Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Information Technology

Department of Software Engineering

Master's thesis

Tool for authoring of HTML5 presentations with capabilities of real-time users' interactions.

Bc. Petr Mikota

Supervisor: Doc. Ing. Tomáš Vitvar, Ph.D.

4th May 2012

Acknowledgements

I would like to sincerely thank my supervisor Tomáš Vitvar for provided guidance, overall technology insight, helpful comments and new ideas throughout the thesis work.

I'm also thankful to my family and friends for unconditional support and facing difficulties on making my education foremost priority.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on 4th May 2012 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Petr Mikota. *Tool for authoring of HTML5 presentations with capabilities of real-time users' interactions.: Master's thesis.* Czech Republic: Czech Technical University in Prague, Faculty of Information Technology, 2012.

Abstract

This thesis presents design and implementation of environment for creating, managing and hosting of HTML5 presentations with backend server written in Node.js. Node.js is coming out as an emerging JavaScript server-side technology for creating efficient and scalable network applications with high performance and yet low memory consumption. The framework also provides ways for users to interact with presentation and it presents a feedback to a lecturer.

Keywords HTML5, Node.js, presentations, social networks

Abstrakt

Tato práce pojednává o návrhu a implementaci nástroje pro tvorbu, správu a hostování prezentací vytvořených v technologii HTML5. Součástí nástroje je server napsaný v Node.js, což je server využívající jazyk JavaScript na straně serveru pro tvorbu výkonných a škálovatelných aplikací. Nástroj zároveň umožňuje interakci uživatelů se samotnou prezentací a poskytuje zpětnou vazbu přednášejícímu.

Klíčová slova HTML5, Node.js, prezentace, sociální sítě

Contents

Introduction **1**

1 Analysis **7**
1.1 Objectives . . . 7
1.2 Requirements . . . 7
1.3 Definitions . . . 9
1.4 Mockup design . . . 12
1.5 Related work . . . 14

2 Principles and Technologies **17**
2.1 Web 2.0 . . . 18
2.2 REST . . . 19
2.3 Cloud . . . 21
2.4 HTML5 . . . 22
2.5 JavaScript . . . 23
2.6 Application Server . . . 27
2.7 Database . . . 32

3 Design and Implementation **37**
3.1 Architecture Design . . . 37
3.2 Libraries . . . 38
3.3 Server . . . 39
3.4 Client . . . 46
3.5 Portal . . . 51

4 Evaluation **55**
4.1 Requirements . . . 55
4.2 Integration . . . 57
4.3 Project License and Hosting . . . 58

Conclusion **59**

Bibliography **61**

A Acronyms **63**

B Installation **65**

C Screenshots **67**

D Contents of enclosed CD **69**

List of Figures

0.1 Humla architecture overview 5

1.1 User interactions with Humla 10
1.2 Mockup of Portal application 12
1.3 Mockup of comments on slides 14

2.1 Protocols, data collected on March 11th, 2012 from `http://www.programmableweb.com/apis/directory/` 20
2.2 Architecture of Node.js (Courtesy: [5]) 28
2.3 CAP Theorem and visual guide to NoSQL (Courtesy: [10]) 33

3.1 Full application stack 38
3.2 Activity diagram of server execution 40
3.3 Humla REST Endpoints 44
3.4 Humla client architecture 46
3.5 Humla integration of web APIs 50
3.6 Sequence diagram of OpenID authentication (Courtesy: [1]) 52

C.1 Humla Portal Application 67
C.2 Humla Client Application 68

List of Tables

3.1 Extension Methods 49

4.1 Supported browsers 56

Introduction

One of the main characteristics of current era is the vast amount of data produced in every single second. If we desire to provide those data to somebody, we need to analyze, process and conclude the right answers, which are complete yet simplified enough to be presented to a man and lead him to his own goals. With the raise of social networking sites, like Facebook or Google+, the number of human interactions has increased significantly and as a consequence the Bacon's number applied on all humans' relations has decreased[1]. One single person is able to give away information to 300 people he has listed as his "friends" and he has a good chance that majority will get that information. One could say that new technologies allow one person to have bigger one-to-many impact that he could have before 20 years. But what about school lectures? Has there been any improvement lately?

Not that far in the history the only way to give a lecture was with white chalk on the blackboard, from which students would remember, for teacher inaudible, scraping sound of sliding chalk. With information technologies the abilities to give a presentation have increased significantly and in many of lectures is the spoken word supported with showing slides with data-projector. This development provides not only relief for student's ears, but it also means a big challenge for new methods of education.

So what is the main goal of someone giving a lecture? He wants to deliver information to his listeners and for that he needs to attract their attention. Apart from choosing correct words and have a great presenting abilities, he has to react to student's feedback even if it's only from their facial reactions. Standard desktop slides have no ability to track any feedback. They are only static entity, which don't make use of any interaction from listeners.

And this is exactly the problem that is solved in following chapters. The focus is on improvements of current presentation possibilities and also on work with feedback from listeners since majority of currently used tools doesn't actually count with any feedback at all.

[1] Kevin Bacon defines Bacon's number at http://findthebacon.com/ as average actor-to-actor distance through common movies to Kevin Bacon.

Motivation

Let's imagine one example situation. A teacher is giving a lecture about some programming related subject and he is trying to educate all his 150 students. As he is going through his lecture he has noticed three different students. First student writes everything down to his notes. He is focused whole lecture and from time to time he asks various topic related questions. He even found one teacher's error. Second student, somehow average, is listening the whole presentation, but he is struggling as his mind keeps slipping away. He makes few notes from what he thinks is interesting or useful. Third student occasionally pays an attention to the lecture and when he finally does he just takes few notes from what the teacher just said. He is that exact type that keeps all his actual studying for home. At home he reads all slides and other sources and he is trying to understand all those things only by himself.

Teacher's goal is to figure out the way how to teach them all. He needs something that would help the first student with his questions and it would provide him with extended information. The second student would welcome different interesting ways of interacting with presentation to keep his mind focused. And the third one would be happy with a system where he could ask questions and interact with the presentation even after the lecture itself is over. For all that they would need a web application, which is fast and responsive along with support for large number of simultaneous users, but also which is easy to use and doesn't impose any extra requirements on users.

Project

The objective of this thesis is to create a presentation environment for managing slides with social interaction possibilities. Humla, a front-end framework originally created by doc. Ing. Tomáš Vitvar, PhD.[2], is extended with server-side part and further extensions. The name of the project remains **Humla**[3] and as that it is reffered in further chapters. Tomáš Vitvar uses Humla as a tool to present HTML5 slides at two main subjects in field Software and Web Engineering. Those subjects are *Web 2.0* (MI-W20) and *Middleware and Web Services* (MI-MDW). One of the goals is to make a framework for even wider support of these subjects and for wider use at the Czech Technical University in Prague.

This thesis is a part of a larger project maintained by a group of 3 people including myself. Each of us aims at different goals and implements different part of the whole project. Vladimír Říha is aiming at faceted browsing of studying materials; Vojtěch Smrček is improving presentation possibilities with HTML5 technology. Each of us had tasks as shown on the Figure 0.1.

[2]Original front-end version can be found at https://github.com/tomvit/humla
[3]The name Humla is derived from Nepal district called Humla.

List of functionalities which original Humla provides:

- **Humla Core** — browsing through presentations, definition of presentation structure, configurable loading system of core files and extensions,
- **Default styling** — default CSS stylesheet template,
- **View modes** — support for 4 different view modes and switching among them using keyboard shortcuts,
- **Client-side extensions system** — provides user extensions and some extensions are already incorporated (Github, Google Drawings etc.),
- **Syntax highlighter** — highlighting source code's syntax.

Complete list of functionalities I've developed:

- **Server Core.** I've designed architecture and implemented the server side application with minor help from Vladimír Říha. This server core can also be used as a module in other applications.
- **Portal.** I've designed and developed Humla Portal application for manipulation with presentations (browsing, editing etc.).
- **Comments extension.** As a part of managing social interactions between lecturer and listeners, I've implemented both client and server side extension for posting comments under single slide.
- **Likes extension.** Every logged user can like/dislike any slide to provide feedback information to a lecturer.
- **Tests.** The creator of the presentation can add simple yes/no tests to the presentation using Tests extension.
- **OpenID login.** I've implemented core functionality to both server and client to authenticate user using OpenID protocol.
- **Menu interface.** I've implemented on-slide Menu with ability to show menu items from extensions.
- **Social networks integration.** Every user can share presentation on the three most common social networks (Facebook, Twitter, Google+).

Thesis Structure

This thesis is divided into five different chapters. Chapters are sorted in the same order as the development phases.

Chapter 1 describes requirements, system analysis and user interface design.

Chapter 2 describes a survey of suitable technologies regarding following implementation.

Chapter 3 describes implementation, structures and different framework parts.

Chapter 4 describes fulfilling of previously stated requirements and tests developed framework with different setups under different circumstances.

Chapter Conclusion makes final statement, summarizes work done and makes recommendation for future work and improvements.

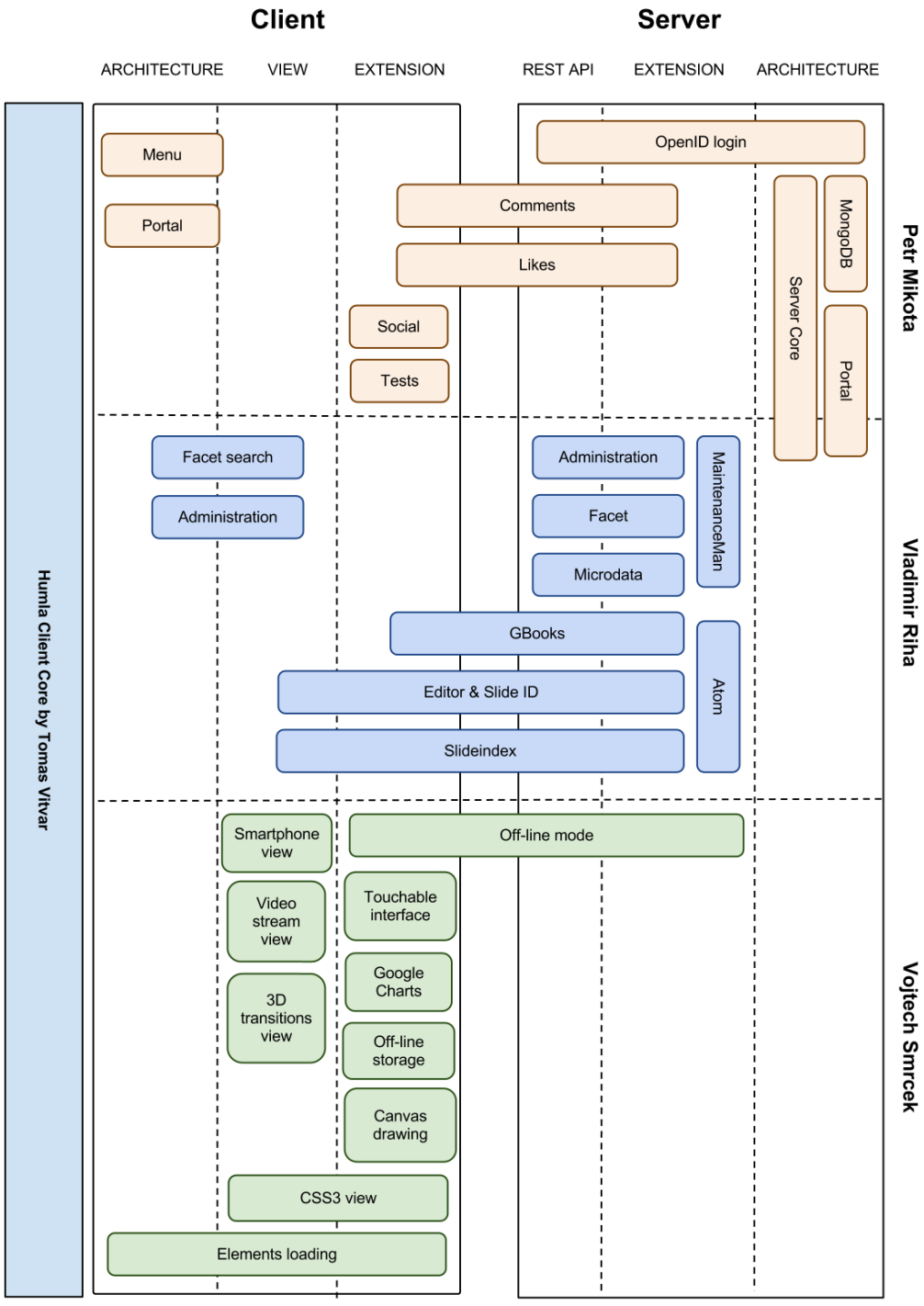

Figure 0.1: Humla architecture overview

CHAPTER 1

Analysis

This chapter is devoted to the definitions and system requirements for design of Humla presentation environment. System design follows user interface design and finally, it discusses research of related works to compare and evaluate efforts in making new presentation environment.

1.1 Objectives

The first goal is to create presentation environment that is based on original Client-side application Humla and to extend it with server application for hosting presentations and ability to provide further information about presentations.

The second goal of this thesis is to enrich both server-side and client-side application with capabilities of real-time users' interactions. The social element is based on possibility to comment and rate each slide in similar way as it's already implemented on users' posts on social websites. This goal also introduces ways of getting feedback from listeners using tests on slides. The goal is to make all those interaction easier for both lecturers and listeners.

The third goal is to make Humla easy extendable with users' extensions and to make Humla able to deploy in various situations and environments.

1.2 Requirements

The main requirement for Humla presentation environment is ability to show presentation itself. Since Humla uses quite lot of advanced features it is appropriate to make a requirements analysis. Requirements analysis is usually divided between two parts, Non-functional and Functional requirements. The first stands for requirements that are laid as conditions to system whereas the latter specifies requirements, that define specific behavior or functions.

Note again that since Humla is developed as a collaboration of three students, in the following list are mostly requirements that I'm responsible for or that I've participated on.

1.2.1 Non-functional requirements

Platform Independent. Humla can run on different most used platforms. The server part runs on all major operating systems (Linux, Windows and Mac OS). The Front-end application works in all major browsers that adopt HTML5 specification.

Various Devices Support. Humla can be used on mobile devices (tablets, mobile phones etc.) as well as on personal computers.

Used Technologies. Humla runs on Node.js (JavaScript based server-side technology) and it extends the original Humla project with new extensions as well as core modifications. The front end application uses HTML5 since most of current smart-phones, tablets and PCs are capable of displaying HTML5 web pages.

RESTful API. Humla provides access to 3rd party applications through web services using RESTful technology. This enables other applications to get and use information about lectures.

1.2.2 Functional requirements

Managing courses and presentations. Lecturer can create course with one or more lectures. On those lectures Humla provides way to fill additional information (e.g. abstract) and select if the presentation is visible to other users or not. After the lecture is created Humla provides way to select and edit desired lecture details.

Browsing through lectures. User can browse and filter all courses, select desired course and then choose desired lecture from that course. Humla provides further information about lecture (abstract and contents). Humla is open source and thus it's not intention to restrict access to presentations. All presentations stored in Humla are public.

Tests in presentations. Lecturer can add a simple quiz with yes/no questions into the presentation. Test usually takes place at the end of the presentation or at the end of some significant chapter. After the listener finishes the test Humla will show the test results. The purpose of those tests is to refresh students' memory on what they've learned in previous parts of the lecture.

Menu in presentations. During presentation Humla provides the way to display menu in which the user can change different views on presentation and view comments or likes/dislikes. Menu also provides an interface for extensions to add further menu items.

Adding Comments on slides. Any authenticated user can add comments on a single slide and read comments from other users.

Adding Likes on slides. Any authenticated user can provide interest by *"liking"* or *"disliking"* any slide. Humla displays statistics of likes/dislikes from other users for each slide.

Administration. Humla Server allows simple configuration. User can configure server setups and the list of enabled extensions.

Social networks integration. Humla provides way to share presentations on a widely used social networks such as *Facebook* or *Twitter*.

Different sizes of presentations. Humla supports wide range of presentation sizes from one-to-one presentation to presentations for huge audiences. There should be no problem to interact with many users at the same time and everybody should have the same user experience.

1.3 Definitions

User roles that appear in further parts of this thesis are:

Lecturer presents presentation to one or more listeners using various types of supplementary tools (e.g. slides, live examples, blackboard writing).

Listener listens to a lecture, asks further questions, and usually takes notes, either to his paper notebook or his laptop.

Note that those are only roles and in real situation listeners also may create and present slides before the whole class (for example to show the results of their research assignments).

The lecture materials are also used by listeners for study purposes. Listeners try to understand things that didn't make it the first time, and they possibly have questions which they can usually ask on the following lecture.

Other desktop tools for creating presentations are Microsoft PowerPoint, or Impress from LibreOffice[1]. Some presentations may be made in LaTeX and there may be some innovative approaches as replacement of black board with writing by hand on tablets.

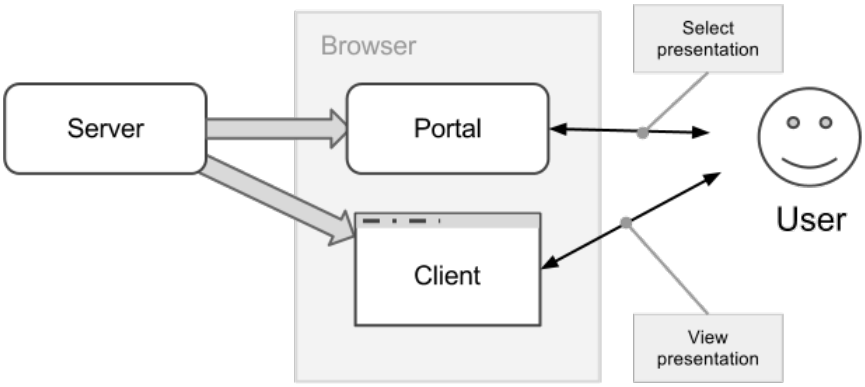

Figure 1.1: User interactions with Humla

1.3.1 Architecture

Humla acts in 3 different roles:

Application. In this particular context it is Web Application that is accessible through commonly known web address. User is able to create and show presentations as well as setup some of the presentation settings.

Framework. Easy extendable application which provides ability to be run from your own computer and make your own presentation in simple markup. Humla can be executed either as a standalone front-end web application or front-end application with server behind. For the first option user only needs to include one file into your HTML presentation file without any need for running server. The second option provides ability to host Humla on users' own environment. Example usage would be to host Humla for single university or its department. Both Server and Client part can be extended by user written extensions.

Module. Humla can be used as a module, which is to be included and run in other projects. This provides ability to use utilities and functions from Humla in other applications (build on top of similar technologies).

Humla can be divided into 3 main parts (also see Figure 1.1):

Server

Server is the heart of the whole system and it has to take care of all components of the system. It is the only part that can access a database. It has to take care of answering requests and database accesses.

[1] OpenOffice.org stopped its development and its biggest successor is LibreOffice.org

Portal

Portal is mostly standard web application that provides capability to create and modify courses and lectures filter and show available lectures (after login). Another portal goal is to provide list of available REST endpoints and services and further information about lectures.

Client

Client is responsible for the presentation itself. It provides different ways of showing slides. Client allows listeners to post their comments and further questions, or provide information about how they liked the slides. Client also provides little tests at the end of the lecture, where every listener chooses answers and when the test ends it will show correct results as well as further statistics. Note that this part is able to run without server.

1.3.2 Data Layer

As mentioned earlier, information for each Course and Lecture is stored in database, so the server can retrieve them later and use them as a user expierence enhancement.

Courses are collection of single course records, which stores information about a course owner and a course name as well as a base course URI.

Lectures are associated with single Course via `courseID` and in Lectures collection are defined lecture metadata about author and about lecture itself (its Abstract).

Comments define collection of comment records. Each comment has fields for identifying its author and for linking it with particular Course, Lecture and Slide. Note that slides are not persisted in database because the actual presentation is serialized to a HMTL file.

Likes collection is used for persisting likes/dislikes for each slide. It is coupled with slide through `slideID`, `lectureID` and `courseID` and it stores array of users which had clicked on like or dislike button.

Tests holds data assigned to each test. Each entry in this collection represents one test on one particular presentation and slide. Each test has questions and answers whith correct ones marked as well as the list of people's answers and additional information.

For persisting data is used NoSQL Key-Value pair database, so the actual data model may differ and even different lectures (e.g. newer) may have slightly different data model (e.g. due to another set of enabled extensions) and everything will still be able to work correctly. That is also one reason why I haven't defined that precisely all of the fields in all collections.

1.4 Mockup design

As said before our application is composed of three parts out of which only two interact directly with user. The first one is the portal that hosts all presentations and course materials and the second one is the presentation itself. Both user interfaces has to be enough simple and minimalistic, yet they have to follow common usage standards and UI patterns. For creating smart UI designs it's important start with making sketches and mockups to realize what will user see and what interactions will be needed. The goal is to display for user only what he needs at that particular moment and make the next step more obvious and easy to distinguish.

The main Portal window (as shown on the Figure 1.2) is the first thing user will see and it provides him with all information needed to select and open presentation. The second part is the presentation itself (HTML5 slides and Client underneath) with menu and extensions. Design is slightly different as shown on the Figure 1.3, because both parts serve a different purpose. Portal is mostly for searching and selecting presentations and Client is for viewing presentations.

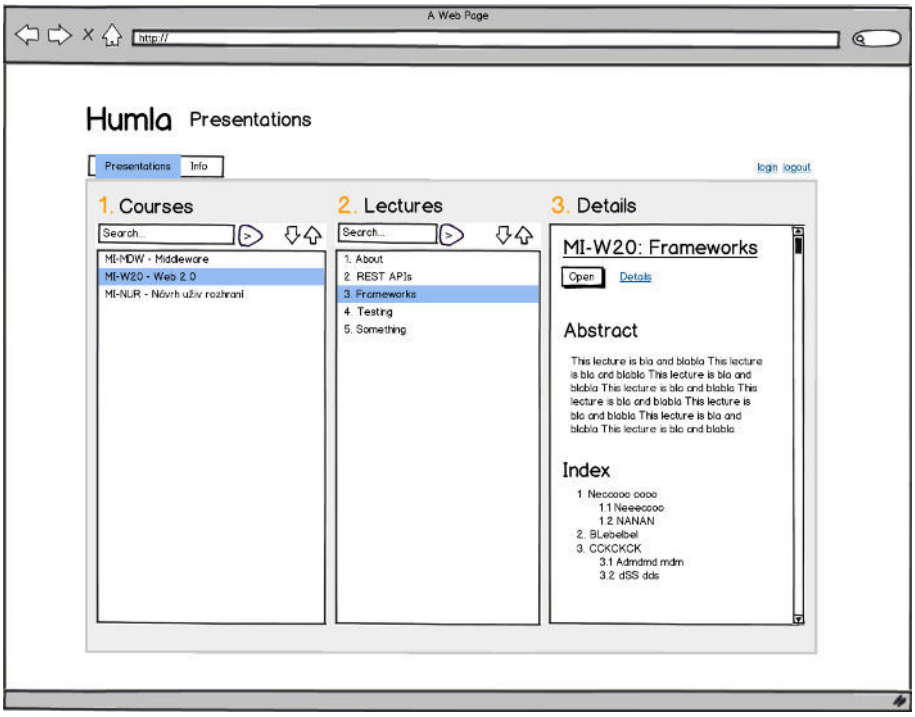

Figure 1.2: Mockup of Portal application

1.4.1 Heuristic Analysis

The main goal of heuristic evaluations is to identify and give a name to any problems caused by wrong design of user interfaces. This method was developed by usability consultant Jakob Nielsen based on his experience in teaching and consulting about usability engineering.

Jakob Nielsen's ten rules of Heuristic Analysis [17] of Graphical User Interface applied on Humla Portal and Humla Client:

1. **Visibility of System Status.** Humla doesn't make any high performance computations, so there's no need for any progress-bars, but the visibility of system status is assured even during loading of lectures. In the Portal part are guiding labels (i.e. *"Choose lecture first"*) to make sure user knows what next step he should take.

2. **Match between system and the real world.** Humla's presentations structure matches real world lectures, because in the real world is usually course that have one or more lectures associated with it. Buttons have consistent names and in the Client application are used intuitive pictographs as menu buttons.

3. **User control and freedom.** User freedom on Portal is assured by displaying all needed information and ability to change every chosen option and application behaves almost statelessly. In the Client user can browse through slides freely using keyboard arrows, mouse buttons or touch gestures on devices with touch support.

4. **Consistency and standards.** In both Portal and Client are used standard button names and design is consistent thanks to standard HTML controls.

5. **Error prevention.** In Humla there are only few user input fields (mostly while creating or managing new presentations). Validation is executed on those fields before the form is submitted. Whenever an error occurs, Humla shows an Error PopUp window explaining what caused the error.

6. **Recognition rather than recall.** On Portal user doesn't need to remember the previous steps in presentation search or selection, because all steps are still visible.

7. **Flexibility and efficiency of use.** Humla can be controlled using keyboard and mouse with all standard keyboard shortcuts (either from web browser or operating system). Humla uses intuitive slide control using keyboard arrows. Portal controls are similar to other web applications.

8. **Aesthetic and minimalist design.** In the application is displayed only the information needed and specially during viewing the presentation, there are no distracting elements, just the slides and menu bar with opacity.

9. **Help users recognize, diagnose, and recover from errors.** Humla displays error messages directly both on Portal and on Client application.

10. **Help and documentation.** Humla Portal offers About tab, where are further details about whole environment and links to documentation. The client application doesn't offer any type of help or documentation as it's control is very intuitive.

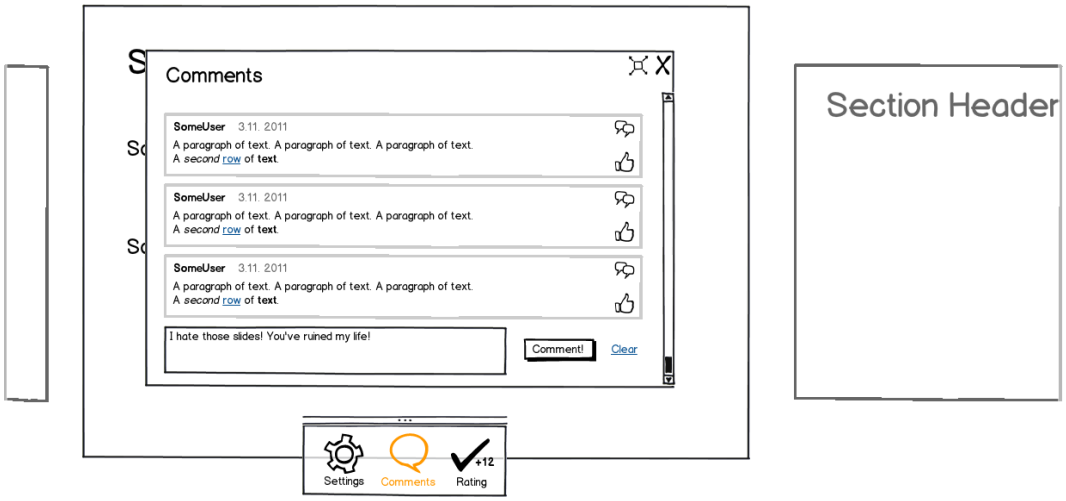

Figure 1.3: Mockup of comments on slides

1.5 Related work

This section provides an overview of related projects in the field of displaying slides online. Main focus is on possible ways of social interactions between slides and users.

Slideshare

Slideshare[2] is probably the most used service for sharing presentations on the internet with more than 1 million of registered users. After uploa-

[2] http://www.slideshare.net

ding in Microsoft Office PowerPoint or PDF format users can provide name, description, tags, language and visibility of presentation. User can later attach a soundtrack in MP3 to his presentation. Presentations are provided using HTML5 and they show various links for sharing on other social networks. Advantages of Slideshare are login via Facebook account and ability to search through presentations index. One of the disadvantages is that it's not possible to make an interactive presentations and the second disadvantage is that the uploaded file will generate HTML5 code without correct usage of semantic tags and with some other problems caused by conversion. Although it's possible to comment the whole presentation, there is no way to comment on a single slide. For every presentation there are usage statistics and buttons to share presentation in common social sites.

Prezi

Prezi[3] provides users with ability to create really beautiful slides using interactive editor. Then you can create slides with defined path in 2D space to move around slides with zooming and other effect. Application is made using Adobe Flash and it allows users to login using Facebook. Prezi allows comments and likes, but there are no further social interactions that would be usable in school. Prezi's main goal is to make more "marketing" and appealing slides that attract listeners' attention.

SlideRocket

SlideRocket[4] enables users to create own slides as well as upload files in PowerPoint format. Presentation editor is quite similar to PowerPoint. Presentation is in Adobe Flash, so most of the mobile devices won't be able to display it, although recently SlideRocket introduced HTML5 viewer. Even though it's possible to import YouTube videos or other web pages, the resulting slides are mostly static with only few animation possibilities. SlideRocket doesn't offer any direct interaction between lecturer and his listeners, but it provides measurements of presentation effectiveness through analytics page.

Authorstream

Authorstream[5] supports uploading of presentations created in Microsoft PowerPoint (i.e. files extensions PPT, PPS, PPTX and PPSX). Users can upload presentation either anonymously or under registered username. Presentation frontend is made using Flash technology with available control menu. Users can comment on presentations and also

[3] http://prezi.com
[4] http://www.sliderocket.com
[5] http://www.authorstream.com

like them using standard Facebook components. It's possible to embed voice into the presentation.

SlideVenue

SlideVenue[6] comes with ability to upload PPT and PPS files and then the presentation is converted to JPEG files runnable as a slideshow. Users can rate and comment on presentations, but the system is full of advertisements that may spoil the overall impression.

Prezentit

Prezentit[7] is quite simple tool with JavaScript based editor and user interface similar to what is known from PowerPoint. It allows users to cooperate on presentation creation. Thanks to JavaScript the Prezentit is usable on most of the smart mobile devices, but the system is still under ongoing development and it doesn't provide users with any ways of social interaction.

MyPlick

MyPlick[8] provides, apart from standard uploading of ODP, ODR, PPT, PPS, PDF, an interesting ability to upload also sound files. Presentation itself is again made using Flash and provides an ability to insert presentation to users webpage. User can rate, comment or share presentations.

As shown in the list of related project mentioned above, there are already available solutions for creating and showing presentations. There's no solution directly usable for social interaction between listeners and lecturer and yet usable for schools. Other options like drawing on slides or various view modes are also usually missing. There is no way to make a polls or small tests at the end of presentation.

[6] http://www.slidevenue.com
[7] http://prezentit.com
[8] http://www.myplick.com

CHAPTER 2

Principles and Technologies

> Any application that can be written in JavaScript, will eventually be written in JavaScript.
>
> Jeff Atwood

World Wide Web is only 23 years old, since it has been proposed by Tim Berners-Lee at CERN in 1989 [13] as a system for "high energy physicists to share data, news, and documentation." The first release consisted of a web server, which would communicate with web browser, both written in Objective-C. From that little project came something that its authors couldn't outguess. Web went through astonishing and dramatic evolution and became one of the most important technological milestones in the last century. World Wide Web had complied its name and became used world-widely with current rate of 32.7% of people with access to the internet (78.6% in North America)[1]. Nowadays, because of the wide usage, the World Wide Web can be referred only as Web. The main principles of Web architecture are:

- Standard-based — builds on top of a common agreement and allows big spread and adoption,
- Separation of Concerns — enables independent innovation (using standard interface between layers),
- Royalty-free technology — a lot of open source, no fees.

[1] Stats from 12/31/2011, http://www.internetworldstats.com/stats.htm

2.1 Web 2.0

First web pages contained only static content, which was simply served as it was stored on servers' hard drives, later the content got generated on the server from data usually stored in relational databases. And even later it became possible to customize pages by manipulating with a HTML Document Object Model (DOM) using JavaScript and separate styling to CSS style rules. The meaning of the Web had moved from static pages to highly interactive Web applications. Another huge milestone happened thanks to eBay, that launched a Beta version of their Application Programming Interface (API) in November 2000 [12]. This was a beginning of third party applications integration and web mashups. During these years the Internet sector and related fields noticed steady commercial growth and became interesting for many Internet-based companies and venture capital specialist, which created an environment, where investors were able to forget everything they've learned, forget the risks and invest huge amount of money for Web pages that had "e-" prefix and ".com" at the end. This era is called *dot-com* bubble.

After the dot-com bubble burst (around Fall 2001) all investors were suspicious about everything Internet related, so Internet companies had to think about new approaches and new paradigms. This is where Web 2.0 comes into the play. The term Web 2.0 came from a name of a series of web conferences by Tim O'Reilly [9].

Web became a platform and the way users and developers interact with the Web changed significantly. Web 2.0 enables users to not only consume information but also produce their own values and information on various web applications. This change was based on ability to dynamically send and receive data using Asynchronous JavaScript and XML (AJAX) with `XmlHTTPRequest`. Web applications became usable as a desktop applications and it became possible to use the application with any computer connected to the internet, since the application downloaded from a remote server and after that executed locally. The only required condition from the user is a compatible web browser. The trend is to move applications from desktop to web browser, since this provides benefits for both, users and developers. Users have synchronized content through all their smart mobile phones, laptops or desktops and developers don't have to keep in mind differences between platforms since they can just write application only for one application server with frontend written in JavaScript (though they ought to solve some browser incompatibilities). There is also almost instant update of new versions and to push bug-fixes to users is fairly easier.

The current web as a platform extends user possibilities and with growing number of web applications, where users create their own content by posting to various social services. Developers can also create mashup application of other applications using their APIs. The importance of APIs has increased significantly [15]. And developers can publish their applications much early,

so they are often in *Beta* state even though they are widely used.

Web is based on three main architectural concepts:

- Identification — universal linking of resources using URI,
- Interaction — different protocols to retrieve resources (e.g. HTTP),
- Formats — standardized resource representation of data and metadata (HTML).

Let's have a look at Identification, because it is crucial to have a single global identification system and for Web it's Uniform Resource Identifier (URI). With URI can be addressed concrete resources and navigate through them via hypertext. URI identifies a resource, but the resource doesn't have to physically exist, so for locating the resource is used Uniform Resource Locator (URL).

```
scheme":"["//"authority]["/"path]["?"query]["#"fragment]
```

Listing 2.1: Structure of URI

In Listing 2.1 are stated different parts of URI. URI consists of scheme (usually used for specifying protocol), authority (registered name, or server address), path and query (identifies resource) and fragment (refers to a part of a resource).

One of the most important Web protocols is Hypertext Transfer Protocol (HTTP)[2] which is used for exchange of hypertext documents. Hypertext documents are usually represented using Hypertext Markup Language (HTML). However hypertext is not only HTML, but every representation of a resource with URI links will do.

2.2 REST

Representational State Transfer (REST) is a resource-oriented model of Web APIs for Web Services created by co-author of HTTP, Roy Fielding [7]. The REST is strongly based on Client-Server architecture and it is possible to simply use HTTP for realization of REST (then the realization is called RESTful), which provides many features of HTTP like Caching, Addressability and Uniform Interface. Thanks to HTTP it is possible to use highly scalable Web infrastructure (caching and/or proxy servers) and that's one of the reasons of good world-wide adoption of REST. In Figure 2.1 it's apparent that REST is the most used protocol (at least on http://programmableweb.com). As a side note, SOAP[3] was there first so big companies and state administration use

[2] Currently used version of HTTP is 1.1.

[3] http://www.w3.org/TR/soap/

SOAP more since it is easier to standardize using WSDL[4]. REST can also be formalized for example using WADL[5], but formalization is still under ongoing research and good practice is to create a developer-friendly documentation.

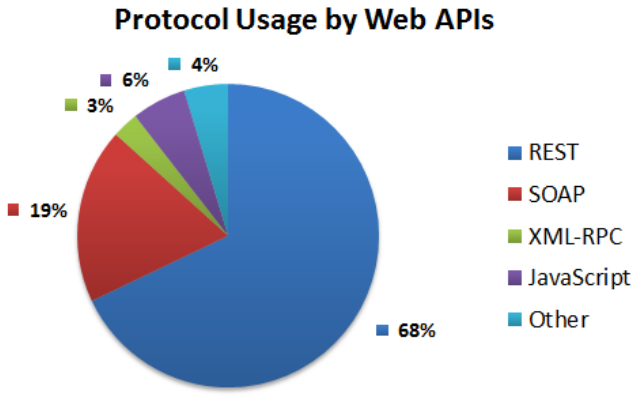

Figure 2.1: Protocols, data collected on March 11th, 2012 from http://www.programmableweb.com/apis/directory/

RESTful uses HTTP methods (GET, PUT, POST, DELETE and others) to manipulate with resources. Data representation can be in different formats, like XML, YAML or JSON. For Humla project the most appropriate to use is JSON because of its JavaScript nature, but some of the main REST endpoints also return XML to reach wider group of developers.

The status of HTTP response is expressed by one of the defined HTTP response codes. Server has to set correct response codes to express situation caused by clients request.

Response codes are categorized into 6 different classes[6]:

- **1xx** Informational — indicates a provisional response,
- **2xx** Successful — indicates that action requested by client has been successful and accepted correctly,
- **3xx** Redirection — indicates that client has to take additional action to complete the request,
- **4xx** Client Failure — informs about error on client side,
- **5xx** Server Failure — informs about server failure to a valid request,

[4] http://www.w3.org/TR/wsdl/

[5] http://www.w3.org/Submission/wadl/

[6] Status codes definition can be found at http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

- **6xx** Global Failure — indicates that there's an unspecified general error.

In RESTful HTTP status codes has to be setted up correctly for each response.

2.2.1 State

"State is a set of values or data that application holds. The state is distributed to both clients and servers and statelessness does not mean there is no state."[21] Every application has some notion of a state and the main difference is in how the state is represented. Application servers can be divided to *Stateless* and *Statefull* — the first one holds state in data being communicated but doesn't save it between connections, whereas the latter holds state in its memory and sometimes it even persists state to DB.

In REST the server is stateless so it uses the Hypertext as the Engine of Application State (HATEOAS), where the link realizes a transition between application states. One of the advantages of this approach is loose coupling of server and clients.

2.3 Cloud

Cloud Computing is sharing hardware or software resources over common network (typically the Internet). Users of "clouds" approach to delivery of computing as a service rather than a product.

The main advantages over the standard hosting are higher availability and reliability, lower costs and easier scalability. Cloud has become a widely used technology and more and more companies start to use Cloud on different levels. As Imad Mouline, CTO of Gomez – a web performance measurement company, says: *"The cloud is being adopted by the biggest, stodgiest companies out there, whether they know it or not."*[16] Big companies use their own Clouds in their datacenters, where different departments lease desired CPU time. There are also companies that allow users to use their datacenters' capacity for their own projects for a little fee based on consumed CPU Time. Examples of this approach would be: Google AppEngine, Microsoft Azure or Amazon AWS.

There are 3 different levels of Cloud Computing[7]:

- **SaaS** — Software as a Service: mainly web-based software for end-users,
- **PaaS** — Platform as a Service: preconfigured development environment for developers,
- **IaaS** — Infrastructure as a Service: physical computers or virtual machines with network middlewares and available CPU time.

[7] There are also few humorous abbreviations: EaaS - Everything as a Service or FaaS - Fun as a Service.

Humla is the example of SaaS cloud application where users create their presentations online.

2.4 HTML5

HTML5 is backward compatible specification of HTML markup language proposed by World Wide Web Consortium (W3C) which builds on top of HTML 4. HTML5 is in state *Last Call Working Draft* and is expected to get to *Recommendation* state around year 2022 [11]. HTML5 is widely supported by various companies including Google, Adobe and Microsoft.

2.4.1 History of HTML

We can date beginnings of HTML to year 1993 [19], when HTML 1.0 was proposed as a draft to the Internet Engineering Task Force (IETF). HTML spread around the world with the expansion of Internet but there always has been an argument between browser vendors, web developers and people from W3C. Most of HTML revisions were defined just to reflect what browsers already support and most of the web sites have at least one syntactic problem. So to satisfy users browser vendors had to make their browsers able to display even very corrupted HTML page (at least somehow) and then web developers have started optimizing their code for those browsers which meant a big trouble. One example would be initial versions of Internet Explorer (Microsoft licensed first browser Mosaic to create Internet Explorer), which added new proprietary features instead of fixing bugs and obeying standards (e.g. `<font>`, `<marquee>`). With Internet explorer (IE) started the first "browser war". Other web browsers were in worse starting position because the IE was bundled with every Windows installation. This led IE to dominating of market, which has culminated in 2002 with 95% share. The noteworthy fact is that IE during it's highest shares wasn't following standards in corner cases and it gained a label of being "buggy". This was ideal opportunity that other browser vendors could take advantage of. As a result, other browsers started to fight about market share with browsers obeying standards (more or less).

In October 2006, Tim Berners-Lee, the founder of the W3C, announced that the W3C would work together with the WHAT Working Group to evolve HTML. One of the first things the newly re-chartered W3C HTML Working Group decided was to rename currently developed "Web Applications 1.0" to "HTML5."

2.4.2 HTML5

HTML5 is a collection of individual features, so it's currently impossible to just simply detect "HTML5 support". HTML5 is successor of HTML 4 which and everything that worked in HTML 4 is still functional in HTML5 although

there may be a better and simple way. Browsers that don't support HTML5 markups still work (almost) correctly but perhaps the result wont't be so user friendly. Current versions of most browsers support HTML5 at least partially and since the HTML5 specification still isn't in state of Recommendation it can't be stated if browser has full support or not. But it appears that in the future all major browser vendors will support HTML5 as they take part in W3C HTML Group.

From the name it may look like HTML5 is only markup language, but the markup itself is only one small part. One could say that HTML5 is actually set of different JavaScript APIs which evolved from use cases that users already used, but had to implement them with some (mostly JavaScript) workarounds.

Local Storage

Developers have desire to save user settings and user local information inside browser. The main reason is usually better user experience and lower amount of transferred data and server load. Before HTML5 developers had two options:

Save data inside hidden HTML elements — This option is still used by large server frameworks to manage application state,

Cookies — Cookies are simple key–value pairs for saving almost any serializable data, but they are limited in size (4KB) and they are sent back to the server with every request. That leads to unnecessary data communications.

HTML5 comes with better solutions. HTML5 storage provides way to save various string based data in user's browser (usually in designated browser's folder). User can load saved data even after browser restart. Local Storage API used to be part of the main HTML5 specification, but it got separated, because HTML5 Working Group realized that HTML5 specification is getting too big. So as said by Mark Pilgrim in his book [18]: "If that sounds like slicing a pie into more pieces to reduce the total number of calories... well, welcome to the wacky world of standards." `LocalStorage` (and `SessionStorage` as a matter of fact) is object that implements the HTML5 Storage interface. Local Storage is accessible via `localStorage` variable inside browser's DOM (Data Object Model) window object.

Local Storage has quite simple API for clearing all items, getting item, setting item and deleting item.

2.5 JavaScript

JavaScript is multi-platform, object-oriented scripting language developed originally by Brendan Eich from the company Netsape. JavaScript derives its

syntax from Java, its first-class functions from Scheme and inheritance model from language called Self [8]. Standardized version of JavaScript is called ECMAScript, which refers to the name of the standardization organization ECMA. Current version of ECMAScript is ECMA-262 version 5.1.[8] ECMAScript 5 offers new properties to make objects immutable and other new features. From ECMAScript are derived other implementations like ActionScript.

"JavaScript is an important language because it is the language of the web browser. Its association with the browser makes it one of the most popular programming languages in the world. At the same time, it is one of the most despised programming languages in the world. The API of the browser, the Document Object Model (DOM), is quite awful and JavaScript is unfairly blamed. The DOM would be painful to work with in any language. The DOM is poorly specified and inconsistently implemented."[3]

JavaScript is built on top of very good ideas, but it has also many flaws. Let's have a look at both sides:

The Good Parts

The really good parts of JavaScript are its functions implementation as first class object with (mostly) lexical scoping, loose typing, dynamic object and expressive object literal notation. It has prototypal inheritance which means one object can inherit properties from other object without using concept of classes. "Deep down, JavaScript has more in common with Lisp and Scheme than with Java. It is Lisp in C's clothing. This makes JavaScript a remarkably powerful language."[3] One of the strengths is also build in event-loop as a language feature — this is particularly important for asynchronous I/O servers like Node.js.

JavaScript is fairly stable language. There have been no new design errors since 1999, but this changes again with ECMAScript 5.

The Bad Parts

The bad ideas include a programming model based on global variables, non-intuitive scoping problems with closures (functions returned from another functions) and other implementation failures like non-transitive equals operator. Let's have a look at the following example:

```
undefined==false  // false
false==null       // false
undefined==null   // true
```

JavaScript has also other problems with automatical type coercion of `==` operator, so the good habit is to use `===`, which doesn't coerce automati-

[8] Specification can be found at: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262edition5.1,June2011.pdf

cally. Another really interesting examples of JavaScript implementation oddities are (with results in commment part)[9]:

```
[] + []             // ''                    1
[] + {}             // {}                    2
{} + []             // 0 (or {} in V8)       3
{} + {}             // NaN                   4
```

Those are reasons why JavaScript programming language is referred by Douglas Crockford as *"the most misunderstood language"*. It has gained bad reputation during 95% IE market share era, but its biggest strength is the availability almost in every single web browser.

In Humla the JavaScript is used for both client and server side.

2.5.1 JSON

JSON[10] is simple universal data interchange format made as part of the ECMAScript standard. JSON (the abbreviation stands for JavaScript Object Notation) format has been defined by Douglas Crockford and it is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition — tangibly it is based on JavaScript's object literal notation. JSON's MIME (Internet media type) is `application/json`. These days many popular sites, like Google or Yahoo, publish their data through various mostly REST webservices also in JSON format.

Big benefits of JSON are the speed of processing and the easiness of use, because it is recognized natively by JavaScript. ECMA has defined in 1999 the `eval()` function that parses the format. But all modern browsers support more secure approach `JSON.parse(param)` and `JSON.stringify(param)` functions. A correctly implemented JSON parser will accept only valid JSON, preventing potentially malicious code from running.

JSON actually differs slightly from JavaScript definition in line ending characters, because Douglas Crockford forget to add that to standard [3].

JSON is built on top of two popular structures:

- **Array** is ordered list of values, which is surrounded by two brackets,
- **Object** is unsorted collection of name/value pairs, which is surrounded by two braces.

Both structures can be nested inside each other and values and pairs are separated by commas. Value can be one of string, number, object, array, `true`, `false` or `null`. An example of JSON with object and inner arrays representing single presentation would be:

[9] Examples from Gary Bernhardt's lightning talk from CodeMash 2012 https://www.destroyallsoftware.com/talks/wat

[10] http://www.json.org/

```
{                                                                    1
    "title": "Diploma thesis presentation",                          2
    "keywords": ["Humla"," HTML5"," presentation"],                  3
    "numberOfSlides": 17,                                            4
    "images": [],                                                    5
    "codeBlocks": [                                                  6
        {   "slideURL": "/Diploma/lecture2.html#!/14",               7
            "language": "JavaScript"                                 8
        }]                                                           9
}                                                                    10
```

Listing 2.2: JSON Example

One of the modifications based on JSON is JSONP, which is JSON with Padding, a usage pattern commonly employed when retrieving JSON from a webpage from another domain.

JSONP is based on loading data through `src` attribute of the `<script>` HTML element and thus outsmart the Same Origin Policy security concept.

First, the browser reads `src` attribute of the `<script>` element. Then browser sends a GET request to the server:

```
http://example.com/user?Id=1234&jsonp=parseResponse                  1
```

Then server returns JSON padded with a function call:

```
parseResponse({"Name": "Foo", "Id" : 1234})                          1
```

Received JSON data are evaluated by the JavaScript interpreter, because browsers don't parse them using JSON parser. This allows to save received data to any variable in client's `parseResponse()` function. Without JSONP the data would embedded into the HTML document.

Modern alternative to JSONP would be Cross Origin Resourse Policy (CORS), which uses designated HTTP headers to negotiate ability to access resources from other domains. CORS is supported by majority of modern browsers.

2.5.2 Humla

The original Humla (OH) presentation environment by Tomáš Vitvar is JavaScript library that uses HTML5 and ECMAScript 5 to show presentations in browser window. Supported browsers are Google Chrome and Safari. It offers different ways do display slides (via Views) and it is controlled by keyboard. The presentation is in pure HTML5 and the only thing that's needed is to include `humla.js` and `humla.css` files into the HTML header. OH loads other necessary files dynamically. OH provides a `config.json` file in which it is possible to set up behavior, views and extensions.

OH doesn't have any underlying server included so it has to be hosted in form of static files from a server. It's currently not possible to run OH directly

from the file system, because the dynamic script file loading is disabled by default in the majority of browsers.

2.5.3 jQuery

JQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.[11] JQuery is used on the Humla's Portal administration sites for filtering, field manipulation and events handling.

2.6 Application Server

Building frameworks represents a set of challenges which needs to be solved and the solution itself depends a lot on the chosen technology. Since most of the use cases defined before can be done asynchronously for backend of the Humla is used Node.js, which has got peak popularity between its enlarging community due to its non-blocking nature and event handling mechanism.

2.6.1 Node.js

Node.js[12] is young[13] event driven JavaScript environment with non-blocking I/O based on JavaScript implementation V8. V8 also stays under the hood of Google Chrome web browser. One of the Node.js' goals is to make an easy way of developing scalable network programs. It is influenced by other event driven frameworks and systems like Twisted (Python) or Event Machine (Ruby). Node's author, Ryan Dahl, comments Node as a "bunch of sugar on top of a very complex virtual machine written by Google"[14]

Even some major technology companies invested in and use Node.js for solving some of their performance and other problems. Examples would be[15]:

Microsoft invested money and time to make Node.js working correctly on Windows and also made it possible to use Node.js in Azure, Microsoft's cloud service,

Ebay choosed Node.js as the runtime stack in ql.io: a data-retrieval and aggregation gateway for HTTP APIs,

LinkedIn has build their entire mobile software stack on top of Node.js,

[11] http://jquery.com/

[12] http://nodejs.org

[13] Ryan Dahl announced Node.js on 02/09/2009 at http://four.livejournal.com/963421.html.

[14] 11/08/2009 Ryan's presentation of Node.js at JSConf 2009 resulted in applauded ovation.

[15] extended list from http://nodejs.org

Yahooo uses Node.js for some inside startups and for supporting browsers without JS enabled by generating plain HTML,

Facebook is using Node.js for experiments with JSGameBench (an HTML5 game benchmarker), mobile JS framework and for traffic analysis and load testing.

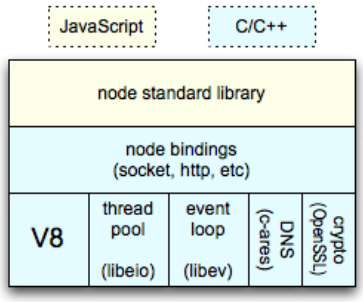

Figure 2.2: Architecture of Node.js (Courtesy: [5])

As shown in figure 2.2 Node.js' core is built in C/C++ on top of Google's V8 JavaScript implementation. V8 compiles the source JavaScript directly to machine code the first time it is executed, so there is no bytecode format and JavaScript is not interpreted. Although the main part is V8, Node.js also relies on other libraries:

- **libeio** for thread pool and asynchronous I/O,
- **libev** for event-loop mapping to underlying operating system,
- **c-ares** for asynchronous DNS support,
- **OpenSSL** for cryptography and SSL/TLS support.

All those libraries are written in C/C++, whereas Node.js standard library is written in JavaScript and there are plenty of helpful bits like REPL command-line interface and URL parser. Standard library layer can only access to the main thread, but in lower C levels it is possible for Node.js to take and advantage of multi-threading programming.

Node.js is capable of handling thousands of incoming connections at the same time on a single computer[16] and that's the limit where other technologies have often failed. The reason for this behavior is that other technologies like Apache usually scale by spawning threads for each incoming connection, Node does it in a different way by firing event for every HTTP request using only

[16]The theoretical problem is called C10K http://www.kegel.com/c10k.html

single process (although there are situations where few more processes are possible and more accurate).

Tedd Ziuba mentioned reasons to or not to choose Node.js prior to other technologies in his article *"Node.js is Cancer"* [4].

The objections are following:

- *"'Node.js is scalable because it never blocks' is a lie"* — There will always be blocking in the sense of CPU computation, but in the world of Web applications that isn't usually the main problem causing regressions. Node.js is using JavaScript to make it easier not to block on I/O operations, which usually take by magnitude more time than CPU operations.
- *"Node.js doesn't employ separation of responsibility and is not enough loosely coupled"* — This objections is accurate and even though Node.js includes own HTTP server it is often proxied using Nginx or other HTTP proxies. Node.js architecture is more similar to Python's standard library, which is written in Python, but with performance crucial blocks of code implemented in C.
- *"Node.js is server written in JavaScript"* — This is completely reasonable objection since JavaScript is simple dynamic programming language with many flaws, but if used correctly it becomes easier and faster to write applications. The stability of Node.js is still questionable due to its maturity.

Simple HTTP server in Node.js returning text with HTTP header 200 OK would look like this:

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, "127.0.0.1");

console.log('Server running at http://127.0.0.1:1337/');
```

Listing 2.3: Node.js Server Example

As shown in Listing 2.3, there are two important facts, which acquired Node.js a lot of attention. First, a function is passed as a parameter to be executed as a callback to incoming request using JavaScript language constructs and inner event loop. Second, due to JavaScript it is fairly easy to understand what is happening and still be able to write high-performance application.

From previous paragraphs it is apparent that event-loop approach may provide better performance conditions and lower memory consumption, but let's take a look at real example of how event-loop works:

```
function example() { console.log('A: In next tick'); }
process.nextTick(example);
console.log('B: Standard');
```

Output is:

```
B: Standard
A: In next tick
```

Considering the order of commands it would be intuitive that Node.js interpreter prints output "A: In next tick" and then "B: Standard". It is actually the opposite because of `process.nexTitck()` function, which is scheduling execution of function `example()` for next loop of JavaScript's event-loop.

Node.js or Java EE

"Support for multiple users in Enterprise Java applications is provided by the application server in the form of multi-threading. Each user-request is mapped to a thread, which takes care of responding to the user-request. Asynchronous event-driven frameworks do exist in the Java world. Examples are the jboss.org project Netty and Apache MINA, which are both based on the Java NIO (New I/O) API. ... Problem is that these frameworks are outside of the JEE specifications. In the Enterprise Java world it is the JEE specification that lays out the way applications deal with concurrency, and so far they are sticking to the multi-threaded model. If you want to be JEE conform you have to use multi-threading for concurrency."[6]

The conclusion is that Node.js is useful for both highly concurrent applications with a large user base and with thousands of concurrent requests and it is still easy to write small prototype applications. Node.js uses almost exclusively asynchronous I/O, whereas Java EE usually maps requests to threads.

2.6.2 Node.js and libraries

Node.js without its growing community of developers couldn't raise interest around the internet and it would come back to dust as other startup applications and companies. After three years of Node's lifespan there has been amazing spread between users and developers of Node.js.

With larger community comes bigger amount of libraries (Node.js developers call them modules), that have been created to interact with other services (e.g. database connectors and object mappers, web services' clients or different authentication plugins) and to make development easier (various frameworks and middlewares). Modules can be installed using Node Package Manager (npm)[17] using simple command line interface.

Examples of most used and forked modules would be:

[17] As of Node.js version 0.6.3 npm has been incorporated into the Node's distribution.

- **Forever** by Nodejitsu Inc — A simple CLI tool for ensuring that a given node script runs continuously (i.e. forever),
- **coffee-script** by Jeremy Ashkenas — CoffeeScript is a little language that compiles into JavaScript,
- **Express** by TJ Holowaychuk — Sinatra inspired web development framework,
- **socket.io** by Guillermo Rauch — Real-time apps made cross-browser & easy with a WebSocket-like API,
- **jsdom** by Elijah Insua — A javascript implementation of the W3C DOM.

All these modules can be found in the GitHub (http://github.com) or in the website of npm.

Express

Express.js is Sinatra[18] inspired web development framework for Node.js developed by TJ Holowaychuk. Express is built on top of Connect, which is a set of common HTTP server related middlewares for Node.js. Express makes it easy to create high performance applications with its caching and HTTP header manipulating middlewares [20].

Features as listed from `http://expressjs.com`:

- Built on Connect
- Robust routing
- HTTP helpers (redirection, caching, etc)
- View system supporting 14+ template engines
- Content negotiation
- Focus on high performance
- Environment based configuration
- Executable for generating applications quickly
- High test coverage

[18] Sinatra is web application library and domain specific language written in Ruby. Further details can be found at `http://www.sinatrarb.com/`

2.7 Database

One of the problems of real-life communication is that it's not from definition recorded and to obtain some only spoken information again, student has to ask teacher to repeat that and even then it's not guaranteed that student will remember the information later. And there comes into play our presentation environment, which may save some comments and other student-teacher interactions to persistent database (DB). One of the goals when choosing the right Database Management System (DBMS) is good integration with JavaScript and Node.js.

Database plays really important role on scaling of web application. Standard relational database management systems may cause read/write delays and in some cases may be a bottleneck. Regarding this problem there's an option to use NoSQL[19] databases, which are principally designed for web application with focus on their performance and scalability.

2.7.1 CAP Theorem

"In theoretical computer science, the CAP theorem, also known as Brewer's theorem, states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:" [2]

- **Consistency** means that all nodes are able to see the exactly same data at the same time,
- **Availability** provides us with a guarantee that for every request there will be a response about the result, whether it was successful or not,
- **Partition tolerance** means that system will keep operating and be able to even if some data is lost due to communication or hardware problems.

The theorem says that it is possible to satisfy any two guarantees, but not all three at the same time.

As shown in Figure 2.3, there are various databases on every side of CAP triangle. The most used NoSQL Database engines are:

- CouchDB,
- MongoDB,
- Riak,
- Cassandra,

[19] NoSQL is an umbrella term for a loosely defined class of non-relational data stores that break with a long history of relational databases and ACID (atomicity, consistency, isolation, durability) guarantees. Data stores that fall under this term may not require fixed table schemata, and usually avoid join operations. The term was first popularized in early 2009.

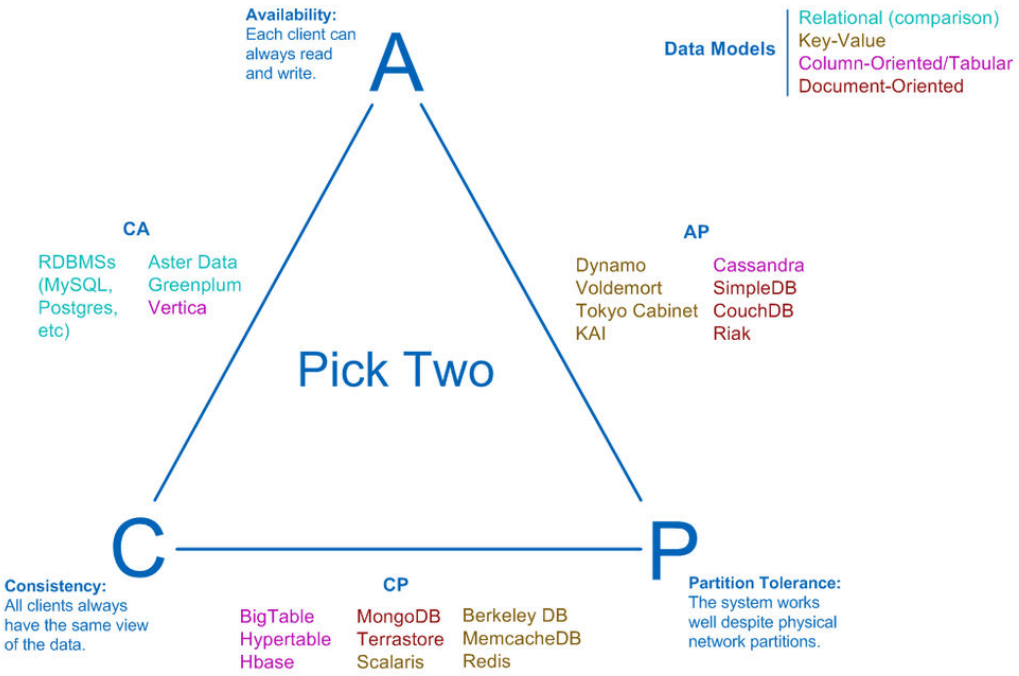

Figure 2.3: CAP Theorem and visual guide to NoSQL (Courtesy: [10])

- Memcache,
- BigTable.

The most appropriate database system for Humla is MongoDB because of its good integration with Node.js, wider user-base, replication abilities and because of an option to use Mongoose, which would bring us a schema like approach for easier access to data. It is also possible to migrate database from MongoDB to similar databases like CouchDB.

2.7.2 MongoDB

MongoDB[20] is a scalable, high-performance, open source non-relational BSON based database management system, which provides well-known API with JavaScript & JSON like syntax.

"MongoDB is a document-oriented database system with a strong focus on flexibility, scalability and performance. Document-orientation involves leaving the row-centric concept of the relational database model, and introdu-

[20] http://www.mongodb.org

cing the much more flexible notion of a document. Document-orientation avoids rigid database schemata and also promotes a certain degree of denormalization which allows embedding documents into each other, leading to potentially much better performance by avoiding the need for expensive join operations."[14]

MongoDB provides:

- Document oriented storage,
- Full Index Support,
- Replication & High Availability,
- Auto-Sharding,
- Rich Document based queries,
- Map/Reduce,
- Command line interface.

MongoDB is document-oriented DBMS, so instead of focusing on the "row" as primary data entity, the core of MongoDB is a "document". Documents are in collections in which no predefined schema is enforced, thus the documents in a collections don't have to share the same structure.

During replication process MongoDB uses master-slave architecture and data is replicated from master instance to configured slaves.

BSON

BSON[21] is the main data format which is used in MongoDB to store data to disk as well as use them for communication. BSON is based on JSON and the main difference is that BSON is binary format whereas JSON is plain-text. The reason for encoding data in binary format is efficiency — numbers don't have to be converted to/from text, which is faster than with JSON numeric strings. BSON defines few own data types that it parses directly. The JSON string `{"hello": "world"}` is represented (in standard JavaScript/Python notation) as:

```
\x16\x00\x00\x00\x02hello\x00
\x06\x00\x00\x00world\x00\x00
```

[21] http://bsonspec.org/

Mongoose

Mongoose[22] is a JavaScript library designed to run within the Node.js environment. Mongoose makes working with MongoDB easier by providing a more intuitive API. Mongoose also provides the ability to define data models that act as a gatekeeper protecting your data. Models offer typecasting on field, validation, default values, output formats and other options.

Mongoose holds DB connection in internal variable and thus it is possible to access to MongoDB via Mongoose only by requiring `mongoose` module and then making requests like in the following example:

```
var mongoose = require('mongoose/').Mongoose,
    db = mongoose.connect('mongodb://localhost/test'),
    Collection = mongoose.noSchema('test',db);

Collection.find({}).each(function(data){
    // request result is in the variable data
});
```

Listing 2.4: MongoDB data access using Mongoose

[22] http://mongoosejs.com/

CHAPTER 3

Design and Implementation

> Everything runs in parallel
> except your code.
>
> Mikito Takada

With an introduction to the most suitable technologies in previous chapter, this chapter is dedicated to architecture design and implementation of Humla presentation environment.

3.1 Architecture Design

As shown in Figure 3.1 Humla is composed of three main parts:

Client is downloaded from Server and executed in user's browser and it basically allows users to view presentations. Client interacts with Server through REST API and it is extendable through JavaScript extensions.

Portal is also downloaded from Server and executed in user's browser. Portal allows users to create courses and lectures and to edit further information about them. User can select desired course and open the presentation.

Server is Express.js HTTP server with access to MongoDB (via Mongoose module). Server runs on top of Node.js. Server hosts both Client and Portal and it is extendable via extensions.

3.1.1 Behavior

Whole process starts when user opens Portal Page (with standard HTTP GET request), the webpage is being downloaded to user's browser and then it makes an AJAX request for available Courses and their Lectures. User can

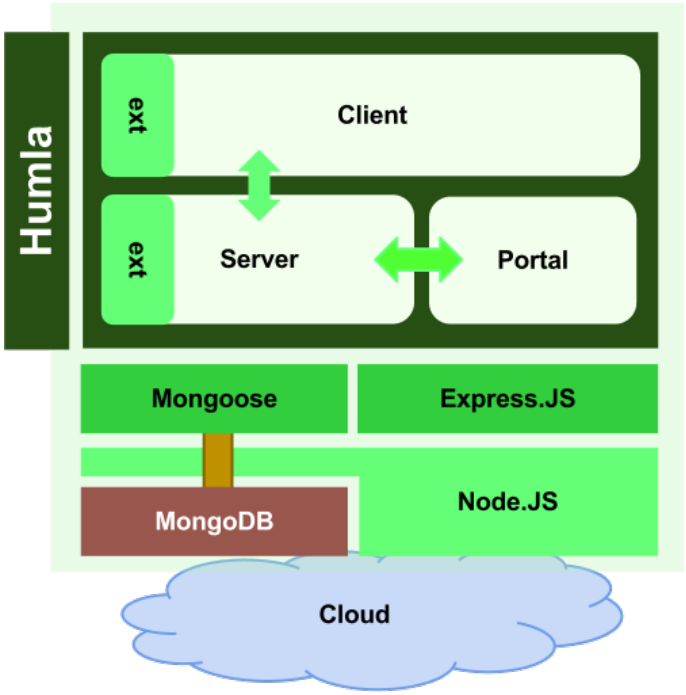

Figure 3.1: Full application stack

login using OpenID and then, if he has sufficient rights (e.g. he is a Lecturer), he can create course, or lecture in selected course.

Let's get back to motivation situation mentioned in the first chapter, where teacher is making presentation for his students. Teacher creates Course in which he creates new Lecture and then he just gives his students information, where Humla is hosted (if that's not the first lecture, they will already know). After that teacher opens a presentation, which is represented as a single HTML file with link to the Client application (There's only one link to `humla.js` and it dynamically loads other necessary files through dynamic content loading) and the client will take care of the whole initialization, so the teacher is able to start lecturing right away. The lecturer can send a URL link to students or students can open the Portal page, search and select the currently narrated lecture and open it. Then they are able to interact live with the teacher.

Humla is easily extendable via extensions on both server and client side. Extensions usually exist in pairs, so the client part is able to communicate with its Server side extension through REST API.

3.2 Libraries

For server part are used following Node.js modules (with current versions):

- Mongoose 2.5.13
- ExpressJS 2.5.1
- Cron 0.1.3
- Jsdom 0.2.10
- Passport 0.1.1

Libraries may have their own dependency packages, since the Node package resolution finds the nearest `node_modules` folder in the filesystem tree. All those packages are easily maintaned using NPM.

Humla can be run in Cloud, because both MongoDB and Node.js are able to run on Cloud computing hosting. For IaaS it's possible to deploy Node.js for example to Amazon EC2[1] or JoyentCloud[2], for PaaS it's possible to deploy Node.js to Nodejitsu, Heroku and many others.[3]

3.3 Server

Humla is built on top of Node's asynchronous I/O. And the asynchronous approach is realized through the set of callbacks for every single I/O task, which avoids executions blocking. Also there is no overhead as in systems with one thread per request. Some of the initialization steps have to be done in a synchronous way to be able to load all necessary system parts before server starts to listen on a selected port.

The whole initialization process is shown in Figure 3.2. The most important steps are:

- **Server execution.** User can run Humla using console command *node index.js*.
- **Load and parse configuration file.** In this step the server configuration (from `server-config.json`) is passed to the server instance. Configuration file consists of paths, domain, port and list of enabled plugins. This configuration is passed to Express.js server with other middlewares (e.g. sessions, logger, passport).
- **Load Models, Handlers and Extensions.** Humla loads files representing Models (`/models/`), Handlers (`/handlers/`) and Extensions (`server_ext`) using standard Node.js `require()` method. So for models the requiring loop looks like follows:

[1] http://aws.amazon.com/ec2/
[2] http://www.joyentcloud.com/
[3] The list of hostings is at https://github.com/joyent/node/wiki/Node-Hosting

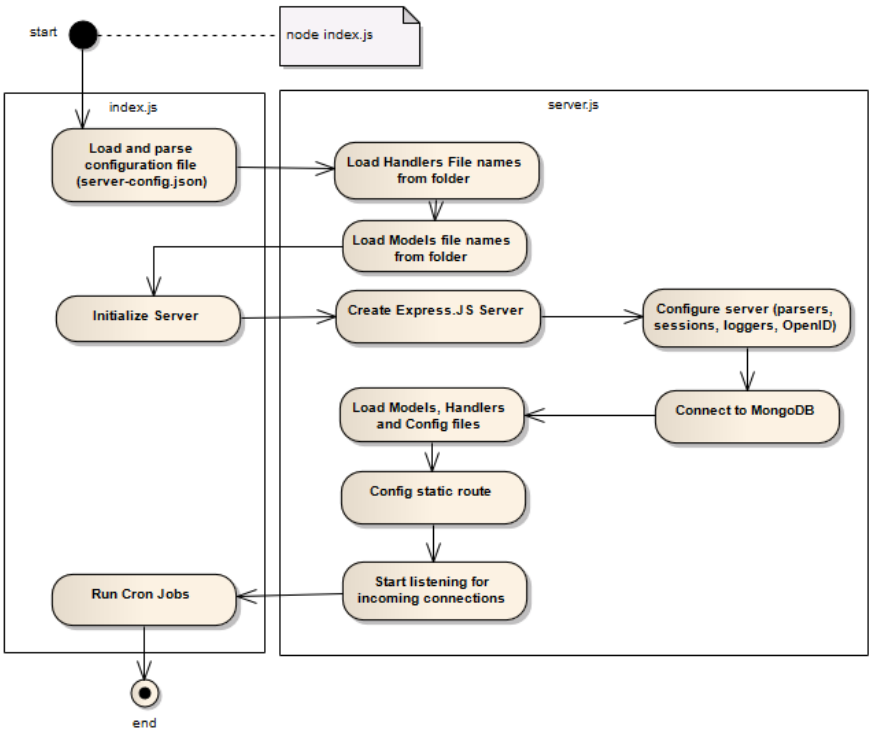

Figure 3.2: Activity diagram of server execution

```
models.forEach(function (model){
    require(model);
});
```

For models and handlers all files from their folders are loaded, whereas for extensions only files that are defined directly in `server-config.json` are loaded.

After all steps are done the server starts listening on a given port.

Server has few different responsibilities and the first one is to statically serve requested files over the internet. Humla uses Express static route (see Listing 3.1).

```
app.configure(function() {
    var oneYear = 31557600000;
    app.use(express["static"](webroot), {
        maxAge: oneYear
    }));
});
```

Listing 3.1: Usage of express static route

Then for the following HTTP GET request to the Humla:

```
GET / HTTP/1.1
User-Agent: curl/7.21.1 (i686-pc-mingw32)
Host: localhost:1338
Accept: */*
```

Humla server responds with the correct parameters as set in Express.js configure:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Date: Thu, 22 Mar 2012 21:02:54 GMT
Cache-Control: public, max-age=31557600
Last-Modified: Tue, 20 Mar 2012 19:37:52 GMT
ETag: "6643-1332272272000"
Content-Type: text/html; charset=UTF-8
```

Then for every incoming request the Express routing chain is executed as well as various middlewares[4], like body and cookie parsers as well as session storage with OpenID support.

Express utilizes the HTTP methods names through routing API. For HTTP GET request we simply call `get` methods with parameters of route strings that are parsed internally in Express, and function callbacks to execute when the request is ready:

```
app.get('/', function(req,res) {
    res.redirect("/pages/index.html");
});
```

In this example is shown the default redirect to the main portal page.

With default options come few handlers defined in `/handlers/default.js` for retrieving default data for Portal and standard error pages.

3.3.1 Humla as a Node.js Module

Apart from using Humla as a standalone server it is also possible to use Humla server extensions in other Node.js modules and applications. The definition file for Humla used as a module is `humla.js` in the project root. When other developers desire to use Humla in their own projects, they can simply require "humla" module. Example of Humla initialization looks like following:

```
var humla = require("humla");
humla.init({
        "administration":{"enable":true},
        "atom":{"enable":true},
        "slideindex":{"enable":true}
},true);
```

[4]Express.JS uses middlewares as tools to interact, manipulate and filter data in incoming request.

```
                                                                    7
console.log(humla.administration);                                  8
```

This example loads Humla (line 1), then Humla loads extensions as listed in the `init` method. The first parameter of the `init` method on the line 6 is object with listed extensions. This parameter is optional and when it is not provided, Humla will load all default extensions from `server-config.json`. The second parameter is boolean value from which Humla chooses if it should enable or disable database. The last line (line 8) in this example will list all available methods for administration extension.

After initialization, Humla object provides access to all enabled extensions' methods simply as attributes of `humla` variable. Example usage of extensions is following:

```
humla.administration.getCourse("MI-MDW",undefined,                  1
    function(err,data){                                             2
        if(!err) console.log(JSON.stringify(data));                 3
        else console.log("MI-MDW not found");                       4
});                                                                 5
```

In this example Humla returns course data for course with `courseID` *"MI-MDW"*.

3.3.2 Humla module definition

In the file `package.json` is definition of the whole structure of Humla environment with dependencies. As shown on Listing 3.2 the versions of modules and of Node.js are defined, so the NPM is able to download and install all necessary modules if needed.

```
{                                                                   1
    "version"       : "0.1.0",                                      2
    "name"          : "Humla",                                      3
    "description"   : "Presentation Environment.",                  4
    ...                                                             5
    "dependencies"  : {                                             6
        "express": ">= 2.4.1",                                      7
        "jsdom": ">= 2.0.1"                                         8
        ...                                                         9
    },                                                              10
    "engines": {                                                    11
        "node": ">= 0.4.1 < 0.7.0"                                  12
    },                                                              13
    "repository": "git://github.com/bubersson/humla",               14
    "main": "index"                                                 15
}                                                                   16
```

Listing 3.2: Humla module definitions.

This definition file is required for publishing of Humla into the standard NPM modules registry. It describes required versions of all modules and the version of Node.js.

3.3.3 Database

For mapping of database entries are used Mongoose schemata stored in folder `models/`. Every model has definition of the schema and then the server registers all schemata to Mongoose's internal models array. To illustrate schema definition see the following shortened version of `CommentSchema`:

```
var CommentSchema = new Schema({                                    1
    courseID: String,                                               2
    body: String });                                                3
mongoose.model('Comment', CommentSchema);                           4
```

On the line 4 is registration of the new Comment model and after this step extensions can access Comment schema using:

```
var Comment = mongoose.model("Comment");                            1
```

Models are loaded automatically during server start-up and accessible from all handlers and extensions.

3.3.4 REST API

Humla provides API for interacting with data using RESTful. Every user can interact with public REST endpoints. Also the communication between Client and Server is done using the same REST API's.

3.3.5 Extensions

Server loads extensions during initialization process using Node's `require()` method. Each extension then registers its routes on globally accessible `app` variable that represents Express.js server.

Extensions are simple JavaScript files, which are stored in `/server_ext/` folder. It's a good habit to create separate folder for each extension. For server extensions no explicit initialization in extension file is needed, but path to extension has to be defined in `plugins` array in `server-config.json`. For example extension called "example", the definition should look like this:

```
"plugins": [                                                        1
  { "id":"example","enable":true,                                   2
    "src":"./server_ext/example/example_ext.js"},                   3
...]                                                                4
```

The JavaScript object in `plugins` array also includes the `enabled` property to setup if extension should be loaded during server initialization. Extensions contain two types of functions. Private functions are accessible only from

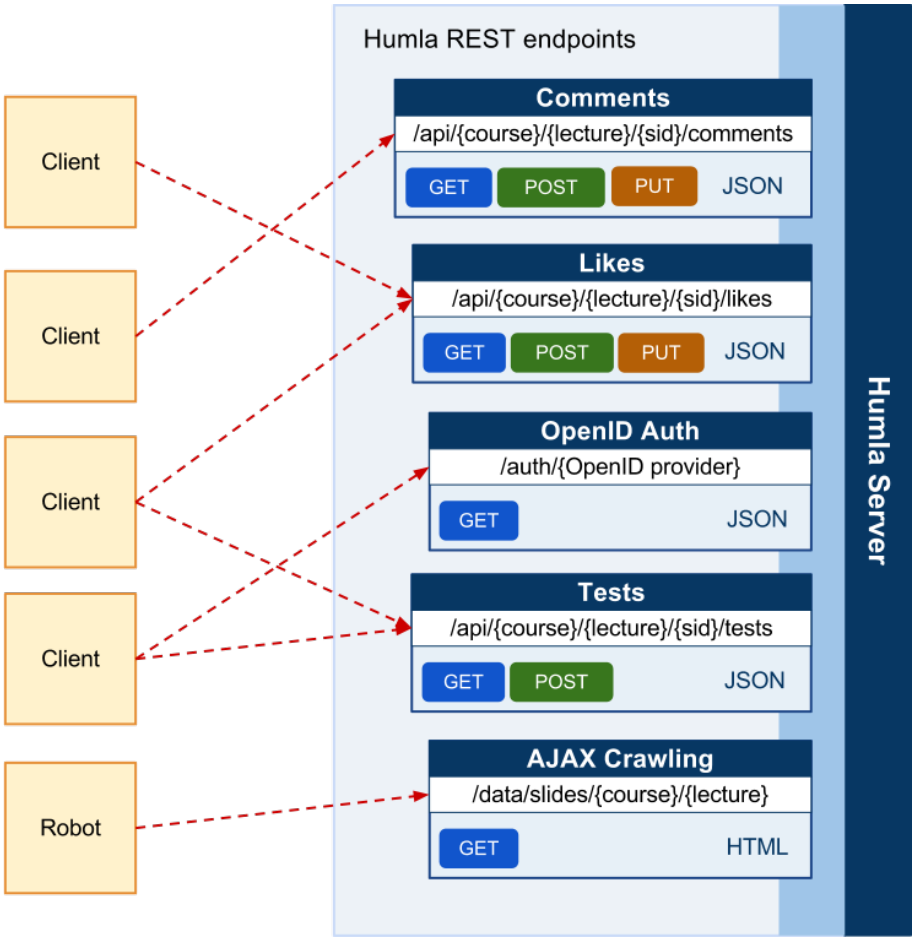

Figure 3.3: Humla REST Endpoints

the current file and globally accessible functions are accessible in the whole project. Global functions are assigned to the `exports` global Node.js variable. This provides the way to encapsulate functionality and provide only desired interface.

Comments

The Comments extension is a pair of server and client extensions. On the server side, it is simple RESTful API that provides the way to list all comments for selected slide. The client side is discussed in Section 3.4.3. The route for Express.js is following:

```
app.get('/api/:course/:lecture/:slide/comments',
    function(req,res,next){
    ...
```

```
});
```

During path resolution Express.js replaces all strings that start with a colon for actually passed values and injects them into the request object variable (`req`) passed to the callback function.

When client Humla application, or any other HTTP client, makes a GET request to this URI and provides `courseid`, `lectureid` and `slideid`, then Comments extension makes a query to MongoDB and returns an array of comments serialized in JSON.

The client can save new comment by sending a POST request to the same URI with a comment text as a body of this request, but the User has to be authorized. When user is not authorized, the server will return HTTP Code 401 Unauthorized.

Likes

The Likes extension is quite similar to the comments extensions in the sense of implementation. Instead of comments users vote on slides. The Express.js path is following:

```
'/api/:course/:lecture/:slide/likes'
```

For GET request on this URI the extension returns number of likes and dislikes associated with selected slide. With POST request users can vote on selected slide. Then URI is slightly different:

```
'/api/:course/:lecture/:slide/likes/:op'
```

The `:op` parameter can be either *"like"* or *"dislike"*.

When User tries to vote on one slide more than once, this extension checks whether he or she is listed in between users that already voted. If the same user is found Humla won't count his second vote.

Ajax Crawling

AJAX Crawling extension checks if the query parameter in received URI is *"_escaped_fragment_"*. If it is, the extension will return only the HTML of selected slide instead of the whole presentation. This makes AJAX crawling possible so the crawling robots can index single slides and then provide links to them without need to execute Humla JavaScript core files.

Crawler (for example Google) searches through the page and finds pretty AJAX URLs containing fragment part. Rather than using a hash, `#`, the Ajax Crawling protocol requires using a hash and an exclamation point: `#!` (this is usually called *"hashbang"*). Then the crawler requests the content with *"_escaped_fragment_"* query parameter instead of the hashbang parameter. Humla provides single slide HTML page for this URL so Crawler can index only the desired data.

3.4 Client

The client is responsible for the presentation itself. It provides different ways of showing slides and interacting with users. The client can be configured via `config.json` file which is in the `/public/humla/lib/` folder.

3.4.1 Architecture

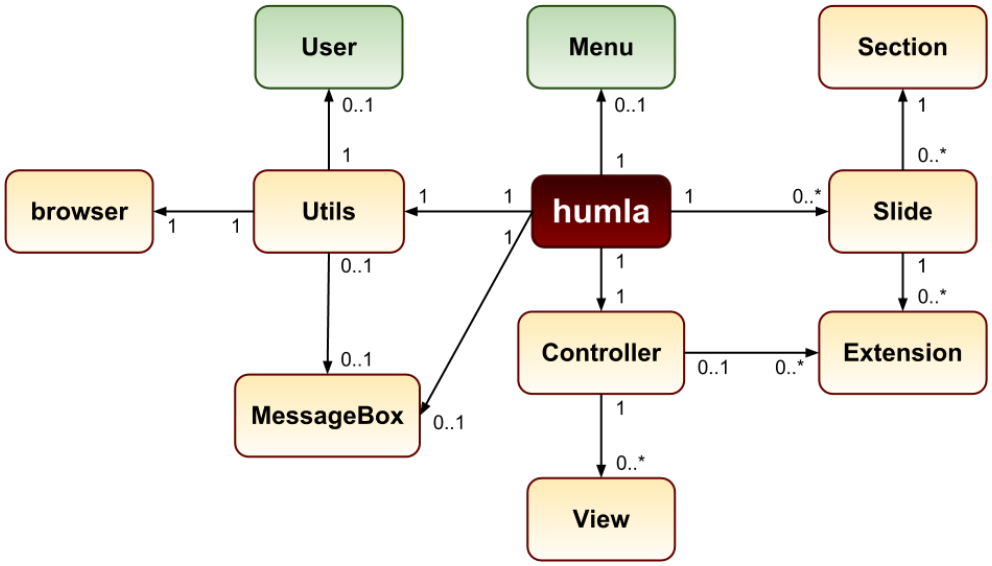

Figure 3.4: Humla client architecture

The architecture and structure of the client-side Humla remains almost the same as in the original Humla, although various parts were slightly modified and extended. There are two new classes, *Menu* and *User*, which provide common user interaction interface and ways to authenticate user, respectively. As shown in Figure 3.4 Humla builds on top of Model-View-Controller architecture, where array of *Slides* can be referred as a Model of the whole presentation.

The following list explains roles of all objects in Humla:

- **Humla** is the main class and for every active page (i.e. one browser Tab) there is only one running instance of Humla. In the system runs only one instance of Humla which loads the code written by users and transforms it into a presentation. It uses an instance of Utils class to load data about extensions and views from configuration files and a Controller object to load them into the system afterwards. Humla objects also create

listeners for pressed keys. This object will start the presentation using an instance of Controller class. Humla class also contains an instance of Utils class. Humla class also provides a way to store and show errors from the code. Humla contains all Slides of the presentation which are loaded on the Window load.

- **Controller** Controller class is also a singleton. It has only one instance which is an attribute of Humla class. It manages to load the scripts and extensions and holds a list of instances of Extension class and View class. The Controller decides what view is the current view and enables us to change it. It is used to run all the Extensions.
- **Utils** class loads the HTML document and parses it into logical parts (head, body). It also provides methods to create HTML elements (scripts, styles) and parses JSON data from configuration files.
- **MessageBox** enables the client application to show messages and errors to the user. It allows user to dismiss messages and other classes to add messages.
- **Browser** is a singleton class representing the user's browser. It is used to determine whether the browser in its current version is supported or not (older browsers do not support modern HTML5 features).
- **Slide** class represents a single slide of a presentation. It contains information about the Section the slide belongs to and about the contents of the slide. Slide has a title and a footer and contains methods to process the slide and load all extensions it needs.
- **Section** class represents a Section in a HTML code which contains either a group of Slides and/or a group of Sections. It is always an element from the HTML document (usually div element).
- **View** class represents a way the presentation is represented to the user. The document can offer several different Views according to the desired use of the document. For example one View can be used to show the presentation during a lecture and another one can be used to print the document. The View contains a current Slide which is to be shown to the user.
- **Extension** class represents any extension loaded from the configuration file. It usually contains JavaScript file and optional CSS style.
- **Menu** enables to show menu on the bottom of each slide. There is only one default item in the menu — selection of Views, but you can add other items from extensions using `processMenu` handler. The menu itself is loaded during the switch of Views, or the initial load of default View.

- **User** class represents a currently logged user. It provides public method `isLogged(cb)` to check whether the user is logged in (using OpenID). This is only client-side check used for deciding what user should see in terms of UI. Posting new comments, for example, is still performed as a server-side check.

3.4.2 Presentation Structure

There are two possible ways to create presentation. The first is using our Portal application for creating a new blank presentation and second is to create presentation in one of the common editors as a standard HTML file. If user chooses to edit HTML source, he has to insert between `<head></head>` tags the following code:

```
<link type="text/css" rel="stylesheet"
      href="/lib/core/humla.css">
<script type="text/javascript"
      src="/lib/humla.js"></script>
```

User has to provide correct URLs to `humla.js` script and `humla.css` cascade style sheet. The script `humla.js` loads other important scripts itself.

Inside `<body>` tag we can structure our presentation using standard semantic HTML5 tags. First we can divide our presentation into the sections `<section>`(but it's not required), which represents a group of slides. Each section should have its header (using `<header>` tag) and then one or more slides (noted as `<div class="slide">`). Apart from that we can nest sections into each other (supported level of nesting is 2).

So the basic one section, one slide presentation would look like this:

```
<section>
    <header>Humla</header>
    <div class="slide">
        <hgroup>
            <h1>Framework Humla</h1>
        </hgroup>
        <ul class="small">
            <li>Humla is</li>
            <li><ul>
                <li>Presentation environment</li>
            </ul></li>
        </ul>
    </div>
</section>
```

Elements like `<code>` and `<table>` will also work as expected and they are styled correctly.

3.4.3 Extensions

All client-side extensions (plugins) are stored in `/humla/lib/ext` directory. Every extension has to be enabled in `config.json`, which also specifies which JS and CSS files should the extension load[5] as well as other parameters. When there are more than just few files it's reasonable to put all inside a folder as the structure becomes more readable.

Simple example extension, that yields *"Hello world"* for every slide, would look like this:

```
var ex_example = {
    enterSlide: function(slide){
        alert('Hello world');
    }
};
```

Humla loads the file with this extensions and then try to call all methods as listed in Table 3.1.

Method
`enterSlide(slide)`
`processSlide(slide)`
`leaveSlide(slide)`
`leaveView(view)`
`enterView(view)`
`processMenu(menu)`

Table 3.1: Extension Methods

Apart from what I've implemented, Humla provides following extensions in standard installation: SlideIndex, Github, Outline, Params, Latex, Syntax highlighter, RSS feed, Google Analytics, Google Drawings, Google Books. Usage examples can be found in Humla documentation. In figure 3.5 a contextual diagram with the content that Humla interacts with is shown. Each extension interacts with different service.

Menu API

One of the most visible extensions is the Menu. It is visible on every slide, although with CSS style `opacity:0.3`, so it won't distract listeners from the presentation. It gets a bigger opacity as the mouse hovers above it. In default setting and without any other extensions, Menu only allows switching between different Views. However when other extensions are provided, Menu becomes more useful. It provides a common user interface for all other extensions that

[5] Good habit is to name all files with the same name as the extension has (e.g. likes.js and likes.css).

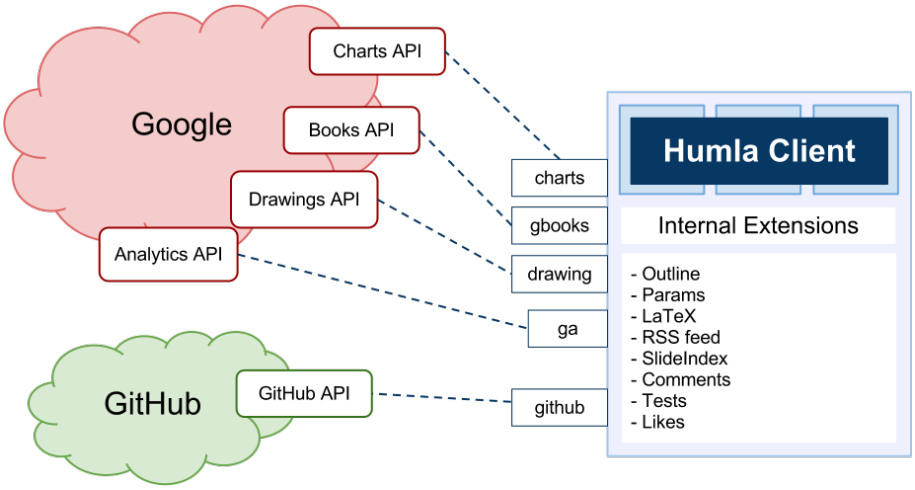

Figure 3.5: Humla integration of web APIs

desire to interact with user. If extension wants to be represented as a menu item, it has to implement method `processMenu(menu)`. Example extension showing menu item and menu layer would be defined like following:

```
var ex_example = {
    processMenu: function(menu) {
        menu.addTab("example",{
            name:"Example",
            show_layer:true,
            enable_login:true,
            html:"<h1>Example</h1>"
                +"<div>Menu Layer</div>"
        });
    }
}
```

Menu layer represents the bigger visible container, which is usually filled with extension settings. Menu layer can be switched on with `show_layer:true` parameter as shown above.

Comments

If the Comments extension is enabled, users can add a comment on every slide. Menu layer is used for displaying comments and a `textarea` for posting a new comment. When user adds a new comment, then this extension sends AJAX POST request with the comment text and further information about slides to the Comments REST endpoint. Humla Server then stores a comment to the database and returns information, whether the operation was successful or not.

Likes

Likes extension is again quite similar to the Comments extension. It also has a designated Menu item, it shows a Menu layer where user can vote for the slide.

Social Networks Integration

This extension provides ways to share presentations on three leading social networks. Twitter, Google+ and Facebook. This extension adds a Menu item with clickable icons to share current slide on selected network. To be able to share links to slide, it is necessary to be logged on to a selected network; otherwise the login will be requested.

Tests

Tests are intended for Listeners to refresh their knowledge after some major part in lecture. Author of the presentation can simply add a new slide, but in this case with `div` element attribute `class="slide test"`. For example the test with one simple question with three answers and only one of those correct would look like following:

```
<div class="slide test">
    <hgroup><h1>Test header</h1></hgroup>
    <div class="question">
        <h2>Do you like this test?</h2>
        <ul class="answers">
            <li class="true">I love it!</li>
            <li class="false">No, I don't.</li>
            <li class="false">I don't know.</li>
        </ul>
    </div>
</div>
```

One question is surrounded by `<div>` element with `class="question"`. Inside this element is a `<h2>` header and unordered list with `class="answers"` Every item in this list represents one answer to the question. The answer can be either right or wrong, which is defined by class `"true"` or `"false"` respectively. Note that those classes are not visible to listeners since the whole slide inner content is regenerated in client and correct answers are cached in `ex_tests.results` object.

3.5 Portal

Portal is simple set of HTML5 pages with JavaScript page-script which sends requests to server REST endpoints. It lets user to search for lectures and to view additional information about lectures and server statistics.

On the main page it's possible to search through courses and lectures using simple jQuery pattern matcher and filter.

Portal also allows users to create and edit courses and lectures details, although user has to be logged in to be able to create course or lecture.

3.5.1 Authentication

For authentication Humla doesn't force user to remember his own password and username, but instead it uses some of the common servers where user may have been already registered. There are two possible ways of that kind of authentication: Pseudo-Authentication using OAuth and OpenID. With OAuth application gets a "key" to access to the user account, whereas with OpenID the application just gets information about user. Humla uses OpenID, because it doesn't intend to commit any changes to users' account (e.g. Gmail account) and it doesn't need to get any information about user apart from username.

During OpenID login authentication the sequence of steps has to be taken. It involves communication between our Humla, OpenID provider and the end user. In Figure 3.6 are shown steps that are taken during whole process.

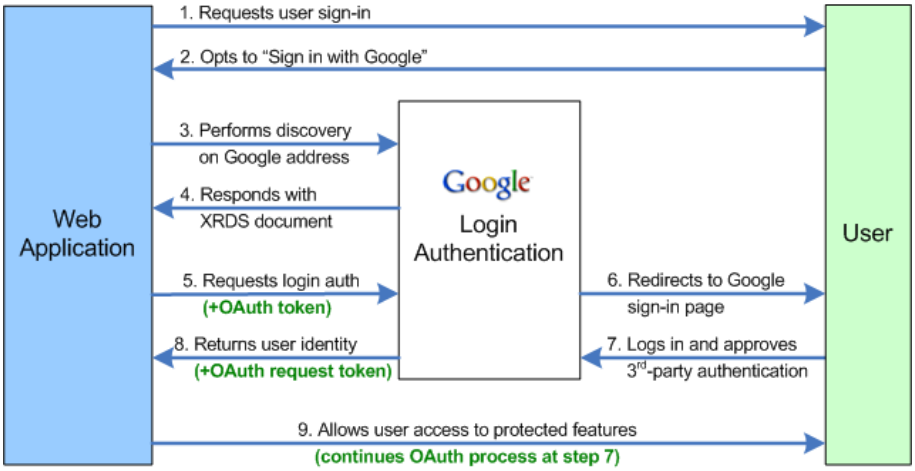

Figure 3.6: Sequence diagram of OpenID authentication (Courtesy: [1])

1. In the first step Humla shows user that he has to sign in to use the desired functionality,

2. User selects to sign in using one of the provided authentication end-points,

3. Humla authorizes user requests through authentication endpoint and if the user is not logged to that service, Humla redirects user to that service login page,

4. if the authentication endpoint returns user identity, than Humla allows user to use the desired functionality.

On the server side for OpenID authentication is used Passport Node.js module, which provides universal way to interact with the major OpenID providers.

CHAPTER 4

Evaluation

This chapter evaluates the Humla application and discusses fulfilled requirements, project status and project integration.

4.1 Requirements

Following lists of functional and non-functional requirements discuss fulfillment of requirements as stated in Chapter 1.

4.1.1 Non-functional requirements

- **Platform Independent.** Server side Humla runs under Windows, Linux and Mac OS operating systems (depending on MongoDB and Node.js installations).

  Installation and execution of server side Humla was successfully performed on selected operating systems:

  - Windows 7 64b
  - Windows XP 32b
  - Linux Mint 12 Lisa 64b
  - Mac OS 10

  Humla Portal application runs in all modern browsers (Google Chrome, Safari, Opera, Firefox and Internet Explorer).

  Humla Client application is able to run in the web browsers listed in Table 4.1. Note that Microsoft Internet Explorer 9 doesn't support used HTML5 and CSS3 features, so it is not supported. Also Opera and Firefox may have some problems with displaying slides due to unsupported HTML5 features and DOM differences.

Browser	Version	Result
Google Chrome	13 or higher	OK
Safari	5.1 or higher	OK
Opera	12	minor CSS problems
Firefox	11	datafields problems
Internet Explorer	9	Not supported

Table 4.1: Supported browsers with versions

- **Various Devices Support.** Humla runs on iPads, iPhones, Android smart-phones and tablets with WebKit based browser and also on the personal computers with supported browser.
- **Used Technologies.** Technologies remain the same as defined in Chapter 1.
- **RESTful API.** This requirement is fulfilled by Humla server-side application.

4.1.2 Functional requirements

- **Managing courses and presentations.** Requirement is fulfilled by Portal application. User can click on the "plus" button to add new course or lecture, although user has to be logged in.
- **Browsing through lectures.** Requirement is fulfilled by Portal application. User can filter and select both courses and lectures. When selecting a lecture, additional info like lecture index and abstract is shown.
- **Tests in presentations.** This requirement is fulfilled by tests extension in Humla client. Creator of the presentation can simply add test questions at the end of presentation using built-in editor.
- **Menu in presentations.** Menu bar is visible on every slide and it is implemented in Humla Client Core. Extensions can add new tabs to menu bar and bind JavaScript event listeners to them.
- **Adding Comments on slides.** Every logged user can add comment on each slide thanks to comments extension in Humla client.
- **Adding Likes on slides.** Logged user can like or dislike selected slides using likes extension.
- **Administration.** For setup it's necessary to edit `server-config.json` file in the project root. In this file is list of enabled extensions.

- **Social networks integration.** Every user can share presentation on Facebook, Twitter and Google+.
- **Different sizes of presentations.** Presentation can contain any number of slides. Also thanks to Node.js' nature it is no problem to interact with many users at the same time.

4.2 Integration

Humla is versatile presentation environment, that can act in 3 different roles as defined in the Chapter 1. Each role has its pros and cons and each role is convenient in different situations. Following example situation provides short overview of two possible ways of integrating Humla with current systems.

4.2.1 Example situation

At the Czech Technical University in Prague on the Faculty of Information Technology a portal and content management system using open-source enterprise applications Liferay[1] and Alfresco[2] are currently being deployed. This combination, apart from other functionalities, provides web interface for interacting with files. It is useful to store presentations in this system and thus prevent redundancies and solve problems with versioning and file hosting. There are two simple solutions to integrate those systems with Humla.

Solution 1

The first solution is to host Humla as a Front-end only framework. This solution is quite simple in terms of deployment. The only required modification or configuration of the CMS Alfresco is to setup a common storage for front-end only Humla files. Then users or teachers can host HTML files of their lectures only with their dependencies (e.g. images). In Humla a link to the `humla.js` and `humla.css` has to be set up accordingly and also `config.json` would have to be changed to disable all Humla server dependent extensions (e.g. likes, comments).

Both HTML presentation files and Humla core files should be hosted on the same domain to prevent problems with Cross Origin Policy. Although it's possible to solve this problems using CORS protocol, Alfresco would have to be configured accordingly.

Solution 2

The second solution is to run standalone Humla server, which is able to host all presentations files. Humla server provides a way to create new presenta-

[1] http://www.liferay.com/
[2] http://www.alfresco.com/

tions, it provides full support for comments, likes and other *"social"* interactions. The requirement for this solution is to configure Humla server (via key *"slides_raw_path"* in `server-config.json`) to store presentations in Alfresco data store (using SMB/CIFS, FTP or WebDAV).

Conclusion

Although the deployment of the portal and CMS systems on the Faculty of Information Technology still isn't in its final stage, the integration with Humla is possible. Similar to this particular problem would be the integration with other systems.

4.3 Project License and Hosting

Humla is developed under the GPL version 3[3], which grants its recipients rights to copy, modify and redistribute the software and ensure that the same rights were preserved in all derivative works. Apart from that GPLv3 is compatible with Apache License 2.0.

The project itself is hosted on `http://github.com` and as said on the Github page: "We make it easier to collaborate with others and share your projects with the universe". Those promises are not far from truth. Github is one of the largest code hosting sites in the world with almost 2.5 millions repositories[4]. Github is based on Git, the Distributed Version Control Software. Github provides social development, so apart from abilities to fork or clone anybody's repositories and host your source codes, it is also possible to comment on each commit, create wikis, or report found bugs in integrated issue tracking system.

User with preinstalled Git can download Humla with following short console command:

```
git clone git://github.com/bubersson/humla.git
```

Then the Humla is cloned to current directory. For further information see Appendix B.

Humla also provides installation script for Debian based operating systems (see `ubuntu_debian_installer.sh` in the project root).

[3] License agreement can be found at `http://www.gnu.org/licenses/gpl.txt`.
[4] The number is still growing as shown at `https://github.com/home`

Conclusion

I've designed and implemented full-range framework on top of original project Humla, which covers creation, modification and presentation of lectures. I've implemented back-end server using JavaScript server-side technology Node.js. My solution is fully configurable and extendable using extensions. I've also added extensions, which enable users to interact with each other and give a feedback to a lecturer. For each lecture I've added possibility for listeners to post their comments and their rating of slides. Author of presentations can also add a simple test at the end of the presentation for refreshing students' memory.

In the first two chapters I've provided analysis and research on available technologies suited for implementation.

In the second part of this thesis I've presented design and implementation of a server, which is able to host lectures and provide REST APIs for interaction with details about lectures. Building web applications using Node.js does not represent any problem. When JavaScript is used properly it's possible to take advantage of its most useful features and its event-based nature with lambda functions and closures. JavaScript seems to be usable for this kind of tasks and still simple enough to reach wide audience of developers.

Future Improvements

Resulting implementation for sure isn't complete, but current state provides platform for creation of many different kinds of extensions. Since changes in Node.js related projects happen every day, there are many possibilities as the Node.js community grows.

This project doesn't have to be extended only by Node.js extensions, there also could be for example desktop application for lectures creation with WYSIWYG editor, or installation and configuration application with *"single-click"* controls.

One of the improvements that could also be done is to support more web browsers. Humla is built on top of new technologies and standards and therefore the client application fully supports only few current browsers (Google Chrome and Safari) in this moment.

Thanks to current Humla architecture based on separated components, there's a room for various projects and extensions. Humla can also serve as a source for so-called mashup applications, which combine data or functionalities from two or more applications.

Bibliography

[1] Federated Login for Google Account Users. April 2012. Available at WWW: https://developers.google.com/accounts/docs/OpenID

[2] Browne, J.: Brewer's CAP Theorem. January 2009. Available at WWW: http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

[3] Crockford, D.: *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[4] Dziuba, T.: Node.js is Cancer. October 2011. Available at WWW: http://teddziuba.com/2011/10/node-js-is-cancer.html

[5] Essel, M.: Dynamic scripting with static speed, the best of both worlds. January 2011. Available at WWW: http://www.victusspiritus.com/2011/01/28/dynamic-scripting-with-static-speed-the-best-of-both-worlds

[6] Fasel, M.: Node.js From the Enterprise Java Perspective. June 2011.

[7] Fielding, R. T.: *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[8] Flanagan, D.: *JavaScript: The Definitive Guide Activate Your Web Pages*. O'Reilly Media, Inc., 6th edition, 2011.

[9] Graham, P.: Web 2.0. November 2005. Available at WWW: http://www.paulgraham.com/web20.html

[10] Hoon, L. S.: Visual Guide to NoSQL Systems. March 2011. Available at WWW: http://blog.beany.co.kr/archives/275

[11] James, J.: Features, Pain Points, Adoption Rate, and More. August 2008. Available at WWW: http://tek.io/fhTEjf

[12] Lane, K.: History of APIs. January 2011. Available at WWW: http://www.apievangelist.com/2011/01/26/history-of-apis-ebay/

[13] Lee, T. B.: Information Management: A Proposal. March 1989. Available at WWW: http://www.w3.org/History/1989/proposal.html

[14] Lundström, Sebastian: Design and Implementation of a MongoDB Driver for Prolog. 2011.

[15] Munandar, F.: The Increasing Importance of APIs in Web Development. October 2011. Available at WWW: http://fredonfire.com/2011/10/the-increasing-importance-of-apis-in-web-development/

[16] Needle, D.: Peer Network Tests 'Real World' Site Performance. May 2008. Available at WWW: http://www.internetnews.com/infra/article.php/3744331/Peer+Network+Tests+Real+World+Site+Performance.htm

[17] Nielsen, J.; Mack, R.: *Usability Inspection Methods*. New York, NY: John Wiley & Sons, 1994.

[18] Pilgrim, M.: *HTML5: Up and Running*. O'Reilly Media, Inc., August 2010.

[19] Reynen, S.: A Brief History of HTML. September 2009. Available at WWW: http://atendesigngroup.com/blog/brief-history-of-html

[20] Souders, S.: *High performance web sites*. O'Reilly Media, Inc., first edition, 2007.

[21] Vitvar, T.: Lecture 6: Representational State Transfer. University Lecture, 2012. Available at WWW: http://vitvar.com/courses/slides/mdw/lecture6.html

APPENDIX A

Acronyms

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

BSON Binary JSON

CORS Cross Origin Resourse Policy

CSS Cascading Style Sheets

DBMS Databse Management System

DOM Document Object Model

ECMA European Computer Manufacturers Association

GPL General Public License

GUI Graphical User Interface

HATEOAS Hypertext as the Engine of Application State

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IT Information Technologies

JSON JavaScript Object Notation

NPM Node Package Manager

OS Operating System

PDF Portable Document Format

REPL Read–Eval–Print Loop

REST Representational State Transfer

SOAP Simple Object Access Protocol

SQL Structured Query Language

UI User Interface

UML Unified Modeling Language

URI Uniform Resource Identifier

URI Uniform Resource Locator

WADL Web Application Description Language

WYSIWYG What You See Is What You Get

XML Extensible Markup Language

APPENDIX B

Installation

Installation is pretty straight forward and consists of installation the Humla itself using Git and then of optional Node.js backend server and optional MongoDB database.

Humla also provides installation script for Debian based operating systems (see `ubuntu_debian_installer.sh` in the project root).

Step by step guide for installation on Windows (other operating systems installations are similar):

1. Download Node.js server from http://nodejs.org/. Installation executables for Windows and Mac as well as packages for Linux are provided.
2. Download MongoDB database from http://mongodb.org/. Then either follow installation guide at http://www.mongodb.org/display/DOCS/Quickstart+Windows or do following steps:
   a) Unzip the downloaded binary package to the location of your choice,
   b) In this folder create default data directories using:

```
C:\> mkdir \data
C:\> mkdir \data\db
```

   c) Run the database itself using `mongod.exe`,
   d) To check if the database is ready start the administrative shell `mongo.exe`

```
C:\mongo_dir\bin> mongo
> db
test
> db.foo.insert( { a : 1 } )
> db.foo.find()
{ _id : ..., a : 1 }
```

3. Install Git from http://git-scm.com/. Windows and Mac installers are provided. To check if Git is ready run:

```
D:\downloads>git --version
git version 1.7.6.msysgit.0
```

4. Clone Humla to your desired directory using

```
git clone git://github.com/bubersson/humla.git
```

5. Install NPM packages with following commands (NPM is part of the Node.js installation):

```
npm install express
npm install mongoose
npm install jsdom
npm install cron
npm install passport
npm install passport-local
npm install passport-google
```

6. Run humla server using command:

```
D:\humla>node index.js
Humla (server) has started, 127.0.0.1:1338, Using
    Express 2.5.1, Node v0.6.14
```

7. You may change the configuration in `server-config.json` file.

APPENDIX C

Screenshots

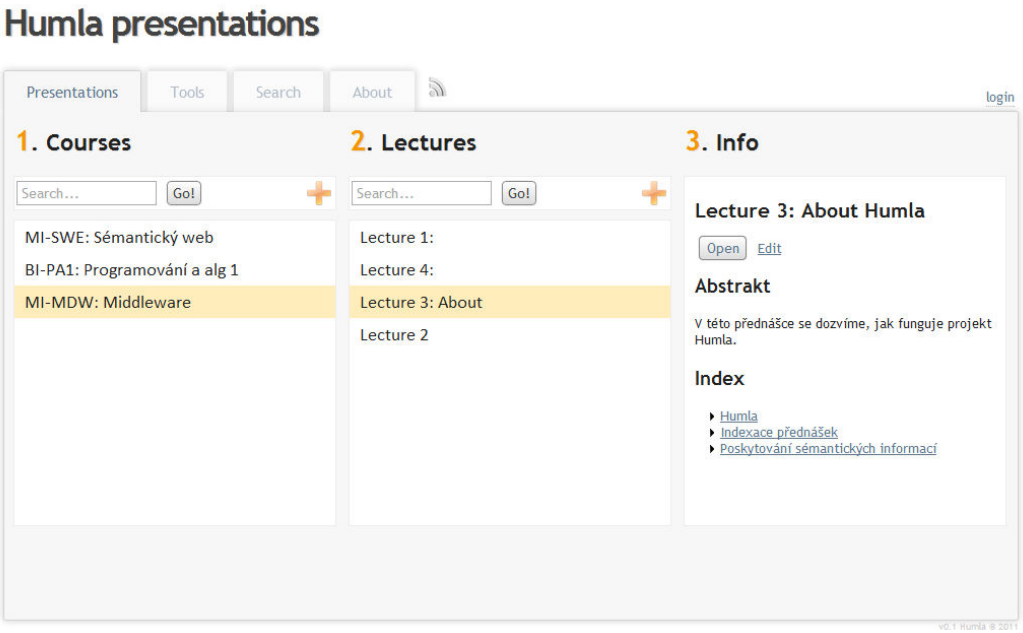

Figure C.1: Humla Portal Application

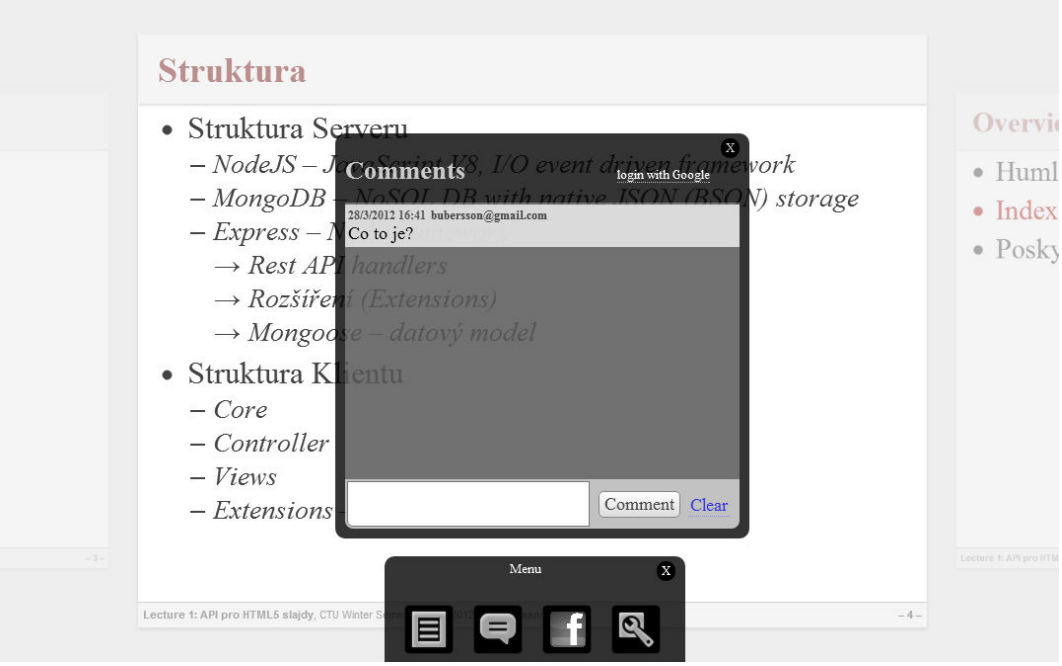

Figure C.2: Humla Client Application

APPENDIX D

Contents of enclosed CD

- `readme.txt` ..... the file with CD content description
- `src` ..... the directory of source codes
  - `humla` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format