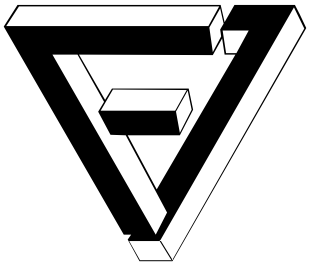
Masaryk University
Faculty of Informatics



Information System for an Orienteering Club

Bachelor's Thesis

David Štorek

Brno, 2025

Declaration

Hereby I declare that this thesis is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source. While preparing this thesis, I have utilised the ChatGPT AI tool as aid in searching for information. I have not used any AI tools to write code or text as part of this thesis.

Advisor: RNDr. Jaroslav Pelikán, Ph.D.

Abstract

In this thesis, I describe the design, implementation and deployment of an information system for a Czech orienteering club. The system handles entries to club events, such as trainings and races, and manages the entry fees for official Czech orienteering events. It also serves as a public website for the club, with select users being able to post news articles to the front page.

The application is implemented in the ASP.NET Core framework for C# and deployed on a Linux machine.

Keywords

ASP.NET, C#, information systems, orienteering, web development

Contents

Chapter 1

Introduction

The goal of this thesis is to develop a web information system for an orienteering [2] club. Many such systems already exist, but they are usually tailored to the needs of a single specific club and not available to others. Admittedly, my implementation will not have as many features as some of these existing systems. Instead, I aim to create a solid foundation that will be deployed early at my local club in Prostějov, being generic enough to be usable in most clubs, yet also allowing extension or modification to suit specific clubs.

The basis for this thesis was my local club's dire need to have a consistent entry system for events organised by the club: mainly trainings and training camps. This constitutes the central section of the system, which is the event calendar. The second requirement has to do with yearly registration fees in the club and entry fees to official Czech orienteering events. Currently, all of the administration required is done manually by the club's accountant. In addition to these features, the system would also serve as a public web page for the club, replacing an old, outdated one. This constitutes the third major section of the system, which is the news page.

Chapter 2

Analysis

In this chapter, I go over the existing implementations of orienteering club information systems and briefly explain the situation of Czech orienteering in regards to the implementation. Then I continue with the overview of requirements for the system and present some use cases.

2.1 Czech orienteering

Every orienteer, who is a member of a club in the Czech Republic, is registered with the Czech Orienteering Federation. [8] This comes with access to the ORIS [37] information system, which is used to enter official competitions. Thanks to the fact that every club member already has login credentials to ORIS, we will offload our user authentication to it via it's API. [36]

Competitions in ORIS typically come with an entry fee which is, in the case of my club, covered by the yearly club registration fee. An exception to this is when a user entered late and thus has been charged a surcharge for the entry. The application will have a way to monitor these late entries and inform club officials when a member should pay this surcharge.

2.2 Existing implementations

As I mentioned before, there are many information systems with characteristics similar to this one. In addition to those developed in-house and deployed only in one specific club, the major ones are eob.cz [30] and "Privátní zóna" (Private Zone). [24]

Eob is a basic system for managing event entries. It was created some time around the year 2002. It is written in PHP, and is outdated by modern web development standards. There is limited information available about it online.

Private Zone is a modern, feature-rich and relatively recent club information system that is used mainly by clubs in north-west Bohemia. Also built in PHP, it is targeted primarily at mid-size to larger clubs. While it's feature set could potentially satisfy the needs in my local club, the club events agenda[1] does not support public events and is otherwise not very extensive. In addition, Private Zone does not come with a public news page, where users could post articles.

2.3 Requirements

This section details the functional and non-functional requirements set for the implementation.

2.3.1 Functional requirements

- Club members will be able to log into the system with their ORIS credentials.
- Select users will be able to create and manage events, and users will be able to enter these events.
- Users that are not members of the club will be able to enter select events without the need to log in.
- Select users will be able to post news articles to the front page.
- Select users will be able to manage the club registration fees and get an overview of entry fees to competitions in ORIS.

2.3.2 Non-functional requirements

- The interaction with the ORIS API when logging in must be properly secured.
- Login attempts must be properly cached in order to not overload the ORIS API.[2]
- The application will be written in C#, using the ASP.NET Core framework. The front end will utilise ASP.NET Core MVC.
- Updating the application must be seamless, with no noticeable downtime.

[1] The non-official events that exist only in the Private Zone and not in ORIS.

[2] This requirement is set by the maintainers of ORIS. The login API endpoint is undocumented to prevent abuse.

- The application will be available at least 99% of the time. If the need should arise to bring the application down for an extended period of time, it will be carried out outside the most frequent usage hours if possible.

Even though vertical scaling[3] is important in an application that expects to be running for a long time, this was not a priority, and so was not included as a requirement. Due to the relatively small amount of users and traffic, the growth will be manageable without any special attention.

2.4 Use cases

The following is a use case diagram of the application. There are several actors who interact with the system in various ways:

- The `Visitor` is a user who is not logged in. They are typically not a club member and only visit the site to read the news page or enter one of the public events.
- The `User` is a club member who has an account in the system and is currently logged in. They can do everything the Visitor does as well as view and enter non-public events.
- The managers are special types of users with elevated privileges. They each manage a component of the application.
- The `ČSOS`[4] `ORIS API` is a system actor that is contacted by the application to fetch requested data from the ORIS API.

[3] Optimising the performance of the single instance of the application when it's utilisation increases.

[4] ČSOS is the Czech abbreviation for the Czech Orienteering Federation.

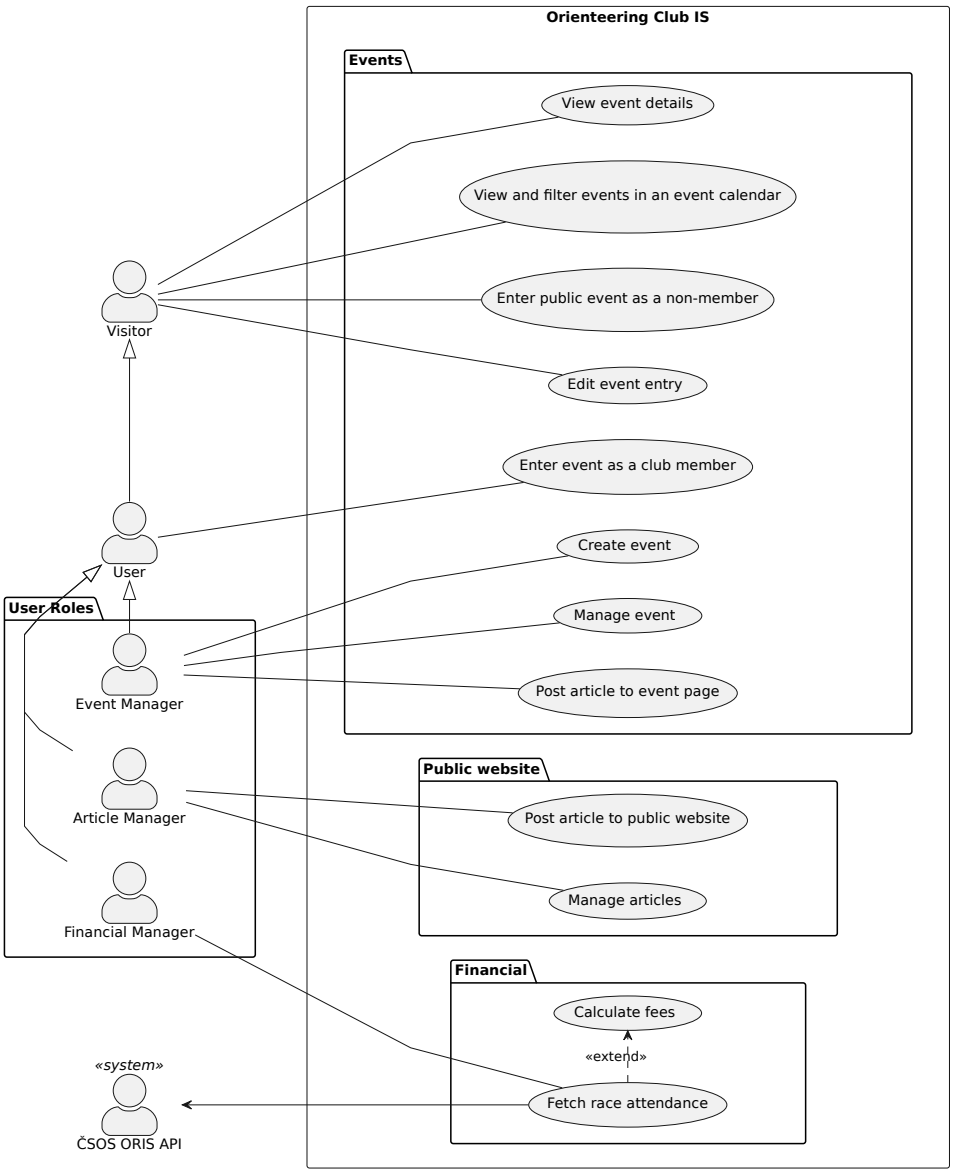


Figure 2.1: Use case diagram

Chapter 3

Design

As mentioned before, the club members will log into the system using their ORIS credentials. Upon first login, the user account will be created. Some users will have to have elevated privileges to be able to create events or post news articles. These permissions will be evaluated based on roles that can be assigned to each user.

The system is structured into three layers. The Data Access Layer (DAL) defines the database schema and communicates with the database. The Business Layer (BL) handles user permissions, communication with the ORIS API and other business logic. Lastly, the Presentation Layer serves as the front end for the application, defines the URL endpoints and presents content to the user.

3.1 Data Access Layer

The DAL is based on the Entity Framework Core ORM (hereafter referred to as EF Core, or EF). The framework has two distinct approaches to defining the database: Database First and Code First. The former takes an already existing database schema, while the latter defines it in code using C# classes. These classes are called the DB models, or DAL models. Although using Database First is "the most common scenario in real life" [23, p. 515], I chose to go with Code First as it is database-agnostic and makes it easier to track schema changes with version control.

3.1.1 Database schema

The data model is composed of three, largely independent, agendas: the news articles, the events and the finances. In addition, the users and user roles are also defined. In this section, I go into detail about these agendas, one by one. For clarity, the Entity Relationship Diagram is split into parts, each accompanying the explanation of the respective agenda. The full ERD

is available in Appendix B.

Users and user roles

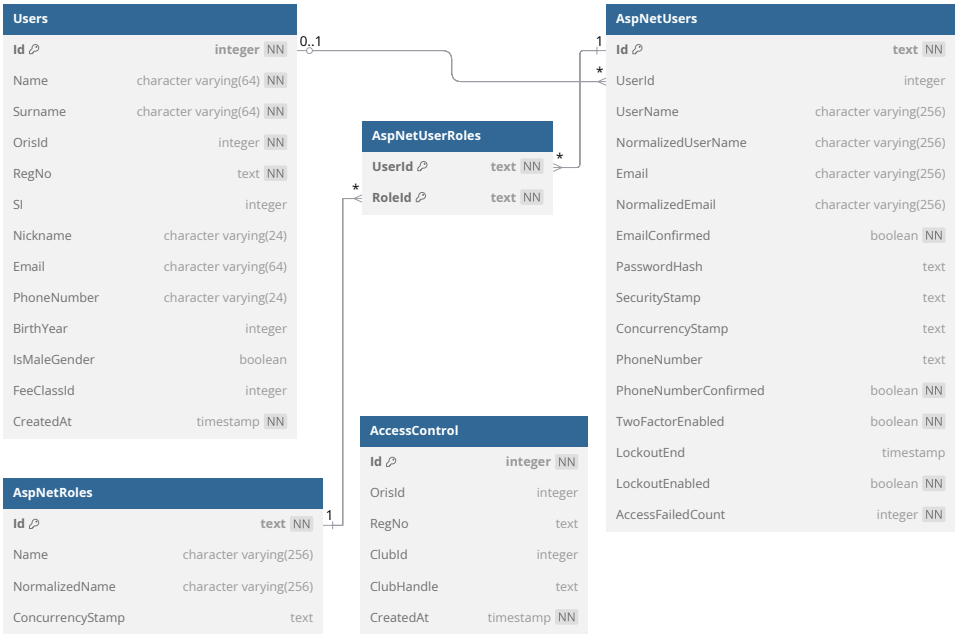


Figure 3.1: Users and roles ERD

The `Users` table stores the user information. Note that there are no columns associated with passwords, as the users are instead authenticated via ORIS using their `OrisId`, which is unique among all registered orienteers. The `AspNetUsers` table is needed by "Identity", a component of the ASP.NET Core framework. More on that in chapter 4.1.2. Most of the columns in this table are unused in the application. The framework unfortunately requires them to be present, so they cannot be removed. The `AspNetRoles` table is also part of Identity and the user roles are stored here. At last, the `AccessControl` table stores fields that are evaluated against newly registered users to determine if they should be allowed to access the application.

Most of the columns in these tables are self explanatory. Other, less clear columns serve these purposes:

- `ClubHandle` is a three-letter abbreviation for a club. Unique among clubs in ORIS.
- `RegNo` is the registration number in the Czech Orienteering Federation, also unique. It consists of the club handle and 4 numbers, from which we can determine the gender and year of birth.

- `SI` represents the identification number of SportIdent [26] cards, which are widely used to mark ("punch") control points in orienteering.

Blogs and news articles

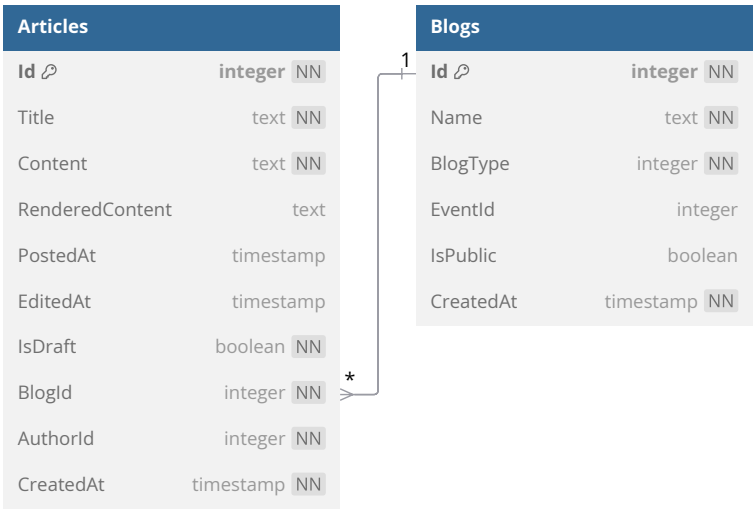


Figure 3.2: Blogs and news articles ERD

A blog is a collection of articles that can either be stand-alone (for articles on the front page) or connected to an event. This is determined by the `BlogType` column in the `Blogs` table. Although the `EventId` column is nullable, it must not be null when the type of the blog is `EventBlog`. EF Core helps to enforce this. The Blog model is defined in Code First, and looks like this:

```
public class Blog
{
    public string Name { get; set; }

    public BlogType BlogType { get; set; } // this is an enum
}

public class EventBlog : Blog
{
    public int EventId { get; set; }
}
```

This hierarchy is stored in the `Blogs` table and the inherited entity is distinguished from the base one (or other inherited entities) by a discrimin-

ator, which, in this case, is the `BlogType` column. This approach is called Single Table Inheritance[1]. [12, p. 55]

Events

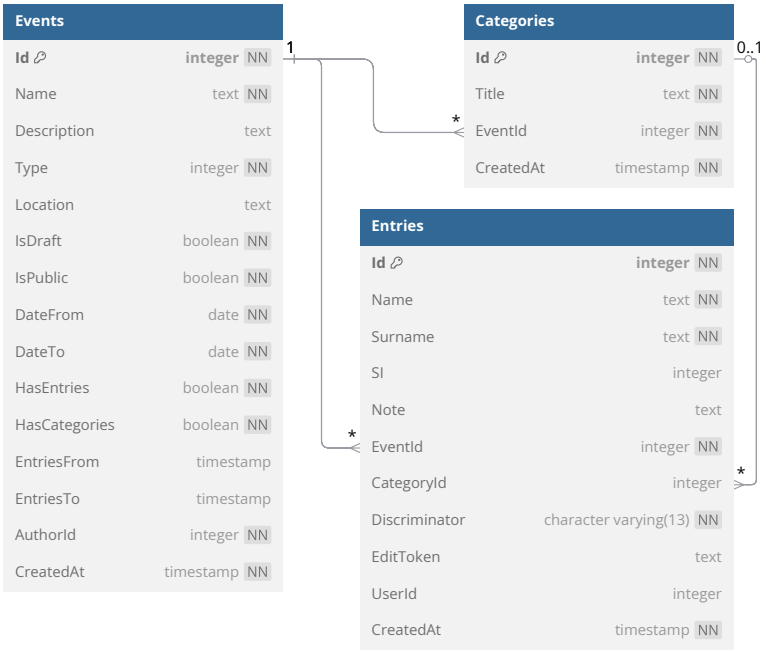


Figure 3.3: Events ERD

An event can be either public or not – this determines whether users need to be logged in to see and enter the event. It can have entries and categories (both of which are optional). It can be marked as draft, in which case it is only visible to it's author.

A category represents the different classes that one can enter in a race, training or other type of event.

There are two types of entries, distinguished by the same method as blogs in the previous section. These are the `UserEntry` – associated with an existing user, and the `UnifiedEntry`, which is instead managed using a randomly generated `EditToken`, and is meant for public events that allow unregistered users to enter.

[1] In Entity Framework, this is referred to as Table-per-hierarchy [9]

Finances

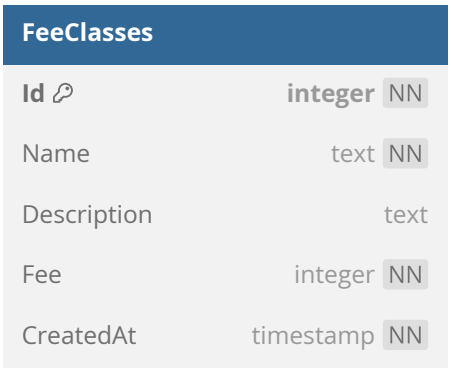


Figure 3.4: Finances ERD

The financial section is rather simple in terms of the entity model. A `FeeClass` is optional for users and it only contains the amount to be paid for the yearly fee. Other fees are calculated based on ORIS and not stored in the database.

3.1.2 Repositories

The Repository Pattern is a "layer of abstraction where query construction code is concentrated". [12, p. 294] The ORM vastly simplifies query construction, so this layer is quite small in terms of lines of code when compared to the other layers. As mentioned in [19, p. 242], the Repository Pattern, while having it's benefits, is best when not overused. In cases where it would make sense not to force all query building into a repository, I chose to not enforce this design pattern.

In the application, each repository corresponds to a table in the database, for example event related queries are handled by the `EventRepository`.

3.1.3 Unit of Work

A Unit of Work manages database transactions, keeping track of related queries and coordinating writes to the database . [12, p. 174]

This is mostly handled by EF Core automatically. There is a `UnitOfWork` class present in the implementation, but it's main purpose is to organise the repositories into a single class in order to simplify dependency injection in the Business Layer.

3.2 Business Layer

The Business Layer is composed of services. Services handle the data fetched by repositories in ways that cannot be done on the database level. They handle the main logic of the application and determine if users have the necessary permissions to access resources. There is also a service that handles communication with the ORIS API.

Instead of raw database models – the entities that directly correspond to the tables in the database schema, the Business Layer sometimes uses Data Transfer Objects (DTOs) to hide properties that are not necessary or intended to be available outside the repository logic.

3.2.1 User permissions

The user permissions are organised with roles and policies. A user has zero or more roles, and multiple roles form a policy. Users and policies are not connected directly. A protected resource[2] can be locked behind a single role or a policy. In the former case, only users with that role are authorised. In the latter, all users whose roles are allowed by the policy are authorised. This allows more fine-grained control over user permissions. For example, an administrator role would allow the user to manage events, news articles and finances thanks to the role being included in all the respective policies. Thus, administrators can have automatic access to new policies, without needing to be assigned a role one by one.

3.3 Presentation Layer

The presentation layer, or the front end, is built on top of the Model-View-Controller pattern (MVC). [13, p. 53]

- The **Models** represent the data that is presented to users or submitted by them.
- The **Views** render and display the data from the models to the user.
- The **Controllers** handle routing, requests and responses to them. They operate with the model data and select views to render. This is typically also where the bridge between the presentation and business layers lie.

[2] A resource that requires the user to be authorised.

Chapter 4

Implementation

This chapter focuses on the implementation details. It presents an overview of used technologies, sheds light on the structure of the code and goes into detail about how the design principles and patters described in the previous chapter were implemented.

4.1 Overview of technologies

To develop the web application, I chose the ASP.NET Core framework, because it is stable, robust and widely used. Documentation is also extensive, although navigating it proved to be challenging due to many differences between versions of the framework.

4.1.1 C# and .NET

C# is a high-level object-oriented multi-purpose programming language invented and developed by Microsoft. .NET[1] is a software framework supporting multiple languages (including C#), which are integrated into a "Common Language Infrastructure" [29]. The application is using C# version 12 and .NET version 8. Newer versions are available now, but when the development first started, .NET 8 was the latest version of the framework and currently has long term support.[2]

4.1.2 ASP.NET Core

ASP.NET Core[3] is a web application framework built on top of the .NET ecosystem. It implements the Model-View-Controller pattern in ASP.NET

[1] Not to be confused with the .NET Framework, which is a Windows-only framework that is no longer being developed, or .NET Core, which is only a component of .NET.

[2] Long term support versions of .NET are typically supported for 3 years, while standard term last 2 years. So .NET 8 will actually be supported for longer than .NET 9.

[3] Again, not to be confused with ASP.NET. The "Core" attribute distinguishes new, open-source and cross-platform versions of Microsoft's products from earlier versions,

Core MVC using the Razor templating engine [15] to render views as html pages.

4.1.3 Entity Framework Core

EF Core is an ORM (Object-Relational Mapper), that serves as a layer between the database and the application. It supports multiple relational database providers, and if implemented properly, allows to switch to a different one without much trouble. It maps C# classes to tables in the database and constructs SQL queries from LINQ, a fluent[4] query language integrated in the .NET platform.

4.1.4 PostgreSQL

PostgreSQL describes itself as an "open source object-relational database system with over 35 years of active development". [21] I chose this database system specifically because it is easy to use on Linux (where the application is deployed), is stable and has a large community. Originally, part of my motivation was also to utilise the large objects feature of PostgreSQL for storing binary files efficiently in the database, however this idea was abandoned.

4.1.5 Bulma CSS

Bulma [4] is a CSS framework that comes with no JavaScript. It has a built-in automatic dark mode[5] and comes with powerful colour scheme customisation. I've specifically chosen to use a framework with no JavaScript to not make the front end excessively heavy, and keep the implementation simple. Having said that, JavaScript is necessary in modern websites to make some elements accessible to mobile users. Some JavaScript snippets are used to perform various small tasks, mostly concerned with UI, on the client.

4.1.6 Other libraries

Mapster is a library that maps objects to different objects. I use it to transform DAL models 3.1, DTOs 3.2 and view models 3.3 to and from each other.

---

which were Windows-only and typically closed-source.

[4]When something is referred to as fluent in the .NET ecosystem, it means that an object (in this case a query) is built by chaining various method calls that each transform the object in some way. The concept is inspired by functional programming.

[5]Automatic dark mode was a feature that caught my attention when picking this framework. As it turns out however, the dark mode does not look very pretty, which is why it is disabled in the application.

The *Markdig* [17], *HtmlSanitizer* [16] and *AngleSharp* [3] libraries are used to convert Markdown to HTML, sanitise user input and transform the HTML further, respectively. *NodaTime* [20] is a library that improves upon the C# default date and time types and integrates well with EF Core and PostgreSQL. *Htmx* [31] is a JavaScript library that makes it easier to render partial HTML – like pop-ups – without having to re-render the entire page. It also comes with a C# library that integrates it with ASP.NET Core. *Flurl* [11] is a fluent URL builder that I use to communicate with the ORIS API.

4.2 Methodology

How the development of a project is managed is an important consideration when developing large applications such as this information system. I wanted to make sure that the application is properly versioned, the bugs and issues are kept track of, and features are implemented fully before being released to the public. To keep track of development, I primarily used the Git version control system [14] and GitHub [1], which hosts it and provides some additional features.

4.2.1 Git and GitHub

The git repository containing the implementation has two main branches – `master` and `devel`. The development branch is where the development happens. The state of the application on devel might not be ready for release yet. When a new version is ready to be released, the individual commits on the devel branch are squashed and merged into the master branch. A new git "tag" [5] that describes the version is created. Squashing the commits helps with debugging any potential future issues, as it makes it easy to determine in what version the bug first occurred by analysing the repository with `git bisect` [6]. Since the original commits are still available on the devel branch, we can use bisect on it as well to make the search more granular. However, the commits on devel might be work-in-progress, and the application is only guaranteed to be buildable on the master branch.

As for GitHub, the feature I primarily used to track bugs, improvements or possible future features is GitHub Issues. An issue can be tagged with various tags to distinguish different kinds of problems and it can be resolved by commits so that it is clear in the future what the incident was and how it was solved. GitHub comes with more advanced project management features, but I found them to be unnecessary for a project of this scale with a single maintainer.

4.2.2 Semantic Versioning

Semantic Versioning [27] is a set of guidelines for versioning the application and making it clear whether different versions are compatible. A "major version" is a version that contains breaking changes – it is incompatible with previous versions. I use this version to mark changes to the database schema that cannot be rolled back without risking losing data. A "minor version" is compatible with the previous version – it typically constitutes changes to application logic that are backwards compatible or changes to the database schema that only add columns, i.e. can be easily reverted without losing data from the previous version. A "patch" is even smaller that a minor version and it typically marks a hotfix or a small adjustment that does not necessarily constitute a full release. The version number is represented as a "version string" in the following format: `v<major>.<minor>.<patch>`.

4.3 Implementation structure

The code is structured in line with the three application layers – corresponding to the `DAL`, `BL` and `MVC` projects[6].

4.3.1 DAL

The DAL is structured into the following directories, each one containing a piece of the DAL logic:

- `Models` holds all the DB models – classes that correspond to tables in the database.
- `Repositories` contains the repositories implementing the Repository Pattern.
- `UnitOfWork` holds the `UnitOfWork` class, which wraps the repositories and the `DbContext`[7].
- `Query` defines helper methods that extend LINQ querying.

The DAL also contains the `Seeder` class, which contains static data that must always be created with a new database, such as the `Blog` that holds the front page news articles. Separate from the main DAL project is the `DAL.PostgreSQL.Migrations` project, which holds the database migrations. Thanks to this separation, it is possible to introduce a new database provider with it's own migrations (migrations are always specific to the DB provider) while keeping the old ones intact.

[6] A project here refers to a C# project – a compilation unit in the C# ecosystem. All projects fall under the .NET solution, which encompasses the entire thesis implementation.

[7] The `DbContext` is the single point of communication with the database in EF Core. Repositories use it internally, but it can also be used on its own to interact with the ORM.

4.3.2 BL

The BL is composed of these directories:

- `Services` contain the main business logic.
- `DTOs` are used to transfer data between the BL and other application layers.
- `Auth` defines ASP.NET Core roles and policies as enums, preventing possible issues with referring to them with hard-coded string names, which is what the framework does by default.
- `Tokens` handles generating edit tokens for event entries.
- `Result` is a helper class that simplifies returning errors from service methods. Inspired by `std::result` from Rust. [25]

4.3.3 MVC

The `Models`, `Controllers` and `Views` directories correspond to the three parts of the MVC architecture. Views are split into subdirectories by their respective controller, as are the models. Two types of views exist: regular ones and partial views. Regular views use a layout and represent a full web page, while partials only represent a part of the DOM[8]. Partial views are typically named with an underscore prefix. `ViewComponents` are more powerful versions of partial views. [28]. View components can be "invoked" from within views, which triggers some application logic.

The static files for the website are located inside the `wwwroot` directory. This includes images, CSS and JavaScript files. This directory also contains the JavaScript libraries used on the front end. JavaScript libraries are installed and managed with `npm` [34] and deployed to the `wwwroot` directory using a shell script.

The `Program.cs` file is where the application is configured and ran. It includes the build parameters defined in `appsettings.json`[9], resolves dependency injection, creates database connections and runs the application.

4.3.4 Dependency Injection

Dependency Injection is how it all comes together. DI separates the object's dependencies from it's behaviour, instead providing them externally. [22, p. 493] In practice, this means we do not have to worry about integrating the different application layers – we just define an interface for a service[10],

[8] Document Object Model

[9] These parameters are serialised into a C# class for use in the code.

[10] This does not necessarily have to be a service as in the business layer. A repository is also a service in this context, as is the ORIS API client.

and the service gets injected. The injected service can be accessed from anywhere by inserting it as a constructor parameter[11]. It is slightly more complicated than this and beyond the scope of this thesis. Dependency Injection is nicely explained in [22].

4.4 ORIS API client

The logic communicating with the ORIS API is implemented by the ORIS API client. It is part of business logic, but I decided to put it in a separate project and treat it as a library. The reason for this separation is that I plan on releasing it under a different licence as a library after all the API methods are implemented. At this time, only the methods needed by the application are implemented.

The API client is based on the *Flurl* [11] library and de-serialises the responses to C# classes that can then be used by the application.

Implementing the API client proved to be a challenge. The API is written in PHP and is inconsistent to say the least. The methods and their parameters are documented, but the API lacks documentation for the format of responses, which made implementing the client possible only with trial-and-error. Sometimes, the API would return entirely different types (an empty array instead of a regular object, among many others) and change the schema of the response based on the values of parameters. This made de-serialising the JSON responses into C# objects quite complicated.

In addition, the API mostly only returns the 200 HTTP status code, even if a request is malformed[12] or the resource is not found, resulting in not being able to provide a clear error message to the user.

4.5 User Interface

The user interface is designed to be responsive – usable on both desktop and mobile devices. The main layout, visible on all pages, is composed of a horizontal navigation bar with the main navigation and a vertical one with additional links.

The vertical navbar takes up one fifth of the width of the page. The rest of the space next to it is taken up by the main page content, which can additionally be split into two parts taking up three fourths and one fourth of the content respectively. Thus the page can be effectively made to have a central piece with two columns of the same width on each side.

Since the application targets a mostly Czech audience, the UI is localised in Czech and does not support localising to different languages. The

[11] As well as by various other means. Constructor Injection is just the most common.
[12] Sometimes, malformed parameters result in seemingly random behaviour.

UI showcases present in the following sections were translated using the browser's built-in translator.

4.6 Features

This section describes the feature set of the application from the user perspective, detailing how the user will typically use it.

4.6.1 Logging in

The user logs in with their ORIS credentials. The ORIS API endpoint is contacted to authenticate the user. This is also where the login attempts are cached to prevent overloading the API. If the user has not logged in prior to this point, they are taken to a registration form, pre-filled with their name fetched from ORIS. They can enter optional user details. Upon continuing, a user account is created with the respective name.

Currently, the user roles have to be assigned manually by inserting the respective records into the database as this is unfortunately not yet implemented in the UI.

4.6.2 Posting articles

A user in the NewsManager policy can post an article to the front page. This brings up a Markdown editor for the article content. The user can decide to leave the article unpublished and save it in draft mode, at which point the article content is saved so that the user can continue editing it in the future. Publishing the article renders it and makes it available on the front page.

The Markdown editor supports live-previewing the rendered article. In order to save on unnecessary requests to the server just to render Markdown, the live preview is rendered on the client and only the final article renders on the server. Consistency between the different Markdown renderers (markdown-it [32] on the client and Markdig on the server) is ensured by configuring them to strictly follow the CommonMark [7] specification.

When rendering the article on the server, special care is taken to sanitise the resulting HTML with the *HtmlSanitizer* library. The articles support including external images, which are processed with the *AngleSharp* library to display as a clickable preview that expands the full image over the entire page. This ensures that inline images inserted in the markdown have a consistent size in the render.

Figure 4.1: Front page with an article, from the point of view of an News-Manager.

4.6.3 Creating and entering events

The event calendars displays the planned, ongoing and past events. It can be filtered. The user can also view all events created by them or events they are entered into. From this page, events are also created. A user in the EventManager policy can choose to create an event with or without categories, fill in the form and publish the event. Events can be marked as drafts in a similar way to articles – preventing them from being published until ready.

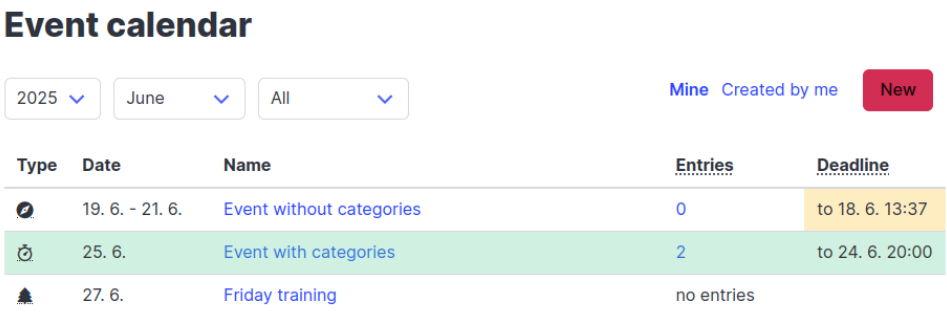


Figure 4.2: The event calendar from the point of view of an EventManager. Entered events are highlighted in green and entry deadlines in orange, if the deadline is coming soon.

Clicking the event in the calendar brings users to the event detail page.

This page lists the main info about the event, as well as the list of entries.

If entries are enabled for the event, users and visitors[13] alike can enter the event by filling out the entry form. A confirmation box is shown and the edit token is displayed in the case of unified entries. The user's own entry is displayed on top of the entries in the entries tab and can be easily edited or deleted. The unified entry is edited by clicking the appropriate button and inputting the edit token. Unified entries can only be edited while the user is not logged in.

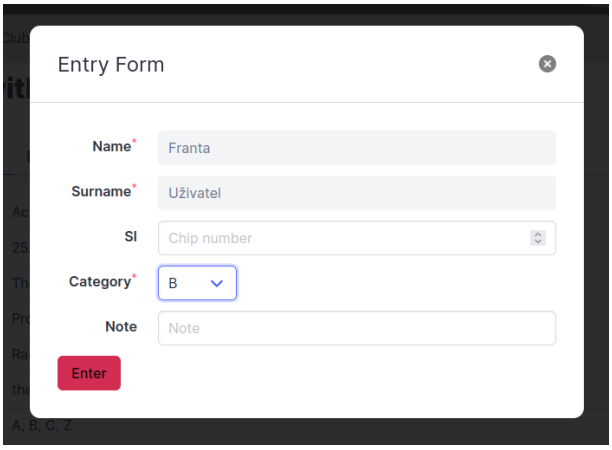


Figure 4.3: The logged-in user's name is pre-filled when entering an event.

4.6.4 Managing entry fees

The surcharges for entry fees are managed by users in the FinancesManager policy. They are presented with a table detailing the ORIS events in which any members had late entries or other extra fees, specifying how much they should pay.

The original plans for this section included the management of club registration fees, as mentioned previously in chapter 3.1.1. Struggling with the ORIS API revealed that retrieving all the necessary would have taken too long, rendering the approach impractical. Even if the data was copied and stored in the database, it would have still needed to be updated periodically, and I was not able to find a way to make this feature viable in practice.

I plan on contacting the ORIS developers to ask about potential API improvements to make features like this viable, but I have not found the time to do this in time of submitting the thesis.

[13] Unregistered users able to enter public events.

Chapter 5

Deployment

This chapter describes the deployment of the application. The application is hosted on a Linux server, where it runs as a system service. An `nginx` [33] reverse proxy server listens for requests and passes them to the service. At the time of writing this thesis, the application is currently deployed in production and being used by an orienteering club in Prostějov, Czechia. Until the date of the thesis defence, a demo version is also available. More details on this in Appendix C.

5.1 Preparing the environment

The application binary is not self-contained, meaning it does not contain the .NET and ASP.NET runtimes. Those need to be installed on the server for the application to be able to run. This makes the resulting binary smaller and reduces the build time, as well as the time it takes to copy it onto the server. PostgreSQL is also running on the production server as well as nginx, which is configured to handle SSL and redirects all HTTP requests to HTTPS.

5.2 Publishing the application

To build the release version of a dotnet application, one would use the `dotnet publish` command from the `.NET CLI` [18] tool. The next step is to bundle the EF Core database migrations, so that they can be applied on the server. We do this with the `dotnet-ef` [10] tool, which extends the .NET cli. The detailed build instructions are present in the README file, located in the implementation archive in Appendix A. The resulting directory, containing the published app and the EF bundle, is then copied to the `/opt/jpvis/<application version>` directory on the server.

5.3 Running and updating

The application runs as a system service supervised by `runit` [35]. The service is defined by a short run script and the application is updated with a slightly longer script. Both of them are available in Appendix D. The update script takes the application version as an argument, corresponding to the version published to the `/opt` directory. The EF migration bundle is applied and a symbolic link is updated to point to the current version. With this approach, it is possible to switch versions at will, provided Semantic Versioning principles are respected.

5.4 Why so complicated?

In part, the motivation behind deploying the application in this manner instead of utilising a managed service for hosting .NET projects, which would have most likely been easier, was the cost. Most orienteering clubs run on volunteers and do not have money to spare. The yearly cost for the server is around 1500 CZK (roughly 60 EUR) and I estimate that the processing power could withstand running ten more instances of the application at the same time, should it be deployed in other clubs as well. Having full root access to the server also allows hosting complementary services.

Chapter 6

Conclusion

The goal of this thesis was to design and implement an information system for an orienteering club. First, I explained the motivation and the specifics of Czech orienteering in regards to the system. I went over some existing implementations, outlined the requirements and use cases. In the following chapter, I explained the software design patterns and principles I used while developing the application, and presented the database schema. Chapter four applied those principles in practice, outlined the chosen technologies and reasoning behind the choices and explored some of the finer implementation details. The penultimate chapter presented the solution for deploying the application in a production environment, bringing us here, to the conclusion.

The application is now deployed at a real orienteering club, being used to enter weekly training events. I expect to continue the development, especially further extending the financial section.

6.1 Feedback from the users

As the application is now deployed at a real orienteering club, I have had an opportunity to collect feedback from real users. The feedback form was filled out by nobody, yet I still got some valuable verbal feedback, including a few bug reports, which were promptly fixed. One user described the application as "boring but functional". The application has not been running for very long in the club, so I have not received any detailed feedback on specific features. The received feedback mostly only concerned the UI

6.2 Future of the application

Some of the planned functionality includes:

- *Administrator area* – A user interface for administrators should be implemented to allow them to manage users, especially assign roles to

them.

- *Localisation* – While the app is primarily in Czech, some parts of the ASP.NET Core framework (like form validation) are presented in English and are quite tricky to localise. A plan for the future is to translate all hard-coded Czech strings to English so that the code is strictly English-only and implement a localisation middleware to then translate to different languages.
- *Finances* – Extending the financial section by connecting it to a banking API and managing all registration fees automatically. Figuring a way to make this approach viable with the ORIS API.
- *Email notifications* – The edit token of unified entries could be sent as an email with a link to edit the entry so that the user does not have to remember it.
- *File uploads* – I have already started developing a file upload feature that would allow event managers to upload files, such as event bulletins, start lists or results, to event pages. This can also be utilised to upload and insert images into news articles.

Appendix A

Implementation archive

The electronic version of the thesis[1] contains an archive with the implementation (the file `impl.tar.gz`). All the necessary build instructions are located in the `README.adoc` file.

[1] Available at https://is.muni.cz/th/evps3

Appendix B

Entity Relationship Diagram

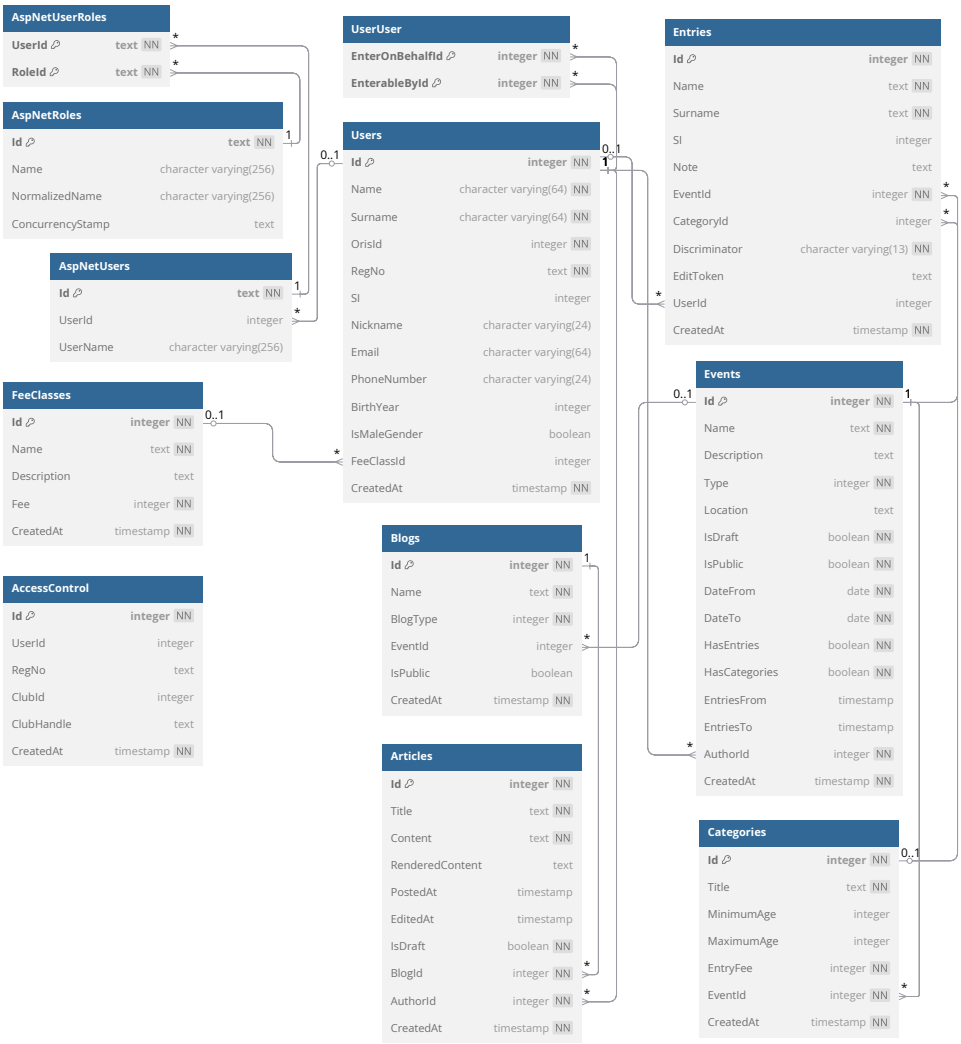


Figure B.1: Entity Relationship Diagram

Appendix C

Demo

The application is currently running in production and available at https://obprostejov.cz/. Due to the fact that accessing most of the features requires having an account in ORIS and being a member of the club, a slightly modified demo version is available temporarily at https://demo.obprostejov.cz/.

The demo disables logging in with ORIS and instead provides the ability to log in as one of two fake users: A regular user with no extra permissions and an administrator, who has all available permissions (event manager, news manager and finance manager).

The demo will be available up to the date of the thesis defence.

Appendix D

Scripts

D.1 Deploy script

```
#!/bin/sh
export DOTNET_ROOT=/opt/dotnet
export PATH=$PATH:/opt/dotnet
export ASPNETCORE_ENVIRONMENT=Production

TEMP=$(getopt -o Mf --longoptions nomigrate,force -- "$@")

if [ "$#" -lt 1 ] || [ "$#" -gt 2 ]; then
        echo "Usage: $0 [--nomigrate] <jpvis_version>" >&2
        exit 1
fi

eval set -- "$TEMP"

MIGRATE=true
FORCE=false

while true; do
        case "$1" in
                -M | --nomigrate ) MIGRATE=false; shift;;
                -f | --force ) FORCE=true; shift;;
                -- ) shift; break;;
                * ) break;;
        esac
done

if $MIGRATE && [ ! -f /opt/jpvis/"$1"/efbundle ]; then
        echo "Cannot apply migrations: efbundle doesn't exist" >&2
```

```
        exit 1
fi

if ! $MIGRATE && ! $FORCE && [ -f /opt/jpvis/"$1"/efbundle ]; then
        echo "Ran with --nomigrate yet migration bundle exists. \
            Are you sure? To confirm, re-run with -f|--force" >&2
        exit 1
fi

sv stop jpvis || exit 1

if $MIGRATE; then
        cd /opt/jpvis/"$1" && ./efbundle || exit 1
fi

ln -nsf /opt/jpvis/"$1" /www/jpvis && sv start jpvis
```

D.2 System service run script

```
#!/bin/sh
exec chpst -u html:html -C /www/jpvis \
    env DOTNET_ROOT=/opt/dotnet \
        ASPNETCORE_ENVIRONMENT=Production \
        ASPNETCORE_URLS=http://127.0.0.1:10001 \
    /opt/dotnet/dotnet MVC.dll
```

Bibliography

[1] *About GitHub*. URL: https://github.com/about (visited on 15/05/2025).

[2] *About Orienteering*. URL: https://orienteering.sport/orienteering/ (visited on 12/05/2025).

[3] *AngleSharp library*. URL: https://github.com/AngleSharp/AngleSharp (visited on 13/05/2025).

[4] *Bulma CSS framework*. URL: https://bulma.io/ (visited on 13/05/2025).

[5] Scott Chacon and Ben Straub. *Pro Git: Git Basics – Tagging*. URL: https://git-scm.com/book/en/v2/Git-Basics-Tagging (visited on 15/05/2025).

[6] Scott Chacon and Ben Straub. *Pro Git: Git Tools – Debugging with Git*. URL: https://git-scm.com/book/en/v2/Git-Tools-Debugging-with-Git (visited on 15/05/2025).

[7] *CommonMark*. URL: https://commonmark.org/ (visited on 15/05/2025).

[8] *Czech Orienteering Federation (ČSOS)*. URL: https://orientacnisporty.cz (visited on 12/05/2025).

[9] *Entity Framework Core inheritance*. URL: https://learn.microsoft.com/en-us/ef/core/modeling/inheritance (visited on 12/05/2025).

[10] *Entity Framework Core tools reference – .NET Core CLI*. URL: https://learn.microsoft.com/en-us/ef/core/cli/dotnet (visited on 13/05/2025).

[11] *Flurl*. URL: https://flurl.dev/ (visited on 14/05/2025).

[12] Martin Fowler. *Patterns of Enterprise Application Architecture*. 1st ed. Addison-Wesley Professional, 2002. ISBN: 0321127420,9780321127426. URL: http://gen.lib.rus.ec/book/index.php?md5=d68d544e079fb1bca54f70ff63b4e1a3.

[13] Adam Freeman. *Pro ASP.NET Core MVC 2*. 7th. Apress, 2017. ISBN: 978-1-4842-3150-0. URL: http://gen.lib.rus.ec/book/index.php?md5=376fd3dd73996ddbe39c16c8c71e605a.

[14] *Git – a free and open source distributed version control system*. URL: https://git-scm.com/ (visited on 15/05/2025).

[15] Dave Glick. *What is Razor Templating, really?* URL: https://www.twilio.com/en-us/blog/what-is-razor-templating (visited on 14/05/2025).

[16] *HtmlSanitizer library*. URL: https://github.com/mganss/HtmlSanitizer (visited on 13/05/2025).

[17] *Markdig library*. URL: https://github.com/xoofx/markdig (visited on 13/05/2025).

[18] *.NET CLI overview*. URL: https://learn.microsoft.com/en-us/dotnet/core/tools/ (visited on 13/05/2025).

[19] *.NET Microservices: Architecture for Containerized .NET Applications*. URL: https://aka.ms/microservicesebook (visited on 12/05/2025).

[20] *NodaTime library*. URL: https://nodatime.org/ (visited on 13/05/2025).

[21] *PostgreSQL*. URL: https://www.postgresql.org/ (visited on 13/05/2025).

[22] Mark J. Price. *Tools and Skills for .NET 8*. EXPERT INSIGHT. Packt, 2024. ISBN: 9781837635207,9780138203368. URL: http://gen.lib.rus.ec/book/index.php?md5=C6167437533A81D54737A6A296194CEE.

[23] Mark Price. *C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals: Start building websites and services with ASP.NET Core 8*. 8th ed. Packt, 2023. ISBN: 9781837635870. URL: http://gen.lib.rus.ec/book/index.php?md5=1890666F349972CB66ACA9D096933B03.

[24] *Privátní zóna (Private Zone)*. URL: https://www.privatni-zona.cz/ (visited on 12/05/2025).

[25] *Rust language documentation – std::result*. URL: https://doc.rust-lang.org/std/result/ (visited on 14/05/2025).

[26] *SPORTident GmbH Germany*. URL: https://www.sportident.com/company/about-us (visited on 12/05/2025).

[27] *Semantic Versioning*. URL: https://semver.org/ (visited on 14/05/2025).

[28] *View components in ASP.NET Core*. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/view-components?view=aspnetcore-9.0 (visited on 14/05/2025).

[29] *Wikipedia: Common Language Infrastructure*. URL: https://en.wikipedia.org/wiki/Common_Language_Infrastructure (visited on 13/05/2025).

[30] *eob.cz*. URL: https://members.eob.cz (visited on 12/05/2025).

[31] *htmx library*. URL: https://htmx.org/ (visited on 13/05/2025).

[32] *markdown-it – Markdown parser, done right*. URL: https://github.com/markdown-it/markdown-it (visited on 15/05/2025).

[33] *nginx*. URL: https://nginx.org/ (visited on 13/05/2025).

[34] *npm – the Javascript package manager*. URL: https://github.com/npm (visited on 14/05/2025).

[35] *runit init system and service supervisor*. URL: https://smarden.org/runit/ (visited on 13/05/2025).

[36] *ČSOS ORIS API*. URL: https://oris.orientacnisporty.cz/api?lang=en (visited on 12/05/2025).

[37] *ČSOS ORIS*. URL: https://oris.orientacnisporty.cz?lang=en (visited on 12/05/2025).