



**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

| | |
|---|-----------|
| Introduction | 3 |
| 1 Status quo and solution | 4 |
| 1.1 Problem description | 4 |
| 1.2 Target audience of our application | 5 |
| 1.3 Storyboards of tasks | 5 |
| 1.4 Model proposition | 6 |
| 1.5 Frontend practices | 9 |
| 2 Architecture overview | 10 |
| 2.1 Technologies | 10 |
| 2.2 Application flow | 11 |
| 2.3 Managers | 11 |
| 2.4 Start file | 13 |
| 2.5 Templating of web and administration | 14 |
| 2.6 Responsive layout | 15 |
| 2.7 Administrative tasks | 15 |
| 2.8 Club Manager tasks | 15 |
| 2.9 Referee tasks | 15 |
| 3 Implementation Documentation | 16 |
| 3.1 Database design | 16 |
| 3.1.1 Schema | 16 |
| 3.1.2 Object tables | 16 |
| 3.1.3 Relation tables | 18 |
| 3.1.4 Content adjustment tables | 20 |
| 3.2 Managers documentation | 21 |
| 3.2.1 PostsManager.php | 21 |
| 3.2.2 UsersManager.php | 21 |
| 3.2.3 ClubsManager.php | 22 |
| 3.2.4 CupsManager.php | 22 |
| 3.2.5 PositionsManager.php | 23 |
| 3.3 Application structure - files defined structure | 23 |
| 3.4 JavaScript functions documentation | 25 |
| 3.4.1 Previous post | 25 |
| 3.4.2 User statistics - year change | 25 |
| 3.4.3 Club statistics - year change | 26 |
| 3.4.4 Filtering referees | 26 |
| 4 Testing | 27 |
| 4.1 Performance testing | 27 |
| 4.2 User testing | 28 |
| 5 Deployment | 30 |

| | | |
|----------|------------------------------|-----------|
| 6 | User Manual | 31 |
| 6.1 | Public part | 31 |
| 6.2 | Administration | 31 |
| | Conclusion | 34 |
| | Bibliography | 35 |
| | List of Figures | 36 |
| | List of Tables | 37 |
| | List of Abbreviations | 38 |
| A | Attachments | 39 |
| A.1 | First Attachment | 39 |

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 stories of Microsoft and Apple. These companies put computers on our tables and iPhones in our pockets. This is, however, just brief overview of my motivation about 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 cuts or making extra cash. To save money, one didn't call a taxi but UBER operated by another individual on the other side. To make extra money one started renting an extra room at Airbnb. Distrust in banking industry and monetary policy created Bitcoin. It is not hard to see that some of these things are not as technically complicated as one would think. A good software engineer should be able to deploy MVP of each thing previously mentioned in couple of weeks or months.

Motivation

This thesis is a fullstack web application meant for my fellow friend to save him time for more important tasks which he has to perform as a chief swimming referee and club 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, insights and lessons. These lessons are hopefully going to help me in my future endeavors and career.

Software engineering is a crucial craftsmanship for delivering positive changes in contemporary world. Building things is this modern adventure.

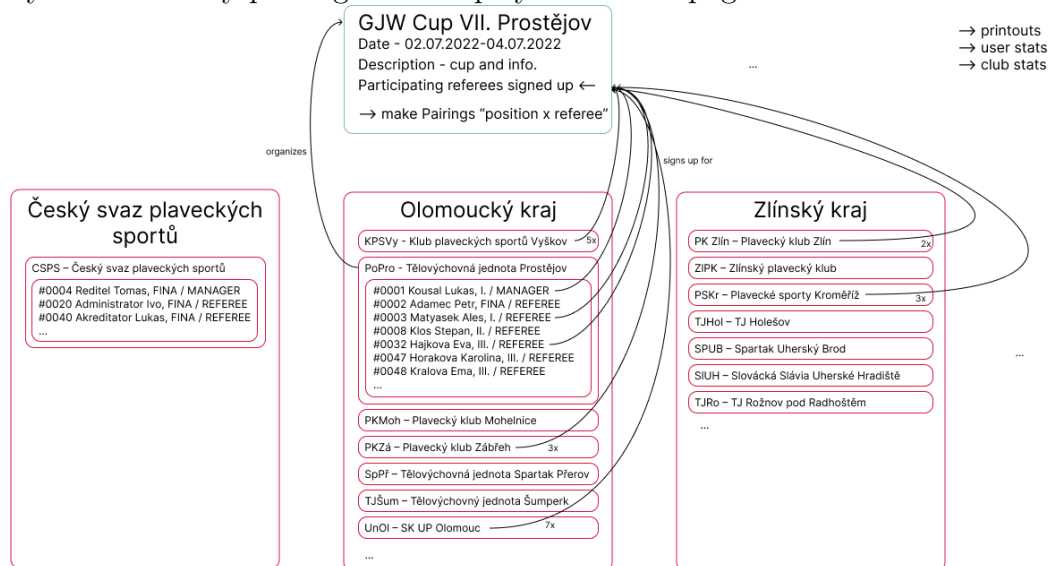
1. Status quo and solution

Section dedicated to description of problem and proposition of our 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 are executed in same straightforward order.

The Czech Swimming Federation ¹ structure has to be modeled as objects in the application and database records as a storage. Thus, logical structure should be decided and implemented. Swimming referees belong to clubs, clubs are located in geographical regions. Swimming cup is hosted by club. Each club contains dozen of swimming referees and one of them is a club manager. When a cup is online each referee can sign himself or herself up as available for the cup. Club manager can also sign up members of his club for to attend a cup. At the end of the day, organizer of the cup assigns available referees to positions (dedicated task-related roles during cup) that he finds them suitable for. My friend, the chairman of referee committee should be able to perform additional administration related to the database as whole - such as adding and removing users, creating new clubs and modifying whole structure. Administrator can also notify all visitors by posting news displayed on homepage.



The SwimmPair system should deliver public listing of all **users**, **cups**, **news**, **individual statistics** and **club statistics**. System should allow to browse stats on a yearly basis. Structure from this image then has to be appropriately modeled with objects. Proposition of database schema will be shown further down.

¹<https://www.czechswimming.cz>

1.2 Target audience of our application

Our users are Czech Swimming Federation members. If their **region** is **participating in this application**, clubs and referees from this region must be in our system. With regards to referee's competence level within these clubs, one will have one of these roles:

- **system administrator** ($\sim 1-3$)
- **club manager** + referee ($\sim 10s$),
- **referee** ($\sim 100s$).

Roles are self-descriptive and previously casually mentioned. My friend, who came up with this idea is **system administrator** because he's been keeping track all this agenda. Club managers are taking care of competitions on behalf of the club and referees are common people who have some degree of knowledge about competitions. Collected statistics will then be used for accreditation granting, activity monitoring and categorization overall.

We were iterating form and features of the web application with two future system administrators during time of development. We then tested usability on all three groups of users via. SUS ² questionnaire.

1.3 Storyboards of tasks

These are the 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 for accreditation renewal purposes,
- statistics of referees,
- statistics of clubs,
- referees managment,
- clubs managment,
- clubs overview,
- referees managment,

²**System Usability Scale** is a questionnaire to reveal how friendly tested system is to target audience. We carried on initial testing for 20 people belonging to one of these 3 categories to find out if we met at least an average score which was determined to be 68/100.

- referees overview,
- news publishing,
- overall categorization of federation,
- database archivation of federation.

1.4 Model proposition

Let's go through things that have to be represented in this system one by one starting from the most important ones. We will outlay objects, their relations and their relational connections. After that basic idea of entities and their relations will be established we continue to project specification for further implementation. We, however, worked more iteratively so this is just retrospective model for implementation that we were specifying during the time of development.

Cup

Cup is the most important object of SwimmPair. A swimming Cup contains name, description, date and is affiliated to organizing Club. Cup serves two purposes. **Firstly** - assigning referees for specific tasks (time tracking, computer support, head of the cup, etc.) 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 statistical 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 **system administrator**. User has to 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 email address as a login information is going to be used here. User must also contain SwimmPair hierarchy listed above³ and indicator of one's skill and knowledge in the swimming field⁴, i.e. referee category. User must also belong to exactly one club in our system.

Club

Club is an administrative unit grouping people (in the same city). Club has a 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

³**Rights** - referee 0 / club manager 1 / system administrator 2

⁴**Referee Rank** - 1/2/3/4/FINA - <https://www.czechswimming.cz/index.php/rozhodci>

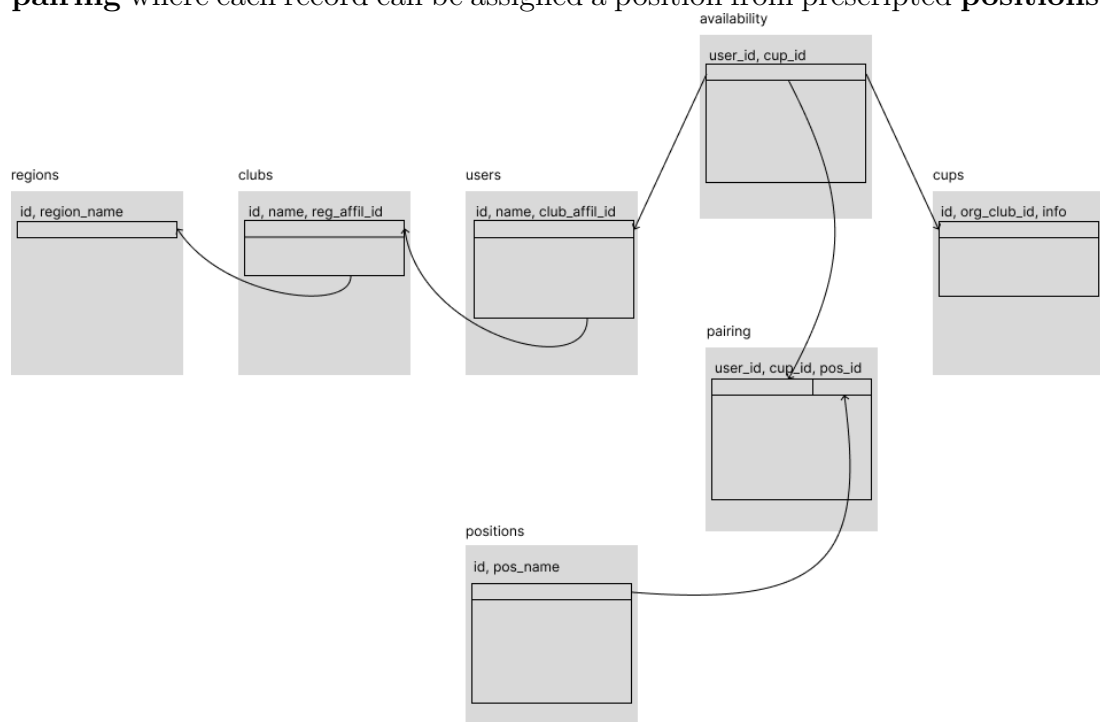
a User who is club manager. Club is unanimously affiliated to Region. Statistics regarding performance of members of Club at swimming competitions must be implemented. Statistics have informative characted and will save time in the current status quo - keeping track of presence and work descriptions in Excel spreadsheets.

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** will then be brought together into table called **availability** which contains referees that are available for specific cups. Availabile users for cup are then paired in **pairing** where each record can be assigned a position from prescribed **positions**.



Availability

Takes track of **User** to **Cup** availability of who is signed either by their team manager or themselves for the cup. Availability has extra flag going - in case User is paired but can't suddenly go then flag is switched to zero, in order to keep him in pairing but mark him as not going.

Pairing

Record from **Availability** is then taken and **Positions** is 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 in administration.

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 should 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 queries⁵ 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 printed⁶ 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.

⁵https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries

⁶https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries#targeting_media_types

2. Architecture overview

Reader will be familiarized with architecture of our application. There are two logical parts, **public web** and **private administration**. Private 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 headerer, 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 ¹ - application pages are templated in HTML by PHP,
- **CSS** is Cascading Style Sheets ²,
- **PHP** is a general-purpose scripting language geared toward web development ³ - object model and backend services are provided by it,
- **JavaScript** is a general-purpose scripting language that conforms to the ECMA Script specification ⁴,
- **MySQL** is an open-source relational database management system ⁵,
- **Git** is a distributed version control system: tracking changes in any set of files - this project is versioned and kept in public GitHub repository ⁶,
- **Docker** is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers ⁷ - used for deployment of our application,
- **Kubernetes** is an open-source container orchestration system for automating software deployment, scaling, and management ⁸ - used for production deployment of our application into cluster.

¹[WHATWG, 26 December 2022]

²[W3C, 31 December 2022]

³[The PHP Group, 28 November 2019]

⁴[ecma INTERNATIONAL, June 2022]

⁵[Oracle, 2023]

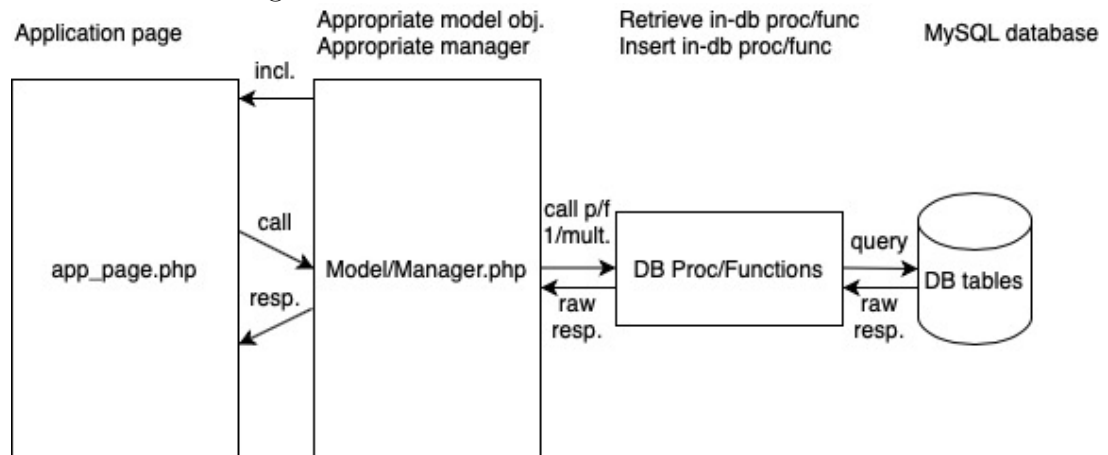
⁶<https://github.com/KlosStepan/SwimmPair-Www>

⁷[Docker, 2023]

⁸[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 as follows.



2.3 Managers

Managers are written to provide API functionality for system administration. 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.

- Cup / CupsManager
- User / UsersManager
- Club / ClubsManager
- Page / PagesManager
- **Post / PostsManager**
- 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.

Object - 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()
}

```

Manager - 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);
}

```

Objects and their managers are made in the same manner. They contain different set of data and different number public functions. Description is included in documentation chapter below.

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** by passing reference to live 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 inclusion*/
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**.

Access to different pages is then discriminated based on rights code on each page.

Rights check on each page in administration

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

Static requireRole function on Auth class for access permission

```
class Auth
{
    public static function requireRole($role)
    {
        if (!isset($_SESSION['rights']))
        {
            header('Location: ../prihlaseni.php');
            exit();
        }
        //Rights sharply lower than 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 tasks

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

3. Implementation Documentation

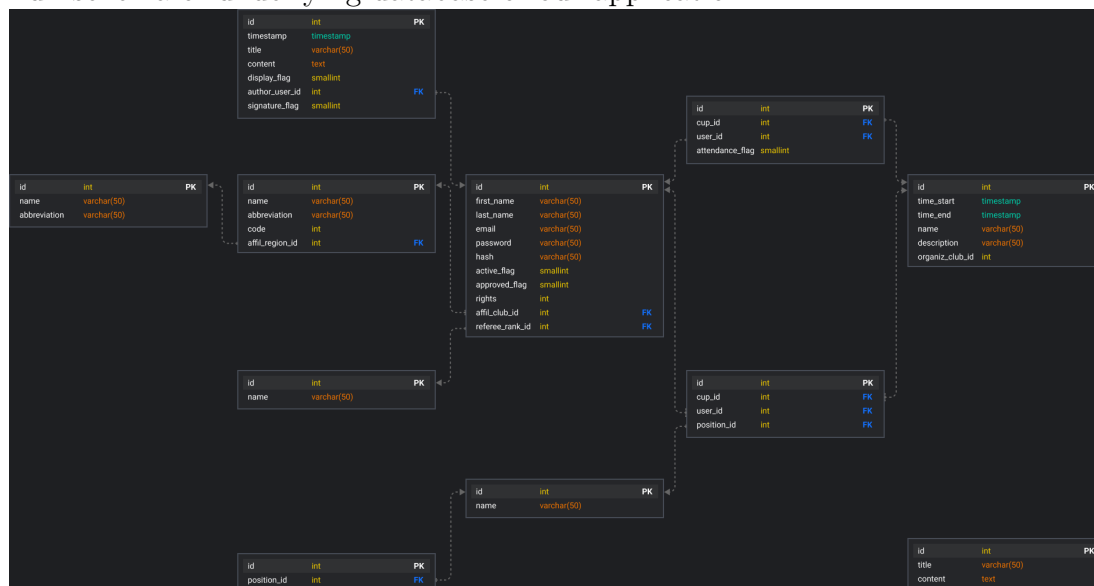
Detailed description of **database** and **backend components** and **functions**.

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

Full schema of underlying database of our application.



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

- $\underline{\text{Post}} \mid \text{null} \leftarrow \mathbf{GetPostById}(\$id)$
 $\searrow _CreatePostOrNullFromStatement(\$stmt)$
 $\searrow _CreatePostFromRow(\$row)$
- $\underline{\text{Post}}[] \mid \text{null} \leftarrow \mathbf{GetLastThreePosts}()$
 $\searrow _CreatePostsFromStatement(\$stmt)$
 $\searrow _CreatePostFromRow(\$row)$
- $\underline{\text{Post}}[] \mid \text{null} \leftarrow \mathbf{GetLastNPosts}(\$N)$
 $\searrow _CreatePostsFromStatement()$
 $\searrow _CreatePostFromRow(\$row)$
- $\underline{\text{true}} \mid \text{false} \leftarrow \mathbf{AddNewPost}(\$title, \$content)$
- $\underline{\text{Post}}[] \mid \text{false} \leftarrow \mathbf{FindAllPostsOrderedByIdDesc}()$
 $\searrow _CreatePostsFromStatement(\$stmt)$
 $\searrow _CreatePostFromRow(\$row)$
- $\underline{\text{true}} \mid \text{false} \leftarrow \mathbf{UpdatePost}(\$id, \$title, \$article)$

3.2.2 UsersManager.php

- $\underline{\text{User}} \mid \text{null} \leftarrow \mathbf{GetUserById}(\$id)$
 $\searrow _CreateUserOrNullFromStatement(\$stmt)$
 $\searrow _CreateUserFromRow(\$row)$
- $\underline{\text{User}}[] \mid \text{null} \leftarrow \mathbf{FindAllActiveUsersOrderByNameDesc}()$
 $\searrow _CreateUsersFromStatement(\$stmt)$
 $\hookrightarrow _CreateUserFromRow(\$row)$
- $\underline{\text{User}}[] \mid \text{null} \leftarrow \mathbf{FindAllInactiveUsersOrderByNameDesc}()$
 $\searrow _CreateUsersFromStatement(\$stmt)$
 $\hookrightarrow _CreateUserFromRow(\$row)$
- $\underline{\text{User}}[] \mid \text{null} \leftarrow \mathbf{FindAllRegisteredMatesForTheCup}(\$cupId, \$teamId)$
 $\searrow _CreateUsersFromStatement(\$stmt)$
 $\hookrightarrow _CreateUserFromRow(\$row)$
- $\underline{\text{User}}[] \mid \text{null} \leftarrow \mathbf{FindAllMates}(\$teamId)$
 $\searrow _CreateUsersFromStatement(\$stmt)$
 $\hookrightarrow _CreateUserFromRow(\$row)$
- $\underline{\text{User}}[] \mid \text{null} \leftarrow \mathbf{FindAllRegisteredUsersForTheCup}(\$cupId)$
 $\searrow _CreateUsersFromStatement(\$stmt)$
 $\hookrightarrow _CreateUserFromRow(\$row)$
- $\underline{\text{User}}[] \mid \text{null} \leftarrow \mathbf{FindAllNametagsForTheCup}(\$cupId)$
 $\searrow _CreateUsersFromStatement(\$stmt)$
 $\hookrightarrow _CreateUserFromRow(\$row)$

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

3.2.3 ClubsManager.php

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

3.2.4 CupsManager.php

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

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

3.2.5 PositionsManager.php

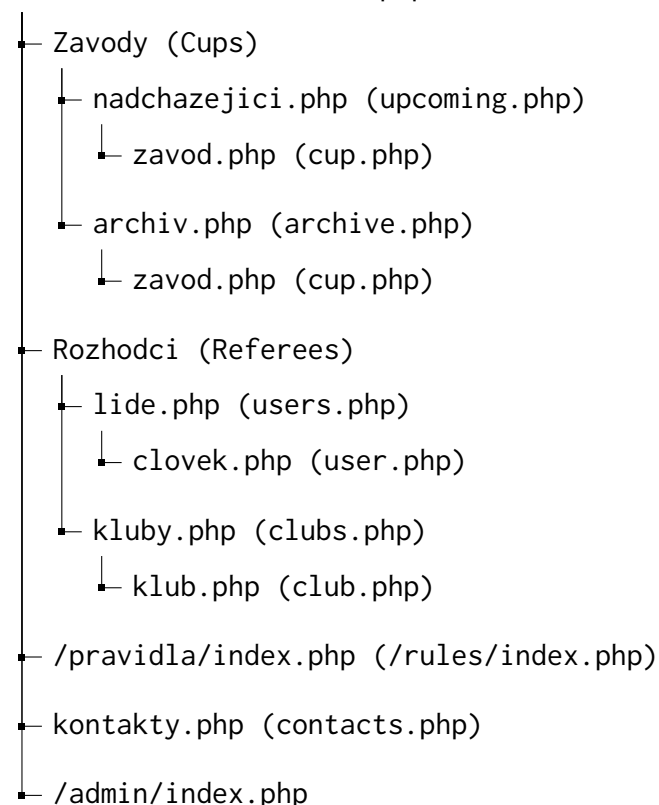
- $\text{Position}[] \mid \text{null} \leftarrow \text{FindAllPositions}()$
 $\searrow _CreatePositionsFromStatement(\$stmt)$
 $\hookrightarrow _CreatePositionFromRow(\$row)$
- $\text{string} \mid \text{null} \leftarrow \text{GetPositionNameById}(\$id)$
 $\searrow _GetSingleResultFromStatement(\$stmt)$

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



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...

AdamecPetrI.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**¹ 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 | 7.02 | 7.04 | 6.61 | 7.89 | 6.73 | 6.62 | 6.66 | 7.34 | 7.19 | 6.54 |
| Test #2 | 0.08 | 0.06 | 0.06 | 0.06 | 0.06 | 0.10 | 0.58 | 0.60 | 0.50 | 0.15 |
| Test #3 | 7.02 | 7.05 | 6.61 | 7.90 | 6.74 | 6.63 | 6.66 | 7.35 | 7.20 | 6.55 |
| Test #4 | 1.24 | 1.05 | 1.14 | 1.05 | 1.18 | 1.17 | 1.24 | 1.15 | 0.96 | 1.59 |
| Test #5 | 8.02 | 7.87 | 7.39 | 8.95 | 7.70 | 7.81 | 7.44 | 8.12 | 7.98 | 7.36 |
| Test #6 | 1.29 | 1.10 | 1.19 | 1.12 | 1.23 | 1.22 | 1.28 | 1.19 | 1.00 | 1.62 |
| TOTAL | <u>9.31</u> | <u>8.97</u> | <u>8.59</u> | <u>10.06</u> | <u>8.93</u> | <u>9.03</u> | <u>8.72</u> | <u>9.31</u> | <u>8.98</u> | <u>8.98</u> |

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 | <u>8.57</u> | <u>8.01</u> | <u>8.25</u> | <u>8.40</u> | <u>8.31</u> | <u>8.17</u> | <u>8.07</u> | <u>8.39</u> | <u>7.95</u> | <u>8.09</u> |

¹In <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 ².

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 calculated³ SUS feedbacks based on responses from 20 people.

| Respondent / Q. no. | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---------------------------|----|----|----|----|----|----|----|----|----|-----|
| Petr A - 87.5 | 5 | 1 | 3 | 1 | 5 | 1 | 3 | 1 | 5 | 2 |
| Olga A - 72.5 | 2 | 1 | 3 | 2 | 4 | 1 | 5 | 1 | 3 | 3 |
| Marin H - 75 | 3 | 1 | 4 | 2 | 5 | 1 | 3 | 2 | 3 | 2 |
| Michaela H - 60 | 2 | 3 | 3 | 4 | 3 | 2 | 4 | 2 | 5 | 2 |
| Stepan K - 85 | 5 | 2 | 4 | 1 | 3 | 1 | 3 | 1 | 5 | 1 |
| Matylda K - 80 | 4 | 1 | 4 | 2 | 4 | 1 | 4 | 2 | 4 | 2 |
| Lukas Kour. - 67.5 | 2 | 2 | 5 | 2 | 4 | 1 | 4 | 2 | 2 | 3 |
| Jana K - 60 | 1 | 2 | 3 | 2 | 5 | 2 | 3 | 2 | 2 | 2 |
| Lukas Kous. - 92.5 | 5 | 1 | 4 | 1 | 5 | 1 | 4 | 1 | 5 | 2 |
| Zuzana K - 70 | 3 | 2 | 5 | 1 | 3 | 1 | 3 | 2 | 3 | 3 |
| Eva K - 80 | 3 | 2 | 5 | 1 | 3 | 1 | 3 | 1 | 4 | 1 |
| Michael P - 75 | 2 | 1 | 4 | 3 | 4 | 1 | 4 | 1 | 3 | 1 |
| Lenka P - 70 | 3 | 2 | 5 | 1 | 3 | 2 | 3 | 2 | 3 | 2 |
| Daniela S - 77.5 | 3 | 2 | 5 | 2 | 3 | 1 | 4 | 2 | 4 | 1 |
| Magdalena S - 85 | 4 | 1 | 4 | 1 | 4 | 1 | 5 | 1 | 3 | 2 |
| Jiri S - 62.5 | 3 | 3 | 5 | 3 | 4 | 2 | 3 | 3 | 2 | 1 |
| Hana S - 80 | 2 | 2 | 4 | 1 | 5 | 1 | 4 | 1 | 4 | 2 |
| Alena T - 90 | 4 | 1 | 5 | 1 | 3 | 1 | 4 | 1 | 5 | 1 |
| Magda Z - 85 | 3 | 2 | 5 | 2 | 4 | 1 | 4 | 1 | 5 | 1 |
| Vera Z - 75 | 3 | 3 | 4 | 1 | 3 | 1 | 3 | 1 | 4 | 1 |

²John Brooke [1995]

³ $((A1-1)+(5-A2)+(A3-1)+(5-A4)+(A5-1)+(5-A6)+(A7-1)+(5-A8)+(A9-1)+(5-A10))*2,5$

Rest of this stuff mby **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);
$sstats_cups = $usersManager->CountOverallStatisticsOfUserGivenYear(
    $users[$user_idx]->id, $year);
$sstats_users = $usersManager->CountClubSeasonalStatistics($clubs[
    $club_idx]->id, $year);
```

This benchmark snippet probably **depr.**

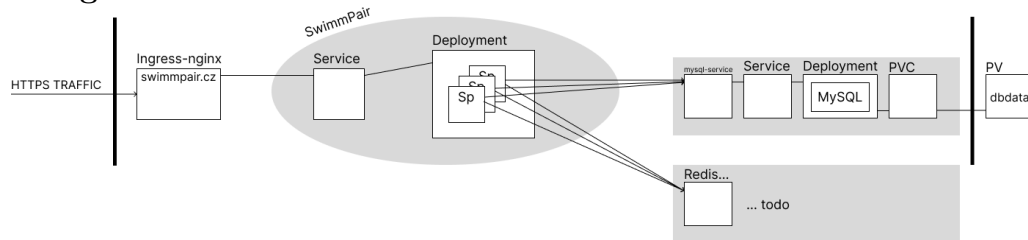
SWIMMPAIR DUMMY DATA & BENCHMARK

```
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

Development of our application was done locally - using **Docker compose** to glue up three components necessary to sufficiently run our system - PHP runtime for web application, MySQL Database and Adminer. We have to, however, run our application in Kubernetes Cluster. **Service** and **Deployment** have to be written, Service for purposes of routing and taking care of container spawn addresses and Deployment to describe container replicas and appropriate images. **Database** is run as separate entity within the Cluster.

Running SwimmPair in Kubernetes Cluster:



Dockerization of SwimmPair

File called **Dockerfile** has to be created in the project folder.

```
FROM thecodingmachine/php:7.4-v4-apache
COPY —chown=docker . /var/www/html
```

This image of Apache/PHP¹ was chosen because it correctly dockerizes part of so-called LAMP stack. In order to build this image and push it into Dockerhub.com we run these commands:

```
docker build -t stepanklos/swimmpair .
docker push stepanklos/swimmpair
```

This image is then pullable as stepanklos/swimmpair from Deployment.

Kubernetes

We run 2 replicas on 2 Nodes in order to ensure reliability and uptime of application.

Database and Redis

As mentioned before, our application doesn't come with Database and Adminer, which have to be set up separately using persistent storage PV on which we write PVC and reference from deployment.

¹Image **thecodingmachine/php:7.4-v4-apache** by TheCodingMachine - <https://github.com/thecodingmachine/docker-images-php>

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

SwimmPair administration after login has many options.



New Cup

 Přidat závod

Název závodu *

Datum konání od *

28.02.2023

Datum konání do *

28.02.2023

Pořadatel

Tělovýchovná jednota
Prostějov z.s.


Popis závodu *

Informace o závodu...

Pole označená * jsou povinná

Přidat

Participation availability

 O pohar Starosty 1

2023-04-26

Můj klub

Lukáš Kousal

Štěpán Klos

Sofie Fialova

Sofie Dvorakova

Stepan Hajek

Dominik Cerny


Přihlášení rozhodčí

Štěpán Klos


Sofie Dvorakova

Aktualizovat

Pairing



O pohar Starosty 1



2023-04-26

Pozn.: Rozhodčí na závody přiřadíte PŘETAŽENÍM. Dvojklikem člověka z role odeberete.

Dostupní rozhodčí

Marek Dvorak

Julie Dvorakova

Sofie Dvorakova

Tereza Dvorakova

Nela Fialova

Ondrej Horak

Jan Horak

Štěpán Klos

Adam Kral

Petr Kucera

Adela Kucerova

Lucie Nemcova

Filip Novak

Matyas Novak

Nela Novakova

Jiri Novotny

Katerina Pospisilova

Lucie Svobodova

Karolina Vesela

Matej Vesely

Vrchní rozhodčí

Lucie Svobodova

Startér

Pomocný startér

Julie Dvorakova

Petr Kucera

Hlasatel

Matej Vesely

Časomíra

Nela Fialova

Obsluha PC

Rozhodčí plav. způsobů

Katerina Pospisilova

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 ¹.

¹<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.
- John Brooke. SUS: A quick and dirty usability scale. on-line, 1995. URL https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale. Accessed: 2023-02-28.
- Oracle. MySQL Documentation. on-line, 2023. URL <https://dev.mysql.com/doc/>. Accessed: 2022-12-27.
- 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. 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