times were they executed, because some scenarios had similar steps, e.g., the 8 different of filling the password fields on the settings page.

7.2.1 Istanbul

Istanbul [47] is a code coverage software tool for JavaScript. It checks coverage for:

- *Function:* number of functions, that have been called.
- *Statement:* number of statements, that have been executed.
- *Branch:* Number of branches, that have been executed.

7.3 Automatized Tests

First follow the instructions in chapter 8, then start the mock server. Use the following command to run the Cucumber tests:

```
npm test
```

I have implemented 14 scenarios and 91 steps. Use the following command to run the Cucumber tests with code coverage:

```
npm test --coverage
```

The following features were implemented:

1. see the general information about the classes
1. see the results
2. see a list of commits and tagging a commit as final version
2. set new password, e-mail and SSH public key
3. summarized view of student's grades

As I mentioned in section 2.3, I tested the features with priority levels 1 and 2.

During the testing I was comparing the HTML elements' texts with the example data that came form the user story and the elements' existence. If a label contains a date, I used a regex for pattern matching:

```
(\d4)\.(\d2)\.(\d2)\.  (\d2):(\d2)
```

Because I cannot test features, that needs to change the database, without a real back end, I tested for outgoing requests. If there was an outgoing request for the specific URL, the test passed. This also gave me the option to test for not outgoing requests. For example, to set a new password, the user has to fill out three input fields: old password once and new password once. If any of these fields were empty, or the new password fields do not contain the same password, then the client should not send a request to the server. I wrote scenarios to all 8 possible states. These tests will change when the back end modules will be implemented.

In this phase the client has a save button on the settings page. As a future improvement in the next version, I will set the save mechanism to the fields on change events. Because of this the server only requires two parameters in the HTTP request's body: mailing list subscription and notification subscription. These parameters will always have a valid value, because the check box fields return a boolean value, either true or false.

7.4 Test Results

In behavior-driven development the developer implements first the tests, that fail. Then the developer implements the missing features, until all the tests for that feature passes. After running the tests, all scenarios and steps passed. This means that all the features, that are prescribed in the functional specification were implemented and tested. The coverage summary for the test Javascript files are almost 100%. The branch summery is only 55 %, because the lines, that contain a command to send an error did not run.

```
  Scenario: Try to set a new password incorrectly #6   # features/saveSettings.feature:50
    Given I am using laboradmin website                # features/step_definitions/saveSet
    And I am on the settings page                      # features/step_definitions/saveSet
    And I fill the email field                         # features/step_definitions/saveSet
    And I fill the ssh-key field                       # features/step_definitions/saveSet
    Given I fill the new password field                # features/step_definitions/saveSet
    And I fill the old password field                  # features/step_definitions/saveSet
    When I press the "Mentés" button                   # features/step_definitions/saveSet
    Then I could not save a new password               # features/step_definitions/saveSet

  Scenario: Try to set a new password incorrectly #7   # features/saveSettings.feature:56
    Given I am using laboradmin website                # features/step_definitions/saveSet
    And I am on the settings page                      # features/step_definitions/saveSet
    And I fill the email field                         # features/step_definitions/saveSet
    And I fill the ssh-key field                       # features/step_definitions/saveSet
    Given I fill the new password again field          # features/step_definitions/saveSet
    And I fill the old password field                  # features/step_definitions/saveSet
    When I press the "Mentés" button                   # features/step_definitions/saveSet
    Then I could not save a new password               # features/step_definitions/saveSet

14 scenarios (14 passed)
91 steps (91 passed)
0m05.109s
================================================================================
Writing coverage object [/home/lenny/thesis/laboradmin/coverage/coverage.json]
Writing coverage reports at [/home/lenny/thesis/laboradmin/coverage]
================================================================================

============================== Coverage summary ==============================
Statements   : 99.39% ( 163/164 )
Branches     : 55.56% ( 10/18 )
Functions    : 100% ( 53/53 )
Lines        : 99.39% ( 163/164 )
================================================================================
```

Figure 7.1. Part of the Tests' Output

Chapter 8

Deployment

For implementation and deployment I used Linux Mint 17.2. The commit, that is presented in this thesis is tagged with *thesis-final2* [81].

8.1 Installation Steps

Git Repository
First you need to clone the git repository. If you do not have git installed, use the following command:

```
sudo apt-get install git-all
```

Use the following command to clone the repository into a new folder. This will clone the latest commit and every commit before it, including the *thesis-final2* tagged commit. To get the tagged commit create a new branch with checkout and add the tag.

```
git clone https://github.com/lordblendi/student
cd student/
git checkout -b new_branch thesis-final2
```

Output:

```
lenny@lenny-laptop-3 ~/temp/new $ git clone https://github.com/lordblendi/student
Cloning into 'student'...
remote:  Counting objects:  5041, done.
remote:  Compressing objects:  100% (191/191), done.
remote:  Total 5041 (delta 94), reused 0 (delta 0), pack-reused 4836
Receiving objects:  100% (5041/5041), 14.21 MiB | 7.47 MiB/s, done.
Resolving deltas:  100% (1095/1095), done.
Checking connectivity...  done.
lenny@lenny-laptop-3 ~/temp/new $ cd student/
lenny@lenny-laptop-3 ~/temp/new/student $ git checkout -b new_branch
thesis-final2
Switched to a new branch 'new_branch'
```

Node
I use node package manager (npm) and the node version manager (nvm) to install software tools and node versions. Use the following command to install npm and nvm:

```
sudo apt-get install curl nodejs
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.29.0/install.sh
| bash
```

Close and reopen your terminal to start using nvm. For testing Cucumber needs node v4.2. and the mock server needs node v0.10. Use the following commands to install these:

```
sudo nvm install v4.2
sudo nvm install v0.10
```

Use the following commands to see the installed node versions:

```
nvm ls
```

Output:

```
 v0.10.41
-> v4.2.3
node -> stable (-> v4.2.3) (default)
stable -> 4.2 (-> v4.2.3) (default)
iojs -> N/A (default)
```

Use the following command to switch to node v4.2 in the current terminal:

```
nvm use 4.2
```

Use the following command to check which node version is being used in the current terminal:

```
node -v
```

Gulp
Gulp is a software tool to automatize tasks. I use it to concatenate and minimize files, transform HTML-like syntax in JavaScript files and move them to another directory. Gulp can be installed with the following command (npm will not work without using a node version):

```
npm install gulp
```

Dependencies
The git repository includes a *package.json* file, that holds the project's metadata, like: name, version, author and dependencies. Dependencies are required to run the gulp file (see section 8.3).

Use the following command to install the dependencies listed in package.json:

```
npm install
```

8.2 Apache2

I used the Apache Web Server to serve the laboradmin's files on localhost. Use the following commands to install Apache:

```
sudo apt-get update
sudo apt-get install apache2
```

To set the document root to */var/www/html* add the *laboradmin.conf* file:

```
sudo vi /etc/apache2/sites-enabled/laboradmin.conf
```

And add this:

```
<VirtualHost *:80>
# The ServerName directive sets the request scheme, hostname and port that
# the server uses to identify itself.  This is used when creating
# redirection URLs.  In the context of virtual hosts, the ServerName
# specifies what hostname must appear in the request's Host:  header to
# match this virtual host.  For the default virtual host (this file) this
# value is not decisive as it is used as a last resort host regardless.
# However, you must set it for any further virtual host explicitly.
#ServerName www.example.com
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html/laboradmin/dist

# Available loglevels:  trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host.  For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf
</VirtualHost>

# vim:  syntax=apache ts=4 sw=4 sts=4 sr noet
```

Make sure to use the same folder name *laboradmin* in */var/www/html* in the *laboradmin.conf*, because the URL is hard coded in the tests.

Now all you have to do is to create a softlink from your *student/* directory to the */var/www/html* directory. Example command:

```
sudo ln -s /home/lenny/temp/student/ /var/www/html/laboradmin
```

The */home/lenny/temp/* part should be switched to where you saved the git repository.

And change the permissions on your student folder:

```
sudo chmod -R 0755 /home/lenny/temp/student/
```

Now you have to reboot apache2:

```
sudo service apache2 restart
```

8.3 The Gulp File

The client's gulp file is added in the root directory.

First gulp uses the msx plugin, to transform the HTML-like syntax in the JavaScript files, then concatenates them. The concatenated JavaScript file is minimized and moved to the *dist* folder. I used the MSX plugin's example code for the msx transformation task [8]. Then gulp minimizes the HTML file and moves it to the *dist* folder. At last the CSS files are concatenated, minimized and the minimized file is moved to the *dist* folder.

The final three files are in the *dist* folder. These files will be downloaded from the server, when a user opens the portal in a browser.

Use the following command to run the gulp file:

```
gulp
```

Output:

```
[12:34:50] Using gulpfile ~/thesis/laboradmin/gulpfile.js
[12:34:50] Starting 'default'...
[12:34:50] Starting 'browserify'...
[12:34:50] Starting 'msx'...
[12:34:50] Finished 'msx' after 30 ms
[12:34:50] Starting 'html'...
[12:34:50] Finished 'html' after 7.27 ms
[12:34:50] Starting 'css'...
[12:34:50] Finished 'css' after 5.4 ms
[12:34:50] Finished 'browserify' after 48 ms
[12:34:50] Finished 'default' after 49 ms
```

8.4 Mock Server Usage

Before opening the laboradmin in a browser, you need to start the drakov mock server. Use the following commands to switch to node v0.10 and install drakov and his dependencies:

```
sudo apt-get install make g++
nvm use 0.10
npm install -g drakov
```

Open a new terminal in the *student* folder and use the following commands to start drakov:

```
drakov -f "api-blueprint/*.md" --autoOptions
```

The *--autoOptions* flag lets the server to answer preflight OPTIONS requests. If a browser sets a header, e.g., content-type in the POST request, then the browser will send a preflight OPTIONS request instead of a post request.

Output:

```
 lenny@lenny-laptop-3 ~/temp/student $ drakov -f "api-blueprint/*.md"
--autoOptions
DRAKOV STARTED
[LOG] Setup Route:  GET /resources Get resources
[LOG] Setup Route:  OPTIONS /resources
[LOG] Setup Route:  GET /student/:userid/general Get student general
informations
```

```
[LOG] Setup Route:  GET /student/:userid/results Get student results
[LOG] Setup Route:  GET /student/:userid/laboratory Get student laboratory
details example for full lab
[LOG] Setup Route:  GET /student/:userid/settings Get student settings
[LOG] Setup Route:  OPTIONS /student/:userid/general
[LOG] Setup Route:  OPTIONS /student/:userid/results
[LOG] Setup Route:  OPTIONS /student/:userid/laboratory
[LOG] Setup Route:  OPTIONS /student/:userid/settings
[LOG] Setup Route:  POST /student/:userid/setfinalcommit Post new final
commit for student
[LOG] Setup Route:  POST /student/:userid/setsettings Post new settings
for student
[LOG] Setup Route:  OPTIONS /student/:userid/setfinalcommit
[LOG] Setup Route:  OPTIONS /student/:userid/setsettings
Drakov 0.1.12 Listening on port 3000
```

And now you can open the following page in a web browser:

`http://localhost/laboradmin/dist/`

Chapter 9

Conclusion

In this chapter I summarize my work and I introduce the future plans.

9.1 Summary of my Work

I achieved the followings:

- wrote the new portal's functional specification,
- compared three JavaScript frameworks and chose one for implementation,
- created the front end system design based on the behavior of the chosen,
- created the database schema with József Marton,
- created a high level system design with Bence Golda,
- designed the communication between the client and the server,
- implemented the front end student module,
- tested the student module with acceptance tests, and
- documented the deployment process.

9.2 Future Work

After this thesis I will continue to work on the front end modules and the system's design. Although I have created a database scheme and the system design, they are still in development. The front end modules will be implemented first, and then I will work more on the graphic design if the team is not satisfied with it. Because the project is still under development, it is possible, that the first official version, what is going to be used in the Software Laboratory 5 course from February 2016, will be different from the one presented in this thesis.

Acknowledgements

First, I would like to thank Bence Golda for all his help and advice and Sándor Gajdos for his mentorship, advice and continued involvement throughout this project.

My deep gratitude for József Marton for offering me his office and granting me the academic support and advice throughout this challenging endeavor.

In conclusion, I would like to acknowledge my family, my boyfriend, Lars Vandenbergh, and my friends Kálmán Tarnay, Gábor Váradi, Zoltán Czirkos, Dalma Babinszki, Dániel Stein, Ferenc and Anni Karpati for their unconditional love, encouragement, and continuous commitment to me both throughout my undergraduate studies and the preparation of this thesis.

List of Figures

List of Tables

Bibliography

[1] Studied in Software Engineering and Software Laboratory 3 courses.

[2] Studied in Developing Data-driven Applications course.

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

[4] Suprotim Agarwal. The MVC Pattern and ASP.NET MVC - Back to Basics. http://www.dotnetcurry.com/aspnet-mvc/922/aspnet-mvc-pattern-tutorial-fundamentals. Accessed: 2015-12-06.

[5] apiblueprint. API Blueprint - API Documentation with powerful tooling. https://apiblueprint.org/. Accessed: 2015-12-01.

[6] assaf. Zombie by assaf. http://zombie.js.org/. Accessed: 2015-12-09.

[7] Jonny Buchanan. MSX. https://github.com/insin/msx. Accessed: 2015-10-20.

[8] Jonny Buchanan. MSX Gulp Example. https://github.com/insin/msx/blob/master/gulpfile.js. Accessed: 2015-12-02.

[9] Karn Bulsuk. An Introduction to 5-why | Karn G. Bulsuk: Full Speed Ahead. http://www.bulsuk.com/2009/03/5-why-finding-root-causes.html. Accessed: 2015-12-06.

[10] Mathias Bynens and Hans Christian Reinl. html5-boilerplate/html.md at 5.2.0 · h5bp/html5-boilerplate. https://github.com/h5bp/html5-boilerplate/blob/5.2.0/dist/doc/html.md. Accessed: 2015-11-30.

[11] The Daring Fireball Company. Daring Fireball: Markdown. http://daringfireball.net/projects/markdown/. Accessed: 2015-12-01.

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

[13] Refsnes Data. JavaScript Style Guide. http://www.w3schools.com/js/js_conventions.asp. Accessed: 2015-12-09.

[14] Josh Davis. The Difference Between TDD and BDD - Josh Davis. https://joshldavis.com/2013/05/27/difference-between-tdd-and-bdd/. Accessed: 2015-12-06.

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

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

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

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

[19] Sándor Gajdos. *Adatbázisok*. A-SzínVonal 2000 Nyomdaipari Kft, 2015. Language: Hungarian.

[20] Git. Git - Git Basics. https://git-scm.com/book/en/v2/Getting-Started-Git-Basics. Accessed: 2015-12-02.

[21] GitHub. Git - Basic Branching and Merging. https://git-scm.com/book/en/v1/Git-Branching-Basic-Branching-and-Merging. Accessed: 2015-12-02.

[22] GitHub. GitHub. https://github.com/. Accessed: 2015-12-02.

[23] GitHub. Understanding the GitHub Flow · GitHub Guides. https://guides.github.com/introduction/flow/index.html. Accessed: 2015-12-02.

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

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

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

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

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

[29] Google. Color - Style - Google design guidelines. https://www.google.com/design/spec/style/color.html#color-color-palette. Accessed: 2015-11-25.

[30] Google. Google Apps update alerts: End of support for Internet Explorer 9. http://googleappsupdates.blogspot.hu/2013/11/end-of-support-for-internet-explorer-9.html. Accessed: 2015-11-30.

[31] Google. Supported browsers - Google Apps Administrator Help. https://support.google.com/a/answer/33864?hl=en. Accessed: 2015-11-30.

[32] Gulp. Gulp. https://github.com/gulpjs/gulp. Accessed: 2015-12-06.

[33] Leo Horie. Getting Started - Mithril. https://lhorie.github.io/mithril/getting-started.html. Accessed: 2015-12-06.

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

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

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

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

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

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

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

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

[42] Leo Horie. Tools - Mithril. http://lhorie.github.io/mithril/tools.html. Accessed: 2015-11-30.

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

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

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

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

[47] Istanbul. gotwarlost/istanbul. https://github.com/gotwarlost/istanbul. Accessed: 2015-12-09.

[48] JetBrains. IntelliJ IDEA — The Most Intelligent Java IDE. https://www.jetbrains.com/idea/. Accessed: 2015-12-09.

[49] JetBrains. JetBrains Plugin Repository :: Cucumber.js. https://plugins.jetbrains.com/plugin/7418?pr=phpStorm. Accessed: 2015-12-09.

[50] JetBrains. JSLint. https://www.jetbrains.com/idea/help/jslint.html. Accessed: 2015-12-09.

[51] Yakov Khalinsky and Marcelo Garcia de Oliveira. drakov. https://www.npmjs.com/package/drakov. Accessed: 2015-12-01.

[52] BME Database Laboratory. bme-db-lab/szglab5-backend. https://github.com/bme-db-lab/szglab5-backend. Accessed: 2015-12-09.

[53] LaTeX. LaTeX – A document preparation system. https://www.latex-project.org/. Accessed: 2015-12-06.

[54] Cucumber Ltd. Cucumber. https://cucumber.io/docs/reference#scenario. Accessed: 2015-11-23.

[55] Cucumber Ltd. Cucumber. https://cucumber.io/. Accessed: 2015-11-23.

[56] Cucumber Ltd. Cucumber. https://cucumber.io/docs/reference#gherkin. Accessed: 2015-11-23.

[57] Cucumber Ltd. cucumber/cucumber-js. https://github.com/cucumber/cucumber-js. Accessed: 2015-12-09.

[58] Cucumber Ltd. Given When Then · cucumber/cucumber Wiki. https://github.com/cucumber/cucumber/wiki/Given-When-Then. Accessed: 2015-12-06.

[59] Robert C. Martin. The Truth about BDD - Clean Coder. https://sites.google.com/site/unclebobconsultingllc/the-truth-about-bdd. Accessed: 2015-12-06.

[60] Bertrand Meyer. *Touch of Class: Learning to Program Well with Objects and Contracts*. Springer, 2009. Corrected printing 2013.

[61] 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.

[62] Microsoft. Professional Flow Chart Diagram Software | Microsoft Visio. https://products.office.com/en-US/visio?legRedir=true&CorrelationId=95b14d1b-87e4-46b6-b02a-cba7af78dfd6&omkt=en-US. Accessed: 2015-12-06.

[63] Microsoft. Specifying legacy document modes (Internet Explorer). https://msdn.microsoft.com/en-us/library/jj676915.aspx. Accessed: 2015-11-30.

[64] Moment.js. Moment.js. http://momentjs.com/. Accessed: 2015-12-06.

[65] Dan North. Introducing BDD. http://dannorth.net/introducing-bdd/. Accessed: 2015-12-06.

[66] 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.

[67] 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.

[68] Budapest University of Technology and Economics (BME). DreamSpark (MSDNAA) @ BME. http://msdnaa.bme.hu/. Accessed: 2015-12-06.

[69] 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.

[70] 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.

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

[72] Mark Otto and Jacob Thornton. Bootstrap Themes. http://themes.getbootstrap.com/collections/all. Accessed: 2015-10-30.

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

[74] Selenium. Selenium WebDriver. http://www.seleniumhq.org/projects/webdriver/. Accessed: 2015-12-09.

[75] Sibboleth. Sibboleth. https://shibboleth.net/. Accessed: 2015-12-10.

[76] StatCounter. Top 12 Desktop Browser Versions in Hungary from Nov 2014 to Nov 2015 | StatCounter Global Stats. http://gs.statcounter.com/#desktop-browser_version-HU-monthly-201411-201511-bar. Accessed: 2015-12-10.

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

[78] Nóra Szepes. lordblendi/student. https://github.com/lordblendi/student. Accessed: 2015-12-09.

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

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

[81] Nóra Szepes. Release Final version for thesis · lordblendi/student · GitHub. https://github.com/lordblendi/student/releases/tag/thesis-final2. Accessed: 2015-12-10.

[82] Nóra Szepes. student/api-blueprint at master · lordblendi/student. https://github.com/lordblendi/student/tree/master/api-blueprint. Accessed: 2015-12-09.

[83] Nóra Szepes. Eiffel. Technical report, Budapest, 2014. Language: Hungarian. This document is not publicly available.

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

[85] 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.

[86] 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 [18], and an online explanatory dictionary [17].

- **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 attends the laboratories and solves a list of tasks.

Appendix B

User Stories

Feature: Student module

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

Before laboratory:

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 information 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

During laboratory:

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

During the meetings the team always comes up with new ideas, and these sketches present only the first version of the graphic design. The sketches were drawn by hand and only three of them were digitalized, because the graphic design is still under development.

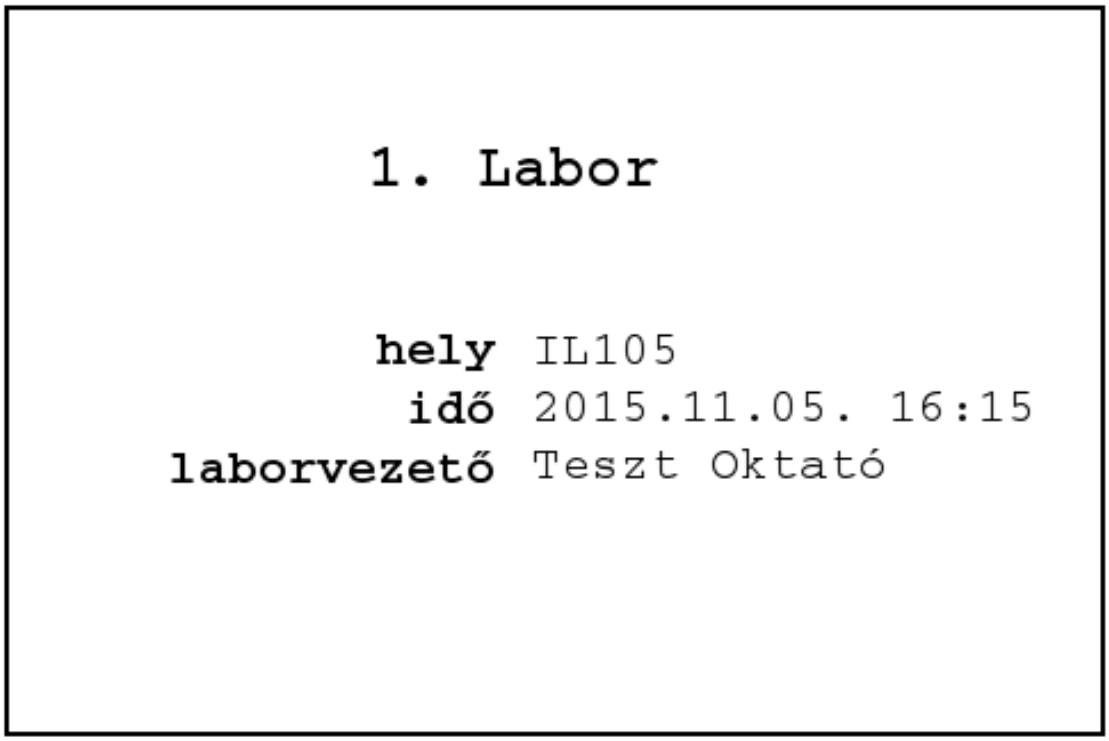

Figure C.0.1. Laboratory page sketch, before lab

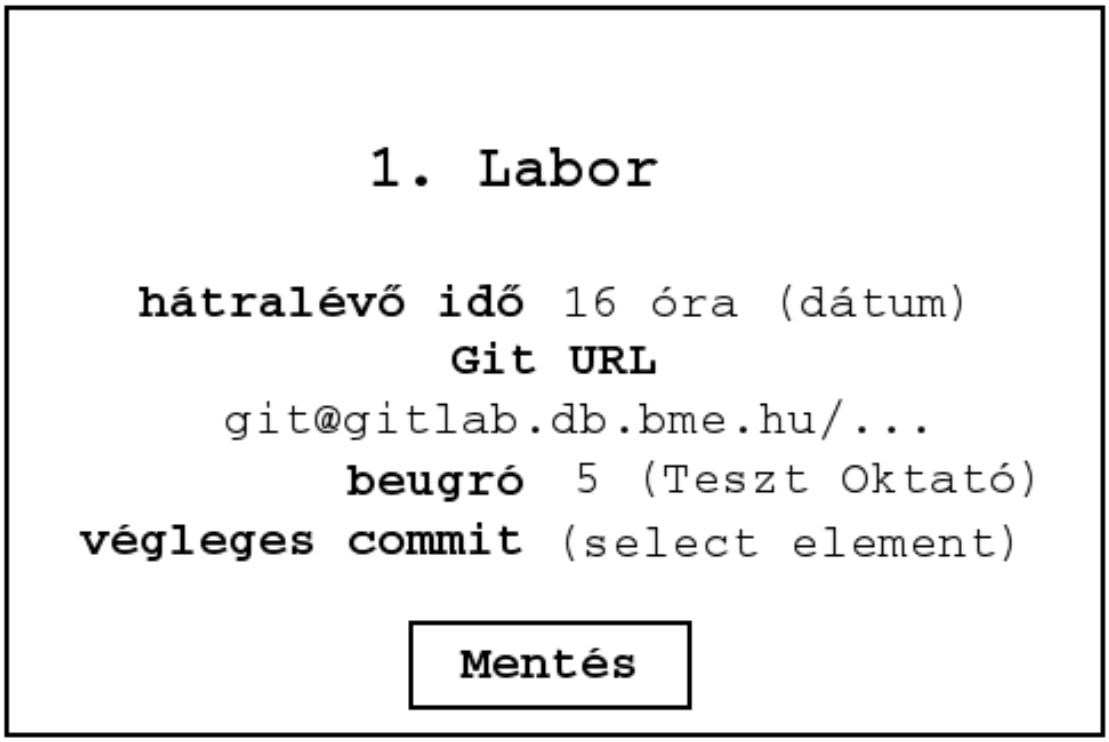

Figure C.0.2. Laboratory page sketch, during lab

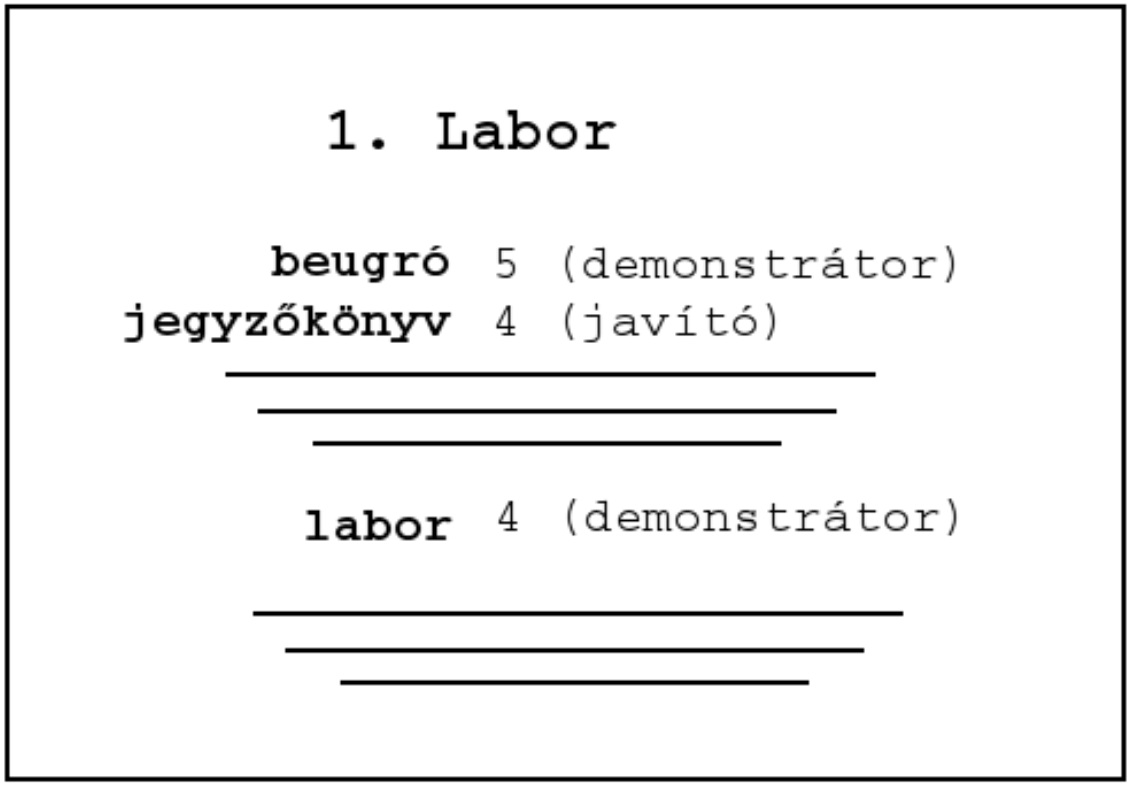

Figure C.0.3. Laboratory page sketch, after lab

Appendix D

Attribute List

About 50 percent of the attribute list was created by me and 50 percent of it was created by József Marton.

- Appointments
  - attributes
    - id - number
    - date - datetime
    - location - text
  - foreign keys
    - eventtype - EventTypes
    - studentgroup - StudentGroups
- Courses
  - attributes
    - id - number
    - name - text
    - codename - text
- Deliverables
  - attributes
    - id - number
    - deadline - datetime
    - submitteddate - datetime
    - grade - number - 1-5
  - foreign keys
    - deliverabletype - DeliverableTypes
    - evaluator - Staffs
    - related - Deliverables
  - Deliverables/Repositories
    - attributes
      - url - text - type (svn, git) depends on the url

            - commit - text - chosen final commit
    - Deliverables/Files
        - attributes
            - size - number
            - sha256sum - text
            - filename - text

- DeliverableTypes
    - attributes
        - id - number
        - type - text - file / repository
        - deadline - datetime operator - +4 days from the event's date (oracle interval)
        - description - text - for example: report
    - foreign keys
        - eventtype - EventTypes

- RegisteredStaffs
    - foreign keys
        - evaluator - Staffs
        - exercisetype - ExcerciseTypes

- Events
    - attributes
        - id - number
        - date - datetime
        - location - text
        - number - number - 1-5
        - title - text - DBMS, SQL, JDBC, X*, SOA
        - attempt - number - starting number: 1
        - shortdescription - text - generated
    - foreign keys
        - related - Events
        - eventtype - EventTypes
        - exercisetype - ExerciseTypes
        - demonstrator - Staffs
        - student - Students
        - studentgroup - StudentGroups

- EventTypes
    - attributes
        - id - number
        - title - text
        - number - number

- ExerciseCategories
  - attributes
    - id - number
    - type - text - SQL, DBMS, SOA, JDBC, XML
  - foreign keys
    - course - Courses
- ExerciseTypes
  - attributes
    - id - number
    - name - text - for example: Car Rental
    - shortname - text - for example: AUTO
    - exerciseid - number - for example: 22
    - codename - text - generated, for example: 22-AUTO
    - language - text
  - foreign keys
    - exercisecategory - ExerciseCategories
- RegisteredStaffs
  - attributes
    - id - number
    - isadmin - boolean
    - isdemonstrator - boolean
  - foreign keys
    - staff - Staffs
    - semester - Semesters
- RegisteredStudents
  - attributes
    - id - number
    - neptunsubjectcode - text
    - neptuncoursecode - text
  - foreign keys
    - student - Students
    - semester - Semesters
- Semesters
  - attributes
    - id - number
    - academicyear - number
    - academicterm - number
    - description - text - generated
  - foreign keys

        * course - Courses

- StudentGroups
  - attributes
    * id - number
    * name - text
    * language - text
  - foreign keys
    * demonstrator - Staffs
    * semester - Semesters

- Users
  - attributes
    * id - number
    * givenname - text
    * surname - text
    * title - text
    * displayname - text - generated
    * loginname - text, unique
    * eppn - text - Sibboleth
    * email - text, unique
    * sshpubliykey - text
    * password - text
  - Users/Students
    * attributes
      · neptun - text
      · university - text
  - Users/Staff