

Kristian Wrang

LAATUPUNTARI – DEVELOPMENT AND DOCUMENTATION

Information Technology

Software engineering

2012

LAATUPUNTARIN KEHITYS JA DOKUMENTOINTI

Wrang, Kristian
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Joulukuu 2012
Ohjaaja: Laine, Kari
Sivumäärä: 74
Liitteitä: 0

Asiasanat: living lab, ohjelmointi, verkko-ohjelmointi

Tämän työn tarkoituksena oli rakentaa Laaturpuntari, Living lab – arviointityökalu, keskeneräisestä aiheesta valmiiksi tuotteeksi ja dokumentoida ohjelmointiprosessi. Dokumentointi laadittiin, jotta Laaturpuntarin jatkokehitys ja käyttö olisi mahdollisimman helppoa.

Työssä käytiin läpi Laaturpuntarin tilannetta vuoden 2012 alussa ohjelmoijan vaihtuessa ja työvaiheita, joiden kautta päästiin valmiiseen tuotokseen. Työssä tutkittiin työkalut, joita Laaturpuntarissa käytetään ja ominaisuudet, joita tarvittiin. Työssä selvitettiin parhaat tavat toteuttaa kukin tarvittu ominaisuus. Käytiin läpi ohjelmistokehitystä varten vaadittavat ohjelmat, niiden asennus ja käyttöönotto. Tämän jälkeen tarkasteltiin uusien tarvittujen ominaisuuksien luonti ajatusasteelta valmiiksi, testatuiksi ja toimiviksi. Lisäksi tutkittiin, miten Laaturpuntari otetaan käyttöön uudelle palvelimelle tai uuteen kehitysympäristöön.

Lopuksi todettiin, mitä materiaalista tulosta työllä oli ja mitä taitoja Laaturpuntarin kehitys toi tekijälleen.

LAATUPUNTARI – DEVELOPMENT AND DOCUMENTATION

Wrang, Kristian

Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences

Degree Programme in Information Technology

December 2012

Supervisor: Kari, Laine

Number of pages: 74

Appendices: 0

Keywords: living lab, programming, web programming

The purpose of this thesis was to develop Laatupuntari, a Living lab reviewing tool, from its basic form to a polished and ready tool and document the process. The document was written to aid in possible continued development and usage of Laatupuntari.

The topics covered in the start included the status of Laatupuntari in the beginning of 2012 when the developer changed and the stages of work that lead to the finished product. In the planning stage the tools used in Laatupuntari and the requirements are covered. The best methods of implementing each required feature are thought through. Software required for web developing are discussed as well as the installation and usage of those programs. Installation of Laatupuntari on a new server and cloning from version control and developing on a new machine are covered. The creation of new, required features is covered from the idea stage to ready, tested and working parts of Laatupuntari.

In the end the outcome of the project is discussed and what making Laatupuntari taught to its creator.

CONTENTS

1	INTRODUCTION.....	7
2	REQUIREMENTS	7
2.1	Main functionality.....	7
2.2	Additional functionality	8
3	TECHNOLOGIES.....	8
3.1	Server-side	8
3.1.1	CakePHP	8
3.1.2	MySQL	9
3.2	Client-side	9
3.2.1	jQuery	9
3.2.2	XHTML and CSS	10
3.3	Add-ons and other libraries.....	10
3.3.1	dompok	10
3.3.2	reCAPTCHA	11
3.4	Coding tools	12
3.4.1	Eclipse	12
3.4.2	Bitbucket	12
3.4.3	WampServer	13
3.4.4	Git for Windows	13
4	PLANNING.....	13
4.1	What was already done	14
4.1.1	Database	14
4.1.2	Site	15
4.1.3	Dynamic questionnaire	17
4.1.4	PDF creation	18
4.2	What needed to be done	18
4.2.1	Question categories	18
4.2.2	Report saving	19
4.2.3	Analysis of questionnaires.....	19
4.2.4	Listing and viewing old reports	20
4.2.5	User control	20
4.2.6	User feedback	21
4.2.7	Questionnaire database tool.....	21
4.2.8	Miscellaneous smaller features.....	22
4.2.9	Bug hunting	23
5	PROGRAMMING.....	24
5.1	Getting started.....	24

5.1.1	Installing WAMP and Eclipse	24
5.1.2	Setting up a local copy of Laatupuntari.....	33
5.2	Starting with the easy ones	34
5.2.1	Progress indicator	34
5.2.2	View previous question	34
5.2.3	Prevent going forward without answering	36
5.2.4	Answer exclusion	36
5.2.5	Question exclusion	37
5.2.6	Selective enabling of input boxes.....	39
5.2.7	Text input	41
5.2.8	Client-side element naming convention	42
5.2.9	‘Ready’ -button.....	43
5.3	Off to the bigger challenges	43
5.3.1	Report saving	43
5.3.2	Tool for adding questions to database	45
5.3.3	Enabling the last two categories	46
5.3.4	Analysis of reports.....	47
5.3.5	User accounts	48
5.3.6	Listing of reports	51
5.3.7	Viewing of old reports.....	52
5.3.8	Enhancing PDF printing	53
5.3.9	User feedback	54
6	CLONING LAATUPUNTARI FROM GIT	56
6.1	Installing msysgit	56
6.2	Cloning Laatupuntari from Bitbucket.....	58
6.3	Downloading CakePHP	59
6.4	Downloading jQuery.....	60
6.5	Downloading dompdf	61
6.6	Downloading reCAPTCHA-plugin	62
6.7	Configuration and possible problems	64
7	FINAL DATABASE	68
8	WHAT WAS LEARNED	70
8.1	Working in a project as part of a team.....	70
8.2	Using new tools.....	71
8.3	Laatupuntari now and possibilities in the future.....	71
	REFERENCES.....	73

GLOSSARY

ESF	European Social Fund
PHP	PHP: Hypertext Preprocessor
MVC	Model View Controller
SQL	Structured Query Language
Windows	Family of operating systems by Microsoft
Linux	A Unix-like operating system running on the Linux kernel
JavaScript	A scripting language commonly used in the creation of dynamic websites
Open-source	Software with source code freely available
UI	User Interface
HTML	Hypertext Markup Language
XHTML	eXtensible Hypertext Markup Language
CSS	Cascading Style Sheets
PDF	Portable Document Format
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
IDE	Integrated Development Environment
Java	An object-oriented programming language, typically compiled to bytecode to be run on a Java virtual machine on any platform
Git	A distributed revision control and source code management system created by Linus Torvalds to replace BitKeeper
OS X	The operating system used in Apple computers
Apache	An open-source web server, officially known as Apache HTTP Server
GUI	Graphical User Interface
ID	Identifier
Primary Key	A unique key to define the characteristics of every database record
JSON	JavaScript Object Notation, a syntax for storing and exchanging text information
HTTP POST	One of several request methods supported by HTTP
Google Chrome	Google's web browser based on WebKit
phpMyAdmin	A free open-source tool for managing a MySQL server
Zip	A file format for data compression and archiving
Serialization	The process of converting a data object into a format more suitable for storing in a database or a file
HTTP GET	An HTTP request method where additional data is transmitted along the URL
URL	Uniform Resource Locator
AJAX	Asynchronous JavaScript And XML
Bash	Bourne again shell
OpenSSH	OpenBSD Secure Shell
Unix	A multitasking, multiuser operating system, the grandfather of nearly every operating system used today

1 INTRODUCTION

The purpose of this thesis is to create a web-based tool for analyzing living lab - projects. The tool itself is a dynamic questionnaire that gives you scoring on your answers and stores everything for future reference.

Laatupuntari is part of a project funded by ESF called “Ammattikorkeakoulujen neloskierre” that consists of several different Living Lab -related tools and other projects (Website of Neloskierre 2012).

In this written part of the thesis I will go through the planning and development of Laatupuntari. One goal is to serve as documentation for possible future developers and administrators, if this project is to be continued or installed on a new server.

2 REQUIREMENTS

In this chapter I will go through the requirements set for this project. What is necessary and what features are to be added if there are still resources left after the main functionality has been implemented.

2.1 Main functionality

The goal is to build a tool that allows the presentation of a dynamic series of questions and an analysis based on how the user answers those questions.

The basic idea is that we have three categories of questions, each consisting of three or four different Living lab maturity levels. The user has to be able to easily select any single maturity level from one of the three categories. After answering all questions the user is presented with what he just answered along with an analysis.

Another aspect of the core functionality is user management. Everyone must be able to register as a user to Laaturuntari. Registered users will have benefits such as storing report for viewing later.

2.2 Additional functionality

The project workgroup came up with ideas that could be implemented if resources prove adequate. A form for submitting feedback could be used to enable users to give critique about Laaturuntari. A separate form makes sure that even the smallest ideas from users are heard. Having users email their critique cuts out a part of feedback because of the additional hassle.

Any registered user could create series of questions. These questionnaires could then be commented on and after approval used in the tool for analysis.

3 TECHNOLOGIES

In the beginning of the project the technologies to be used were already decided. In this chapter I will go through server-side and client-side technologies for Laaturuntari.

3.1 Server-side

3.1.1 CakePHP

CakePHP is an open source framework to build web applications on. It's written in PHP and uses well-known concepts such as MVC. The purpose of CakePHP is to ease the work of the developer by providing simplified functionality for things like database access, form creation and user control. (Website of CakePHP 2012).

CakePHP provides excellent documentation for anyone and creating rich web applications is a breeze. Minimal configuration is required, just database access and the majority of scenarios are covered.

CakePHP is used on more than three hundred sites, web applications and blogs around the world (CakePHP: Sites in the wild 2012). The underlying technology, PHP, is currently the most widely used server-side programming language on the internet (Web Technology Surveys (W3Techs): Usage statistics and market share of PHP for websites 2012).

3.1.2 MySQL

MySQL is the world's most popular open source database. It is highly scalable, fast and well documented. Connectors, even ready-to-use libraries are available for all popular programming languages. (Website of MySQL 2012).

Being open source, MySQL is installable on nearly any platform available. Installation packages can be downloaded for Windows-based servers and every Linux distribution with a package management solution provides you with a precompiled MySQL-server.

3.2 Client-side

3.2.1 jQuery

jQuery is an open source JavaScript library to make coding more efficient. Everything from UI design and manipulation to dynamic data retrieval and logic can be handled with ease and speed. Minimal JavaScript knowledge is required, since jQuery has its own syntax that can be used exclusively. If preferred, regular JS can be inserted among jQuery when optimized routines are required for raw number crunching or similar tasks that need to be quick. (Website of jQuery 2012).

jQuery is widely used by big companies such as Google, Dell, NBC, CBS and the Bank of America (Website of jQuery 2012).

3.2.2 XHTML and CSS

To create a site that is as widely compatible as possible, strictly formatted XHTML was chosen. That stricter formatting also has a slight speed advantage, which is crucial especially on mobile devices.

According to best practice, Cascading Style Sheets should be used to provide the visual style of the site. All styling can be written inside one file and then referenced from all over the site. This not only makes editing easy since everything can be found in one place, but it speeds up browsing because the users client only has to download that CSS file once.

CSS has a good work-around to style components that don't have an established standard: filters can be created so that certain style definitions are only read by the browser they are meant for (Olsson, O'Brien 2007). This makes going around possible browser bugs easier, since you can code so it works on the majority of browsers – the ones that work properly – and make the small fixes only for the one that does not work.

3.3 Add-ons and other libraries

3.3.1 dompdf

dompdf is an HTML to PDF converter written in PHP. It handles most CSS properties and creates clean PDF files as long as it is provided with properly formatted HTML. (Website of dompdf 2012)

Since it was required that you could save a PDF report of a previously answered questionnaire, dompdf was chosen.

dompdf is open source and requires little more than PHP 5.0 or newer.

3.3.2 reCAPTCHA

reCAPTCHA is Google's free anti-bot service. The differentiating idea behind reCAPTCHA is that it helps digitize books all while providing security for your web application. One word is taken randomly from a dictionary and the other word is a scan from a book. (Website of reCAPTCHA 2012)

An easy-to-use plugin is provided for CakePHP and Google servers are used to display the security snippet.



Picture 1. A reCAPTCHA example

If presented with a word too hard to read, the refresh button can be pressed for a new set of two words. Audio playback is provided for blind people, making for example user registration accessible to a wider user base.

3.4 Coding tools

3.4.1 Eclipse

Eclipse is one of if not the most popular free IDE for software development. Eclipse was created for Java development and it is written in Java. Various plugins enable the use of all well-known programming languages, but lesser-known languages are also supported. (Website of Eclipse 2012)

With the correct plugins Eclipse recognizes PHP, HTML, CSS and JavaScript files with ease and managing a project with multiple file types is smooth. Sharing Eclipse workspaces is a possibility through version control, but can be left out if each developer has clear preferences on how they want their windows and other settings.

3.4.2 Bitbucket

Bitbucket is a version control service hosted by Atlassian. It uses Git and is free for small projects; unlimited public and private repositories with 5 users (Picture 2). (Website of Bitbucket 2012)

5 USERS	10 USERS	25 USERS	50 USERS	100 USERS	UNLIMITED
Free	\$10/mo	\$25/mo	\$50/mo	\$100/mo	\$200/mo
Current Plan	Choose	Choose	Choose	Choose	Choose

Picture 2. Bitbucket pricing plans

In addition to providing unlimited repositories to use with any Git-client available for Windows, Linux or OS X, Bitbucket also includes a website to manage your repositories or browse code. Pull requests from repository forks are easily manageable and an issue tracker is included in every repository, be it private or public. Commits and other basic version management tasks are done the same way Git works everywhere else, but several functions made available by the web UI frees up more time for coding.

3.4.3 WampServer

WampServer is a web development environment for Windows and it stands for Windows, Apache, MySQL and PHP. It comes in an easily installable package containing all the server binaries you need to locally develop and test web pages and applications. (Website of WampServer 2012)

Since uploading to a server for each test takes away precious time, having a local development environment with a complete web-server makes the developers life much easier.

3.4.4 Git for Windows

Git for Windows is a version of msysgit. It is targeted at Windows-users that need a stable and easy Git client. Git for Windows can be used from the command line or with a GUI, whichever is preferred. (Website of msysgit 2012)

4 PLANNING

The basic architecture in this project had already been done. All choices regarding languages and libraries – save for reCAPTCHA – were done by the previous programmer.

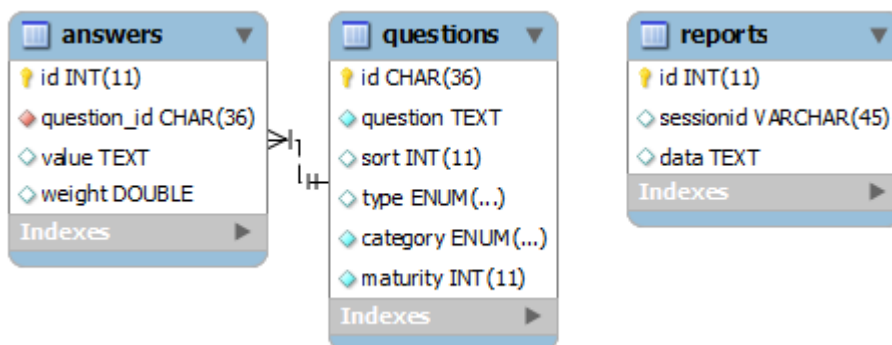
The layout of the site had been perfected over several iterations and is still in use with little modifications for usability's sake.

One of the main ideas from the start was to make the tool as dynamic as possible. Series of questions should be requested from the server according to what the user chooses. Saving a report of what the user answered should be done without user input; in the background the moment the questionnaire is finished.

4.1 What was already done

4.1.1 Database

A basic version of the database to be used was already in place.



Picture 3. Laatumuntari beta database

Questionnaires are spread into two tables: questions and answers. Questions -table includes:

- ID; generated by CakePHP. CakePHP database ID's are created to be unique, not just inside one table but across a whole system.
- actual question (for example: How would you rate the current state of your Living Lab?)
- sort; for telling the tool in which order to present chosen questions
- type; this tells us what type the answers are. Can be 'radio' for choose one, 'checkbox' for multiple choice or 'inputbox', which assumes one or more input boxes for each answer and no selectable components.
- category; choose from 'toiminta' (operation), 'case' or 'opetus' (teaching/coaching)
- maturity; for storing the different levels of Living Lab maturity. Depending on category, three or four levels are used.

The other important part to showing users series of questions comes from the answers -table:

- ID; automatically incrementing primary key

- question_id; foreign key that links answers to questions. Each question can have one to many different answers.
- value; the text containing the answer
- weight; the score value of the answer, to be used in the analysis component

Lastly, reports:

- ID; primary key, automatically incrementing
- sessionid; to easily connect browsing sessions with different reports one user might create during a day we store the session id created by CakePHP
- data; the report itself containing the answers the user chose

At first questions were placed in only one category, the rest to be added when the codebase could accommodate for them.

4.1.2 Site

[Etusivu](#) [Anna palaute](#) [Tietoja](#)

LAATUPUNTARI

Tervetuloa Laaturpuntariin!

Laaturpuntari kuvaa, mikä Living Lab- toiminnassa on laadukasta ja minkä osa-alueiden kehittämiseen tulisi jatkossa kiinnittää erityistä huomiota. Laaturpuntaria voidaan käyttää tarkistuslistana Living Lab- toiminnassa hyviksi havaittuihin käytänteisiin.

Valitse arvioitava osa-alue



Living Lab- toiminta

Miten Living Lab- toiminta on suunniteltu ja määritetty ja mitä käytäntöjä noudatetaan?



Living Lab- case

Miten Living Lab- casen toteutus on suunniteltu ja toteutettu?



Living Lab ja opetus

Miten Living Lab- toiminta integroituu ammattikorkeakoulun opetukseen ja TKI- toimintaan?



Picture 4. First beta of Laatupuntari.

The version of the site left by the previous programmer had all design elements of the home page in place. Three categories are visible as big and clear buttons. Two of the categories didn't yet do anything and the visual effect was accomplished by lowering opacity.

When the button for the first category is clicked, the user will be taken to a page that holds information about that category and the different maturity levels it contains. Instructions on how to answer the questions and how the questionnaire generally works are also presented.

When the user presses 'Aloita testi' (Start test) on that page, the questionnaire begins. First the user is presented with a question regarding the level of maturity of the Living Lab he/she wants to validate. After that question has been answered, the actual questions are presented one at a time. After all chosen questions have been answered, the user is presented with all the answers on one page. Included are several buttons: Print a PDF, Give Feedback and Back to the Homepage.

With 'Print a PDF' the server stores a PDF file containing all selected answers and some other data and sends it to the user's browser. After that it can be either saved or viewed right away, depending on the browsers settings.

'Give Feedback' opens a new browser window for the user and displays a survey on Webropol. The survey was created by other project personnel and is not part of this thesis.

Lastly we have a button to send the user back to the home page of the tool. This can be accomplished by pressing the orange and black 'Laatupuntari' -logo on the top of the page, but was created due to user feedback.

4.1.3 Dynamic questionnaire

Arvioi mikä on oman Living Lab -toiminnan ja ekosysteemin kypsyytaso. Valintasi perusteella sivusto tarjoaa kypsyytasoosi soveltuvat kysymykset.

- Living Lab -toimintaa suunnitellaan (valmisteilla)
- Toteutusta on suunniteltu ja yksittäisiä caseja toteutettu (käynnistymässä)
- Living Lab -toiminta ja ekosysteemi on vakiintunut (käynnissä)

Seuraava →



Picture 5. The static first question in ‘Toiminta’ –category.

After starting the questionnaire, the user is presented with one static question: what level of maturity is his/her Living Lab. When ‘Seuraava’ (next) is pressed, jQuery is used to read the value of these radio buttons. The value of the selected radio button is sent to the server as a JSON query. Then CakePHP’s ‘find’ is used recursively to get questions of chosen maturity and all answers linked to those questions (Golding 2008, 93).

That data is sent back to the jQuery -script. HTML to present the questions is then dynamically created from the database query results. All questions are rendered hidden and after everything has been created a function is used to fade the first question on to the users display.

After that every time a question is answered jQuery is used to fade out the one being answered and fade in the next one. By having everything already hidden in the HTML there is no need to reload the page at all.

The ‘Next’ -button is not a traditional button, but a div -element styled to look like a button. The button-style comes from class ‘button’. Additionally the class ‘next’ is added to the button for jQuery. Live click -feature from jQuery is then used to register a click on that div -element.

Example:

```
$("#div.next").live('click', function() { handle_fading; });
```

After the questionnaire is finished and last question faded out jQuery notices this is the last one. A text in the top of the page is added along with the three buttons to the bottom of the page.

4.1.4 PDF creation

PDF creation is handled server-side by rendering an HTML page and letting dompdf do its work on that page.

Form data is sent through regular HTTP POST. The data is parsed to one array and then passed on for the PDF creation. Since only the values of answers are sent through POST, a CakePHP recursive find is used to get questions and answers for those values sent by the client side.

As we now have only the answers the user selected we can just go through the array we have and print it out on the PDF. After all data is on the page a header with PDF metadata is sent along with the actual file, letting the browser decide what to do with that file.

4.2 What needed to be done

4.2.1 Question categories

One of the major things to be added was the support for the two other categories; 'opetus' (teaching/coaching) and 'case'. Since it also required the addition of the actual questions and answers, it was put at the bottom of the list to wait till a workgroup from the project personnel had finalized the questions.

For now a decision was made that all that would be needed is a starting page for all three categories, static questions for the maturities of each category and more flexible

handling of question and answer arrays. That way the same code works for all categories and is easily implementable in a totally different environment.

4.2.2 Report saving

The first major step was the addition of report saving. A database for the reports had already been created but nothing was being stored.

Since the plan had been all along to automatically store all reports without the user having to activate saving, it was decided the feature would be implemented with jQuery.

To avoid having a database table for reports and another table for report data (with has many / belongs to one -relationships) a decision was made that the code would be streamlined to use the same function of array parsing for both PDF creation and report storing. The difference would be where PDF creation starts to throw in question titles; instead the report data array would just be serialized and stored into the database. That way it will be easier to implement a feature for the user to view old reports he/she has done in the past.

4.2.3 Analysis of questionnaires

The point of a Living Lab reviewing tool like Laatupuntari is not to gather statistics about Finnish Living Labs but to let users analyze them.

All questions and answers would have a score; a weight value. Questions are chosen so that it's easy to determine what answer is a good one and what is not. They are then scored accordingly.

It was decided, that this feature would be implemented by someone else, as a sole job.

4.2.4 Listing and viewing old reports

One of the features thought important but not crucial if resources run low was the listing and viewing of old reports. The basic idea is that if a user fills in a questionnaire one day, he can come back to view it after days, weeks, months or even years. That way the user will get a good picture on how his/her Living Lab has improved over a period of time.

The user should have a page where he/she can list all reports stored from filled questionnaires. Along with category and maturity, date and time were chosen as adequate parameters for the user to tell the difference between reports and open the right one the first time without having to browse for extended periods of time.

For the sake of consistency it was also decided that when a stored report is opened it should look as close as possible to how it was when the questionnaire was completed in the first place. This decision makes it possible to use as much of the same jQuery code as possible.

4.2.5 User control

Since both report saving and listing of old reports is meaningless without the ability of linking reports to users that filled them, user control is required.

The requirement from the project group was that anyone can register as a user to the tool and that the only things we store from the users are an email address and a name, be it their forename or a nickname. Protection against bots and spam is provided by reCAPTCHA. Retyping of the user's desired password is required to avoid having spelling errors in the password.

Basic user control functions to implement include a means to change password. Also required is a way to change password in the event the user forgets what it is. Since it is not best practice to store passwords as plain text, the best decision is to send a password reset link to the users email address.

The link would then direct the user to a page, where he/she can change the password. Retyping password again required to make sure there are no errors in it.

Additionally, a dedicated ‘administrator’ status should be available to be given to users for more access to certain elements of Laatuduntari.

4.2.6 User feedback

An optional, low-priority feature suggestion was a dedicated feedback page instead of the survey-link to Webropol. The ability to give feedback would be limited to registered users. A design similar to modern smartphone marketplaces was suggested; a star-based rating along with some identifier to the user, date, time and the actual text.

Ability for the user to delete his/her own feedback was requested, along with administrators being able to delete any feedback deemed inappropriate.

Since Laatuduntari is not a tool where all the feedback goes to telling other people whether or not they should use it, users will be allowed to hide their feedback from public. Hidden feedback will only be seen by administrators and the user responsible for said feedback.

4.2.7 Questionnaire database tool

A less requested feature was a tool to aid in the adding of questions and answers to the database. In the beginning of the project the idea was that any registered user could create series of questions for reviewing and commenting. Those series could then be accepted to the site for anyone to use. However, after it became apparent that resources wouldn’t be enough for that, it was dropped.

Before the programmer change in the project, one colleague had been inserting all questions and answers manually to the database. A simple tool for adding questions

and answers would take little time to add, but save huge amounts of time for the ones adding the questions.

4.2.8 Miscellaneous smaller features

The list of small but important features is partly populated by things the beta testers felt would make Laaturpuntari a better tool and most of all easier to use.

Ready: a button to the last page of the questionnaire for the user to press when he/she is ready watching the results. After pressing this button the user would be shown a page like ‘Thank you for using Laaturpuntari, your results have been stored’, because people expect there to be a ‘save’ -button. Naturally, here it will be just for the sake of the mental health of users, since saving will be done without user input.

Previous: a button to let the user go back inside the questionnaire. Earlier it was only possible to go forward and in the event of an accidental click or a sudden realization it is preferred that the option to go back and correct things is there.

Text input: the first beta version of Laaturpuntari included text boxes. Those boxes could be created by using the ‘inputbox’ -type in questions -table or typing {input} in the answers -table. The first option added an input box in the beginning of each answer and the answer’s text after that. The second one replaced the text {input} inside the answer with an input box. This has several drawbacks, such as not being able to put more than one text box inside one answer. The plan is to create a parser to allow using braces in the answers-table for text boxes.

Question excluding: in first beta all questions from a single category and a single maturity are read to an array and presented one by one to the user. In several cases there are questions where one or more answers will render the next question meaningless. It was being handled by placing a text in the question, stating that it was an extension to the previous one. A feature is needed that allows the tool to automatically skip questions that are not relevant to the user based on a previous answer.

Answer excluding: often the questions used in Laatuduntari will have an answer option like ‘not relevant’ or ‘none of the above’, even when the question type is ‘checkbox’. It created a problem, because the user could choose any number of positive answers in addition to a ‘not relevant’ or similar option. A binary flag in the database will have to be added for excluding answers. An accompanying function in jQuery will handle the client-side of that exclusion.

Progress indicator: users beta testing Laatuduntari were reporting that it’s difficult to know how much questions are left in a questionnaire. Three choices were considered: a time-based indicator using average values, a percentage-based indicator and a basic X/Y –style indicator. The latter was chosen due to giving the best information right from the start.

Force answering each question: in the first beta each question had the first answer chosen as a default. This led to some users skip questions because the default may have been ‘good enough’. That default has to be removed. After that it has to be made sure that each question is answered; if the user tries to go to the next question before answering a message will be shown.

Small tweaks to answering: all input boxes are enabled whether their respective checkboxes or radio buttons are active or not. This is a problem, since the input boxes send HTTP POST data whenever there is data in them. A function is needed to enable input boxes only in answers that have been chosen. Question type ‘inputbox’ will naturally be unaffected by this.

4.2.9 Bug hunting

Resources allocated in the beginning didn’t include time for dedicated bug hunting. That is why the testing was a silently ongoing process from the beginning and bugs would be fixed as soon as they are discovered. One meeting scheduled for a couple of weeks before release will work partly as a bug hunting day as several people with different devices and browsers will get together to test Laatuduntari.

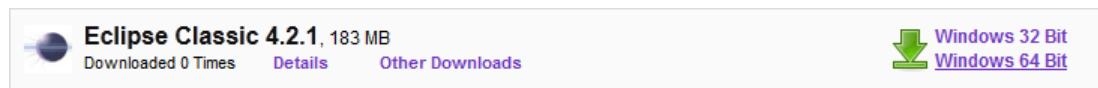
5 PROGRAMMING

5.1 Getting started

The first steps to starting work the actual work on Laatupuntari included setting up a local web development environment.

5.1.1 Installing WAMP and Eclipse

Before developing can begin, all tools need to be installed. First to be installed is Eclipse. It can be downloaded from the Eclipse website (Website of Eclipse 2012). For web developing Eclipse Classic is chosen.


















Picture 6. Eclipse Classic download

Since computers with a 64-bit Windows 7 are used, the 64-bit version of Eclipse is chosen. Eclipse runs on Java, which needs to be installed now if it is not already there.

Java Platform, Standard Edition		
<p>Java SE 7u9 This releases address security concerns. Oracle strongly recommends that all Java SE 7 users upgrade to this release. JavaFX 2.2.3 is now bundled with the JDK on Windows, Mac and Linux x86/x64. Learn more ▶</p> <p>"What Java Do I Need?" You must have a copy of the JRE (Java Runtime Environment) on your system to run Java applications and applets. To develop Java applications and applets, you need the JDK (Java Development Kit), which includes the JRE.</p>	<p>JDK</p> <p>DOWNLOAD ↓</p> <p>JDK 7 Docs</p> <ul style="list-style-type: none"> ▪ Installation Instructions ▪ ReadMe ▪ ReleaseNotes ▪ Oracle License ▪ Java SE Products ▪ Third Party Licenses ▪ Certified System Configurations 	<p>JRE</p> <p>DOWNLOAD ↓</p> <p>JRE 7 Docs</p> <ul style="list-style-type: none"> ▪ Installation Instructions ▪ ReadMe ▪ ReleaseNotes ▪ Oracle License ▪ Java SE Products ▪ Third Party Licenses ▪ Certified System Configurations

Picture 7. Java Standard Edition 7 download

Java 7 (also known as 1.7) download comes in two varieties: Java Development Kit and Java Runtime Environment (Website of Java 2012). Since Java development is not required the runtime is enough to get Eclipse up and running.

Java SE Runtime Environment 7u9		
<p>You must accept the Oracle Binary Code License Agreement for Java SE to download this software.</p> <p> <input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement </p>		
Product / File Description	File Size	Download
Linux x86	54.54 MB	 jre-7u9-linux-i586.rpm
Linux x86	45.77 MB	 jre-7u9-linux-i586.tar.gz
Linux x64	52.71 MB	 jre-7u9-linux-x64.rpm
Linux x64	44.52 MB	 jre-7u9-linux-x64.tar.gz
Max OS X	50.05 MB	 jre-7u9-macosx-x64.dmg
Max OS X	46.62 MB	 jre-7u9-macosx-x64.tar.gz
Solaris x86	45.27 MB	 jre-7u9-solaris-i586.tar.gz
Solaris SPARC	48.56 MB	 jre-7u9-solaris-sparc.tar.gz
Solaris SPARC 64-bit	17.33 MB	 jre-7u9-solaris-sparcv9.tar.gz
Solaris x64	14.78 MB	 jre-7u9-solaris-x64.tar.gz
Windows x86 Online	0.85 MB	 jre-7u9-windows-i586-iftw.exe
Windows x86 Offline	29.72 MB	 jre-7u9-windows-i586.exe
Windows x86	39.42 MB	 jre-7u9-windows-i586.tar.gz
Windows x64	31.18 MB	 jre-7u9-windows-x64.exe
Windows x64	41.19 MB	 jre-7u9-windows-x64.tar.gz

Picture 8. Different versions of Java SE 7 Runtime

It is highly important, that the same version of JRE and Eclipse are installed; if you chose 64-bit Eclipse, you NEED 64-bit JRE. After Java is installed it's time to download WampServer.

DOWNLOAD WAMP SERVER (64 BITS & PHP 5.3) 2.2E

WampServer is an open source project, free to use (GPL licence). You can also fill in this form in order to receive training news from Alter Way, the company behind WampServer. If you do not want to, [you can download it directly](#).

WARNING : You must install Visual C++ 2010 SP1 Redistributable Package x86 or x64
 VC10 SP1 vcredist_x86.exe 32 bits : <http://www.microsoft.com/download/en/details.aspx?id=8328>
 VC10 SP1 vcredist_x64.exe 64 bits : <http://www.microsoft.com/download/en/details.aspx?id=13523>

WARNING : Do not try to install WampServer 2 over WAMP5.
 If WAMP5 is installed on your computer, save your data, uninstall it and delete the WAMP5 directory before installing WampServer 2.

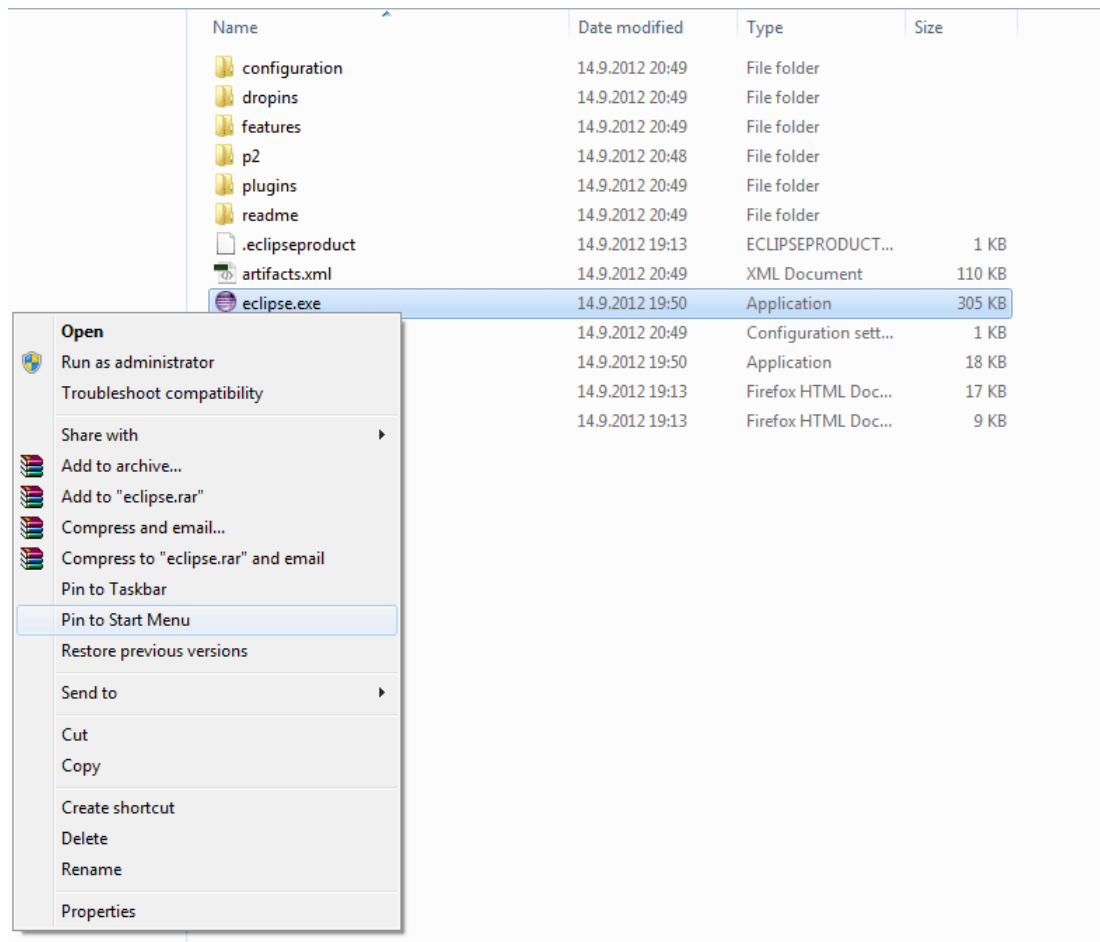
WARNING : All the components of the v2.2 WampServer stack have been compiled with VC9 version of Microsoft compiler.
 Earlier versions of Wampserver have been made with VC6 version of Microsoft compiler.
 So, You can't mix components of 2.2 stack with previous version of Wampserver Stack components.
 If you do it you will get an instable Wampserver.

Picture 9. Downloading WampServer

WAMP provides several different packages; both 32- and 64-bit versions and packages using either Apache 2.2 or 2.4 (Website of WampServer 2012). Newest PHP is also available if it is desired. Some changes introduced in PHP 5.4 would require slight modifications to Laatupuntari's code and therefore PHP 5.3 will be used as long as possible.

Because Laatupuntari will be put to the www-root of WAMP, the server has to be installed before Eclipse. The default directory is chosen. Near the end of the installation procedure you are asked, what browser to use as a default for WampServer functions. Because Google Chrome's features are perfect for web developers, Chrome is chosen here. It should be noted that this has very little effect and all the configuration pages and phpMyAdmin can be opened with any browser by just pointing to localhost and the port set in WampServer configuration. Port 80 is the default and under most situations can be left unchanged. The only server configuration related thing that has to be enabled in WampServer is the rewrite module. How to enable mod_rewrite will be covered later along with other configuration for Laatupuntari.

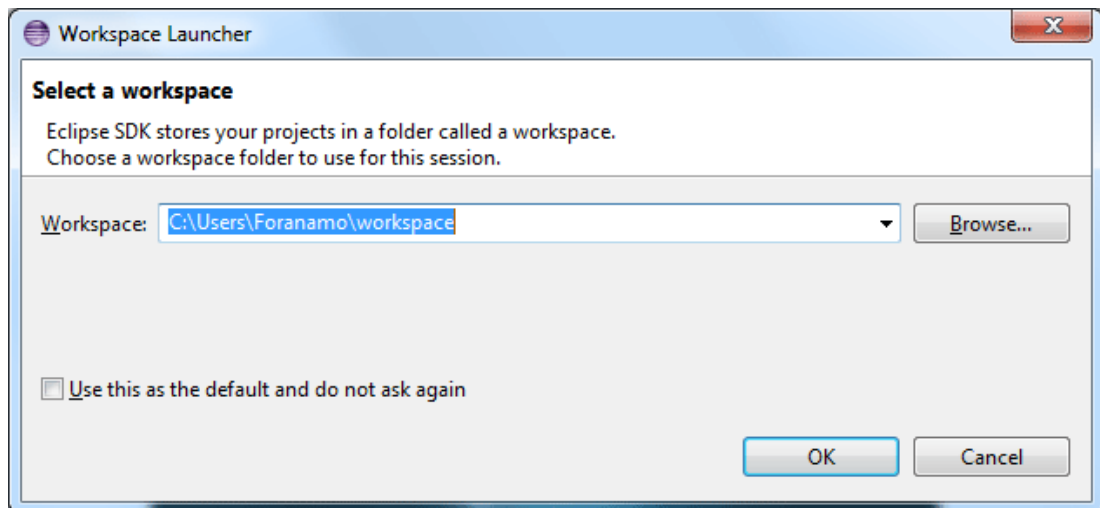
Eclipse is a bit different to most programs in that it doesn't come with an installer by default. Instead a regular Zip archive is provided. That archive can be extracted anywhere. To make opening Eclipse easier the 'Pin to Start Menu' option of Windows 7 is used (Picture 10).



Picture 10. Using Pin to Start Menu in Windows 7

For the sake of covering Eclipse setup in one shot, let's assume Laatusuntari is placed where it needs to be. Installation and possible configuration of Laatusuntari will be covered later.

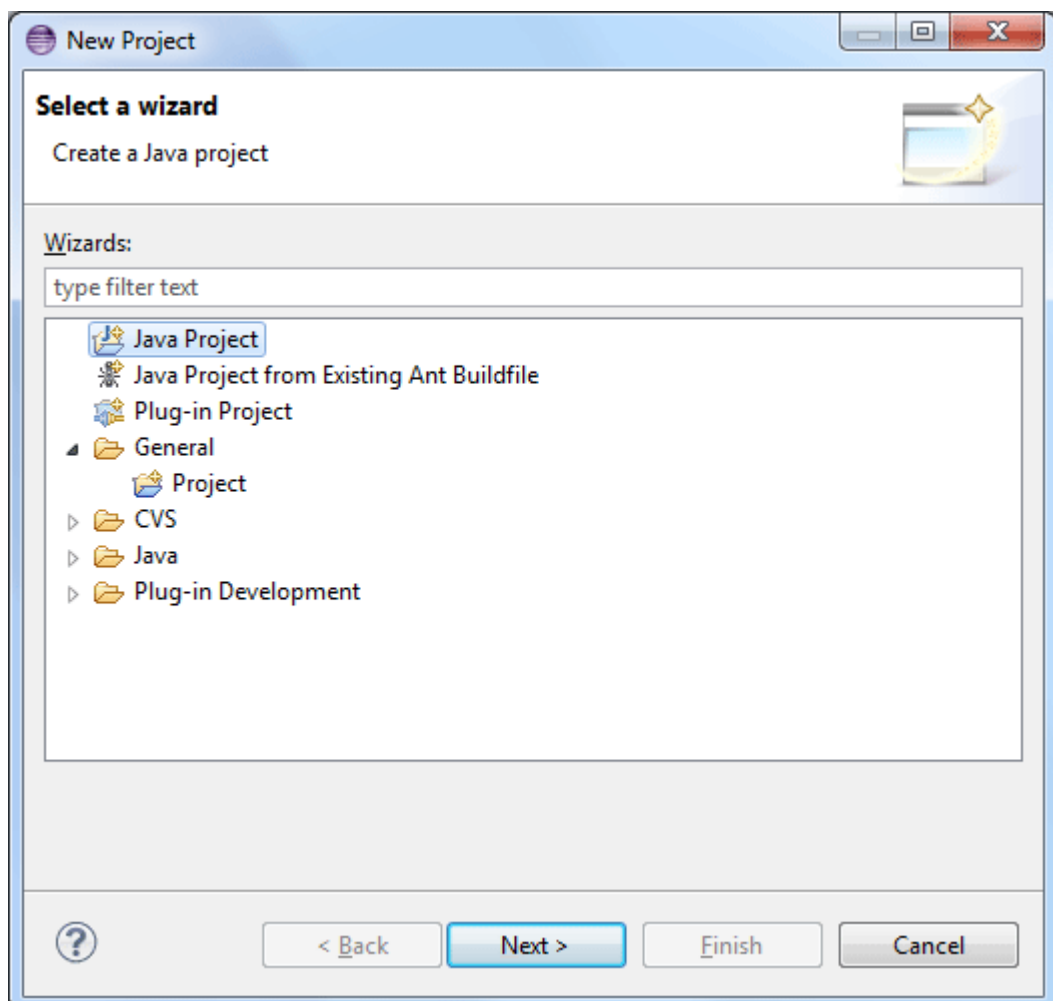
When Eclipse is opened for the first time a window about workspace location is presented. Workspace is where Eclipse stores all personal preferences; window sizes, toolbars, opened projects and other settings. Default option is chosen here.



Picture 11. Eclipse Workspace Launcher

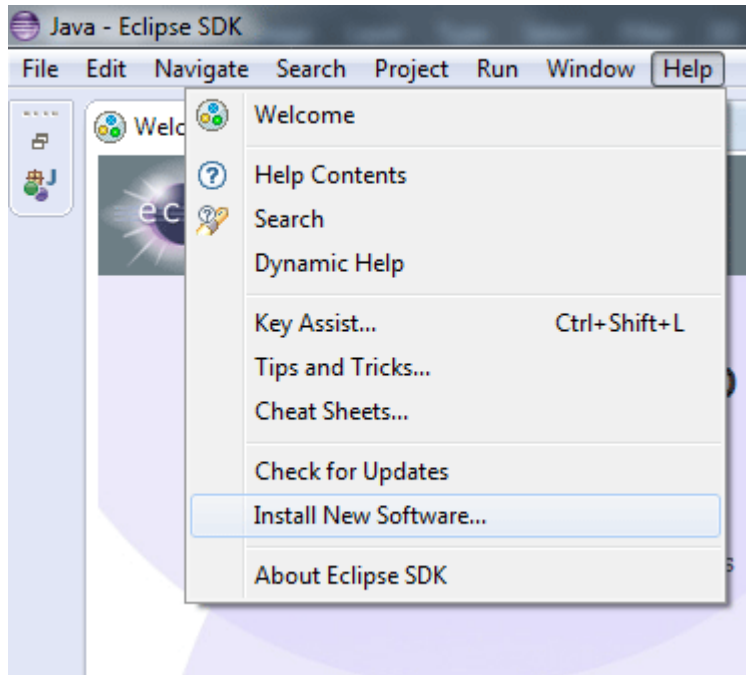
To stop Eclipse from asking about workspaces on every launch, the chosen workspace can be set as default with the checkbox on the bottom left corner.

Before programming can be started, Lautupuntari needs to be opened as a project.



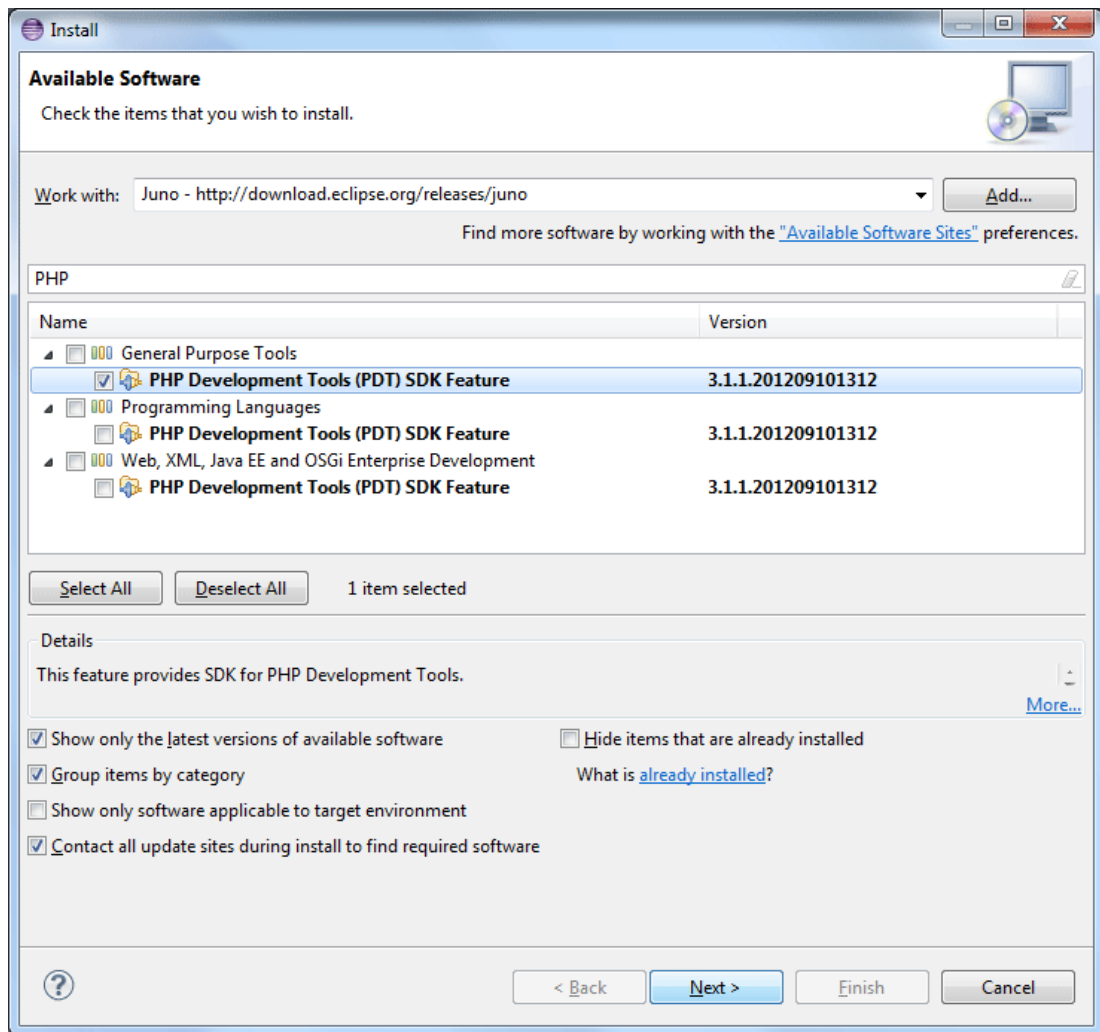
Picture 12. Eclipse New Project dialog on first start

As Eclipse was created first and foremost for Java development, PHP projects are not supported out-of-the-box. Next the dialog needs to be closed.



Picture 13. Eclipse Help -menu

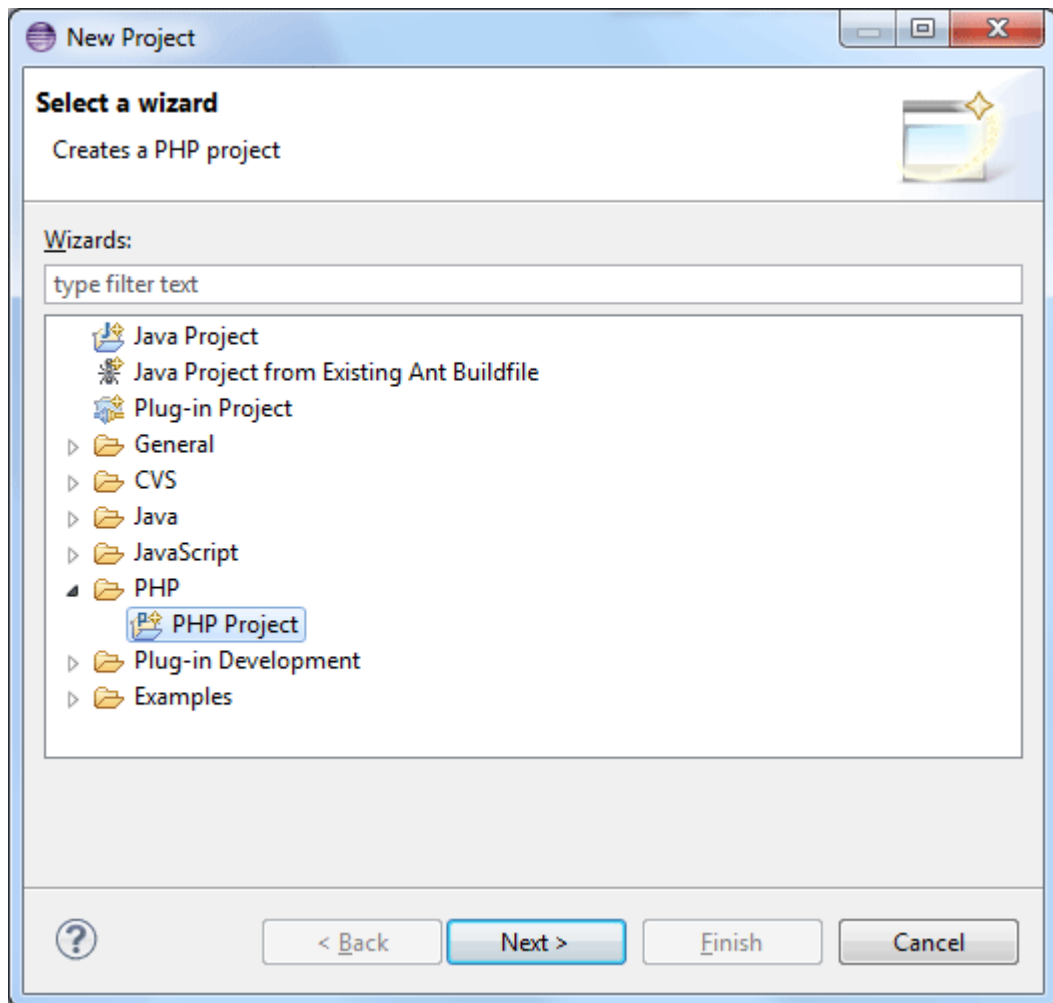
From Eclipse's Help -menu, 'Install New Software' is chosen (Picture 13). After the new dialog opens, Eclipse Juno -repository is chosen. Once the repository is downloaded, 'php' is written in the filter box to minimize browsing through irrelevant options (Picture 14).



Picture 14. Eclipse Install New Software -dialog

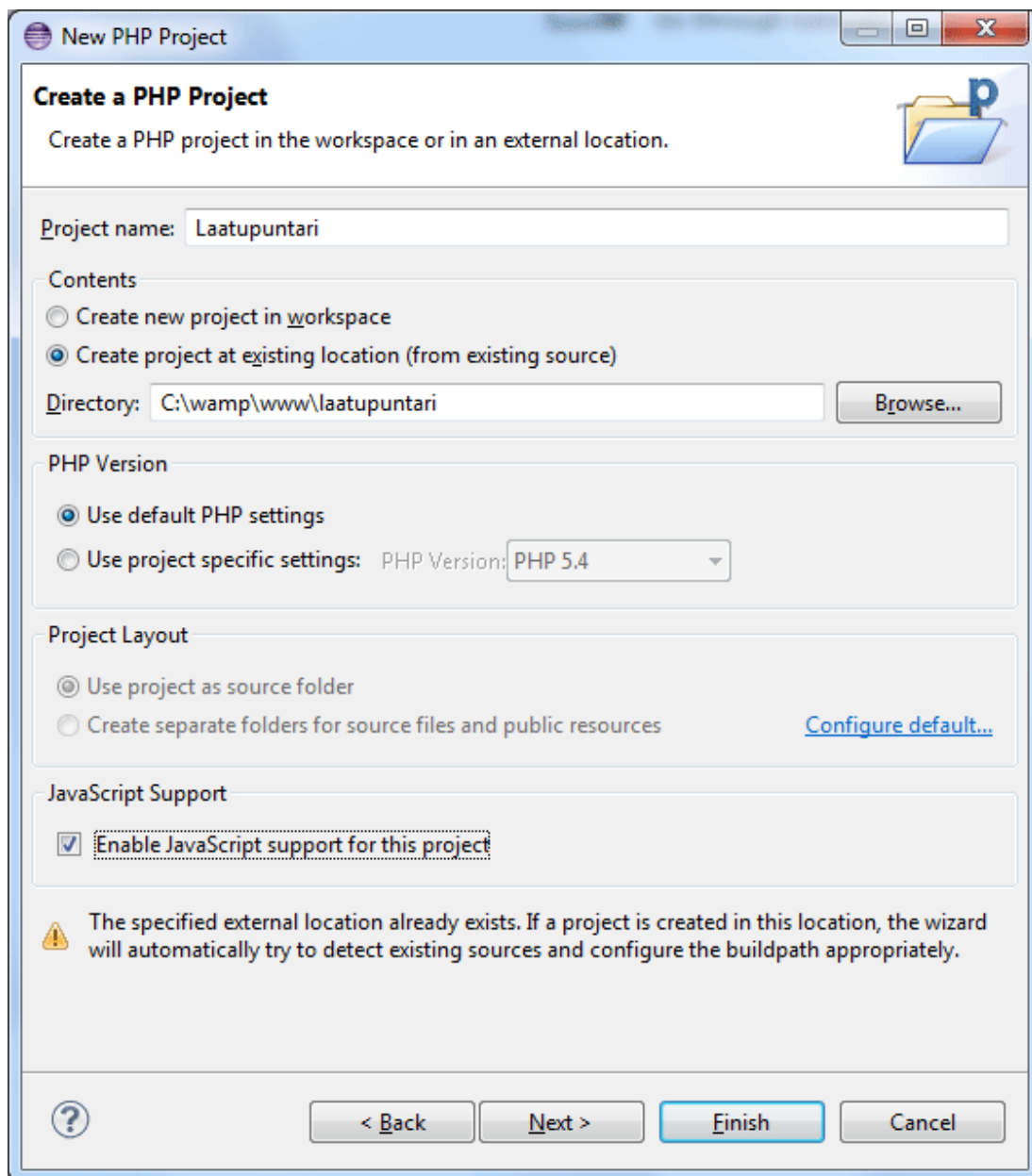
Any of the three shown packages can be shown as the same package is referenced for three different categories. After approving to the license agreement, PHP Development Tools will be downloaded and installed. A restart for Eclipse is required before the new tools are available.

When Eclipse is started again it's time to bring up the New Project -dialog.



Picture 15. Eclipse New Project -dialog with PHP Project available

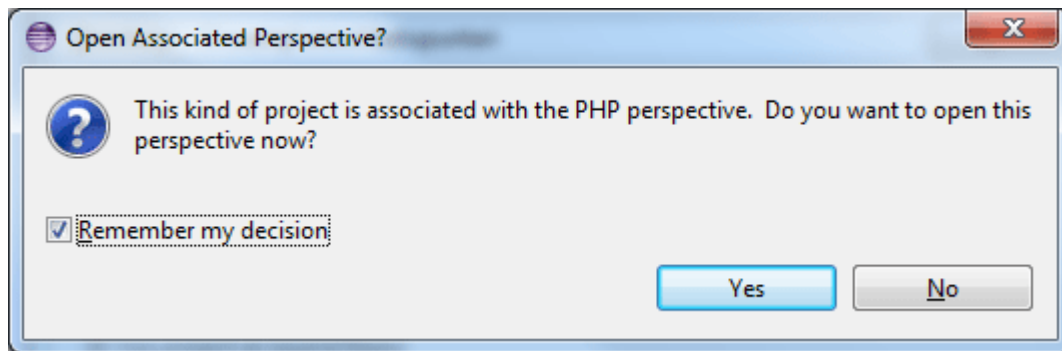
PHP Project can now be selected. Next Eclipse will present a window about the new project.



Picture 16. Create PHP Project -dialog in Eclipse

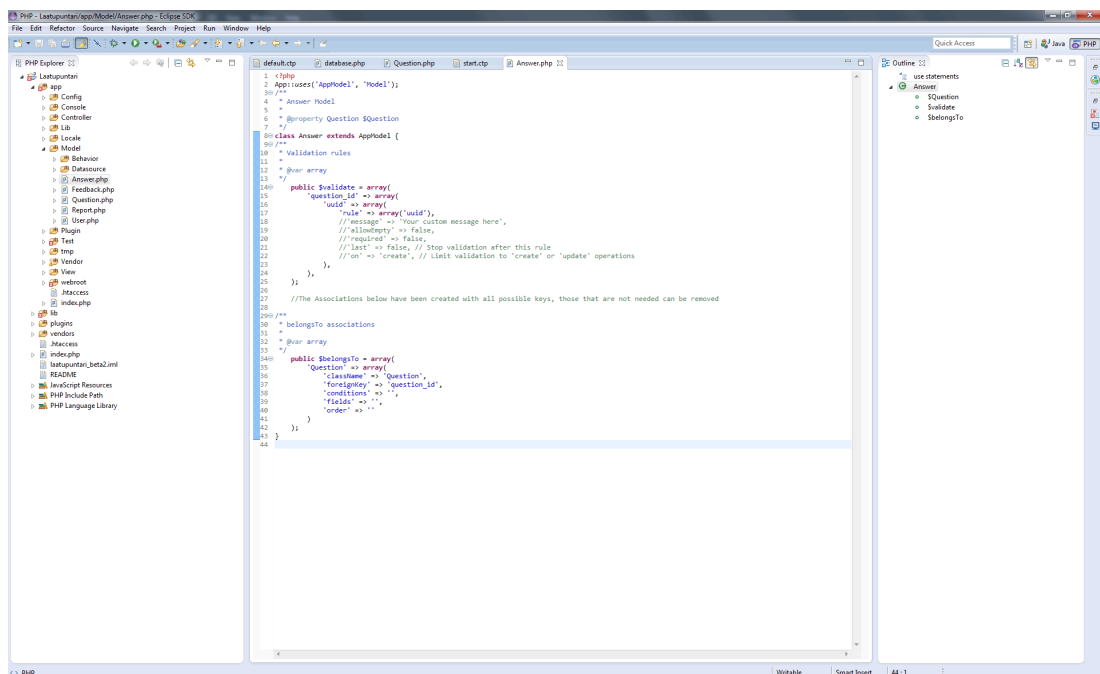
In this dialog Eclipse prompts for several things about the project to be created (Picture 16). The important thing is that ‘Create project at existing location’ is used and that it is pointed to where Laatupuntari is stored. Here Laatupuntari is WampServer’s www-root. JavaScript support is wise to enable so Eclipse provides you with syntax highlighting there as well.

When everything is set it is best to click Finish as the additional setup is not required. Before code is displayed Eclipse asks if you would like to open the ‘PHP Perspective’ that has window functions more suited for PHP coding (Picture 17).



Picture 17. Open Associated Perspective -dialog in Eclipse

After answering this you may still need to minimize the Eclipse ‘home page’. After all setup, Lautupuntari is finally opened in Eclipse (Picture 18).



Picture 18. Lautupuntari open in Eclipse

5.1.2 Setting up a local copy of Lautupuntari

A local copy of Lautupuntari was pulled from Github (Website of Github 2012), but because the Lautupuntari repository in Github was deleted very soon, Github will not be covered in this thesis.

The local copy was placed straight into WampServer's www-root; in this case `c:\wamp\www\laatupuntari`. WAMP's default installation directory is `C:\wamp\`. An up-to-date MySQL-dump was attained from the previous developer.

Installation of Laatupuntari will be covered in chapter 6. Cloning from Git, configuration, MySQL import and possible problems will all be talked through.

5.2 Starting with the easy ones

Since both CakePHP and jQuery were completely unknown to me before starting in this project, it was best to start with small features and bugs that had been reported during first beta.

5.2.1 Progress indicator

The simplest to implement, the progress indicator gave good time to find out how the jQuery -script did its job. The function rendering all questions and answers to the page has everything neatly formatted inside an array. It was only a matter of going inside the 'for' -loop rendering the elements and printing the key of the current element in the loop along with the maximum number of elements, also known as '.length'.

Once it was surrounded by brackets, it was given its own div -element. With a little CSS -styling, the first item on the board was done.

5.2.2 View previous question

In the very beginning it was considered that each answer would be stored to the database the instant the user moved to the next one. After careful consideration it became obvious, that this type of saving would pose a problem in a number of different scenarios. To implement a feature to allow the user to go backward in a questionnaire,

Laatupuntari would have to request answers to the previous questions from the database each time the user goes back.

It was then finally decided, that saving would happen only when every question has been answered. That allows Laatupuntari to show previous questions in a near identical fashion to how the next question is displayed.

jQuery provides great tools for listing elements matching a certain tag, class or ID. Using those tools, a list of previous question-elements is created, but only the first one of them selected and faded in. The current question is naturally faded out before that.

A 'Previous' -button was created next to nearly every 'Next' -button, with the exception being the first question. This exception comes from the fact that if the user could go back to the maturity selection, all questions would have to be removed and new ones retrieved from the database.

After some time, a bug was discovered in this 'Previous question' functionality. If the user first pressed 'Next' and then very quickly 'Previous', the current question would fade out and both the next and the previous question would fade in. After that there was no way of getting the additional element to disappear. Work began on figuring out the way jQuery does animation.

It was found out that jQuery puts all animation inside a queue. The key to fixing the problem was stopping all queued animation when fading out using the 'previous question' -function. This led to the animation being less smooth, but since it only happened when going back it was considered a non-issue. Final code was complete:

```
$("#div.prev").live('click', function() {  
    $(this).parent().fadeOut().stop(true, true).fadeOut(function() {  
        $(parent).prevAll(".question:first").stop(true, true).fadeIn();  
    });  
});
```

5.2.3 Prevent going forward without answering

Preventing the user from proceeding in the questionnaire without answering was a matter of checking if any input element had been selected. The most trouble stemmed from the difference of text boxes and other input elements. The final solution was two different jQuery loops, going through all inputs of the active element. The first loop goes through all regular inputs of the active question element and checks if any single one of them has the attribute 'checked'. The second loop goes through all inputs of type 'text' and checks if any single one of them has the 'value' attribute longer than zero. If none of these conditions are met a message is shown and animation from the current question to the next one will not happen.

5.2.4 Answer exclusion

Answer exclusion was the first one requiring some thought to how it should work. The first idea that came to mind was:

Before nothing is selected, everything is enabled. When a 'regular' answer is selected the excluding options are disabled and vice versa. The drawback in this is that a function would have to run each time any answer is selected, creating unnecessary stress on the user's computer.

Work began on creating a more efficient solution. The final idea started from the exclusion flag in the 'answers' database. When we have a flag we can put an additional class on all the answer options that have that flag. Now we can call a function only when an exclusion modifier enhanced option is selected.

Let's say the user first selects an answer like 'Budget is handled by a small workgroup'. Next, the user is feeling adventurous and selects 'None of the above' before deselecting the first option. A jQuery live click event is triggered by the 'excl' class in that answer. The function goes through all the other options to see if one has been selected. If the loop comes back positive, the user is informed that he/she cannot choose that option with another option and the excluding option that the browser checked is unchecked.

The opposite scenario is handled so that when the user first selects an excluding answer the live click -event is fired up again. Since no other options were selected the function allows the users action and then goes on to disable all the other options inside that question. Now if the user wants to change his/her answer all he needs to do is deselect the excluding answer and all the other answer become enabled again.

Here's what the final code looks like:

```

$(".excl").live('click', function() {
    var allow = 1;
    var thisvalue = $(this).attr('value');
    var checked = $(this).attr('checked');
    $(this).parent().parent().parent().find('input').each(function(index){
        if($(this).attr('checked') && $(this).attr('value') != thisvalue){
            allow = 0;
        }
    });
    if(allow == 0){
        $(this).attr('checked', false);
        alert('Et voi valita tätä vastausta muiden vastausten kanssa!');
    } else {
        $(this).parent().parent().parent().find('input[name*=input]').each(
            function(index) {
                if($(this).attr('value') != thisvalue) {
                    if(checked) {
                        $(this).attr('disabled', true);
                    } else {
                        $(this).attr('disabled', false);
                    }
                }
            }
        );
    }
});

```

5.2.5 Question exclusion

Excluding questions that are not relevant to the user based on the answer of a previous question was one of the first slightly bigger tasks after first beta was completed. Like before, the first task to completing the required feature was figuring out how it needs to work.

When asked, the workgroup was unsure about several aspects: will there be several answer choices to trigger one 'definitive' question and can one choice trigger multiple definitive questions. The ultimate decision was to make it as flexible as possible,

since too much function is of little trouble, whereas too little function will probably need to be modified along the road.

Two additional columns were added to the database for this function:

- def: a binary modifier to indicate that a question has the ‘power’ to trigger other questions
- subid: this will be used on a question that can be excluded by some other question. Will include the id of the triggering question and all the answer options for triggering surrounded in vertical bars. Example:

```
d2d36d18-23d2-11e1-8984-000c29829b30|0|1|2|
```

The functionality is then built using HTML classes and jQuery.

All questions flagged with ‘def’ will have the ‘def’ -class in every answer option. That way ‘live click’ can be used whenever the user clicks an answer option that has the ability to hide or show a question further along the questionnaire.

All questions that have a ‘subid’ set are given that ‘subid’ as a class. Those questions are also marked with the ‘delete’ class. The ‘delete’ -class will be used to remove all non-relevant questions when the questionnaire is complete, before submitting results to the database. Because the way the next question is always faded in is by using the ‘question’ -class, these possibly non-relevant questions are not given that class and therefore are not found by jQuery.

```
$(parent).nextAll(".question:first").stop(true, true).fadeIn();
```

This completely skips the questions marked with ‘delete’, since either ‘delete’ or ‘question’ is used, never both.

Now, when an answer triggers the ‘def’ class ‘live click’, the questions are hidden and shown using jQuery’s wildcard selector. All questions that have the current questions id in their class will be hidden by removing the ‘question’ class and adding the ‘delete’ -class. Of course neither of those happen if there is no ‘question’ class and the ‘delete’ -class is already there. Because jQuery selectors work fast enough this

method is preferred over going through every possible ‘definitive’ question and enabling/disabling them depending on what answer options are chosen.

After all these sub-questions are disabled, a selector is built from:

- the id of the current question
- the number or value of the answer option selected; the one that called the function in the first place

After the answer option number is enclosed in vertical bars, both of the selectors are thrown together along with wildcards. jQuery gives a good match to a ‘subid’ with these selectors and all matching elements can safely be enabled by removing the ‘delete’ -class and adding the ‘question’ -class, effectively including them in the questionnaire.

Final code follows:

```
$(".def").live('click', function() {
    var id = $(this).parent().parent().parent().attr('id');
    $('[class*=" '+id+'"]').removeClass('question').addClass('delete');
    //First hide all possible sub-questions
    var sub = "|" + $(this).attr('value') + "|";
    $('[class*="'+id+'"][class*="'+sub+'"]').removeClass("delete")
    .addClass("question");
    //Then show the set of sub-questions that was chosen.
});
```

5.2.6 Selective enabling of input boxes

As mentioned earlier, input boxes proved a bit of a problem. The user could type text to an input box but leave the respective answer button unchecked, leading to problems server-side. The remedy to this was to disable all input boxes in questions of type ‘radio’ or ‘checkbox’ by default and then enable those inputs only when needed.

A new HTML class was come up with; ‘control’. Every checkbox and radio button is given that class and the rest handled in jQuery. All input box -elements are looped through and dealt with properly.

Most of the functionality is achieved through the element naming conventions used in Lautupuntari. Since radio buttons and checkboxes work differently, separate methods were written for each. Handling of checkboxes was written first, since it is simpler:

1. Take the name of the checkbox, use split to get rid of []
2. Split away the 'input' in the checkbox's name, end up with question ID
3. Use a wildcard to find if there are any input boxes for the selected input
4. Enable/disable accordingly. In case of disable the input box is cleared

Handling of radio buttons differs a bit:

1. Create a loop of each radio button in the question
2. Split away 'input' to end up with question ID
3. Use a wildcard to find if there are input boxes for the current input in the loop
4. Enable/disable accordingly and clear on disable

The way radio button type questions are handled is slightly more taxing to the client but as of date has not posed a problem. Final code follows:

```

$(".control").live('click', function() {
  if ($(this).attr('type') == 'radio') {
    $('input[name='+$(this).attr('name')+']').each(function(index) {
      var name = $(this).attr('name').split('_');
      var textname = 'textinput_'+$(this).attr('value');
      if ($('#input[name*="'+textname+'"][name*="'+name[1]+'']')
        .length > 0){
        if($(this).attr('checked')) {
          $('#input[name*="'+textname+'"][name*="'+name[1]+'']').
            attr('disabled', false);
        } else {
          $('#input[name*="'+textname+'"][name*="'+name[1]+'']').
            attr('disabled', true);
          $('#input[name*="'+textname+''][name*="'+name[1]+'']').
            attr('value', '');
        }
      }
    });
  } else {
    var inputname = $(this).attr('name');
    inputname = inputname.split(' ');
    var name = inputname[0].split('_');
    var textname = 'textinput_'+$(this).attr('value');
    if ($('#input[name*="'+textname+''][name*="'+name[1]+'']')
      .length > 0){
      if($(this).attr('checked')) {
        $('#input[name*="'+textname+''][name*="'+name[1]+'']')
          .attr('disabled', false);
      } else {

```



```

        $( 'input[name*="'+textname+'"][name*="'+name[1]+'"]' )
        .attr( 'disabled', true );
        $( 'input[name*="'+textname+'"][name*="'+name[1]+'"]' )
        .attr( 'value', '' );
    }
}
});

```

5.2.7 Text input

The way text boxes were handled in the beginning was very static and numerous reasons required a change in that. A solution was needed that didn't just put an input box in front of every 'inputbox' typed answer and replace every {input} -text with an actual input box.

It was decided before that {} -braces would be perfect for this and all that remained was to implement it. In report storing and PDF printing this required little else than a simple PHP replace function call but the client-side took consideration.

“A regular expression (regex or regexp for short) is a special text string for describing a search pattern.” (Wikipedia: Regular expression 2012)

The perfect way of finding all brace-enclosed tags and saving them was the use of regular expressions. After some time studying regular expressions and a couple of hours of trial and error, a suitable regexp was found.

```
\{[a-ö]*\}(?!\\)
```

Though unlikely that Scandinavian letters (ä, ö, å) are ever used, it seemed appropriate to include them inside the regexp. However, due to the nature of regular expressions and charsets, in some cases this may not work and it's best to stick with basic letters from A to Z.

Now that a suitable regexp is complete, it is run against every answer for as many times as it comes back with a hit. The hit is replaced with an input box and part of

that hit is used in the name of the input box. That naming convention is covered in the next chapter.

```
var value = answerVal.value;
var re = new RegExp("\{[a-ö]*\}(?!\\_)");
var m = re.exec(value, 'i');
while(m != null) {
    value = value.replace(re, '</label><label><input type="text" '
    +disabled+' name="textInput_'+answerKey+'_'+m+'_'+val.Question.id+
    '"/>');
    m = re.exec(value, "i");
}
```

Because regexp execution returns null when no hits are found, a while loop against 'm' not being null is the easiest way to go. The 'disabled' variable simply stores the text 'disabled' in cases where the question type is 'checkbox' or 'radio' and it is cleared for questions of type 'inputbox', since there are no other elements to use for enabling the input boxes.

5.2.8 Client-side element naming convention

The input elements in questions have clear naming conventions to make it easier to, for example, write jQuery selectors for them. The basic input name is formed like this:

input_<question-id>

Naturally, <question-id> is replaced with the actual question ID, like in this example: input_47cfc072-1a59-11e1-8984-000c29829b30

The naming format for input boxes is slightly different:

textInput_<number-or-value-of-answer>_{inputtag}_<question-id>

An example from the site:

textInput_0_{input}_47cfc072-1a59-11e1-8984-000c29829b30

When split with the underscore, here's how it's constructed:

1. The tag, naturally. Three possible tags are used: input, textinput and score. Score will be covered later.
2. The number or value of the answer the text input is tied to. For the first answer in a question, this will be 0, for the second answer it's 1 and so on.
3. The tag from database, used to replace the answer text's tag with the value that is typed in the input box.
4. The question ID

With this underscore splitting it is nearly trivial to look up text boxes related to a checkbox or hide questions further up the questionnaire.

5.2.9 'Ready' -button

Adding a 'Ready' -button included throwing in a button and a small page that thanks the user for taking the time to fill out the questionnaire.

The button was created with this simple call:

```
$("#questions").append('<a href="'+root+'/questions/valmis" class="button small">Valmis</a>');
```

A small file was added as the page the button redirects to, with a link to the Laatu-puntari home page and a link to the Webropol survey. Included was a text assuring the user that his/her answers were saved. Not having a button for saving created uncertainty among beta-testers and having a page assure them that everything is saved and secure gives the user some peace of mind.

5.3 Off to the bigger challenges

5.3.1 Report saving

The first major new feature was coding the saving of reports. Since a good parser had already been written it was chosen best to make it even better and use it as a part of report storing.

Here is an example of the parsers work. First we have data as it comes from the browser:

```
[input_1e7a4152-1a59-11e1-8984-000c29829b30] => Array
(
    [0] => 0
    [1] => 1
    [2] => 5
    [3] => 7
)
[input_47cfc072-1a59-11e1-8984-000c29829b30] => 0
[textinput_0_{input}_47cfc072-1a59-11e1-8984-000c29829b30] => 2015
```

The first array is from a question of type ‘checkbox’ with multiple choices. Here first, second, sixth and eighth answers were chosen. HTTP POST automatically sends that data as an array, provided that all checkboxes are named ‘name-of-checkbox[]’ with [] being the mark of an array.

The second one is from a question of type ‘radio’. The first answer option included an input box and this is how it is sent to the server. Let’s see the same data after the parser:

```
[1e7a4152-1a59-11e1-8984-000c29829b30] => Array
(
    [0] => 0
    [1] => 1
    [2] => 5
    [3] => 7
)
[47cfc072-1a59-11e1-8984-000c29829b30] => 0|{input}|2015
```

For the first one very little is modified; only ‘input_’ has been removed. In this form it is very easy to use CakePHP’s find recursively to gather all question and answer data and then use this parsed report array to show the user what he/she selected.

The second one is less similar to how it was before. It stores the number of the answer, the input tag for use with a replace -function and the actual answer the user typed. Vertical bars are used to separate the data. In case of several answer options an array is formed and each cell will contain similarly formatted data.

In the event we have an answer option with two or more input boxes, they are parsed as follows:

```
[question-id] => Array
```

```
(
    [0] => 2|{input_one}|two years
    [1] => 2|{input_two}|of solitude
)
```

After the parsed data is securely in an array, a new report is opened. Following data is stored inside the report:

1. User id. If the user is not logged in a -1 is stored instead.
2. Session id; automatically created by CakePHP.
3. Date and time. Helsinki is set as time zone and the date is then stored like this: 2012-12-02 20:53:00
4. The data itself. The array is taken and serialized with PHP. Out comes textual data that can be stored in the database. That data can then later be unserialized and an identical array to the one originally stored is retrieved.

As the storing now works, it's time to work on something else for a while. In that time several reports will be stored and they will be useful for debugging when report listing and viewing are implemented.

5.3.2 Tool for adding questions to database

In the early stages of development, question series were added manually through phpMyAdmin. This took away hours of time and a tool for help was requested.

Since the database tool would only be used internally - at least in the beginning - user interface wouldn't be much of a problem. CakePHP's FormHelper component was used to create a form with all input element names formatted so that it would be as easy as possible to store in the database.

A form was created with all the elements that the 'questions' table had in MySQL. Possibly the most important part of the tool would be answers linked to the question. Previously it was required to manually first add the question, then copy the questions ID and add all related answers one at a time.

With the tool a form with the answer data was added along with the question. That way the answer would automatically link with the question in the database. To avoid having to first determine if 1 or 7 answer options are wanted, some jQuery was added. A small button was included that would call a function in a dedicated jQuery script. That script would then take the first answer element with all its data and copy it to the bottom of the page. All names would then be modified to make sure everything is stored correctly by CakePHP. Some of CakePHP's beauty is apparent in the code that stores the question and its answer options to the database:

```
function add() {
    /* add is used both in showing the form for addition and saving data
    * check requesthandler, if there is data to save
    */
    if($this->RequestHandler->isPost() && !empty($this->data)) {
        $this->Question->saveAll($this->data);
    }
}
```

The tool was used to add the remaining two categories of questions to the database. Time spent to question adding was lowered by 80-90%.

5.3.3 Enabling the last two categories

The easy part of enabling the second and third category was creating information pages for them and enabling the links in the home page. Before actual texts were received from the workgroup, the info page of the first category was used for all three.

As the way Laatumuntari was built doesn't allow for the category questions coming from database – not without significant rewrites at least – all the category questions are hardcoded inside the same PHP-file. HTTP GET -parameters are used to tell which maturity question form from which category to display.

More complications rose when the workgroup decided that it would be great if the 'case' questions could all be shown in succession; that the user could choose more than one maturity level at once. At that point some rewriting was done to use the same code for both situations, be it a single maturity or multiple maturities in an ar-

ray. Luckily, CakePHP's model find allows for the use of arrays in a condition, in this case retrieving questions of multiple maturities at the same time. Close to no modification was required client-side, since all questions are still rendered in one list and displayed one at a time.

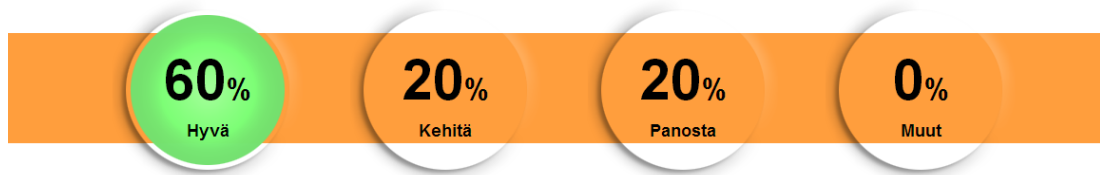
5.3.4 Analysis of reports

A key function of Laaupuntari was to be able to tell users how they fared in the Living Lab -related questions. This component was assigned to Razvan Constantinescu during his student exchange at the Turku University of Applied Sciences. A thesis was written about the creation of that component (Constantinescu 2012, 32-40). However, some modifications had to be made to make the analysis component more suitable for use with Laaupuntari and to work better with the existing code.

The principle of the analysis component is that every answer option has a score value, a weight. These scores are added up for a max score for every question. Percentages are then used to determine whether the answer(s) a user chose were:

- a) Good
- b) Bad
- c) Ok
- d) Other

The 'Other' -category had to be added later, because the user couldn't see the question he/she had answered in a way that left them out of the analysis. Additionally questions that only have input boxes are not analyzed by the component at all, since an algorithm would be required to analyze text written in a box and resources had to be directed to more important aspects of Laaupuntari.

Lopputulos:**Keiden tarpeista tuotteiden tai palveluiden kehittämisprosessi (Living Lab Case) on käynnistynyt?**

- hyödyntäjien ja/tai käyttäjien
- kehittäjien
- operaattoreiden ja/tai mahdollistajien

Mitkä tahot ovat osallistuneet projektiainion suunnitteluun?

- operaattori ja suunnitellut casen toteuttajatahot
- suunnittelu on ollut yhden henkilön varassa
- ei ole tehty

Onko Living Lab -casen toteuttamisesta tehty sopimus (tarjous hyväksytty tai erillinen sopimus)?

- kirjallinen casesopimus on tehty
- case käynnistettiin ilman sopimusta

Picture 19. Example of the analysis component in Lautupuntari.

Four colored circles are used to depict good, ok, bad and others. Texts used here are: ‘Hyvä’ (good), ‘Kehitä’ (improve), ‘Panosta’ (invest) and ‘Muut’ (others). The circles reveal a color suitable to the situation when hovered over. A click on the circle shows the questions linked to that review. Showing the questions is handled by adding a class to each question, classes being ‘good’, ‘ok’, ‘bad’ and ‘other’. On click of any of the four circles all ‘question’ classed elements are faded out and elements of the same class as the circle are shown.

5.3.5 User accounts

The first feature requiring a new table to the database was user control. A table was added containing:

- an id
- username
- password
- email
- administrator flag

- a text field to store a token used when a user forgets his/her password

The project workgroup felt that no more information was required of the users than a username, password and an email address. As little information as possible is best.

Several different CakePHP features came in handy at creating the user control:

- Denying access to pages if a user is not logged in (Golding 2008, 198)
- Encrypting passwords
- Matching passwords
- Using session to flash messages when a user is registered, logged in and so on
- Creating a registration form with FormHelper, then saving data through a User model

The basic user account control was created; a form to change password and a form to send the user a password reset link in case the password is forgotten.

To provide some anti-bot protection, reCAPTCHA was chosen for the user registration page. After registering at Google for a reCAPTCHA key, a plugin for use with CakePHP was found. An open source plugin was cloned from Github and set up using instructions from the CakePHP Bakery. (Jahdrien 2012).

Once setup has been done and the reCAPTCHA-plugin loaded in the AppController, all that is needed to display reCAPTCHA in the desired page is:

```
echo $this->Recaptcha->show(); //Display the plugin
echo $this->Recaptcha->error(); //Display any error from the plugin
```

Sähköpostiosoite

Salasana

Salasana uudelleen



Kirjoita näkemäsi kaksi sanaa:

reCAPTCHA™ stop spam. read books.

Picture 20. Displaying reCAPTCHA in the registration page.

When a new user is registering for Laatupuntari, he/she will first fill out all the other information and then the reCAPTCHA. If the CAPTCHA doesn't match, the user is informed about this.

In the 'forgot my password' function, a 10 character randomly generated token is used. The token is sent as an email to the user and also stored in the database with the user's information. When a page is opened with the token, the token is used to find the correct user. Then the user is required to type a new password twice. No password history is recorded and neither are plain text passwords, reducing the severity of a possible hacking attempt. An example function from the user control follows:

```
function register() {
    if(!empty($this->data)) {
        if ($this->User->save($this->data)) {
            $this->Session->setFlash('Your account was created',
                'default',
                array(), 'auth');
            $login = $this->Auth->login();
            $this->redirect('/');
        } else {
            $this->Session->setFlash('Your account could not be created.',
                'default',
                array(), 'auth');
        }
    }
}
```

If save is successful a message is displayed and the user is redirected to the home page of Laatupuntari. In case of an error, like passwords do not match or email address already in use, an error is displayed and the user is directed back to the registration page.

After some time the project workgroup felt there was too much data for the user to remember and requested that the username be removed and email address used for login instead. Since the beautiful automatic functions of CakePHP relied on the data names being username and password, the email address is handled as username server-side and only labeled as an email address to the user. At the moment of writing,

the email field still exists in the database. A thorough check is required to see if it is still used before it can be removed permanently.

5.3.6 Listing of reports

After user control was done it was time to start implementing the listing of old reports. To be used through the users own page, the list would feature some information about the report and questionnaire that was filled. That information should be enough for the user to tell reports apart.

The data to be displayed in the list was chosen as: date, category and maturity. Category is stored as text in lower-case, so all it takes to display is the ucfirst()-function of PHP to convert the first letter to upper-case. Maturity is stored as a number, so a three dimensional array of categories and maturities was written. That array is then referenced by both the category and maturity, as many times as there are maturities in a report. Remember that the second category allows up to 4 maturity levels in one report.

In the end the list was simple the construct. All controller-side code fit in two rows:

```
function listReports() {
    $reports = $this->Report->find('all', array('conditions' =>
        array('userid' => $this->Auth->user('id'))));
    $this->set('data', $reports);
}
```

Aikaisemmin täyttämäsi kyselyt

Päivä ja aika	Kategoria	Kypsyystaso
0000-00-00 00:00:00	Toiminta	Valmisteilla
2012-09-11 15:22:31	Case	Arviointivaihe
2012-09-14 16:20:23	Case	Arviointivaihe
2012-09-14 16:21:05	Case	Tilausvaihe
2012-09-14 16:22:02	Case	Toteutusvaihe
2012-09-25 15:19:55	Case	Toteutusvaihe, Arviointivaihe
2012-09-25 15:41:05	Case	Tilausvaihe
2012-10-01 14:35:04	Toiminta	Valmisteilla



Picture 21. Listing of reports

The final list is simple and easy to read, with a big font for older users. The first report in the list is from a time the date and time of a report was not yet stored, hence all zeros.

5.3.7 Viewing of old reports

Possibly the feature that required the most thought to get right was the viewing of old reports. Essentially the idea was to load a report, then load the questions in that report. The main jQuery script of Laaturpuntari was copied, since while lot of the code was exactly the same, some of it was different enough that it wasn't a good idea to try and push it all in one file.

First, a page is called with the reports id as one parameter through HTTP GET. If a report with the ID exists, the userID is checked. If the userID of the report matches with the user that is logged in, the page is loaded. The report is stored into a hidden input box and then read with jQuery. jQuery then uses AJAX to get the report data. Maturities and category from the report array is used for a second AJAX call to get the question titles and answers. After that, a modified version of the Laaturpuntari question render function throws all questions and answers on to the page, checking all correct boxes and writing correct data to input boxes.

Because scores are NOT stored along with a report, the scores are calculated again at this point. After everything has been rendered, a similar view to the analysis view of a questionnaire is displayed. The only difference is that there is no 'Ready' -button at the bottom of the page because nothing will be stored. A PDF can be printed here as well and it will look exactly like the one that may have been printed when the report was stored.

Because of the number of lines involved in viewing the report, only a part is showed as an example:

```
function getData() {  
    $this->autoRender = false;  
  
    if($this->request->is('ajax')) {
```

```

    $report = $this->Report->find('first', array(
        'conditions' => array(
            'id' => $this->request->query['id'],
            'recursive' => '1'
        ));
    $report['Report']['data'] =
        unserialize($report['Report']['data']);
    echo json_encode($report);
} else {
    $this->redirect($this->referer());
}
}

```

This is the function used to fetch report data from the database. For retrieving questions -data from the database the same function is used as when filling out a questionnaire.

5.3.8 Enhancing PDF printing

In the first beta of Laatuspunti, the PDF printout featured question titles and one answer from however many were initially chosen. After some work this was fixed to printing out all the chosen answers and included input box data in the right place, thanks to the regular expression and replace -function.

In a meeting of the workgroup it was brought to attention that a PDF printout is useless if it doesn't show any score from the analysis. Two possible solutions to this came in mind: calculate the scoring again server-side or add a hidden box for each question to store the score from 3 to 0 (good to others). The box was created, named score_<question-id> and the parser was modified to ignore the score everywhere else but in the PDF printing.

When the score was available at the PDF template, work began on showing the analysis in the PDF. Because no interactivity can be had in a PDF, it was decided that the PDF would contain the four circles and then all of the questions in the same order they are originally presented. A small colored circle would be added next to each question title to present how that particular question was answered. It was time to place the circles and change the layout. Since the only way to get everything in the

right place was to change a few pixels and print a new PDF, the fine tuning took some time. After a hundred or so PDF files later the graphics were in place.



Picture 22. PDF print with graphics in place

5.3.9 User feedback

After a while it was getting cumbersome to receive feedback regarding the site to email, a feature to give feedback was requested. A similar look to mobile marketplace feedback components was proposed, with a star-based rating, email address, date and the actual review.

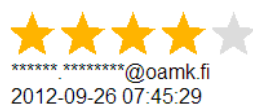
Anyone could view public feedback, but only registered users could post it. A table for feedbacks was created:

- id; the identifier of the feedback. Automatically incrementing.
- userID; the id of the user who posted the feedback
- star; the amount of stars the user wishes to give the site from 0 to 5
- text; textual review of the site
- date; date and time when feedback was given
- private; a flag to tell whether the feedback is public or for administrators' eyes only

The requirements were a list to browse feedback, a page to add feedback and ability to remove feedback. The latter is given to the author of the feedback and to all administrators, as a means of moderating without touching the database.

As one feedback takes some screen real estate, it was decided that a maximum of ten feedbacks would be shown at a time. A page selector was written to be sufficiently dynamic. jQuery was deemed unnecessary here and for each page change a new database query is done for only ten results, making queries fast and light.

Käyttäjien antamaa palautetta



Bugeja etsimässä, eikä turhaan. Tästä se paranee!



Osio: Living Lab ja opetus, arviointi ja palaute
 - Palaute jää vielä aika orvoksi, koska vastaaja ei saa tietoa, että mikä olisi optimaalista / hyvää toimintaa...
 olisiko sitä mahdollista vielä lisätä?
 - "Miten palautetta käsitellään" -kysymyksessä vastaus: dokumentoidaan ja arkistoidaan antaa punaisen laikan?
 Pitäisikö sen olal mieluummin keltainen? Vai mitkä ovat



Opetusvaihtoehto / en ymmärrä kysymystä
 Kysymys 4/9 on aika vaikea vastata

Picture 23. List feedback

Censoring email addresses was added after questions about invading privacy were raised at a workgroup meeting. Email addresses are visible for administrators in case some contact is desired. The beta version of the feedback feature raised many questions. Some modifications were made because of them:

- Added information on how to use the feedback form
- Made the text boxes in list view not look like they could be edited

Anna palautetta Laatupuntarille

Palaute

Arviosi (0-5 tähteä)

0
 1
 2
 3
 4
 5

Yksityinen palaute

Ohje:

Kirjoita palautelaatikkoon palautteesi, vapaa sana.
 Jos haluat, että palautteesi näkyy vain pääkäyttäjille, valitse 'yksityinen'. Muussa tapauksessa palaute on julkinen.
 Muista vielä arvioida Laatupuntaria asteikolla 0 - 5 tähteä, viiden ollessa paras arvio.



Picture 24. Feedback form

In the end the workgroup was content and some feedback came from the release event of Laatupuntari on 12.10.2012. Safe to say at that point Laatupuntari was a complete product since no critique was given.

6 CLONING LAATUPUNTARI FROM GIT

6.1 Installing msysgit

First a Git client is downloaded (Website of msysgit 2012).

Welcome to the home page of Git for Windows



Git is a powerful version control system aiming to be the fastest decentralized source code management tool on this planet.

Having its root in the Linux development community, Git used to be quite dependent on POSIX features usually only provided by Unix-style Operating Systems. Thanks to the efforts of a few contributors, this project succeeded in providing an almost feature-complete fork of Git on Windows. Being solely driven by volunteers in their spare time, it is nevertheless quite stable.

In order to develop Git for Windows, these volunteers rely on a build environment that is based on the MSys/MinGW project. To sort out the confusion revolving around the naming scheme, let's have a look at this table:

Links:

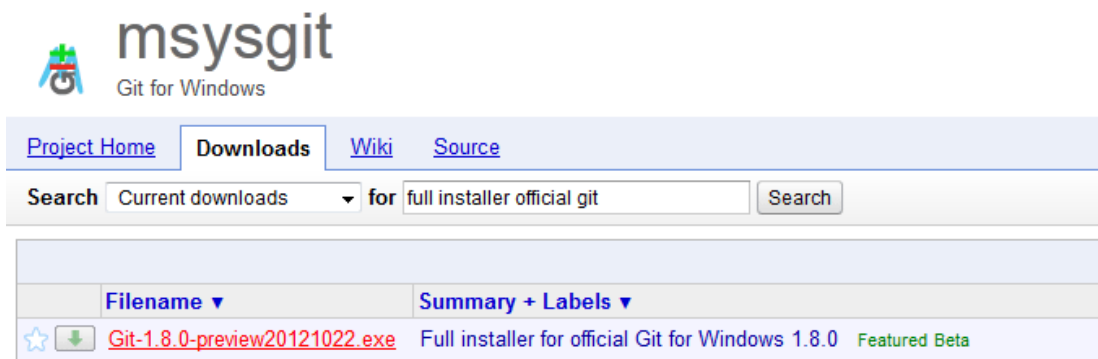
- [FAQ](#)
- [Homepage](#)
- [Wiki](#)
- [Downloads](#)
- [Downloads \(build environment\)](#)
- [Repository](#)
- [Repository \(build environment\)](#)
- [Mailing list](#)

	Git for Windows	msysGit
Logo:		
Audience:	Pure users of Git	Testers, developers, custom installer maintainers
Support:	Unfortunately none	Questions regarding building Git for Windows should be directed to the msysGit mailing list
Downloads:	On Google Code	On Google Code
How to start:	<ul style="list-style-type: none"> • Startmenu • Desktop shortcut • QuickLaunch icon • Explorer context menu 	<ul style="list-style-type: none"> • <code>msys.bat</code> • <code>git-cmd.bat</code> • Other, after running <code>share/msysGit/add-shortcut.tcl</code>
Bug tracker:	Closed	Please ask for help on the msysGit mailing list
Git repository:	On GitHub	On GitHub
Former names:	WinGit	N/A

Stay tuned, this page will have more information later!

Picture 25. Git for Windows homepage

Git for Windows is chosen here. It is best to download the newest version as they are sufficiently stable even if marked as a beta.



msysgit
Git for Windows

[Project Home](#) [Downloads](#) [Wiki](#) [Source](#)

Search for

Filename ▼	Summary + Labels ▼
Git-1.8.0-preview20121022.exe	Full installer for official Git for Windows 1.8.0 Featured Beta

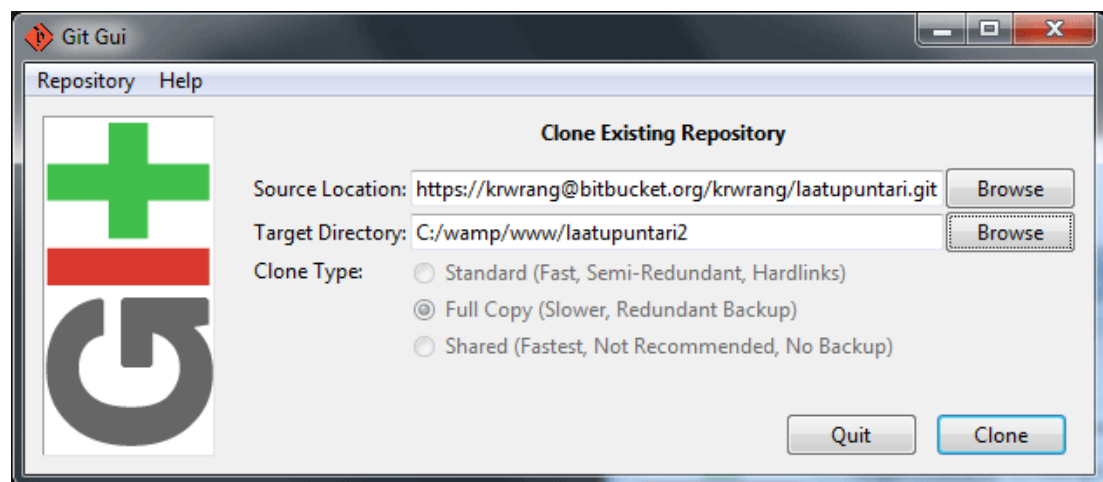
Picture 26. Git 1.8.0 download

Git for Windows installation asks several questions about how it should work. Because no special behavior is required, nearly everything is left at their default settings:

- Use Git Bash only. Doesn't modify Windows PATHs
- Use OpenSSH
- Checkout as-is, commit unix-style line endings

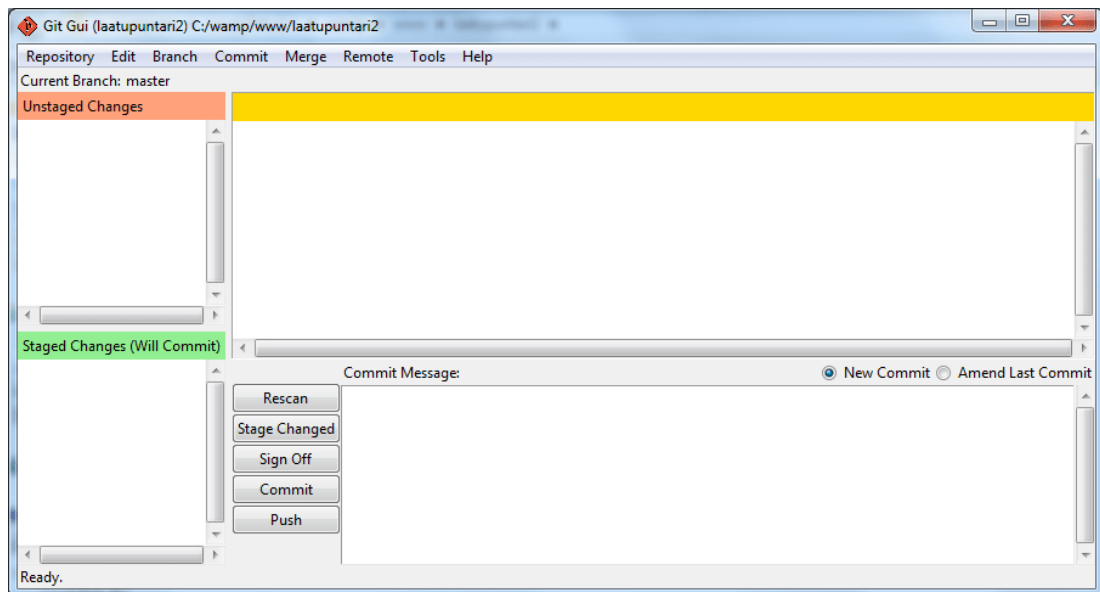
6.2 Cloning Laatupuntari from Bitbucket

Now that Git is installed, it's time to clone Laatupuntari from Bitbucket (Bitbucket: Laatupuntari 2012). Git for Windows opens automatically after installation is complete.



Picture 27. Clone Existing Repository in Git for Windows

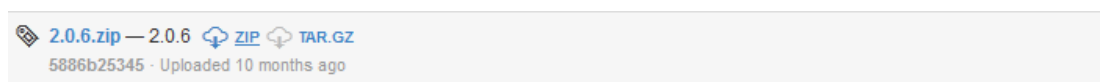
When Git for Windows opens, 'Clone existing repository' is opened. Then Laatupuntari's location and a desired target location can be set (Picture 27). Here, '/laatupuntari2' is used since '/laatupuntari' is already in use. When cloning is complete the main window is opened and no messages are given to indicate completion.



Picture 28. Git GUI

At this moment you can go check the directory and see if everything is in place. After that there are some libraries needed to be downloaded. First it's time to download a 'piece of Cake'.

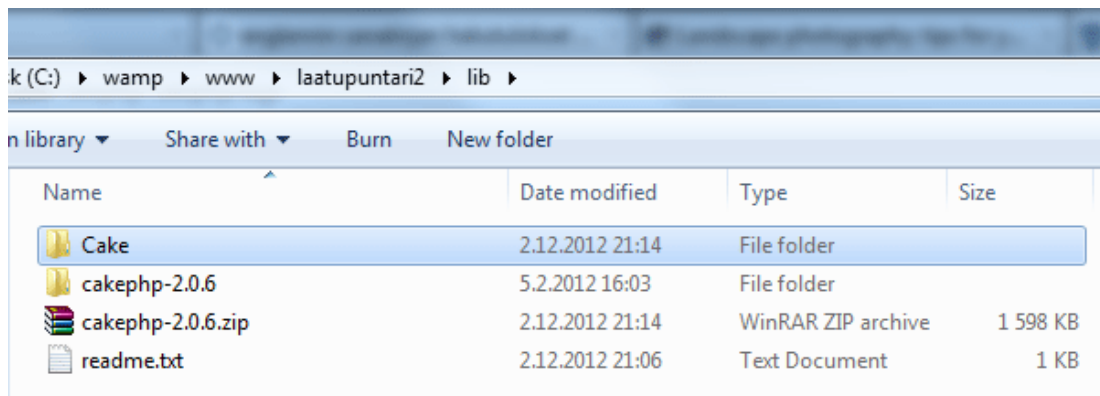
6.3 Downloading CakePHP



Picture 29. CakePHP 2.0.6 download (Website of CakePHP 2012)

The newest version BEFORE 2.1.0 of CakePHP is chosen. CakePHP 2.1 introduced some changes that would require modifications to Laatuspuntari. At this moment the improvements in newer versions of CakePHP are not critical and upgrading is not a priority.

CakePHP comes in an archive of your choosing; here we demonstrate usage on Windows so ZIP is chosen. Normally the archive would be extracted to www-root or any place of preference, but now only one directory is needed: 'lib/Cake'.

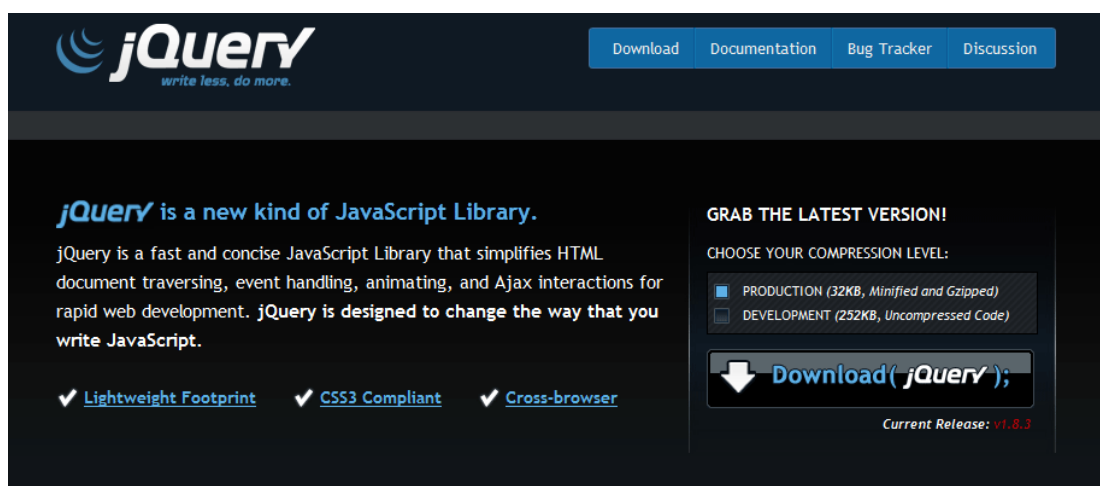


Picture 30. Placement of CakePHP

CakePHP is extracted to 'laatupuntari/lib'. Inside the newly extracted cakephp-folder is 'lib/Cake'. Take the 'Cake' -folder and place it to 'laatupuntari/lib', like illustrated (Picture 30).

6.4 Downloading jQuery

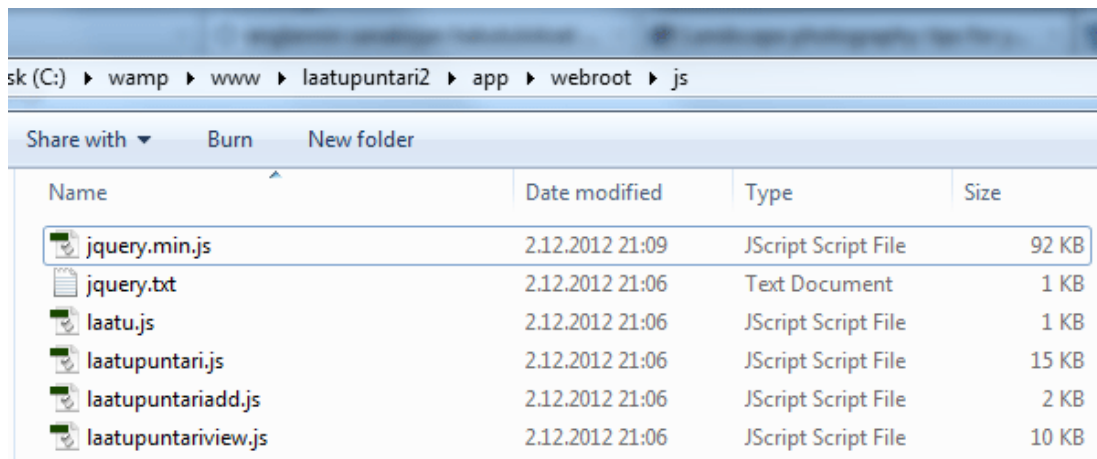
The most important part is now installed, but because the main functionality of Laatupuntari is dynamic and uses jQuery, a jQuery package is needed.



Picture 31. jQuery home page. (Website of jQuery 2012)

Production version of jQuery is chosen, because we do not intend to do any modification to jQuery itself. The latest version is good for the foreseeable future and no code altering modifications have been made as of late.

On the development machine the jQuery download opened in the browser window as plain text. ‘Save Page As’ can then be used to save the file where we need it: ‘laatu-puntari/app/webroot/js/jquery.min.js’.

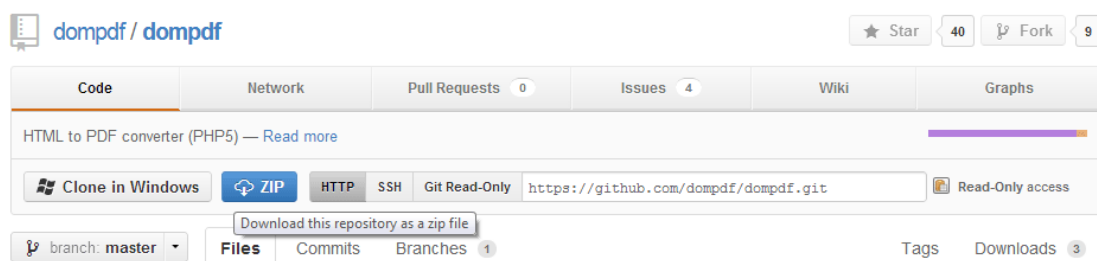


Picture 32. Placement of jQuery

By default the file name contains the version that was downloaded, so it is important to rename it as ‘jquery.min.js’.

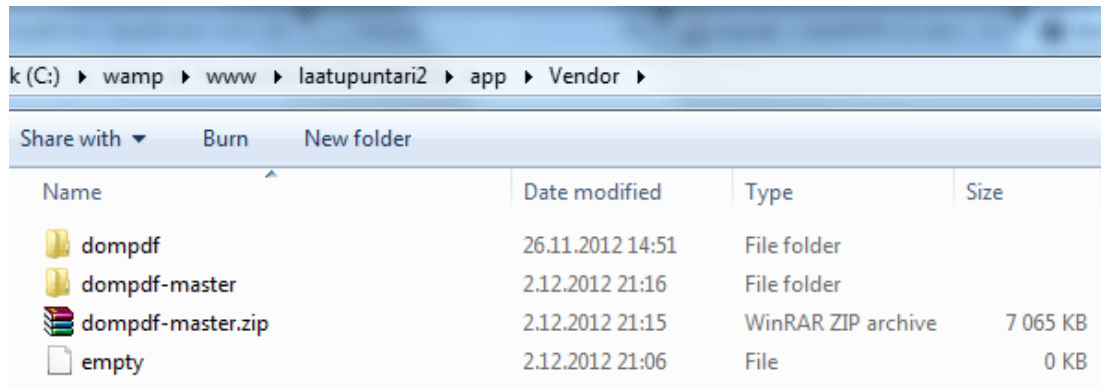
6.5 Downloading dompdf

At this moment the important files are in place. Before going forward to set up the database, dompdf will be downloaded.



Picture 33. dompdf repository on Github. (Website of dompdf 2012)

Since no interest is had in keeping dompdf as up-to-date as possible, Git Clone is skipped and a plain ZIP archive downloaded. The contents of the ZIP are then extracted to ‘laatupuntari/app/Vendor’.

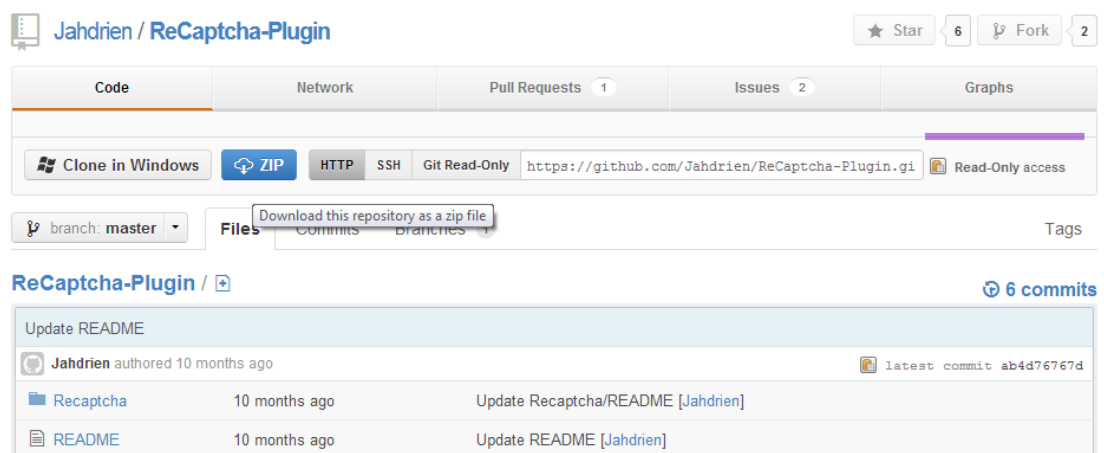


Picture 34. dompdf-master.zip extracted.

The file extracts to folder ‘dompdf-master’, which can be renamed to ‘dompdf’. As of writing, the default configuration in dompdf is good for use with LaatuPuntari.

6.6 Downloading reCAPTCHA-plugin

Finally, the last piece of software needed is the reCAPTCHA-plugin for CakePHP. For this, Jahdrien’s ReCaptcha-Plugin is used (Jahdrien 2012). First the plugin is downloaded from Github (Picture 35).



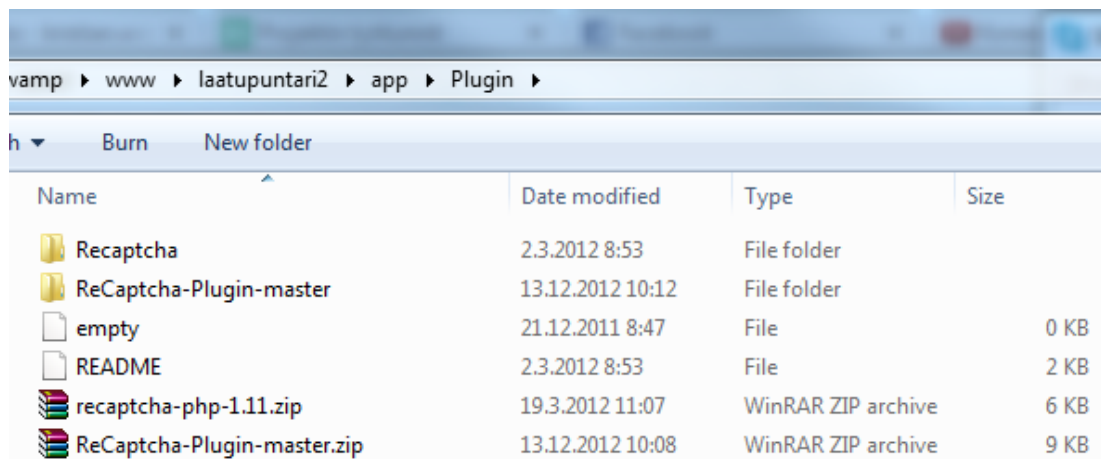
Picture 35. ReCaptcha-Plugin on Github.

Following the instructions on either site, an official reCAPTCHA Library file is downloaded from Google Code (Website of Google Code 2012) (Picture 36).



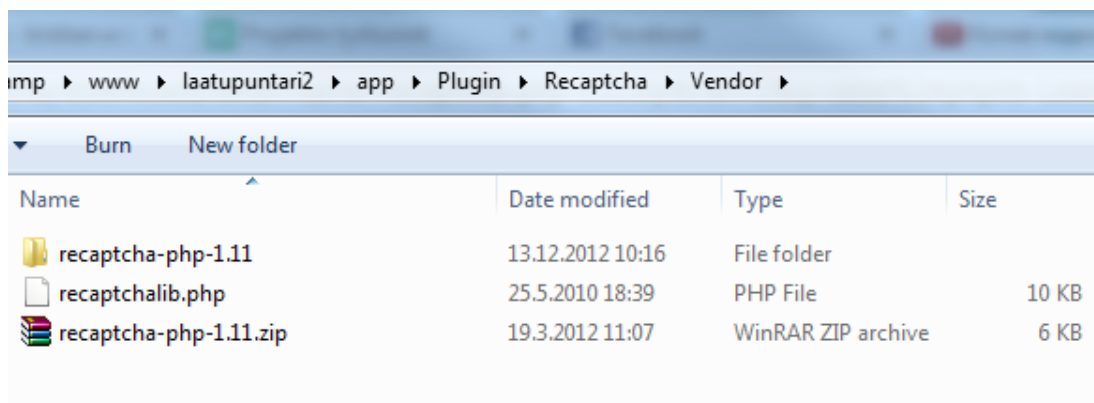
Picture 36. reCAPTCHA Library for PHP.

The plugin is then extracted to 'laatupuntari2/app/Plugin' (Picture 37).



Picture 37. ReCaptcha-Plugin extracted in 'app/Plugin'.

The folder named 'Recaptcha' inside the extracted file is then moved to 'app/Plugin', placing it where CakePHP and Laatupuntari can use it. The official library file from Google Code is then extracted as 'app/Plugin/Recaptcha/Vendor/recaptchalib.php' (Picture 38).

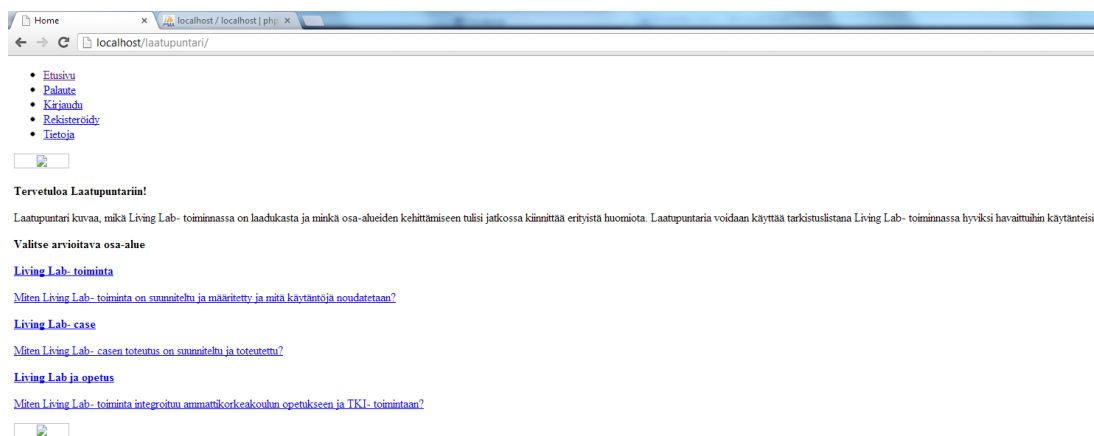


Picture 38. Official reCAPTCHA Library extracted.

The final part is setting up the reCAPTCHA key that can be acquired from Google. For this an example file 'key.php.default' exists in 'app/Plugin/Recaptcha/Config'. After renaming the file to 'key.php', both public and private keys can be entered. Bootstrapping and other configuration is already placed in LaatuPuntari so reCAPTCHA is now functional and available for security in user registration.

6.7 Configuration and possible problems

Now that everything - besides the database - is installed, the home page of LaatuPuntari is checked.



Picture 39. First look at a local LaatuPuntari copy.

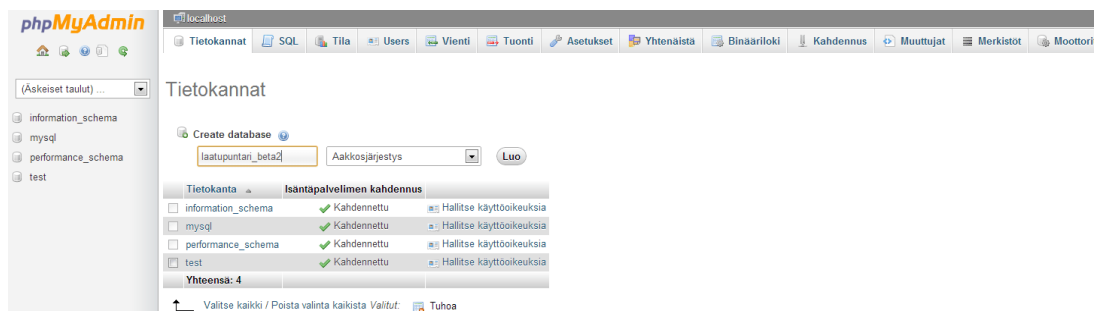
The main page opens, but without images or styling (Picture 39). This is most likely caused by the servers rewrite module not being loaded (Website of Apache HTTP

Server 2012). In WampServer the easiest way to enable this is to left click the WampServer icon in system tray, go to Apache->Apache modules and click on ‘rewrite module’. After an automatic server restart a refresh of the home page yields better results (Picture 40).



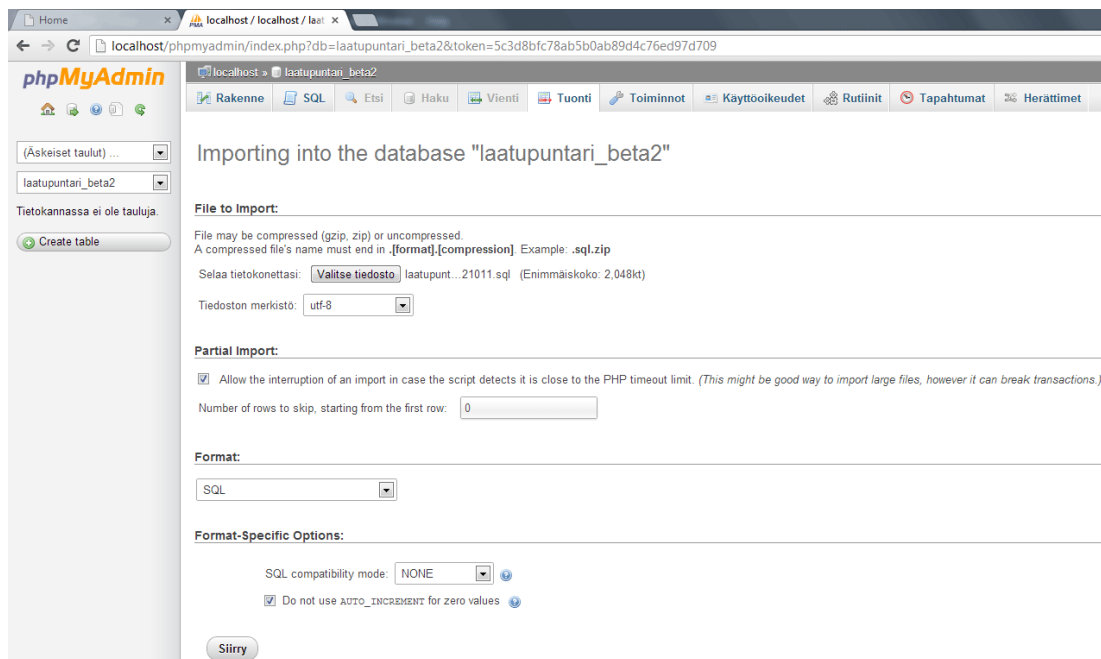
Picture 40. Second try at viewing the main page of LaatuPuntari.

At this moment it's best to handle the database. A dump of the production database is used in this example. First, localhost/phpmyadmin is opened.



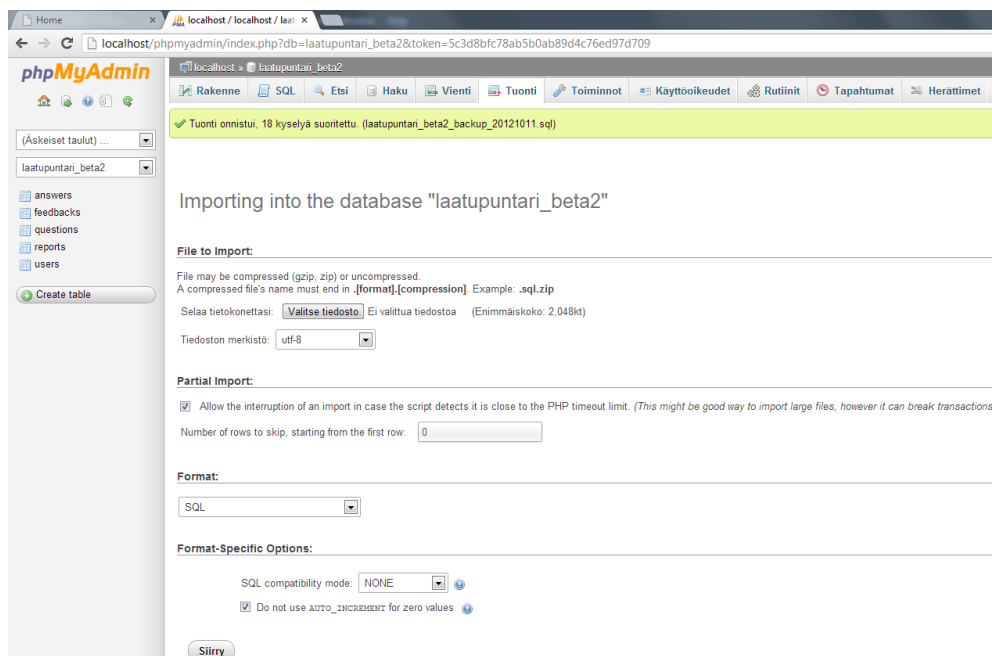
Picture 41. Create a new database in phpMyAdmin

The database can be given any name desired as it will be set later to the database configuration of LaatuPuntari. Here ‘laatuPuntari_beta2’ name is used to avoid having to change configuration.



Picture 42. Import to database

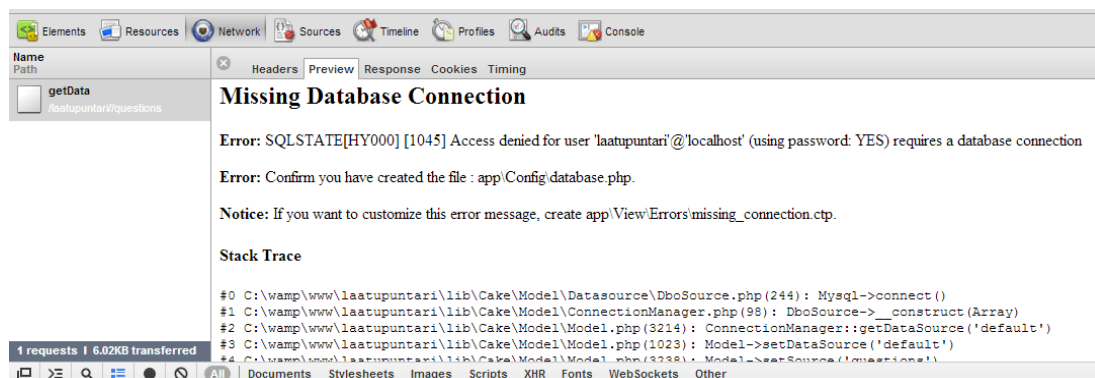
Now that the new database is created, we can select the database and go into Import. In Import the MySQL dump is loaded.



Picture 43. Database import successful

Now that the database is in place, let's see how Laatupuntari behaves. The first category is chosen and the first questions answered. However, when the actual questions from the database should be presented, a blank page is all that is shown.

The Element Browser in Chrome provides excellent debugging tools for web developers and that will be used here. A look at the Network tab proves that something is wrong.



Picture 44. Missing Database Connection in Chrome Element Browser

The database configuration was not done and a new database user needs to be created, too.

In phpMyAdmin a new user is created under the name 'laatupuntari'. The user is created for use with localhost only and full privileges are granted to the newly created 'laatupuntari_beta2' -database. After that, some configuration needs to be made in Laatupuntari. That configuration can be found in 'laatupuntari/app/Config/database.php'.

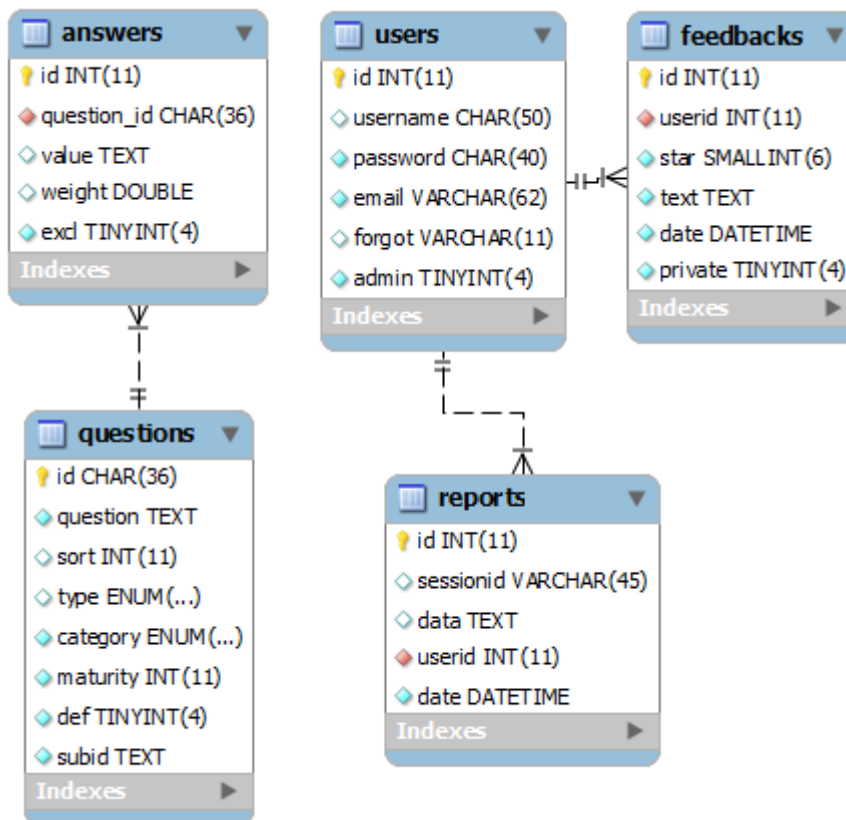
```
public $default = array(
    'datasource' => 'Database/Mysql',
    'persistent' => false,
    'host' => 'localhost',
    'login' => 'laatupuntari',
    'password' => 'laatu',
    'database' => 'laatupuntari_beta2',
    'prefix' => '',
    'encoding' => 'utf8',
);
```

Host is best left to 'localhost' for development environments. However, in a production environment this naturally needs changing if a separate database server is used. Database name, login and password should all be set according to the user and database just added in MySQL.

After these changes everything should be in working order. One thing that needs to be remembered is that the directory 'laatupuntari/app/tmp' and its subdirectories need write access. If used on a *nix-environment these permissions have to be included or an error message will be displayed. Fortunately the error messages in CakePHP are normally very informative and problems are easy to find.

7 FINAL DATABASE

After all the new functionality and features have been implemented, it is time to see how the database has changed. The preliminary version was covered in chapter 4.1.1 and the final version is as follows:



Picture 45. Final database of Laaupuntari.

Because some of these tables have already been explained in the beginning, let's go through the new tables and changes.

Answers:

- weight; stores the score, or weight, of the question. For example, a good answer could have the weight value of 3, while a mediocre answer has the value of 1. A bad answer has no value at all and an answer that cuts the question out from the analysis has the value of -1.
- excl; marks the answer as exclusive; it cannot be selected with any other answer option.

Questions:

- def; marks the question as having the power to hide or show a question further along the questionnaire. For example, a question with this flag might be along the lines of "Do you know what a Living Lab is?" and a follow up would be "How long have you known?" or "Where did you find out?" Then, according if the first one is answered as "Yes", the next one is enabled.
- subid; for the question that depends on the answer of a previous question. Holds the ID of the former question and the number of the answer that can enable it.

Users:

- id; identification number of the user, automatically incrementing.
- username; holds the username of the user. Email address is used as username.
- password; the password of the user. Encrypted by CakePHP.
- email; the email address of the user. DEPRECATED, exists just in case for now.
- admin; a flag to tell if the user has administrator rights or not
- forgot; a randomly generated string to use when a user forgets his/her password

Reports

- userID; the id of the user the report is saved to. For anonymous users this will be -1

- date; the time and date when the report was saved

Feedbacks

- id; identification number of the feedback, automatically incrementing
- userID; the id of the user who submitted the feedback. Since anonymous users are not able to post feedback, this will always be >0
- star; the amount of stars the user gives as feedback. Rating from 0 to 5 is used.
- text; the textual content of the feedback
- date; time and date feedback was given
- private; a flag indicating whether the feedback should be only viewable by Laaturuntari administrators

The database that was initially three tables grew into five. This can be counted as a small victory, as many projects like this often have significant technical changes along the development cycle. A perfect document of requirements in the beginning would have largely eliminated even these small changes, but even in a scenario like that preferences change and people get new ideas.

8 WHAT WAS LEARNED

8.1 Working in a project as part of a team

Developing Laaturuntari taught me what it is like to work in a project with budgets, deadlines and everything. What is important is to know your limits and when the project workgroup asks for the moon from the sky, you have to know what you can do in the time that is given to you. Distances between project members were long and nearly all communication took place online. Adobe Connect Pro meetings were held every month to discuss what had been done and what is still needed. Often new ideas destroyed old ones and keeping track was not optional. One of the most troublesome things was when a new feature required big changes and every detail was not yet thought out. Outside of ACP meetings, emails were often the only means of getting

hold of more than one project coordinator at a time. To minimize the risk of having to rewrite heaps of code, it was often best to wait for a response on big changes without making uninformed decisions.

A larger group has benefits such as more efficient testing. Every user is different, every user accomplishes tasks differently. This difference is what makes for good testing of software. The developer can never find all bugs in his/her software and that is where different habits of using a computer come in handy.

Workshops and demonstration events produced heaps of suggestions and bug reports, making the end product that much more polished. Not all feedback was about bugs or feature suggestions; Laatuspuntari is all about Living labs and many of the comments about Laatuspuntari were content related.

8.2 Using new tools

When I started working on Laatuspuntari, much of the tools used were completely new to me. Even though I had used XHTML, PHP, CSS and JavaScript extensively before, CakePHP and jQuery opened huge new possibilities. Because jQuery and CakePHP do much of otherwise time consuming work for you automatically, more time can be directed at better functionality, usability and design. After this project I am more confident than ever at taking on bigger projects using these open source tools.

8.3 Laatuspuntari now and possibilities in the future

The version of Laatuspuntari that was created is a great tool for making dynamic questionnaires, not just living lab related but any type of survey that benefits from having an analysis in the end. Depending on the type of use, Laatuspuntari could easily be converted to all sorts of uses with the amount of changes depending on the intended usage.

The final version of Laaupuntari does nearly everything that was dreamed of in the beginning (Website of Laaupuntari 2012). In case of future use and continued development, the biggest change that Laaupuntari would need is probably making everything related to a series of questions come from the database. Information pages, titles, preliminary questions. Along with a dedicated editing tool, this change would make it possible for new users to create questionnaires and propose their use as part of Laaupuntari. Endless possibilities like commenting on those proposed question series, proposing changes, accepting changes and others would make Laaupuntari even better than it is now.

REFERENCES

Website of Neloskierre. Referred 8.12.2012. <http://www.neloskierre.fi/>

Website of CakePHP. Referred 8.12.2012. <http://www.cakephp.org/>

CakePHP: Sites in the wild 2012. Referred 8.12.2012.
<http://book.cakephp.org/1.2/view/510/Sites-in-the-wild>

Web Technology Surveys (W3Techs): Usage statistics and market share of PHP for websites 2012. Referred 8.12.2012.
<http://w3techs.com/technologies/details/pl-php/all/all>

Website of MySQL 2012. Referred 8.12.2012. <http://www.mysql.com/>

Website of jQuery 2012. Referred 8.12.2012. <http://www.jquery.com/>

Olsson, T., O'Brien, P 2007. Workarounds, Filters and Hacks. Sitepoint. Referred 8.12.2012. <http://reference.sitepoint.com/css/workaroundsfiltershacks>

Website of dompdf 2012. Referred 8.12.2012. <http://code.google.com/p/dompdf/>

Website of reCAPTCHA 2012. Referred 8.12.2012.
<http://www.google.com/recaptcha>

Website of Eclipse 2012. Referred 8.12.2012. <http://eclipse.org/>

Website of Bitbucket 2012. Referred 8.12.2012. <https://bitbucket.org/>

Website of WampServer 2012. Referred 8.12.2012. <http://www.wampserver.com/en/>

Website of msysgit 2012. Referred 8.12.2012. <http://msysgit.github.com/>

Website of Java 2012. Referred 8.12.2012. <http://www.java.com/en/>

Website of Github 2012. Referred 8.12.2012. <https://github.com/>

Constantinescu, R. 2012. Case Neloskierre.fi. Final Year Project. Turku: Turku university of applied sciences.

Golding, D. 2008. Beginning CakePHP: From Novice to Professional. Berkeley: Apress.

Bitbucket: laatupuntari 2012. Referred 21.12.2012.
<https://bitbucket.org/krwrang/laatupuntari>

Wikipedia: Regular expression 2012. Referred 8.12.2012.
http://en.wikipedia.org/wiki/Regular_expression

Jahdrien 2012. reCAPTCHA Plugin for CakePHP2. Referred 10.12.2012.
http://bakery.cakephp.org/articles/jahdrien/2012/02/15/recaptcha_plugin_for_cakephp_2

Website of Apache HTTP Server 2012. Referred 12.12.2012. <http://httpd.apache.org/>

Website of Laatupuntari 2012. Referred 21.12.2012. <http://www.laatupuntari.fi/>