



**FACULTY  
OF MATHEMATICS  
AND PHYSICS**  
Charles University

## **BACHELOR THESIS**

Štěpán Klos

# **Web application for swimming competitions management**

Department of Software Engineering

Supervisor of the bachelor thesis: doc. Mgr. Martin Nečaský, Ph.D.

Study programme: Computer Science

Study branch: Software and Data Engineering

Prague 2022

Sth

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....  
Author's signature

Dedication.

Title: Web application for swimming competitions management

Author: Štěpán Klos

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., Software and Data Engineering

Abstract: The goal of this work is to create a system that facilitates management of swimming competitions in the Czech Republic. This system must contain necessary infrastructure with easy-to-use web interface that is also mobile friendly. SwimmPair is using MySQL database for storing data, extensible PHP managers for performing all backend tasks. Frontend is implementet via custom drag'n'drop DOM API in JavaScript.

Keywords: key web application, web, automation, catalogization, administration, cms, full stack, frontend, backend

# Contents

|   |           |
|---|-----------|
| <b>Introduction</b>   | <b>3</b>  |
| <b>1 Status quo and solution</b>                              | <b>4</b>  |
| 1.1 Problem description . . . . .                             | 4         |
| 1.2 Target audience . . . . .                                 | 5         |
| 1.3 Scenarios of storyboards . . . . .                        | 5         |
| 1.4 Model proposition . . . . .                               | 5         |
| 1.5 Frontend practices . . . . .                              | 8         |
| <b>2 Architecture overview</b>                                | <b>9</b>  |
| 2.1 Technologies . . . . .                                    | 9         |
| 2.2 Application flow . . . . .                                | 10        |
| 2.3 Managers . . . . .  | 10        |
| 2.4 Start file . . . . .                                      | 12        |
| 2.5 Templating of web and administration . . . . .            | 13        |
| 2.6 Responsive layout . . . . .                               | 14        |
| 2.7 Administrative tasks . . . . .                            | 14        |
| 2.8 Club Manager tasks . . . . .                              | 14        |
| 2.9 Referee . . . . .   | 14        |
| <b>3 Implementation Documentation</b>                         | <b>15</b> |
| 3.1 Database design . . . . .                                 | 15        |
| 3.1.1 Schema . . . . .  | 15        |
| 3.1.2 Object tables . . . . .                                 | 15        |
| 3.1.3 Relation tables . . . . .                               | 17        |
| 3.1.4 Content adjustment tables . . . . .                     | 19        |
| 3.2 Managers documentation . . . . .                          | 20        |
| 3.2.1 PostsManager.php . . . . .                              | 20        |
| 3.2.2 UsersManager.php . . . . .                              | 20        |
| 3.2.3 ClubsManager.php . . . . .                              | 21        |
| 3.2.4 CupsManager.php . . . . .                               | 21        |
| 3.2.5 PositionsManager.php . . . . .                          | 22        |
| 3.3 Application structure - files defined structure . . . . . | 22        |
| 3.4 JavaScript functions documentation . . . . .              | 24        |
| 3.4.1 Previous post . . . . .                                 | 24        |
| 3.4.2 User statistics - year change . . . . .                 | 24        |
| 3.4.3 Club statistics - year change . . . . .                 | 25        |
| 3.4.4 Filtering referees . . . . .                            | 25        |
| <b>4 Testing</b>  | <b>26</b> |
| 4.1 Performance testing . . . . .                             | 26        |
| 4.2 User testing . . . . .                                    | 27        |
| <b>5 Deployment</b>   | <b>29</b> |
| 5.1 Administration . . . . .                                  | 29        |

|          |                              |           |
|----------|------------------------------|-----------|
| <b>6</b> | <b>User Manual</b>           | <b>30</b> |
| 6.1      | Public part . . . . .        | 30        |
| 6.2      | Administration . . . . .     | 30        |
|          | <b>Conclusion</b>            | <b>31</b> |
|          | <b>Bibliography</b>          | <b>32</b> |
|          | <b>List of Figures</b>       | <b>33</b> |
|          | <b>List of Tables</b>        | <b>34</b> |
|          | <b>List of Abbreviations</b> | <b>35</b> |
| <b>A</b> | <b>Attachments</b>           | <b>36</b> |
| A.1      | First Attachment . . . . .   | 36        |

# Introduction

Being born in mid 90s has given me the opportunity to observe development of personal computing and advent of internet first-hand. By the time I was three I was fortunate enough to experience my father's first computer running Windows 98. By the time I was five I already knew I wanted to be a programmer when I grow up. I realized that I could write some lines and make a public website. I've been fascinated by Microsoft and Apple, by Bill Gates, Steven Balmer, Paul Allen and by Steve Jobs and Steve Wozniak. They are the geniuses who put computers on our tables and iPhones in our pockets. This is, however, just brief overview of my fascination by IT world.

## Why web applications

Dot-com bubble crash was correction of overhyped optimism stemming from new technologies in early 2000s, subsequently helping whole industry to mature. It was year 2008 and financial crisis that brought the real opportunities in the web space. Despite having made an average American customer poorer, it has brought the world new trend of money saving services that were meant for cutting the costs or making extra cash. To save money, one didn't call a taxi, you called UBER. To make extra money one started renting an extra room at AirBnB. Distrust in banking industry created Bitcoin and made us think of other uses of blockchain. It is not hard to see that these things are not as technically complicated as one would think. A CS grad with correct amount of enthusiasm should be able to deploy a MVP of each thing previously mentioned in couple of weeks or months.

## Motivation

This thesis is a fullstack system meant for my fellow to save time for more important tasks which he has as a chief swimming referee and manager. This is valuable training for me since I have to forge a solution of problem vaguely resembling one of the MVPs listed above. Having delved into this problem rewarded me with valuable experiences and insights. These experiences are hopefully going to help me in my future endeavors and career.

I realized that Software and Data Engineering is a crucial craftsmanship for delivering these positive changes. Building things is the modern adventure.



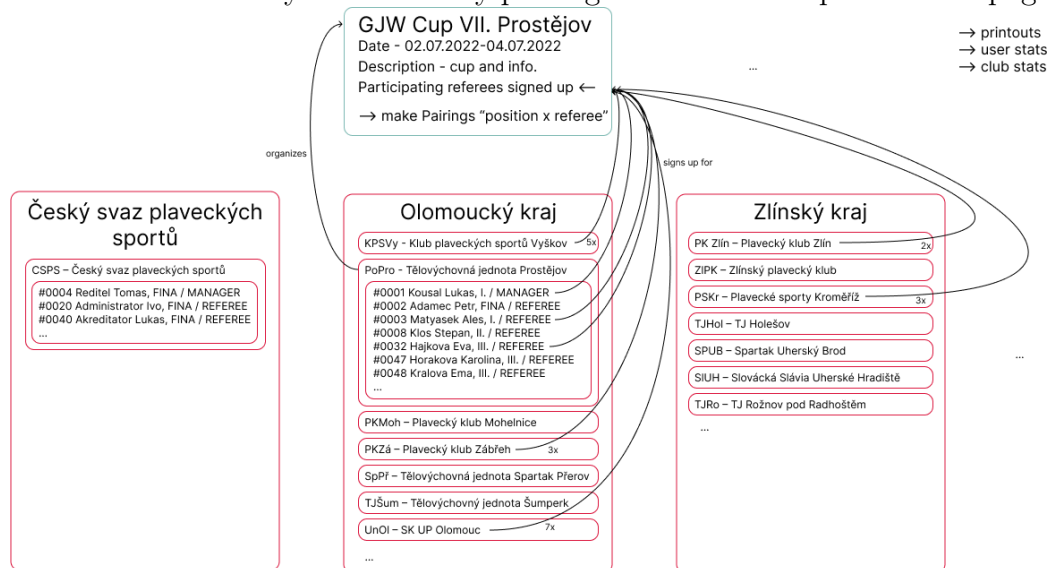
# 1. Status quo and solution

This section is an introduction into problem how the proposed solution looks like and what practices are used in the SwimmPair web application.

## 1.1 Problem description

A friend of mine reached out to me to ask me in order to ask if I could automate part of his agenda work agenda. Administration of swimming competitions and creating statistics is very repetitive and error-prone list of tasks. However, almost all the tasks from about organization are executed in the same order.

The Czech Swimming Federation <sup>1</sup> structure has to be modeled as objects in the application and database records as a storage. Thus, logical structure should be set and implemented in following order. Swimming referees belong to clubs, clubs are located in geographical regions. Swimming cup is organised by a club. Each Club contains several swimming referees and one of them is a club manager. When a Cup is online each Swimming referee can sign himself or herself up as available for the Cup. Club manager can also sign members of his club for a cup. At the end of the day, organizer of the cup assigns available referees that signed up to positions that he finds them suitable for. My friend, the chairman of referee committee should be able to perform additional administrative other tasks, such as adding and removing users, creating new clubs and modifying whole structure. Administrator can notify all visitors by posting an information psa on homepage.



The SwimmPair system should deliver public listing of **users**, **cups**, **news**, **individual statistics** and **club statistics**. System should allow to browse stats on a yearly basis. This schema will then be appropriately modeled by objects and the schema will be shown below.

<sup>1</sup><https://www.czechswimming.cz>

## 1.2 Target audience

Lukas and his pals.

## 1.3 Scenarios of storyboards

These are tasks that have to be performed:

- create swimming cup,
- perform pairing for swimming cup,
- manage swimming club,
- preview cup + print pairing,
- participate in cup or participate with teammates,
- statistics prereviews for accreditations renewal,
- statistics of referees,
- statistics of clubs,
- referees managment,
- clubs managment,
- clubs overview.

## 1.4 Model proposition

### Cup

Cup is the most important object of SwimmPair. A swimming Cup contains name, description, date and is affiliated to organising Club. Cup serves two purposes. Firstly - assigning referees for specific tasks (time tracking, computer support, head of the cup) has to be ready by the time the event takes place. Secondly - statistics summing up participations for Users and Clubs have to be calculated for each year over all cups in this time period. We also have to discriminate between upcoming and already past cups. Upcoming cups are displayed on the top, past cups should reside in the archive to be revisited for statistics purposes.

### User

User is an entity modelling swimming referee. A referee participating in this system falls in one of three categories. These categories or levels if you wish are **referee**, **club manager** and **administrator**. User must be uniquely identifiable. A person i.e. User in the system is going to have profile information such as first name, family name, email address. Good practice of using an email address as

a login information is going to be used here. User must also contain SwimmPair hierarchy listed above and indicator of one's skill and knowledge in the swimming field, i.e. referee category. User must also belong to exactly one club in our system.

## Club

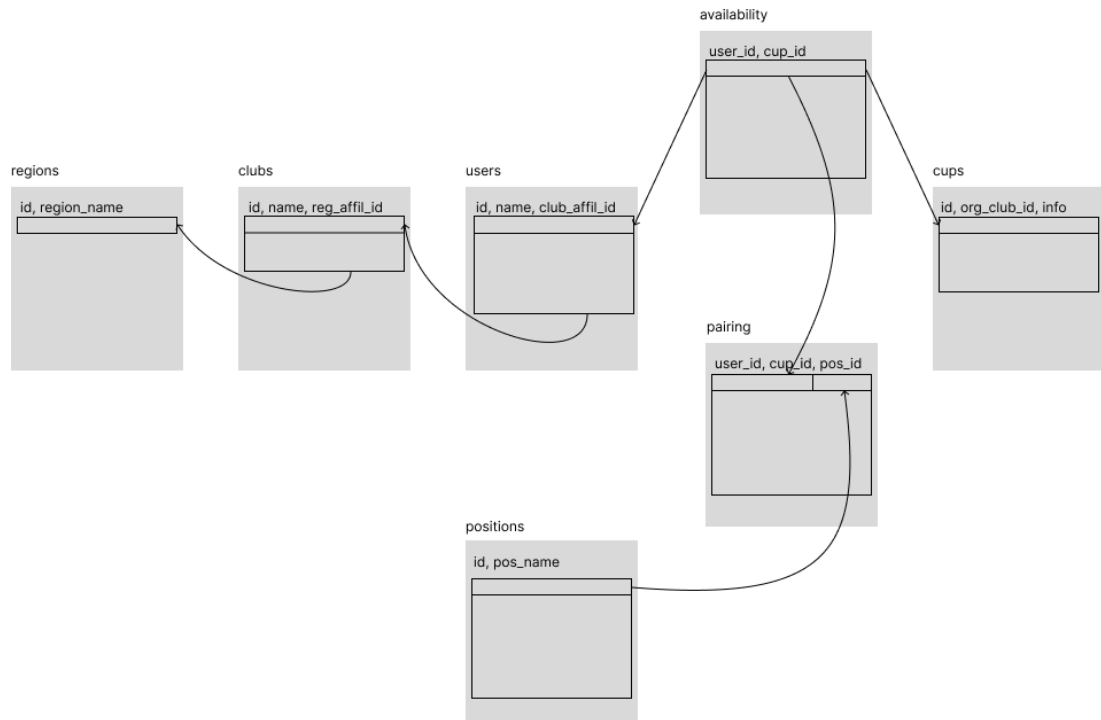
Club is an administrative unit of people. Club has specific name, abbreviation and ID in Czech Referee Federation. An image can be included as well. A club will be serving as a formal authority organising Cup - by a User who is Club Manager. Club is unanimously affiliated to Region. Statistics regarding performance of members of Club at swimming competitions must be implemented. Statistics have informative characted and will save time in the current status quo - keeping track of presence and work descriptions in Excel spreadsheets. Post is an informative snippet to be displayed at homepage to notify other swimmers about new event or anything worth paying attention to. Homepage should display last 3 posts and should be allowed to load more.

## Region

One of the 13 regions of the Czech Republic in which this system is used. Clubs are located in one of these regions. When new Club starts using SwimmPair, new region has to be added and potential clubs created and attached to this Region. We list objects of our application model and describe their properties and purpose.

## Schema mockup

Majority of focus should be on tables **users** and **cups** in presented schema. Users belong to clubs that belong to regions. These two entities **user** and **cups** are then brought together into table **availability** which lists referees for specific cups. Availabile users for cup are then paired in **pairing** where each couple can be assigned a position from **positions**.



## Availability

Takes track of **User** available for **Cup** who are signed either by their team manager or themselves.

## Pairing

Record from availability is then taken and one or more **positions** are assigned to it.

## Position

Predefined list of tasks necessary to be done at each cup. This list is probably never going to change since there is a fixed set of roles. Referees are going to be assigned to these positions for each cup by drag'n'drop user interface.

## **1.5 Frontend practices**

Several good practices have to be implemented to make SwimmPair easy to use. These practices are either well known or situation specific but they have one thing in common - they make the application good to use.

### **Smooth frontend browsing**

Frontend of SwimmPair should be easy to use. There are several options and use cases of JavaScript that can come in handy. Reduction of page reloads is definitely a good way to go. Therefore there are going to be asynchronous JavaScript calls for obtain semi-partial data. After, next function will modify the DOM based on data received from asynchronous call.

### **Multiple device types**

Today is certain that there are users who want to browse our system from pc, tablet or smartphone and responsive design is a necessity. Since CSS3 supports media queries we are going to use them for creation of device specific styling.

### **Assigning referees to positions via. drag'n'drop**

Assigning referees to positions for cups should be implemented via drag'n'drop. Dragging a referee, moving referee over the region specified for the positions and releasing mouse button. Double clicking this person is a good way of removing it.

### **Printouts of pairing**

Upcoming Cup can be directly printed from website and hanged as data printout.

### **Appropriate design**

Red blue and grey are colors that appear pretty much at a swimming pools. These colors will be used in our system as well. The elements should have fresh lightweave look and not appear heavy.

## 2. Architecture overview

This overview should familiarize the reader with architecture of application. There are two parts, **public web** and **private administration**. Administration is hidden behind login/password.

When designing such system, object oriented approach and grouping of similar functions together is a must. There are objects that have to be moved around the web application described in previous chapter. These objects are Post, User, Club, Cup, Position and Region. Therefore we came up with a concept of managers. Each page of SwimmPair is composed of same header, menu, footer. The content part is filled with page's specific results of manager call used to construct data UI page layout. These managers are included and used in all pages via **start file**.

### 2.1 Technologies

Following technologies are used to implement SwimmPair application:

- **HTML** is HyperText Markup Language <sup>1</sup> - application pages are templated in HTML by PHP,
- **CSS** is Cascading Style Sheets <sup>2</sup>,
- **PHP** is a general-purpose scripting language geared toward web development <sup>3</sup> - object model and backend services are provided by it,
- **JavaScript** is a general-purpose scripting language that conforms to the ECMAScript specification <sup>4</sup>,
- **MySQL** is an open-source relational database management system <sup>5</sup>,
- **Git** is a distributed version control system: tracking changes in any set of files - this project is versioned and kept in public GitHub repository <sup>6</sup>,
- **Docker** is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers <sup>7</sup> - used for deployment of our application,
- **Kubernetes** is an open-source container orchestration system for automating software deployment, scaling, and management <sup>8</sup> - used for production cluster of our application.

---

<sup>1</sup>[WHATWG, 26 December 2022]

<sup>2</sup>[W3C, 31 December 2022]

<sup>3</sup>[The PHP Group, 28 November 2019]

<sup>4</sup>[ecma INTERNATIONAL, June 2022]

<sup>5</sup>[Oracle, 2023]

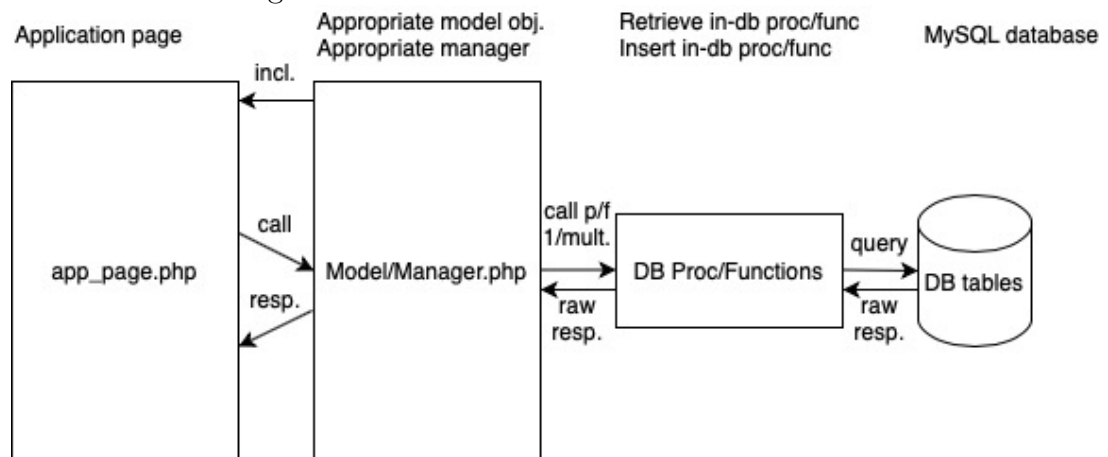
<sup>6</sup><https://github.com/KlosStepan/SwimmPair-Www>

<sup>7</sup>[Docker, 2023]

<sup>8</sup>[The Kubernetes Authors, 2023]

## 2.2 Application flow

Visitor comes to **app page**, where **managers** are included. From page there are API calls on Managers that retrieve and store data data.



## 2.3 Managers

Managers are written to provide API functionality for system administration in PHP. These managers are populating pages or taking new input from them and administer process of storing them. Each object has a manager handling it and accomodates database loads and stores controlled by transactions.

- Post/PostsManager
- User/UsersManager
- Page/PagesManager
- Club/ClubsManager
- Cup/CupsManager
- Position/PositionsManager
- Region/RegionsManager

Managers are implemented to extract and store data of class by which they are named after. Let's take PostsManager as an example. This manager handles Post and is implemented as follows.

### Post.php

```
class Post
{
    public $id;
    public $timestamp;
    public $title;
    public $content;
    public $display_flag;
    public $author_user_id;
    public $signature_flag;
```

```

    public function __construct($id, $timestamp, $title, $content,
        $display_flag, $author_user_id, $signature_flag)

    //7/7: {id, timestamp, title, content, display_flag,
        author_user_id, signature_flag}
    public function Serialize()
}

```

### **PostsManager.php**

```

class PostsManager
{
    private $mysqli;

    //Constructor - setting $mysqli to $this->mysqli
    public function __construct(mysqli $mysqli)

    //Handling functions retrieve/store
    public function GetPostById($id)
    public function FindLastNPosts($N)
    public function InsertNewPost($title, $content, $display_flag,
        $author, $signature_flag)
    public function UpdatePost($id, $title, $content, $display_flag,
        $signature_flag)

    //Private functions - auxiliary controller functions
    private function _CreatePostOrNullFromStatement(mysqli_stmt
        $statement)
    private function _CreatePostsFromStatement(mysqli_stmt $statement)
    private function _CreatePostFromRow(array $row)
}

```

### **Demonstration - public function GetPostById(\$id)**

```

public function GetPostById($id)
{
    $statement = $this->mysqli->prepare("CALL _GetPostById`(`?);");
    $statement->bind_param('i', $id);
    return $this->_CreatePostOrNullFromStatement($statement);
}

```



## 2.4 Start file

Start file is included the in beginning of each page. It serves for **connection to database**, **sanitization of input**, **definition of error handling** and most importantly **includes objects and managers** and subsequently **instantiates all managers** while passing reference to the database connection `$mysqli` their only constructor argument.

```
/*Database credentials from environment*/
$host = getenv("DATABASE_HOST");
$user = getenv("DATABASE_USER");
$pass = getenv("DATABASE_PASS");
$db = getenv("DATABASE_NAME");
/*Database connection and charset set*/
$mysqli = new mysqli($host, $user, $pass, $db) or die($mysqli->error
);
$mysqli->set_charset('utf8');
/* Sanitization function */
function h($string)
{
    return htmlspecialchars($string);
}
/* Exception handling*/
error_reporting(E_ALL);
ini_set("display_errors", 1);
set_exception_handler(function () {
    echo "<h3_style=\"color:red;\">INVALID REQUEST</h3>";
    exit();
});
/* Objects and Managers*/
require __DIR__ . '/model/Sanitizer.php';
require __DIR__ . '/model/Auth.php';
require __DIR__ . '/model/Post.php';
require __DIR__ . '/model/PostsManager.php';
require __DIR__ . '/model/Page.php';
require __DIR__ . '/model/PagesManager.php';
require __DIR__ . '/model/StatUserCnt.php';
require __DIR__ . '/model/StatPositionCnt.php';
require __DIR__ . '/model/RefereeRank.php';
require __DIR__ . '/model/Region.php';
require __DIR__ . '/model/RegionsManager.php';
require __DIR__ . '/model/User.php';
require __DIR__ . '/model/UsersManager.php';
require __DIR__ . '/model/Cup.php';
require __DIR__ . '/model/PairPositionUser.php';
require __DIR__ . '/model/CupsManager.php';
require __DIR__ . '/model/Position.php';
require __DIR__ . '/model/PositionsManager.php';
require __DIR__ . '/model/Club.php';
require __DIR__ . '/model/ClubsManager.php';
/* Construction of Managers w/ reference to $mysqli */
$postsManager = new PostsManager($mysqli);
$pagesManager = new PagesManager($mysqli);
$usersManager = new UsersManager($mysqli);
$clubsManager = new ClubsManager($mysqli);
$cupsManager = new CupsManager($mysqli);
$positionsManager = new PositionsManager($mysqli);
$regionsManager = new RegionsManager($mysqli);
```

## 2.5 Templating of web and administration

Each page layout of public website has common characteristics such as header, menu and footer. These sections are unified and included everywhere, therefore they are included everywhere. They are:

- HEADER,
- MENU,
- Generated from result obtained by one or more manager calls. this section might be further updated via XMLHttpRequest calls & DOM modifications of newly delivered data,
- FOOTER.

Homepage of administration panel /admin/profile.php after login gets assembled with regards to the rights of logged user. Ordering is following: Admin (2) > Club manager (1) > Swimming referee (0) and each user gets snippet of his and lower role snippets:

- SUPERUSER menu snippet - 2,
- CLUB MANAGER menu snippet - 1,
- SWIMMING REFEREE menu snippet - 0.

Following flow is then discriminated based on rights code on each page.

### Rights check

```
<?php
require __DIR__ . '/../start.php';
session_start();
Auth::requireRole(UserRights::SuperUser);
...
?>
...
```

### requireRole on Auth class

```
class Auth
{
    public static function requireRole($role)
    {
        if (!isset($_SESSION['rights']))
        {
            header('Location: ../prihlaseni.php');
            exit();
        }
        //Rights sharply lower that user has, throw RuntimeException
        if ($_SESSION['rights'] < $role)
        {
            echo '<h1>Not enough rights</h1>';
            echo $_SESSION['rights'];
            echo $role;
            throw new RuntimeException();
        }
    }
}
```

## 2.6 Responsive layout

Listed media queries are used to provide design of the web by manually overriding specific classes for desired user experience outcome.

- Basic CSS design
- @media (max-width: 768px)
- @media (print)

**Basic CSS design** gives definition of colors and desktop layout of our application. **Media query with max-width: 768px** supports tablets and mobile devices while **media print** of cup pairing hides redundant controll and informative elements while it keeps the pairing of cup to be printed.

## 2.7 Administrative tasks

- **Add Post/Edit Post** - from PostsManager call InsertNewPost/UpdatePost
- **Approve Newly Registered Users** - swap flag **approved** to **1**
- **Pair Available Users On Cup Positions** - from UsersManager call UpdatePairing - calls several SQL Procs for different things in transaction and commits/rollbacks
- **Add User/Edit User** - from UsersManager call AddUser/UpdateUser
- **Add Cup/Edit Cup** - from CupsManager call AddCup/UpdateCup
- **Add Club/Edit Club** - from ClubsManager call AddClub/UpdateClub
- **Add Region/Edit Region** - from RegionsManager call AddRegion/UpdateRegion
- **Configure Stats Ordering** - delete ordering, insert ordering **Nth-statId**
- **Edit Contacts** - from PagesManager call UpdatePage

## 2.8 Club Manager tasks

- **Add Cup** - from CupsManager call AddCup
- **Sign Up People From My Club As Available For Cup** - prihlasi\_moje\_lidi\_na.php then XMLHttpRequest/call\_update\_availability.php

## 2.9 Referee

- **Sign Myself As Available For Cup** - add my Id to table **cupId-userId**

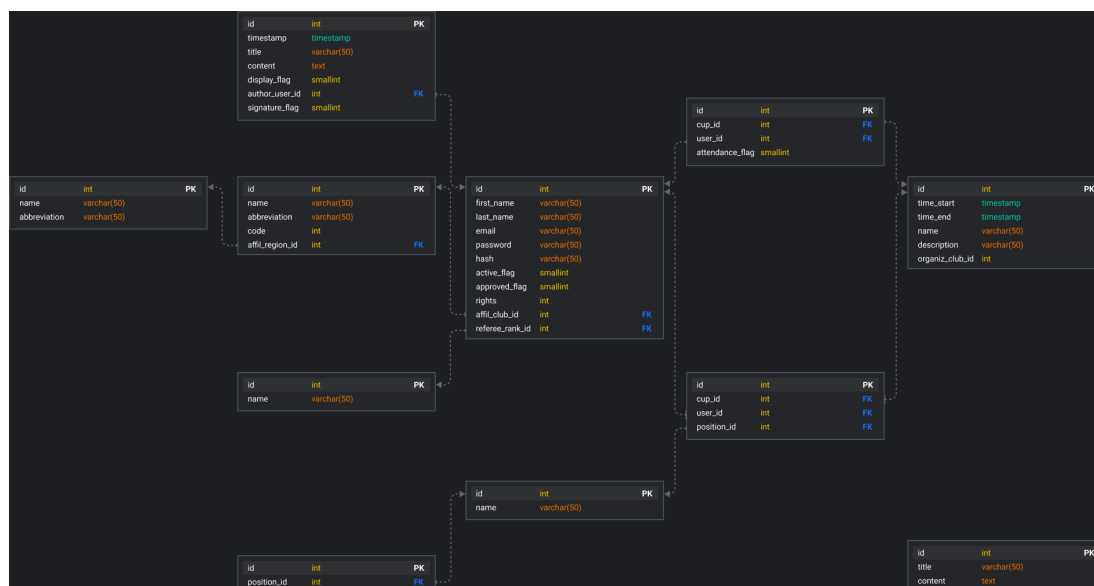
# 3. Implementation Documentation

Detailed description of **database** and **backend components and functions**.

## 3.1 Database design

For this purpose well defined database is a necessity. In this chapter we learn how the proposed objects are represented how relations between these objects are maintained and what information tables exist.

### 3.1.1 Schema



### 3.1.2 Object tables

These are the tables in database modeling the object to satisfy the primary motivation defined as the **problem paradigm**. These rows are then being converted to Objects and returned to user by appropriate Manager.

#### sp\_posts

Posts for main page are stored in this table.

Table preview

| id  | timestamp   | title      | content      | display_flag | author | sign_flag |
|-----|-------------|------------|--------------|--------------|--------|-----------|
| 1   | 2018-01-... | New Por... | SwimmPair... | 1            | 21     | 0         |
| 2   | 2018-03-... | Updates    | This web...  | 1            | 21     | 0         |
| ... | ...         | ...        | ...          | ...          | ...    | ...       |

Columns description

1. **id**, PK, int(11), AUTOINCREMENT
2. **timestamp**, datetime
3. **title**, text
4. **content**, text
5. **display\_flag**, tinyint
6. **author**, FK, int(11) | NULL
7. **sign\_flag**, tinyint

### sp\_users

Users are stored in this table.

Table preview

| id  | first_name | last_name | email         | password | hash   | ... |
|-----|------------|-----------|---------------|----------|--------|-----|
| 1   | Lukáš      | Kousal    | lukas@swim.cz | -PASS-   | -HASH- | ... |
| ... | ...        | ...       | ...           | ...      | ...    | ... |
| N   | ...        | ...       | ...           | ...      | ...    | ... |

Columns description

1. **id**, PK, int(11), AUTOINCREMENT
2. **first\_name**, varchar(50)
3. **last\_name**, varchar(50)
4. **email**, varchar(100) //unique identifier
5. **password**, varchar(100)
6. **hash**, varchar(32)
7. **active**, tinyint(1)
8. **approved**, tinyint(1)
9. **rights**, int(11)
10. **klubaffil**, FK, int(11)

### sp\_clubs

Clubs are stored in this table.

Table preview

| id  | name                          | zkratka | idklubu | img      |
|-----|-------------------------------|---------|---------|----------|
| 1   | Klub plaveckých sportů Vyškov | KPSVy   | 614     | null.jpg |
| ... | ...                           | ...     | ...     | ...      |
| 14  | TJ Rožnov pod Radhoštěm       | TJRo    | 0       | null.jpg |

Columns description

1. **id**, PK , int(11), AUTOINCREMENT
2. **name**, varchar(512)

3. **zkratka**, text
4. **idklubu**, int(11)
5. **img**, text

### sp\_cups

Cups are stored in this table.

Table preview

| id  | date       | name       | description                  | owningclub |
|-----|------------|------------|------------------------------|------------|
| 1   | 2017-06-12 | GJW Cup I. | Cup organized by GJW PV, ... | 2          |
| ... | ...        | ...        | ...                          | ...        |

Columns description

1. **id**, PK , int(11), AUTOINCREMENT
2. **date**, date
3. **name**, text
4. **description**, text
5. **owningclub**, int(11)

### sp\_positions

List of Positions for which we are pairing users are stored here.

Table preview

| id  | poz             |
|-----|-----------------|
| 1   | Vrchní rozhodčí |
| ... | ...             |
| 19  | Ostatní         |

Columns description

1. **id**, PK , int(11), AUTOINCREMENT
2. **poz**, varchar(512)

## 3.1.3 Relation tables

Relation tables hold the most important information stored in the SwimmPair system - the **pairings** and **data for underlying statistics**. Both availability for cups and pairings to positions are represented here.

### sp\_cup\_user\_availability

This table stores relationships between referees/users and cups called availability. Referees are signed up by their team manager or themselves as available for the cup. In case of sudden inability to participate, the `attendance_flag` is switched to 0 in case the user is already assigned to some position. In that case the administrator is going to see the user in red box.

Table preview

| id  | cup_id | user_id | attendance_flag |
|-----|--------|---------|-----------------|
| 1   | 3      | 21      | 1               |
| 2   | 3      | 1       | 1               |
| 7   | 3      | 19      | 0               |
| ... | ...    | ...     | ...             |

Columns description

1. **id**, PK, int(11), AUTOINCREMENT
2. **cup\_id**, FK, int(11)
3. **user\_id**, int(11)
4. **attendance\_flag**, tinyint(1)

### sp\_position\_user\_pairing

This table stores pairing information about available referees/users on positions for each cup. This is the most time saving utility of the SwimmPair.

Table preview

| id  | cup_id | position_id | user_id |
|-----|--------|-------------|---------|
| 46  | 5      | 5           | 21      |
| 484 | 3      | 1           | 21      |
| 485 | 3      | 1           | 22      |
| 486 | 3      | 2           | 7       |
| 487 | 3      | 3           | 15      |
| 487 | 3      | 5           | 12      |
| 487 | 3      | 7           | 14      |
| ... | ...    | ...         | ...     |

Columns description

1. **id**, PK, bigint(11), AUTOINCREMENT
2. **cup\_id**, FK, int(11)
3. **position\_id**, FK, int(11)
4. **user\_id**, FK, int(11)

### 3.1.4 Content adjustment tables

#### sp\_public\_stats\_config

Configuration table of which positions in what order should be displayed in statistics on frontend. For frontend then LEFT-JOIN **position\_id** from table **sp\_positions** ON **id** and display **poz**.

Table preview

| id  | position_id |
|-----|-------------|
| 148 | 1           |
| 149 | 8           |
| 150 | 2           |
| 151 | 4           |
| 152 | 6           |

Columns description

1. **id**, PK, int(11), AUTOINCREMENT
2. **position\_id**, FK, int(11)

#### sp\_pages

SwimmPair static Pages.

Table preview

| id | title    | content  |
|----|----------|--|
| 1  | Kontakty | <h1>Title</h1><p>Contact information +420...</p> |

Columns description

1. **id**, PK , int(11), AUTOINCREMENT
2. **title**, text
3. **content**, text



## 3.2 Managers documentation

These five controllers work with objects and provide login (i.e. joining more tables in varios ways to achieve all functionality).

### 3.2.1 PostsManager.php

- Post | null  $\leftarrow$  **GetPostById**(\$id)  
     $\searrow$  \_CreatePostOrNullFromStatement(\$stmt)  
     $\searrow$  \_CreatePostFromRow(\$row)
- Post[] | null  $\leftarrow$  **GetLastThreePosts**()  
     $\searrow$  \_CreatePostsFromStatement(\$stmt)  
     $\searrow$  \_CreatePostFromRow(\$row)
- Post[] | null  $\leftarrow$  **GetLastNPosts**(\$N)  
     $\searrow$  \_CreatePostsFromStatement()  
     $\searrow$  \_CreatePostFromRow(\$row)
- true | false  $\leftarrow$  **AddNewPost**(\$title, \$content)
- Post[] | false  $\leftarrow$  **FindAllPostsOrderedByIdDesc**()  
     $\searrow$  \_CreatePostsFromStatement(\$stmt)  
     $\searrow$  \_CreatePostFromRow(\$row)
- true | false  $\leftarrow$  **UpdatePost**(\$id, \$title, \$article)

### 3.2.2 UsersManager.php

- User | null  $\leftarrow$  **GetUserById**(\$id)  
     $\searrow$  \_CreateUserOrNullFromStatement(\$stmt)  
     $\searrow$  \_CreateUserFromRow(\$row)
- User[] | null  $\leftarrow$  **FindAllActiveUsersOrderByNameDesc**()  
     $\searrow$  \_CreateUsersFromStatement(\$stmt)  
     $\hookrightarrow$  \_CreateUserFromRow (\$row)
- User[] | null  $\leftarrow$  **FindAllInactiveUsersOrderByNameDesc**()  
     $\searrow$  \_CreateUsersFromStatement(\$stmt)  
     $\hookrightarrow$  \_CreateUserFromRow (\$row)
- User[] | null  $\leftarrow$  **FindAllRegisteredMatesForTheCup**(\$cupId, \$teamId)  
     $\searrow$  \_CreateUsersFromStatement(\$stmt)  
     $\hookrightarrow$  \_CreateUserFromRow (\$row)
- User[] | null  $\leftarrow$  **FindAllMates**(\$teamId)  
     $\searrow$  \_CreateUsersFromStatement(\$stmt)  
     $\hookrightarrow$  \_CreateUserFromRow(\$row)
- User[] | null  $\leftarrow$  **FindAllRegisteredUsersForTheCup**(\$cupId)  
     $\searrow$  \_CreateUsersFromStatement(\$stmt)  
     $\hookrightarrow$  \_CreateUserFromRow(\$row)
- User[] | null  $\leftarrow$  **FindAllNametagsForTheCup**(\$cupId)  
     $\searrow$  \_CreateUsersFromStatement(\$stmt)  
     $\hookrightarrow$  \_CreateUserFromRow(\$row)

- $\underline{\text{User}}[] \mid \text{null} \leftarrow \mathbf{FindPairedUsersOnCupForPosition}(\$cupId, \$posId)$   
 $\searrow \_CreateUsersFromStatement(\$stmt)$   
 $\hookrightarrow \_CreateUserFromRow(\$row)$
- $\underline{\text{Pair}}[] \mid \text{null} \leftarrow \mathbf{FindPairedPozIdUserIdOnCup}(\$cupId)$   
 $\searrow \_CreatePairsFromStatement(\$stmt)$   
 $\hookrightarrow \_CreatePairFromRow(\$row)$
- $\underline{\text{string}} \mid \text{null} \leftarrow \mathbf{GetClubAbbreviationByAffiliationId}(\$id)$   
 $\searrow \_GetSingleResultFromStatement(\$stmt)$
- $\underline{\text{string}} \mid \text{null} \leftarrow \mathbf{GetUserFullNameById}(\$id)$   
 $\searrow \_GetSingleResultFromTwoColsStatement(\$stmt)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \mathbf{UserWithEmailExists}(\$email)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \mathbf{RegisterUserFromAdmin}(\$first\_name, \$last\_name, \$email, \$password, \$rights, \$klubaffil)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \mathbf{SendYouWereRegisteredFromAdmin}(\$email, \$password)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \mathbf{ApproveUser}(\$userId)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \mathbf{UpdatePairing}(\$JSON)$

### 3.2.3 ClubsManager.php

- $\underline{\text{Club}} \mid \text{null} \leftarrow \mathbf{GetClubById}(\$id)$   
 $\searrow \_CreateClubFromStatement(\$stmt)$   
 $\searrow \_CreateClubFromRow(\$row)$
- $\underline{\text{Club}}[] \mid \text{null} \leftarrow \mathbf{FindAllClubs}()$   
 $\searrow \_CreateClubsFromStatement(\$stmt)$   
 $\hookrightarrow \_CreateClubFromRow(\$row)$

### 3.2.4 CupsManager.php

- $\underline{\text{Cup}}[] \mid \text{null} \leftarrow \mathbf{FindAllUpcomingCupsEarliestFirst}()$   
 $\searrow \_CreateCupsFromStatement(\$stmt)$   
 $\hookrightarrow \_CreateCupFromRow(\$row)$
- $\underline{\text{Cup}}[] \mid \text{null} \leftarrow \mathbf{FindAllPastCupsMostRecentFirst}()$   
 $\searrow \_CreateCupsFromStatement(\$stmt)$   
 $\hookrightarrow \_CreateCupFromRow(\$row)$
- $\underline{\text{string}} \mid \text{null} \leftarrow \mathbf{GetCupNameById}(\$id)$   
 $\searrow \_GetSingleResultFromStatement(\$stmt)$   
 $\searrow \_GetSingleResultFromStatement(\$stmt)$
- $\underline{\text{Cup}} \mid \text{null} \leftarrow \mathbf{GetCupById}(\$id)$   
 $\searrow \_CreateCupOrNullFromStatement(\$stmt)$   
 $\searrow \_CreateCupFromRow(\$row)$
- $\underline{\text{Pair}}[] \mid \text{null} \leftarrow \mathbf{FindPairingsForThisCup}(\$id)$   
 $\searrow \_CreatePairsFromStatement(\$stmt)$   
 $\hookrightarrow \_CreatePairFromRow(\$row)$

- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \text{InsertNewCupFromAdmin}(\$name, \$date, \$club, \$content)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \text{IsUserAvailableForTheCup}(\$userId, \$cupId)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \text{UpdatePairingForThisCup}(\$cupId, \$JSON)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \text{UpdateAvailabilityForThisCup}(\$cupId, \$JSON)$
- $\underline{\text{true}} \mid \underline{\text{false}} \leftarrow \text{AddAvailableUserForTheCup}(\$cupId, \$userId)$

### 3.2.5 PositionsManager.php

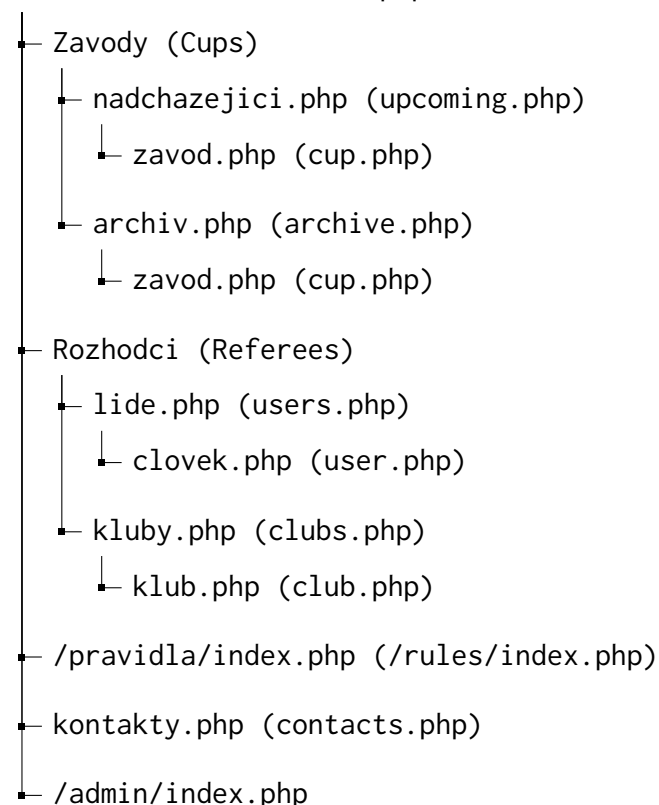
- $\text{Position}[] \mid \text{null} \leftarrow \text{FindAllPositions}()$   
 $\searrow \_CreatePositionsFromStatement(\$stmt)$   
 $\hookrightarrow \_CreatePositionFromRow(\$row)$
- $\text{string} \mid \text{null} \leftarrow \text{GetPositionNameById}(\$id)$   
 $\searrow \_GetSingleResultFromStatement(\$stmt)$

## 3.3 Application structure - files defined structure

### User part of the system

The system is running on Czech URLs for convinience reasons of browsing. English equivalentents of route pages are attached in brackets to demonstrate what the pages do for non-czech speaker. There is no client side routing with traditional LAMP stack.

`www.SwimmPair.cz/index.php`



## Administrative part of the system

The administration has following structure. After going to /admin/index.php user gets logs in and goes to /administration/profile.php. Regarding user's rights (that are passed around along with other information in **SESSION**, retrievable like **\$SESSION['rights']**) one has following structure (**Administration, My Club, Me**). Each user has profile settings for resetting password and other stuff.

[www.SwimmPair.cz/administration/profile.php](http://www.SwimmPair.cz/administration/profile.php)



## 3.4 JavaScript functions documentation

Several features of public website implemented for interactive browsing.

### Library js/SwimmPairFrontendJSLib.js

This library is created to support Ajax calls and DOM operations on frontend. Functions are self-descriptive and **this** means reference to caller DOM element.

- **GetPostAppendPost(PushLastId())**  
    ↘ ConstructNextPost(id, timestamp, title, content, author, signed)
- **ProcessClubForTheSeason(clubId, this)**  
    ↘ CommunicateClubStatsXhrAndUpdateTable(clubId, year)  
    ↘ UpdateClubStatsTable(returnedJSON)
- **ProcessPersonForTheSeason(userId, this)**  
    ↘ CommunicateUserStatsXhrAndUpdateTable(userId, year)  
    ↘ UpdateUserStatsTable(cnt, arr\_str)

### XMLHttpRequest endpoints

- get\_post\_following.php, **GET** args: id
- get\_person\_statistics\_for\_the\_season.php, **GET** args: user\_id, year
- get\_club\_statistics\_for\_the\_season.php, **GET** args: club\_id, year

#### 3.4.1 Previous post

This button on the main page serves as a tool for loading next post. It has **onclick="GetPostAppendPost(PushLastId())"**. Both are JavaScript functions, **PushLastId()** detects id **<article class="post" id="X"...** of last article **class="post"** from DOM by **querySelector** and returns it. This value is then used as an argument of call **GetPostAppendPost(id)**. This function requests article by GET request **XMLHttpRequest/get\_following\_post.php?id=X**. If the result is

- i) **null** button is deleted since there are no other articles to pull from DB,
- ii) **post** next article is constructed and appended from response.

#### 3.4.2 User statistics - year change

All individual referees have seasons years picker when opened. Default season is the current season. Clicking different season visibly changes selected year and obtains appropriate statistics and updates the stats table. Clicking **<span onclick="ProcessPersonForTheSeason(userId, this)"...** calls inside **CommunicateStatsXhrAndPopulateStats(userId, year)** gets data from **XMLHttpRequest/call\_get\_person\_statistics\_for\_the\_season.php?id=userId&year=YYYY** and updates table. Also via this reference in call the button is marked as selected.

### 3.4.3 Club statistics - year change

Club statistics are updated by clicking appropriate year that gets switched. Year onclick calls **ProcessClubForTheSeason(clubId, this)** which gets statistics by calling **CommunicateClubStatsXhrAndUpdateTable(clubId, year)** by calling **XMLHttpRequest/get\_club\_statistics\_for\_the\_season.php?id=clubId&year=Year** and subsequently calling **UpdateClubStatsTable(returnedGetJSON)** which literally updates stats.

### 3.4.4 Filtering referees


This function is triggered by one of these:

- i) **RegionPickerChanged(this)**,
- ii) **RefereeRankPickerChanged(this)**,
- iii) **SearchBarChanged()**.

## Registrovaní rozhodčí

VŠE OLK ZLK

VŠE I. II. III. FINA

 Hledat...

Adamec      Petr      I.      Olomoucký kraj

**FilterQueriedReferees("kraje", "tridy", "inputTrida", "nopplfound")** is called every time one of 3 controls is changed. We then loop all users visible/hidden and check if this one's **Region** **IsOptionPermissible(raid, args[])** (referee area id), **Rank** **IsOptionPermissible(rrid, args[])** (referee rank id) and then if one's Name **IsNamePermissible(args[])**. We then set one's element style to **style=""** and continue cycle execution. If we fail one of these three conditions we proceed to code below which sets element style to **style="display:none"**.

```
...
//Querying
if (IsOptionPermissible(raid , krajeIDs)) {
    if (IsOptionPermissible(rrid , tridyIDs)) {
        if (IsNamePermissible(jmeno , first_name , last_name)) {
            articlePerson.setAttribute("style", "");
            empty = false;
            continue;
        }
    }
}
//Some Condition Fails – Not Permissible
articlePerson.setAttribute("style", "display:none");
}
```

## 4. Testing

There are two main ways to make sure that a web application works properly and fulfills it's role. On one hand there is a code performance testing, performing test on backend level with dummy data insertion and performance benchmarking. On the other hand there is testing to assure that users are able to use system and to get inspiration for future ux iterations via SUS.

### 4.1 Performance testing

Performance script **dummy\_data\_benchmark.php**<sup>1</sup> is located in main swimm-pair folder. It is performed on default database installation (w/ 2 admin users, w/ already existing clubs administered by application requesters, and default referee positions).

**The script has several tasks (tests) which are performed and benchmarked.**

1. Create 98 Users (no. 3-100) - each random affiliation to existing Club (no. 1-15).
2. Create 12 Cups - each random affiliation to existing Club (no. 1-15).
3. Fetch new Users, fetch new Cups (+ fetch static Positions).
4. Create Availabilities (20 Users available per Cup).
5. Create Pairings (each Availability gets 1 random position).
6. Call stats queries (20 - randomly either Clubs or Users stats w/ random club.id or user.id).

**Docker Compose** - 2.3 GHz Core i5 (I5-8259U) RAM 16GB Storage 512GB

| T / rep no. | #1   | #2   | #3   | #4   | #5   | #6   | #7   | #8   | #9   | #10  |
|-------------|------|------|------|------|------|------|------|------|------|------|
| Test #1     | 6.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Test #2     | 0.05 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Test #3     | 6.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Test #4     | 1.14 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Test #5     | 7.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Test #6     | 1.18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| TOTAL RT    | 8.75 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

**Kubernetes - DOKS** Kubernetes v 1.25.4-do.0, **2x Node** s-1vcpu-2gb-intel

| T / rep no. | #1   | #2   | #3   | #4   | #5   | #6   | #7   | #8   | #9   | #10  |
|-------------|------|------|------|------|------|------|------|------|------|------|
| Test #1     | 7.08 | 6.87 | 6.79 | 6.85 | 7.20 | 7.10 | 6.77 | 6.85 | 6.81 | 6.79 |
| Test #2     | 0.04 | 0.03 | 0.04 | 0.04 | 0.04 | 0.03 | 0.05 | 0.05 | 0.03 | 0.04 |
| Test #3     | 7.08 | 6.88 | 6.79 | 6.86 | 7.20 | 7.10 | 6.77 | 6.85 | 6.81 | 6.79 |
| Test #4     | 0.84 | 0.59 | 0.68 | 0.81 | 0.60 | 0.55 | 0.70 | 0.82 | 0.60 | 0.67 |
| Test #5     | 7.72 | 7.40 | 7.55 | 7.56 | 7.69 | 7.60 | 7.35 | 7.56 | 7.33 | 7.40 |
| Test #6     | 0.86 | 0.61 | 0.70 | 0.84 | 0.62 | 0.57 | 0.72 | 0.84 | 0.62 | 0.69 |
| TOTAL RT    | 8.57 | 8.01 | 8.25 | 8.40 | 8.31 | 8.17 | 8.07 | 8.39 | 7.95 | 8.09 |

<sup>1</sup>In <https://github.com/KlosStepan/SwimmPair-Www> **dummy\_data\_benchmark.php**

## 4.2 User testing

We carried on testing of our application by handing SUS questionnaire to 20 respondents. We then evaluated the scores in order to find out how our application stands. These people are either managers or common referees <sup>2</sup>.

**Questionare is made of 10 questions scored 1-5.**

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

**We collected SUS feedbacks from 20 people using our system.**

| Respondent / Q. no. | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---------------------|----|----|----|----|----|----|----|----|----|-----|
| Petr A              | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Olga A              | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Marin H             | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Michaela H          | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Stepan K            | 4  | 2  | 4  | 1  | 3  | 1  | 3  | 1  | 5  | 1   |
| Matylda K           | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Lukas Kour.         | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Jana K              | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Lukas Kous.         | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Zuzana K            | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Eva K               | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Michael P           | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Lenka P             | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Daniela S           | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Magdalena S         | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Jiri S              | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Hana S              | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Alena T             | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Magda Z             | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| Vera Z              | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

<sup>2</sup>SUS bois [2000]



rest mbz depr.

```
//1. Create 98 Users – random affil to 1–15
$usersManager->RegisterUser($first_name, $last_name, $email, 12345,
    $rights[$rights_idx], $ranks[$rrid_idx]->id, $clubs[$club_idx]->
    id);
//2. Create 12 Cups
$scupsManager->InsertNewCup($scups_names[$scup_name_idx]."_".rand(1, 8)
    , "2023-".str_pad($j, 2, '0', STR_PAD_LEFT)."-26", "2023-".
    str_pad($j, 2, '0', STR_PAD_LEFT)."-28", $clubs[$club_idx]->id,
    $content[$content_idx]);
//3. Fetch new Users and Cups (&positions)
$users = $usersManager->FindAllActiveUsersOrderByLastNameAsc();
$scups = $scupsManager->FindAllUpcomingCupsEarliestFirst();
$positions = $positionsManager->FindAllPositions();
//4. Create Availabilities
$scupsManager->InsertNewAvailability($scups[$k]->id, $users[$user_idx[
    $kk]]->id, 1);
//5. Create Pairings (availabilities 1 random pos. for each)
$scupsManager->InsertNewPairing(($l+1), $positions[$position_idx]->id
    , $avails[$l]->id);
echo("6. Call stats queries (20 either Clubs/Users stat queryings)<
    br/>\r\n");
$personCupsCount = $usersManager->CountCupsAttendanceOfUserGivenYear
    ($users[$user_idx]->id, $year);
$stats_cups = $usersManager->CountOverallStatisticsOfUserGivenYear(
    $users[$user_idx]->id, $year);
$stats_users = $usersManager->CountClubSeasonalStatistics($clubs[
    $club_idx]->id, $year);
```

## SWIMMPAIR DUMMY DATA & BENCHMARK

### 1. Register Users #3-#100

Stop 1: 6.5730640888214 sec.

### 2. Insert Cups #1-#12

Stop 2: 0.052016973495483 sec.

### 3. Fetch Users and Cups (&Positions)

Stop 3: 6.5764610767365 sec.

### 4. Insert Availability (20 per Cup)

Stop 4: 1.1495687961578 sec.

### 5. Insert Pairing (availabilities 1 random pos. for each)

Stop 5: 7.5706551074982 sec.

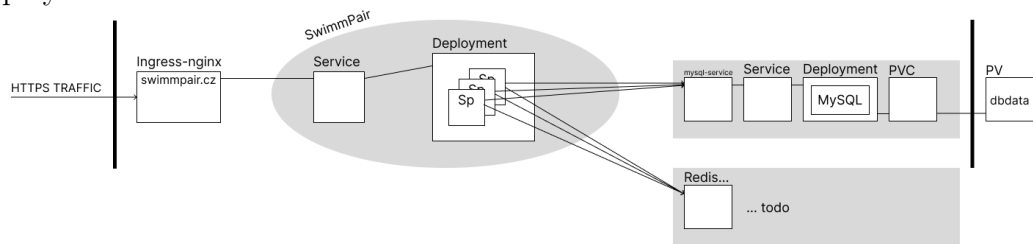
### 6. Call stats queries (20 either Clubs/Users stat queryings)

Stop 6: 1.1841568946838 sec.

TOTAL RUNTIME: 8.7548229694366 sec.

# 5. Deployment

Deployment of SwimmPair into Kubernetes cluster is realized as follows:



## Dockerization

Create Docker image and copy project appropriately into folder.

## Kubernetes

Orchestration of stuff

## Database and Redis

Navigate the cups, users and statistics as a user interested in public information.

## Application overview

### Cups printouts

### Public statistics

## 5.1 Administration

Show to different administrator tiers how to use this system.

### Participate in Cup

### Your Club

### Administrate SwimmPair as whole

# 6. User Manual

## 6.1 Public part

Navigate the cups, users and statistics as a user interested in public information.

### Application overview

text about what is where

### Cups printouts

example screen

### Public statistics

example screen

## 6.2 Administration

Show to different administrator tiers how to use this system. example screen of login 2/1/0

### Participate in Cup

sign self for cup

### Your Club

people i think, create cup, etc.

### Administrate SwimmPair as whole

# Conclusion

Endeavor of designing, development and shipping of this web application was overallly successful. There are some parts, that can be improved or extended in the future, however, our system is ready for this and lessons have been learned.

Speaking of lessons, design/development part was not as straightforward as theoretics would have appreciated, but rather it was done iteratively with cooperation of system requester.

The stages of iterations are roughly:

1. Problem description + basic pages layouts programming (homepage, cup, user, club).
2. Model formalization and proper division of code into system objects and task functions.
3. Statistics on data (user, club), additional pages for categorization purposes.
4. Addition of Regions for futher extensible hierarchisation.
5. Major final refactoring of database, backend and testing with dummy data insertion and querying.
6. Cloud ready, Docker image of web application and Kubernetes Deployment.

Further system extesions might be related to adding **new public pages, statistical queries, administrative tasks addition** - these are pages addition, model adjustment and minor design changes. All types of modifications can be accomodated thanks to divided code and the changes might consist of modifying style, model classes enhancement, database procedures addition etc. Future modifications will reside in the the project GitHub repository <sup>1</sup>.

---

<sup>1</sup><https://github.com/KlosStepan/SwimmPair-Www>

# Bibliography

- Inc Docker. Docker Documentation. on-line, 2023. URL <https://docs.docker.com/desktop/>. Accessed: 2022-12-27.
- ecma INTERNATIONAL. ECMA-262 ECMAScript® 2022 language specification. on-line, June 2022. URL <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>. Accessed: 2023-01-03.
- Oracle. MySQL Documentation. on-line, 2023. URL <https://dev.mysql.com/doc/>. Accessed: 2022-12-27.
- SUS bois. SUS Methodic. on-line, 2000. URL [https://www.researchgate.net/publication/228593520\\_SUS\\_A\\_quick\\_and\\_dirty\\_usability\\_scale](https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale). Accessed: 2023-02-28.
- The Kubernetes Authors. Kubernetes Documentation. on-line, 2023. URL <https://kubernetes.io/docs/home/>. Accessed: 2023-01-03.
- The PHP Group. PHP 7.4 Specification. on-line, 28 November 2019. URL [https://www.php.net/releases/7\\_4\\_0.php](https://www.php.net/releases/7_4_0.php). Accessed: 2022-12-27.
- W3C. CSS Specifications. on-line, 31 December 2022. URL <https://www.w3.org/Style/CSS/specs.en.html>. Accessed: 2022-12-27.
- WHATWG. HTML Living Standard. on-line, 26 December 2022. URL <https://html.spec.whatwg.org/>. Accessed: 2022-12-27.

# List of Figures

# List of Tables

# List of Abbreviations



# A. Attachments

## A.1 First Attachment