CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering

Bachelor's Thesis

Redaction and reservation system for project SHerna

Jozef Bugoš
Study program: Open Informatics
Branch: Software

May 2020
Supervisor: Ing. Richard Vachula

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bugoš** Jméno: **Jozef** Osobní číslo: **466219**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Redakční a rezervační systém pro projekt SHerna

Název bakalářské práce anglicky:

Redaction and reservation system for project SHerna

Pokyny pro vypracování:

Navrhněte a implementujte rozšíření a případné úpravy systému projektu SHerna.
Info o projektu na súčasnom webe1.
1) Seznamte se s aktuálním řešením systému. Dohodněte s ostatními členmi pracujícími na tomto projektu funkcionalitu pro jeho rozšíření. Provedenou analýzu zdokumentujte pomocí UML diagramů
2) Navrhněte architekturu nového řešení webu a následně ji implementujte.
2) Na základe analýzy navrhněte a následně implementujte daná rozšíření.
3) Implementaci otestujte jednotkovými a integračními testy.
4) Navrhněte možnosti budoucího rozšíření systému.

Seznam doporučené literatury:

[1] Sherna. Sherna [online]. Praha: Lukáš Figura, 2017 [cit. 2020-01-21]. Dostupné z: https://sherna.siliconhill.cz/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Richard Vachula, Fakulta Dopravní ČVUT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Richard Vachula podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Acknowledgement / Declaration

I would like to thank Ing. Richard Vachula, the supervisor of the bachelor thesis for his valuable advice and guidance, which was very helpful.

I declare, that I have done assigned project alone led by supervisor. I used only literature, that is listed in work. In Prague 10. 5. 2020

.....

Abstrakt / Abstract

Tento projekt sa zaoberá analýzou súčasného a návrhom nového webového redakčného a rezervačného systému pre projekt SHerna. Web je navrhnutý na základe potrieb používateľov aj administrátorov a zameriava sa na jednoduchosť pre používateľov a rozšíriteľnosť. Výsledkom tohto projektu je analýza súčasného riešenia, zhromaždenie požiadaviek od používateľov a navrhnutie a prevedenie krokov na implementáciu týchto požiadaviek v novom systéme.

Kľúčové slová: Webový systém, Redakčný systém, PHP, Laravel, SHerna, rezervácie

This project deals with analysing the current redaction and reservation system and designing a new one for project SHerna. The web is designed based on needs of both users and administrators, and focuses on simplicity for the user and extensibility. The result of this project is an analysis of the current solution, gathering a requirements from users and designing and performing steps to implement these requirements to in the new system.

Keywords: Web system, Content Management System, PHP, Laravel, SHerna, reservations

/ Contents

/ Figures

Chapter 1
Introduction

Project SHerna is project from students living at Strahov dormitory to their fellow students. While having 2 rooms equipped with all manners of things that helps people relax while enjoying playing games, like VR, but also old DOS games, and organising events and tournaments, people will not use our services if the information is hard to find, or if they are hard to use. Therefore, having a nice, easy web service is crucial. However, our currents system is outdated and after people who built it left, it is also almost unmanageable.

In short, the purpose of the web for users is to inform students about our services and inventories, announce news like acquiring new equipment or announcing tournaments, and enabling them to reserve one of the two fully equipped game rooms for their use. From the administrative side, the purpose is to have easy and intuitive interface to manage the whole web site, administration, reservations and redaction system to create and edit articles.

In this project, my primary goal is to address these issues. Firstly, I will have to analyse current solution and its design, write up all the problems we as administrators and also users face. Secondly, gather up requirements about functionality that is critical and is missing right now, but also about things that would be useful and that would help to ease the usability of our system. Lastly, design and implement all the requirements from the previous step.

As I will address later in this thesis, the current solution is not made out to be scalable, and the architecture and implementation requires complete makeover. Therefore, in this project, I will redesign the whole architecture and implementation from the scratch with all the requirements in mind.

This thesis will consist of two main parts. First part will deal the current implementations and its issues with throughout analysis and breakdown the problems of it. In the second part, I will address those issues with my own implementation. In this part, I will talk about how I managed to avoid the same problems our current system has, and I will discuss the new architecture and implementation.

Chapter 2
Analysis

In this chapter, I will be looking into the current implementation of the web system. I will discuss technologies used but also I will address the primary goals our system is trying to achieve and how are they being fulfilled. I will also briefly discuss target audience.

2.1 Target audience

As of the definition, I would like to split the targeted audience into two. The first would be the users themselves and second group would be the people administrating our services.

Firstly I would define the users. The user of the application is a user between 18 - 25 years old student living at Strahov dormitory that likes to play video games. The definition would be used during feature design and testing of the application. The age range has been chosen like this because users in this range are most likely to live at Strahov dormitory.

The second group as mentioned before are the administrators. There are few reasons behind this. First of all the application could simplify the process for managing reservations and all information about them and thanks to this some problems and misunderstandings could be avoided. The last reason is that it would propagate their events and services, as the users that are using it is mostly interested in the video games themselves. With this in mind, I can analyse current workflows and in my new design try to tackle the problems they are now facing.

2.2 Technology used

Web systems have two parts, that are separate but tightly coupled. There is the backend side and frontend side of the system. I will now address both of them and in the end I will discuss how they are working together. First, let me define what backend and frontend is.

The backend part of the application, sometimes called `the server-side` is basically how the site works, updates, and changes. This refers to everything the user cannot see in the browser, like databases and servers. On backend side, we manipulate and store all the data, like user profiles, images, blog posts, etc.

The front-end is everything involved with what the user sees and interacts, including design.

For our backend we use PHP as its main programming language. We are using PHP in old version 5.6.4. The database with which we are working is MySQL.

Frontend, so everything a user sees, is made up from HTML and CSS. For better ease of use we are using Bootstrap[1], version 3. Bootstrap is an open source toolkit for developing with HTML, CSS, and JS, that helps you design websites faster and easier.

[1] `https://getbootstrap.com/`

For joining both side of the system we are using PHP web application framework, Laravel. From their official documentation: „Laravel is a web application framework with expressive, elegant syntax. ... Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.“[1] It means, that we do not have to tackle every single problem that can come up during developing bigger systems and their architecture.

Laravel architecture is based on the MVC (Model — View — Controller) : „The Model-View-Controller design pattern is simply stated: 1. Data model objects encapsulate information. 2. View objects display information to the user. 3. Controller objects implement actions. 4. View objects observe data model objects and update their display whenever it changes. 5. View objects gather user input and pass it to a controller object that performs the action.“[2]programming paradigm which forces separation of concerns of application. We can say that Model represents backend side of the system (all the data we work with) and View represents frontend side (everything users see). The Controller is what is mediating cooperation between the Model and the View, and therefore, it is joining both our sides of the system.

As I said, Laravel is PHP Framework, therefore it is using special template for Views, called Blade, „the simple, yet powerful templating engine provided with Laravel. Unlike other popular PHP templating engines, Blade does not restrict you from using plain PHP code in your views.“[3]. This way, our frontend consist not only of HTML, CSS and JavaScript, but using Laravel's templating engine we can use PHP code and special directives to address the data we get from our backend.

2.3 Goals

There are two primary functions of our system right now: 1. Informational — To inform students living at Strahov dormitory about our services 2. Reservation system — Service to reserve one of our rooms for chosen day and time The first goal is to have system that can not only communicate the information to users in eloquent and also elegant way, but also that can post visible updates and news in a way, that is visible and consistent with our social media. We want a blog-like feature, that would enable us to publish articles about various events, review of games, etc, and announce our events. Another goal is to have interactive inventory of our equipment and hopefully connect to gaming consoles to show data. For this, we need system that will enable us to easily maintain and manage all the information. The second goal is to make our services easily available to everyone who wants to use them and consistent for both of our game rooms. This consists of making reservations for one of our game rooms. Also, as administrators, we want to maintain all the reservation, and be notified in time of all and every change that we need to address. Therefore, we need a nice and intuitive administrator interface.

2.4 Conclusion

This chapter aims to introduce our current implementation and explaining what the goals of our system are. I will build on this in the later chapters.

Chapter 3
Current Problems

In this chapter, I will be looking into the problems of our current implementation of the web system. I will discuss the technical side, but also the way our system fulfils goals we established in the previous chapter. I will be considering problem from two sides, with customers and with administrators in mind. Firstly, I will look at the problem we as administrators have right now.

3.1 Access to code and documentation

The first problem came up as I tried to get access to the code and to the machine on which the server is running. This project was developed more than 3 years ago, and after 1 year of maintaining it, the people who worked on it did not have time to continue the maintenance and nothing was done for some period of time. It started as a simple side project, and nobody bothered with any documentation or readability of code. Therefore, after the handful of people who had knowledge about this project left, there was no easy way to find all the parts needed to even locally run it. In the end, I found out that this whole project was in a private github repository, so only two people have access to it.

Second problem arose after I finally got the github repository and wanted to get familiar with the code. As the project is using Laravel framework, the architecture was the same as for every Laravel project, and everyone can understand the basic logic behind it. But after digging just bit deeper, you will realise the lack of documentation. Controllers, which should have only one primary responsibility, are managing multiple pages and models, most of the time absolutely unrelated to each other. Some of the views and their forms where all managed by one big Javascript file, sending data using AJAX even though it was unnecessary. For this kind of bigger project I would expect at least generated PHPDoc for public methods, but there is none. Furthermore, there are almost zero comments in the code, which makes it hard for people to understand what is going on. Without documentation or commented code, it is easy to get lost in ambiguous calls and methods.

3.2 Redaction system

One of the biggest problems for us as administrators is the redaction system. Right now, the only way to add a new page with content, we need to manually add a title to the database, then add to database also page content, and after this, we need to update code in the corresponding view, to layout navigation bar view and also into the controller. After this, we can use the web interface to edit the content and make it public. The biggest problem with all this is that no one from the administrators has access to the code and to the server where the web is running. The only one with that access left the project more than a year ago, and nobody requested the access yet.

However, even if we do everything I mentioned, there are only two possible subcategories of articles.

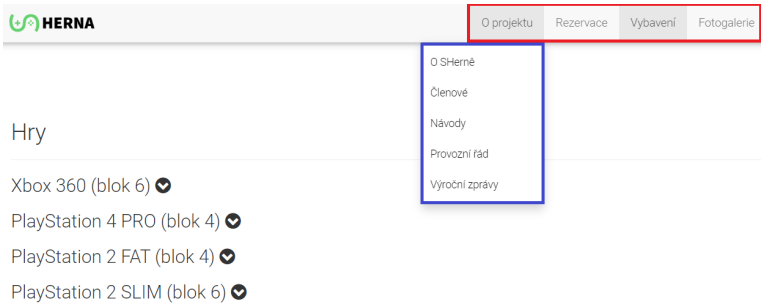

Figure 3.1. (red) navigation bar, (blue) About submenu

Firstly, you can add another article to the navigation bar. But this is not ideal, as it violates the hierarchy of information, as well as you will soon run out of place on the screen. Second option is to add new article to the subsection of `O projektu` (About). However, this does not make sense for the majority of articles, that are thematically different. It would only confuse the viewer, as he would not expect the information to be placed there. The resolution of this problem would be to edit the whole layout view, and all the corresponding views with hardcoded values of the navigation bar and its pages and subpages.

The problem is not only creating and placing of new pages. Right now, every article is treated and considered as a static page that is displaying important information about the project. However, this goes against the reason why we wanted redaction system. It was supposed to provide blog-like function — creating not `pages`, but `articles`. Semantic difference aside, we wanted to be able to create new articles, sort and place them under categories, and search through all articles. Also, as part of this, it would be nice for users to add comments. This cannot be supported with the current implementation, as we cannot add any metadata to the pages.

3.3 Reservation system

As I said in previous chapters, we have two equipped game rooms. For both you can make reservations using our web, by picking a date and a time, and then you will have access for the chosen time into one of the rooms. These uses card readers that are installed to the doors, so it will open only for the one who made the reservation. However, in one of the rooms the reader has been broken for more than 2 years. Therefore, to reserve that room, you must also write an email so someone can give you key to the room. Since the reader never worked in that room, it is inconvenient that you cannot actually reserve it via our service. It would make sense to have the reservation system work for both rooms the same, but for the one without card reader, it would automatically email available administrator. This way the user interacts with our system the same way, and the difference will be in the background, unknown to him.

Users face another obstacle when trying to reserve a time slot. In one of our rooms, we have Virtual Reality with many games set up. However, to use this, you need to have an initial 'How to operate VR' training, and only those that have the training can request access to VR, which is locked in the room. Right now, there is no direct way of requesting the key however, requiring the user to write another email requesting the access.

Even after successfully creating a reservation, the problems for users do not end there. If the user wants to prolong his reservation, the room is available he cannot do it. The only way would be to cancel the actual reservation, and create a new one.

Another problem appears when we look at the reservation from the administrator's side. The normal user can have only 1 active reservation with maximal duration that is globally set up by the administrators. However, as active members of the project we have the right to have more reservations, and also exceeding the time limit. Sometimes, it is even required. When we organise an event in one of the locations, we must create a reservation that will last minimally for the whole day, so we can make preparations. Right now, the only way to do this is to navigate to administration section, and create a new reservation there. That is not only unnecessary time consuming, you do not have a clear view of when are the current reservations, and you need to make sure you do not create a reservation that overlaps with another one. This is one of the biggest pain points for us, the administrators, as we are using our game rooms periodically multiple times a week, and having to always go to the not that much user friendly view, that is created mostly for managing and editing the reservations, can be quite frustrating.

There is one last, but not least, problem with reservations, and that is the fact that it is maintained and served by javascript. I will address this issue in a separate section 3.6.

3.4 Testing

One of the biggest problems is with testing. Better said, lack of it. As this project was not small at all, without proper testing, any change could break the application. There are no units tests for any part of the application, so every bug that is detected is only thanks to reports of our users. Every change is potentially a breaking one. Changing or adding a new feature would always require throughout control of the system functions manually, to see if the change did not break anything. This would prolong any kind of bigger system modification almost indefinitely.

3.5 Architecture

Current implementation and architecture matches the kind of project this was at the beginning. We need to understand that it was a small project that was supposed to help SHerna to gain a bit of attention as it was just starting. The main goal at the time was to have a nice eye-pleasing web that would attract users and let them use our services, and to have it up and running as soon as possible.

Therefore, by using Laravel framework and its predefined foundation, they achieved the separation of concerns. It was okay until this point. They followed good practices of MVC of creating controller for every page and its view in an Administration section. However, for the client side of the app, there is only one ClientController, that has huge amount of responsibilities, servicing request from all pages that user can navigate to. It shows every view, that user has access to, but also it create and manage reservations, login and authorisation services that could have been created by its own specialised class or factory.

```
/**
 * @return array
 */
private function getISService()
```

```
{
/**
 * Create a new instance of the URI class with the current URI
 */
$uriFactory = new UriFactory();
$currentUri = $uriFactory->createFromSuperGlobalArray($_SERVER);
$currentUri->setQuery('');

// Setup the credentials for the requests
$credentials = new Credentials(
env('IS_OAUTH_ID'), //Application ID
env('IS_OAUTH_SECRET'), // SECRET
action('Client\ClientController@oAuthCallback') //callback url
);

// Session storage
$storage = new Session();

// Instantiate the service using the credentials, http client and storage
$serviceFactory = new ServiceFactory();
$service = $serviceFactory->createService('IS', $credentials, $storage);

return [$currentUri, $service];
}
```

This is also true for views, where sometimes there is twice as much php code as it has mark-up language. As an example, there is a view, where 3 different views are situated, each one with it is own logic, and there is a simple if that determines which one should be shown. This way, the whole solution is extremely unintuitive and disarranged, where even trying to understand the working of one method can take you on a journey throughout a whole system. The current implementation does not follow SOLID[4]. That is a problem, as right now, the whole solutions uses almost no abstraction, and because of that, the code is not scalable and repairable.[5]The problem behind this is that models in this current solutions are only `java beans-like`. They consists of only properties and Laravel's default CRUD operations. This means that all the logic is left to controllers and sometimes even views.

Also, because of this, throughout the project, there is a lot of parts that violate the DRY rule and the code is hard to read and understand. For example, you can see a lot of reservations querying in controllers:

```
$reservationExist = Reservation::whereNull('canceled_at')
->where('location_id', '=', $request->location)
->where(function ( $q ) use ( $request, $startTime, $endTime ) {
$q->where(function ( $query ) use ( $request, $startTime, $endTime ) {
$query->where('end', '>', $startTime)
->where('start', '<', $startTime);
})->orWhere(function ( $query ) use ( $request, $startTime, $endTime ) {
$query->where('end', '>', $endTime)
->where('start', '<', $endTime);
})->orWhere(function ( $query ) use ( $request, $startTime, $endTime ) {
$query->where('end', '>=', $endTime)
->where('start', '<=', $startTime);
})->orWhere(function ( $query ) use ( $request, $startTime, $endTime ) {
```

```
$query->where('end', '<=', $endTime)
->where('start', '>=', $startTime);
});
})
->exists();

$parallelReservationExist = Reservation::whereNull('canceled_at')
->where('location_id', '!=', $request->location)
->where('tenant_uid', '=', $request->userUID)
->where(function ( $q ) use ( $request, $startTime, $endTime ) {
$q->where(function ( $query ) use ( $request, $startTime, $endTime ) {
$query->where('end', '>', $startTime)
->where('start', '<', $startTime);
})->orWhere(function ( $query ) use ( $request, $startTime, $endTime ) {
$query->where('end', '>', $endTime)
->where('start', '<', $endTime);
})->orWhere(function ( $query ) use ( $request, $startTime, $endTime ) {
$query->where('end', '>=', $endTime)
->where('start', '<=', $startTime);
})->orWhere(function ( $query ) use ( $request, $startTime, $endTime ) {
$query->where('end', '<=', $endTime)
->where('start', '>=', $startTime);
});
})
->exists();
```

By using models without any logic except the basic one provided by Laravel, you are forced to repeat the same code many times. This is a repeating occurrence in the project. By splitting some logic to functions, it would follow the DRY rule, but it would also make code mode understandable, with just a bit of documentation. Right now, you need to go line by line to understand what exactly is going on, because, as I stated earlier, there are no comments throughout the code.

The whole point of separation of concerns that the MVC architecture brings is laid to waste with some of uses in the current solution. Let's take the whole ClientController as an example:

```
/**
 * ClientController constructor.
 */
public function __construct(){}
public function index(){}
public function show( $code ){}
public function getAuthorize(){}
public function getLogout(){}
/**
 * @param $result
 */
private function controlLoginUser( $result ){}
/**
 * @return array
 */
private function getISService(){}
public function changeLang( $langCode ){}
public function oAuthCallback(){}
```

```
public function postUserData( Request $request ){}
public function postCreateEvent( Request $request ){}
public function postUpdateEvent( Request $request ){}
public function postDeleteEvent( Request $request ){}
public function getDeleteEvent( $event ){}
public function postEvents( Request $request ){}
public function postConsoles( Request $request ){}
public function getReservations(){}
public function getReservationICS( $reservationID ){}
public function getBadges(){}
public function postEvent( Request $request ){}
```

This is the whole class, except implementation. No documentation and no comments either. Is it clear what the responsibility of this controller is? The name is ClientController, that tells us it is something on the client side. But what is it doing, exactly? What does the method 'show(code)' shows? Why is it doing authorisation? It even handles and manages reservations. In fact, everything the client can do, is handled in this one class. Does not matter whether is creating a new reservation, navigating to information pages, or reviewing his own profile, everything is in there.

This class demonstrates another problem I mentioned before. The lack of documentation. One way around it is usually by searching where in the code are the methods called. However, even this is not possible. As most of the methods are never mentioned in the PHP code again. This brings us to another problem, the javascript handling the business logic. This I will discuss in the next section.

3.6 Javascript

One of the biggest problems I faced when analysing and trying to get grasp of the current solution was the fact that most of the client side is handled by javascript. It is used to initiate all the special widgets, which is understandable, as there is often no other way around it. However, it would help to have the functionality broke down to a functions at least, better yet to have separate javascript files based on the responsibility of the code. This was not the case. The javascript was in the end compressed into one big file. That alone would not be bad, if not for the fact that even the base javascript was all over the place, with some of the code never used, some declared multiple times or even copy pasted to another file. It was used to create a request using AJAX, instead of simply submitting it by the form in blade view. Because of this, many methods are never used in the PHP code, only called via AJAX, therefore making the understanding what action is called when more difficult, as I mentioned earlier.

3.7 Conclusion

This chapter aims to sum up the biggest and the most obvious errors and problems we are currently facing. Some of it will be addressed in chapters with requirements that I collected from people associated with the project and also from the users. As anyone can see, there are numerous features that need update, but there are also problems that are limiting the continuous maintenance and development of the current solution and its upgrade. Therefore, the decision was made that I will create a new implementation of the system from scratch, using all the knowledge gained in school and at work to create a better solution that would address all the issues mentioned in this chapter and attempt to satisfy all the requirement which I will talk about in the next chapter.

Chapter 4
Requirements

In this chapter I will be presenting the list of requirements I collected from administrators and users.

4.1 Administrators requirements

As an administrator, I need to be able to create a new article, edit it and categorise it, because that way I can inform our viewers with a blog. (x)[1]

As an administrator, I need to be able to manage and modify comments under blog articles, because that way I can protect our users from potential hate speech. (x)

As an administrator, I need to be able to manage reservations, because that way I have a full control over our services.

As an administrator, I need to be able to make reservation that exceeds time limit and other settings, because that way I can use game rooms for special events.

As an administrator, I need to be able to change limits and settings for reservations, because that way I can change it according to current situation.

As an administrator, I need to be able to manage locations and its statuses, because that way I can close or open reservations for our rooms.

As an administrator, I need to be able to close Location for a specified amount of time, because that will help us close the locations for events or maintenance. (x)

As an administrator, I need to be able to get notified if reservation contains request for VR, because that way I can provide them with key to a locked VR. (x)

As an administrator, I need to be able to edit our inventory, because that way I have full control over it and may update it based on the actual situation.

As an administrator, before any impactful edits, I need to be notified and asked for confirmation, because that way I am less prone to errors. (x)

As an administrator, I need to be able to manage navigation bar, create a new part or reorder it, because that way I have control over how the users see the website. (x)

As an administrator, I need to be able to assign roles to users, because that way, I can control what actions can users take. (x)

As an administrator, I need to be able to create tournaments with registration forms, because that way I can ease managing the event. (x)

As an administrator, I want to be able to create tournament brackets, because that will save time and allow me to control everything on one site. (x)

4.2 User requirements

As a user, I need the web to have nice, pleasing and intuitive design and user interface.

[1] (x) means not implemented in the current solution)

As a user, I need to be able to reserve game room for chosen date and time, so I can go there to play and relax.

As a user, I need to be able to reserve game room at Block 6 via web page (same as for Block 4), so I do not have to write them email. (x)

As a user, I need to be able to cancel my reservation, so I will not take up space for somebody else.

As a user, I need to be able to prolong my reservation -if it does not exceed time limit or collide with another reservation-, so I do not need to create another reservation. (x)

As a user, I need to be able to edit my reservation, so I can have control over it and do not need to delete it and create again. (x)

As a user, I need to be able to use the calendar widget to create and manage my reservations, co I can use intuitive way of creating even for specific times. (x)

As a user, I need to be able to request VR for my reservation, so I do not need to contact members of SHerna via mail to get keys for VR. (x)

As a user, I need to be able to comment on blog articles, so I can express my point of view. (x)

As a user, I need to be able to informed about new events organised by SHerna, so I can decide if I want to attend or not. (x)

As a user, I need to be able to find information about SHerna projects and its members, so I know who is responsible for what and what exactly is that project.

As a user, I need to be able to contact representatives of project SHerna, so I can ask them questions. (x)

As a user, I want to be able to be notified of new changes or articles via other channels, e.g. Facebook, Instagram, so I do not need to check only one site for information. (x)

4.3 Conclusion

In this chapter, I listed all the requirements I was able to collect, from users and the administrators of current system alike.

I have highlighted the requirements that are not yet implemented in the current system. As you can see, there are a lot of them, therefore a lot of features users and administrators are yet to be implemented. This only shows that the original purpose of the web was only to satisfy the most fundamental requirements.

Chapter 5
Application design

After analysing the current solution, it is clear that I must redesign the whole system, as the state in which it is right now is not fulfilling our requirements, and is not extendable. The reason for this are:

1. Code is hard to understand
2. Code doesn't follow good practices
3. Solutions in not easily expandable and modifiable
4. Project does not follow separations of concerns thoroughly, and because of this
5. Every new feature will only add to the difficulty of understanding the solution

The cost of using the current solution as a whole would be even higher than using some of its components and creating the whole design and system from scratch.

Therefore, in this chapter, I will discuss how I will proceed in designing the system from scratch.

5.1 Technology

For the technology, I will continue to use the Laravel framework even for the new solutions. There are few reasons that lead me to this decision:

1. Possible usage of some of the spare parts from the previous solution
2. Previous experience with web development using PHP
3. Opportunity to learn new and popular framework, that I found very interesting and intriguing during my analysis of the previous system
4. Experience of my supervisor with Laravel projects and technology

For all the reasons mentioned above, I will stick with the same technology, although in its most recent version, 7.x.x.

5.2 Architecture

As it is the main cornerstone of Laravel, the main architecture will again be MVC, as it is also the standard for such apps. The main goal of the new architecture will however be to be more scalable and modular, creating layers of services and reusable groups. I will try follow the separation of concerns, and have one controller for one specific responsibility, if possible, with logic for only one model. My first basic idea for an architecture was to have one central controller that will be used as main communication access point, selecting the right action and controller. The main reason behind this was that this way, the whole app and its logic would be connected, and it would help code understanding and expandability. However, this part of the application is already provided by Laravel, although a little bit differently. Routes are stored in separate files, defining the url, parameters, request method, and the action that should be taken. For example, if I wanted to declare what will happen when I type url /login, it is done as easy as this:

```
Route::get('/login', 'Auth\LoginController@login')->name('login');
```

First parameter is the url, second is the action, and lastly I name the route 'login', and now I can reference it by this name anywhere in code. All in all, this one line tells us, that after calling url /login with GET method, LogincController's action login should be called. Here is the abstract sequence diagram capturing the abstraction of the flow:

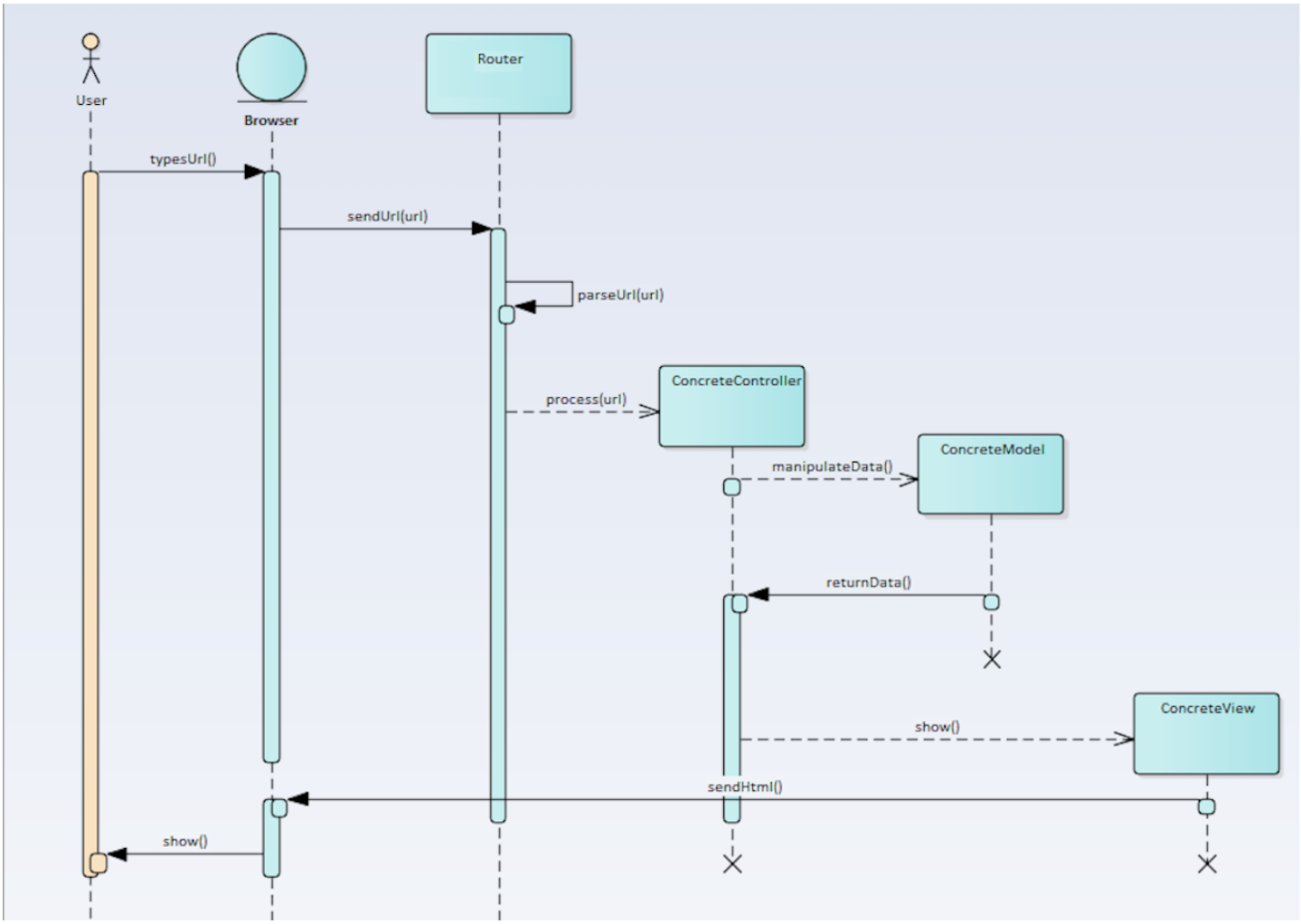

Figure 5.1. Sequence diagram of url processing

Another attempt to make the code easier to understand will be to create an easy-to-follow project structure, splitting the files into logical directories which makes it easier to find them when working on large projects. This will be the case for all three parts of MVC - splitting models, controllers and views files in the same logical structure. As our system will have 2 main parts, Administration and the Client side, both with multiple subparts, following structure suits us the best.

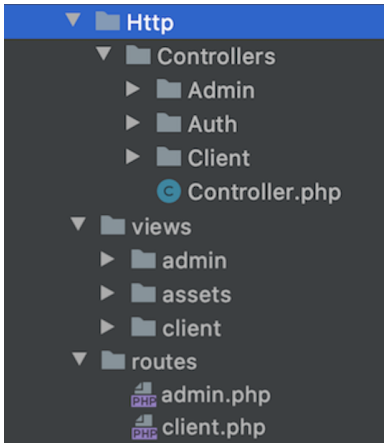

Figure 5.2. Project Structure

With the logical project structure I established above, it is very intuitive what code will be executed based on the user's action, and where to look in code. If the user viewed the Blog, the route he called is stored in routes/client.php, the controller named BlogController inside Controllers/Client folder was called, and it showed view from views/client/blog.

Another advantage is that this way, the adding of new functionality is well documented and repeatable process. After creating all the files needed for the new feature, you will just simply follow the example in the code, and add the specific route.

For models, I will make sure that they not only have java beans-like properties and defined basic CRUD operations, but also that some of the logic is happening in this layer, in order to follow separation of concerns. For models that are handled by multiple Controllers, e.g. Reservation that can be created via both Admin and Client sides, I will create a service layer where the logic will take place.

In a very simple and basic MVC workflow, when a user interacts with our application, the steps in the following image are performed.

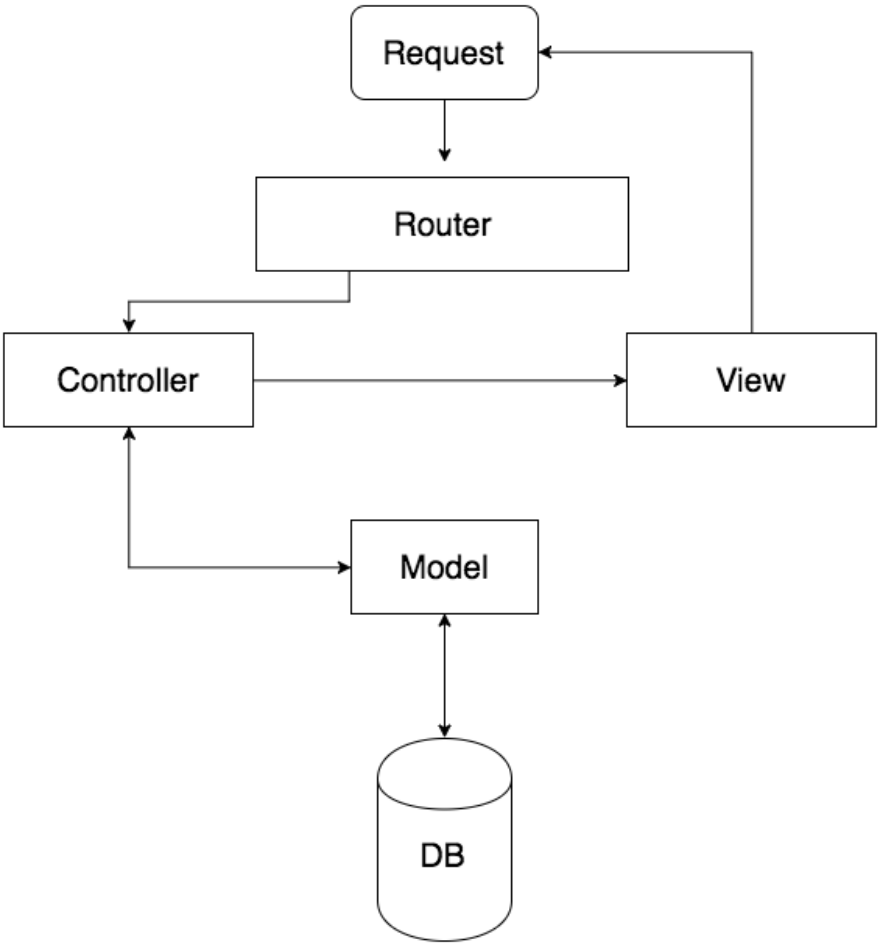

Figure 5.3. MVC workflow [6]

5.3 Development process

During the development process, I shall focus on creating comprehensive documentation, so in addition to understandable architecture, future developers that may come after me will have an easy time adapting to the system.

I will try to follow agile style with its sprints, in order to focus on one specific feature at time.This will in turn encourages simple designs and it will incrementally improve the Minimum viable product(MVP), as „Unfortunately adding features doesn't necessarily improve the business case. It may take longer, make the product less usable, and carry more risk...We define MVP as that unique product that maximizes return on risk for both the vendor and the customer... The MVP solves a variety of problems, especially on a product's first release. Products without required features fail at sunrise but products with too many features cut return and increase risk for both vendor and customer.“[7], as instead of wasting resources in a feature that no one would use, it is crucial to focus on the key features that contribute to the product. Furthermore, with components covered by tests, every future change could be easily tested and it would be easy to recognise that change broke something it the system.

With this comes cyclic incrementing development. This style will also benefit from my closeness with the project users. By being in contact with them during the whole process, I can spot wrong approaches right away and fix it in the next cycle.

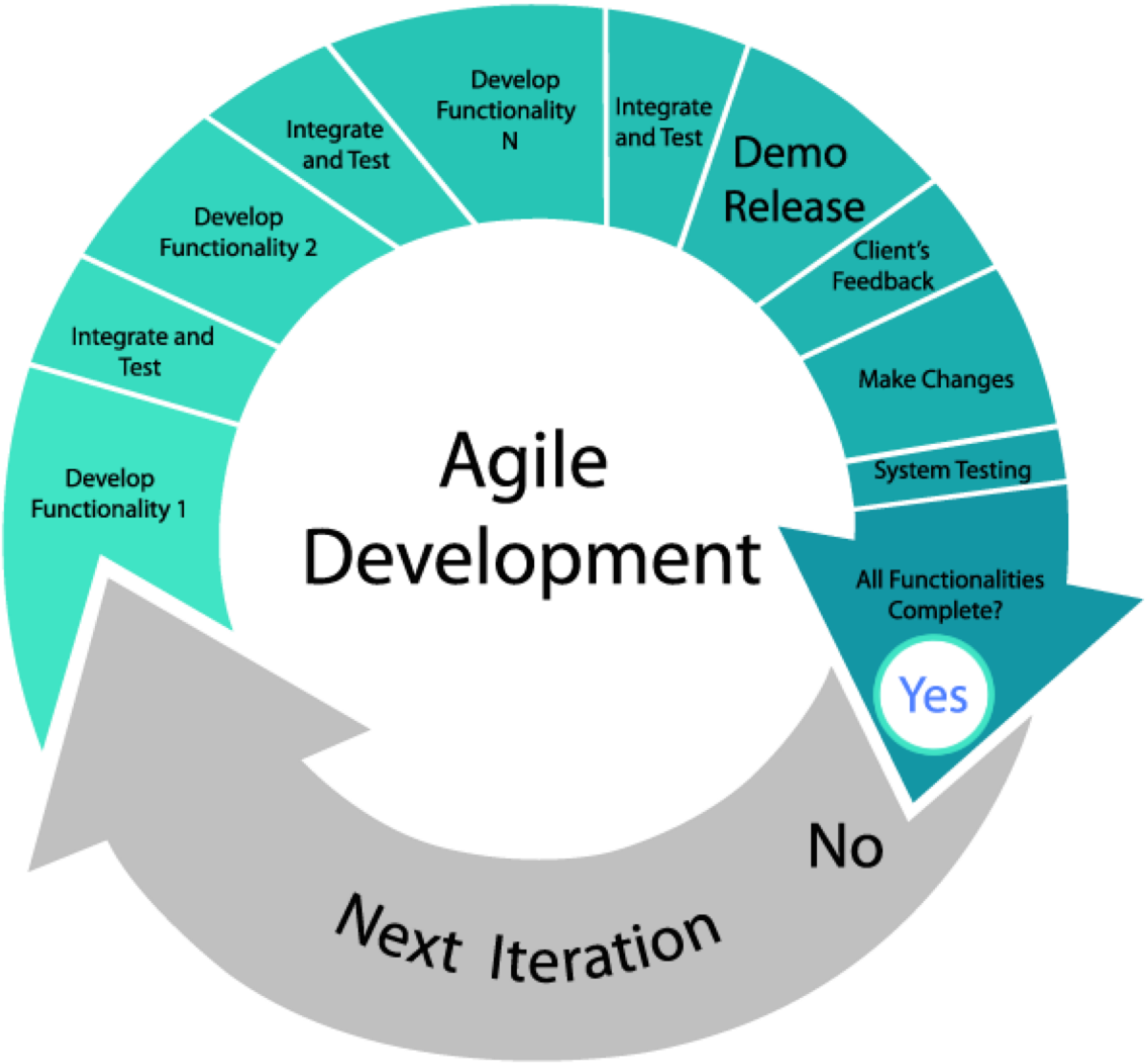

Figure 5.4. Life cycle of development[8]

5.4 Conclusion

In this chapter, I aimed to outline the architecture and the development process alongside providing the reasons for my choices. In the next chapter, I will build onto this and I will talk about the whole implementation, problems I have encountered and how did I resolve them and the problems of the previous solution.

Chapter 6
Implementation

In this part, I would like to go through a high-level overview of the implementation of the front-end and the back-end of the application.

6.1 Front-end

For the front-end, the idea was to use the parallel programming that MVC enables us to do, with me dealing with the architecture and the implementation, and our graphic creating a new design for the website. However, due to the covid pandemic, this was not possible. Therefore, I mainly used the same styles and design as the previous solution. As I stated multiple times before, our web consists of 2 main parts, administration, and the client section. I will briefly talk about these in the next few sections.

6.1.1 Administration

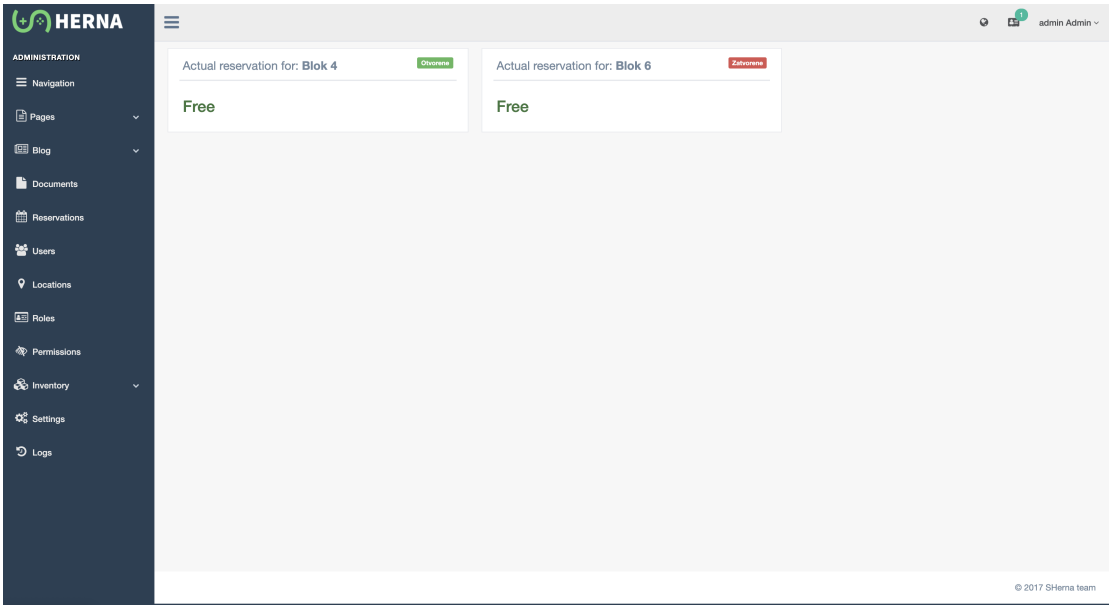

Figure 6.1. Administration

For the administration, the Gentelella admin template is used[1]. Navigation bar is located on the left side. Of course, the whole template is responsive thanks to Bootstrap. The purpose of the administration is to manage the whole web page, as well as all the data and settings, like creating a new location, adding a new game to an inventory, etc.

Even though the template is the same as it was in the previous solutions, multiple changes has been made. For starters, the template was updated to the latest version. Other changes came with the functionality change that I will mostly talk about in the

[1] `https://colorlibhq.github.io/gentelella/`

Back-end section 6.2. Here I will provide only small overview. The basic responsibility of every page in this section is to manage and handle CRUD operations. Therefore, there are 3 different views for every model that can be managed - Create, Edit, Index. Create and Edit views are almost the same, with only minor changes as some data can not be modified after creation. Their whole purpose is to put data in a database.

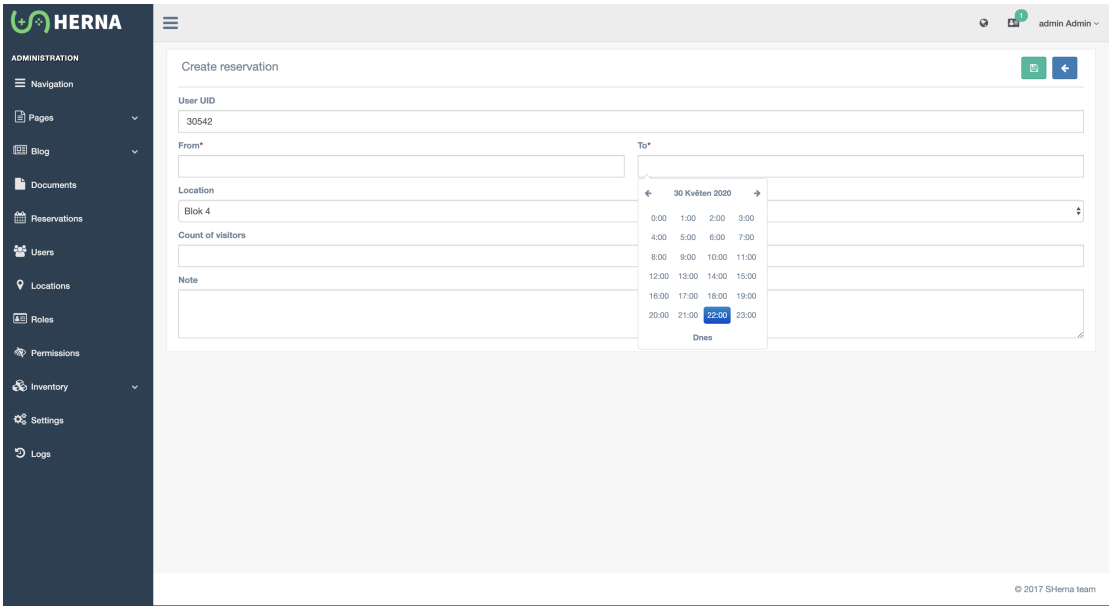

Figure 6.2. Create view

Index page, on the other hand, gathers all the data from the database, and shows overview of it. Due to possible numerous of data, it is paginated with the Laravel's build in solution. In the next screenshot, I manually set the number of instances shown per page to 5, in order to demonstrate it.

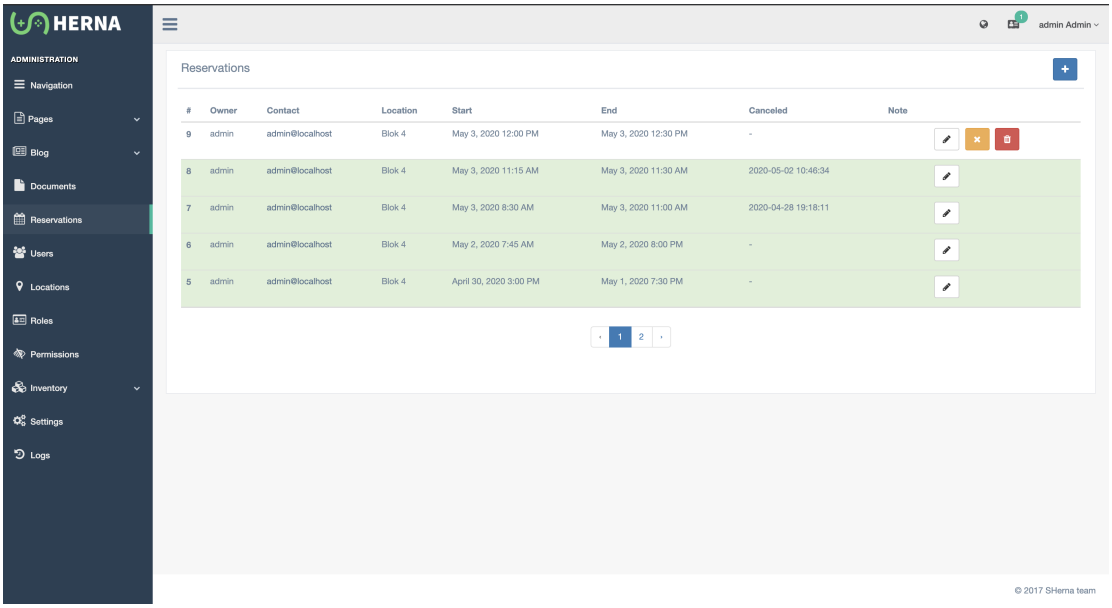

Figure 6.3. Index view

The most changes came with the new functionality. As I said before, the previous system lacked the complete and functioning content management system(CMS), there-

fore I had to create new one. More information will be provided in the back-end section 6.2. Example of using the CMS for blog articles:

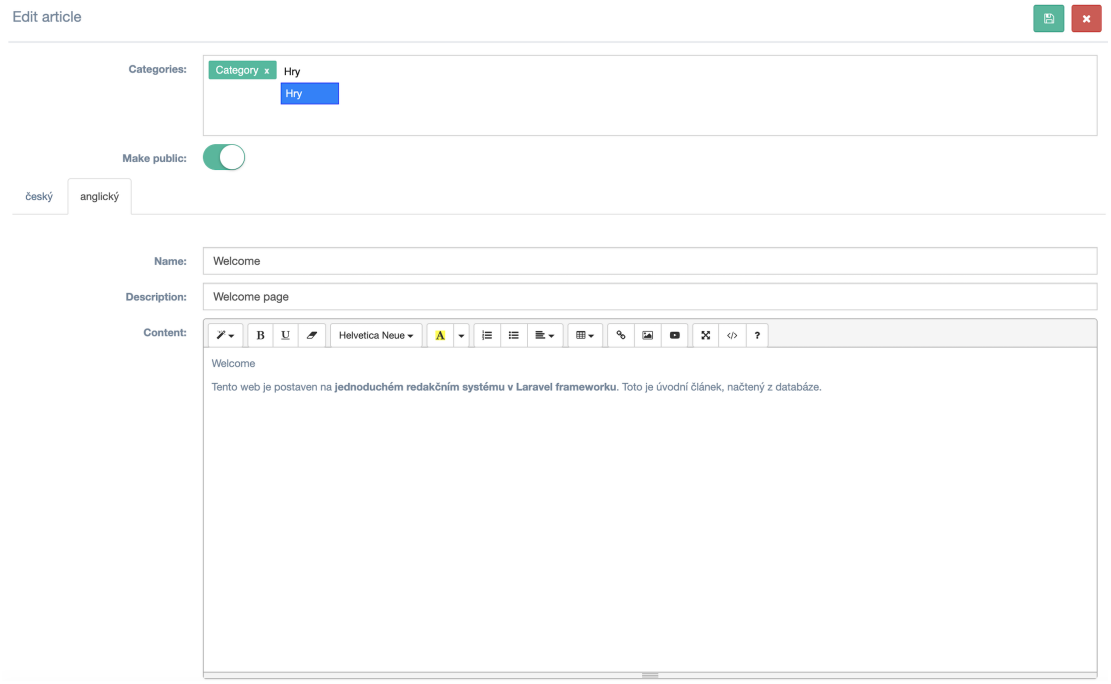

Figure 6.4. Content management system

Here you can see the edition of an article. Every article can have many categories, which are highlighted using the javascript widget Tags Input [1]. For this input, there is also an autocomplete configured, so users get suggestions to prefill already existing categories. If the category user typed does not yet exists, it will be created.

One of the biggest changes comes with the new possibility to manage the client navigation bar, create new parts for it and more. For that, I used drag and drop functionality in the index view to easily and intuitively reorder the navigation bar. You simply need to drag the row and select in what order you want the user to see the pages.

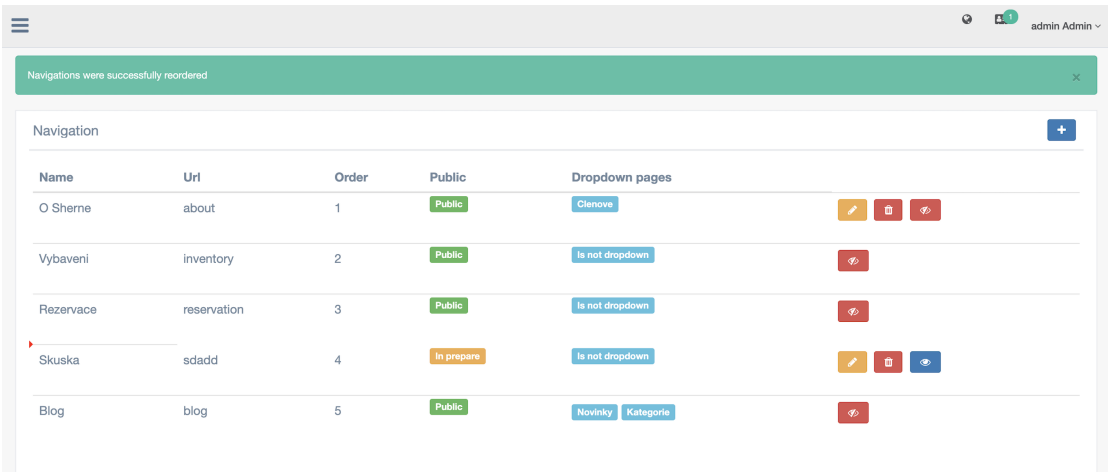

Figure 6.5. Reordering of navigation bar

[1] `http://xoxco.com/projects/code/tagsinput/`

Other parts of the administration look similar, following the unified layout, appearance and functionality.

6.1.2 Client

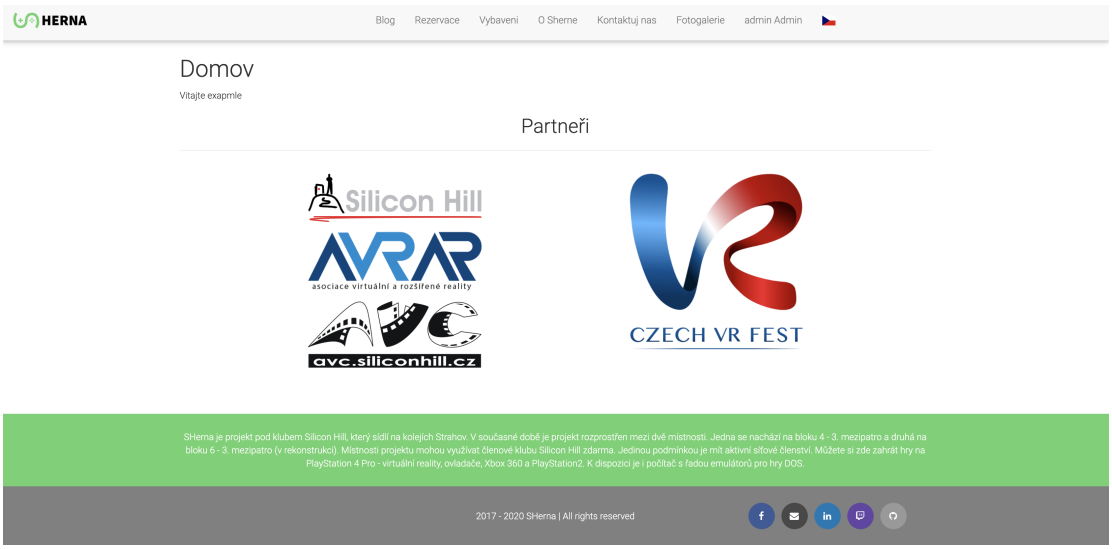

Figure 6.6. Client side

The first big change on the client side is the navigiation bar. As I talked about in the previous section, the navigation bar can be edited. Therefore, it is now loaded dynamically from the database, not statically hardcoded in the view as it was before.

For the clients side, there are 3 constants pages (excluding home page and simple contact us form): Reservation, Inventory and Blog with its articles. Other pages and subpages can be created and managed, but these 3 cover special functionality.

First, the most important one, reservations page. On this page, users can create and manage their reservations. For that, I am using the fullcalendar javascript widget[1] Users can use the control of the calendar to create, prolong, or completely move the reservation. This is a big improvement from the previous solution, where users had to manually fill the form to create a reservation, and there was no way to edit it. The user's own reservation is shown in green colour, all others are blue.

[1] `https://fullcalendar.io/`

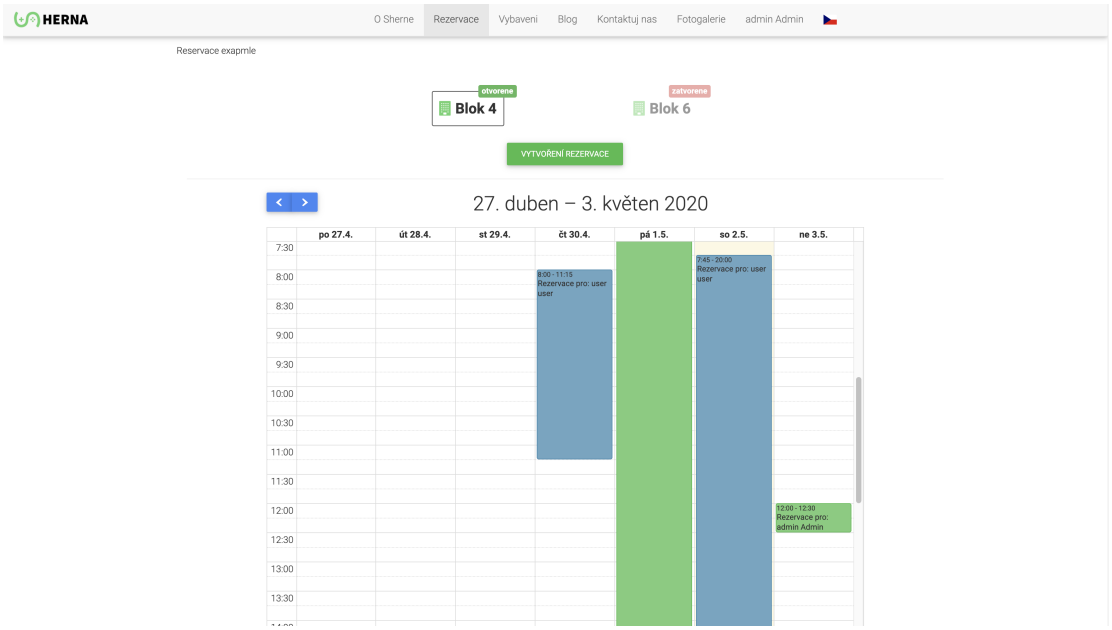

Figure 6.7. Reservations

The next constant page comes with the new Blog feature, The article created from the CMS can have comments, so that logged in users can share their opinion on the article, and have a discussion with each other, as they can even reply to another comments. These nested comments have predefined limit of nesting to 5, in order to avoid the comment to get smaller and smaller. Every article has an option to have comments disabled, and also, only logged in user can post comments. In the next iteration of work, edition of comments will be enabled, for users or administrators alike, but right now this feature is not coded yet. However, guests can view all the comments without the requirement to be logged in. The users icon that there is right now will be replaced by the users' images taken from the Information System.

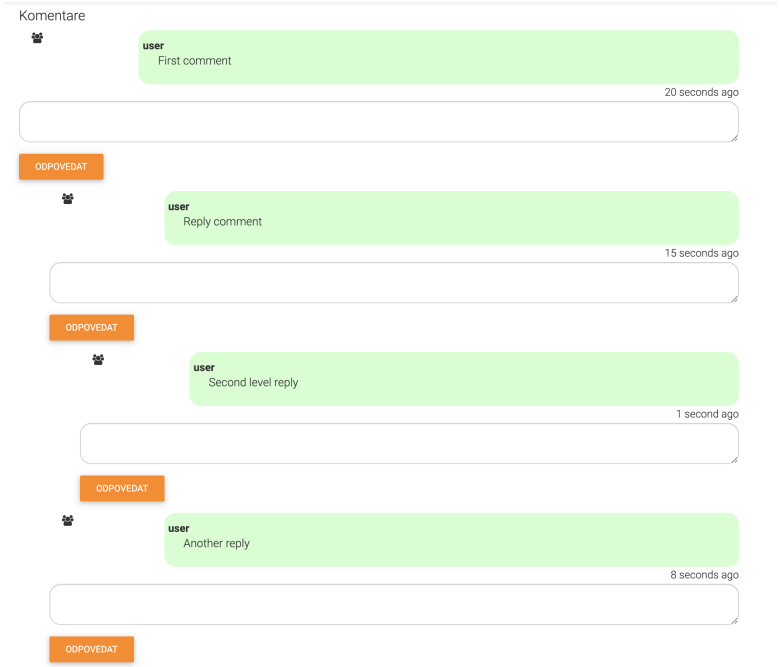

Figure 6.8. Comments

The inventory shows all the items that are located in the game rooms, sorted in the categories, and of course all the consoles and their games. The content of this page is gathered dynamically from the database with data managed in the administration section. Every location has its own games and inventory, and inventory is split into categories defined by administrators. For games, they are separated for each console in the location, and there are also additional information displayed. Right now it is shown only in its base form, as this page is ought to be redesigned by our graphic to have more pleasing and intuitive form.

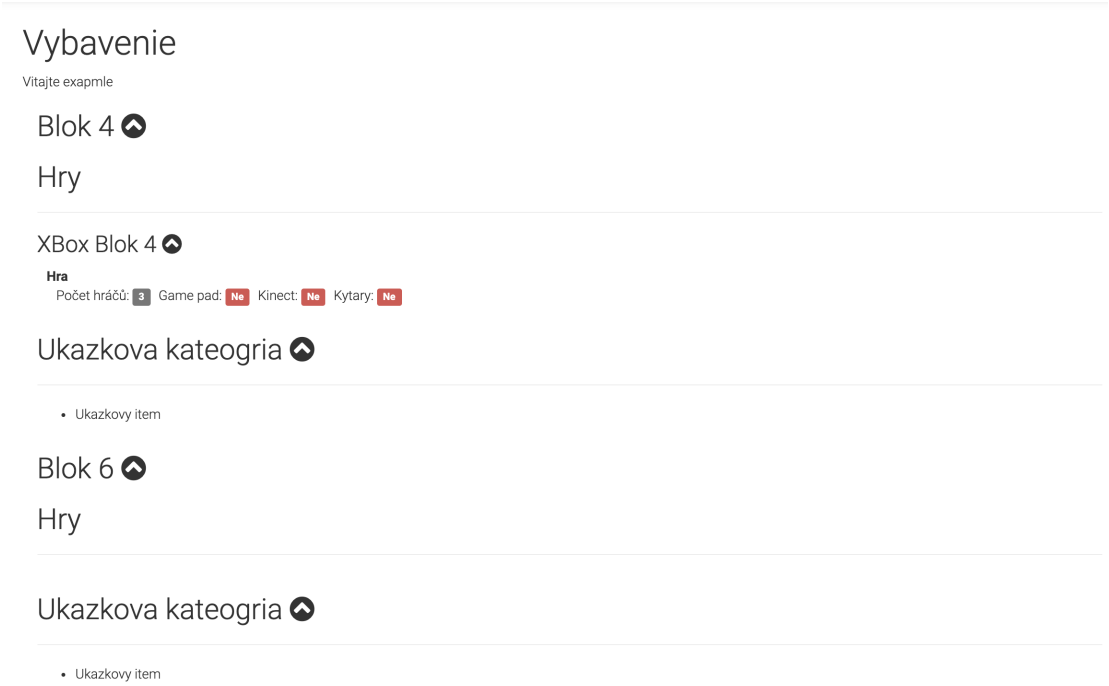

Figure 6.9. Inventory

Whole web is localised, with every article and page having content for every language supported (right now they are per requirements 2 languages supported - English, and Czech). Static messages and notifications also have translations. So after changing the language, everything will show the correct version.

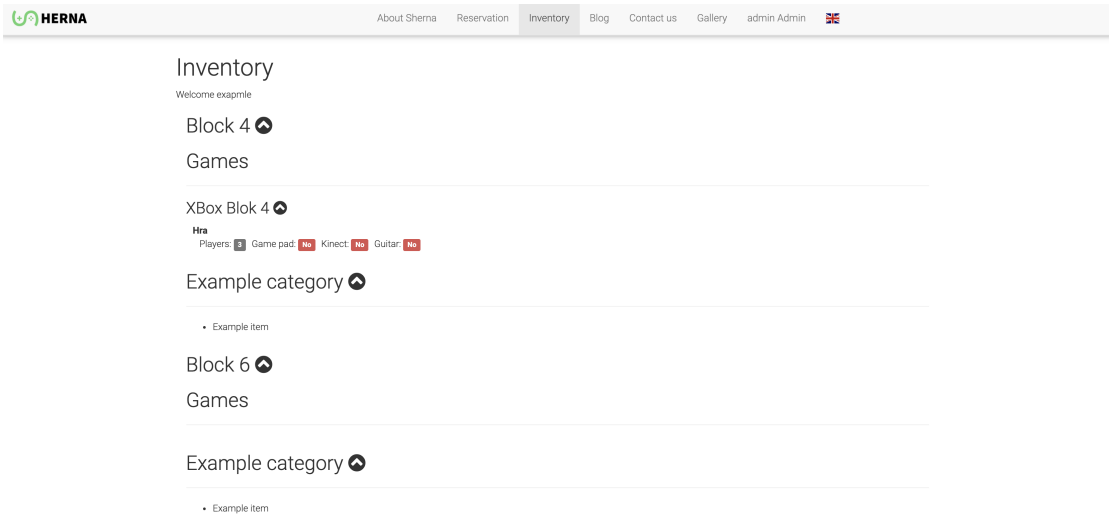

Figure 6.10. Localised Inventory

There are few more screens, however, there is no special behaviour in them, only showing the content. Although bear in mind, I am no graphic and was using just base styles and templates. Once the COVID pandemic will be through, it will enable us the meet and work more closely, as well as test the new User Interface the new design will for sure bring.

6.2 Back-end

The main difference between the previous system and this new one is in the back-end. As I started from scratch, I had to design every part of this. I followed the cornerstones of the design I mention in the Application Design section 5. For the administration side, it was easy. Every model has its own controller handling the index, edit and create view for data for CRUD operations. For the client side, it was a bit more difficult, considering my aim was to create a web that could be configured and managed almost entirely from the Administration section. In this section, I will start be describing the development process, then I will move and describe the database, and after that, I will touch upon the most interesting features and problems I encountered.

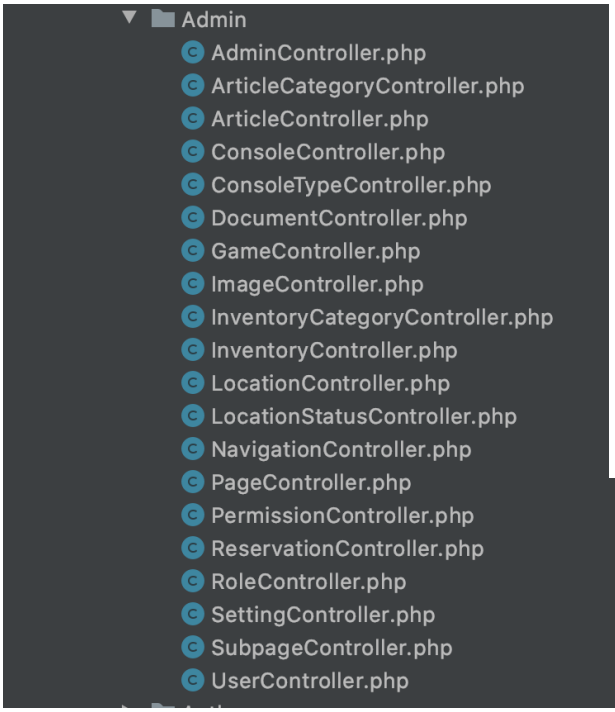

Figure 6.11. Admin controllers

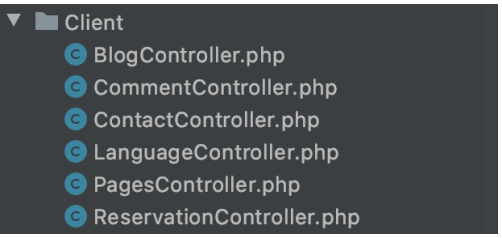

Figure 6.12. Client controllers

6.2.1 Development

As I stated before, I was trying to adopt some king of agile development, where I had sprints during which I worked on chosen tasks and created a MVP. For this, I created a dashboard with epics, stories, and tasks, using a Trello [1].

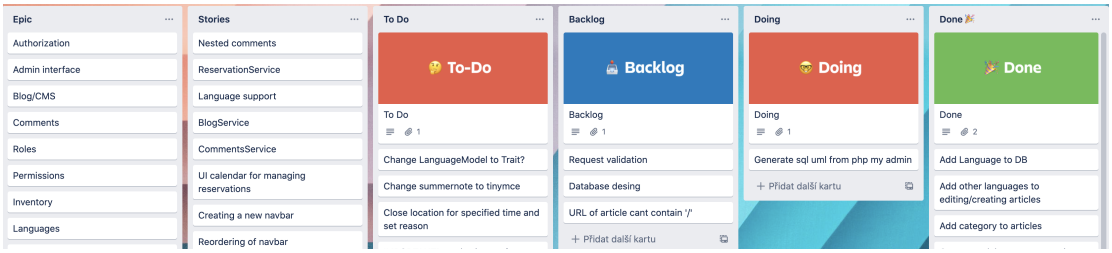

Figure 6.13. Trello dashboar

The next step to ease the process was to use GIT[2] as a versioning system. For every sprint, I selected one epic I wanted to implement, created tasks, and created a new branch in my repository. After the functionality was tested and working, I merged the branch. This way, I achieved easily revertible incrementing changes, therefore creating an MVP in a cyclic process.

[1] https://trello.com/
[2] https://github.com/SHernaSH/sherna-web-v2

6.2.2 Database design

The current implementation is using MySQL database, and I decided to stick with that for a few reasons:

1. Popularity - with large number of users and developers, it is bound to stay supported and at the top of the game for a long time

2. Ease to use - From installation to actual working with the database, MySQL is very intuitive

3. High performance - „MySQL is designed to meet even the most demanding applications while ensuring optimum speed, full-text indexes and unique memory caches for enhanced performance.“[9]

4. Costs - For our intended purposes, the MySQL database is free of charge

I chose to use the relational database(SQL), instead of non-relational (NoSQL), as it is easier to understand and use, due to its traditional status, and the experience I have with this kind of databases. Also, the main advantages that NoSQL provides, would not be used much in our use case: „NoSQL tends to be a better option for modern applications that have more complex, constantly changing data sets, requiring a flexible data model that doesn't need to be immediately defined. Most developers or organizations that prefer NoSQL databases, are attracted to the agile features that allow them to go to market faster, make updates faster. Unlike traditional, SQL based, relational databases, NoSQL databases can store and process data in real-time.“[10]

Laravel provides a powerful ORM implementation called Eloquent for work with databases. „The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding `Model` which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.“[11] Eloquent creates an abstract layer over the database, so it - in most cases- does not depend on the concrete database realisation, and it works with majority of currently available databases. All in all, every model has a corresponding table in the database, and has methods to query this table, automatically mapping retrieved results to the right object, or a collection of objects. Models are not self-contained. They have relationships that intertwine with each other, representing the 1:1, 1:N and M:N relations.

The main problem with designing the database was how to implement localisation. In order to resolve that, the languages table was created, and every table that needs localised data has a foreign key referencing it. Furthermore, localised models consists of 2 tables. One the main table with common information, and second table with information that is different for each language. For example, there is table articles with id, url and user that created it, and another table articlestexts with all the information in the associated language.

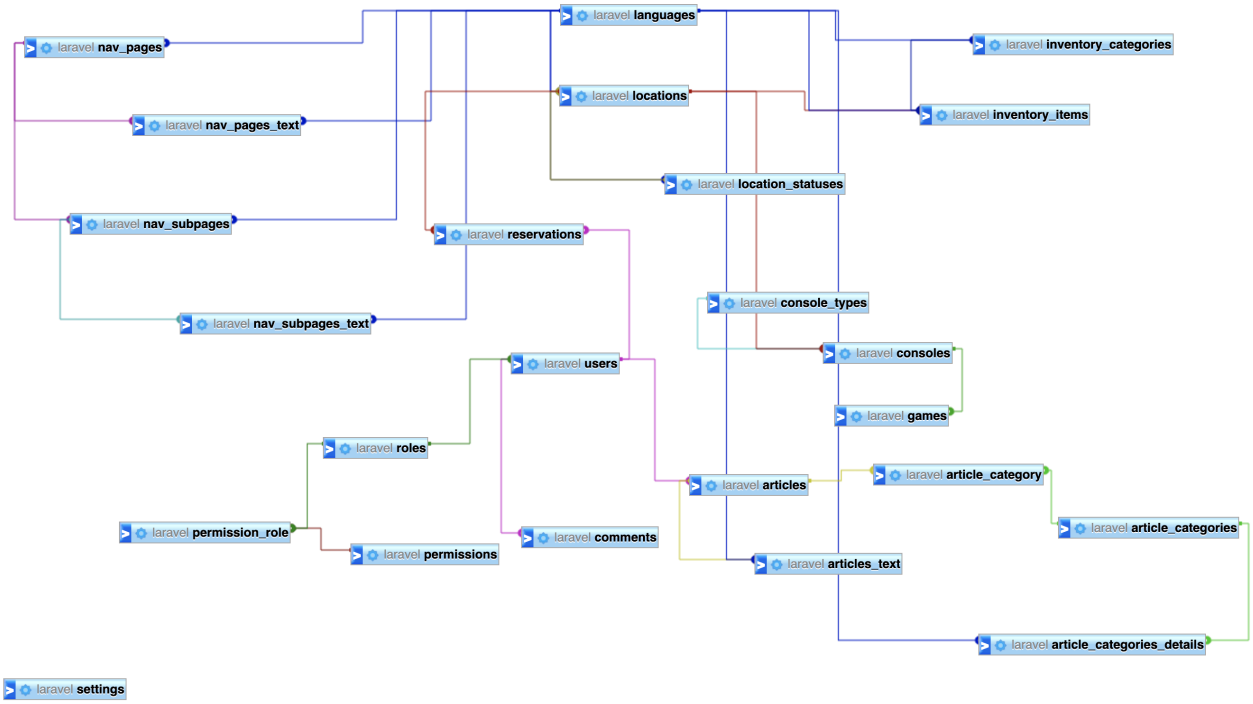

Figure 6.14. Database

6.2.3 Localisation in the code

The localisation had to be resolved not only in the database, but also in the code. For this reason, I created a new LocalisedModel with local scope, ensuring that every query will return only results with the current language. Therefore, when I wanted to make a model localised, all I had to do was to extend LanguageModel, instead of Model, and that was all. The handling of the model is the same as if it was not localised, as everything is resolved in the background.

6.2.4 Roles and permissions

The next obstacle appeared in a form of permissions. The previous solution just had in code check if the user is admin, and that was all for the roles. That is why I had to create a whole new system.

I used the build-in functionality of Laravel to generate all actions possible, and store them as a permission, assign a permission to a role and create a middleware that after every request is sent, checks whether the user's role has the permission to perform this action.

This way, we can dynamically create new roles, or edit the existing ones, based on our needs.

As every action is by default disabled for everyone, after creating a new Controller or action, there is a need to generate the new permission. By doing that, the permission is assigned to a super admin, that can then assign it to other roles. This way, by using a whitelist, instead of a blacklist, we can ensure no unauthorised person can access something he does not have permissions for.

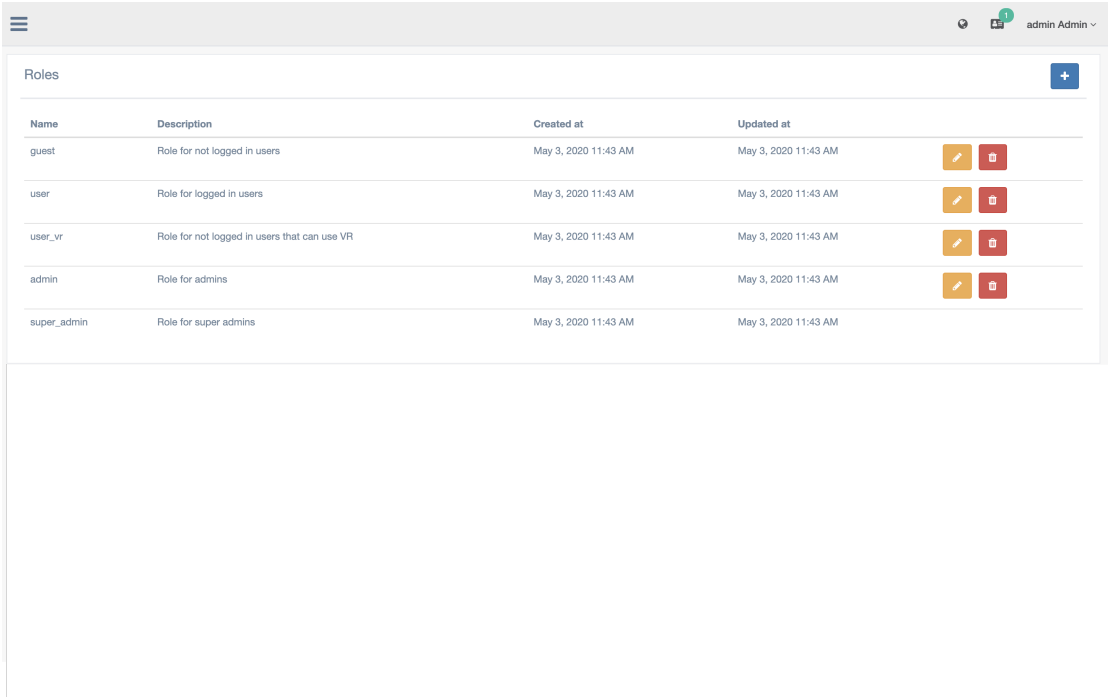

Figure 6.15. Roles

6.2.5 Navigation bar

Maybe the biggest setback I encountered was with a navigation bar. I wanted it to be completely modifiable though administration. To chose if the navigation will be a dropdown with subpages, or whether it would lead right the the page with the content. Change an order in which the navigation bar is showed, to edit the contents of all the pages. But there was a small problem. Some of the pages in the navigation have their own logic and their own controller to handle them. Even the way the content are shown is different. For example, in reservations page, there is a calendar for every location. In order to resolve this, I added a new attribute to the database, a special code, that determines whether the navigation page should be treated a bit differently.

For the reordering, as I mentioned, I used javascript that enables to simply and intuitively drag and reorder to navigation as user wishes.

However, the real problem that took a long time to crack was how to create a new navigation dropdown page. I wanted to enable user to create the page, and the subpages associated with it in the same page, as it is intuitive. But to create a new subpage, I must first create the parent page and save it into the database, which would mean the creation would have to have at least 2 steps. One to create and save the parent page, and afterwards create the subpages, and associate it with the parent.

I found the solution for that using the session variable. After creating a subpage, it is stored inside the session for the next redirect, refreshing everytime the edition of the page go on. After the whole page is saved, every subpage is consequently saved and associated with it.

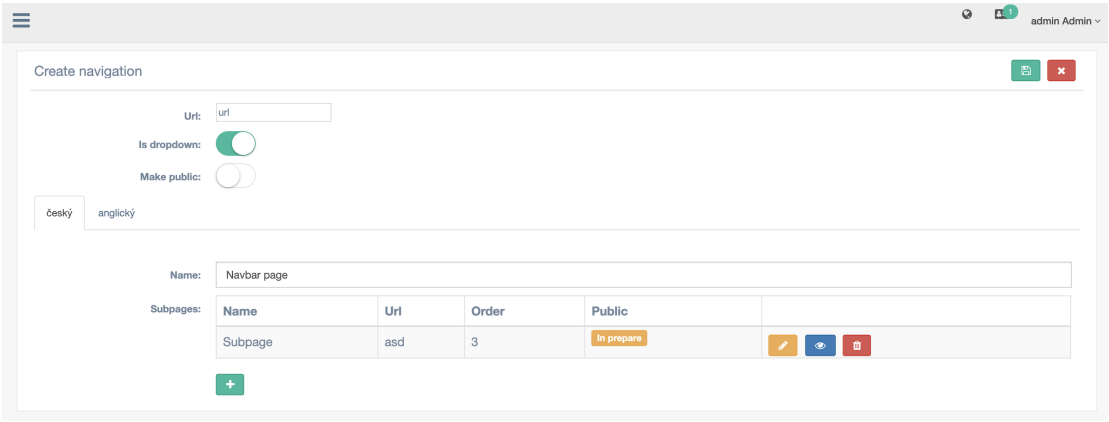

Figure 6.16. Creating navigation page

Even though the subpages are currently stored only inside the sessions, you can still manipulate them as if they were stored in the database. You can even reorder them the same way the navigation pages can be reordered.

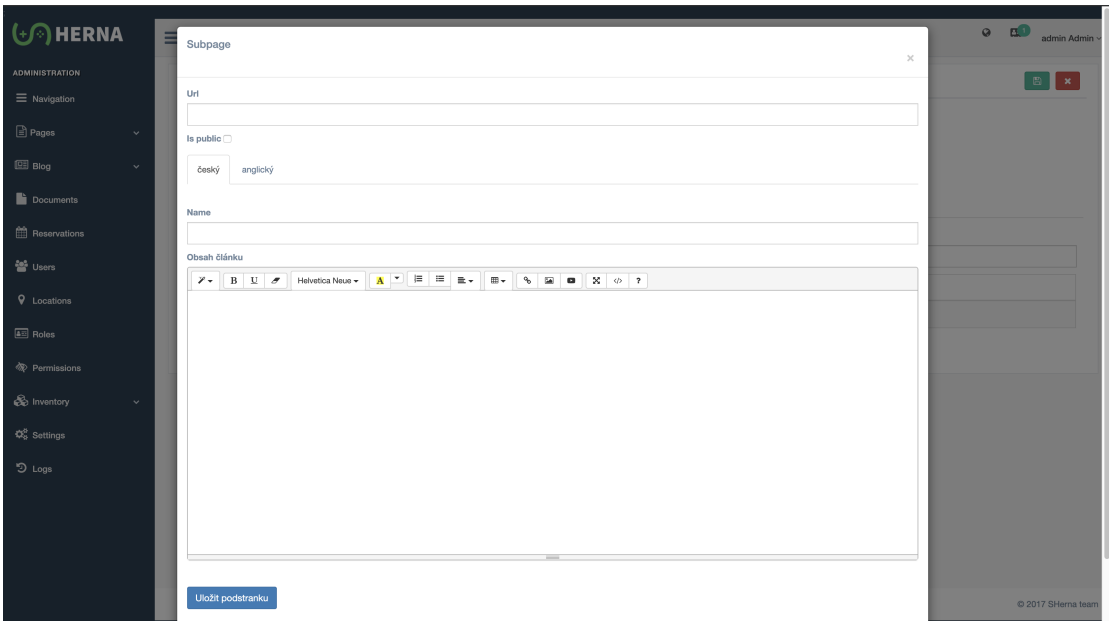

Figure 6.17. Creating navigation subpage

6.2.6 Services

Usually, one model is handled by one controller. However, sometimes the model has more controller associated with it, e.g. Reservation can be created both from administration and the client side. Therefore, for these cases (namely it is Reservation, Page, and User), I created a Service layer, in order to avoid repetition of code.

In both - Admin and Client - Reservations controllers, this is the only repetition of code, instead of having the whole setting and storing logic repeated in two places.

```
$validation = $this->reservationService
 ->makeReservation($request, Auth::user());
if (!is_string($validation)) {
    flash(trans('reservations.success_added'))->success();
} else {
    flash(trans($validation))->error();
}
```

6.2.7 Authentication

For the Authentication our application is connecting to the Information System(IS) of SiliconHill, via the OAuth2 Server Api. For that, I used PHPoAuthLib[1].

This way, our application does not need to store the passwords of our users, and can use the users of the IS, which makes sense, as it is also the requirement to use our Services, and it stores all the data the application needs. After the authentication, if the users is not in our database, we create him from the data from IS.

6.2.8 Notifications

Another new feature I added is the notification. Users and administrators are automatically notified - for only only per email, slack can be easily added' - when they create a reservation that depends on more than just using the reader on game room doors. For instance, everytime users request Virtual Reality, email is send to the administrators, to contact the user about providing him the key for the VR. Same goes when the users creates a reservation for a location that has access on key, both user and administrators are notified, as per requirement, for this enables the user to treat every location the same during creation of the location

These notification used the fact that the Users in Laravel automatically use Notifiable trait[2].

[1] https://github.com/Lusitanian/PHPoAuthLib
[2] https://laravel.com/docs/7.x/notifications

Chapter 7
Testing

As I stated previously, on of the biggest problems of the previous solution was the total lack of testing. My goal was to cover all and every functionality of the application with tests. For that, I used the LaravelShift[12] service, to generate the template test for every route and create unit test covering my custom request validation.

7.1 Tests

Laravel provides us with 2 distinctive test categories. Unit and Feature. For Unit test, I am testing the Form Validation and its rules for every form request, to validate the rules are set up correctly. This is paired with a feature that that asserts an action is using the correct Form Request validation. In doing so, it ensures everything is wired together as required by Laravel to perform the validation.

However, the most important are the feature tests that check whether the action, or in other words, the feature is working as intended. This is ensured by using the requests to send data, and after the action is performed, asserting the state of application is as desired. For example, after sending request index reservations, I want to assert that at the page I can see all the reservations with the correct data. Another example, after sending request to update the user role, the test needs to ensure the user has a correct role assigned after the action was performed.

Lets take the following test as an example:

```
public function store_returns_an_ok_response()
    {
        $user = factory(\App\Models\Users\User::class)->create();
        $status = Helpers::createLocationStatus();
        $data = [
            'location_uid' => $this->faker->word,
            'reader_uid' => $this->faker->word,
            'status' => $status->id,
        ];
        foreach (Language::all() as $language) {
            $data['name-' . $language->id] = $this->faker->name;}
        $response = $this->actingAs($user)->post(route('location.store'),
         $data);
        $response->assertStatus(302);
        $response->assertRedirect(route('location.index'));
        foreach (Language::all() as $language) {
            $this->assertDatabaseHas('locations', [
                'name' => $data['name-' . $language->id],
                'language_id' => $language->id,
                'status_id' => $status->id,]);
        }
    }
```

This test is asserting that the storing of Location is working as intended. Firstly, the test must create a new user and new location status - whether the location is opened/closed, etc.). Then, data is populated using Faker[1], so everytime, there u is new and unique data, not hardcoded values. Afterwards, request is sent with this data to create a new location. Then, the test asserts that response has correct status and is redirecting to the right page. In the next step, the test is asserting that the database contains the newly created location with correct data.

As of right now, the application has 162 passing test, with 24 more that are incomplete, and need more assertions to enable them - most of them are Form Assertions.

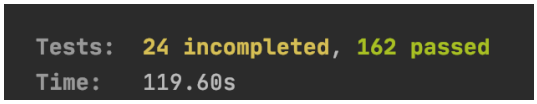

Figure 7.1. Tests passing

7.2 Conclusion

In conclusion, the current state of tests covers all functionality, making it easy for developers to spot and find the errors. Everytime the application is changed, or a feature is added, these test can help to ensure all the functionality is woking as intended, or, in case something is broken, quickly find the cause of the error.

This will ease the workflow of adding new features and changes, greatly increasing the scalability of the solution. More work is still needed, to tests wrong data, assert errors are shown, etc. But for know, I wanted to have a coverage of basic test for every action, to ensure everything runs smoothly.

[1] https://github.com/fzaninotto/Faker

Chapter 8
Future work and deployment

In this chapter, I would talk about the future work and features that will the application have during public release. The next part would be discussing deployment of the web on a server.

8.1 Future features

There are many features that I plan on adding before the web is released. One of them, of course, it the new design. For this, I plan on using the Vue.js[1] framework, which has a great integration with Laravel framework.

The next features will be to satisfy the requirements not yet implemented in my new solution. This includes a notification integration with Slack, and also with Instagram and Facebook. As our project is active on social sites, it would be nice to have our followers notified every time a new event is created, article published, etc.

For the integration, if it is possible, I would like to connect to our consoles and create a page showing the games we have, the gametime played, achievements achieved.

The next big feature would be to have a system created and ready to host events, register our users and create a tournament brackets for the participants. The reason for this is that we often host a tournaments in one of our competitive games, and right now the user registration is manual, they write their name on a paper, and we then create a tournament brackets using some of the online tools.

One more small feature based on requirements is to enable an option to closed location for a specified amount of time, in order to enable us to close the location for an event, or for a maintenance, instead of just reserving a time slot for that, as it is more informative.

[1] `https://vuejs.org/`

8.2 Deployment

For deployment, we have our own machine where the server is hosted. However, I plan on creating a container for the application. „A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.“[13]. By dockerizing the application, new docker image will be automatically released when changes are pushed into our git repository. On the server, the only thing that needed to be done is to automatically fetch the latest image. This way it would all be automated and changes and new features easily deployed. Another advantage is having a complete control over the environment, so at all times we know exactly what versions of the dependant services are we using. This would be a big improvement over the way changes to the application are made today. Right now, you need to log in into the server, and manually change the code.

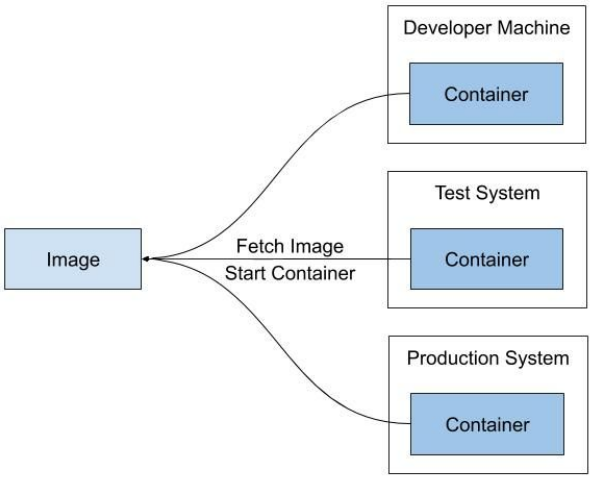

Figure 8.1. Deployment docker[14]

Right now, the application is deployed online using Heroku [1] on the url [2]. The app is deployed automatically once changes are pushed to the github repository on branch deployment.

[1] `https://dashboard.heroku.com/`
[2] `https://sherna-web.herokuapp.com/`

Chapter 9
Conclusion

The purpose of the bachelor thesis was to analyse the current implementation of the SHerna web system, design and implement a new version of this application that would satisfy the requirements and would be expandable.

First, I have discussed the technologies used, the architecture, and its perks and disadvantages. I have analysed and listed all the problems with the current solution. I have written down the requirements for the system. Then I presented the draft of the new implementation.

Afterwards, I followed the implementation process, describing the new architecture and obstacles I faced, and how I resolved the problems of the previous solution. Then I moved on to describe the testing process.

At the end, I drafted the future work that needs to be done on the application, with new features that are still missing or could be improved. Briefly I talked about the deployment and how the system will be managed in the future.

Appendix A
Symbols

PHP	Recursive initialism for PHP: Hypertext Preprocessor
MVC	Model View Controller
CRUD	Create read update and delete
SOLID	Single responsibility, Open–closed principle, Liskov substitution principle, Interface segregation principle principles, Dependency inversion principle
DRY	Don't repeat yourself
MVP	Minimum viable product
SQL	Structured Query Language
NoSQL	Not Only Structured Query Language

References

[1] *Laravel philosophy*.
https://laravel.com/docs/4.2/introduction.

[2] James Bucanek. *Learn Objective-C for Java Developers*. Apress, 2009. ISBN 978-1-4302-2369-6.
https://link.springer.com/book/10.1007/978-1-4302-2370-2.

[3] *Laravel Blade*.
https://laravel.com/docs/7.x/blade#introduction.

[4] Steve Fenton. *Pro TypeScript: Application-Scale JavaScript Development*. Edition 1 edition. Apress, 2014. ISBN 978-1430267911.

[5] *WHY SHOULD EVERY MAGENTO DEVELOPER FOLLOW SOLID PRINCIPLES?*
https://dckap.com/blog/why-should-every-developer-follow-solid-principles.

[6] *The basic architecture of Laravel applications*.
https://subscription.packtpub.com/book/web_development/9781788833912/1/ch01lvl1sec11/the-basic-architecture-of-laravel-applications.

[7] Frank Robinson. *MINIMUM VIABLE PRODUCT*.
http://www.syncdev.com/minimum-viable-product/.

[8] *Common Mistakes Agile Software Development Teams Make*.
https://number8.com/common-mistakes-using-agile-method.

[9] *8 Major Advantages of Using MySQL*.
https://www.datamation.com/storage/8-major-advantages-of-using-mysql.html.

[10] *NoSQL vs Relational Databases*.
https://www.mongodb.com/scale/nosql-vs-relational-databases.

[11] *Eloquent ORM*.
https://laravel.com/docs/7.x/eloquent.

[12] *Laravel Tests Generator*.
https://laravelshift.com/laravel-test-generator.

[13] *What is a Container?*
https://www.docker.com/resources/what-container.

[14] *How to dockerize your PHP application for AWS Fargate?*
http://cloudonaut.io/how-to-dockerize-your-php-application-for-aws-fargate.

[15] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Edition 1 edition. Prentice Hall, 2008. ISBN 978-0132350884.

[16] Matt Stauffer. *Laravel: Up & Running: A Framework for Building Modern PHP Apps*. 2nd ed. edition edition. O'Reilly, 2019. ISBN 978-1492041214.