Timo-Pekka Kemppainen

Data Archive Project

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Media Engineering

Thesis

28 April 2015

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Timo-Pekka Kemppainen Data archive project 31 pages 28 April 2015
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	
Instructor(s)	Aarne Klemetti, Senior Lecturer

In the modern world, digital data archiving and accessing is an issue that needs tailored solutions. This thesis gives one example of designing and implementing a data archiving solution using time tested technologies.

Helsinki Metropolia University of Applied Sciences had a need to archive their contracts in digital format instead of using old file cabinets. Specifications were created together with the customer at the start of the project. The project was to produce a complete product for the end user to work with.

The project followed a principle where the end user created ‘user stories’ which were used to create product specifications. Database designs were created using user requirements. A server side implementation was done with the PHP scripting language and MySQL database. A Web based user interface was selected for simplicity.

The project was more of a learning experience and did not produce a usable product for the end user. All material produced in the project can be used as future reference for database design and PHP implementation. Other notable points include usage of pre-existing frameworks and development environment setup. Database design and creation scripts can be used as a basis for future projects.

In conclusion the project primarily provided guidance about workload management.

Keywords	PHP, MySQL, LAMP

Tekijä Otsikko Sivumäärä Aika	Timo-Pekka Kemppainen Tiedon arkistointisovellus 31 sivua 27.4.2015
Tutkinto	Bachelor of Engineering
Koulutusohjelma	Media Engineering
Ohjaaja	Lehtori Aarne Klemetti

Insinöörityön päämääränä oli tuottaa toimiva ohjelmisto sopimusten digitaaliseen arkistointiin. Asiakkaan kannalta ohjelmiston tärkeimmät ominaisuudet olivat kattava tiedon etsintä, metadatan tallennus ja helppokäyttöisyys. Projektin tavoitteet päätettiin yhdessä asiakkaan työryhmän kanssa.

Projektin mittakaava luotiin asiakaslähtöisessä ’käyttäjälähtöisen kehityksen’ kehitysmallissa, jossa asiakastyöryhmä otettiin tuottamaan käyttäjätarinoita. Käyttäjätarinoiden tavoitteena oli luoda kattava yleiskuva siitä, mitä ohjelmistolta vaadittiin.

Insinöörityössä tuotettiin MySQL-tietokannan suunnitelmat asiakkaan toiveiden perusteella. Tietokantasuunnitelmaa seuraten luotiin testiympäristössä toimiva toteutus. Tietokantaa käsittelevät osat luotiin PHP-ohjelmointikielellä, mikä mahdollisti web-pohjaisten ratkaisujen käyttämisen käyttöliittymässä.

Projektissa onnistuttiin tuottamaan toimivat tietokantasuunnitelmat, palvelimella toimiva tietokanta sekä PHP-komentosarjoja tiedonkäsittelyyn. Projektissa ei onnistuttu tuottamaan kokonaista toimivaa tuotetta asiakkaalle.

Projektissa saadut kokemukset osoittivat kokonaisen ohjelmiston luomisen tyhjästä erittäin suureksi työksi. Valmiita ohjelmistokirjastoja käyttämällä olisi voitu säästää paljon aikaa ja ehkä mahdollistaa projektin onnistuminen. Insinöörityö toimi oppimisprojektina, joka tuotti referenssimateriaaliksi sopivan tietokantasuunnitelman sekä palvelinkomentosarjoja.

Avainsanat	PHP, MySQL, LAMP

Abstract

1. Introduction

The purpose of the project was to produce a working database solution for Tutkimus-, Kehitys- ja Innovaatiotoiminta (TKI) department at Helsinki Metropolia University of Applied Sciences, located in southern Finland. A solution was needed to allow faster access to different projects in which Metropolia is participating. Currently the TKI department is using file cabinets to store information about projects.

The TKI department is responsible for keeping track of all the past, present and the future projects at Metropolia. This information is used by the school management to decide where to invest their money. Such projects can be simple such as updating workstations at school.

Data Archive Project was important for several reasons. The TKI group had need for a tailored solution to store documents electronically. To understand the project needs properly, it was critical to keep close contact between the developer and customer. The customer needed robust tools to control data searches and required specially tailored a user interface. Because of this, the project was decided to be built from scratch instead of using readymade software.

The technologies chosen for this project were all robust and time tested. All of them have been used by the Information Technology industry for years and they were very well documented. This allowed quick development and removed risk from projects overall success.

Scope of this project was to implement a database on a live server, along with data access layer, business logic and an HTML 5 based graphical user interface. This web application would help the TKI department greatly by eliminating the need to browse manually through project archives, locate projects and do all estimation calculations by hand.

The goal was to have a complete web application, done completely free of charge, from ground up by a student in one year.

2. Theoretical Background

2.1 LAMP

LAMP (Linux, Apache, MySQL, PHP) is an acronym that comes from technologies that are together used to provide a basis for any dynamic web application. The acronym was created by Michael Kunze in an article written for a German IT magazine. All LAMP technologies are free to use. [2,8.]

The technologies are the following:

Linux is a UNIX based operating system that gives a basis for all other technologies. Linux is not mandatory and other operating systems can be used instead [2,8].

Apache is a popular web service platform that provides an interface with server side scripts and clients [2,8].

MySQL is a free database management system [2,9]. (See section 2.4 for more information about MySQL.)

PHP Hypertext preprocessor is a server side scripting language that outputs HyperText Markup Language (HTML) [2,9]. (See section 2.5 for more information.)

2.2 Relational Database

Relational data is a set of information that is related to other information with an identifier. In relational databases, all data is stored in data tables. A data table can be described as rows and columns. Each column holds a single typed value and each row holds data for a single entry in a table. To have a relational database, two tables are minimum requirement. For example if a table is created that holds information about teachers, the columns in the table could be Name, Teacher ID and Salary. Each row would hold an entry for single teacher. [5, 143.]

It is possible to define some columns uniquely. This means that two rows can have the same value in the column defined as unique. For example Teacher ID could be a unique column. This means that a single teacher could be identified by ID alone, even though

there might be teachers with the same name. [5,150.] Teacher ID could also be defined as a key column (which would automatically mean it is unique). A key column means that other tables can refer to this table by using values in the key column. [5,150-154.]

The second table could be about the courses offered by the school with columns: Course name, Course ID, Teacher ID and Credit points. Course ID could be a unique key value. Instead of writing the teacher manually on each row, it is possible to relate these two tables using a primary key and a foreign key pair. The teacher ID in the course table can be designated as a foreign key which takes values from the teacher table's Teacher ID.

This way the course table and teacher table are related. If information on the teacher table is changed, for example a teacher receives a pay rise, this does not cause a problem where two tables would have different information [5,151-154].

It should be noted that in relational databases, each cell should be atomic. Atomic means that a cell only contains basic data that cannot be divided into components. In the example given above, the name column would contain both the first and the last names. The columns should be divided so that one column holds information for the first name and the other column holds information for the last name [5,147].

2.3 Temporal Relations

Temporal relations or temporal database means database where information is related to the time component. For example in healthcare, patient information needs to be kept intact so doctors can see what treatments have been given and when [5,819].
One example of temporal database is Valid Time relation. Valid time can be described as a perioid of time when the entry is considered to be true.

For example an employee salary can change during his/her time in a company. The employee can get a pay rise but for taxing purposes the company might want to keep track of previous salaries in given time period. In practice, valid time is used as an two column in a database. The column valid_start_time indicates us point in time when the entry started and valid_end_time tells when the entry becomes old [5,822].

When the entry gets old or is updated in valid time relations, the column valid_end_time is closed by adding a current point in time. In case of an update, a new row is inserted with modified data and a new time period [5,825].

It should be noted that since there are multiple entries with the same ID, it is not possible to mark it as unique and to use as a primary key. A primary key is a combination of an ID and valid_start_time to identify latest certain entry [5,825].

2.4 MySQL

MySQL is a database that works with scripting languages. It is regarded as medium-scale and is suited for medium sized web applications. [1, 21.] MySQL allows multiple users over multiple threads to access relational data [2,13].

The term MySQL was created by a Sweden based TcX DataKonsult company as an internal project. The project gained popularity since it was released for public and soon it was spinned into a own separate company.

The first release was in 1996 and it lacked most of the sophisticated features of standard enterprise databases but MySQL was designed with performance and scalability in mind. This proved more important for users than unused features of enterprise databases [4,621].

MySQL supports multiple tools for management. One of the popular tools is the phpMyAdmin graphical management tool written in PHP [2,14].

The popularity of MySQL is due to its flexibility, solid performance and useful features. MySQL can be deployed into multiple operating systems, including Windows platforms, Mac OS X and many Linux distributions. It also has ready-made application programming interfaces for most popular programming languages like: C, C++, Java, Perl, PHP and Ruby [4,622].

2.5 PHP

PHP is a web server based programming language that works well in web development [1,10]. PHP works closely with HTML by having easy tools for data input and output. PHP

code is always executed on the server side and its output is pure HTML. This means that the end user does not know what underlying PHP code actually does. PHP code can only be viewed by accessing the script files on the server itself. It is also very newcomer friendly while incorporating advanced features for senior developers [1,11].

All scripting languages have ready-made tools to access databases and PHP is no exception. PHP works well with popular MySQL database [1,16]. PHP is free to use and works well with Apache servers [1, 20].

PHP can be described as open and forgiving. It is loosely typed, meaning all variables and pointers can be assigned any type value or reference instead of strictly assigning single type for single variable. Originally PHP was used as procedural language but as language was developed, Object-orientated features were added to programmer tools in later PHP versions [3, xix].

```
<HTML>
          <head>
          </head>
          <body>
                    <?php
                              echo("Hello all");
                    ?>
          </body>
</HTML>


<HTML>
          <head>
          </head>
          <body>
                    Hello all
          </body>
</HTML>
```

Listing 1. Example PHP code and its output.

Listing 1 shows how PHP code can be integrated into html code. First set of HTML tags shows a PHP script and the second set shows its output. It is important to note how PHP code is formatted under <?php ?> tags to tell the server to parse everything in between as a script.

3. Methods and Materials

3.1 Project Start

Project was started when developer was contacted by one of senior lecturers in Metropolia University of Applied Sciences. Tutkimus-, kehitys- ja innovaatiotoiminta (TKI) group in Metropolia was in need of database solution for storage of project information.

Developer contacted TKI-group representative and she gave me their first attempt at database creation. After studying Microsoft access database for a week, developer agreed on a meeting with TKI group. In meeting developer gathered information by discussing with group members about their needs.

The initial contact revealed that the TKI group needs were not that urgent but they visioned quite extensive application that could synchronize with other Metropolia systems. Since first meeting only gave developer a piece of information what TKI group actually wanted, it was decided to gather user stories by using Google Drive public documents. Access was given to whole group including customer.

As user stories were added, the scope of the project became more apparent. The customer wanted easy to use application that could handle storage of project metadata, original documents and various queries which would fetch data by multiple different parameters.

For convenience purposes it was decided that application would only work in the Metropolia intranet. Creating secure web application would only complicate project needlessly.

More meetings were held about closer details of user stories and different Metropolia systems what customer wanted to include in project. It was decided that application uses its own user control system.

Tools

Tools chosen for this project were quite simple. Eclipse was chosen as the development editor for several reasons. It was free software. It had a plugin for PHP development which provided advanced tools for language debugging, project file management and task listing.

Eclipse also worked with the virtual Apache server used in project quite well. Code could be debugged in runtime and the editor had all ready-made PHP libraries installed. Eclipse text editing features were also part of the reason to choose Eclipse. Options to edit the text view were extensive and editor pointed out syntax issues immediately. [7.]

Due to the nature of PHP being a server side scripting language, a virtual server was required for development. Normally PHP cannot be tested locally unless the Apache web server is running locally. This was remedied by installing free software called XAMPP. XAMPP provides easy to install virtual web server that has Apache and MySQL capabilities. [8.]

The Reason to use XAMPP was simple. Acquiring virtual server from school proved problematic, development environment needed to be outside of school (for convenience), installing a secondary computer with LAMP would be time consuming and pointless considering that the final platform would be in the Metropolia intranet.

Database planning was done with Libre Office Draw. Libre Office is a free competitor for the Microsoft Office family. The Libre Office package contains tools for document editing, an Excel equivalent and drawing tools. [9.]

The Dropbox service was used for data backup and files transfer between workstations. [10.]

Environment

Development environment was selected to be local. It was important that project could be developed outside school, which ruled out virtual servers in the intranet. The project

was also developed at multiple workstations so environment was required to be easy to setup.

Money was also an issue because project had no funding, all solutions needed to be no cost solutions. This practically meant that all workstations needed to be already available. A Decision to use a personal computer and a workstation at work was made.

Environment involved installation of Eclipse, XAMPP and document manipulation tools. After installing XAMPP, virtual servers were tested and Eclipse workspace was pointed under htdocs folder of XAMPP allowing PHP script execution from the editor. [11.]

All working files were backed up to Dropbox except for code binaries and database. Code binaries were backed up manually and all temporary data in the database was expendable. [10.]

Workflow

The work was divided into several parts. The first part was designing the database using the Libre Office Draw software. User stories were used in drafting. The work mainly consisted of creating database tables and designing relations.

The second part was database implementation to the XAMPP MySQL server. The creation was based on trial and error. All SQL creation commands were created by hand and placed into correct order so relations could be formed by SQL engine.

The third part consisted of PHP layer creation, database I/O and overall infrastructure. The PHP layer was divided into input code, output code and tool code. The input code handled everything that was supposed to go into the database, the output was a semi-dynamic querying system and tool scripts provided development tools, such as the Populate Database Script (used to create dummy data for testing).

The Fourth part was the HTML layer with AJAX commands. HTML was written with the Eclipse editor.

3.2 User Stories

User stories were collected from customer using Google Drive (back then Google Docs). The reason for this was simple. It is hard to remember everything that is needed in contact meetings. Online document could be edited as soon as new idea emerged. This allowed every needed detail to be recorded the very instant the customer remembered it.

User stories were strictly formatted. Customer had to fill in rows of the following columns: “Kuka haluaa tehdä”, “Mitä haluaa tehdä”, “Miksi haluaa tehdä” and “Prioriteetti”. Loosely translated they mean: “Who wants to do”, “What he/she wants to do”, “Why he/she wants to do” and “Priority”.

Since the project aimed to have different levels of users, there was a need to distinguish different users from each other and what said users could do. The “Who wants to do” column sorted out all different roles the customer wanted to have.

It was also needed to know what customer wanted to do with application. The “What wants to do” column collected individual actions customer might want from the software. Each of the items was a single action that the user could do. This made it easy to create business logic from direct customer input.

3.3 Database Design

The database was designed to revolve around projects. All related permits and metadata would be tied to the project table by using special Join Tables in between. All tables were also designed to be temporal, the Join Tables would take care of keeping correct versions of metadata tied to the correct version of the project.

Since metadata and permits were their own tables, this allowed the same instance to be used in multiple projects. This also removed confusion where multiple instances of the same metadata would have to be added and maintained. Maintenance of multiple metadata entries would have been tedious and error prone process.

Each primary table included at least: ID, valid_starttime and valid_endtime. Current instance could be identified by combining id and valid_starttime. When the instance was being replaced, valid_endtime would be filled in with a point of time and new table entry would be created with valid_starttime using the same point of time.

Updating primary tables would also require updating of related join tables. For example if the ProjectPartner table was modified by creating a new row and closing old row's valid_endtime, ProjectPartner_In_Project would have to be updated to reflect a new projectpartner_id and projectpartner_vst (vst is acronym for valid_starttime).

Documents would be stored on their own database table. The table would include all related metadata and directory where the document was stored. The documents itself were not stored in the database and would have to be backed up separately. The reason to leave documents out of the database was optimization. Fewer actions required by database seemed logical.

Updating the document would happen by uploading a file into the server, creating a new row with metadata and updating related join tables. This would preserve old files, so they could be accessed when something went wrong or when their data would be needed.

The EndUser table is the only one that is not related to the project directly. Multiple users could be created and each entry can be temporal. This would allow users to change their metadata. EndUser would be used to log into the website and used to see who had made the most recent change.

3.4 Database Implementation

Database implementation began by deciding the table creation order. All tables were interconnected through a series of join tables and table creation order became crucial because of foreign key dependencies. The first created table was EndUser, since all other primary tables were pointing at the 'ID' column in the EndUser table. To allow temporal relations to work, primary key was combination of 'id' and 'valid_starttime'. In columns where character input was required VARCHAR(255) type was selected to allow variable character lengths up to 255 characters. [12.]

All other primary tables were created after ‘EndUser’. Foreign key references were pointed to the ‘EndUser’ table columns ‘id’ and ‘valid_starttime’. At this point, the sequence in which all primary tables were created did not matter.

After the primary tables were created to the database, the Join Tables could be created. The Join Tables did not have primary keys and were connected to primary tables through two foreign keys. Each pointed to respective tables ‘id’ and ‘valid_starttime’ columns.

After all tables were created, a PHP was created to populate all tables with dummy data. This script was run in order to test if relations were working correctly. The script would go through each primary table and assign random amount of rows. Each row was populated by gibberish, random arrangement of letters. After the primary tables were populated, the script would add random join table connections.

Database queries were tested against dummy data. The first test was to query one project, then to see separately all related primary tables. This was achieved by using INNER JOINS to query data.

After manual queries were proven to be working, PHP query scripts were created by using manual queries as a basis for development.

3.5 PHP Layer Implementation

PHP files were divided by their functionalities into several categories: database access functions, common function files, class files, test files and tools.

Database access functions are simple script files where each file has a singular function. For example insert.php would handle insertion to database while fetch.php would pass queried data to end user. Reason to divide files by function was to ease AJAX implementation. By giving AJAX simple files to call, client side code would be much simpler, when compared to a situation where AJAX would need to call multiple files and process data itself.

Common function files included all common actions that the server was required to do. For example all database related sequences were stored here for easy access in all files. This would allow rapid PHP script development and reuse of code.

Each programming class was divided into their own files. This allows easier script maintenance and removed need to include unnecessary code into memory while scripts are running on the server. Examples of class files include Log.php, which would take care of all logs written by server. Logged actions include user actions, server maintenance actions and possible errors.

As stated in the database implementation part, database development required script files to test database functionalities. Testing that code works lowers the amount of mistakes, which results in better working code. Also catching possible mistakes early on made them easier to fix. Test file examples also included AJAX testing while server was keeping track of sessions.

Tools were created in order to ease some development processes. Early in the development, database design was not final. This resulted in multiple database wipes. The Problem arose from populating the database. Temporary testing data needed to be inserted again after each wipe. This proved to be slow and tedious process, which lead to creation of populateDatabase.php script file. The script inserted temporary data with relations for testing.

The PHP layer was iterated multiple times during the development. The first iteration was base files that could be reused in upcoming iterations. Such files included common files, session management and database access. On later iterations The PHP layer structure become more refined. The latest iteration changed how layer was accessed by introduction of database access functions.

3.6 Project Halt and Issues

The project was halted nine months into development. There are multiple reasons for halting which included, project scope, reform of customer team and personnel problem.

As with modern projects, scope of this particular project was defined to be quite narrow. The original scope only included an extremely simple interface to the database. As the project went forward and user stories were collected, interface requirements became more complex with each user story iteration. This increased the workload beyond one thesis.

Reform of a customer team caused some communication problems. It was unclear if the team had been disbanded by school completely or just reformed. This caused the developer to stop development to determine what to do if customer team was disbanded completely. Increased project scope could be remedied by assigning more students.

Unfortunately finding extra persons with enough knowhow or even motivation proved to be a daunting task. Several students were added to the project team but no one was competent enough to ease the workload.

The combination of these issues resulted in the project that could not be completed with available resources. Because of this, project was axed.

4. Results

4.1 Database

This project managed to produce database design to be suitable for Metropolia TKI group, implement said database and create working queries.

Listing 2 illustrates how project table was created. As with every table, id is defined as NOT NULL, meaning id must be inserted in order to avoid failure when linking tables. Most of the columns are defined as varchar(255). This gives more freedom to the end user since any string between 0-255 characters can be inserted. Columns valid_starttime and valid_endtime are related to temporality. Because of this they use DATETIME to exact point of time.

```
CREATE TABLE Project
(
id INT NOT NULL,
identifier varchar(255),
diary_number varchar(255),
projectcode varchar(255),
project_name varchar(255),
acronym varchar(255),
focuse_areas varchar(255),
metropolia_coordinator varchar(255),
degree_program_related varchar(255),
start_day DATE,
end_day DATE,
inspector varchar(255),
inspected_day DATE,
funding_application varchar(255),
funding_application_signed_day DATE,
deniend_application varchar(255),
targeted_budget varchar(255),
offer varchar(255),
invitation_to_tender varchar(255),
invention_announcement varchar(255),
```

```
ethical_estimates_on varchar(255),
general_project_information varchar(255),
valid_starttime DATETIME,
valid_endtime DATETIME,
user_id INT,
user_vst DATETIME,
PRIMARY KEY (id, valid_starttime),
FOREIGN KEY (user_id, user_vst) REFERENCES EndUser(id,
valid_starttime)
);
```

Listing 2. Project table creation SQL.

Id and valid_starttime are designated as primary key, this allows temporal relations since other tables will access the latest row version automatically. Foreign keys point at Enduser table and the latest iteration of user.

Listing 3 shows an example of a join table. Join tables keep track which versions of tables are related to each other. Each time a table row is updated, join tables are updated to reflect the latest version of the row. For example if the Project table had an update on a single project, a new row would be created. Since all tables must have id and valid_start-time, a join table can be updated to the latest iteration by an SQL update command

```
CREATE TABLE ExternalStakeHolder_In_Project
(
project_id INT,
project_vst DATETIME,
ESH_id INT,
ESH_vst DATETIME,
FOREIGN KEY (project_id, project_vst) REFERENCES Project(id,
valid_starttime),
FOREIGN KEY (ESH_id, ESH_vst) REFERENCES ExternalStakeHolder(id,
valid_starttime)
);
```

Listing 3. Join table SQL example.

Document storage table in listing 4 required multiple iterations. The original idea of file storage was to upload all files directly into database. This idea was later re-evaluated in order to ease the load on the database server.

```
CREATE TABLE Document
(
id INT NOT NULL,
document_name varchar(255),
document_type varchar(255),
document_role varchar(255),
document_data_id INT,
sign_day DATE,
valid_starttime DATETIME,
valid_endtime DATETIME,
user_id INT,
user_vst DATETIME,
PRIMARY KEY (id, valid_starttime),
FOREIGN KEY (user_id, user_vst) REFERENCES EndUser(id,
valid_starttime),
FOREIGN KEY (document_data_id) REFERENCES Document_data(id)
);
```

Listing 4. Document table creation SQL.

Files would be sent to standard fileserver in the latest iteration and they would be stored in Document table. It can be seen that this table does not follow the latest design iteration and would require more work in order to work with file server. Currently Document table would store reference to file location in database to document_data_id.

Single query was created to pull a single project out of the database. The main idea was that the user could see a list of projects and all data from a single project. These goals were defined before user stories were created and they diverge from the final goal of the project. The whole query is very long and only snippets are shown here.

Listing 5 shows the beginning of a large SQL query. The main points in listing 5 are ‘AS’ commands. With SQL queries, it is possible to select something and give it an alias for later use.

```
SELECT
project.id AS "project.id",
project.identifier AS "project.identifier",
project.diary_number AS "project.diary_number",
project.projectcode AS "project.projectcode",
project.project_name AS "project.project_name",
```

Listing 5. Snippet of project query SQL.

The code snippet in listing 6 shows how join tables are used to pull correct rows from two tables.

```
INNER JOIN (
projectpartner_in_project
          INNER JOIN
          projectpartner
          ON
          projectpartner.id = projectpartner_in_project.project-
partner_id AND
          projectpartner.valid_starttime = projectpartner_in_pro-
ject.projectpartner_vst
)
ON
project.id = projectpartner_in_project.project_id AND
project.valid_starttime = projectpartner_in_project.project_vst
```

Listing 6. Inner join example of project query SQL.

SQL database design and implementation almost met project requirements. End user defined queries are still missing.

4.2 Code

Code implementation in this thesis project did not meet user the requirements but could be produced groundwork results that can be used as reference. Produced code include the database access layer written in PHP, business logic layer written in PHP and the client layer written in HTML 5 and Javascript.

Figure 1 shows how scripts were stored on the server side. The common folder contains all scripts that are common everywhere else, such as database access scripts, common functions and common regular expression functions.

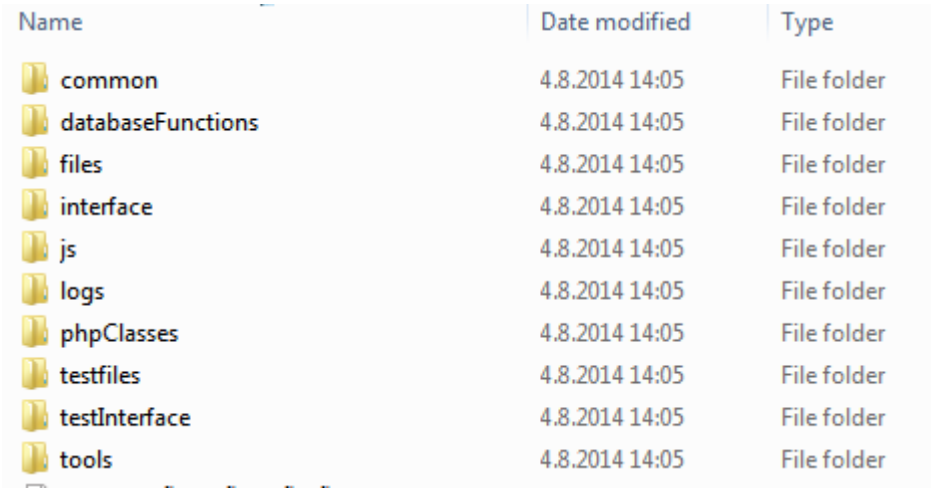

Figure 1. Server folder structure.

The folder databaseFunctions contains procedural functions for single database commands (such as insert, update and fetch). All documentation stored by users would be placed inside the files folder. Js folder contains all Javascript files.

The logs folder contains all logs made by server side scripts. Each user event is logged and all errors are collected. The folder phpClasses contains all programming classes created in this project. It should be noted that most of the code in this project is procedural, and object oriented programming has not been used properly.

Two folders, testfiles and testInterface, contained temporary files used in development and they would be removed from the final product. The tools folder contained scripts that are used as development tools, such as populateDatabase.php.

Listing 7 demonstrates an array conversion function from common functions. This particular function converts associative arrays into numbered arrays. This was used to cycle through arrays using integers.

```
//converts associative array to numbered one
function convertAssocToNum($array){
          $i = 0;
          $numberedArray = array();
          foreach($array as $num){
                    $numberedArray[$i] = $num;
                    $i++;
          }
          return $numberedArray;
}
```

Listing 7. Example function from common functions.

```
include_once("../phpClasses/Log.php");
include_once("../common/dbConnect.php");
include_once("../common/functions.php");
include_once("../common/dbFunctions.php");

if(isset($_POST['fetch'])){
          switch($_POST['fetch']){
                    case "projectList":
                              echo(json_encode(queryProject-
List($DBH)));
                              break;
                    case "project":
                              echo(json_encode(queryPro-
ject($DBH, $_POST['id'])));
                              break;
          }
}
```

Listing 8. Example from fetch.php

Listing 8 demonstrates how simple database functions are. Function checks what the user wants to fetch from the database, calls for a general queryProject() function, changes return value into JSON format and sends information back to the user. Because of time constraints, some of the database functions are completely empty. It is currently impossible to insert data into the database using insert.php. This implementation is also missing data validation. It is possible to inject scripts into server from client side. Further development is needed in order to secure server.

```
/*
 * Closes old entry by closing timestamp and inserting new one. Requires id.
 */
function updateEntry($DBH, $innoDBH, $table, $id, array $columns, $return = false){
          try{
                    $timestamp = getTimeStamp();
                    $fixedColumns = selectWhichColumnsToUpdate($DBH, $innoDBH, $table, $id, $columns);

                    $fixedColumns[1][searchNumArray("valid_starttime", $fixedColumns[0])] = $timestamp; //change timestamp
                    $insertStatement = insertEntry($DBH, $table, $fixedColumns, true);

                    $statement = "
                    UPDATE $table SET
                    valid_endtime = '" . $timestamp . "'
                    WHERE
                    valid_endtime IS NULL AND id=$id;

                    ";
                    if($return){
                              $statement .= updateRelatedJoins($DBH, $innoDBH, $table, $id, $timestamp, true);
                              return $statement;
```

```
                    }else{
                                $STH = $DBH->exec($statement);
                                $STH   =   $DBH->exec($insertState-
ment);
                                $log = new Log();
                                $log->log($statement);
                                $log->log($insertStatement);
                                updateRelatedJoins($DBH,  $innoDBH,
$table, $id, $timestamp);
                                return true;
                    }
          }catch(Exception $e){
                    $log = new Log();
                    $log->error($e);
                    return false;
          }
}
```

Listing 9. How database entries are updated.

Function in Listing 9 would select correct columns to be updated, create SQL queries for an update, update columns, update related join tables and log results into .txt files.

This project also produced multiple Javascript files. The main idea in Javascript was to use AJAX to access database, thus removing wait time from page loads. Jquery library was used to help out webpage manipulation and AJAX queries.

Javascript functions are roughly divided into two categories: database related AJAX queries and webpage manipulation scripts. Webpage scripts would alter information on a webpage on realtime while AJAX queries would work on the background with the server to provide required information to the user.

```
class Filehandler extends Log{

          function __construct(){
                    parent::__construct();   //initialize   error
logging
                    //check if current days folder has been cre-
ated and create one if not
                    if(!file_exists("../files")){
                                        try{

          mkdir("../files");
                                        }catch(Exception $e){
                                                  $this->er-
ror($e);
                                        }
                    }
          }

          //stores file into files folder
          //is tied to documentId
          function  storeFile($documentId,  $fileTempName,  $file-
Name){
                    if(!file_exists("../files/$documentId")){
                                        try{
                                                  if(file_ex-
ists("../files/$documentId/" . $fileName)){
                                                            throw
new Exception("File $fileName already exists");
                                                  }else{

          mkdir("../files/$documentId");
                                                  move_up-
loaded_file($fileTempName, "../files/$documentId/" . $fileName);
                                                  $this->Log("Up-
loaded File: $fileName");
                                                  }
                                        }catch(Exception $e){
```

```
                                                    $this->er-
ror($e);
                                                }
                    }
          }

          //returns link to specified file
          // prolly not needed because js should already have file
directory from db
          function retrieveFile($documentId, $fileName){
                    return("../files/$documentId/$fileName");
          }

}
```

Listing 10. Filehandler class

The Filehandler class in listing 10 is a special class created for this project. It is based on another class called Log, the purpose of which is to write log files into the server. Initializing filehandler causes it to create correct folder for the files.

When storing a file, Filehandler will check if the file has been already been inserted into the system and insert file. The current implementation does not handle file updating. The business logic behind here is that the updated file would be stored under a different document id and the old file would be left intact as a reference.

This system works well until server needs to be scaled up. There is no way to divide files between different hard-drives.

```
function populateDB($DBH, $innoDBH){


          for($i = 0; $i < 100; $i++){
          echo("//<br>");
          $projectId = fetchNextFreeId($DBH, 'project', 'id');
          echo("projectId " . $projectId . "<br>");
```

```
        $projectData = array(
        array("id", "valid_starttime", "identifier", "diary_number", "projectcode", "project_name", "acronym", "focuse_areas", "metropolia_coordinator", "degree_program_related"),
        array($projectId, getTimestamp(), generateRandomWord(), generateRandomWord(), generateRandomWord(), generateRandomWord(), generateRandomWord(), generateRandomWord(), generateRandomWord(), generateRandomWord()),
        array("int", "datetime", "varchar", "varchar", "varchar", "varchar", "varchar", "varchar", "varchar", "varchar")
        );
        insertEntry($DBH, 'project', $projectData);

        for($i = 0; $i < rand(0,10); $i++){
                $contractId = fetchNextFreeId($DBH, 'contract', 'id');
                $contractData = array(
                array("id", "valid_starttime", "lname", "fname", "sign_day"),
                array($contractId, getTimestamp(), generateRandomWord(), generateRandomWord(), generateRandomDate()),
                array("int", "datetime", "varchar", "varchar", "date")
                );
                insertEntry($DBH, 'contract', $contractData);
                joinTables($DBH, $innoDBH, "project", $projectId, "contract", $contractId);
                echo("contractId " . $contractId . "<br>");
        }
```

Listing 11. Part from populated function from tools.

The idea behind the populate database script in listing 11 was to create tool that inserts temporary data into the database in order to test querying. The script creates from 0 to

100 projects into the database, each with 0 to 100 related tables together with join tables. All table rows are filled in with a random sequence of numbers and letters.

A lot of code was produced for this project. Attention had to be given into the backend server code, client code and usability. This resulted in fractured nature of code. For example Javascript code contains quite an amount of temporary solutions and outright mistakes. Some database access scripts are completely unusable (such as insert.php) and most of the code is not as reusable as it could be. Also object orientated programming could have been used more efficiently.

All functions to control data from and to the database are ready. File insertion is the only part which would require refactoring (it is currently in the old implementation where the files would be stored inside database itself).

5 Discussion

5.1 Database Design & Implementation

Database design was problematic in the beginning. There are several reasons for this, such as developer inexperience, changing requirements and setting up development environment. The only prior knowledge about database development came from a course called ‘Database management’. Everything else had to be studied separately. Putting learned concepts into practice also proved problematic in the beginning. Database error messages are hard to debug and finding mistakes can be a daunting task.

In every project, requirements change all the time. One of the mistakes in this project was not to take this into account while designing the database. It is not possible to anticipate every change required completely, but by creating good overall design helps when the customer starts to plan details and changing project goals. It was also problematic to keep every detail in check when requirements started to change. Good visualization would have helped to keep the project focused.

The Development environment was full of beginner mistakes. There was no easy way to rapidly test the implemented design. Because of this, trying out database implementation or just testing ideas was quite slow. Automation of database population solved some of the issues. Also, the development environment lacked proper servers and everything (such as design, coding and server simulation) was done on a single machine.

The most problematic part in design was the temporality aspect. Since all data had to be logged, this meant that the number of rows would increase at a rapid rate. Database scripts were never tested against a huge amount of data. It can be assumed that query times would increase at the same pace with inserted data. Since users were added quite early in the development cycle, the ‘user’ table has wrong temporal implementation.

5.2 Code Design & Implementation

Code design suffered from the same issues as the database. The developer only had some experience doing PHP code, mainly from previous school courses. Object orientated programming was known but it was not utilized in this project properly. This would have made the database access layer easier to implement.

Most of the code components were not created to be reused and were mostly created by quickly prototyping. This backfired each time the customer changed the project specifications. Revising code would have also simplified data queries. Each time data was queried from the server, a different algorithm was used. Maintaining these algorithms to keep up with changing requirements proved tedious.

At first, all client side calls would have called a single PHP file, where a huge amount of ‘if’ code blocks and ‘elseif’ code block happened. At halfway through development, this proved too cumbersome to maintain and the PHP access layer was divided into multiple files. This also simplified the AJAX layer calls (no need for a huge amount of parameters in each call).

After using other programming languages, PHP development feels really slow. This comes from PHP’s need to run on a server. Although Ruby on Rails is also run on a server, normal Ruby code can be run locally for quick testing of programming logic. This would have helped with simple programming questions, like how to go through multiple dimensions of nested arrays.

5.3 Customer Relations

The relationship with the customer group had a great start. Early meetings were useful in getting a grasp of the scope of the project and helped the customer to explain what they wanted. Creation of user stories also helped conveying the customer vision.

The biggest problems with customer relations were caused by time constraints. Lack of energy coupled with increasing demands were a lot to handle. The customer cannot be blamed for this. It is natural that features start to come in after all parties get more acquainted. In retrospect, this could have been avoided with some common sense.

Customer relations became problematic when not enough progress had been made in agreed time and the developer was afraid of confronting the customer group. This resulted in delayed response times and emotional distance between two parties. Eventually all customer relations boiled down to the developer lacking resources to provide what the customer wanted.

5.4 Project management

After working for a software company for several years, it can be said that this particular thesis project did not have any kind of project management. For example the project did not have a proper version control system in place. Different versions of files were handled by copy pasting old files into different parts of the same hard drive. This meant that if the hard drive were to fail during development, there would have been no way to restore files and all work would have been lost. Also doing versioning by hand is tedious work.

The project also lacked a proper task system. All code tasks were only in the head of the developer. This practically meant that new people who would have started working on this project, would have had absolutely no idea what to do next.

6 Conclusions

The main goal of this project was to produce a working database solution for the TKI department at Metropolia. This project did not result in a working product but it laid groundwork if such a project were attempted in the future. This thesis can be used as reference for future database development.

This project produced a complete database with complete tools for the data access layer on the server side, with unfinished implementation for responsive AJAX. Finishing this project would require finishing touches to the backend, completely redoing the frontend and the user experience.

Carrying out a project like this again would require creating new user stories, new database implementation and a data access layer based on this implementation. The development environment should use version control and future development could also be done using Ruby on Rails.

No further action is recommended for this project.

References

[1] Zhao S. Dynamic Website Implementing PHP and MySQL. Espoo: EVTEK University of Applied Sciences; 2003.

[2] Wu H, Zheng W. Building a Website with LAMP. Espoo: Metropolia University of Applied Sciences; 2008.

[3] MacIntyre P, Danchilla B, Gogala M. Pro PHP Programming. Berkeley CA: Springer Ebooks, Apress; 2011.

[4] Gilmore WJ. Beginning PHP and MySQL: From Novice to Professional. Berkeley CA: Springer eBooks, Apress; 2008.

[5] Elmasri R, Navathe SB. Fundamentals Of Database Systems. Addison-Wesley; 2000.

[6] Yurdakul S. Tietoturvallinen Web-Ohjelmointi. Metropolia University of Applied Sciences; 2013.

[7] Eclipse Plugin Documentation [online]. Ottawa, Canada. Eclipse Foundation.
URL: http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Fconcepts%2Fconcepts-25.htm.
Accessed 26 March 2015.

[8] XAMPP [online]. Apache Friends.
URL: https://www.apachefriends.org/index.html?ModPagespeed=noscript.
Accessed 26 March 2015.

[9] Libre Office [online]. Kaufbeuren, Germany. The Document Foundation.
URL: http://www.libreoffice.org/discover/libreoffice/.
Accessed 26 March 2015.

[10] Dropbox [online]. San Francisco, United States. Dropbox, Inc.
URL: https://www.dropbox.com/business/why-dropbox-for-business.
Accessed 26 March 2015.

[11] Apache FAQ [online]. Apache Friends.
URL: https://www.apachefriends.org/faq_windows.html?ModPagespeed=noscript.
Accessed 26 March 2015.

[12] MySQL Reference Manual [online]. Redwood City, United States. Oracle, Inc.
URL: http://dev.mysql.com/doc/refman/5.0/en/char.html.
Accessed 26 March 2015.