jamk.fi

Building a custom timesheet service using LAMP stack

Carita Tammelin

Bachelor’s thesis
May 2020
School of Technology
Information and Communications Technology
Software Engineering

jamk.fi

Description

Author(s)	Type of publication	Date
Tammelin, Carita	Bachelor's thesis	May 2020
		Language of publication: English
	Number of pages 50	Permission for web publication: Yes

Title of publication
Building a timesheet service using LAMP stack

Degree programme
Software Engineering

Supervisor(s)
Rantala Ari, Mieskolainen Matti

Assigned by
EV-metalli Ky

Abstract

The task the thesis addresses was assigned by EV-metalli Ky to improve the bookkeeping of the employees' working hours, and better track the progress of work tasks. The research was made to find out if a timesheet service could be developed using a stack of Ubuntu, Apache, MySQL, and PHP. A virtual system was utilized to replace the use of paper. The objectives of the service were to add work data into the database via web forms, to connect the work tasks with the working hours spent on them and to track whether the task in question was performed. The web forms were to handle string, integer, float, date, Boolean, and file data.

The end goal of the thesis was to develop a minimum viable product that would perform the most essential functions. The main goal was the data insertion, being able to access the data through web pages and showing data in connection to each other. The Ubuntu operating system served as the server machine and memory. Apache HTTP server transferred data from Ubuntu to a web browser. The MySQL database server stored the acquired data. MySQL also provided a base for table relationships to connect different data tables. PHP and Laravel were used to store data using the web forms as well as to find data and present it on a web page. It was necessary for the service to be able to differentiate data based on the user. Timesheet data entered through a web form would be connected only to the current user. Laravel provided a built-in authentication system to handle user data.

The primary goals of the thesis assignment were achieved. However, it was not yet enough for the service being deployed; yet, with the research made during the development of the minimum viable product, the needed improvements were considered possible in the future. It proved that using the LAMP stack and Laravel collecting data via web forms, adding relations to them, and storing them into database was possible and suitable for such use.

Keywords/tags (subjects)
Ubuntu, Apache, MySQL, PHP, Web development, Timesheet, Laravel, LAMP

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Tammelin, Carita	Opinnäytetyö, AMK	Toukokuu 2020
		Julkaisun kieli Englanti
	Sivumäärä 50	Verkkojulkaisulupa myönnetty: Kyllä

Työn nimi

Tuntikirjausjärjestelmän kehittäminen käyttäen LAMP-teknologioita

Tutkinto-ohjelma

Ohjelmistotekniikan tutkinto-ohjelma

Työn ohjaaja(t)

Rantala Ari, Mieskolainen Matti

Toimeksiantaja(t)

EV-metalli KY

Tiivistelmä

Opinnäytetyön toimeksianto oli kehittää tuntikirjausjärjestelmä pienyritykselle. Työn tavoitteena oli kehittää yrityksen tuntikirjausta, sekä parantaa työtehtävien seurantaa. Tutkimuksessa selvitettiin, onko mahdollista kehittää tuntikirjausjärjestelmä käyttäen Ubuntu-, Apache-, MySQL- ja PHP-teknologioita, siirtäen järjestelmän paperilta web-sivuille. Palvelun tavoitteena oli mahdollistaa työn tietojen siirtämisen tietokantaan käyttäen web-lomakkeita. Sen tuli myös yhdistää työtehtävät tehtyihin työtunteihin, sekä seurata työtehtävien tilaa. Lomakkeiden oli käsiteltävä sujuvasti erilaisia datatyyppejä.

Opinnäytetyön lopputavoite oli kehittää pienin toimiva tuote, joka suorittaisi kaikista tärkeimmät toiminnot. Päätavoite oli tallentaa tietoa ja päästä siihen käsiksi web-sivujen kautta. Tärkeää oli myös näyttää tietojen suhde toisiinsa. Ubuntu-käyttöjärjestelmä toimi palvelimen alustana ja muistina. Apache HTTP-palvelin siirsi tietoa käyttöjärjestelmän ja web-selaimen välillä. MySQL-relaatiotietokantapalvelin tallensi saadun tiedon tauluihin. Se myös tarjosi alustan, jolla tietokantataulujen suhteet määriteltiin. PHP ja Laravel mahdollistivat tietokantakyselyt web-sivuilla. Niiden avulla kerättiin, etsittiin ja esitettiin dataa selaimessa. Järjestelmän oli myös tärkeää erottaa käyttäjät toisistaan. Tämä toteutettiin Laravelin sisäänrakennetun todennusjärjestelmän kautta.

Opinnäytetyön päätavoitteet saavutettiin, mutta se ei riittänyt käyttöönotettavaksi järjestelmäksi. Tehdyn työn avulla tarvittavat parannukset kyseisellä järjestelmällä todettiin mahdolliseksi. Työllä todistettiin, että käyttäen LAMP-teknologioita ja Laravelia on mahdollista kerätä monenlaista dataa web-lomakkeilla. Mahdollista on myös yhdistää dataa ja tuoda sitä esille.

Avainsanat (asiasanat)

Ubuntu, Apache, MySQL, PHP, Web-ohjelmointi, tuntikirjausjärjestelmä, Laravel, LAMP

Muut tiedot

Contents

Figures

Terminology

Artisan	A command-line interface in Laravel.
Atom	An open source text editor.
Bootstrap	A frontend HTML, CSS, and JavaScript library.
CSRF	Cross-site request forgery.
CSS	Cascading Style Sheets. A programming language organizing web elements on the page.
DigitalOcean	A cloud infrastructure provider.
HTTP	HyperText Transfer Protocol.
HTML	HyperText Markup Language.
JavaScript (JS)	A programming language used in web development.
LAMP	Linux, Apache, MySQL and PHP.
Linux	An open source operating system.
Open source software	Software that has source code available to anyone. Free use is under different licenses.
PuTTy	A free SSH and telnet client.
PuTTygen	An SSH key generator tool.

RSA	Rivest-Shamir-Adleman. An encryption algorithm.
SQL	Structured Query Language.
SSH	Secure Shell. Encryption protocol for secure communication.
UFW	Uncomplicated Firewall.
WinSCP	Windows Secure Copy. An open source SFTP client for secure file transfer.

1 Introduction

1.1 Background

As technology has evolved massively in the last decade, there are multiple different solutions and opportunities in problem-solving. However, the vastness of options can also be overwhelming and sometimes offer too much in one package. In that case, instead of making matters easier like they intend to, having many options, and moving parts only frustrates an inexperienced user.

1.2 Client

In the world of complicated timesheet tools catering mainly for big companies, it can be overwhelming to find a tool that is easy to use and only holds the necessary features for a smaller business. To this problem, EV-metalli Ky in Jyväskylä, has requested a solution.

EV-metalli KY is a small metal subcontractor that also specializes in machine shop industry and metalwork. They are a family business, currently in second generation management. They were established in 1989 as a limited partnership and they employ around 1 to 4 people at a time. Their workshop locates in Jyväskylä, Finland. Further in the documentation they will be referred to as the client. (EV-metalli KY.)

1.3 Objectives

The objective of the project was to develop a simple timesheet system for the client's specific needs. The client had requested a light-weight system which would only have the features they need. Most timesheet services have been made to meet the needs of bigger companies with dozens of employees, and thus they contain plenty of unnecessary features making the service heavy and impractical for a smaller firm to use.

One of the requests was also that the system be accessible outside the workplace to make it easier for them to handle the calculation of salaries at home. One of the objectives is to make it as easy to use as possible for it to be usable for users regardless of their age and familiarity with technology. More about the features the product holds can be read further into the documentation, in chapter 4.

The client had also suggested that there could be other small companies interested in the requested system, and there might be market for a user license. To make the product sold elsewhere was not one of the main objectives but it was put into consideration while in development.

2 Available solutions

2.1 What can be learned

In order to create a solution to the problem at hand, it is recommendable to look at available solutions. By researching what has already been developed, it is easier to understand the ways to record one's working hours and what must be taken into consideration. It is also a good way to come up with ideas. The overviews of these services are brief as they most often require a subscription to have full access.

2.2 Visma Entry

The first search result for Finnish timesheet systems was Visma Entry. They market themselves as flexible and able to be shaped for any industry. With a quick glance it was clear that the system has plenty of different functions and would work on multiple platforms. In most cases, this service seems like a good choice for most businesses. However, the client has emphasized that they want a system that is simple and trimmed from any excess features. (Visma N.d.)

Without going into the cost of Visma Entry it is visible that there are too many features for the client's needs, and it would require training before being efficiently used. By looking at this solution it has become clearer what some of the most important aspects of the system needed to be. It is not possible to know all the features Visma Entry provides without using the actual application. But it seems that the focus is the timesheet entry.

Unlike the Visma Entry software, the timesheet service addressed in this documentation was developed with a small business in mind. To save the client's time and money, the service needs to be ready to use immediately and have only the necessary features. The focus of this service is not only the timesheet entry. It is a small part of the real purpose. The client wanted a service that tracks the work tasks provided by the customers and thus the timesheet entries in this aspect. Visma Entry claims to be customizable, however, they might not be equipped with the parts the client needs.

2.3 Reportronic

Reportronic is not mainly a timesheet service but a project management system (Reportronic N.d.). However, working hour entries are a part of it. It is a partly familiar system. A Reportronic entry is executed by inserting the date, hours worked and what has been performed into a web form. It was a simple, easy to use system. However, these entries were usually entered once a day and had only a few forms. The client has requested that there would be more data to be entered per entry, such as the customer and methods used. Regardless, a look at Reportronic has showed how a simple form can be efficient as well.

The timesheet service aspires to track work task progress and the hours spent on them. In this aspect, Reportronic is closer to the intended service than Visma Entry as it tracks projects. However, the timesheet service needs to be as simple as possible and have industry specific options on the user interface. It needs to handle the customer data and the relationship between work tasks and customers as well.

Reportronic is better for general use. The timesheet service is supposed to be mindful of the client's industry.

3 Methods

3.1 Decision process

The first issue was to determine which methods to use and what kind of application would be the smartest choice to create. Use of the LAMP stack was initially a suggestion; however, as Linux, Apache, MySQL and PHP proved to be to the most familiar technological choices for the developer, it was quickly chosen. It also utilized open-source technologies and proved to be a cost-efficient choice.

3.2 Linux

Ubuntu, the operating system chosen for the service, belongs to the Linux family and thus serves as the first letter L in LAMP stack. To make the service always run, it was necessary to purchase a virtual operating system in a cloud as it is more cost efficient than an actual server computer. This virtual server machine served as a foundation and a storage space for the actual servers. DigitalOcean provided an easy way to get Ubuntu as well as a comprehensive supply of tutorials and documentation to aid in building the stack. The version of the Ubuntu system used in development was 18.04 and its terminal was accessed using PuTTY client.

3.3 Apache

The next letter in LAMP stack, A, signifies the open source HTTP server Apache. Apache is an old server celebrating its 25th anniversary in 2020 but remains regardless a popular choice (The Number One HTTP Server On The Internet N.d.). As a web server, Apache delivers data from the server storage, in this case Ubuntu, to the

Internet browser. It receives requests from the browser and searches the corresponding files from the server machine to display. (G. 2020.)

3.4 MySQL

A database was an essential part of the system, which is why MySQL serves as the M in LAMP stack. MySQL is a server that manages relational databases. It is open source and belongs to Oracle Corporation. (What is MySQL? N.d.) The task of a relational database is to store different types of data in tables and manage them in connection to each other. Whereas the Apache server transfers web data, the MySQL server transfers data from and to the database using queries. The MySQL database queries are carried out using the SQL language. (What is a Relational Database Management System? N.d.)

3.5 PHP

The last letter of the LAMP stack, P, stands for the programming language PHP essentially used in the development of the service. PHP is used to run scripts on the server, and it is mostly incorporated into web files containing the programming language HTML. It can only be used if it has been installed on the server. PHP was chosen because it is open source and can be used to access MySQL. (What is PHP? Write your first PHP Program N.d.)

3.6 Laravel

3.6.1 Overview

Laravel is a web application framework that was used to make the project easier as it offers many useful commands that help create a database and web files (The PHP Framework for Web Artisans N.d.). It creates dependencies between files without any hassle and fills new files with hints on what to do with them. It has a large and easily accessible documentation that offers knowledge on how to best use Laravel.

Laravel version 7 was used in the project to create and administer the database with Eloquent ORM, to query data from the database, paste it on a web site and the other way around in an effortless manner. Most service functionality were performed utilizing Laravel. Laravel needs up to date servers to reach its full potential.

3.6.2 Migrations

> *Migrations are like version control for your database, allowing your team to modify and share the application's database schema*
>
> *(Database: Migrations N.d.).*

Database migrations were carried out using Artisan commands at the root folder of the project on the Ubuntu command line. By making a migration by running *php artisan make:migration file_name*, a base file for a database table was created. It was then filled with the needed information, the fieldname, datatypes of fields, keys, and restrictions. After the table was outlined, another command, *php artisan migrate*, was then run on the command line to deploy the table.

3.6.3 Eloquent ORM

Models are provided by Eloquent ORM which is also a part of Laravel. When a database table is created in Laravel, a Model file of it is also created. The Model communicates with the corresponding table and is used for data queries and adding records into that table without having to manually access the database. In these Models, it is possible to also define relations between that specific Model table and another table in the database. (Eloquent: Getting Started N.d.)

3.7 Alternate methods

3.7.1 LEMP

The LEMP stack signifies Linux, Nginx, MySQL and PHP, which means that in regard to the LAMP stack used in the project, only Apache would have been substituted. As Apache, Nginx is an open source HTTP server and can be installed on Ubuntu. Nginx

is known for its ability to handle over 10 000 clients on one server. It does not have many differences compared to Apache when it comes to performance or security. However, Apache is more flexible as it is customizable through dynamic writing modules while this is not possible with Nginx. Unlike Nginx, Apache also has the .htaccess file, which is used to change settings around the system efficiently. Regardless of these issues, Apache was chosen over Nginx because of its certain compatibility with Ubuntu and the familiarity of it. (Leslie 2018.)

3.7.2 LAPP

The LAPP stack includes Linux, Apache, PostGreSQL and Perl, replacing the database server and programming language of the stack used in the project. PostGreSQL is an open source object-relational database, which means that it is a compilation of two different database types. In addition to tables with relations, PostGreSQL can store objects used in some programming languages. (Object-Relational Database [ORD] N.d.) In comparison to MySQL, PostGreSQL is more diverse as it is compatible with multiple other technologies, and it also performs more complicated queries (PostgreSQL vs MySQL: What's the Difference? N.d.). Although PostGreSQL sounds overall like a better choice, the developer had no experience using it; hence, MySQL was chosen for its familiarity.

Perl is an old scripting language. Perl can be used in web development, and it works well with Apache, multiple different database servers and runs on Ubuntu. It has a large documentation to seek help from. (Why Perl is a Valid Choice N.d.) However, Perl is a complex language to learn and cannot be called inside HTML code unlike PHP (Perl vs PHP N.d.). Perl is also not used as much anymore and learning a completely new language that probably would not be used in any other project did not seem practical.

4 Requirements specification

4.1 Service overview

The product is meant to serve as a timesheet for the client. Its purpose is to be a simple and light-weight service that the client could also access outside the workplace. It is going to replace paper which could easily be misplaced.

The product would involve a service with which the client could add employees, customers and phases of work into the system as well as edit and delete them. The client's employees would use the system most as its main feature was for them to be able to log their work hours, the date of the entry, what kind of procedures they have performed and to which customer, as well as how close to completion the objective is. In case there was time left after the main service was finished, an additional service of payroll computations based on the hours worked could be added to the system.

4.2 Service requirements

4.2.1 Functional requirements

Functional requirements define the way the system is supposed to operate by introducing the required functions and the way they should work. They explain how the system is supposed to act when certain requirements are met. In a way, they define the processes that make the system realize itself. They require the system to perform a certain function. (Vaatimukset [perinteinen hanke] N.d.)

The most important functional requirement of a timesheet service is to correctly handle login information. When the employer logs into the service, their view is different from an employee's view. They have more authority and can add, edit or delete employees and customers from the system. The employees' information only

allows them to view their own data and add their working hour entries. It is important to distinguish these two different user types.

Another fundamental requirement is for the data to be saved to a database. A database is the heart of this system as the entries, user information and customer information are stored into it. The information should be possible to be stored via web forms and displayed on web pages when needed. A database needs to be well organized and secure.

The service should in real-time sum an employee's working hours per day, per month and per year. It should be able to store this information to an employee's data depending on the one using the service at the time. This is necessary in case the service is to handle payroll computations as the salary is mostly calculated according to one's active hours.

The client had two requests regarding specifically the customer data. One was that when entering a new customer to the database, there should be a possibility to add an image to the entry. Another request was that an existing customer's data should be possible to be printed as a PDF as the client still wanted to have the customer info on paper as well.

4.2.2 Non-functional requirements

Non-functional requirements include the requirements that do not require a function but define the requirements of the necessary background processes – the restrictions and constraints. These processes are associated with performance, accountability, security, and usability of the system. Non-functional requirements are vital to the system and need to be specified with utmost care. (Nonfunctional requirements N.d.)

Regarding the performance of the service, its most important non-functional requirements deal with the capacity and response time of the system. However, the client does not have many employees and thus the service has only a few active users at a

time. The web pages within the service do not hold that much functionality or data to make capacity a relevant issue either. Therefore, it was not designed beforehand. The response times are a more pressing issue as in current times people are used to quick operations. It was decided that the response time should always be less than 5 seconds.

The timesheet service handles important personal data and as such it had to be secure. Different employees should not be able to access each other's data, and guests of the page should not see anything without credentials to the service. Only an admin has the authority to edit customer data and enter new customers to the system. The admin would also be the only one to see all employees' data.

The web service should be usable. Web forms would need to be as simple as possible, and in case there was data that needed to be entered in a certain way, it would need to be explained on the page. Every link and button needed to be logical and lead to a page they clearly indicated they would lead to. Everything in the system would be designed with the end user in mind.

5 Implementation

5.1 Service architecture

5.1.1 Database structure in the beginning

The first version of the database was designed to be simple and straightforward. It was also designed with less knowledge of the possibilities that the technologies used offered and thus changed in the process. The changes will be discussed further in chapter 5. Figure 1 presents the database's core tables, which are further discussed below the table.

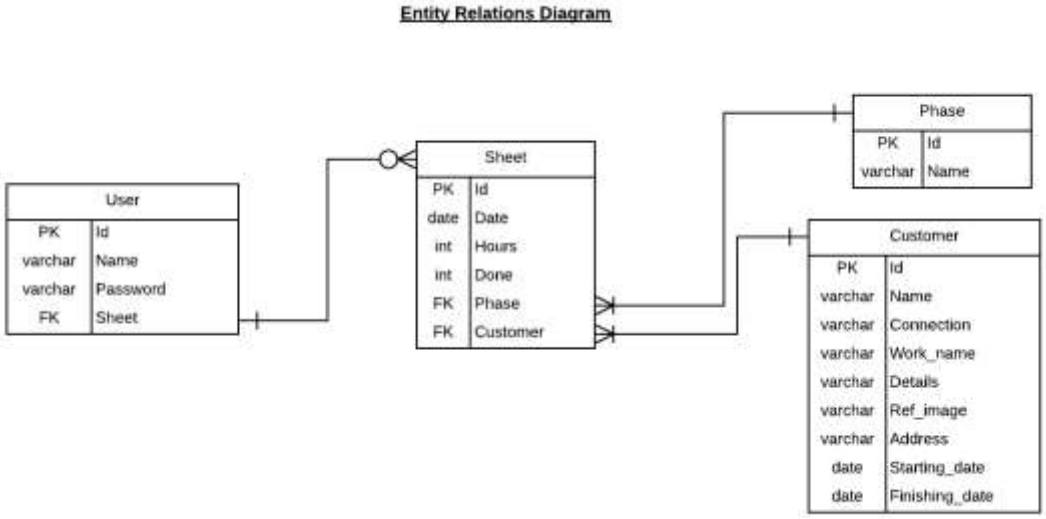

Figure 1. Entity Relations Diagram

The relations shown in Figure 1 present the core tables in the timesheet service. The first table on the left, the User (or Users) table, represents the employees. In the table model, the first vertical column holds the data types while the second one presents the data as it is named in the database. The different data types in the diagram are primary key (PK), varchar, integer, foreign key (FK) and date. When specified, each data type only accepts a certain type of data to be stored in the database column. Primary key serves as an identifier and is unique to every record. Varchar means that the data entered needs to be string-type while int signifies that it needs to be an integer. Date is self-explanatory containing date data. Foreign key data refers to data from another table. It serves as a bridge between the tables.

In the diagram in Figure 1, the user table holds the data needed to log into the system. The line from user table's foreign key to the sheet table signifies that these two tables have a relation. The vertical line on the user end of the line designates that the relation the sheet table has towards the user table is one to many which means that a timesheet entry may only have one user as their composer. The other end of the line that has a circle and many lines connecting it to the table means that the user table's relation to the sheet table is none or many to one. One user may have various

amounts of timesheet entries from zero to many, although having no sheet entries defeats the purpose of the service.

The phase and customer tables also have a relation to the sheet table. The timesheet entries may have only one customer and only one specified work phase. As a user may have many timesheet entries, phases and customers may belong to many different entries. For example, one customer's request may be worked on multiple occasions and thus it belongs to many different entries.

5.1.2 Site map

As the timesheet service is provided via a web browser, it is important to predetermine a smooth web page flow for better accessibility. After clicking a link leading to the next page the user should already have an inkling on what the content of the following page is. The content should not be available to every guest visiting the site either. Figure 2 presents the design of the web site flow as a map.

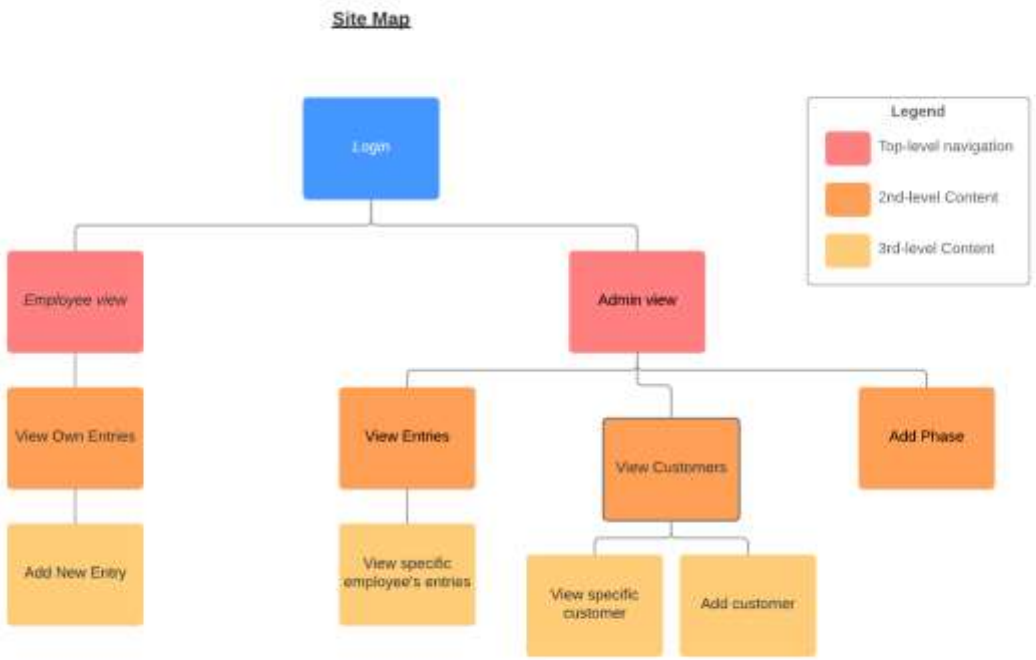

Figure 2. Site map

The order of the pages is descending, the first page being the login page. Only a user or an admin may view the pages following it. A regular user, an employee, may only view their own timesheet entries and create new ones. They land first on a view showing them their previous entries. By pressing a button on the page, they will be able to move on to the other available view, the page holding the form to enter a new timesheet entry. After the entry is ready, the user will be returned to the landing page.

An admin would have more functionality available. Their landing page presents them with three choices; to view their employees' timesheet entries and to add another phase to the database through a web form or to manage the customers. By viewing the employees' timesheet entries, the admin can also decide to view a certain employee's entries by choosing an employee. Managing the customers opens first to a view of all the customers already entered to the database. The admin may choose to either delete or edit a customer's data or add a new customer. They may also view a single customer's data on a new page. Choosing to add a new phase from the landing page opens a simple form. Entering a new phase returns the admin back to the landing page.

5.2 Environment building

The project was started by purchasing DigitalOcean credits on their website and then an Ubuntu operating system-based droplet. Before the droplet was created, an SSH key was added to the profile. To increase the security of the connection, SSH keys were created to serve as authentication when accessing the droplet. To use SSH keys, they were first generated using PuTTYgen. A passphrase was entered to serve as something close to a password when using the keys. Generating created two files – a public and a private key file. The public key file was then entered to the security section of the DigitalOcean account. As a Ubuntu droplet was purchased, that same

public key was chosen when asked to add an SSH key. (How to Create SSH Keys with PuTTY on Windows N.d.)

To use the droplet, its IP address was pasted as the destination IP field on PuTTY's configuration Session tab view. The private SSH key file created earlier was linked for authentication on the auth view under the Connection and SSH tabs. Back on the Session tab view the port was set to 22 and connection type to SSH. The session was named and saved for future use. Then, using the key phrase set when creating the SSH keys, a connection to the Ubuntu droplet was established. (How to Connect to your Droplet with PuTTY on Windows N.d.)

After logging into Ubuntu for the first time as a root user, it was necessary to create a new user with admin rights to prevent the further use of root.

> *This is because part of the power inherent with the **root** account is the ability to make very destructive changes, even by accident (Ellingwood 2018).*

With the new user, it was possible to make root level changes with the terminal command sudo as the first part of the command. However, sudo was not used all the time and thus the system changes were not always as deep. The new user was used from then on frequently. A basic UFW firewall was also set up to act as a gatekeeper for new applications as they were installed into the system. (Ellingwood 2018.)

Before adding MySQL to the bundle, Apache 2.4 server and different PHP libraries were installed using the *sudo apt-get install* command. Then, to initialize the use of Laravel, a composer was installed, and the user directory of Apache server was configured. The public_html folder was created at the root of Ubuntu and inside this folder the Laravel project called tunnit was created using a composer command. Further on, this project folder will be referred to as the route public_html/tunnit. (Rantala 2019a.)

After Laravel was ready for use, the MySQL server was installed using *sudo apt install*. The MySQL default credentials, root user and blank password were used to access the MySQL command line for the first time. The MySQL user password was then added, and the authentication changed to native password. After the MySQL server was properly deployed, it was ready for use. (Rantala 2019a.)

5.3 Web development

To access the files on a visual file explorer, WinSCP was connected to the server using the SSH key. This session was also saved to be easily reopened. Every time there is a mention of a file been accessed further on, WinSCP file explorer was open and connected as a default. Atom text editor was used to edit and compose .php files.

The first table created was the customers tables. To create a table, a migration file was first created by using an artisan command, php artisan *make:migration create_customers_table,* in the Ubuntu terminal. This command creates a base file that only needs to be filled with the needed functionality. All migration commands were performed at the root folder of the project, public_html/tunnit. After creating the file, it was opened in Atom and the customers table's data was written according to the first version of the entity relations diagram presented in Figure 1.

Figure 3 presents a snippet of code from the migration file. In this migration file resided the up and down functions of the database table, defining the manner the table records were created and deleted.

```php
class CreateCustomersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('customers', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('name', 50);
            $table->string('connection', 100);
            $table->string('work_name', 50);
            $table->string('details', 100);
            $table->string('ref_image')->nullable();
            $table->string('address', 50);
            $table->date('starting_date');
            $table->date('finishing_date');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('customers');
    }
}
```

Figure 3. create_customers_table.php

The Schema class functions were used in table management. The create method was given the name of the database table, customers, and a blueprint object to help set up the table. (Database: Migrations N.d.) The method was then populated with the definitions of the table's columns according to the entity relations diagram presented in Figure 1. The down function defines how the table was deleted from the database with the drop command.

By taking a closer look at the way how the fields are defined in Figure 3, it becomes clearer how the method works. To a table object ($table), the data type of the field was first defined by typing the wanted type, for example string. Inside the brackets

and apostrophes was the name of the field. If there was a number inside the brackets after the apostrophes, it defined the amount of characters the field accepted. A semicolon closed the statement. One of the statements in Figure 3 also had the expression nullable after the field definition. This meant that the specific field may also be null, empty. After all the needed fields had been defined, the table was migrated using the *php artisan migrate* command in the terminal.

To access the created tables, controllers were generated using artisan. Figure 4 presents the first function inside the CustomerController.

```
class CustomerController extends Controller
{
    use UploadTrait;

    public function index() {
      $customers = Customer::all();
      return view('customers/index')->with('customers', $customers);
    }
```

Figure 4. CustomerController index function

The CustomerController was created with the command *php artisan make:controller ControllerName*. This created a CustomerController.php file in the HTTP folder. A controller holds the functionality connected to the table in question. CustomerController handles the needed procedures to bring the accurate data from the table for a blade file - of the same name as the function - to display on a web page. The index function presented in Figure 4 creates an object called customers. The Customer class uses the function all to bring all customer data to the customers object. Then the index function returns a view defined in the brackets. The latter brackets transfer the customers object to be used in the index blade file as customers.

Figure 5 presents all routes of customer views in the web.php file residing in the routes folder. This file defines how the routes function.

```
Route::get('/customers/index', 'CustomerController@index');
Route::get('/customers/create', 'CustomerController@create');
Route::get('/customers/{id}', 'CustomerController@show');
Route::post('/customers', 'CustomerController@store');
Route::get('/customers/{id}/edit', 'CustomerController@edit');
Route::patch('/customers/{id}', 'CustomerController@update');
Route::delete('/customers/{id}', 'CustomerController@destroy');
```

Figure 5. Customer routes

The different web routes are defined as functions of the Route class. By using different functions, such as get or post, the Route class performs that certain function for the view route in the brackets. The controller in the brackets shows the function taking place inside that view. The post function of the Route class stores data into the database, while the patch function updates the database with new data and delete destroys the record from the database (HTTP Controllers N.d.).

By using the get function, the Route class brings the defined view from the resources/views folder and performs the specific controller's function. For example, the first route is the index route which uses the controller's function of the same name. With the Route class get function the CustomerController's index function brings all the customer data from the database. How it is displayed on the web page is defined in the index.blade.php file in the resources/views/customers folder as illustrated in Figure 6.

Name	Size	Changed
..		30/04/2020 11:19:39
create.blade.php	1 KB	30/04/2020 11:41:46
edit.blade.php	2 KB	30/04/2020 11:43:30
index.blade.php	2 KB	30/04/2020 11:39:23
show.blade.php	1 KB	30/04/2020 11:44:20

Figure 6. Inside resources/views/customers folder

Figure 6 shows the inside of the resources/views/customers folder which holds the blade files matching the different CustomerController functions. A Route class function fetches a customer view from this folder and displays it.

To be able to add customer data into the database without accessing Ubuntu, there needs to be an easily accessible web form. The CustomerController's functions allowing this to be carried out can be seen in Figure 7.

```
public function create() {
  return view('customers/create');
}


public function store() {

  $customer = new Customer();

  $customer->name = request('name');
  $customer->connection = request('connection');
  $customer->work_name = request('work_name');
  $customer->details = request('details');
  $customer->ref_image = request('ref_image');
  $customer->address = request('address');
  $customer->starting_date = request('starting_date');
  $customer->finishing_date = request('finishing_date');

  $customer->save();

  return redirect('/customers/index');
}
```

Figure 7. CustomerController create and store functions

The create function in CustomerController only returns the user to a create.blade.php file view, which has a form that collects the needed data from the user. The store function creates a new Customer object and stores the variable data collected with the web form into the right fields. After the data has been directed to the corresponding fields, the customer object saves the data and returns the user back to the index view as shown in Figure 8.

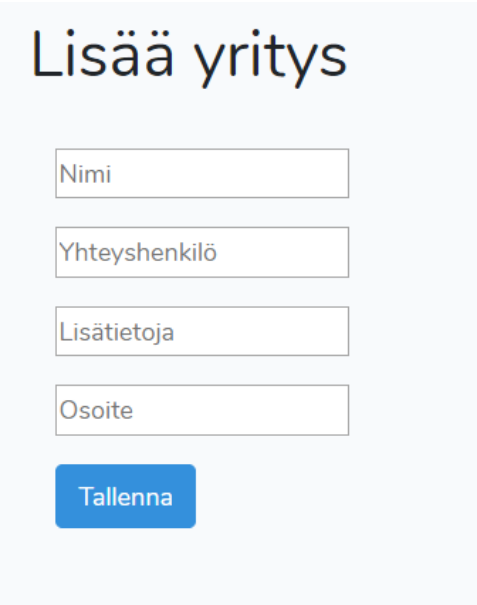

Figure 8. Add a new customer web form

The web form presented in Figure 8 lacks the date, work name and image download fields as it was created according to a more recent version of the database displayed in Figure 31. These four forms receive the customer name, connection, details, and address. The missing forms are later introduced with the Work table and its form. Figure 9 presents an example of the form creation inside the view file.

```
<form method="POST" action="../customers" role="form" enctype="multipart/form-data">

  {{ csrf_field() }}

  <div class="form-group">
    <input type="text" name="name" placeholder="Nimi">
  </div>
```

Figure 9. Form code in create.blade.php file

Inside the create.blade.php file resides the code defining the functionality of the presented forms and their appearance on the page. The service's user interface uses

bootstrap as its framework and the appearance definitions are executed according to it. In Figure 9, the form was first defined with the HTML tag form. Inside the form tag, the method attribute was set to post, which defines the HTTP method used in the form. This form sends the data to the store function of the CustomerController. The action attribute defines the route of the get method. The role and enctype attributes were not important regarding the current form.

The CSRF token in Figure 9 signifies that the form is protected from CSRF attacks. The CSRF token is a feature provided by Laravel. When it is applied in a file with a form, it protects the form from being accessed by unauthorized parties. The token is generated every time a user logs into the system. (CSRF Protection N.d.)

The div tag designates a section inside the code. The div in Figure 9 arranges the section according to bootstrap tag form-group to fit it onto the web page better. It contains an input box that collects data into a variable called name. This variable is then sent to CustomerController's store method. Figure 9 does not show the end of code, which holds more form-groups that collect variable data and a button that verifies the collected data to be sent. The form is then closed with a *</form>* tag.

Because there is a need for a customer's data to be edited through a web page as the client does not have access to the server manually, the functions edit and update are needed. These CustomerController functions are presented in Figure 10.

```
public function edit($id) {
  $customer = Customer::find($id);
  return view('customers/edit')->with('customer', $customer);
}

public function update($id) {

$customer = Customer::find($id);

$customer->name = request('name');
$customer->connection = request('connection');
$customer->details = request('details');
$customer->address = request('address');

$customer->save();

return redirect('/customers/index');
}
```

Figure 10. CustomerController edit and update functions

The edit function in Figure 10 populates the customers object with data of a single customer record by using the Customer class find function and giving it the parameter id. Every customer was generated a unique id to differentiate them from each other and by using that id as a parameter, the function could find the specific customer field.

The update function also uses the id as a parameter. In the function, the customer object is again given the data of a single customer by using the Customer class find function. Then that customer's data is changed by fetching the variable data submitted on the edit.blade.php page and overriding the old fields with the new data. Figure 11 below shows how the CustomerController functions handle showing and deleting specific customer data.

```
public function show($id) {
  $customer = Customer::find($id);
  // Kokeile myös usein parempaa
  //$customer = Customer::findOrFail($id);
  return view('customers/show')->with('customer', $customer);
}

public function destroy($id) {
Customer::find($id)->delete();
return redirect('/customers');
}
```

Figure 11. CustomerController show and destroy functions

The CustomerController show function searches a specific customer's data and passes it to a blade file called show. This blade file displays the data on a web page. The destroy function first finds the preferred customer record and then deletes it, then redirecting the user back to the index page.

5.4 Features

5.4.1 Timesheet

After the and customer table had been created and their functionality added, the next table needed was the Work table. The Work table was different from the previous table because it had foreign key in the table referencing the customer table. This was to be able to access this table and present data from it in affiliation to the Work table. The Work table was crucial after the database changes presented in Figure 31. A foreign key was then used in every table created.

The Works table contained new data types in contrast to the Customers table. These data types and their initialization can be seen in Figure 12.

```
public function up()
{
    Schema::create('works', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->string('person')->nullable();
        $table->string('mark')->nullable();
        $table->string('details', 100)->nullable();
        $table->string('ref_image')->nullable();
        $table->boolean('done')->default(0);
        $table->boolean('charged')->default(0);
        $table->boolean('paid')->default(0);
        $table->date('starting_date')->nullable();
        $table->date('finishing_date')->nullable();
        $table->bigInteger('customer_id')->unsigned()->nullable();
        $table->foreign('customer_id')->references('id')->on('customers')->onDelete('cascade');
        $table->timestamps();
    });
}
```

Figure 12. create_works_table.php

The new data types used in creating Work table fields were foreign, and Boolean. Regarding the foreign key, there were two different statements with the same field name. In the first statement, the foreign key field was initialized by defining the data type of the id field in the other table, so the receiving table matches it. The unsigned attribute allows the field to receive a large number (Numeric Type Attributes N.d.) A Boolean value can only be true or false. With the Boolean data type the done, charged, and paid fields enabled a checkbox to be ticked or empty, the ticked value being 1. The default value set to these fields was 0, false. Figure 13 presents how the new foreign key relationship were handled inside the tables' Model files.

```
class Work extends Model
{
    public function sheet(){
    return $this->hasMany('App\Sheet');
  }

  public function customer(){
    return $this->belongsTo('App\Customer');
  }

  public function getImageAttribute() {
    return $this->ref_image;
  }
}
```

Figure 13. Inside the Work Model

To access the foreign key relationship in the controller and blade files, the tables' relationship needs to be defined inside their Model files as well. The function carrying the name of the other table in the relationship returns the Model and allows it to have the defined relationship with the Model inside the brackets. The Work table has a one to many -relationship with the Sheet table, in the code implicated with the hasMany method. The relationship with the Customer table is a many to one and that is specified with the method belongsTo. Figure 14 presents how the relationship is handled in the Customer Model.

```
class Customer extends Model
{
    public function work(){
      return $this->hasMany('App\Work');
    }
}
```

Figure 14 Customer Model file

To finalize the relationship between the tables, there needs to be a reference on the other table's Model as well. In Figure 14, the Customer table is defined to have a one to many relationship with the Work table. When creating a new record, the other table needs to be accessed as well to enable the relationship to another table's record. Figure 15 addresses the addition of customer data inside a work record.

```
public function create() {
  $customers = Customer::all(['id','name']);
  return view('works/create', compact('customers'));
}
```

Figure 15. WorkController create function

Because the Work table has a relationship with the Customer table, it needs to be taken into consideration when creating a new record. In Figure 15, the customers variable is defined to include the id and name field of the customer table, using the customer class all function. The customers variable is then passed into the create.blade.php using the php function compact that creates an array of the variables inside the brackets (Compact N.d.) Figure 16 presents how the customer data is collected into a dropdown menu.

```
<div class="form-group">
   <select class="form-control" name="customer_id">
      @foreach($customers as $customer)
     <option value="{{$customer->id}}">{{$customer->name}}</option>
      @endforeach
   </select>
 </div>
```

Figure 16. Customers dropdown code in works folder create.blade.php

Inside the create.blade.php file the customers list is gone through with a foreach loop. The loop presents all the customers currently available in a dropdown view as a customer is first brought using their id. Then their name was found in the record matching the id and shown in the dropdown. By defining the select tag's name to be the variable passed into the work record being created, the customer clicked from the dropdown view is chosen as seen in Figure 17.

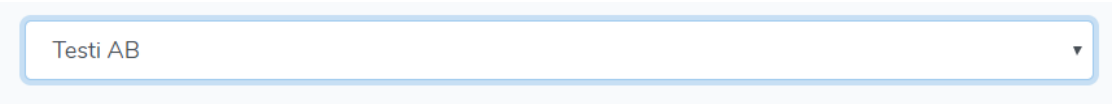

Figure 17. Dropdown to select customer in create Work view

Figure 17 presents the dropdown that contains the customers available. By hovering over it, it expands downwards and reveals more choices. The chosen customer can then be shown in relation to the added Work record as seen in Figure 18.

Figure 18. Work folder index.blade.php view in browser

Figure 18 shows that the choices carried out using the dropdown were successfully inserted into the database. This method was also utilized in the SheetController as it is possible to choose from work records, which job the employee is working on.

The Sheets table needs the Work and Customers tables to exist before being initialized as it is a child to the Work table. After these tables were successfully created, it was possible to add the Sheets table and records. A timesheet entry was executed via a web form presented in figure 19.

Figure 19. Sheets folder create.blade.php view in a browser

The date and working hours data were entered by hand into form fields. The work phases displayed as checkboxes were ticked to indicate which phases were performed. A task was then chosen from a dropdown view of a list of work records. By pressing the Tallenna button, the new entry was saved into the database. Figure 20 shows how the checkboxes in Figure 19 were outlined in the view file.

```
<div class="form-group">
    <input type="checkbox" name="phases[]" id="hitsaus" value="hitsaus">Hitsaus
    <input type="checkbox" name="phases[]" id="maalaus" value="maalaus">Maalaus
    <input type="checkbox" name="phases[]" id="sahaus" value="sahaus">Sahaus
    <input type="checkbox" name="phases[]" id="korjaus" value="korjaus">Korjaus
</div>
```

Figure 20. Hardcoded checkbox array of work phases in create.blade.php

The phase checkboxes were inserted into a form group inside the timesheet creation form. The name was arranged to be an array to signify that all phase values could be inserted into an array that would then be passed to the SheetController store function presented in Figure 21. The ids and values of the checkboxes set the values of the single variables that were passed into the array.

```
public function store(Request $request) {
  $sheet = new Sheet();
  $request->merge([
    'phases' => implode(', ', (array) $request->get('phases'))
  ]);
  $sheet->date = request('date');
  $sheet->hours = request('hours');
  $sheet->phases = request('phases');
  $sheet->user_id = auth()->user()->id;
  $sheet->save();
  return redirect('/sheets/index');
}
```

Figure 21. SheetController store function

Inside the SheetController store function, the request object injected the collected data into a merge function. The merge function joined the array data received from the checkboxes with the local variable. The local phases variable was populated with the implode method that created a collection where the single variables in the array are sorted with a comma. After the local phases' variable had been populated with the array data, it was passed to be inserted into the new sheet record.

Figure 22 presents the timesheet entry on a web page. It shows that the checkbox data was successfully inserted into the database.

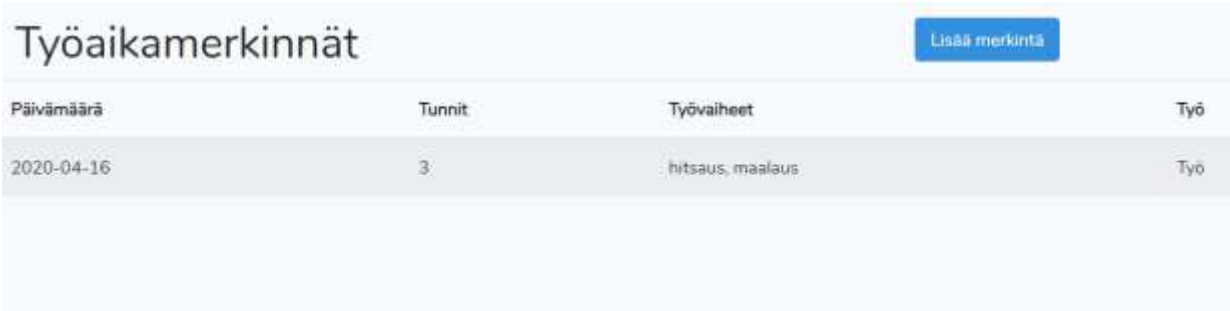

Figure 22. Timesheet index view in browser

5.4.2 Image upload

One of the requirements listed in chapter 3 was uploading an image into the database. An image cannot be inserted into the database as is. The string saved into the ref_image field was used as a path that leads to the right image inside the folder system. Figure 23 shows how the image data was collected using a form.

```
<div class="form-group">
  <input id="ref_image" type="file" class="form-control" name="ref_image">
</div>
```

Figure 23. Work folder create.blade.php file upload

A file upload was added inside the create.blade.php view file containing the form that collects data to be inserted into the Work table. By defining the input's type as file, the system added a button opening to the local computer's file explorer. After the file was chosen, it was passed to the WorkController with the id ref_image. The *enctype="multipart/form-data"* presented in figure 9 was used in this form to handle the file data. The enctype defined the manner the data entered was handled. The multipart/form-data specified that the form included multimedia and needed to take that into account. (HTML <form> enctype Attribute N.d.) After the form data was

verified, the image data was handled inside the WorkController store function as shown in Figure 24.

```
public function store(Request $request) {
  $work = new Work();

  $this->validate($request, [
    'ref_image' => 'image|nullable|max:1999'
  ]);
  if ($request->hasFile('ref_image')) {
            //get filename with extension
  $fileNameWithExt = $request->file('ref_image')->getClientOriginalName();
            //get hyst filename
  $filename = pathinfo($fileNameWithExt, PATHINFO_FILENAME);
            //get just ext
  $extension = $request->file('ref_image')->getClientOriginalExtension();
            //filename to store
  $fileNameToStore = $filename.'_'.time().'.'.$extension;
            // upload image
  $path = $request->file('ref_image')->storeAs('public/images', $fileNameToStore);
    }else {
      $fileNameToStore = 'noimage.jpg';
    }

  $work->name = request('name');
  $work->mark = request('mark');
  $work->customer_id = request('customer_id');
  $work->person = request('person');
  $work->details = request('details');
  $work->ref_image = $fileNameToStore;
```

Figure 24. WorkController store function

In the store function, the received data was tested if it contained ref_image uploaded file data. In case there was an image uploaded, the fileNameWithExt variable got the name of the image uploaded as it was in the file explorer with the getClientOriginalName function. Then the filename variable was set to hold the name of the original file without the file extension using the pathinfo function. The extension variable was then initialized with just the file extension of the original image file using the getClientOriginalExtension function. The fileNametoStore was the variable that would be passed to the Work record, and it was constructed of the original filename without the extension, underscore, the time it was uploaded, and just the file extension of the image. This way there would not be multiple images with the same

name. The image was then stored to a storage/public/images folder, where it would be brought from in case it was requested on a view. The storage was initiated using the *php artisan storage:link* command on the Ubuntu terminal. (Laravel From Scratch [Part 12] – File Uploading & Finishing Up 2017.)

When there was no image data, the else expression used a default image to serve as a substitute and filled the variable fileNameToStore with the empty image name (Laravel From Scratch [Part 12] – File Uploading & Finishing Up 2017). The image name was then passed to the new Work record as the ref_image data. This same method was also used in the update function.

5.4.3 Authentication

Keeping the application data accessible to any guest would have been unwise as it handles personal information as well as business information. To only allow those involved to use the website and see the information, an authentication system was created. Fortunately, creating a log in a system using Laravel was easy and typing a single command on Ubuntu command line initialized the whole feature ready for use. By running the command *php artisan ui vue –auth*, the authentication controllers, such as LoginController, as well as the Register and Login views were generated and ready.

The user_id field was added into the sheets table afterwards, using the artisan command *php artisan make:migration add_user_id_to_sheets_table*. After this migration file was filled with the integer type user_id field data as in previous migration files, the migrations were run to verify the new field.

To make the timesheet data available to an authorized user, the auth middleware was used to differentiate guests from users. A middleware functions to allow only the kind of requests to the system that it is programmed to. If someone unauthorized tried to access pages that are run through the auth middleware, it returned them back to the login page. (Middleware N.d.) In Figure 25 presenting the way SheetController handles a specific authenticated user's data, the function

__construct variable *this* refers to the SheetController and initializes the use of auth middleware when SheetController is operated, not allowing any unauthorized user to access the SheetController pages (Rantala 2019b).

```
class SheetController extends Controller
{
    public function __construct(){
      $this->middleware('auth');
    }

    public function index() {
        $user_id = auth()->user()->id;
        $user = User::find($user_id);
        return view('sheets/index')->with('sheets', $user->sheets);
    }
```

Figure 25. SheetController authentication

In the index function also presented in figure 25, the user entering the page is first found in the database. The variable user_id takes the value of an authenticated user's id. Then a user variable is initialized to match the user_id passed in the previous expression by using the User class find function to find it. The index view then returns only the Sheet data registered to that one user.

To ensure that every timesheet entry is stored with the right user_id matching the current user, the auth middleware is used to get the current session. Then the current user is checked, and their id is returned from the database to create the user_id stored into the field as presented in Figure 26.

```
$sheet->user_id = auth()->user()->id;
```

Figure 26. An expression inside the SheetController store function

An Admin middleware was generated to restrict certain actions and pages from common users. Figure 27 presents the functionality of this middleware.

```
public function handle($request, Closure $next)
{
    //dd($this->auth->getUser());
    if ($this->auth->getUser()->email !== "testi@gmail.com") {
        return redirect('authproblem');
        abort(403, 'Unauthorized action.');
    }

    return $next($request);
}
```

Figure 27. AdminMiddleware handle function

Inside the AdminMiddleware.php file handle function, the authenticated user was brought with getUser function, and it was tested whether the email of the user did not match the one inside the quotation marks. If that were the case, the system would throw an error. If the user email matched the one in the quotation marks, the system would allow the user to proceed with what they were doing.

To ensure that the admin middleware was considered a middleware, it was initiated inside the routeMiddleware object handling the middleware in the system as shown in Figure 28 (Rantala 2019b). This object resided inside the app/http/Kernel.php file.

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'admin' => \App\Http\Middleware\AdminMiddleware::class,
];
```

Figure 28. RouteMiddleware in the app/http/Kernel.php file

To easily specify the pages only available to an admin, the group function of the Route class was used inside web.php file as presented in Figure 29. Inside this function were routes as shown in Figure 5. To get to the array of pages, a user would have to go through the middleware auth and admin. If the user trying to access the pages was indeed a user but not an admin, they would be returned to home page.

```
Route::group(['middleware' => ['auth', 'admin']], function(){
```

Figure 29. Route group only admin can see

A navigation bar file was created as an individual file, so it was easy to include it on every page without the need to copy the whole navbar. However, every item on the navbar was not accessible to every user type. The navigation items were wrapped in a hierarchy of if statements as seen in Figure 30.

```
@if(!Auth::guest())
 @if (Auth::user() && Auth::user()->type == 'admin')
 <li><a class="nav-link" href="{{ url('/') }}/sheets/showall">Työaikamerkinnät</a></li>
 <li><a class="nav-link" href="{{ url('/') }}/customers/index">Asiakkaat</a></li>
 <li><a class="nav-link" href="{{ url('/') }}/works/index">Työt</a></li>
  @else
  <li><a class="nav-link" href="{{ url('/') }}/sheets/index">Työaikamerkinnät</a></li>
  <li><a class="nav-link" href="{{ url('/') }}/works/index">Työt</a></li>
  @endif
@endif
```

Figure 30. Navbar restrictions

The first *if* statement checked if the user was not a guest. In case the user was an actual user, the next *if* statement activated. In the next *if* statement, the Auth middleware checked if the user type of the current user was admin. In case the user was an admin, the navigation links to admin specific pages were presented. If that was not the case, the *else* statement activated, showing the navigational items that lead to user specific pages. If the user was a guest, none of the navigational links were shown. Using this method, some buttons leading to admin only pages were hidden on user accessible pages.

5.5 Structural changes

5.5.1 Why there were changes

As it is possible to see in the files presented before, the structure of the service changed in the process. After an earlier version of the project was presented to the client, it was clear that the focus of the timesheet service was not, in fact, supposed to be on the employees but on the work projects the client receives. Some changes were carried out to bring this aspect to light.

5.5.2 Database changes

The database required some significant changes as a table was deleted and another one added. To clarify the new structure, a new entity relations diagram was visualized. Figure 31 presents the changes performed to the database structure.

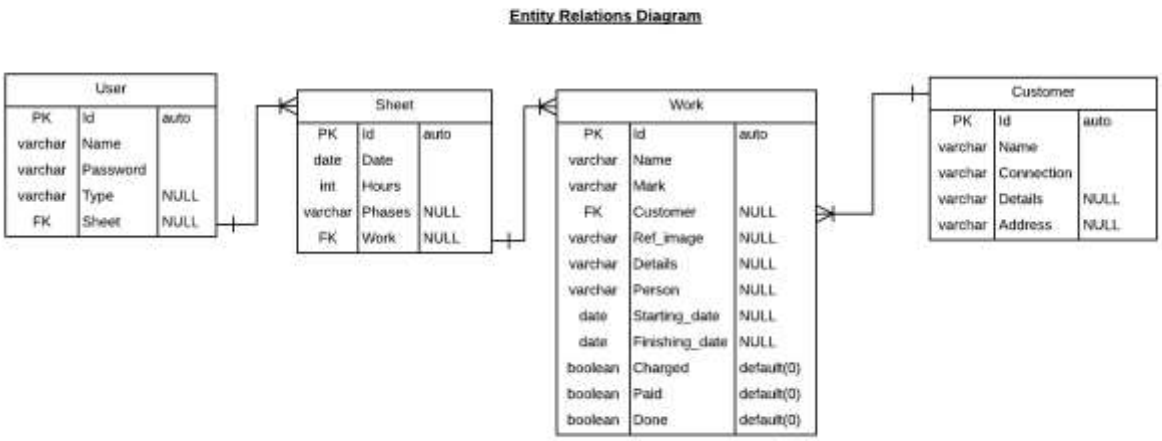

Figure 31. Entity Relations Diagram second version

The Phases table was deleted as it was not important and could be replaced with a hardcoded array that was stored in Sheet table's phases column. User type was added to the user table. This enabled the service to differentiate admins from regular users in the code. The Work table was also added as it became the centre of the service. This table contained columns related to information about the work request, such as name of the job and name of the contact. The Customer table was stripped of multiple columns that were more useful in the Work table. The Customer table held the information regarding recurring customers and was attached to different jobs via foreign key on the Work table.

Unlike the first version of the database diagram, the second version also showed some details about the data. Every primary key Id field in the database was auto incremented and thus marked as auto. The data marked as null contained fields that

could be null. The new data type, Boolean, appearing in the Work table signified that the data was either true or false. In this case, the default value of the fields was set to be 0 – false. The fields without a detail were those that always needed to be filled with data.

5.5.3 Web site map changes

As there was a whole new table, there also needed to be new views to utilize the table. Figure 32 presents the changes to the web page flow.

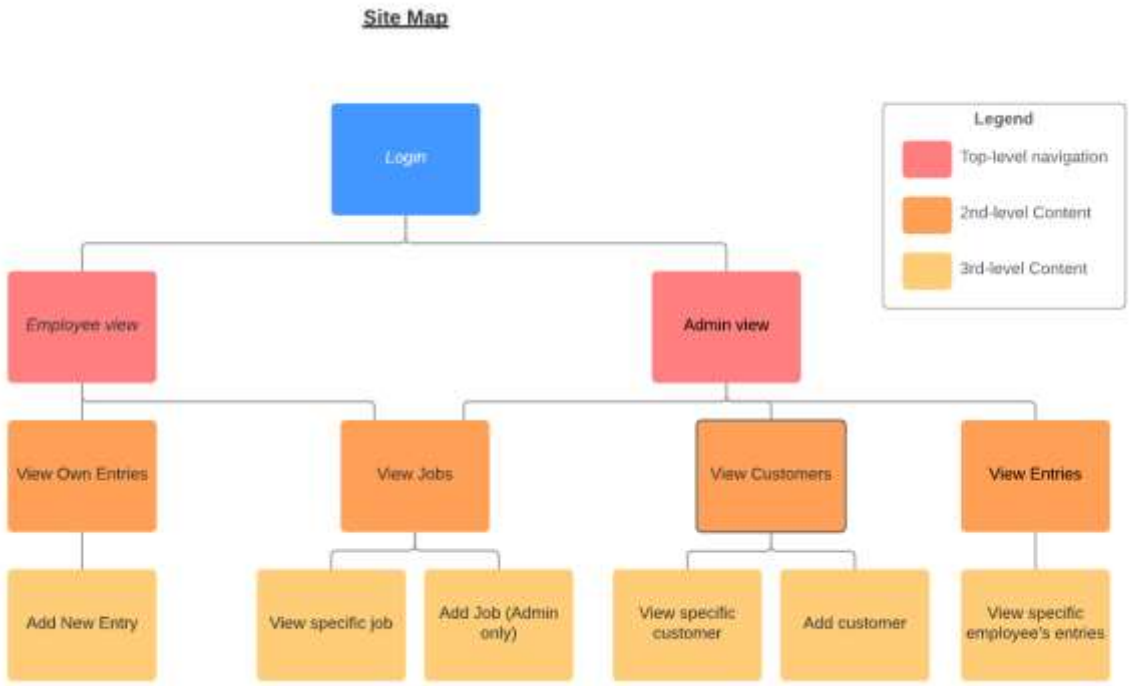

Figure 32. Web site map second version

The phase views were deleted as there was no longer need for them. Both user types were able to view all work records and have a look at a certain record. However, only the admin was able to see the add work button and follow it to a page that allowed the creation of a new work record.

5.6 Problems and solutions

5.6.1 Summary

The development did not always go smoothly and there were some problems in the way. Mostly the smaller problems revolved around certain commands or code snippets being dated to earlier versions of Laravel. However, these problems were solved eventually. Following are a few problems that slowed down the development process but after looking into them were solved.

5.6.2 Database access problem

One of the first pressing problems was not being able to access the MySQL database after it was installed and the administrator authentication was set to native password. When the command to open MySQL workspace, *mysql -u root -p(password) laraveldb*, was entered, an error message came up saying that the access was denied and the user root did not have the rights to access the workspace. After trying to no effect access the database with an extra argument *–skip-grant-tables*, a previous snapshot of the Ubuntu droplet was loaded from DigitalOcean. The server and database were installed again but the error message appeared yet again. After taking a better look at the written commands it became clear that a typographical error when rewriting the administer credentials of the database had been the culprit.

5.6.3 Not enough space for bootstrap

Bootstrap was installed to make the web site easily look more appeasing. Bootstrap also worked well with Laravel. However, installing it was not as straightforward as it was thought to be. When the commands to install it were run on the Ubuntu command line, there were error messages informing that there was not enough space in the system to install the add-on. To avoid having to buy more space, a swap memory was set up on Ubuntu.

5.6.4 Error 1215

Error code 1215 in Laravel signifies not being able to add a foreign key constraint to a database table. This problem was rather easy to solve as the type of the foreign key data had just been defined wrong. As the foreign key uses the other table's primary key id to signify which record it is attached to, the field needs to be the same type as the primary key is in the other table. In this case, the primary keys are big increments and thus the field with the foreign key constraint also needs to be a big increment.

6 Results

6.1 Current functionality

At the moment of this document's finalization the timesheet service is still under development. The service presented in this documentation is a minimum viable product and as such is not yet ready to be deployed for regular use. It lacks some features that make it more user friendly and which will be added later as the cooperation continues.

The service was able to store data of users, customers, work, and timesheets in their own tables via web forms. Figure 33 shows the user interface of a work form, which has the most functionality.

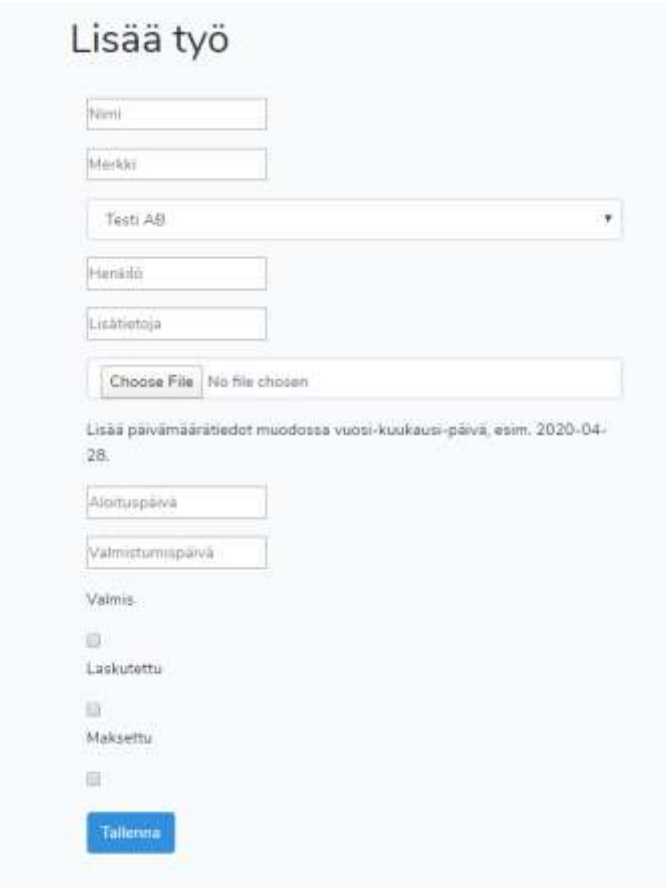

Figure 33. Work form user interface

The inserted Work data could also be viewed individually, and that page could be printed by pressing a button at the bottom of the page. Figure 34 presents an individual Work record with a placeholder image loaded from the database.

Figure 34. Individual Work record on a web page

It could store data from a table to another using dropdowns on the user interface, connected with foreign key relationships. The timesheet entries also stored array data of work phases, collected using checkboxes as seen in Figure 35.

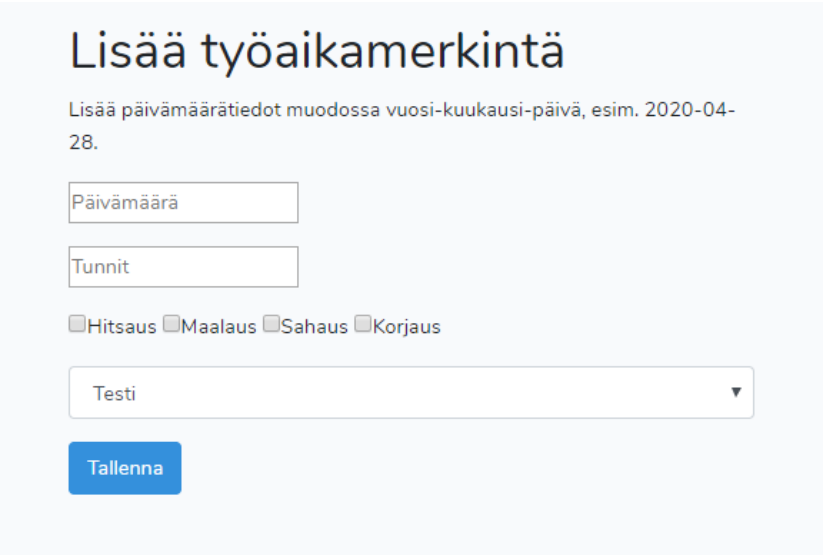

Figure 35. Timesheet entry form

The stored data was displayed on the web page depending on the user. Figure 36 shows one user's timesheet entries in a table view.

Figure 36. Timesheet entries of a user

A common user could not view other users' entries. Only an admin could view every user's entries as shown in Figure 37, access the customer data, and add new jobs. A common user was able to view the jobs but not mark them done, billed, or paid, unlike the admin.

Työaikamerkinnät

Työntekijä	Päivämäärä	Tunnit	Työvaiheet	Työ
Testi Henkilö	2020-04-16	3	hitsaus, maalaus	Työ
Toinen Henkilö	2020-05-05	5	maalaus, korjaus	Työ
Toinen Henkilö	2020-05-08	7	sahaus, korjaus	Testi
Toinen Henkilö	2020-05-10	2	hitsaus	Testi

Figure 37. Admin's view of all timesheet entries

6.2 Future improvements

In the future the service should have a feature that counts the user's working hours per day and per month, collecting them into a monthly report. This was one of the features earlier considered crucial. However, it was not realized in the version this document addresses as the focus shifted from the employees to work tasks, mid development. This could be done by searching the timesheet table for a certain user, a certain date, and hours. After all the entries from that user and date would be selected, the float values in the hours fields would be summed taking into consideration how to handle time data. Then the summed values would be passed into a view.

A calendar feature could show the most urgent jobs in a calendar view to make it easier to prioritize what to do next. A visual calendar could be added using JavaScript and finding all the deadline dates of the work tasks from the database. By pressing a date with a task, the task's show page would open. This way it would be easy to have a perception of what to do next.

7 Conclusion

The development had a few difficulties resulting from the inexperience of the developer as well as the short attention span that led to some delays and the incompleteness of the service. Fortunately, there was a great amount of help to be found in online manuals and tutorials. Many of the problems faced in this project had also been experienced by various other developers beforehand; by studying the code written and by furious googling, the problems were mostly defeated. DigitalOcean's tutorials were a great help as they are easy to follow, explanatory and detailed. The communication with the client was most often smooth but there were some misunderstandings, which are common when trying to realize someone else's idea. The application version addressed in this documentation may not have met with the idea they had in mind but was in the right direction.

The learning process during the development was a great one. In the beginning, the skill level was quite low and programming a whole timesheet service seemed intimidating. After spending hours fixing mistakes and learning how to run things correctly, the fear began to falter. It became clear that just sleeping overnight or waiting a day or two could sometimes make the problem easier to solve. Reading technical documentation and YouTube videos on the topics at hand also helped and understanding them became much easier. This documentation was written to be descriptive not only to explain the matter to a reader but to realize that these issues were truly understood. Not only understanding what the code snippets carried out but also why.

To conclude, using the LAMP stack to make a timesheet service was proven to be possible and recommendable. Its use was beginner friendly, presenting a less experienced programmer with a good framework. Inserting Laravel into the equation proved to be more convenient than having to create the SQL queries by hand in every case. Laravel had many inbuilt, user friendly functions and features. The syntax Laravel used was also easy to read. Purchasing a Ubuntu droplet and running it in a cloud saved space and some time, and it was easy to do. Installing the other parts of the stack was also easy using the previously mentioned tutorials and documentation.

The LAMP stack could do much more than this document lets on. It will be explored further as the development of the service continues.

However, the Laravel basic authentication provided a public registration form that allowed anyone to register as a user. As the timesheet service should only be accessible to the personnel of the client, the registration should either be hidden and only accessible through a shared link or an admin would register new employees into the system. Laravel also has had many changes in syntax between different versions, which sometimes complicated the task to find the right command. There were also many variations in query and function building which, in some cases, complicated understanding the solution to a problem that other developers had deemed worked. However, after these hardships were conquered, Laravel was quite conventional and provided answers to problems.

References

Compact. N.d. PHP documentation. Accessed on 3 May 2020. Retrieved from https://www.php.net/manual/en/function.compact.php

CSRF Protection. N.d. Laravel documentation. Accessed on 3 May 2020. Retrieved from https://laravel.com/docs/7.x/csrf

Database: Migrations. N.d. Laravel documentation. Accessed on 23 April 2020. Retrieved from https://laravel.com/docs/7.x/migrations

Drake, M. 2018. How to Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04. *DigitalOcean, 27 April 2018.* Accessed on 21.4.2020 Retrieved from https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04

Ellingwood, J. 2018. Initial Server Setup with Ubuntu 18.04. *DigitalOcean, 27 April 2018*. Accessed on 21 April 2020. Retrieved from https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04

Eloquent: Getting Started. N.d. Laravel documentation. Accessed on 23 April 2020. Retrieved from https://laravel.com/docs/7.x/eloquent

EV-metalli KY. N.d. Finder.fi business information. Accessed on 23 April 2020. Retrieved from https://www.finder.fi/Metallin+alihankintateollisuus/EV-Metalli+Ky/Jyv%C3%A4skyl%C3%A4/yhteystiedot/211662

G., D. 2020. What is Apache? An In-Depth Overview of Apache Web Server. *Hostinger Tutorials, 25 March 2020*. Accessed on 1 May 2020. Retrieved from https://www.hostinger.com/tutorials/what-is-apache

How to Connect to your Droplet with PuTTY on Windows. 2018. DigitalOcean tutorial. Accessed on 21 April 2020. Retrieved from https://www.digitalocean.com/docs/droplets/how-to/connect-with-ssh/putty/

How to Create SSH Keys with PuTTY on Windows. 2018. DigitalOcean tutorial. Accessed on 21 April 2020. Retrieved from https://www.digitalocean.com/docs/droplets/how-to/add-ssh-keys/create-with-putty/

HTML <form> enctype Attribute. N.d. W3Schools.com tutorial. Accessed on 5 May 2020. Retrieved from https://www.w3schools.com/tags/att_form_enctype.asp

HTTP Controllers. N.d. Laravel documentation. Accessed on 3 May 2020. Retrieved from https://laravel.com/docs/5.1/controllers

Laravel From Scratch [Part 12] – File Uploading & Finishing Up. 2017. A video by Traversy Media. *Youtube, 3 June 2017.* Accessed on 5 May 2020. Retrieved from https://www.youtube.com/watch?v=AL8PCThJ9c4

Leslie, A. 2018. NGINX vs. Apache (Pro/Con Review, Uses, & Hosting for Each). *Hostingadvice, 31 May 2018.* Accessed on 2 May 2020. Retrieved from https://www.hostingadvice.com/how-to/nginx-vs-apache/

Nonfunctional requirements. 2018. *Scaled agile framework, 18 September 2018.* Accessed on 13 April 2020. Retrieved from https://www.scaledagileframework.com/nonfunctional-requirements/

Numeric Type Attributes. N.d. MySQL documentation. Accessed on 4 May 2020. Retrieved from https://dev.mysql.com/doc/refman/5.7/en/numeric-type-attributes.html

Middleware. N.d. Laravel documentation. Accessed on 5 May 2020. Retrieved from https://laravel.com/docs/7.x/middleware

Object-Relational Database (ORD). N.d. Technopedia article. Accessed on 2 May 2020. Retrieved from https://www.techopedia.com/definition/8714/object-relational-database-ord

Perl vs PHP. N.d. Educba article. Accessed on 4 May 2020. Retrieved from https://www.educba.com/perl-vs-php/

PostgreSQL vs MySQL: What's the Difference? N.d. Guru99 article. Accessed on 2 May 2020. Retrieved from https://www.guru99.com/postgresql-vs-mysql-difference.html

Rantala, A. 2019a. Laravel – käyttöönotto. *Web-palvelinohjelmointi – Materiaali, 31 August 2019*. Accessed on 24 April 2020. Retrieved from http://netisto.fi/matsku/php/?id=25

Rantala, A. 2019b. Laravel::DB::ScoreTronic02 – autorisoidut reitit ja omat SQL-kyselyt. *Web-palvelinohjelmointi – Materiaali, 31 August 2019*. Accessed on 5 May 2020. Retrieved from http://netisto.fi/matsku/php/?id=37

Reportronic. N.d. Page on Reportronic website. Accessed on 22 April 2020. Retrieved from http://www.reportronic.fi/

The Number One HTTP Server On The Internet. N.d. Page on Apache website. Accessed on 22 April 2020. Retrieved from https://httpd.apache.org/

The PHP Framework for Web Artisans. N.d. Page on Laravel website. Accessed on 22 April 2020. Retrieved from https://laravel.com/

Vaatimukset (perinteinen hanke). N.d. Page on Helsingin kaupunki website. Accessed on 1 April 2020. Retrieved from https://kehmet.hel.fi/menetelmalaari/vaatimukset/

Visma Entry Työajanseuranta. N.d. Page on Visma website. Accessed on 22 April 2020. Retrieved from https://www.visma.fi/ohjelmistoratkaisut/visma-

entry/?utm_source=google&utm_medium=paid-google_cpc&utm_campaign=tuote_entry&utm_content=Androidty%C3%B6ajanseuranta&utm_term=&type=paid-google&utm_source=google&utm_medium=paid-google_cpc&gclid=CjwKCAjwkPX0BRB-KEiwA7THxiP6dbHeZjAhQr0TJnBb7urY5Bi2g3rtgcu-joLMiZWIPsSSVJb43AnBoCUT4QAvD_BwE

What is a Relational Database Management System? N.d. Codecademy article. Accessed on 1 May 2020. Retrieved from https://www.codecademy.com/articles/what-is-rdbms-sql

What is MySQL? N.d. MySQL documentation. Accessed on 23 April 2020. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

What is PHP? Write your first PHP Program. N.d. Guru 99 tutorial. Accessed on 1 May 2020. Retrieved from https://www.guru99.com/what-is-php-first-php-program.html

Why Perl is a Valid Choice. N.d. Perl documentation. Accessed on 4 May 2020. Retrieved from https://www.perl.org/advocacy/whyperl.html