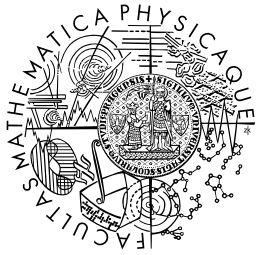


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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 2023

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

I would like to thank my supervisor doc. Mgr. Martin Nečaský, Ph.D. for his valuable advice and suggestions. Thanks to his guidance and expertise, I was able to take my knowledge to the next level. Each consultation was very interesting and provided valuable insights from a highly experience person in software engineering that I have internalized and continue to apply in my work.

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 project is to streamline the management process for swimming competitions in the Czech Republic. To achieve this, we are developing a system with a user-friendly web interface that is also mobile-friendly. The system will include all necessary infrastructure and utilize a MySQL database for data storage. Backend tasks will be handled by extensible PHP managers. For the frontend, we are implementing a custom drag-and-drop DOM API in JavaScript to improve the user experience.

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

Contents

Introduction

As someone born in the mid-1990s, I've had a front-row seat to the evolution of personal computing and the rise of the internet. Even as a three-year-old, I was captivated by my father's first computer, which ran Windows 98. By the age of five, I already knew I wanted to be a programmer when I grew up, after realizing that I could write code and create public websites. Over time, I became increasingly fascinated by the stories of technology giants like Microsoft and Apple, who brought computers to our desks and smartphones to our pockets. Of course, my interest in the IT world runs much deeper than this brief overview can convey.

Why web applications

The dot-com bubble crash in the early 2000s marked a necessary correction of the overhyped optimism surrounding new technologies, helping the industry as a whole to mature. However, it was the financial crisis of 2008 that truly opened up opportunities in the web space. While it left the average American customer poorer, it also spurred a new trend of money-saving services designed to help people cut costs or earn extra cash. Suddenly, instead of calling a taxi, people could use Uber to find a ride from an independent driver. And if they had a spare room, they could list it on Airbnb to make some extra money. Meanwhile, the lack of trust in the banking industry and monetary policy gave rise to Bitcoin and other cryptocurrencies. While some of these innovations may seem technically complex, a skilled software engineer could deploy a minimum viable product of each one in just a matter of weeks or months.

Motivation

I designed this thesis as a full-stack web application for a friend who works as a chief swimming referee and club manager, in order to help him save time for more important tasks. The process of developing this application has been invaluable training for me, as it required me to devise a solution to a problem that was similar in some ways to the minimum viable products mentioned earlier. Throughout this project, I gained a wealth of experience, insights, and lessons that I hope to apply in my future endeavors and career. As I see it, software engineering is a crucial craft for effecting positive change in the contemporary world. Building new things is an exciting adventure that holds endless possibilities.

Software engineering is a crucial craft for effecting positive change in the contemporary world, as it enables us to create new tools, platforms, and experiences that make people's lives easier, more productive, and more fulfilling. In a sense, building things has become a modern adventure, filled with opportunities to innovate, explore, and improve the world around us.

1. Status quo and solution

In this section, we will describe the problem we aim to address and present our proposed solution in the form of an application design.

1.1 Problem description

A friend of mine recently asked me if I could help him automate part of his work agenda. Administration of swimming competitions and creating statistics is very repetitive and error-prone array of tasks. All the tasks are executed in the same and straightforward order.

The Czech Swimming Federation [1] structure has to be layed out and subsequently modeled as objects in the application and interface with database via managers (ie. APIs for the application). Thus, a logical structure has to be prepared and implemented.

Swimming referees belong to clubs. Clubs are located in geographical regions. Swimming cup is hosted by a club. Each club contains dozens of swimming referees and one of them must be a club manager. Once a cup is scheduled, referees can sign up to make themselves available for it. Club manager can also sign members of his club up to be available for a cup. At the end of the day, the cup organizer assigns referees to specific task-related positions based on their availability and suitability.

The chairman of referee committee (my friend) should be able to perform additional administration related to the whole dataset - be it adding and removing users, creating new clubs or modifying whole structure. Administrator can also notify all visitors by posting news displayed on homepage.

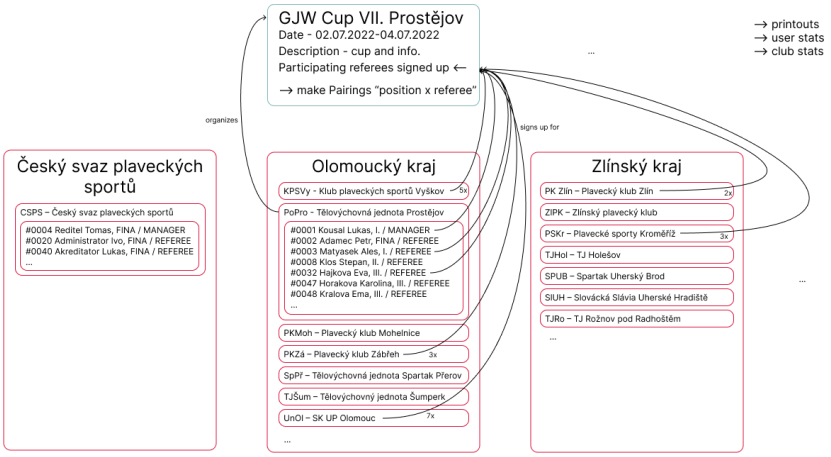


Figure 1.1: Preview of region-club-user grouping & availability for the cup.

The SwimmPair application should deliver public listing of all **users**, **cups**, **news**, **individual statistics** and **club statistics**. Application should allow to browse statistics on yearly basis. Structure (Figure 1.1) then has to be appropriately modeled as entities (Figure 1.2). Proposition of entities-objects mappings (Figure 2.6 Figure 2.7) and database (Figure 2.8) will be shown further down.

[1] `https://www.czechswimming.cz`

1.2 Stakeholders

Groups direclty and indirectly interested in existence of this application and breakdown of its active/passive users.

Interest groups

There are several entities that are interested in existence of this application. All these stakeholders will have their job facilitated and better organized to some extent thanks to existence of this application.
Interested stakeholders are:

- **Czech Swimming Federation** - organization for competitive swimmers,
- **Olomouc Region**, **Zlin Region** - CSF regions administered together,
- **Lukas K.** from **TJ Prostejov** - coordinator who asked for this application.

Users of the application

Users of our application will be Czech Swimming Federation members. If their **region** is **participating in this application**, clubs and referees belonging to this region must be in our system. Depending on their rank within the structure, individuals will be assigned one of these roles:

- **system administrator** (∼1-3),
- **club manager** + also a swimming referee (∼10s),
- **swimming referee** (∼100s).

Roles are self-descriptive. The coordinator (my friend), who came up with this idea for this application will be **system administrator** because he's been running all this agenda in Excel spreadsheets. **Club managers** are taking care of competitions on behalf of their club and **swimming referees** are common people who have some degree of knowledge about competitions and can participate as referees.

The collected statistics will be used for accreditations granting, monitoring activity, and overall categorization.

During the development process, we worked with two coordinators (who will also serve as system administrators) to iterate on the requirements and features for the application.

After the development was over, we performed test usability on **all three groups of users** via. SUS[2] questionnaire. We then version controlled, unit tested and CI pipelined [3] our application. Despite advancements in automation, deployments are still performed manually.

[2]**System Usability Scale** is a questionnaire to reveal how friendly a 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.

[3]Brief overview of CI/CD pipelining `https://resources.github.com/ci-cd/` on GitHub.

1.3 Functional requirements

There are areas of similar tasks that we would like address and solve by our application by implementing features and ui pages for these purposes.

”C” as Cup administration

1. [club manager] needs to [create swimming cup] in order to [publish cup and invite others to participate]
2. [club manager] needs to [create pairing for swimming cup] in order to [finalize preparations of the cup the day before it takes place]
3. [club manager] needs to [preview cups and print pairing] in order to [perform inspection and publish information offline]
4. [club manager] needs to [participate in cup or participate with teammates] in order to [help swimming cup to take place]

”R” as Referees administration & overview

1. [club manager] needs to [manage swimming club] in order to [keep information and users up to date]
2. [club manager] needs to [perform referees managment] in order to [keep referees up to date]
3. [referee] needs to [view statistics of referees] in order to [have track record about personal participation]
4. [club manager] needs to [view statistics of clubs] in order to [have information about performance within one’s own club]
5. [club manager] needs to [perform club managment] in order to [keep own club up to date]
6. [system administrator] needs to [manage referees] in order to [add, remove, update users in the dataset]
7. [system administrator] needs to [list referees overview] in order to [see activity of referees]

”S” as Stakeholders interests

1. [system administrator] needs to [have overview of clubs] in order to [be informed about various things]
2. [stakeholder] needs to [have overall categorization of federation] in order to [use application for administrative purposes]
3. [stakeholder] needs to [have database archivation of federation] in order to [use application as an archivation tool]

4. [stakeholder] needs to [have information about participations] in order to [grant accreditations to referees for next season]

5. [system administrator] needs to [publish news] in order to [notify everybody about important whereabouts]

6. [system administrator] needs to [edit page/s] in order to [change static public info in the application]

1.4 Domain model

Let's look at the entities which have to be represented in our system one by one - starting from the most important ones. We will outlay entities and their relations. After basic idea of entities and their relations is established we proceed to project specification to delve further into the implementation details. Our approach was, however, more iterative - this retrospective domain model represents only a snapshot of the implementation details that we were specifying throughout our development process.

Cup

Cup is the most important entity. A swimming **cup** contains name, description, date and is affiliated to organizing club. Cup serves two purposes. **Firstly**, referees assignment for specific tasks (time tracking, computer support, main organizer, etc.) has to be **ready by the time the event takes place**. **Secondly**, statistics summing up participations of referees and clubs have to be calculated for each year from all cups in this time period. We also have to discriminate between upcoming and already past cups. Upcoming cups should be displayed, past cups should reside in archive to be revisited for statistical purposes.

Referee

Referee is a person and a main workforce during swimming cup. A referee is a member of club who participates on the club's behalf in cups. A referee's level of expertise is described by one of the several ranks[4]. Referee is assigned to one or more Positions, such as **Timekeeping** or **PC Support** and is in charge of the task for duration of the cup.

Club manager

Club manager is usually one person who is in charge of club administration. Club manager organizes cup on behalf of club and acts as a main figure during it. Club managers may also assist with certain tasks, but their role is primarily administrative. The club manager is responsible for pairing referees with suitable positions and creating work schedules for all participants.

[4]**Referee Rank** - 1/2/3/4/FINA at `https://www.czechswimming.cz/index.php/rozhodci`

Coordinator

The coordinator is the head of swimming in a specific region. They prepare budgets, plan tournaments and manage the administration and database of referees, clubs, and cups. They are the most important person in the administrative hierarchy of swimming.

Club

A club is an administrative unit that groups people together in the same city. The club has a specific name, abbreviation, and id in the Czech Swimming Federation, and a club logo can be included as well. The club serves as the formal authority organizing the cup by a user who is the club manager. The club is affiliated to the region. Performance statistics of members of the club at swimming competitions must be implemented. These statistics have an informative character and will save time compared to the current status quo of keeping track of presence and work descriptions in Excel spreadsheets.

Region

There are 13 regions in the Czech Republic, and we are currently addressing this problem for two regions that are being administered together. Other regions may also choose to join. Each club is located in one of these regions. When a new club starts using SwimmPair in the future, a new region will need to be added and potential clubs created and attached to this region.

Pairing

Pairing is simple list of pairs (**position** x **referee**) $\rightarrow$ **cup**.

Position

Predefined list of tasks necessary to be taken care of 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.

Schema of entities and their connections

Majority of focus should be on **referee**, **club manager** and **cup** in the presented schema(Figure 1.2). Referees belong to clubs which belong to regions. These two entities **referee** and **cup** along with **position** will then be brought together as **pairing** which contains referees that are available for specific cups and will be performing work at a specified position. Referees must be available in specific time but **pairing** is where each record can be assigned a position from prescripted **positions**.

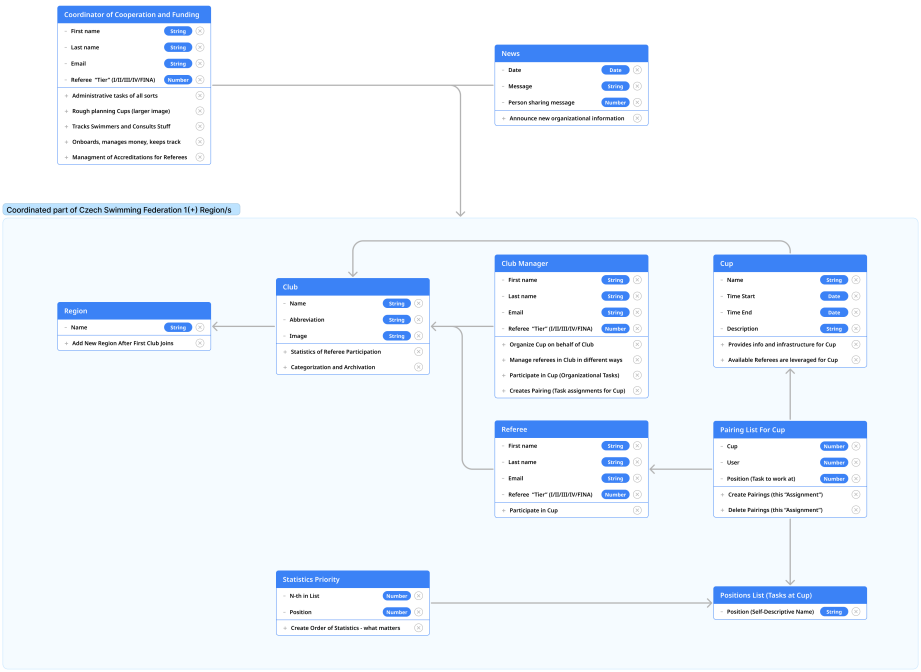


Figure 1.2: UML Class Diagram outlaying the administrative structure.

1.5 Quality/Usability Requirements

Several good practices have to be implemented to make SwimmPair easy to use. Although some of these practices are well-known and others are situation-specific, they all share a common goal: to improve the usability of the application.

Smooth frontend browsing

To make SwimmPair user-friendly, the frontend should be designed to be easy to use. This includes reducing page reloads, which can be achieved through asynchronous JavaScript calls to obtain partial data. The obtained data can then be used to modify the DOM using the appropriate functions.

Multiple device types

It is certain that there are users who want to browse our application on a PC, tablet, or smartphone so a responsive design is a necessity. CSS3 supports media queries[5], which we will use to create 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[6] from website and hanged as data printout.

Appropriate design

Red, blue, and gray are common colors found at swimming pools, and they will be used in our system's design. Our UI elements should have a fresh, lightweight look that doesn't feel heavy or overwhelming to users. Specifically, we'll use these colors to create an interface that's easy to navigate and visually appealing. For example, we might use blue to highlight important information, or red to indicate errors or warnings. By using these colors consistently throughout the system, we can create a cohesive and user-friendly experience for our users.

1.6 Scalability/Usability Requirements

Application is initially going to be used for 2 regions and approximately 100 referees. Maximum saturiation would mean that system is used in the whole Czech Republic. Maximum traffic is then 5-6x larger load than the current one.

Potentially looming issues:

- **performance issues** - shouldn't be problem using LAMP stack,
- **larger traffic** - can be solved via Kubernetes autoscaling [7],
- **simulaneous application edits** - can be solved by locking and comparing hashes of states before comitting changes to database,
- **sessions consistency** - can be solved (even throughout cluster restart) using Redis[8] for (persistent) session handling.

These issues can be easily tackled if they are kept in mind during the development.

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

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

[7] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[8] https://redis.io

2. System design

This chapter aims to familiarize the reader with architecture of our application. The system consists of two logical parts: **public website** and **private administration**. The private administration is secured with simple login/password, allowing access only to authorized personnel.

When designing such system, an object-oriented approach and grouping of similar functions together is a must. There are **objects that have to be moved around our web application**, as described in the previous chapter. These objects will be: post, user, club, cup, position, and region. Thus, we came up with a **concept of managers**. Each page of SwimmPair is composed of the same/unified header, menu, and footer. The content part is filled with the page's specific results of manager calls used to construct the data UI page layout. These managers are included and used in all pages via the start file.

2.1 Technologies

There are several technologies and tools used for creating the application.

- **HTML** is HyperText Markup Language [1] - application pages are templated 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 delivered by PHP.
- **JavaScript** is a general-purpose scripting language that conforms to the ECMAScript 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 project 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.
- **GitHub Actions** is a platform for automating workflows in GitHub repositories - our application uses GitHub Actions to automate testing. Specifically, we checkout with **actions/checkout@v3** and then run PHP using **shivammathur/setup-php@v2** for our backend tests using PHPUnit,

[1] [WHATWG, 26 December 2022]
[2] [W3C, 31 December 2022]
[3] [The PHP Group, 28 November 2019]
[4] [INTERNATIONAL, June 2022]
[5] [Oracle Corporation, 2023]
[6] `https://github.com/KlosStepan/SwimmPair-Www`
[7] [Docker, Inc., 2023]

along with a **docker container of our database filled with database schema** for integration testing. After each push we can see success of CI wokload run.

- **Kubernetes** is an open-source container orchestration system for automating software deployment, scaling, and management [8] - used for production deployment of our application.
- **Redis** is an in-memory data store often used for caching, messaging, and session persistence. Our application uses Redis to store session data and share it between running container instances, allowing traffic to be efficiently redirected and load-balanced.

[8] [The Kubernetes Authors, 2023]

2.2 Architecture overview

A visitor comes to the **app page** where **managers** are included. From the page, there are API calls on managers that retrieve and store data as follows (Figure 2.1).

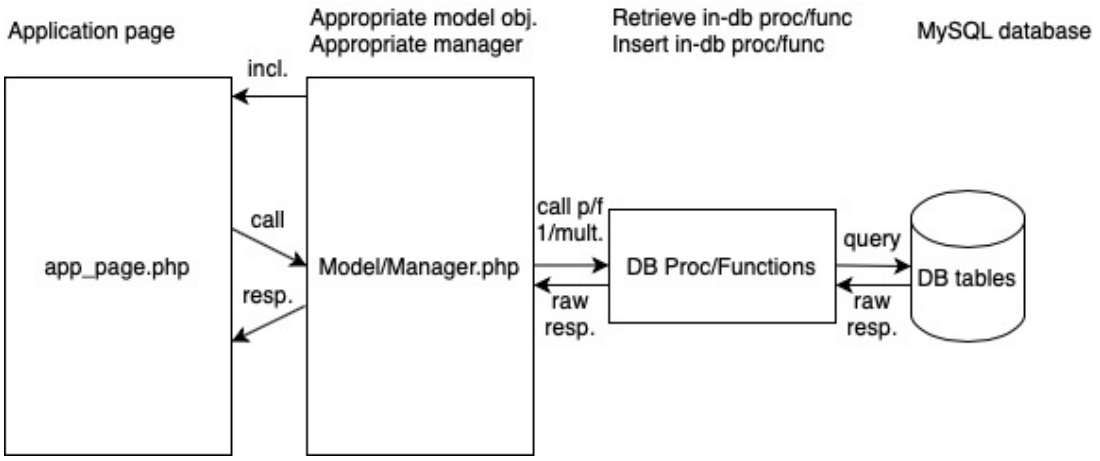


Figure 2.1: From page to manager-database function-database and back.

2.3 Model Managers

Managers are designed to provide API functionality for the system administration. Managers populate pages with data and process new input from users, administering the storage of this data. Each object has a corresponding manager responsible for handling it, which helps to accommodate database loads and stores. These operations are controlled by transactions to ensure data consistency.

Classes and their corresponding managers:

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

Managers are designed to retrieve and store data related to the class they are named after.

2.4 User Interface mockups

In this chapter, we present UI mockups for both public and private parts of our application. These mockups serve as an initial visualizations of the real UI and should not be considered exact guidelines. Instead, they provide a starting point for readers and stakeholders to understand the overall direction of the UI design.

Public website mockups

This part is concerned with displaying view-only data for public access.

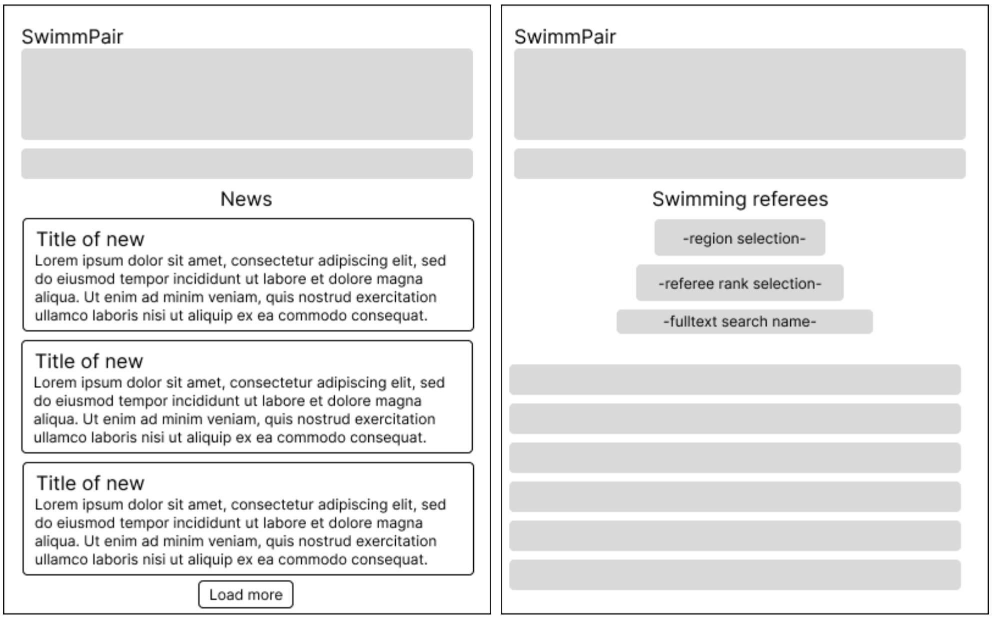


Figure 2.2: Public pages - homepage (S5) and listing of users (R3/S4).

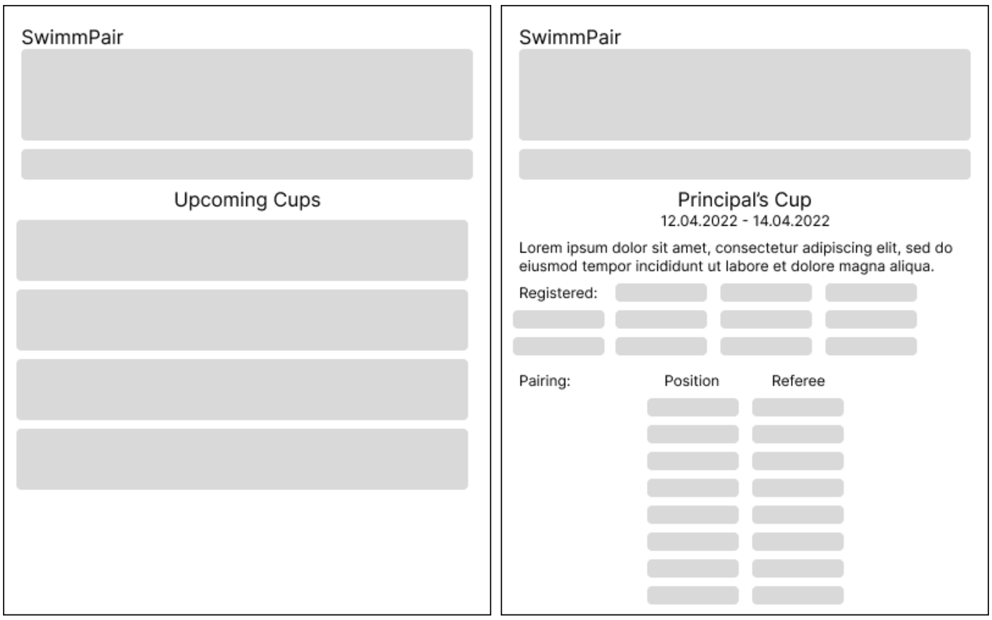


Figure 2.3: Public pages - cups listing and cup preview (C3).

Administration mockups

After logging in, the user can access the administrative menu, which is tailored to their level of access (2/1/0). This menu provides a list of administrative functions related to various aspects of the application. In the following sections, we will present mockups that demonstrate how these functional requirements may be programmed and designed.

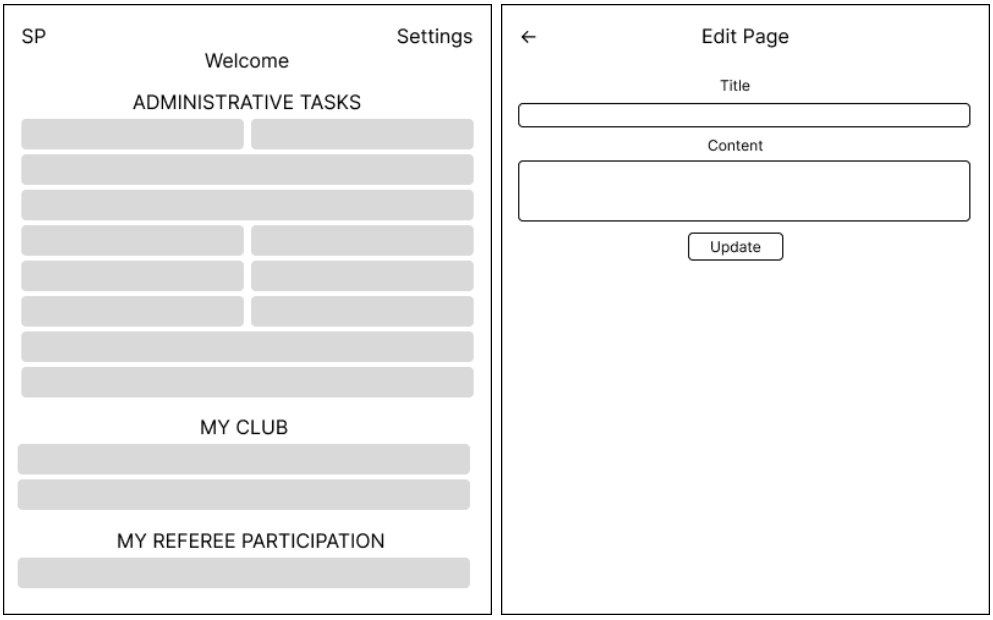


Figure 2.4: Administration menu gets assembled for rights, page edit (S6).

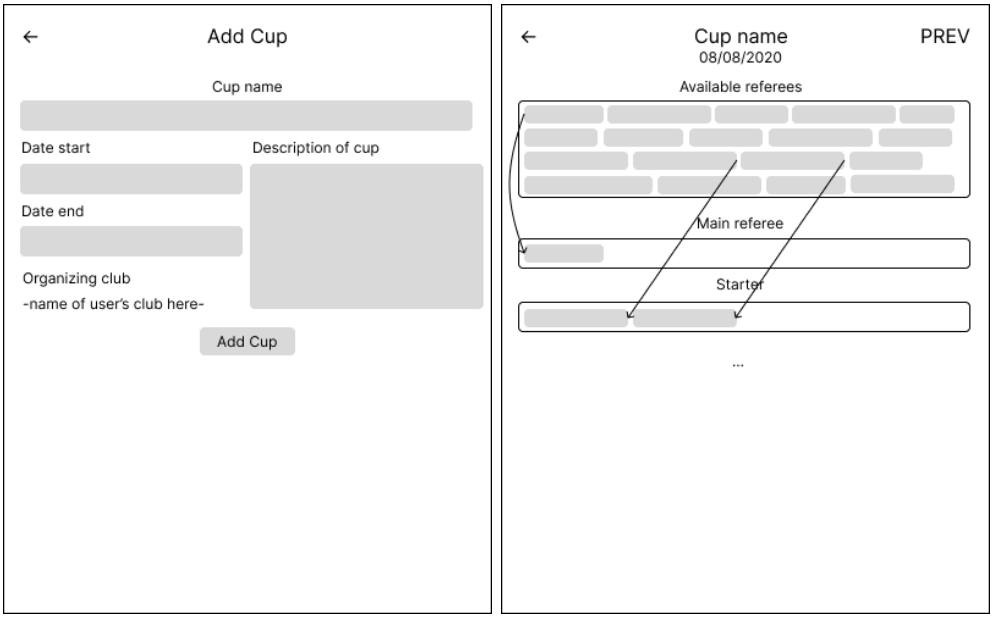


Figure 2.5: Add Cup (C1) and drag'n'drop pairing (C2).

2.5 Database design

In this chapter, we will discuss the importance of a well-defined database schema that accurately models the functional requirements and entities (Figure 1.2) of the system. To achieve this, we will illustrate the process of converting real-world requirements into a rigorous database schema, highlighting the necessary mappings and merges. By doing so, we aim to design a robust system that accurately represents the needs of the stakeholders.

People entities merge

We've merged three entities (**coordinator**, **club manager**, and **referee**) that model people into a single object called user, which includes additional parameters such as access rights and club affiliation to be able to distinguish between them.

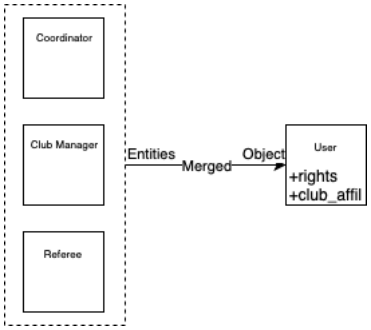


Figure 2.6: These 3 people entities got merged into one single object.

Pairing of entities as 3-tuple object

Majority of logic here is modelled as entities converted to objects. Furthemore, the **availability** must preceed **pairing** in the process.

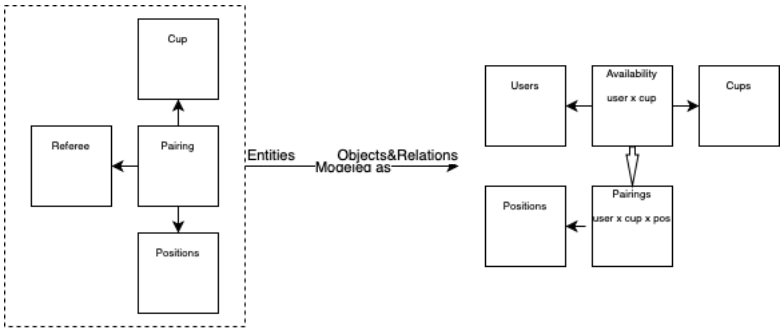


Figure 2.7: Entities modeled as 3 objects and 2 relational tables.

Remaining entities

The process of modeling the remaining entities as objects is relatively straightforward and self-evident. For example News from homepage are objects called Post and Regions are simply Region objects; and so on...

Full database schema used for our application

Based on all these merges, conversions and connections we can assemble a full database schema (Figure 2.8) with all columns for fields in the objects.

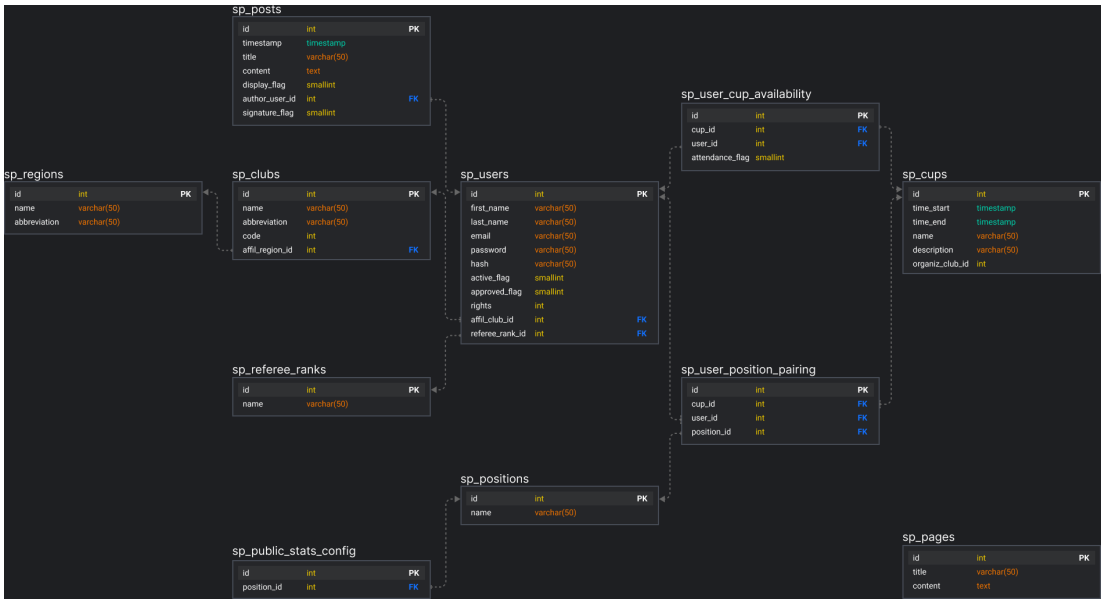


Figure 2.8: Full database schema of our application.

2.6 Functional requirements mapping to API

We will demonstrate how specific administrative tasks, as outlined in the functional requirements, are achieved through the use of model API functions [9].
Table has following structure: **Task - Role - Function(s)**.

Add Post	system admin	InsertNewPost
Edit Post	system admin	UpdatePost
Approve New Users	system admin	SetApprovedForUser
Create Pairing For Cup	system admin	DeleteOldPairing InsertNewPairing
Add User	system admin	RegisterUser
Edit User	system admin	SetLoginEmailForUser SetRefereeRankForUser SetPasswordForUser
Add Club	system admin	InsertNewClub
Edit Club	system admin	UpdateClub
Add Region	system admin	InsertNewRegion
Edit Region	system admin	UpdateRegion
Configure Stats	system admin	DeleteOldStatsPositions InsertNewStatPosition
Edit Contacts	system admin	UpdatePage
Add Cup	club manager	InsertNewCup
Sign People From My Club Available For Cup	club manager	DeleteOldAvailability InsertNewAvailability
Sign Myself As Available For Cup	referee	SetAvailabilityRegister SetAvailabilityCantGo SetAvailabilityCanGo

[9] http://docu-swimmpair.stkl.cz/functions_func.html

3. Implementation Documentation

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

3.1 Database

Full database schema of the application was already presented to the reader in previous chapter (Figure 2.8). This chapter will introduce the reader to detailed description of tables and their relations.
Each database table is accompanied with an example table preview, where:

- abbreviated column name: (a),
- contracted/ommited info: ...,

followed by full listing of attributes - datatypes, keys, etc.

3.1.1 Object tables

These are the tables in database modeling of objects to satisfy the primary motivation defined as the (Figure 1.2). These row/s are then being converted to **Object** or **Array<Object>** (Club, Cup, Page, PairPositionUser, Position, Post, RefereeRank, Region, StatPositionCnt, StatUserCnt, User) and returned to the application page by appropriate Manager (Figure 2.1).

sp_posts

Post is a static snippet of news for homepage of the web application.
Table preview

id	timestamp	title	content	d_flag(a)	auth_id(a)	sign(a)
1	2023-01-...	Running...	SwimmPair...	1	21	0
2	2023-03-...	Updates	This web...	1	21	0
...	...	...	...	...	...	...

Columns description

1. **id**|**PK**, int(11) *Auto Increment*
2. **timestamp**, datetime *NULL* [**CURRENT_TIMESTAMP**]
3. **title**, text
4. **content**, text
5. **display_flag**, tinyint(1)
6. **author_user_id**|**FK**, int(11) *NULL*
7. **signature_flag**, tinyint(1)

sp_users

In the system, a **User** is usually a referee affiliated to the **Club**, with a **user right** and a **tier** from the **Referee rank** list (Figure 2.6). The User's statistics are presented in public sites via the PositionsManager and the StatUserCnt.
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) *Auto Increment*
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_flag**, tinyint(1) [**0**]
8. **approved_flag**, tinyint(1) [**0**]
9. **rights**, tinyint(1)
10. **referee_rank_id**|**FK**, int(11)
11. **affiliation_club_id**|**FK**, int(11)

sp_clubs

A Club is an administrative unit that groups bunch of users together within the swimming application (and the real world). One User has the ClubManager/1 from UserRights, while the others have the Referee/0 from UserRights.
Table preview

id	name	abbrev(a)	club_id	img
1	Klub plaveckých sportů Vyškov	KPSVy	614	vyskov.jpg
...	...	...	...	...
14	TJ Rožnov pod Radhoštěm	TJRo	0	roznov.jpg

Columns description

1. **id**|**PK**, int(11) *Auto Increment*
2. **name**, varchar(80)
3. **abbreviation**, text
4. **code** int(11) *NULL*
5. **img**, text *NULL*
6. **affiliation_region_id**|**FK**, int(11)

sp_cups

This table stores information about **Cups**, including their name, start and end dates, location, and associated Club.
Table preview

id	t_st(a)	t_e(a)	name	desc(a)	org_c_id(a)
1	2023-...	2023-...	GJW Cup I.	Cup organized by ...	2
...	...	...	...	...	...

Columns description

1. **id**|**PK**, int(11) *Auto Increment*
2. **time_start**, date
3. **time_end**, date
4. **name**, text
5. **description**, text
6. **organizer_club_id**|**FK**, int(11)

sp_positions

A **Position** is an object within the system that represents a task to be performed by a User for a given Cup. Each Position is assigned a unique internal ID that is used to link it to other objects within the system.
Table preview

id	name
1	Vrchní rozhodčí
...	...
19	Ostatní

Columns description

1. **id**|**PK**, int(11), *Auto Increment*
2. **name**, varchar(45)

3.1.2 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_user_cup_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) *Auto Increment*
2. **cup_id**|**FK**, int(11)
3. **user_id**|**FK**, int(11)
4. **attendance_flag**, tinyint(1) [**1**]

sp_user_position_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(20) *Auto Increment*
2. **cup_id**|**FK**, int(11)
3. **position_id**|**FK**, int(11)
4. **user_id**|**FK**, int(11)

3.1.3 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 **sp_positions.name**.
Table preview

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

Columns description

1. **id**|**PK**, int(11) *Auto Increment*
2. **position_id**|**FK**, int(11)

sp_pages

Page is a static website page with information in our application. It has a title and content.
Table preview

id	title	content
1	Kontakty	Title Contact information +420...

Columns description

1. **id**|, int(11) *Auto Increment*
2. **title**, text
3. **content**, text

3.2 Managers documentation

These five managers work with objects and provide views and functions (i.e. joining more tables in various ways to achieve all functionality). We are providing an overview of which API functions are calling which internal functions (plain function call ↘, repeated calls in cycle ↪) and what are desired return values. **Documentation of model - classes and public functions** [1].

3.2.1 PostsManager.php

PostsManager has API functions to handle Post object/s and delivers it through web application.

- Post | null ← **GetPostById**($id)
  ↘ _CreatePostOrNullFromStatement($stmt)
  ↘ _CreatePostFromRow($row)
- Post[] | null ← **FindLastNPosts**($N)
  ↘ _CreatePostsFromStatement($stmt)
  ↪ _CreatePostFromRow($row)
- true | false ← **InsertNewPost**($title, $content, $d_flag, $auth_id, $sign)
- Post[] | false ← **FindAllPostsOrderedByIdDesc**()
  ↘ _CreatePostsFromStatement($stmt)
  ↪ _CreatePostFromRow($row)
- true | false ← **UpdatePost**($id, $title, $content, $d_flag, $sign)

3.2.2 UsersManager.php

UsersManager has API functions to handle User object/s and delivers is through web application.

- User | null ← **GetUserById**($id)
  ↘ _CreateUserOrNullFromStatement($stmt)
  ↘ _CreateUserFromRow($row)
- User[] | null ← **FindAllActiveUsersOrderByLastNameAsc**()
  ↘ _CreateUsersFromStatement($stmt)
  ↪ _CreateUserFromRow ($row)
- User[] | null ← **FindAllInactiveUsersOrderByLastNameAsc**()
  ↘ _CreateUsersFromStatement($stmt)
  ↪ _CreateUserFromRow ($row)
- User[] | null ← **FindAllRegisteredTeamMembersForTheCup**($cupId, $teamId)
  ↘ _CreateUsersFromStatement($stmt)
  ↪ _CreateUserFromRow ($row)

[1] Additional documentation for this work can be found at `http://docu-swimmpair.stkl.cz`, or as a PDF attachment (also accessible here `https://github.com/KlosStepan/SwimmPair-Www/blob/master/_doc/pdf/refman.pdf`), providing further details.

- User[] | null ← **FindAllTeamMembers**($teamId)
  ↘ _CreateUsersFromStatement($stmt)
  ↪ _CreateUserFromRow($row)
- User[] | null ← **FindAllRegisteredUsersForTheCup**($cupId)
  ↘ _CreateUsersFromStatement($stmt)
  ↪ _CreateUserFromRow($row)
- User[] | null ← **FindPairedUsersOnCupForPosition**($cupId, $posId)
  ↘ _CreateUsersFromStatement($stmt)
  ↪ _CreateUserFromRow($row)
- string | null ← **GetClubAbbreviationByAffiliationId**($id)
  ↘ _GetSingleResultFromStatement($stmt)
- string | null ← **GetUserFullNameById**($id)
  ↘ _GetSingleResultFromTwoColsStatement($stmt)
- true | false ← **IsEmailPresentAlready**($email)
- true | false ← **RegisterUserFromAdmin**($first_name, $last_name, $email, $password, $rights, $klubaffil)
- true | false ← **EmailNewPersonRegistered**($email, $password)
- true | false ← **SetApprovedForUser**($userId)

3.2.3 ClubsManager.php

ClubsManager has API functions to handle Club object/s and delivers it through web application.

- Club | null ← **GetClubById**($id)
  ↘ _CreateClubFromStatement($stmt)
  ↘ _CreateClubFromRow($row)
- Club[] | null ← **FindAllClubs**()
  ↘ _CreateClubsFromStatement($stmt)
  ↪ _CreateClubFromRow($row)

3.2.4 CupsManager.php

CupsManager has API functions to handle Cup object/s and delivers it through web application.

- Cup[] | null ← **FindAllUpcomingCupsEarliestFirst**()
  ↘ _CreateCupsFromStatement($stmt)
  ↪ _CreateCupFromRow($row)
- Cup[] | null ← **FindAllPastCupsMostRecentFirst**()
  ↘ _CreateCupsFromStatement($stmt)
  ↪ _CreateCupFromRow($row)
- Cup | null ← **GetCupById**($id)
  ↘ _CreateCupOrNullFromStatement($stmt)
  ↘ _CreateCupFromRow($row)

- Pair[] | null ← **FindPairingsForThisCup**($id)
  ↘ _CreatePairsFromStatement($stmt)
  ↪ _CreatePairFromRow($row)
- true | false ← **InsertNewCup**($name, $t_st, $t_end, $club, $content)
- true | false ← **IsUserAvailableForTheCup**($userId, $cupId)

Called together in XMLHttpRequest/**update_availability.php** in transaction.

- true | false ← **DeleteOldAvailability**($cupId)
- true | false ← **InsertNewAvailability**($cupId, $userId, 1)

Called together in XMLHttpRequest/**update_pairing.php** in transaction.

- hash | null ← **GetPairingHashForThisCup**($cupId)
  ↘ _GetSingleResultFromStatement($stmt)
- true | false ← **DeleteOldPairing**($cupId)
- true | false ← **InsertNewPairing**($cupId, $posId, $userId)

3.2.5 PositionsManager.php

PositionsManager has API functions to handle Position object/s and delivers it through web application.

- Position[] | null ← **FindAllPositions**()
  ↘ _CreatePositionsFromStatement($stmt)
  ↪ _CreatePositionFromRow($row)
- string | null ← **GetPositionNameById**($id)
  ↘ _GetSingleResultFromStatement($stmt)

3.3 Start file

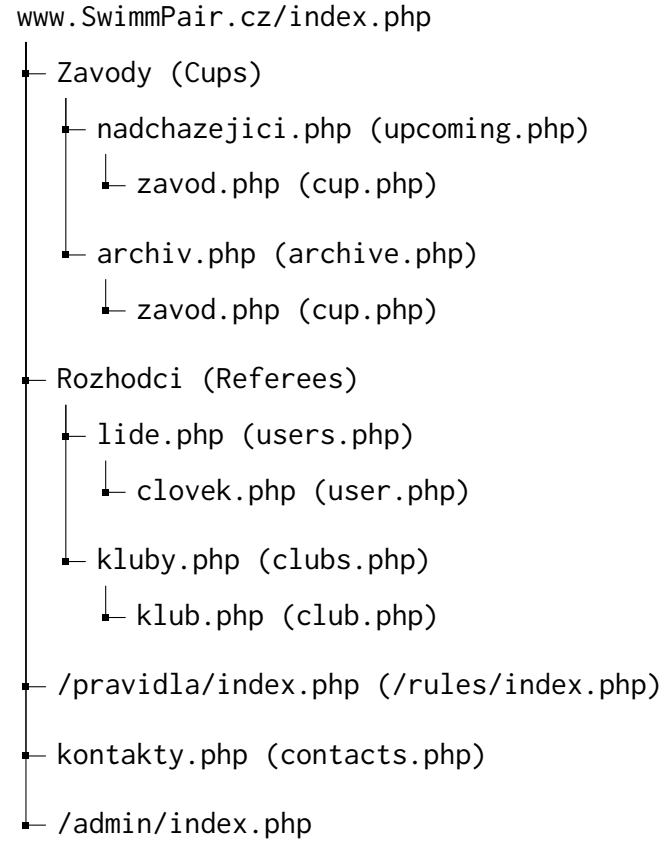
Start file is included in the beginning of each page. It serves for **connection to database**, **sanitization of input**, **definition of error handling** and most importantly **includes objects and managers** and subsequently **instantiates all managers** by passing reference to live database connection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

```
/*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);
```

3.4 Application structure - files defined

User part of the system

The system is running on Czech URLs for convinience reasons of browsing. English equivalents 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.



Admin 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 reseting password and other stuff.

```
www.SwimmPair.cz/administration/profile.php
├─ pridat_aktualitu.php (add_post.php)
├─ editovat_aktuality.php (edit_posts.php)
│  └─ editovat_aktualitu.php (edit_post.php)
├─ nove_registrovani.php (newly_registered.php)
├─ rozhodci_zavody.php (referees_cups.php)
│  └─ pairing.php (pairing.php)
├─ zaregistrovat_uzivatele.php (register_user.php)
├─ editovat_profily.php (edit_profiles.php)
│  └─ editovat_profil.php (edit_profile.php)
├─ novy_klub.php (add_club.php)
├─ sprava_klubu.php (edit_clubs.php)
│  └─ editovat_klub.php (edit_club.php)
├─ novy_kraj.php (new_region.php)
├─ sprava_kraju.php (edit_regions.php)
│  └─ editovat_kraj.php (edit_region.php)
├─ konfigurace_statistik.php (configure_stats.php)
└─ editovat_stranku.php (edit_page.php)
My Club
├─ pridat_zavod.php (add_cup.php)
└─ prihlasit_moje_lidi.php (sign_availability_mates.php)
   └─ prihlasit_moje_lidi_na.php (sign_availability_mates_for.php)
Me
└─ sebe_na_zavod.php (myself_for_cup.php)
   └─ prihlasit_se_na.php (sign_myself_for.php)
```

3.5 Templating of web and administration

Each page on the public website has a common layout, consisting of a header, a menu, and a footer. These sections are unified and included on every page to maintain consistency throughout the website. Specifically, these sections include:

- **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 as well as the sections of lower-ranked users:

- **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 that user has, throw RuntimeException
    if ($_SESSION['rights'] < $role)
    {
      echo '<h1>Not enough rights</h1>';
      echo $_SESSION['rights'];
      echo $role;
      throw new RuntimeException();
    }
  }
}
```

3.6 JavaScript functionality documentation

Frontend functionality is implemented via Vanilla JavaScript. Majority of interactive stuff takes place on **public part** of the application. This section will give reader a glimpse into how things work on the frontend.

Functions in js/SwimmPairFrontendJSLib.js

Functions in this library were created to support Ajax calls and DOM operations for **public part** of our application. Functions have self-descriptive names, function parameter and **this** means reference to the caller DOM element.
Functions are general & contain **XMLHttpRequest** and **DOM modification**:

- **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)

Backend endpoints for XMLHttpRequests

- 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**

3.6.1 Previous post - retrieve older 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**. The result can be:

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

3.6.2 User statistics - year change

All individual referees have seasons years picker when opened. Default season is the current season. Clicking different season visibily 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.

3.6.3 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 function modifying DOM **UpdateClubStatsTable(returnedGetJSON)** which literally updates stats.

3.6.4 Filtering referees

Refiltering of referees is triggered by one of these (Figure 3.1) functions on events:

- **region changed** - onclick="RegionPickerChanged(this)",
- **ref. rank changed** - onclick="RefereeRankPickerChanged(this)",
- **searchbar changed** - onkeyup="SearchBarChanged()".

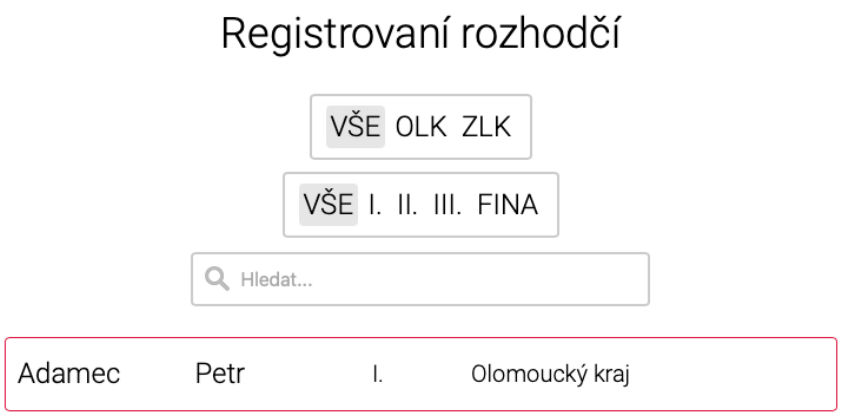


Figure 3.1: Filtering of referees for statistical purposes (requirements R3/S4).

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[]) (region id), **Rank** - IsOptionPermissible(rrid, args[]) (referee rank id) and then if one's **Name** - IsNamePermissible(args[]) starts with queried piece of string from the search bar. We then set one's element style to **style=""** and continue looping. If we fail one of these three conditions we know that referee does not satisfy search conditions and set referee's element's style to **style="display:none"** - to make referee not visible in the listing.

```
...
 //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 its role. On one hand, there is automated code testing, which involves testing the backend by dummy data insertion with performance benchmarking and unit testing. On the other hand, there is testing to ensure that users are able to use the system and to get inspiration for future UX improvements via SUS.

4.1 Performance evaluation

Performance testing script **dummy_data_benchmark.php** [1] is located in the main project folder. Benchmark is performed on default database installation (with 2 admin users, with already existing clubs administered by application requesters and with default referee positions)[2].

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

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 with 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	9.31	8.97	8.59	10.06	8.93	9.03	8.72	9.31	8.98	8.98

Kubernetes - DOKS Kubernetes v 1.25.4-do.0, 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	8.57	8.01	8.25	8.40	8.31	8.17	8.07	8.39	7.95	8.09

[1] In `https://github.com/KlosStepan/SwimmPair-Www` file **dummy_data_benchmark.php**

[2] Database schema for this script and unit testing CI pipeline `https://github.com/KlosStepan/SwimmPair-Www/blob/master/_db/1_create_proc_schema_init_data.sql`

4.2 System Usability Scale testing

We carried on testing of our application by handing SUS questionare to 20 respondents. We then evaluated the scores in order to find out how our application stands. These people are either coordinators, managers or referees [3].
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 calculated[4] 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

[3] John Brooke [1995]

[4] Simple formula for calculating SUS results: ((A1-1)+(5-A2)+(A3-1)+(5-A4)+(A5-1)+(5-A6)+(A7-1)+(5-A8)+(A9-1)+(5-A10))*2,5

We plotted test results so we can further scrutinize them.

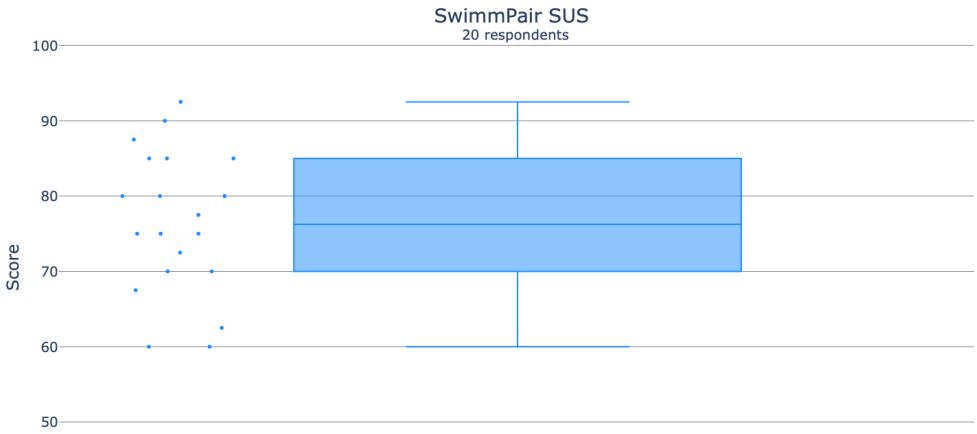


Figure 4.1: Boxplot of 20 System Usability Scale score results.

4.3 Unit testing

Testing serves two puposes:

- **firstly** - to ensure that the implemented functionality works as intended,
- **secondly** - not only that newly added functionality works as intended but also that any previous functions, mechanisms or database were not broken either.

There is bunch of PHPUnit tests for each Manager called ManagerTest.php in folder **tests/Unit** in our project. Tests check ordinary CRUD [5] functionalities.

4.3.1 Local execution of tests

Preview the results of test during local development by attaching VS Code to the running Container [6] and seeing results in PHP Tools by DEVSENSE [7] on the left side(Figure 4.2). Alternatively, we can also open command line (within the container environment already) **docker@6bd3d752da84:/var/www/html** and run tests simply by using **phpunit** command.

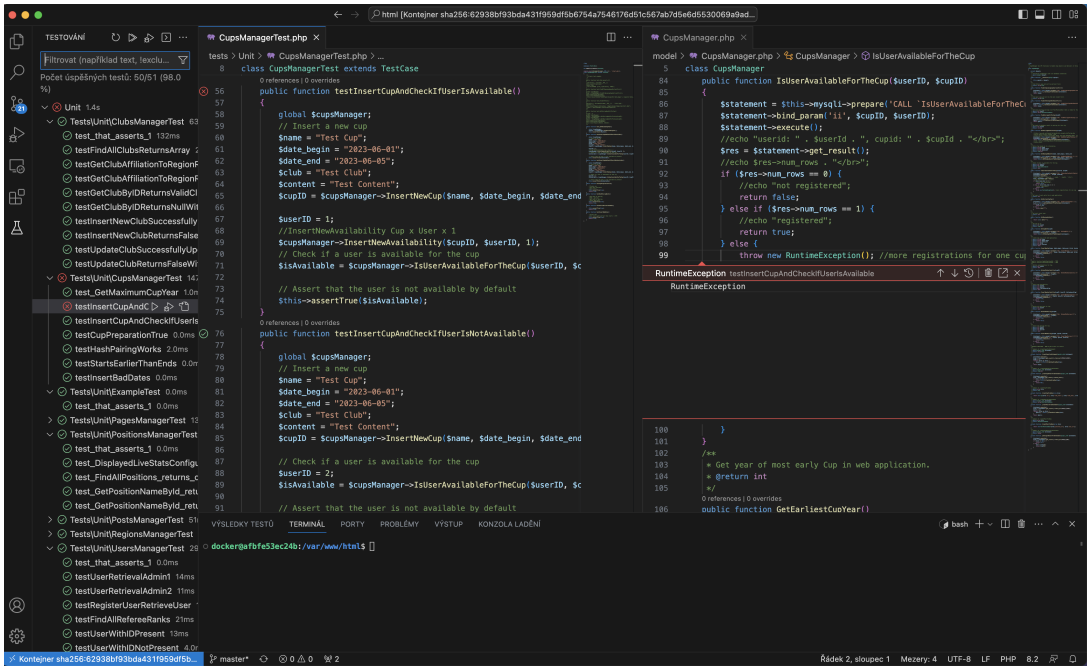


Figure 4.2: Attached VS Code to container running PHPUnit tests.

As we can see there was problem with execution on one of the tests. After brief investigation, we can conclude that the reason why our **testInsertCupAndCheckIfUserIsAvailable()** failed was because we did run the tests on already populated database. Thus, we added more than one availability of user for cup. This error wouldn't have occurred if we had run the tests on a fresh database schema or used drag'n'drop in our application to perform desired operation (Figure 2.5).

[5] CRUD = Create, Read, Update, Delete

[6] Volume within **. : /var/www/html** is, in fact, our working folder.

[7] `https://www.devsense.com/en/features#vscode`

4.3.2 GitHub Actions workflow

We load our project repository with a folder containing tests and then run automated testing each time we push new code to the repository. Once the testing is complete, we can review the results of all the steps taken. If everything passes, a green checkbox is added to our repository header next to the hash as confirmation.

Steps that are crucial for our CI pipeline's QA[8] purpose:

1. code gets pushed into master brach of the repositor[9],
2. GitHub Action workload[10] gets triggered,
3. database created, filled with dummy data, app connects to it,
4. then **phpunit** command is run,
5. **results are reported** for preview for us (Figure 4.3).

Result of all these steps are visible on GitHub website. All steps are openable for more deliberate investigation.

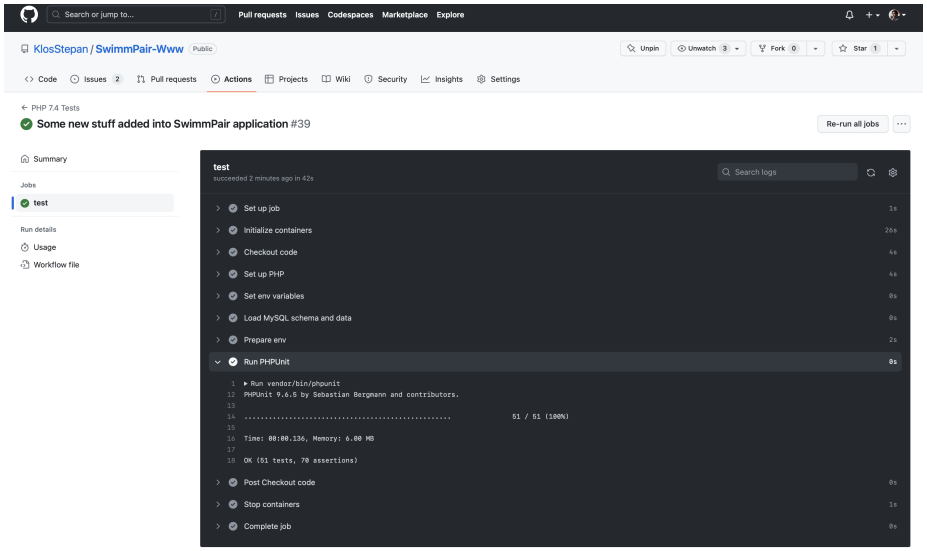


Figure 4.3: GitHub Actions workflows - steps of execution, PHPUnit step opened.

Once the pipeline runs successfully, a checkbox appears next to the commit hash, providing a quick overview.

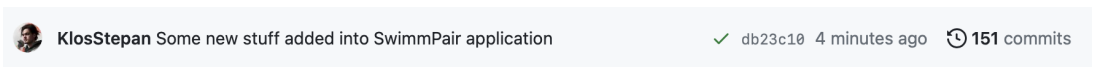


Figure 4.4: Green checkbox next to the commit hash as confirmation.

[8] QA = quallity assurance

[9] `https://github.com/KlosStepan/SwimmPair-Www`

[10] `https://github.com/KlosStepan/SwimmPair-Www/blob/master/.github/workflows/main.yml`

5. Deployment

Development of our application was performed locally using **docker-compose** to glue up 4 components necessary to sufficiently orchestrate PHP container for web application, MySQL container + Adminer container and Redis container.

We have to, however, run our application in **Kubernetes Cluster**. **Services** and **Deployments** have to be written. Services for the purpose of routing and taking care of new container spawn addresses. Deployments to describe container definitions - images & volumes, number of replicas and environment variables. **Database** is run as a separate standalone entity within the cluster with persistent storage, **Redis** for syncing and preserving sessions as well.

SwimmPair running in the Kubernetes Cluster

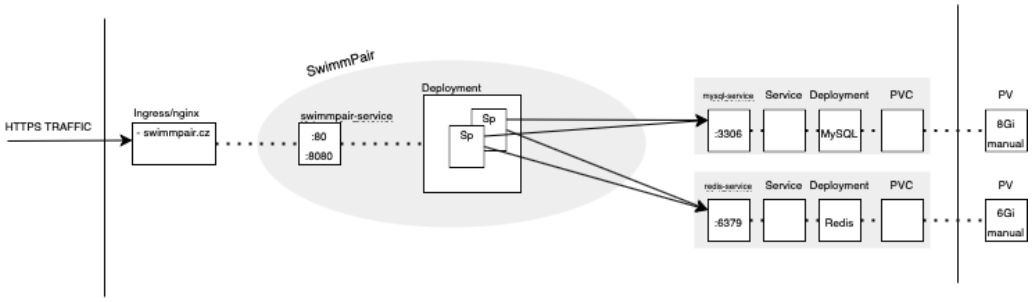


Figure 5.1: Preview of our Kubernetes setup with running application.

Both **Database** and **Redis** can be substituted by Managed Databases[1] or their equivalents in different cloud providers. They can be accessed remotely as dedicated self-hosted services within your own company as well.

SwimmPair can be run using Container Service or "1-click app" as well.

Dockerization of SwimmPair application

A file called **Dockerfile** has to be ready in the project folder.

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

This image of PHP7.4/Apache[2] was chosen because it correctly dockerizes part of so-called LAMP stack. In order to build this image and push it into Docker Hub we run following commands:

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

This image is then pullable as stepanklos/swimmpair:latest by Deployment[3].

[1] https://www.digitalocean.com/products/managed-databases

[2] Image **thecodingmachine/php:7.4-v4-apache** by TheCodingMachine - https://github.com/thecodingmachine/docker-images-php

[3] https://hub.docker.com/repository/docker/stepanklos/swimmpair/general

Kubernetes

We run two replicas on two nodes to ensure reliability and uptime. Additionally, we have set up an autoscaler to accommodate future increases in traffic.

Kubernetes autoscaler for increased workloads

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: swimmpair-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: swimmpair
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 70
```

Figure 5.2: Autoscaler for Deployment to accomodate larger application traffic.

Deployment app-swimmpair.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: swimmpair
spec:
 replicas: 2
 selector:
  matchLabels:
   app: swimmpair
 template:
  metadata:
   labels:
    app: swimmpair
  spec:
   containers:
   - name: swimmpair
    image: stepanklos/swimmpair:latest
    securityContext:
     allowPrivilegeEscalation: true
    ports:
    - containerPort: 80
    env:
    - name: MESSAGE
     value: Hello from swimmpair Deployment!
    - name: DATABASE_HOST
     value: 'mysql-service'
    - name: DATABASE_USER
     value: 'root'
    - name: DATABASE_PASS
     value: 'heslo'
    - name: DATABASE_NAME
     value: 'plavani'
```

```
- name: PHP_INI_SESSION__SAVE_HANDLER
  value: 'redis'
- name: PHP_INI_SESSION__SAVE_PATH
  value: 'tcp://redis-service:6379?auth=aGVzbG8='
```

Database and Redis

As mentioned, our application does not come with a database (and Adminer) or Redis, so we had to set them up separately using a PV (persistent storage) on which a PVC (persistent volume claim) was created. Due to Digital Ocean's policy on PersistentVolume implementation, we had to run these services as 1 pod with manual volume, which is sufficient for our workload.
These two services we set up are internally accessible as follows:

- MySQL Database[4]: **mysql-service:3306** ,
- **Re**mote **Di**ctionary **S**erver: [5] **redis-service:6379** ,

in our Kubernetes Cluster.

[4] https://github.com/KlosStepan/DOKS-configs/tree/main/mysql-deployment
[5] https://github.com/KlosStepan/DOKS-configs/tree/main/redis-deployment

Conclusion

The process of designing, developing, and shipping this web application was overall successful, although there are still some parts that could be improved or extended in the future. However, our system is ready for these changes and we have learned valuable lessons along the way.

In particular, the design and development process was not as straightforward as we had initially anticipated. Instead, it was an iterative process that required close collaboration with the system requester. The stages of iteration were roughly as follows.

1. Problem description and basic page layout programming (homepage, cup, user, club).
2. Formalization of the model and proper division of code into system objects and task functions.
3. Analysis of user and club data, additional pages for categorization purposes.
4. Addition of regions for further extensible hierarchization.
5. Major final refactoring of the database, backend, and testing with dummy data insertion and querying.
6. Cloud readiness, Docker image of the web application, and Kubernetes deployment.

In the future, the system can be extended by adding new public pages, statistical queries, and administrative tasks. These modifications might involve adding new pages, adjusting the model, or making minor design changes. Thanks to the divided code, these changes can be easily accommodated and will reside in the project GitHub repository [6].

Overall, we are pleased with the outcome of this project and look forward to seeing it evolve further in the future.

[6] https://github.com/KlosStepan/SwimmPair-Www

Bibliography

Docker, Inc. 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 Corporation. 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 Abbreviations

CSPS Cesky Svaz Plaveckych Sportu, *Czech Swimming Federation* unites swimming clubs in the Czech Republic and provides competitions infrastructure and operations.

LAMP Linux Apache MySQL PHP, standard stack for running applications.

SUS System Usability Scale, is a standardized questionnaire consisting of 10 questions, which assesses the usability of a web application on a scale of 0-100. It provides a measure of how easy the application is to use, helping to identify areas for improvement.

DOKS DigitalOcean Kubernetes, managed Kubernetes service by `https://www.digitalocean.com` with various products, scaling and functional options.

SwimmPair Swimming Pairing, application that we developed and abbreviated and branded it like *SwimmPair*.

UML Unified Modeling Language, style of represent class relations between modelled objects in functional design.

CRUD Create Read Update Delete, set of basic functions for performing I/O of data via Managers in web applications.

CI Continuous Integration, is a software development practice that involves frequently integrating code changes from multiple developers into a shared repository. It aims to detect and resolve integration issues early by automatically building, testing, and validating code changes. This helps to ensure that the software is always in a working state and ready for deployment.

K8s is a shorthand way of writing "Kubernetes". The "8" in "K8s" represents the eight letters between the "K" and the "s" in "Kubernetes".

QA auality assurance, refers to the process of ensuring that a product or service meets a certain level of quality and reliability.