LAB University of Applied Sciences

# Using server-side web technologies for rapid development of internal survey tooling

LAB University of Applied Sciences
Bachelor of Business Administration, Business Information Technology
2023
Alexander Sobolev

**Abstract**


| Author(s) | Publication type | Completion year |
|---|---|---|
| Alexander Sobolev | Thesis, UAS | 2023 |
| | Number of pages<br>28 | |


Title of the thesis

**Using server-side web technologies for rapid development of internal survey tooling**

Degree, Field of Study

Bachelor of Business Administration, Business Administration Technology

Name, title and organisation of the client


Abstract

The project aimed to implement an interface for an existing web survey creation system originally developed at LAB. Available professional literature was reviewed to create a holistic understanding of the relevant theoretical concepts. Following that, the target project was implemented, based the author's own experience and founded in the concepts identified during the literature review.

As a result, conclusions were established with regard to viability and process of rapid development of internal tooling using web technologies. The technology in question was Symfony, a versatile PHP framework that allowed to accomplish project goals within the identified constraints.



Keywords

Client-server architecture, Tool development, Web frameworks

Contents

# 1 Introduction

## 1.1 Background

The aim of this thesis project is to develop an improved interface for a survey-making service developed and internally updated by LAB for LUT, used to provide versioned business surveys to third-party companies. A survey interface example can be seen in Image 1.

The feature which sets this service apart from other existing technologies of the same purpose is the support for creating and maintaining derivative versions of the same survey. By maintaining the ability to track responses across several – likely chronological – versions, the system's users obtain a chance to gather information which would otherwise be tedious or practically impossible to collect in an automated fashion.

In its current state, the system has an inadequate interface for creating and updating the registered surveys. This problem has been reported multiple times by its administrators. The primary difficulty lies in the absence of any "user"-oriented interface for creating surveys, which necessitates manual interaction with the service's database in order to create or update a survey's contents. Considering that each survey may contain up to hundreds of individual pieces of information, which must consistently reference parts of itself, the current process of entering survey data makes for a highly tedious, time-consuming, and error-prone activity.

Within this thesis project's scope, the aim is set to provide a better, more user-friendly way of working with the survey system without overhauling its internal structure and/or implementation. The current implementation uses a traditional PHP-based back-end service (both concepts explained further), and thus whatever improvements may be added are required to work well with current technology. Due to that, the topic of this thesis report specifies that the technologies involved in this project are strictly server-side, as opposed to some relatively novel ways of designing services and their interfaces.

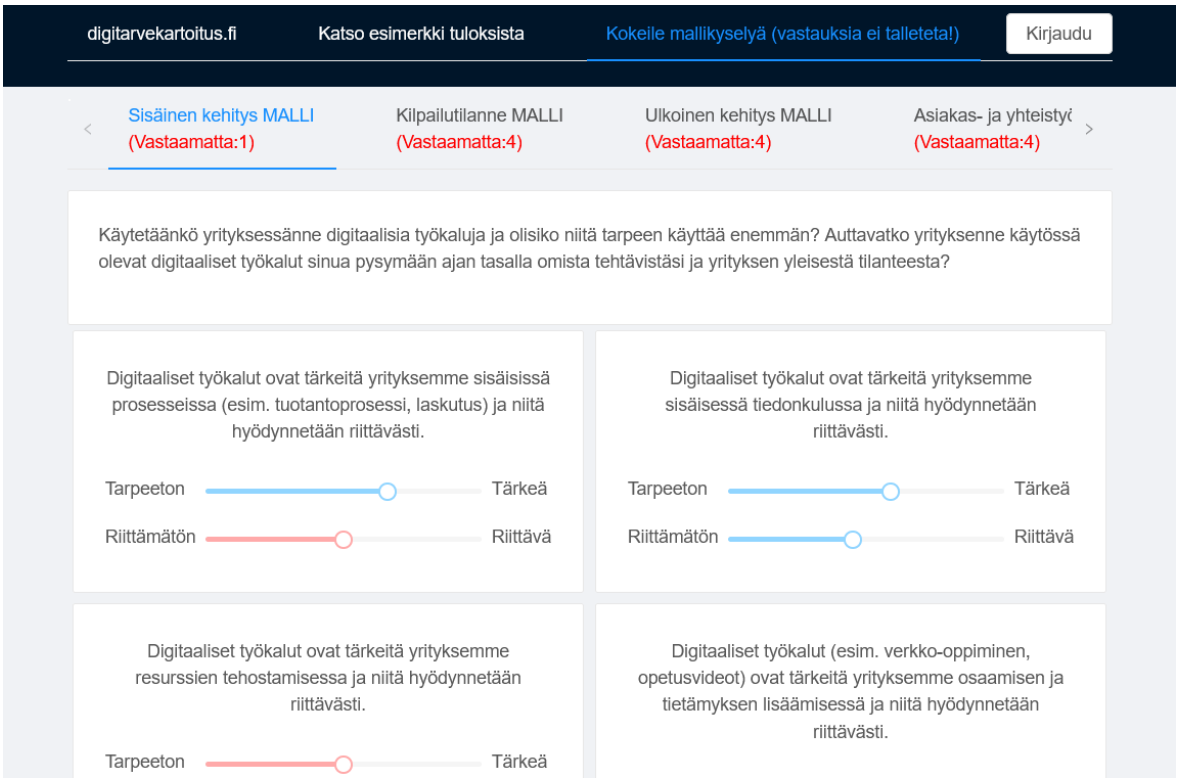


Image 1. Front page of the survey-making service

## 1.2 Research questions

Research question formulated for this thesis is:

***How to rapidly develop internal survey creation tooling with server-side web technologies?***

The overarching question is split into three sub-questions to help formulate usable and valuable answers, and to better guide the writing process of this report. These sub-questions are detailed below:

- ***What technologies are available within the project constraints?***
- ***What are the main advantages of the selected technology?***
- ***What are the difficulties encountered while developing the required service?***

In answering these questions by the end of this thesis, the author hopes to provide an overview, grounded both in theory and practise, of the viability and development process of the target project service. The constraints set out in the next subsection (1.3) narrow the research field even further, which adds to the value of this thesis's output.

## 1.3 Objectives and constraints

The primary objective of the project backing this thesis report is to create a sufficiently user-friendly interface for adding surveys with the pre-existing system. Because the current way it works is highly technical, involving manual editing and input of dozens of parameters via database code, the process is highly error prone. Thus, even a very basic interface can be expected to make a noticeable improvement in terms of the system's administrator's user experience.

For the thesis report itself, the objective is to establish a conclusive answer to the research questions set out in Section 1.2, by applying pre-existing knowledge and analysing the case service implementation.

### Limitations and delimitations

Limitations of a research are the key factors which may cause incorrect interpretation of its results. Regardless of the thoroughness of the researcher, certain background aspects and assertions need to be assumed true without formally satisfying the burden of proof, and research limitations serve as the part which details these assumptions. (Ellis & Levy 2010, 115.)

Delimitations, on the other hand, are set apart from limitations by the nature of their origin. According to Theofanidis and Fountouki (2018, 156-157), the main quality differentiating delimitations from limitations is their being set by the researcher themselves. Specifying them improves the overall work quality by allowing its readers to critically evaluate applicability and limitations of the research done and / or report written.

When it comes to constraints for this project and thesis, the limitations are:

- Adherence to the existing data structure and database layout.
- Time constraints.

The primary delimitation of the background project is focusing on pre-existing technologies already used in the service being developed. While it would have been possible to apply the more novel tools for solving this problem, introducing alternative technologies would significantly increase the project's scope while adding difficulties for its future support and maintenance. In addition, the breadth of programming languages, architectures, and frameworks available would require setting more opinionated delimitations, reducing the overall value of this work.

## 1.4 Approach and methodology

### Approach

There are three primary approaches to doing research: deductive, inductive, and abductive. Illustrating the difference between them is out of scope for this thesis, but a brief comparison is provided below to explain the choices made by the author.

Deductive approach is used to describe a research process where existing knowledge concerning a subject is used to formulate a hypothesis. After a period of collecting and analysing data in relation to the hypothesis, it is then either accepted or rejected. In practice, this means that research done with a deductive approach uses the broader available concepts to come to a more specific conclusion. (Bryman & Bell 2015, 23.)

Inductive approach, on the other hand, leads the researcher from specific observations (usually from prior research) to a more generic conclusion. In other words, in inductive approach, the broader theory is seen as an output, rather than input, of the research. (Bryman & Bell 2015, 25.)

The third approach mentioned above, which is the *abductive*, aims to mitigate weaknesses of deductive and inductive approach. For deductive reasoning, the difficulty lies in selecting a precise hypothesis. For inductive reasoning, the problem is the understanding that there can be no such amount of data collected that would always lead to a truly conclusive theory. Abductive approach serves to overcome these difficulties by looking at a surprising fact and trying to explain or simplify its peculiarities via the “best” available interpretations. (Bryman & Bell 2015, 27.)

Through this thesis, the deductive approach is used. The original research question can be rephrased as a hypothesis, for example: the more traditional server-side technologies are a good way to rapidly build internal tooling. Depending on the answer reached by the end of this work, that hypothesis is either accepted or rejected.

### Methodology

To select the kind of methods to use, this section provides a brief overview of qualitative and quantitative methodology.

Qualitative methodology focuses on studying complex phenomena as a whole. It lends itself well to researching processes, causations, and narratives. Qualitative research methods may also be also applied for studying loosely connected subjects, where identifying trends and key findings takes priority over the purely numerical methods. (Ahmad et al. 2019, 2829.)

On the other hand, quantitative methods are more statistical in nature, using mathematical methods. They are best applied for fields where hard, precise measurements are desired and/or possible. (Ahmad et al. 2019, 2829.)

Empirical nature of this thesis and emphasis on analysing the practical effort required, by means of implementing the case project, lead to a strong preference for qualitative methodology. As a result, it is the kind of methodology used for research in this thesis. Further in this paper, a literature review is done to provide an overview of the fundamental technological concepts required, and that of the exact technologies used while implementing the project.

## 1.5 Thesis structure

The fundamental theoretical foundation of this thesis is detailed in chapter 2. It discusses the concepts which are relevant to the technologies used in the project. Furthermore, the actual technologies used or considered through the project implementation are described in terms of those concepts.

Following that, chapter 3 explains thinking and choices made during the development process. It also details the operational principles of this project's primary software output artifact – the creation interface for the survey system.

Finally, the summary chapter describes the results of the project and this thesis report. It reflects on the author's findings with respect to questions and goals set out in sections 1.2 & 1.3, and considers the limitations of the work done, as well as possible future work. Table 1 presents a more succinct overview of this thesis report's structure.

Table 1. Brief explanation to the chapters of this thesis report

| | |
|---|---|
| **Introduction** | Background, questions, goals, limitations, approach. |
| **Concepts and technologies** | Fundamental aspects, exact technologies used. |
| **Project implementation** | Development process, observations, features. |
| **Summary** | Research question answers, limitations, future work. |

The overarching goal of this thesis report is to overview the developed software project from a broader research perspective. This report is structured to bring the reader from goals and

questions posed in its beginning to the findings reached through reviewing theory and executing the case project. This is done through a flow of discussing several fundamental theoretical concepts, using them to introduce the exact technologies used in the project, and finally utilizing those in a description of the project's implementation process.

# 2 Theoretical and applied concepts

## 2.1 Fundamental concepts

This subsection introduces several theoretical concepts used in web service development, which are required for understanding the technologies used during the thesis project's implementation.

### 2.1.1 Client-server architecture

Client-server architecture, as a concept, is used to define a mode of operation between some local computer, the client, and some other remote computer, the server (Oluwatosin et al. 2014, 67). The latter part, "server" may not necessarily be a server in a conventional understanding of the word, i.e. a big computer in a dedicated room. As long as the computer or the software application has data and functions to share with other computers or software applications, it may be described as a server. This includes, for instance, providing database access, user authentication, some logic dispatch, etc.

An operational model common to software using the client-server architecture is "request-response". True to its name, this model is based around the concept of two directions of communications between the client and the server. Requests are the well-structured messages sent (usually, via network) from the client to the server, i.e., "requesting" some action or a piece of data, whereas responses are the messages the server sends back. (Potapov et al. 2019, 1; Sallow et al. 2020, 7.)

One definitive feature of the "request-response" style of communication is its reactive nature. In other words, no action happens on a server until a client requests it. There are other ways of exchanging information between networked computers, including those where the information is sent without any explicit request for it (Tarkoma 2012, 1). However, while they can be commonly found in modern web software solutions, they are of little interest to this report due to the case project's limitations.

### 2.1.2 Relational databases

The concept of a database is broadly defined in colloquial use, usually meaning anything which may store data. It is worth nothing that from a more technical perspective, an additional constraint is required, since not just every chunk of disorganized data can be understood and otherwise used in a way that would make sense to a program or a software engineer. Harrington (2016, 5) states that, *to be considered a database, the place where data are stored must contain… also information about the relationships between those data*.

In terms of computer databases, the traditional software suites providing database functions (known as DBMS, or database management systems) provided primarily what is known as *relational* databases, especially over the last two decades.

Relational databases are based on the so-called relational model invented by E.F. Codd. His paper, published in June 1970, introduced a model for managing databases in an easy and error-resistant way. (Melton & Simon 1993, 6.) From the applied perspective, relational databases feature several basic, easily recognizable concepts (Jatana et al. 2012, 2):

- Tables, which are strictly defined to describe one and only one category of data.
- Columns, which describe a single characteristic of a data category stored in a table.
- Rows, which are the uniquely comparable entries within a table.
- Relations, commonly expressed via key columns, which uniquely and unambiguously connect several tables.

### 2.1.3 Web service front-end and back-end

When discussing development of web services, they are often separated into the broadly defined segments of “back-end” and “front-end”. Even though finding a formal definition of either proves to be challenging due to these terms’ ubiquitous and colloquial nature, it can be said that “back-end” software means all kinds of applications running on the server, while “front-end” refers to the public-facing software elements (Smith 2013, 138; Shetty et al. 2020, 5734).

In the client / server model, as discussed in section 2.1.1, back-end software is what implements the *server* part and can thus be referred to as *server* software. It ranges from small scripts running in the background to the more complex applications managing web content to database suites, etc.

On the other hand, mapping the notion of front-end software to the client / server model is not as straightforward. In terms of web development, the front-end is *the behaviour, design, content, and structure of everything visible on the screen* (Shetty et al. 2020, 5734). When applying this to a conventional web service model where there is some *server* software running over the network, and a *client*-side interface used through a web browser, the relation between “front-end software” and “client software” is illustrated in Figure 1.

In this figure, the web browser serves as the client software, while the interface available through it constitutes the front-end part of the service in question.

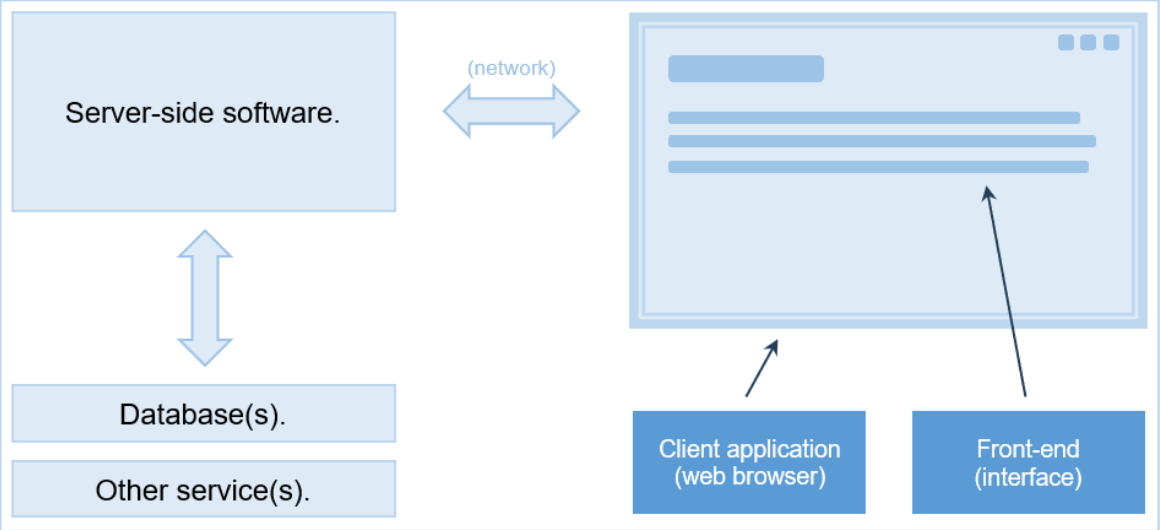


Figure 1. Relation between notion of client software and front-end part of a web service.

### 2.1.4 Web framework

A web framework is a structured, documented software source code provided for other developers to use, which is designed to serve as a foundation for creating custom web applications (Porebski et al. 2011, 2). By abstracting away the common logic prevalent across any sort of web application, while possibly being difficult to implement fully correctly, web application frameworks serve as both the architectural basis and implementation helpers for higher-level end-product services.

The primary benefit which developers get by relying on one web framework or another is the ability to focus on implementing their task at hand. Time is known to be a precious resource, often virtually equating money in business processes, and the time not spent on designing a service's internal architecture or building the more basic functions required, is the time which can be spent on better testing and more features instead. Porebski et al. (2011, 2) identify several key benefits to using a web application framework:

- higher development speed
- minimization of risk of errors in common logic
- promotion of code reusability
- higher default code quality.

For all these advantages, using a web framework is not without disadvantages. By adhering to the overall design implemented by someone else, the developer loses control over the high-level code structure and service architecture. Furthermore, smaller issues such as

conflicting code style (e.g., variable naming, class organization, etc.) may introduce more friction into the development process.

**Distinction from library**

A software library, much like a framework, is an organized collection of source code created and provided by a developer or a company for other to use. Software libraries in this sense of the word tend to be at least partly documented and aim to solve one or several closely related problems. The idea is to implement some functionality once and do it well, and then reuse it in the future to trim development time.

The similarities between a library and a framework are easily seen. However, there is a notable distinction between them, which in essence delimits the applicability of the concept of framework. Porebski et al. (2011, 2) distinguish these two concepts the following way:

- Libraries are akin to attachable parts which an application's code may use in a way more convenient to the developer. Using a library does not require conforming to a vast pre-defined architecture, but rather provides a way to quickly plug in a third-party implementation of some feature.
- Frameworks are the skeleton for an application, which instead calls the developer's own code in a pre-defined way, sensible in terms of the framework's own design.

## 2.2 Concrete technologies

This subsection provides a description of the key technologies considered and selected for the implementation of this project, based on the basic concepts described in the previous subsection. This should give the reader the necessary background to understand the thinking process and decisions made during the development of this thesis project.

### 2.2.1 SQL

Subsection 2.1.2 introduces the notion of relational databases. SQL is a language which serves as the primary of interfacing with most relational database implementations. While many data languages have been created over the years, SQL is the one most strictly and widely standardized, for reasons of being popular in both usage and implementation. (Melton & Simon 1993, 4-5.)

One highly important distinction is that SQL does not fully implement the formal relational model as originally defined by Codd. Rather, it is closely based on it, allowing database administrators and developers certain leeway which the model does not. For instance, while

the formal relational database model prohibits having duplicate rows in a table, it is a common thing to have in modern relational databases. (Melton & Simon 1993, 18.)

As a language, SQL provides a common way of managing database contents and descriptions, and the software systems governing one or more databases themselves. This means that it allows creating (inserting), reading (selecting), updating (altering) and deleting (removing) of database instances, tables, columns, rows, constraints, as well as allowed users, among other things. These four types of operations are colloquially referred to as CRUD, though the historical background of this acronym is not strictly connected to SQL (Chris 2022).

**Namespace and schema**

Two concepts crucial to understanding the way SQL is used to manage data in a database are *namespace* and *schema*.

A namespace is a way to use multiple tables which require having the same name in a single database system installation. This might be necessitated, for example, by deploying several instances of the same application on one database server. To mitigate a naming conflict, SQL allows tables to have a higher-order name, the namespace. Using it, a table of the same name can be referred to unambiguously by prefacing its logical name with its namespace. (Melton & Simon 1993, 25.)

With this in mind, a schema can be defined in terms of the namespace concept. A schema in SQL is a set of named objects (primarily, tables) which belong to the same namespace (Melton & Simon 1993, 25). This way, whenever a database structure required by an application is deployed to some database server, it is said to create its schema – the collection of tables, views, indices, etc. necessary for its functioning.

**Example**

Image 2 shows an example of a table definition using an SQL statement. The piece of code shown is automatically generated from an actual database used for this thesis project. Multiple statements like this make the full database schema required by the developed project.

The SQL statement shown lists columns with their names, types, constraints, and additional modifiers, which altogether make up a table “qpage”. To provide context, in the developed thesis project, this table stores distinct large parts of a survey visually separated into pages (or tabs).

To illustrate the concept of SQL columns, the “pid” column serves as the default, automatically incremented ordinal index for all rows in the table. The column “qid” references another

table's index (namely, the questionnaire to which this page belongs), while "ptitle" is the name of this particular page as presented to the user.

```
create table qpage
(
    pid            int auto_increment
        constraint `PRIMARY`
        primary key,
    ptitle         varchar(4096)                                  not null,
    pseqnum        int                                            null,
    qid            int                                            null,
    pgeneralinline text                                           not null,
    pgeneralfile   varchar(255)                                   not null,
    pscope         enum ('MAINUSER', 'ALL', '', '') default 'ALL' not null,
    constraint pid
        unique (pid, pseqnum),
    constraint qpage_ibfk_1
        foreign key (qid) references questionnaire (qid)
)
    charset = latin1;
```

Image 2. Table definition example using an SQL statement

The rationale for exact table structure, constraints and type selection is not discussed in this thesis because it is inherited from the pre-existing service implementation and is not subject to change, as mentioned in section 1.3.

### 2.2.2 PHP

PHP is a programming language, one of many used for developing server-side applications and services. Originally, it was introduced by Rasmus Lerdorf in 1994 for his personal blog implementation. (Welling & Thomson 2003, 3.)

Since then, it has become one of the most widely used technologies in web development. According to Sklar (2016, 5), over 200 million websites use it, ranging from personal pages to technological giants. Despite numerous alternatives arising over the last decade – most of which may be subjectively seen as superior, more reliable, or more developer-friendly – PHP still maintains a vast presence running the Internet, and continues to receive security and feature updates, as well as fundamental modernizations.

There are several reasons for its rise in the beginning of the century and its continued popularity. Sklar (2016, 4-5), Welling and Thomson (2003, 5), mention some of these reasons:

- It is free, both in monetary and philosophical sense (it is an open-source project).
- It is cross-platform, i.e. can be run in virtually any server environment.
- It is built for web, i.e. with the client/server, request-response model in mind, and thus the developer does not need to "fight" the language to use it meaningfully.
- It has a lot of relevant functionality built-in via expansive set of libraries (for example, for low-level database querying, image manipulation, and others).
- It is highly popular, meaning that there is plenty of learning materials, development tools and third-party software available to anyone using it.
- It is easy to learn, which somewhat contributes to its popularity.

With regard to this thesis, PHP as a language is already used for parts of the background project's implementation. Additionally, the deployment environment used for the web survey service is strongly oriented toward PHP. For this reason, virtually any kind of service or application accessing the surveys' database requires having at least some part to be written in this language.

**Operational principle**

As mentioned above, PHP is historically tailored for server-side web development. This is seen, first and foremost, in the execution process of a PHP script (a piece of source code).

First, a usually external server receives a client's request and forwards it to PHP. Second, the PHP reads and executes the requested script. Third, that script build a usually textual output and sends it back to the server. Fourth, the server responds to the client. (Sklar 2016, 3.)

In other words, PHP is designed to execute its scripts in a short-lived manner: every time a request is received, the executing engine spins up, processes the script and exits. Implementations may store and cache some internal execution-related data between running the scripts, but it is not exposed to application programmer. This approach, in the author's experience, has one major advantage and one major disadvantage.

On one hand, the short-lived nature of PHP scripts removes the concerns of managing complex application state in a reliable and secure fashion. Since all data which needs to be stored between different script invocations is explicitly required to be persisted elsewhere – for instance, in a relational database – the complexity of ensuring a consistent and error-free access to that data is pushed onto the developers of third-party database or cache

implementations. As a result, the programmer working on a web service only needs to concern themselves with their application's own logic, which frees up time and allows to redistribute effort to their area of competence.

On the other hand, the same short-lived nature presents a difficulty when managing some small state which needs to be present. For example, a client-server session identifier, or an intermediate calculation. Even if it only needs to be done in one place across the entire service code base, the developer working with PHP will need to persist it using another service – as opposed to simply using a variable in most other languages. This makes PHP ill-suited for developing services which exchange data with clients continuously, such as a large file uploading progress, or some external event notifications.

To summarize, the PHP's major strength and weakness are two sides of the same feature. Depending on the technical requirements for a developed service, this either makes it a fully suitable tool for the job or necessitates finding an alternative.

**Example**

Image 3 demonstrates an example of PHP code. It shows an abridged part of the thesis project's real code. While many of the applied concepts necessary for understanding its exact purpose are introduced later, in the section 2.2.3, this code shows several core features of modern PHP. Those include: a traditional (C-like) syntax, classes and methods corresponding to the object-oriented programming paradigm (OOP), code namespaces, and importing.

```php
<?php
namespace App\Controller;

use App\Service\LLDatabaseAccessor;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController
{
    public function __construct(private LLDatabaseAccessor $accessor) {
        // ...
    }

    #[Route('/v1', name: 'default_index')]
    public function index(): Response
    {
        $buckets = $this->accessor->listQuestionnaireBuckets();
        return $this->render('index.html.twig', [
            'buckets' => $buckets,
            'buckets_json' => json_encode($buckets, JSON_PRETTY_PRINT),
        ]);
    }
}
```

Image 3. Example PHP code

### 2.2.3 Symfony

Symfony is an open-source back-end web framework, originally developed in 2005 by a company called Sensio Labs. Since then, it has undergone multiple iterations – currently being at version 6.3 – and has become one of biggest PHP frameworks available. Nowadays, it powers several other open-source platforms, including Prestashop (an e-commerce suite), Drupal (a content management system), as well as many smaller open-source and in-house projects. (Porebski et al. 2011, 8; Symfony c.)

The Symfony project is built as a well-interoperating set of loosely coupled components, usable both as a part of the overarching framework, and individually as libraries (Symfony c). As a result, its architecture mitigates some of the problems and difficulties inherent to application frameworks, as discussed in section 2.1.4.

However, experience suggests that embracing the full Symfony ecosystem is still beneficial due to being able to use its many features within their native environment, leading to higher productivity and lower number of programmer errors. That being said, it is a complicated and time-consuming endeavour due to the scale and variety of functions implemented by Symfony components. These functions include, but are not limited to, interface translation (localization) system, error handlers, configuration loaders, email helpers, caching implementations, and database accessors. As of writing this paper, there are 69 standalone components available for the developers to use. (Symfony b.)

**Database interaction**

Computer applications in general tend to process data. For web applications, the data in question may vary from blog posts and user credentials to message feeds and notifications. That data is usually stored in a database, which is why it is crucial for a web framework to provide ways of accessing one or more databases.

Because Symfony is a highly modular framework, as described previously, there is nothing preventing a developer from designing their own way of interacting with any database available. As long as their application’s server can reach the database server, it is possible to implement an interaction layer on top of Symfony which would be able to use it. However, it is impractical to do so, given that Symfony is a framework which already has an idiomatic way of working with databases: Doctrine.

Doctrine is a set of open-source libraries which includes two ways of accessing and modifying database contents: Doctrine DBAL and Doctrine ORM. While a thorough analysis of

these two options is out of scope for this paper, a quick overview of the difference between them is provided below.

**Doctrine DBAL** stands for Database Abstraction Layer. It is a library which provides abstractions for communicating with relational databases, such as MySQL. Its interface is verbose and requires direct writing of SQL statements in order to retrieve or modify data. (Symfony a.)

An example PHP code using Doctrine DBAL is shown in Image 4. In this example, the code prepares (creates) an SQL statement for inserting a record into the “questionnaire” table Further instructions associate the inserted questionnaire parameters with the statement, and finally it is executed to perform the insertion.

```
216   private function insertQuestionnaire(array $package): int {
217       $questionStatement = $this->connection->prepare(
218           "INSERT INTO questionnaire (predecessor, description, root, active)"
219           . " VALUES (:predecessor, :description, :root, :active)"
220       );
221
222       $questionStatement->bindValue('root', $package['root']);
223       $questionStatement->bindValue('predecessor', $package['predecessor']);
224       $questionStatement->bindValue('active', $package['active']);
225       $questionStatement->bindValue('description', $package['description']);
226
227       $questionStatement->executeStatement();
228       return $this->getLastInsertId();
229   }
```

Image 4. A PHP function using Doctrine DBAL

**Doctrine ORM** on the other hand stands for Object Relational Mapper. The ORM as a concept means an additional abstraction layer which automatically maps classes and objects (as in Object Oriented Programming paradigm) to database tables and records (Chen et al. 2016, 1). In Doctrine, the ORM is implemented on top of DBAL (Symfony a). This means that Doctrine ORM allows the developer to avoid direct implementation of SQL code, making the overall project code more resilient to errors and easier to update.

# 3 Project implementation

## 3.1 Implementation architecture

This section provides an overview of the thinking process and reasoning of the thesis author throughout the project implementation, with an emphasis on the decisions made during the early stages of development. While in some parts the reasoning draws from personal experience acquired during prior studies, employment, and personal side-projects, the principial decisions made here reflect the author's understanding of theoretical concepts discussed and summarized in the previous chapter.

**Initial considerations**

The initial task for developing a better survey creation interface did not mandate any single form of interface. Anything ranging from a desktop application with a graphical user interface, to a command-line utility, to a dedicated web service would have suited the project parameters.

However, in the beginning of the development process, several practical constraints became apparent in early research. Firstly, regardless of the interface form, it needed to be able to get data in and out of the existing service's database. Given that the current deployment environment all but required PHP, the options for possible implementations were narrowed down. If the interface were primarily built as a client-side application, be it a quick command-line script or an extensive desktop application, it would still require a server component written in PHP to communicate with the database. In particular, with a MySQL database.

Furthermore, the time limitation was accounted for. Developing a fully interactive interface for creating highly interconnected content, such as surveys in this case, was estimated to be a very time-demanding path forward. Personal experience suggested that implementing the required controls, testing, and mapping data from whichever front-end framework I could choose to the underlying database structure would have taken over a month of tedious work. Though not technically complex, this would have pushed the project beyond the available timeframe limits.

Thus, a more efficient implementation was required. During the discussions with the thesis supervisor, two alternative interfaces were suggested:

- generating SQL statements from given survey parameters
- survey import / export via some kind of markup language.

The immediately apparent problem with the first option was the little impact on the overall usability of the system it would have made. If the automation were to be limited to generating raw SQL, creating new queries would indeed be easier than the current state. However, the system would still require surveys to be entered by an administrator with good knowledge of SQL, and relational database operation principles in general. While an improvement, it was considered to be a minor one.

The second option looked promising because it could be implemented as a fully functioning solution. Despite not being user-friendly and still requiring some technical acumen, automated import / export of records via some data markup language, such as XML or JSON, can be documented well enough to be used by stakeholders without any hard skill in the field of Information Technology. This made for a significant improvement over the current state of the service, and over the first option as well. Moreover, if the import / export were to be implemented as a part of a web service, the resulting solution would be usable without any direct interaction with the database, which was assumed to make things simpler even for people who are familiar with database operations.

Due to having prior experience with developing back-end web services in PHP and processing structured markup formats for data export, the work on this project proceeded with the second considered option.

**Technology selection**

Within the project constrains, the primary question in terms selecting an appropriate technology was whether it would be done from ground up, or by utilizing a web framework. With the strengths and weaknesses of web frameworks in mind, it was decided to use a framework.

The conceptual web framework advantages as described in section 2.1.4 allowed the project to be implemented within the identified limitations. Having experience with Symfony, the author selected it as the primary technological foundation for the service. Moreover, its modularity allowed for a quicker and more straightforward implementation of the database interactions, while benefitting from the established software architecture, as well as various tools and helpers for building parts of the service.

Symfony’s flexibility in terms of database communication was a major consideration which reinforced the decision to use this framework instead of others, such as Laravel. As discussed in section 2.2.3, Symfony’s idiomatic way of working with databases is the Doctrine library, which exposes both a lower-level abstraction layer (DBAL) and a higher-level object relation mapper (ORM).

Both options for communicating with the database were considered. After building a small experimental project to try these options, some advantages and disadvantages were identified.

On one hand, ORM was beneficial in multiple ways:

- It provided automatic SQL code generation, making the code more resilient to programmer errors.
- It allowed for easier extensibility: instead of manually rewriting SQL in the event of updates to the database schema, the developers would be able to adjust the self-documenting record class definitions.
- It supported better integration with the rest of Symfony framework: while it was possible to use other database utilities without significant complications, all the infrastructure built up around the Doctrine ORM remained unavailable. For example, this led to additional complications when creating the form for uploading XML files.

On the other hand, DBAL was beneficial in the context of the project's constraints:

- It worked better with the externally defined database schema. On a slightly broader note, ORMs – and Doctrine ORM in particular – shine when they are allowed full control over the database structure and management. This includes creation and modification of tables, entity referencing, and other processes. However, the developed service had to work with a database schema established and managed externally. Wrangling ORM into the pre-defined schema was estimated to be possible, but unnecessary complicated and brittle in case of future database structure updates.
- It made no assumptions about how the relations between various entities (tables) were to be implemented. One of the main aspects of the complexity that ORM aims to alleviate is managing the relations between tables, which it does well when in full control over the schema. However, when it is not, the assumptions made by ORM to hide this complexity start working against the developer, which was discovered early into the initial practical experimentation.

By carefully considering the identified strengths and weaknesses of DBAL and ORM in relation to the project's constraints, the DBAL was selected as the database communication layer of choice.

**Import and export logic**

The logic at the core of the developed service's implementation is conversion of survey definitions between SQL and XML. The data can be accessed within the database using

SQL and can be imported and exported via XML. Bridging these formats constituted the main part of the entire development process.

The most straightforward, but also the naïve way in which this conversion could be done was intermingling XML parsing / building code with DBAL code. This approach fulfilled the time constraint but presented significant difficulties when debugging the work-in-progress implementation. Moreover, it was assumed that the service will receive some maintenance and updates even after the thesis work is done, and this approach resulted in a large volume of poorly maintainable code.

To address these issues, the logic was split into two equally important parts: database accessor class and schema converter class. Both parts were implemented as PHP classes, exposed by Symfony for the rest of the codebase.

In its current implementation, the database accessor class is only responsible for getting survey data from and into the database. It heavily relies on Doctrine DBAL to accomplish that.

The schema converter class is only responsible for parsing and building survey XML for import and export. This is done via PHP's built-in SimpleXML library, which exposes an object-oriented interface for structured access and modification of XML files.

Both classes needed to exchange survey data with one another, which required a common way of structuring survey data in PHP process memory. PHP's native associate array type was used for this purpose since a nested array was found to be able to unambiguously express the entirety of a survey definition. Images 5 and 6 demonstrate parts of an example layout of such an array.

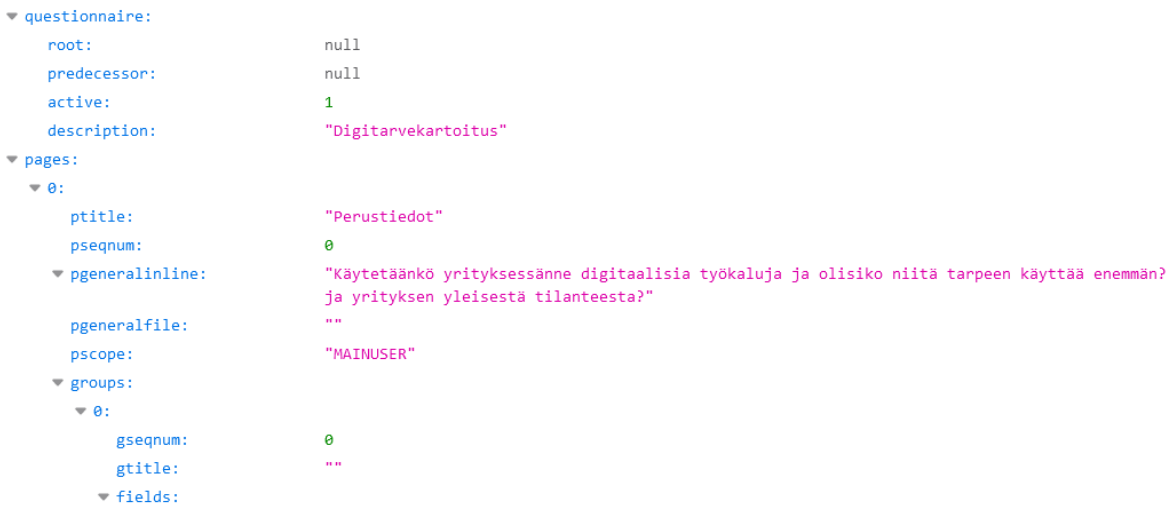


Image 5. Top level of a questionnaire array

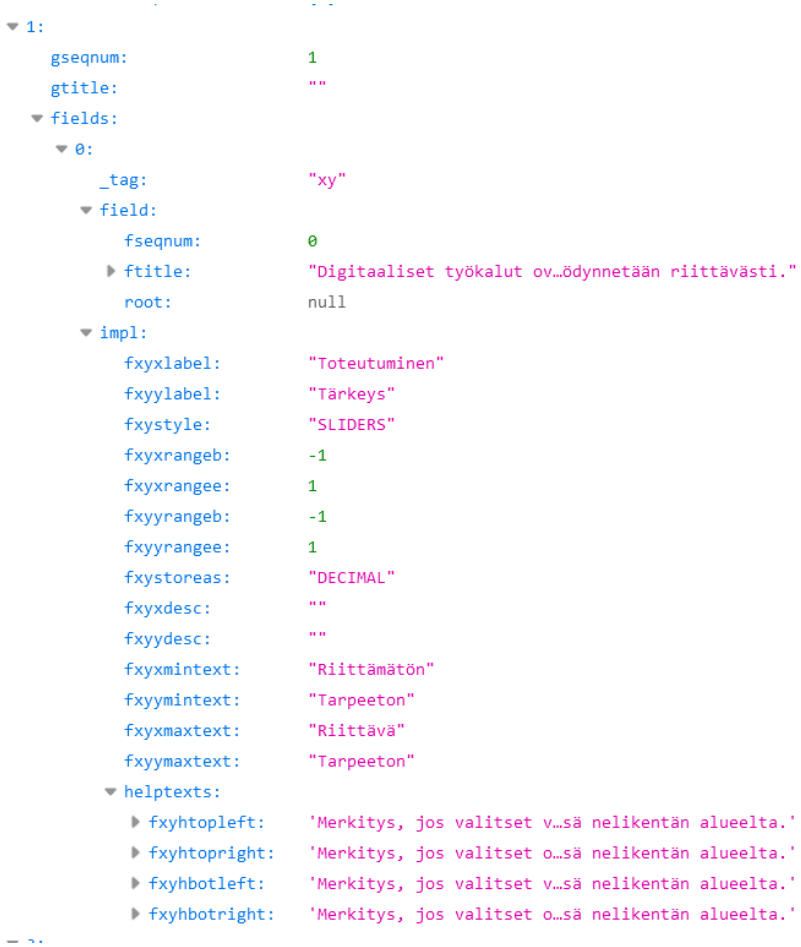


Image 6. Group of survey questions in a questionnaire array

The most problematic aspect of the current way of implementation is connected to this intermediate array representation. Because it is read and written in two separate classes, any change to the database schema, and thus the array, requires adjusting code in both classes. However, it is seen by the author of this thesis as an acceptable trade-off for decoupling SQL and XML manipulation logic.

## 3.2 Description of the developed service

The resulting interface as implemented throughout this project consists of a web service running fully server-side, which has two pages and two more endpoints for downloading survey XML data.

Image 7 demonstrates the first page, which is the survey index. This page lists the surveys registered in the system, grouped by their root, which is the earliest connected version of the survey present. Root surveys are highlighted in light grey for the users' convenience.

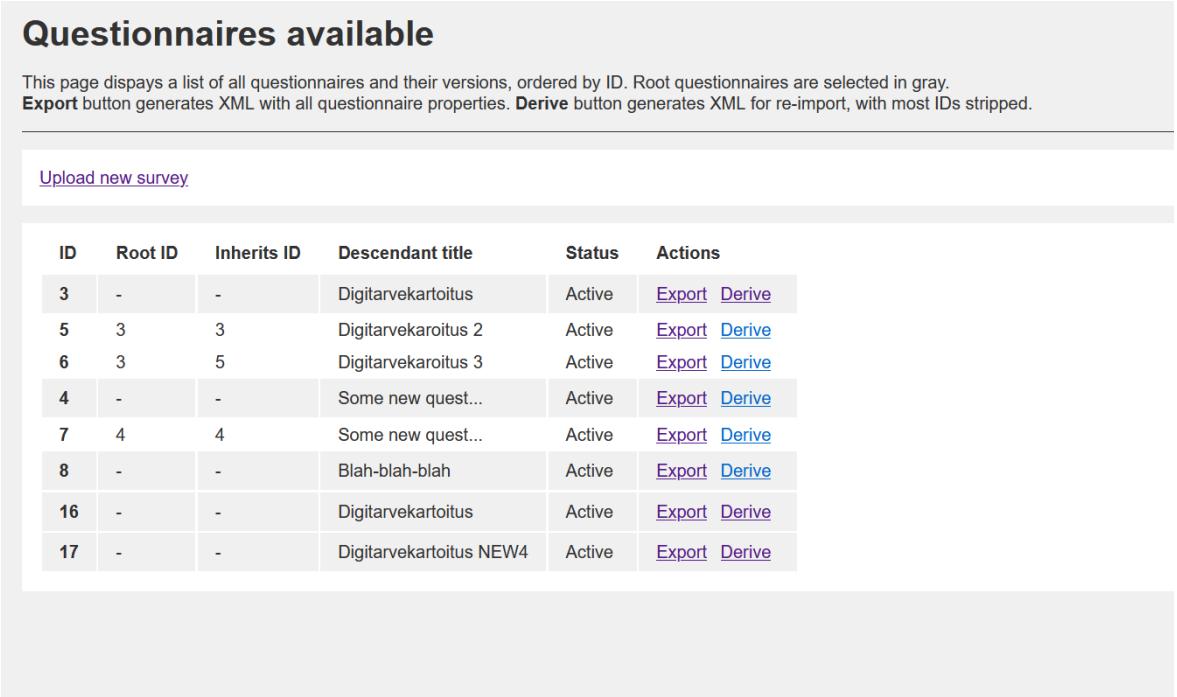


Image 7. Index page screenshot with sample data

There are two buttons in the right-most column of the index table, present for each survey. Both generate XML files describing their respective survey and open it in another tab, but their functions are different:

- **Export** button produces the reference XML, which includes all record IDs and sequencing data. This is highly useful for getting an overview of the survey, as well as for referencing the exact IDs in newer versions of the survey.
- **Derive** button produces the derivative XML, which strips the IDs. This form of exported XML is more convenient for modifying surveys, since it excludes the data which is not even considered by the importer implementation. This is designed to help avoid confusion when creating newer survey versions.

Image 8 shows the import page, used for uploading survey XMLs into the system. The selector in the middle part of this screenshot serves as the double-check for the user. If they want to upload a new questionnaire, the imported XML must not reference any roots or

predecessors. On the other hand, if they are uploading a new version of an existing questionnaire, their XML must describe this relation by means of dedicated XML attributes. When either of these constraints is not met, the page aborts processing and displays the errors, as may be seen on the screenshot.

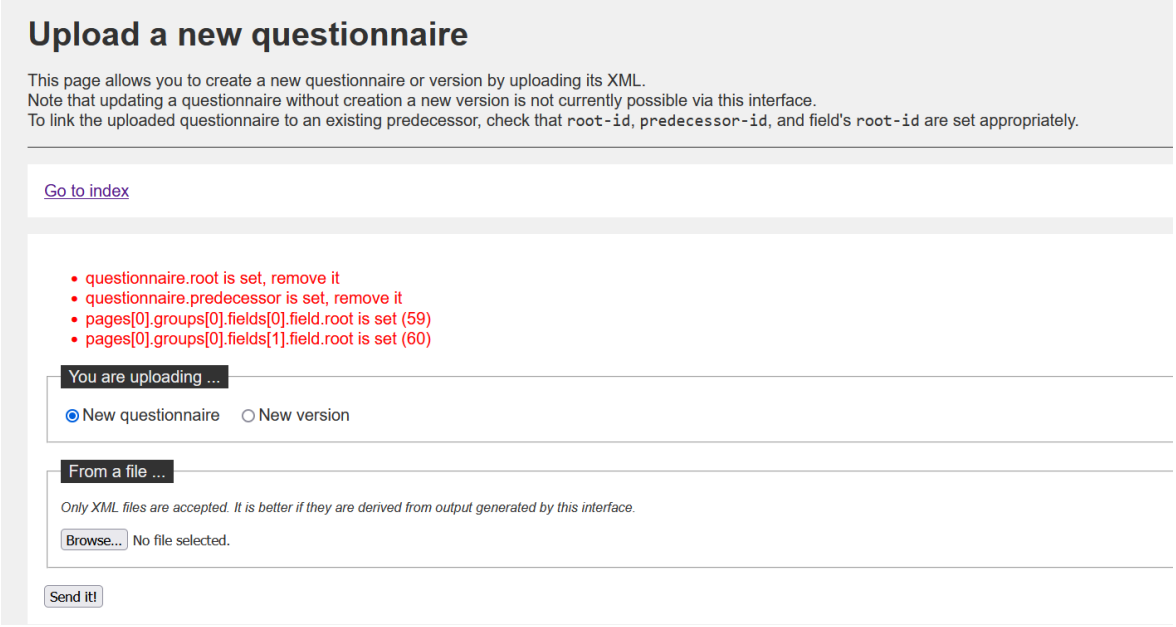


Image 8. Import page screenshot with processing errors

Image 9 below demonstrates a part of an example XML schema that the system generates or anticipates during import. A more formal definition for its structure was considered, but disregarded due to time limitation.

```
<?xml version="1.0"?>
<questionnaire is-active="1" root-id="19" predecessor-id="17">
    <description><![CDATA[Digitarvekartoitus NEW4.1]]></description>
    <pages>
        <page scope="MAINUSER">
            <title><![CDATA[Perustiedot]]></title>
            <general-inline><![CDATA[Käytetäänkö yrityksessänne digitaalisia työkaluja ja olisiko
            <general-file />
            <groups>
                <group>
                    <title />
                    <fields>
                        <field root-id="59">
                            <title><![CDATA[Työntekijöiden määrä 2]]></title>
                            <text store="INTEGER_POS">
                                <label />
                            </text>
                        </field>
                        <field root-id="60">
                            <title><![CDATA[Yrityksen ikä (vuosina) 3]]></title>
                            <text store="INTEGER_POS">
                                <label />
                            </text>
                        </field>
                        <field root-id="">
                            <title><![CDATA[Yrityksemme toiminnan pääpaino on]]></title>
                            <choice>
                                <choice presentation="HOVERDROPDOWN" subseqnum="0">
                                    <label><![CDATA[Tuotannossa]]></label>
                                </choice>
                                <choice presentation="HOVERDROPDOWN" subseqnum="0">
                                    <label><![CDATA[Palveluissa]]></label>
                                </choice>
                            </choice>
                        </field>
                        <field root-id="">
                            <title><![CDATA[Yrityksemme asiakkaita ovat enimmäkseen]]></title>
                            <choice>
                                <choice presentation="HOVERDROPDOWN" subseqnum="0">
                                    <label><![CDATA[Toiset yritykset]]></label>
                                </choice>
                                <choice presentation="HOVERDROPDOWN" subseqnum="0">
                                    <label><![CDATA[Kuluttajat]]></label>
```

Image 9. Example XML code showing the import schema

# 4 Summary

## 4.1 Research results

The main goal of this thesis is two-fold. First, to implement a better survey creation interface for the target project within the recognized constraints. Second, to answer the research questions specified in section 1.2.

In terms of the actual project implementation, the goal has been achieved: there is a working service which allows the system administrator to create new surveys and new versions of existing surveys. It has been implemented within the specified limitations and delimitations, including those of time limit and technological selection.

As for the research questions, the literature review and project work done in this thesis allows the author to formulate answers to them.

***How to rapidly develop internal survey creation tooling with server-side web technologies?***

It is possible to rapidly develop internal survey creation tooling with server-side web technologies by considering the available technologies and selecting ones which work best within the project's constraints.

- ***What technologies are available within the project constraints?***

In this case, the technologies in question proved to be PHP language, Symfony framework, Doctrine DBAL library, and XML files for data import/export.

- ***What are the main advantages of the selected technology?***

Speed of development, flexibility, modularity, and reliability.

- ***What are the difficulties encountered while developing the required service?***

Working outside of the eco-system for which the used tools were originally developed created difficulties, namely in wrangling the Symfony framework to work with the established data structure. This resulted in a bigger volume of development work and certain logical duplication, though was ultimately considered to be a viable trade-off.

## 4.2 Validity and reliability

The results discussed in the section above have been accomplished by means of reading and analysing the available technical literature on various fundamental theoretical concepts, as well as practical technologies. With a small exception, the sources selected for review

consists of professional literature and academic papers, including those submitted for well-known professional conferences. This mutually reinforces the author's personal experience gained through prior studies, employment, and multiple personal side-projects.

Additionally, the author has used many of the technologies reviewed in this thesis in work and other projects, and the results reached by the end of this thesis match the personal experience gained through that.

## 4.3 Future work

The implemented project is functional but can be improved in many ways. There is a breadth of potential suggestions available to enhance the developed service, listed in the order of estimated difficulty:

- Make the current web interface more user-friendly, especially its accessibility.
- Formalize the import / export XML schema and use a more thorough validation.
- Develop a full survey editing front-end interface.

## References

Ahmad, S., Wasim, S., Irfan, S., Gogoi, S, Srivastava, A., Fahreen, Z. 2019. Qualitative v/s. Quantitative Research – A Summarized Review. Journal of evidence-based medicine and healthcare, Vol. 6 (13), 2829. Retrieved on 15 November 2023. Available at DOI 10.18410/jebmh/2019/587

Bryman, A. & Bell, E. 2015. Business research methods. Fourth Edition. Oxford: Oxford University Press.

Chen, T., Shang, W., Yang, J., Hassan, A. E., Godfrey, M. W. 2016. An empirical study on the practice of maintaining object-relational mapping code in Java systems. Proceedings of the 13th International Conference on Mining Software Repositories. 165-176. Retrieved on 29 November 2023. Available at: https://dl.acm.org/doi/10.1145/2901739.2901758

Chris, K. 2022. CRUD Operations – What is CRUD? Retrieved on 29 November 2023. Available at https://www.freecodecamp.org/news/crud-operations-explained/

Ellis, T. J. & Levy, Y. 2010. A guide for novice researchers: design and development research methods. Proceedings of Informing Science & IT Education Conference (InSITE) 2010, Vol. 10, 107-118. Retrieved on 28 November 2023. Available at DOI 10.28945/1237

Harrington, J. L. 2016. Relational database design and implementation. Fourth Edition. Cambridge: Morgan Kaufmann.

Jatana, N., Piru, S., Ahuja, M., Kathuria, I., Gosain, D. 2012. A survey and comparison of relational and non-relational database. International Journal of Engineering Research & Technology, Vol. 1 (6), 1-5. Retrieved on 18 November 2023. Available at https://www.ijert.org/a-survey-and-comparison-of-relational-and-non-relational-database

Melton, J. & Simon, A. R. 1993. Understanding the new SQL: a complete guide. San Fransisco: Morgan Kaufmann Publishers, Inc.

Oluwatosin, H. S. 2014. Client-server model. IOSR Journal of Computer Engineering, Vol. 16 (1), 67-71. Retrieved on 18 November 2023. Available at DOI 10.9790/0661-16195771

Porebski, B., Przystalski, K., Nowak, L. 2011. Building PHP applications with Symfony, CakePHP, and Zend Framework. Indianapolis: Wiley Publishing, Inc.

Potapov, V. I., Shafeeva, O. P., Gritsay, A. S., Makarov, V. V., Kuznetsova, O. P., Kondratuokva, L. K. 2019. Reliability in the model of an information system with client-server architecture. Journal of Physics: Conference Series, Vol. 1260 (2), 1-5. Retrieved on 18 November 2023. Available at DOI 10.1088/1742-6596/1260/2/022007

Sallow, A. B., Dino, H. I., Ageed, Z. S., Mahmood, M. R., Abdulrazaq, M. B. 2020. Client/Server remote control administration system: design and implementation. International Journal of Multidisciplinary Research and Publications, Vol. 3 (2), 5-11. Retrieved on 18 November 2023. Available at http://ijmrap.com/wp-content/uploads/2020/07/IJMRAP-V3N1P95Y20.pdf

Shetty, J., Dash, D., Joish, A. K., C, G. 2020. Review paper on web frameworks, databases and web stacks. International Research Journal of Engineering and Technology, Vol. 7 (4), 5734-5738. Retrieved on 18 November 2023. Available at https://www.irjet.net/archives/V7/i4/IRJET-V7I41078.pdf

Sklar, D. 2016. Learning PHP. First Edition. Sebastopol: O'Reilly Media, Inc.

Smith, P. G. 2013. Professional website performance: optimizing the front end and the back end. Indianapolis: John Wiley & Sons, Inc.

Symfony a. How to use Doctrine DBAL. Retrieved on 29 November 2023. Available at https://symfony.com/doc/current/doctrine/dbal.html

Symfony b. Symfony Components. Retrieved on 29 November 2023. Available at https://symfony.com/components

Symfony c. What is Symfony. Retrieved on 29 November 2023. Available at https://symfony.com/what-is-symfony

Tarkoma, S., 2012. Publish / subscribe systems: design and principles. Chichester: John Wiley & Sons Ltd.

Theofanidis, D. & Fountouki, A. 2018. Limitations and delimitations in the research process. Perioperative Nursing (GORNA), Vol. 7 (3), 155-163. Retrieved on 17 November 2023. Available at DOI 10.5281/zenodo.2552022

Welling, L. & Thomson, L. 2003. PHP and MySQL web development. Second Edition. Indianapolis: Sams Publishing.